LivingApps-Dokumentation

LivingLogic AG

27.02.2024

Inhaltsverzeichnis

KAPITEL 1

Importfunktion

Bevor Sie Daten importieren, gibt es einige wichtige Punkte, die Sie vorab durchdenken sollten, um sicherzustellen, dass der Import erfolgreich und ohne Probleme abläuft.

1.1 Grundsätzliches

Der Datenimport gelingt, wenn Daten aus einer CSV- oder eine Excel-Datei importiert werden. Vorteil dieser Vorgehensweise ist die Möglichkeit, die Daten vorher auf ihre Qualität zu überprüfen. Das bedeutet, dass die Daten vollständig, konsistent und korrekt sind. Überprüfen Sie, ob es leere Felder oder Duplikate gibt und ob die Daten in der richtigen Form sind.

Beispielsweise sollte der Inhalt der Spalten immer in der gleichen Form vorliegen: Eine Internetadresse sollte in LivingApps als `https://ab.de` angegeben sein. Probleme bereiten unterschiedliche Einträge wie `www.ab.de` und `https://cd.de` in einer Spalte. Man kann damit aus LivingApps heraus bestimmte Funktionalitäten nicht nutzen.

1.2 Unterschiedliche Szenarien für einen Import

Es gibt unterschiedliche Anlässe, warum und wie in LivingApps Daten importiert werden:

1. Es sollen in eine bestehende LivingApp Daten übernommen werden
    a. Es kommen weitere Datensätze dazu
    b. In bestehende Datensätze werden weitere Daten hinzu importiert
2. Erzeugen einer neuen LivingApp beim Import

1.3 Übernahme von Daten in eine bestehende LivingApp

Mit der Übernahme in eine bestehende LivingApp werden weitere Datensätze dieser LivingApp hinzugefügt.

Das Vorgehen läuft folgendermaßen: Wählen Sie in der App, die mit weiteren Datensätzen ergänzt werden soll, den Menüpunkt *Daten* → *Import*.

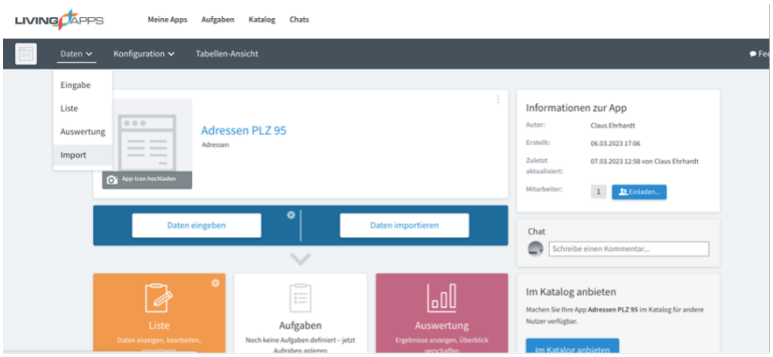


Abb. 1: Das Import-Menü

Kopieren Sie die zu importierenden Excel-Daten mit den dazugehörigen Spaltenköpfen in die Zwischenablage und kopieren Sie die Daten in das geöffnete Import-Fenster:

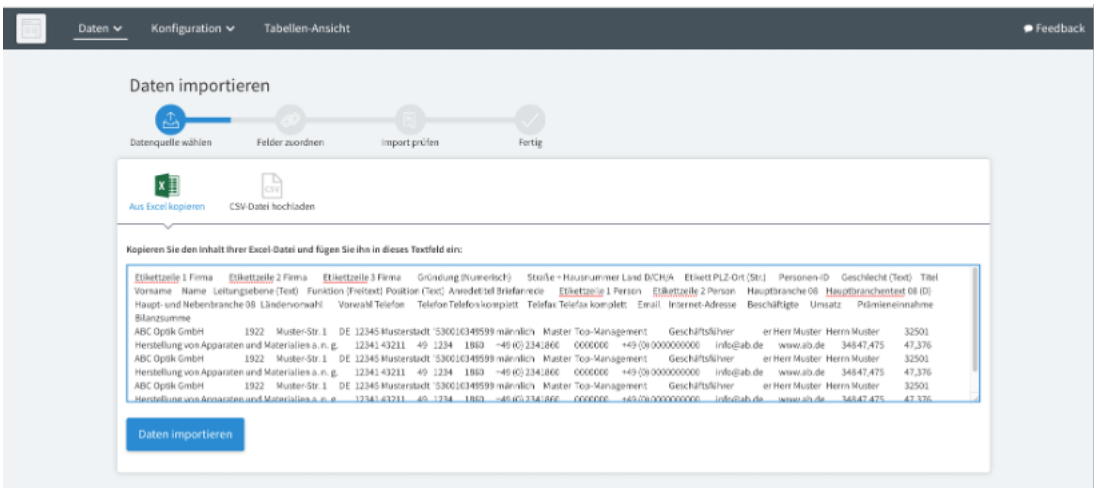


Abb. 2: Daten kopiert

Klicken Sie *Daten importieren* an.

Sie erhalten die Anzahl der Datensätze — hier werden 3 Datensätze angezeigt —, die das System erkannt hat sowie die Zuordnungs-Tabelle mit den jeweiligen Feldern (*Verfügbare Elemente* und *App-Eingabefelder*). Auf der linken Seite sehen Sie die aus dem Excel mitkopierten Spaltenköpfe.

Vergleichen Sie die Anzahl der Datensätze in LivingApps mit den Zeileneinträgen in Excel, die Sie importieren wollen. Die Zahl ist ein Indiz für die korrekte Übernahme in die LivingApps. (Die Anzahl Datensätze und die Anzahl der Excelzeilen (ohne die Spaltenköpfe) sollten gleich sein.)

Nun können Sie die verfügbaren Elemente den App-Eingabefeldern zuordnen. Sie können das manuell nacheinander tun, alternativ so machen, wie Sie es beim letzten Import (wird gespeichert) zugeordnet haben, nach Namen (das System sucht automatisch nach den identischen Namen für die App-Eingabefelder) oder nach Reihenfolge (also von oben nach unten) zuordnen.

Die Spaltenköpfe müssen nicht notwendigerweise den identischen Namen haben. Sie sollten bei der manuellen Zuordnung jedoch sicher sein, dass die verfügbaren Elemente inhaltlich mit den App-Eingabefeldinhalten zusammenpassen.

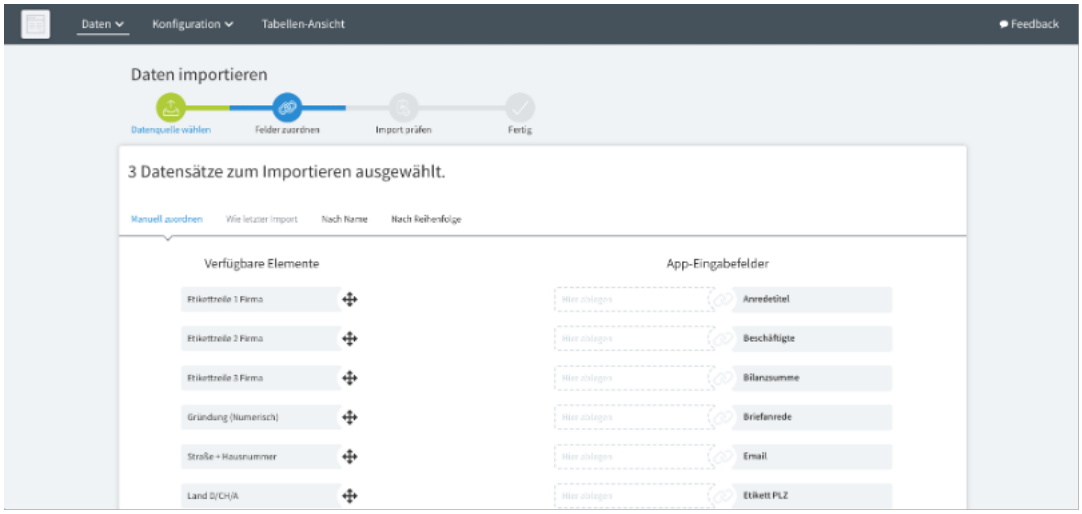


Abb. 3: Felder zuordnen

Beginnen Sie im Anschluss mit einem Klick auf *Import starten* den Import. Es kann bei großen Datenmengen ein paar Minuten dauern, bis alle Daten in der Datenbank an der richtigen Stelle angekommen sind. Den Status können Sie über die Anzeige des importieren Prozentsatzes der Daten mitverfolgen.

Im Anschluss können Sie das Ergebnis über das Menü *Daten* → *Liste* begutachten.

1.4 Übernahme von Daten in eine bestehende LivingApp (Daten ergänzen)

Eine weitere Form des Imports von Daten ist das Ergänzen bestehender Datensätze mit zusätzlichen Informationen.

Wählen Sie in der App, die mit weiteren Datensätzen ergänzt werden soll, über Menü *Daten* → *Import*.

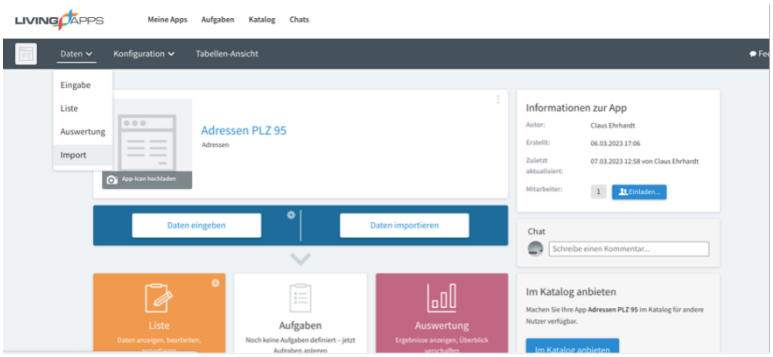


Abb. 4: Das Import-Menü

Kopieren Sie die Excel-Datei oder den Ausschnitt daraus mit den neuen, zu ergänzenden Spaltenköpfen (Feldern) sowie dem Spalteninhalt in das sich öffnende Fenster. Klicken Sie dann *Daten importieren* an.

Sie erhalten die Anzahl der Datensätze — hier werden 3 Datensätze angezeigt —, die das System erkannt hat sowie die Zuordnungs-Tabelle mit den jeweiligen Feldern (*Verfügbare Elemente* und *App-Eingabefelder*). Auf der linken Seite sehen Sie die aus dem Excel mitkopierten Spaltenköpfe.

Vergleichen Sie die Anzahl der Datensätze in LivingApps mit den Zeileneinträgen in Excel, die Sie importieren wollen. Die Zahl ist ein Indiz für die korrekte Übernahme in die LivingApps.

Zur Beachtung: Beim Aktualisieren von bestehenden Datensätzen kann es auch dazu kommen, dass ausgewählte Felder überschrieben werden, die z.B. beim Import oder durch Eingabe bereits gespeichert waren!

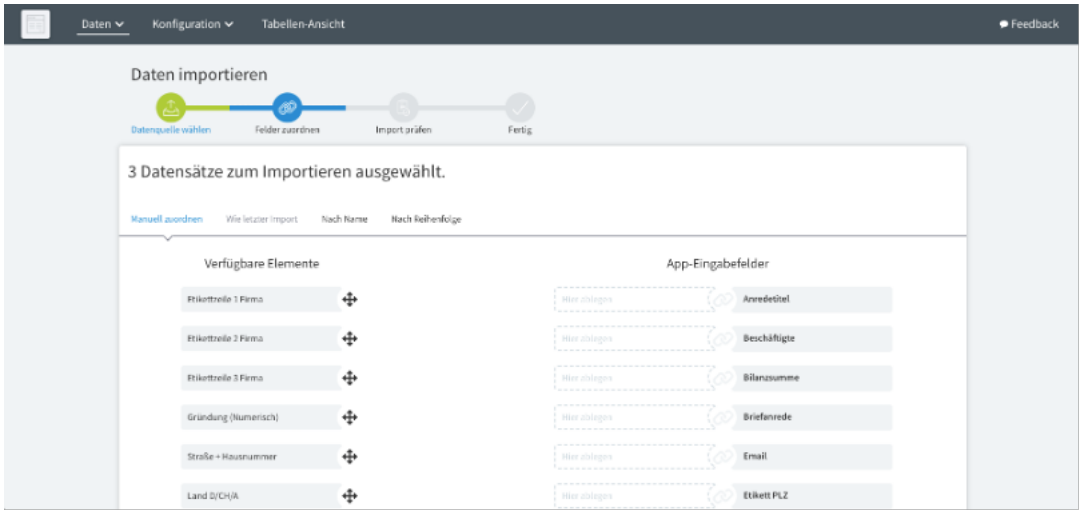


Abb. 5: Felder zuordnen

Wenn Sie den Datenbankinhalt der bestehenden Datensätze ergänzen wollen, benötigen Sie hier ein eindeutiges Schlüsselfeld. Wählen Sie ein Schlüsselfeld, z.B. der eindeutige Firmenname, und kreuzen das Kästchen hierfür an, damit das System erkennen kann, zu welchem Datensatz der Inhalt überschrieben oder hinzugefügt werden soll.

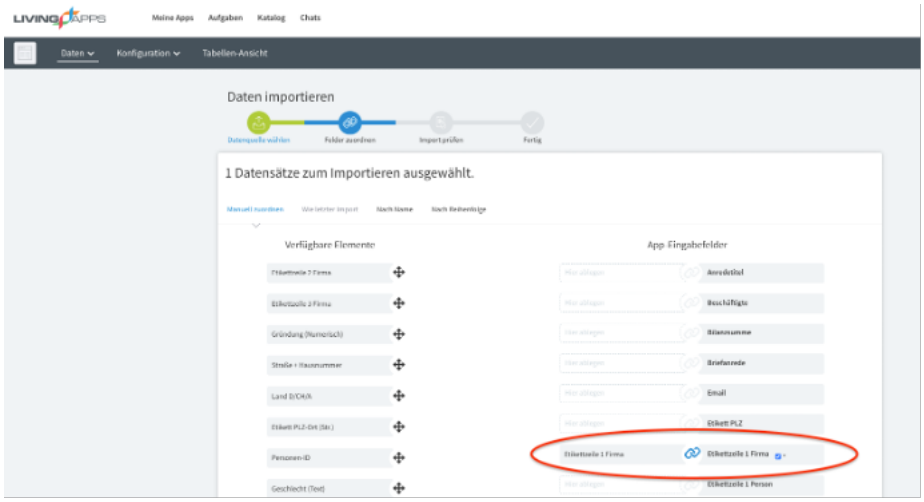


Abb. 6: Schlüsselfeld ausgewählt

Nun können Sie die restlichen benötigten Elemente den App-Eingabefeldern per „Drag and Drop“ zuordnen.

Sie können das manuell nacheinander tun, oder so machen, wie Sie es beim letzten Import (wird gespeichert) zugeordnet haben, nach Namen (das System ordnet Spalten zu Feldern mit demselben Namen) oder nach Reihenfolge (also von oben nach unten) zuordnen.

Zur Beachtung: Möglicherweise ist es notwendig, zwei oder mehrere Felder als Schlüsselfeld zu definieren, damit der passende Datensatz identifiziert werden kann. Zum Beispiel müssen Sie Nachname und Vorname ankreuzen, um den eindeutigen Datensatz zu identifizieren. Dies kann dann der Fall sein, wenn beispielsweise der Nachname „Müller“ öfter als nur einmal vorkommt.

Die Spaltenköpfe müssen nicht notwendigerweise den identischen Namen haben. Sie sollten bei der manuellen Zuordnung jedoch sicher sein, dass die verfügbaren Elemente inhaltlich mit den App-Eingabefeldinhalten zusammenpassen.

Beachten Sie, dass es zu allerlei Fehlern beim Import kommen kann, wenn bei :guilabel`Verfügbare Elemente` Spalten mit Werten ausgewählt sind, deren Inhalt aber nicht zum Feldtyp passt. Beispielsweise erwartet die LivingApp beim Typ URL einen Feldinhalt nicht in der Form `www.blafasel.de`. Richtig ist an dieser Stelle ein Eintrag

in der Form `https://blafasel.de`. Darüber hinaus gibt es Probleme in den Feldinhalten, die zum Beispiel Anführungszeichen beinhalten. Bei einer Kopie direkt aus Excel heraus kann der Inhalt nicht richtig interpretiert werden und der Import bricht an dieser Stelle ab. Ein Beispiel für einen Importabbruch ist folgender Inhalt in einem Feld, welches Sie aus einer Exceltabelle kopiert haben: `Hase "Paul" GmbH`. Suchen Sie nach derartigen Inhalten und löschen Sie die Anführungszeichen.

Bemerkung: Führen Sie den Import durch Hochladen einer CSV–Datei durch, die Daten in Anführungszeichen enthält, heilt das System den Fehler und der Import gelingt problemlos.

Auch Mehrfachinhalte in den Datenzellen führen in der Regel zum Abbruch, wenn der Feldtyp einzelne Inhalte erwartet. Z.B. führt ein Eintrag `ab@cd.de, fg@df.de` für ein E-Mail-Feld zum Abbruch des Imports.

Achten Sie auf absolut saubere Daten!

Beginnen Sie im Anschluss mit einem Klick auf *Import starten* den Import. Es kann bei großen Datenmengen ein paar Minuten dauern, bis alle Daten in der Datenbank an der richtigen Stelle angekommen sind. Den Status können Sie über die Anzeige des importieren Prozentsatzes der Daten mitverfolgen.

Im Anschluss können Sie das Ergebnis über das Menü *Daten* → *Liste* überprüfen.

1.5 Import in eine neue LivingApp

Eine weitere, sehr effiziente Möglichkeit, Daten mit LivingApps zu verarbeiten, ist der Import aus einer Excel-Datei bzw. einer CSV-Datei und dem Aufbau einer neuen App aus den importierten Daten.

Hierzu geben Sie derzeit folgenden Link in den Browser ein, in dem Sie die LivingApps geöffnet haben:

```
https://my.living-apps.de/import-neue-app.htm#
```

Sie gelangen in dieses Fenster:

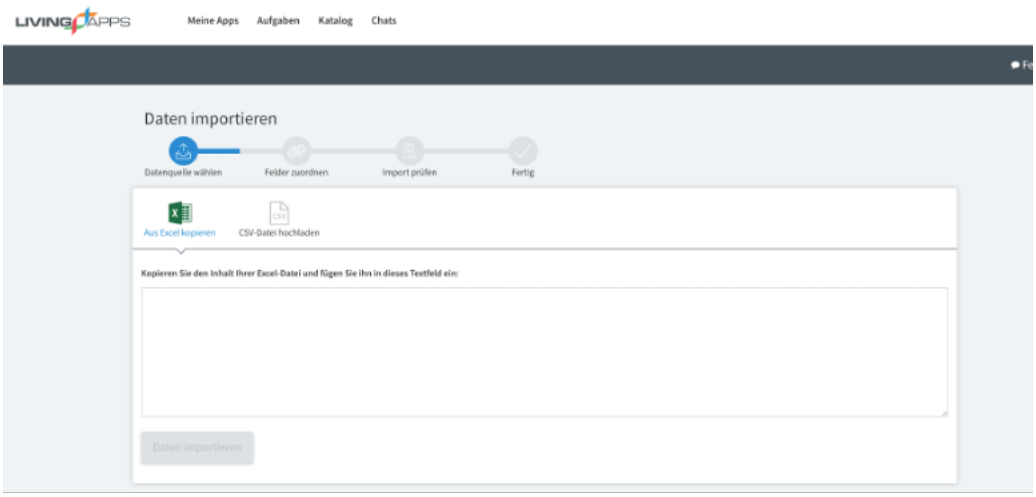


Abb. 7: Daten kopieren

Kopieren Sie die zu importierenden Excel-Daten mit den dazugehörigen Spaltenköpfen in die Zwischenablage und kopieren Sie die Daten in das geöffnete Import-Fenster:

Klicken Sie *Daten importieren* an.

Sie erhalten die Anzahl der Datensätze — hier wird 1 Datensatz angezeigt —, die das System erkannt hat sowie die Zuordnungs-Tabelle mit den jeweiligen Feldern (*Verfügbare Elemente*).

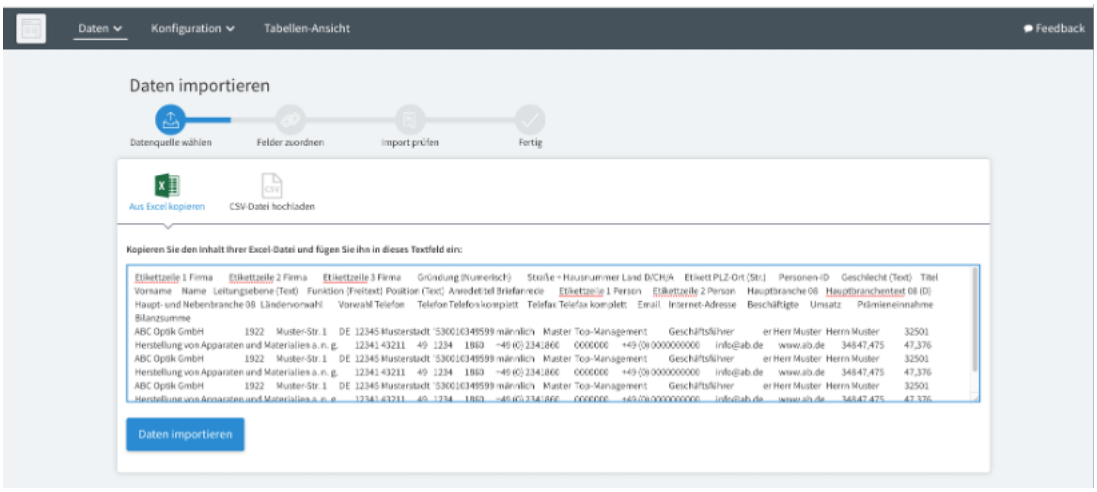


Abb. 8: Daten kopiert

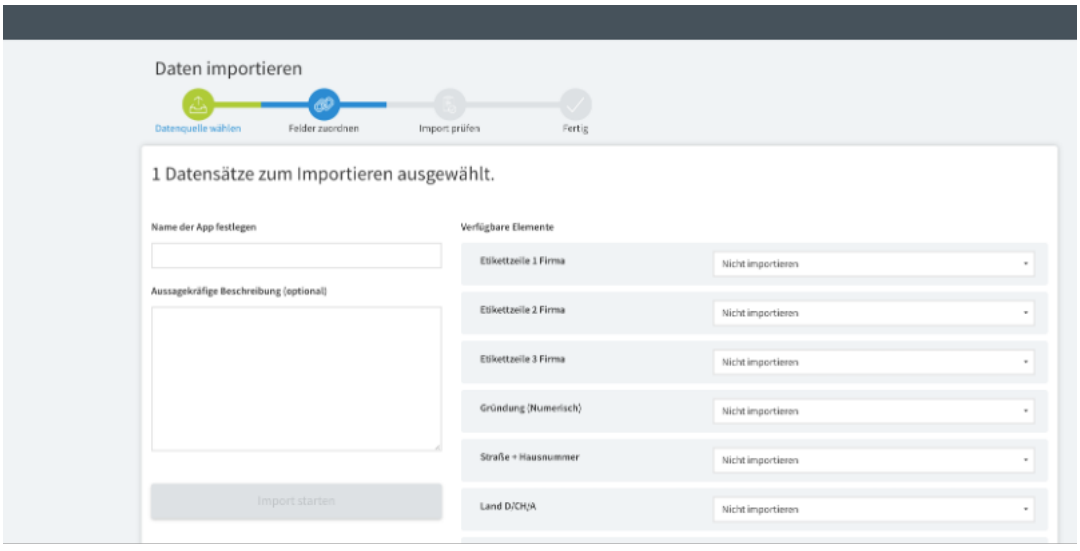


Abb. 9: Feldtypen festlegen

Vergleichen Sie die Anzahl der Datensätze, die Sie in die LivingApps importieren wollen, mit den Zeileneinträgen in Excel. Die Zahl ist ein Indiz für die korrekte Übernahme in die LivingApps. (Die Anzahl Datensätze und die Anzahl der Excelzeilen ohne die Spaltenköpfe sollten gleich sein.)

Bevor Sie die Daten importieren, benötigt die neu in LivingApps erzeugte LivingApp weitere Angaben zu den Feldtypen. Sie können zu den verfügbaren Elementen folgende Feldtypen kreieren:

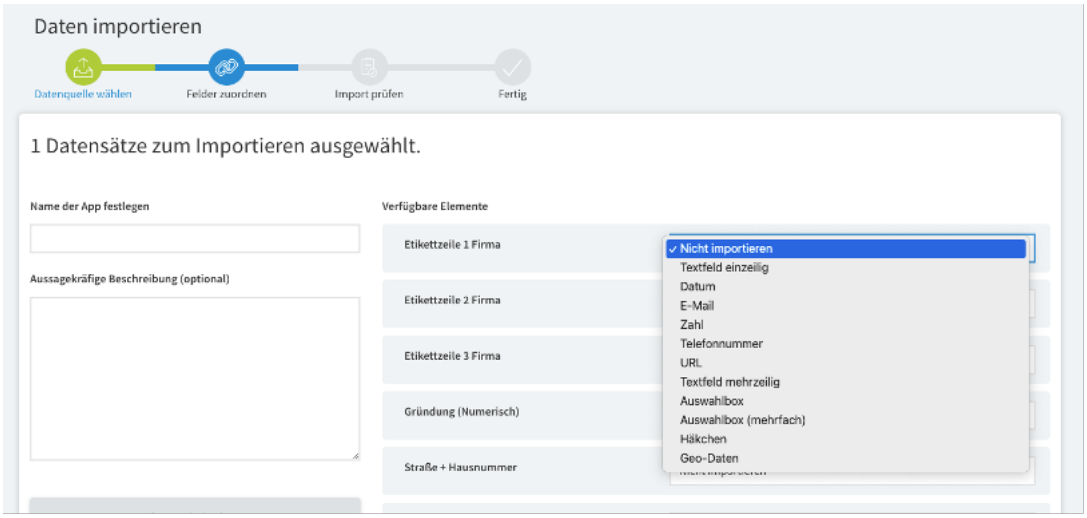


Abb. 10: Auswahl der Feldtypen

Es ist Ihnen bei der Auswahl der „Verfügbaren Elemente“ auch möglich, einzelne Felder nicht zu importieren. Hierzu lassen Sie die Voreinstellung *Nicht importieren* einfach ausgewählt stehen.

Auf der linken Seite dieser Maske werden Sie nach dem Namen der neuen LivingApp sowie einer möglichen weiteren Beschreibung der LivingApp gefragt.

Beachten Sie, dass es zu allerlei Fehlern beim Import kommen kann, wenn für die „Verfügbaren Elemente“ Feldtypen ausgewählt wurden, die nicht zu den vorhandenen Feldwerten passen. Beispielsweise erwartet die LivingApp beim Typ URL einen Feldinhalt nicht in der Form `www.blafasel.de`. Richtig ist an dieser Stelle ein Eintrag in der Form `https://blafasel.de`. Darüber hinaus gibt es Probleme bei Feldinhalten, die zum Beispiel Anführungszeichen beinhalten. Bei einer Kopie direkt aus Excel heraus kann der Inhalt nicht richtig interpretiert werden und der Import bricht an dieser Stelle ab. Ein Beispiel für einen Importabbruch ist folgender Inhalt in einem Feld, welches Sie aus einer Exceltabelle kopiert haben: `Hase "Paul" GmbH`. Suchen Sie nach derartigen Inhalten und löschen Sie die Anführungszeichen.

Bemerkung: Führen Sie den Import durch Hochladen einer CSV–Datei durch, die Daten in Anführungszeichen enthält, heilt das System den Fehler und der Import gelingt problemlos.

Auch Mehrfachinhalte in den Datenzellen führen in der Regel zum Abbruch, wenn der Feldtyp einzelne Inhalte erwartet. Z.B. führt ein Eintrag `ab@cd.de, fg@df.de` für ein E-Mail-Feld zum Abbruch des Imports.

Achten Sie auf absolut saubere Daten!

Sobald Sie im Anschluss auf *Import starten* klicken, beginnt der Import. Sollten es viele Datenzeilen sein, die geladen werden, kann sich der Import bis zu einigen Minuten hinziehen. Sie erkennen den Importstatus am Prozentsatz der geladenen Daten.

Nach dem Import erhalten Sie folgende Meldung:

Abb. 11: Import erfolgt

1.5.1 Begutachtung der neuen App

Um den Import zu prüfen und mit dem Aufbau eines Formulars der LivingApp zu beginnen, klicken Sie auf *Zur Detailansicht der App* und gelangen in das Startfenster der App. Diese sieht so aus:

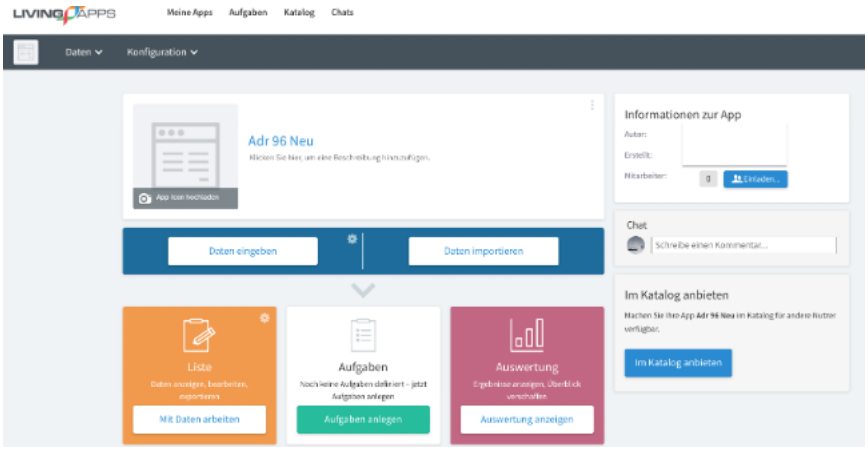


Abb. 12: App-Startseite

Um zur Anzeige der importierten Daten zu gelangen, wählen Sie im Menü *Daten* → *Liste* aus oder klicken weiter unten in der orangenen Kachel auf *Liste*.

Im Anschluss sehen Sie nun die Liste der importieren Daten:

1.5.2 Konfiguration der neuen App

Die Anzeige oder Reihenfolge der in der Liste verwendeten Spalten können Sie ändern, indem Sie im Menü *Konfiguration* → *Erweitert* wählen, in der linken Navigation *Datenmanagement*, und dort den Punkt *Felder* auswählen.

Um die Spaltenreihenfolge für alle Spalten anzupassen, klicken Sie die Spaltenüberschrift *Reihenfolge Datenmanagement* an, wählen *Bearbeiten ein* und ändern dann in der Spalte *Reihenfolge Datenmanagement* die Werte. Klicken Sie anschließend den erscheinenden Button *Alle speichern* an.

Um festyzulegen, welche Spalten in der Liste dargestellt werden gehen Sie ähnlich vor: Klicken Sie die Spaltenüberschrift *In Liste?* an, wählen *Bearbeiten ein* und selektiern oder deselektieren dann in der Spalte *In Liste?* den haken. Klicken Sie anschließend den erscheinenden Button *Alle speichern* an.

Um für diese App ein Formular zu bauen, wählen Sie im Menü *Konfiguration* → *Eingabe*. Sie erhalten die Maske des „Formularbuilders“ mit den Feldnamen zur Platzierung auf dem Formularbogen. Diese Felder finden Sie unter *Private Felder*. Gestalten Sie Ihr Formular, wie gewünscht.

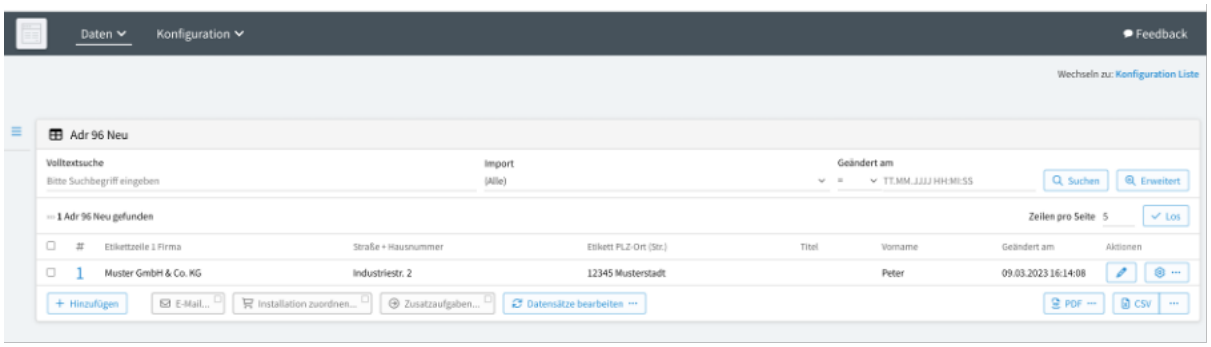


Abb. 13: Datenmanagement

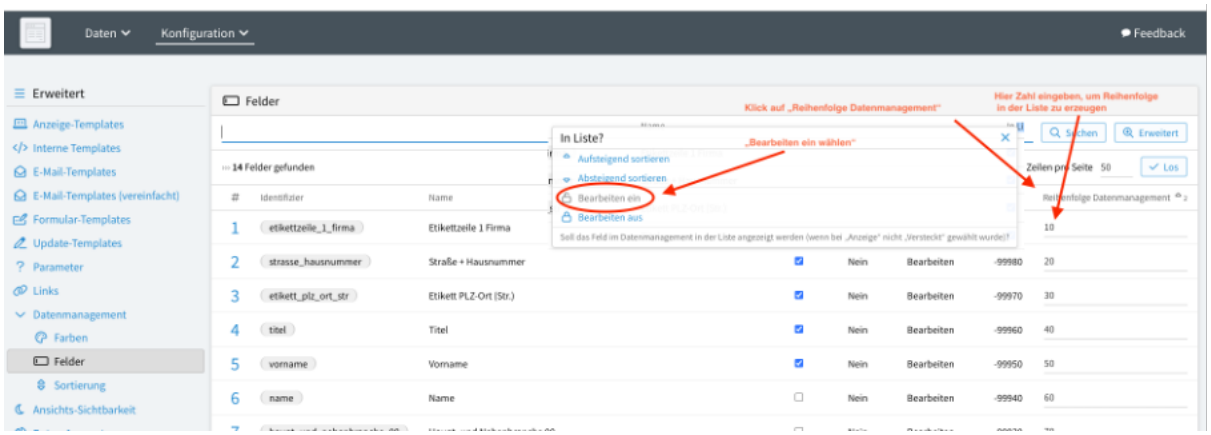


Abb. 14: Spaltenreihenfolge festlegen

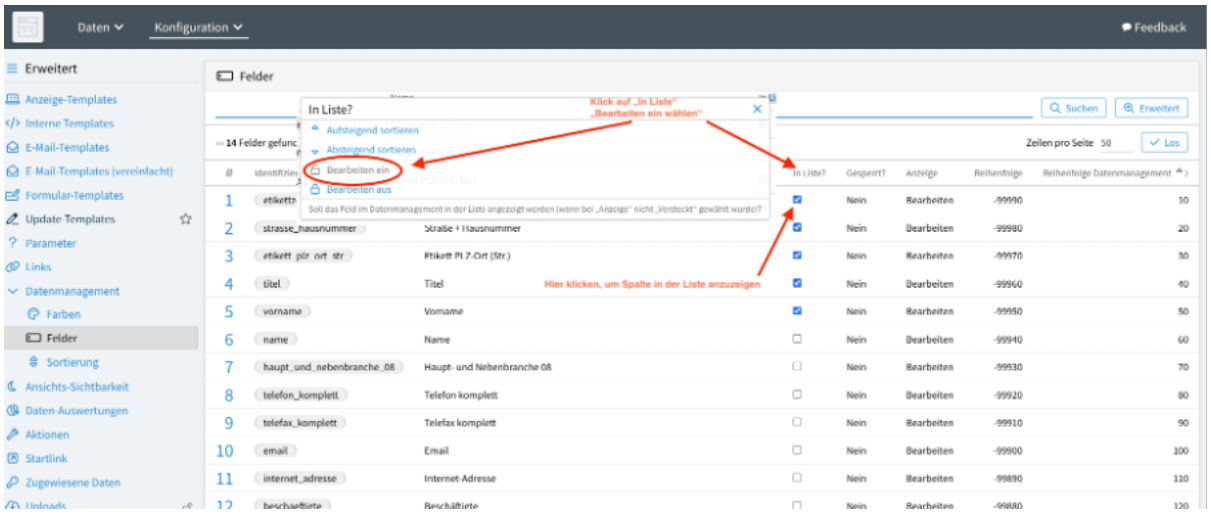


Abb. 15: Spaltenanzeige festlegen

1.6 Unterstützung des Imports durch aktive Hinweise auf Fehler

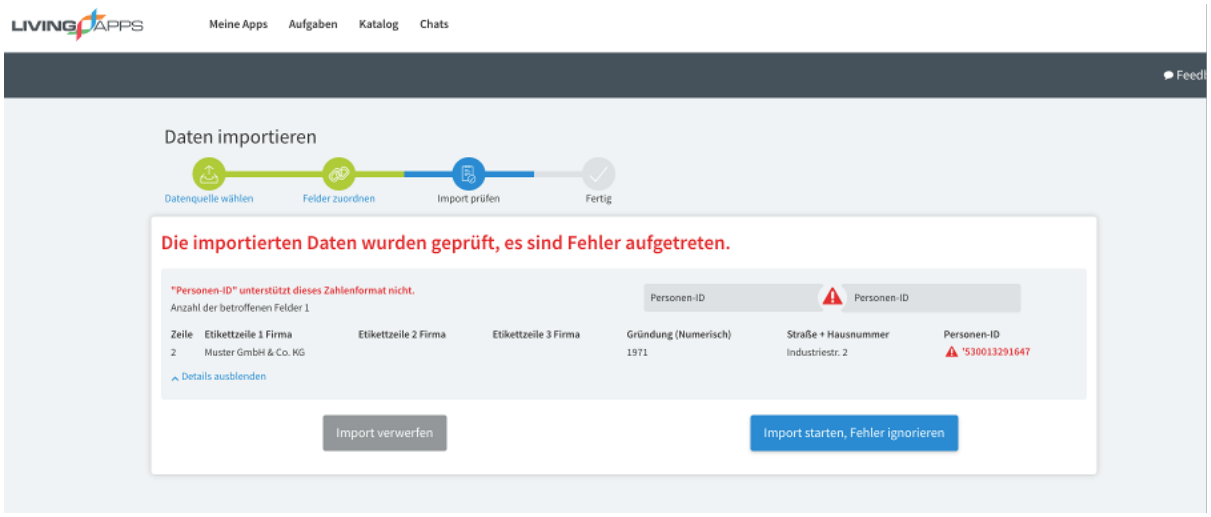


Abb. 16: Importfehler

Beim Import können Fehler auftreten, die durch Dateninhalte verursacht werden, die nicht zu den gewählten Feldtypen passen. In diesem Falle unterstützt Sie das System mit Fehlermeldung wie oben dargestellt. Hier passt z.B. der Feldinhalt aufgrund eines vorangestellten Anführungszeichens nicht zu einem reinen Zahlenfeld. Sie haben die Möglichkeit, den Fehler zu ignorieren und den Import trotzdem zu starten. Andererseits stoppen Sie den Import und korrigieren den Fehler in Excel oder den Daten der CSV-Datei.

KAPITEL 2

Erweiterte Funktionen

In dieser Dokumentation werden die erweiterten Funktionen von LivingApps erklärt, die Erstellung und Anwendung demonstriert und anhand eines Anwendungsbeispiels vorgestellt.

Für die Nutzung der erweiterten Funktionen müssen Sie Administrationsrechte an der App besitzen.

2.1 Allgemeines

In dieser Dokumentation werden die *erweiterten Funktionen* von LivingApps erklärt, die Erstellung und Anwendung demonstriert und anhand eines Anwendungsbeispiels vorgestellt.

Für die Nutzung der erweiterten Funktionen müssen Sie Administrationsrechte an der App besitzen.

2.1.1 Erweiterte Funktionen

Wählen Sie im Menü Ihrer App *Konfiguration* → *Erweitert*. In der linken Spalte finden Sie alle verfügbaren erweiterten Funktionen. Eine ausführliche Beschreibung der einzelnen Funktionen können Sie im jeweiligen Kapitel nachlesen.

Wenn Sie in einer der erweiterten Funktionen auf Ihre Datensätze zugreifen möchten, so muss hierzu die Templatesprache UL4 verwendet werden. Die UL4-Dokumentation finden Sie unter https://python.livinglogic.de/UL4.html. Dort finden Sie unter anderem auch eine Beschreibung, wie Sie lokale Variablen, Dictionaries und Listen anlegen. Die Formulierung der UL4-Ausdrücke wird jedoch auch im Folgenden an den entsprechenden Stellen erläutert.

Die Ausgabe der Daten Ihrer App erfolgt über die *LivingAPI*. Hierüber können Sie im entsprechenden Kapitel und in der nachfolgenden Dokumentation mehr lesen.

Anzeige-Templates
: Erstellen Sie hier sogenannte „UL4-Templates“, die es Ihnen ermöglichen, die Daten Ihrer LivingApp in jeder denkbaren Form darzustellen und aufzubereiten.

Interne Templates
: Sie haben hier die Möglichkeit, sich eine Bibliothek von „UL4-Templates“ anzulegen. Diese Templates können Sie dann in Ihren Anzeige- und E-Mail-Templates aufrufen.

E-Mail-Templates
: Gestalten Sie hier Vorlagen für E-Mails, die von LivingApps oder vom Benutzer in bestimmten Situationen automatisch verschickt werden:

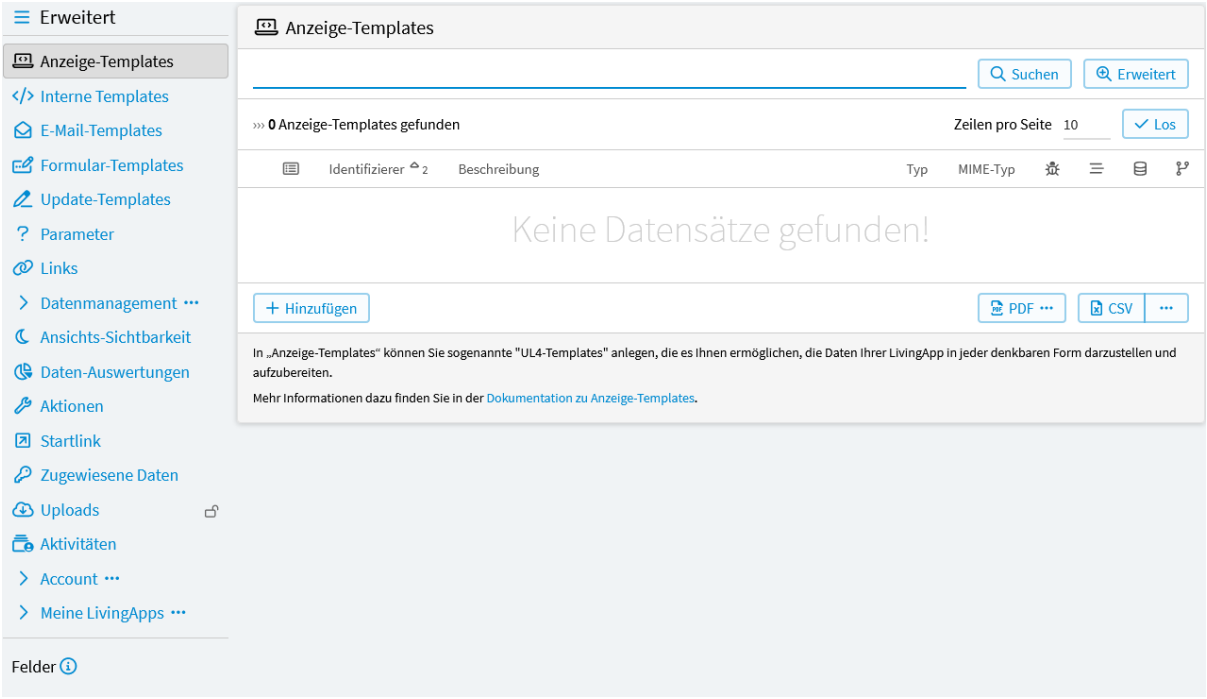


Abb. 1: Anmeldeformular Konfiguration - Erweitert

Beim Anlegen eines neuen Datensatzes, beim Ändern eines Datensatzes, oder wenn ein Benutzer im Datenmanagement eine E-Mail an die in den Datensätzen hinterlegten E-Mail-Adressen verschickt.

Formular-Templates

Mit Formular-Templates können Sie Aktionen definieren, die im Eingabeformular der jeweiligen Ansicht beim Aufruf, beim Speichern, nach dem Speichern oder im Fehlerfall des Formulars ausgeführt werden.

So können z. B. Feldwerte vorbelegt, die Auswahl von Optionen eingeschränkt usw. . . .

Update-Templates

Mit Update-Templates können Sie Aktionen definieren, die im Eingabeformular der jeweiligen Ansicht ausgeführt werden, während der Benutzer Daten eingibt.

Damit ist es möglich — in Abhängigkeit von den Eingaben des Benutzers — Werte in Eingabefeldern zu manipulieren, sowie Felder zu deaktivieren, oder komplett auszublenden. Außerdem kann der Absendebutton in Abhängigkeit von bestimmten Bedingungen aktiviert oder deaktiviert werden.

Parameter

Legen Sie hier Konfigurationswerte fest, auf die Sie dann in den Anzeige- und E-Mail-Templates zugreifen können.

Links

Erstellen Sie Links, um das Navigationsmenü auf der Detailseite Ihrer LivingApp zu erweitern, oder weiterführende Links in einem Panel auf der rechten Seite dieser Detailseite anzuzeigen.

Datenmanagement

Konfigurieren Sie hier die Anzeige der Daten in der Datenliste.

Farben

So können Sie in der Datenliste Datensätze einfärben, wenn diese bestimmte Bedingungen erfüllen.

Felder

Legen Sie hier fest, in welcher Reihenfolge die Felder Ihrer App in der Datenliste (Spalten) angezeigt werden und ob sie in der Datenliste sichtbar sein sollen.

Sortierung

Legen Sie hier die Sortierung der Datensätze im Datenmanagement (Zeilen) fest. Wird nichts ange-

geben, so werden die Datensätze absteigend nach dem Erzeugungsdatum sortiert (d. h. die neuesten Datensätze sind oben).

Ansichts-Sichtbarkeit
Definieren Sie hier die Zeiträume für die Formular-Ansichten, zu welchen diese verfügbar sein sollen und gestalten Sie Meldungen, die vor und nach dem Anzeigezeitraum angezeigt werden.

Daten-Auswertungen
Hier können zusätzliche Darstellungen der Daten in der Liste. Z. B. Monatsauswertungen, Summen, usw. erstellt werden.

Aktionen
Erstellen Sie hier Aktionen, die automatisch (z. B. wenn ein neuer Datensatz vom Benutzer angelegt wird) oder manuell (z. B. durch Anklicken einer URL in einer E-Mail oder eines Buttons im Datenmanagement) Datensätze verändern, löschen, neue Datensätze generieren, oder E-Mails verschicken.

Startlink
Legen Sie hier fest, zu welcher Seite der Benutzer kommt, wenn er diese App in der App-Übersichtsseite anklickt.

Zugewiesene Daten
Hier können Sie festlegen, welche Bedingung erfüllt sein muß, damit ein Datensatz einem Benutzer zugewiesen wird.

Uploads
In *Uploads* können Sie den Zugriff auf die hochgeladenen Dateien konfigurieren.

Aktivitäten
In *Aktivitäten* können Sie konfigurieren, ob für diese App bei Ereignissen (Datensatzänderungen, versendete E-Mails, Chat-Nachrichten) Aktivitäten für beteiligte Benutzer angelegt werden sollen.

Account
Die Funktionen unter *Account* sind nicht app-spezifisch.

- ***Login-Token***
  In *Login-Token* können Sie einen alternativen Login-Mechanismus aktivieren.
- ***App-Kategorien***
  In *App-Kategorien* können Sie hierarchische Kategorien anlegen, und Ihren Apps, zur besseren Übersicht, Kategorien zuweisen. Diese Kategorien tauchen auf der App-Übersichtsseite als Navigation auf.
- ***App-Zuordnungen***
  App-Zuordnungen zeigen die Zuordnungen Ihrer Apps zu den unter *App-Kategorien* aufgelisteten Kategorien.
- ***Kalender***
  Hier können Sie konfigurieren, wie die Termine der Arbeitsaufgaben in Ihren Apps publiziert werden. Dies ermöglicht es Ihnen diese Termine mit Ihrem Kalender-Program zu synchronisieren (z.B. „Outlook Express“ (auf Windows) oder „Kalender“ (auf Mac OS X)).
- ***SMTP-Server***
  Unter *SMTP-Server* können Sie den Zugang zu einem eigenen E-Mail-Server für E-Mails einrichten, die über LivingApps versendet werden. Dieser E-Mail-Server kann dann bei Versendungen und E-Mail-Templates verwendet werden.
- ***Installationen***
  Hier werden alle unter diesem Account durchgeführten Installtionen aufgeführt.
- ***Template-Libraries***
  Hier finden Sie eine Auswahl an Hilfstemplates, die Sie kopieren und für Ihre Apps einsetzen können.

Meine Living-Apps
Hier finden Sie eine Übersicht und alle Konfigurationsmöglichkeiten der Apps, bei denen Sie Admin-Berechtigung haben.

Living-Apps
: ist eine Liste aller Apps, auf die der Benutzer Zugriff hat. Diese Maske bietet eine Kombination der Masken *Startlink*, *Uploads*, *Aktivitäten*, und *Zugewiesene Daten*. Außerdem kann man die Kennung, ob im Datenmanagement die *Reihenfolge* bestehen bleiben soll, hier ändern.

Alle Anzeige-Templates
: Die Anzeige-Templates aller Apps, die ich administriere.

Alle Internen Templates
: Die Internen Templates aller Apps, die ich administriere.

Alle E-Mail-Templates
: Die E-Mail-Templates aller Apps, die ich administriere.

Alle Formular-Templates
: Die Formular-Templates aller Apps, die ich administriere.

Alle Update-Templates
: Die Update-Templates aller Apps, die ich administriere.

Alle Parameter
: Die konfigurierten Parameter aller Apps, die ich administriere.

Alle Links
: Die erstellten Links aller Apps, die ich administriere.

Farben (Datenmanagement)
: Die Farbkonfigurationen aller Apps, die ich administriere.

Alle Felder (Datenmanagement)
: Die Felderkonfigurationen aller Apps, die ich administriere.

Sortierung (Datenmanagement)
: Die Sortierkonfigurationen aller Apps, die ich administriere.

Alle Ansichtssichtbarkeiten
: Die Ansichtssichtbarkeiten aller Apps, die ich administriere.

Alle Aktionen
: Die Aktionen aller Apps, die ich administriere.

Alle Daten-Auswertungen
: Die Daten-Auswertungen aller Apps, die ich administriere.

2.1.2 Felder und deren Identifizierer

Die Daten einer App werden in *Feldern* mit unterschiedlichen Feldtypen organisiert. Diese Felder legen Sie an wenn Sie in Ihrer App unter *Konfiguration* → *Eingabe* ein Formular erstellen. In den erweiterten Funktionen unter dem Punkt *Felder* können Sie die *Bezeichnung*, den *Identifizierer*, den *Typ*, sowie das *Ziel* der Felder Ihrer App einsehen.

Jedes der Felder hat eine bestimmte *Bezeichnung*, die Sie selbst wählen. Da diese Bezeichnung Sonder- und Leerzeichen enthalten oder mehrfach vergeben werden kann, wird zusätzlich für jedes Feld ein eindeutiger *Identifizierer* erstellt. Wenn Sie in einer der erweiterten Funktionen auf die Daten in Ihren Feldern zugreifen möchten, benötigten Sie diese Identifizierer.

Ist Ihr Feld vom Typ `lookup` bzw. `applookup` finden Sie unter *Ziel* die *entsprechende Auswahl* bzw. die *verknüpfte App* mit der entsprechenden Auswahl.

Zum *Typ* können Sie im Kapitel *Übersicht der Feldtypen* mehr lesen.

Feld-Information

Anmeldung

Bezeichnung	Identifizierer	Typ	Ziel
bezahlt	`bezahlt`	`bool`	
bezahlt am	`bezahlt_am`	`date/date`	
Teilnahmegebühr	`teilnahmegebuehr`	`number`	
abgemeldet	`abgemeldet`	`bool`	
Zimmer sind gebucht	`zimmer_sind_gebucht`	`bool`	
Veranstaltung	`veranstaltung2`	`applookup/select`	Veranstaltungen
Vorname	`vorname`	`string/text`	
Nachname	`nachname`	`string/text`	
Straße Hsnr.	`strasse_und_hsnr2`	`string/text`	
PLZ Ort	`plz_ort`	`string/text`	
E-Mail-Adresse	`e_mail_adresse`	`string/email`	
Telefon	`telefon`	`string/tel`	
Anzahl Personen	`anzahl_personen2`	`number`	
Wird Übernachtung benötigt?	`wird_uebernachtung_benoetigt`	`lookup/radio`	Auswahl
Einzelzimmer	`einzelzimmer`	`number`	
Euro / Nacht	`euro_nacht`	`number`	
Doppelzimmer	`mehrbettzimmer`	`number`	
Euro / Nacht	`euro_nacht2`	`number`	

Abb. 2: Feld-Information

Feld-Information

Anmeldung

Bezeichnung	Identifizierer	Typ
Wird Übernachtung benötigt?	`wird_uebernachtung_benoetigt`	`lookup/radio`

↓

Auswahl

Bezeichnung	Identifizierer
Ja	`ja`
Nein	`nein`

Abb. 3: Feld-Information Ziel - lookup

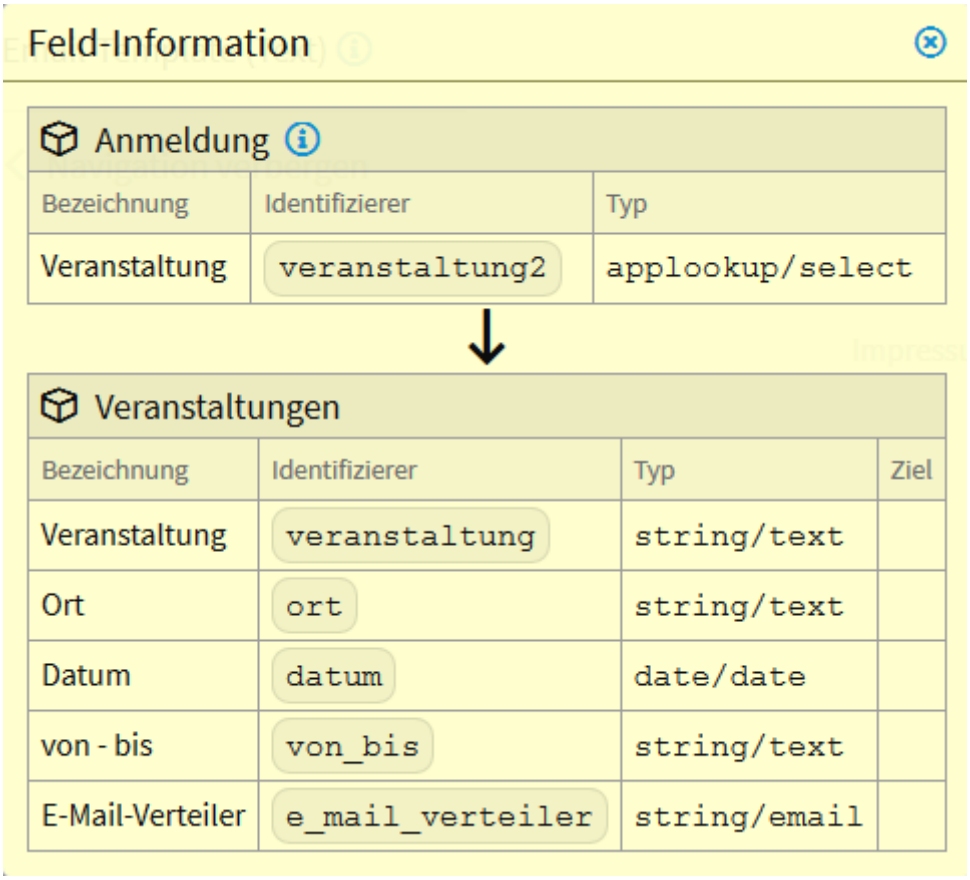


Abb. 4: Feld-Information Ziel - applookup

2.1.3 Anwendungsbeispiel

Die erweiterten Funktionen von LivingApps werden im Folgenden anhand der Beispiel-LivingApp „Anmeldung“ veranschaulicht. Über das *Formular* der App soll man sich zu Veranstaltungen anmelden und gleichzeitig Angaben zu Übernachtungen machen können. Die anmeldende Person soll nach der Anmeldung automatisch über die erfolgreiche Anmeldung informiert werden und gleichzeitig eine Bestätigung per E-Mail erhalten, in der nochmal alle gemachten Angaben aufgelistet sind. Außerdem soll das Anzeigetemplate „Teilnehmerliste“ erstellt werden, das alle Teilnehmer und die jeweils gewählte Veranstaltung auflistet.

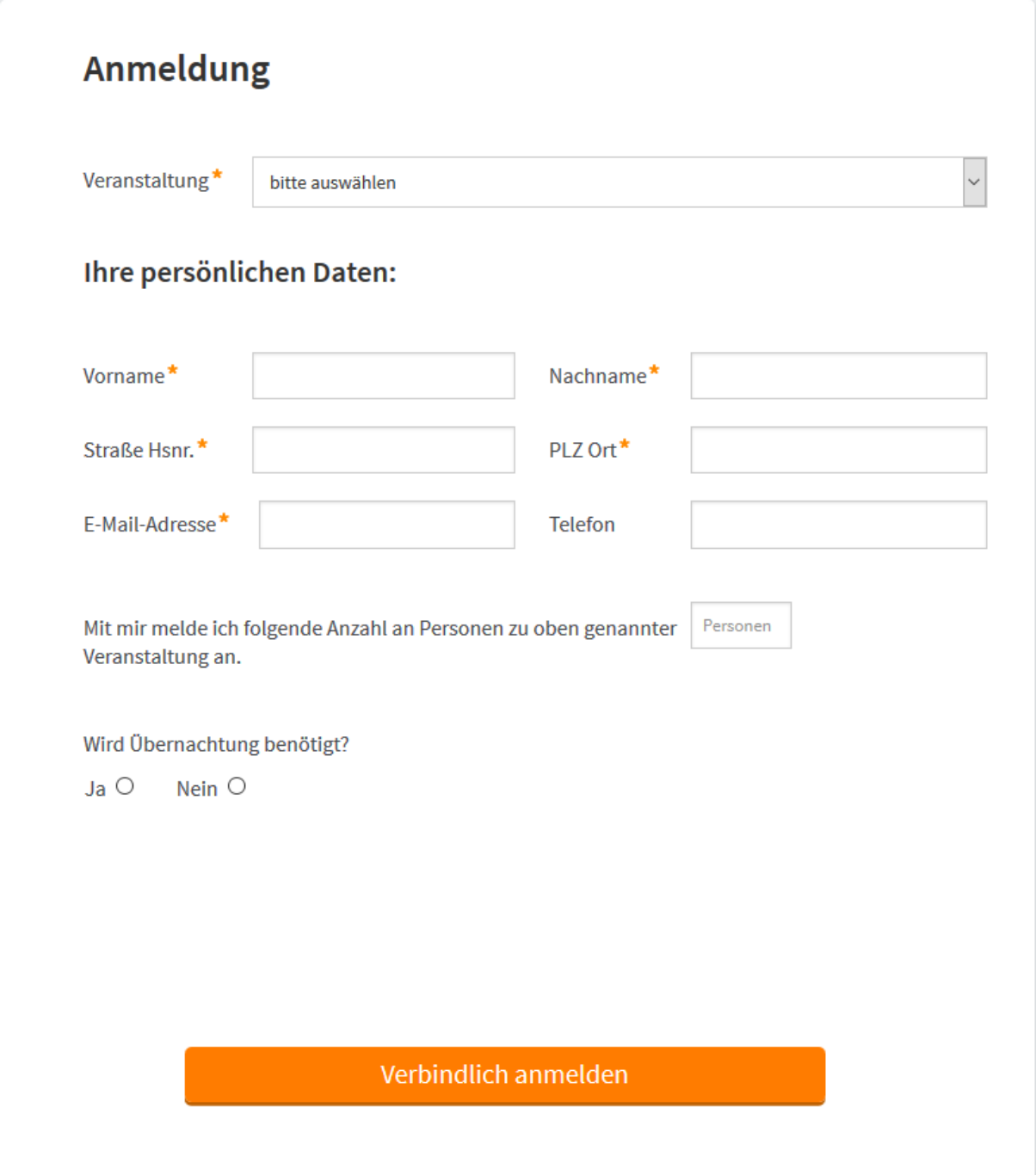


Abb. 5: Anmeldeformular

Für die folgenden Kapitel ist es notwendig, dass die *Identifizierer der Felder* des Formulars bekannt sind. Wie bereits erwähnt, können diese durch Klicken auf *Felder* eingesehen werden. Hier können auch die Typen der Felder abgelesen werden. Die *Veranstaltung*-Auswahlbox hat z. B. den Identifizierer `veranstaltung2` und ist vom Typ `applookup/select`, das Feld zur Eingabe der *E-Mail-Adresse* hat den Identifizierer `e_mail_adresse` und

ist vom Typ `string/email` und das Feld *Einzelzimmer* hat den Identifizierer `einzelzimmer` und ist vom Typ `number`. Über Feldtypen mit Ihren Untertypen können Sie im Kapitel *Übersicht der Feldtypen* mehr lesen.

2.2 Übersicht der Feldtypen

Die Daten einer App werden in Feldern mit unterschiedlichen Feldtypen organisiert. Felder können bei der Erstellung einer neuen App, oder bei einer bereits bestehenden App unter *Konfiguration* → *Eingabe* im Formulardesigner angelegt werden. Der Feldtyp (`type`) legt grundsätzlich die Art des Inhaltes fest. Untertypen (`subtypes`) stellen entweder eine Einschränkung des erlaubten Feldtyps dar, oder legen fest, wie das Feld dem Benutzer präsentiert wird. Wird kein Untertyp gewählt, wird der Standardwert verwendet.

Folgende Feldtypen mit ihren Untertypen stehen zur Auswahl:

2.2.1 Feldtyp `string`

Feld für die Eingabe von Text, E-Mail-Adressen, Internet-Adressen, Telefonnummern.

Untertyp `text` (Standard)

Feld für die Eingabe von Text (bis zu 4000 Zeichen) Z. B. Name, Straße, Ort, Beschreibung, usw. Wird im *Formulardesigner* angelegt durch Auswahl von *Eingabefeld (einzeilig)* bei *Formularelemente* und *Text* bei *Allgemein* → *Art des Inhalts*. So sieht ein `string/text`-Feld in der *Dateneingabe* aus.

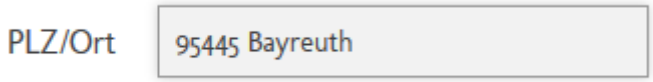


Abb. 6: `string/text`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `string/text`-Feldes in Templates ausgeben:

```
<?printx record.fields.identifier.value?>
```

Shortcut:

```
<?printx record.v_identifier?>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht.

Untertyp `textarea`

Feld für die Eingabe von mehrzeiligem Text (unbegrenzt). Z. B. für ausführliche Beschreibungen, Kommentare, Notizen, usw. Wird im *Formulardesigner* angelegt als *Textfeld (mehrzeilig)*. So sieht ein `string/textarea`-Feld in der *Dateneingabe* aus.

Sie haben die Möglichkeit, den Inhalt Ihres mehrzeiligen Textfeldes zu verschlüsseln. Wählen sie dafür im *Formulardesigner* unter *Feldinhalt verschlüsseln…* die entsprechende Option aus.

Nicht anwenden
: Der Inhalt wird nicht verschlüsselt.

Erzwingen
: Der Inhalt muss beim Speichern mit einem Passwort verschlüsselt werden.

Optional
: Der Inhalt kann nach dem Speichern mit einem Passwort verschlüsselt werden.

Abb. 7: `string/textarea`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `string/textarea`-Feldes in Templates ausgeben:

```
<?printx record.fields.identifier.value?>
```

Shortcut:

```
<?printx record.v_identifier?>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht.

Untertyp `email`

Textfeld für die Eingabe von E-Mail-Adressen, z. B. `max.mustermann@beispiel.com`. Die Eingabe der E-Mail-Adresse wird auf Gültigkeit überprüft. Wird im *Formulardesigner* angelegt als *Eingabefeld (einzeilig)* bei *Formularelemente* und *E-Mail* bei *Allgemein* → *Art des Inhalts*. Ein E-Mail-Feld ist z. B. notwendig für E-Mail-Versendungen aus dem Datenmanagement oder für Bestätigungs-Mails über E-Mail-Templates. So sieht ein `string/email`-Feld in der *Dateneingabe* aus.

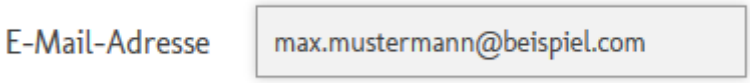


Abb. 8: `string/email`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `string/email`-Feldes in Templates als Link ausgeben:

```
<a href="mailto:<?printx record.fields.identifier.value?>">
   <?printx record.fields.identifier.value?>
</a>
```

Shortcut:

```
<a href="mailto:<?printx record.v_identifier?>">
   <?printx record.v_identifier?>
</a>
```

Wobei `identifier` für den *Identifizierer des E-Mail-Feldes* steht. Beim Anklicken des Links im Template öffnet sich Ihr E-Mail-Programm.

Untertyp `url`

Textfeld für die Eingabe von Internet-Adressen, z. B. `http://www.livinglogic.de/`. Die Eingabe der URL wird auf Gültigkeit überprüft. Wird im *Formulardesigner* angelegt als *Eingabefeld (einzeilig)* bei *Formularelemente* und *URL* bei *Allgemein* → *Art des Inhalts*. So könnten z. B. bei Recherche-Arbeiten im Internet besonders interessante Links in URL-Feldern hinterlegt werden. Ansicht eines `string/url`-Feldes in der *Dateneingabe*.

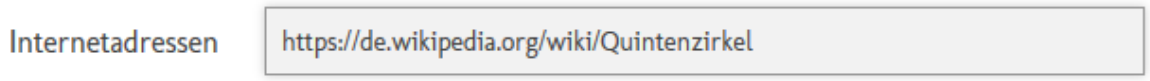


Abb. 9: `string/url`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `string/url`-Feldes in Templates als Link ausgeben:

```
<a href="<?printx record.fields.identifier.value?>">
   <?printx record.fields.identifier.value?>
</a>
```

Shortcut:

```
<a href="<?printx record.v_identifier?>">
   <?printx record.v_identifier?>
</a>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht.

Untertyp `password`

Passwortfeld - momentan nicht aktiv

Untertyp `tel`

Textfeld für die Eingabe von Telefonnummern. Die Eingabe wird auf Gültigkeit überprüft. Erlaubt sind Zahlen von `0`-`9`, `/`, Leerzeichen und `+`. Wird im *Formulardesigner* angelegt als *Eingabefeld (einzeilig)* bei *Formularelemente* und *Telefonnummer* bei *Allgemein* → *Art des Inhalts*. So sieht ein `string/tel`-Feld in der *Dateneingabe* aus.

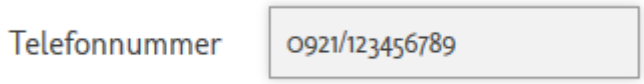


Abb. 10: `string/tel`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `string/tel`-Feldes in Templates als Link ausgeben:

```
<a href="tel:<?printx record.fields.identifier.value?>">
   <?printx record.fields.identifier.value?>
</a>
```

Shortcut:

```
<a href="tel:<?printx record.v_identifier?>">
   <?printx record.v_identifier?>
</a>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht. Beim Anklicken des Links kann die Telefonnummer direkt angewählt werden (bei App auf dem Handy).

2.2.2 Feldtyp `int`

Textfeld mit Ganze-Zahlen-Validierung - momentan nicht aktiv

2.2.3 Feldtyp `number`

Textfeld für die Eingabe von Zahlen, z. B. `12345,6`. Die Eingabe wird auf Gültigkeit überprüft. Wird im *Formulardesigner* angelegt als *Eingabefeld (einzeilig)* bei *Formularelemente* und *Zahl* bei *Allgemein → Art des Inhalts*. Sollen für ein Feld Summierungen oder sonstige Berechnungen erstellt werden, dann muss das Feld ein Zahlenfeld sein. Z. B. für Berechnungen der Summe oder des Durchschnitts in Daten-Auswertungen. So sieht ein `number`-Feld in der *Dateneingabe* aus.

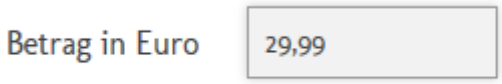


Abb. 11: `number`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `number`-Feldes in Templates ausgeben:

```
<?printx record.fields.identifier.value?>
```

Shortcut:

```
<?printx record.v_identifier?>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht.

Mit folgender Funktion werden die Zahlen mit 2 Nachkommastellen ausgegeben:

```
<?def formatfloat(v)?>
   <?if v is not None?>
      <?code v = str(v)?>
      <?if "." not in v?>
         <?code v += "."?>
      <?end if?>
      <?code v += "00"?>
      <?code v = v[:v.find(".")+3]?>
   <?end if?>
   <?return v.replace(".", ",")?>
<?end def?>
<?printx formatfloat(record.v_identifier)?>
```

2.2.4 Feldtyp `date`

Textfeld für die Eingabe von Datum und Uhrzeit. Eingabe wird auf Gültigkeit überprüft. Wird im *Formulardesigner* angelegt als *Eingabefeld (einzeilig)* bei *Formularelemente* und *Datum und Uhrzeit* bei *Allgemein → Art des Inhalts*. Bei einem Datumsfeld öffnet sich bei der Dateneingabe ein Kalender zur Auswahl von Datum und Uhrzeit. Datumsfelder sind sinnvoll um Zeiträume darzustellen, Zeiterfassungen zu machen, usw. So sieht ein Datumsfeld mit dem Untertyp `datetimeminute` in der *Dateneingabe* aus.

Folgende *Datumsformate* stehen zur Auswahl:

Abb. 12: `date/datetimeminute` in der Dateneingabe

Untertyp `date` (Standard)

Tag, Monat, Jahr (z. B. 29.02.2000)

Auf folgende Weise können Sie den Inhalt eines `date/date`-Feldes in Templates ausgeben:

```
<?if record.fields.identifier.value?>
   <?printx format(record.fields.identifier.value, "%d.%m.%Y", "de")?>
<?end if?>
```

Shortcut:

```
<?if record.v_identifier?>
   <?printx format(record.v_identifier, "%d.%m.%Y", "de")?>
<?end if?>
```

Untertyp `datetimeminute`

Tag, Monat, Jahr, Stunden, Minuten (z. B. 29.02.2000 15:30)

Auf folgende Weise können Sie den Inhalt eines `date/datetimeminute`-Feldes in Templates ausgeben:

```
<?if record.fields.identifier.value?>
   <?printx format(record.fields.identifier.value, "%d.%m.%Y %H:%M", "de")?>
<?end if?>
```

Shortcut:

```
<?if record.v_identifier?>
   <?printx format(record.v_identifier, "%d.%m.%Y %H:%M", "de")?>
<?end if?>
```

Untertyp `datetimesecond`

Tag, Monat, Jahr, Stunden, Minuten, Sekunden (z. B. 29.02.2000 15:30:45)

Auf folgende Weise können Sie den Inhalt eines `date/datetimesecond`-Feldes Templates ausgeben:

```
<?if record.fields.identifier.value?>
   <?printx format(record.fields.identifier.value, "%d.%m.%Y %H:%M:%S", "de")?>
<?end if?>
```

Shortcut:

```
<?if record.v_identifier?>
   <?printx format(record.v_identifier, "%d.%m.%Y %H:%M:%S", "de")?>
<?end if?>
```

Wobei `identifier` jeweils für den *Identifizierer des Feldes* steht.

2.2.5 Feldtyp `lookup`

Feld zur Auswahl einer Option aus einer Liste von Auswahloptionen.

Hier ist in der Dateneingabe die Auswahl maximal einer Option möglich.

Untertyp `select`

Auswahl-Box einfach (Standard)

Wird im *Fomulardesigner* angelegt als *Auswahlbox*. Hier können unter *Auswahlen* → *Optionen Standard* beliebig viele Auswahl-Optionen angelegt werden. Z. B. verschiedene Kategorien, denen Datensätze zugeordnet werden sollen. So sieht ein `lookup/select`-Feld in der *Dateneingabe* aus.

Abb. 13: `lookup/select`-Feld in der Dateneingabe

Untertyp `choice`

Dieser Untertyp ist sinnvoll für große Auswahlmengen. Allgemeine Beschreibung siehe Untertyp `select`. Zusätzlich muss hier bei *Auswahlen* → *Optimieren für große Auswahlmenge* das Häckchen gesetzt werden. In der Dateneingabe werden dann erst nachdem man Zeichen eingegeben hat, die passenden Auswahlen vorgeschlagen. Groß-/Kleinschreibung ist dabei von Bedeutung. So sieht ein `lookup/choice`-Feld in der *Dateneingabe* aus.

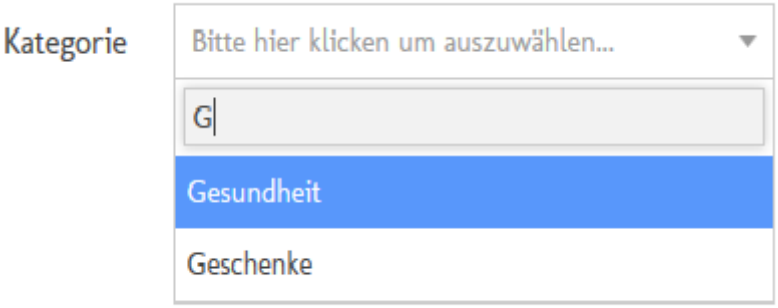


Abb. 14: `lookup/choice`-Feld in der Dateneingabe

Untertyp `radio`

Wird im *Fomulardesigner* angelegt als *Optionen*. Z. B. soll als Anrede `Herr` oder `Frau` gewählt werden. So sieht ein `lookup/radio`-Feld in der *Dateneingabe* aus.

Abb. 15: `lookup/radio`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `lookup`-Feldes in Templates ausgeben:

```
<?if record.fields.identifier.value?>
   <?printx record.fields.identifier.value.label?>
<?end if?>
```

Shortcut:

```
<?if record.v_identifier?>
   <?printx record.v_identifier.label?>
<?end if?>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht.

2.2.6 Feldtyp `applookup`

Dieser Typ ist ähnlich zum Typ `lookup` jedoch legen Sie hier die Auswahloptionen nicht beim Feld fest, sondern die Datensätze einer festgelegten, ausgewählten App werden als Auswahloptionen verwendet.

Hier ist in der Dateneingabe die Auswahl maximal einer Option möglich.

Untertyp `select`

Wird im *Fomulardesigner* angelegt als *Auswahlbox*. Hier kann unter *Auswahlen → Optionen einer App nutzen* eine App gewählt werden. Weiterhin können Sie festlegen welche Felder dieser App in der Auswahl angezeigt werden. Z. B. könnten einer App „Anmeldung“ die Daten der App „Veranstaltungen“ mit den Feldern „Veranstaltung“ und „Datum“ als Auswahl zur Verfügung gestellt werden. So sieht ein `applookup/select`-Feld in der *Dateneingabe* aus.

Untertyp `choice`

Dieser Untertyp ist sinnvoll für große Auswahlmengen. Allgemeine Beschreibung siehe Untertyp `select`. Zusätzlich muss hier bei *Auswahlen → Optimieren für große Auswahlmenge* das Häckchen gesetzt werden. In der Dateneingabe werden dann erst nachdem man Zeichen eingegeben hat, die passenden Auswahlen vorgeschlagen. Groß-/Kleinschreibung ist dabei von Bedeutung.

Auf folgende Weise können Sie den Inhalt eines `applookup`-Feldes in Anzeige- und internen Templates ausgeben:

```
<?code field = record.f_identifier?>
<?if field.value?>
   <?for reffieldidentifier in field.control.lookupcontrols?>
      <?code reffield = field.value.fields[reffieldidentifier]?>
      <?printx reffield.control.label?>: <?printx reffield.value?>
   <?end for?>
<?end if?>
```

Abb. 16: `applookup/select`-Feld in der Dateneingabe

Um alle anzuzeigenden Felder des zugeordneten Datensatzes auszugeben, muss man eine Schleife über diese Felder benutzen. Dabei steht `identifier` für den *Identifizierer des anzuzeigenden Feldes* und `reffieldidentifier` durchläuft die Feld-Identifizierer die für die Auswahl konfiguriert wurden. Im oben genannten Beispiel werden so die Inhalte der Felder „Veranstaltung“ und „Datum“ aus der verknüpften App „Veranstaltungen“ ausgegeben.

2.2.7 Feldtyp `multiplelookup`

Feld zur Auswahl mehrerer Optionen aus einer Liste von Auswahloptionen. Hier ist in der Dateneingabe die Auswahl mehrerer Optionen möglich. Dafür muss bei der Auswahl die `Strg`-Taste der Tastatur gedrückt werden.

Untertyp `select`

Auswahl-Box mehrfach (Standard)

Wird im *Fomulardesigner* angelegt als *Auswahlbox*. Hier können unter *Auswahlen* → *Optionen Standard* beliebig viele Auswahl-Optionen angelegt werden. Bei *Auswahlen* → *Mehrfachauswahl zulassen* muss das Häkchen gesetzt werden. Es können z. B. verschiedene Kategorien angelegt werden, denen Datensätze zugeordnet werden sollen. Ein Datensatz kann hier mehreren Kategorien zugeordnet werden. So sieht ein `multiplelookup/select`-Feld in der *Dateneingabe* aus.

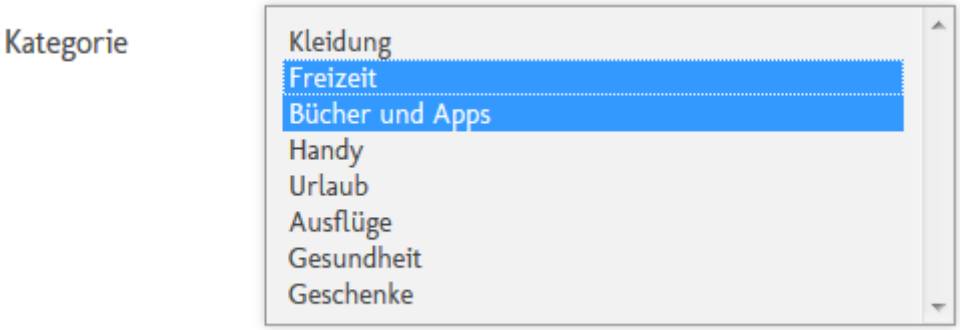


Abb. 17: `multiplelookup/select`-Feld in der Dateneingabe

Untertyp `choice`

Dieser Untertyp ist sinnvoll für große Auswahlmengen. Allgemeine Beschreibung siehe Untertyp `select`. Zusätzlich muss hier bei *Auswahlen → Optimieren für große Auswahlmenge* das Häckchen gesetzt werden. In der Dateneingabe werden dann erst nachdem man Zeichen eingegeben hat, die passenden Auswahlen vorgeschlagen. Groß-/Kleinschreibung ist dabei von Bedeutung. So sieht ein `multiplelookup/choice`-Feld in der *Dateneingabe* aus.

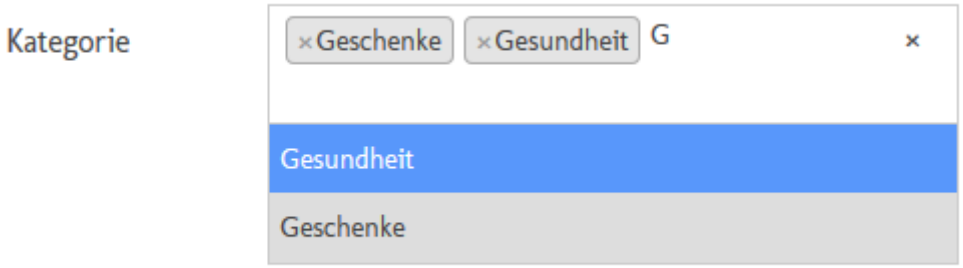


Abb. 18: `multiplelookup/choice`-Feld in der Dateneingabe

Untertyp `checkbox`

Wird im *Fomulardesigner* angelegt als *Optionen (mehrfach)*. Kann z. B. verwendet werden für Multiple Choice-Fragen. So sieht ein `multiplelookup/checkbox`-Feld in der *Dateneingabe* aus.

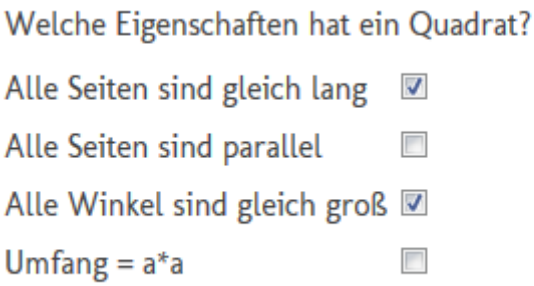


Abb. 19: `multiplelookup/checkbox`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `multiplelookup`-Feldes in Templates ausgeben:

```
<?for lookup in record.fields.identifier.value?>
   <?printx lookup.label?>
<?end for?>
```

Shortcut:

```
<?for lookup in record.v_identifier?>
   <?printx lookup.label?>
<?end for?>
```

Um alle zugeordneten Optionen anzuzeigen, muss man eine Schleife über der Wert des Feldes benutzen. Wobei `identifier` für den *Identifizierer des Feldes* steht.

2.2.8 Feldtyp `multipleapplookup`

Feld, das die Datensätze einer festgelegten, ausgewählten App zur Auswahl nimmt. Hier ist in der Dateneingabe die Auswahl mehrerer Optionen möglich. Dafür muss bei der Auswahl die `Strg`-Taste der Tastatur gedrückt werden.

Untertyp `select`

Wird im *Fomulardesigner* angelegt als *Auswahlbox*. Hier kann unter *Auswahlen* → *Optionen einer App nutzen* eine App gewählt werden. Und Sie können festlegen, welche Felder dieser App in der Auswahl angezeigt werden. Bei *Auswahlen* → *Mehrfachauswahl zulassen* muss das Häkchen gesetzt werden. Z. B. könnten hier für eine Veranstaltung in der App „Veranstaltungen“ mehrere zuständige Mitarbeiter aus der App „Mitarbeiter“ mit den Feldern „Nachname“, „Vorname“ und „E-Mail-Adresse“ ausgewählt werden. So sieht ein `multipleapplookup/select`-Feld in der *Dateneingabe* aus.

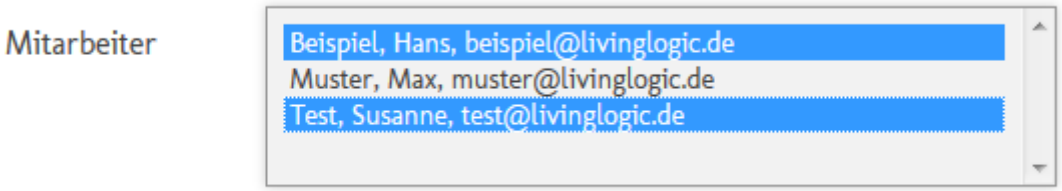


Abb. 20: `multipleapplookup/select`-Feld in der Dateneingabe

Untertyp `choice`

Dieser Untertyp ist sinnvoll für große Auswahlmengen. Allgemeine Beschreibung siehe Untertyp `select`. Zusätzlich muss hier bei *Auswahlen* → *Optimieren für große Auswahlmenge* das Häckchen gesetzt werden. In der Dateneingabe werden dann erst nachdem man Zeichen eingegeben hat, die passenden Auswahlen vorgeschlagen. Groß-/Kleinschreibung ist dabei von Bedeutung.

Auf folgende Weise können Sie den Inhalt eines `multipleapplookup`-Feldes in Templates ausgeben:

```
<?code field = record.f_identifier?>
<?for refrecord in field.value?>
   <?for reffieldidentifier in field.control.lookupcontrols?>
      <?code reffield = refrecord.fields[reffieldidentifier]?>
      <?printx reffield.control.label?>: <?printx reffield.value?>
   <?end for?>
<?end for?>
```

Um alle anzuzeigenden Felder der zugeordneten Datensätze auszugeben, muss man eine Schleife über den Wert dieses Feldes benutzen. Dabei steht `identifier` für den *Identifizierer des anzuzeigenden Feldes*, `refrecord` durchläuft die ausgewählten, verknüpften Datensätze und `reffieldidentifier` durchläuft die Felder, die für die Auswahl konfiguriert wurden. Im oben genannten Beispiel können so die Inhalte der Felder „Nachname“, „Vorname“ und „E-Mail-Adresse“ aus der verknüpften App „Mitarbeiter“ ausgegeben werden.

2.2.9 Feldtyp `bool`

Feld zur Eingabe und Anzeige von Ja/Nein-Werten (Boolesche Variable). Wird im *Formulardesigner* angelegt als *Häkchen*. `bool`-Felder werden verwendet um z. B. AGBs zu akzeptieren, Newsletter zu abonnieren, oder ähnliche Ja/Nein-Fragen zu beantworten. So sieht ein `bool`-Feld in der *Dateneingabe* aus.

Abb. 21: `bool`-Feld in der Dateneingabe

Auf folgende Weise können Sie den Inhalt eines `bool`-Feldes in Templates ausgeben:

```
<?if record.fields.identifier.value?>
   Ja
<?else?>
   Nein
<?end if?>
```

Shortcut:

```
<?if record.v_identifier?>
   Ja
<?else?>
   Nein
<?end if?>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht. Gibt „Ja“ aus, wenn das Häkchen gesetzt ist, ansonsten „Nein“.

2.2.10 Feldtyp `file`

Feld für Datei-Uploads. Wird im Fomulardesigner angelegt als *Datei-Upload*. Hier können Sie Dateien sämtlicher Formate, Bilder, Videos, Audio-Dateien, usw. hochladen. Kann z. B. verwendet werden, um in Online-Bewerbungen das Anschreiben, Lebenslauf, Zeugnisse, usw. hochzuladen. So sieht ein `file`-Feld in der *Dateneingabe* aus.

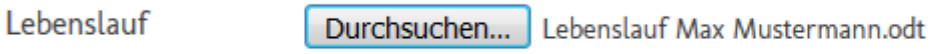


Abb. 22: `file`-Feld in der Dateneingabe

Untertyp `signature`

Hier können Sie direkt in der Dateneingabe mit der Maus im grauen Feld unterschreiben und diese Unterschrift dann in Templates einfügen. So sieht ein `file/signature`-Feld in der *Dateneingabe* aus.

Auf folgende Weise können Sie den Inhalt eines `file`-Feldes entweder als `<img/>`-Element oder als Download-Link in Templates ausgeben:

```
<?code file = record.v_identifier?>
<?if file and file.mimetype and file.mimetype.startswith("image/")?>
   <img style="max-width: 150px; max-height: 150px" src="<?printx file.url?>"/>
```

(Fortsetzung auf der nächsten Seite)

Abb. 23: `file/signature`-Feld in der Dateneingabe

(Fortsetzung der vorherigen Seite)

```
   <?printx file.filename?>
<?elif file?>
   <a target="_top" href="<?printx file.url?>"><?printx file.filename?></a>
<?end if?>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht. Bei `style` können Sie festlegen, in welcher Größe das Bild im Template angezeigt werden soll.

2.2.11 Feldtyp `geo`

Feld zur Erfassung von Adressen oder Geo-Koordinaten. Wird im *Fomulardesigner* angelegt als *Geodaten*. Hier können Sie unter *Allgemein* → *Dateneingabe* wählen zwischen *Adresse/Straße* (Standard) oder *Koordinaten*. Kann z. B. verwendet werden um fotografisch lohnenswerte Orte zu sammeln und zu katalogisieren. So sieht ein `geo`-Feld in der *Dateneingabe* aus.

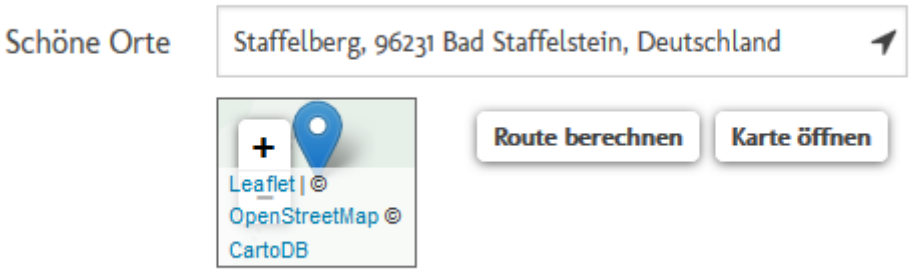


Abb. 24: `geo`-Feld in der Dateneingabe

Folgende Konfigurationen können Sie zusätzlich bei einem `geo`-Feld vornehmen:

Kartenausschnitt anzeigen
: Wird hier das Häkchen gesetzt, wird bei der Dateneingabe ein Kartenausschnitt angezeigt.

Kartenanzeige im Umkreis von
: Der Kartenausschnitt wird im Umkreis von der gewählten Option angezeigt.

Route berechnen anzeigen
: Wird hier das Häkchen gesetzt, kann bei der Dateneingabe eine Route berechnet werden.

Karte öffnen anzeigen
: Wird hier das Häkchen gesetzt, kann die Karte z. B. in google maps geöffnet werden.

Automatisch füllen mit…
: Das Feld wird automatisch mit *Aktuellem Standort* gefüllt.

Auf folgende Weise können Sie den Inhalt eines `geo`-Feldes als Link in Templates ausgeben:

```
<?if record.v_identifier?>
   <a href="https://www.google.de/maps?q=<?printx record.v_identifier.lat?>,<?printx␣
↪record.v_identifier.long?>">
      <?printx record.v_identifier.info?>
   </a>
<?end if?>
```

Wobei `identifier` für den *Identifizierer des Feldes* steht. Wird der Link im Template angeklickt, öffnet sich Google Maps mit dem ausgewählten Standort.

2.3 Maximale Feldanzahl

Pro App kann es nur eine bestimmte maximale Anzahl von Feldern eines bestimmten Typs geben. Aktuell gibt es folgende Beschränkungen:

Typen	Maximalanzahl
`string/textarea`, `string/html`	100
Sonstige `string`-Felder, `lookup`, `geo`	200
`applookup`	20
`multipleapplookup`	20
`file`	20
`bool`, `int`	100
`date`	50
`number`	100
`multiplelookup`	160

2.4 URLs für Formulare

2.4.1 Systemparameter

Mit den folgenden Systemparametern im Request können Aussehen und Verhalten des Formular-Templates gesteuert werden:

`la-editheader`
: Wenn dieser Parameter nicht vorhanden ist oder den Wert *true* hat, dann wird anhand der Benutzer-Berechtigungen geprüft, ob der User auf der Formularseite die Überschriften und das Bild ändern kann. Bei jedem anderen Wert wird die Editiermöglichkeit nicht angeboten.

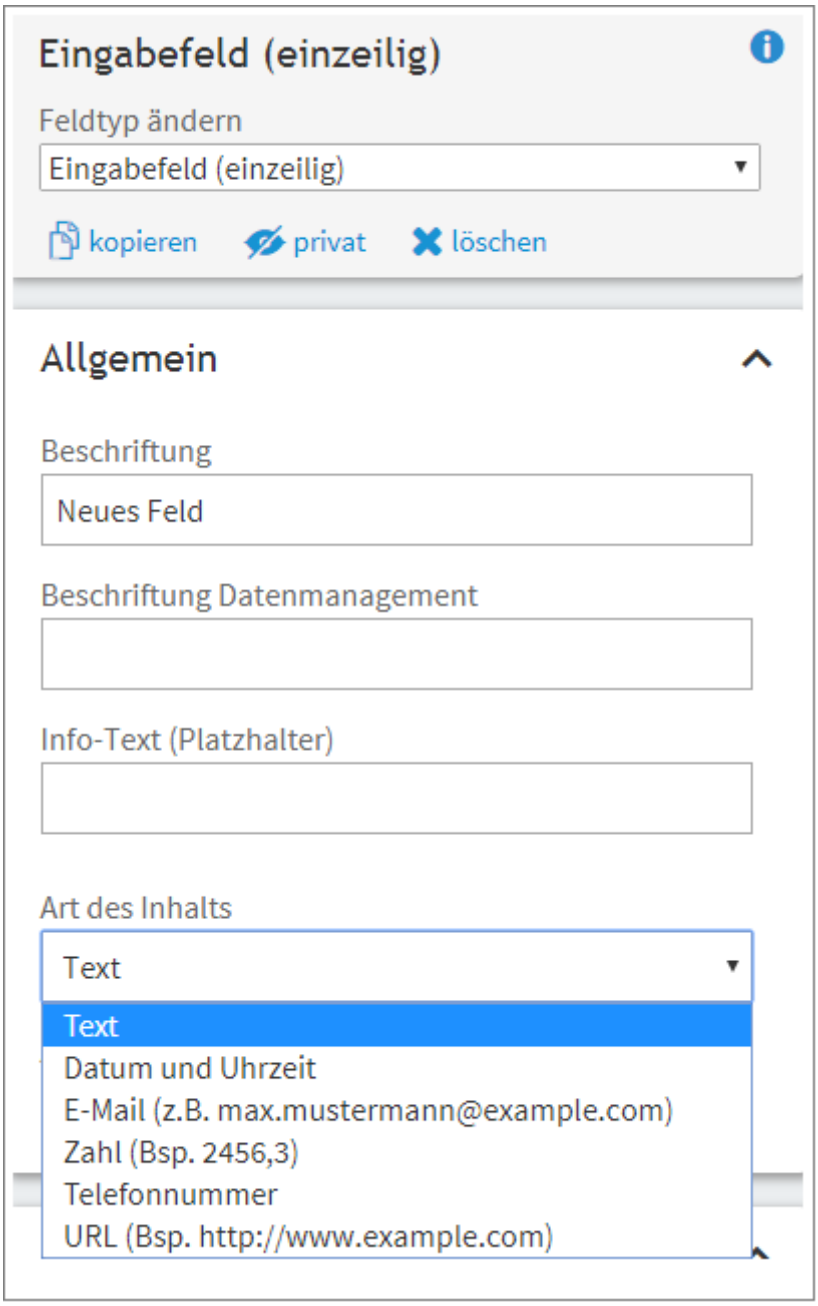


Abb. 25: Eingabefeld (einzeilig)

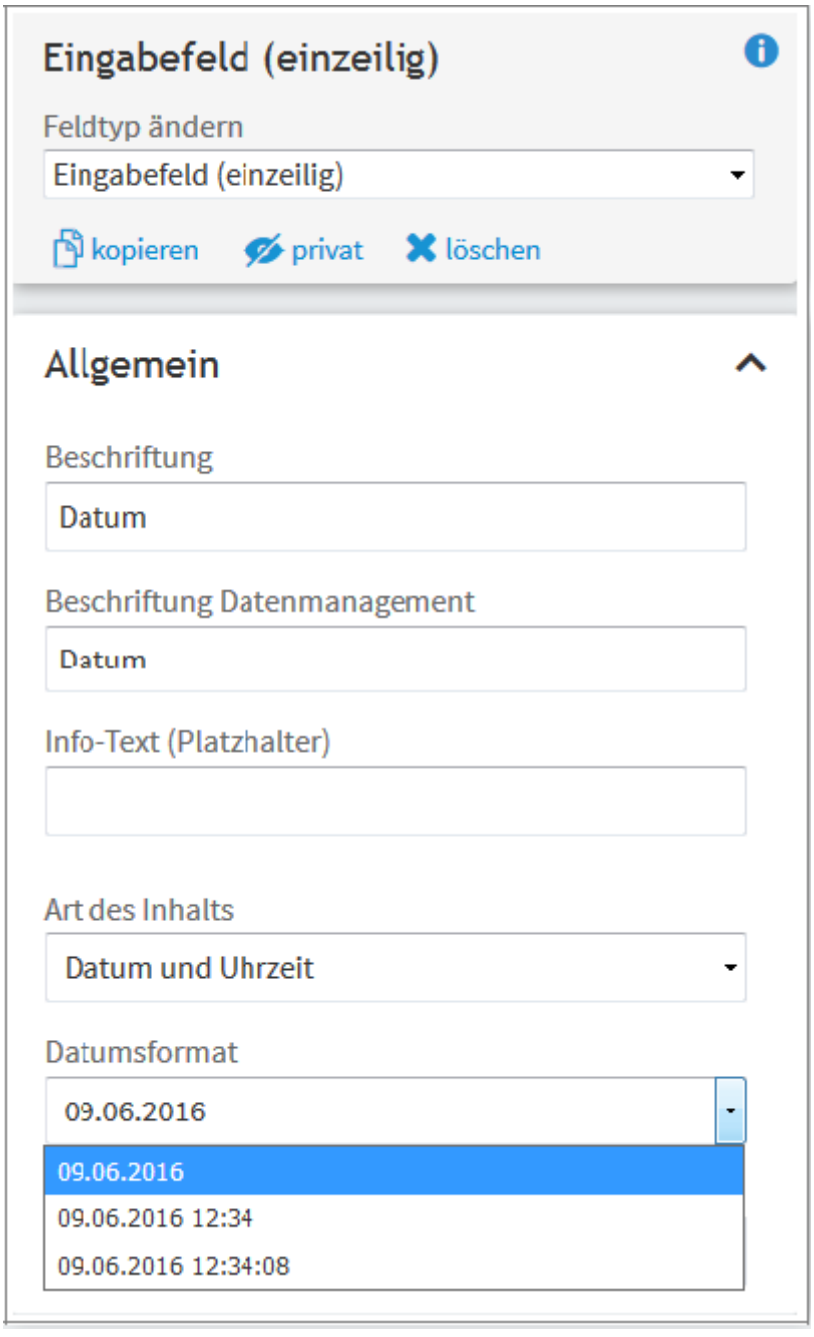


Abb. 26: Eingabefeld (einzeilig)- Datumsformat

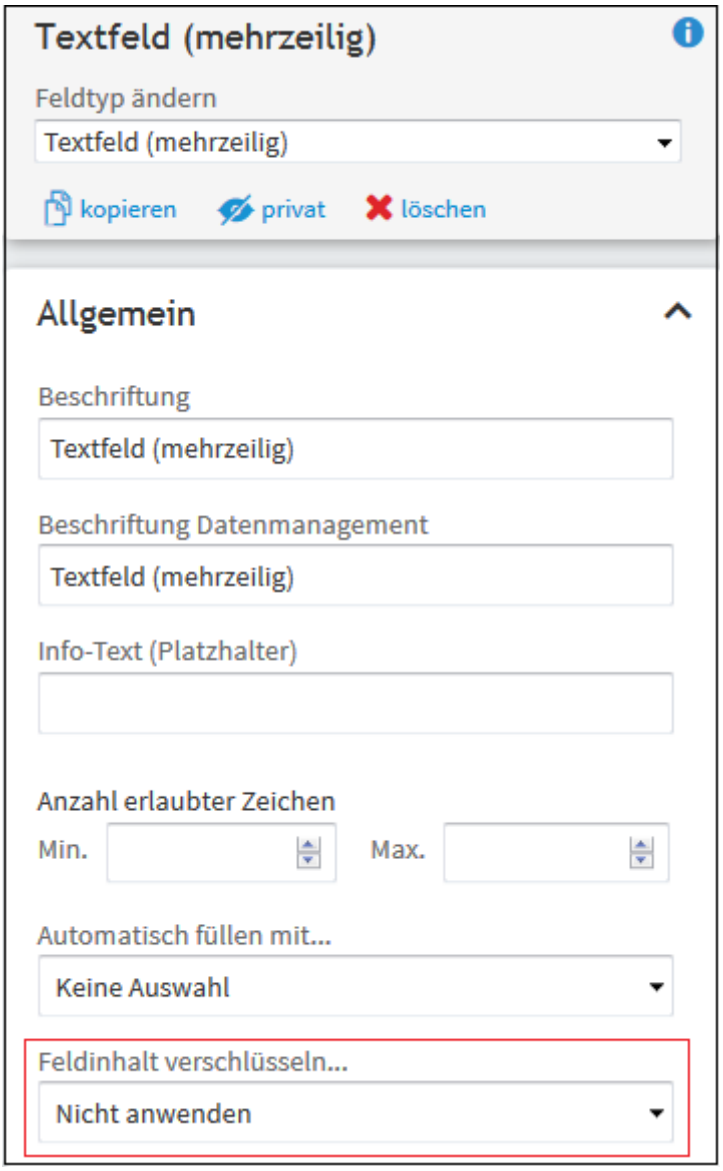


Abb. 27: Textfeld (mehrzeilig)

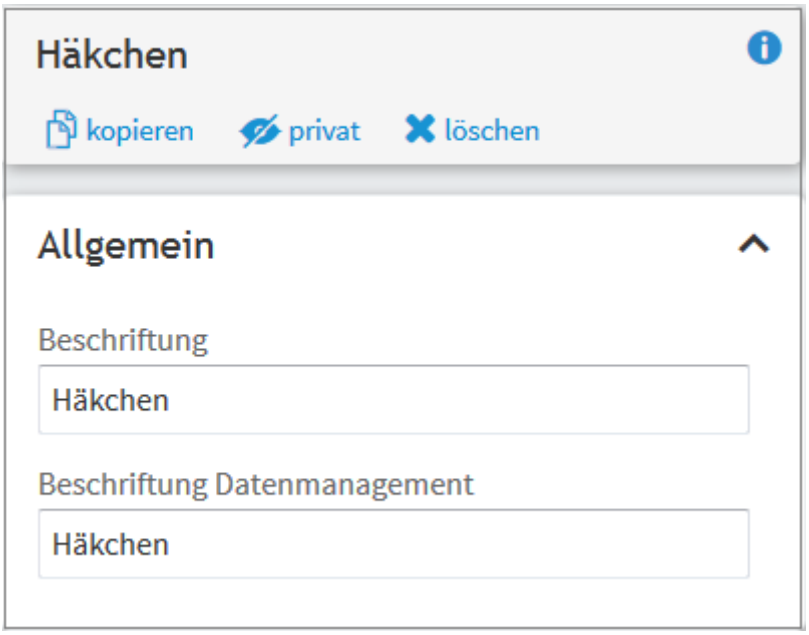


Abb. 28: Häkchen

2.5 Anzeige-Templates

In *Anzeige-Templates* können Sie beliebig viele sogenannte „UL4-Templates“ anlegen. In diesen Templates können Sie auf die vorhandenen Datensätze zugreifen und mit den Daten arbeiten, sie analysieren und ausgeben. Beispielsweise können die Daten mit Anzeige-Templates nach bestimmten Kriterien ausgewertet und die Ergebnisse in tabellarischer Form dargestellt werden.

Je nach Funktion Ihrer LivingApp sind die Anwendungsmöglichkeiten sehr vielfältig. Anzeige-Templates können sowohl für einfache Ausgaben von Datensätzen dienen, als auch alle möglichen Auswertungen der Daten vornehmen.

Anzeige-Templates sind also für alles nützlich, was mit Datenanalyse, -Übersicht und -Ausgabe zu tun hat.

2.5.1 Erstellung eines Anzeige-Templates

Um ein Anzeige-Template für Ihre LivingApp zu erstellen, wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Anzeige-Templates* und dann auf *Hinzufügen*.

Im nun geöffneten Fenster sehen Sie die *Eingabeansicht* des Anzeige-Templates.

Identifizierer
: Der Identifizierer ist eine eindeutige Kennung für das Template. Er kann verwendet werden, um das Template abzurufen. Der Identifizierer darf nur Buchstaben, Ziffern und _ enthalten.

Quelltext
: Hier erfolgt die *Formulierung des Templates*.

Wird im Quelltext ein <?whitespace?>-Tag oder ein <?doc?>-Tag eingegeben, dann erscheint unter *Templateeigenschaften* folgendes:

Whitespace
: *Whitespace* konfiguriert, wie Einrückungen/Zeilenvorschübe im Template-Quelltext behandelt werden (wird automatisch aus dem <?whitespace?>-Tag im Template-Quelltext extrahiert).

Beschreibung
: Beschreibung des Templates (aus dem <?doc?>-Tag im Template-Quelltext extrahiert)

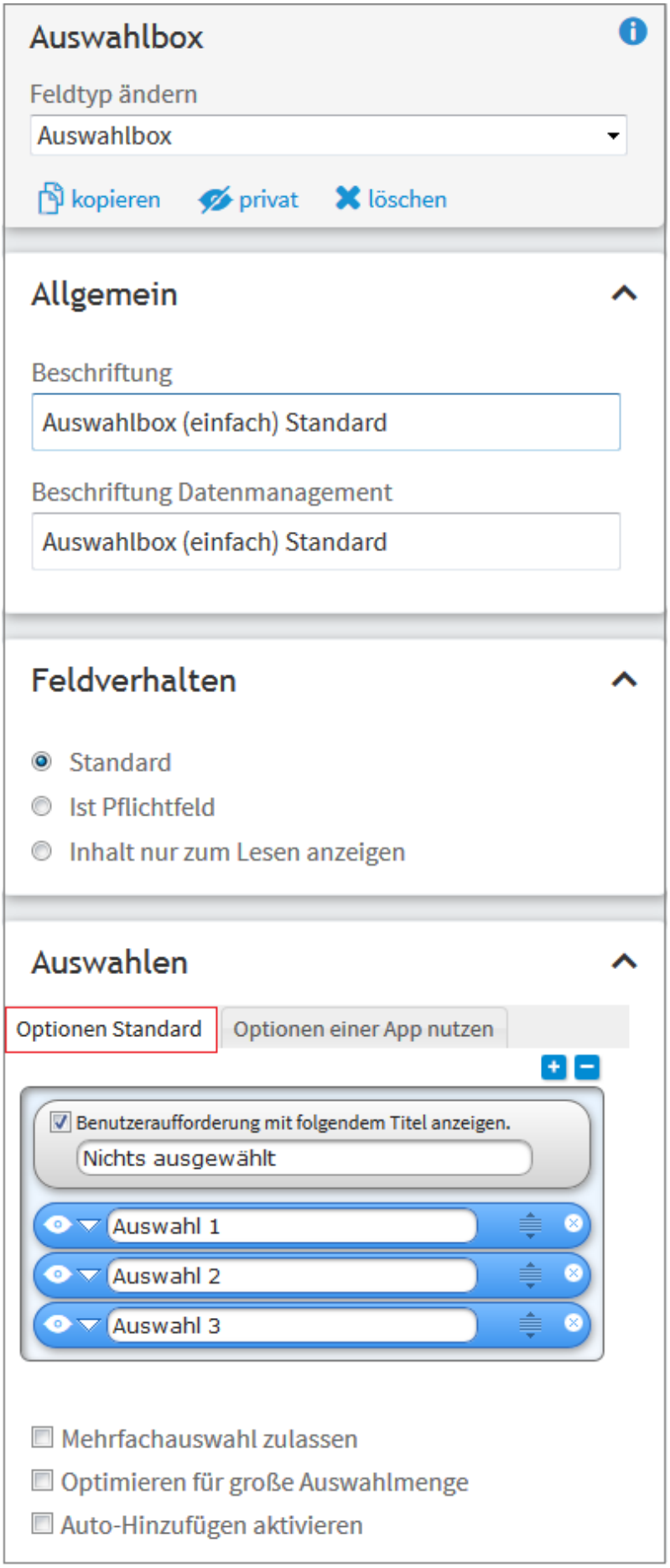


Abb. 29: Auswahlbox (einfach) - Optionen Standard

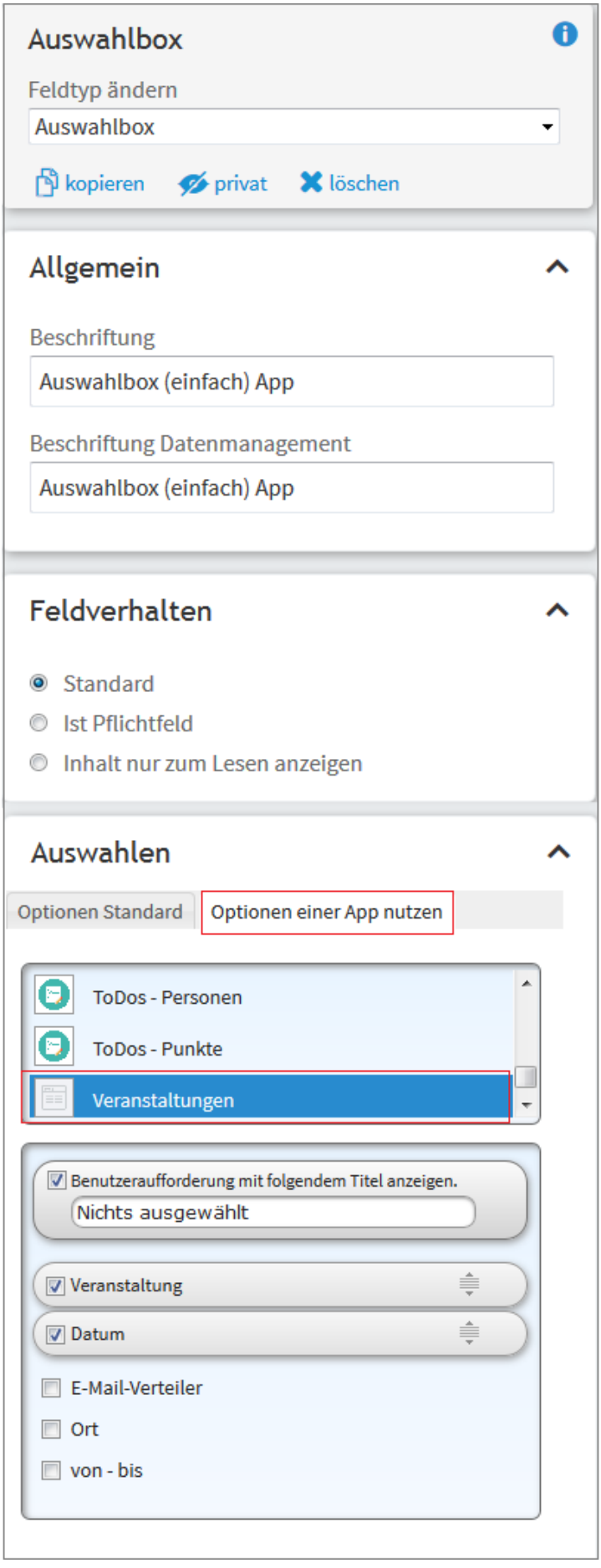


Abb. 30: Auswahlbox (einfach) - Optionen einer App nutzen

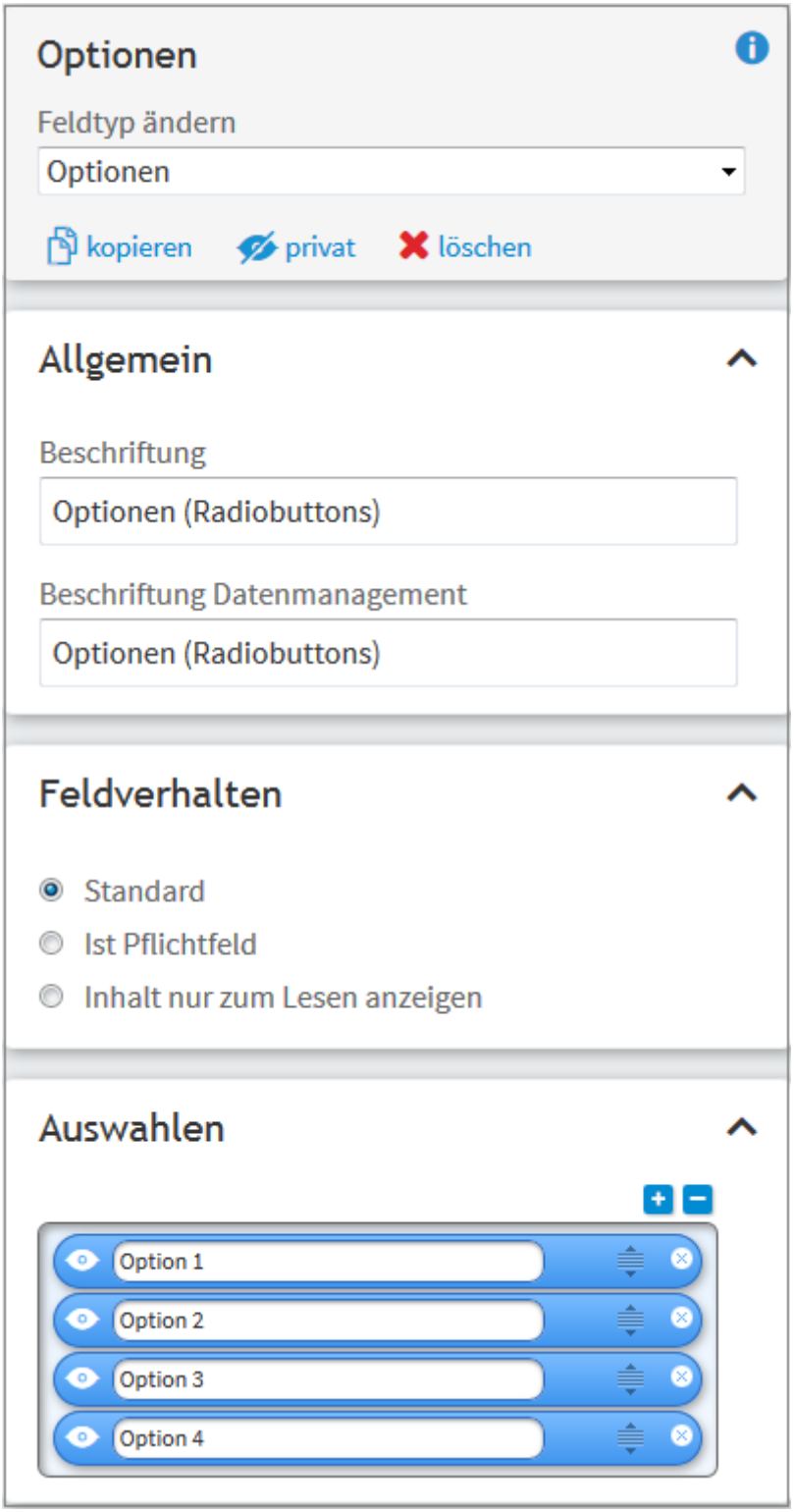


Abb. 31: Optionen einfach (Radiobuttons)

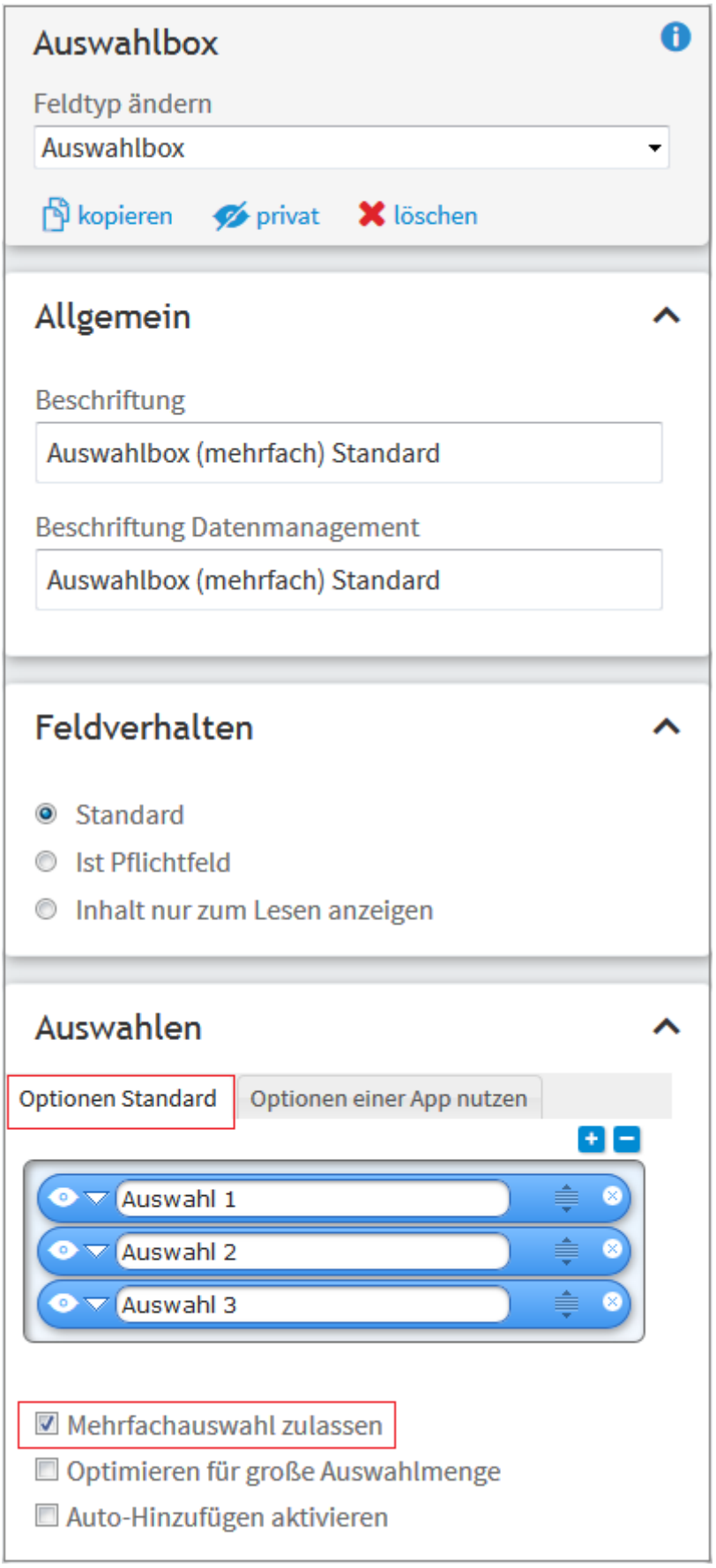


Abb. 32: Auswahlbox (mehrfach) - Optionen Standard

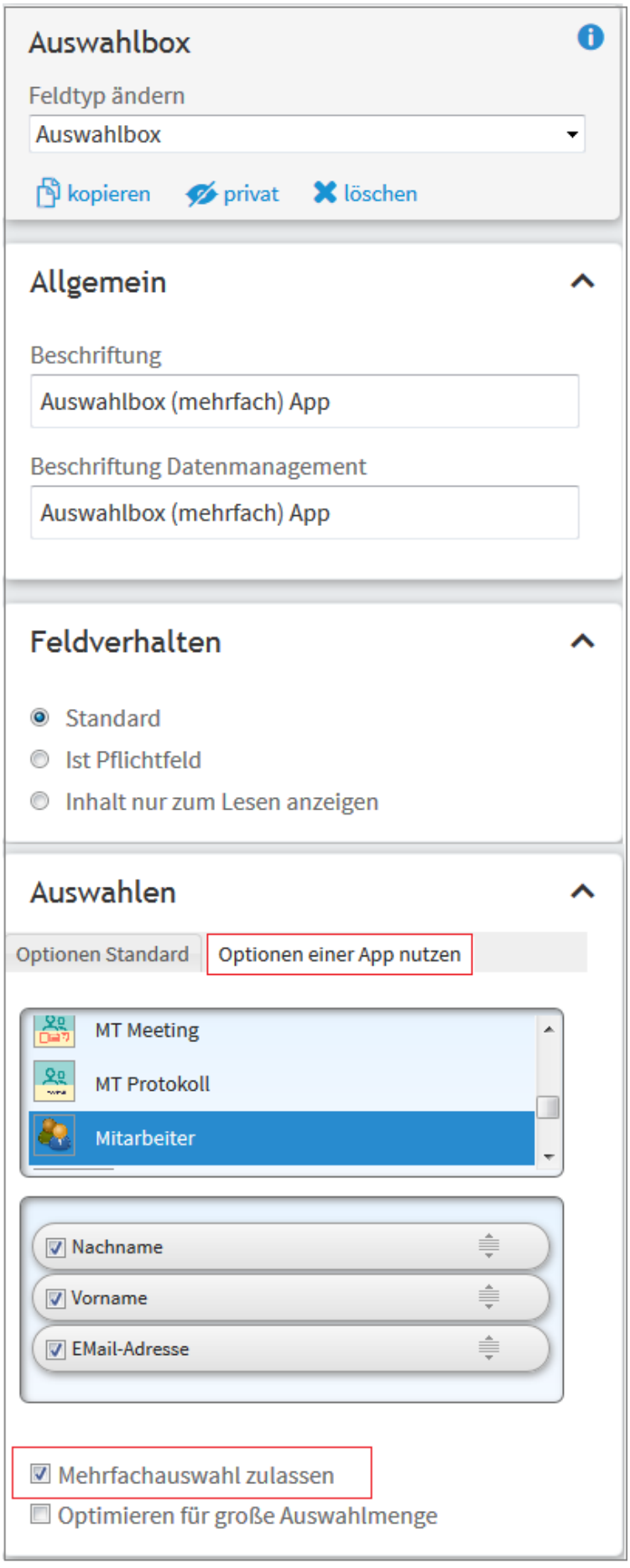


Abb. 33: Auswahlbox (mehrfach) - Optionen einer App nutzen

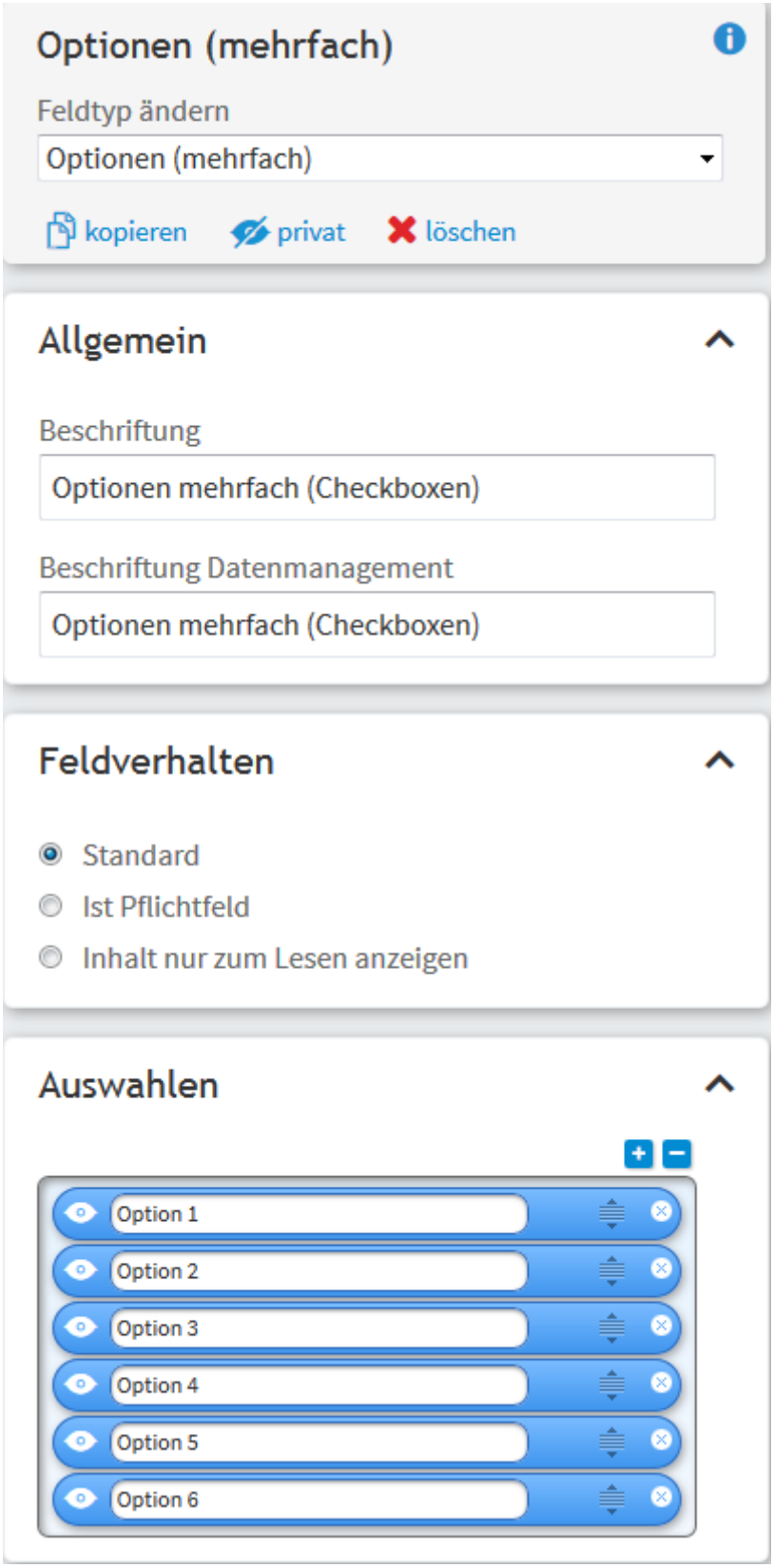


Abb. 34: Optionen mehrfach (Checkboxen)

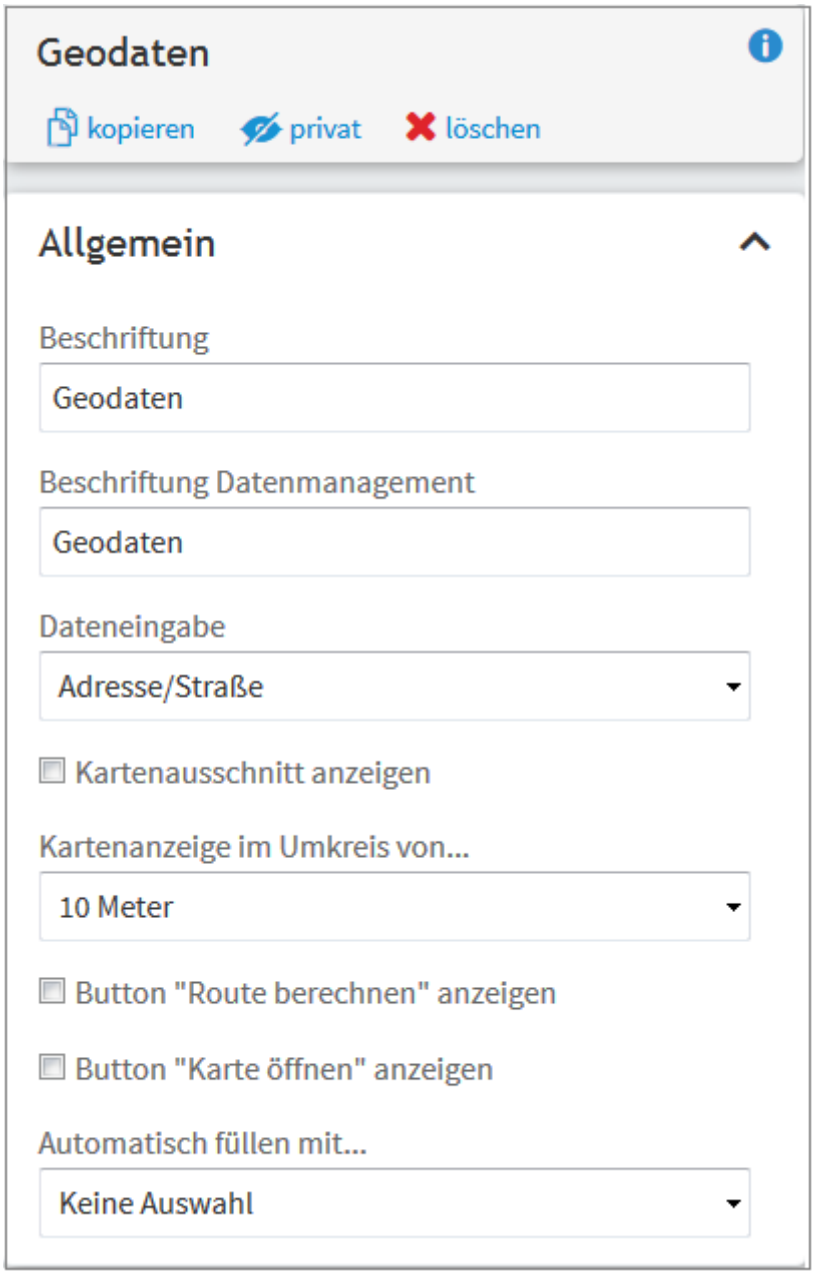


Abb. 35: Geodaten

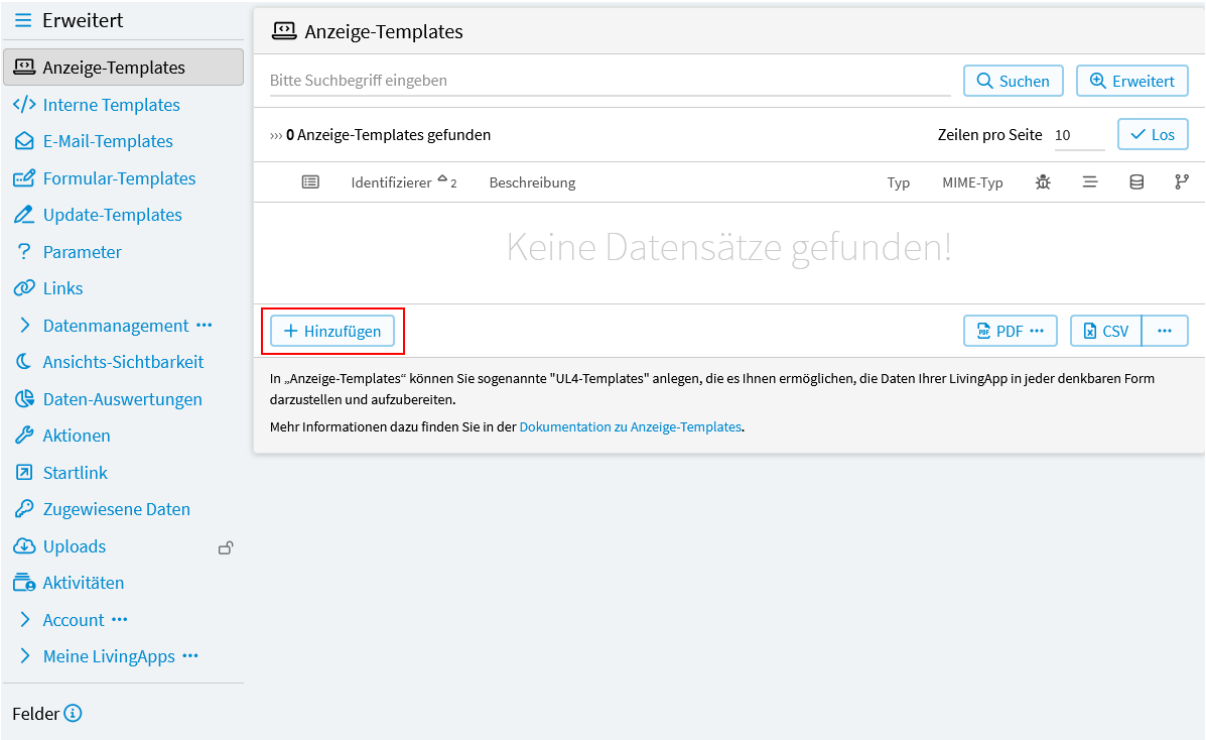


Abb. 36: Anzeige-Template hinzufügen

Typ

Es muss ein Template-Typ ausgewählt werden.

Liste

Es kann auf alle Datensätze, die in der *Datenquelle* konfiguriert sind, zugegriffen werden. Soll das Template eine Auswertung aller Daten beinhalten, so muss dieser Typ gewählt werden.

Liste Standard

Beschreibung siehe oben unter *Liste*. Dieses Template wird ausgeliefert, wenn die URL keinen Parameter `template` hat. Darüber können Sie unter *Aufruf eines Anzeige-Templates* mehr lesen. (Es kann nur ein Anzeige-Template mit dem Typ Liste (Standard) geben.)

Liste (incl. Datenmanagement)

Hier steht der Link zum Anzeige-Template auch im Datenmanagement zur Verfügung (und zwar in der Fußzeile der Liste).

Detail

Es kann nur auf einen einzelnen Datensatz zugegriffen werden.

Detail (Ergebnisseite)

Hier wird dieses Anzeige-Template verwendet um den Inhalt der Ergebnisseite beim Abspeichern eines Formulars zu erzeugen. In dem Fall muß das Template nur den Seiteninhalt erzeugen, der Seitenrahmen wird automatisch erzeugt. (Es kann nur ein Anzeige-Template mit dem Typ Detail (Ergebnisseite) geben.)

Detail (incl. Datenmanagement)

Hier steht der Link zum Anzeige-Template auch im Datenmanagement zur Verfügung (und zwar bei jedem einzelnen Datensatz).

Support

Ein Support-Template kann für andere Anzeigen verwendet werden, die keine Datensätze benötigen. Z. B. für CSS oder Javascript.

MIME-Typ

Dateiformat, in welchem das Template ausgegeben wird. Hier kann z. B. das CSS das in einem Listen- oder

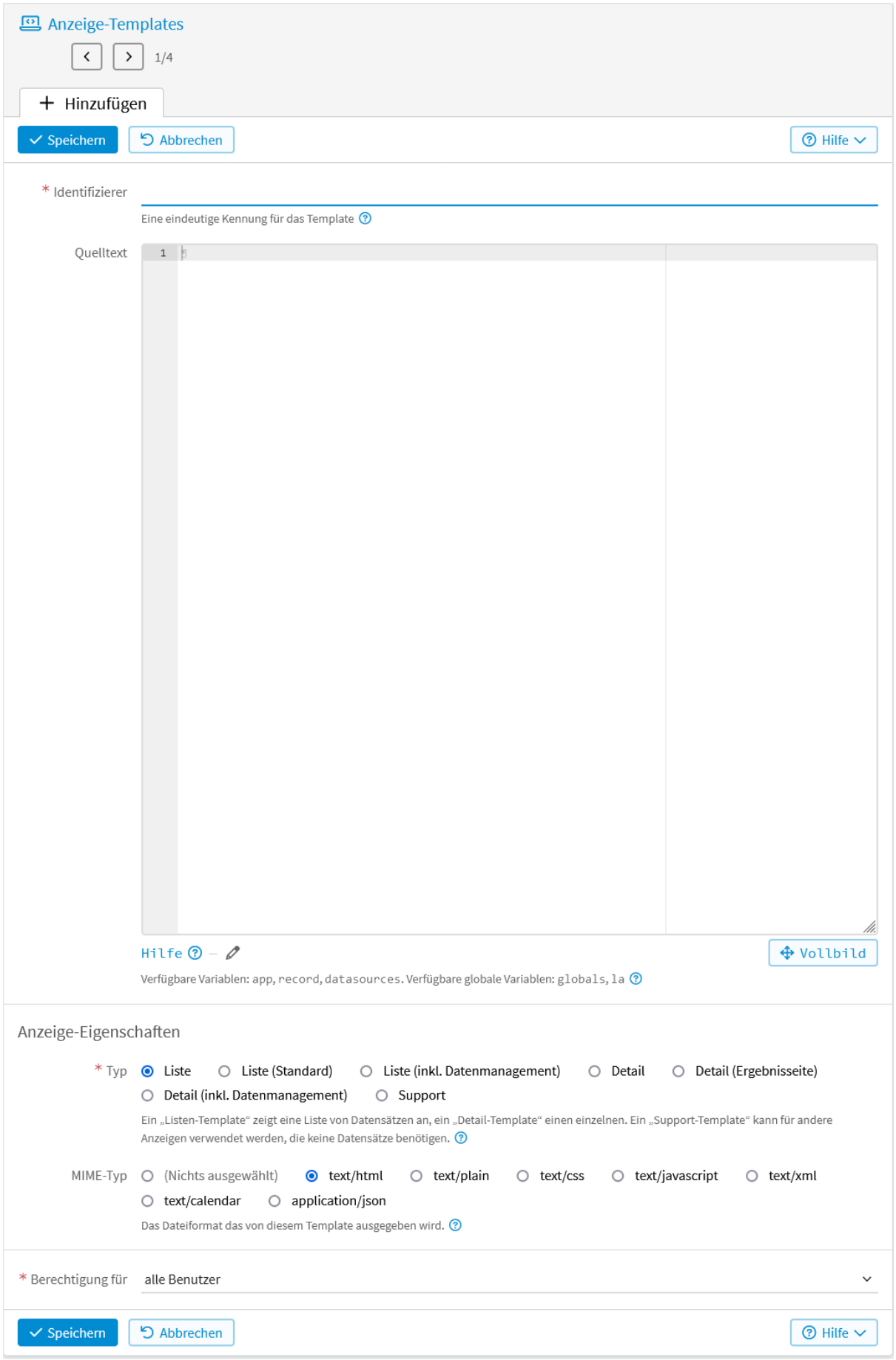


Abb. 37: Anzeige-Template Eingabeansicht

Detail-Template verwendet wird ausgelagert und anschließend im HTML-Template eingebunden werden.

Berechtigung für
Hier kann ausgewählt werden, wer das Template aufrufen kann. In der jeweiligen LivingApp können unter *Konfiguration* → *Berechtigungen* verschiedenen Personen unterschiedliche Rechte zugewiesen werden. Je nachdem, wer welche Rechte hat und für wen das Template freigeschaltet wurde, haben unterschiedliche Personen Zugriff auf das jeweilige Template.

Alle Benutzer
Das Template ist frei zugänglich.

Eingeloggte Benutzer
Um das Template aufrufen zu können, muss diejenige Person auf LivingApps eingeloggt sein.

User mit Rechten auf der Applikation
Bei dieser Auswahl kann das Template nur durch Personen aufgerufen werden, die irgendein Recht an der App besitzen. Dies kann sowohl ein Einzelrecht sein, als auch eine Rolle (z. B. Minimal, Standard oder Administrator).

User mit Datenverarbeitungsrechten auf der Applikation
Bei dieser Auswahl kann das Template nur durch Personen aufgerufen werden, die die Berechtigung haben, in die Datenliste der App zu gelangen.

User mit Admin-Rechten an der Applikation
Bei dieser Auswahl kann das Template nur durch Personen aufgerufen werden, die Administrationsrechte an der App haben.

Ein Anzeige-Template ist in der Regel aus zwei Teilen aufgebaut. Der erste Teil führt die Datenaufbereitung durch. Hier wird auf die Datensätze zugegriffen und die gewünschte Auswertung berechnet. Der zweite Teil sorgt für die HTML-Ausgabe der ermittelten Daten. Dazu können sämtliche HTML-Elemente verwendet werden. Zur Formatierung kann zudem CSS eingesetzt werden. Hierzu können Sie im Kapitel *Formulierung des HTML-Codes* mehr lesen.

Formulierung des Templates

Um auf die Daten Ihrer App zugreifen zu können, benötigten Sie die Templatesprache UL4. Die UL4-Dokumentation finden Sie unter https://python.livinglogic.de/UL4.html. Die Ausgabe der Daten erfolgt über die *LivingAPI*. Im zugehörigen Kapitel sind die Objekte ausführlich beschrieben.

Durchlaufen von Datensätzen bei Templates vom Typ *Liste*
Die Datensätze der jeweiligen App sind im *Record*-Dictionary der *App* (`app.records`) gespeichert.

Damit die Datensätze dem Template zur Verfügung stehen, muss dafür eine Datenquelle angelegt werden. Über die Datenquelle können Sie dann auch z. B. auf *App-Parameter* zugreifen oder Daten mehrerer Apps ausgeben. Näheres können sie in der Dokumentation der *Datenquelle* nachlesen.

Wählen sie in der Datenquelle dieselbe App aus, in der das Template erstellt wird, durchlaufen Sie mit folgender `for`-Schleife alle vorhandenen Datensätze.

```
<?for r in app.records.values()?>
   Anweisungen
<?end for?>
```

Möchten Sie Datensätze aus anderen Apps in Ihrem Template anzeigen, durchlaufen Sie die Datensätze mit folgender `for`-Schleife:

```
<?for r in datasources.beispiel.app.records.values()?>
   Anweisungen
<?end for?>
```

Wobei `beispiel` für den Identifizierer der Datenquelle steht, die Sie in Ihrem Template angelegt haben.

Ausgabe von Feldwerten

Die Attribute der Datensätze, wie das Erstellungsdatum, die Felder oder Anhänge sind im *Record*-Objekt gespeichert. Im *Field*-Objekt befinden sich die Werte des Feldes. *Control*-Objekte beinhalten die Metadaten eines Feldes einer Applikation. Je nach Feld-Typ ist dies ein spezieller Objekt-Typ (z.B. TextControl für ein normales einzeiliges Text-Feld). Zu den Feldtypen mit Ihren Untertypen können Sie im Kapitel *Übersicht der Feldtypen* mehr lesen.

Auf die einzelnen Attribute kann per Punktnotation zugegriffen werden. Zur Ausgabe von Daten aus dem Datensatz verwenden Sie die Anweisung `printx` und geben das gewünschte Element des Datensatzes mit Punktnotation an.

Der Feldinhalt eines `string/text`-Feldes kann dann beispielsweise innerhalb der obigen `for`-Schleife und verkapselt in einer `if`-Anweisung wie folgt ausgegeben werden:

```
<?if r.fields.identifier.value?>
   <?printx r.fields.identifier.value?>
<?end if?>
```

Wie die Werte anderer Feldtypen ausgegeben werden, können Sie im Kapitel *Übersicht der Feldtypen* nachlesen.

Desweiteren stehen die Werte auch über „Shortcut"-Attribute zur Verfügung. `record.fields.identifier` steht auch direkt als `record.f_identifier` zur Verfügung. `record.fields.identifier.value` steht auch direkt als `record.v_identifier` zur Verfügung. Wobei `identifier` jeweils für den Identifizierer des Feldes steht, das sie ausgeben möchten.

Auf den Feldwert kann analog mit `<?printx r.fields['identifier'].value?>` zugegriffen werden.

Folgender Code durchläuft alle Datensätze und alle Felder und gibt, wenn ein Wert vorhanden ist, dann deren Werte aus:

```
<?for r in app.records.values()?>
   <?for f in r.fields.values()?>
      <?if f.value is not None?>
         <?printx f.value?>
      <?end if?>
   <?end for?>
<?end for?>
```

Prüfung von Feldwerten auf Inhalt

Bevor Sie den Inhalt eines Formularfeldes ausgeben, können Sie prüfen, ob ein Inhalt eingegeben wurde. Dafür ist die Ausgabe mit einer `if`-Bedingung zu umschließen, die prüft, ob eine Angabe im betrachteten Feld vorhanden ist.

Als „leer" gelten:

- der boolsche Wert False
- leere Zeichenketten
- leere Listen
- leere Dictionaries
- numerische Null-Werte
- der Wert `None`

Es gibt folgende drei Möglichkeiten, den Feld-Wert zu überprüfen. Wobei `identifier` für den Identifizierer des jeweiligen Feldes steht.

Prüfen, ob ein Wert eingegeben wurde:

```
<?if record.fields.identifier.value?>
   Anweisung für den Fall, dass ein Wert vorhanden ist
<?end if?>
```

Shortcut:

```
<?if record.v_identifier?>
   Anweisung für den Fall, dass ein Wert vorhanden ist
<?end if?>
```

Besonderheiten:

Bei leeren Listen in `multiplelookup`/`multipleapplookup`-Feldern wird der `if`-Block in obiger Logik nicht ausgeführt, verhält sich also korrekt. Die Zahl „0“ in Feldern vom Typ `number` ist `False` und wird deshalb nicht ausgeführt.

Prüfen, ob der Wert nicht `None` ist:

```
<?if record.fields.identifier.value is not None?>
   Anweisung für den Fall, dass ein Wert vorhanden ist
<?end if?>
```

Shortcut:

```
<?if record.v_identifier is not None?>
   Anweisung für den Fall, dass ein Wert vorhanden ist
<?end if?>
```

Besonderheiten:

Bei leeren Listen in `multiplelookup`/`multipleapplookup`-Feldern wird der if-Block ausgeführt. Die Zahl „0“ in Feldern vom Typ `number` ist `True` und wird ausgeführt.

Methode `is_empty()` von *Field*-Objekt:

```
<?if not record.fields.identifier.is_empty()?>
   Anweisung für den Fall, dass ein Wert vorhanden ist
<?end if?>
```

Shortcut:

```
<?if not record.f_identifier.is_empty()?>
   Anweisung für den Fall, dass ein Wert vorhanden ist
<?end if?>
```

Je nach Typ ist das Richtige implementiert.

Bei leeren Listen in `multiplelookup`/`multipleapplookup`-Feldern wird der `if`-Block nicht ausgeführt. Die Zahl „0“ in Feldern vom Typ `number` ist `True` und wird ausgeführt.

Über die Verwendung der Methode `is_empty()` können Sie in der Dokumentation der LivingAPI - Objekt-Typen unter *Field* mehr nachlesen.

Whitespace

Befinden sich im Template UL4-Ausdrücke oder werden andere Templates eingebunden, empfiehlt es sich zur besseren Darstellung, einen <?whitespace?>-Tag ins Template einzufügen. Folgende drei Varianten stehen zur Auswahl:

<?whitespace keep?>
: Leerzeichen und Zeilenvorschübe werden 1:1 übernommen.

<?whitespace strip?>
: Leerzeichen und Zeilenvorschübe zwischen den Tags werden ignoriert.

<?whitespace smart?>
: Bei Zeilen die außer einem <?print?>-, <?printx?>- oder <?render?>-Tag nur Whitespace beinhalten, wird die Einrückung und der Zeilenvorschub ignoriert. Weiterhin wird die zusätzliche Einrückung ignoriert, die innerhalb eines <?for?>-, <?if?>-, <?elif?>-, <?else?>- oder <?def?>-Tags verwendet wird.

Formulierung des HTML-Codes

Abgesehen von Daten der Datensätze können Sie natürlich auch alle anderen HTML-Elemente in Ihre Auswertung integrieren. Sie können einen gesamten HTML-Quellcode mit `head`- und `body`-Bereich als Anzeige-Template eingeben und an den entsprechenden Stellen UL4 verwenden, um auf Datensätze zuzugreifen, sie zu verarbeiten und zu analysieren. Mit HTML-Notation werden diese Daten dann beispielsweise in Tabellen- oder Listenform dargestellt. Außerdem können Überschriften, Verlinkungen und Bilder in Ihre Auswertung eingefügt werden. Ihnen stehen alle Möglichkeiten der HTML-Notation zur Verfügung.

Um gewisse Formatierungen festzulegen und um die optische Darstellung der Auswertung ansprechend zu gestalten, kann zudem CSS verwendet werden.

Mehr Informationen zu HTML und CSS sowie ausführliche Anleitungen finden Sie unter https://wiki.selfhtml.org/

CSS-Template

Für das nachfolgende Anwendungsbeispiel wurde ein CSS-Template angelegt und im Anzeigetemplate `teilnehmerliste` eingebunden.

Um ein CSS-Template anzulegen, wählen Sie dafür im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Anzeige-Templates*. Klicken Sie auf *Hinzufügen*.

Im nun geöffneten Fenster wurden für das Anwendungsbeispiel folgende *Konfigurationen* vorgenommen:

Wie das CSS-Template im Anzeigetemplate `teilnehmerliste` eingebunden wird, können Sie im *Anwendungsbeispiel* nachlesen.

Ergebnisseite

Sie haben die Möglichkeit ein Anzeigetemplate zu erstellen, das dem Teilnehmer/ der Teilnehmerin nach der Eingabe eines Datensatzes oder nach einer Änderung über Formular, als Ergebnisseite angezeigt werden soll. Diese Ergebnisseite könnte folgendermaßen aussehen.

Um eine Ergebnisseite anzulegen, wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Anzeige-Templates*. Klicken Sie auf *Hinzufügen*.

Im nun geöffneten Fenster werden folgende Konfigurationen vorgenommen:

Bei *Typ* muss *Detail (Ergebnisseite)* gewählt werden.

Wie eine Ergebnisseite aufgerufen wird, können Sie unter *Aufruf eines Anzeige-Templates* nachlesen.

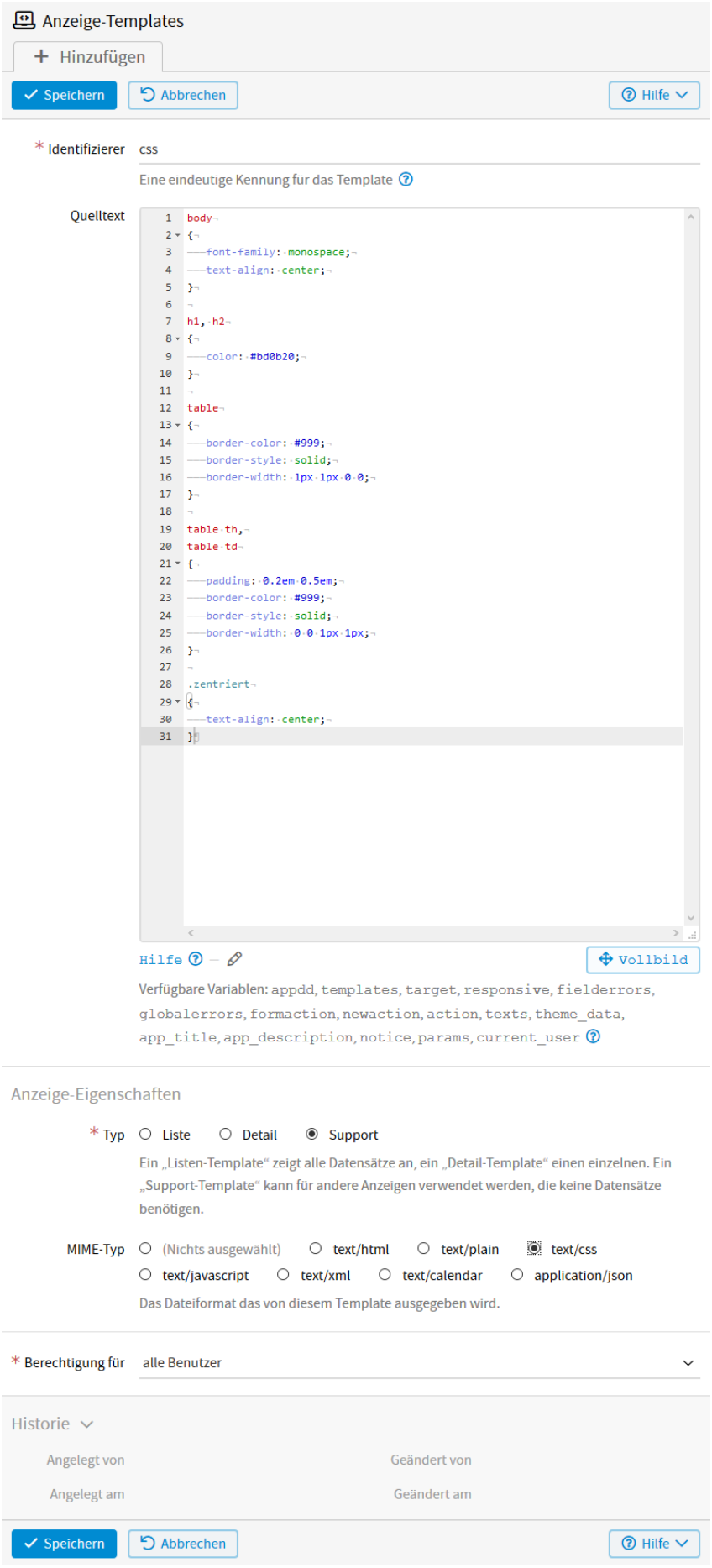


Abb. 38: Anzeige-Template CSS - Konfiguration

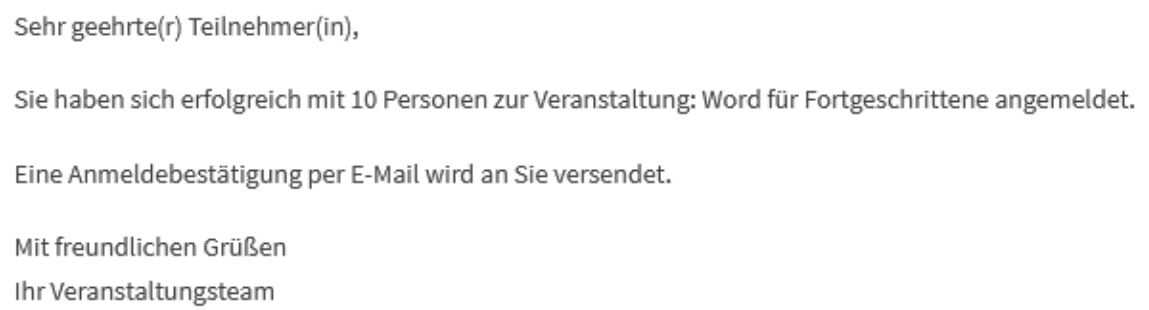


Abb. 39: Anzeige-Template Ergebnisseite

Aufruf eines Anzeige-Templates

Wenn Sie das Formular einer beliebigen LivingApp öffnen, sehen Sie in der URL hinter `apps/` eine Kombination aus Buchstaben und Zahlen. Dies ist die App-UUID (Universally Unique Identifier), der eindeutige Identifizierer Ihrer App. Im Formular steht hinter der UUID `/new`, da hier ein neuer Datensatz angelegt werden kann. An das `/new` ist die View-ID angehängt. Der Link zum Formular der App könnte beispielsweise wie folgt lauten:

```
https://my.living-apps.de/gateway/apps/563a2616591c6ca7a9800ef2/new?
↪view=584f56445afcda56b7716252
```

Wobei `563a2616591c6ca7a9800ef2` die App-UUID ist und `584f56445afcda56b7716252` die View-ID. Ein Listen-Template, bei dem bei *Standard* das Häkchen gesetzt ist, erreichen Sie, indem Sie `/new?view=584f56445afcda56b7716252` aus der URL löschen und keinen weiteren Parameter anhängen. Im Beispiel wäre das folgende URL:

```
https://my.living-apps.de/gateway/apps/563a2616591c6ca7a9800ef2
```

Haben Sie hingegen bei Ihrem Listen-Template das Häkchen bei *Standard* nicht gesetzt, erreichen Sie dieses Template, indem Sie `?template=mein_identifizierer` hinter die App-UUID an die URL anhängen. Heißt der Template-Identifizierer z. B. `uebersicht`, so lautet die Beispiel-URL:

```
https://my.living-apps.de/gateway/apps/563a2616591c6ca7a9800ef2?template=uebersicht
```

Wird bei einem Detail-Template bei *Ergebnisseite* das Häkchen gesetz, dann wird nach Eingabe oder Änderung eines Datensatzes über Formular, dieses Template als Ergebnisseite angezeigt. Der Identifizierer wird in dem Fall automatisch an die URL angehängt. Diese würde dann mit dem Template-Identifizierer `beispiel` wie folgt lauten:

```
https://my.living-apps.de/gateway/apps/563a2616591c6ca7a9800ef2/
↪56683221 0d3450b5949b9c00?template=beispiel
```

Im vorherigen Beispiel wird außerdem sichtbar, dass auf Datensatz-Ebene hinter die App-UUID eine weitere ID (dat_id) gehängt wird, die für jeden Datensatz eindeutig ist.

Sie haben auch die Möglichkeit einen Link zum Aufruf Ihres Anzeige-Templates zu erstellen. Darüber können Sie in der Dokumentation der *App-Menüs*, *Benutzer-Menüs* bzw. *App-Panels* oder *Benutzer-Panels* mehr lesen.

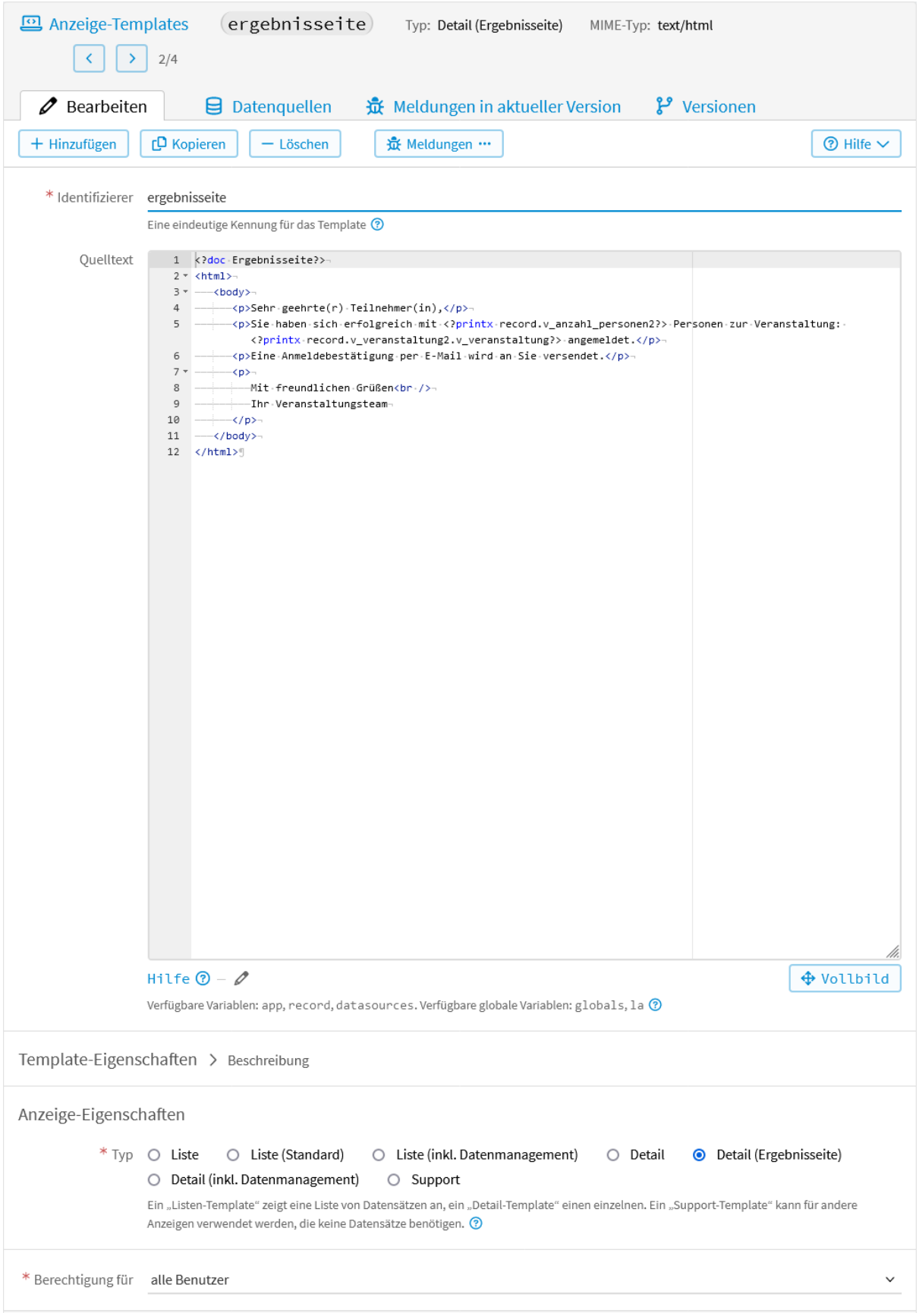


Abb. 40: Anzeige-Template Ergebnisseite - Konfiguration

2.5.2 Anlegen von Datenquellen

Nachdem das Template das erste Mal gespeichert wurde, kann im Tab *Datenquellen* die Datengrundlage für das Template konfiguriert werden.

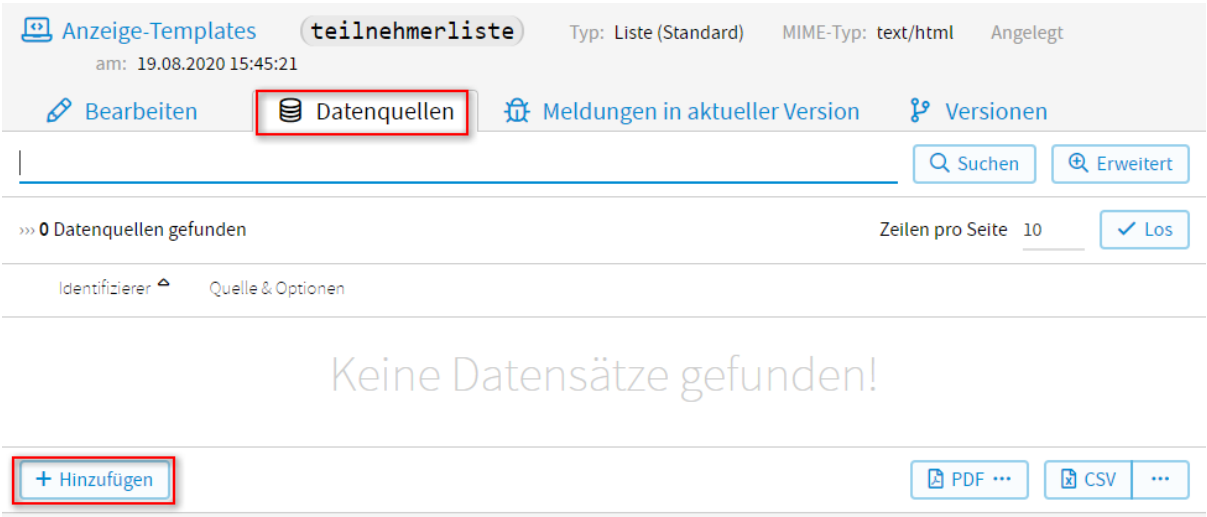


Abb. 41: Datenquelle hinzufügen

Dies ermöglicht es, auf Daten aus mehreren Apps zuzugreifen, die zur Verfügung stehenden Daten im Vorhinein einzuschränken und auf App-Parameter zuzugreifen. Außerdem haben Sie die Möglichkeit bei *Sortierung* Ihre *Daten zu sortieren* und bei *Felder* einzelne *Felder Ihrer App aus- oder abzuwählen*.

Möchten Sie auf Daten mehrerer unterschiedlicher Apps zugreifen, müssen Sie für jede App eine Datenquelle anlegen. Sie können im Anzeige-Template auf die Datenquelle zugreifen mit `datasources.beispiel`. Wobei `beispiel` für den Identifizierer der Datenquelle steht. Über die Attribute der Datenquelle können Sie in der Dokumentation der LivingAPI unter *DataSource* mehr lesen.

Klicken Sie auf *Hinzufügen* um eine Datenquelle anzulegen. Im nun geöffneten Fenster können folgende *Konfigurationen* vorgenommen werden.

Identifizierer
: Der eindeutige Name dieser Datenquelle. Der Name darf nur Buchstaben, Ziffern und _ beinhalten. Eine Datenquelle mit dem Namen `beispiel` kann in den Anzeige-Templates mittels des Ausdrucks `datasources.beispiel` angesprochen werden.

App
: Hier kann die App ausgewählt werden, deren Datensätze die Datenquelle dem Template zur Verfügung stellen soll. Möchten Sie die Informationen der *System-Apps* zur Verfügung stellen, müssen Sie hier die entsprechende System-App wählen.

  Wird eine App ausgewählt, haben Sie die Moglichkeit bei *Kopien einbeziehen* das Häkchen zu setzen. Damit wird nicht nur diese App selbst, sondern auch alle ihre Kopien dem Template zur Verfügung gestellt.

  App-Filter
  : Einen App-Filter können Sie nur dann eingeben, wenn Sie bei *App Alle Apps* gewählt oder bei einer bestimmten App das Häkchen bei *Kopien einbeziehen* gesetzt haben.

    Diese Bedingung muss eine App `a` erfüllen, um in die Liste der Apps zu kommen, aus der die Datensätze in die Datenquelle aufgenommen werden. Es stehen die *Variablen* `a` (zu filternde App), `app` (App zu der diese Datenquelle gehört), `user` (eingeloggter Benutzer), `record` (Detail-Datensatz; nur in Detail-Templates) und `params` (Request-Parameter) zur Verfügung.

    Mit `a.owner.id == user.id` können Sie sich zum Beispiel nur die Apps anzeigen lassen, bei denen der eingeloggte User der Besitzer ist. Mit `a.identifier == p.str.app` wird die App angezeigt, deren UUID als Parameter in der URL mitgegeben wird.

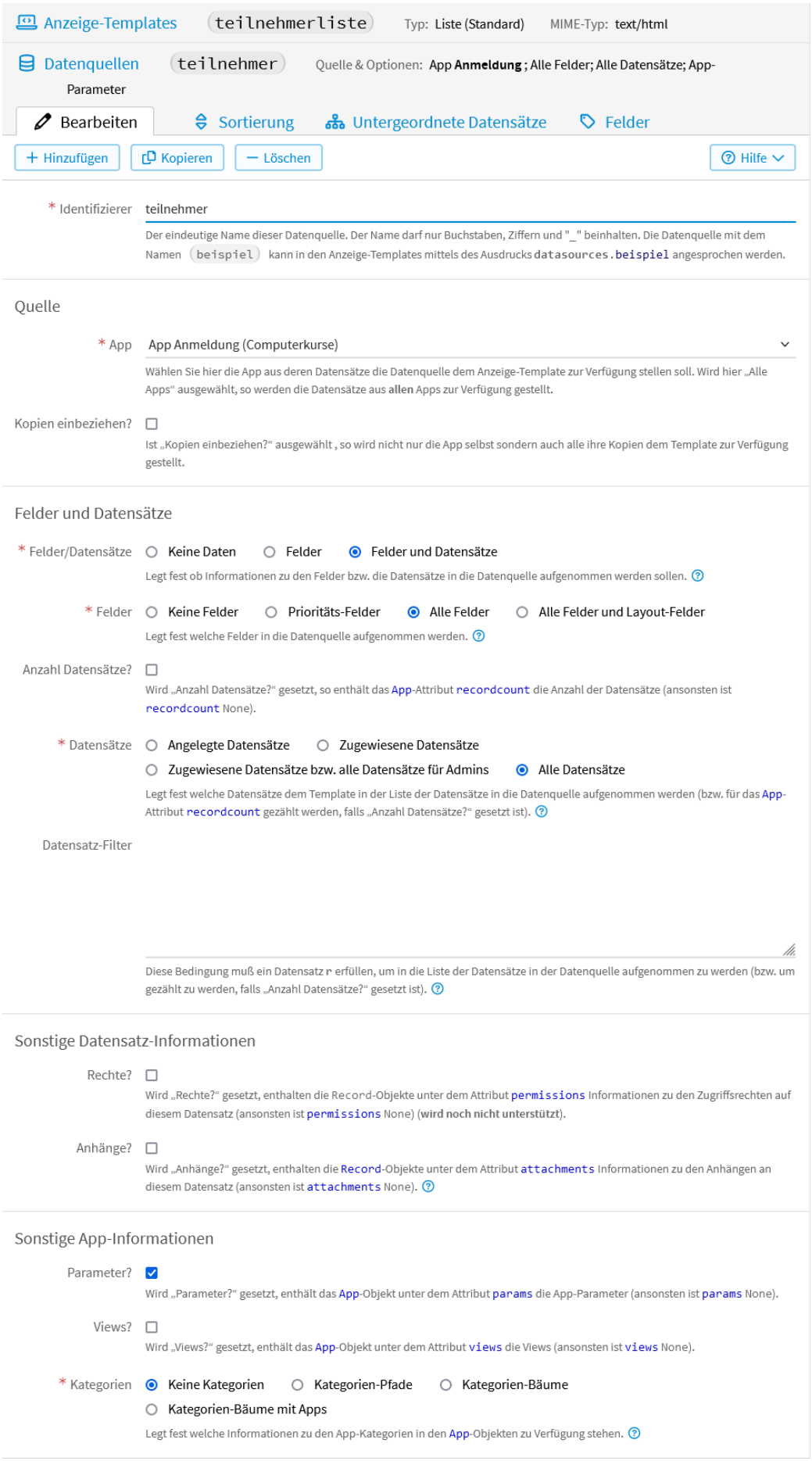


Abb. 42: Datenquellen

Ist der Ausdruck nicht vom Typ `BOOL`, wird er mittels der Funktion bool konvertiert. Mehr Informationen zu vSQL-Ausdrücken finden Sie in der Dokumentation zu *vSQL*.

Felder/Datensätze
Hier können Sie festlegen, ob Informationen zu den Feldern (`controls`), oder die Datensätze (`records`) in die Datenquelle aufgenommen werden sollen.

keine Daten
Es werden keine Informationen zu den Feldern bzw. Datensätze in die Datenquelle aufgenommen, sondern nur die App-Information (Name der App, Beschreibung der App, Icon, ...). Dies kann z.B. sinnvoll sein, wenn Sie eine Übersicht über alle Ihre Apps erstellen wollen.

Felder
Hier werden nur Feldinformationen (Name des Feldes, Feldtyp, App zu der das Feld gehört, Info ob Listenfeld, Reihenfolge der Felder, ...) in die Datenquelle aufgenommen. Dies kann z.B. für reine Eingabemasken sinnvoll sein.

Felder und Datensätze
Zusätzlich zu den Feldinformationen werden hier die Datensätze der App mit in die Datenquelle aufgenommen. Diese ist die übliche Auswahl. Damit können Sie dann die Daten Ihrer App im Anzeigetemplate auswerten oder darstellen.

Felder
Erscheint bei der Auswahl von *Felder* oder *Felder und Datensätze*. Legt fest, welche Felder in die Datenquelle aufgenommen werden.

keine Felder
Es werden gar keine Felder aufgenommen.

Prioritätsfelder
Es werden nur die Felder aufgenommen, die in der Datenliste im Datenmanagement ausgewählt sind.

Alle Felder
Es werden alle Felder aufgenommen.

Alle Felder und Layoutfelder
Es werden alle Felder aufgenommen. Zusätzlich enthält `View.layout_controls` alle Layout-Felder, wie die „Formatierten Textfelder“, „Dekobilder“ und den „Absenden“-Button.

(Damit View-Objekte in die Datenquelle aufgenommen werden, muß der Haken bei *Views?* gesetzt sein.)

Anzahl Datensätze
Wird hier das Häkchen gesetzt, wird die Anzahl der Datensätze mit in die Datenquelle aufgenommen (als App-Attribut `recordcount`).

Datensätze
Erscheint bei der Auswahl von *Felder und Datensätze*. Legt fest, welche Datensätze dem Template in der Liste der Datensätze in die Datenquelle aufgenommen werden.

Angelegte Datensätze
Es werden nur die vom Benutzer angelegten Datensätze aufgenommen.

Zugewiesene Datensätze
Es werden nur Datensätze aufgenommen, die dem Benutzer über die Konfiguration *Zugewiesene Daten* zugewiesen wurden. Ohne diese Konfiguration wird diese Option wie *Angelegte Datensätze* behandelt.

Zugewiesene Datensätze bzw. alle Datensätze für Admins
Hat der Benutzer die Rolle Adminstrator oder Datenmanager, so werden alle Datensätze aufgenommen, sonst wird diese Option wie *Zugewiesene Datensätze* behandelt.

Alle Datensätze
Es werden alle Datensätze aufgenommen.

Datensatz-Filter
Diese Bedingung muss ein Datensatz `r` erfüllen, um in die Liste der Datensätze in der Datenquelle aufgenommen zu werden. Es stehen die *Variablen* `r` (zu filternder Datensatz), `app` (App zu der dieses Anzeige-

Template gehört), `user` (eingeloggter Benutzer) und `params` (Request-Parameter) zur Verfügung. Zum Beispiel könnten Sie nur die Daten, die in den letzten 30 Tagen eingegeben wurden anzeigen lassen mit `r.createdat >= now() - days(30)`. Folgenden Filter können Sie eingeben, wenn Sie zum Beispiel nur die Daten aus der App „Vorgänge“ anzeigen lassen wollen, bei denen als Vorgangstyp Rechnung in der Auswahl gewählt wurde `r.v_vorgangstyp == "rechnung"`. Wobei `vorgangstyp` für den identifier des Feldes Vorgangstyp steht.

Ist der Ausdruck nicht vom Typ `BOOL`, wird er mittels der Funktion bool konvertiert. Mehr Informationen zu vSQL-Ausdrücken finden Sie in der Dokumentation zu *vSQL*.

Rechte?
Wird hier das Häkchen gesetzt, wird das Attribut `permissions` gefüllt mit allen der App zugeordneten Benutzern und deren Berechtigungen. (Wird noch nicht unterstützt.)

Anhänge?
Wird hier das Häkchen gesetzt, werden auch die Anhänge (`attachments`) der jeweiligen Datensätze mit in die Datenquelle aufgenommen. Bei dem Datensatz der bei einem Detail-Template angezeigt wird, stehen die Anhänge immer zu Verfügung, unabhängig davon, ob das Häkchen bei *Anhänge?* gesetzt ist oder nicht.

Parameter?
Wird hier das Häkchen gesetzt, dann stehen dem Template die unter *Konfiguration → Erweitert → Parameter* erstellten *App-Parameter* (`params`) zur Verfügung.

Views?
Wird hier das Häkchen gesetzt, werden alle Formularvarianten (Formularansichten — `views`) dem Template zur Verfügung gestellt.

Kategorien
Legt fest, welche Informationen zu den *App-Kategorien* in den App-Objekten zur Verfügung stehen. Zur Demonstration der folgenden Möglichkeiten wird das Beispiel der unter *Konfiguration → Erweitert → Account → App-Kategorien angelegten Kategorien* verwendet.

Keine Kategorien
Es steht keine Information zur Verfügung, d.h. das Attribut `categories` der App-Objekte ist immer `None`.

Kategorien-Pfade
Hier ist das App-Attribut `categories` ein Dictionary mit den Kategorien, zu denen diese App gehört. Die Schlüssel sind der jeweilige Identifizierer der Kategorie und der Wert sind Kategorie-Objekte. Bei diesen verweist das `parent`-Attribut jeweils auf die übergeordnete Kategorie. D.h. über die den Apps zugeordneten Kategorien können über das parent-Attribut die Pfade zu diesen Kategorien im Kategorien-Baum rekonstruiert werden, aber nicht die Bäume selbst. Zum Beispiel *Familie → Privat*, oder *Verein → Privat*.

Kategorien-Bäume
Diese Option funktioniert ähnlich wie *Kategorien-Pfade*, jedoch enthält das Kategorien-Attribut `children` zusätzlich noch die dieser Kategorie untergeordneten Kategorien als Dictionary. (Bei *Kategorien-Pfade* ist `children` immer `None`). Zum Beispiel die übergeordnete Kategorie *Privat* mit den Unterkategorien *Familie* und *Verein*.

Kategorien-Bäume mit Apps
Diese Option funktioniert ähnlich wie *Kategorien-Bäume*, jedoch enthält das Kategorien-Attribut `apps` zusätzlich noch die dieser Kategorie zugeordneten Apps als Dictionary (bei *Kategorien-Pfade* und *Kategorien-Bäume* ist `apps` immer `None`). Zum Beispiel die übergeordnete Kategorie *Privat* mit der Unterkategorie *Familie* und deren Apps *Haushaltsbuch* und *Urlaubsplanung*, sowie der Unterkategorie *Verein* und deren Apps *Mitglieder* und *Spenden*.

Nach dem Speichern der Datenquelle erscheinen neben *Bearbeiten* die Tabs *Sortierung*, *Untergeordnete Datensätze* und *Felder*.

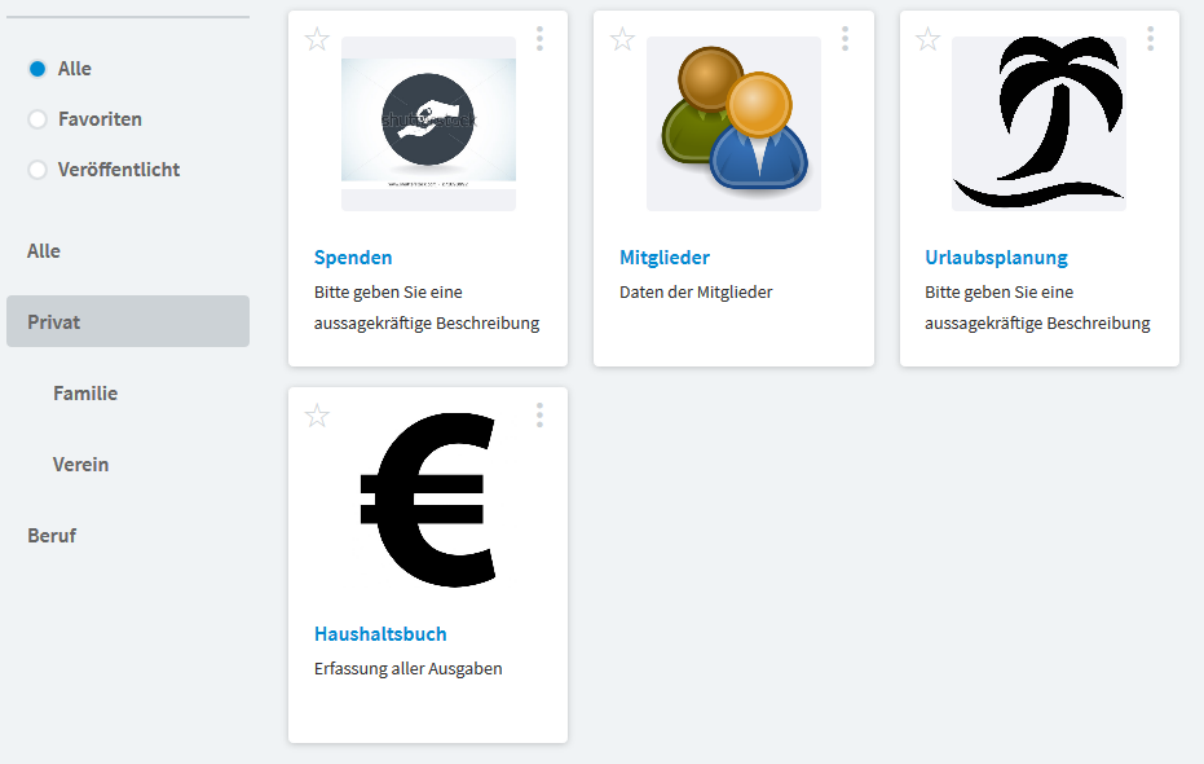


Abb. 43: Beispiel Kategorien

Sortierung

Bei *Sortierung* kann nach einem bestimmten Ausdruck *sortiert*, die Sortier-Richtung und das Handling von Null-Werten eingestellt werden. Wird hier keine Sortierung festgelegt, so wird die Sortierung, wie sie unter *Datenmanagement* → *Sortierung* für das Datenmanagement konfiguriert wurde, verwendet.

Klicken Sie auf den Tab *Sortierung* und dann auf *Hinzufügen*, um eine *Sortierung anzulegen*.

Im nun geöffneten Fenster können folgende Konfigurationen vorgenommen werden.

Reihenfolge
: Bei mehrerern Sortierungen werden die Datensätze in dieser Reihenfolge sortiert.

Ausdruck
: Nach dem Wert dieses Ausdrucks werden die Datensätze sortiert. Für die Formulierung wird *vSQL* verwendet. Es stehen die *Variablen* `r` (zu sortierender Datensatz), `app` (App zu der das zugehörige Anzeige-Template gehört) `record` (Detail-Datensatz; nur in Detail-Templates), `user` (eingeloggter Benutzer) und `params` (Request-Parameter) zur Verfügung.

Sortier-Richtung

Aufsteigend
: Aufsteigende Sortierung (A-Z)

Absteigend
: Absteigende Sortierung (Z-A)

Nullwerte

Zuerst
: Nullwerte werden zuerst angezeigt.

Zuletzt
: Nullwerte werden zuletzt angezeigt.

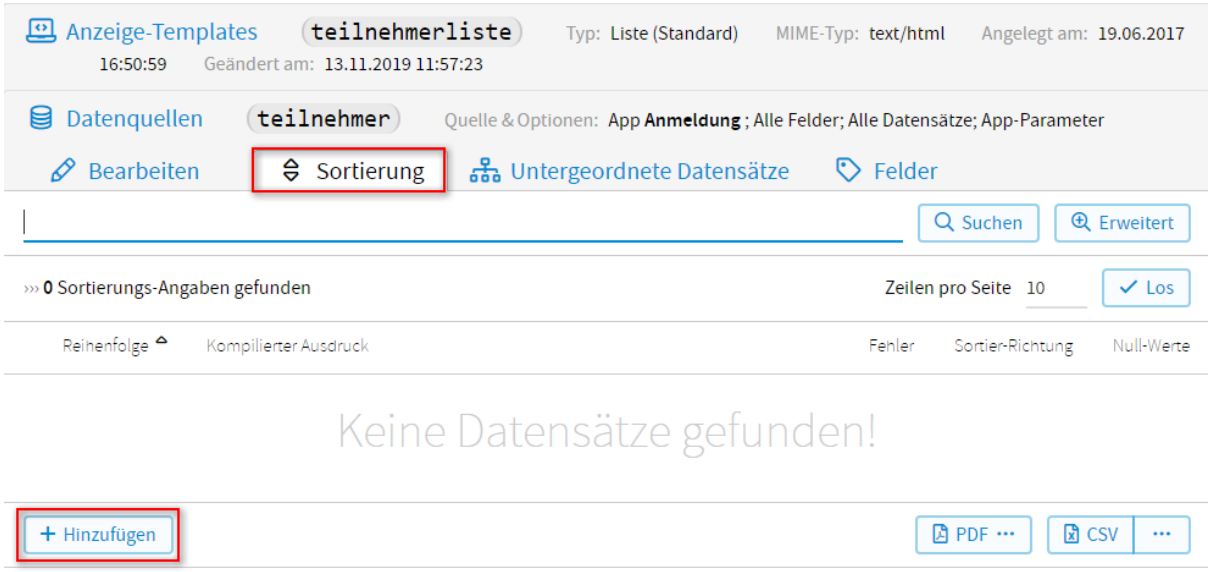


Abb. 44: Datenquellen - Sortierung hinzufügen

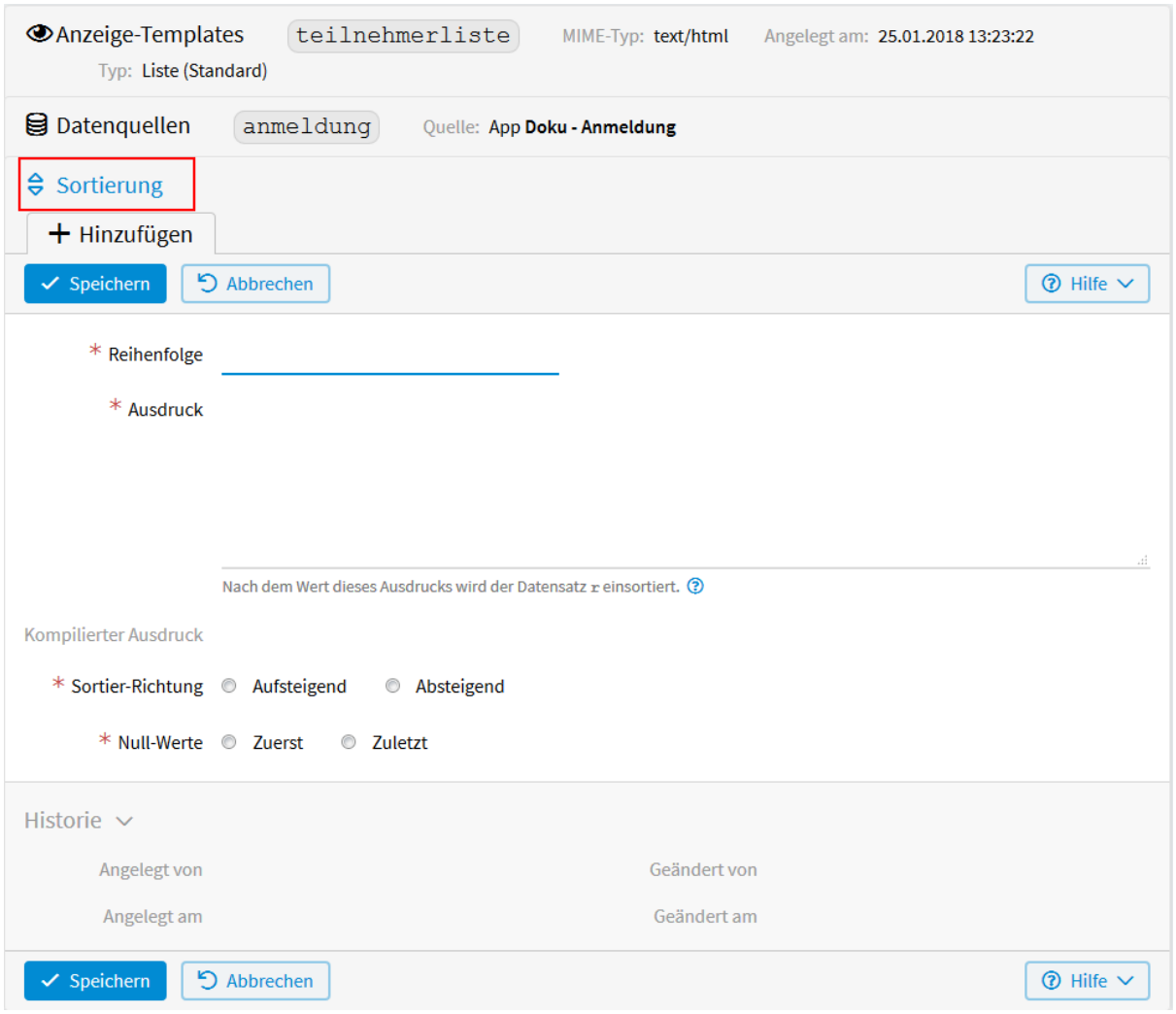


Abb. 45: Datenquellen - Sortierung

Untergeordnete Datensätze

Wenn Sie in einer App eine Auswahl auf eine andere App definieren, dann erzeugen Sie damit eine Eltern-Kind-Beziehung. In unserem Anwendungsfall z. B. greift das Feld *Veranstaltung* in der App *Anmeldung* auf die App *Veranstaltungen* zu. Das heißt die App *Veranstaltungen* hat eine Eltern-Kind-Beziehung zu der App *Anmeldung* (eine Veranstaltung kann mehrere Anmeldungen haben). Diese Beziehung können Sie für untergeordnete Datensätze nutzen. So könnten Sie sich z. B. die Teilnehmer zu Ihren Veranstaltungen in einem Template anzeigen lassen. Dafür müssen Sie in der App *Veranstaltungen* ein *Anzeigetemplate* und dazu eine *Datenquelle* anlegen. Für dieses Beispiel wurde folgende *Datenquelle* konfiguriert.

Nach dem Speichern der Datenquelle klicken Sie auf den Tab *Untergeordnete Datensätze* und dann auf *Hinzufügen*.

Im nun geöffneten Fenster werden folgende Konfigurationen vorgenommen.

Identifizierer
: Der eindeutige Identifizierer für diese Konfiguration. Der Identifizierer darf nur Buchstaben, Ziffern und _ beinhalten. Die Konfiguration mit dem Namen `beispiel` kann in den Anzeige-Templates für einen Datensatz `record` mittels des Ausdrucks `record.children.beispiel` angesprochen werden. Zu einem Veranstaltungsdatensatz wäre z. B. bei o.g. Beispiel ein Zugriff auf die zugehörigen Anmeldungen möglich über `record.children.anmeldungen`.

Ziel
: Hier können Sie die App auswählen, deren Datensätze den Datensätzen dieser Datenquelle zugeordnet sind. Angezeigt werden die Apps, die ausgewählt werden können, sowie die Felder worüber die beiden Apps miteinander verknüpft sind. Ist eine App mehrfach mit einer anderen App verbunden, dann können sie über das Feld, das Ihnen bei *Ziel* mitangezeigt wird, diese beiden Verknüpfungen unterscheiden.

Filterbedingung
: Diese Bedingung muß ein Datensatz `r` erfüllen, um in die Liste der untergeordneten Datensätze aufgenommen zu werden (zusätzlich zu der Bedingung, dass er dem Hauptdatensatz zugeordnet ist). Es stehen die *Variablen* `r` (untergeordneter zu filternder Datensatz), `app` (App zu der das zugehörige Anzeige-Template gehört), `record` (Detail-Datensatz; nur in Detail-Templates), `user` (eingeloggter Benutzer) und `params` (Request-Parameter) zur Verfügung. Ist der Ausdruck nicht vom Typ `BOOL`, wird er mittels der Funktion `bool` konvertiert. Für das *Formulieren der Bedingung* wird *vSQL* verwendet.

Felder

Hier haben Sie die Möglichkeit, einzelne Felder Ihrer App aus- oder abzuwählen, welche in die Datenquelle aufgenommen werden sollen. Das ist z. B. sinnvoll, wenn Sie nur wenige von vielen Feldern anzeigen lassen möchten oder wenn Sie entgegen der Angabe in der Datenquelle einzelne Felder mitaufnehmen oder ausschließen möchten.

Durch Klicken auf den Tab *Felder* gelangen Sie zu folgender Seite.

Nichts ausgewählt
: Das Feld wird entsprechend der Angabe in der Datenquelle behandelt.

Nein
: Das Feld wird nicht mit in die Datenquelle aufgenommen.

Ja
: Das Feld wird mit in die Datenquelle aufgenommen.

In der Übersicht Ihrer Anzeige-Templates können Sie durch Klicken auf den Pfeil neben dem entsprechenden Template einsehen, was in der *Datenquelle konfiguriert* wurde.

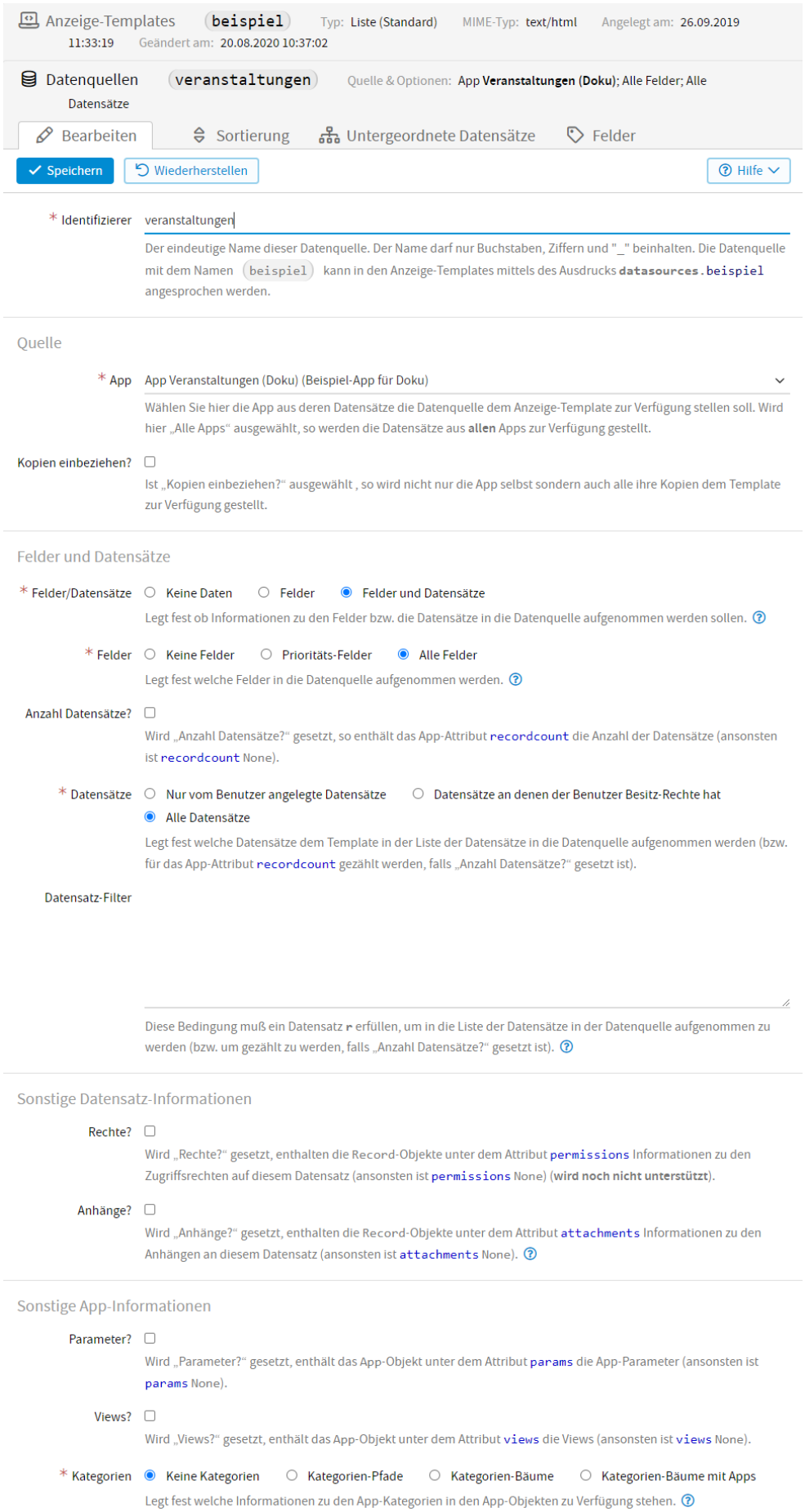


Abb. 46: Datenquelle - untergeordnete Datensätze

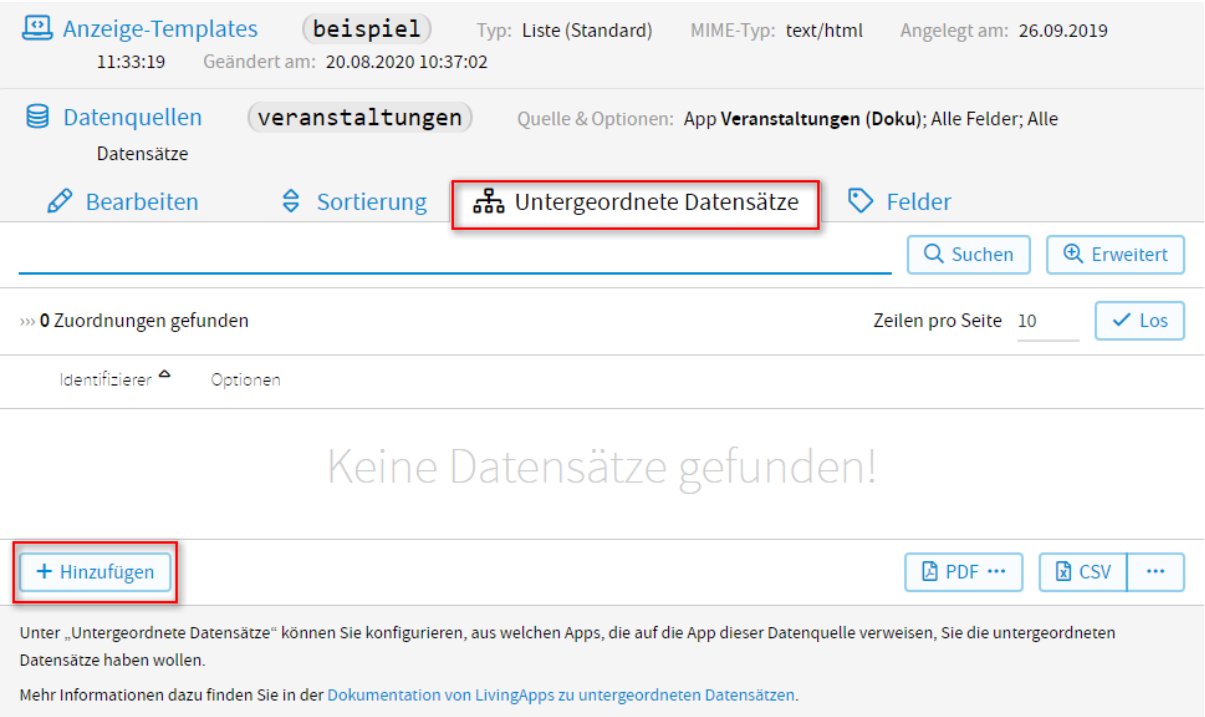


Abb. 47: Datenquelle - untergeordnete Datensätze hinzufügen

2.5.3 Zugriff auf System-Apps

Über System-Apps können Sie auf Informationen aus dem Living-Apps-System zugreifen, als wären diese Informationen Datensätze einer App (eben einer System-App). Überall wo Sie eine normale App als Datenquelle wählen können ist es auch möglich, statt dessen eine System-App zu wählen. Um auf eine System-App zugreifen zu können, müssen Sie dafür eine Datenquelle anlegen und dort als Quelle bei *Apps* die entsprechende System-App auswählen. Darüber können Sie in der Dokumentation der *Datenquelle* mehr lesen.

Folgende System-Apps stehen zur Verfügung.

System-App „Aktivitäten“

Aktivitäten stellen Ereignisse dar, die im Living-Apps-System stattgefunden haben, von denen der eingeloggte Benutzer verständigt werden soll. In der System-App sind alle Aktivitäten vorhanden, die noch nicht als gelesen markiert wurden, aber nicht von Benutzer selbst ausgelöst wurden.

Die Sytem-App „Aktivitäten“ hat folgende Felder:

Name	Identifizierer	Typ
Übergeordnete Aktivität	`parent`	`applookup/select`
Typ	`type`	`string/text`
App	`app`	`applookup/select`
Datensatz	`record`	`applookup/select`
Inhalt	`content`	`string/textarea`
Quittiert am	`seen`	`date/datetimeminute`

Das Feld `type` (d.h. der Aktivitäts-Typ) kann folgende Werte haben:

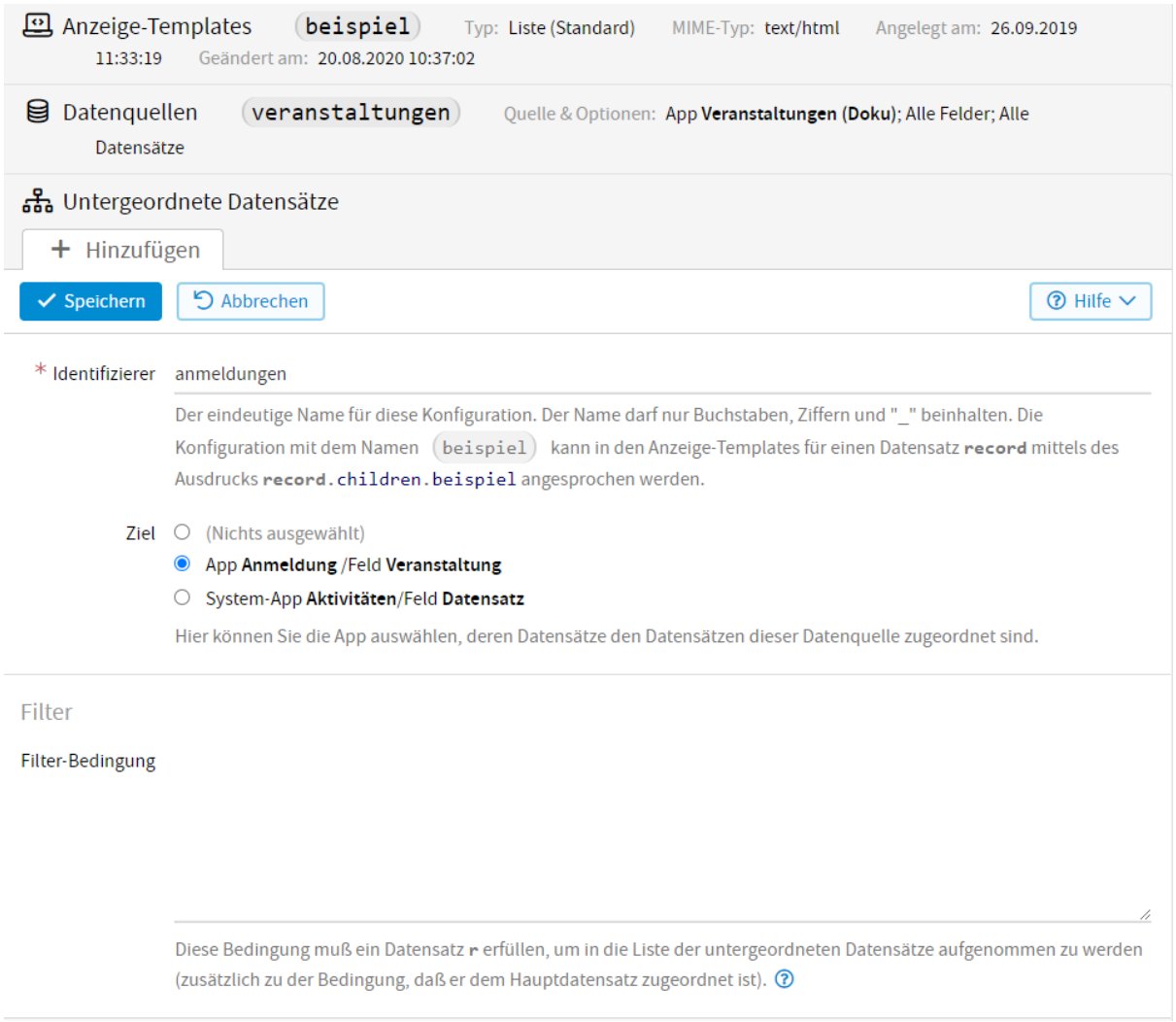


Abb. 48: Datenquellen - untergeordnete Datensätze - Konfiguration

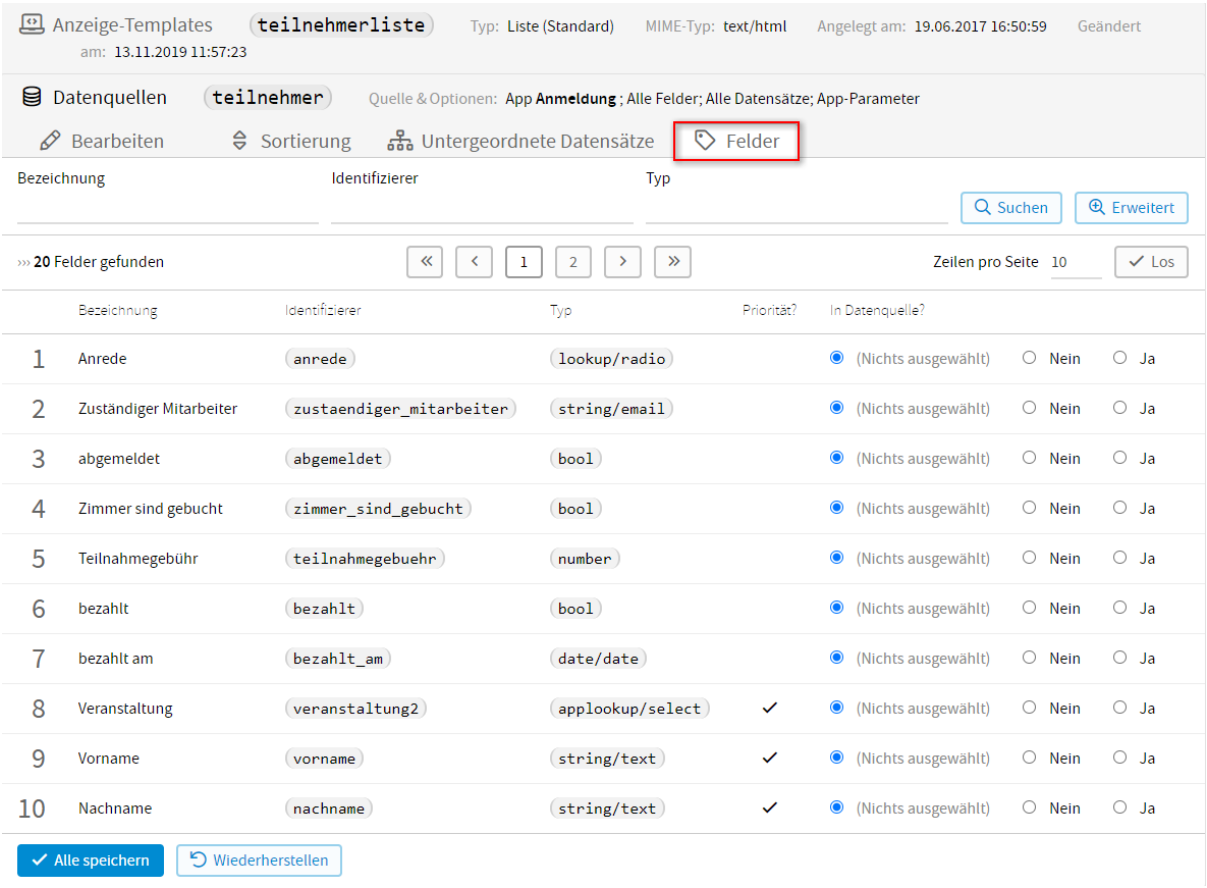


Abb. 49: Datenquellen - Felderwahl

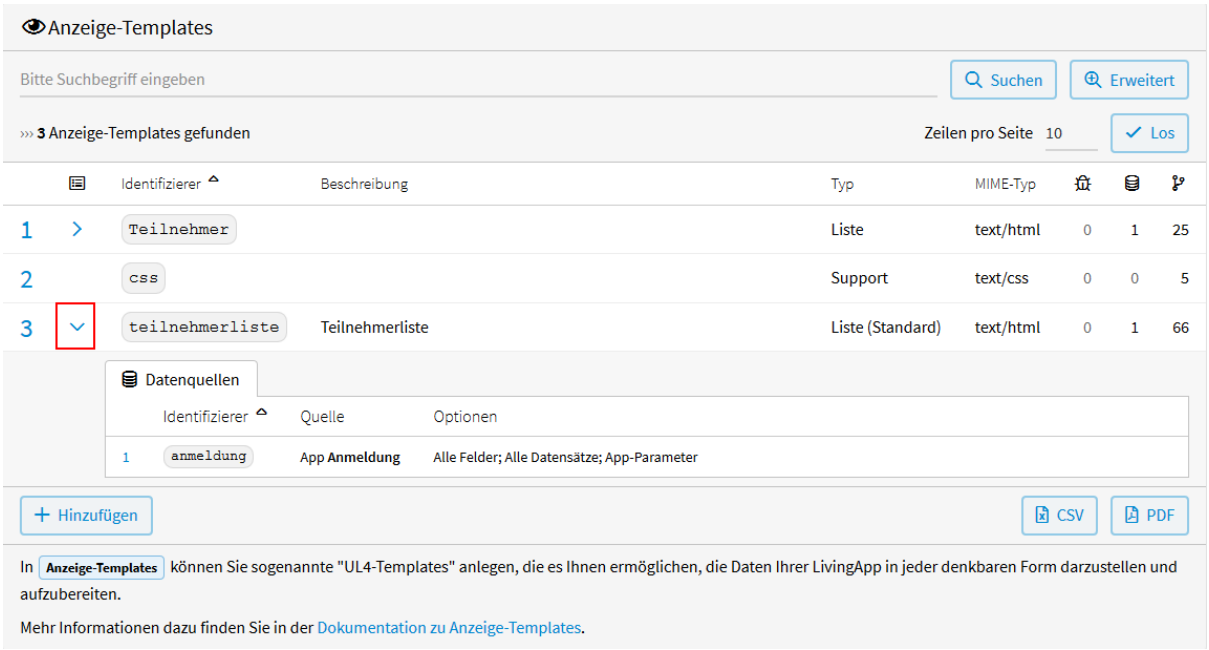


Abb. 50: Datenquellen - Ansicht in der Template-Übersicht

Wert	Beschreibung
attachment.insert	Anhang erzeugt
attachment.update	Anhang geändert
attachment.delete	Anhang gelöscht
massdata.insert	Mehrere Datensätze erzeugt
massdata.update	Mehrere Datensätze geändert
massdata.delete	Mehrere Datensätze gelöscht
data.insert	Datensatz erzeugt
data.update	Datensatz geändert
data.delete	Datensatz gelöscht
linkaction.execute	Link-Aktion ausgeführt
email.send	Email versendet
email.fail	Emailversendung fehlgeschlagen
task.insert	Aufgabe erzeugt
task.done	Aufgabe erledigt
task.delete	Aufgabe gelöscht
chat	Chat-Nachricht
app.chat	Chat-Nachricht zur App
data.chat	Chat-Nachricht zum Datensatz

So können Sie z. B. in der Datenquelle bei *Datensatz-Filter* folgende Bedingung angeben:

```
r.v_type != "data.insert"
```

Wenn ein Datensatz erzeugt wird, soll keine Aktivität angezeigt werden, ansonsten schon.

Folgende Bedingung zeigt nur die Aktivitäten zu den Chats an und keine anderen:

```
"chat" in r.v_type
```

System-App „Apps“

Bei dieser System-App werden die Apps selbst als Datensätze interpretiert. Diese Datensätze haben folgende Felder:

Name	Identifizierer	Typ
Name	`name`	`string/text`
Beschreibung	`description`	`string/text`
Kleines Icon	`icon_small`	*File*
Großes Icon	`icon_large`	*File*
Anzahl unquittierter Aktivitäten	`activity_count`	`int`

Dies ermöglicht es Ihnen z.B. ein Template zu erstellen, das alle ihre Apps anzeigt.

System-App „Controls“

System-App „Emailtemplates“

System-App „Versendungen“

2.5.4 Anwendungsbeispiel

Um die Darstellung eines Anzeige-Templates zu veranschaulichen, wird im Folgenden auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen. Es soll das Anzeige-Template *Teilnehmerliste* vom Typ *Liste* erstellt werden, welches als Übersichtsseite über sämtliche Anmeldungen dient und verschiedene Links und Parameter einbindet. Die Übersicht soll eine Tabelle anzeigen, mit den Spalten `Kurs`, `Teilnehmer`, `Anzahl Personen`, `E-Mail-Adresse` und `Bearbeiten`. Außerdem soll die Summe der angemeldeten Personen berechnet und ausgegeben werden.

Zusätzlich wird das Anzeige-Template *Ergebnis* vom Typ *Detail* (Ergebnisseite) erstellt, welches die Teilnehmer über eine erfolgreiche Anmeldung informieren soll.

Das zugehörige *CSS-Template* wird in einem separaten *Support-Template* ausgelagert und dann eingebunden.

Teilnehmerliste

Anmeldung (Computerkurse)

Kurs	Teilnehmer	Anzahl Personen	E-Mail-Adresse	Bearbeiten
Grundlagen Computerwissen (Aufbaukurs)	Beispiel Karl	2	karl.beispiel@example.org	
Grundlagen Computerwissen (Grundkurs)	Beispiel Hans	4	hans.beispiel@example.org	
Power Point Grundkurs	Muster Klara	4	klara.muster@example.org	
Power Point Grundkurs	Mustermann Max	1	max.mustermann@example.org	
Power Point Grundkurs	Sorglos Sabine	1	sabine.sorglos@example.org	
Power Point Grundkurs	Test Peter	2	peter.test@example.org	
Word für Einsteiger	Hinz Anne	1	anne.hinz@example.org	
Word für Fortgeschrittene	Mustermann Max	2	max.mustermann@example.org	

Insgesamt angemeldete Personen: 17

Anmeldung hinzufügen

Abb. 51: Anzeige-Template Teilnehmerliste

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Anzeige-Templates*. Klicken Sie auf *Hinzufügen* um ein neues Anzeige-Template anzulegen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Bei *Quelltext* wird folgender Code eingegeben.

```
<?doc?>
Teilnehmerliste
<?end doc?>
<?code teilnehmerapp = datasources.teilnehmer.app?>
<html>
   <head>
      <title>Teilnehmerliste</title>
      <?render globals.t_la_static_font_awesome()?>
      <link rel="stylesheet" href="<?printx teilnehmerapp.template_url('css')?>" />
   </head>
   <body>
      <h1><?print app.pv_ueberschrift?></h1>
      <h2>
         <?if teilnehmerapp.image is not None?>
            <img src="<?printx teilnehmerapp.image.url?>" width="32"/>
         <?end if?>
```

(Fortsetzung auf der nächsten Seite)

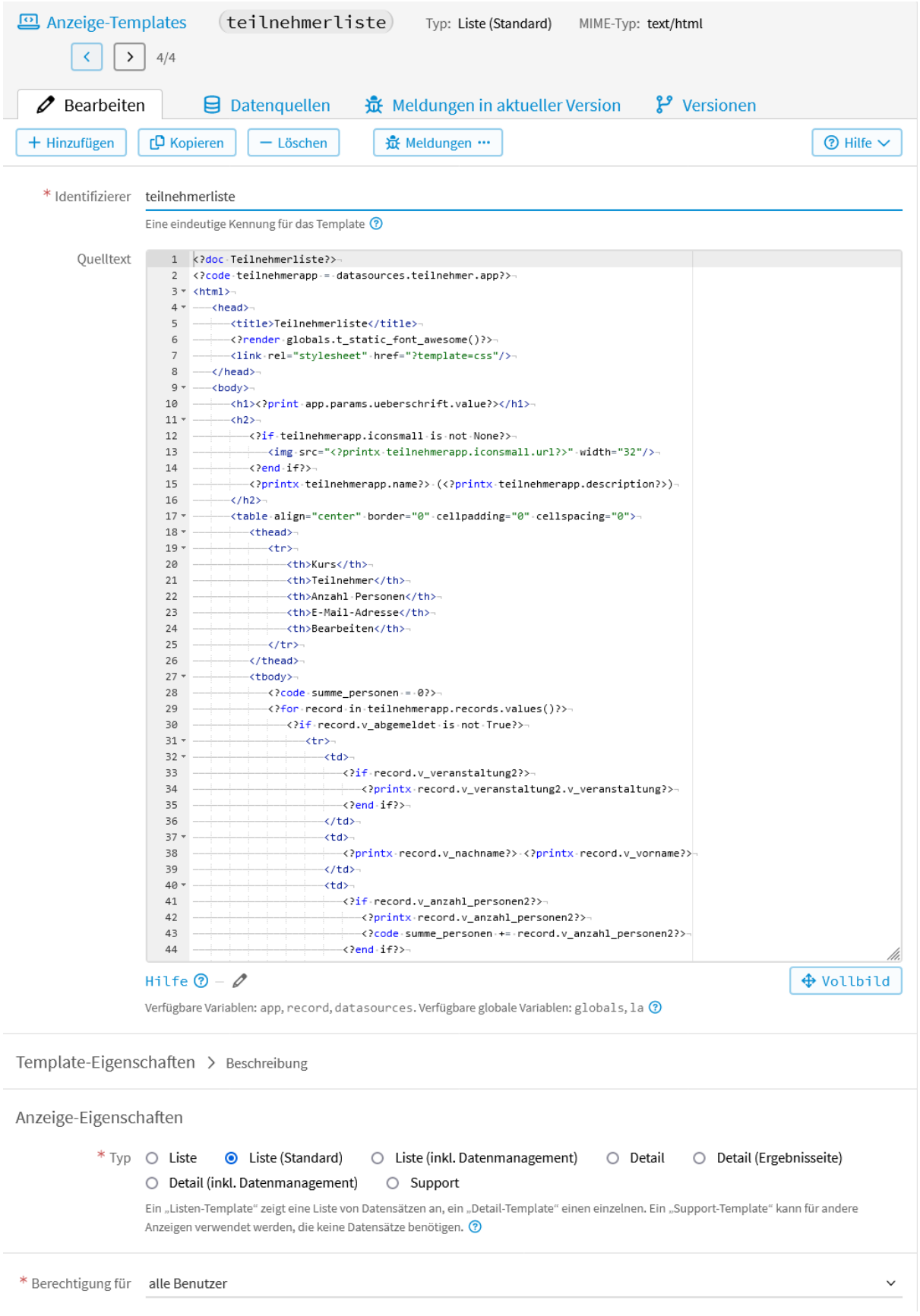


Abb. 52: Anzeige-Template Konfiguration

(Fortsetzung der vorherigen Seite)

```
17          <?printx teilnehmerapp.name?> (<?printx teilnehmerapp.description?>)
18       </h2>
19       <table align="center" border="0" cellpadding="0" cellspacing="0">
20          <thead>
21             <tr>
22                <th>Kurs</th>
23                <th>Teilnehmer</th>
24                <th>Anzahl Personen</th>
25                <th>E-Mail-Adresse</th>
26                <th>Bearbeiten</th>
27             </tr>
28          </thead>
29          <tbody>
30             <?code summe_personen = 0?>
31             <?for record in teilnehmerapp.records.values()?>
32                <?if not record.v_abgemeldet?>
33                   <tr>
34                      <td>
35                         <?if record.v_veranstaltung2?>
36                            <?printx record.v_veranstaltung2.v_veranstaltung?>
37                         <?end if?>
38                      </td>
39                      <td>
40                         <?printx record.v_nachname?> <?printx record.v_vorname?>
41                      </td>
42                      <td>
43                         <?if record.v_anzahl_personen2?>
44                            <?printx record.v_anzahl_personen2?>
45                            <?code summe_personen += record.v_anzahl_personen2?>
46                         <?end if?>
47                      </td>
48                      <td>
49                         <?printx record.v_e_mail_adresse?>
50                      </td>
51                      <td class="zentriert">
52                         <a target="_top" href="<?printx record.edit_embedded_url()?>">
↪<?render globals.t_la_icon("pencil", fw=True)?></a>
53                      </td>
54                   </tr>
55                <?end if?>
56             <?end for?>
57          </tbody>
58       </table>
59       <p>Insgesamt angemeldete Personen: <?printx summe_personen?></p>
60       <p><a target="_top" href="<?printx teilnehmerapp.new_embedded_url()?>">
↪Anmeldung hinzufügen</a></p>
61    </body>
62 </html>
```

Zeile 1
: Die Beschreibung des Templates mit dem `<?doc?>`-Tag.

Zeile 4
: Belegung der Variablen `teilnehmerapp` mit den Informationen aus der App der Datenquelle.

Zeilen 6 - 10
: Der *title* der Seite wird festgelegt. Das interne Template `la_static_font_awesome` (Zeile 8) und das ausgelagerte CSS-Template (Zeile 9) werden eingebunden.

Zeile 12
Die Überschrift des Templates wird über den App-Parameter `ueberschrift` eingebunden und ausgegeben. Über App-Parameter können Sie in der Dokumentation der *Parameter* mehr lesen.

Zeilen 14 - 16
Wenn ein App-Icon hochgeladen wurde, wird dieses hier ausgegeben.

Zeile 17
Der Name und die Beschreibung der App wird ausgegeben.

Zeile 18 - 26
Der Kopf der Tabelle wird festgelegt, mit den Spalten-Überschriften `Kurs`, `Teilnehmer`, `Anzahl Personen`, `E-Mail-Adresse` und `Bearbeiten`.

Zeile 30
Die Variable `summe_personen` wird initialisiert, zu der in jedem Datensatz die Anzahl der angemeldeten Personen aufaddiert wird.

Zeile 31
Die `for`-Schleife durchläuft die Werte der Datensätze in der Datenquelle.

Zeile 32
Der Code im `if`-Block von Zeile 32 - 55 wird nur ausgeführt für Teilnehmer, die sich nicht wieder abgemeldet haben.

Zeilen 35 - 37
Wenn eine Veranstaltung ausgewählt wurde, wird der Wert des Feldes „Veranstaltung“ aus der verknüpften App „Veranstaltungen“ ausgegeben.

Zeile 40
Der Nachname und der Vorname des Teilnehmers wird ausgegeben.

Zeile 43 - 46
Wenn das Feld `Anzahl Personen` gefüllt ist, dann wird dieser Wert hier (Zeile 44) ausgegeben und anschließend auf die in Zeile 30 definierte Variable aufsummiert (Zeile 45).

Zeile 49
Die E-Mail-Adresse des Teilnehmers wird ausgegeben.

Zeile 52
Link zum Bearbeiten des jeweiligen Datensatzes. Mit `<?render globals.t_la_icon("pencil", fw=True)?>` wählen Sie das Icon zum Editieren aus Font-Awesome aus.

Zeile 59
Ausgabe der Variablen `summe_personen`.

Zeile 60
Link zur Dateneingabe, in dem ein neuer Datensatz angelegt werden kann.

Nachdem sie das Template gespeichert haben, können Sie Datenquellen dazu anlegen. Die Datenquelle, die für dieses Anwendungsbeispiel angelegt wurde, finden Sie in der Dokumentation der *Datenquelle*.

Wie ein Template aufgerufen wird, können Sie unter *Aufruf eines Anzeige-Templates* nachlesen.

2.6 Interne Templates

In *Interne Templates* können Sie sich eine Bibliothek von „UL4-Templates“ anlegen. Diese Templates können Sie dann in Ihren E-Mail- oder Anzeige-Templates aufrufen.

Ein internes Template kann nur von anderen Templates aufgerufen werden, nicht von außen. Z. B. für eine Funktion, die sich um die Anzeige eines Feldes kümmert und mit den entsprechenden Parametern aufgerufen wird.

2.6.1 Erstellung eines Internen Templates

Um ein Internes Template für Ihre LivingApp zu erstellen, wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Interne Templates* und dann auf *Hinzufügen*.

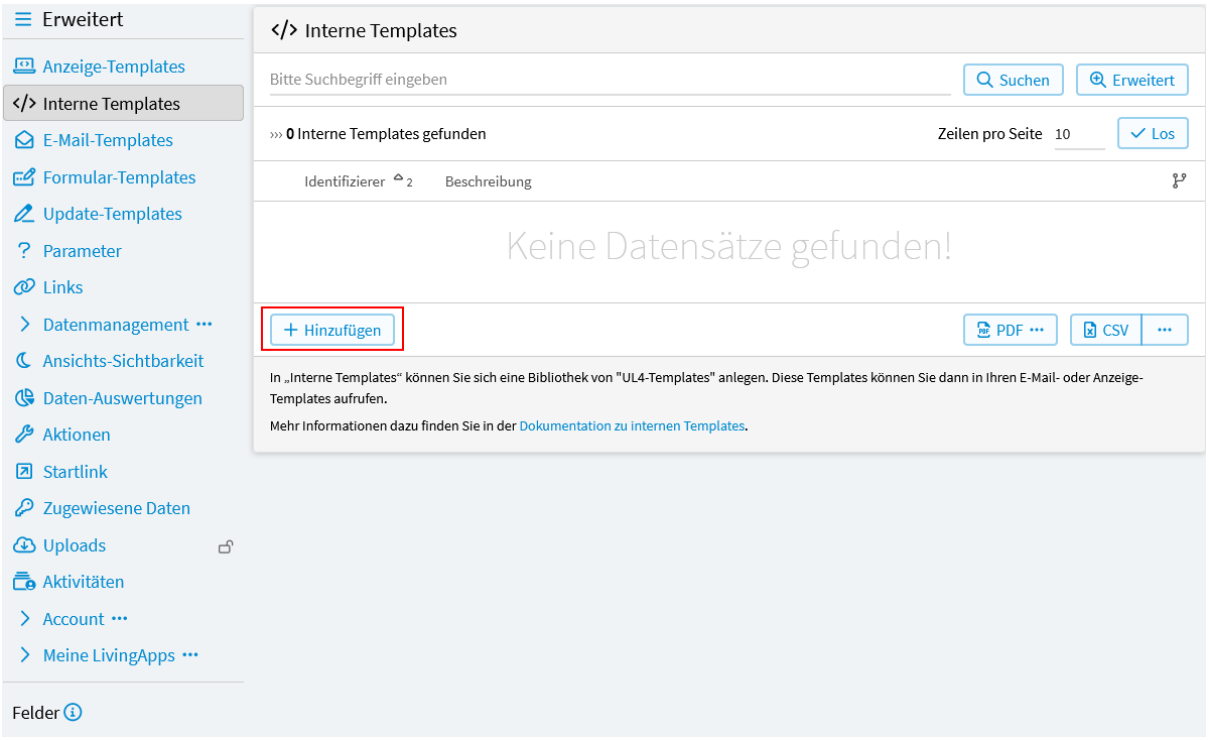


Abb. 53: Internes Template hinzufügen

Im nun geöffneten Fenster sehen Sie die *Eingabeansicht* des Internen Templates.

Identifizierer
: Der Identifizierer ist die eindeutige Kennung für das Template. Er kann verwendet werden, um das Template aufzurufen. Der Identifizierer darf nur Buchstaben, Ziffern und _ enthalten.

Quelltext
: Hier erfolgt die *Formulierung des Templates*.

Wird im Quelltext ein <?ul4>-Tag, ein <?whitespace?>-Tag oder ein <?doc?>-Tag eingegeben, dann erscheint unter *Templateeigenschaften* folgendes:

Signatur
: Die Liste der Parameter, die dieses Template erwartet (wird automatisch aus dem <?ul4?>-Tag im Template-Quelltext extrahiert). Für interne Templates sollte eine Signatur angegeben werden, damit das Template aufgerufen werden kann, ohne die Parameter per „Keyword“ übergeben zu müssen. D.h. wenn z.B. eine Signatur `<?ul4 berechnung(wert)?>` für ein Template namens `berechnung` verwendet wird, kann dieses mittels `globals.t_berechnung(42)` aufgerufen werden, statt `globals.t_berechnung(wert=42)` verwenden zu müssen.

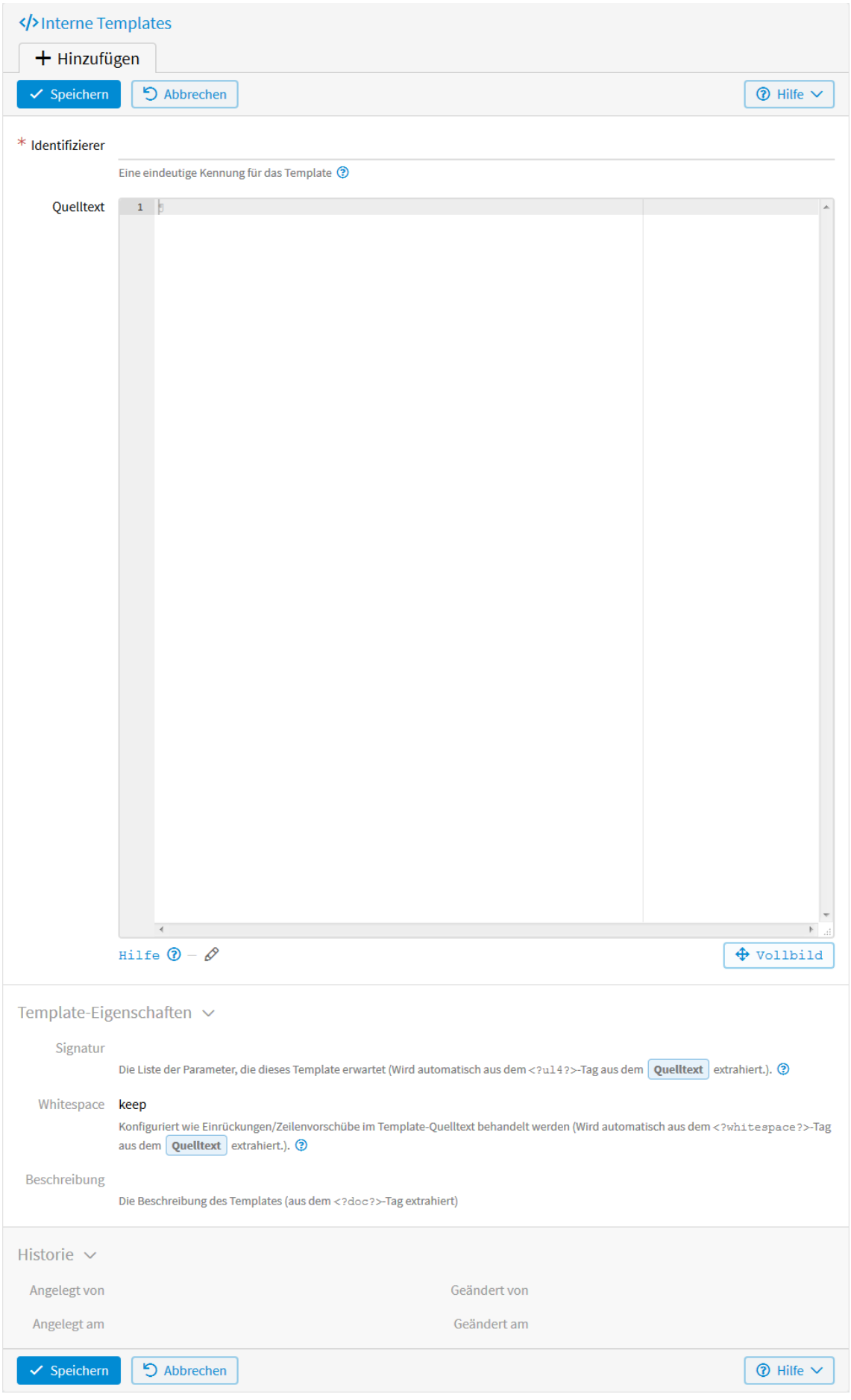


Abb. 54: Internes Template - Eingabeansicht

Whitespace
: *Whitespace* konfiguriert, wie Einrückungen/Zeilenvorschübe im Template-Quelltext behandelt werden (wird automatisch aus dem `<?whitespace?>`-Tag im Template-Quelltext extrahiert).

Beschreibung
: Beschreibung des Templates (wird automatisch aus dem `<?doc?>`-Tag im Template-Quelltext extrahiert). Z. B. `<?doc Teilnehmerliste?>` oder `<?doc?>Teilnehmerliste<?end doc?>`.

Aufgerufen werden kann das Interne Template mit dem Identifizierer `beispiel` mit:

```
<?render globals.templates.beispiel(argumente)?>
```

Shortcut:

```
<?render globals.t_beispiel(argumente)?>
```

Haben Sie z. B. eine *Datenquelle* mit dem Identifizierer `muster` angelegt und in dieser nicht die aktuelle App sondern eine andere ausgewählt, können Sie auf die internen Templates dieser App zugreifen mit:

```
<?render datasources.muster.app.templates.beispiel(argumente)?>
```

Shortcut:

```
<?render datasources.muster.app.t_beispiel(argumente)?>
```

Wobei `beispiel` jeweils für den Identifizierer des internen Templates steht.

2.6.2 Anwendungsbeispiel

Im Anwendungsbeispiel soll ein Internes Template angelegt werden, das die Font-Awesome-Icons einbindet. Diese Icons können Sie dann in Ihren Anzeige-Templates verwenden.

Um eine neues Internes Template zu erstellen, wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Interne Templates*.

Klicken Sie auf *Hinzufügen* um ein Internes Template anzulegen.

Im nun geöffneten Fenster werden folgende Konfigurationen vorgenommen.

Über den *Identifizierer* kann das Interne Template z. B. in Ihrem Anzeigetemplate aufgerufen werden mit:

```
<?render globals.templates.static_font_awesome()?>
```

Shortcut:

```
<?render globals.t_static_font_awesome()?>
```

In *Quelltext* erfolgt die *Formulierung des Templates*.

2.7 E-Mail-Templates

Mit den E-Mail-Templates können Vorlagen für E-Mail-Benachrichtigungen erstellt werden, die in bestimmten Situationen verschickt werden. Zum Beispiel kann automatisch eine E-Mail nach dem Erstellen eines Datensatzes (d. h. nach dem Absenden des Formulars) oder nach dem Ändern eines Datensatzes versendet werden. Zudem gibt es die Möglichkeit, E-Mail-Vorlagen zu erstellen, die dann manuell in der Datenliste versendet (Vorausssetzung dafür ist ein E-Mail-Feld im Formular der App) oder in der *Aktion E-Mail-Versendung* eingebunden werden können.

Sie können festlegen, an welchen Empfänger die E-Mail verschickt werden soll und wer eine Kopie der E-Mail erhält. So können beispielsweise einzelne Mitarbeiter über einen neuen Datensatz informiert und auf anfallende

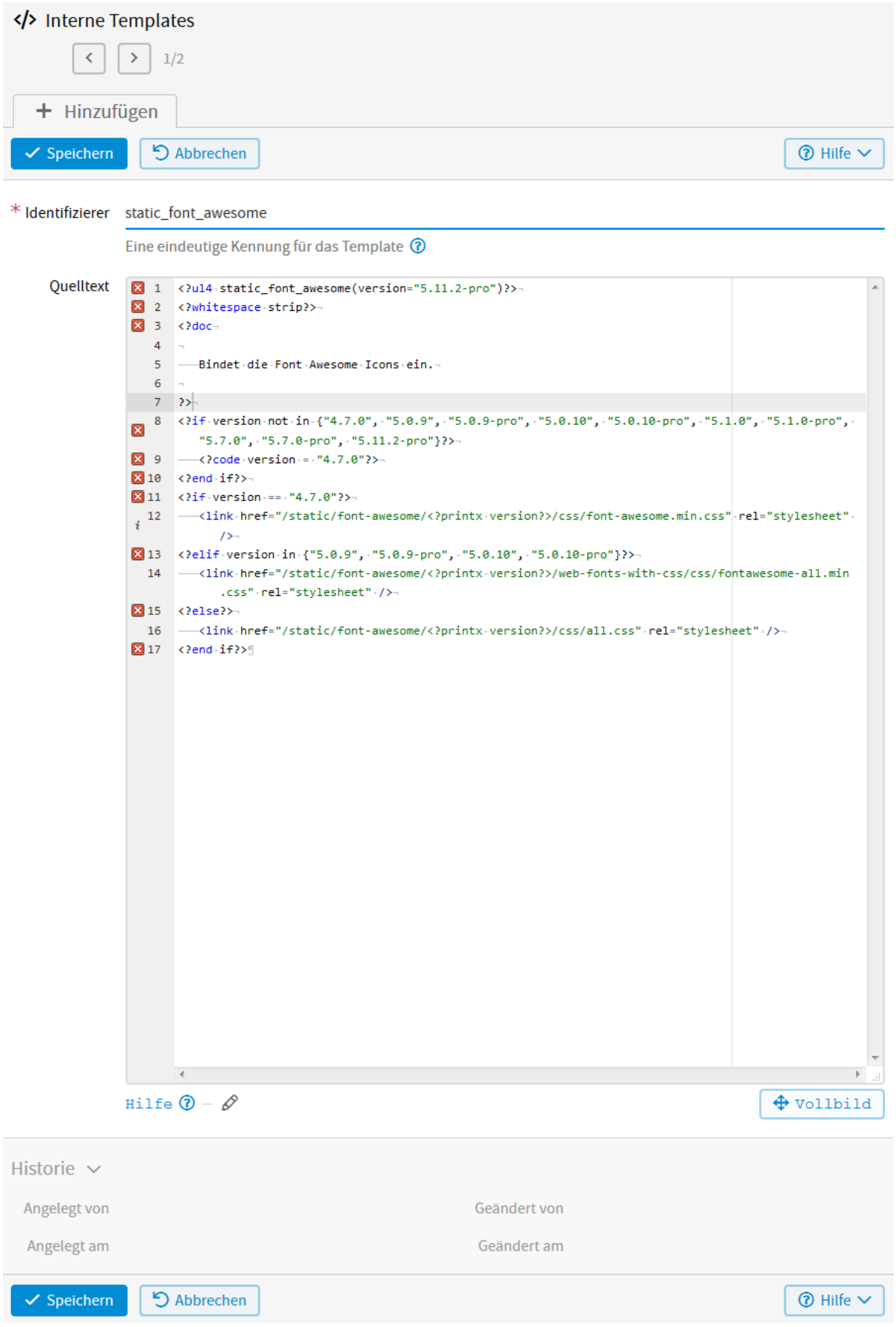


Abb. 55: Internes Template Konfiguration

Aufgaben hingewiesen werden. Die E-Mail-Funktion kann auch genutzt werden, um dem Absender des Formulars eine Empfangsbestätigung zu schicken, in der z. B. auch seine angegebenen Daten nochmals aufgelistet werden.

Innerhalb der E-Mail kann auf die einzelnen Felder des jeweiligen Datensatzes und wenn Sie Datenquellen anlegen, auch auf die Felder anderer Apps zugegriffen werden. So können wichtige Informationen in den Inhalt der Benachrichtigung integriert werden und der Empfänger erhält in der E-Mail einen Überblick über die neuen oder geänderten Daten.

2.7.1 Erstellung eines E-Mail-Templates

Um ein neues E-Mail-Template zu erstellen, wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken anschließend in der linken Menüleiste auf *E-Mail-Templates* und dann auf *Hinzufügen*.

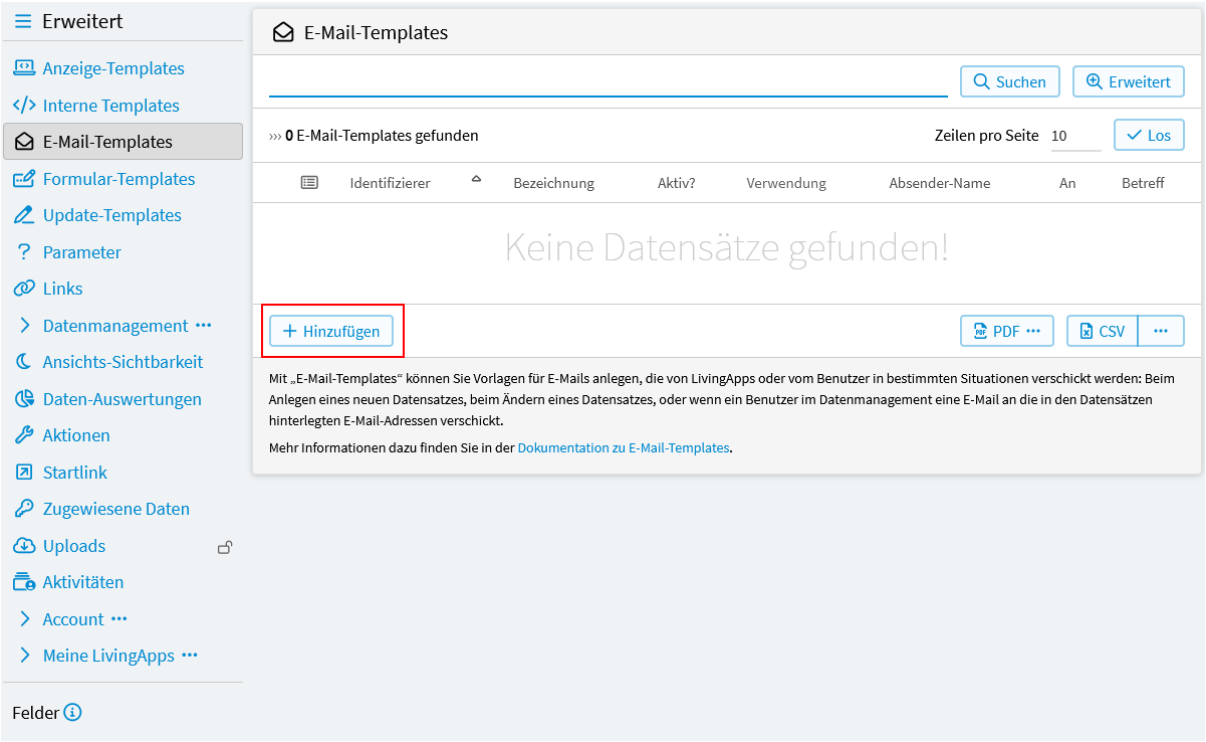


Abb. 56: E-Mail-Template hinzufügen

Im nun geöffneten Fenster sehen Sie die *Eingabeansicht* des E-Mail-Templates.

Identifizierer
: Der Identifizierer ist eine eindeutige Kennung für das Template und darf nur Buchstaben, Zahlen und _ enthalten.

Bezeichnung
: Hier können Sie eine Beschreibung des E-Mail-Templates eingeben, um eine bessere Übersicht über mehrere Templates zu haben. Unter dieser Bezeichnung finden Sie das E-Mail-Template, wenn Sie es aus der Datenliste manuell verschicken wollen.

Aktiv
: Es werden nur die E-Mail-Templates ausgeführt, bei denen hier das Häkchen gesetzt wurde.

Verwendung
: Hier können Sie festlegen, wann eine E-Mail verschickt werden soll.

  Bei neuem Datensatz
  : Versendet eine Benachrichtigung, wenn ein neuer Datensatz angelegt wurde.

  Beim Bearbeiten eines Datensatzes
  : Versendet eine Benachrichtung, wenn ein bestehender Datensatz verändert wurde.

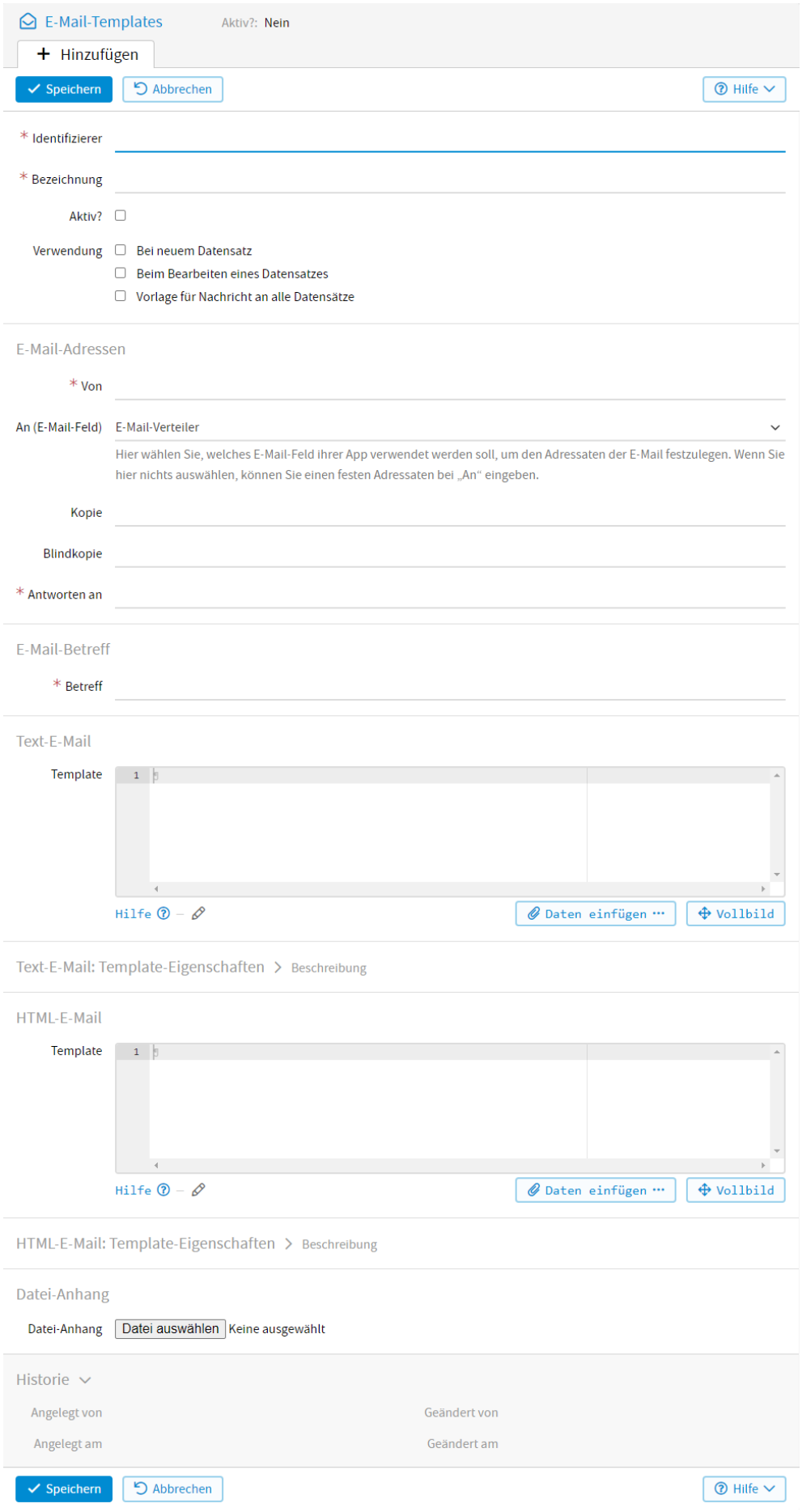


Abb. 57: E-Mail-Template Eingabeansicht

Vorlage für Nachricht an alle Datensätze
Hier können E-Mail-Vorlagen erstellt werden, die dann aus dem Datenmanagement manuell, oder über eine Aktion *(E-Mail-Versendung)* verschickt werden.

E-Mail-Header:

Von
Geben Sie hier den Namen ein, der dem Empfänger der E-Mail als Absender angezeigt werden soll.

An (E-Mail-Feld)
Empfänger der E-Mail. Gibt es im Formular Ihrer App ein oder mehrere Felder vom Typ „E-Mail“, können Sie hier das entsprechende auswählen. Dieses E-Mail-Feld sollte im Formular ein Pflichtfeld sein, sonst kann an die Personen, die keine E-Mail-Adresse angeben, z. B. keine Anmeldebestätigung verschickt werden. Möchten Sie feste Adressaten eingeben, wählen Sie hier *(Nichts ausgewählt)* und geben die E-Mail-Adresse(n) bei *An* ein.

An
Haben Sie bei *An (E-Mail-Feld) (Nichts ausgewählt)* gewählt oder gibt es im Formular Ihrer App kein E-Mail-Feld, können Sie hier feste Adressaten eingeben. Soll die E-Mail an mehrere Empfänger versendet werden, listen Sie hier deren E-Mail-Adressen kommagetrennt auf.

Kopie
Hier können Sie weitere Empfänger angeben. Wenn Sie mehr als eine E-Mail-Adresse eingeben wollen, dann trennen Sie diese durch Kommas.

Blindkopie
An die hier aufgelisteten, kommagetrennten Empfänger wird eine Blindkopie der E-Mail versendet.

Antworten an
Der Empfänger der Benachrichtigung kann auf die E-Mail antworten. Hier legen Sie fest, an welche E-Mail-Adresse die Antwort verschickt werden soll.

Betreff
Schreiben Sie hier den Betreff Ihrer E-Mail.

E-Mail-Header - letzte Änderungsoptionen

Sie haben die Möglichkeit, im Quelltext des E-Mail-Templates den Inhalt der E-Mail-Header noch ein letztes Mal zu verändern. Die hier gemachten Angaben haben Vorrang vor den oben gemachten Einträgen. Durch eigene programmierte Bedingungen können Sie z. B. Einfluss darauf nehmen, wann, ob und an wen eine E-Mail verschickt wird.

Beispiele:

Möchten Sie z. B. nicht, dass als Absender der E-Mail „Max Mustermann <benachrichtigung@living-apps.de>“ steht oder möchten Sie einen eigenen SMTP-Server verwenden, schreiben Sie folgendes in den Quelltext Ihres E-Mail-Templates:

```
<?code globals.response.from = "Max Mustermann <mm@beispiel.de>"?>
```

Soll die E-Mail z. B. an unterschiedliche E-Mail-Adressen eines Empfängers gehen, geben Sie folgendes ein:

```
<?code to = record.v_e_mail?>
<?if record.v_email2?>
   <?code to += "," + record.v_email2?>
<?end if?>
<?if record.v_email3?>
   <?code to += "," + record.v_email3?>
<?end if?>
<?code globals.response.to = to?>
```

Wobei `e_mail`, `email2` und `email3` die jeweiligen Identifizierer der E-Mail-Felder sind. An dieser Stelle hätten Sie auch die Möglichkeit, eine Versendung der E-Mail noch zu stoppen, wenn bestimmte eigene programmierte Bedingungen nicht erfüllt wären.

Sind die E-Mail-Adressen der Empfänger nicht direkt in der App verfügbar, sondern stehen im Verteiler einer verknüpften App, können Sie darauf zugreifen mit:

```
<?code globals.response.to = record.v_verteiler.v_email?>
```

Wobei `verteiler` und `email` die Identifizierer der jeweiligen Felder sind.

Eine Kopie der E-Mail soll nur dann verschickt werden, wenn bei *Wird Übernachtung benötigt? Ja* gewählt wurde:

```
<?if record.v_wird_uebernachtung_benoetigt.key == "ja"?>
   <?code globals.response.cc = "zimmerbuchung@livinglogic.de"?>
<?else?>
   <?code globals.response.cc = None?>
<?end if?>
```

Dasselbe gilt auch für die Blindkopie (bcc).

Auch was in *Antworten an* stehen soll, kann hier an Bedingungen geknüpft werden:

```
<?if record.v_veranstaltung2.v_veranstaltung == "powerpoint"?>
   <?code globals.response.replyto = "beispiel@livinglogic.de"?>
<?else?>
   <?code globals.response.replyto = record.v_zustaendiger_mitarbeiter?>
<?end if?>
```

Wenn Sie keinen festen Betreff eingeben, sondern hier auf die Daten Ihrer App zugreifen möchten, könnten Sie das folgendermaßen tun:

```
<?code globals.response.subject = record.v_betreff?>
```

Wobei `betreff` für den Identifizierer des Feldes steht.

Mithilfe folgender Variablen können Sie z. B. ein E-Mail-Rahmentemplate erstellen, das nur die Anrede und die Signatur enthält. Der Text wird dann über die Varianblen eingefügt.

Mit `<?print globals.request.bodyhtml?>` wird als E-Mail-Text ausgegeben, was in der Daten-Liste unter *E-Mail versenden* bei *Nachricht* eingegeben wird oder bei einer *Aktion* mit dem Anweisungstyp *E-Mail-Versendung* im Feld *Inhalt (HTML)*.

Mit `<?print globals.request.bodytext?>` wird als E-Mail-Text ausgegeben, was in einer *Aktion* mit dem Anweisungstyp *E-Mail-Versendung* im Feld *Inhalt (Text)* eingegeben wird.

Bespiel für ein E-Mail-Rahmentemplate:

```
Sehr geehrte(r) <?printx record.v_vorname?> <?printx record.v_nachname?>,

<?print globals.request.bodyhtml?>

Mit freundlichen Grüßen
Ihr Veranstaltungsteam

Signatur
...
```

E-Mail-Inhalt bereitstellen

Die Daten Ihrer App sind über die *LivingAPI* zugänglich. Im zugehörigen Kapitel sind die Objekte ausführlich beschrieben. Haben Sie zusätzlich zum Template eine *Datenquelle* konfiguriert, können Sie über diese z. B. auf *App-Parameter* zugreifen, die zur Verfügung stehenden Daten im Vorhinein einschränken und Daten mehrerer Apps ausgeben.

Den Inhalt Ihrer E-Mail können sie als einfachen Text bei *Text-E-Mail* und/oder als HTML-Code bei *HTML-E-Mail* eingeben. Außerdem stehen jeweils unter *Daten einfügen…* E-Mail-Vorschläge zur Verfügung. Bei Verwendung dieser Vorschläge werden alle Felder Ihrer App, bei denen Eingaben gemacht wurden, entsprechend ihres Typs, richtig ausgegeben.

Template Text-E-Mail
: Schreiben Sie den Inhalt als reine Text-E-Mail, werden die Absätze so in die E-Mail übernommen, wie sie hier eingegeben wurden. Arbeiten Sie mit UL4-Ausdrücken oder möchten Sie andere Templates einbinden, empfiehlt es sich zur besseren Darstellung, einen *<?whitespace?>-Tag* ins Template einzufügen. In der Text-E-Mail haben Sie keine Formatierungsmöglichkeiten.

Template HTML-E-Mail
: Möchten Sie Ihren E-Mail-Text formatieren und mit CSS gestalten, schreiben Sie hier den E-Mail-Inhalt in HTML-Tags.

Datei-Anhang
: Hier können Sie eine Datei auswählen, die an Ihre automatische E-Mail-Benachrichtigung angehängt wird. Achten Sie hier auf eine möglichst geringe Dateigröße.

Zugriff auf Elemente des Datensatzes und Ausgabe von Feldwerten

Die Datensätze der jeweiligen App sind im *Record*-Dictionary der *App* gespeichert. Um auf einzelne Elemente des Datensatzes zuzugreifen und diese in Ihrer E-Mail auszugeben, benötigen Sie, wie in Kapitel 1 erläutert, deren *Identifizierer*. Diese finden Sie in der linken Menüleiste unter *Felder*.

Ist Ihnen der Identifizierer des jeweiligen Feldes bekannt, so können Sie auf dieses mit Punktnotation zugreifen:

```
<?record.fields.identifier.value?>
```

Die Werte stehen auch über „Shortcut“-Attribute zur Verfügung. `record.fields.identifier` steht auch direkt als `record.f_identifier` zur Verfügung. `record.fields.identifier.value` steht auch direkt als `record.v_identifier` zur Verfügung. Wobei `identifier` jeweils für den Identifizierer des Feldes steht, auf das Sie zugreifen möchten.

Um Elemente des Datensatzes in der Benachrichtigungs-E-Mail auszugeben, benutzen Sie den Befehl `printx`:

```
<?printx record.fields.vorname.value?>
```

Shortcut:

```
<?printx record.v_vorname?>
```

greift auf den Wert des Text-Feldes `Vorname` zu und gibt den eingegebenen Vornamen des betrachteten Datensatzes (z. B. Max) aus. Hierbei gilt es zu beachten, dass `vorname` der Identifizierer des Feldes `Vorname` ist.

```
<?printx record.fields.wird_uebernachtung_benoetigt.value.label?>
```

Shortcut:

```
<?printx record.v_wird_uebernachtung_benoetigt.label?>
```

greift auf den Wert des Lookup-Feldes `Wird Übernachtung benötigt` zu und gibt die Bezeichnung der gewählten Option („Ja“ oder „Nein“) aus. :

```
<?printx record.fields.veranstaltung2.value.fields.veranstaltung.value?>
```

Shortcut:

```
<?printx record.v_veranstaltung2.v_veranstaltung?>
```

greift auf den Wert des Applookup-Feldes `Veranstaltung` zu und gibt den Wert des Feldes `Veranstaltung` (z. B. Excel Grundkurs) der verknüpften App aus.

Mehr darüber können Sie in der Dokumentation der Anzeigetemplates unter *Formulierung des Templates* nachlesen. In der *Übersicht der Feldtypen* erfahren Sie mehr darüber, wie die einzelnen Feldtypen in Templates ausgegeben werden können.

Bevor Sie den Inhalt eines Formularfeldes in der E-Mail ausgeben, sollten Sie prüfen, ob ein Inhalt eingegeben wurde. Einige Felder Ihres Formulars könnten nicht-verpflichtende Felder sein. Somit ist nicht sichergestellt, dass die Ausgabe des Feldinhaltes möglich ist.

Deshalb ist es sinnvoll, Ausgaben immer mit einer `if`-Bedingung zu umschließen, die prüft, ob eine Angabe im betrachteten Feld vorhanden ist. Für das Formularfeld mit dem Identifizierer `vorname` sieht diese `if`-Bedingung bespielsweise so aus:

```
<?if record.fields.vorname.value?>
   <?printx record.fields.vorname.value?>
<?end if?>
```

Shortcut:

```
<?if record.v_vorname?>
   <?printx record.v_vorname?>
<?end if?>
```

Im Kapitel *Prüfung von Feldwerten auf Inhalt* können Sie mehr darüber nachlesen.

Einbinden eines Links in das E-Mail-Template

Haben Sie unter *Aktionen* einen *Link für E-Mails* angelegt, gibt es zwei Möglichkeiten diesen Link in Ihr E-Mail-Template einzubinden. Zum einen über den Link mit `{actionlink:identifier}` und zum anderen über die URL mit `{actionurl:identifier}`. Wobei `identifier` für den Identifizierer der Aktion steht.

`{actionlink:identifier}`
: In der HTML-Email wird daraus: `<a href="http://lapp.io/1234567">Aktions-Beschriftung</a>`

  In der Text-Email wird daraus: `(Aktions-Beschriftung)[http://lapp.io/1234567]`

`{actionurl:identifier}`
: In beiden Fällen wird daraus: `http://lapp.io/1234567`

Haben Sie bei den *Anzeige-Template*s ein *Detailtemplate* angelegt und möchten in Ihrem E-Mail-Template auf dieses Detailtemplate zugreifen, gibt es auch hier die folgenden zwei Möglichkeiten. Wobei hier `identifier` für den Identifizierer des Detailtemplates steht.

`{detaillink:identifier}`
: In der HTML-Email wird daraus: `<a href="http://lapp.io/1234567">Detailtemplate-Identifizierer</a>`

  In der Text-Email wird daraus: `(Detailtemplate-Identifizierer)[http://lapp.io/1234567]`

`{detailurl:identifier}`
: In beiden Fällen wird daraus: `http://lapp.io/1234567`

Einbinden eines Bildes in der HTML-E-Mail

Sie können in der HTML-Version des E-Mail-Templates Bilder einbinden, dies könnte beisipielsweise folgendermaßen aussehen:

```
<img src="https://<?printx globals.hostname?><?printx file.url?>" />
```

Aus Datenschutzgründen wird jedoch dieses Bild möglicherweise vom E-Mail-Program nicht angezeigt, da ein Abrufen des Bildes von einem Webserver es möglich macht, nachzuvollziehen, daß die E-Mail gelesen wurde.

Daher können Sie das E-Mail-Template dazu veranlassen, daß Bild direkt in die E-Mail einzubetten. Dies geschieht über das HTMl-Attribute `data-embed`. Folgende Werte sind bei diesem Attribut zulässig:

`dataurl`
: Das Bild wird per `data`-URL eingebunden. Das empfielt sich nur, wenn das Bild nur an einer Stelle verwendet wird.

  In diesm Fall müßten Sie folgendes im E-Mail-Template schreiben:

  ```
  <img data-embed="dataurl" src="https://<?printx globals.hostname?><?printx file.
  ↪url?>" />
  ```

`attachment`
: Das Bild wird als zusätzliches Attachment an die Mail gehängt, und im HTML wird darauf verwiesen.

  In diesm Fall müßten Sie folgendes im E-Mail-Template schreiben:

  ```
  <img data-embed="attachment" src="https://<?printx globals.hostname?><?printx␣
  ↪file.url?>" />
  ```

`external` (oder auch jeder andere Wert)
: Das Bid wird wie bisher vom externen Server geholt.

Datenquellen

Nachdem Sie das E-Mail-Template abgespeichert haben, können Sie Datenquellen dazu anlegen. Über die Datenquelle können Sie z. B. auf App-Parameter zugreifen, die Daten mehrerer Apps ausgeben und die zur Verfügung stehenden Daten im Vorhinein einschränken. Mehr darüber können Sie in der Dokumentation der Anzeigetemplates unter *Anlegen von Datenquellen* und weiter unten im *Anwendungsbeispiel* nachlesen.

2.7.2 Anwendungsbeispiel

Beim Eingang einer neuen Anmeldung zu einer Veranstaltung, soll der Teilnehmer / die Teilnehmerin automatisch eine Anmeldebestätigung erhalten. Diese E-Mail soll in der Anrede den entsprechenden Namen der Person und in der Einleitung die gewählte Veranstaltung ausgeben. Außerdem sollen alle vom Teilnehmer gemachten Eingaben aufgelistet werden. Zusätzlich enthält die E-Mail einen *Linkf für E-Mails*, über den sich der Teilnehmer wieder von der Veranstaltung abmelden kann.

Am Ende der Anmeldebestätigung sollen noch die neuen Veranstaltungen 2020 zur Information aufgelistet werden. Weil nicht nur auf die Veranstaltung des Datensatzes zugegriffen werden soll, sondern auf alle Veranstaltungen, muss eine *Datenquelle* dafür angelegt werden.

Die *Anmeldebestätigung* soll folgendermaßen aussehen:

Um ein neues E-Mail-Template anzulegen, wählen Sie im Menü *Konfiguration* → *Erweitert* und dann *E-Mail-Templates*. Klicken Sie anschließend auf *Hinzufügen*.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen werden.

Sehr geehrte(r) Max Mustermann,

vielen Dank für die Anmeldung zu unserer Veranstaltung: **Word für Einsteiger**.

Wir haben folgende Informationen von Ihnen erhalten:

Name:	Max Mustermann
Straße Hsnr.:	Musterstraße 3
PLZ Ort:	88888 Musterstadt
E-Mail:	max@example.org
Anzahl Personen:	2
Übernachtung:	Ja
Einzelzimmer:	2
Euro / Nacht:	76

Wir werden uns um die Reservierung Ihres Zimmers / Ihrer Zimmer kümmern.

Rechtzeitig vor Veranstaltungsbeginn erhalten Sie noch weitere Informationen von uns.

Sollten Sie doch nicht teilnehmen wollen, klicken Sie bitte hier: Von Veranstaltung abmelden

Dies ist eine automatisch generierte E-Mail.

Mit freundlichen Grüßen
Ihr Veranstaltungsteam

Abb. 58: Anmeldebestätigung (HTML-Beispiel)

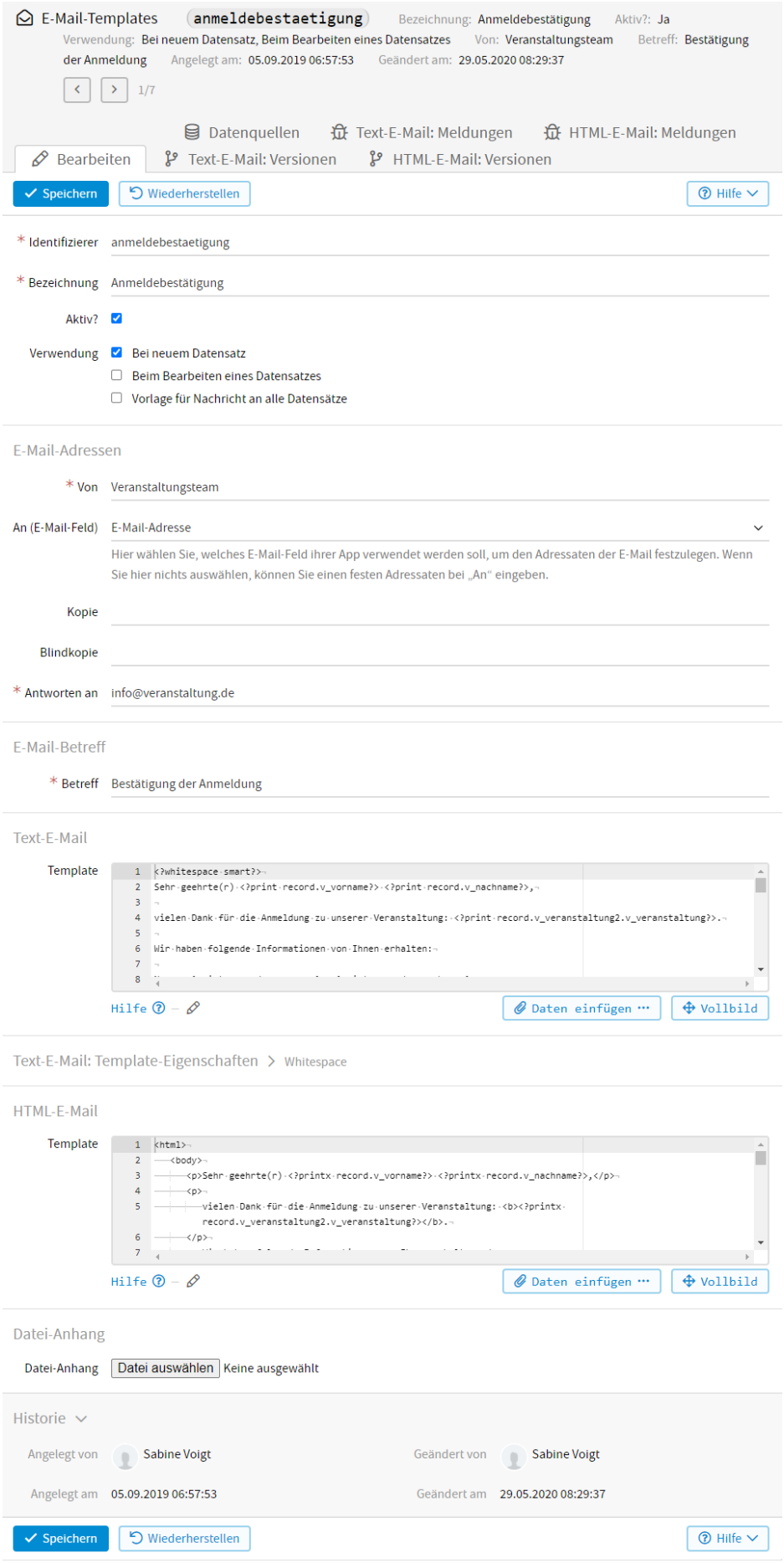


Abb. 59: E-Mail-Template Konfiguration

Unter der *Bezeichnung* `Anmeldebestätigung` finden sie das E-Mail-Template, wenn sie es z. B. aus der Datenliste manuell verschicken möchten. Da die Anmeldebestätigung bei jeder Neuanmeldung verschickt werden soll, wird bei *Verwendung Bei neuem Datensatz* ausgewählt.

Bei *An (E-Mail-Feld)* wurde *E-Mail-Adresse* gewählt, weil die Anmeldebestätigung an die Adresse gehen soll, die an dieser Stelle vom Teilnehmer im Anmeldeformular eingegeben wurde.

Folgendes Template wurde unter *HTML-E-Mail* angelegt:

```
<html>
<body>
   <p>Sehr geehrte(r) <?printx record.v_vorname?> <?printx record.v_nachname?>,</p>
   <p>
      vielen Dank für die Anmeldung zu unserer Veranstaltung: <b><?printx record.v_
↪veranstaltung2.v_veranstaltung?></b>.
   </p>
   <p>Wir haben folgende Informationen von Ihnen erhalten:</p>
   <table>
      <tr>
         <td>Name:</td>
         <td><?printx record.v_vorname?> <?printx record.v_nachname?></td>
      </tr>
      <tr>
         <td>Straße Hsnr.:</td>
         <td><?printx record.v_strasse_und_hsnr2?></td>
      </tr>
      <tr>
         <td>PLZ Ort:</td>
         <td><?printx record.v_plz_ort?></td>
      </tr>
      <tr>
         <td>E-Mail:</td>
         <td><?printx record.v_e_mail_adresse?></td>
      </tr>
      <?if record.v_telefon?>
         <tr>
            <td>Telefon:</td>
            <td><?printx record.v_telefon?></td>
         </tr>
      <?end if?>
      <tr>
         <td>Anzahl Personen:</td>
         <td><?printx record.v_anzahl_personen2?></td>
      </tr>
      <?if record.v_wird_uebernachtung_benoetigt.key == "ja"?>
         <tr>
            <td>Übernachtung:</td>
            <td>Ja</td>
         </tr>
         <?if record.v_einzelzimmer?>
            <tr>
               <td>Einzelzimmer:</td>
               <td><?printx record.v_einzelzimmer?></td>
            </tr>
         <?end if?>
         <?if record.v_euro_nacht?>
            <tr>
               <td>Euro / Nacht:</td>
               <td><?printx record.v_euro_nacht?></td>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
            </tr>
         <?end if?>
         <?if record.v_mehrbettzimmer?>
            <tr>
               <td>Doppelzimmer:</td>
               <td><?printx record.v_mehrbettzimmer?></td>
            </tr>
         <?end if?>
         <?if record.v_euro_nacht2?>
            <tr>
               <td>Euro / Nacht:</td>
               <td><?printx record.v_euro_nacht2?></td>
            </tr>
         <?end if?>
            <tr>
               <td colspan="2">Wir werden uns um die Reservierung Ihres Zimmers /␣
↪Ihrer Zimmer kümmern.</td>
            </tr>
      <?else?>
         <tr>
            <td>Übernachtung:</td>
            <td>Nein</td>
         </tr>
      <?end if?>
   </table>
   <p>Rechtzeitig vor Veranstaltungsbeginn erhalten Sie noch weitere Informationen␣
↪von uns.</p>
   <p>Sollten Sie doch nicht teilnehmen wollen, klicken Sie bitte hier:
↪{actionlink:abmelden}</p>
   <p>Dies ist eine automatisch generierte E-Mail.</p>
   <p>
      Mit freundlichen Grüßen <br />
      Ihr Veranstaltungsteam
   </p>
   </body>
</html>
```

Mit der `if`-Bedingung wird überprüft, ob das jeweilige Feld gefüllt ist. Wenn ja, wird der Wert mit `printx` ausgegeben. Diese Überprüfung kann bei Pflichtfeldern weggelassen werden, da sie ja gefüllt sein müssen. Die Auflistung der vom Teilnehmer gemachten Angaben wird in einer Tabelle dargestellt.

Nur wenn bei *Wird Übernachtung benötigt? Ja* gewählt wurde, sollen die Zimmer, Euro/Nacht und der Hinweis zur Reservierung aufgeführt werden. Mit *{actionlink:abmelden}* wird der Link eingebunden, über den sich der Teilnehmer/die Teilnehmerin wieder von der Veranstaltung abmelden kann. Mehr darüber können Sie in der Dokumentation der Aktionen unter *Link für E-Mails* nachlesen.

Als *Text-E-Mail* sieht das Template folgendermaßen aus:

```
<?whitespace smart?>
Sehr geehrte(r) <?print record.v_vorname?> <?print record.v_nachname?>,

vielen Dank für die Anmeldung zu unserer Veranstaltung: <?print record.v_
↪veranstaltung2.v_veranstaltung?>.

Wir haben folgende Informationen von Ihnen erhalten:

Name: <?print record.v_vorname?> <?print record.v_nachname?>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
Straße Hsnr.: <?print record.v_strasse_und_hsnr2?>
PLZ Ort: <?print record.v_plz_ort?>
E-Mail: <?print record.v_e_mail_adresse?>
<?if record.v_telefon?>
   Telefon: <?print record.v_telefon?>
<?end if?>
Anzahl Personen: <?print record.v_anzahl_personen2?>
<?if record.v_wird_uebernachtung_benoetigt.key == "ja"?>
   Übernachtung: Ja
   <?if record.v_einzelzimmer?>
      Einzelzimmer: <?print record.v_einzelzimmer?>
   <?end if?>
   <?if record.v_euro_nacht?>
      Euro / Nacht: <?print record.v_euro_nacht?>
   <?end if?>
   <?if record.v_mehrbettzimmer?>
      Doppelzimmer: <?print record.v_mehrbettzimmer?>
   <?end if?>
   <?if record.v_euro_nacht2?>
      Euro / Nacht: <?print record.v_euro_nacht2?>
   <?end if?>
   Wir werden uns um die Reservierung Ihres Zimmers / Ihrer Zimmer kümmern.
<?else?>
   Übernachtung: Nein
<?end if?>

Rechtzeitig vor Veranstaltungsbeginn erhalten Sie noch weitere Informationen von uns.

Sollten Sie doch nicht teilnehmen wollen, klicken Sie bitte hier: {actionurl:abmelden}

Dies ist eine automatisch generierte E-Mail.

Mit freundlichen Grüßen

Ihr Veranstaltungsteam
```

Folgendermaßen sieht die Anmeldebestätigung als reine Text-E-Mail aus:

Datenquelle

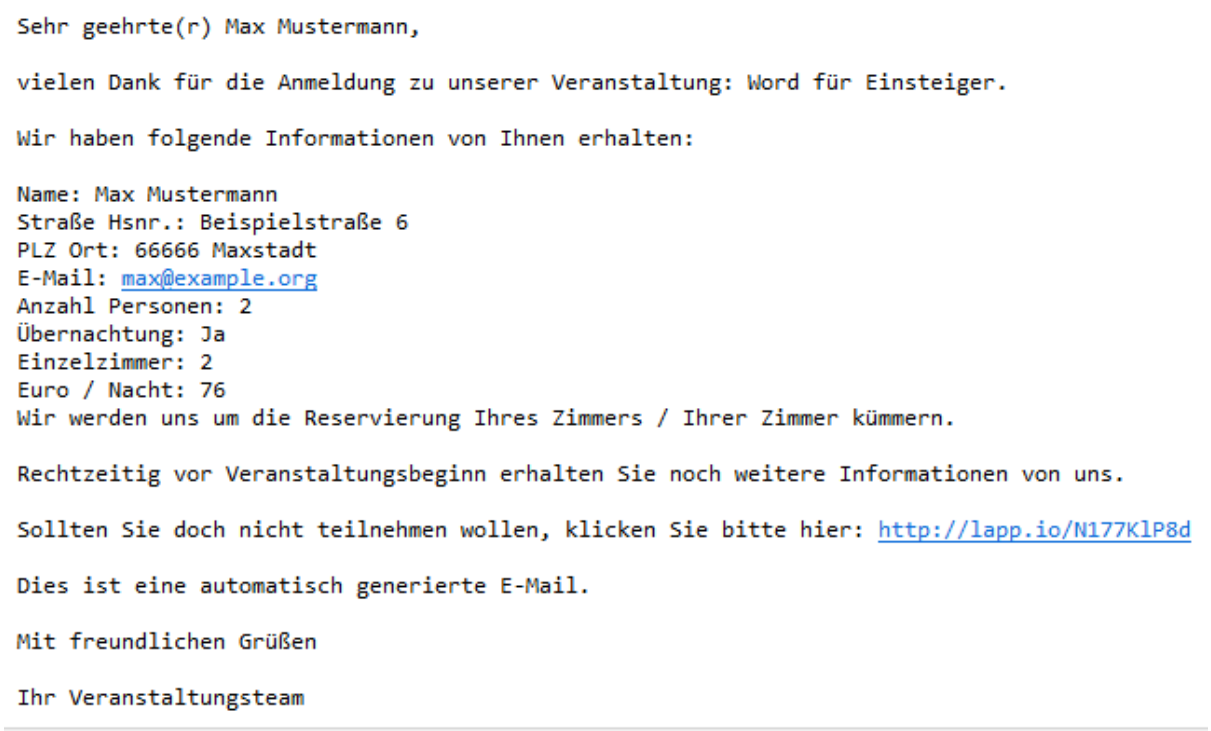
In unserem Beispiel soll dem Teilnehmer/der Teilnehmerin in der Anmeldebestätigung eine Übersicht der neuesten Veranstaltungen zur Information aufgelistet werden. Um auf alle Veranstaltungen der App zugreifen zu können, müssen Sie dafür eine Datenquelle anlegen. Nachdem Sie Ihr E-Mail-Template das erste Mal gespeichert haben, erscheint neben dem *Bearbeiten* - Button der Button *Datenquelle*.

Um eine Datenquelle anzulegen, klicken Sie auf *Datenquelle* und dann auf *Hinzufügen*. In unserem Beispiel wurden folgende Konfigurationen vorgenommen.

Über den *Identifizierer* `veranstaltungen` können Sie im E-Mail-Template auf die Datenquelle zugreifen. Bei *App* wählen Sie die App aus, auf deren Daten Sie zugreifen wollen. In unserem Beispiel ist es die App *Veranstaltungen (Doku)*. Da nur die Veranstaltungen ab 2020 aufgelistet werden sollen, muss folgender *Datensatzfilter* eingegeben werden:

```
r.v_datum.year > 2019
```

Eine ausführliche Beschreibung zu den *Datenquellen* und zur *Formulierung der Bedingung* können Sie in der Dokumentation der Anzeigetemplates unter Datenquelle nachlesen.



```
Sehr geehrte(r) Max Mustermann,

vielen Dank für die Anmeldung zu unserer Veranstaltung: Word für Einsteiger.

Wir haben folgende Informationen von Ihnen erhalten:

Name: Max Mustermann
Straße Hsnr.: Beispielstraße 6
PLZ Ort: 66666 Maxstadt
E-Mail: max@example.org
Anzahl Personen: 2
Übernachtung: Ja
Einzelzimmer: 2
Euro / Nacht: 76
Wir werden uns um die Reservierung Ihres Zimmers / Ihrer Zimmer kümmern.

Rechtzeitig vor Veranstaltungsbeginn erhalten Sie noch weitere Informationen von uns.

Sollten Sie doch nicht teilnehmen wollen, klicken Sie bitte hier: http://lapp.io/N177KlP8d

Dies ist eine automatisch generierte E-Mail.

Mit freundlichen Grüßen

Ihr Veranstaltungsteam
```

Abb. 60: Anmeldebestätigung (Text-Beispiel)

Folgendermaßen wird im E-Mail-Template auf die Datenquelle und deren Daten zugegriffen:

```
<?for v in datasources.veranstaltungen.app.records.values()?>
   <?printx format(v.v_datum, "%d.%m.%Y")?> - <?printx v.v_veranstaltung?> in <?
↪printx v.v_ort?>
<?end for?>
```

Die *for*-Schleife durchläuft alle vorhandenen Datensätze der Datenquelle. Mit `printx` wird dann das Datum, die Veranstaltung und der Ort ausgegeben.

Folgendes wird dann in der E-Mail ausgegeben:

```
23.05.2020 - Kommunikation/Gesprächsführung im Personalwesen in Bayreuth
12.03.2020 - Telefontraining in Bayreuth
18.01.2020 - Basiswissen Existenzgründung in Bamberg
25.01.2020 - Basiswissen Existenzgründung in Bayreuth
```

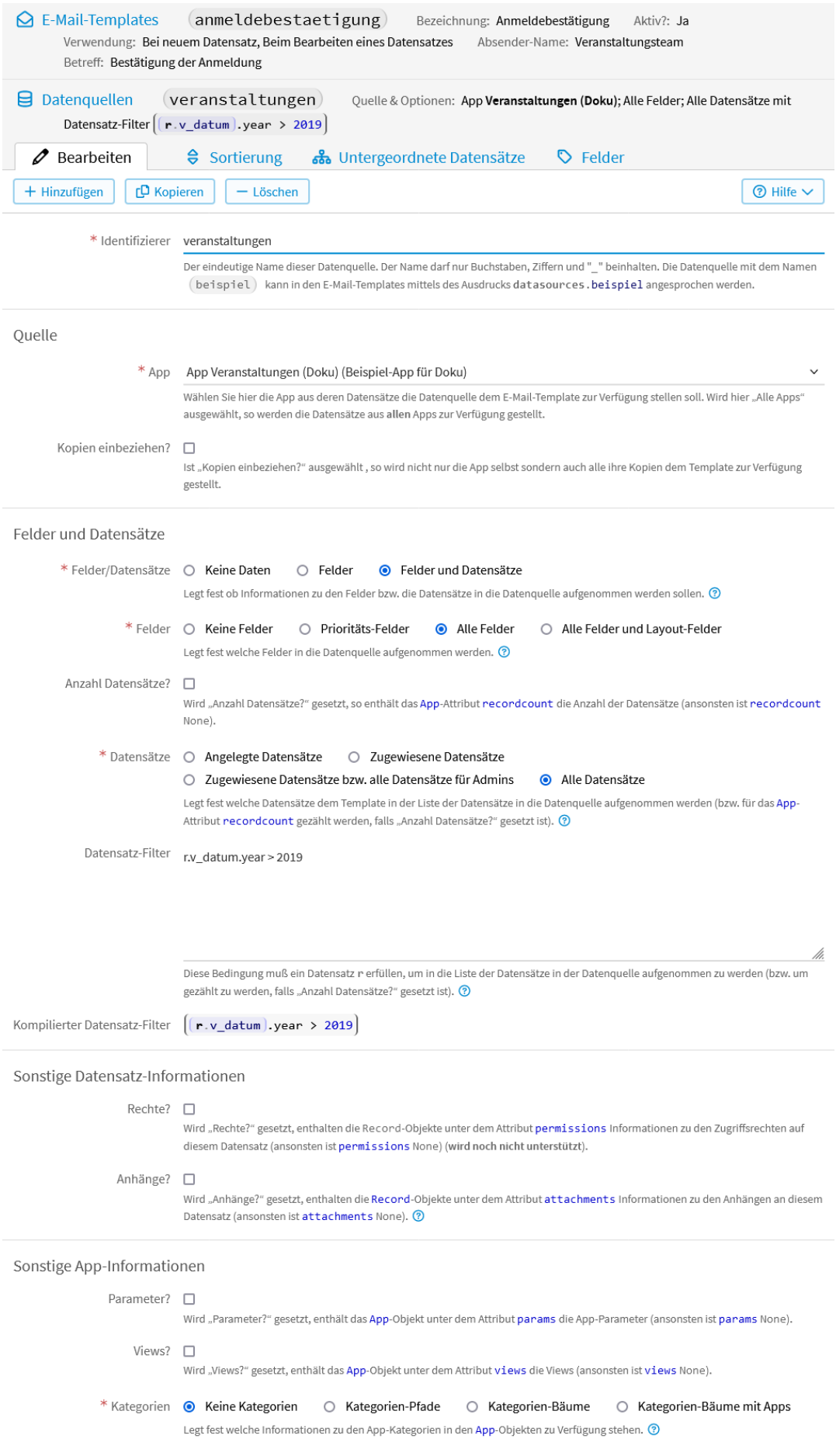


Abb. 61: Beispiel Datenquelle

2.7.3 Versendete E-Mails

Klicken Sie im Datenmanagemt unter *Daten* → *Liste* in der linken Navigation auf *E-Mail-Historie*, sehen Sie alle E-Mails, die bisher über die App versendet wurden.

2.8 Formular-Templates

Formular-Templates werden vor und nach dem Ausfüllen eines Eingabeformulars ausgeführt. Damit ist es möglich:

- die Vorbelegung der Felder zu beeinflussen;
- bei Applookup-Feldern eine eigene Auswahl festzulegen, d.h. aus welchen Datensätzen ausgewählt werden kann, wie und in welcher Reihenfolge diese Datensätze angezeigt werden;
- den Datensatz bevor er gespeichert wird nochmals zu verändern;
- nachdem der Datensatz gespeichert wurde noch zusätzliche Datensätze anzulegen, die auf diesen neuen Datensatz verweisen.

Wenn ein Formular-Template definiert ist, so wird dieses Formular-Template während der Phasen der Formular-Verarbeitung ein- oder mehrmals aufgerufen. In welcher Phase man sich befindet, erfährt man mittels *globals.mode*.

2.8.1 Ablaufplan bei „Neu“-Formularen

Bei einen „Neu“-Formular erfolgt die Verarbeitung in folgenden Phasen:

1. Neues Datensatz-Objekt anlegen;
2. Im Formbuilder konfigurierte Feld-Standard-Werte setzen (Felder des eingeloggten Users etc.);
3. Felder das Datensatzes mit den in der URL übergebenen Parametern überschreiben;
4. Formular-Template im Modus `form/new/init` ausführen.

Bei einem Fehler im „Neu“-Formular (z.B. wegen nicht ausgefüllter Pflichfelder) passiert folgendes:

5. Daten nach Schritt 4 (siehe oben) laden, wenn nicht vorhanden, Schritte 1. - 4. wiederholen)
6. Felder das Datensatzes mit den im Formular eingegebenen Werten überschreiben;
7. Formular-Template im Modus `form/new/fail` ausführen.

Wenn der Datensatz gespeichert wird passiert folgendes:

8. Daten nach Schritt 4 (siehe oben) laden, wenn nicht vorhanden, Schritte 1. - 4. wiederholen;
9. Felder das Datensatzes mit den im Formular eingegebenen Werten überschreiben;
10. Formular-Template im Modus `form/new/presave` ausführen;
11. Datensatz abspeichern;
12. Datenaktion *Nach Anlegen des Datensatzes ausführen (auch bei Import)* ausführen;
13. Formular-Template im Modus `form/new/postsave` ausführen.

2.8.2 Ablaufplan bei „Bearbeiten“-Formularen

Bei einem „Bearbeiten“-Formular sieht die Ausführungsreihenfolge ähnlich aus:

1. Datensatz laden;
2. Datenaktion *Vor dem Anzeigen des „Bearbeiten“-Formulars ausführen“* ausführen;
3. Felder des Datensatzes mit den in der URL übergebenen Parametern überschreiben;
4. Formulartemplate im Modus `form/edit/init` ausführen.

Bei einem Fehler im „Bearbeiten“-Formular (z.B. wegen nicht ausgefüllter Pflichfelder) passiert folgendes:

5. Daten nach Schritt 4 (siehe oben) laden, wenn nicht vorhanden, Schritte 1. - 4. wiederholen;
6. Felder das Datensatzes mit den im Formular eingegebenen Werten überschreiben;
7. Formulartemplate im Modus `form/edit/fail` ausführen.

Wenn der Datensatz gespeichert wird passiert folgendes:

8. Daten nach Schritt 4 (siehe oben) laden, wenn nicht vorhanden, Schritte 1. - 4. wiederholen);
9. Felder das Datensatzes mit den im Formular eingegebenen Werten überschreiben;
10. Formular-Template im Modus `form/edit/presave` ausführen;
11. Datensatz abspeichern;
12. Datenaktion *Nach Änderung des Datensatzes ausführen (auch bei Import)* ausführen;
13. Formular-Template im Modus `form/edit/postsave` ausführen.

2.8.3 Erstellung eines Formular-Templates

Um ein Formular-Template für Ihre LivingApp zu erstellen, wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Formular-Templates* und wählen dann die Formular-Ansicht für die Sie ein Formular-Template erstellen wollen.

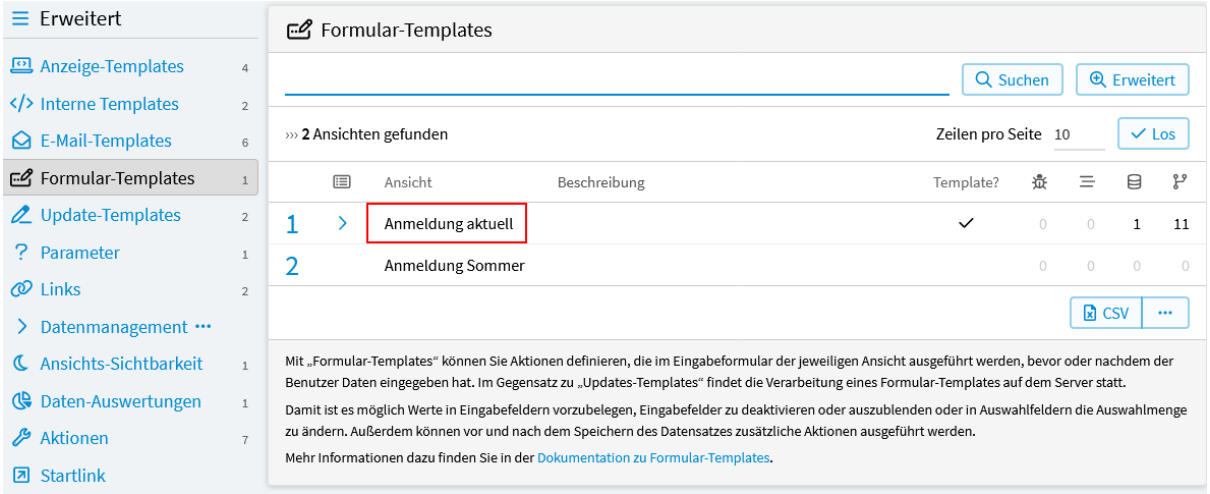


Abb. 62: Formular-Template hinzufügen

Um auf die Daten Ihrer App zugreifen zu können, benötigten Sie die Templatesprache UL4. Die UL4-Dokumentation finden Sie unter https://python.livinglogic.de/UL4.html. Die Ausgabe der Daten erfolgt über die *LivingAPI*. Im zugehörigen Kapitel sind die Objekte ausführlich beschrieben.

Die einzelnen Felder der App werden über den jeweiligen Identifizierer angesprochen. Darüber können Sie unter *Felder und deren Identifizierer* mehr nachlesen.

Möchten Sie in Ihrem Formular-Template z. B. auf App-Parameter zugreifen, die Daten mehrerer Apps ausgeben oder die zur Verfügung stehenden Daten im Vorhinein einschränken, dann können Sie dafür Datenquellen anlegen. Mit `globals.datasources` können Sie auf die Datenquelle zugreifen. Mehr darüber können Sie unter *Anlegen von Datenquellen* nachlesen. Auf die Datenquellen, die Sie im Formular-Template anlegen, können Sie auch von den Update-Templates aus zugreifen.

2.8.4 Bearbeiten eines Formular-Templates

Gehen Sie vor wie oben beschrieben und wählen die Ansicht aus, deren Formulartemplate sie bearbeiten wollen.

2.8.5 Anwendungsbeispiele

Im Folgenden wird ein Formular-Template in der App „Anmeldung“ aus Kapitel 1 angelegt.

Im Anwendungsbeispiel wird gezeigt, wie Sie Felder mit einer Auswahl vorbelegen können, wie Sie die Auswahl im Applookup einschränken und sortieren und Feldbezeichnungen ändern können.

Beim Feld *Wird Übernachtung benötigt?* soll *Nein* vorausgewählt sein. Die Anzahl der Personen, die mit einer Anmeldung eingegeben werden können, wird auf 10 begrenzt und es sollen nur ganze Zahlen eingegeben werden können.

Als nächstes wird die Auswahl der Veranstaltungen im Applookup *Veranstaltung* eingeschränkt und zwar sollen nur die Veranstaltungen angezeigt werden, die in der Zukunft liegen und sie sollen in der Auswahl aufsteigend sortiert, also immer die aktuellste oben angezeigt werden. Außerdem soll die Feldbezeichnung in der Auswahl der Dateneingabe nicht zu sehen sein, sondern nur der Inhalt. Also z. B. `Word für Fortgeschrittene` anstatt `Veranstaltung: Word für Fortgeschrittene`.

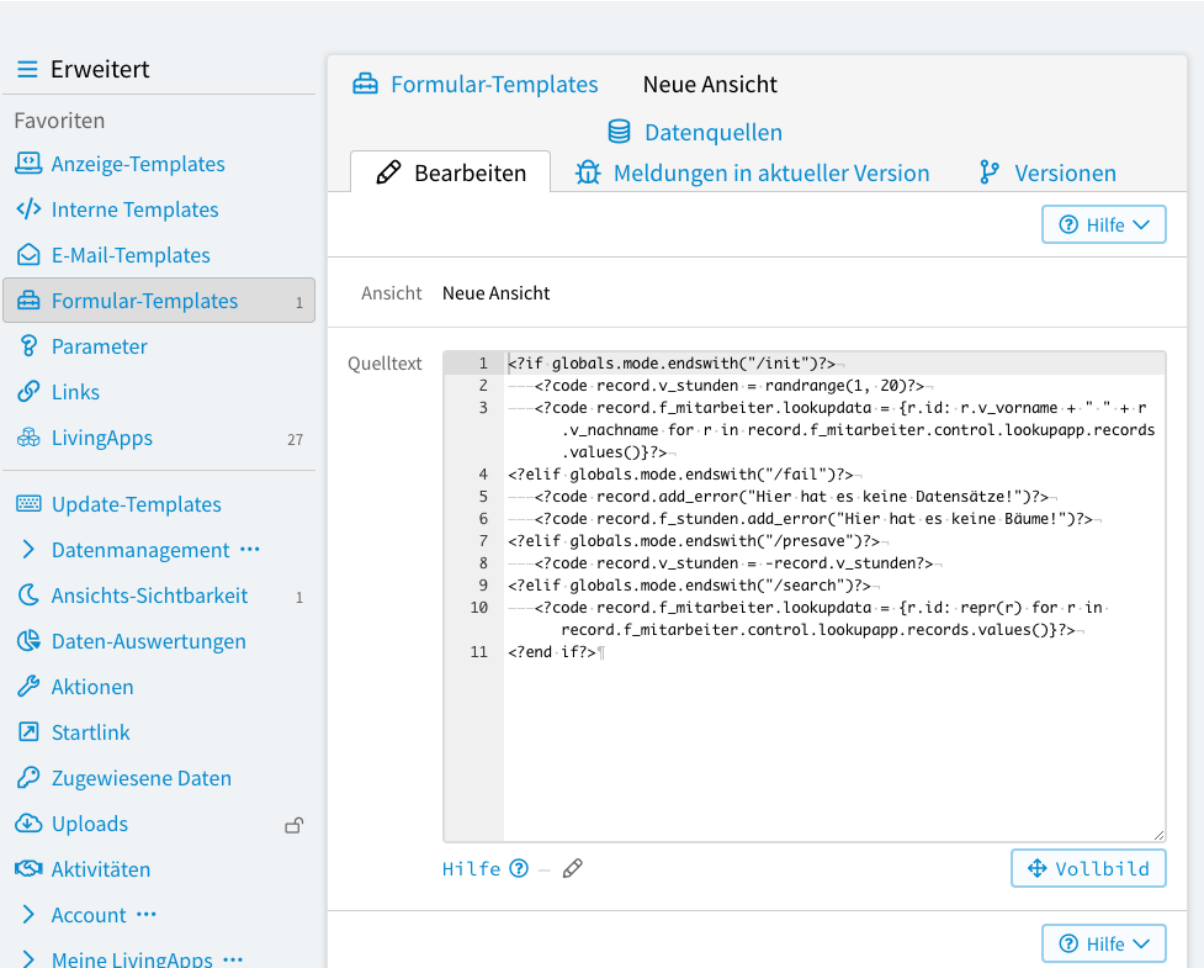


Abb. 63: Formular-Template bearbeiten

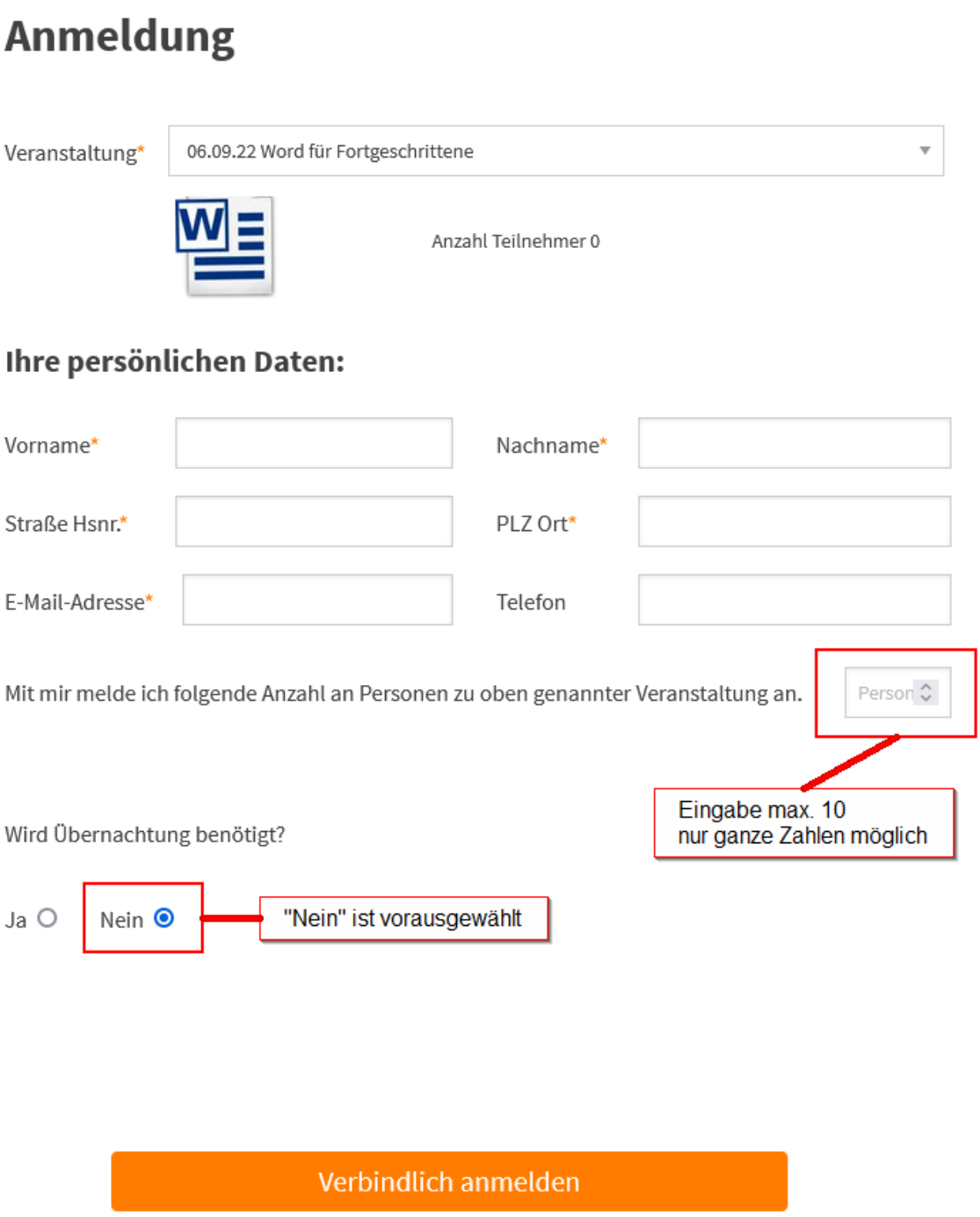


Abb. 64: Formular-Template - Felder vorbelegen

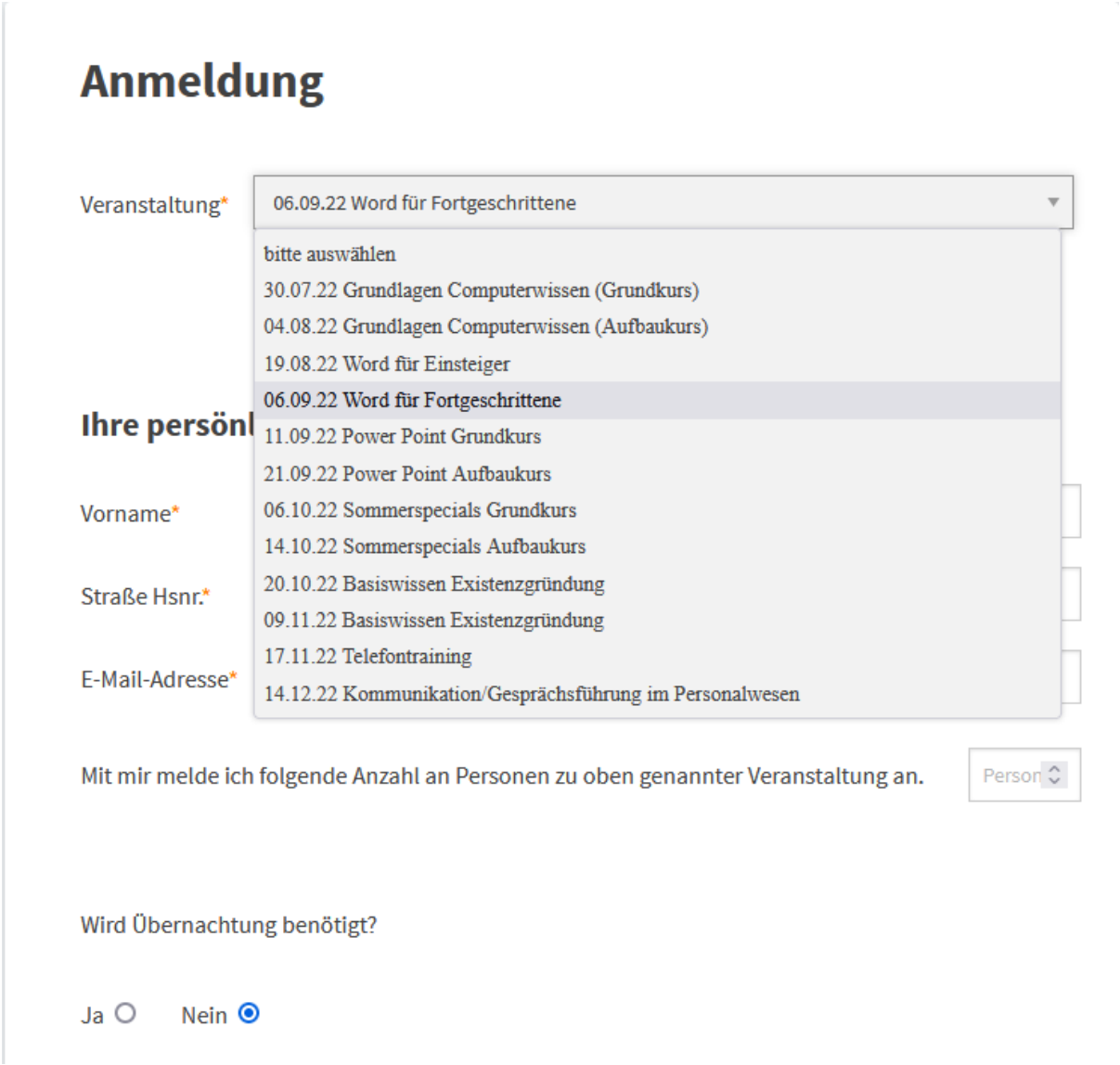


Abb. 65: Formular-Template - Auswahl einschränken, Sortierung

Dafür muss eine Datenquelle angelegt werden, weil auf die App „Veranstaltungen“ zugegriffen werden soll und die Einschränkung und Sortierung auch über die Datenquelle eingestellt werden. Wie Sie eine Datenquelle anlegen, können Sie unter *Anlegen von Datenquellen* nachlesen.

Folgende Konfigurationen wurden im Anwendungsbeispiel in der Datenquelle vorgenommen:

Datenquelle

Sortierung

Untergeordnete Datensätze (wurde angelegt für Update-Template)

Für das Anwendungsbeispiel wurde die *Ansicht aktuell* gewählt und folgender Code eingegeben:

```
<?code record.f_veranstaltung2.lookupdata = {
   r.id: format(r.v_datum, "%d.%m.%Y") + " " + r.v_veranstaltung
   for r in record.app.c_veranstaltung2.lookupapp.records.values()
}?>
<?note Ändert die Anzeige der Veranstaltungen von Datum:
   01.01.2022, Veranstaltung: Word für Fortgeschrittene
   in 01.01.2022 Word für Fortgeschrittene.?>
<?code record.f_anzahl_personen2.control.precision = 0?>
<?note Für Zahlfelder (Typ = number) können die Anzahl der
   Nachkommastellen angegeben werden.?>
<?code record.f_anzahl_personen2.control.minimum = 0?>
<?note Außerdem kann ein Minimalwert...?>
<?code record.f_anzahl_personen2.control.maximum = 10?>
<?note ... und ein Maximalwert angegeben werden. Bei Integer-Zahlen
   (ganze Zahlen) kann man das im Formbuilder einstellen.?>

<?code record.v_wird_uebernachtung_benoetigt = "nein"?>
<?note Vorbelegung des Lookup-Feldes "Wird Übernachtung benötigt" mit Nein?>
```

Kurze Erklärungen zum Code finden Sie dort als `<?note ...?>`.

2.9 Update-Templates

Die *Update-Templates* ermöglichen es, im Formular einer LivingApp die Werte von bestimmten Feldern aus anderen Feldern zu berechnen, sowie Felder zu aktivieren, deaktivieren, oder auszublenden. Wenn z. B. einzelne Eingabefelder unter bestimmten Voraussetzungen nicht sichtbar sein sollen, können Sie das Update-Template nutzen, um diese Eingabefelder auszublenden. Sie können für jede Formular-Ansicht ein Update-Template konfigurieren.

Das Update-Template wird ausgeführt bei der Eingabe oder der Bearbeitung eines Feldes, also wenn sich der Inhalt des Feldes ändert.

Möchten Sie in Ihrem Update-Template z. B. auf App-Parameter zugreifen, die Daten mehrerer Apps ausgeben oder die zur Verfügung stehenden Daten im Vorhinein einschränken, dann können Sie auf die Datenquellen zugreifen, die unter Formular-Templates angelegt wurden. Mit `globals.datasources` können Sie auf die Datenquelle zugreifen. Mehr darüber können Sie unter *Anlegen von Datenquellen* nachlesen.

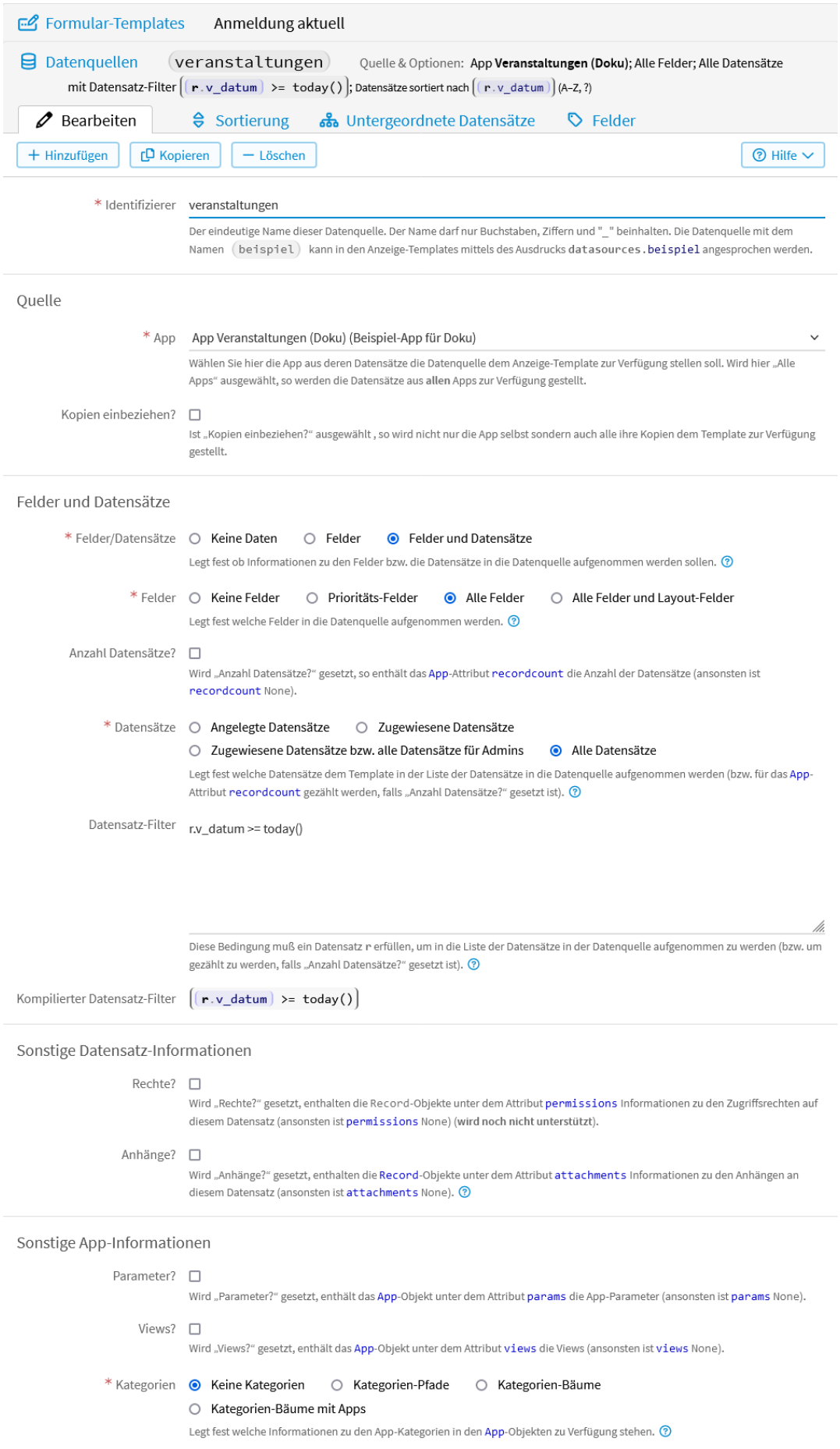


Abb. 66: Formular-Template - Datenquelle

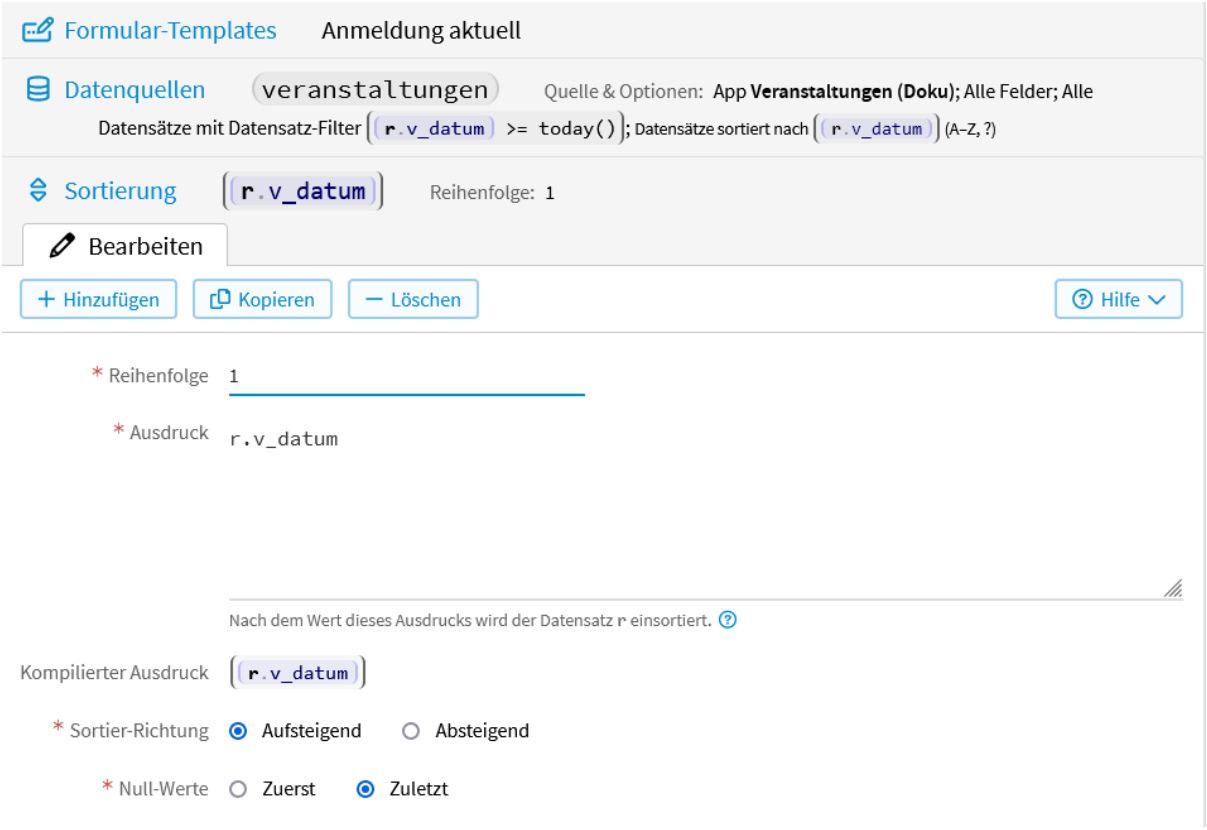


Abb. 67: Formular-Template - Datenquelle Sortierung

2.9.1 Erstellung eines Update-Templates

Wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken dann in der linken Menüleiste auf *Update-Template*. Hier sehen Sie nun die verschiedenen Formular-Ansichten Ihrer LivingApp.

Um für eine der Ansichten ein *Update-Template zu erstellen*, klicken Sie auf diese und füllen das Feld *Template*.

Geo-Koordinaten?
: Hier können Sie angeben, ob das Update-Template Zugriff auf den Standort des Endgerätes erhalten soll.

Um auf die Daten Ihrer App zugreifen zu können, benötigten Sie die Templatesprache UL4. Die UL4-Dokumentation finden Sie unter https://python.livinglogic.de/UL4.html. Die Ausgabe der Daten erfolgt über die *LivingAPI*. Im zugehörigen Kapitel sind die Objekte ausführlich beschrieben.

Die einzelnen Felder der App werden über den jeweiligen Identifizierer angesprochen. Darüber können Sie unter *Felder und deren Identifizierer* mehr nachlesen.

Dem Update-Template werden drei Parameter übergeben:

`record`
: `record` enthält alle aktuellen Daten des in Bearbeitung befindlichen Datensatzes. Details dazu finden Sie in der Dokumentation der LivingAPI unter *Record*.

  Außerdem gibt es in Update-Templates unter bei den *Field*-Objekten folgende Attribute:

  `writable`
  : Mit diesem Attribut kann das Feld beschreibar bzw. unbeschreibar gemacht werden. D. h. `record.f_name_writeable = True` entfernt das `readonly`-Attribut vom Eingabe-Feld mit dem Identifizierer `name`, `record.f_name.writable = False` fügt es hinzu. Die Werte aus Feldern, die auf diese Weise als „readonly“ konfiguiert sind, werden ins Datenmanagement übertragen.

  `visible`
  : Mit diesem Attribut kann das Feld ausgeblendet werden. `record.f_name.visible = False` ent-

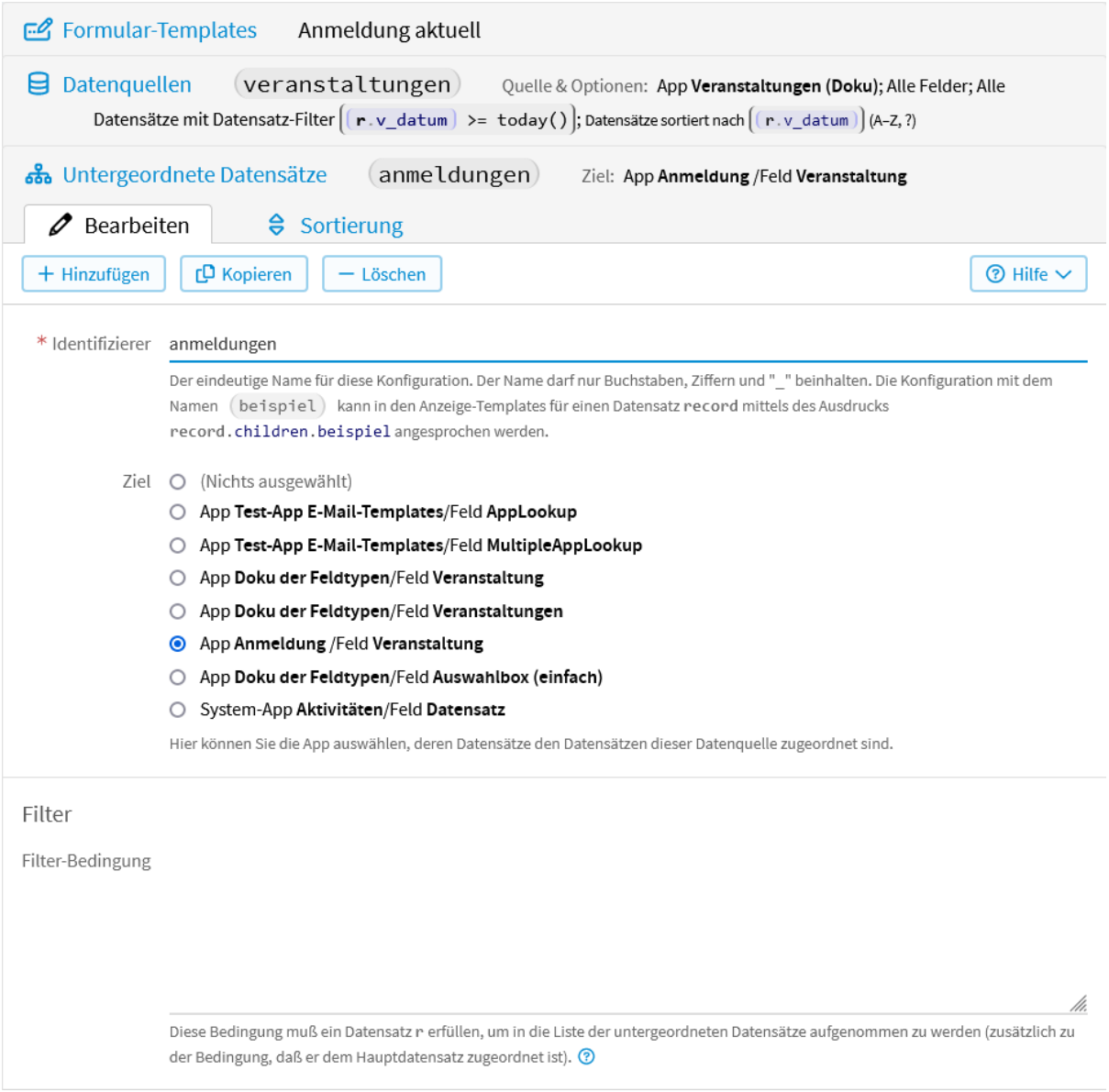


Abb. 68: Formular-Template - Datenquelle untergeordnete Datensätze

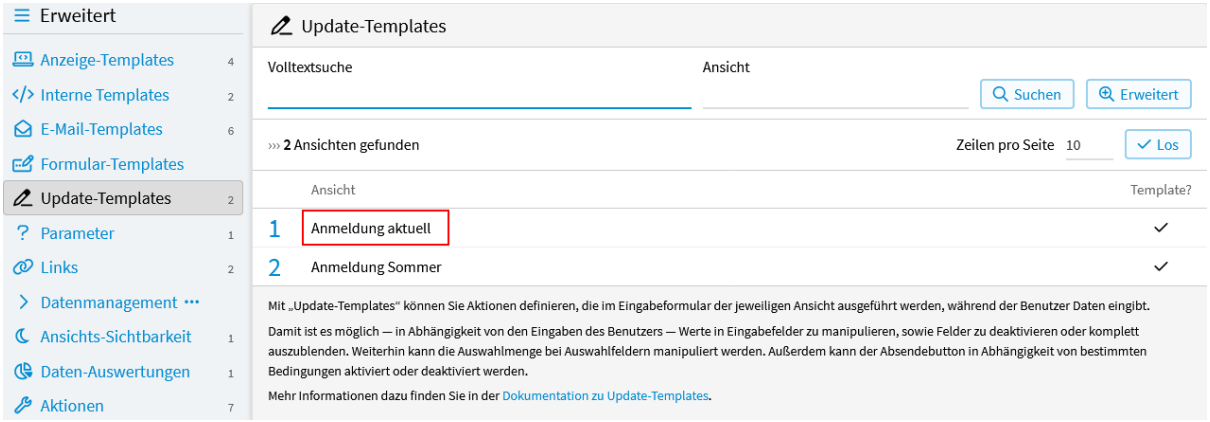


Abb. 69: Update-Template hinzufügen

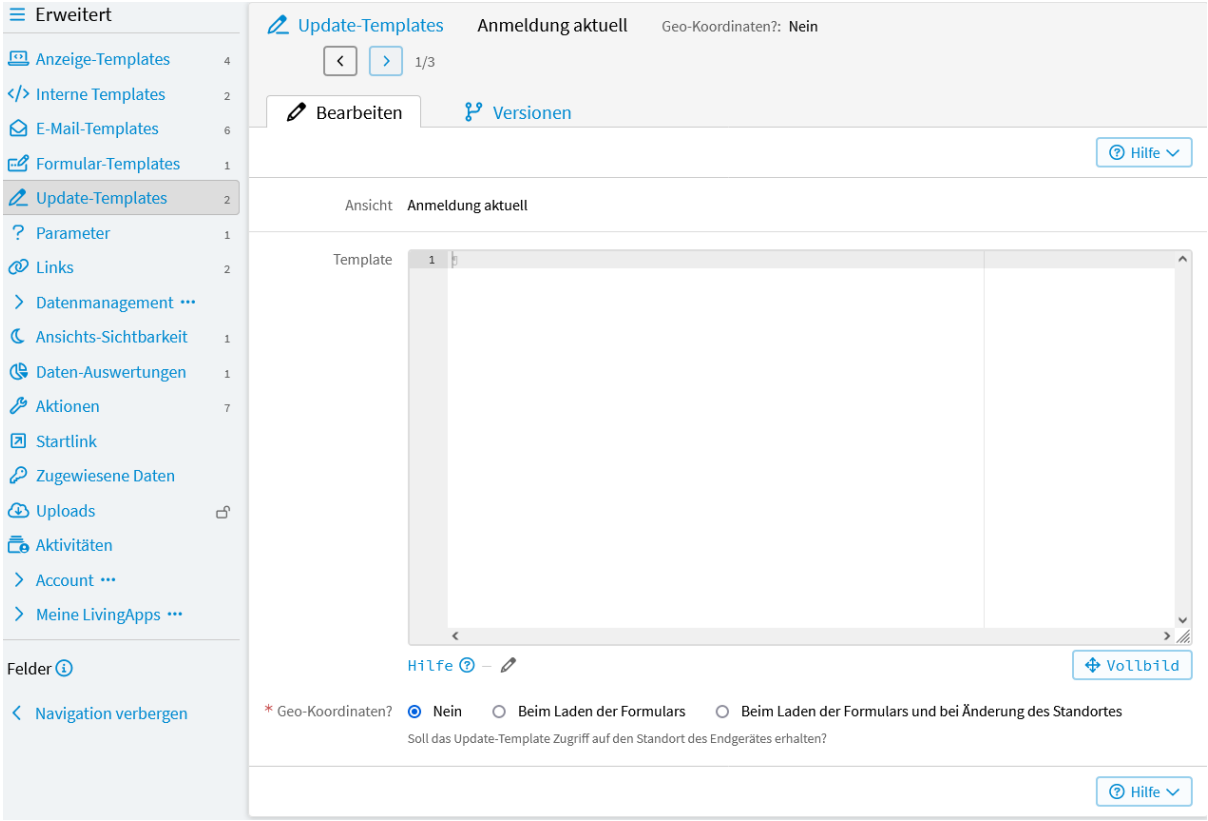


Abb. 70: Update-Template erstellen

fernt das Feld mit dem Identifizierer `name` vom Formular. Mit `record.f_name.visible = True` kann man es wieder sichtbar machen. Somit können im Formular-Template und im Update-Template Bedingungen programmiert werden, mit denen man Felder ein- und ausschalten kann. Auch unsichtbare Werte werden ins Datenmanagement übertragen.

`enabled`
: Damit kann das Feld aktiviert und deaktiviert werden. D.h. `record.f_name.enabled = True` entfernt das `disabled`-Attribut vom Eingabe-Feld mit dem Identifizierer `name`, `record.f_name.enabled = False` fügt es hinzu, sodass keine Eingabe mehr möglich ist. Hierbei gilt es zu beachten, dass Werte aus Feldern, die disabled sind, nicht ins Datenmanagement von LivingApps übertragen werden. Sie sind somit nur für den Nutzer des Formulars zu sehen.

`identifier`
: Das Template wird einmal nach dem Laden der Seite aufgerufen. Dies ermöglicht es beispielsweise, im Formular dynamische Default-Werte zu setzen oder ähnliches. In diesem Fall hat `identifier` den Wert `None`.

  Des Weiteren wird das Template jedes Mal aufgerufen, wenn sich eines der Felder ändert (das heißt bei jedem `change`- oder `keyup`-Event). In diesem Fall wird in `identifier` der Identifizierer des Feldes übergeben, das sich gerade geändert hat.

`globals`
: Globals enthält wichtige Grundinformationen über z. B. den angemeldeten Benutzer, die Datenquelle, die im Formular-Template-init geöffnet wurden, die App usw. Details dazu finden Sie in der Dokumentation der LivingAPI unter *Globals*.

  Eine Besonderheit in Bezug auf Update-Templates ist die Verwendung der Log-Funktionen wie z.B. `globals.log_info`. Diese Debuginformationen landen in der Javascriptkonsole.

2.9.2 Anwendungsbeispiel

Im Folgenden wird das Update-Template der App „Anmeldung“ aus Kapitel 1 konfiguriert.

Abhängig von der Auswahl bei *Wird Übernachtung benötigt?* sollen die Felder *Einzelzimmer*, *Doppelzimmer* sowie die zwei Felder *Euro / Nacht* für die Übernachtungskosten *ein- bzw. ausgeblendet* werden.

Ist keine Option oder die Option *Nein* ausgewählt, sollen die restlichen Felder ausgeblendet sein. Ist die Option *Ja* ausgewählt, sollen die Felder eingeblendet werden. Der Nutzer soll dann die Anzahl der Einzel- oder Doppelzimmer eintragen und in den Kosten-Feldern sollen mit dem Update- Template die *Kosten berechnet* werden.

Außerdem soll, abhängig von der Auswahl der Veranstaltung, das entsprechende *Icon* dazu eingeblendet werden und die *Anzahl* der bereits angemeldeten Teilnehmer zu dieser Veranstaltung angezeigt werden.

Dafür muss im Formular der App „Anmeldung“ jeweils ein „Platzhalter“ für das Bild und für die Anzahl der Teilnehmer in Form eines „Formatierten Textfeldes“ hinzugefügt werden. Wählen Sie dafür im Menü *Konfiguration* → *Eingabe* und ziehen dann das *Formularelement* *Formatierter Text* zweimal ins Formular.

Als Text wurde im Anwendungsbeispiel „ohne Icon“ für das Bild und „Anzahl Teilnehmer 0“ eingetragen. Um diese „Platzhalter“ im Update-Template anzusprechen, brauchen Sie deren Identifizierer.

Dafür gehen Sie folgendermaßen vor:

Wählen Sie in der Dateneingabe eine Veranstaltung aus, für die ein Icon hochgeladen wurde und klicken dann mit der rechten Maustaste auf das Icon. Im nun geöffneten Fenster wählen Sie *„Untersuchen“*.

Es öffnet sich der Inspektor im Browser. Suchen Sie dort nach dem `div` in dem die *formatierten Textfelder* stehen.

Im Anwendungsbeispiel lautet der Identifizierer für den Platzhalter des Icons `formatierter_text` und für den Platzhalter der Teilnehmeranzahl `formatierter_text2`.

Zum Erstellen des Update-Templates wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken dann in der linken Menüleiste auf *Update-Template*. Wählen Sie die entsprechende Ansicht aus, für die Sie das Update-Template konfigurieren möchten.

Im Anwendungsbeispiel wurde die *Ansicht aktuell* ausgewählt und dafür folgender Code eingegeben:

```
<?code fields.euro_nacht.writable = False ?>
<?code fields.euro_nacht2.writable = False ?>
<?note Damit kann man in dem Feld "euro_nacht" und "euro_nacht2"
   nichts mehr ändern.?>

<?if identifier is None or identifier == "wird_uebernachtung_benoetigt" ?>
   <?code fields.einzelzimmer.visible = fields.wird_uebernachtung_benoetigt.value.key␣
↪== "ja" ?>
   <?code fields.einzelzimmer.value = 0?>
   <?code fields.euro_nacht.visible = fields.wird_uebernachtung_benoetigt.value.key␣
↪== "ja" ?>
   <?code fields.euro_nacht.value = 0?>
   <?code fields.mehrbettzimmer.visible = fields.wird_uebernachtung_benoetigt.value.
↪key == "ja" ?>
   <?code fields.mehrbettzimmer.value = 0?>
   <?code fields.euro_nacht2.visible = fields.wird_uebernachtung_benoetigt.value.key␣
↪== "ja" ?>
   <?code fields.euro_nacht2.value = 0?>
<?end if?>

<?if identifier is None or fields.einzelzimmer.visible ?>
   <?code fields.euro_nacht.value = fields.einzelzimmer.value * 38 ?>
   <?code fields.euro_nacht2.value = fields.mehrbettzimmer.value * 60 ?>
<?end if?>

<?if identifier is None or identifier == "veranstaltung2"?>
```

(Fortsetzung auf der nächsten Seite)

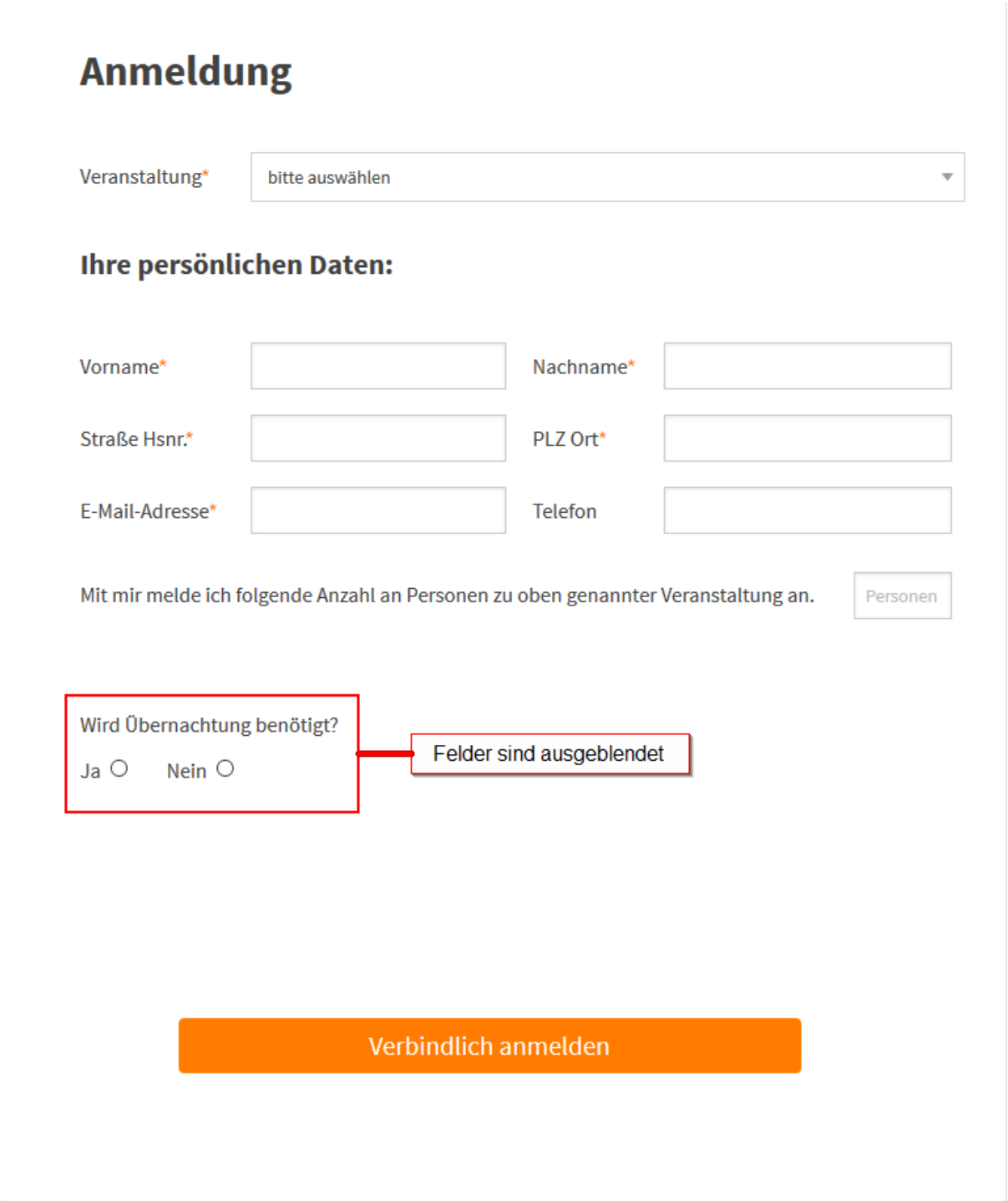


Abb. 71: Update-Template - Felder ein- und ausblenden

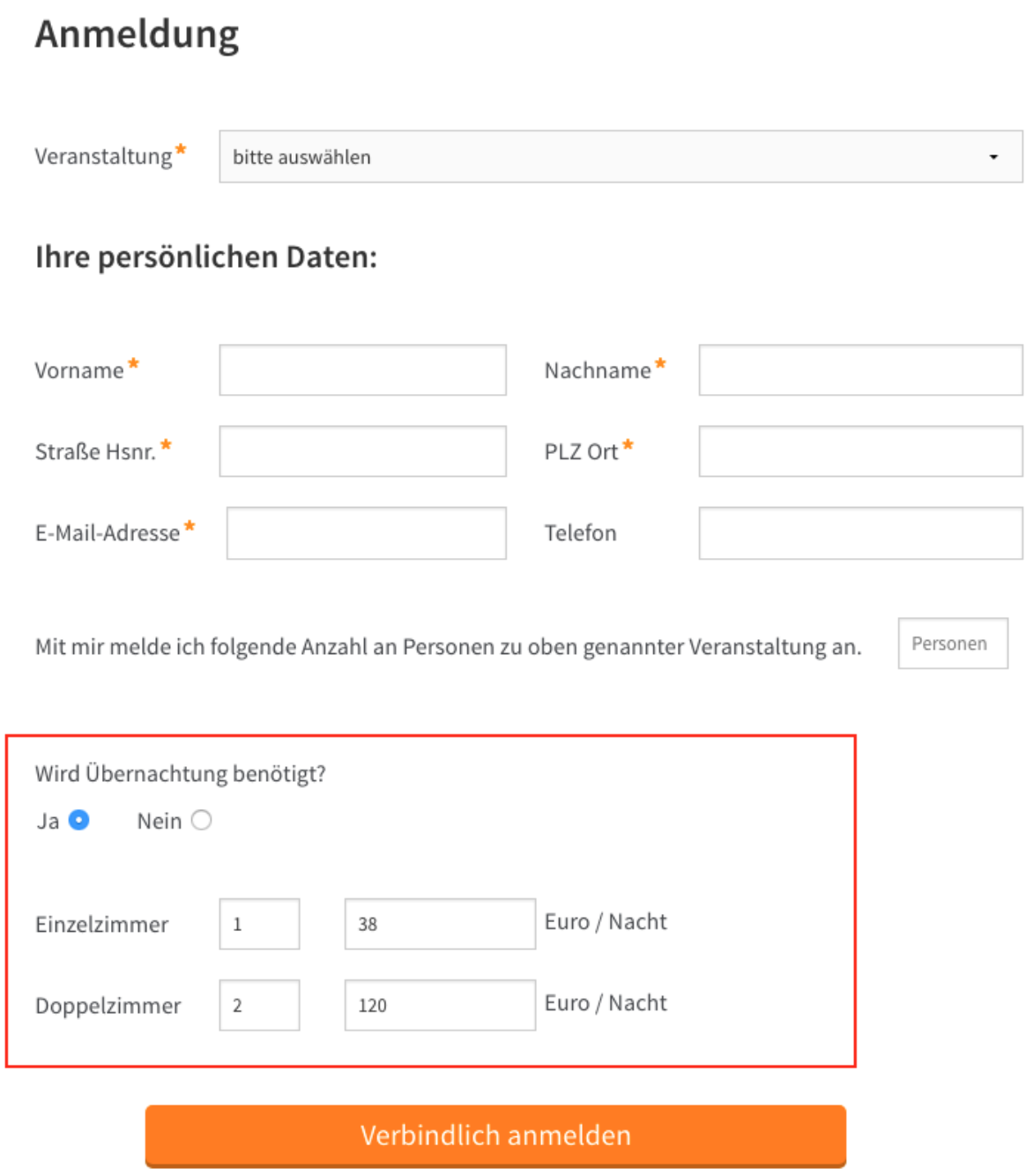


Abb. 72: Update-Template - Kosten berechnen

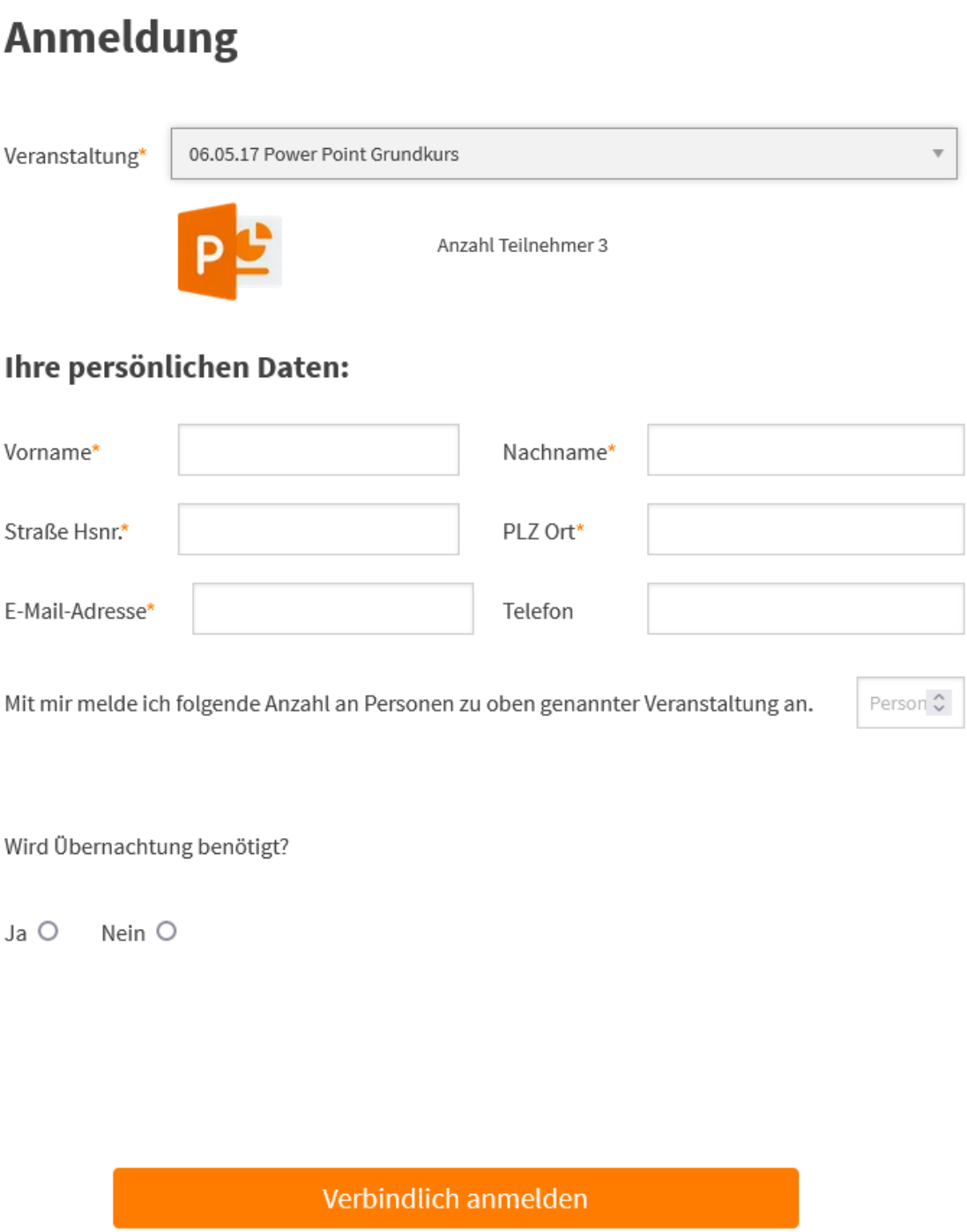


Abb. 73: Update-Template - Icon einblenden und Teilnehmer anzeigen

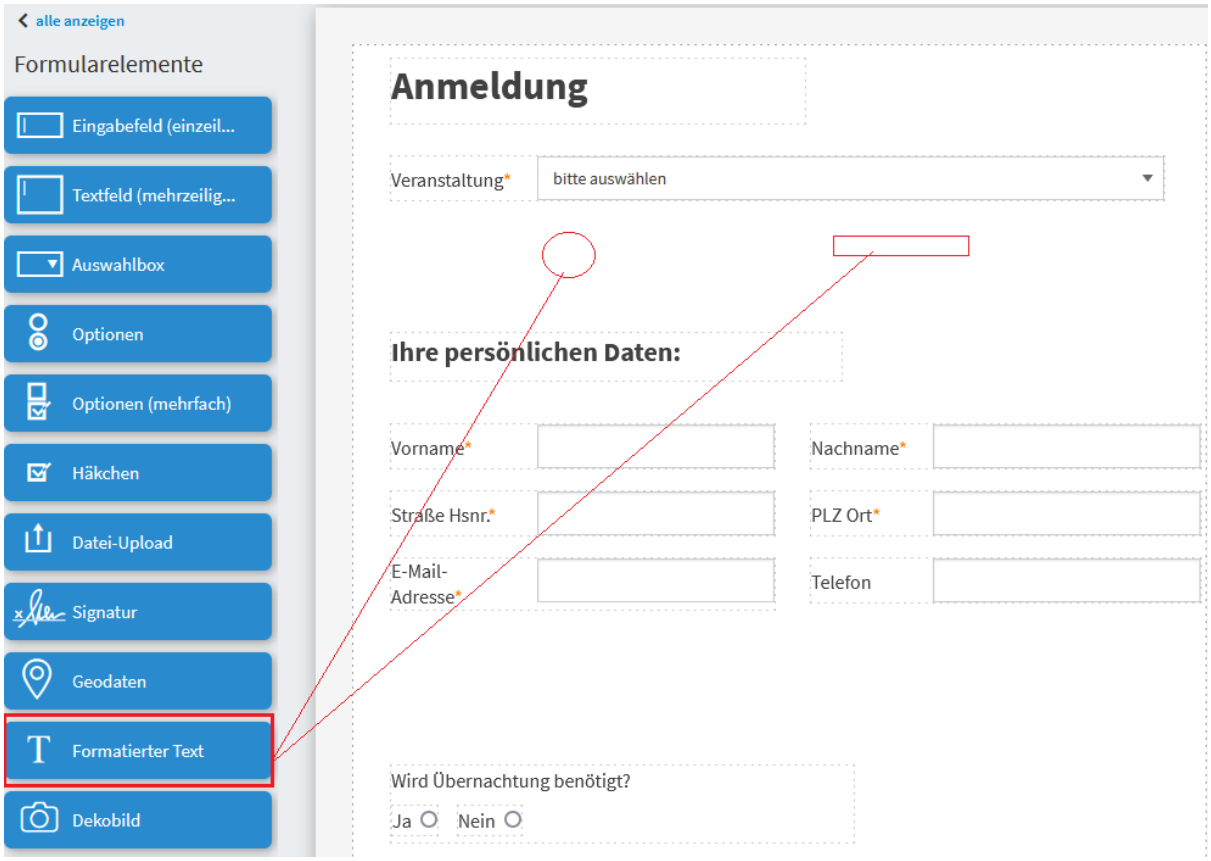


Abb. 74: Update-Template - Platzhalter im Formular erstellen

(Fortsetzung der vorherigen Seite)

```
<?note zunächst das Icon der Veranstaltung anzeigen?>
<?if record.v_veranstaltung2?>
   <?note Berechnungen zur Veranstaltung machen nur Sinn, wenn eine
      Veranstaltung ausgewählt wurde.?>
   <?if record.v_veranstaltung2.v_icon?>
      <?code texticon = '<p><img src="' +
         globals.scaled_url(record.v_veranstaltung2.v_icon,
         30, 30)+'" /></p>'?>
      <?note Wir bauen den HTML-Textschnipsel für die Anzeige des
         Icons. "scaled_url" ist hierbei eine Hilfsfunktion unter
         globals mit der man die URL eines Bildes berechnen
         lassen kann, inklusive der Abmessungen des Bildes
         (hier 30px, 30px).?>
      <?code record.app.lc_formatierter_text.value = texticon?>
      <?note Hier wird der HTML-Schnipsel dem Layoutfeld
         "formatierter_text" zugeordnet. Wie man an den Identifizierer
         des Formatierten Textfeldes rankommt, wird weiter oben in der
         Dokumentation beschrieben.?>
   <?else?>
      <?code record.app.lc_formatierter_text.value = ""?>
   <?end if?>
   <?code record.app.lc_formatierter_text2.value = "Anzahl Teilnehmer " +
      str(sum(1+(a.v_anzahl_personen2 or 0) for a in
      record.v_veranstaltung2.c_anmeldungen.values()))?>
   <?note Das Layoutfeld "formatierter_text2" wird mit der Anzahl der
      Teilnehmer als Zusatzinformation bestückt. Dazu haben wir in
```

(Fortsetzung auf der nächsten Seite)

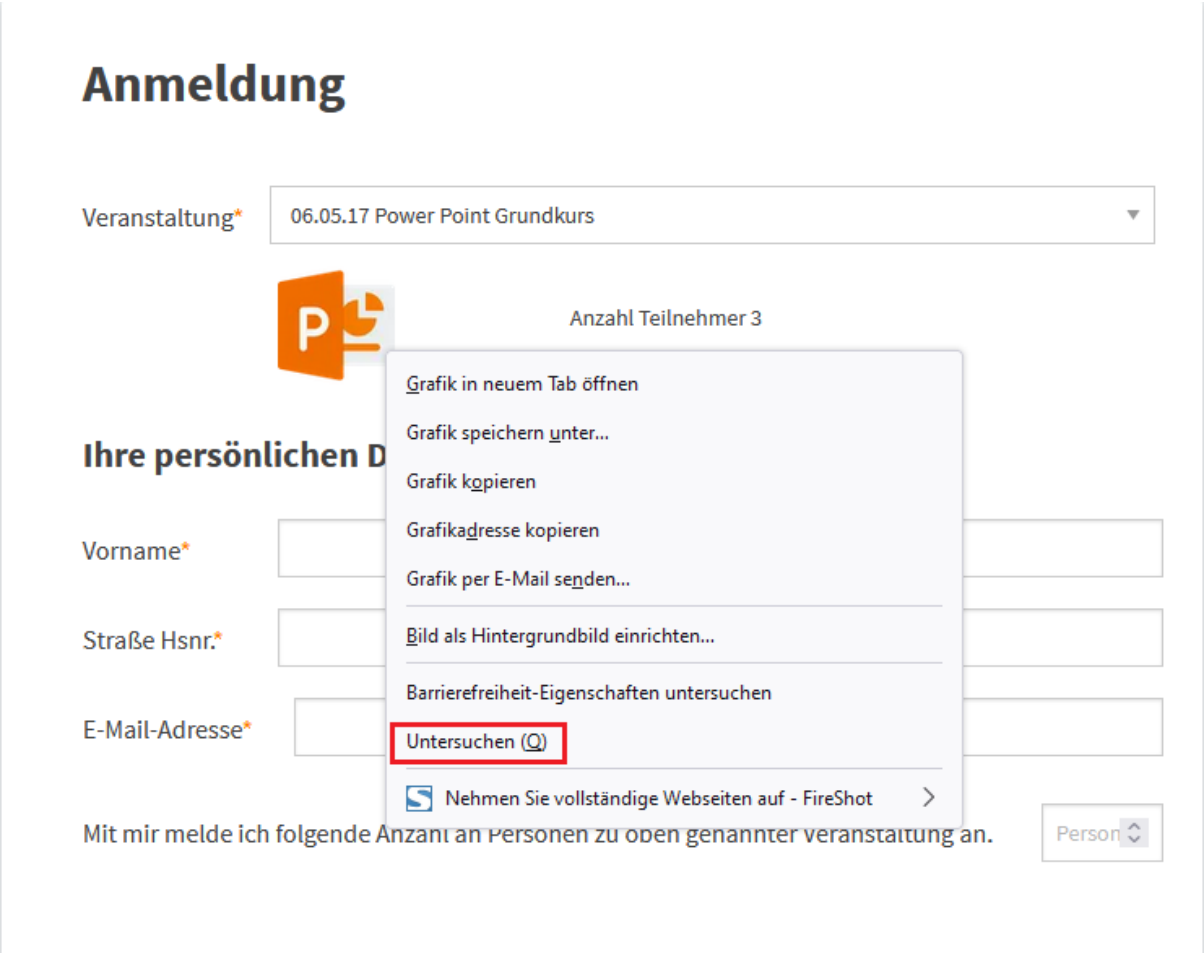


Abb. 75: Update-Template - Identifizierer für Platzhalter finden (Teil 1)

Abb. 76: Update-Template - Identifizierer für Platzhalter finden (Teil 2)

(Fortsetzung der vorherigen Seite)

```
        der Datenquelle "veranstaltung" noch die "Untergeordneten Daten"
        "anmeldungen" konfiguriert. Damit stehen beim Veranstaltungsdatensatz
        unter "c_anmeldungen" die Anmeldedatensätze zu dieser Veranstaltung
        zur Verfügung. Über "sum" können wir also die Anzahl der Teilnehmer
        ermitteln.?>
   <?end if?>
<?end if?>
```

Kurze Erklärungen zum Code finden Sie dort als `<?note ...?>`.

2.10 Parameter

App-Parameter sind Konfigurationswerte, auf die in den Anzeige-Templates zugegriffen werden kann. Durch deren Nutzung können Werte und Dateien ohne Anpassung des Template-Codes geändert werden.

2.10.1 Erstellung von App-Parametern

Wählen Sie im Menu *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Parameter* und dann auf *Hinzufügen*, um einen App-Parameter anzulegen.

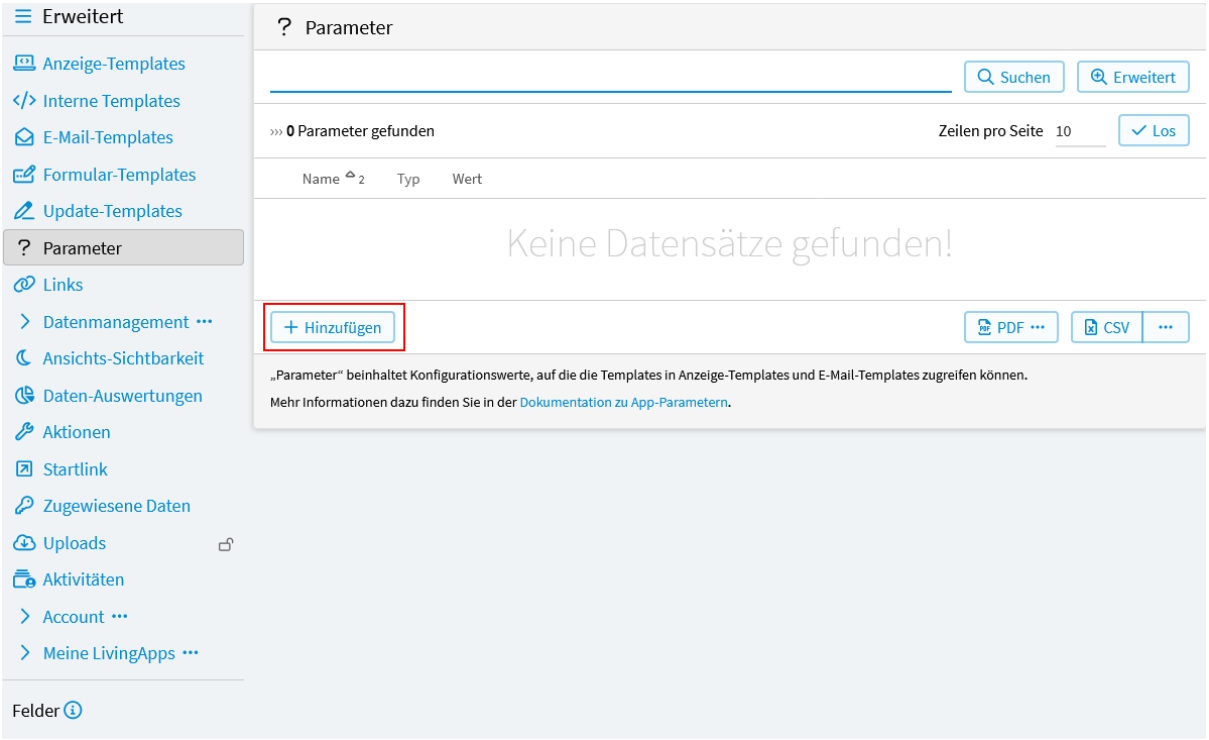


Abb. 77: App-Parameter hinzufügen

Im nun geöffneten Fenster sehen sie die *Eingabeansicht* Ihrer App-Parameter.

Name
: Hier muss ein eindeutiger Name für den App-Parameter festgelegt werden. Der Name darf nur Buchstaben, Ziffern und „_“ beinhalten. Über diesen Namen können sie in Ihren Templates auf den App-Parameter zugreifen.

Typ
: Wählen Sie hier den Typ Ihres Parameters.

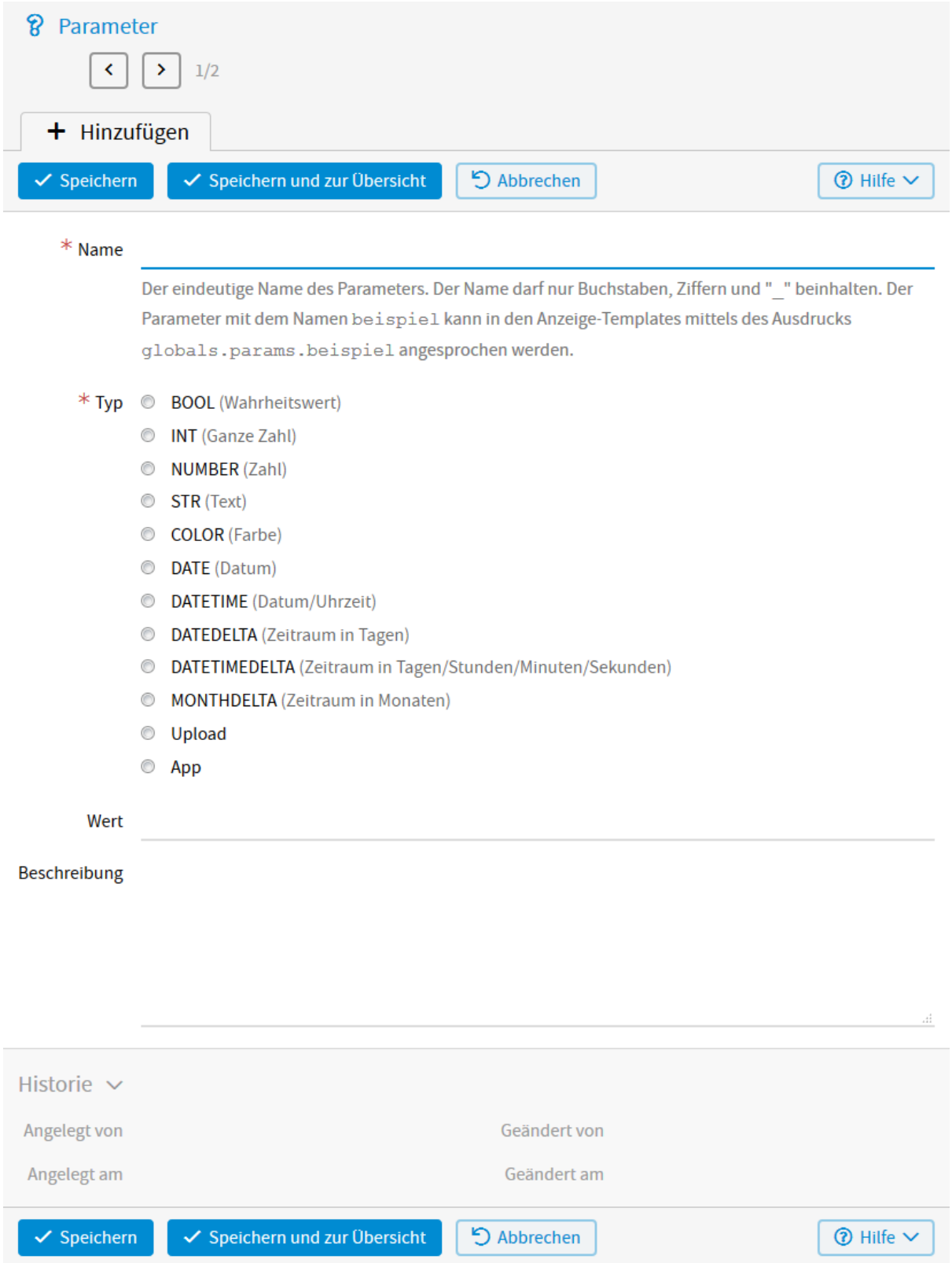


Abb. 78: App-Parameter Eingabeansicht

BOOL Wahrheitswert (Ja/Nein)

INT Ganze Zahl (z.B. `2`)

NUMBER Zahl (z.B. `2.394`)

STR Text (z.B. `gurk`)

COLOR Farbe (z.B. `#ff0000`)

DATE Datum (z.B. `29.02.2000`)

DATETIME Datum/Uhrzeit (z.B. `29.02.2000 16:05:54`)

DATEDELTA Zeitraum im Tagen (z.B. `10`)

DATETIMEDELTA Zeitraum in Tagen/Stunden/Minuten/Sekunden (z.B. 5 `days, 12:33:15`)

MONTHDELTA Zeitraum in Monaten (z.B. `6`)

Upload Dateien (z.B. ein Bild oder eine PDF-Datei)

App die entsprechende App kann hier ausgewählt werden.

Wert
Je nach Typ des Parameters wird hier der Wert festgelegt. Beispiele siehe oben in der Klammer.

Beschreibung
Die Beschreibung der Parameter ist nur intern sichtbar.

2.10.2 Zugriff in E-Mail- und Anzeige-Templates

Um auf die App-Parameter zugreifen zu können, muss im Template eine *Datenquelle* konfiguriert und in dieser bei *Parameter?* das Häkchen gesetzt werden.

In den E-Mail- und Anzeige-Templates kann auf die App-Parameter mittels des Ausdrucks `app.params.beispiel` oder `app.p_beispiel` (Shortcut-Attribut) zugegriffen werden. Wobei `beispiel` jeweils der Name des Parameters ist.

Auf den Wert kann direkt mit dem Shortcut-Attribut `app.pv_beispiel` zugegriffen werden.

Der Wert des App-Parameters kann mit dem `print`-Statement:

```
<?printx app.pv_beispiel?>
```

ausgegeben werden.

Ist der App-Parameter vom Typ *DATE* kann das Datum in deutscher Schreibweise ausgegeben werden mit:

```
<?printx format(app.pv_beispiel, "%d.%m.%Y", "de")?>
```

Bei App-Parametern vom Typ *Upload* erhält man die URL, unter welcher die Datei zu finden ist mit:

```
<?printx app.pv_beispiel.url?>
```

Ist der App-Parameter vom Typ *App*, kann die App-ID ausgegeben werden mit:

```
<?printx app.pv_beispiel.id?>
```

Mehr Informationen zum `App`- und `AppParameter`-Objekt finden Sie unter *App* und *AppParameter*.

2.10.3 Anwendungsbeispiel

Um die Anwendung der App-Parameter zu demonstrieren, wird wieder auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen. Es soll ein App-Parameter angelegt werden, der als Überschrift des Anzeigetemplates eingesetzt werden soll.

Wählen Sie im Menü *Konfiguration → Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Paramter* und anschließend auf *Hinzufügen*.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Der *Name* soll `ueberschrift` lauten. Damit können Sie im E-Mail- oder Anzeigetemplate auf den App-Parameter zugreifen. Als *Typ* wird *STR (Text)* ausgewählt. Bei *Wert* wird die Bezeichnung der Überschrift - `Teilnehmerliste` eingegeben.

Wie Sie den App-Parameter in Ihren Templates ausgeben, können Sie im Abschnitt *Zugriff in E-Mail- und Anzeige-Templates* nachlesen.

2.11 App-Menüs

Die erweiterte Funktion *App-Menüs* ermöglicht es Ihnen im Menü auf beliebige Webseiten zu verlinken. Sollten Sie Links auf der App-Übersichtsseite anlegen wollen, lesen Sie bitten bei *Benutzer-Menüs* nach.

Haben Sie beispielsweise Auswertungen mittels der Anzeige-Templates erstellt, so können Sie auf diesem Wege einfach und schnell auf diese zugreifen (siehe *Link zu einem Anzeige-Template*).

2.11.1 Erstellung von App-Menüs

Wählen Sie im Menü *Konfiguration → Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *App-Menüs* und dann auf *Neuen Link oder neues Menü zur Menüleiste hinzufügen*.

Im nun geöffneten Fenster können folgende Konfigurationen zur *Erstellung eines Links* vorgenommen werden.

Identifizierer
: Der eindeutige Name des Links. Der Name darf nur Buchstaben, Ziffern und „_“ beinhalten.

Beschriftung
: Der Link-Text.

Icon
: Das Icon für den Link. Verwendet werden können alle Namen aus dem FontAwesome-Icon-Set.

Anzeigen auf
: Gibt an, wo das Menü erscheint. Der Link kann sowohl auf der *App-Startseite?*, *Eingabeformular?*, *iframe-Seite?* und *eigener Übersichtsseite?* erscheinen.

Position in der Menüleiste
: Hier kann, wenn ein Untermenü angelegt werden soll, mit *Übergeordnetes Menü* ein Menü ausgewählt werden. Mit *Reihenfolge* kann die Reihenfolge bestimmt werden, in der Menüs der gleichen Ebene sortiert werden. Je grösser die Zahl, desto weiter hinten erscheint der Menüpunkt.

Link-Ziel
: Hier kann aus verschiedenen Zielen ausgewählt werden, auf die ein Link zeigen kann. Bei den Zielen *Eingabeformular (öffentlich)*, *Eingabeformular (einebettet)*, *Daten-Management*, *Eigene Übersichtsseite*, *Auswertung*, *Import/Export*, *Arbeitsaufgaben*, *Konfiguration: Eingabe*, *Konfiguration: Arbeitsaufgaben*, *Konfiguration: Daten*, *Konfiguration: Berechtigungen*, *Konfiguration: Erweitert*, *Anzeige-Template* und *Daten-Auswertung* wird eine Auswahlmöglichkeit für den jeweiligen Typ angeboten. Bei *Eigener Link* kann eine beliebige URL angegeben werden. Der letzte Punkt *Kein Link* muss angewählt werden, wenn ein Menüeintrag erstellt werden soll, an dem ein Untermenü hängt.

Link-Attribute
: In diesem Abschnitt können die HTML-Attribute *Title*, *Target* und *Class* festgelegt werden.

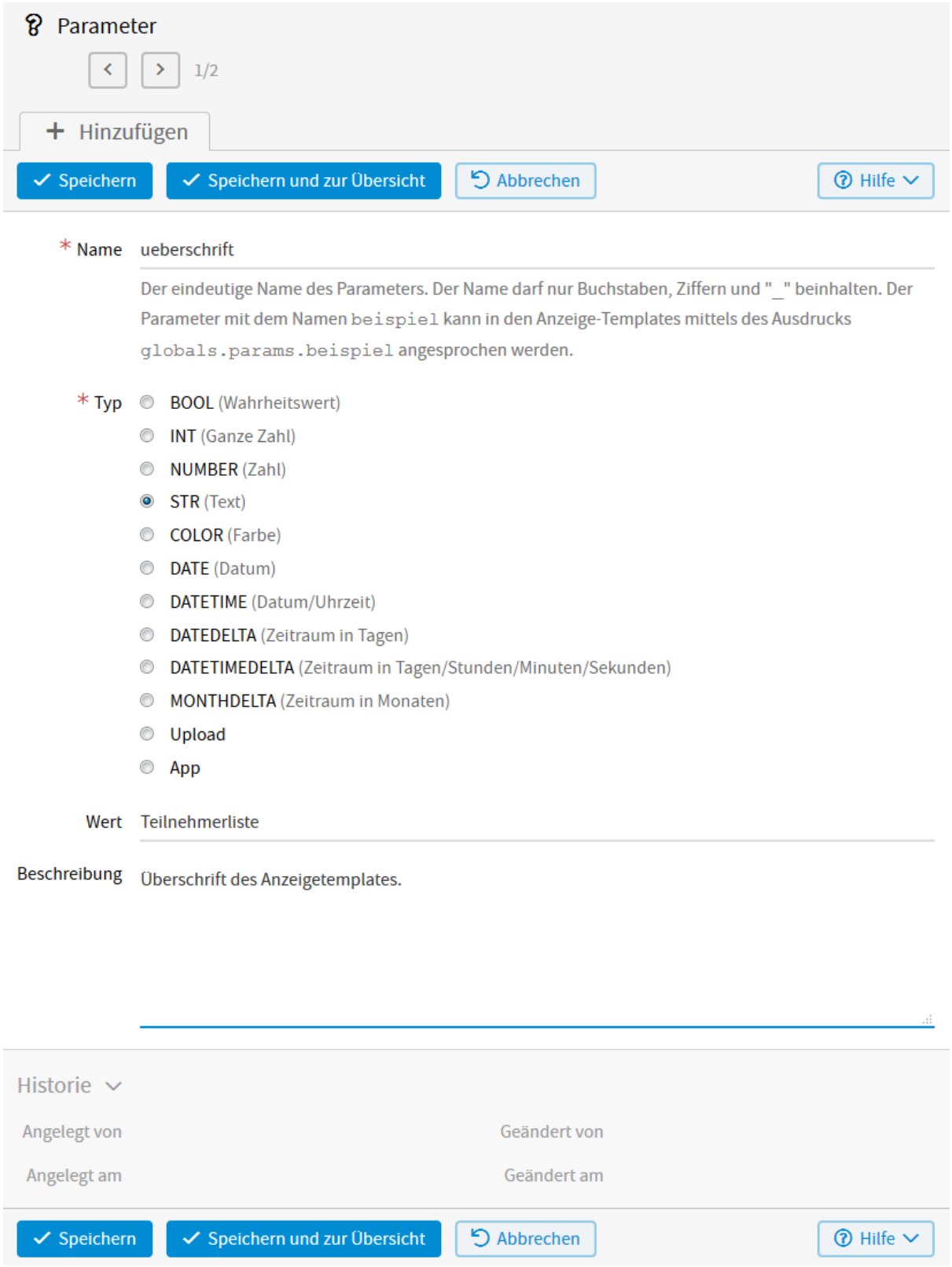


Abb. 79: App-Parameter erstellen

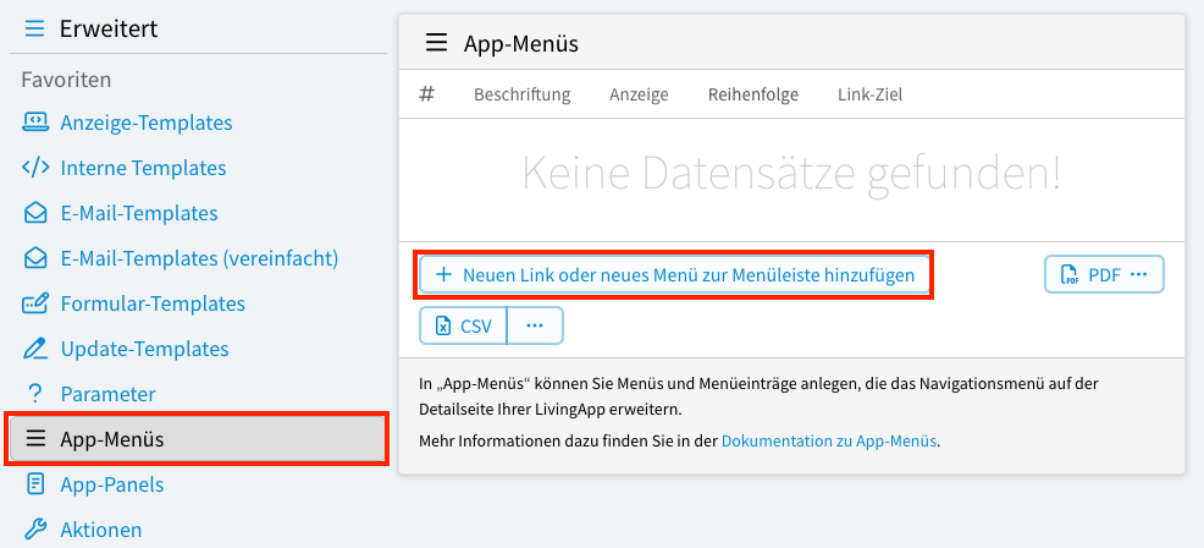


Abb. 80: Hinzufügen eines Menü-Links

Zeitsteuerung
: Hier kann festgelegt werden in welchem Zeitraum das Menü angezeigt werden soll. Mit *Aktivierung* legt man den Zeitpunkt fest, ab dem der Link angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link sofort aktiviert. Mit *Deaktivierung* kann angegeben werden ab wann der Link nicht mehr angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link nicht automatisch deaktiviert.

2.11.2 Anwendungsbeispiel

Um die Funktionsweise der Links zu veranschaulichen wird im Folgenden wieder auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen.

Link zu einem Anzeige-Template

Zuerst soll ein Link als *Menüpunkt in der Detailmaske* erstellt werden, der zur Teilnehmerliste führt, die mittels eines *Anzeige-Templates* angelegt wurde.

Wählen Sie dafür im Menü *Konfiguration* → *Erweitert* und klicken anschließend in der linken Menüleiste auf *App-Menüs* und dann auf *Neuen Link oder neues Menü zur Menüleiste hinzufügen* um einen neuen Link zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Dabei wird insbesondere bei *Link-Ziel* der Typ *Kein Link* ausgewählt.

Anschliessend ist in der Menü-Liste der neue *Link* Datensatz zu sehen, bei dem der Button *Neuer Menüpunkt* angeboten wird.

Nach dem Klick auf *Neuer Menüpunkt* ist im Abschnitt *Position in der Menüleiste* das Feld *Übergeordnetes Menü* bereits mit dem Wert *Links* vorbelegt.

Dann wird im Abschnitt *Link-Ziel* bei *Typ* der Wert *Anzeige-Template* ausgewählt.

Danach kann mit einem Klick auf den Button *Auswählen* ein Anzeige-Template gewählt werden:

Nach dem Abspeichern erscheint *Links* als *Menüpunkt in der Detailmaske*.

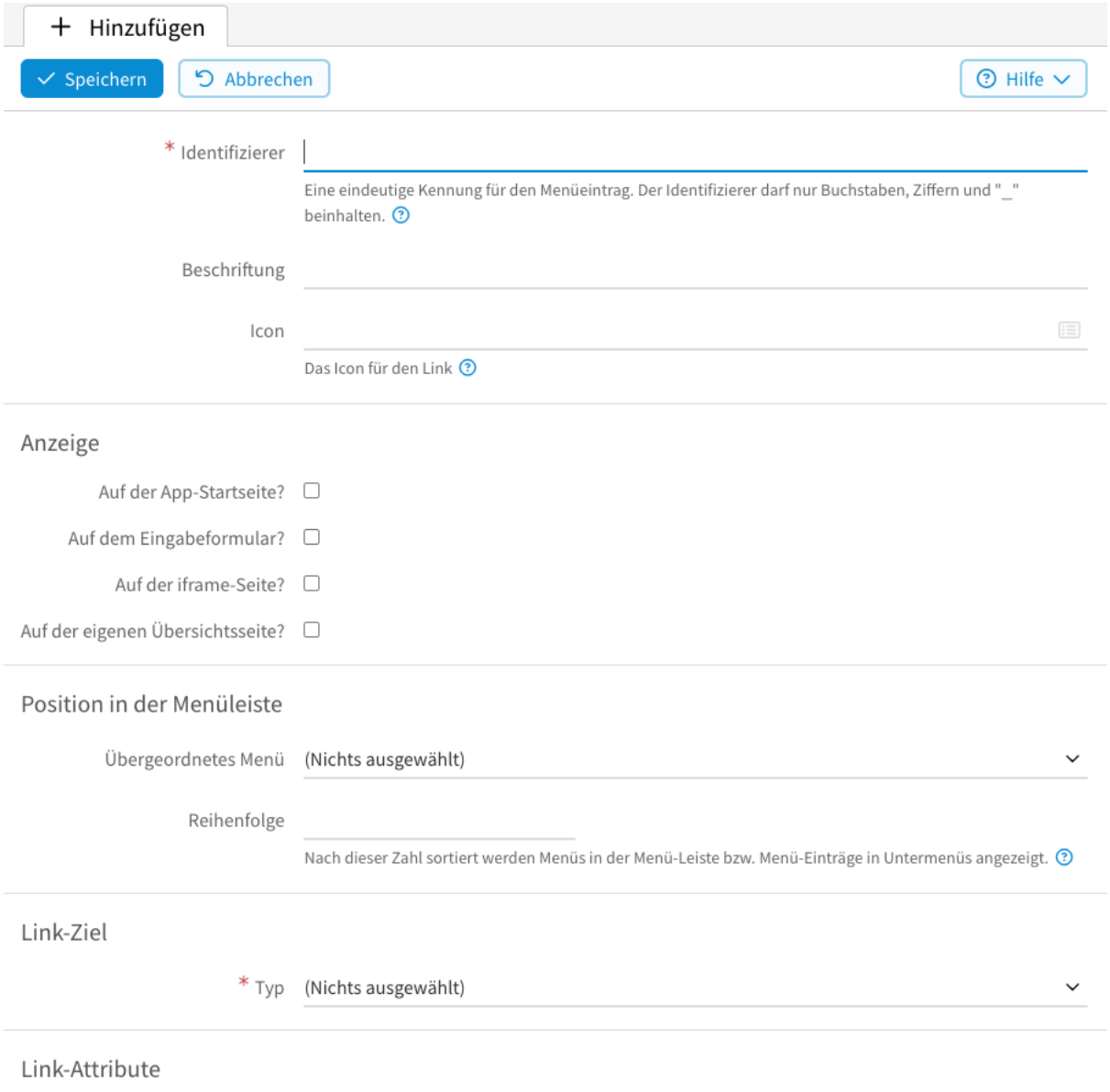


Abb. 81: Erstellung eines Menüs

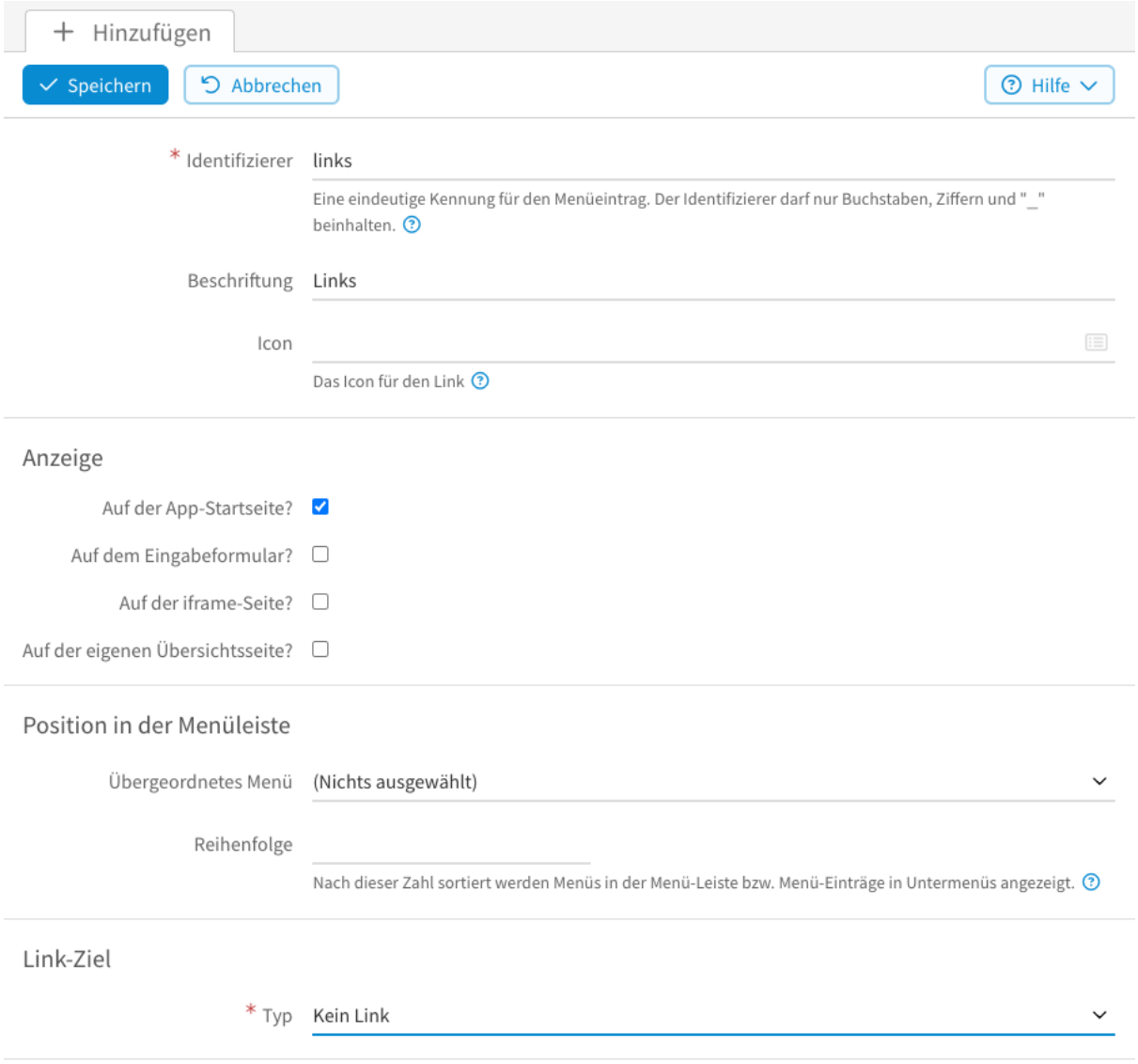


Abb. 82: Übergeordnetes Menü ‚Links' erstellen

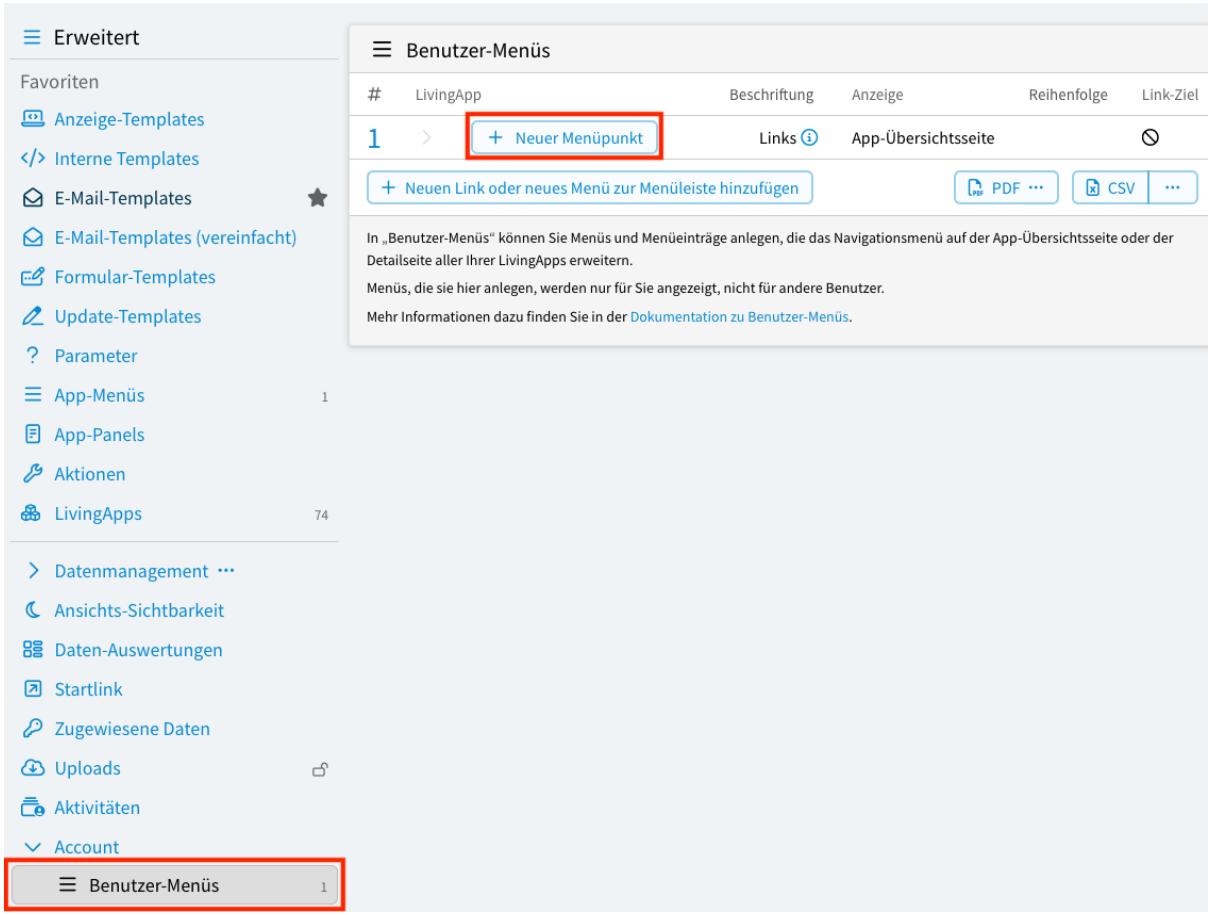


Abb. 83: Neues Menü ‚Links‘

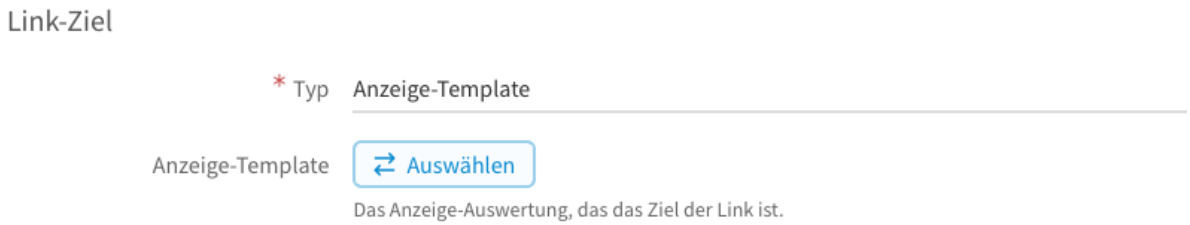


Abb. 84: Link-Ziel mit Auswählen-Button

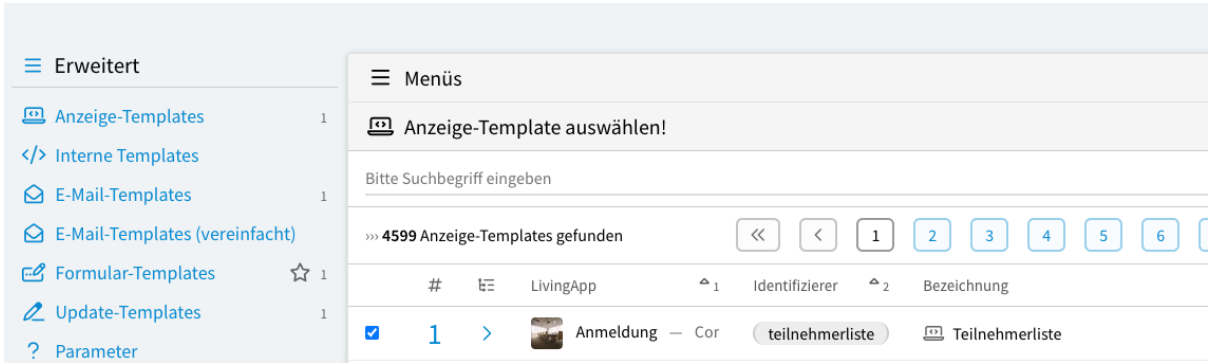


Abb. 85: Auswahl des Anzeige-Templates

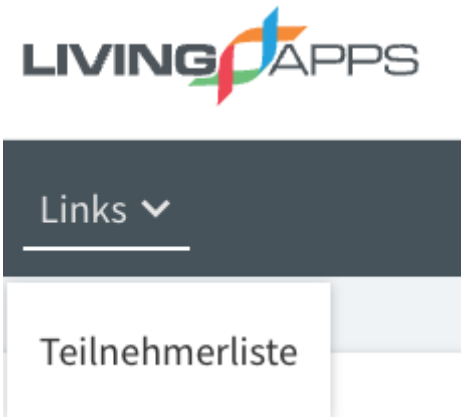


Abb. 86: Link als Menüpunkt in der Detailmaske

2.12 App-Panels

Die erweiterte Funktion *App-Panels* ermöglicht es Ihnen auf verschiedenen Seiten Kacheln mit z.B. Hinweistexten oder Links zu platzieren. Sollten Sie Links auf der App-Übersichtsseite anlegen wollen, lesen Sie bitten bei *Benutzer-Panels* nach.

Z.B. Sie erstellen einen Link und nutzen diesen als Textelement, das z. B. Erläuterungen zum Formular enthält und neben der Dateneingabe anzeigt wird (siehe *Link zum Anlegen von Textelementen*).

2.12.1 Erstellung von App-Panels

Wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *App-Panels* und dann auf *Neues Panel hinzufügen*.

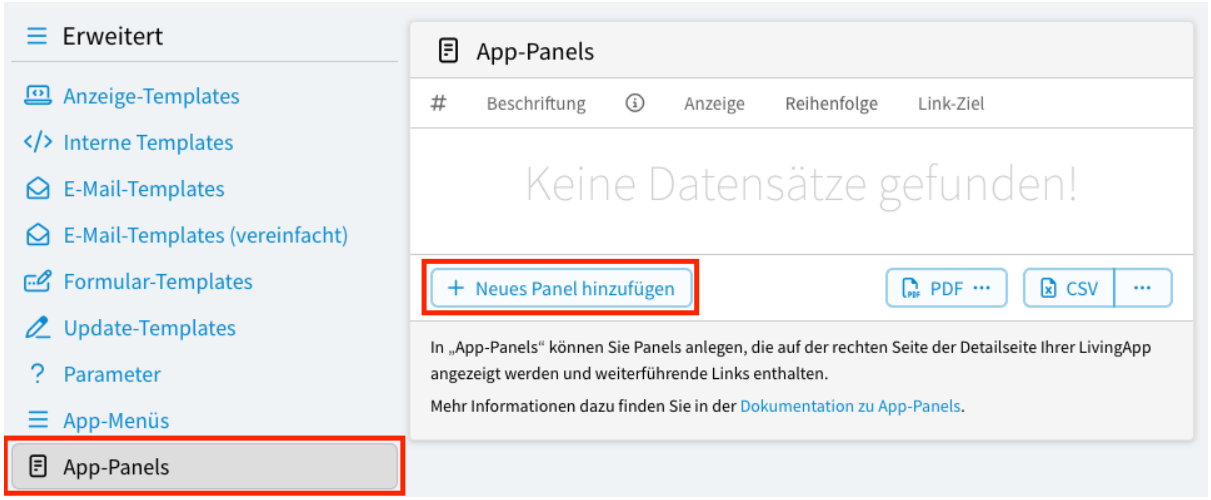


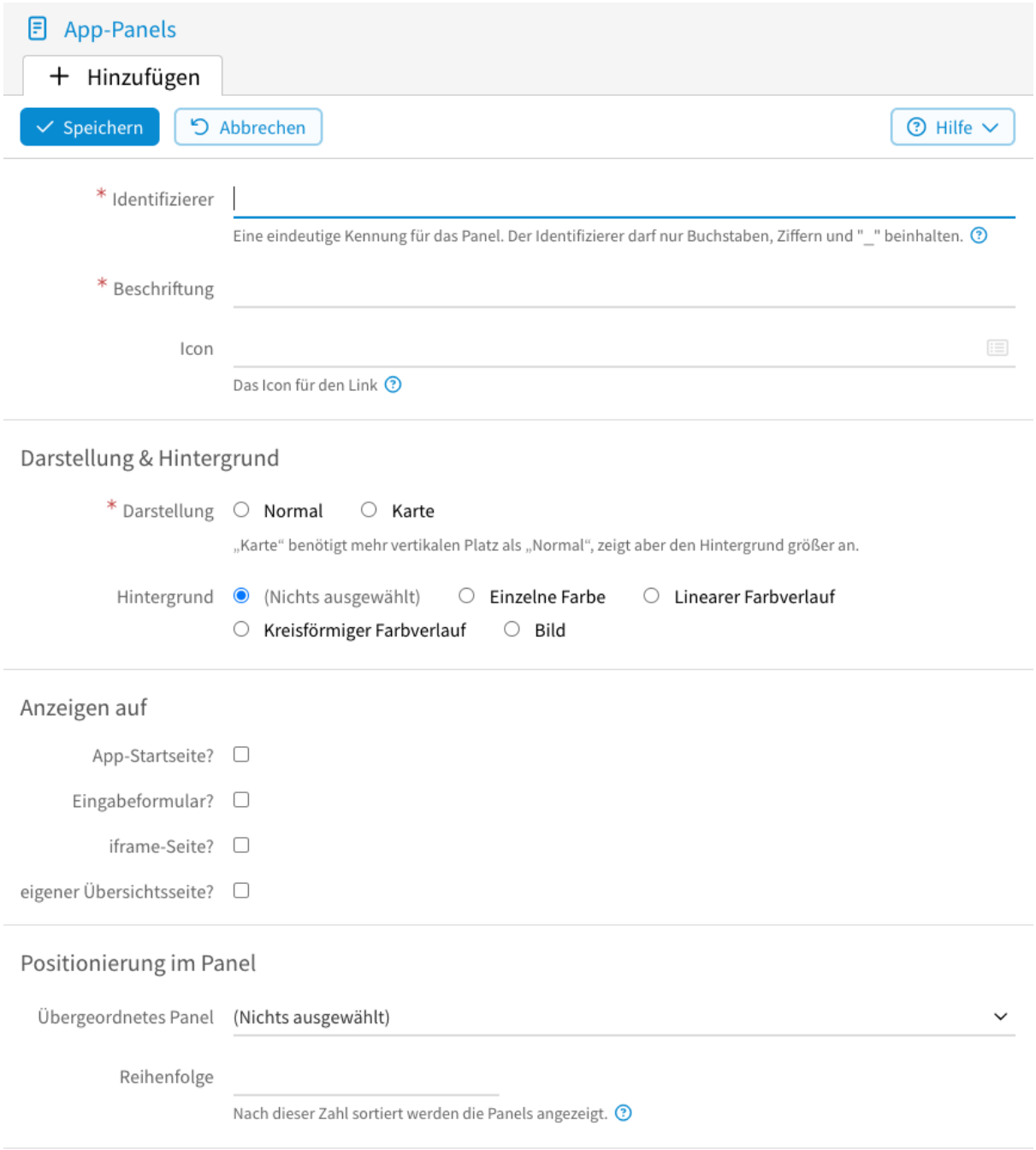
Abb. 87: Hinzufügen eines App-Panels

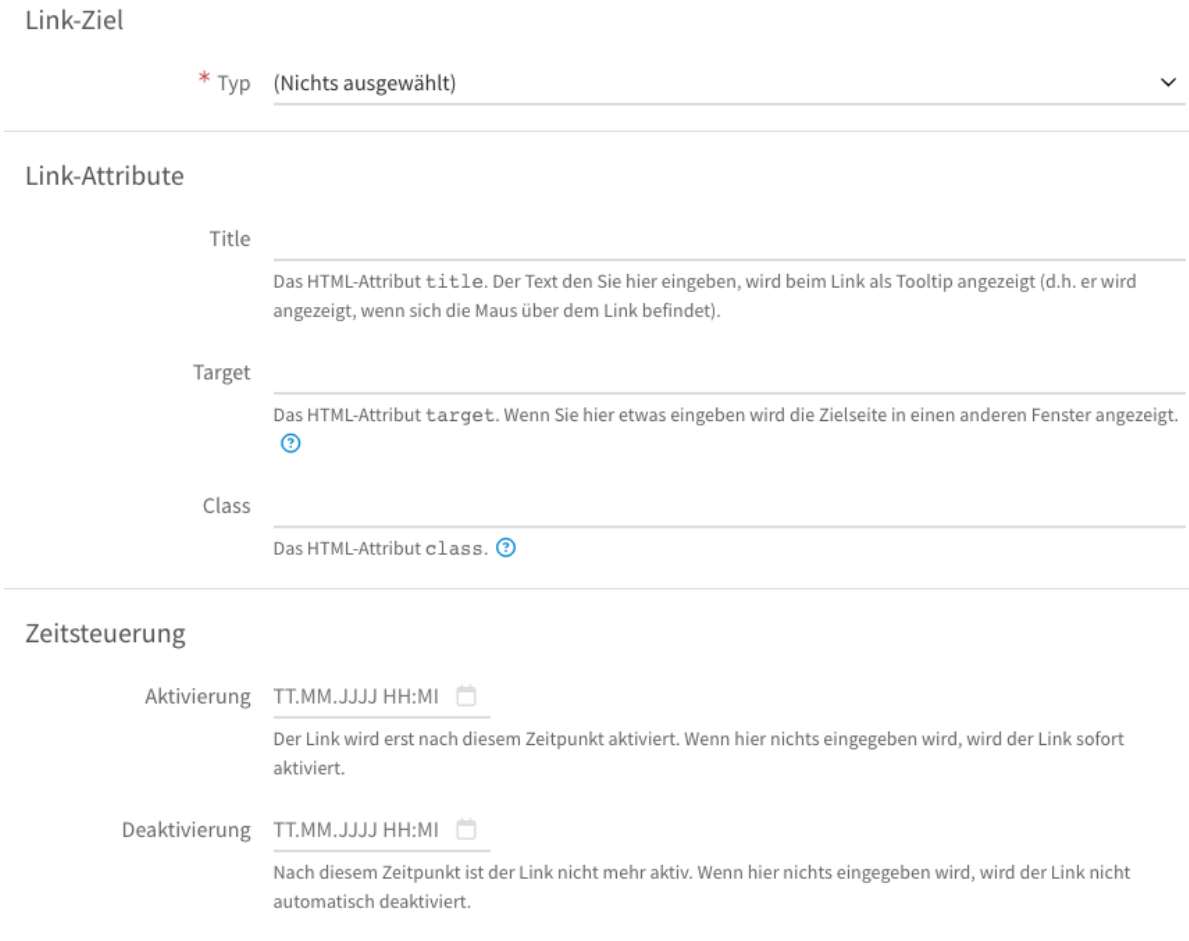
Im nun geöffneten Fenster können folgende Konfigurationen zur *Erstellung eines Links* vorgenommen werden.

Identifizierer
: Der eindeutige Name des Links. Der Name darf nur Buchstaben, Ziffern und „_“ beinhalten.

Beschriftung
: Der Text im Kopfbereich des Panels.

Icon
: Das Icon für den Link. Verwendet werden können alle Namen aus dem FontAwesome-Icon-Set.


App-Panels
Hinzufügen
Speichern
Abbrechen
Hilfe
* Identifizierer
Eine eindeutige Kennung für das Panel. Der Identifizierer darf nur Buchstaben, Ziffern und "_" beinhalten.
* Beschriftung
Icon
Das Icon für den Link
Darstellung & Hintergrund
* Darstellung
Normal
Karte
„Karte" benötigt mehr vertikalen Platz als „Normal", zeigt aber den Hintergrund größer an.
Hintergrund
(Nichts ausgewählt)
Einzelne Farbe
Linearer Farbverlauf
Kreisförmiger Farbverlauf
Bild
Anzeigen auf
App-Startseite?
Eingabeformular?
iframe-Seite?
eigener Übersichtsseite?
Positionierung im Panel
Übergeordnetes Panel
(Nichts ausgewählt)
Reihenfolge
Nach dieser Zahl sortiert werden die Panels angezeigt.


Link-Ziel
* Typ
(Nichts ausgewählt)
Link-Attribute
Title
Das HTML-Attribut title. Der Text den Sie hier eingeben, wird beim Link als Tooltip angezeigt (d.h. er wird angezeigt, wenn sich die Maus über dem Link befindet).
Target
Das HTML-Attribut target. Wenn Sie hier etwas eingeben wird die Zielseite in einen anderen Fenster angezeigt.
Class
Das HTML-Attribut class.
Zeitsteuerung
Aktivierung
TT.MM.JJJJ HH:MI
Der Link wird erst nach diesem Zeitpunkt aktiviert. Wenn hier nichts eingegeben wird, wird der Link sofort aktiviert.
Deaktivierung
TT.MM.JJJJ HH:MI
Nach diesem Zeitpunkt ist der Link nicht mehr aktiv. Wenn hier nichts eingegeben wird, wird der Link nicht automatisch deaktiviert.

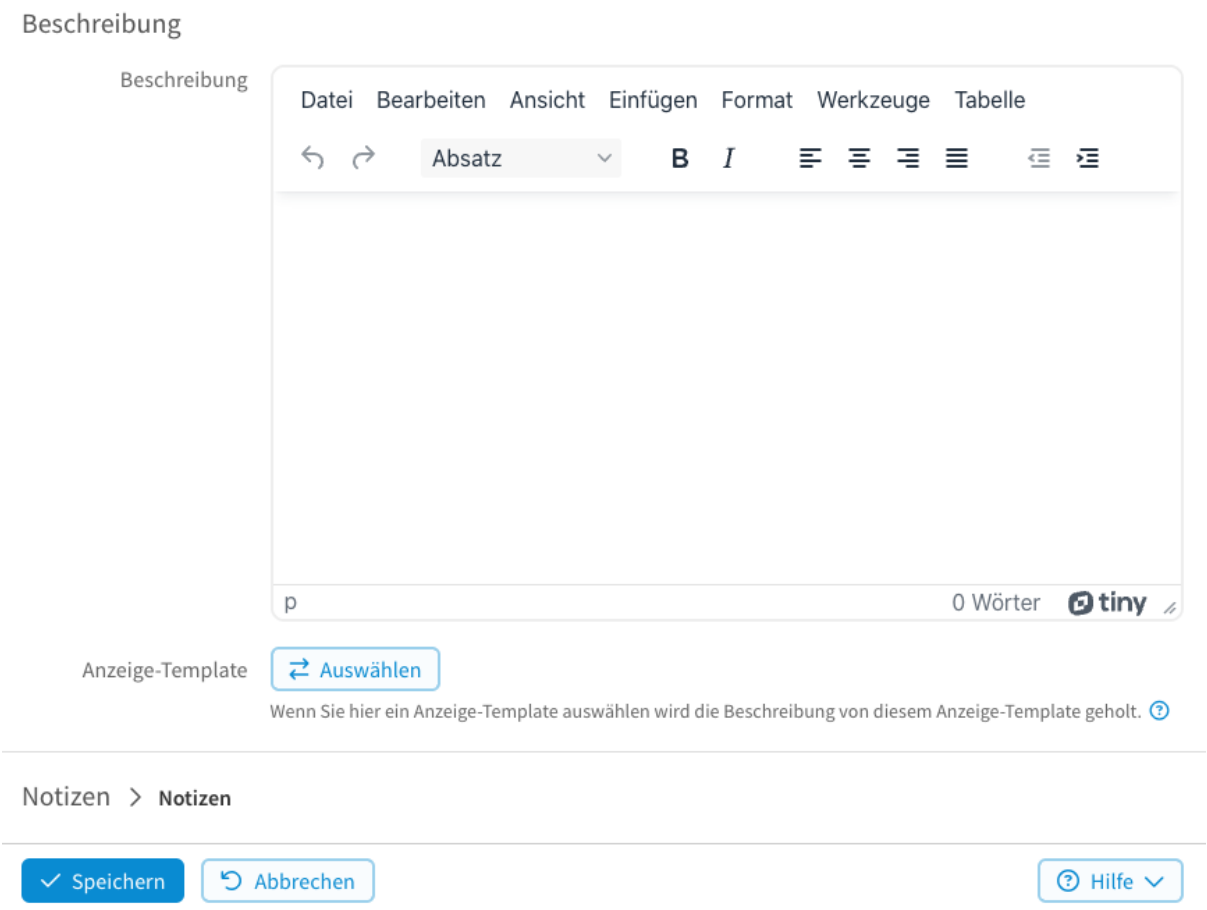


Abb. 88: Erstellung eines App-Panels

Darstellung
Normal bzw. *Karte* kann hier ausgewählt werden. *Karte* führt zu einer etwas höheren Darstellung mit mehr Hintergrund.

Hintergrund
Hier besteht die Möglichkeit zwischen *Einzelner Farbe*, *Linearer Farbverlauf*, *Kreisförmiger Farbverlauf* und *Bild* zu wählen. Je nach Auswahl werden anschliessend Felder für *Textfarbe*, *Hintergrundfarbe oben*, *Hintergrundfarbe unten* oder ein *Bild* angeboten.

Anzeigen auf
Gibt an, wo das Panel erscheint. Der Link kann sowohl auf der *App-Startseite?*, *Eingabeformular?*, *iframe-Seite?* und *eigener Übersichtsseite?* erscheinen.

Positionierung im Panel
Hier kann, wenn Panels ineinander verschachtelt werden sollen, mit *Übergeordnetes Panel* ein Panel ausgewählt werden. Mit *Reihenfolge* kann die Reihenfolge bestimmt werden, in der Panels der gleichen Ebene sortiert werden. Je grösser die Zahl, desto weiter hinten erscheint die Kachel.

Link-Ziel
Hier kann aus verschiedenen Zielen ausgewählt werden, auf die ein Link zeigen kann. Bei den Zielen *Eingabeformular (öffentlich)*, *Eingabeformular (einebettet)*, *Daten-Management*, *Eigene Übersichtsseite*, *Auswertung*, *Import/Export*, *Arbeitsaufgaben*, *Konfiguration: Eingabe*, *Konfiguration: Arbeitsaufgaben*, *Konfiguration: Daten*, *Konfiguration: Berechtigungen*, *Konfiguration: Erweitert*, *Anzeige-Template* und *Daten-Auswertung* wird eine Auswahlmöglichkeit für den jeweiligen Typ angeboten. Bei *Eigener Link* kann eine beliebige URL angegeben werden. Der letzte Punkt *Kein Link* muss angewählt werden, wenn ein Panel erstellt werden soll, in dem sich andere Panels befinden.

Link-Attribute
In diesem Abschnitt können die HTML-Attribute *Title*, *Target* und *Class* festgelegt werden.

Zeitsteuerung
Hier kann festgelegt werden in welchem Zeitraum das Panel angezeigt werden soll. Mit *Aktivierung* legt man den Zeitpunkt fest, ab dem der Link angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link sofort aktiviert. Mit *Deaktivierung* kann angegeben werden ab wann der Link nicht mehr angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link nicht automatisch deaktiviert.

Beschreibung
Hier kann ein Beschreibungstext angegeben werden, der in dem Panel angezeigt wird. Alternativ kann auch ein *Anzeige-Template* ausgewählt werden, das den Beschreibungstext ausgibt.

2.12.2 Anwendungsbeispiel

Um die Funktionsweise der App-Panels zu veranschaulichen wird im Folgenden wieder auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen.

Link zum Anlegen von Textelementen

Es soll ein Panel erstellt werden, das zum Anlegen eines Textelements verwendet wird. Dieses Textelement soll Hinweise des Veranstalters enthalten und bei der Eingabe einer Anmeldung rechts vom Eingabeformular angezeigt werden (*Eingabeformular*).

Wählen Sie dafür im Menü *Konfiguration* → *Erweitert* und klicken anschließend in der linken Menüleiste auf *App-Panels* und dann auf *Neues Panel hinzufügen* um ein neues Panel zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Nach Abspeichern des Links erscheint dieser dann rechts vom Formular in der Dateneingabe (siehe *Link als Textelement*).

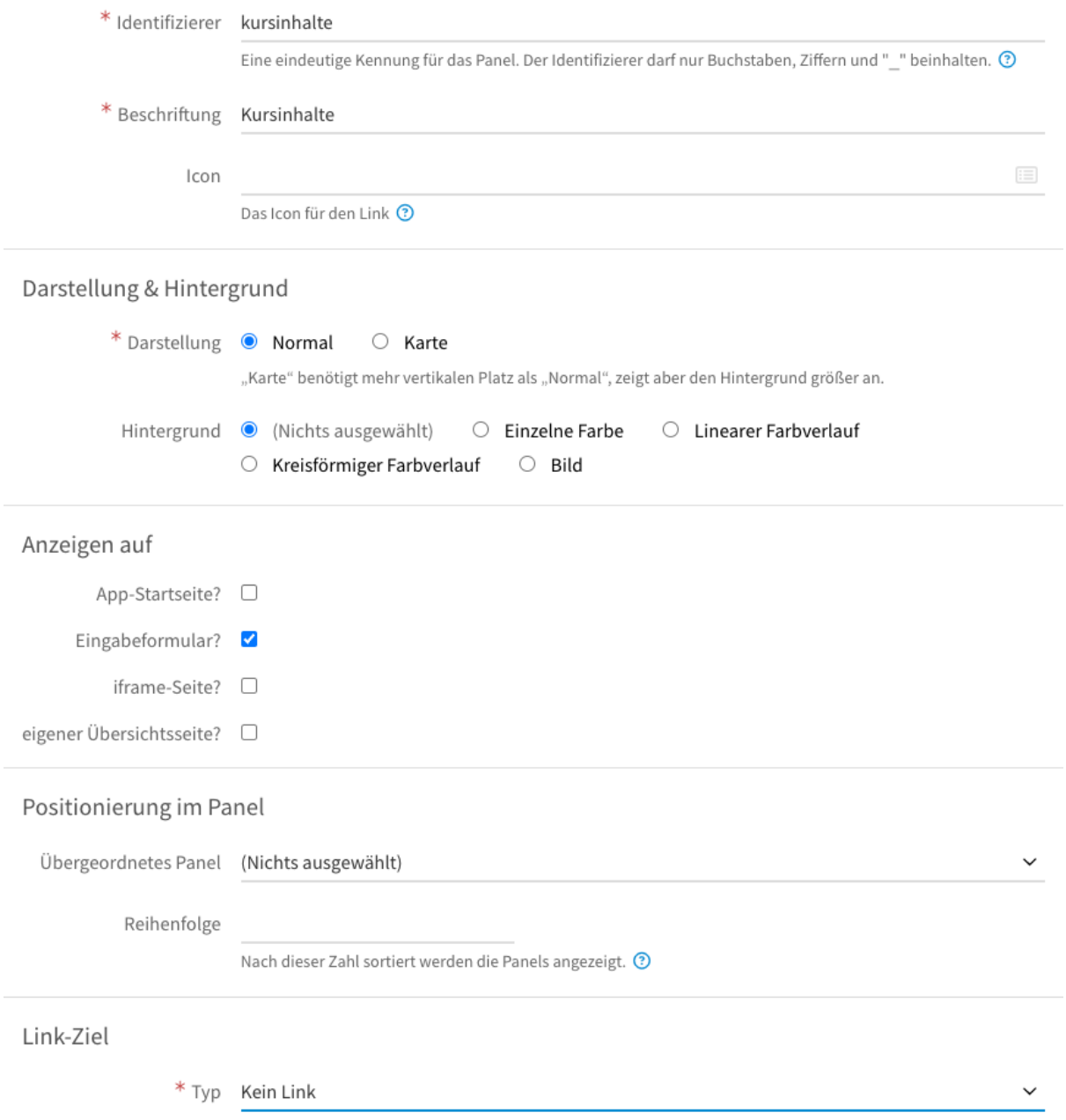


Abb. 89: Erstellen eines App-Panels als Textelement

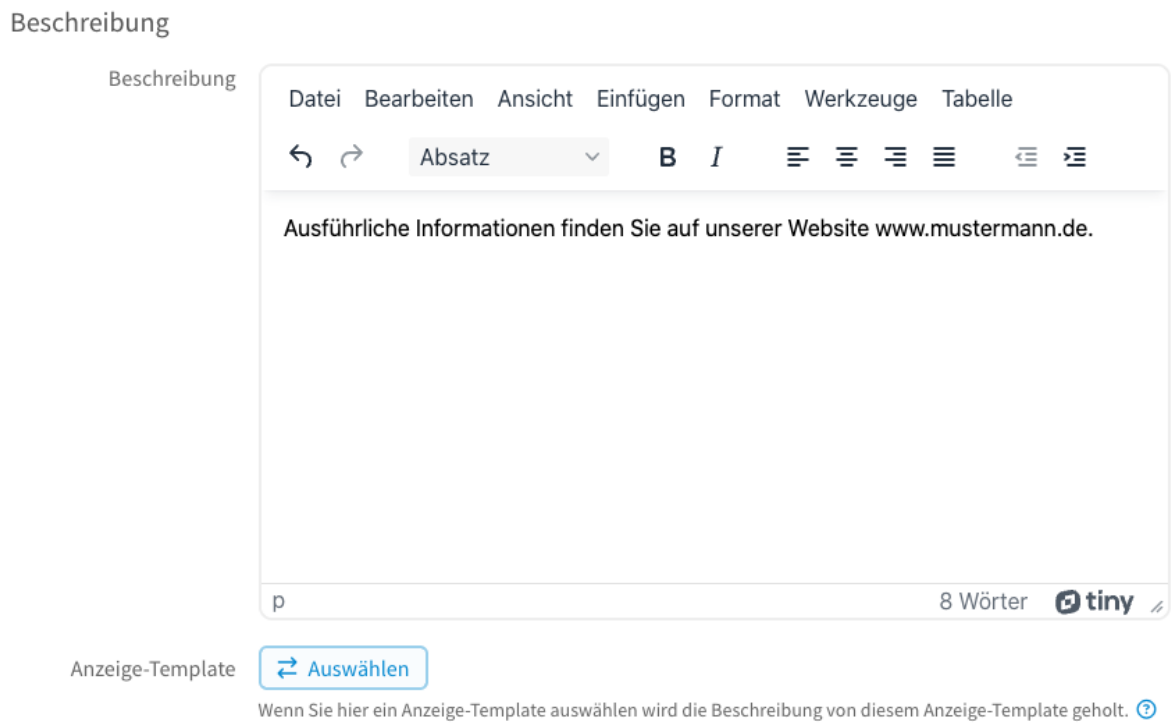


Abb. 90: Erstellen eines App-Panels als Textelement

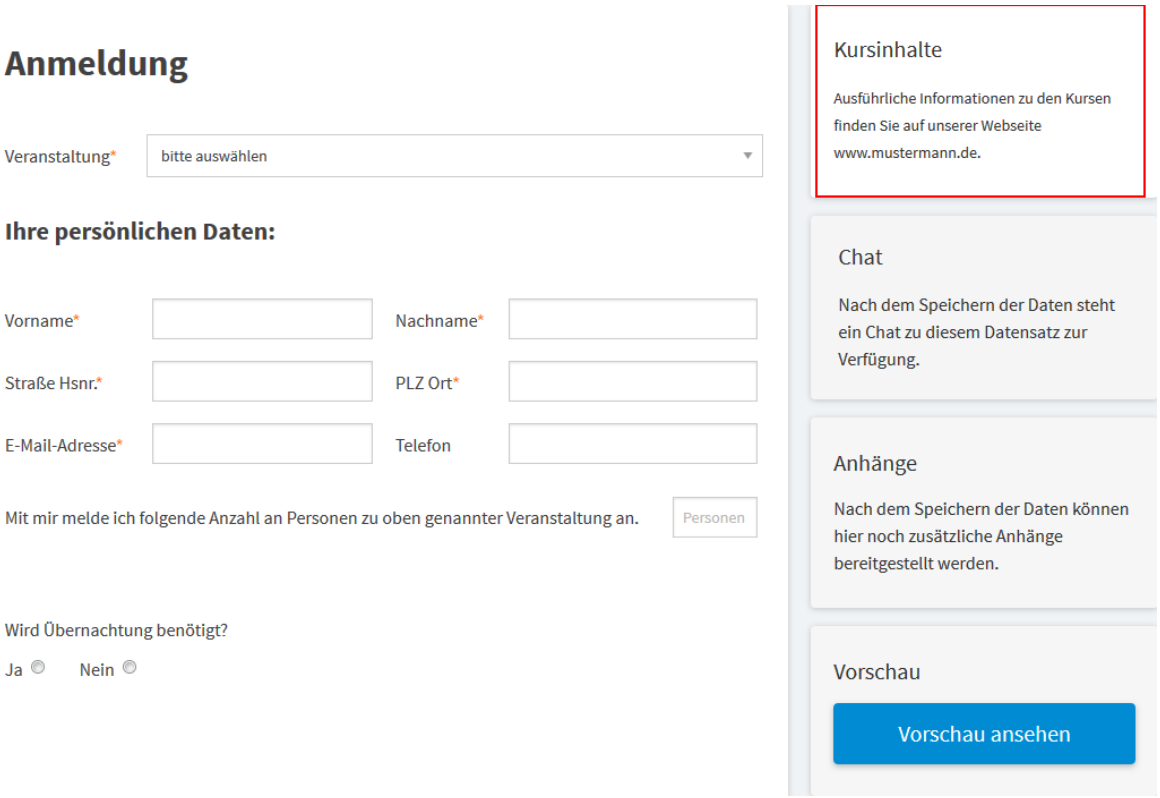


Abb. 91: Links als Textelement in der Eingabemaske

2.13 Datenmanagement

Unter *Konfiguration* → *Erweitert - Datenmanagement* können verschiedene Einstellungen für die Datenliste vorgenommen werden.

2.13.1 Farben

Mit *Farben* kann konfiguriert werden, dass Datensätze in der Liste unter bestimmten Bedingungen eingefärbt werden. So können z. B. Datensätze, die bestimmte Werte besitzen, optisch hervorgehoben werden. Im Beispiel sollen die Datensätze von Teilnehmern, die noch nicht bezahlt haben, rot eingefärbt werden.

Erstellung

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Datenmanagement* → *Farben* aus. Klicken Sie auf *Hinzufügen* um eine neue Einfärbung einzustellen. (Siehe *Einfärbung hinzufügen* und *Einfärbung erstellen*)

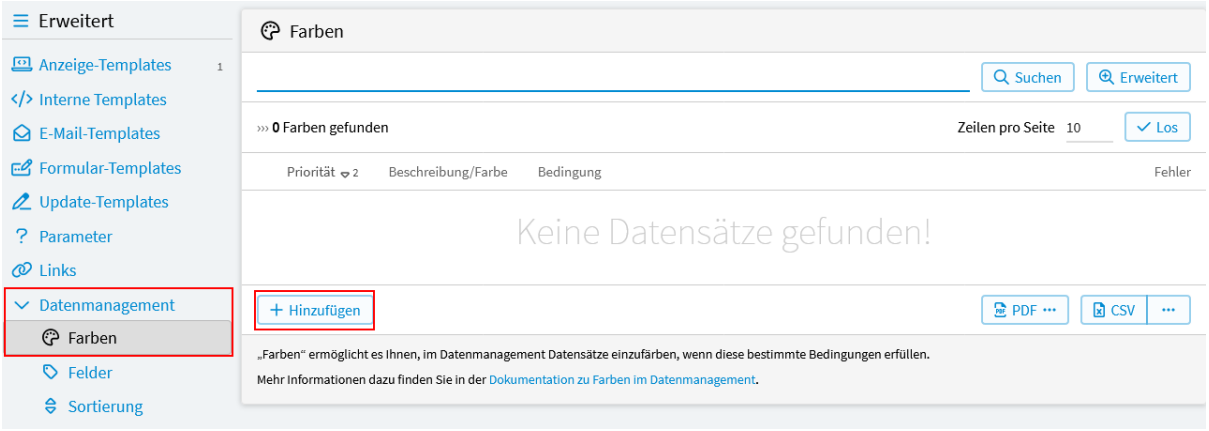


Abb. 92: Einfärbung hinzufügen

Priorität
: Die Bedingungen werden in absteigender Priorität getestet. Die erste Bedingung, die erfüllt ist, bestimmt die Farbe.

Farbe
: Wählen Sie hier die Farbe, mit der die Zeile im Datenmanagement eingefärbt werden soll, wenn die angegebene Bedingung erfüllt ist.

Beschreibung
: Diese Beschreibung wird im Datenmanagement in der Legende unter der Liste angezeigt.

Bedingung
: Diese Bedingung muss ein Datensatz erfüllen, um die zugehörige Farbe zugewiesen zu bekommen.

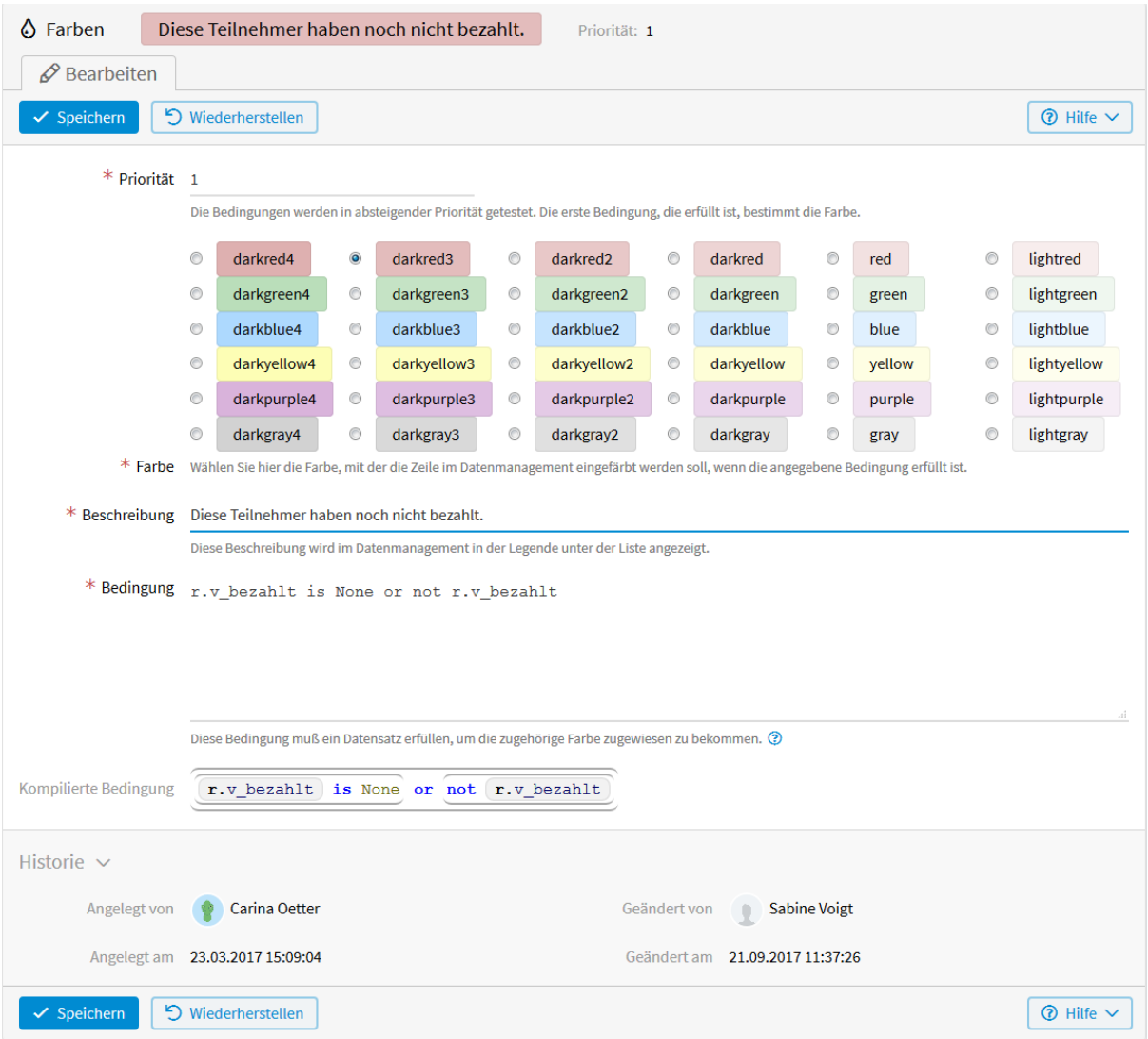


Abb. 93: Einfärbung erstellen

Formulieren der Bedingung

Für die Formulierung wird *vSQL* verwendet.

Um Werte von Feldern Ihres Formulars zu prüfen, werden die Identifizierer der Felder benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*. Auf die Werte kann dann zugegriffen werden mit `r.v_identifier` .

Sie haben unterschiedliche Prüfmöglichkeiten. Die Datentypen, Operationen, Attribute und Methoden, die Ihnen hier zur Verfügung stehen, finden Sie im Kapitel *vSQL* bzw. wenn Sie das `?` unterhalb des Feldes anklicken.

Um beispielsweise ein Text-Feld mit dem Identifizierer `name` auf einen bestimmten Wert zu prüfen, benötigen Sie folgende Bedingung: `r.v_name == "LivingApps"`.

Ist die Bedingung nicht vom Typ `BOOL` (d.h. liefert `True` oder `False`), wird sie mittels der vSQL-Funktion `bool` konvertiert.

Sobald Sie auf *Speichern* gedrückt haben, erscheint unterhalb des Bedingungsfelds die kompilierte Bedingung und falls beim Kompilieren ein Fehler aufgetreten ist, auch dieser Fehler.

2.13.2 Felder

Unter *Felder* können die Spalten in der Datenliste konfiguriert werden. Wählen Sie hier die Felder aus, die in der Liste angezeigt werden sollen und legen Sie die Reihenfolge fest, in welcher die Felder aufgelistet sein sollen.

Erstellung

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Datenmanagement* → *Felder* aus. Hier können Sie durch Anklicken die entsprechenden *Felder konfigurieren*.

Möchten Sie Änderungen direkt in der Übersicht vornehmen, um einen besseren Überblick zu behalten, klicken Sie auf *In Liste?*, *Reihenfolge* oder *Versteckt?* und dann jeweils auf *Bearbeiten ein/aus*.

In Liste
: Ist das Häkchen gesetzt, erscheint das Feld als Spalte in der Datenliste.

Reihenfolge
: Legt die Reihenfolge der Spalten in der Datenliste fest.

Versteckt
: Wird hier das Häkchen gesetzt, ist das Feld in der Datenliste und den Datensätzen nicht mehr sichtbar.

2.13.3 Sortierung

Bei *Sortierung* kann festgelegt werden, wonach die Datensätze in der Datenliste sortiert sein sollen. Wird nichts angegeben, so werden die Datensätze absteigend nach dem Erzeugungsdatum sortiert.

Erstellung

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Datenmanagement* → *Sortierung* aus. Klicken Sie auf *Hinzufügen* um eine *neue Sortierung* anzulegen.

Im nun geöffneten Fenster können folgende *Einstellungen* vorgenommen werden.

Reihenfolge
: Legt fest, in welcher Reihenfolge die Sortierungen (wenn mehrere konfiguriert sind) abgehandelt werden.

Ausdruck
: Nach dem Wert dieses Ausdrucks werden die Datensätze sortiert. Für die Formulierung wird *vSQL* verwendet. Um die Werte der Felder auszugeben, werden die eindeutigen *Identifizierer der Felder* benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*. Auf die Werte kann dann zugegriffen werden mit: `r.v_identifier`.

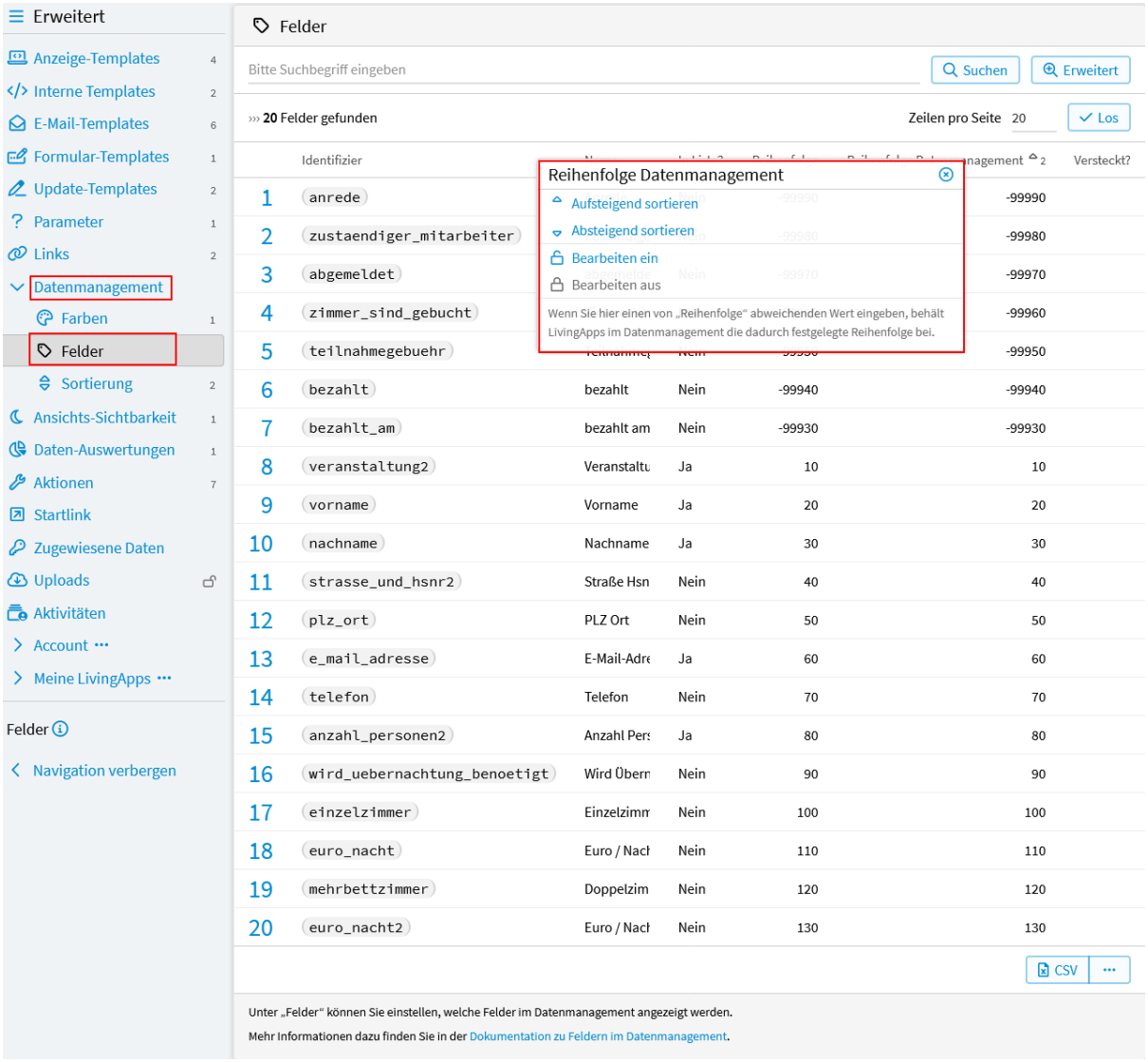


Abb. 94: Datenmanagement - Felder konfigurieren

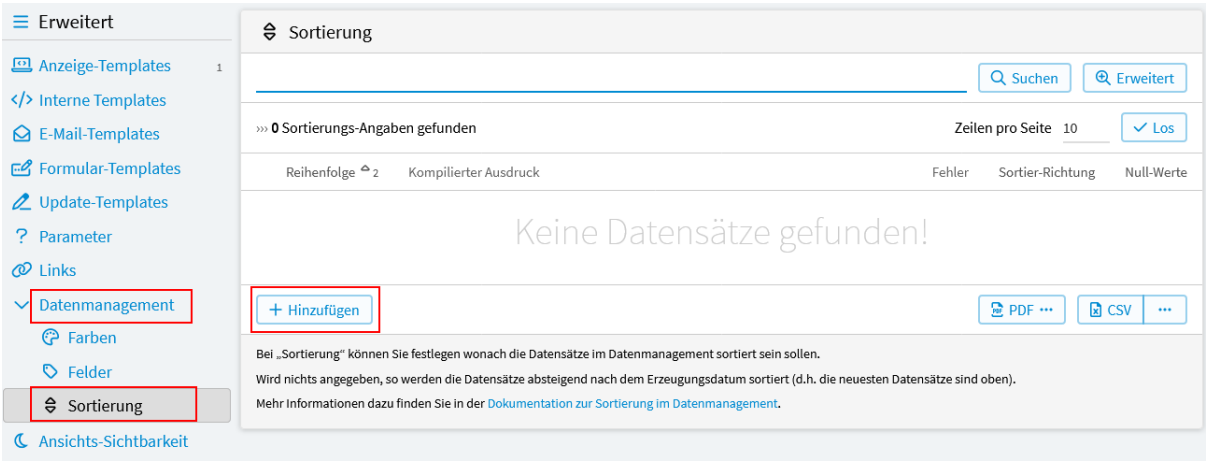


Abb. 95: Sortierung hinzufügen

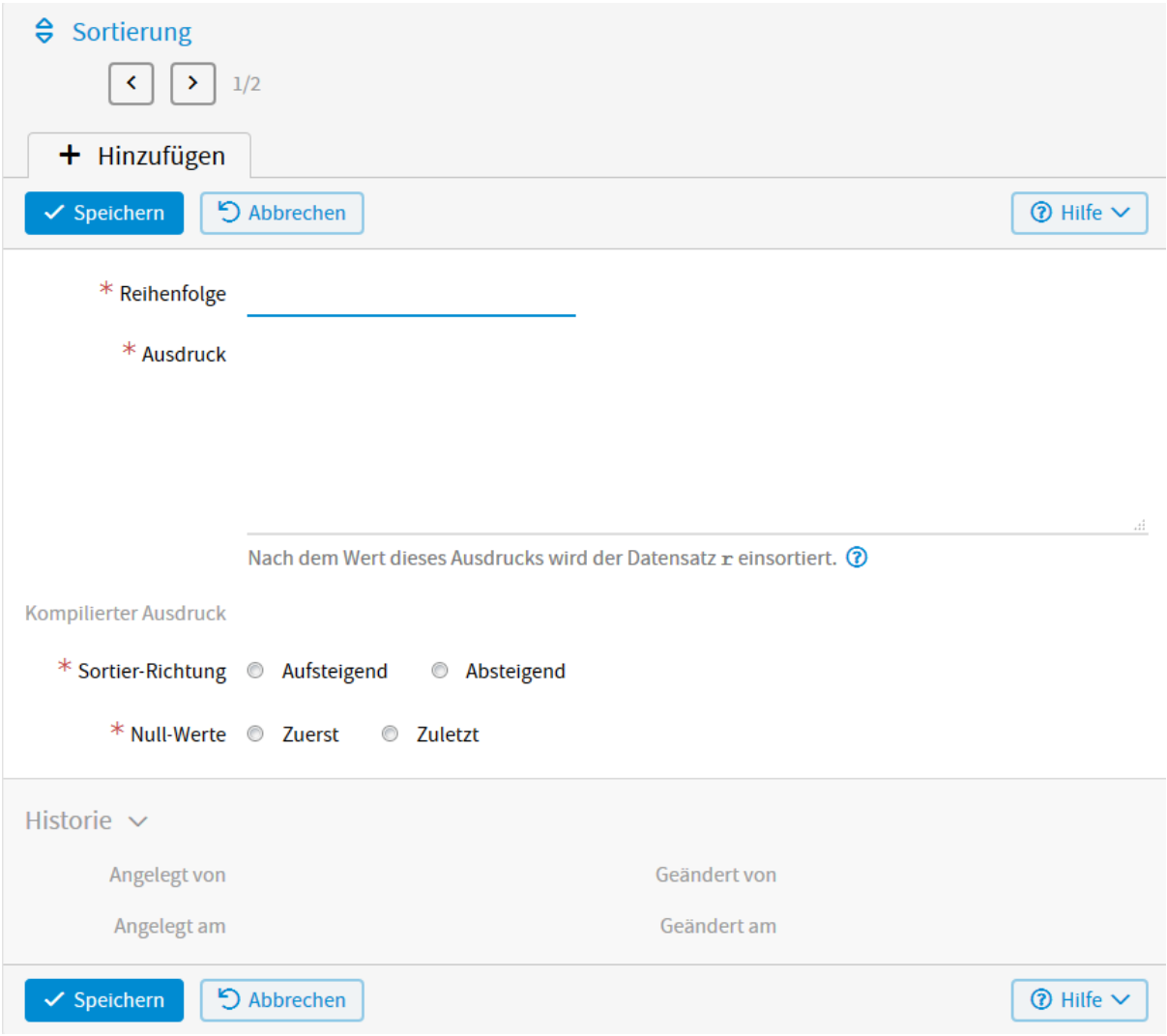


Abb. 96: Sortierung erstellen

Sortier-Richtung
: *Aufsteigend* (A-Z oder 1, 2, 3 oder altes Datum zuerst) *Absteigend* (Z-A oder 3, 2, 1 oder neues Datum zuerst)

Null-Werte
: Können zuerst oder zuletzt angezeigt werden.

Anwendungsbeispiel

Um die Funktionsweise der Sortierung zu veranschaulichen, wird im Folgenden auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen.

Die eingegangenen Anmeldungen sollen in der Datenliste sortiert nach Veranstaltung und nach dem Nachnamen der Teilnehmer angezeigt werden. Dafür müssen zwei Sortierungen angelegt werden.

Zuerst soll *nach Veranstaltung* sortiert werden.

Wählen Sie dafür im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Datenmanagement* → *Sortierung* aus und klicken auf *Hinzufügen*.

Für die *1. Sortierung* werden folgende Konfigurationen vorgenommen.

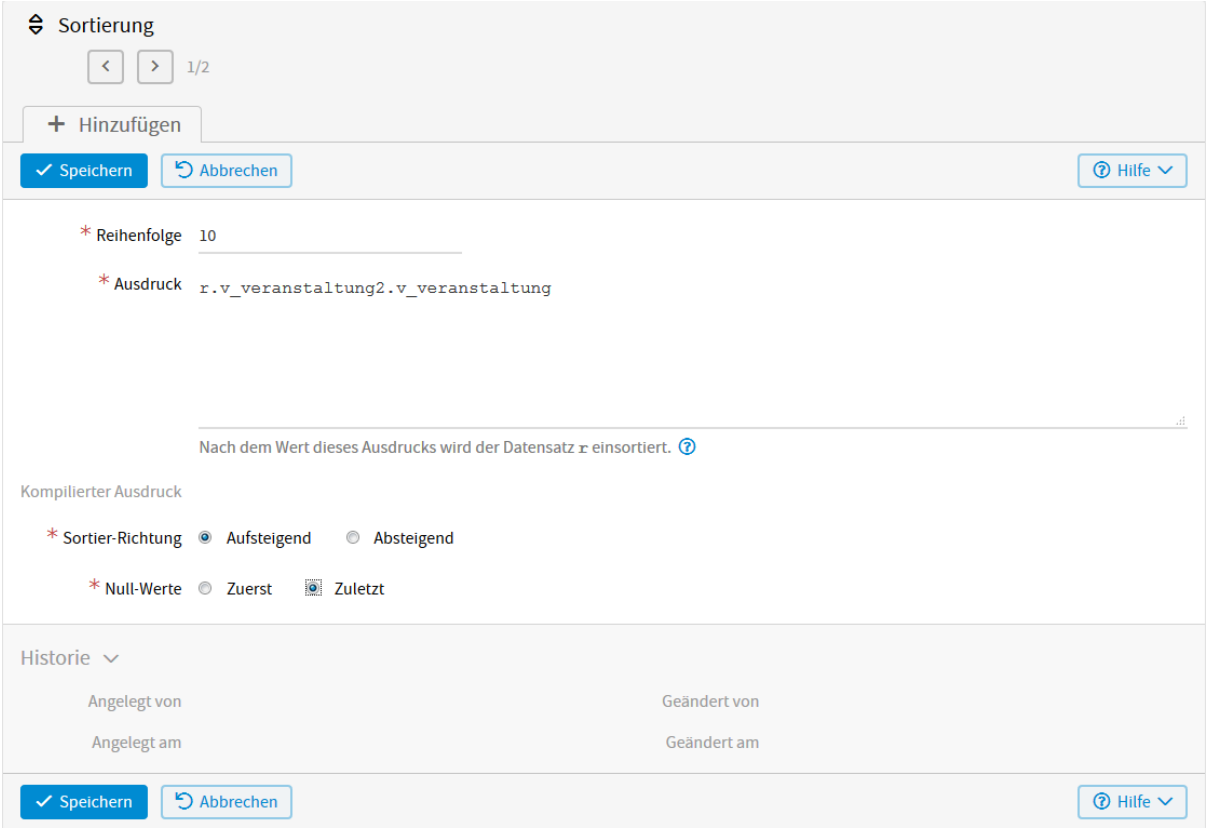


Abb. 97: Sortierung 1 erstellen

Im Anwendungsbeispiel lautet der Ausdruck `r.v_veranstaltung2.v_veranstaltung`. Dabei wird über das Feld `veranstaltung2` (applookup/select) der App „Anmeldung“ auf das Feld `veranstaltung` der verknüpften App „Veranstaltungen“ zugegriffen.

Bei der zweiten Sortierung soll zusätzlich nach dem *Nachnamen* sortiert werden.

Klicken Sie auf *Hinzufügen* um die *2. Sortierung* anzulegen.

So sieht dann das *Ergebnis der Sortierung* aus. Ohne Sortierung würden die Datensätze absteigend nach Erzeugungsdatum angezeigt werden.

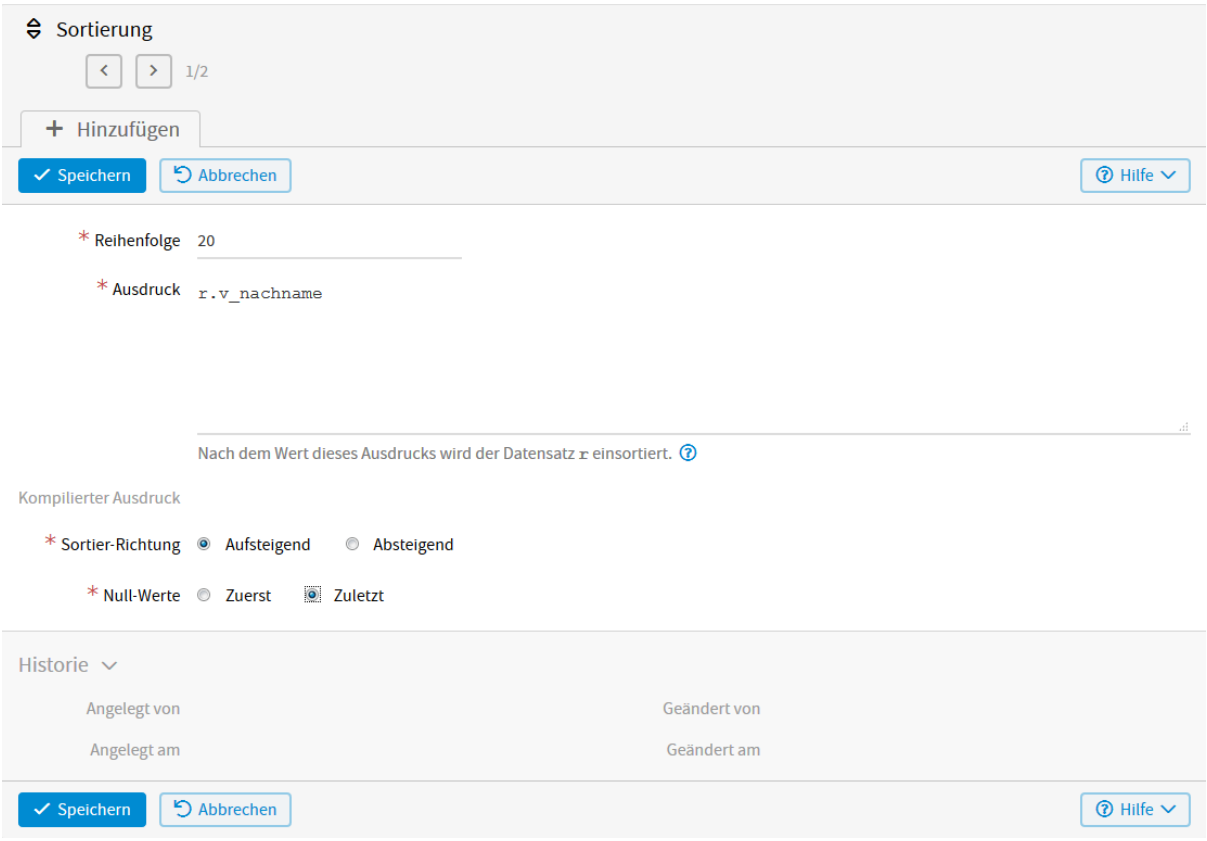


Abb. 98: Sortierung 2 erstellen

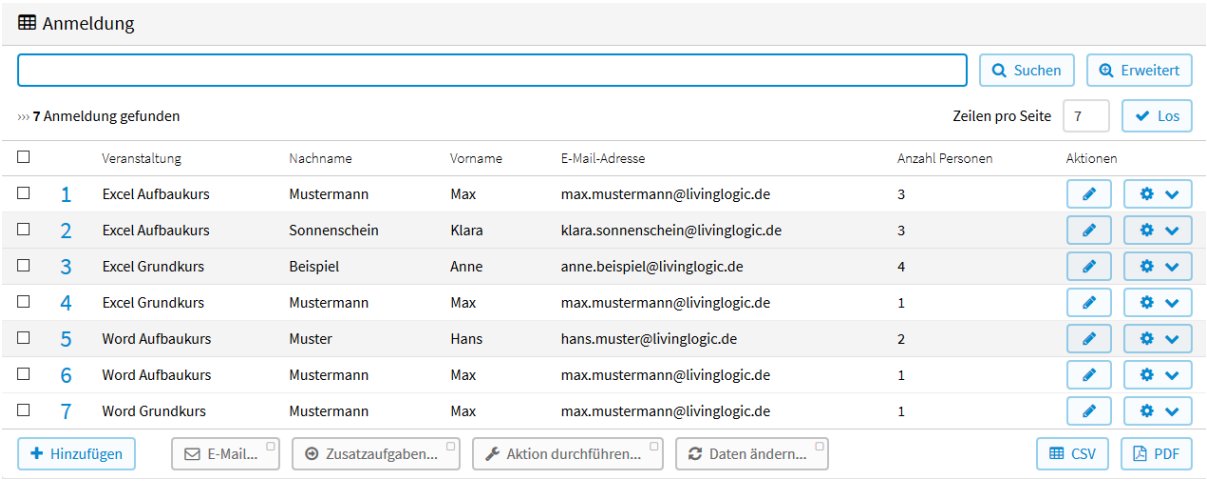


Abb. 99: Ergebnis der Sortierung

2.14 Ansichts-Sichtbarkeit

Unter *Ansichts-Sichtbarkeit* können Sie einen Gültigkeits-Zeitraum für Formular-Ansichten festlegen, sowie konfigurieren, was vor und nach diesem Zeitraum auf dem Formular angezeigt werden soll. Unterschiedliche Formular-Ansichten können erstellt werden unter *Konfiguration* → *Eingabe*.

2.14.1 Erstellung

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Ansichts-Sichtbarkeit*. Klicken Sie auf die *jeweilige Ansicht*, um dafür *Sichtbarkeits-Konfigurationen* festzulegen.

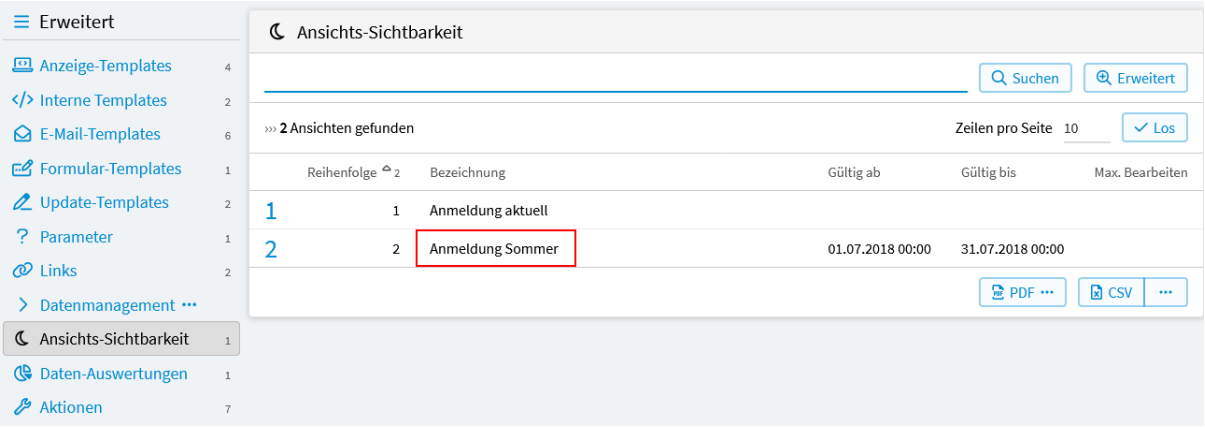


Abb. 100: Ansicht auswählen

Gültig ab
: Wählen Sie hier ein Datum, ab wann diese Ansicht sichtbar sein soll.

Gültig bis
: Wählen Sie hier ein Datum, bis wann diese Ansicht sichtbar sein soll.

Anzeigetemplate vorher
: Für den Zeitraum davor wird der hier eingegebene Text als Inhalt der Formularseite eingestellt. Zur Gestaltung können Sie HTML-Elemente nutzen. Informationen dazu finden Sie unter https://wiki.selfhtml.org/.

Anzeigetemplate nachher
: Für den Zeitraum danach wird der hier eingegebene Text als Inhalt der Formluarseite eingestellt. Zur Gestaltung können Sie HTML-Elemente nutzen. Informationen dazu finden Sie unter https://wiki.selfhtml.org/.

Max. Anzahl bearbeiten
: Hier kann eingestellt werden, wie oft ein Datensatz in dieser Ansicht bearbeitet werden kann. Das ist sinnvoll z. B. bei Umfragen. Nach der Teilnahme an der Umfrage werden die Daten sofort weiterverarbeitet und sollen nicht mehr geändert werden dürfen.

Anzeigetemplate max. Anzahl überschritten
: Geben Sie hier den Text ein der angezeigt werden soll, wenn die max. Anzahl überschritten wurde. Zur Gestaltung können Sie HTML-Elemente nutzen. Informationen dazu finden Sie unter https://wiki.selfhtml.org/.

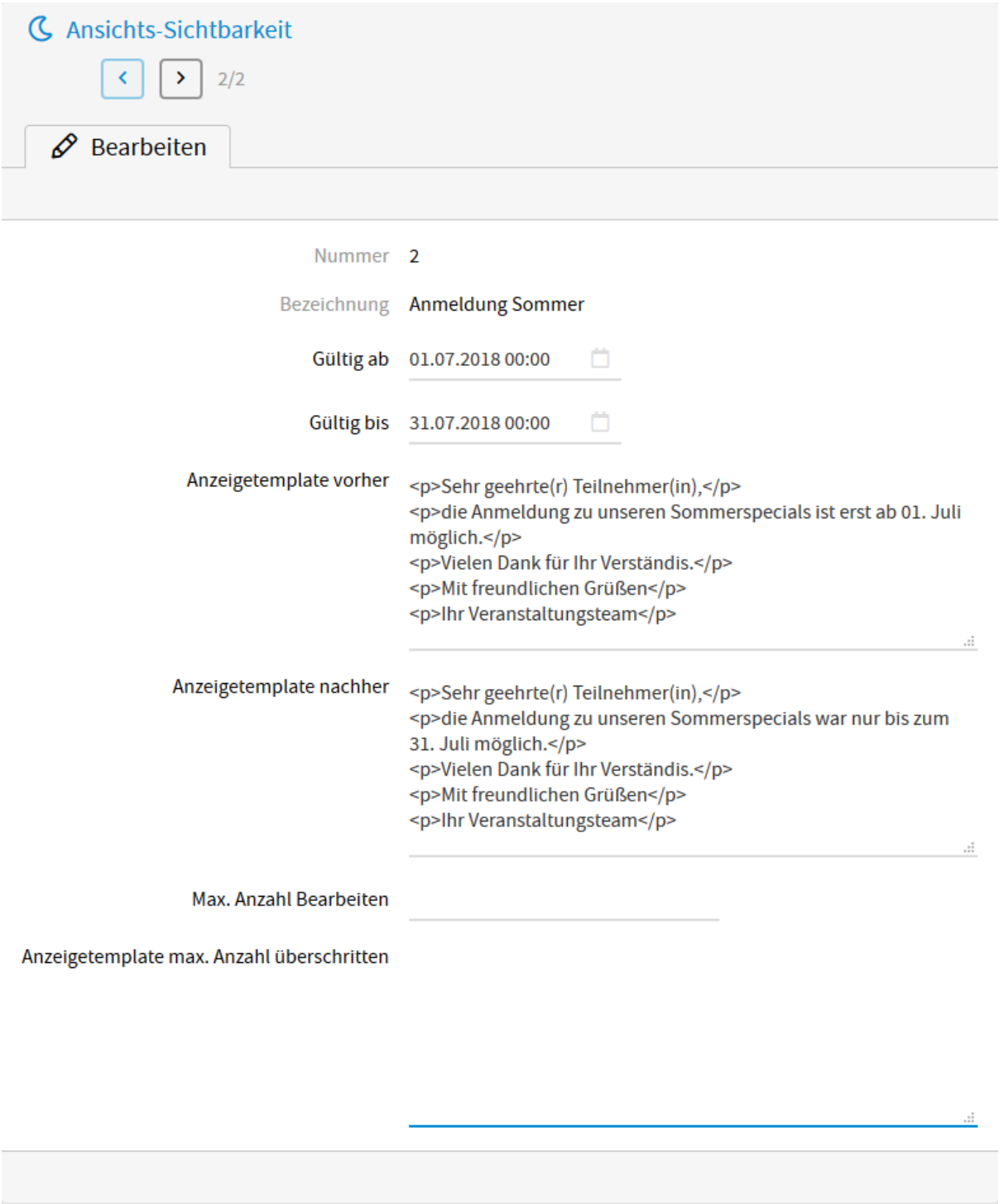


Abb. 101: Ansichts-Sichtbarkeit erstellen

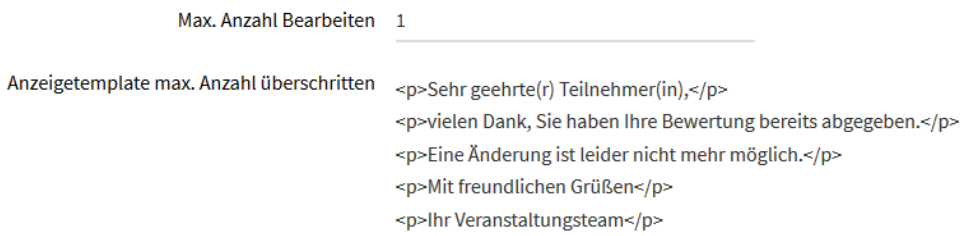


Abb. 102: Beispiel max. Anzahl

2.15 Daten-Auswertungen

Daten-Auswertungen ermöglicht es Ihnen, im Datenmanagement zusätzliche Auswertungen für Ihre Daten einzurichten.

Welche Felder in diesen Auswertungen angezeigt werden, lässt sich flexibel konfigurieren. Weiterhin ist es möglich, die Daten zu aggregieren (d.h. die Datensätze anhand bestimmter Kriterien zu gruppieren).

2.15.1 Erstellung von Daten-Auswertungen

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Daten-Auswertungen*. Klicken Sie auf *Hinzufügen* um eine neue Auswertung zu erstellen.

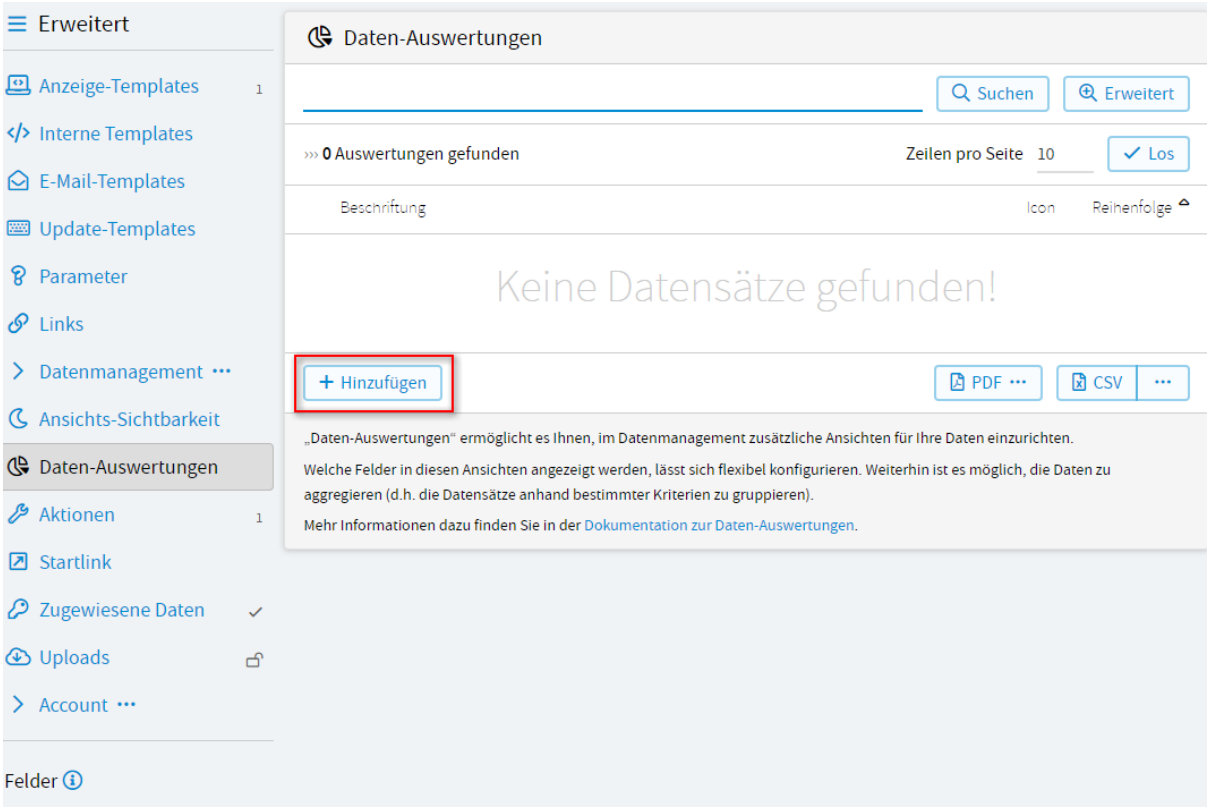


Abb. 103: Daten-Auswertung hinzufügen

Sie gelangen dann in die erste Einstellungsmaske. (Siehe *Daten-Auswertung erstellen Schritt 1*)

Beschriftung
: Unter dieser erscheint die Auswertung in der linken Navigation in der Datenliste.

Icon
: Hier können Sie ein Icon für die Auswertung auswählen. Verwendet werden können alle Namen aus dem FontAwesome-Icon-Set (http://fontawesome.io/)

Reihenfolge
: Geben Sie hier an, an welcher Stelle die Auswertung in der linken Navigation der Datenliste erscheint.

Anzahl in Navigation
: Durch das Setzen des Hakens wird neben dem Icon die Anzahl der Datensätze in dieser Auswertung angezeigt.

Wenn Sie diese Felder gefüllt haben, klicken Sie auf *Speichern*. Anschließend sehen Sie neben *Bearbeiten* die Tabs *Haupt-Felder* und *Filterbedingungen*. (Siehe *Daten-Auswertung erstellen Schritt 2*)

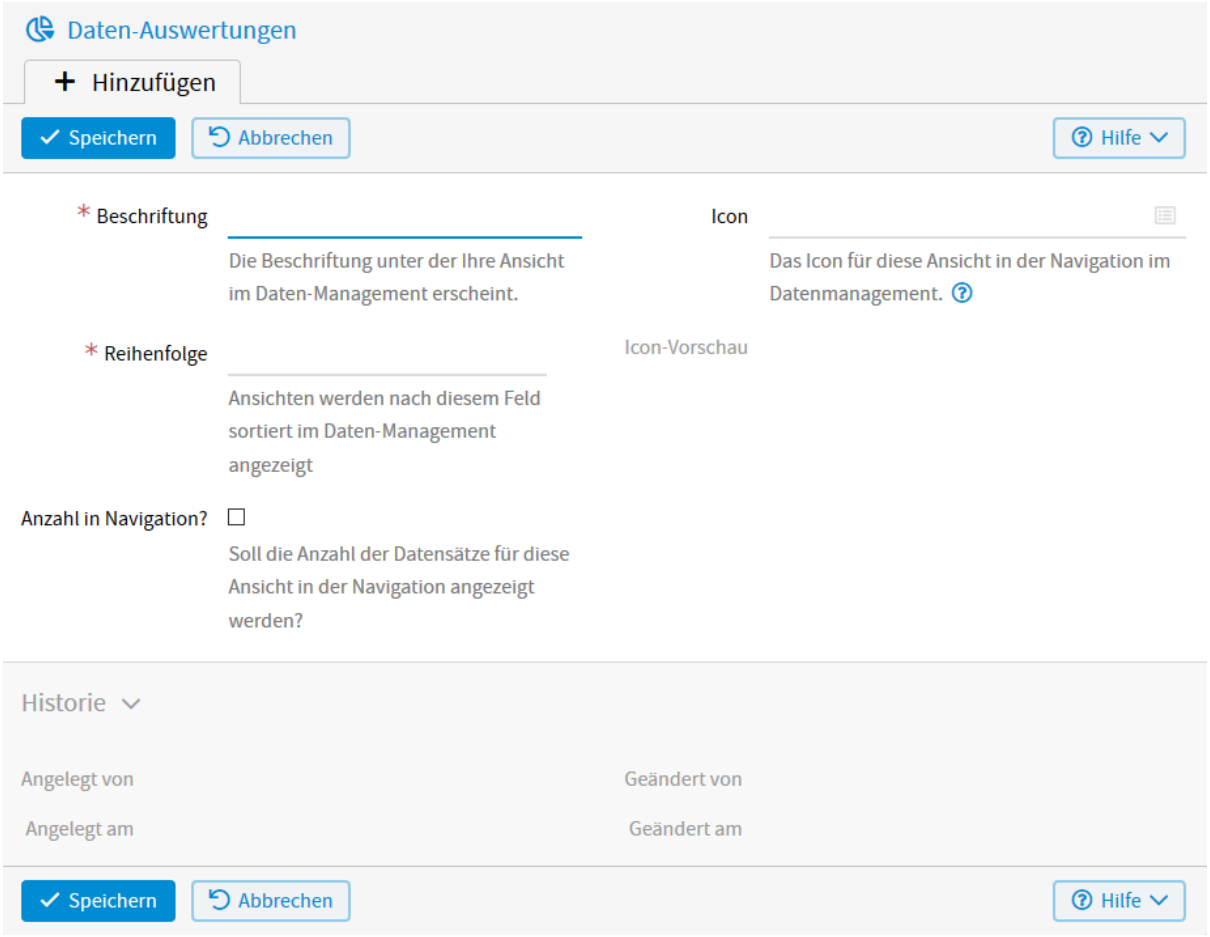


Abb. 104: Daten-Auswertung erstellen Schritt 1

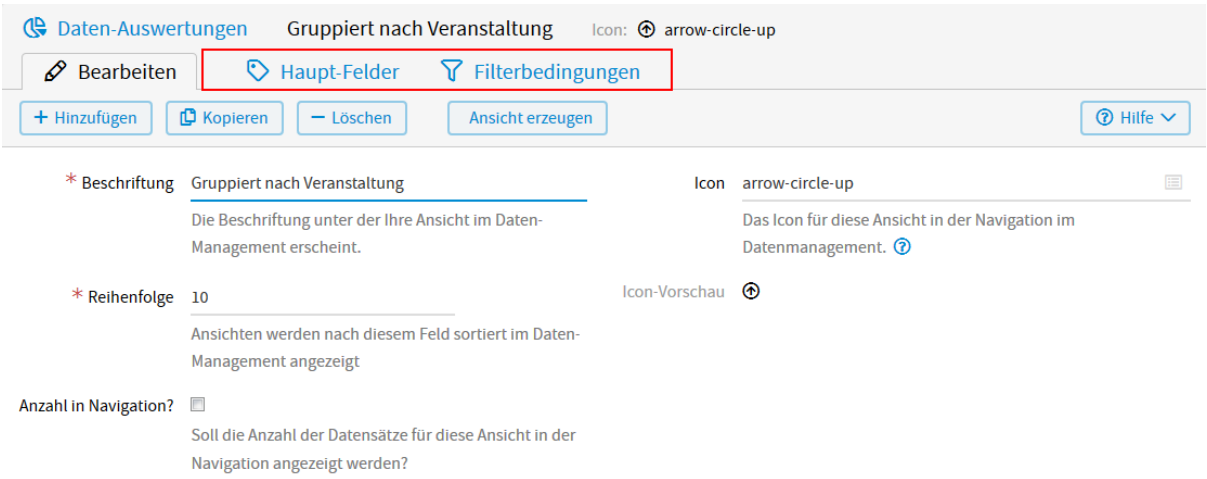


Abb. 105: Daten-Auswertung erstellen Schritt 2

Haupt-Felder

Klicken Sie zunächst auf *Haupt-Felder* und anschließend auf *Hinzufügen*, um festzulegen, welche Felder in die Auswertung aufgenommen werden sollen. (Siehe *Hauptfelder hinzufügen* und *Hauptfelder erstellen*) Die in dieser Auswertung konfigurierten Felder erscheinen in der neuen Auswertung als Spalten.

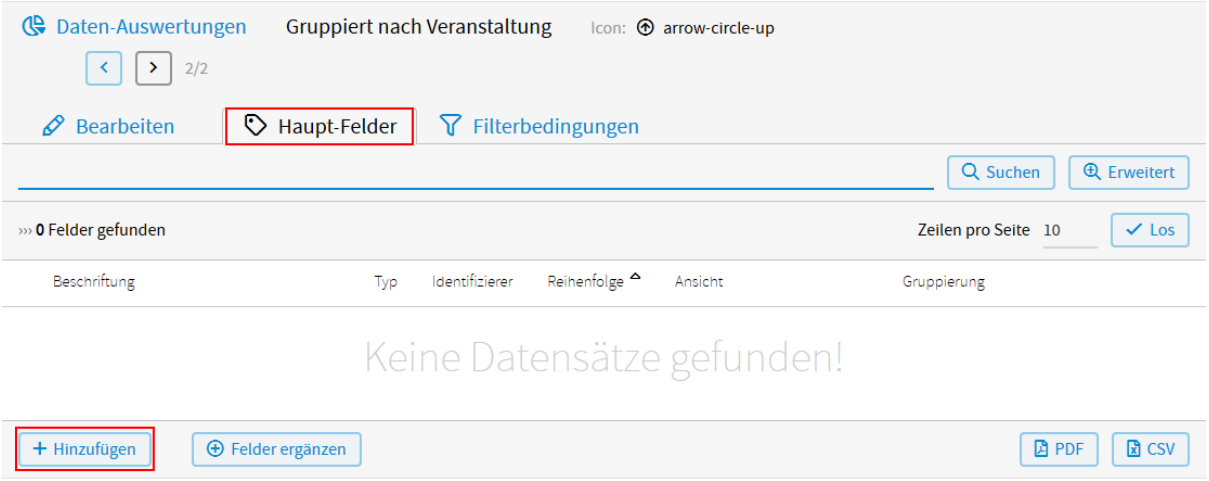


Abb. 106: Hauptfelder hinzufügen

Feld
: Wählen Sie hier das Feld aus, wonach ausgewertet werden soll.

Eigene Beschriftung
: Diese ersetzt die Beschriftung des Feldes in der Spalte der Auswertung.

Reihenfolge
: Legt bei mehreren Hauptfeldern die Reihenfolge in der Übersicht fest.

Ansicht
: Hier können alle Eigenschaften der Standard-Datenliste ausgewählt werden.

  Listenansicht
  : Das Feld soll in der Listenansicht der Auswertung mit angezeigt werden.

  Detailansicht
  : Das Feld soll in der Detailansicht der Auswertung mit angezeigt werden.

  Suche
  : Das Feld bekommt ein eigenes Suchfeld in der Listenansicht der Auswertung.

  Volltextsuche
  : Das Feld wird in die Volltextsuche der Listenansicht der Auswertung aufgenommen.

  Expertensuche
  : Das Feld wird in die erweiterte Suche der Listenansicht der Auswertung aufgenommen.

  Pfad
  : Das Feld wird im Pfad immer angezeigt, wenn aus der Listenansicht heraus Untermasken ausgewählt werden. Zum Beispiel: *Anzeige* oder *Einzeldatensätze*

  Panel
  : Das Feld wird im Pfad immer angezeigt, wenn aus der Listenansicht heraus Untermasken ausgewählt werden. Zum Beispiel: *Anzeige* oder *Einzeldatensätze*. Das Feld steht dann rechts vom Pfad.

  Soll es in Ihrer Auswertung keine Suche geben, oder kein Pfad angezeigt werden, können die entsprechenden Häkchen weggelassen werden.

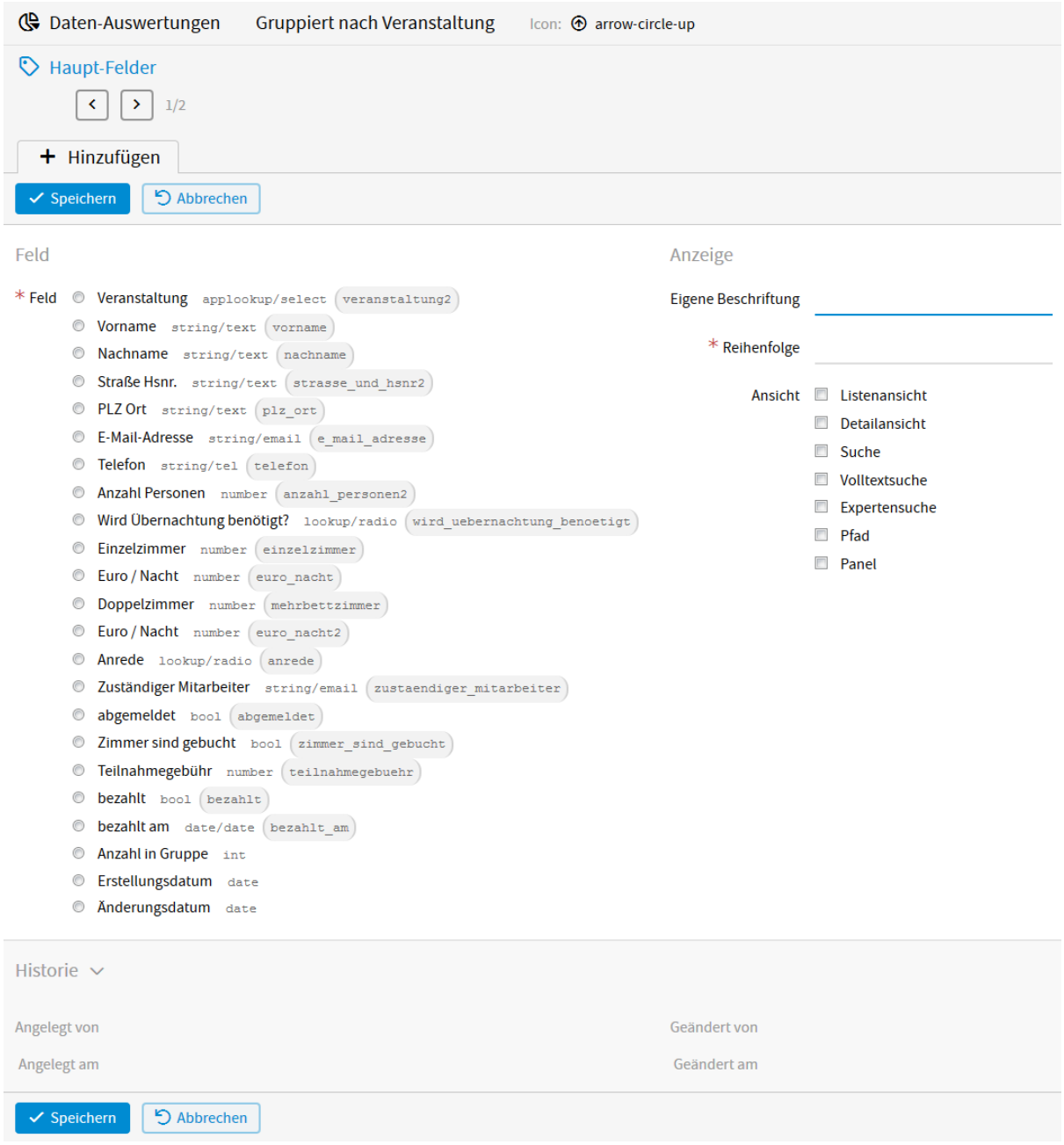


Abb. 107: Hauptfelder erstellen

Gruppierung
Sobald ein Feld ausgewählt wurde, erscheint noch die Auswahl *Gruppierung*. Hier kann eingestellt werden, ob und wie das Feld gruppiert werden soll. Die Auswahl ist abhängig vom Feldtyp.

Gruppiert
Einfaches Gruppieren; mehrfach vorkommende Einträge werden zusammengefasst.

Gruppiert nach Klang
Einträge, die gleich klingen, werden zusammengefasst (z. B. Mayer und Meier). Sinnvoll für Adressdoubletten-Suche.

Gruppiert nach Beginn
Einträge, bei denen die ersten x Zeichen übereinstimmen, werden zusammengefasst. Z. B. Postleitzahlengebiet bei einer PLZ.

Gruppiert nach Ende
Einträge, bei denen die letzten x Zeichen übereinstimmen, werden zusammengefasst. Z. B. die letzten Zeichen einer E-Mail-Adresse.

Gruppiert nach Tag
Einträge, die vom gleichen Tag sind, werden zusammengefasst. Betrifft nur Datumsfelder.

Gruppiert nach Monat
Einträge, die vom gleichen Monat sind, werden zusammengefasst. Betrifft nur Datumsfelder.

Gruppiert nach Jahr
Einträge, die vom gleichen Jahr sind, werden zusammengefasst. Betrifft nur Datumsfelder.

Minimum
Zeigt den kleinsten Wert der betroffenen Datensätze an. Betrifft nur Zahlenfelder.

Maximum
Zeigt den größten Wert der betroffenen Datensätze an. Betrifft nur Zahlenfelder.

Durchschnitt
Zeigt den Durchschnitt der betroffenen Datensätze an. Betrifft nur Zahlenfelder.

Summe
Zeigt die Summe der betroffenen Datensätze an. Betrifft nur Zahlenfelder.

Nach Anklicken von *Gruppiert nach Beginn* oder *Gruppiert nach Ende* kann darunter noch ein entsprechender Wert eingegeben werden. Nach Auswahl der Felder klicken Sie auf *Speichern* und anschließend wieder auf *Haupt-Felder*. Sie können nun zusätzliche Haupt-Felder oder, wenn nötig, *Filterbedingungen hinzufügen*. Für letzteres klicken Sie auf *Filterbedingungen* und anschließend auf *Hinzufügen*.

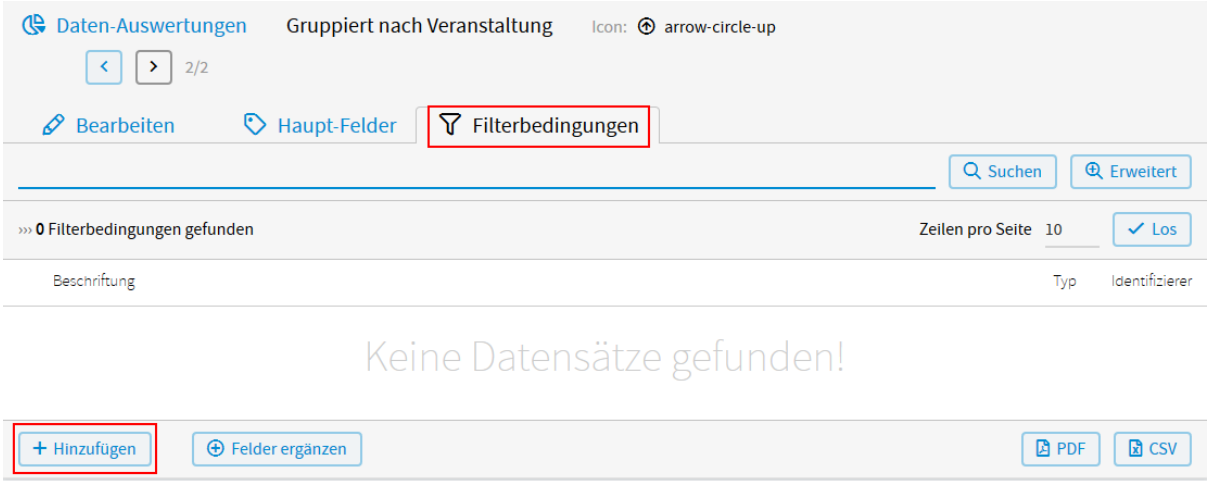


Abb. 108: Filterbedingungen hinzufügen

Filterbedingungen

Mit den *Filterbedingungen* kann eingeschränkt werden, welche Datensätze in der Auswertung angezeigt werden.

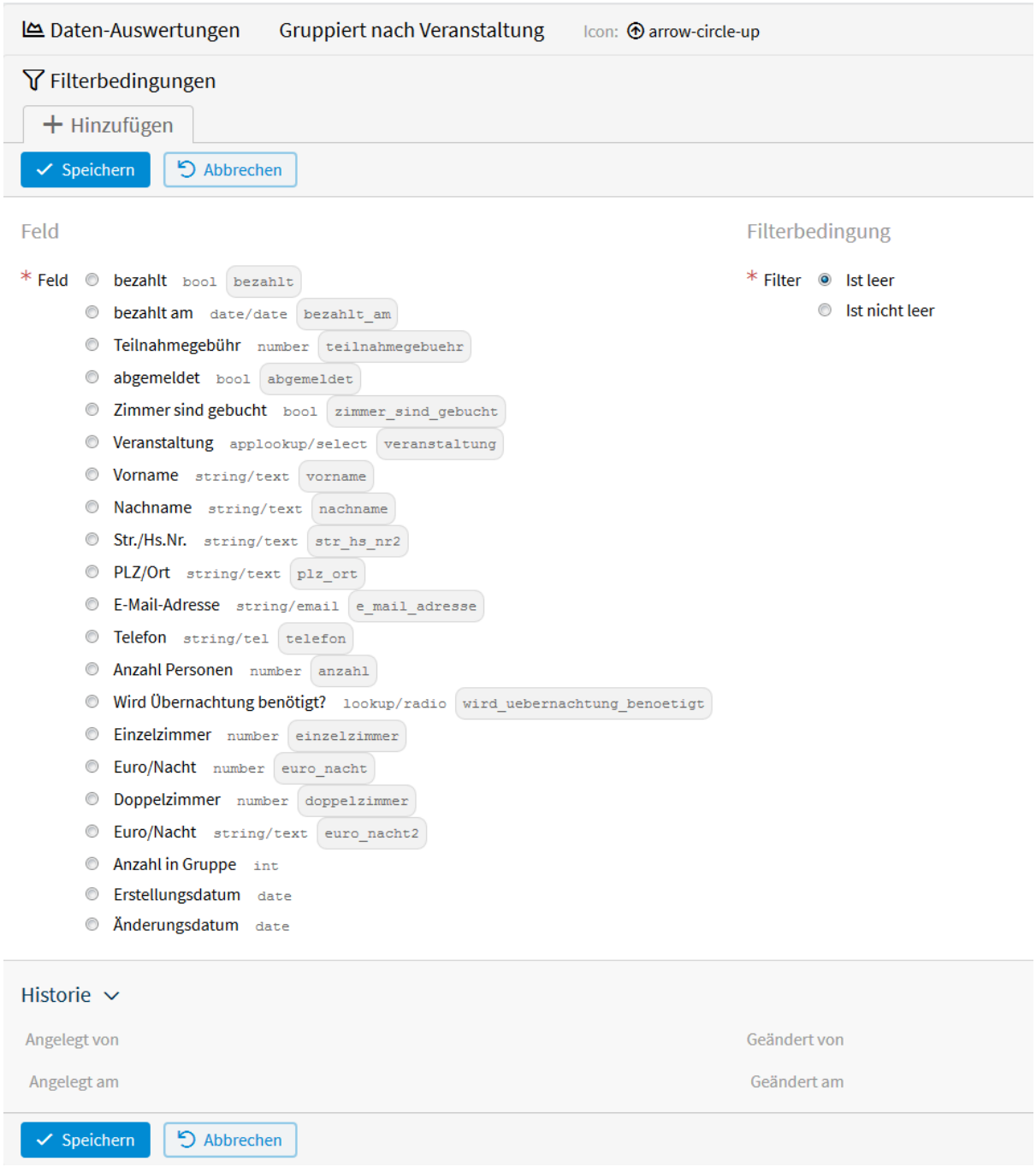


Abb. 109: Filterbedingungen erstellen

Feld
: Geben Sie an, nach welchem Feld gefiltert werden soll.

Filter
: Geben Sie an, welchen Filter Sie nutzen wollen. Die Auswahl ist abhängig vom Feldtyp.

Wert
: Wird zusammen mit dem Filter als Filterbedingung genutzt.

Sind alle gewünschten Einstellungen getroffen und gespeichert, muss im Tab *Bearbeiten* der Button *Ansicht erzeu-*

gen gedrückt werden.

WICHTIG: Nach jeder Änderung muss die Ansicht immer wieder neu erzeugt werden.

Wenn beim Aufrufen der Auswertung ein „Unvorhersehbarer Fehler“ erscheint, ist eine Einstellung ungültig.

2.15.2 Anwendungsbeispiel

Um die Funktionsweise der Sortierung zu veranschaulichen, wird im Folgenden auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen.

Es soll eine Daten-Auswertung erstellt werden, in der nach Veranstaltung gruppiert, die jeweilige Anzahl der Anmeldungen angezeigt wird.

Wählen Sie dafür im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Daten-Auswertungen*. Klicken Sie auf *Hinzufügen* um eine neue Auswertung zu erstellen.

Im *Schritt 1* werden hier folgende Konfigurationen vorgenommen.

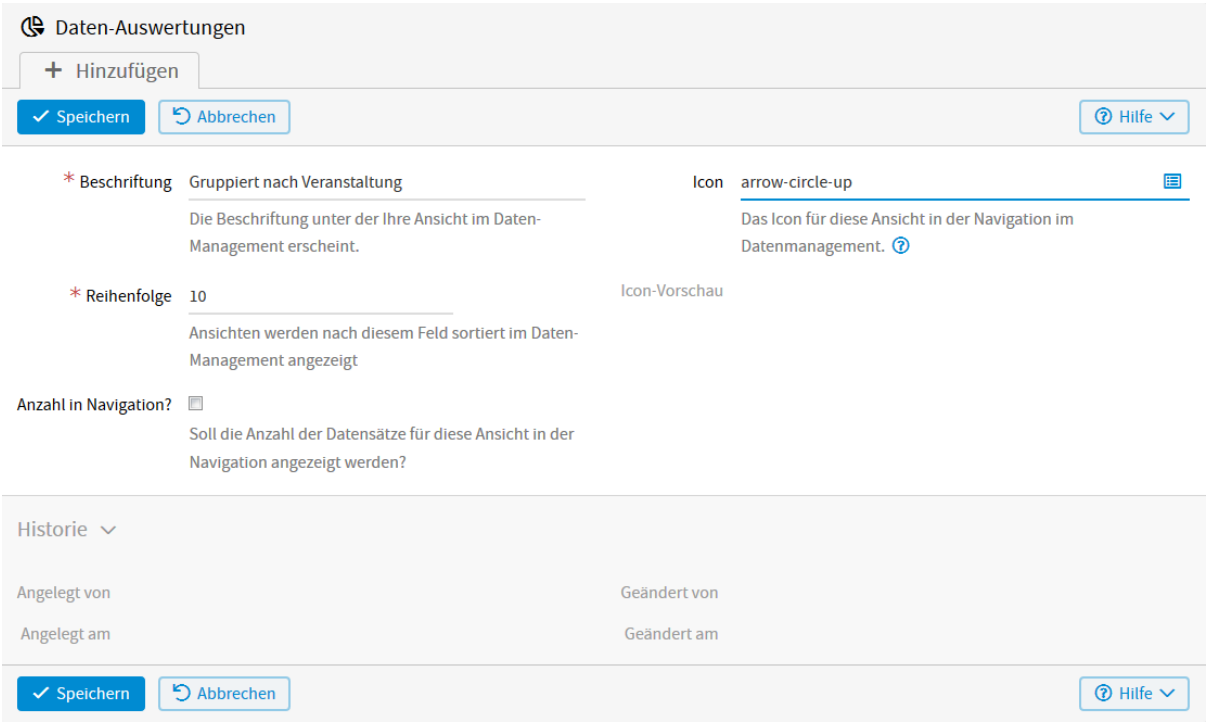


Abb. 110: Anwendungsbeispiel - Schritt 1

Nach dem Speichern klicken Sie auf *Hauptfelder* und *Hinzufügen* um das *1. Hauptfeld* anzulegen.

Da nach Veranstaltung gruppiert werden soll, wird zunächst bei *Feld Veranstaltung* ausgewählt. Bei *Eigene Beschriftung* wird nichts eingegeben, da der Feldname `Veranstaltung` passend ist. Die *Reihenfolge* soll `10` lauten und bei *Ansicht* werden alle Häkchen gesetzt, bis auf *Panel*. Bei *Gruppierung* wird *gruppiert* ausgewählt.

Nach dem Speichern kann durch Klicken auf *Hinzufügen* das *2. Hauptfeld erstellt* werden.

Hier soll die Anzahl der Anmeldungen zu den jeweiligen Veranstaltungen angezeigt werden. Dafür muss bei *Feld Anzahl Personen* ausgewählt werden. Bei *Eigene Beschriftung* wird `Anmeldungen` eingegeben. Das ist dann der Name der Spalte in der Auswertung. Die *Reihenfolge* soll `20` sein und bei *Ansicht* werden alle Häkchen gesetzt, bis auf *Pfad*. Bei *Gruppierung* wird *Summe* gewählt. Zum Schluss müssen alle Eingaben gespeichert werden. *Filterbedingungen* werden in diesem Beispiel nicht benötigt.

Sind alle Hauptfelder erstellt und gespeichert, muss im Tab *Bearbeiten* der Button *Ansicht erzeugen* angeklickt werden.

WICHTIG: Nach jeder Änderung der Daten-Auswertung muss die Ansicht immer wieder neu erzeugt werden.

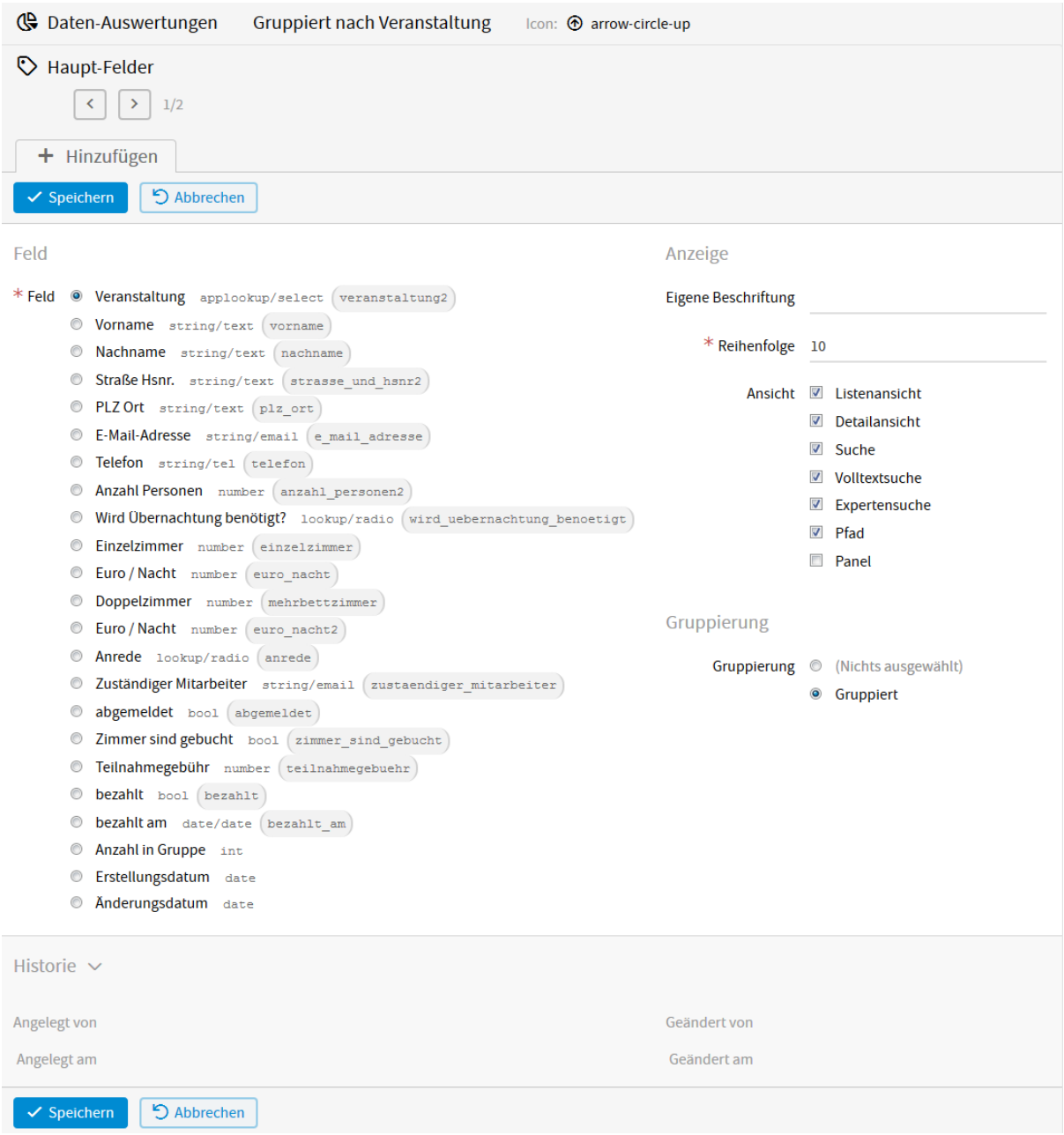


Abb. 111: Hauptfeld 1 erstellen

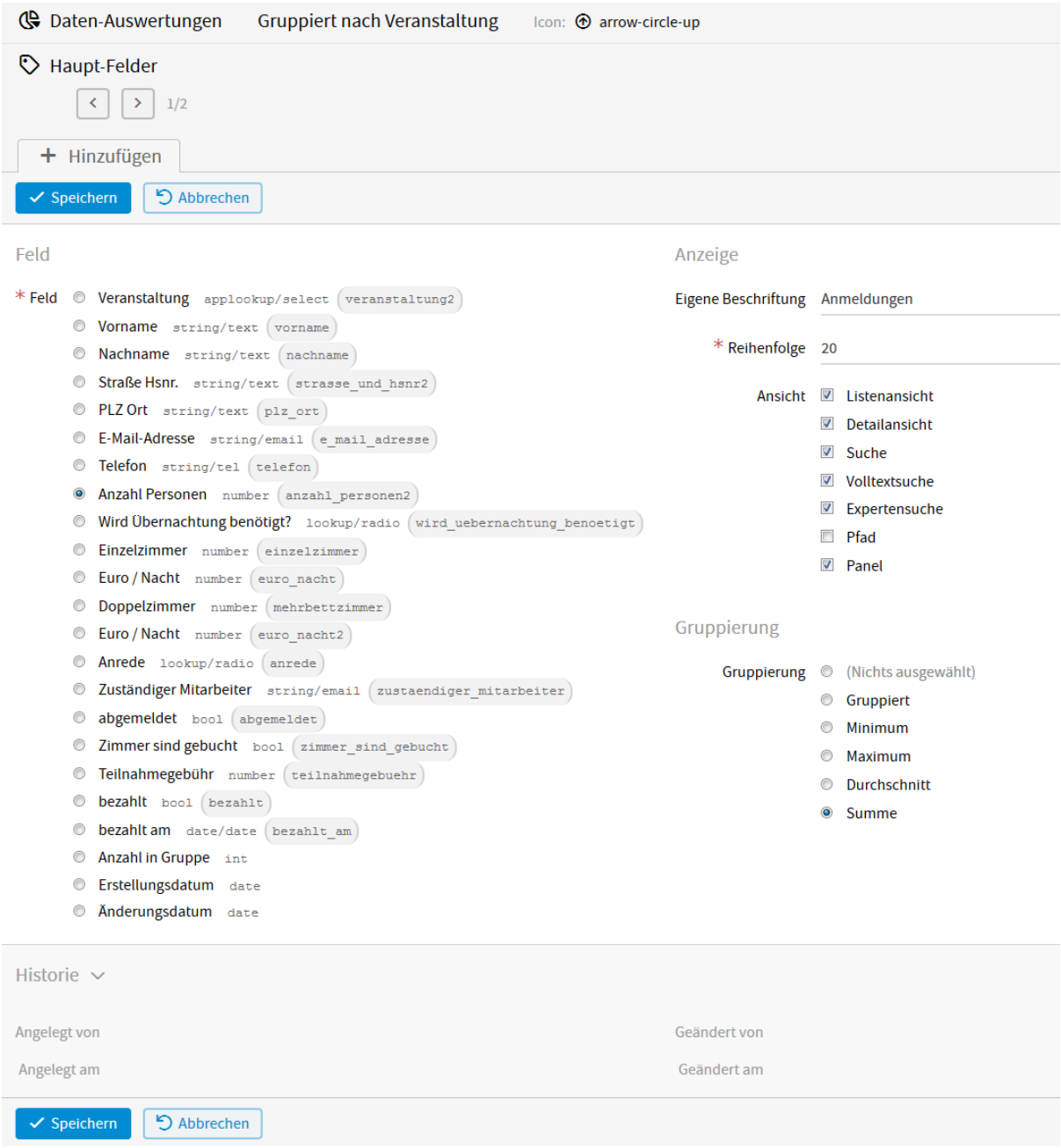


Abb. 112: Hauptfeld 2 erstellen

Das *Ergebnis der Daten-Auswertung* kann jetzt unter *Daten* → *Liste* im Datenmanagement aufgerufen werden.

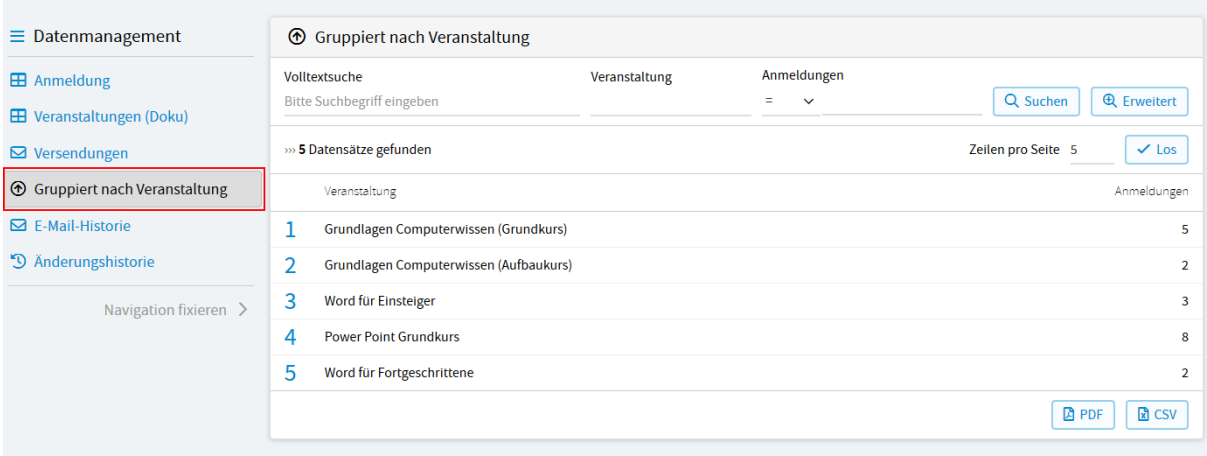


Abb. 113: Ergebnis der Daten-Auswertung

Durch Klicken auf einen Datensatz der Auswertung, erscheinen die einzelnen Datensätze, die hinter dieser Gruppierung stehen.

2.16 Aktionen

Aktionen ermöglicht es Ihnen, Aktionen zu definieren, die bestimmte Datensätze ändern, neue Datensätze anlegen, Arbeitsaufgaben anlegen, oder E-Mails verschicken.

Die Aktionen können entweder automatisch (z.B. wenn ein neuer Datensatz vom Benutzer angelegt oder geändert wird) oder vom Benutzer (z. B. durch Anklicken einer URL in einer E-Mail oder eines Buttons im Datenmanagement) ausgelöst werden.

2.16.1 Erstellung von Aktionen

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine *neue Aktion* zu erstellen.

Im nun geöffneten Fenster können folgende *Konfigurationen* vorgenommen werden.

Identifizierer
: Die eindeutige Kennung für die jeweilige Aktion. (Mit diesem wird die Aktion z. B. bei der Verwendung als Link für E-Mails angesprochen.)

Beschriftung
: Der Name der Aktion, der z. B. in der E-Mail oder in der Datenliste sichtbar ist.

Aktiv?
: Solange hier kein Häkchen gesetzt wird, kann die Aktion nicht ausgeführt werden.

Reihenfolge
: Die Aktionsbuttons werden im Datenmanagement nach dieser Zahl sortiert angezeigt.

Berechtigung für
: Hier muss ausgewählt werden, wer diese Aktion ausführen darf.

Beschreibung
: Eine Beschreibung der Funktion, oder eine Beschreibung, um mehrere Aktionen voneinander zu unterscheiden.

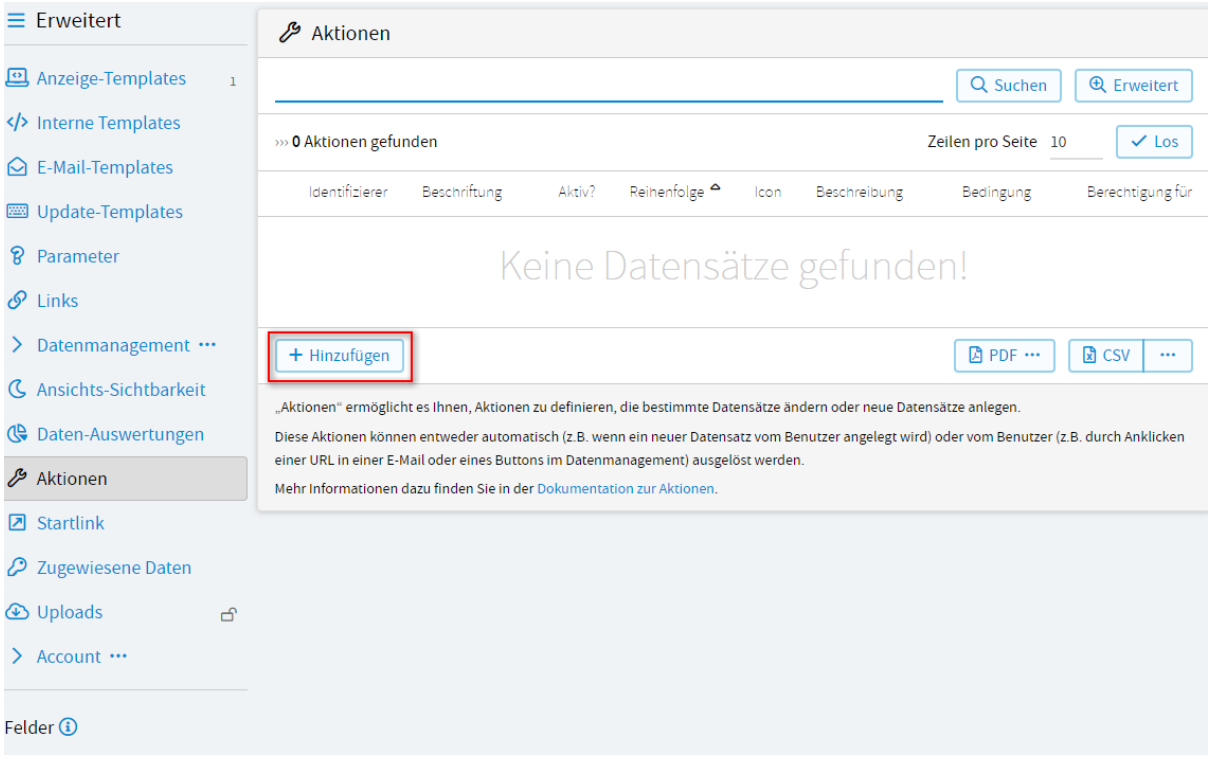


Abb. 114: Aktionen hinzufügen

Icon
: Hier können Sie ein Icon für die Aktion auswählen. Verwendet werden können alle Namen aus dem FontAwesome-Icon-Set (http://fontawesome.io/).

Verwendung
: Legt fest, wo die Aktion zur Verfügung steht.

  Aktions-Button im Datenmanagement für mehrere Datensätze
  : Die Aktion kann in der Datenliste für mehrere ausgewählte Datensätze ausgeführt werden.

  Aktions-Button im Datenmanagement für einzelnen Datensatz
  : Die Aktion kann in der Datenliste für einen einzelnen ausgewählten Datensatz ausgeführt werden.

  Link für E-Mails
  : Die Aktion wird durch Anklicken des Links in einer E-Mail ausgeführt. (*Siehe Anwendungsbeispiel*). Wie Sie einen Link einbinden, können Sie in der Dokumentation der *E-Mail-Templates* nachlesen. Außerdem können Sie auf den Link zugreifen, wenn Sie von der Datenliste aus E-Mails verschicken (sofern ein E-Mail-Feld vorhanden ist). Klicken Sie dafür in der geöffneten E-Mail-Maske auf *Daten einfügen* und wählen den entsprechenden Aktionslink aus.

  Vor dem Anzeigen des Bearbeiten-Formulars ausführen
  : Die Aktion wird beim Aufruf des edit-Formulars ausgeführt. Auf das jeweilige edit-Formular gelangt man über den Bearbeiten-Button des des jeweiligen Datensatzes in der Datenliste oder indem man nach dem Absenden des Formulars `/edit` vor die View-ID `?view=58b3f8b35699ce805c43dc6d` in der URL einfügt. Somit lässt sich z. B. eine Aufrufbestätigung einrichten.

  Nach Änderung des Datensatzes ausführen (auch bei Import)
  : Die Aktion wird beim Speichern, nach dem Ändern eines bestehenden Datensatzes ausgeführt.

  Nach Anlegen des Datensatzes ausführen (auch bei Import)
  : Die Aktion wird beim Speichern eines neuen Datensatzes ausgeführt.

Meldung
: Diese Meldung erscheint nachdem die Aktion ausgeführt wurde.

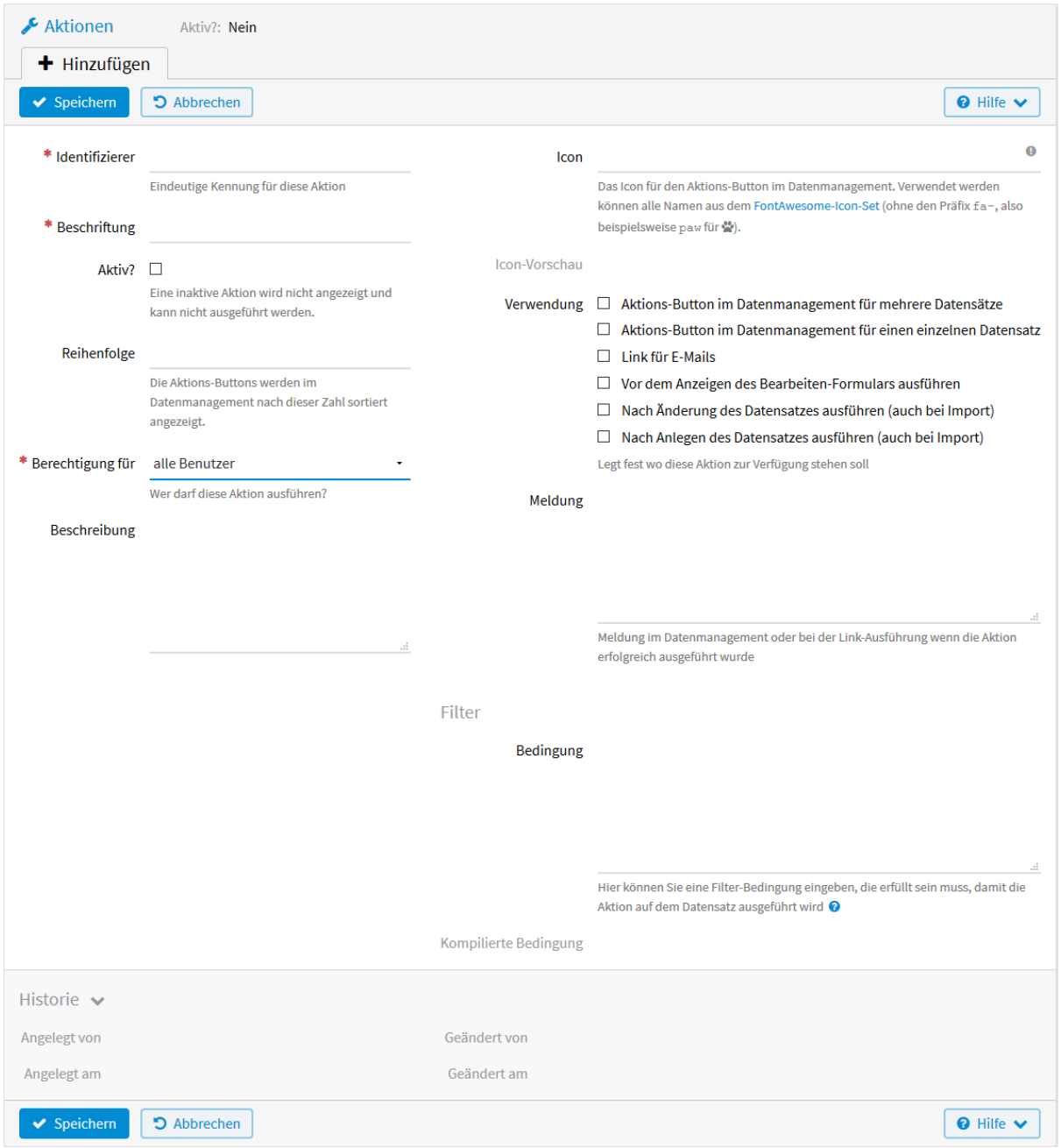


Abb. 115: Aktionen erstellen

Bedingung
Durch eine Filter-Bedingung kann eingeschränkt werden, auf welche Datensätze die Aktion ausgeführt wird.

Formulieren der Bedingung

Für die Formulierung wird *vSQL* verwendet.

Um Werte von Feldern Ihres Formulars zu prüfen, werden die *Identifizierer der Felder* benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*. Auf die Werte kann dann zugegriffen werden mit: `r.v_identifier`.

Sie haben unterschiedliche Prüfmöglichkeiten. Die Datentypen, Operationen, Attribute und Methoden, die Ihnen hier zur Verfügung stehen, finden Sie im Kapitel *vSQL* bzw. wenn Sie das `?` unterhalb des Feldes anklicken.

Um beispielsweise ein Text-Feld mit dem Identifizierer `name` auf einen bestimmten Wert zu prüfen, benötigen Sie folgende Bedingung: `r.v_name == "LivingApps"`.

Ist die Bedingung nicht vom Typ `BOOL` (d.h. liefert `True` oder `False`), wird sie mittels der vSQL-Funktion `bool` konvertiert.

Sobald Sie auf *Speichern* gedrückt haben, erscheint unterhalb des Bedingungsfelds die kompilierte Bedingung und, wenn beim Kompilieren ein Fehler aufgetreten ist, auch dieser Fehler.

Anweisungen

Nach dem Speichern der ersten Maske erscheint neben *Bearbeiten* der Tab *Anweisungen*. Fügen Sie eine neue Anweisung hinzu, um einzustellen, was durch die Aktion ausgeführt werden soll. Siehe *Anweisung hinzufügen* und *Anweisung erstellen*.

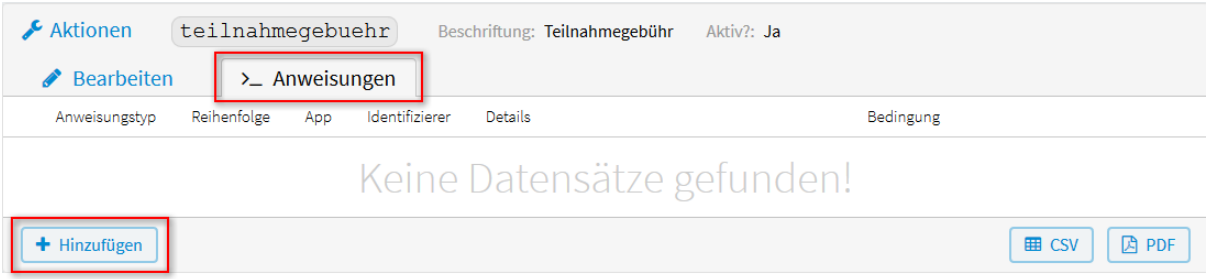


Abb. 116: Anweisungen hinzufügen

Anweisungstyp
Legt die Art der Aktion fest.

Datensatz anlegen
Ein neuer Datensatz wird angelegt, auf den die Konfigurationen angewendet werden.

Datensatz verändern
Der Datensatz, auf dem die Aktion ausgeführt wird, wird entsprechend der Konfigurationen verändert (*siehe Anwendungsbeispiel*).

Arbeitsaufgabe automatisch anlegen
Diese Funktionalität sollte nicht ohne gesonderte Einweisung verwendet werden.

Datensatz löschen
Der Datensatz, auf dem die Aktion ausgeführt wird, wird gelöscht (*siehe Anwendungsbeispiel*).

Datensatz über Formular anlegen
Auf einem bestehenden Datensatz können Sie hiermit ein Eingabeformular derselben oder einer anderen App öffnen. Die Werte des Eingabeformulars können Sie vorbelegen mit festen Werten, oder Ausdrücken, die sich auf den bestehenden Datensatz beziehen. Wird das Eingabeformular dann abgesendet, wird der neue Datensatz angelegt. (*siehe Anwendungsbeispiel*)

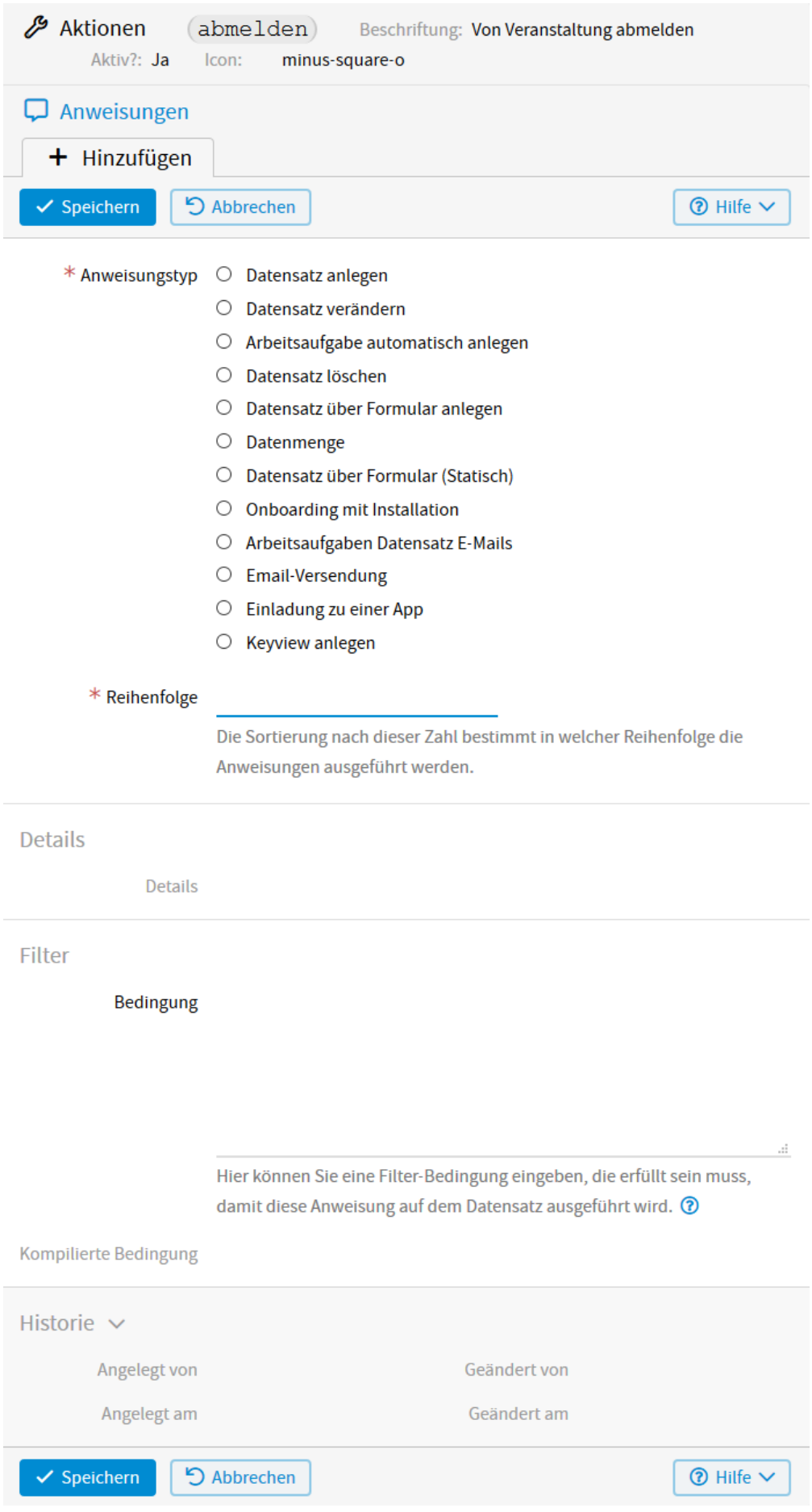


Abb. 117: Anweisungen erstellen

Datenmenge
Diese Funktionalität sollte nicht ohne gesonderte Einweisung verwendet werden.

Datensatz über Formular (Statisch)
Diese Funktionalität wird bei bestimmten Installationen automatisch angelegt. Man sollte sie nicht manuell anlegen.

Onboarding mit Installation
Diese Funktionalität sollte nicht ohne gesonderte Einweisung verwendet werden.

Arbeitsaufgaben Datensatz E-Mails
Diese Aktion legt eine Arbeitsaufgabe für eine oder mehrere Personen an, die auch als Link per E-Mail an die eingegebenen Adressen verschickt werden kann (*siehe Anwendungsbeispiel*).

Email-Versendung
Mit dieser Aktion kann man E-Mails verschicken (*siehe Anwendungsbeispiel*).

Einladung zu einer App
Mit dieser Aktion können Sie Personen zur Mitarbeit an einer oder mehreren Ihrer Apps einladen. (*siehe Anwendungsbeispiel*).

Keyview anlegen
Diese Funktionalität sollte nicht ohne gesonderte Einweisung verwendet werden.

Reihenfolge
Bei mehreren Anweisungen entscheidet die hier festgelegte Reihenfolge über die Abfolge der Ausführung.

Bedingung
Hier kann eine Filter-Bedingung eingegeben werden, die erfüllt sein muss, damit diese Anweisung auf dem Datensatz ausgeführt wird. Hierzu können Sie im Kapitel *Formulieren der Bedingung* mehr lesen.

Bei Auswahl der Felder *Datensatz anlegen* oder *Datensatz über Formular anlegen* erscheinen zwei weitere Konfigurationsfelder in dieser Maske:

App
Hier können Sie die App auswählen, in der der neue Datensatz erstellt werden soll. Wenn nichts ausgewählt wird, ist es die aktuelle App.

Identifizierer
Hier können Sie festlegen, unter welchem Identifizierer der durch diese Aktion gelieferte Datensatz für die untergeordneten Anweisungen zugänglich sein soll.

Felderbelegung

Nach dem Speichern der Anweisungen erscheint neben *Bearbeiten* der Tab *Felderbelegung*. Durch Klicken darauf sehen Sie alle Felder, die durch die Aktion verändert werden können. Ist es die aktuelle App, dann sehen Sie deren *Felder*. Wenn z. B. ein Datensatz in einer anderenApp angelegt werden soll, sehen Sie die Felder der anderen App.

Durch Klicken auf das entsprechende Feld, kann der *Inhalt konfiguriert* werden.

Je nachdem, um welche Aktion es sich handelt, können verschiedene Einstellungen vorgenommen werden.

Anzeige
Diese Konfiguration erscheint nur wenn *Datensatz über Formular anlegen* gewählt wurde.

Bearbeiten
Der Benutzer kann den unter Aktion generierten Wert noch verändern.

Anzeigen
Der Benutzer kann diesen Wert nicht verändern. Er wird unverändert übernommen.

Verstecken
Der Wert wird unverändert übernommen, ohne angezeigt zu werden.

Aktion

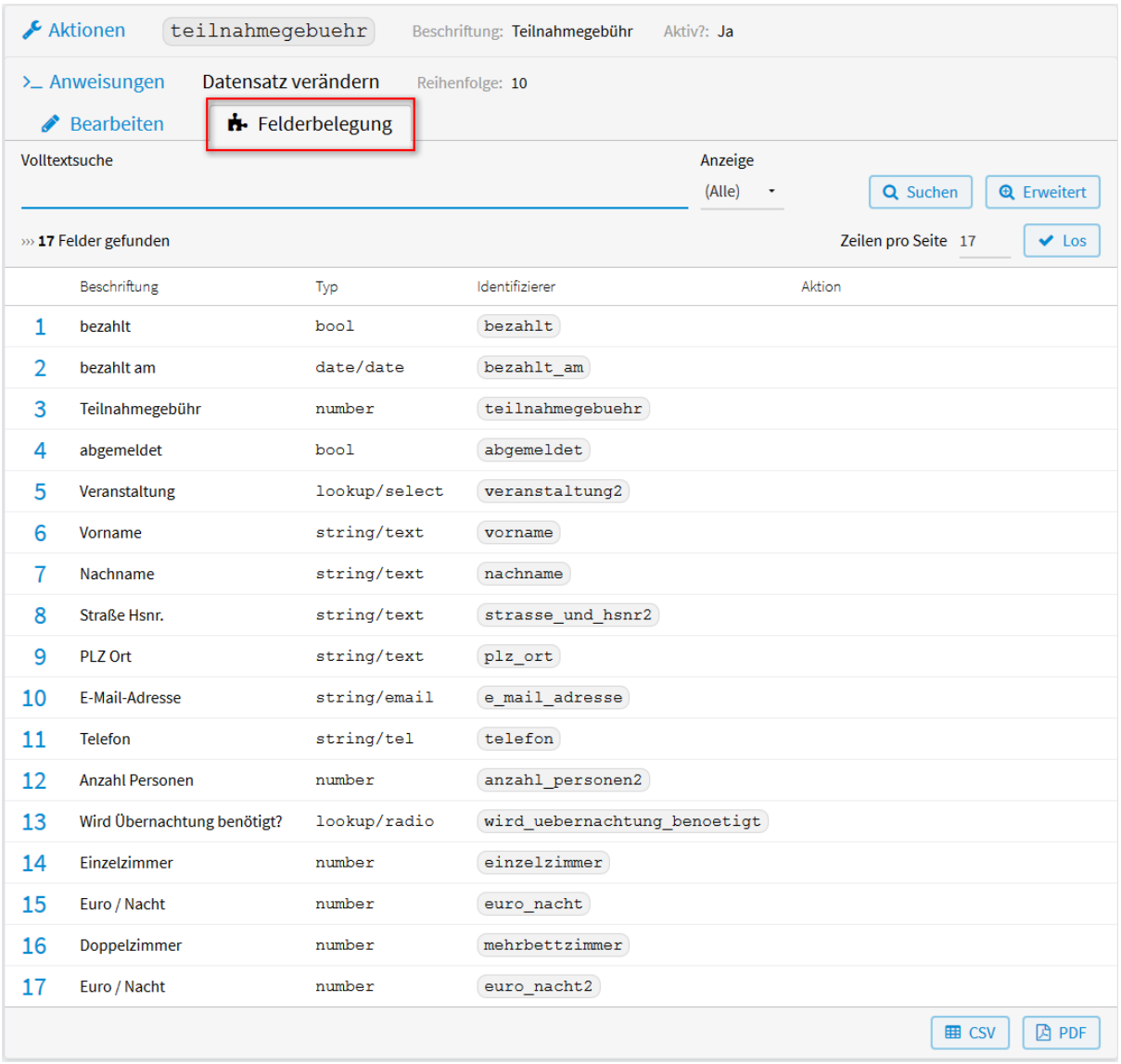


Abb. 118: Felderbelegung

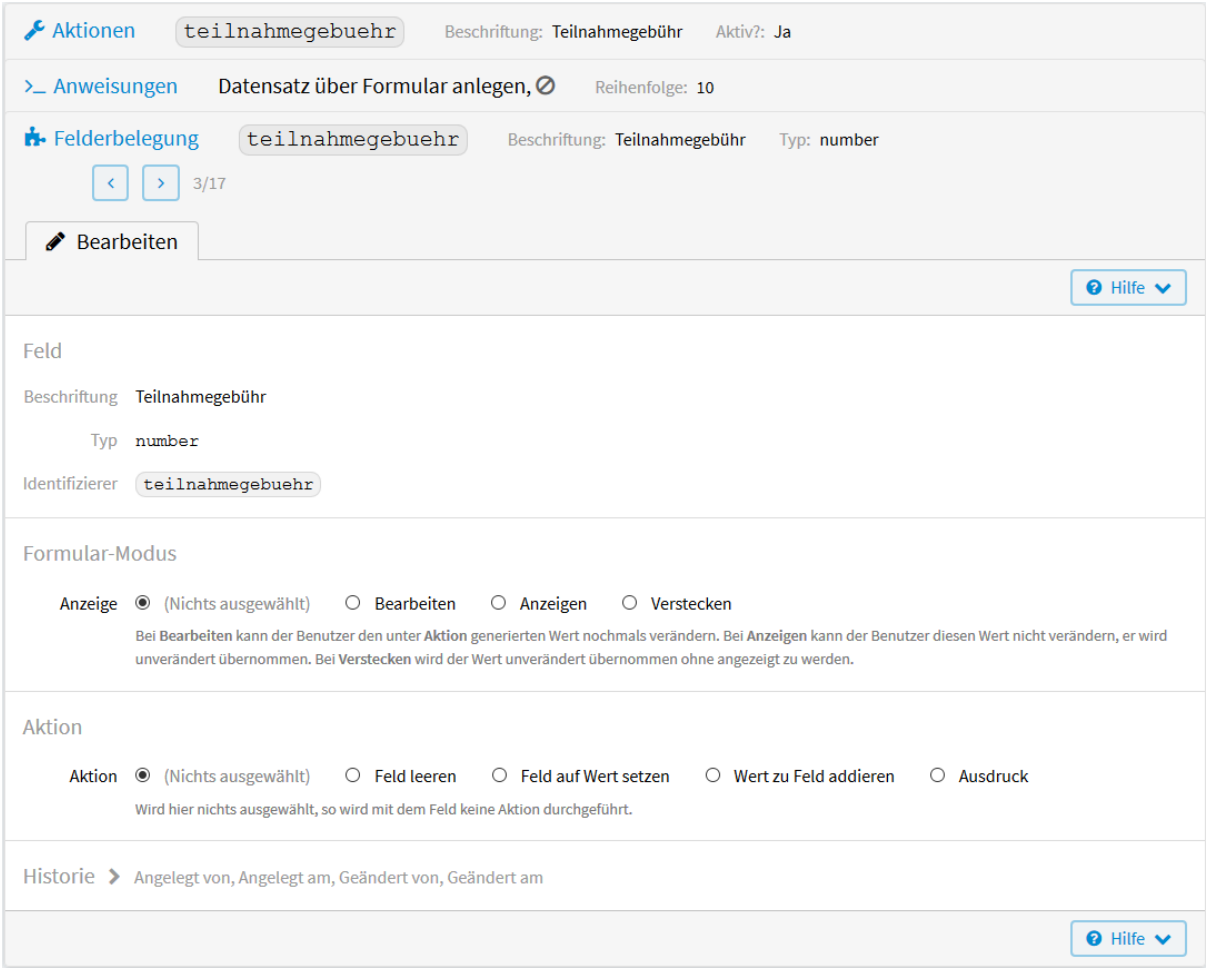


Abb. 119: Konfiguration Felder

Nichts ausgewählt
: Wird hier nichts ausgewählt, so wird mit dem Feld keine Aktion durchgeführt.

Feld leeren
: Der Feldinhalt wird gelöscht. Das ist nur beim Ändern von Datensätzen sinnvoll.

Feld auf aktuelles Datum setzen
: Diese Auswahl erscheint nur, wenn das ausgewählte Feld vom Typ `date`, `datetimeminute` oder `datetimesecond` ist.

Feld auf Wert setzen
: Das Feld übernimmt, nachdem die Aktion ausgeführt wurde, den hier eingegebenen Wert.

Wert zu Feld addieren
: Diese Auswahl erscheint nur, wenn das ausgewählte Feld vom Typ `number` ist. Der hier eingegebene Wert wird zum Wert des Feldes addiert.

Ausdruck
: Sollen die Werte des Feldes nicht gleich, sondern individuell sein, kann hier ein Ausdruck eingegeben werden. Auf die Werte kann dann zugegriffen werden mit: `r.v_identifier`. Für die Formulierung des Ausdrucks wird *vSQL* verwendet. Dafür werden die *Identifizierer der Felder* benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*.

Nach dem Speichern erscheint unterhalb des Feldes *Ausdruck* der kompilierte Ausdruck. Ist ein Fehler in der Formulierung, sehen Sie diesen hier.

2.16.2 Anwendungsbeispiele

Um die Funktionsweise der Aktionen zu veranschaulichen, wird im Folgenden auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen. Es werden sieben verschiedene Aktionen vorgestellt.

Datensatz verändern

Hierbei soll das Feld `Zuständiger Mitarbeiter`, nach dem Speichern einer neuen Anmeldung, automatisch mit der E-Mail-Adresse des zuständigen Mitarbeiters aus der App „Veranstaltungen“ gefüllt werden.

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine neue Aktion zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Nach dem Speichern erscheint neben *Bearbeiten* der Tab *Anweisungen*. Nach *Hinzufügen* einer neuen Anweisung kann im nun geöffneten Fenster eingestellt werden, was durch die Aktion ausgeführt werden soll. Siehe *Anweisungen*.

Nach dem Speichern der Anweisungen erscheint neben *Bearbeiten* der Tab *Felderbelegung*. Durch Klicken darauf sehen Sie alle *Felder der App*, deren Datensätze durch die Aktion verändert werden können. Durch Klicken auf das zu ändernde Feld *Zuständiger Mitarbeiter*, kann dessen *Inhalt konfiguriert* werden.

Im nun geöffneten Fenster wird bei *Aktion Ausdruck* ausgewählt, da kein fester Wert eingetragen sondern auf die Datensätze der App zugegriffen werden soll.

Mit dem *Ausdruck* `r.v_veranstaltung2.v_e_mail_verteiler` wird über das Feld mit Identifizierer `veranstaltung2` auf den Wert des Feldes mit dem Identifizierer `e_mail_verteiler` der verknüpften App `Veranstaltungen` zugegriffen. Die *Feldinformationen und Identifizierer der Felder* können Sie durch klicken auf *Felder* in der Menüleiste unten links, einsehen. Mit *Speichern* ist die Erstellung der Aktion abgeschlossen. Aufgerufen und ausgeführt wird diese Aktion, sobald sich ein Teilnehmer zu einer Veranstaltung angemeldet hat.

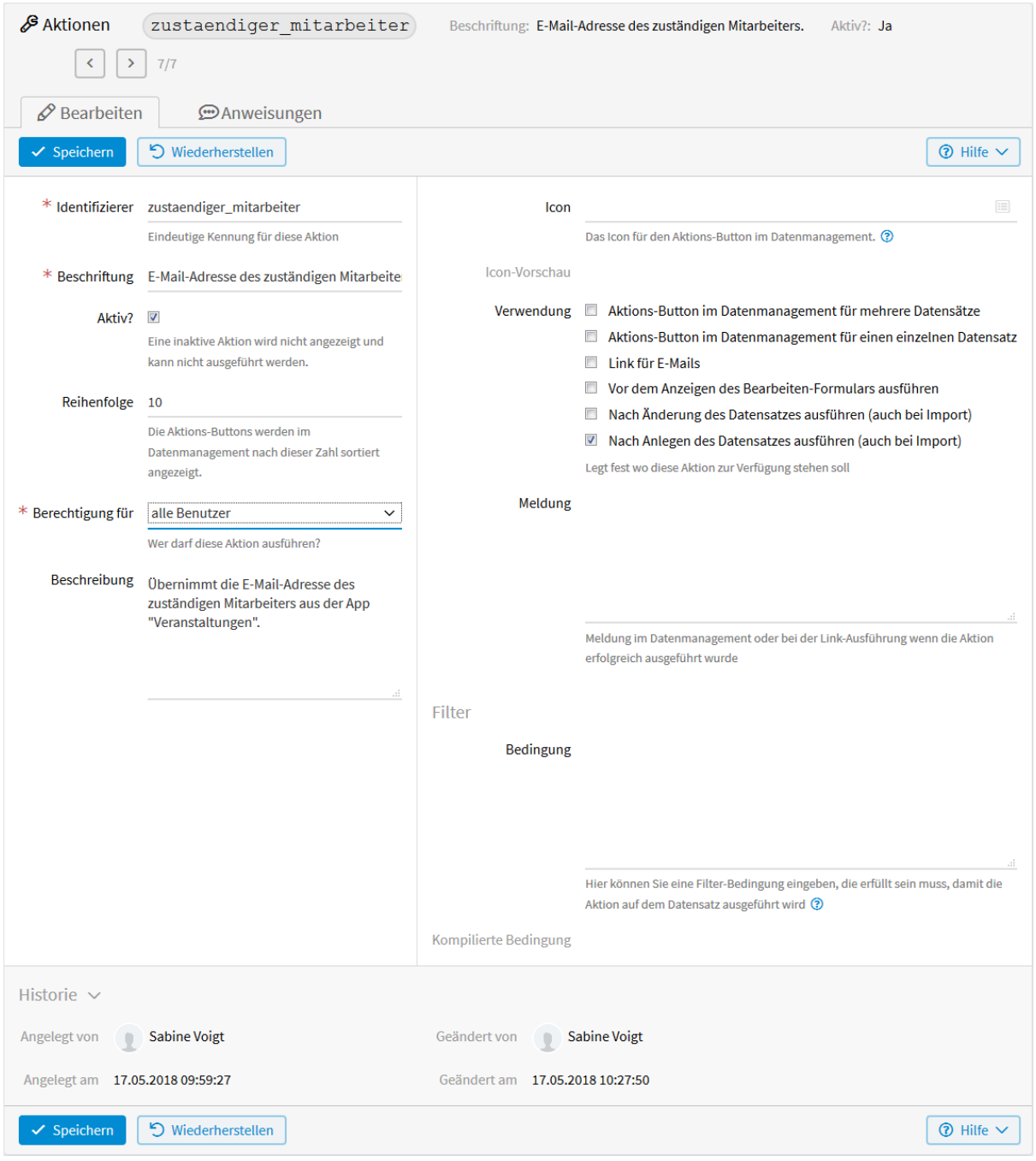


Abb. 120: Datensatz verändern

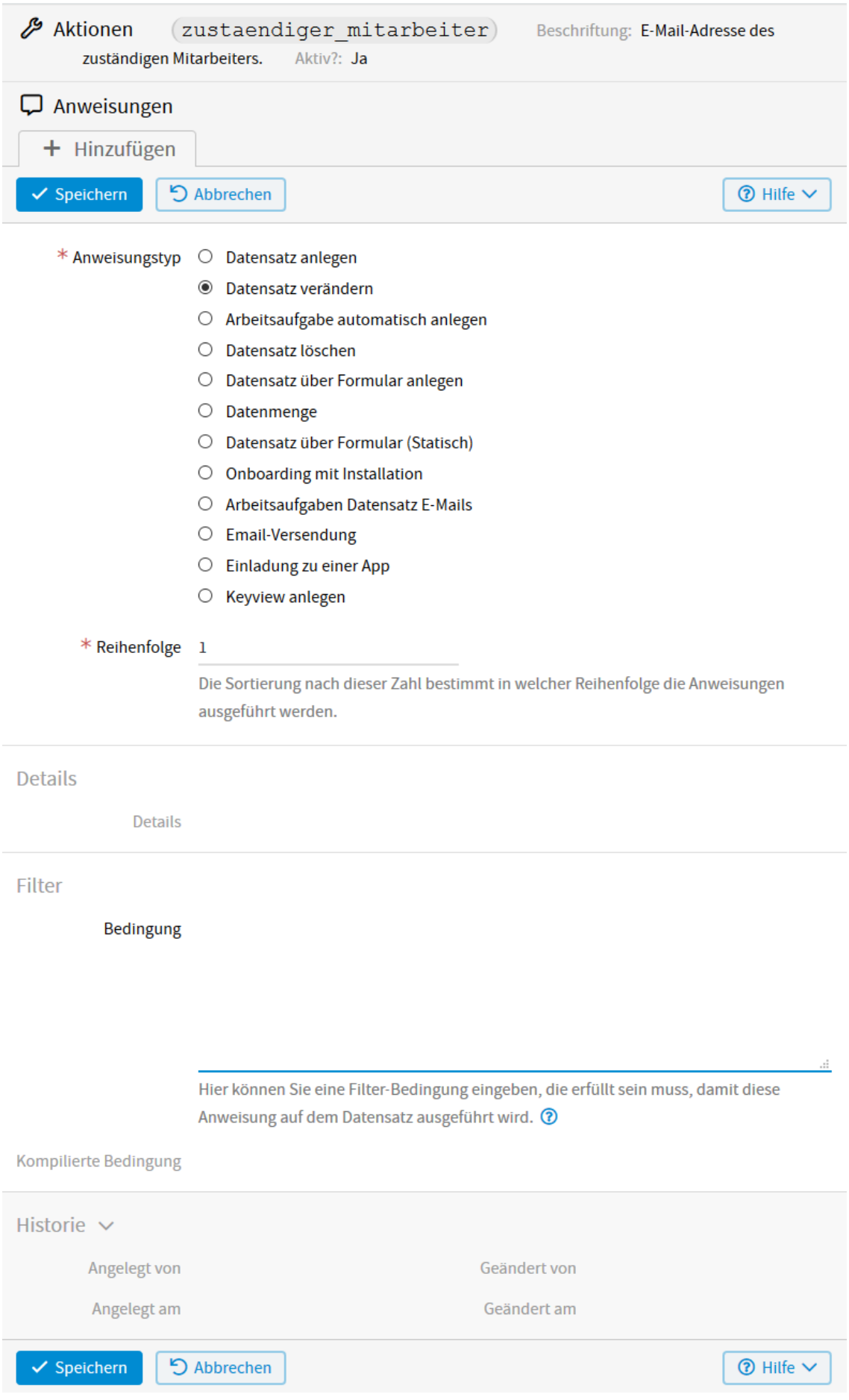


Abb. 121: Anweisungen - Datensatz verändern

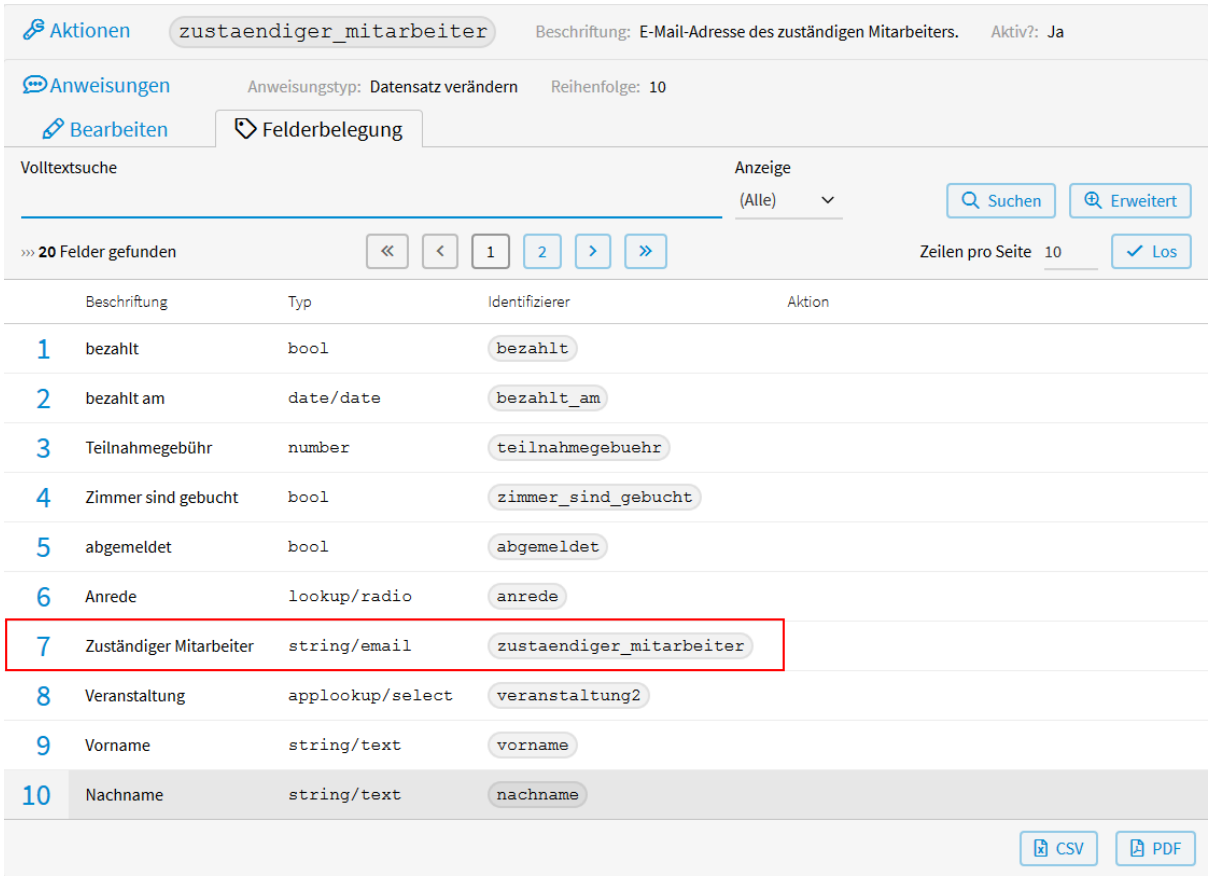


Abb. 122: Felderbelegung - Datensatz verändern

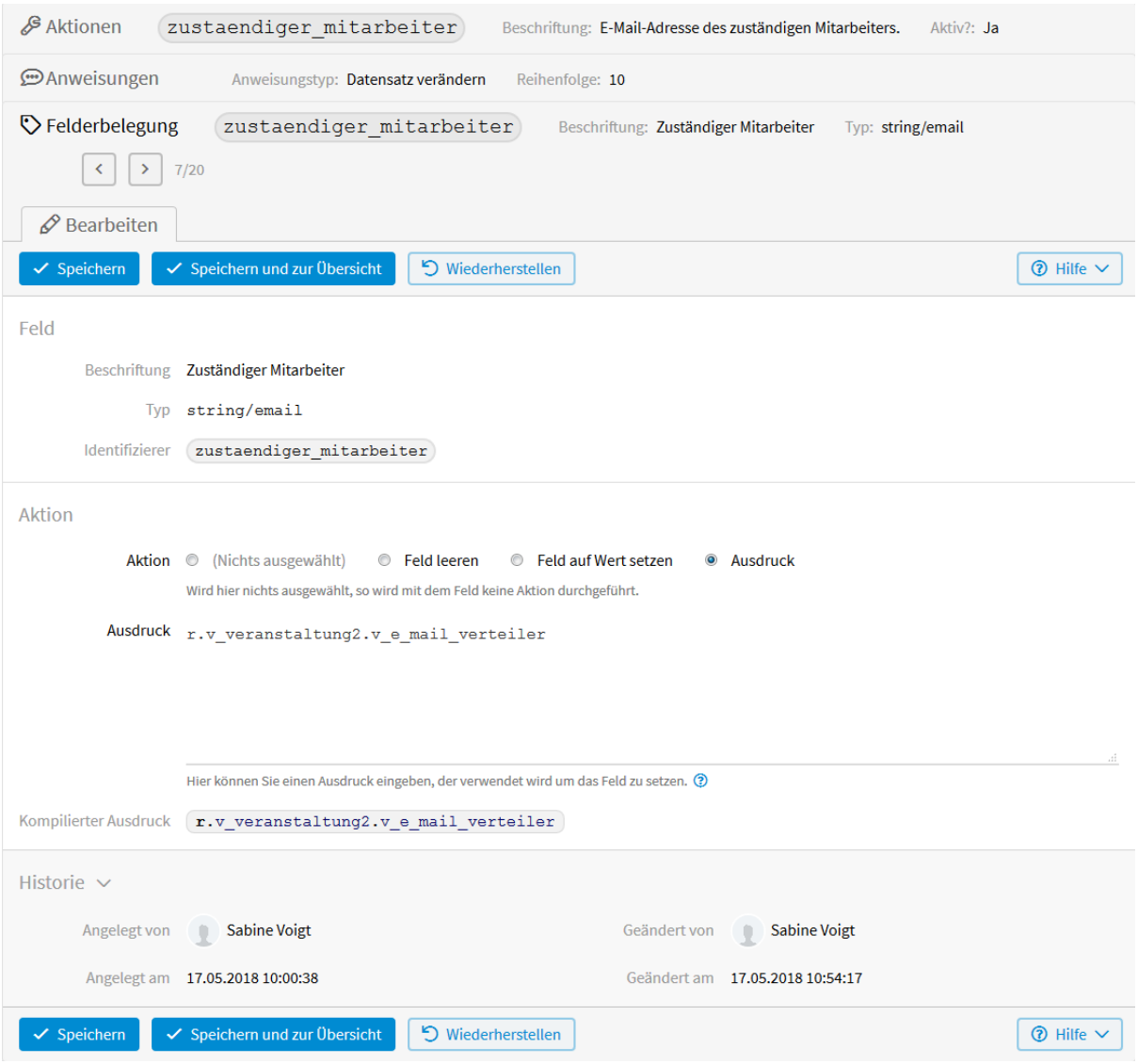


Abb. 123: Felderkonfiguration - Datensatz verändern

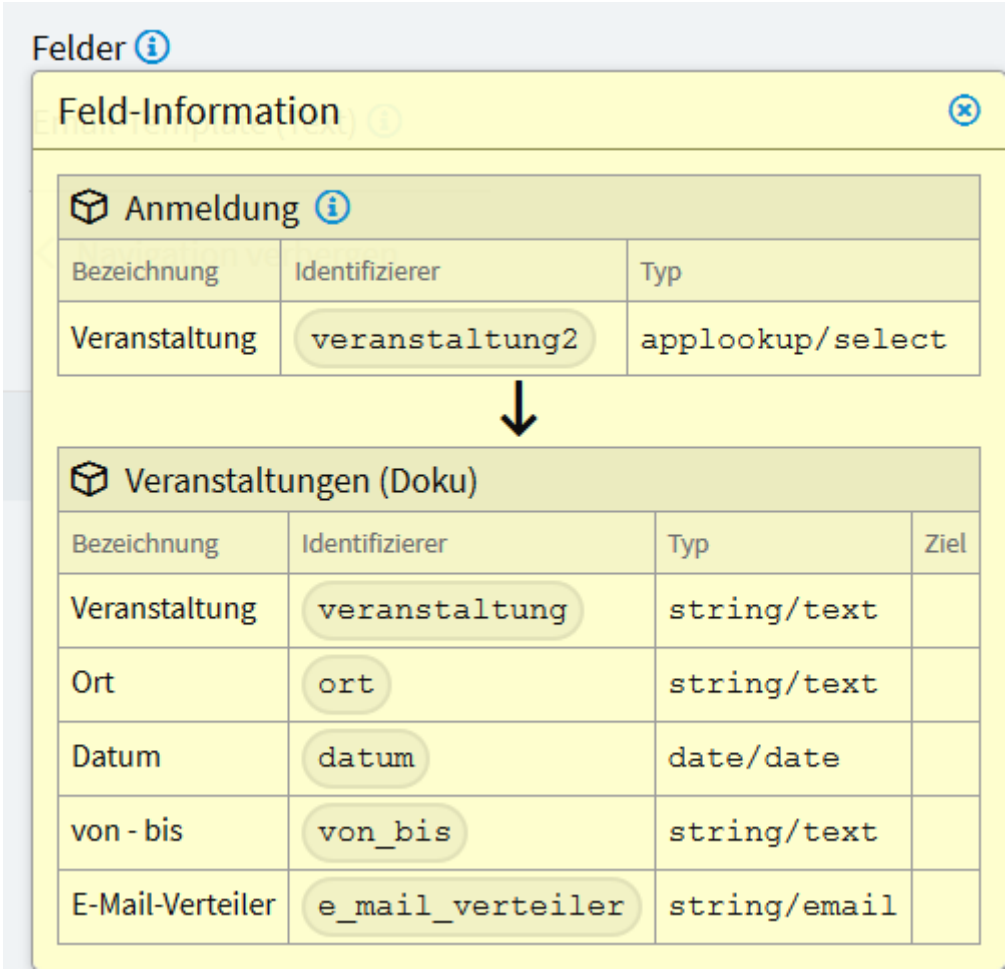


Abb. 124: Feldinformationen - Datensatz verändern

Link für E-Mails

Diese Aktion soll automatisch im Feld *abgemeldet* des Anmeldeformulars ein Häkchen setzen, sobald sich ein Teilnehmer über die Anmeldebestätigung per E-Mail-Link wieder abmeldet.

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine neue Aktion zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

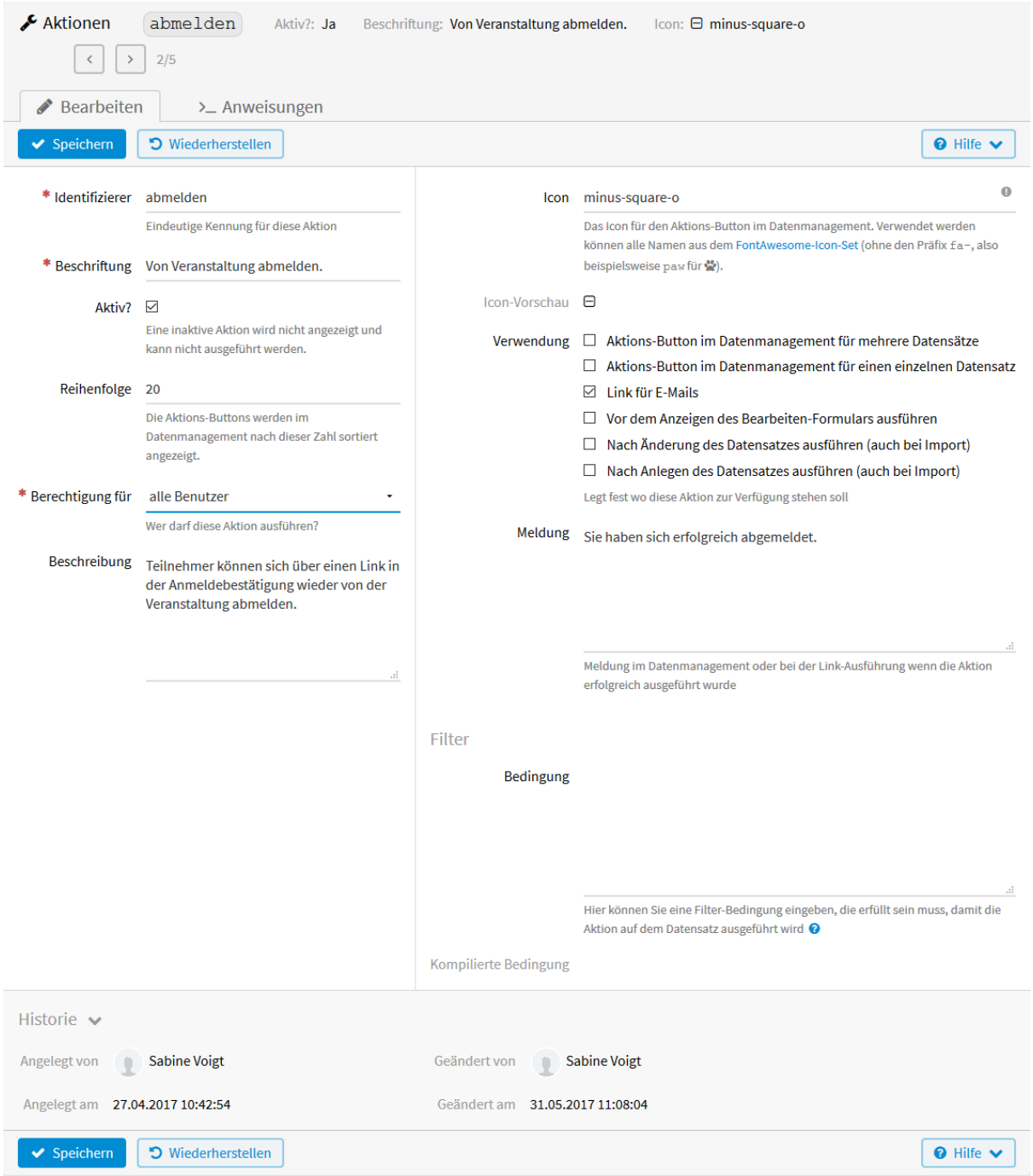


Abb. 125: Link für E-Mails

Über den *Identifizierer* `abmelden` wird die Aktion in die E-Mail eingebunden mit: `{actionlink:abmelden}` (siehe *Code HTML-E-Mail*). Die *Beschriftung* der Aktion `Von Veranstaltung abmelden` steht als *anklickbarer Link* in der HTML-E-Mail des Empfängers. Bei erfolgreicher Ausführung der Aktion wird dem Benutzer die

Meldung *Sie haben sich erfolgreich abgemeldet* angezeigt.

Nach dem Speichern erscheint neben *Bearbeiten* der Tab *Anweisungen*. Nach *Hinzufügen* einer neuen Anweisung kann im nun geöffneten Fenster eingestellt werden, was durch die Aktion ausgeführt werden soll. Siehe *Anweisungen*.

Als *Anweisungstyp* wird *Datensatz verändern* gewählt und bei *Reihenfolge* `10` eingetragen.

Nach dem Speichern der Anweisungen erscheint neben *Bearbeiten* der Tab *Felderbelegung*. Durch Klicken darauf erscheinen alle *Felder der App*, deren Datensätze durch die Aktion verändert werden können. Durch klicken auf das zu ändernde Feld *abgemeldet*, kann dessen *Inhalt konfiguriert* werden.

Im nun geöffneten Fenster wird bei *Aktion Feld auf Wert setzen* und bei *Wert Ja* ausgewählt. Durch Klicken auf *Speichern* ist die Erstellung der Aktion abgeschlossen. Der Link kann mit `{actionurl:abmelden}` oder `{actionlink:abmelden}` in Ihr E-Mail-Template eingebunden werden. Der Unterschied wird in der Dokumentation der *E-Mail-Templates* anhand eines Beispiels erklärt.

Aufgerufen und ausgeführt wird diese Aktion, sobald ein Teilnehmer in der Anmeldebestätigungs-E-Mail den Link zum Abmelden anklickt.

Datensatz löschen

Mit dieser Aktion soll ein einzelner, oder mehrere Datensätze im Datenmanagement gelöscht werden. Mit der Bedingung, dass der (die) Teilnehmer sich abgemeldet hat (haben).

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine neue Aktion zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Über das *Formulieren der Bedingung* können Sie im entsprechenden Kapitel mehr lesen.

Nach dem Speichern erscheint neben *Bearbeiten* der Tab *Anweisungen*. Nach *Hinzufügen* einer neuen Anweisung kann im nun geöffneten Fenster eingestellt werden, was durch die Aktion ausgeführt werden soll. Siehe *Anweisungen*.

Als *Anweisungstyp* wird *Datensatz löschen* gewählt und bei *Reihenfolge* `10` eingetragen. Nach dem Speichern der Anweisungen ist die Erstellung der Aktion abgeschlossen.

Ausgeführt werden kann diese Aktion in der Liste des Datenmanagements unter *Daten* → *Liste*.

Datensatz über Fomular anlegen

Bei dieser Aktion wird mit einer zusätzlichen App gearbeitet. In dieser App sind verschiedene Veranstaltungen erfasst. Durch Auswahl eines dieser Datensätze soll die Aktion „Anmelden“ ausgelöst werden. Dadurch öffnet sich das Eingabeformular der App „Anmeldung“. Hier sollen dann bereits Felder mit festen Werten, oder Ausdrucken, die sich auf den Datensatz beziehen, vorbelegt werden. Wird das Eingabeformuar dann gespeichert, wird der neue Datensatz angelegt. Das heißt, es kann aus der App „Veranstaltungen“ heraus, ein Datensatz in der App „Anmeldung“ angelegt werden.

Wählen Sie die App „Veranstaltungen“ und dort im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine neue Aktion zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Nach dem Speichern erscheint neben *Bearbeiten* der Tab *Anweisungen*. Nach *Hinzufügen* einer neuen Anweisung kann im nun geöffneten Fenster eingestellt werden, was durch die Aktion ausgeführt werden soll. Siehe *Anweisungen* hinzuzufügen.

Weil die Datensätze in der App *Anmeldung* angelegt werden sollen, wird diese bei *App* ausgewählt. Nach dem Speichern erscheint neben *Bearbeiten* der Button *Felderbelegung*. Durch Klicken darauf erscheinen in der Übersicht alle *Felder der ausgewählten App Anmeldung*.

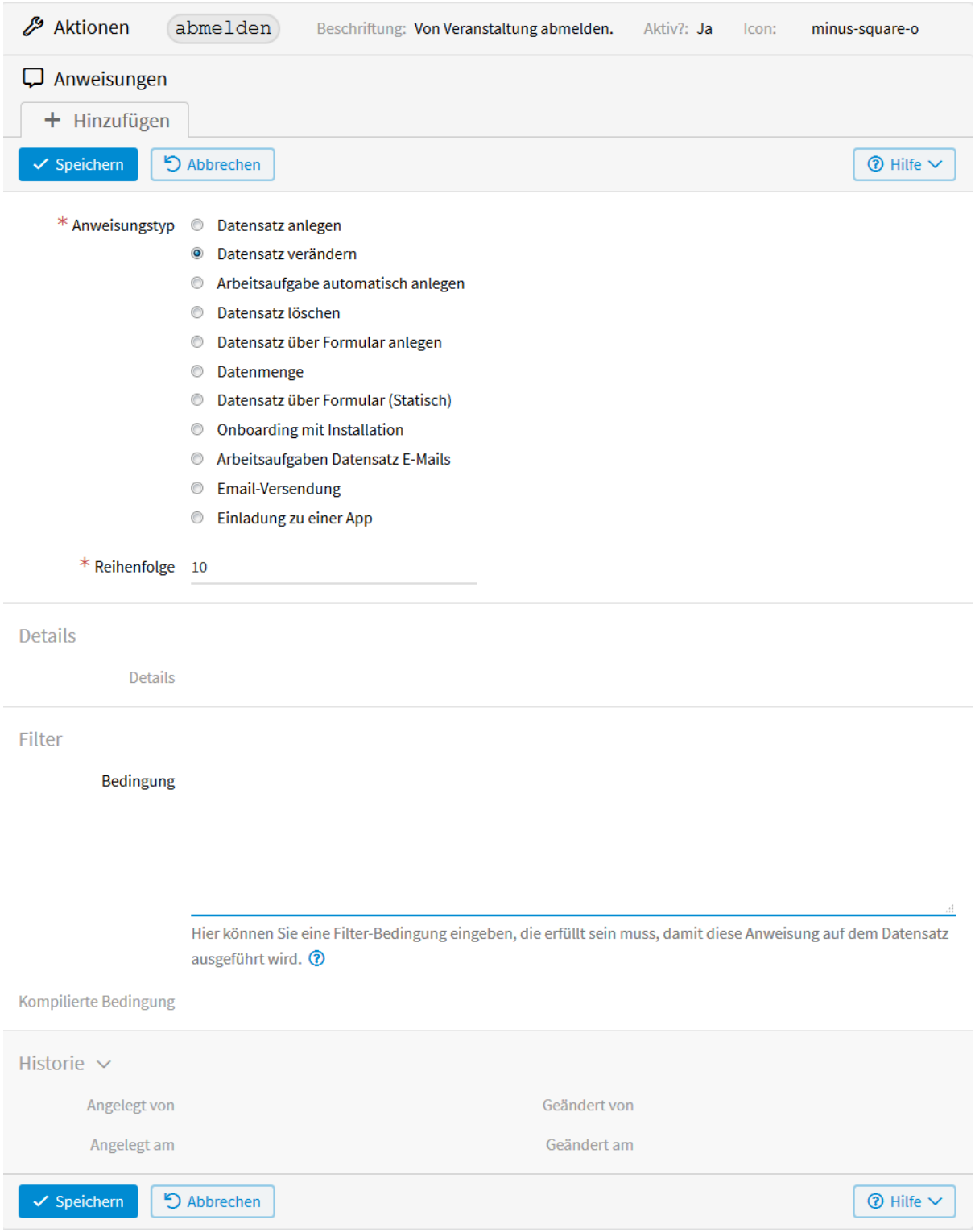


Abb. 126: Anweisungen - Link für E-Mails

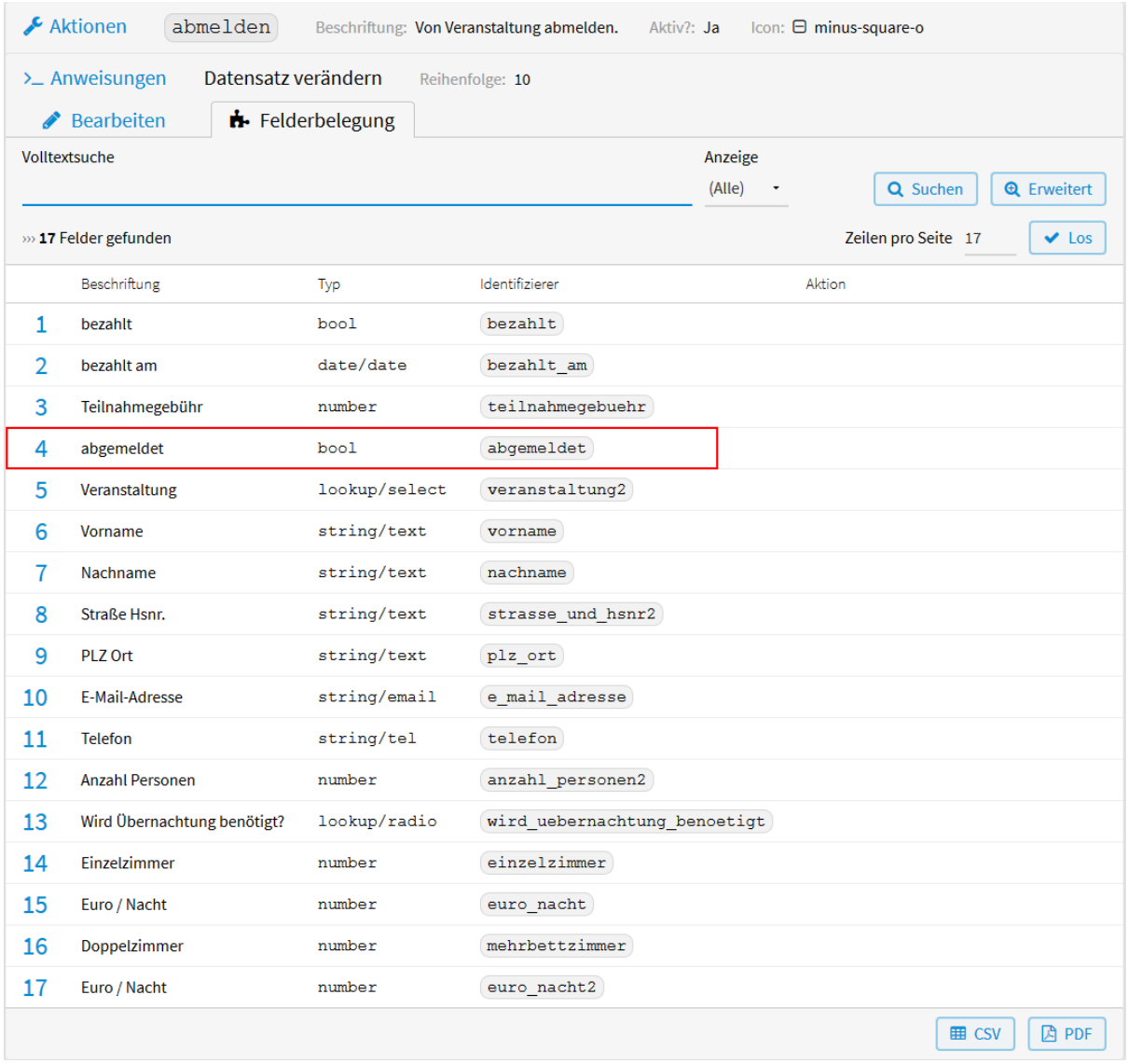


Abb. 127: Felderbelegung - Link für E-Mails

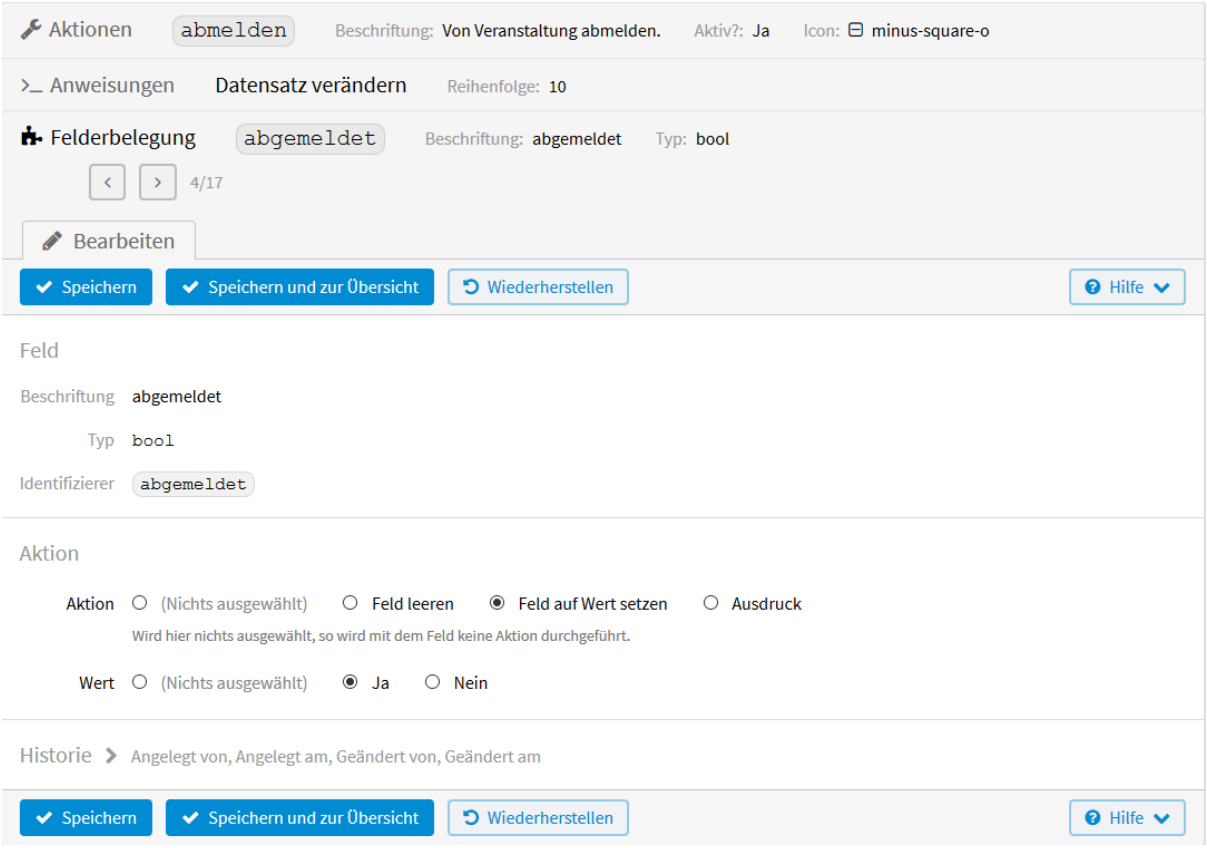


Abb. 128: Felderkonfiguration - Link für E-Mails

Diese können hier mit Ausdrucken der App *Veranstaltungen* belegt werden. In unserem Beispiel betrifft das die Felder der *Veranstaltung*, *Ort*, *Datum* und *von-bis*. Durch Klicken auf die jeweilige Zeile, können diese Felder konfiguriert werden. Dazu können Sie in der allgemeinen Dokumentation über *Felderbelegung* mehr lesen.

Für die Formulierung des Ausdrucks wird *vSQL* verwendet. Dafür werden die Identifizierer der Felder der App *Veranstaltungen* benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*. Auf die Werte kann dann zugegriffen werden mit: `r.v_identifier`. In unserem Beispiel lauten die Ausdrucke `r.v_veranstaltung`, `r.v_ort`, `r.v_datum` und `r.v_von_bis`. Siehe *Ergebnis Felderkonfiguration*.

Nach der Eingabe und dem Speichern aller benötigten Ausdrucke ist die Erstellung der Aktion abgeschlossen. *Aufgerufen* werden kann die Aktion in der Liste des Datenmanagements der App *Veranstaltungen* unter *Daten* → *Liste*. Klicken Sie dafür ganz rechts auf den Button in der Zeile der Veranstaltung, für die Sie eine Anmeldung anlegen wollen und wählen in dem Menü die Aktion *Anmeldung*. . . .

Nach dem Klicken auf *Anmeldung* öffnet sich das Eingabefenster der App *Anmeldung*, in dem dann bereits die vier Felder mit den Daten aus der App *Veranstaltung* vorbelegt sind. Siehe *Ergebnis*.

Jetzt müssen nur noch die persönlichen Daten ergänzt und der Datensatz gespeichert werden.

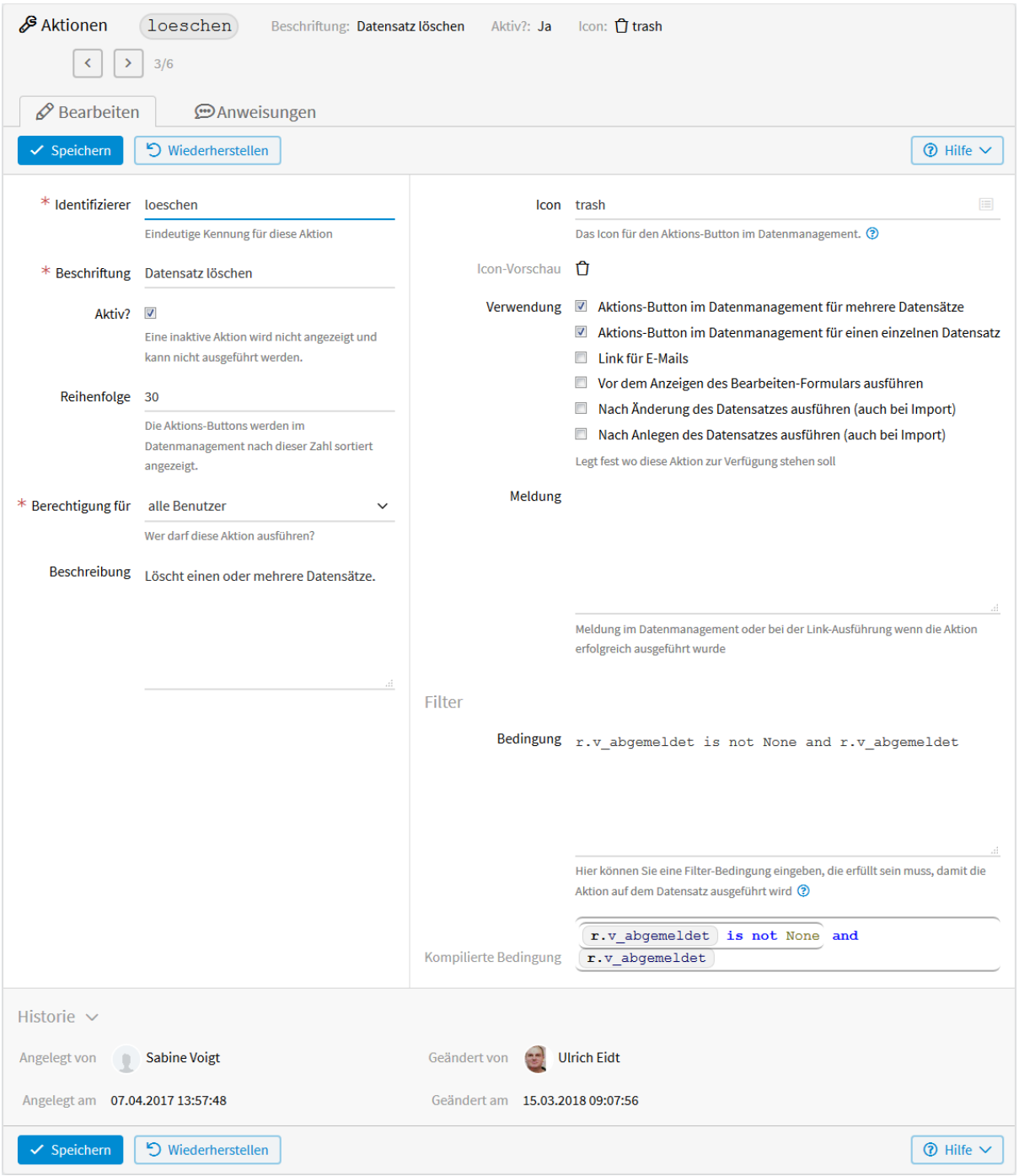


Abb. 129: Datensatz löschen

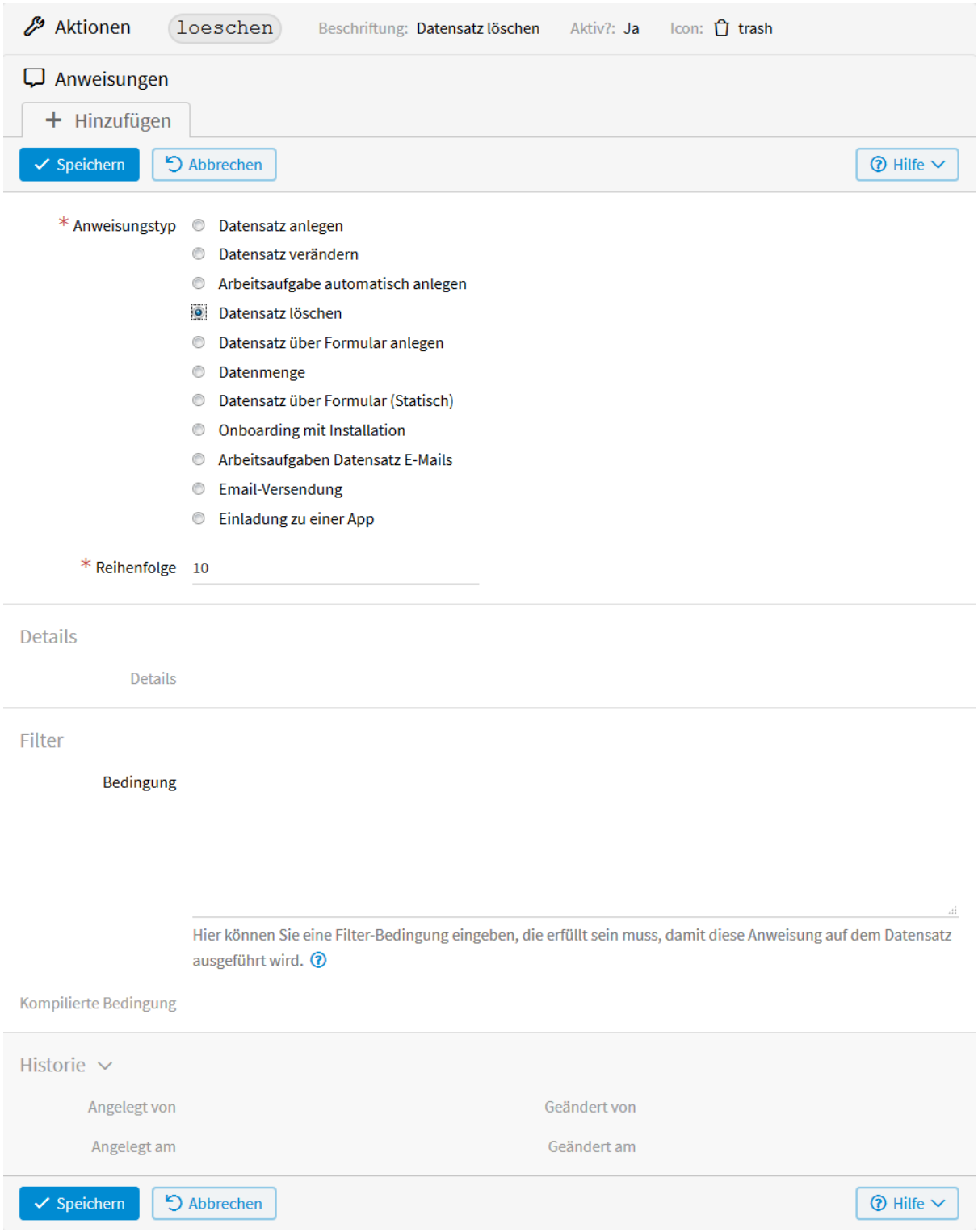


Abb. 130: Anweisungen - Datensatz löschen

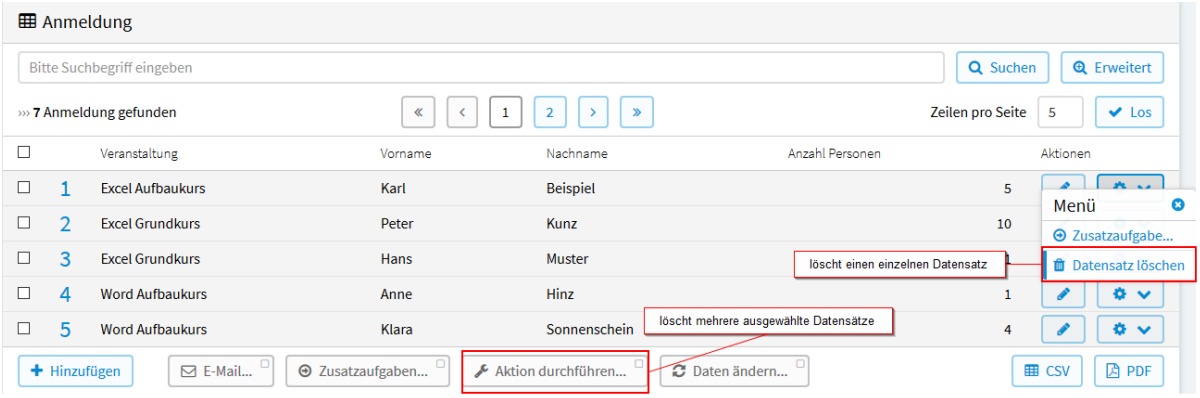


Abb. 131: Aktion ausführen - Datensatz löschen

Arbeitsaufgaben Datensatz E-Mails

Im folgenden Anwendungsbeispiel soll bei einer neuen Anmeldung zur Veranstaltung eine Arbeitsaufgabe angelegt werden, mit der Bedingung, dass im Anmeldeformular bei *Übernachtung benötigt ja* gewählt wurde.

Die Arbeitsaufgabe fordert auf, die entsprechende Anzahl an Zimmern zu buchen. Nach Erledigung der Aufgabe soll durch eine weitere Aktion das Häkchen bei *Zimmer gebucht* im Anmeldeformular gesetzt werden.

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine neue Aktion zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Da die Arbeitsaufgabe nur angelegt werden sollen, wenn im Anmeldeformular bei *Übernachtung benötigt Ja* gewählt wurde, muss folgende *Bedingung* eingegeben werden:

```
r.v_wird_uebernachtung_benoetigt == "ja"
```

Hierzu können sie im Kapitel *Formulieren der Bedingung* mehr lesen.

Nach dem Speichern erscheint neben *Bearbeiten* der Tab *Anweisungen*. Nach *Hinzufügen* einer neuen Anweisung kann im nun geöffneten Fenster eingestellt werden, was durch die Aktion ausgeführt werden soll. Siehe *Anweisungen*.

Als *Anweisungstyp* wird *Arbeitsaufgaben Datensatz E-Mails* gewählt und bei *Reihenfolge* `10` eingegeben.

Nach dem Speichern der Anweisungen erscheint neben *Bearbeiten* der Tab *Felderbelegung*. Durch Klicken darauf sehen Sie folgende *Felder*. Diese sind bei jeder App gleich.

User - string/text - c_user
: Angemeldeter Benutzer - kann nicht geändert werden.

Datensatz - string/text - dat_id
: Datensatz, an den die Aufgabe gehängt wird (dies ist normalerweise `r.id`, d.h. der Datensatz für den die Aktion aufgerufen wurde).

E-Mails als Komma-Liste - string/text - emails
: E-Mail-Adresse(n), an die die Arbeitsaufgabe als Link verschickt wird (werden). Z. B. können Sie `r.v_identifier` (Identifizierer des E-Mail-Feldes) bei *Ausdruck* oder die E-Mail-Adressen kommagetrennt bei *Feld auf Wert setzen* eintragen.

  Bitte beachten: Bei E-Mail-Adressen, die im System sind, werden die Aufgaben im Account erzeugt. Ansonsten kommt die Aufgabe immer als E-Mail-Aufgabe. Wird bei *Per Email immer Ja* gesetzt ist, kommt die Aufgabe immer (auch) als E-Mail-Aufgabe.

Aufgabe - string/text - tsk_name
: Name der Aufgabe

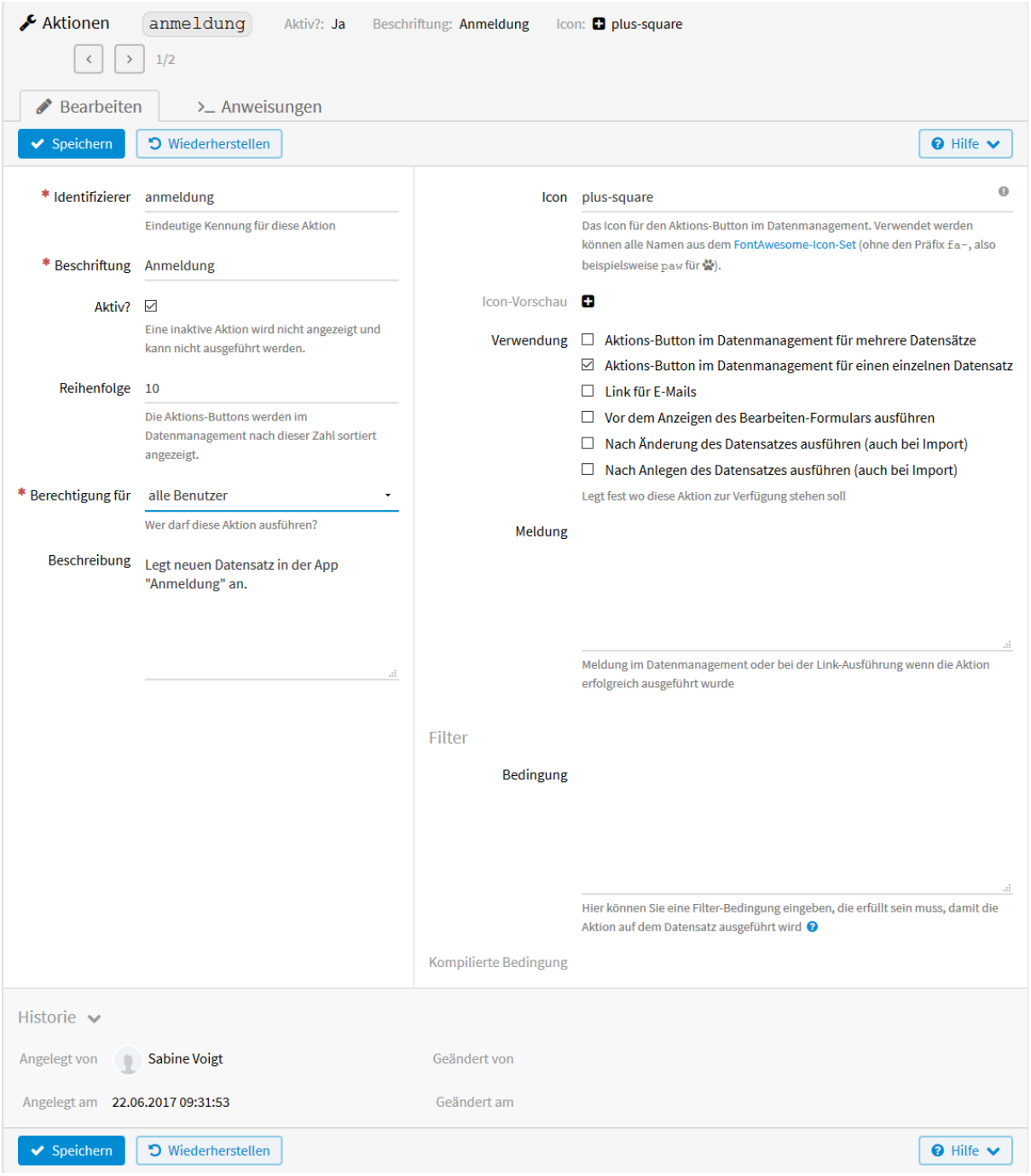


Abb. 132: Datensatz über Formular anlegen

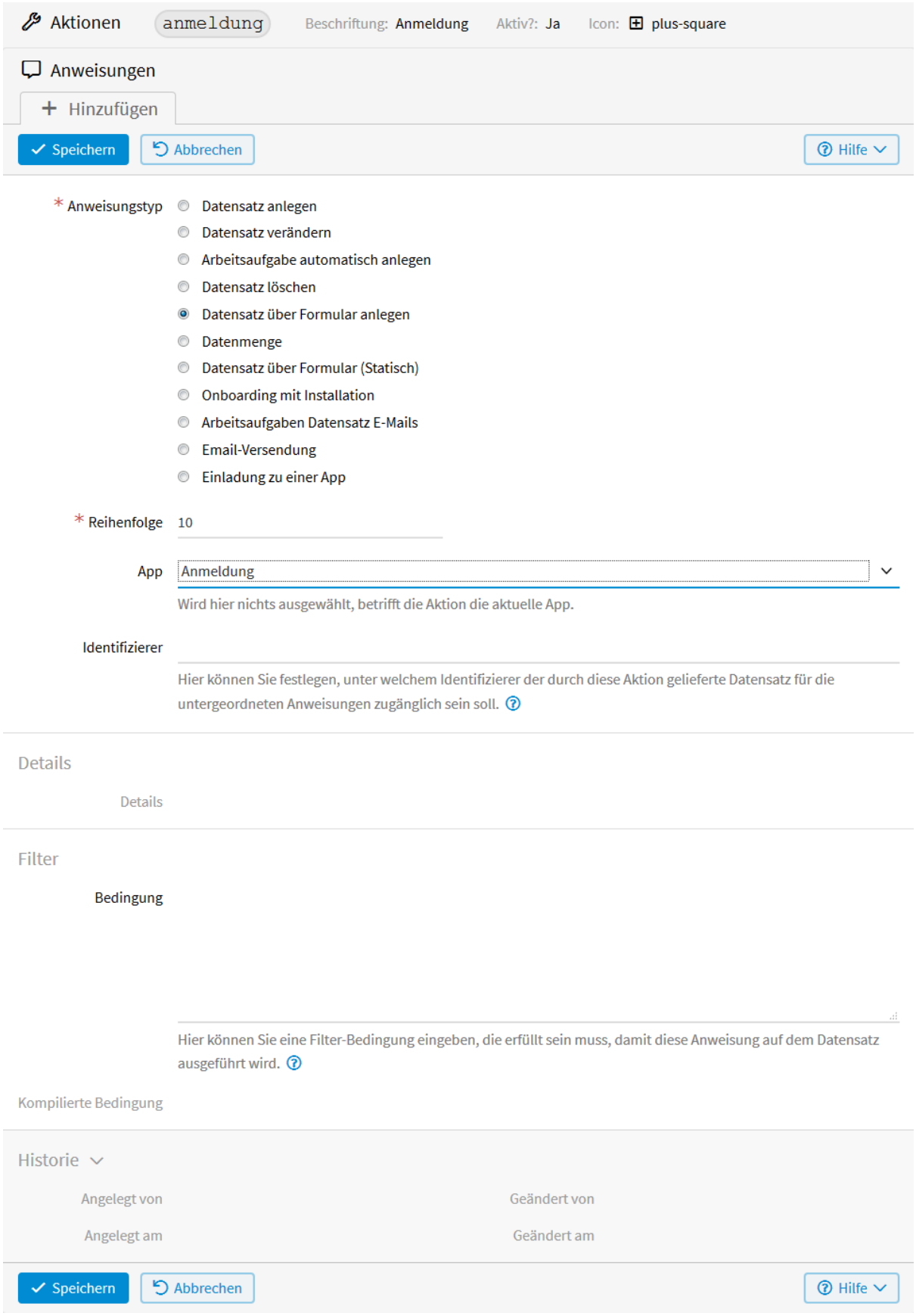


Abb. 133: Anweisungen - Datensatz über Formular anlegen

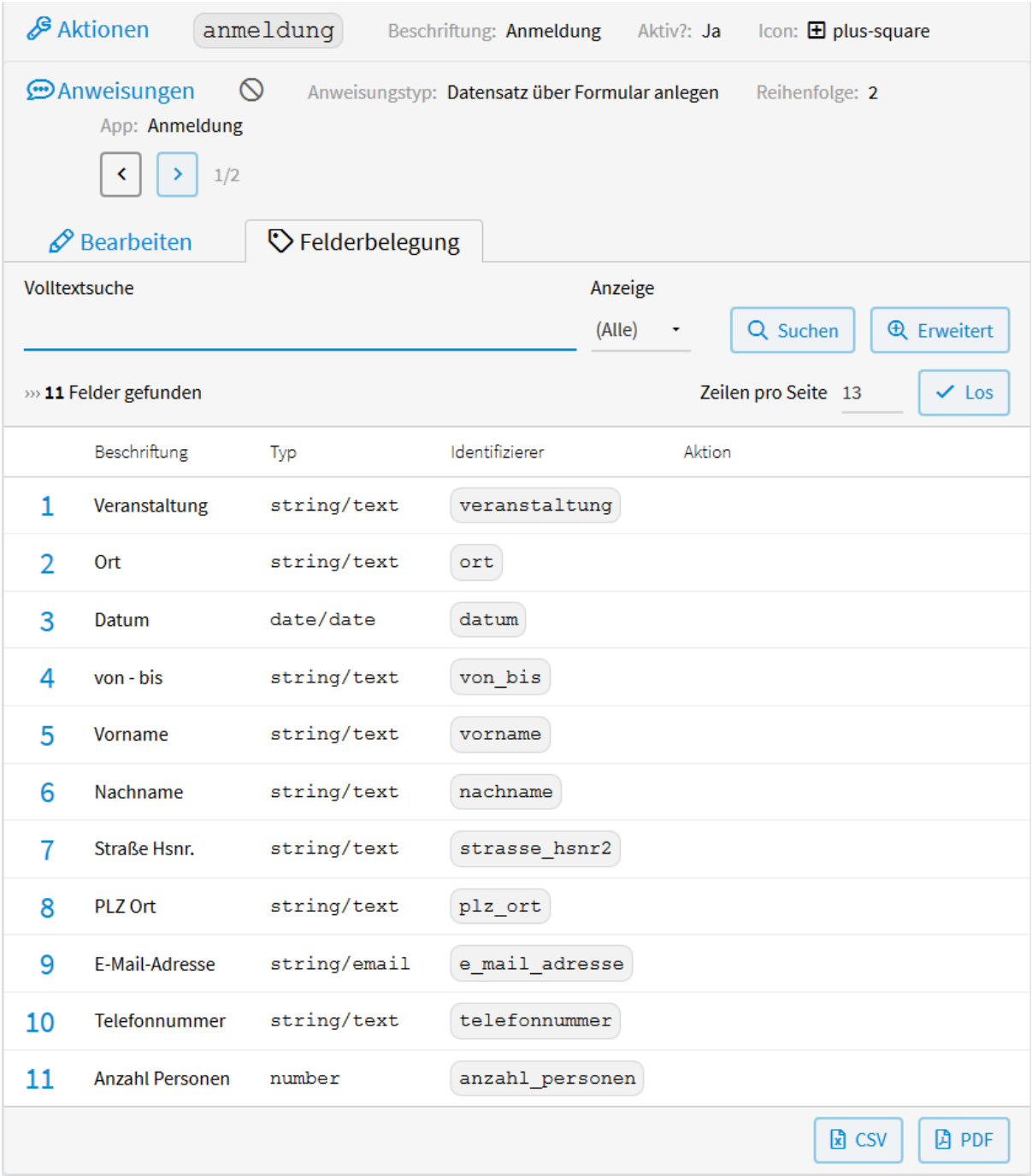


Abb. 134: Felderbelegung - Datensatz über Formular anlegen

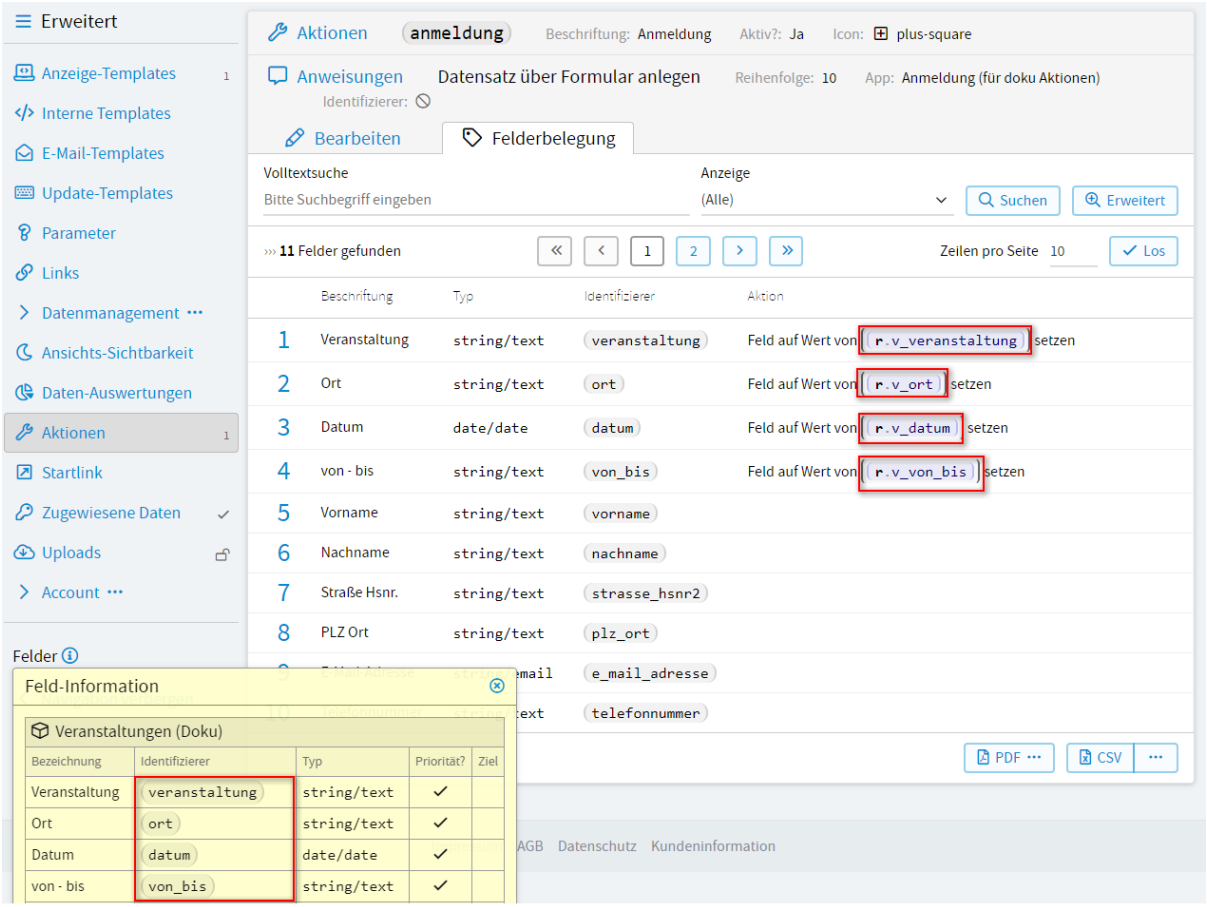


Abb. 135: Ergebnis Felderkonfiguration - Datensatz über Formular anlegen

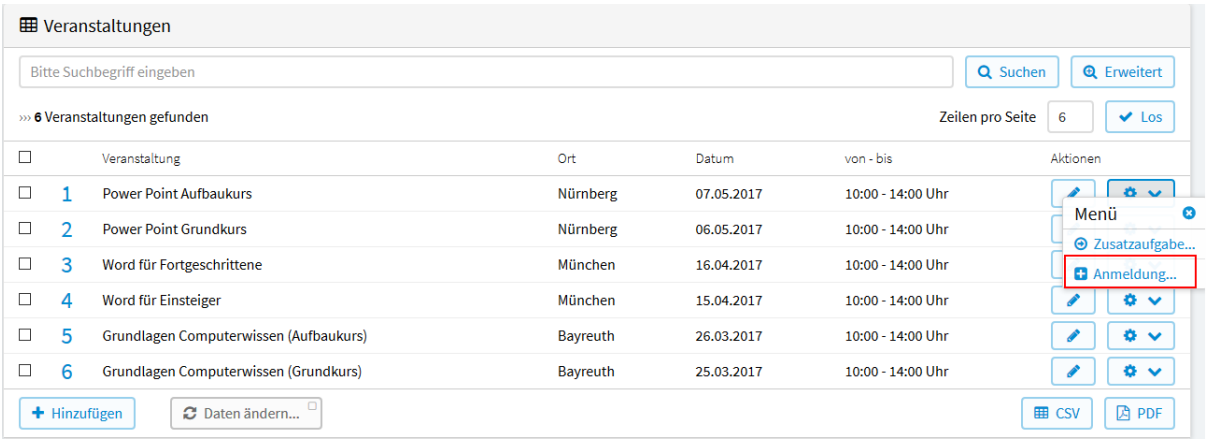


Abb. 136: Aktion aufrufen - Datensatz über Formular anlegen

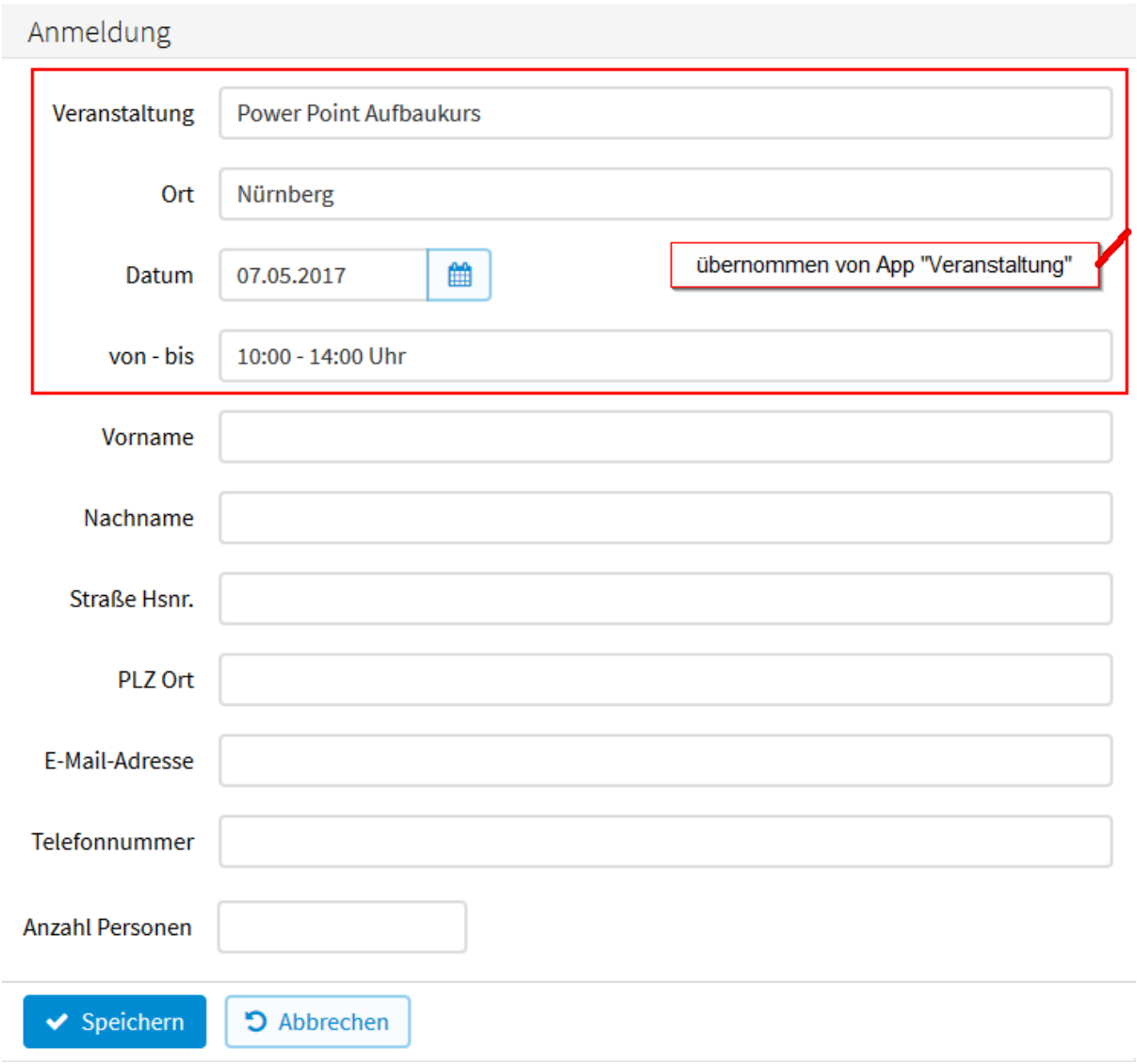


Abb. 137: Ergebnis - Datensatz über Formular anlegen

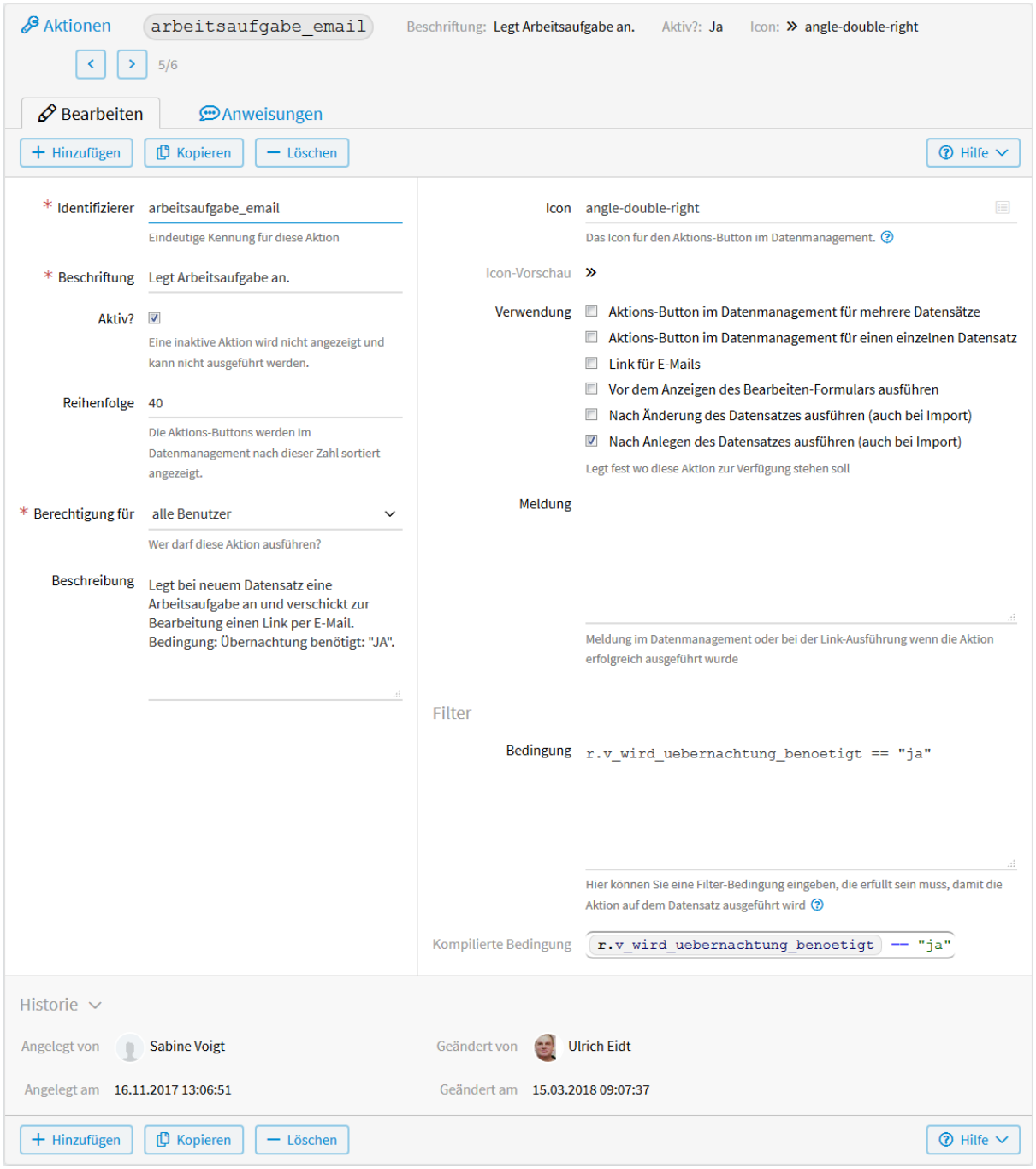


Abb. 138: Arbeitsaufgaben Datensatz E-Mails

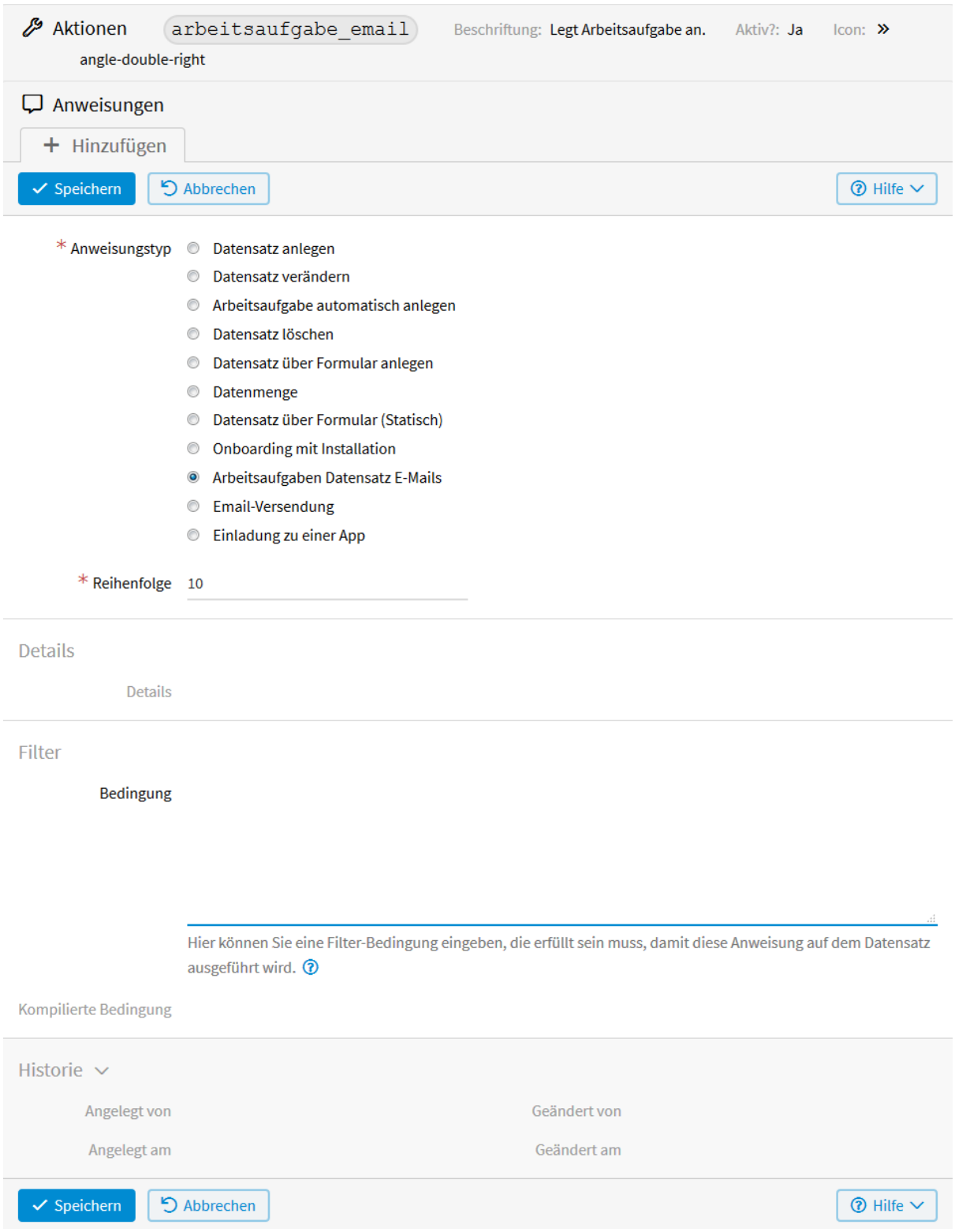


Abb. 139: Anweisungen - Arbeitsaufgaben Datensatz E-Mails

Aktionen arbeitsaufgabe_email Beschriftung: Legt Arbeitsaufgabe an.
Aktiv?: Ja

Anweisungen Arbeitsaufgaben Datensatz E-Mails Reihenfolge: 1

Bearbeiten | Felderbelegung

Volltextsuche | Anzeige: (Alle) | Suchen | Erweitert

››› 25 Felder gefunden | Zeilen pro Seite 25 | Los

	Beschriftung	Typ	Identifizierer	Aktion
1	User	string/text	c_user	
2	Datensatz	string/text	dat_id	
3	E-Mails als Komma-Liste	string/text	emails	
4	Aufgabe	string/text	tsk_name	
5	Beschreibung	string/text	tsk_description	
6	Ergebnis 1	string/text	res_name1	
7	Datenaktion bei 1	string/text	da_id_1	
8	Ergebnis 2	string/text	res_name2	
9	Datenaktion bei 2	string/text	da_id_2	
10	Ergebnis 3	string/text	res_name3	
11	Datenaktion bei 3	string/text	da_id_3	
12	Ergebnis 4	string/text	res_name4	
13	Datenaktion bei 4	string/text	da_id_4	
14	Ergebnis 5	string/text	res_name5	
15	Datenaktion bei 5	string/text	da_id_5	
16	Zu erledigen bis (Datum)	date	wfl_date	
17	Warn-Email an Bearbeiter am	date	escalate_date	
18	Aufgabe deaktivieren am	date	timeout_date	
19	Benachrichtigung (adhoc_data_note)	string/text	adhoc_type	
20	Aufgabe anzeigen ab	date	wfl_datefrom	
21	Aufgabe anzeigen bis	date	wfl_dateto	
22	Kalenderexport	bool	wfl_calendarexport	
23	Kalender-Alarm	bool	wfl_calendaralarm	
24	Per Email immer	bool	email_forced	
25	Felder verstecken (Komma getrennt)	string/text	hidecontrols	

PDF | CSV

Abb. 140: Felderbelegung - Arbeitsaufgaben Datensatz E-Mails

Beschreibung - string/text - tsk_description
Beschreibung der Aufgabe

Ergebnis 1 - string/text - res_name1
Text des Ergebnis-Buttons (z. B. Erledigt, Abgelehnt, …)

Datenaktion bei 1 - string/text - da_id_1
Datenaktion, die bei Wahl von Ergebnis 1 ausgeführt werden soll. Hier muss die id oder der identifier der Aktion bei *Feld auf Wert setzen* eingetragen werden.

Ergebnis 2 - string/text - res_name2
Text des Ergebnis-Buttons (z. B. Erledigt, Abgelehnt, …)

Datenaktion bei 2 - string/text - da_id_2
Datenaktion, die bei Wahl von Ergebnis 2 ausgeführt werden soll. Hier muss die id oder der identifier der Aktion bei *Feld auf Wert setzen* eingetragen werden.

Ergebnis 3 - string/text - res_name3
Text des Ergebnis-Buttons (z. B. Erledigt, Abgelehnt, …)

Datenaktion bei 3 - string/text - da_id_3
Datenaktion, die bei Wahl von Ergebnis 3 ausgeführt werden soll. Hier muss die id oder der identifier der Aktion bei *Feld auf Wert setzen* eingetragen werden.

Ergebnis 4 - string/text - res_name4
Text des Ergebnis-Buttons (z. B. Erledigt, Abgelehnt, …)

Datenaktion bei 4 - string/text - da_id_4
Datenaktion, die bei Wahl von Ergebnis 4 ausgeführt werden soll. Hier muss die id oder der identifier der Aktion bei *Feld auf Wert setzen* eingetragen werden.

Ergebnis 5 - string/text - res_name5
Text des Ergebnis-Buttons (z. B. Erledigt, Abgelehnt, …)

Datenaktion bei 5 - string/text - da_id_5
Datenaktion, die bei Wahl von Ergebnis 5 ausgeführt werden soll. Hier muss die id oder der identifier der Aktion bei *Feld auf Wert setzen* eingetragen werden.

Zu erledigen bis - date - wfl_date
Bis wann soll die Aufgabe erledigt sein.

Warn-Email an Bearbeiter am - date - escalate_date
Erinnerung

Aufgabe deaktivieren am - date - timeout_date
Wenn die Aufgabe bis zu diesem Datum nicht erledigt ist, wird sie deaktiviert.

Benachrichtigung (adhoc_data_note) - string/text - adhoc_type
`adhoc_data_note` gibt dem Ersteller (= c_user) eine Benachrichtigungs-Aufgabe, mit „Aufgabe wurde erledigt“ - einziges Ergebnis = „Zur Kenntnis genommen“.

Bei Eingabe von `adhoc_no_action`, oder wenn nichts eingegeben wird, passiert nichts.

Aufgabe anzeigen ab - date - wfl_datefrom
Ab wann soll die Aufgabe in der Aufgabenübersicht angezeigt werden (ist gut für „Wiedervorlage“). Funktioniert mit E-Mail-Aufgaben noch nicht.

Aufgabe anzeigen bis - date - wfl_dateto
Bis wann soll die Aufgabe angezeigt werden?

Kalenderexport - bool - wfl_calendarexport
Für Aufgaben-Kalender

Kalender-Alarm - bool - wfl_calendaralarm
Alarm - Stunden oder Tage vorher

Per Email immer - bool - email_forced
Versendet die Aufgabe immer (auch) per E-Mail, auch wenn die E-Mail-Adresse im System ist.

Felder verstecken (kommagetrennt) - string/text - hidecontrols
: Felder, die nicht in den allgemeinen Informationen der Arbeitsaufgabe mit angezeigt werden sollen (z. B. private Felder), können hier versteckt werden. Geben Sie dafür die *Identifizierer* dieser Felder bei *Feld auf Wert setzen* kommagetrennt ein.

Durch Klicken auf das zu ändernde Feld, sind folgende Konfigurationen möglich.

Aktion

Nichts ausgewählt
: Wird hier nichts ausgewählt, so wird mit dem Feld keine Aktion durchgeführt.

Feld leeren
: Der Feldinhalt wird gelöscht. Das ist nur beim Ändern von Datensätzen sinnvoll.

Feld auf aktuelles Datum setzen
: Diese Auswahl erscheint nur, wenn das ausgewählte Feld vom Typ `date`, `datetimeminute` oder `datetimesecond` ist.

Feld auf Wert setzen
: Das Feld übernimmt, nachdem die Aktion ausgeführt wurde, den hier eingegebenen Wert.

Ausdruck
: Für die Formulierung des Ausdrucks wird *vSQL* verwendet. Dafür werden die Identifizierer der Felder benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*. Auf die Werte kann dann zugegriffen werden mit: `r.v_identifier`.

Nach dem Speichern erscheint unterhalb des Feldes *Ausdruck* der kompilierte Ausdruck. Ist ein Fehler in der Formulierung, sehen Sie diesen hier.

In unserem Beispiel wurden folgende *Konfigurationen* vorgenommen.

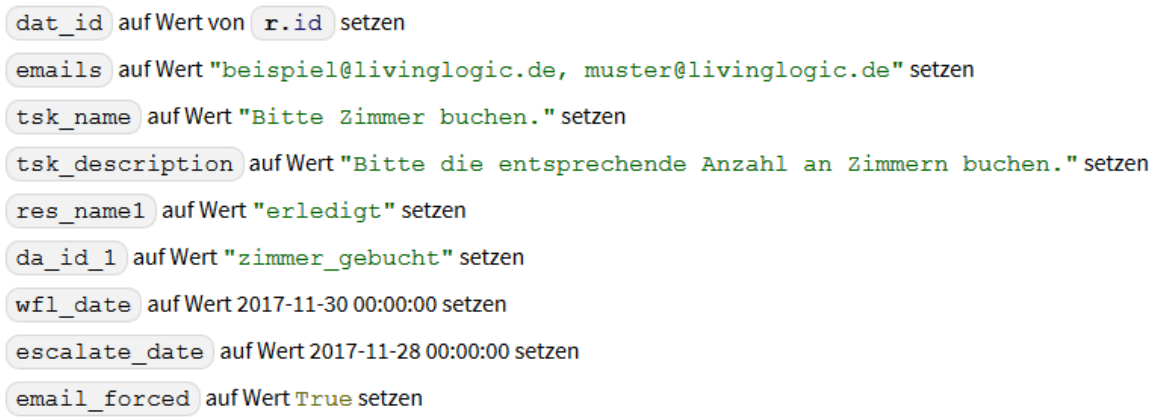


Abb. 141: Felderkonfiguration - Arbeitsaufgaben Datensatz E-Mails

***dat_id* auf Wert von *r.id* setzen**
: Die Arbeitsaufgabe wird an den aktuellen Datensatz gehängt.

***emails* auf Wert "beispiel@livinglogic.de, muster@livinglogic.de" setzen**
: An diese E-Mail-Adressen wird die Arbeitsaufgabe geschickt. Da die E-Mail- Adressen im System sind, wird die *Arbeitsaufgabe im Account* erzeugt.

***tsk_name* auf Wert "Bitte Zimmer buchen" setzen**
: Name der Aufgabe

***tsk_description* auf Wert "Bitte die entsprechende Anzahl an Zimmern buchen." setzen**
: Beschreibung der Aufgabe.

***res_name1* auf Wert "erledigt" setzen**
: Name des Ergebnisbuttons

***da_id_1* auf Wert `"zimmer_gebucht"` setzen**
: Die Aktion mit dem Identifizierer `zimmer_gebucht` wird ausgeführt, sobald die Arbeitsaufgabe `erledigt` ist. Diese Aktion setzt das Häkchen bei „Zimmer sind gebucht“ im Formular der App „Anmeldung“. Wie sie eine Aktion anlegen die einen *Datensatz verändert*, können Sie im entsprechenden Kapitel nachlesen.

***wfl_date* auf Wert `"2017-11-30 00:00:00"` setzen**
: Die Aufgabe soll bis zum 30.11.2017 erledigt sein.

***escalate_date* auf Wert `"2017-11-28 00:00:00"`**
: Erinnerung am 28.11.2017

***email_forced* auf Wert `True` setzen**
: Die Arbeitsaufgabe wird immer auch *per E-Mail* versendet.

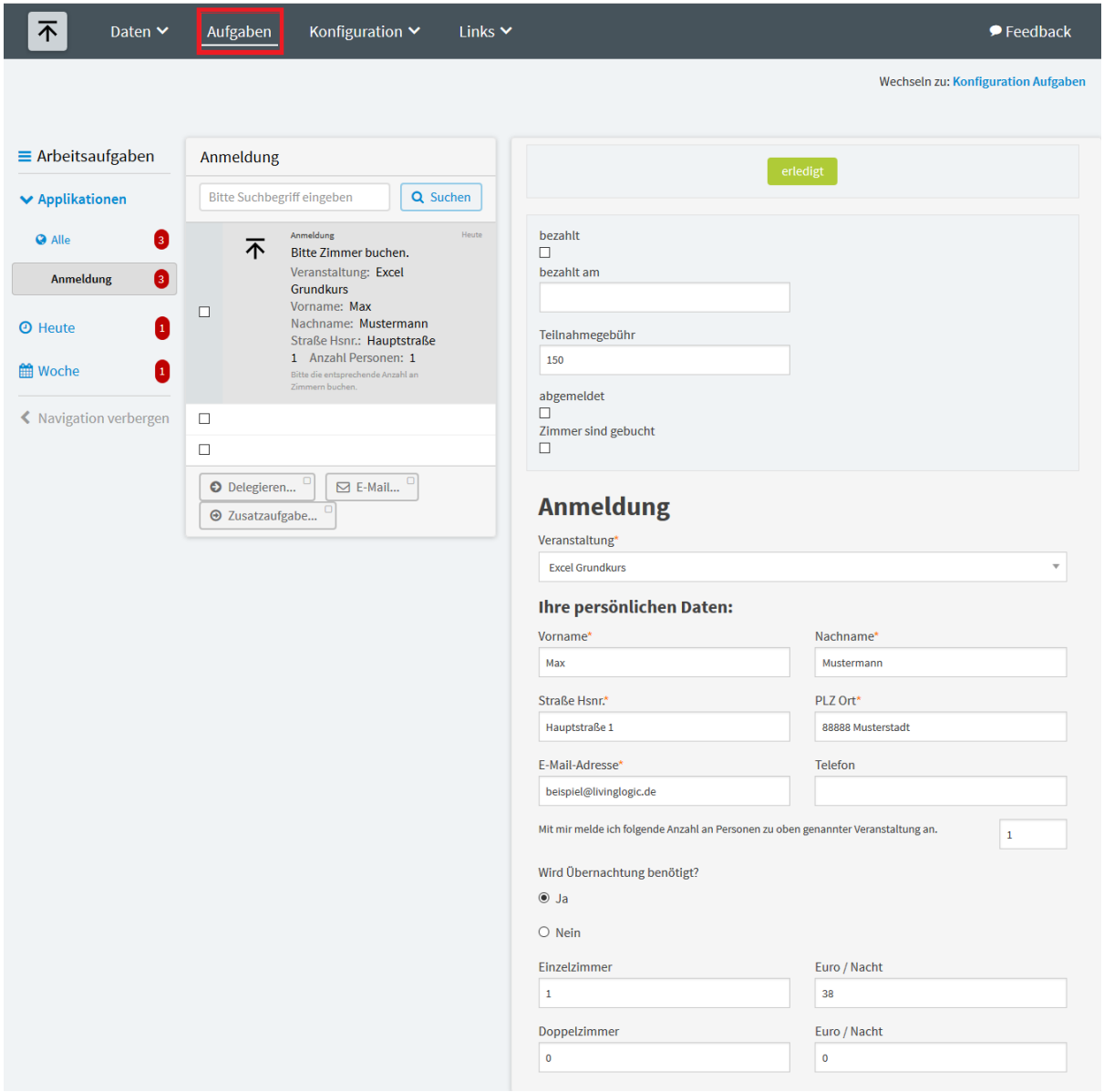


Abb. 142: Arbeitsaufgabe im Account

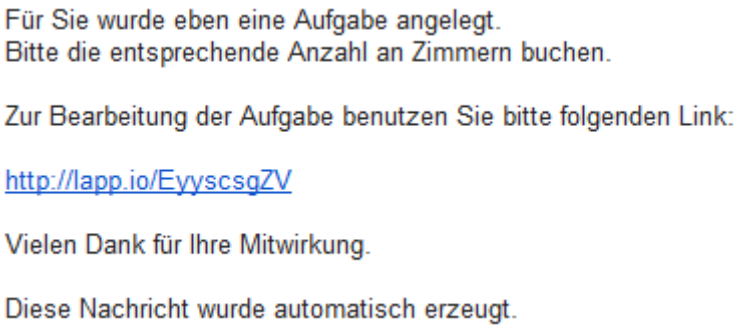


Abb. 143: Arbeitsaufgabe per E-Mail

E-Mail-Versendung

Im folgenden Anwendungsbeispiel sollen Teilnehmer vor einer Veranstaltung per E-Mail erinnert werden. Diese Mails können im Datenmanagement durch eine Aktion verschickt werden. Der E-Mail-Inhalt wird über ein E-Mail-Template eingebunden. Die Erinnerung soll nur an die Teilnehmer verschickt werden, die sich nicht wieder abgemeldet haben.

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine neue Aktion zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Da nur die Teilnehmer erinnert werden sollen, die sich nicht wieder abgemeldet haben, muss folgende *Bedingung* eingegeben werden:

```
not r.v_abgemeldet
```

Hierzu können sie im Kapitel *Formulieren der Bedingung* mehr lesen.

Nach dem Speichern erscheint neben *Bearbeiten* der Tab *Anweisungen*. Nach *Hinzufügen* einer neuen Anweisung kann im nun geöffneten Fenster eingestellt werden, was durch die Aktion ausgeführt werden soll. Siehe *Anweisungen*.

Als *Anweisungstyp* wird *E-Mail-Versendung* gewählt und bei *Reihenfolge* `10` eingegeben.

Nach dem Speichern der Anweisungen erscheint neben *Bearbeiten* der Tab *Felderbelegung*. Durch Klicken darauf sehen Sie folgende *Felder*. Diese sind bei jeder App gleich.

Sprache (=de)
: Wert kann gesetzt werden auf „de“ für Deutsch, oder „en“ für Englisch.

Typ (=dataaction)
: Wert muss immer auf `dataaction` gesetzt werden.

An
: Geben Sie hier den Empfänger der E-Mail an. Die E-Mail kann auch an mehrere Empfänger versendet werden. Dazu müssen Sie deren E-Mail-Adressen unter *Feld auf Wert setzen* kommagetrennt auflisten. Gibt es in Ihrer App ein E-Mail-Feld, können sie, wenn sie *Ausdruck* wählen, auf die dort gespeicherten E-Mail-Adressen zugreifen mit `r.v_identifier`. Wobei identifier für den Identifizierer des E-Mail-Feldes steht.

cc
: Hier können Sie weitere Empfänger angeben. Wenn Sie mehr als eine E-Mail-Adresse eingeben wollen, dann trennen Sie diese durch Kommas.

bcc
: An die hier aufgelisteten, kommagetrennten Empfänger wird eine Blindkopie der E-Mail versendet.

Betreff
: Betreff der E-Mail. Einen gleichbleibenden Wert geben Sie bei *Feld auf Wert setzen* ein. Wählen Sie *Aus-*

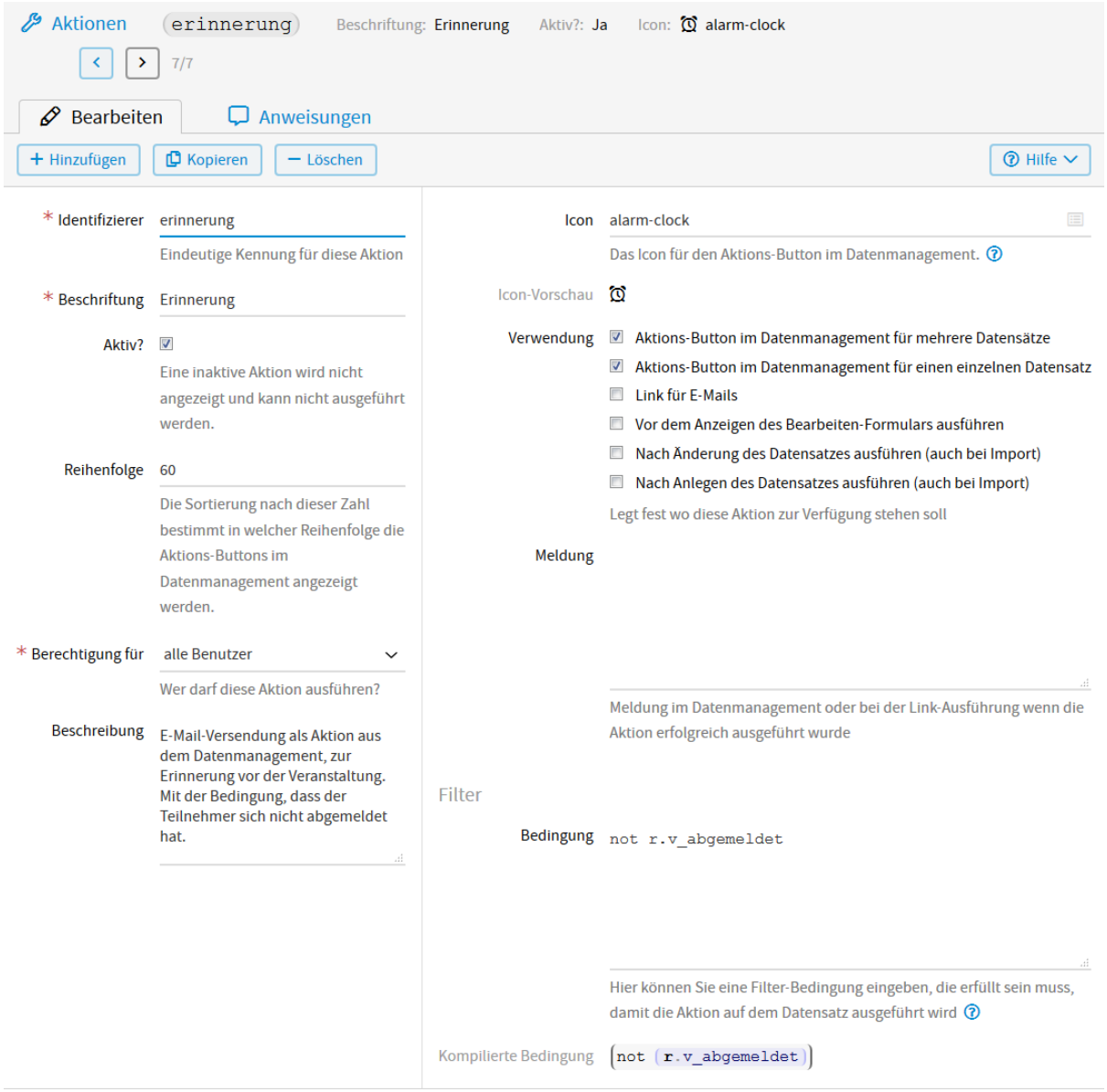


Abb. 144: E-Mail-Versendung

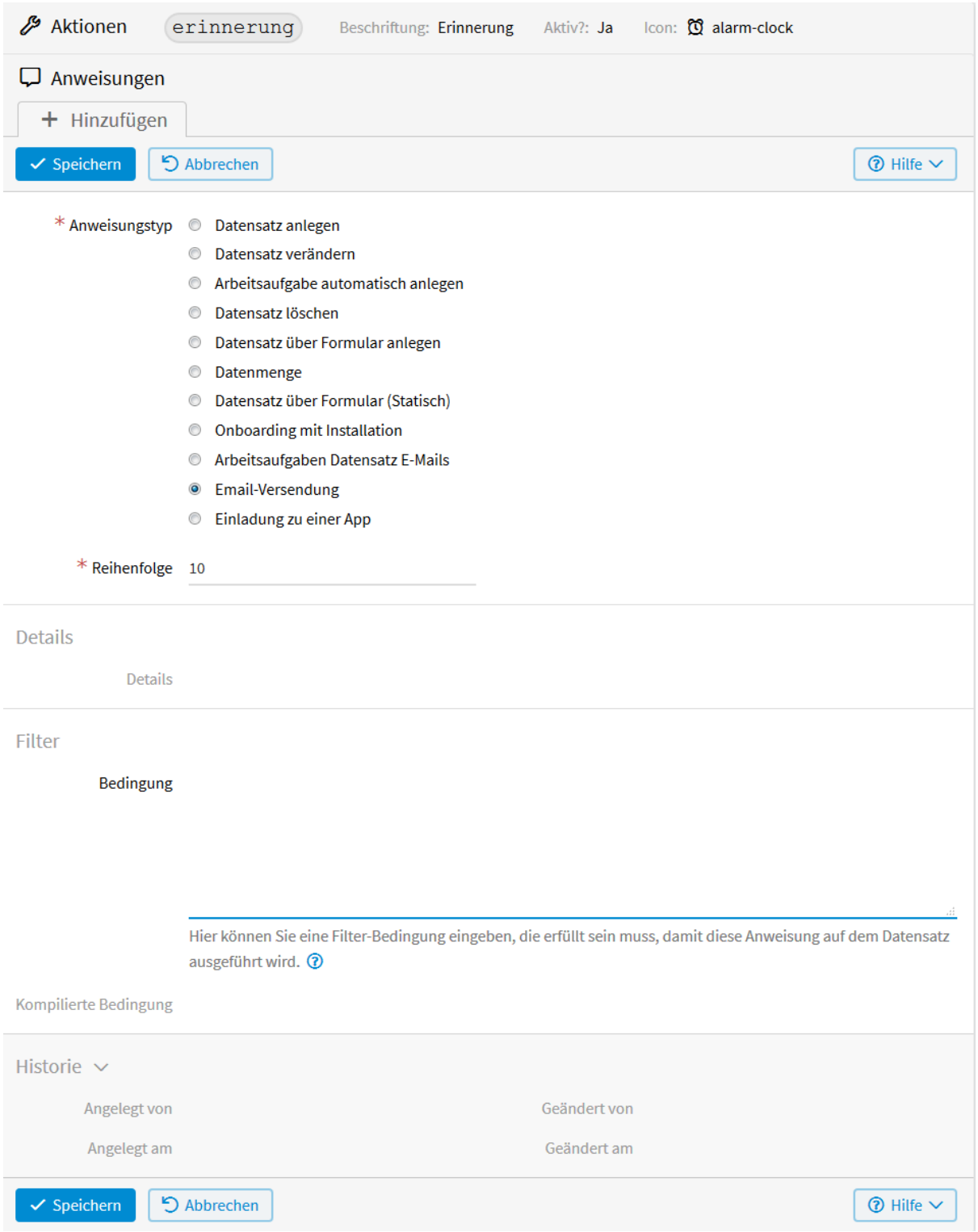


Abb. 145: Anweisungen - E-Mail-Versendung

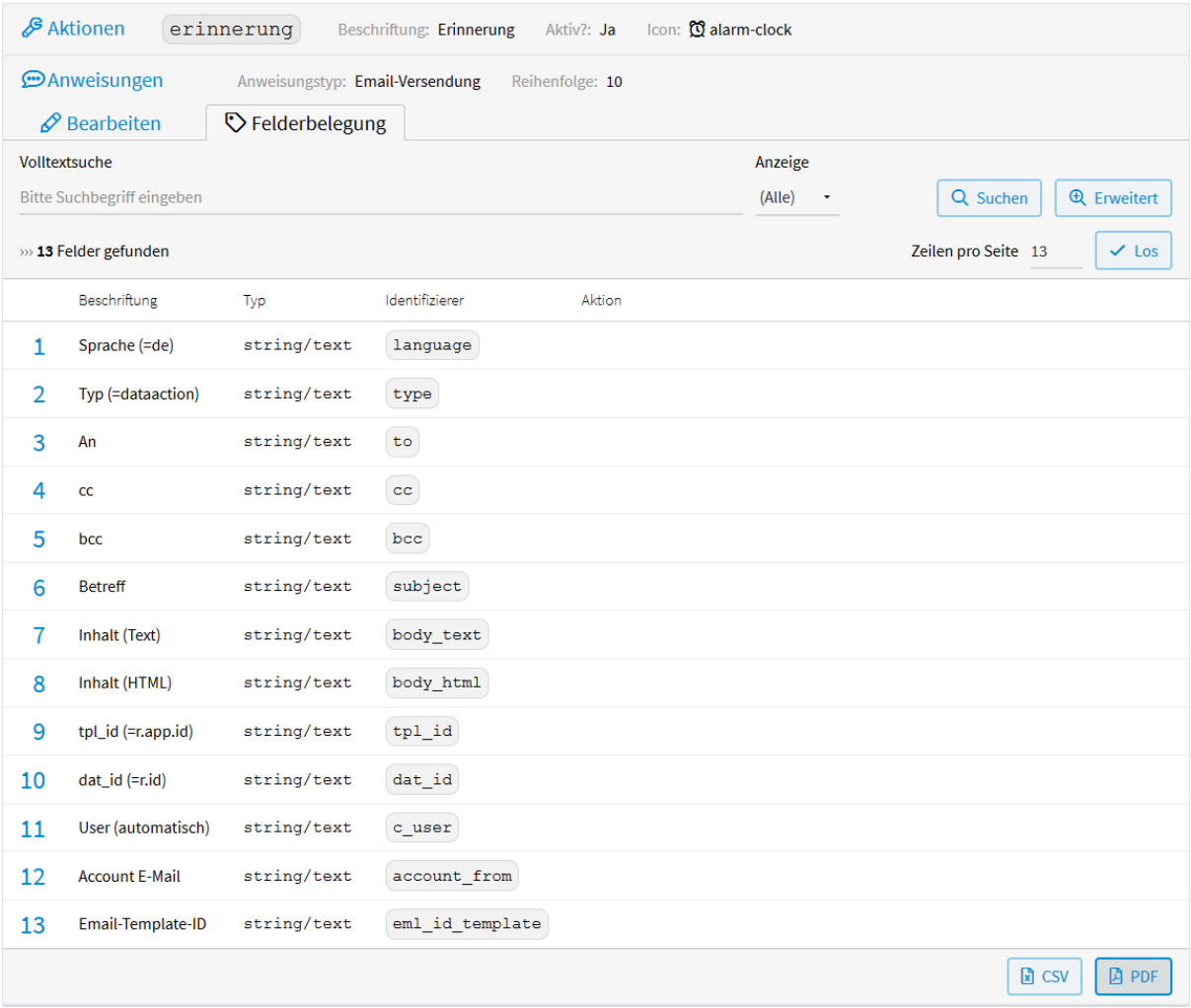


Abb. 146: Felderbelegung - E-Mail-Versendung

druck, können Sie auch hier auf die Felder Ihrer App zugreifen mit `r.v_identifier`. Wobei identifier für den Identifizierer des Feldes steht, auf das Sie zugreifen möchten.

Inhalt (Text)
Hier können sie den Inhalt der E-Mail als Text eingeben.

Inhalt (HTML)
Hier können Sie den Inhalt Ihrer E-Mail mit HTML gestalten.

tpl_id (=r.app.id)
Die App, von der aus die E-Mail verschickt wird. Hier muss als *Ausdruck* `r.app.id` eingegeben werden.

dat_id (=r.id)
Datensatz für den die E-Mail verschickt wird. Hier muss bei *Ausdruck* `r.id` eingegeben werden.

User (automatisch)
Ist immer der Benutzer der die Aktion durchführt (angemeldeter Benutzer) - kann nicht geändert werden.

Account E-Mail
Der bei LivingApps registrierte Account von dem „von“ (Name) und „Antwort an“ (E-Mail-Adresse) gefüllt werden soll. Geben Sie hier nichts ein, wird das Feld automatisch mit dem Namen und der E-Mail-Adresse von dem gefüllt, der die Aktion auslöst. Möchten Sie z. B. dass hier immer der Name und die E-Mail-Adresse des zuständige Mitarbeiters steht, können Sie bei *Feld auf Wert setzen* dessen E-Mail-Adresse eingeben, oder bei *Ausdruck* darauf zugreifen mit `r.v_identifier`. Wobei identifier für den Identifizierer des E-Mail-Feldes steht.

Email-Template-ID
Möchten Sie den E-Mail-Inhalt über ein E-Mail-Template einfügen, geben Sie hier die Email-Template-ID an. Diese erhalten Sie, wenn Sie in der Übersicht Ihrer E-Mail-Templates das entsprechende Template anklicken. Die ID besteht aus dem letzen Ziffern- und Zahlenblock, der hinter `entry=` der angezeigten *URL* steht. Bei Verwendung eines E-Mail-Templates werden die Felder *An Betreff* und *Account E-Mail* in der E-Mail von der Aktion übernommen. Der Inhalt der E-Mail wird vom E-Mail-Template übernommen. Über die Erstellung eines E-Mail-Templates können sie in der Dokumentation der erweiterten Funktionen von LivingApps unter *E-Mail-Templates* mehr lesen. Als *Typ* des E-Mail-Templates muss hierfür *Vorlage für Nachricht an alle Datensätze* gewählt werden.

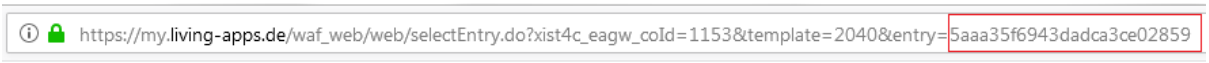


Abb. 147: Felderbelegung - E-Mail-Versendung - E-Mail-Template-ID

Durch Klicken auf das zu ändernde Feld, sind folgende Konfigurationen möglich.

Aktion

Nichts ausgewählt
Wird hier nichts ausgewählt, so wird mit dem Feld keine Aktion durchgeführt.

Feld leeren
Der Feldinhalt wird gelöscht. Das ist nur beim Ändern von Datensätzen sinnvoll.

Feld auf Wert setzen
Das Feld übernimmt, nachdem die Aktion ausgeführt wurde, den hier eingegebenen Wert.

Ausdruck
Für die Formulierung des Ausdrucks wird *vSQL* verwendet. Dafür werden die Identifizierer der Felder benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*. Auf die Werte kann dann zugegriffen werden mit: `r.v_identifier`.

Nach dem Speichern erscheint unterhalb des Feldes *Ausdruck* der kompilierte Ausdruck. Ist ein Fehler in der Formulierung, sehen Sie diesen hier.

In unserem Beispiel wurden folgende *Konfigurationen* vorgenommen.

***language* auf Wert von "de" setzen**
Sprache ist Deutsch.

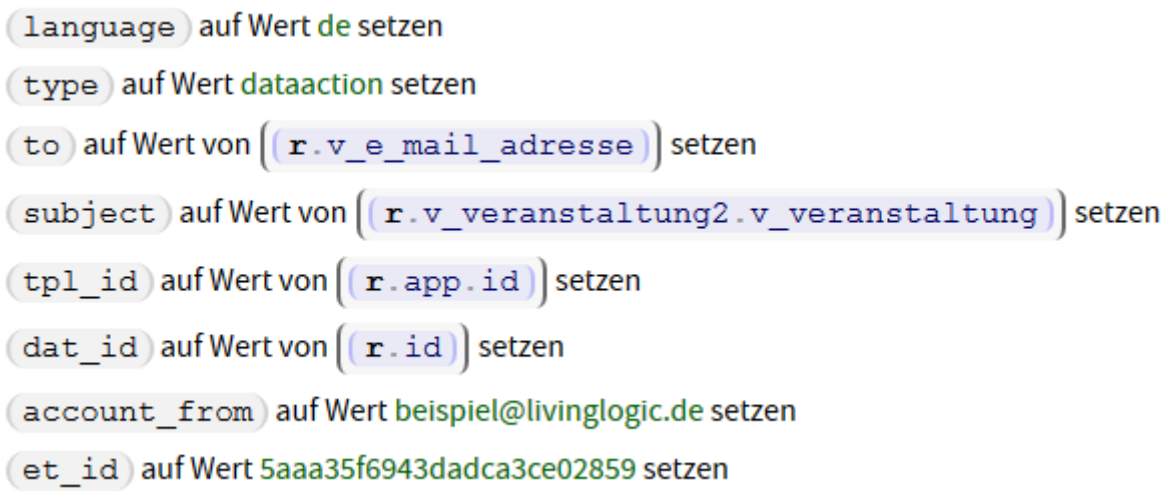


Abb. 148: Felderkonfiguration - E-Mail-Versendung

***type* auf Wert "dataaction" setzen**
Vorgegebener Wert

***to* auf Wert von r.v_e_mail_adresse setzen**
Empfänger der E-Mail. Greift auf die jeweilige E-Mail-Adresse der Teilnehmer zu.

***subject* auf Wert von "Erinnerung an " + r.v_veranstaltung2.v_veranstaltung setzen**
Schreibt in die Betreffzeile der E-Mail `Erinnerung an` und greift dann auf die jeweilige Veranstaltung zu, an die erinnert werden soll.

***tpl_id* auf Wert von r.app.id setzen**
Die App, von der aus die E-Mail verschickt wird. Hier muss als *Ausdruck* `r.app.id` eingegeben werden.

***dat_id* auf Wert von r.id setzen**
Datensatz für den die E-Mail verschickt wird. Hier muss bei *Ausdruck* `r.id` eingegeben werden.

***eml_id_template* auf Wert "5aaa35f6943dadca3ce02859" setzen**
Der E-Mail-Inhalt wird über ein E-Mail-Template eingefügt. Dafür muss hier die E-Mail-Template-ID als Wert eingegeben werden. Mehr dazu können Sie oben unter *E-Mail-Template-ID* nachlesen.

Ausgelöst werden kann die Aktion in der Liste des Datenmanagements der App *Anmeldung* unter *Daten* → *Liste*. Wobei die Aktion *Erinnerung* nur bei den Teilnehmern aktiv ist, die sich nicht wieder abgemeldet haben.

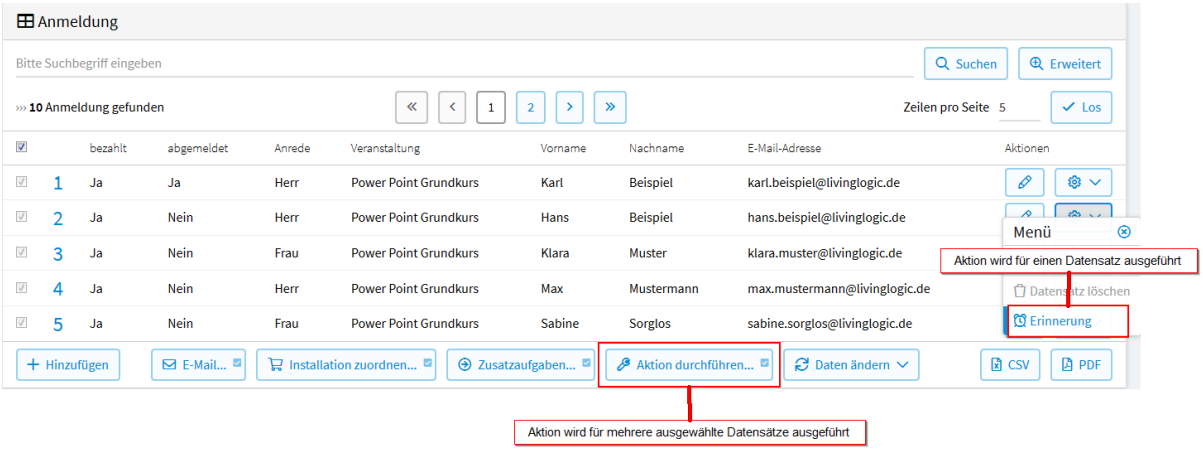


Abb. 149: Aktion „E-Mail-Versendung“ ausführen

Folgende E-Mail erhalten die Teilnehmer in unserem Beispiel nach Auslösen der Aktion:

```
Sehr geehrte Frau Sand,

wir möchten Sie erinnern, dass Sie sich zur Veranstaltung Word für Einsteiger
am 17.03.18 von 10:00 bis 16:00 Uhr,
in Bamberg, Kulturzentrum, Raum 105,
angemeldet haben.

Bei Rückfragen erreichen Sie uns unter 0951/123456789.

Mit freundlichen Grüßen
Ihr Veranstaltungsteam
```

Einladung zu einer App

Im folgenden Anwendungsbeispiel sollen die Personen in der App „Mitarbeiter“ zur Mitarbeit an den Apps „Anmeldung“ und „Veranstaltungen“ eingeladen werden. Der Vorteil dies über eine Aktion zu tun ist der, dass man nicht jeden einzeln einladen muss, sondern alle auf einmal, auch zu mehreren Apps, einladen kann.

Wählen Sie im Menü *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Aktionen*. Klicken Sie auf *Hinzufügen* um eine neue Aktion zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Nach dem Speichern erscheint neben *Bearbeiten* der Tab *Anweisungen*. Nach *Hinzufügen* einer neuen Anweisung kann im nun geöffneten Fenster eingestellt werden, was durch die Aktion ausgeführt werden soll. Siehe *Anweisungen*.

Als *Anweisungstyp* wird *Einladung zu einer App* gewählt und bei *Reihenfolge* `10` eingegeben.

Nach dem Speichern der Anweisungen erscheint neben *Bearbeiten* der Tab *Felderbelegung*. Durch Klicken darauf sehen Sie folgende *Felder*. Diese sind bei jeder App gleich.

User-ID
: Dieses Feld wird automatisch mit dem angemeldete Benutzer gefüllt.

App-ID
: Wenn Sie eine App anklicken, sehen Sie in der URL hinter `apps/` eine Kombination aus Buchstaben und Zahlen:

  Dies ist die *App-UUID* (Universally Unique Identifier), der eindeutige Identifizierer Ihrer App. Die App-ID der App, zu der Sie zur Mitarbeit einladen wollen, muss hier angegeben werden. Bei der Einladung zu mehreren Apps geben Sie die App-IDs kommagetrennt ein.

Vorname
: Vorname der Person, die Sie einladen wollen. Wählen Sie *Ausdruck* und greifen mit `r.v_vorname` auf die Werte des Feldes `Vorname` zu. Wobei `vorname` für den Identifizierer des Feldes `Vorname` steht.

Nachname
: Nachname der Person, die Sie einladen wollen. Einen gleichbleibenden Wert (z. B. Mustermann) geben Sie bei *Feld auf Wert setzen* ein. Wählen Sie *Ausdruck*, können Sie auch hier auf die Felder Ihrer App zugreifen mit `r.v_nachname`. Wobei `nachname` für den Identifizierer des Feldes `Nachname` steht.

E-Mail/Account
: E-Mail-Adresse der Person, die Sie einladen wollen. Wählen Sie *Ausdruck* und greifen mit `r.v_email` auf die Werte des E-Mail-Feldes Ihrer App zu. Wobei `email` für den Identifizierer des E-Mail-Feldes steht.

Berechtigungslevel (2-5)
: Geben Sie hier an, welche Berechtigung die Mitarbeiter an der App haben sollen. Wobei `2` für *Minimal*, `3` für *Standard*, `4` für *Datenverarbeitung* und `5` für *Administrator* steht.

Einzelberechtigungen als String
: Möchten Sie die App-Berechtigungen nicht als Rolle, sondern benutzerdefiniert vergeben, können Sie dies hier tun. Dafür wird jede einzelne Berechtigung jeweils mit `0` „nicht gewählt“ und mit `1` „gewählt“.

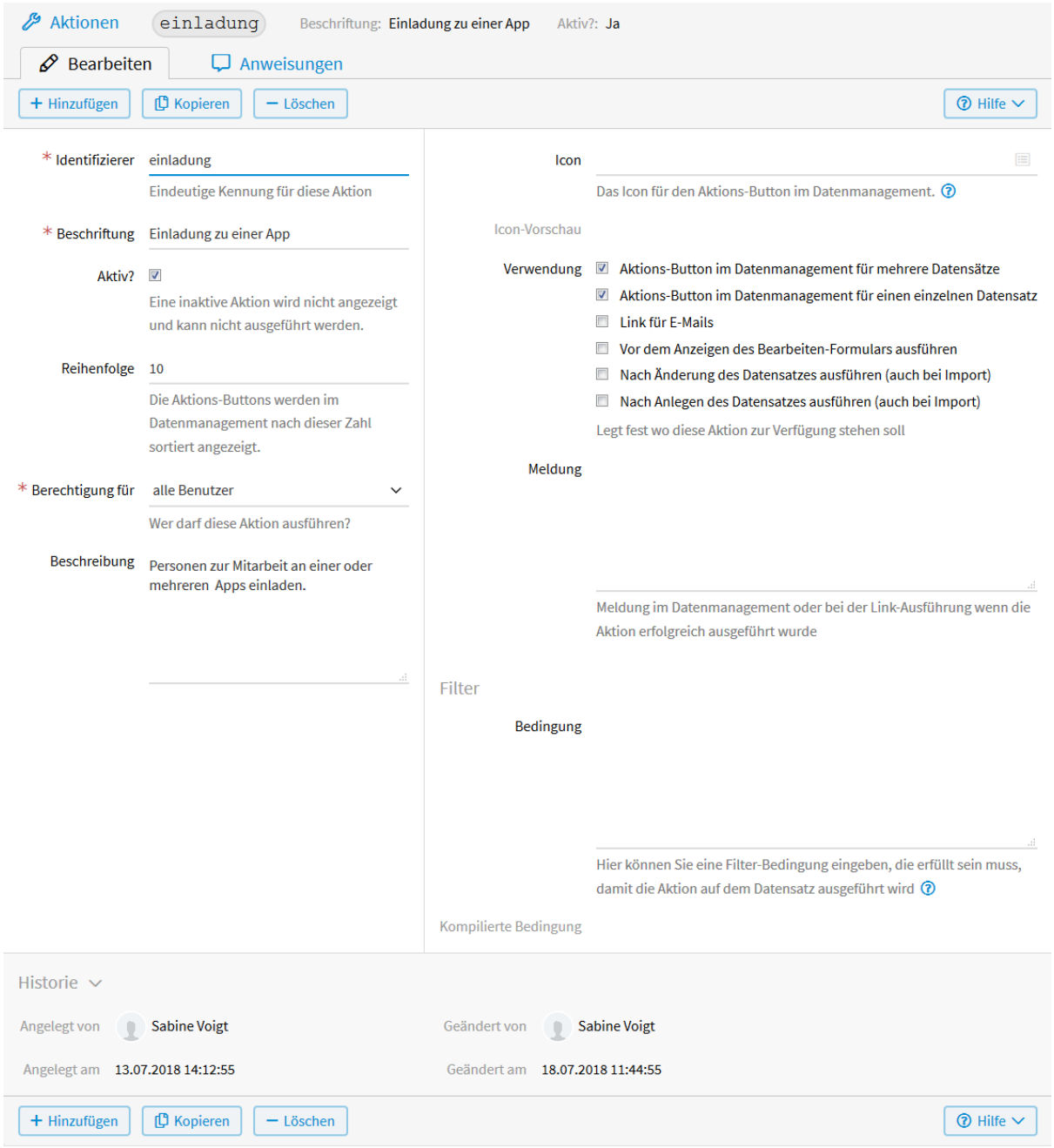


Abb. 150: Einladung zu einer App

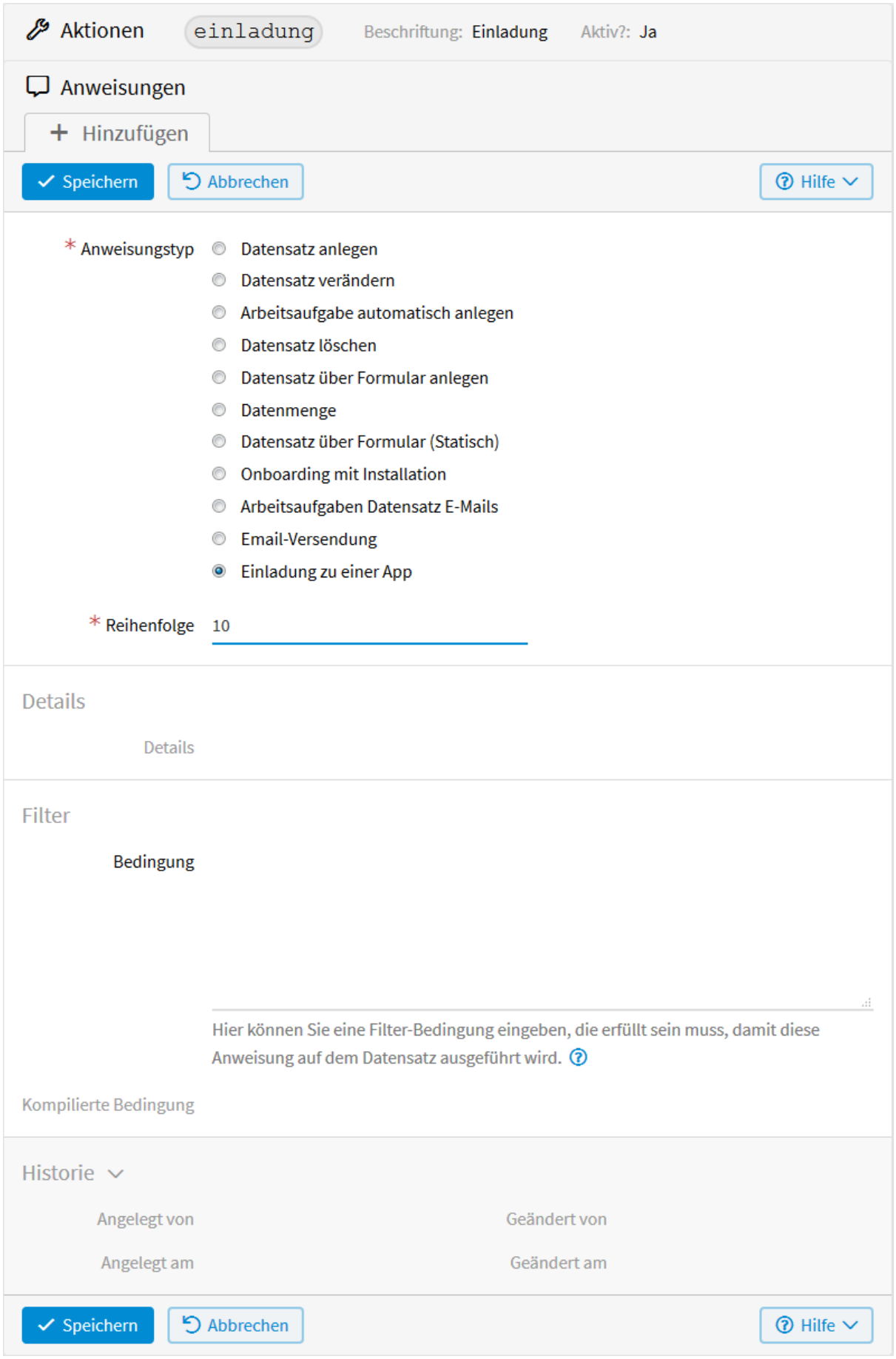


Abb. 151: Anweisungen - „Einladung zu einer App“

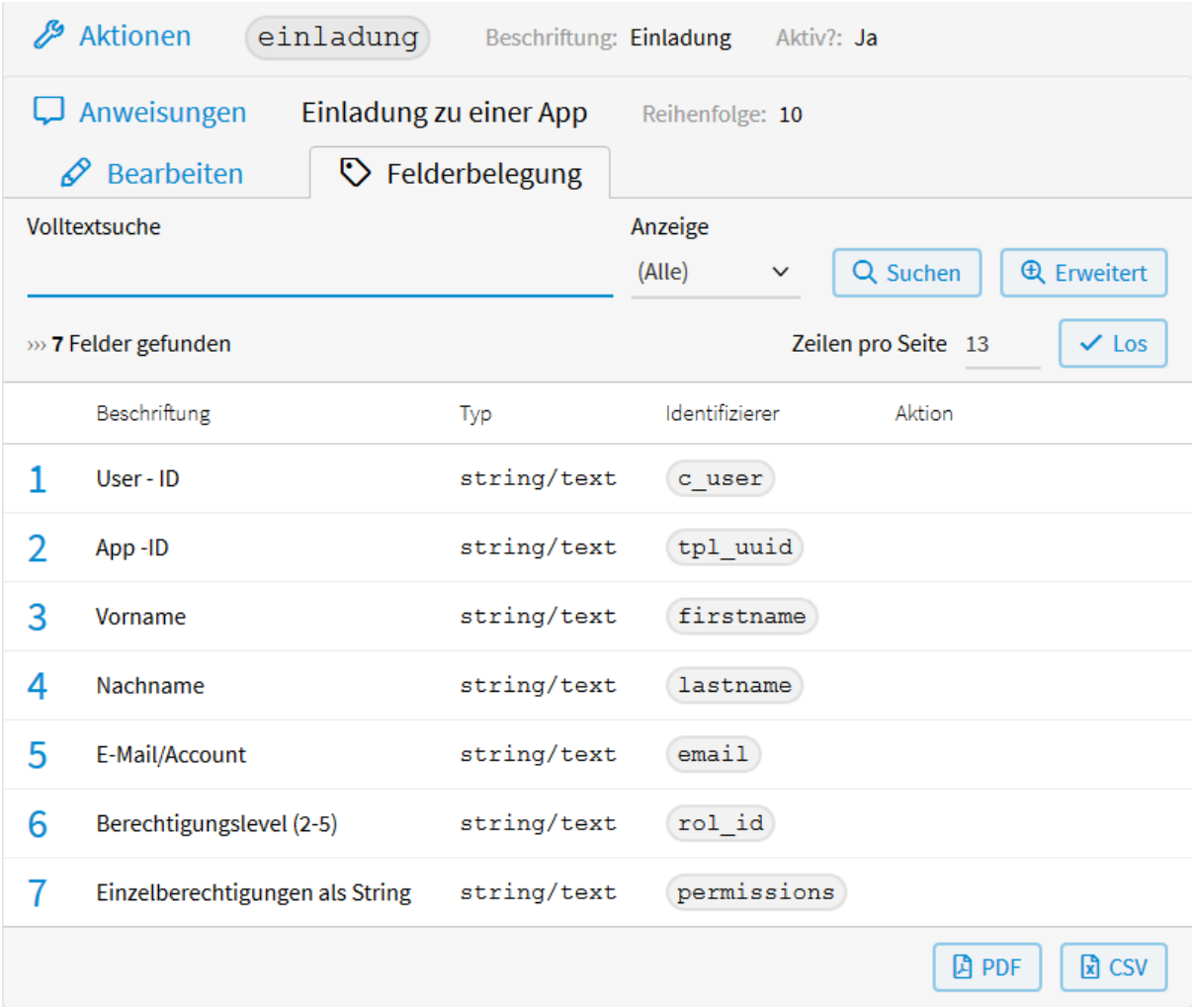


Abb. 152: Felderbelegung - „Einladung zu einer App“

Abb. 153: App-ID

Applikationsrechte

#	Name	Berechtigung	Erlaubt
1	app_edit	Dateneingabe anpassen	den Formularwizard aufzurufen
2	app_konfig	Datenmanagement konfigurieren	Interne Felder definieren, Anzeige im Datenmanagement zu konfigurieren
3	app_delete	Applikation löschen	
4	app_taskmanage	Arbeitsaufgaben konfigurieren	Aufgabenwizard aufrufen, zu Aufgaben einladen
5	app_data_edit	Datenmanagement anzeigen	Daten sichten und editieren im Datenmanagement
6	app_data_view	Daten sichten	Daten sichten im Datenmanagement
7	app_taskview	Arbeitsaufgaben anzeigen	sichten/bearbeiten, der meinen Aufgaben zugeordneten Datensätze
8	app_frontend	Formular aufrufen	
9	app_dataconnectex	nicht mehr relevant!	
10	app_performevalua	Auswertung anzeigen	
11	app_tocatalog	Im Katalog veröffentlichen	
12	app_dataimexport	Daten importieren	
13	app_mydata_edit	meine Daten editieren	nach Login, die von mir erfassten Daten editieren
14	app_mydata_view	meine Daten sichten	nach Login, die von mir erfassten Daten ansehen
15	app_user_admin	Berechtigungen zuweisen	

Um z. B. die Einzelberechtigungen nur für `app_mydata_edit` und `app_mydata_view` zuzuweisen, müssten Sie bei *Feld auf Wert setzen* folgende Angaben machen:

```
0000000000000110
```

Durch Klicken auf das zu ändernde Feld, sind folgende Konfigurationen möglich.

Aktion

Nichts ausgewählt
Wird hier nichts ausgewählt, so wird mit dem Feld keine Aktion durchgeführt.

Feld leeren
Der Feldinhalt wird gelöscht. Das ist nur beim Ändern von Datensätzen sinnvoll.

Feld auf Wert setzen
Das Feld übernimmt, nachdem die Aktion ausgeführt wurde, den hier eingegebenen Wert.

Ausdruck
Für die Formulierung des Ausdrucks wird *vSQL* verwendet. Dafür werden die Identifizierer der Felder benötigt. Diese finden Sie in der linken Menüleiste unter *Felder*. Auf die Werte kann dann zugegriffen werden mit: `r.v_identifier`.

Nach dem Speichern erscheint unterhalb des Feldes *Ausdruck* der kompilierte Ausdruck. Ist ein Fehler in der Formulierung, sehen Sie diesen hier.

In unserem Beispiel wurden folgende *Konfigurationen* vorgenommen.

***tpl_uuid* auf Wert „da11219bb494aff28062f50f, a6df949681835742ab27a967“ setzen**
App-IDs der Apps „Anmeldung und Veranstaltung“

***firstname* auf Wert von *r.v_vorname* setzen**
Greift auf den Vornamen des eingeladenen Mitarbeiters zu.

***lastname* auf Wert von *r.v_nachname2* setzen**
Greift auf den Nachnamen des eingeladenen Mitarbeiters zu.

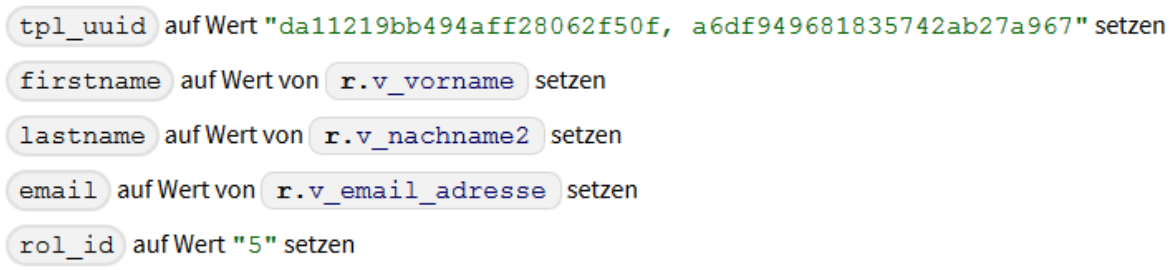


Abb. 154: Felderkonfiguration - „Einladung zu einer App“

***email* auf Wert von *r.v_email_adresse* setzen**
Greift auf die E-Mail-Adresse des eingeladenen Mitarbeiters zu.

***rol_id* auf Wert „5“ setzen**
Weist den eingeladenen Mitarbeitern die Rolle als Administrator zu.

Ausgelöst werden kann die Aktion in der Liste des Datenmanagements der App *Mitarbeiter* unter *Daten* → *Liste*.

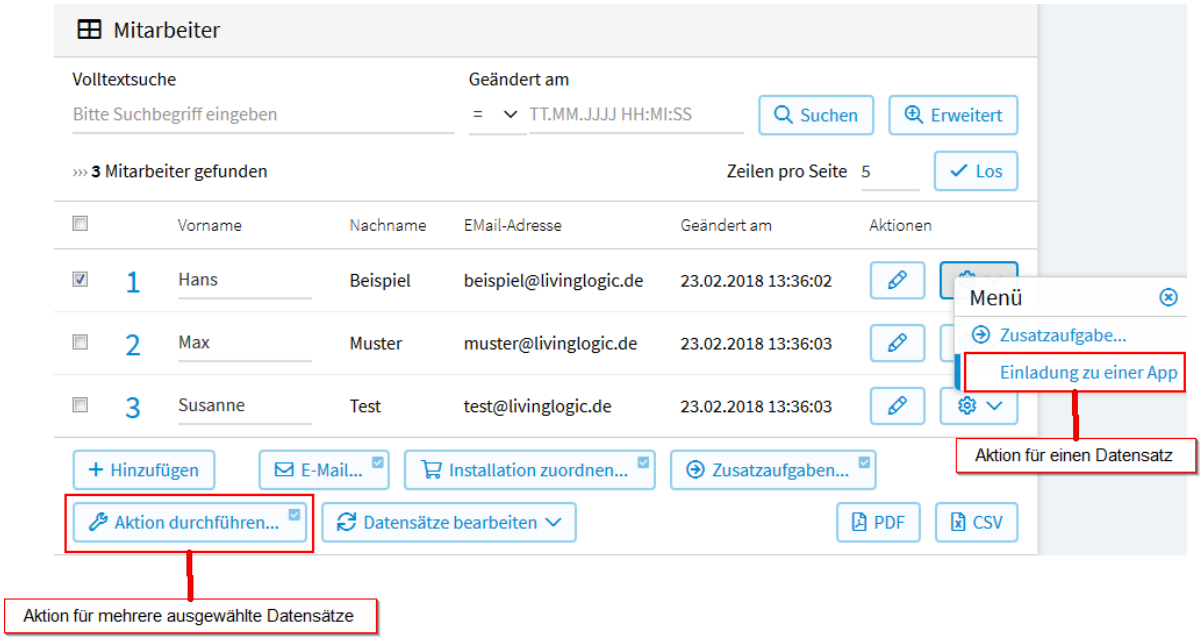


Abb. 155: Aktion „Einladung zu einer App“ ausführen

2.17 Startlink

In *Startlink* können Sie festlegen zu welcher Seite der Benutzer kommt, wenn er diese Applikation in der App-Übersichtsseite anklickt.

2.17.1 Erstellung eines Startlinks

Wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Startlink*. Im nun geöffneten Bereich kann ein Startlink neu angelegt, geändert oder zurückgenommen werden. Siehe *Startlink erstellen*

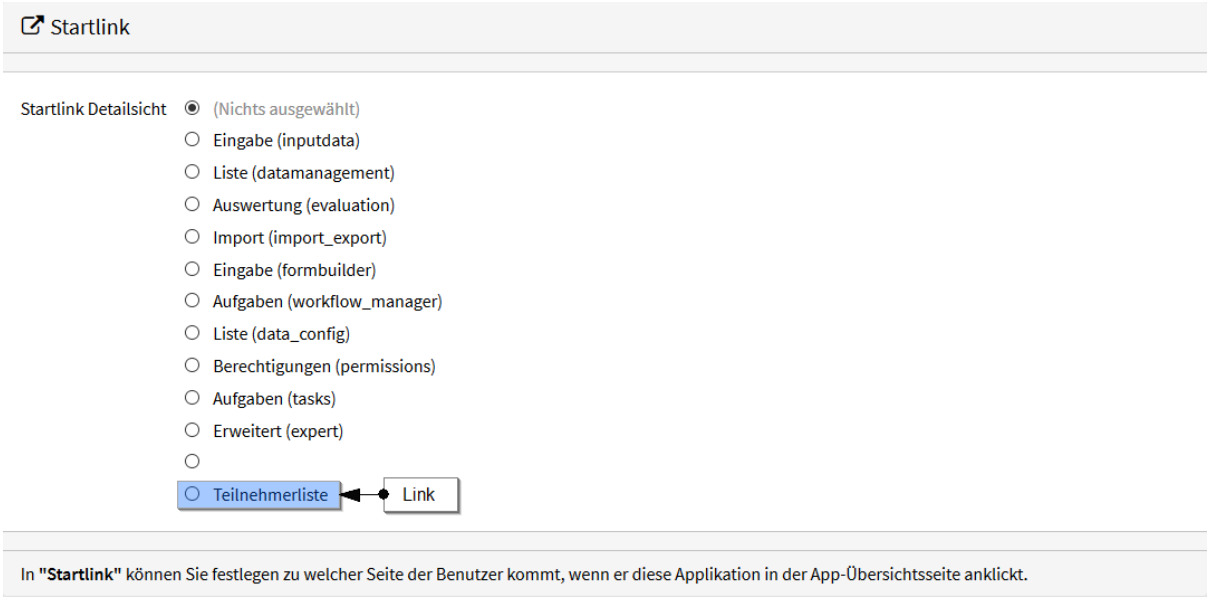


Abb. 156: Startlink erstellen

Eingabe (inputdata)
: Setzt die eingebettete Dateneingabe als Startlink. *Daten* → *Eingabe*

Liste (datamanagement)
: Setzt die Datenliste als Startlink. *Daten* → *Liste*

Auswertung (evaluation)
: Setzt die automatisch bereitgestellten Auswertungen der Daten als Startlink. *Daten* → *Auswertung*

Import (import_export)
: Setzt die Import-Seite als Startlink. *Daten* → *Import*

Eingabe (formbuilder)
: Setzt die Konfiguration der Eingabe (den Formbuilder) als Startlink. *Konfiguration* → *Eingabe*

Aufgaben (workflow_manager)
: Setzt die Konfiguration der Arbeitsaufgaben als Startlink. *Konfiguration* → *Aufgaben*

Liste (data_config)
: Setzt die Konfiguration der Datenliste als Startlink. *Konfiguration* → *Liste*

Berechtigungen (permissions)
: Setzt die Konfiguration der Berechtigungen als Startlink. *Konfiguration* → *Berechtigungen*

Aufgaben (tasks)
: Setzt die Aufgaben-Seite als Startlink. Nur möglich, wenn Aufgaben für diese App angelegt sind. *Aufgaben*

Erweitert (expert)
: Setzt die Erweitert-Konfiguration als Startlink. *Konfiguration* → *Erweitert*

Links
: Sofern in der jeweiligen App Links konfiguriert sind, können diese ebenfalls als Startlink verwendet werden.

Wählen Sie einen beliebigen *Startlink* aus und klicken Sie auf *Speichern*. Testen Sie den Startlink, indem Sie über die obere Navigation zurück zu *Meine Apps* gehen und die eben konfigurierte App durch einen Klick auf das Icon öffnen.

2.18 Zugewiesene Daten

In *Zugewiesene Daten* können Sie festlegen, welche Bedingung erfüllt sein muß, damit ein Datensatz einem Benutzer zugewiesen ist. Wenn Sie hier keine Bedingung eingeben, wird standardmäßig `r.createdby.id == user.id` verwendet, d.h. der Datensatz ist dem Benutzer zugewiesen, wenn er ihn angelegt hat.

2.18.1 Erstellung

Wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Zugewiesene Daten*. Im nun geöffneten Bereich können Sie *die Bedingung eingeben*, die erfüllt sein muss, dass ein Datensatz einem Benutzer zugewiesen wird, auch wenn er ihn nicht angelegt hat. Für das *Formulieren der Bedingung* wird *vSQL* verwendet. Es stehen die *Variablen* `r` (zu testender Datensatz), `app` (die App zu der diese Konfiguration gehört) und `user` (eingeloggter Benutzer) zur Verfügung.

So können Sie z. B. in einer App „Veranstaltungen“, die zuständigen Mitarbeiter zu den Veranstaltungen im Feld „E-Mail-Verteiler“ erfassen. Um zu erreichen, dass die Mitarbeiter nur die Datensätze zu den Veranstaltungen sehen und bearbeiten können, denen sie zugewiesen sind, muss folgende Bedingung eingegeben werden:

```
r.v_e_mail_verteiler == user.email
```

Wobei `e_mail_verteiler` für den Identifizierer des E-Mail-Feldes steht. `user.email` ist die E-Mail-Adresse des eingeloggten Benutzers.

2.19 Uploads

In *Uploads* können Sie festlegen, ob der Benutzer eingeloggt sein und Zugriffsrechte auf die App besitzen muss, um Dateien, die in dieser App hochgeladen wurden, downloaden zu können, oder ob die Dateien frei zugänglich sein sollen.

2.19.1 Erstellung

Wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Uploads*.

Öffentlich
: Wählen Sie *Öffentlich*, wenn Ihre Dateien frei zugänglich sein sollen. Das heißt, Ihre Dateien können von jedem jederzeit heruntergeladen werden. Dies ist z.B. dann sinnvoll, wenn Sie ein Bild in einer Email (oder einem öffentlichen Anzeige-Template) verwenden wollen.

Nur App-Benutzer
: Hier können die Dateien von einem eingeloggten Benutzer heruntergeladen werden, wenn er Zugriffsrechte auf der App besitzt. Dies ist dann sinnvoll, wenn alle Benutzer dieser App gemeinsam die Datensätze der App bearbeiten und hochgeladene Dokumente einsehen können sollen, aber diese vor Zugriff von Dritten geschützt sein sollen (z. B. Bewerbungsunterlagen).

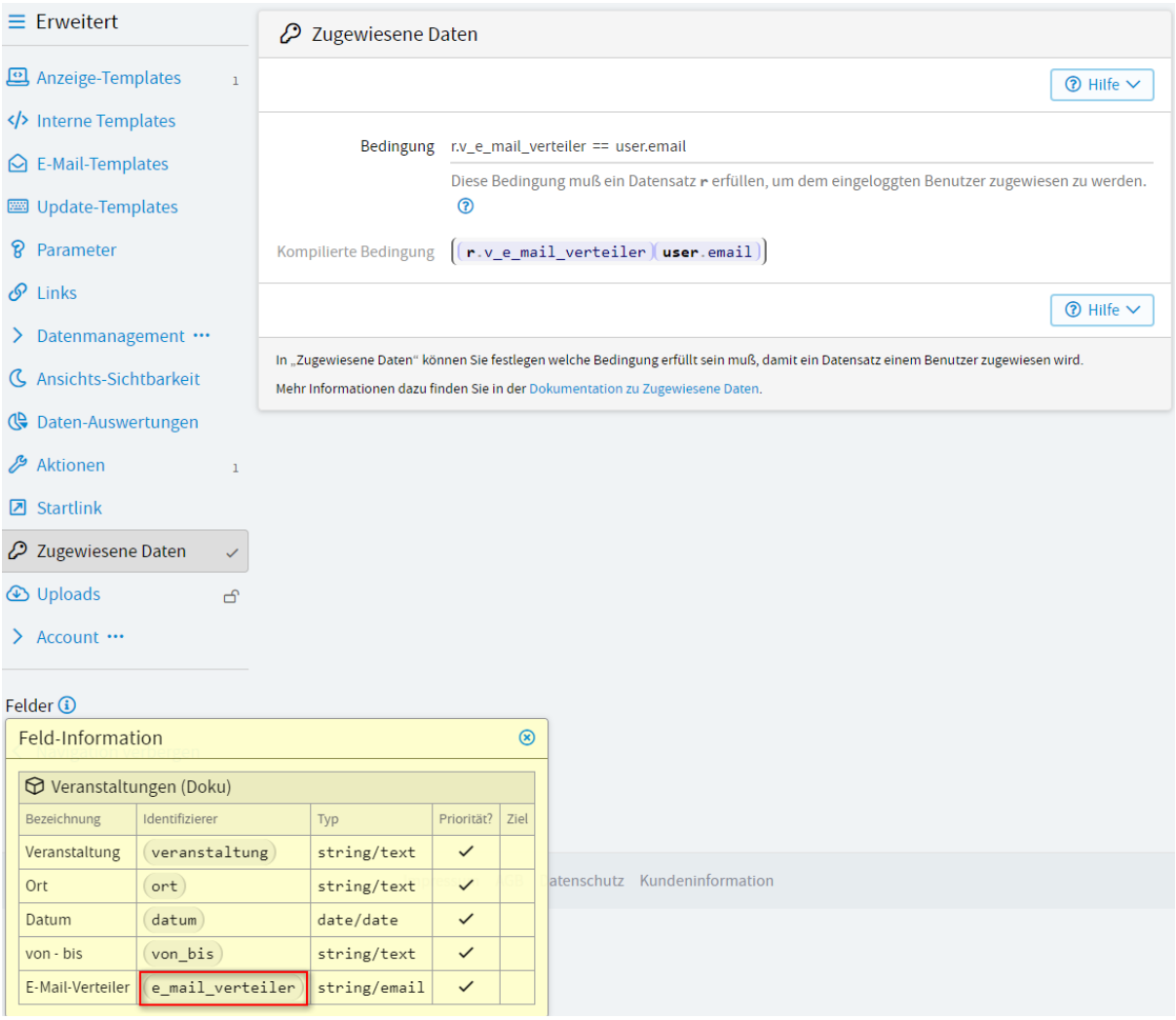


Abb. 157: Zugewiesene Daten - Bedingung

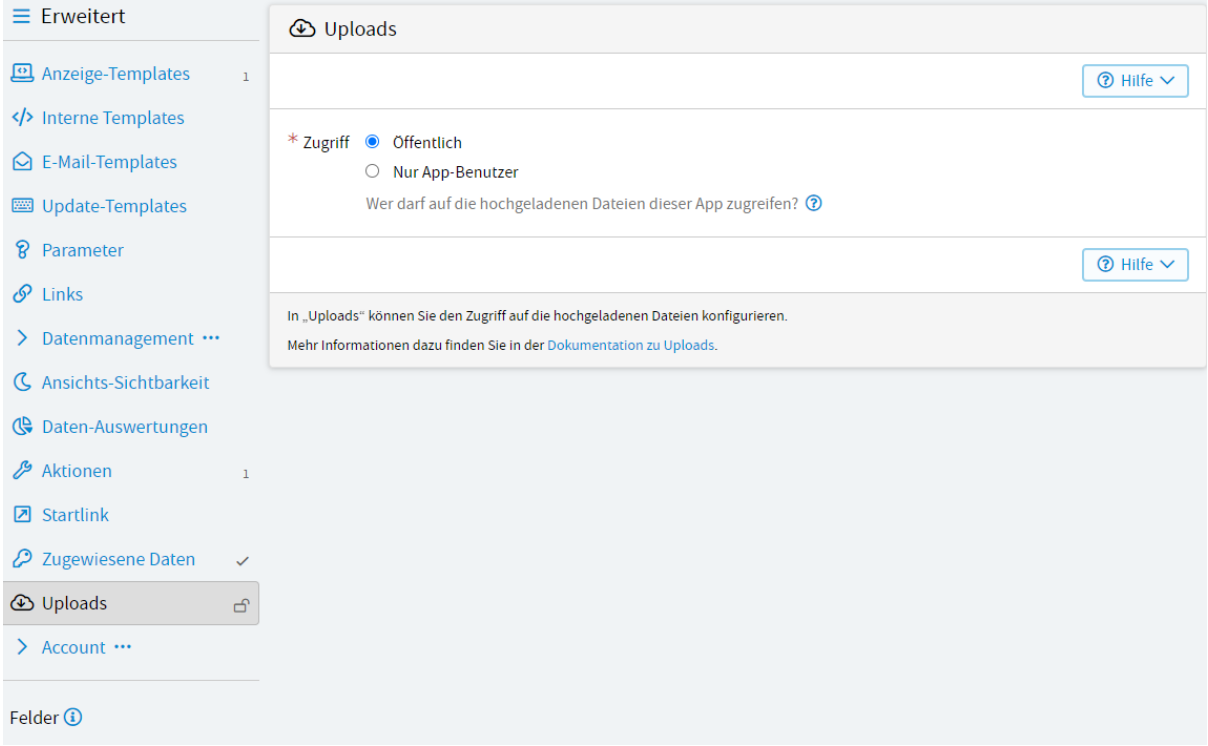


Abb. 158: Uploads

Desweiteren kann eine Datei über die LivingAPI-Funktion *globals.response.send_file()* ausgeliefert werden. Dies ermöglicht es, jeden beliebigen Zugriffsschutz für nicht-eingeloggte oder andere Benutzer im Anzeige-Template selbst zu implementieren.

2.20 Account

Unter *Account* können Konfigurationen vorgenommen werden, die nicht an die jeweilige App gebunden sind, sondern für Ihr gesamtes LivingApps-Konto gelten.

Um *Einstellungen des Accounts* vorzunehmen, wählen Sie im Menü einer beliebigen App *Konfiguration* → *Erweitert* und anschließend in der linken Menüleiste *Account*. Wählen Sie dann den jeweiligen Menüpunkt aus.

2.20.1 Benutzer-Menüs

Die erweiterte Funktion *Benutzer-Menüs* ermöglicht es Ihnen im Menü auf beliebige Webseiten zu verlinken. Im Gegensatz zu *App-Menüs* werden *Benutzer-Menüs* in allen Ihren Apps angezeigt (es sei denn, Sie ordnen das Menü einer bestimmten App zu). Außerdem können diese Menüs auch auf der *App-Übersichtsseite?* platziert werden.

Haben Sie beispielsweise Auswertungen mittels der Anzeige-Templates erstellt, so können Sie auf diesem Wege einfach und schnell auf diese zugreifen (siehe *Link zu einem Anzeige-Template*).

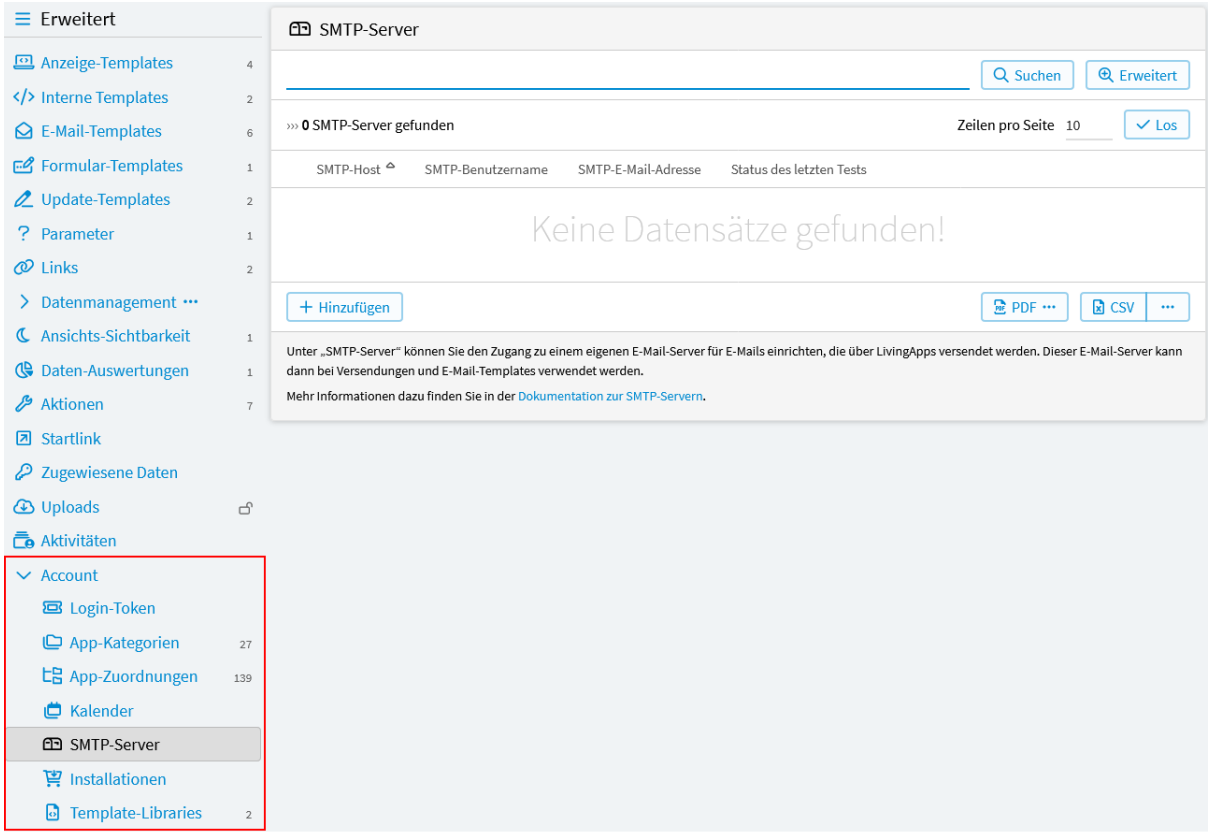


Abb. 159: Account

Erstellung von Benutzer-Menüs

Wählen Sie im Menü *Konfiguration → Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Account* und *Benutzer-Menüs* und dann auf *Neuen Link oder neues Menü zur Menüleiste hinzufügen*.

Im nun geöffneten Fenster können folgende Konfigurationen zur *Erstellung eines Links* vorgenommen werden.

LivingApp
: Die App in deren Kontext dieses Menü angezeigt werden soll. Wenn Sie hier keine App auswählen, wird dieses Menü bei allen Ihrer Apps dargestellt.

Identifizierer
: Der eindeutige Name des Links. Der Name darf nur Buchstaben, Ziffern und „_“ beinhalten.

Beschriftung
: Der Link-Text.

Icon
: Das Icon für den Link. Verwendet werden können alle Namen aus dem FontAwesome-Icon-Set.

Anzeigen auf
: Gibt an, wo das Menü erscheint. Der Link kann sowohl auf der *App-Übersichtsseite?*, *App-Startseite?*, *Eingabeformular?*, *iframe-Seite?* und *eigener Übersichtsseite?* erscheinen.

Position in der Menüleiste
: Hier kann, wenn ein Untermenü angelegt werden soll, mit *Übergeordnetes Menü* ein Menü ausgewählt werden. (Wenn Sie in der Menü-Liste den Button *Neues Menü* bei einem vorhandenen Menü anklicken, so wird dieses Menü bei *Übergeordnetes Menü* vorausgewählt).

  Mit *Reihenfolge* kann die Reihenfolge bestimmt werden, in der Menüs der gleichen Ebene sortiert werden. Je grösser die Zahl, desto weiter hinten erscheint der Menüpunkt.

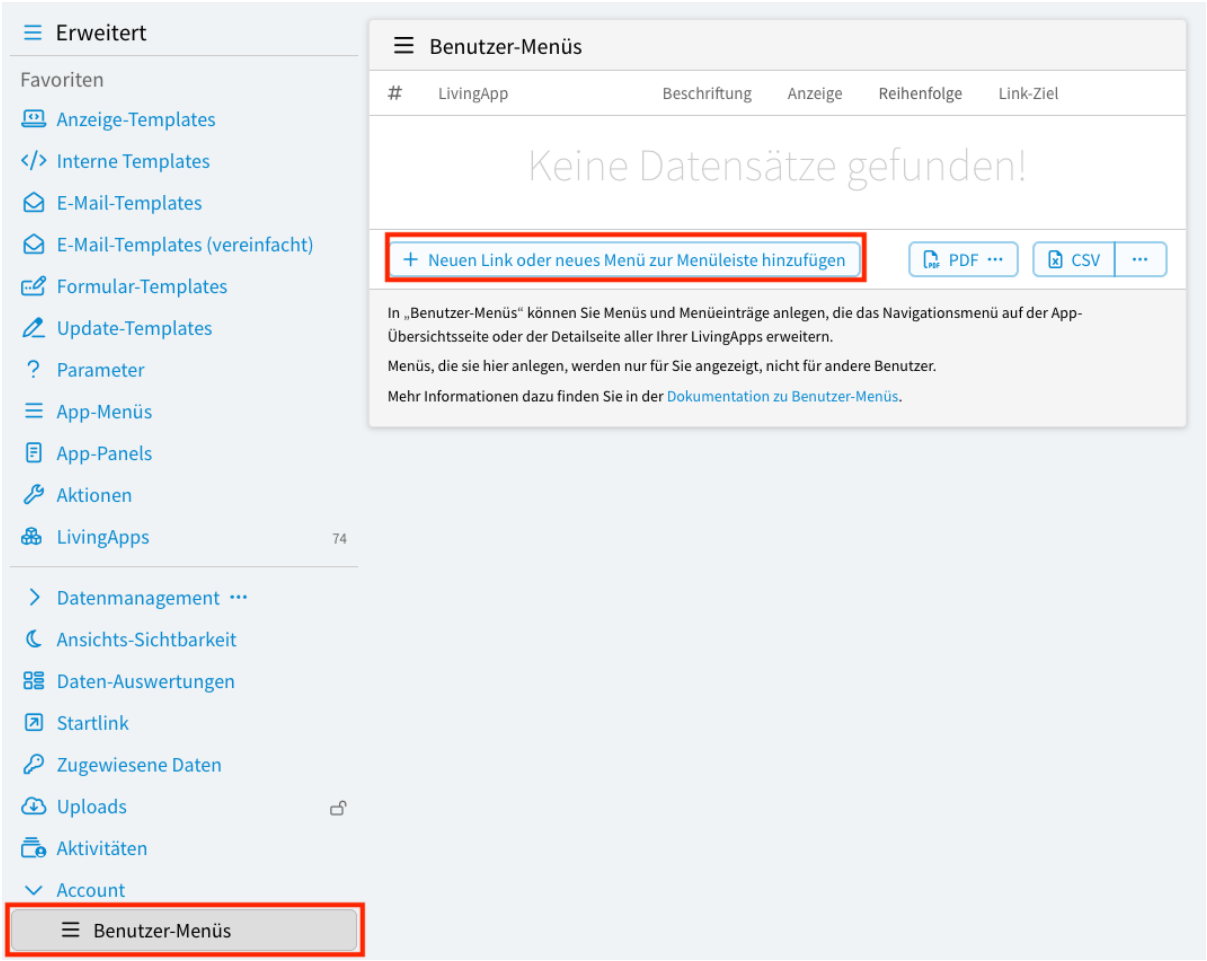


Abb. 160: Hinzufügen eines Menü-Links

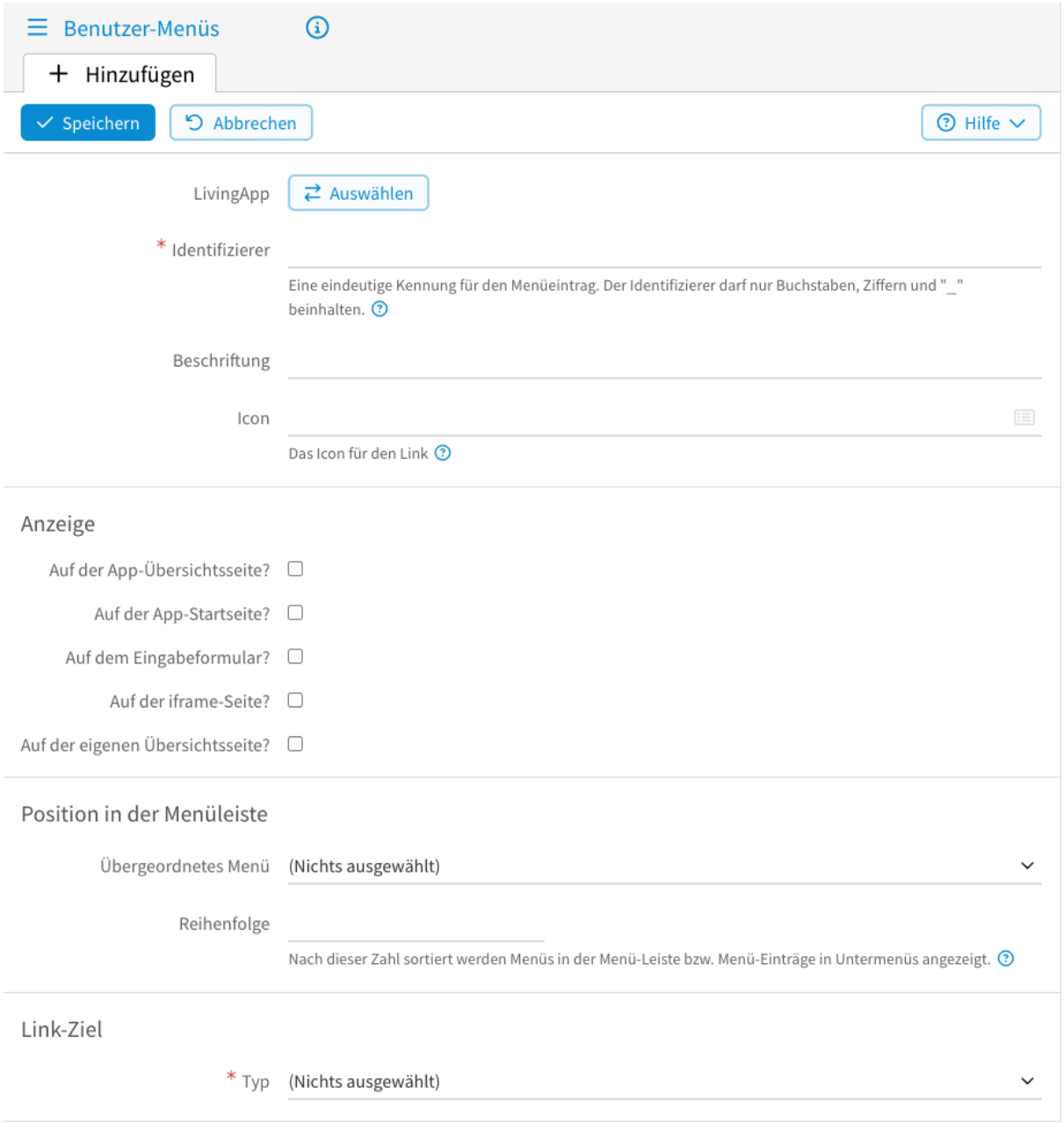


Abb. 161: Erstellung eines Menüs

Link-Ziel
: Hier kann aus verschiedenen Zielen ausgewählt werden, auf die ein Link zeigen kann. Bei den Zielen *Eingabeformular (öffentlich)*, *Eingabeformular (einebettet)*, *Daten-Management*, *Eigene Übersichtsseite*, *Auswertung*, *Import/Export*, *Arbeitsaufgaben*, *Konfiguration: Eingabe*, *Konfiguration: Arbeitsaufgaben*, *Konfiguration: Daten*, *Konfiguration: Berechtigungen*, *Konfiguration: Erweitert*, *Anzeige-Template* und *Daten-Auswertung* wird eine Auswahlmöglichkeit für den jeweiligen Typ angeboten. Bei *Eigener Link* kann eine beliebige URL angegeben werden. Der letzte Punkt *Kein Link* muss angewählt werden, wenn ein Menüeintrag erstellt werden soll, an dem ein Untermenü hängt.

Link-Attribute
: In diesem Abschnitt können die HTML-Attribute *Title*, *Target* und *Class* festgelegt werden.

Zeitsteuerung
: Hier kann festgelegt werden in welchem Zeitraum das Menü angezeigt werden soll. Mit *Aktivierung* legt man den Zeitpunkt fest, ab dem der Link angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link sofort aktiviert. Mit *Deaktivierung* kann angegeben werden ab wann der Link nicht mehr angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link nicht automatisch deaktiviert.

Anwendungsbeispiel

Um die Funktionsweise der Links zu veranschaulichen wird im Folgenden wieder auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen.

Link zu einem Anzeige-Template

Zuerst soll ein Link als *Auf der App-Übersichtsseite?* erstellt werden, der zur Teilnehmerliste führt, die mittels eines *Anzeige-Templates* angelegt wurde.

Wählen Sie dafür im Menü *Konfiguration* → *Erweitert* und klicken anschließend in der linken Menüleiste auf *Acccount*, auf *Benutzer-Menüs* und dann auf *Neuen Link oder neues Menü zur Menüleiste hinzufügen* um einen neuen Link zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Dabei wird insbesondere bei *Link-Ziel* der Typ *Kein Link* ausgewählt.

Anschliessend ist in der Menü-Liste der neue Datensatz mit der Beschriftung *Link* zu sehen, bei dem der Button *Neuer Menüpunkt* angeboten wird.

Nach dem Klick auf *Neuer Menüpunkt* ist im Abschnitt *Position in der Menüleiste* das Feld *Übergeordnetes Menü* bereits mit *Links* vorbelegt.

Dann wird im Abschnitt *Link-Ziel* bei *Typ* der Wert *Anzeige-Template* ausgewählt.

Danach kann mit einem Klick auf den Button *Auswählen* ein Anzeige-Template gewählt werden:

Nach dem Abspeichern erscheint *Links* als *Menüpunkt in der App-Übersichtseite*.

2.20.2 Benutzer-Panels

Die erweiterte Funktion *Benutzer-Panels* ermöglicht es Ihnen auf verschiedenen Seiten Kacheln mit z.B. Hinweistexten oder Links zu platzieren. Im Gegensatz zu *App-Panels* werden *Benutzer-Panels* in allen Ihren Apps angezeigt (es sei denn, Sie ordnen das Panel einer bestimmten App zu). Außerdem können diese Panels auch auf der *App-Übersichtsseite?* platziert werden.

Z.B. Sie erstellen einen Link und nutzen diesen als Textelement, das z. B. Erläuterungen zum Formular enthält und neben der Dateneingabe anzeigt wird (siehe *Link zum Anlegen von Textelementen*).

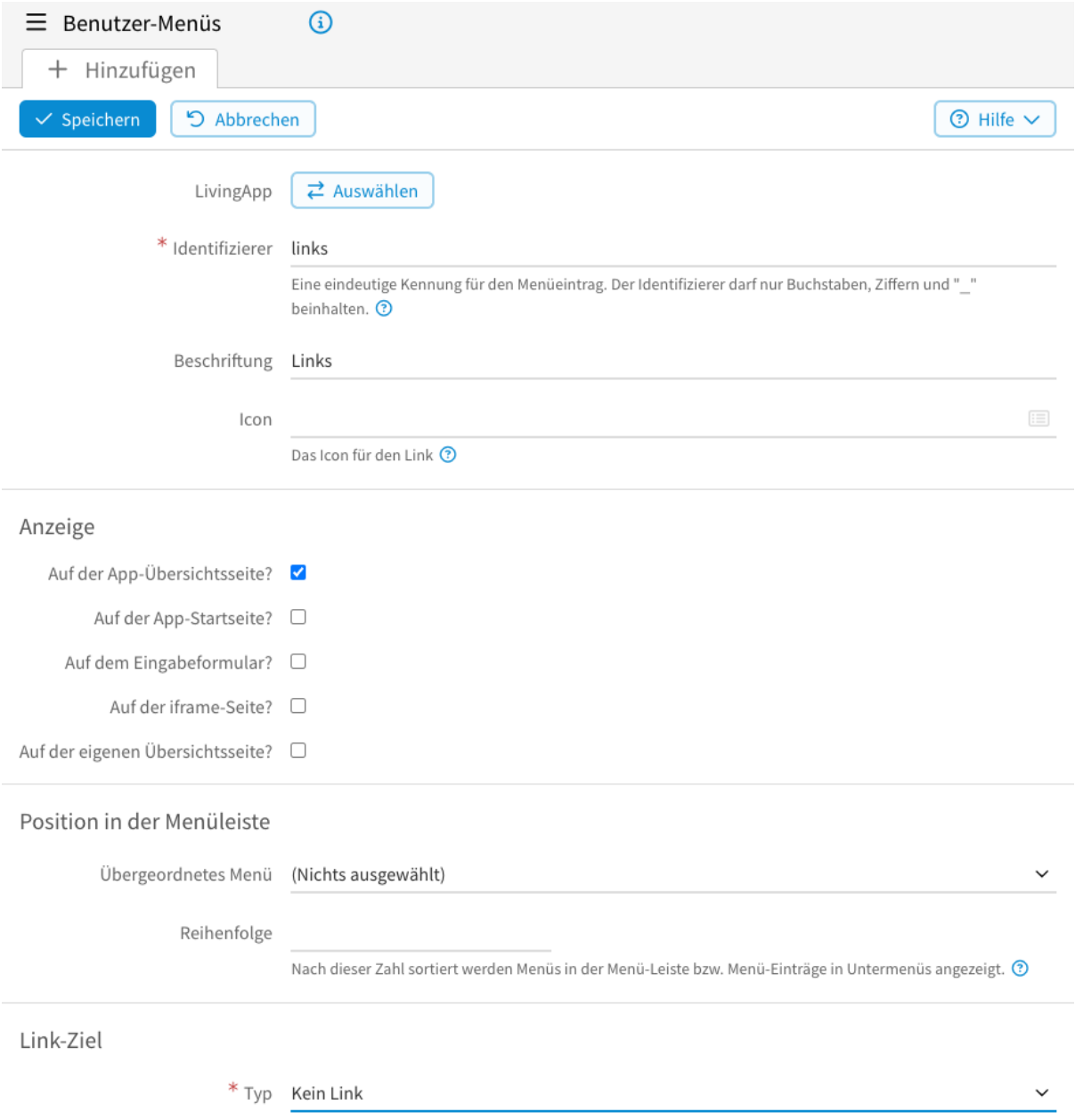


Abb. 162: Übergeordnetes Menü „Links“ erstellen

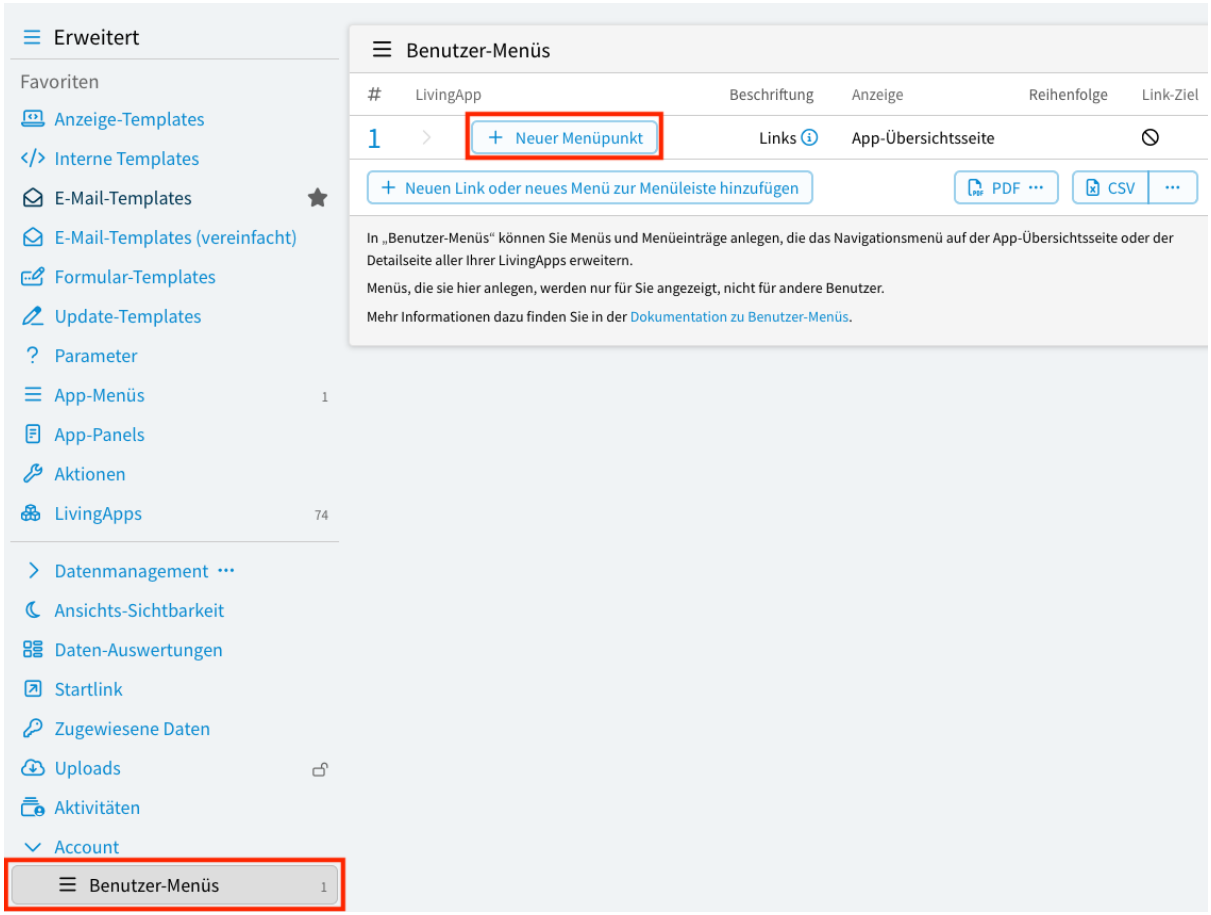


Abb. 163: Neues Menü „Links“

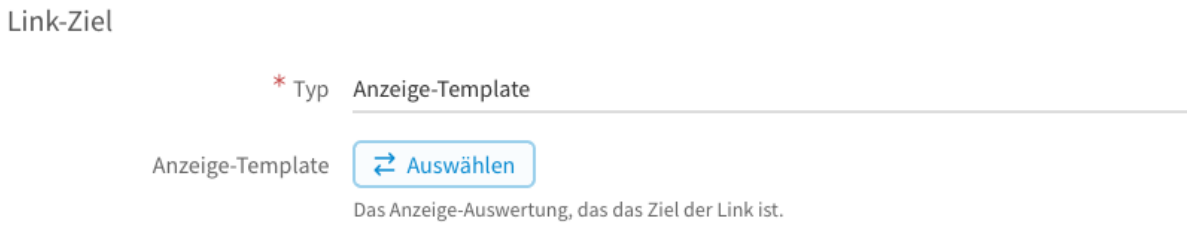


Abb. 164: Link-Ziel mit Auswählen-Button

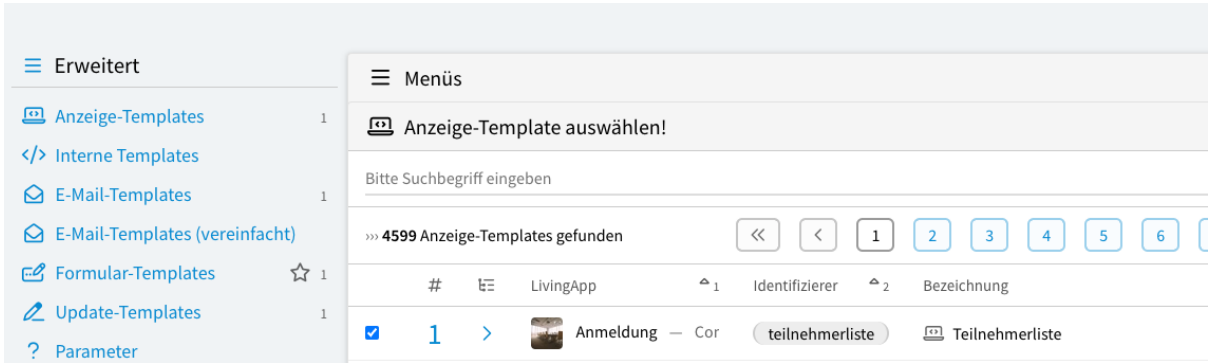


Abb. 165: Auswahl des Anzeige-Templates

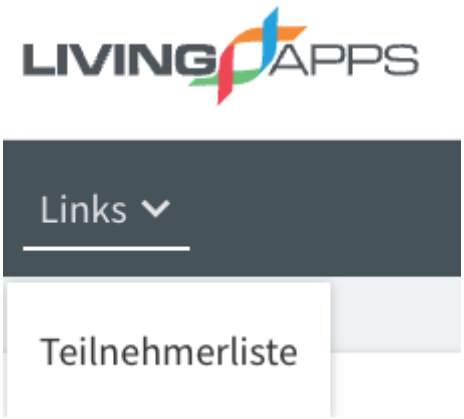


Abb. 166: Link als Menüpunkt in der App-Übersichtseite

Erstellung von Benutzer-Panels

Wählen Sie im Menü *Konfiguration* → *Erweitert* und klicken Sie anschließend in der linken Menüleiste auf *Account*, *Benutzer-Panels* und dann auf *Neues Panel hinzufügen*.

Im nun geöffneten Fenster können folgende Konfigurationen zur *Erstellung eines Links* vorgenommen werden.

LivingApp
: Die App in deren Kontext dieses Panel angezeigt werden soll. Wenn Sie hier keine App auswählen, wird dieses Panel bei allen Ihrer Apps dargestellt.

Identifizierer
: Der eindeutige Name des Links. Der Name darf nur Buchstaben, Ziffern und „_“ beinhalten.

Beschriftung
: Der Link-Text im Kopfbereich des Panels.

Icon
: Das Icon für den Link. Verwendet werden können alle Namen aus dem FontAwesome-Icon-Set.

Darstellung
: *Normal* bzw. *Karte* kann hier ausgewählt werden. *Karte* führt zu einer etwas höheren Darstellung mit mehr Hintergrund.

Hintergrund
: Hier besteht die Möglichkeit zwischen *Einzelner Farbe*, *Linearer Farbverlauf*, *Kreisförmiger Farbverlauf* und *Bild* zu wählen. Je nach Auswahl werden anschliessend Felder für *Textfarbe*, *Hintergrundfarbe oben*, *Hintergrundfarbe unten* oder ein *Bild* angeboten.

Anzeigen auf
: Gibt an, wo das Panel erscheint. Der Link kann sowohl auf der *App-Übersichtsseite?*, *App-Startseite?*, *Eingabeformular?*, *iframe-Seite?* und *eigener Übersichtsseite?* erscheinen.

Positionierung im Panel
: Hier kann, wenn Panels ineinander verschachtelt werden sollen, mit *Übergeordnetes Panel* ein Panel ausgewählt werden. (Wenn Sie in der Panel-Liste den Button *Neues Panel* bei einem vorhandenen Panel anklicken, so wird dieses Panel bei *Übergeordnetes Panel* vorausgewählt)

  Mit *Reihenfolge* kann die Reihenfolge bestimmt werden, in der Panels der gleichen Ebene sortiert werden. Je grösser die Zahl, desto weiter hinten erscheint die Kachel.

Link-Ziel
: Hier kann aus verschiedenen Zielen ausgewählt werden, auf die ein Link zeigen kann. Bei den Zielen *Eingabeformular (öffentlich)*, *Eingabeformular (einebettet)*, *Daten-Management*, *Eigene Übersichtsseite*, *Auswertung*, *Import/Export*, *Arbeitsaufgaben*, *Konfiguration: Eingabe*, *Konfiguration: Arbeitsaufgaben*, *Konfiguration: Daten*, *Konfiguration: Berechtigungen*, *Konfiguration: Erweitert*, *Anzeige-Template* und *Daten-Auswertung* wird eine Auswahlmöglichkeit für den jeweiligen Typ angeboten. Bei *Eigener Link* kann eine

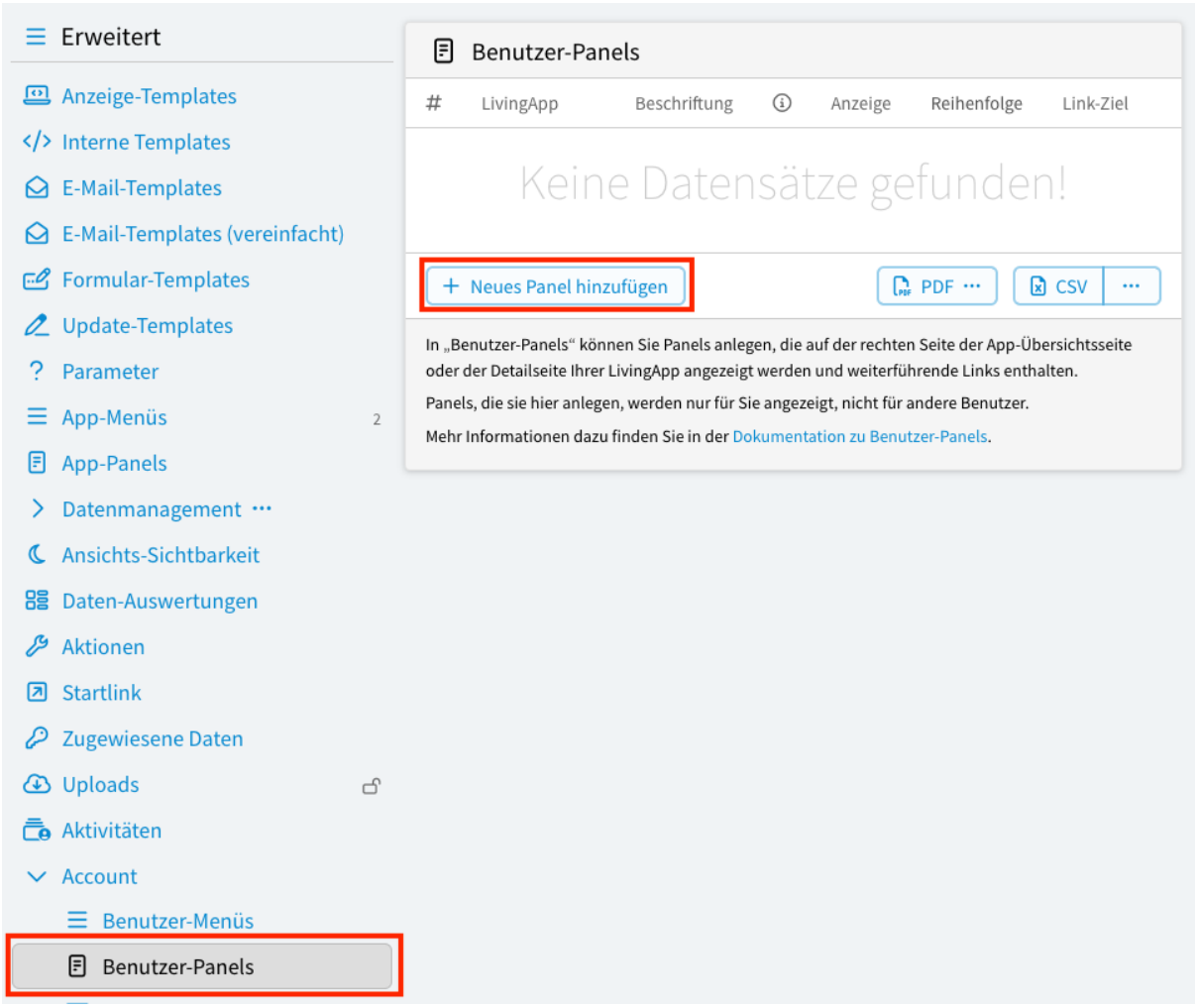
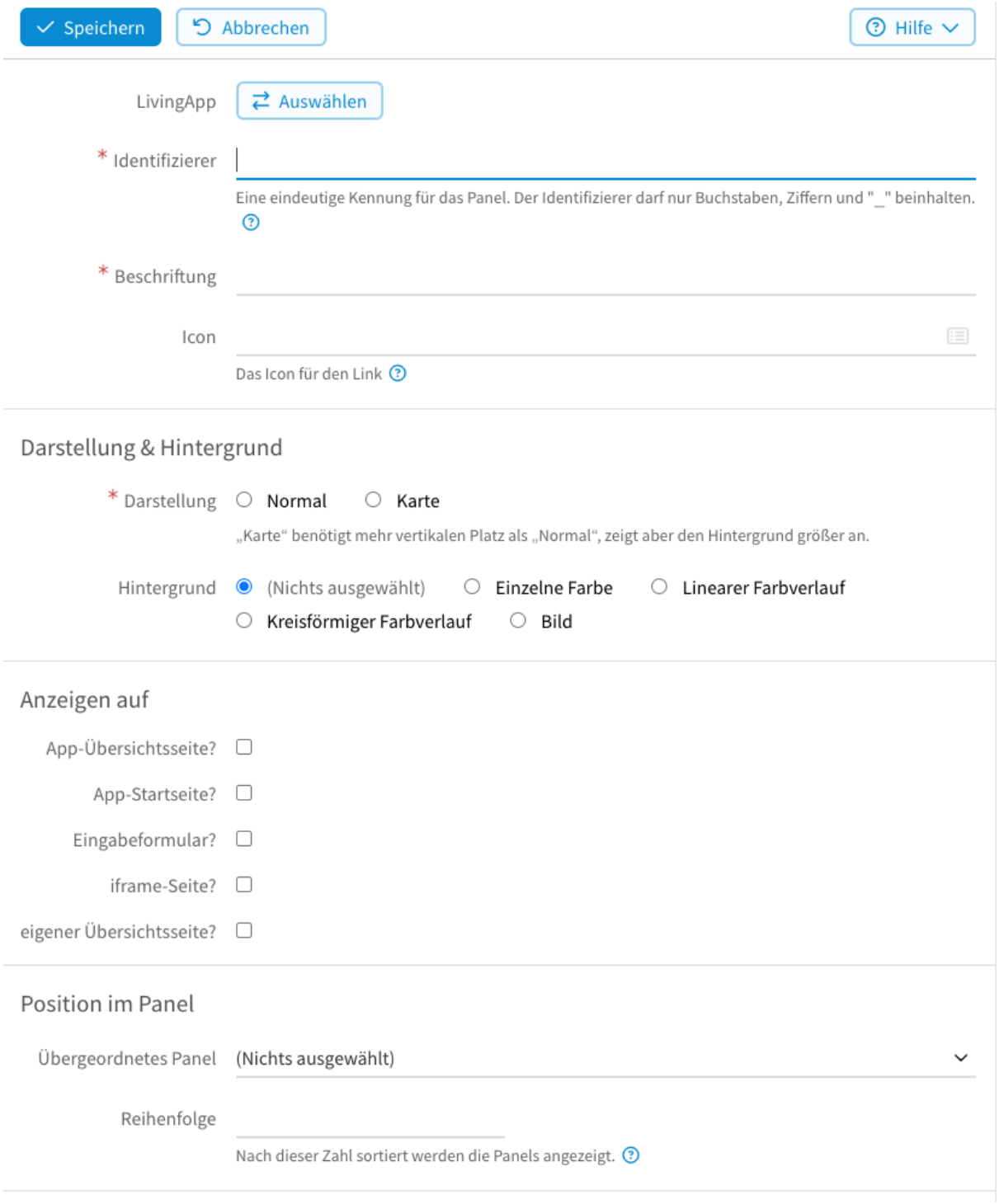
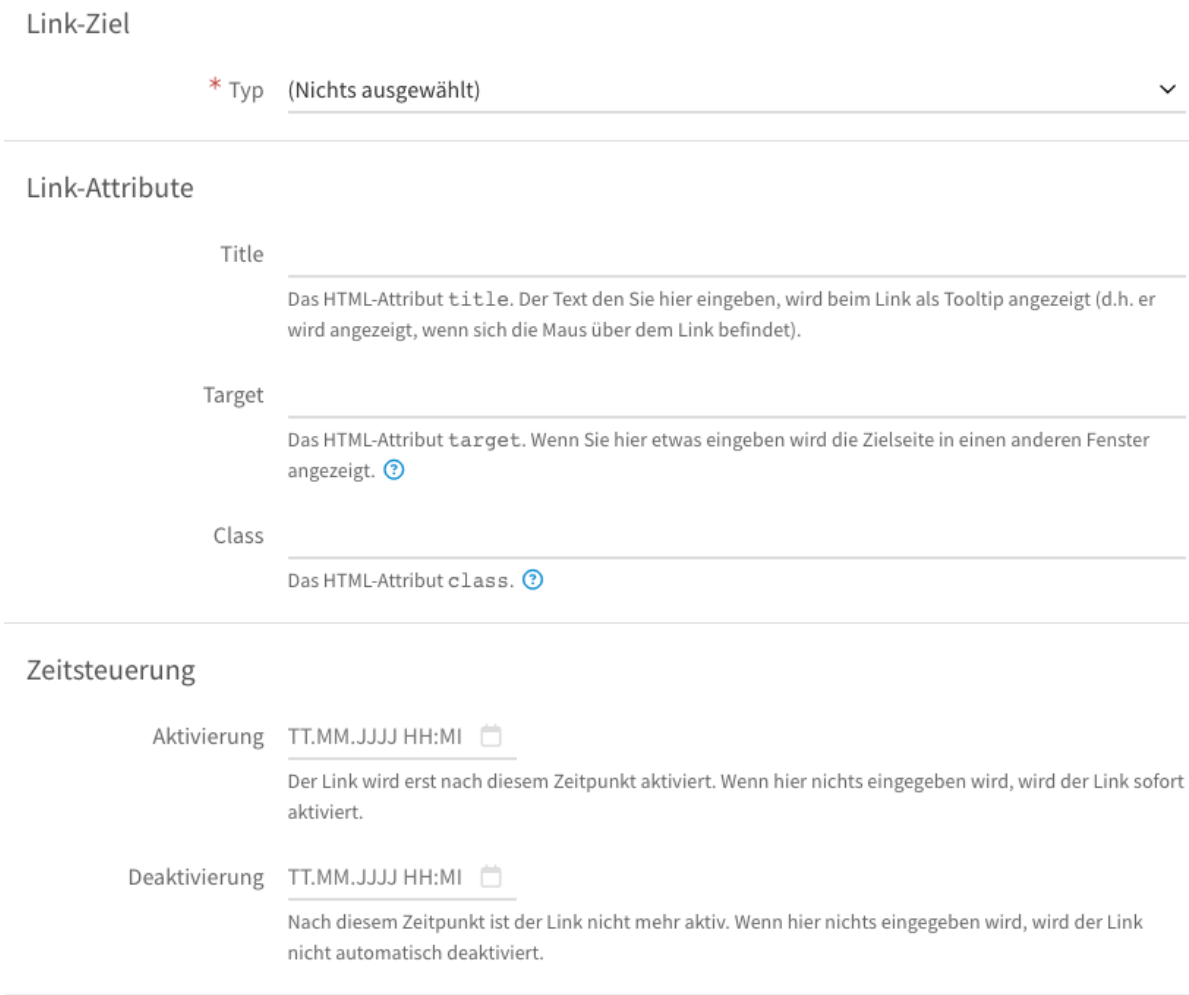


Abb. 167: Hinzufügen eines Benutzer-Panels


Speichern
Abbrechen
Hilfe
LivingApp
Auswählen
* Identifizierer
Eine eindeutige Kennung für das Panel. Der Identifizierer darf nur Buchstaben, Ziffern und "_" beinhalten.
* Beschriftung
Icon
Das Icon für den Link
Darstellung & Hintergrund
* Darstellung
Normal
Karte
„Karte" benötigt mehr vertikalen Platz als „Normal", zeigt aber den Hintergrund größer an.
Hintergrund
(Nichts ausgewählt)
Einzelne Farbe
Linearer Farbverlauf
Kreisförmiger Farbverlauf
Bild
Anzeigen auf
App-Übersichtsseite?
App-Startseite?
Eingabeformular?
iframe-Seite?
eigener Übersichtsseite?
Position im Panel
Übergeordnetes Panel
(Nichts ausgewählt)
Reihenfolge
Nach dieser Zahl sortiert werden die Panels angezeigt.


Link-Ziel
* Typ
(Nichts ausgewählt)
Link-Attribute
Title
Das HTML-Attribut title. Der Text den Sie hier eingeben, wird beim Link als Tooltip angezeigt (d.h. er wird angezeigt, wenn sich die Maus über dem Link befindet).
Target
Das HTML-Attribut target. Wenn Sie hier etwas eingeben wird die Zielseite in einen anderen Fenster angezeigt.
Class
Das HTML-Attribut class.
Zeitsteuerung
Aktivierung
TT.MM.JJJJ HH:MI
Der Link wird erst nach diesem Zeitpunkt aktiviert. Wenn hier nichts eingegeben wird, wird der Link sofort aktiviert.
Deaktivierung
TT.MM.JJJJ HH:MI
Nach diesem Zeitpunkt ist der Link nicht mehr aktiv. Wenn hier nichts eingegeben wird, wird der Link nicht automatisch deaktiviert.

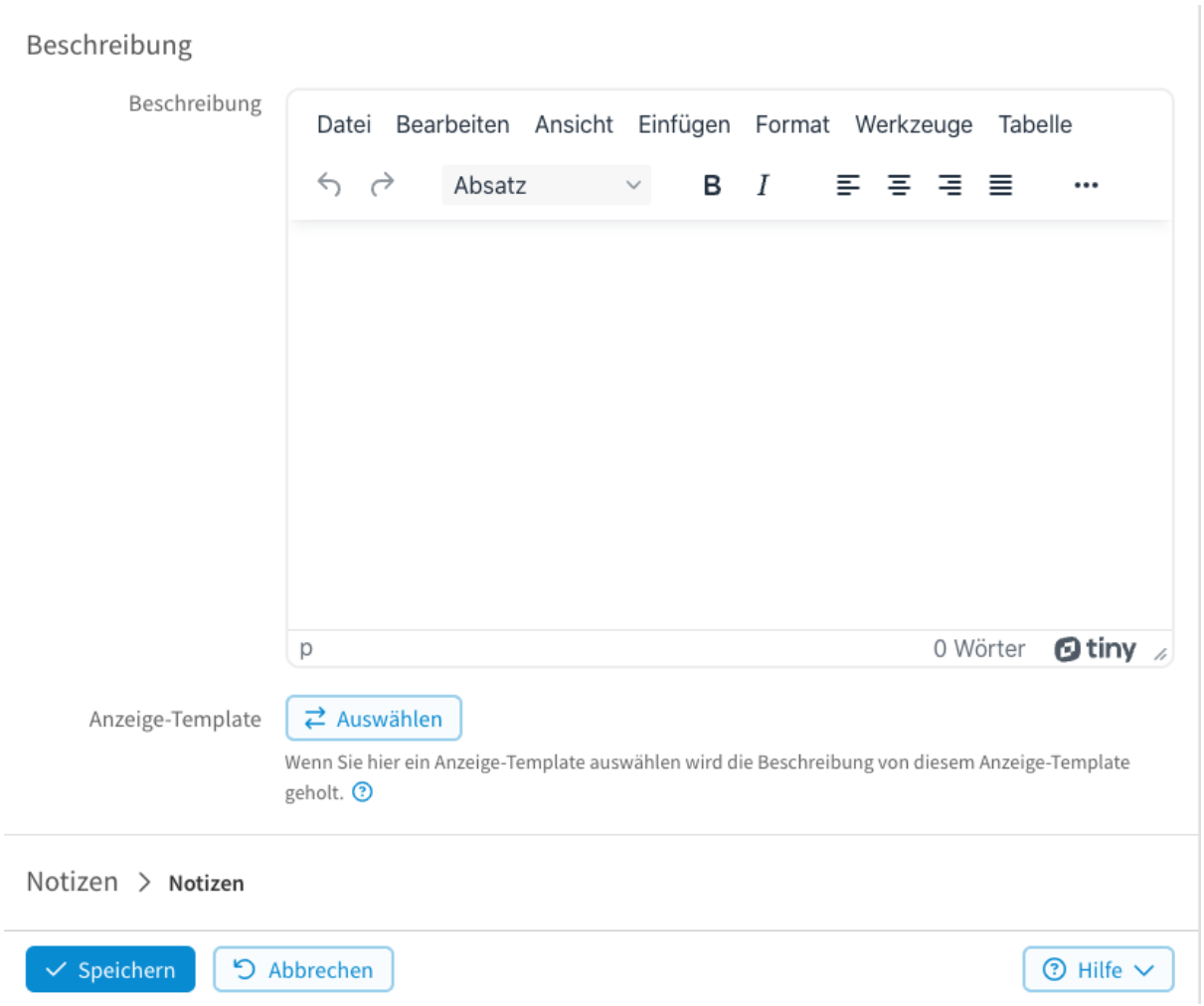


Abb. 168: Erstellung eines Benutzer-Panels

beliebige URL angegeben werden. Der letzte Punkt *Kein Link* muss angewählt werden, wenn ein Panel erstellt werden soll, in dem sich andere Panels befinden.

Link-Attribute
In diesem Abschnitt können die HTML-Attribute *Title*, *Target* und *Class* festgelegt werden.

Zeitsteuerung
Hier kann festgelegt werden in welchem Zeitraum das Panel angezeigt werden soll. Mit *Aktivierung* legt man den Zeitpunkt fest, ab dem der Link angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link sofort aktiviert. Mit *Deaktivierung* kann angegeben werden ab wann der Link nicht mehr angezeigt werden soll. Wenn hier nichts eingegeben wird, wird der Link nicht automatisch deaktiviert.

Beschreibung
Hier kann ein Beschreibungstext angegeben werden, der in dem Panel angezeigt wird. Alternativ kann auch ein *Anzeige-Template* ausgewählt werden, das den Beschreibungstet ausgibt.

Anwendungsbeispiel

Um die Funktionsweise der Benutzer-Panels zu veranschaulichen wird im Folgenden wieder auf das Anwendungsbeispiel aus Kapitel 1 zurückgegriffen.

Link zum Anlegen von Textelementen

Es soll ein Panel erstellt werden, das zum Anlegen eines Textelements verwendet wird. Dieses Textelement soll Hinweise des Veranstalters enthalten und bei der Eingabe einer Anmeldung rechts vom Eingabeformular angezeigt werden (*Eingabeformular?*).

Wählen Sie dafür im Menü *Konfiguration* → *Erweitert* und klicken anschließend in der linken Menüleiste auf *Account*, dann *Benutzer-Panels* und dann auf *Neues Panel hinzufügen* um ein neues Panel zu erstellen.

Im nun geöffneten Fenster werden folgende *Konfigurationen* vorgenommen.

Nach Abspeichern des Links erscheint dieser dann rechts vom Formular in der Dateneingabe (siehe *Link als Textelement*).

Weil es ein Benutzer-Panel ist, das nicht mittels der *LivingApp* Auswahlbox einer speziellen App zugeordnet wurde, erscheint dieses Panel bei **jeder** App auf dem Eingabeformular.

2.20.3 Login-Token

Hier können Sie einen alternativen Login-Mechanismus aktivieren.

Dieser kann beispielsweise in externen Entwicklertools zur Authentifizierung ggü. LivingApps verwendet werden, und ist unabhängig von einer etwaigen Änderung des Passworts.

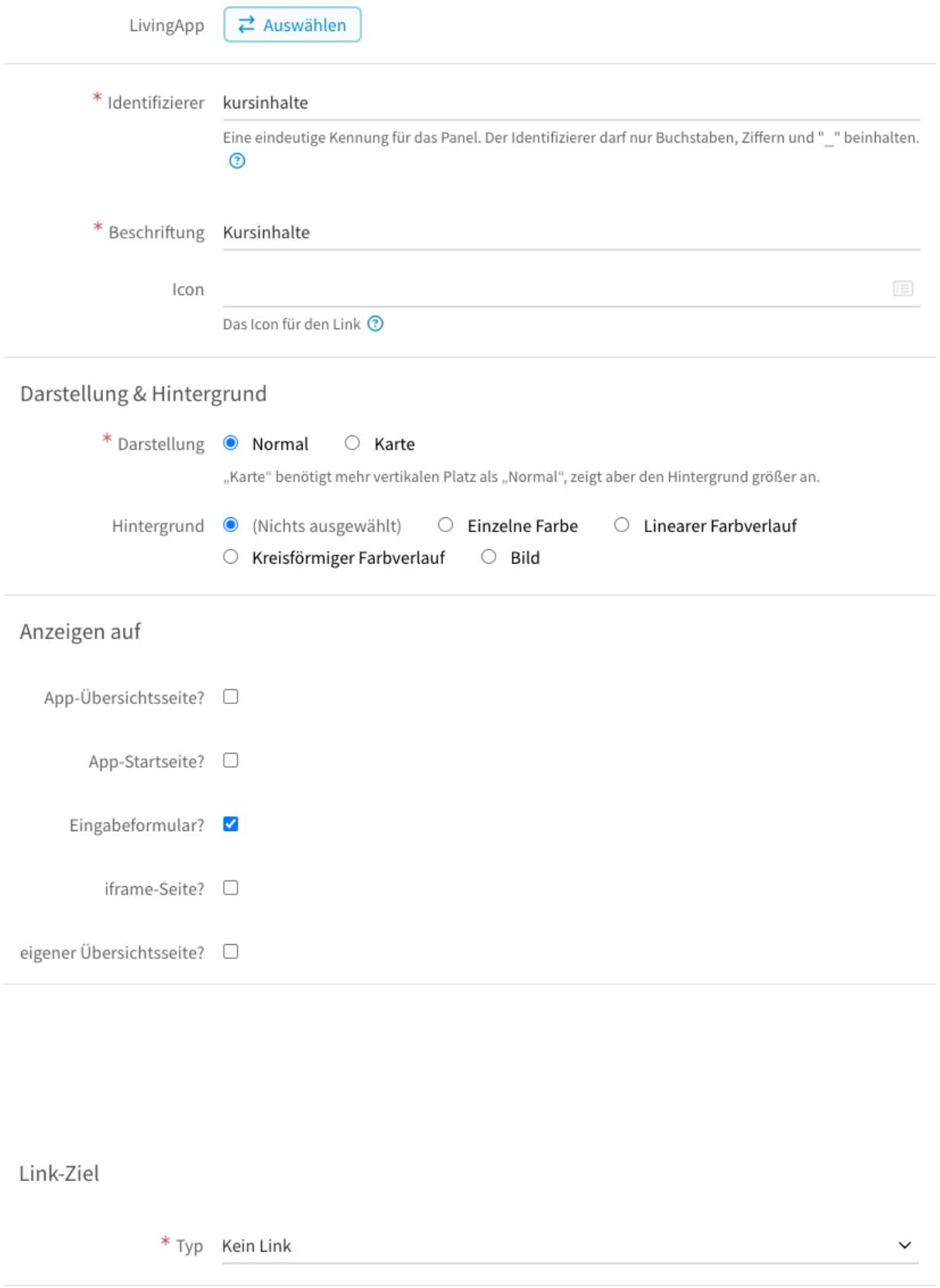
2.20.4 App-Kategorien

Hier können Apps gegliedert werden. Die Kategorien erscheinen dann auf der App-Übersichtsseite *Meine Apps* auf der linken Seite als Navigation. Eine App kann verschiedenen Kategorien zugewiesen werden.

Das Auswahlmenü der App-Kategorien kann auch mehrstufig sein. Zu beachten ist, dass ein Klick auf einen Menüpunkt alle Apps, auch die der darunterliegenden Menüpunkte, anzeigt. Ein Untermenüpunkt filtert immer feiner. Zum Beispiel können hier die Apps in „Privat“ und „Beruflich“ aufgeteilt werden.

Nachdem Sie im Menü *Konfiguration* → *Erweitert* → *Account* → *App-Kategorien* ausgewählt und auf *Hinzufügen* geklickt haben erscheint folgende *Eingabemaske*.

Identifier
Identifizierer der Kategorie.


LivingApp
Auswählen
* Identifizierer
kursinhalte
Eine eindeutige Kennung für das Panel. Der Identifizierer darf nur Buchstaben, Ziffern und "_" beinhalten.
* Beschriftung
Kursinhalte
Icon
Das Icon für den Link
Darstellung & Hintergrund
* Darstellung
Normal
Karte
„Karte" benötigt mehr vertikalen Platz als „Normal", zeigt aber den Hintergrund größer an.
Hintergrund
(Nichts ausgewählt)
Einzelne Farbe
Linearer Farbverlauf
Kreisförmiger Farbverlauf
Bild
Anzeigen auf
App-Übersichtsseite?
App-Startseite?
Eingabeformular?
iframe-Seite?
eigener Übersichtsseite?
Link-Ziel
* Typ
Kein Link

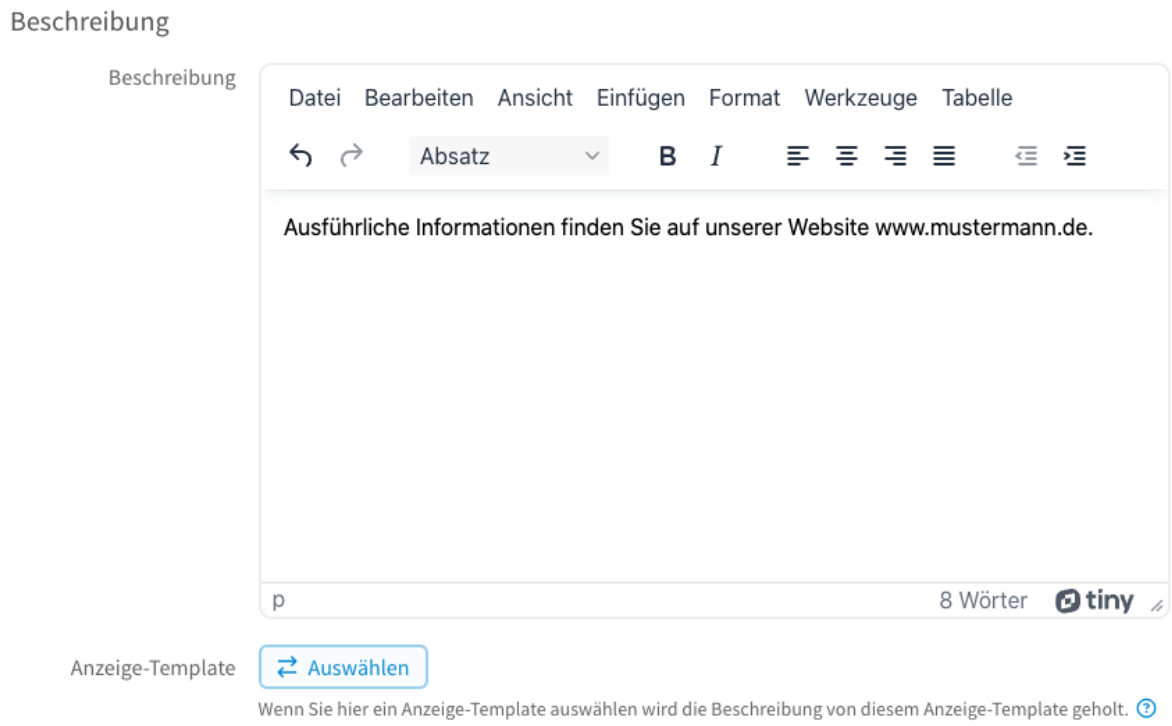


Abb. 169: Erstellen eines Benutzer-Panels als Textelement

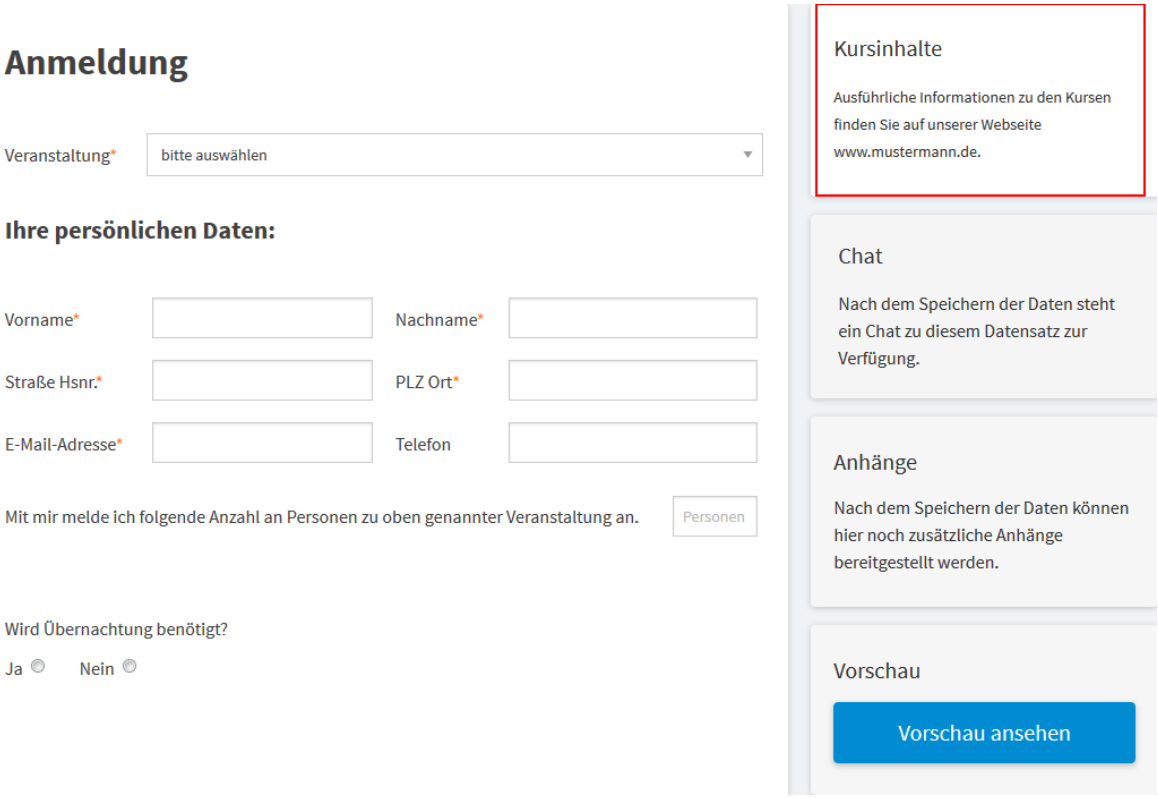


Abb. 170: Links als Textelement in der Eingabemaske

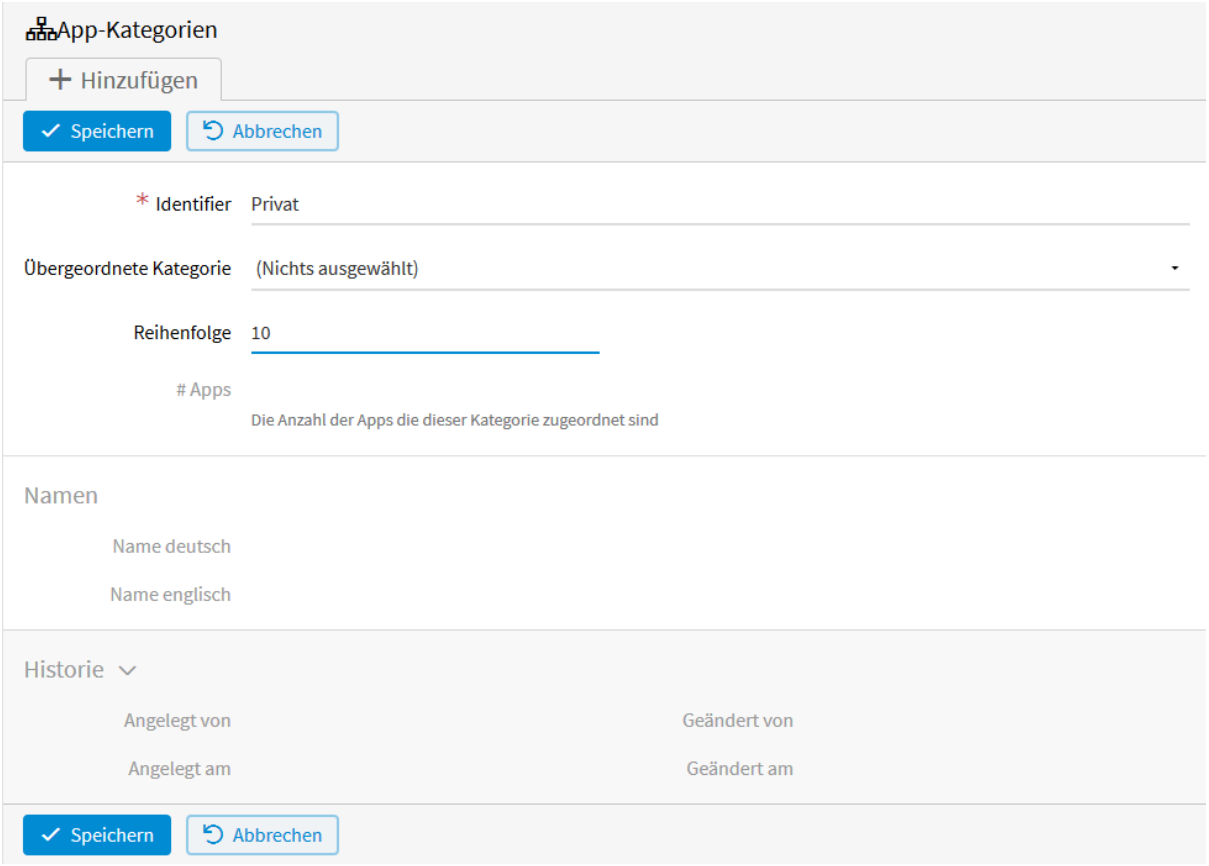


Abb. 171: App-Kategorie anlegen

Übergeordnete Kategorie
: Um eine Kategorie einer anderen als Unterkategorie zuzuordnen, wählen Sie hier die übergeordnete aus.

Reihenfolge
: Legt die Anzeige-Reihenfolge fest.

#Apps
: Sobald der Kategorie Apps zugeordnet sind, wird hier deren Anzahl angezeigt.

Nachdem Sie auf *Speichern* geklickt haben, erscheinen neben *Bearbeiten* die *Tabs* *Übersetzungen* und *Zugeordnete Apps*.

Klicken Sie zunächst auf *Übersetzungen*. Hier muss der *Name* der Kategorie in der entsprechenden Sprache (Deutsch oder Englisch) angegeben werden. Dieser Name steht dann auch in der Navigationsleiste in der App-Übersicht.

Um *Unterkategorien* anzulegen klicken Sie in der Kategorienübersicht auf das + bei der Kategorie, zu der Sie Unterkategorien anlegen wollen.

Es erscheint folgende *Eingabemaske*

Identifier
: Identifizierer der Unterkategorie

Übergeordnete Kategorie
: Die übergeordnete Kategorie ist dann bereits ausgewählt.

Reihenfolge
: Legt die Anzeige-Reihenfolge fest.

#Apps
: Sobald der Kategorie Apps zugeordnet sind, wird hier deren Anzahl angezeigt.

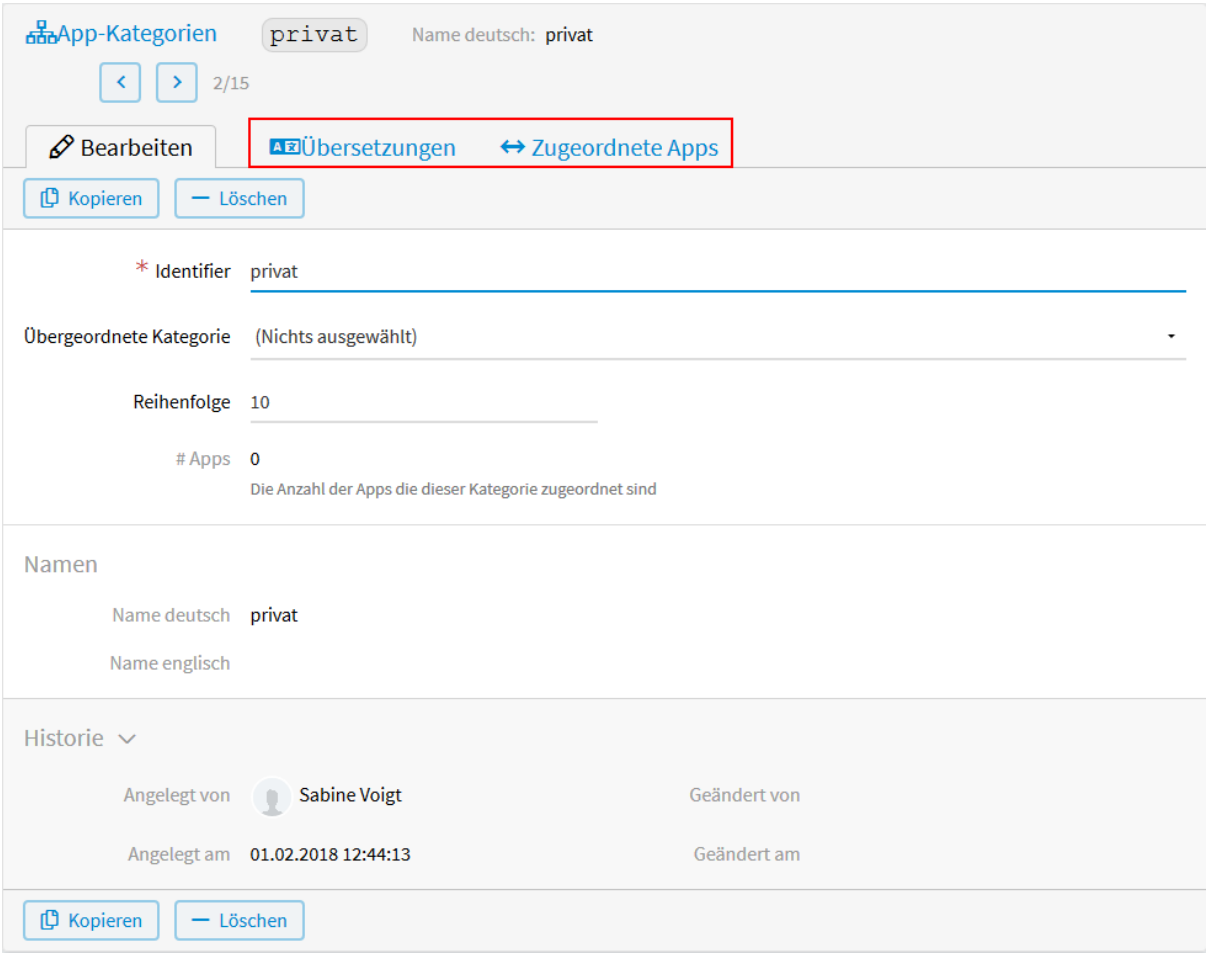


Abb. 172: App-Kategorie Tabs

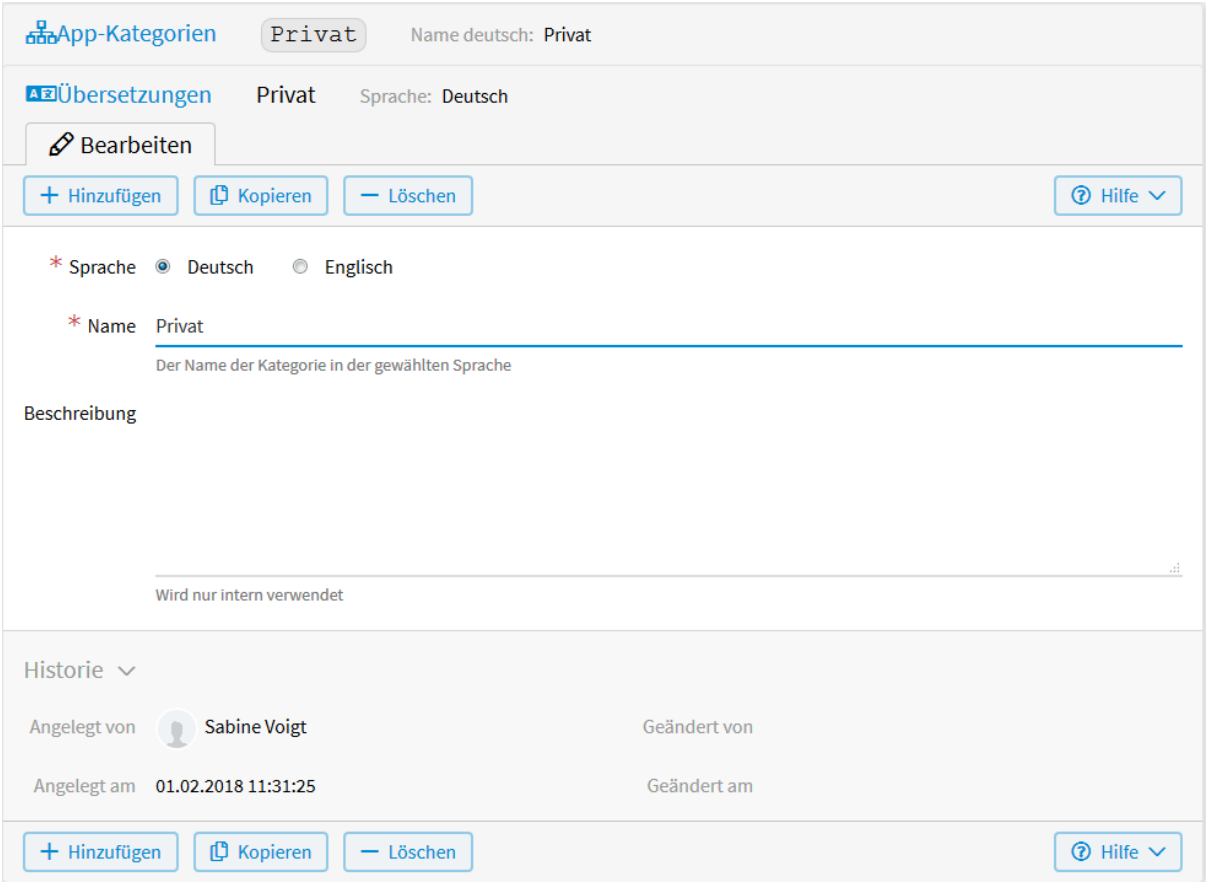


Abb. 173: App-Kategorie Name

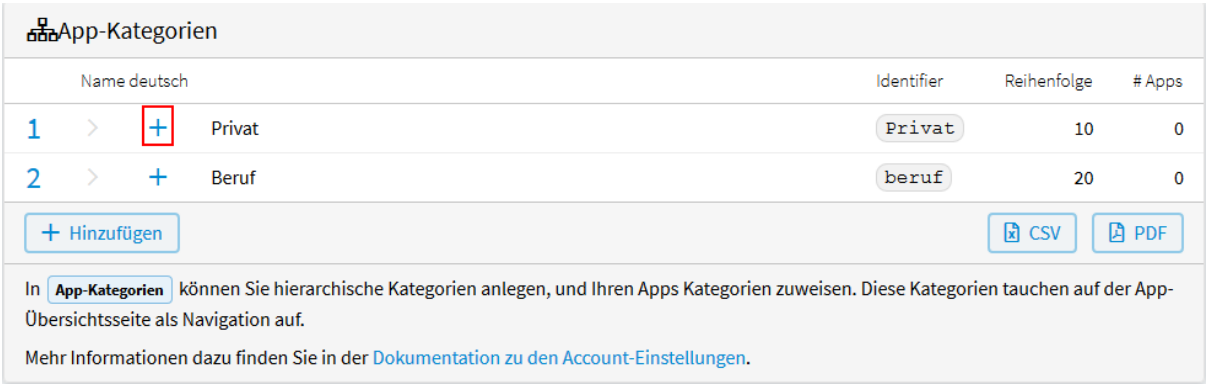


Abb. 174: Unterkategorie hinzufügen

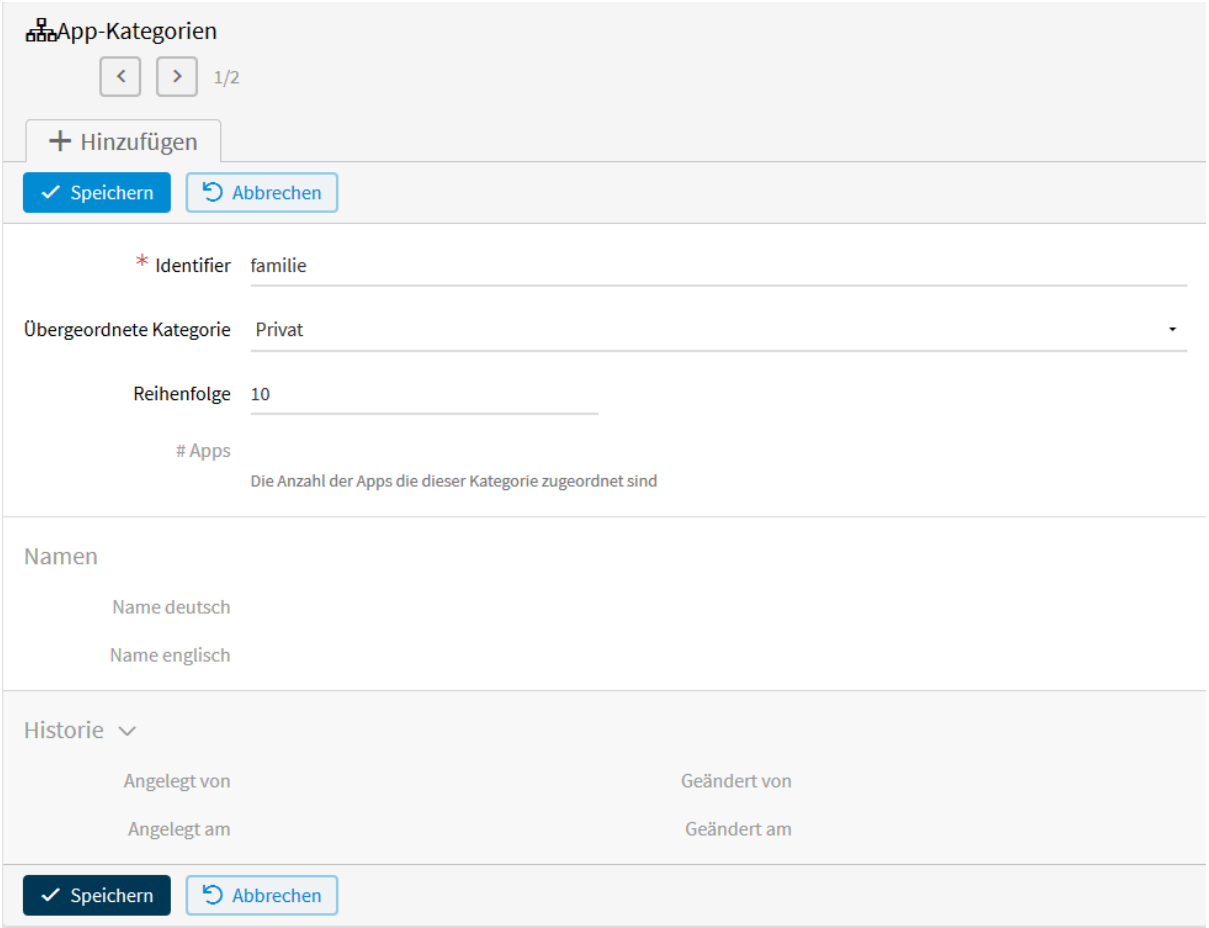


Abb. 175: Unterkategorie anlegen

Nach dem *Speichern* der Unterkategorie muss auch hier bei *Übersetzungen* der Name der Kategorie in der entsprechenden Sprache eingetragen werden.

Nachdem die Kategorien angelegt wurden, können die entsprechenden Apps zugewiesen werden. Das können Sie entweder über *Account* → *App-Zuordnungen* tun, oder über den Tab *Zugeordnete Apps*. Für letzteres klicken Sie in der Kategorien-Übersicht auf die Kategorie, der Sie Apps zuordnen wollen und anschließend auf den Tab *Zugeordnete Apps*. Durch *Hinzufügen* können Sie der Kategorie beliebig viele *Apps zuordnen*.

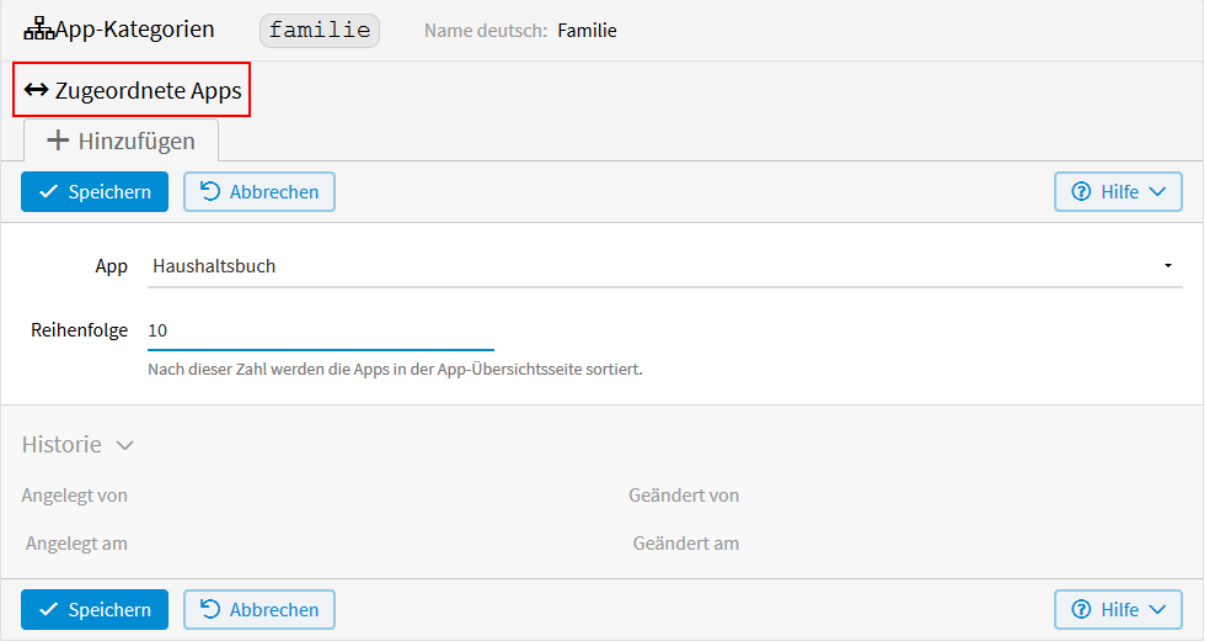


Abb. 176: App-Zuordnung

App
: Wählen Sie die App aus, die Sie dieser Kategorie hinzufügen wollen.

Reihenfolge
: Nach dieser Zahl werden die Apps in der App-Übersichtsseite sortiert. (Wird noch nicht unterstützt)

Das *Ergebnis* der Kategorisierung sehen Sie auf Ihrer App-Übersichtsseite unter *Meine Apps*.

2.20.5 App-Zuordnungen

Sofern App-Kategorien erstellt sind, kann auch hier die Zuordnung der Apps vorgenommen werden. Jede Ihrer Apps kann einer Kategorie zugewiesen werden, sowie die Reihenfolge, nach denen die Apps sortiert werden, festgelegt werden.

Nachdem Sie im Menü *Konfiguration* → *Erweitert* und dann *Account* → *App-Zuordnungen* ausgewählt und auf *Hinzufügen* geklickt haben, erscheint folgende *Eingabemaske*.

App
: Wählen Sie hier die App aus, die einer Kategorie zugeordnet werden soll.

App-Kategorie
: Wählen Sie hier die Kategorie (evtl. auch Unterkategorie) aus, der die oben gewählte App zugeordnet werden soll.

Reihenfolge
: Nach dieser Zahl werden die Apps in der App-Übersichtsseite sortiert. (Wird noch nicht unterstützt)

In unserem *Beispiel* wurden 4 Apps den Kategorien *Privat-Familie* und *Privat-Verein* zugeordnet.

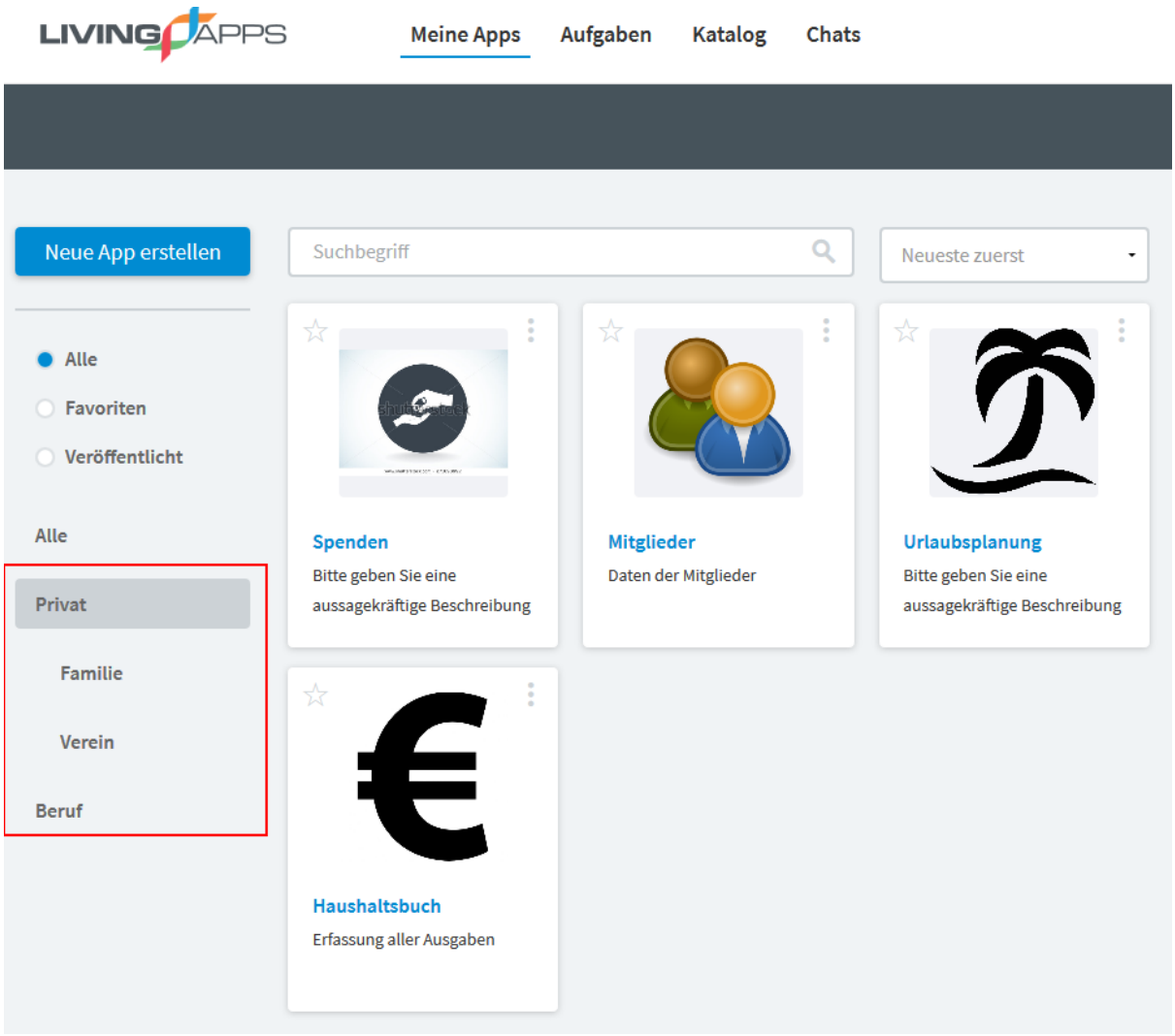


Abb. 177: Ergebnis der Kategorisierung

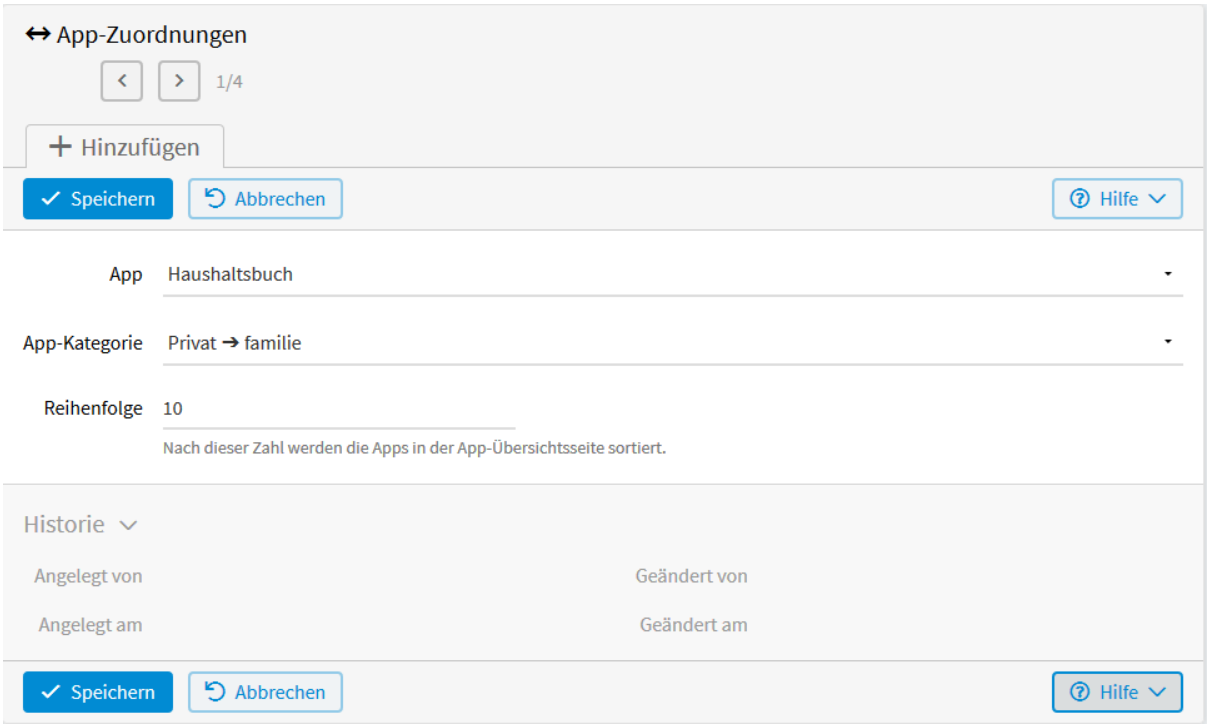


Abb. 178: App-Zuordnungen anlegen

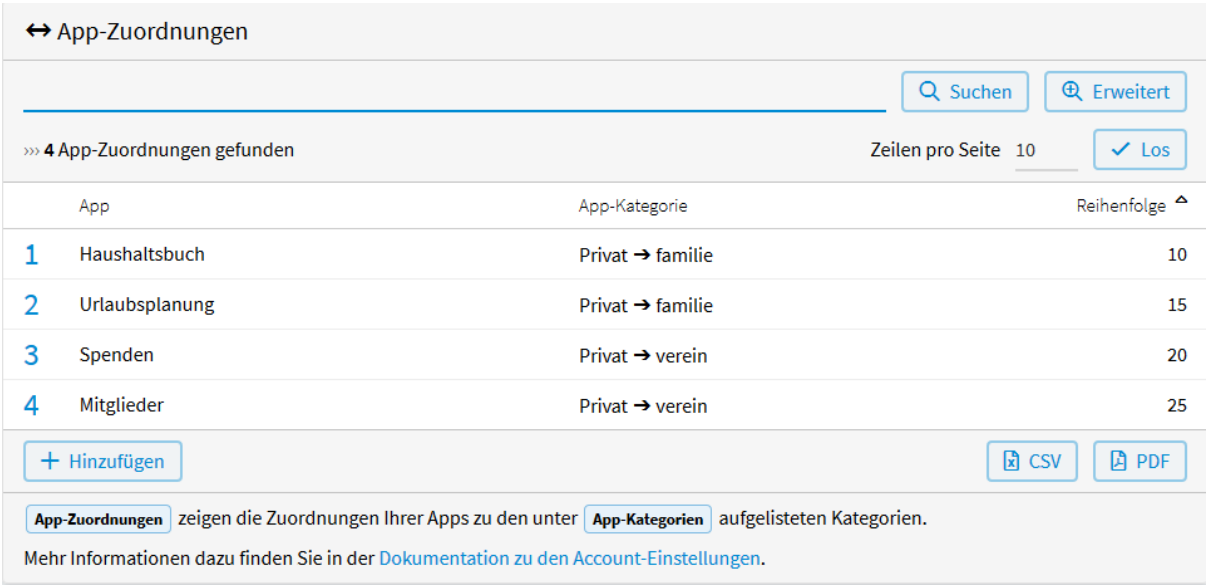


Abb. 179: App-Zuordnungen Ergebnis

2.20.6 Kalender

Hier kann konfiguriert werden, wie Termine der Arbeitsaufgaben in den Apps publiziert werden. Dies ermöglicht es Ihnen, diese Termine mit Ihrem Kalender-Programm zu synchronisieren (z. B. „Outlook Express“ (auf Windows) oder „Kalender“ (auf Mac OS X)).

Nachdem Sie im Menü *Konfiguration* → *Erweitert* und dann *Account* → *Kalender* ausgewählt und auf *Hinzufügen* geklickt haben erscheinen folgende Eingabepunkte.

Name
: Der Name, unter dem dieser Kalender beim Abonnieren in Ihrem Kalender-Programm erscheint.

Dateiname
: Unter diesem Namen wird der Kalender abgespeichert, wenn Sie ihn downloaden statt abonnieren.

App
: Wählen Sie hier die App aus, deren Termine aus den Arbeitsaufgaben Sie im Kalender sehen wollen. Wird hier keine App gewählt, enthält der Kalender alle Ihre Termine aus den Arbeitsaufgaben. Möchten Sie Termine mehrerer Apps sehen, müssen Sie für jede App einen neuen Kalender hinzufügen.

2.20.7 SMTP-Server

Hier kann ein eigener E-Mail-Server für E-Mails, die über LivingApps versendet werden (z. B. bei Versendungen und E-Mail-Templates), eingerichtet werden.

Nachdem Sie im Menü *Konfiguration* → *Erweitert* und dann *Account*/*SMTP-Server* ausgewählt und auf *Hinzufügen* geklickt haben, erscheint folgende *Eingabemaske*.

SMTP-Host
: Geben Sie hier den Namen Ihres Mail-Servers an (z.B. `mail.example.com`).

SMTP-Benutzername
: Tragen Sie hier Ihren Benutzernamen des Mail-Servers ein.

SMTP-Passwort
: Geben Sie hier Ihr Passwort des Mail-Servers ein.

SMTP-E-Mail-Adresse
: Geben Sie hier Ihre E-Mail-Adresse an. Diese E-Mail-Adresse wird bei allen über diesen SMTP-Server versendeten E-Mails als Absender verwendet. Eine vollständige E-Mail-Adresse (inklusive Anzeigename) in der Form `Anzeige Name <name@example.com>` wird unterstützt.

Port
: Die Port-Nummer unter der der SMTP-Server auf dem *SMTP-Host* angesprochen werden kann. Gängige Port-Nummern sind 25, 587, 465 und 2525.

TLS?
: Setzen Sie hier das Häkchen, wenn Sie Ihre Daten sicher und verschlüsselt übertragen wollen.

Authentifizierung
: Wählen Sie hier, wie Ihre Authentifizierung erfolgen soll.

Status des letzten Tests
: Das Ergebnis des zuletzt durchgeführten Zugang-Tests. Wenn Sie die Konfiguration neu abspeichern wird dieser Status zurückgesetzt. Um den Zugangstest erneut durchzuführen, klicken Sie den Button *SMTP-Zugang testen*.

Datenschutz
: Um Ihre E-Mail bei Ihrem SMTP-Server abliefern zu können, muß sich LivingApps bei Ihrem SMTP-Server einloggen. Dazu muß LivingApps Ihre E-Mail-Zugangsdaten speichern. Die E-Mail-Zugangsdaten werden verschlüsselt gespeichert und ausschliesslich zum Zwecke und zum Zeitpunkt des E-Mail-Versands verwendet.

  Sie müssen hier bestätigen, daß Sie mit dieser Speicherung einverstanden sind.

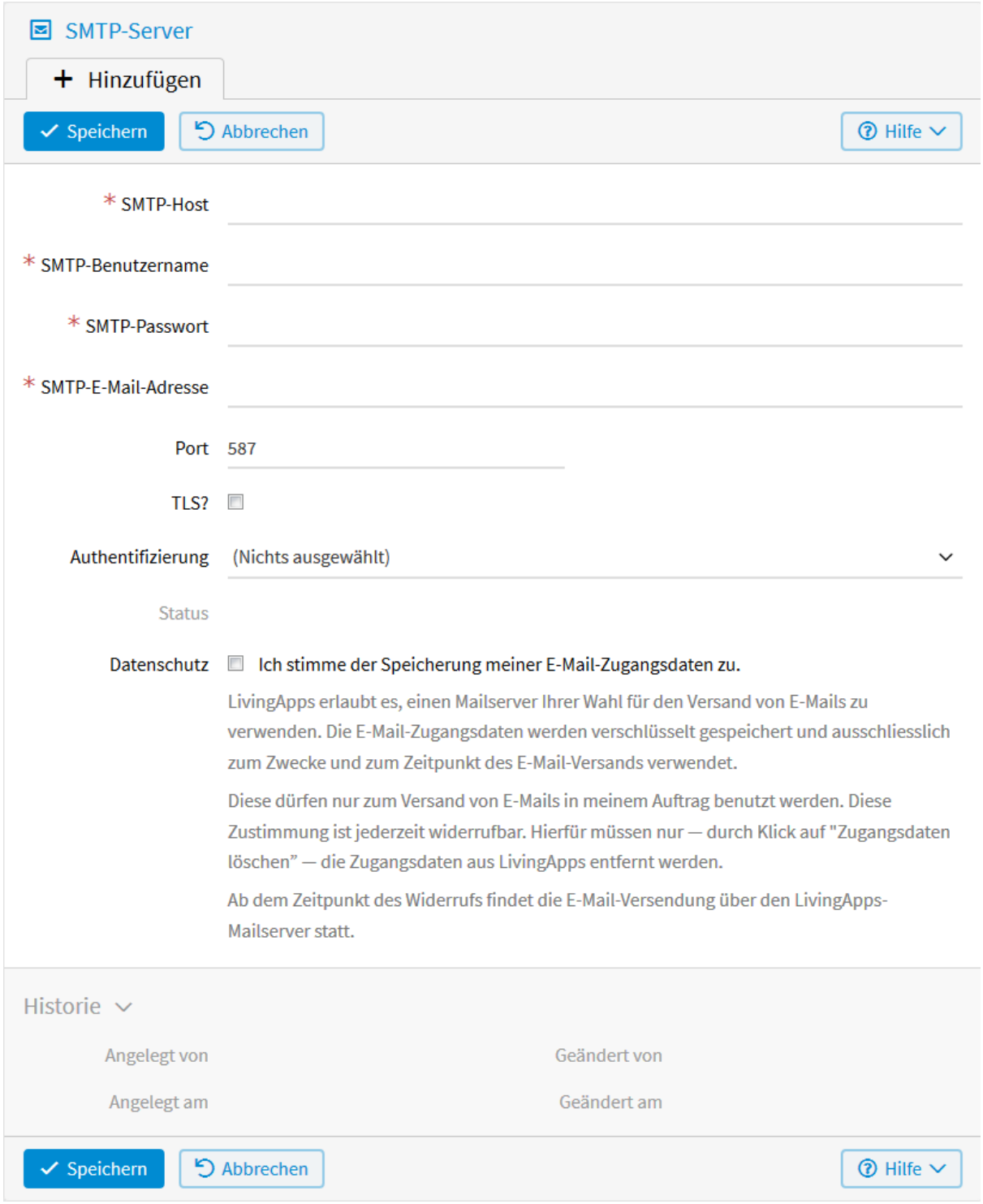


Abb. 180: SMTP-Server anlegen

2.20.8 Installationen

Hier werden alle unter diesem Account durchgeführten Installtionen aufgeführt.

2.20.9 Template-Libraries

Hier finden Sie eine Auswahl an Hilfstemplates, die Sie kopieren und für Ihre Apps einsetzen können.

2.21 Meine LivingApps

Unter *Meine Living-Apps* finden Sie die Übersicht und alle Konfigurationsmöglichkeiten der Apps, bei denen Sie Admin-Berechtigung haben. Das macht es einfacher, vorhandene Konfigurationen zu suchen und von einer App in eine andere zu kopieren.

2.21.1 Living-Apps

Living-Apps ist eine Liste aller Apps, auf die der Benutzer Zugriff hat. Diese Maske bietet eine Kombination der Masken *Startlink*, *Uploads*, *Aktivitäten*, und *Zugewiesene Daten*. Außerdem kann man die Kennung, ob im *Datenmanagement* die *Reihenfolge* bestehen bleiben soll, hier ändern.

Über Maskenlinks sind alle Masken, die es sonst auf oberster Ebene für die per CMS-Navigation ausgewählte App gibt, für die in der WAF-Liste ausgewählte App zugänglich:

Unter *Templates*:
: Anzeige-Templates, Interne Templates, E-Mail-Templates, Formular-Templates, Update-Templates und Parameter.

Unter *Datenmanagement*:
: Farben, Felder und Sorterung

Unter *Sonstiges*:
: Links, Aktionen, Ansichten und Auswertungen

2.21.2 Alle Anzeige-Templates

Die Anzeigetemplates aller Apps, die ich administriere.

2.21.3 Alle Internen Templates

Die Internen Templates aller Apps, die ich administriere.

2.21.4 Alle E-Mail-Templates

Die E-Mail-Teplates aller Apps, die ich administriere.

2.21.5 Alle Formular-Templates

Die Formular-Teplates aller Apps, die ich administriere.

2.21.6 Alle Update-Templates

Die Update-Templates aller Apps, die ich administriere.

2.21.7 Alle Parameter

Die konfigurierten Parameter aller Apps, die ich administriere.

2.21.8 Alle Links

Die erstellten Links aller Apps, die ich administriere.

2.21.9 Farben (Datenmanagement)

Die Farbkonfigurationen aller Apps, die ich administriere.

2.21.10 Alle Felder (Datenmanagement)

Die Felderkonfigurationen aller Apps, die ich administriere.

2.21.11 Sortierung (Datenmanagement)

Die Sortierkonfigurationen aller Apps, die ich administriere.

2.21.12 Alle Ansichts-Sichtbarkeiten

Die Ansichtssichtbarkeiten aller Apps, die ich administriere.

2.21.13 Alle Aktionen

Die Aktionen aller Apps, die ich administriere.

2.21.14 Alle Daten-Auswertungen

Die Daten-Auswertungen aller Apps, die ich administriere.

2.22 Library-Templates

LivingApps bietet Ihnen einige vorgefertigte Templates, die sie in Ihren Anzeige-, Formular- und E-Mail-Templates verwenden können.

2.22.1 Übersicht über die Templates

Um zur Liste der Library-Templates zu gelangen, wählen Sie im Menü einer beliebigen App *Konfiguration* → *Erweitert* und klicken anschließend in der linken Menüleiste auf *Account* und dann auf *Library-Templates*:

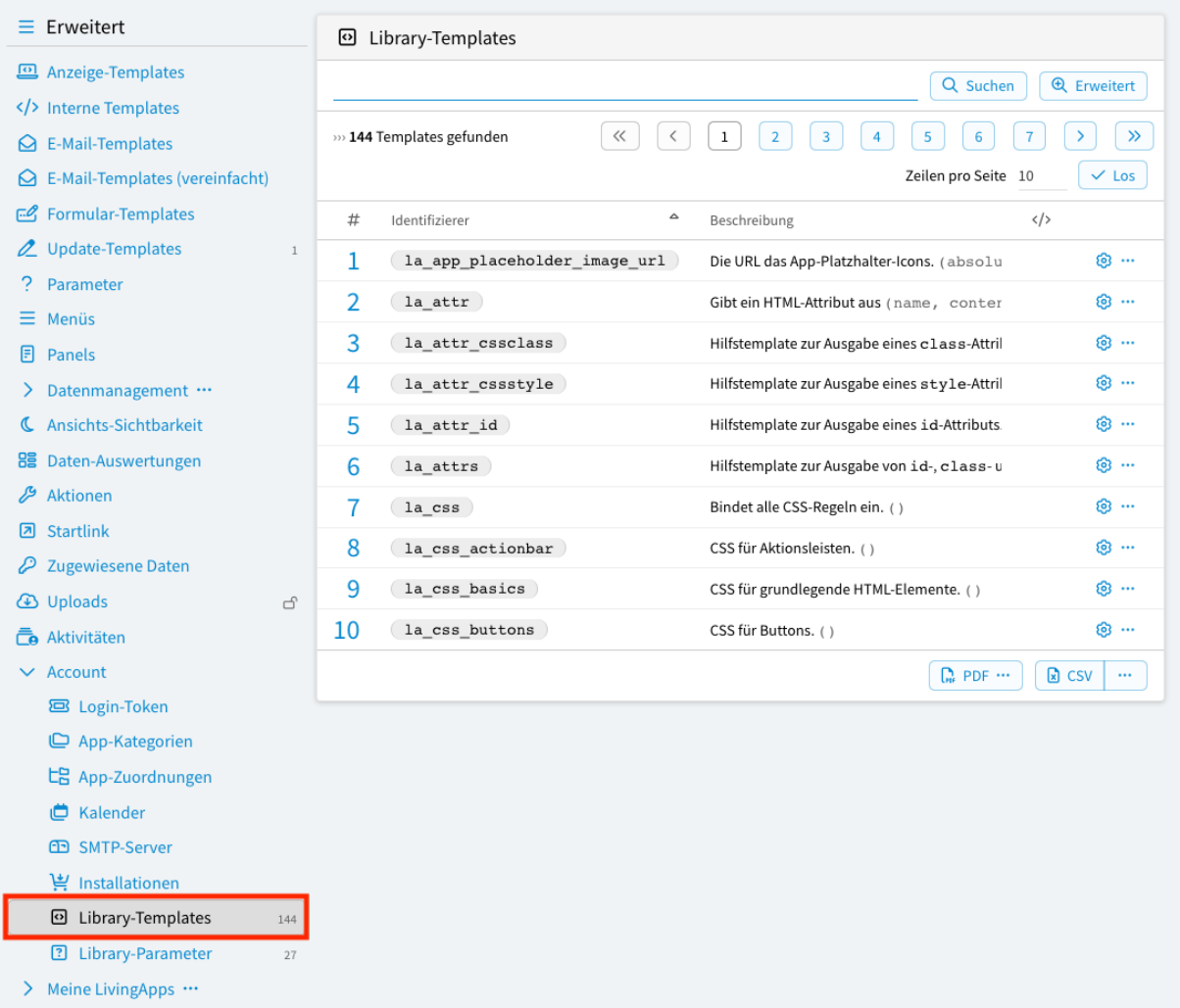


Abb. 181: Library-Templates

Ein Klick auf eines der Templates in der Liste führt Sie zum Quelltext und zur Dokumentation des Templates:

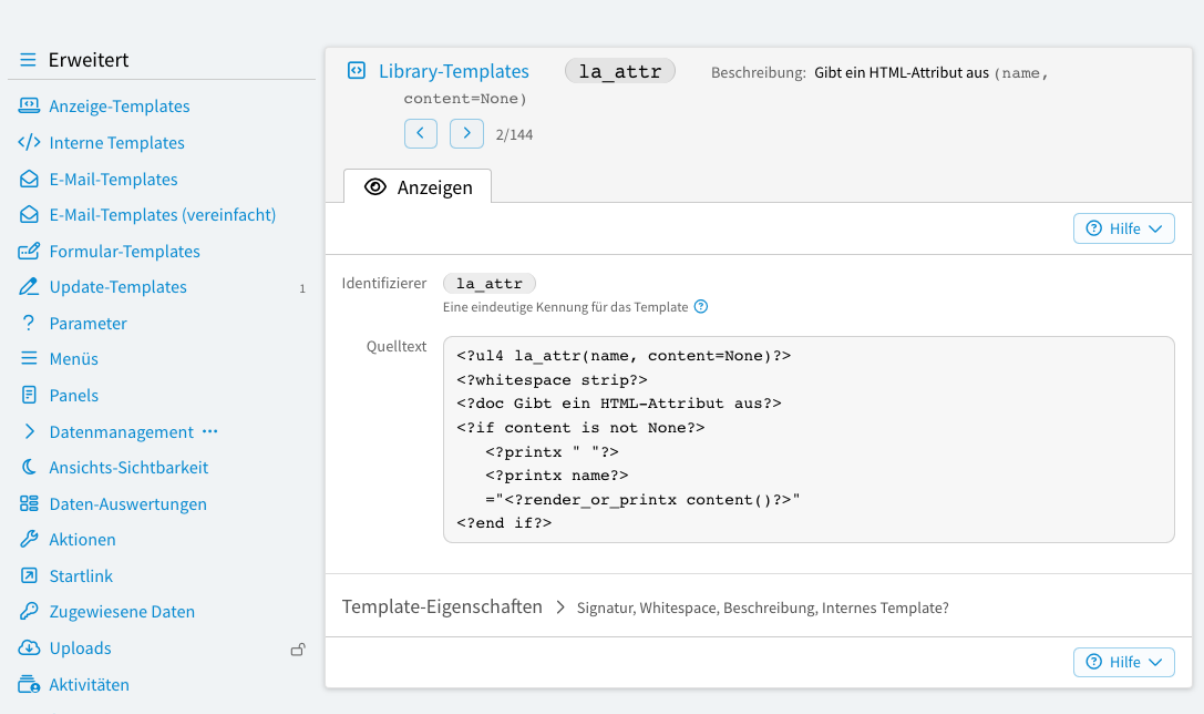


Abb. 182: Library-Template

2.22.2 Einbinden eines Templates

Sämtliche Library-Templates sind auf die selbe Art und Weise aufrufbar, wie wenn sie interne Templates der jeweiligen App wären. Um z.B. das Library-Template `la_static_ul4` in Ihrem Anzeige-Template auszuführen, benutzen Sie folgenden Code:

```
<?render globals.app.templates.la_static_ul4()?>
```

Genauso wie bei den internen Templates sind auch hier folgende alternative Aufrufe möglich:

```
<?render globals.app.t_la_static_ul4()?>
<?render globals.templates.la_static_ul4()?>
<?render globals.t_la_static_ul4()?>
```

2.22.3 Überschreiben eines Templates

Wollen Sie die Funktionalität eines Library-Templates ergänzen oder ersetzen, können Sie das tun indem sie ein Template mit dem gleichen Namen in den internen Templates Ihrer App ablegen.

Sie können dies auch tun, indem Sie ihre eigenen Templates in einer Ihrer Apps gesammelt in deren internen Templates ablegen.

Wenn Sie beispielsweise ein eigenes internes Template `la_static_ul4` anlegen, wird beim Aufruf des Templates mittels folgendem Code:

```
<?render globals.t_la_static_ul4()?>
```

der Reihe nach die Existenz der folgenden Templates überprüft:

- `globals.app.templates.la_static_ul4`
- `globals.app.params.la.value.templates.la_static_ul4`
- `globals.app.params.la.value.params.la.value.templates.la_static_ul4`

- …

Das erste gefundene Template wird ausgeführt. Wird kein Template gefunden, so wird `la_static_ul4` aus der Template-Library verwendet.

Das heißt:

- zuerst wird überprüft ob `globals.app` ein Template `la_static_ul4` besitzt.
- Ist dies nicht der Fall, wird getestet, ob diese App einen App-Parameter `la` hat, der vom Typ `App` ist. In diesem Fall wird das Template dann in dieser App gesucht.
- Ist es auch dort nicht vorhanden, so wird wiederum der App-Parameter `la` **dieser** App getestet usw.
- Diese Suche bricht ab, sobald eine der Apps in der Kette keinen App-Parameter `la` hat oder dieser Parameter nicht vom Typ `App` ist. In diesem Fall wird dann final das Template `la_static_ul4` aus der Template-Library verwendet.

2.22.4 Library-Templates als Methoden eines Typs

In der Liste der Library-Templates fällt auf, dass es Templates gibt, die einen Typ haben. Dieser Typ kann *App*, *Control*, *Field*, *File*, *Geo*, *LayoutControl*, *Record MenuItem* oder *Panel* sein. Das bedeutet, dass die jeweiligen Templates diesem Objekt-Typ zugeordnet sind. Der erste Parameter hat immer den Namen *self* und muss beim Aufruf nicht angegeben werden. Stattdessen kann das Template wie eine Methode des Objekts referenziert werden (mit *t_* als Präfix):

..sourcecode:: html+ul4

    <?code geo = globals.geo(lat=49.9552129, long=11.5901843)?> <?render geo.t_la_displayhtml()?>

2.22.5 Namens-Konvention

Der Namen aller Templates in der LivingApps-Template-Library beginnt mit `la_`.

Sie sollten es daher vermeiden, Ihren eigenen internen Templates einen Namen zu geben, der mit `la_` beginnt. Eine Ausnahme bilden selbstverständlich jene Templates, die Sie anlegen, um eines der Templates aus einer LivingApps-Template-Library zu überschreiben.

2.23 Library-Parameter

Die Template-Library verwendet einige Parameter mit denen das Aussehen Ihrer Anzeige- und E-Mail-Templates angepasst werden können, wenn Sie die Template-Library verwenden.

2.23.1 Übersicht über die Parameter

Um zur Liste der Library-Parameter zu gelangen, wählen Sie im Menü einer beliebigen App *Konfiguration* → *Erweitert* und klicken anschließend in der linken Menüleiste auf *Account* und dann auf *Library-Parameter*:

Ein Klick auf einen der Parameter in der Liste führt Sie zur Detailansicht des Parameters:

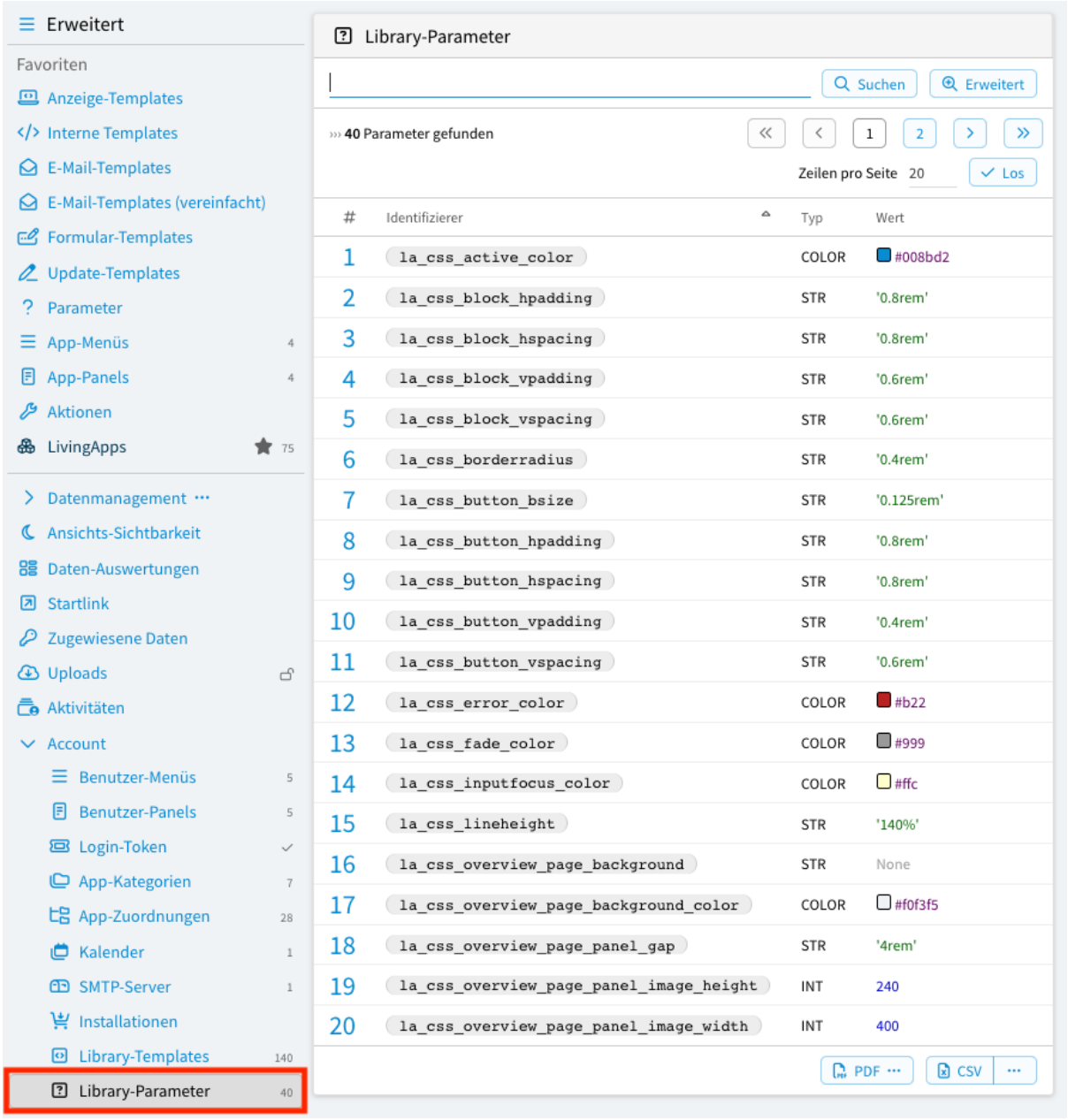


Abb. 183: Library-Parameter

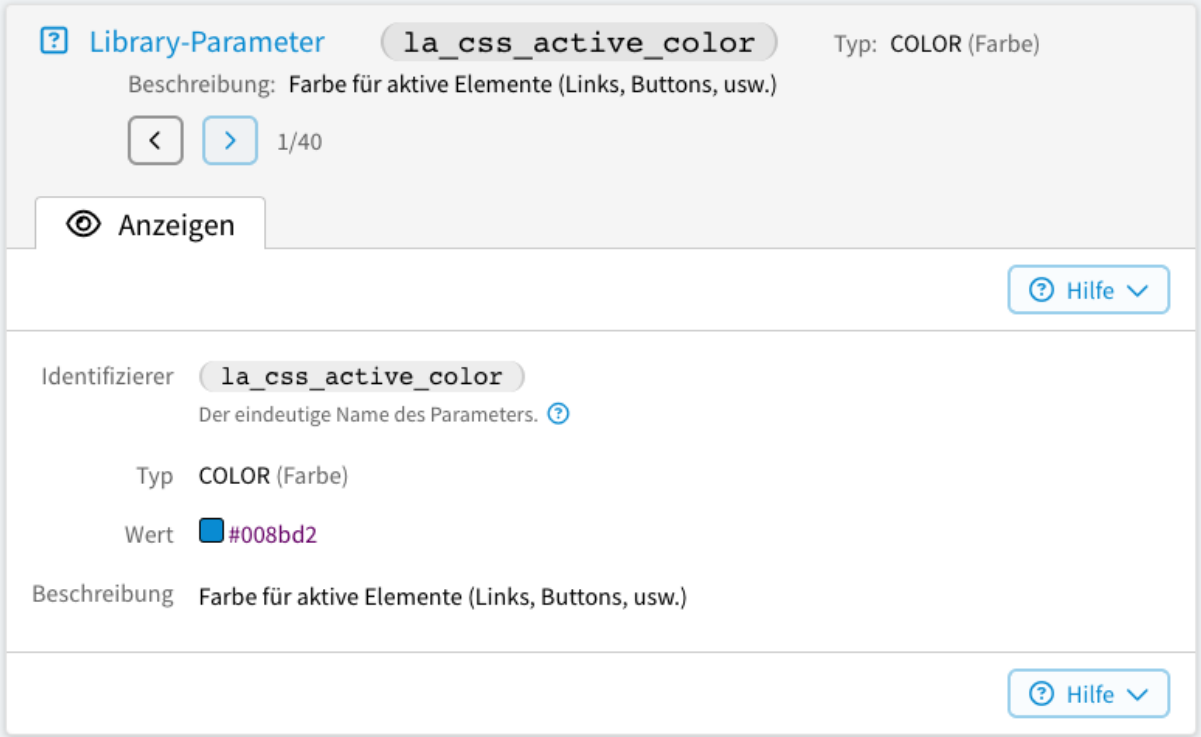


Abb. 184: Library-Parameter

2.23.2 Verwenden eines Parameters

Die Paramter in der Template-Library werden von den Templates in der Template-Library verwendet, daher taucht ein solcher Parameter normalerweise nicht in Ihren eigenen Templates auf. Sie könnten aber natürlich trotzdem auf einen Library-Parameter zugreifen und zwar genauso wie Sie auf App-Parameter zugreifen. Um z.B. den Wert das Parameters `la_css_active_color` auszugeben, können Sie folgenden Code benutzen:

```
<?print globals.app.params.la_css_active_color.value?>
```

Genauso wie bei den App-Parametern sind auch hier folgende alternative Aufrufe möglich:

```
<?print globals.app.params.la_static_ul4.value?>
<?print globals.params.la_static_ul4.value?>
<?print globals.app.p_la_static_ul4.value?>
<?print globals.app.pv_la_static_ul4?>
<?print globals.p_la_static_ul4.value?>
<?print globals.pv_la_static_ul4?>
```

2.23.3 Überschreiben eines Parameters

Wenn Sie innerhalb Ihrer App für einen der Library-Parameter einen anderen Wert verwenden wollen, können Sie das tun indem sie einen App-Parameter mit dem gleichen Namen in den Parametern Ihrer App anlegen.

Sie können dies auch tun, indem Sie ihre Paramter-WErt in einer Ihrer Apps gesammelt in deren App-Parametern ablegen.

Wenn Sie beispielsweise einen eigenen Wert für `la_css_active_color` anlegen, wird beim Abruf des Parameters mittels folgendem Code:

```
<?print globals.pv_la_css_active_color()?>
```

der Reihe nach die Existenz der folgenden Parameter überprüft:

- `globals.app.params.la_css_active_color`
- `globals.app.params.la.value.params.la_css_active_color`
- `globals.app.params.la.value.params.la.value.params.la_css_active_color`
- …

Der erste gefundene Parameter wird verwndet. Wird kein Parameter gefunden, so wird `la_css_active_color` aus der Template-Library verwendet.

Das heißt:

- zuerst wird überprüft ob `globals.app` einen Parameter `la_css_active_color` besitzt.
- Ist dies nicht der Fall, wird getestet, ob diese App einen App-Parameter `la` hat, der vom Typ `App` ist. In diesem Fall wird der Parameter dann in dieser App gesucht.
- Ist es auch dort nicht vorhanden, so wird wiederum der App-Parameter `la` **dieser** App getestet usw.
- Diese Suche bricht ab, sobald eine der Apps in der Kette keinen App-Parameter `la` hat oder dieser Parameter nicht vom Typ `App` ist. In diesem Fall wird dann final der Parameter `la_css_active_color` aus der Template-Library verwendet.

2.23.4 Namens-Konvention

Der Namen aller Parameter in der LivingApps-Template-Library beginnt mit `la_`.

Sie sollten es daher vermeiden, Ihren eigenen Parametern einen Namen zu geben, der mit `la_` beginnt. Eine Ausnahme bilden selbstverständlich jene Parameter, die Sie anlegen, um eines der Parameter-Werte aus der LivingApps-Template-Library zu überschreiben.

2.24 vSQL

vSQL wird verwendet um Filterbedingungen und ähnliche Ausdrücke angeben zu können, die für Datenbankabfragen verwendet werden.

vSQL ist eine auf Ausdrücke reduzierte und eingeschränkte Teilmenge von UL4.

2.24.1 Verwendung

vSQL kommt an folgenden Stellen zum Einsatz:

- als Filter-Bedingung bei *Datenquellen* und *untergeordneten Datensätzen*;
- zur Konfiguration der Sortierreihenfolge im *Datenmanagement*, bei den *Datenquellen* und bei den *untergeordneten Datensätzen*;
- zur Konfiguration der *Zeilenfarben im Datenmanagement*;
- um festzulegen, *welche Datensätze einem Benutzer zugewiesen sind*;
- um festzulegen, *welche Datenaktionen für einen Datensatz zur Verfügung stehen*;
- um festzulegen, *welche Anweisungen einer Datenaktionen ausgeführt werden sollen*;
- um bei einer Datenaktion festzulegen, *auf welchen Wert ein Feld gesetzt werden soll*.

2.24.2 Beispiele

Die folgende Filterbedingung liefert nur Datensätze, die innerhalb der letzten 30 Tage angelegt wurden:

```
r.createdat >= now() - days(30)
```

Die folgende Filterbedingung liefert nur Datensätze, die von jemanden angelegt oder geändert wurden, dessen Accountname auf `@example.org` endet:

```
r.createdby.email.endswith("@example.org") or r.updatedby.email.endswith("@example.org
↪")
```

2.24.3 Variablen

Welche Variablen zur Verfügung stehen, hängt davon ab, an welcher Stelle vSQL eingesetzt wird. Insgesamt gibt es folgende Variablen:

user
: [*Benutzer*]
Der eingeloggte Benutzer ist immer über die vSQL-Variable `user` zugänglich, d.h. dass z.B. `user.firstname` der Vorname des eingeloggten Benutzers ist.

r
: [*Datensätze*]
Der Datensatz der für den jeweiligen Zweck getestet werden soll, also der Datensatz der für die Filterbedingung bei Datenquellen gefiltert werden soll, oder bei den Farben im Datenmanagement eingefärbt werden soll, oder bei der Konfiguration für untergeordnete Datensätze (bei der Filterbedingung und der Sortierung) der untergeordnete Datensatz etc.

record
: [*Datensätze*]
Wenn ein Anzeige-Template als Detail-Template aufgerufen wird und wenn ein E-Mail-Template aufgerufen wird, so ist der Detaildatensatz unter der Variable `record` in der Filterbedingung für Datenquellen zugänglich. Das einzige Attribut das aber tatsächlich vorhanden ist, ist `id` (d.h. `record.id` ist die eindeutige 30-stellige Kennung des Detail-Datensatzes.)

lang
: [String]
Die aktuelle Sprache (`globals.lang` in UL4).

mode
: [String]
Der Template-Modus (`globals.mode` in UL4).

search
: [String]
Der aktuelle Suchausdruck für die Ajax-Suche für Applookups vom Untertyp `"choice"` im Eingabeformular.

params
: In der Filterbedingung für Datenquellen kann mittels `params` auf die Parameter des HTTP-Requests zugegriffen werden (nur bei Anzeige-Templates, nicht bei E-Mail-Templates). `params` besitzt zehn Unter-Attribute.

  params.str.*
  : Enthält die Parameter als Einzelwerte und interpretiert sie als String. Wenn ein Parametername trotzdem doppelt angegeben wurde, ist der Wert der des ersten Auftretens, d.h. in der URL:

    ```
    http://my.living-apps.de/gateway/apps/123456789abcdef12345678?template=gurk&
    ↪x[]=1&y=2&x[]=3
    ```

ergibt `params.str.x` den String `"1"`. Wurde der Parametername gar nicht angegeben, so ist der Wert `None`.

`params.strlist.*`
Enthält die Parameter als Listen und interpretiert sie als String, d.h. wenn ein Parametername mehrmals angegeben wurde, so ist der Wert eine Liste mit mehreren Strings, wurde er nur einmal angegeben, so enthält die Liste nur einen String, wurde der Parametername gar nicht angegeben, so ist die Liste leer. Bei obiger URL ergibt also `params.strlist.x` die Liste `["1", "3"]`, `params.strlist.y` ergibt `["2"]` und `params.strlist.z` ergibt `[]`.

`params.int.*`
Enthält die Parameter als Einzelwerte und interpretiert sie als ganze Zahl. Wird ein Parametername trotzdem doppelt angegeben, ist der Wert der des ersten Auftretens, d.h. in der URL:

```
http://my.living-apps.de/gateway/apps/1234567890abcdef12345678?template=gurk&
↪x[]=1&y=2&x[]=3
```

ergibt `params.int.x` die ganze Zahl `1`. Wurde der Parametername gar nicht angegeben, oder entspricht er nicht dem Format für INT-Parameter, so ist der Wert `None`.

`params.intlist.*`
Enthält die Parameter als Listen und interpretiert sie als ganze Zahl, d.h. wenn ein Parametername mehrmals angegeben wurde, so ist der Wert eine Liste mit mehreren ganzen Zahlen, wurde er nur einmal angegeben, so enthält die Liste nur eine ganze Zahl. Wurde der Parametername gar nicht angegeben, oder entspricht er nicht dem Format für INT-Parameter, so ist die Liste leer. Bei obiger URL ergibt also `params.intlist.x` die Liste `[1, 3]`, `params.intlist.y` ergibt `[2]` und `params.intlist.z` ergibt `[]`.

`params.float.*`
Enthält die Parameter als Einzelwerte und interpretiert sie als Zahl mit Nachkommastellen. Wird ein Parametername trotzdem doppelt angegeben, ist der Wert der des ersten Auftretens, d.h. in der URL:

```
http://my.living-apps.de/gateway/apps/1234567890abcdef12345678?template=gurk&
↪x[]=1.1&y=2.2&x[]=3.3
```

ergibt `params.float.x` die Zahl `1.1`. Wurde der Parametername gar nicht angegeben, oder entspricht er nicht dem Format für NUMBER-Parameter, so ist der Wert `None`.

`params.floatlist.*`
Enthält die Parameter als Listen und interpretiert sie als Zahl mit Nachkommastellen, d.h. wenn ein Parametername mehrmals angegeben wurde, so ist der Wert eine Liste mit mehreren Zahlen, wurde er nur einmal angegeben, so enthält die Liste nur eine Zahl. Wurde der Parametername gar nicht angegeben, oder entspricht er nicht dem Format für NUMBER-Parameter, so ist die Liste leer. Bei obiger URL ergibt also `params.floatlist.x` die Liste `[1.1, 3.3]`, `params.floatlist.y` ergibt `[2.2]` und `params.floatlist.z` ergibt `[]`.

`params.date.*`
Enthält die Parameter als Einzelwerte und interpretiert sie als Datum. Wird ein Parametername trotzdem doppelt angegeben, ist der Wert der des ersten Auftretens, d.h. in der URL:

```
http://my.living-apps.de/gateway/apps/1234567890abcdef12345678?template=gurk&
↪x[]=2018-05-01&y=2018-06-01&x[]=2018-07-01
```

ergibt `params.date.x` das Datum `01.05.2018`. Das Format für `DATE-Parameter` ist `YYYY-MM-DD`. Zum Beispiel kann mit `params.date.datum` in der Filterbedingung der Datenquelle auf `datum=2000-02-29` in der URL zugegriffen werden. Wurde der Parametername gar nicht angegeben, oder entspricht er nicht dem Format für DATE-Parameter, so ist der Wert `None`.

`params.datelist.*`
Enthält die Parameter als Listen und interpretiert sie als Datum, d.h. wenn ein Parametername mehrmals angegeben wurde, so ist der Wert eine Liste mit mehreren Daten, wurde er nur einmal angegeben, so enthält die Liste nur ein Datum. Wurde der Parametername gar nicht angegeben, oder entspricht er

nicht dem Format für DATE-Parameter, so ist die Liste leer. Bei obiger URL ergibt also `params.datelist.x` die Liste `[01.05.2018, 01.07.2018]`, `params.datelist.y` ergibt `[01.06.2018]` und `params.datelist.z` ergibt `[]`.

params.datetime.*
: Enthält die Parameter als Einzelwerte und interpretiert sie als Datum mit Uhrzeit. Wird ein Parametername trotzdem doppelt angegeben, ist der Wert der des ersten Auftretens, d.h. in der URL:

```
http://my.living-apps.de/gateway/apps/1234567890abcdef12345678?template=gurk&
↪x[]=2018-05-01T14:30:55&y=2018-06-01T09:50:22&x[]=2018-07-01T12:30:05
```

ergibt `params.datetime.x` das Datum mit Uhrzeit `01.05.2018 14:30:55`. Das Format für `DATETIME`-Parameter ist `YYYY-MM-DDTHH:MM:SS`. Zum Beispiel kann mit `params.datetime.datum` in der Filterbedingung der Datenquelle auf `datum=2000-02-29T14:30:29` in der URL zugegriffen werden. Wurde der Parametername gar nicht angegeben, oder entspricht er nicht dem Format für `DATETIME`-Parameter, so ist der Wert `None`.

params.datetimelist.*
: Enthält die Parameter als Listen und interpretiert sie als Datum mit Uhrzeit, d.h. wenn ein Parametername mehrmals angegeben wurde, so ist der Wert eine Liste mit mehreren Daten und Uhrzeiten, wurde er nur einmal angegeben, so enthält die Liste nur ein Datum mit Uhrzeit. Wurde der Parametername gar nicht angegeben, oder entspricht er nicht dem Format für `DATETIME`-Parameter, so ist die Liste leer. Bei obiger URL ergibt also `params.datetimelist.x` die Liste `[01.05.2018 14:30:55, 01.07.2018 12:30:05]`, `params.datetimelist.y` ergibt `[01.06.2018 09:50:22]` und `params.datetimelist.z` ergibt `[]`.

2.24.4 Objekte

In vSQL stehen einige Objekte mit Attributen zu Verfügungen, die in Ausdrücken verwendet werden können.

Im folgenden werden diese Objekte beschrieben.

Benutzer

Benutzer-Objekte sind z.B. Informationen über den eingeloggten Benutzer oder über den Benutzer, der eine App oder einen Datensatz angelegt oder geändert hat. Benutzer-Objekte besitzen folgende Attribute:

id
: [*STR*]
Die eindeutige Datenbank-Identifizierer des Benutzers;

email
: [*STR*]
Die Email-Adresse des Benutzers (gleichzeitig auch der Accountname);

firstname
: [*STR*]
Der Vorname des Benutzers;

surname
: [*STR*]
Der Nachname des Benutzers;

initials
: [*STR*]
Die Initialen des Benutzers.

App

App-Objekte enthalten Informationen über die App. Dies kann einerseits die App sein, innerhalb der sich der Benutzer befindet, andererseits auch die App, zu der ein Datensatz gehört. Dies kann evtl. eine andere App sein, da sich z.B. in den Anzeige-, Formular-, Update- oder E-Mail-Templates über Datenquellen konfigurieren läßt, daß dem Template auch Datensätze aus anderen Apps zur Verfügung stehen. Außerdem sind über verknüpfte Apps andere Apps als die Ausgangs-App zugänglich. App-Objekte besitzen folgende Attribute:

id
[*STR*]
Die eindeutige Datenbank-Identifizierer der App.

name
[*STR*]
Der Name der App.

description
[*STR*]
Die Beschreibung der App.

createdat
[*DATETIME*]
Der Zeitpunkt, zu dem die App erstellt wurde.

createdby
[*Benutzer*]
Der Benutzer, der die App erstellt hat.

updatedat
[*DATETIME*]
Der Zeitpunkt, zu dem die App das letzte Mal geändert wurde. (Wurde die App noch nicht geändert, so ist `updatedat` `None`.)

updatedby
[*Benutzer*]
Der Benutzer, der die App zuletzt geändert hat. (Wurde die App noch nicht geändert, so sind alle Attribute des `updatedby`-Benutzers `None`.)

installation
Wenn die Applikation durch einen Installationsvorgang erzeugt wurde, ist `installation` ein `Installation`-Objekt. Ansonsten ist `installation` `None`. Ein `Installation`-Objekt hat folgendes Attribut:

name
[*STR*]
Der Name der Installation.

p_<identifier>
[*App-Parameter*]
Alle vom Benutzer angelegten App-Parameter stehen über Shortcut-Attribute zur Verfügung. D.h., dass beispielsweise der Parameter `beispiel` unter `app.p_beispiel.value` zur Verfügung steht.

App-Parameter

Hier können sie auf die Parameter der Applikation, die unter *Konfiguration* → *Erweitert* in der *Parameter*-Maske angelegt wurden, zugreifen. Dies funktioniert nicht bei zu filternden Apps in der Datenquelle. App-Parameter-Objekte besitzen folgende Attribute:

`id`
: [*STR*]
Der eindeutige Datenbank-Identifizierer des Parameters.

`identifier`
: [*STR*]
Die eindeutige Bezeichnung des Parameters.

`description`
: [*STR*]
Die Beschreibung des Parameters.

`value`
: [Objekt]
Der Wert des Objekts. Dabei hängt der Typ des Wertes vom Typ des Parameters ab.

`createdat`
: [*DATETIME*]
Der Zeitpunkt, zu dem der Parameter erstellt wurde.

`createdby`
: [*Benutzer*]
Der Benutzer, der den Parameter erstellt hat.

`updatedat`
: [*DATETIME*]
Der Zeitpunkt, zu dem der Parameter das letzte Mal geändert wurde. (Wurde der Parameter noch nicht geändert, so ist `updatedat` `None`.)

`updatedby`
: [*Benutzer*]
Der Benutzer, der den Parameter zuletzt geändert hat. (Wurde der Parameter noch nicht geändert, so sind alle Attribute des `updatedby`-Benutzers `None`.)

Datensätze

Datensatz-Objekte haben folgende Attribute:

`id`
: [*STR*]
Die eindeutige Datenbank-Identifizierer des Datensatzes.

`url`
: [*STR*]
Die URL für die Bearbeiten-Ansicht.

`createdat`
: [*DATETIME*]
Der Zeitpunkt, zu dem der Datensatz erzeugt wurde.

`createdby`
: [*Benutzer*]
Der Benutzer, der den Datensatz erstellt hat.

`updatedat`
: [*DATETIME*]

Der Zeitpunkt, zu dem der Datensatz das letzte Mal geändert wurde. (Wurde der Datensatz noch nicht geändert, so ist `updatedat` `None`.)

`updatedby`
[*Benutzer*]
Der Benutzer, der den Datensatz zuletzt geändert hat. (Wurde der Datensatz noch nicht geändert, so sind alle Attribute des `updatedat`-Benutzers `None`.)

`app`
[*App*]
Die App, zu der dieser Datensatz gehört.

`v_<identifier>`
[Objekt]
Alle vom Benutzer angelegten Felder sind über mit Präfixen versehene Attribute verfügbar. D.h. wenn `r` das Datensatz-Objekt ist, ist beispielsweise das Feld mit den Identifier `titel` als `r.v_titel` verfügbar.

2.24.5 Datentypen, Attribute und Methoden

vSQL unterstützt eine Teilmenge der Datentypen, die von UL4 unterstützt werden. Diese und die damit in Verbindung stehenden Methoden werden im folgenden erklärt.

None (`NONE`)

`None` ist die Null-Konstante in vSQL (und UL4).

Boolsche Werte (`BOOL`)

`True` oder `False`

Ganze Zahlen (`INT`)

Beispielsweise `123`

Zahlen mit Nachkommastellen (`NUMBER`)

Beispielsweise `123.45`

Strings (`STR`)

String-Konstanten werden in vSQL (wie in UL4) folgendermaßen geschrieben: `"foo"` oder `'foo'`.

Methoden

Strings besitzen in vSQL folgende Methoden:

str.lower()

Konvertiert den String-Wert `str` in Kleinbuchstaben.

Beispiele:

Ausdruck	Wert
`"GURK".lower()`	`"gurk"`

str.upper()

Konvertiert den String-Wert `str` in Großbuchstaben.

Beispiele:

Ausdruck	Wert
`"Gurk".upper()`	`"GURK"`

str.startswith(präfix)

Testet, ob der String-Wert `str` mit dem übergebenen Präfix `präfix` beginnt. `präfix` kann auch eine Liste von Strings sein, dann gibt `x.startswith(y)` `True` zurück, wenn `x` mit einem der Strings in `y` beginnt.

Beispiele:

Ausdruck	Wert
`"Gurk".startswith("Gu")`	`True`
`"Gurk".startswith("gu")`	`False`
`"Gurk".startswith("")`	`True`
`"Gurk".startswith(["Gu", "Hu", "Ku"])`	`True`
`"Gurk".startswith(["Hu", "Ku"])`	`False`

str.endswith(suffix)

Testet, ob der String-Wert `str` mit dem übergebenen Suffix `suffix` endet. `suffix` kann auch eine Liste von Strings sein, dann gibt `x.endswith(y)` `True` zurück, wenn `x` mit einem der Strings in `y` endet.

Beispiele:

Ausdruck	Wert
`"Gurk".endswith("rk")`	`True`
`"Gurk".endswith("rz")`	`False`
`"Gurk".endswith("")`	`True`
`"Gurk".endswith(["rk", "rz", "nz"])`	`True`
`"Gurk".endswith(["rz", "nz"])`	`False`

`str.strip()` oder `str.strip(None)`

Entfernt Leerzeichen und anderen Whitespace von beiden Enden des String-Wertes `str`.

Beispiele:

Ausdruck	Wert
`"gurk".strip()`	`"gurk"`
`" gurk ".strip()`	`"gurk"`
`" \t\r\n gurk \t\r\n ".strip()`	`"gurk"`
`" \t\r\n gurk \t\r\n hurz \t\r\n ".strip()`	`"gurk \t\r\n hurz"`
`" gurk ".strip(None)`	`"gurk"`

`str.strip(zeichen)`

Entfernt alle Zeichen die in dem String-Wert `zeichen` vorkommen von beiden Enden des String-Wertes `str`.

Beispiele:

Ausdruck	Wert
`"abrakadabra".strip("a")`	`"brakadabr"`
`"abrakadabra".strip("rab")`	`"kad"`

`str.lstrip()`, `str.lstrip(None)` oder `str.lstrip(zeichen)`

Ähnlich wie `strip()`, jedoch werden nur Zeichen am Anfang des Strings entfernt.

Beispiele:

Ausdruck	Wert
`"gurk".lstrip()`	`"gurk"`
`" gurk ".lstrip()`	`"gurk "`
`" \t\r\n gurk \t\r\n ".lstrip()`	`"gurk \t\r\n "`
`" \t\r\n gurk \t\r\n hurz \t\r\n ".lstrip()`	`"gurk \t\r\n hurz \t\r\n "`
`" gurk ".lstrip(None)`	`"gurk "`
`"abrakadabra".lstrip("a")`	`"brakadabra"`
`"abrakadabra".lstrip("rab")`	`"kadabra"`

`str.rstrip()`, `str.rstrip(None)` oder `str.rstrip(zeichen)`

Ähnlich wie `strip()`, jedoch werden nur Zeichen am Ende des Strings entfernt.

Beispiele:

Ausdruck	Wert
`"gurk".rstrip()`	`"gurk"`
`" gurk ".rstrip()`	`" gurk"`
`" \t\r\n gurk \t\r\n ".rstrip()`	`" \t\r\n gurk"`
`" \t\r\n gurk \t\r\n hurz \t\r\n ".rstrip()`	`" \t\r\n gurk \t\r\n hurz"`
`" gurk ".rstrip(None)`	`" gurk"`
`"abrakadabra".rstrip("a")`	`"abrakadabr"`
`"abrakadabra".rstrip("rab")`	`"abrakad"`

`str.find(suchstring)`

Gibt den Offset zurück an dem `suchstring` das erste Mal in `str` auftaucht. Wird `suchstring` nicht gefunden, wird `-1` zurückgegeben.

Beispiele:

Ausdruck	Wert
`"abrakadabra".find("a")`	`0`
`"abrakadabra".find("ra")`	`2`
`"abrakadabra".find("rab")`	`-1`

`str.find(suchstring, start, stop)`

Ähnlich wie der Aufruf mit einem Argument, jedoch wird die Suche auf den Unterstring `str[start:stop]` beschränkt.

Beispiele:

Ausdruck	Wert
`"abrakadabra".find("a", 0)`	`0`
`"abrakadabra".find("a", 1)`	`3`
`"abrakadabra".find("a", 11)`	`-1`
`"abrakadabra".find("a", -1)`	`10`
`"abrakadabra".find("a", -4)`	`7`
`"abrakadabra".find("ra", 3)`	`9`
`"abrakadabra".find("ra", 3, 11)`	`9`
`"abrakadabra".find("ra", 3, 10)`	`-1`
`"abrakadabra".find("ra", 3, -7)`	`9`
`"abrakadabra".find("ra", 3, -8)`	`-1`

`str.rfind(suchstring)` oder `str.rfind(suchstring, start, stop)`

Ähnlich wie `find()`, die Suche beginnt jedoch am Ende des Strings.

Beispiele:

Ausdruck	Wert
`"abrakadabra".rfind("a", 0)`	`10`
`"abrakadabra".rfind("a", 11)`	`10`
`"abrakadabra".rfind("a", 10)`	`7`
`"abrakadabra".rfind("a", 4, -2)`	`7`

`str.split(sep)`

Trennt den String `str` an den Stellen, an denen `sep` vorkommt und liefert die resultierende Liste von Strings zurück.

Beispiele:

Ausdruck	Wert
`"A+B+C".split("+")`	`["A", "B", "C"]`
`"++A++B++C++".split("+")`	`["", "", "A", "", "B", "", "C", "", ""]`

`str.split(sep, maxsplit)`

Ähnlich `str.split(sep)` jedoch wird `str` höchstens `maxsplit` mal getrennt.

Beispiele:

Ausdruck	Wert
`"A+B+C".split("+", 2)`	`["A", "B", "C"]`
`"A+B+C".split("+", 1)`	`["A", "B+C"]`

`str.join(liste)`

Verbindet alle Strings in `liste` mit dem Trennstring `str` zu einen einzigen String.

Beispiele:

Ausdruck	Wert
`"+".join(["A", "B", "C"])`	`"A+B+C"`

Datum (`DATE`) und Datum/Uhrzeit (`DATETIME`)

`@(2000-02-29)`
: Datums-Konstante

`@(2000-02-29T12:34:56)`
: Datums/Zeit-Konstante

Außerdem können `DATE`-Werte mit der Funktion `date` und `DATETIME`-Werte mit der Funktion `datetime` erzeugt werden.

Attribute

`DATE`- und `DATETIME`-Werte besitzen folgender Attribute:

date.year

Das Jahr des Datumswertes `date`.

Beispiele:

Ausdruck	Wert
`@(2000-02-29).year`	`2000`

date.month

Das Monat des Datumswertes `date`.

Beispiele:

Ausdruck	Wert	Bemerkung
`@(2000-02-29).month`	`2`	Februar

date.day

Der Tag des Datumswertes `date`

Beispiele:

Ausdruck	Wert
`@(2000-02-29).day`	`29`

date.weekday

Der Wochentag des Datumswertes `date`. `0` ist Montag, `1` ist Dienstag, ..., `6` ist Sonntag.

Beispiele:

Ausdruck	Wert	Bemerkung
`@(2000-02-29).weekday`	`1`	Dienstag

date.yearday

Die Anzahl der Tage seit Beginn das Jahres.

Beispiele:

Ausdruck	Wert
`@(2000-01-01).yearday`	`1`
`@(2000-02-29).yearday`	`60`
`@(2000-12-31).yearday`	`366`
`@(2001-12-31).yearday`	`365`

Zusätzlich besitzen `DATETIME`-Werte noch die Attribute:

datetime.hour

Die Stundenangabe des Datumswertes `datetime`.

Beispiele:

Ausdruck	Wert
`@(2000-02-29T12:34:56).hour`	`12`

datetime.minute

Die Minutenangabe des Datumswertes `datetime`.

Beispiele:

Ausdruck	Wert
`@(2000-02-29T12:34:56).minute`	`34`

datetime.second

Die Sekundenangabe des Datumswertes `datetime`.

Beispiele:

Ausdruck	Wert
`@(2000-02-29T12:34:56).second`	`56`

Zeiträume (`DATEDELTA` und `DATETIMEDELTA`)

`DATEDELTA`-Werte stellen die Differenz zwischen zwei `DATE`-Werten dar. `DATEDELTA`-Werte können mit den Funktionen *timedelta* und *days* erzeugt werden.

`DATETIMEDELTA`-Werte stellen die Differenz zwischen zwei `DATETIME`-Werten dar und können mit den Funktionen *timedelta*, *hours*, *minutes* und *seconds* erzeugt werden.

Monatszeiträume (`MONTHDELTA`)

`MONTHDELTA`-Werte können mit den Funktionen *monthdelta*, *years* und *months* erzeugt werden.

Farben (`COLOR`)

Farb-Werte (mit oder ohne Alpha-Wert) werden in vSQL (und UL4) ähnlich geschrieben wie in CSS: `#rgba`, `#rrggbb`, `#rgba` oder `#rrggbbaa` (wobei `r`, `g`, `b` und `a` jeweils Hexadezimal-Ziffern sind).

Beispiele:

Wert	Bemerkung
`#000`	Schwarz
`#fff`	Weiß
`#f00`	Rot
`#0f0`	Grün
`#00f`	Blau
`#7f7f7f`	Mittelgrau
`#00ff007f`	Halbtransparentes Grün

Mehr zum RGB-Farbmodel in der Wikipedia: https://de.wikipedia.org/wiki/RGB-Farbraum.

Attribute

Farben haben folgende Attribute:

`color.r`

Rot-Anteil der Farbe `color` (Zahl zwischen 0 und 255);

`color.g`

Grün-Anteil der Farbe `color` (Zahl zwischen 0 und 255);

`color.b`

Blau-Anteil der Farbe `color` (Zahl zwischen 0 und 255);

color.a

Transparenz der Farbe `color` (0 = komplett transparent, 255 = komplett undurchsichtig).

Methoden

Farben haben folgende Methode:

color.lum()

Gibt die Helligkeit der Farbe `color` zurück. Das Ergebnis liegt zwischen 0 (dunkel) und 1 (hell).

Beispiele:

Ausdruck	Wert
`#000.lum()`	`0.0`
`#808080.lum()`	`0.5019607843137255`
`#fff.lum()`	`1.0`

Funktionen

Desweiteren gibt es folgende Funktion um eine Farben zu erzeugen:

rgb(r, g, b, a)

Dabei ist `r` der Rot-Anteil, `g` der Grün-Anteil, `b` der Blau-Anteil und `a` die Transparenz.

Diese Werte müssen jeweils zwischen 0 und 1 liegen. Der Defaultwert für a ist 1.

Beispiele:

Ausdruck	Wert	Bemerkung
`rgb(0, 0, 0)`	`#000`	Schwarz
`rgb(1, 1, 1)`	`#fff`	Weiß
`rgb(1, 0, 0)`	`#f00`	Rot
`rgb(0.5, 0.5, 0.5)`	`#7f7f7f`	Mittelgrau
`rgb(0, 1, 0, 0.5)`	`#00ff007f`	Halbtransparentes Grün

Für Farben gibt es zusätzlich den Farbmisch-Operator `%`. Dabei ist `a % b` die Farbe, die sich ergibt wenn die Farbe `a` auf einem Hintergrund der Farbe `b` aufgetragen würde.

Beispiele:

Ausdruck	Wert	Bemerkung
`#fff % #000`	`#fff`	intransparerentes Weiß auf Schwarz also Weiß
`#fff8 % #000`	`#888`	halbtransparentes Weiß auf Schwarz also Grau
`#f00a % #0f0`	`#a50`	halbtransparentes Rot auf Grün also Gelb

Geo-Koordinaten (GEO)

Geo-Koordinaten können in vSQL mit der Funktion `geo()` erzeugt werden. Es gibt eine Version mit zwei Argumenten (Breiten- und Längengrad) sowie drei Argumenten (Breitengrad, Längengrad und Beschreibung).

Beispiele:

Ausdruck	Beschreibung
`geo(49.955267, 11.591212)`	Breiten- und Längengrad
`geo(49.955267, 11.591212, "LivingLogic AG, Bayreuth")`	Breitengrad, Längengrad und Info

Attribute

Geo-Koordinaten haben folgende Attribute:

geo.lat

Der Breitengrad.

Beispiel:

Ausdruck	Wert
`geo(49.955267, 11.591212, "LivingLogic AG, Bayreuth").lat`	`49.955267`

geo.long

Der Längengrad.

Beispiel:

Ausdruck	Wert
`geo(49.955267, 11.591212, "LivingLogic AG, Bayreuth").long`	`11.591212`

geo.info

Die Beschreibung des Ortes.

Beispiel:

Ausdruck	Wert
`geo(49.955267, 11.591212, "LivingLogic AG, Bayreuth").info`	`"LivingLogic AG, Bayreuth"`

Funktionen

Desweiteren gibt es eine Funktion zur Verarbeitung von Geo-Koordinaten.

`dist(geo1, geo2)`

Berechnet die Entfernung der Geo-Koordinaten `geo1` und `geo2` in Kilomettern.

Beispiel:

Ausdruck	Wert
`dist(geo(49.955267, 11.591212, "LivingLogic AG, Bayreuth"), geo(51.5319695, 9.9416986, "Albanikirchhof, 37085 Göttingen, Deutschland"))`	210,5

Listen

vSQL unterstützt Listen, wenn die Einträge einen der folgende Typen haben: `INT`, `NUMBER`, `STR`, `DATE` oder `DATETIME`. Außerdem müssen alle Einträge denselben Typ haben. Solange nicht alle Werte `None` sind, sind auch `None`-Einträge in der Liste erlaubt.

Beispiele

Ausdruck	Bemerkung
`[1, 2, 3, None, 5]`	`INT`-Liste
`[1.0, 2.0, 3.0, 17.23]`	`NUMBER`-Liste
`["gurk", "hurz", None, "hinz", "kunz"]`	`STR`-Liste
`[@(2000-02-29), @(2019-02-28), today()]`	`DATE`-Liste
`[@(2000-02-29T12:34), @(2019-02-28T12:34), now()]`	`DATETIME`-Liste

2.24.6 Operatoren

vSQL unterstützt folgende Teilmenge der UL4-Operatoren:

`x == y` (Vergleich auf Gleichheit)

Gibt zurück ob `x` und `y` den gleichen Wert haben.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

`x != y` (Vergleich auf Ungleichheit)

Gibt zurück ob `x` und `y` einen unterschiedlichen Wert haben.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

`x < y` (Vergleich auf kleiner)

Gibt zurück ob `x` kleiner als `y` ist.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Dabei müssen die Datentypen von `x` und `y` kompatibel sein. (`"gurk" < 42` wird beispielsweise nicht unterstützt.)

`x <= y` (Vergleich auf kleiner/gleich)

Gibt zurück ob `x` kleiner oder gleich `y` ist.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Dabei müssen die Datentypen von `x` und `y` kompatibel sein. (`"gurk" <= 42` wird beispielsweise nicht unterstützt.)

`x > y` (Vergleich auf größer)

Gibt zurück ob `x` größer als `y` ist.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Dabei müssen die Datentypen von `x` und `y` kompatibel sein. (`"gurk" > 42` wird beispielsweise nicht unterstützt.)

`x >= y` (Vergleich auf größer/gleich)

Gibt zurück ob `x` kleiner oder gleich `y` ist.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Dabei müssen die Datentypen von `x` und `y` kompatibel sein. (`"gurk" >= 42` wird beispielsweise nicht unterstützt.)

`x + y` (Addition)

Addition von Zahlen, außerdem String- und Listen-Verkettung sowie Datumsarithmetik.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Beispiele:

Ausdruck	Wert
`False + True`	`1`
`2 + True`	`3`
`17 + 23`	`40`
`"gurk" + "hurz"`	`"gurkhurz"`
`["gurk", "hurz"] + ["hinz", "kunz"]`	`["gurk", "hurz", "hinz", "kunz"]`
`@(2000-02-29) + days(1)`	`@(2000-03-01)`
`@(2000-02-29T12:34:56) + days(1)`	`@(2000-03-01T12:34:56)`
`@(2000-02-29T12:34:56) + minutes(1)`	`@(2000-02-29T12:35:56)`
`@(2000-02-29) + months(1)`	`@(2000-03-29)`
`@(2000-02-29T12:34:56) + months(1)`	`@(2000-03-29T12:34:56)`
`days(7) + days(7)`	`timedelta(14)`
`days(1) + hours(1)`	`timedelta(1, 3600)`
`hours(1) + minutes(30)`	`timedelta(0, 5400)`
`months(6) + months(6)`	`monthdelta(12)`

`x - y` (Subtraktion)

Subtraktion von Zahlen sowie Datumsarithmetik.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Beispiele:

Ausdruck	Wert
`0 - True`	`-1`
`7 - 3`	`4`
`@(2000-03-01) - @(2000-02-29)`	`timedelta(1)`
`@(2000-03-01) - days(1)`	`@(2000-02-29)`
`days(1) - hours(23)`	`timedelta(0, 3600)`

`x * y` (Multiplikation)

Multiplikation von Zahlen, sowie String- und Listen-Wiederholung sowie Datumsarithmetik.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Beispiele:

Ausdruck	Wert
`False * 2`	`0`
`7 * True`	`7`
`3 * 7`	`21`
`2 * "gurk"`	`"gurkgurk"`
`2 * ["gurk", "hurz"]`	`"["gurk", "hurz", "gurk", "hurz"]`
`False * "hurz"`	`""`
`days(7) * 2`	`timedelta(14)`
`hours(6) * 4`	`timedelta(1)`
`monthdelta(6) * 2`	`monthdelta(12)`

`x / y` (Division)

Divison von Zahlen sowie Datumsarithmetik.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Beispiele:

Ausdruck	Wert	Bemerkung
`4 / True`	`4.0`	
`False / 2`	`0.0`	
`7 / 2`	`3.5`	
`7 / 0`	`None`	Fehler
`days(1) / 24`	`timedelta(0, 3600)`	
`seconds(2) / True`	`timedelta(0, 2)`	

`x // y` (Ganzzahl-Division)

Division von Zahlen mit Rundung auf ganze Zahlen.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Beispiele:

Ausdruck	Wert	Bemerkung
`4 // True`	`4`	
`False // 2`	`0`	
`7 // 2`	`3`	
`7 // 0`	`None`	Fehler
`days(7) // 3`	`timedelta(2)`	
`monthdelta(12) // 3`	`monthdelta(4)`	

`x % y` (Modulo-Operator)

Modulo-Operator bzw. Farbmischung.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

Beispiele:

Ausdruck	Wert
`13 % 10`	`3`
`True % 2`	`1`
`6.5 % 2.5`	`1.5`
`#fff % #000`	`#fff`
`#fff8 % #000`	`#888`
`#f00a % #0f0`	`#a50`

`x and y` (Boolsche „Und“-Verknüpfung)

Das Ergebnis ist `y` wenn `bool(x)` `True` ist, ansonsten `x`.

Dabei müssen die Datentypen von `x` und `y` kompatibel sein. (`"gurk" and 42` wird beispielsweise nicht unterstützt.)

Siehe auch:

bool().

`x or y` (Boolsche „Oder“-Verknüpfung)

Das Ergebnis ist `y` wenn `bool(x)` `False` ist, ansonsten `x`.

Dabei müssen die Datentypen von `x` und `y` kompatibel sein. (`"gurk" or 42` wird beispielsweise nicht unterstützt.)

Siehe auch:

bool().

`x in y` (Enthalten-Test)

Testet ob `x` in `y` enthalten ist.

Beispiele:

Ausdruck	Wert
`"u" in "gurk"`	`True`
`"ur" in "gurk"`	`True`
`"hu" in "gurk"`	`False`
`"u" in ["gurk", "hurz", "hinz", "kunz"]`	`False`
`"hinz" in ["gurk", "hurz", "hinz", "kunz"]`	`True`

`x not in y` (Nicht-Enthalten-Test)

Testet ob `x` nicht in `y` enthalten ist.

Beispiele:

Ausdruck	Wert
`"u" not in "gurk"`	`False`
`"ur" not in "gurk"`	`False`
`"hu" not in "gurk"`	`True`
`"u" not in ["gurk", "hurz", "hinz", "kunz"]`	`True`
`"hinz" not in ["gurk", "hurz", "hinz", "kunz"]`	`False`

`x is y` (Identitäts-Test)

Der Wert ist derselbe wie bei `x == y`, jedoch muß `x` oder `y` ein literales `None` sein.

`x is not y` (Negativer Identitäts-Test)

Der Wert ist derselbe wie bei `x != y`, jedoch muß `x` oder `y` ein literales `None` sein.

`x[index]` (Index-Zugriff)

Holt das `index`-te Zeichen aus dem String `x` oder den `index`-ten Eintrag aus der Liste `x`. Negative Werte für `index` werden als relativ zum Ende des Strings bzw. der Liste interpretiert.

Beispiele:

Ausdruck	Wert
`"gurk"[0]`	`"g"`
`"gurk"[2]`	`"r"`
`"gurk"[5]`	`None`
`"gurk"[-1]`	`"k"`
`"gurk"[-3]`	`"u"`
`"gurk"[-5]`	`None`
`["gurk", "hurz", "hinz", "kunz"][0]`	`"gurk"`
`["gurk", "hurz", "hinz", "kunz"][-2]`	`"hinz"`

`x[start:stop]` (Slice-Zugriff)

Gibt einen Unterstring des Strings `x` von Index `start` bis Index `stop` zurück, bzw. gibt eine Unterliste der Liste `x` von Index `start` bis Index `stop` zurück. Das Zeichen/der Eintrag an Index `start` ist dabei Bestandteil des Ergebnisses, das Zeichen/der Eintrag an Index `stop` nicht. Negative Werte für `start` und `stop` werden als dabei als relativ zum Ende des Strings bzw. der Liste interpretiert. Wird `start` weggelassen, beginnt der Abschnitt am Anfang, wird `stop` weggelassen, endet der Abschnitt am Ende.

Beispiele:

Ausdruck	Wert
`"gurk"[1:3]`	`"ur"`
`"gurk"[-3:3]`	`"ur"`
`"gurk"[:2]`	`"gu"`
`"gurk"[-2:]`	`"rk"`
`"gurk"[3:1]`	`""`
`"gurk"[10:]`	`""`
`"gurk"[:-10]`	`""`

`not x` (Boolsche Negation)

Gibt `False` zurück wenn `bool(x)` `True` ist, ansonsten `True`.

Beispiele:

Ausdruck	Wert
`not None`	`True`
`not False`	`True`
`not True`	`False`
`not 0`	`True`
`not 42`	`False`
`not ""`	`True`
`not "gurk"`	`False`
`not "0"`	`False`
`not days(0)`	`True`
`not days(7)`	`False`
`not months(0)`	`True`
`not months(12)`	`False`
`not [1, 2, 3]`	`False`

Siehe auch:

bool().

`-x` (Unäres Minus)

Kehrt das Vorzeichen der Zahl (oder des Zeitraums) `x` um.

`BOOL`-Werte werden dabei wie die entsprechenden Ganzzahl-Werte 0 und 1 behandelt.

`x if cond else y` (Wenn/Dann-Ausdruck)

Gibt `x` zurück wenn `bool(cond)` `True` ist, ansonsten `y`.

Dabei müssen die Datentypen von `x` und `y` kompatibel sein. (`"gurk" if True else 42` wird beispielsweise nicht unterstützt.)

Siehe auch:

bool().

`x.attrname` (Attribut-Zugriff)

Die Attribute der jeweiligen Datentypen werden in *Datentypen, Attribute und Methoden* beschrieben.

`funktion(...)` (Funktionsaufruf)

Die zur Verfügung stehenden Funktionen werden in *Funktionen* beschrieben.

`x.methode(...)` (Methodenaufruf)

Die methoden der jeweiligen Datentypen werden in *Datentypen, Attribute und Methoden* beschrieben.

2.24.7 Funktionen

`today()`

Gibt das aktuellen Datum als *DATE*-Wert zurück.

`now()`

Gibt den aktuellen Zeitpunkt als *DATETIME*-Wert zurück.

`bool()`

Gibt `False` zurück.

`bool(obj)`

Konvertiert `obj` in einen Boolschen Wert. Das Ergebnis ist `False` für `None`, `False`, die Zahl 0, leere Strings und Listen sowie für leere Zeiträume und `True` für alle anderen Werte.

Beispiele:

Ausdruck	Wert
`bool(None)`	`False`
`bool(False)`	`False`
`bool(True)`	`True`
`bool(0)`	`False`
`bool(42)`	`True`
`bool("")`	`False`
`bool("gurk")`	`True`
`bool("0")`	`True`
`bool(days(0))`	`False`
`bool(days(7))`	`True`
`bool(months(0))`	`False`
`bool(months(12))`	`True`
`bool([1, 2, 3])`	`True`

`int()`

Gibt `0` zurück.

`int(obj)`

Konvertiert `obj` in einen `INT`-Wert.

Beispiele:

Ausdruck	Wert	Bemerkung
`int(False)`	`0`	
`int(True)`	`1`	
`int(42)`	`42`	
`int(42.5)`	`42`	
`int(-42.5)`	`-42`	
`int("42")`	`42`	
`int("42.5")`	`None`	„Fehler“
`int("gurk")`	`None`	„Fehler“

`float()`

Gibt `0.0` zurück.

`float(obj)`

Konvertiert `obj` in einen `NUMBER`-Wert.

Beispiele:

Ausdruck	Wert	Bemerkung
`float(False)`	`0.0`	
`float(True)`	`1.0`	
`float(42)`	`42.0`	
`float(42.5)`	`42.5`	
`float(-42.5)`	`-42.5`	
`float("42")`	`42.0`	
`float("42.5")`	`42.5`	
`float("gurk")`	`None`	„Fehler“

str()

Gibt den leeren String ("") zurück.

str(obj)

Konvertiert `obj` in einen *STR*-Wert.

Beispiele:

Ausdruck	Wert
`str(None)`	`"None"`
`str(False)`	`"False"`
`str(True)`	`"True"`
`str(42)`	`"42"`
`str(42.5)`	`"42.5"`
`str("gurk")`	`None`
`str([1, 2, 3])`	`"[1, 2, 3]"`

date(jahr, monat, tag)

Erzeugt einen `DATE`-Wert aus dem übergebenen Jahr-, Monats- und Tagangaben.

Beispiele:

Ausdruck	Wert	Bemerkung
`date(2000, 2, 29)`	`@(2000-02-29)`	29. Feb. 2000

datetime(ja, mo, ta, st, mi, se)

Erzeugt einen `DATETIME`-Wert aus dem übergebenen Jahr-, Monats-, Tag-, Stunden-, Minuten- und Sekundenangeben.

Beispiele:

Ausdruck	Wert	Bemerkung
`datetime(2000, 2, 29)`	`@(2000-02-29T)`	29. Feb. 2000 00:00
`datetime(2000, 2, 29, 12)`	`@(2000-02-29T12:00)`	29. Feb. 2000 12:00
`datetime(2000, 2, 29, 12, 34)`	`@(2000-02-29T12:34)`	29. Feb. 2000 12:34
`datetime(2000, 2, 29, 12, 34, 56)`	`@(2000-02-29T12:34:56)`	29. Feb. 2000 12:34:56

len(str)

Gibt die Länge des Strings `str` zurück.

Beispiele:

Ausdruck	Wert
`len("")`	`0`
`len("gurk")`	`4`

len(list)

Gibt die Länge der Liste `list` zurück.

Beispiele:

Ausdruck	Wert
`len([1, 2, 3])`	`3`

timedelta()

Gibt einen „leeren“ Zeitraum zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29) + timedelta()`	`@(2000-02-29)`

timedelta(tage)

Gibt einen `DATE`-Wert von `tage` Tagen zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29) + timedelta(1)`	`@(2000-03-01)`
`@(2000-02-29) - timedelta(1)`	`@(2000-02-28)`

timedelta(tage, sekunden)

Gibt einen `DATETIME`-Wert von `tage` Tagen und `sekunden` Sekunden zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29T12:34:56) + timedelta(1, 100)`	`@(2000-03-01T12:36:36)`

monthdelta()

Gibt einen *MONTHDELTA*-Wert für 0 Monate zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29) + monthdelta()`	`@(2000-02-29)`

monthdelta(n)

Gibt einen *MONTHDELTA*-Wert von `n` Monaten zurück.

Beispiele:

Ausdruck	Wert
`@(2000-01-01) + monthdelta(1)`	`@(2000-02-01)`
`@(2000-01-31) + monthdelta(1)`	`@(2000-02-29)`

years(n)

Gibt einen *MONTHDELTA*-Wert von `n` Jahren zurück.

Beispiele:

Ausdruck	Wert
`@(2000-01-01) + years(1)`	`@(2001-01-01)`
`@(2000-02-29) + years(1)`	`@(2001-02-28)`

months(n)

Gibt einen *MONTHDELTA*-Wert von `n` Monaten zurück.

Beispiele:

Ausdruck	Wert
`@(2000-01-01) + monthdelta(1)`	`@(2000-02-01)`
`@(2000-01-31) + monthdelta(1)`	`@(2000-02-29)`

days(n)

Gibt einen *DATEDELTA*-Wert von `n` Tagen zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29) + days(1)`	`@(2000-03-01)`
`@(2000-02-29) - days(1)`	`@(2000-02-28)`
`@(2000-01-31) + days(30)`	`@(2000-03-01)`
`@(2000-02-29) + days(89)`	`@(2000-05-28)`

hours(n)

Gibt einen *DATETIMEDELTA*-Wert von `n` Stunden zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29T12:34:56) + hours(1)`	`@(2000-02-29T13:34:56)`

minutes(n)

Gibt einen *DATETIMEDELTA*-Wert von `n` Minuten zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29T12:34:56) + minutes(1)`	`@(2000-02-29T13:35:56)`

seconds(n)

Gibt einen *DATETIMEDELTA*-Wert von `n` Sekunden zurück.

Beispiele:

Ausdruck	Wert
`@(2000-02-29T12:34:56) + seconds(1)`	`@(2000-02-29T13:34:57)`

md5(s)

Gibt die MD5-Prüfsumme für den String `s` zurück.

Beispiele:

Ausdruck	Wert
`md5("")`	`"d41d8cd98f00b204e9800998ecf8427e"`
`md5("gurk")`	`"4b5b6a3fa4af2541daa569277c7ff4c5"`

random()

Gibt eine zufällige Zahl zwischen `0.0` (einschließlich) und `1.0` (ausschließlich) zurück.

randrange(start, stop)

Gibt eine zufällige ganze Zahl zwischen `start` (einschließlich) und `stop` (ausschließlich) zurück.

cos(x)

Gibt den Cosinus von `x` (im Bogenmaß) zurück.

sin(x)

Gibt den Sinus von `x` (im Bogenmaß) zurück.

tan(x)

Gibt den Tangens von `x` (im Bogenmaß) zurück.

sqrt(x)

Gibt den Quadratwurzel von `x` zurück.

abs(x)

Gibt der Absolutbetrag von `x` zurück.

geo(lat, long)

Gibt einen Geo-Wert mit dem Längengrad `lat` und dem Breitengrad `long` zurück. Das `info`-Atribut ist `None`.

geo(lat, long, info)

Gibt einen Geo-Wert mit dem Längengrad `lat`, dem Breitengrad `long` und der Beschreibung `info` zurück.

dist(x, y)

Gibt den Abstand zwischen den beiden Geo-Koordinaten `x` und `y` zurück.

2.25 LivingAPI

LivingAPI bezeichnet die Gesamtmenge an Objekten die einem Anzeige-Template, einem E-Mail-Template oder anderen Templates übergeben werden.

(Im Gegensatz dazu wird *vSQL* verwendet, um Datensätze zu filtern oder auszuwählen, *bevor* sie an Templates übergeben werden.)

Von welchen Apps Daten über die LivingAPI an ein Anzeige-, Formular-, Update- oder E-Mail-Template übergeben werden, kann unter *Konfiguration* → *Erweitert* in der *Anzeige-Templates*-Maske beim jeweiligen Anzeige-Template unter der *Datenquellen*-Maske konfiguriert werden (sowie über bestimmte Abfrage-Parameter im HTTP-Request beeinflußt werden). Für E-Mail-Templates passiert dies in der *E-Mail-Templates*-Maske beim jeweiligen E-Mail-Template unter der *Datenquellen*-Maske.

2.25.1 Objekt-Typen

Die folgende Objekt-Typen stellen die zentralen Objekte in der LivingAPI dar: Apps (*App*) mit ihren Feldern (*Control*) oder Dekorationsfeldern (*LayoutControl*) sowie Datensätze (*Record*) mit den Feld-Werten (*Field*) und Anhängen zu den Datensätzen (*Attachment*) sowie Auswahl-Objekte (*LookupItem*), Objekte für hochgeladene Dateien (*File*) und für geographische Koordinaten (*Geo*).

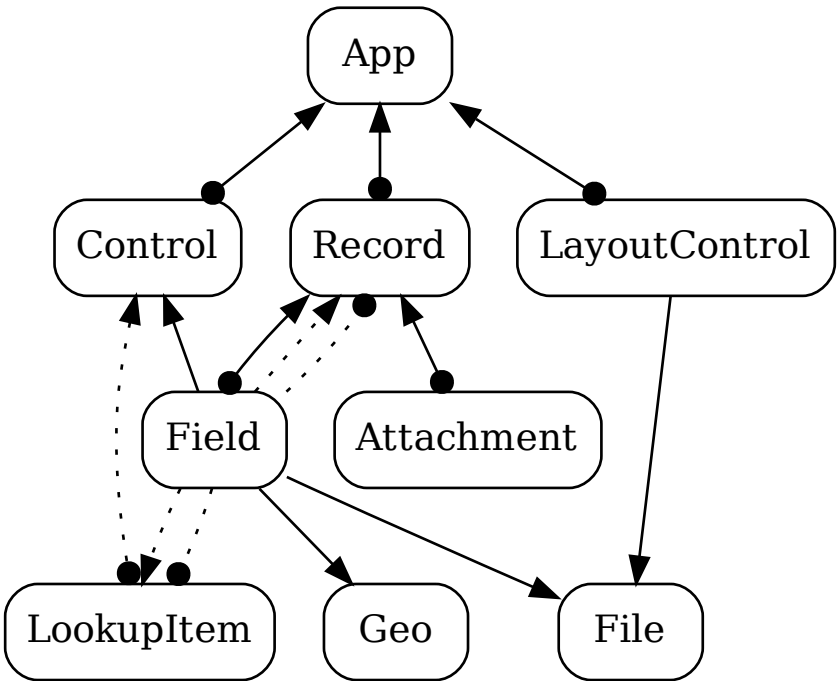


Abb. 185: Die zentralen Objekt-Typen der LivingAPI

Weiterhin werden einem Anzeige-Template im *Globals*-Objekt globale Informationen übergeben. Darüber hat der Benutzer die Möglichkeit auf Informationen über die zu verarbeitende HTTP-Anfrage zuzugreifen (*HTTPRequest*), die zu versendende HTTP-Anwort zu beeinflußen (*HTTPResponse*), und einige Dinge mehr.

Bei einem E-Mail-Template werden ebenfalls im *Globals*-Objekt globale Informationen übergeben. Das *EmailRequest*-Objekt enhält Informationen über den Kontext in dem die E-Mail-Versendung stattfindet. Das *EmailResponse*-Objekt kann dazu verwendet werden, Betreff und Adressaten der Email zu ändern.

Dem *App*-Objekt sind weiterhin einige Informationen zugeordnet: Die App-Kategorien (*Category*), die App-Parameter (*AppParameter*), die Menüs (*MenuItem*), die Panels/Kacheln (*Panel*), die Installation, die diese App erzeugt hat (*Installation*) sowie die Ansichten (*View*).

Die einzelnen Objekt-Typen werden in den folgenden Unter-Kapiteln beschrieben:

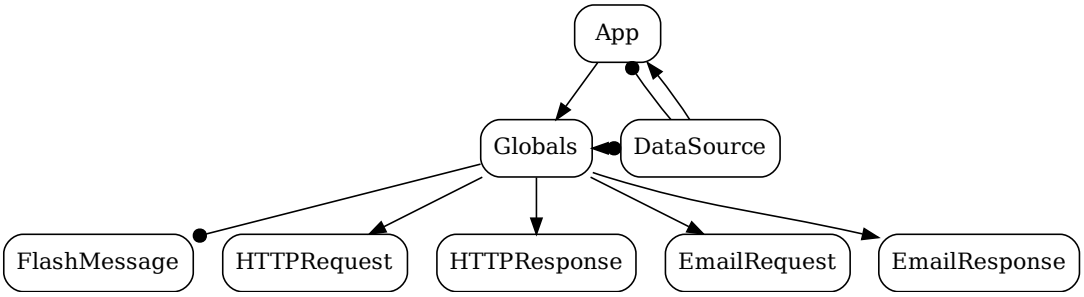


Abb. 186: Globale Informationen

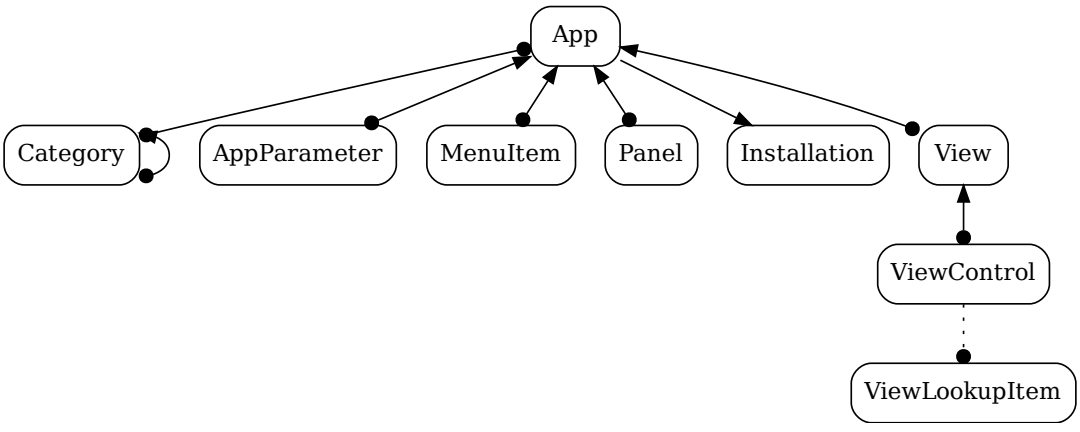


Abb. 187: Zusätzliche Information zu `Apps`

Globals

Ein `Globals`-Objekt wird einem Anzeige-, Formular-, Update- oder E-Mail-Template in der Variablen `globals` übergeben.

Diese Variable ist global, d.h. sie ist auch in aufgerufenen Unter-Templates bekannt, ohne daß `globals` beim Aufruf an diese übergeben werden muß.

Das `Globals`-Objekt enthält globale Informationen und besitzt folgende Attribute:

app
: [*App*]
Die App des aufgerufenen Anzeigetemplates.

record
: [*Record* oder `None`]
Der über die URL mitgegebene Datensatz.

version
: [String]
Die Version der LivingAPI die zur Erzeugung sämtlicher Objekte verwendet wurde.

platform
: [String]
Der Name für die installierte Version von LivingApps. Auf http://my.living-apps.de ist dies z.B. `"LivingApps"`, auf http://my.leadair.de `"LeadAir"`.

hostname
: [String]
Der Hostname des LivingApps-Server (also beispielsweise `my.living-apps.de` oder `my.leadair.de`).

user
: [*User* oder `None`]
Der aktuell eingeloggte Benutzer. `user` ist `None`, falls ein nicht authentifizierter Nutzer ein öffentliches Template nutzt.

lang
: [String]
Die zu verwendende Sprache (d.h. in dieser Sprache sollten die Ausgaben in einem Anzeige-Template erzeugt werden). Wenn ein Benutzer eingeloggt ist, ist dies die Sprache des Benutzers (ansonsten Standard-Sprache der LivingApp-Platform: `"de"` für Deutsch). Dieses Attribut kann geändert werden (z.B. wenn wegen eines `Accept-Language`-Headers eine andere Sprache verwendet werden soll).

In Formular-Templates ist `globals.lang` die Sprache der Formularvariante (nicht die bevorzugte Sprache des eingeloggten Benuzters).

p_<identifier>
: [*AppParameter*]
„Shortcut"-Attribute für die `AppParameter`-Objekte der App. Beispielsweise ist `globals.p_beispiel` äquivalent zu `globals.app.p_beispiel` (und dies wiederum zu `globals.app.params.beispiel`).

pv_<identifier>
: [Objekt]
„Shortcut"-Attribute für die Werte von `AppParameter`-Objekten der App. `globals.pv_beispiel` äquivalent zu `globals.app.pv_beispiel` (und im Endeffekt wiederum zu `globals.app.params.beispiel.value`).

templates
: [Dictionary(String → UL4-Template)]
Alle internen Templates, die in der aktuellen App definiert sind. Diese Templates können unter *Konfiguration* → *Erweitert* in der *Interne Templates*-Maske angelegt werden. Die Schlüssel des Dictionarys sind dabei der Identifizierer des Templates und die Werte die UL4-Templates selbst.

Damit ist es möglich, z.B. mehrfach benötigte Funktionalität in einem Template zu implementieren und diese Funktionalität von anderen Anzeige-Templates aus aufzurufen.

t_<identifier>
[UL4-Template]
Die internen Templates stehen auch über diese „Shortcut"-Attribute zur Verfügung. `globals.t_gurk` is beispielsweise äquivalent zu `globals.templates.gurk`.

datasources
[Dictionary(String → *DataSource*)]
Die Datenquelle des aufrufenden Templates

custom
[Objekt]
Dieses Attribut kann vom Benutzer für beliebige zusätzliche Informationen gesetzt werden.

x_<identifier>
[Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

mode
[String]
Gibt an in welchem Typ von Template man sich gerade befindet bzw. in welcher Ablaufphase dieses Templates. Mögliche Werte sind:

form/new/init
[Formular- und Update-Template]
Der erste Aufruf eines „Neu anlegen"-Formulars.

form/new/search
[Formular-Template]
Ajax-Suche für `applookup/choice`- bzw. `multipleapplookup/choice`-Felder.

form/new/failed
[Formular-Template]
Das Ausfüllen des „Neu anlegen"-Formular ist fehlgeschlagen (wegen nicht ausgefüllter Pflichtfelder etc.).

form/new/presave
[Formular-Template]
Das „Neu anlegen"-Formular wurde erfolgreich ausgefüllt. Der Datensatz wurde noch nicht gespeichert.

form/new/postsave
[Formular-Template]
Das „Neu anlegen"-Formular wurde erfolgreich ausgefüllt. Der Datensatz wurde bereits gespeichert.

form/new/input
[Update-Template]
Der Benutzer hat im „Neu"-Formular ein Eingabefeld geändert. Die Variable `identifier` enthält den Identifier des geänderten Feldes.

form/new/geo
[Update-Template]
In einen „Neu"-Formular liegt Geo-Information vor über den Standort des Benutzers vor.

form/edit/init
[Formular- und Update-Template]
Der erste Aufruf eines „Bearbeiten"-Formulars.

form/edit/search
[Formular-Template]
Ajax-Suche für `applookup/choice`- bzw. `multipleapplookup/choice`-Felder.

form/edit/failed
[Formular-Template]

Das Ausfüllen des „Bearbeiten“-Formular ist fehlgeschlagen (wegen nicht ausgefüllten Pflichtfeldern etc.).

form/edit/presave
[Formular-Template]
Das „Bearbeiten“-Formular wurde erfolgreich ausgefüllt. Der Datensatz wurde noch nicht gespeichert.

form/edit/postsave
[Formular-Template]
Das „Bearbeiten“-Formular wurde erfolgreich ausgefüllt. Der Datensatz wurde bereits gespeichert.

form/edit/input
[Update-Template]
Der Benutzer hat im „Bearbeiten“-Formular ein Eingabefeld geändert. Die Variable `identifier` enthält den Identifier des geänderten Feldes.

form/new/geo
[Update-Template]
In einen „Bearbeiten“-Formular liegt Geo-Information vor über den Standort des Benutzers vor.

view/list
[Anzeige-Template]
In einem Anzeige-Template vom Typ „Liste“.

view/detail
[Anzeige-Template]
In einem Anzeige-Template vom Typ „Detail“.

view/support
[Anzeige-Template]
In einem Anzeige-Template vom Typ „Support“.

email/text
[E-Mail-Template]
Text-Version der E-Mail.

email/html
[E-Mail-Template]
HTML-Version der E-Mail.

geo(...)
[Methode(…) → *Geo*]
Dieser Methode gibt *Geo*-Objekte zurück. Sie besitzt zwei verschiedene Signaturen:

geo(lat, long)
[Methode(Zahl, Zahl) → *Geo*]
Übergeben wird der Breitengrad `lat` und der Längengrad `long`. Die fehlende Beschreibung des Ortes (z.B. eine Adresse) wird daraus ermittelt. Beispielsweise liefert:

```
<?printx globals.geo(lat=49.9552129, long=11.5901843)?>
```

das Geo-Objekt:

```
<com.livinglogic.appdd.Geo
   lat=49.9552129
   long=11.5901843
   info='Markgrafenallee 44, 95448 Bayreuth, Deutschland'
>
```

geo(info)
[Methode(String) → *Geo*]
Übergeben wird eine Beschreibung des Ortes z.B. als Adresse. Der fehlende Längen- und Breitengrad wird daraus ermittelt. Beispielsweise liefert:

```
<?code info = "LivingLogic AG, Markgrafenallee 44, 95448 Bayreuth"?>
<?printx globals.geo(info=info)?>
```

das Geo-Objekt:

```
<com.livinglogic.appdd.Geo
   lat=49.95559
   long=11.59011
   info='LivingLogic AG, Markgrafenallee 44, 95448 Bayreuth'
>
```

Bemerkung: Wenn *Geo*-Objekte dargestellt werden, muss Open Streetmap als Datenquelle genannt werden. Mehr Informationen zur Nennung[1] und zur Lizenz[2].

`current_geo()`
: [Methode() → *Geo*]
Gibt den aktuellen Standort des Benutzers zurück (nur in Update-Templates).

`seq()`
: [Methode() → Integer]
Jeder Aufruf dieser Methode gibt eine eindeutige, fortlaufende Nummer zurück. `seq` benötigt keine Argumente.

`dist(geo1, geo2)`
: [Methode(*Geo*, *Geo*) → Zahl]
Gibt die Entfernung zwischen den zwei Geo-Koordinaten `geo1` und `geo2` in Kilometern zurück.

`scaled_url(...)`
: [Methode(...) → String]
Diese Methode liefert eine URL unter der die skalierte Version eines Bildes ausgeliefert wird. Zum Skalieren der Bilder wird imgproxy[3] verwendet.

  Die Methode unterstützt folgende Argumente:

  `image`
  : [*File* oder String]
  Entweder die absolute URL eines Bildes oder ein *File*-Objekt das ein Bild anthält. Dieses Bild wird skaliert.

  `type`
  : [String oder `None`]
  Erlaubte Werte sind: `"fit"`, `"fill"`, `"fill-down"`, `"force"` und `"auto"`. Der Standardwert ist `"fill"`.

    Für mehr Information siehe die imgproxy-Dokumentation[4].

  `width`
  : [Integer oder `None`]
  Die Zielbreite des skalierten Bilder. Der Standardwert ist `None`.

    Für mehr Information siehe die imgproxy-Dokumentation[5].

  `height`
  : [Integer oder `None`]
  Die Zielhöhe des skalierten Bildes. Der Standardwert ist `None`. `width` und `height` dürfen nicht beide `None` sein.

[1] https://www.openstreetmap.de/faq.html#lizenz
[2] https://opendatacommons.org/licenses/odbl/summary/
[3] https://imgproxy.net/
[4] https://docs.imgproxy.net/generating_the_url?id=resizing-type
[5] https://docs.imgproxy.net/generating_the_url?id=width

Für mehr Information siehe die imgproxy-Dokumentation[6].

enlarge
[Bool]
Soll das Bild vergrößert werden, wenn es kleiner als die angegebene Zielgröße ist? Der Standardwert ist `True`.

Für mehr Information siehe die imgproxy-Dokumentation[7].

gravity
[String oder `None`]
Erlabute Werte sich `"no"`, `"so"`, `"ea"`, `"we"`, `"noea"`, `"nowe"`, `"soea"`, `"sowe"`, `"ce"` und `"sm"`. Der Standardwert ist `"sm"`.

Für mehr Information siehe die imgproxy-Dokumentation[8].

quality
[Integer oder `None`]
Wird dieses Argument angegeben, muß der Wert zwischen 0 und 100 liegen. Der Standardwert ist `None`.

Für mehr Information siehe die imgproxy-Dokumentation[9].

rotate
[Integer oder `None`]
Rotiert das Bild. Wird dieses Argument angegeben, muß der Wert ein ganzzahliges Vielfaches von 90 sein.

Für mehr Information siehe die imgproxy-Dokumentation[10].

blur
[Zahl oder `None`]
Macht das Bild unschwarf.

Für mehr Information siehe die imgproxy-Dokumentation[11].

sharpen
[Zahl oder `None`]
Schärft das Bild.

Für mehr Information siehe die imgproxy-Dokumentation[12].

format
[String oder `None`]
Das Bildformat (PNG, GIF, JPEG usw.) des skalierten Bildes.

Für mehr Information siehe die imgproxy-Dokumentation[13].

cache
[Bool]
Wird `True` gegeben, so wird eine URL zurückgeliefert, die das skalierte Bild zwischenspeichert, sodaß es bei einen erneuten Aufruf nicht nochmal skaliert werden muß. Bei `False` wird das Bild bei jeden Aufruf skaliert.

„Flash“-Meldungen werden dazu verwendet um dem Benutzer über das Ergebnis von Aktionen zu informieren, die er auf der vorhergehenden Seite durchgeführt hat. Z.B. kann es in einer Liste von Datensätzen zu jeden Datensatz einen Button geben, der über ein Formular/einen POST-Request diesen Datensatz ändert. Das UL4-Template das

[6] https://docs.imgproxy.net/generating_the_url?id=height
[7] https://docs.imgproxy.net/generating_the_url?id=enlarge
[8] https://docs.imgproxy.net/generating_the_url?id=gravity
[9] https://docs.imgproxy.net/generating_the_url?id=quality
[10] https://docs.imgproxy.net/generating_the_url?id=rotate
[11] https://docs.imgproxy.net/generating_the_url?id=blur
[12] https://docs.imgproxy.net/generating_the_url?id=sharpen
[13] https://docs.imgproxy.net/generating_the_url?id=format

diese Änderung durchführt setzt dann eine Flash-Meldung und leitet den Benutzer wieder zur Listenansicht zurück. Die Listenansicht zeigt dann diese Flash-Meldung an. D.h. die Flash-Meldungen bleiben auch über Requests hinweg erhaltet.

flashes()
[Methode() → Liste(*FlashMessage*)]
`flashes` gibt eine Liste von „Flash"-Meldungen zurück. Beim Aufruf von `flashes()` wird diese Liste automatisch geleert (d.h. der nächste Aufruf gibt eine leere Liste zurück).

flash_info(title, message)
[Methode(String, String oder `None`) → `None`]
Erzeugt ein neues `FlashMessage`-Objekt von Typ `info` und fügt dieses zur Liste der „Flash"-Meldungen hinzu. Der Parameter `title` muß ein String sein, `message` ein String oder `None`.

flash_notice(title, message)
[Methode(String, String oder `None`) → `None`]
Erzeugt ein neues `FlashMessage`-Objekt von Typ `notice` und fügt dieses zur Liste der „Flash"-Meldungen hinzu. Der Parameter `title` muß ein String sein, `message` ein String oder `None`.

flash_warning(title, message)
[Methode(String, String oder `None`) → `None`]
Erzeugt ein neues `FlashMessage`-Objekt von Typ `warning` und fügt dieses zur Liste der „Flash"-Meldungen hinzu. Der Parameter `title` muß ein String sein, `message` ein String oder `None`.

flash_error(title, message)
[Methode(String, String oder `None`) → `None`]
Erzeugt ein neues `FlashMessage`-Objekt von Typ `error` und fügt dieses zur Liste der „Flash"-Meldungen hinzu. Der Parameter `title` muß ein String sein, `message` ein String oder `None`.

Die folgenden Methoden können dazu verwendet werden die Ausführung eines Templates zu protokollieren. Die erzeugten Meldungen sind beim jeweiligen Anzeige-Template unter *Meldungen in aktueller Version* zu finden.

Alle Methoden unterstützen eine beliebige Anzahl von Argumenten. Ein Argument, das nicht vom Typ String ist wird mittels der UL4-Funktion `repr` in einen String konvertiert.

log_debug(*args)
[Methode(**Objekt) → `None`]
Erzeugt eine neue Meldung vom Typ `debug`.

log_info(*args)
[Methode(**Objekt) → `None`]
Erzeugt eine neue Meldung vom Typ `info`.

log_notice(*args)
[Methode(**Objekt) → `None`]
Erzeugt eine neue Meldung vom Typ `notice`.

log_warning(*args)
[Methode(**Objekt) → `None`]
Erzeugt eine neue Meldung vom Typ `warning`.

log_error(*args)
[Methode(**Objekt) → `None`]
Erzeugt eine neue Meldung vom Typ `error`.

In Anzeige-Templates haben die beiden Attribute `request` und `response` folgende Bedeutung:

request
[*HTTPRequest*]
Informationen über die HTTP-Anfrage über die das Anzeige-Template aufgerufen wurde.

response
[*HTTPResponse*]
Informationen über die HTTP-Antwort über die das Anzeige-Template ausgeliefert werden wird.

Innerhalb von Email-Templates exisistieren die beiden Attribute `request` und `response` ebenfalls, haben jedoch eine andere Bedeutung:

request
: [None]
Wird für Email-Templates nicht benötigt und ist immer `None`

response
: [*EmailResponse*]
Enthält den Betreff der zu versendenden E-Mail sowie die E-Mail-Adressaten. Diese Informationen können vom E-Mail-Template geändert werden.

Die folgenden Methoden geben die relativen URLs der aus der Web-Oberfläche bekannten Navigations-Links zurück.

my_apps_url()
: [Methode() → String]
Gibt die relative URL zur „Meine Apps“-Seite zurück.

Beispiel:

```
<?print app.my_apps_url()?>
```

erzeugt:

```
/apps.htm
```

my_tasks_url()
: [Methode() → String]
Gibt die relative URL zur „Aufgaben“-Seite zurück.

Beispiel:

```
<?print app.my_tasks_url()?>
```

erzeugt:

```
/xist4c/web/aufgaben_id_393_.htm
```

catalog_url()
: [Methode() → String]
Gibt die relative URL zur „Katalog“-Seite zurück.

Beispiel:

```
<?print app.catalog_url()?>
```

erzeugt:

```
/katalog/home.htm
```

chats_url()
: [Methode() → String]
Gibt die relative URL zur „Chats“-Seite zurück.

Beispiel:

```
<?print app.chats_url()?>
```

erzeugt:

```
/chats.htm
```

profile_url()
[Methode() → String]
Gibt die relative URL zur „Profil“-Seite zurück.

Beispiel:

```
<?print app.profile_url()?>
```

erzeugt:

```
/profil/index.htm
```

account_url()
[Methode() → String]
Gibt die relative URL zur „Account“-Seite zurück.

Beispiel:

```
<?print app.account_url()?>
```

erzeugt:

```
/account.htm
```

logout_url()
[Methode() → String]
Gibt die relative URL des Logout-Links zurück.

Beispiel:

```
<?print app.logout_url()?>
```

erzeugt:

```
/login.htm?logout=standardCug
```

HTTPRequest

Ein `HTTPRequest`-Objekt beinhaltet Informationen über die HTTP-Anfrage über die das Anzeige-Template aufgerufen wurde und ist zugänglich als `globals.request`. Es besitzt folgende Attribute:

method
[String]
Die HTTP-Methode über die das Anzeige-Template aufgerufen wurde. Dies kann entweder `"get"` oder `"post"` sein.

headers
[Dictionary(String → String)]
Die HTTP-Header, die bei der HTTP-Anfrage übergeben wurden. Die Dictionary-Schlüssel sind die Header-Namen (Groß-/Kleinschreibung wird dabei ignoriert) und die Werte sind die Werte der Header. Beispielsweise gibt

```
<?print globals.request.headers["Accept"]?>
```

folgendes aus:

```
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
```

Bemerkung: Im Header `Cookie` sind nur Cookies vorhanden, deren Namen mit `custom` beginnt. Alle anderen Cookies werden herausgefiltert. D.h. wenn Sie in Ihren eigenen Templates Cookies setzen wollen, muß deren Namen mit `custom` beginnen, wenn Sie diese Cookies im Request wieder abfragen wollen.

params
[Dictionary(String → (String oder *File*))]
Die Parameter, die bei der HTTP-Anfrage übergeben wurden. Beispielsweise gibt bei folgender URL

```
http://my.living-apps.de/gateway/apps/1234567890abcdef1234567890abcd/
↪fedcba0987654321fedcba09876543?template=gurk&modus=normal
```

der Ausdruck `<?print globals.request.params.modus?>` den Wert `normal` aus.

Wenn über ein Formular mit dem `enctype` `multipart/form-data` eine Datei hochgeladen wurde, so ist der Wert des entsprechenden Parameters ein *File*-Objekt. Diese Objekt kann dann z.B. direkt benutzt werden um einen Datensatz anzulegen:

```
<?code personapp.insert(
   firstname=globals.request.params.firstname,
   lastname=globals.request.params.lastname,
   portrait=globals.request.params.portrait,
)?>
```

dabei wurde dann z.B. folgendes Formular verwendet:

```
<form enctype="multipart/form-data" method="post" action="/gateway/apps/
↪1234567890abcdef1234567890abcd">
   <input type="hidden" name="template" value="gurk">
   <label>First name <input type="text" name="firstname" /></label>
   <label>Last name <input type="text" name="lastname" /></label>
   <label>Portrait <input type="file" name="portrait" /></label>
</form>
```

HTTPResponse

Ein `HTTPResponse`-Objekt enhält Informationen über die HTTP-Antwort über die das Anzeige-Template ausgeliefert werden wird und ist zugänglich als `globals.response`. Das Anzeige-Template kann die Attribute diese Objekts ändern, um die Antwort zu beinflussen. Die Attribute sind:

status
[Integer]
Der HTTP-Statuscode

headers
[Dictionary(String → String)]
Die HTTP-Header, die bei der Antwort geschickt werden (Groß-/Kleinschreibung wird dabei bei den Schlüsseln ignoriert).

send_file(file)
[Methode(*File*) → `None`]
Mit dieser Methode können Sie die LivingAPI anweisen, statt der Ausgabe, die Ihr Anzeige-Template erzeugt, ein *File*-Objekt auszuliefern.

Wenn Sie unter *Uploads* konfiguriert haben, daß nur Benutzer mit Rechten an der App Dateien herunterladen können, können Sie in einem Anzeige-Template Ihren eigenen Zugriffsschutz implementieren und dann mittels `send_file()` die gewünschte Datei ausliefern.

`send_file()` kann mehrmal aufgerufen werden. Das Argument muß ein *File*-Objekt sein. Alle übergebenen *File*-Objekte werden in einer Liste gesammelt. Ist am Ende des Anzeige-Templates diese Liste leer, so

wird die Ausgabe des Anzeige-Templates ausgeliefert. Enthält die Liste ein *File*-Objekt, so wird dieses File ausgeliefert. Enthält die Liste mehrere Files, so werden diese Files in ein ZIP-Archiv verpackt, und dieses ZIP-Archiv wird ausgeliefert.

Werden mittels `send_file()` Dateien ausgeliefert, so werden trotzdem die in `headers` gesetzten Header berücksichtigt, mit Ausnahme des `"Content-Type"`-Headers, der automatisch auf den richtigen Datei-Typ gesetzt wird und auch nicht verändert werden kann. Außerdem wird der `"Content-Disposition"`-Header gesetzt um dem Browser den Original-Dateinamen bekannt zu machen.

clear_files()
[Methode() → `None`]
Leert die Liste der zu auszuliefernden Files, d.h. danach wird wieder die Ausgabe des Anzeige-Templates ausgeliefert.

Beispiele

Ein Redirect läßt sich mit `status` und `headers` beispielsweise folgendermaßen implementieren:

```
<?code globals.response.status = 303?>
<?code globals.response.headers["Location"] = "http://www.example.org/"?>
```

Sie können mit folgendem Code Ihr eigenes Account-Icon ausliefern:

```
<?code globals.response.send_file(globals.user.avatar_large)?>
```

EmailRequest

In einem E-Mail-Template enhält das `EmailRequest`-Objekt (zugänglich als `globals.request`) Informationen über den Kontext in dem die E-Mail-Versendung stattfindet.

bodytext
[String oder `None`]
Wenn das E-Mail-Template in einer Datenaktion verwendet wird, so ist `globals.request.bodytext` der Wert der durch das Feld *Inhalt (Text)* in der Email-Versendungs-Anweisung definiert wurde.

bodyhtml
[String oder `None`]
Wenn das E-Mail-Template in einer Datenaktion verwendet wird, so ist `globals.request.bodyhtml` der Wert der durch das Feld *Inhalt (HTML)* in der Email-Versendungs-Anweisung definiert wurde.

Wenn das E-Mail-Template verwendet wird, um im Datenmanagement eine E-Mail zu versenden, so ist `globals.request.bodyhtml` die im *E-Mail versenden*-Dialog eingegebene E-Mail-*Nachricht*.

EmailResponse

In einem E-Mail-Template enhält das `EmailResponse`-Objekt (zugänglich als `globals.response`) Informationen über die zu versendende E-Mail. Diese Informationen können vom E-Mail-Template geändert werden, um die zu versendende E-Mail zu beinflussen. Die Attribute sind:

from
[String]
Der Name der in der Absender-Adresse auftaucht (die E-Mail-Adresse wird aber immer ein LivingApps-E-Mail-Adresse sein).

to
[String oder `None`]
Der Empfänger der Email. Dies kann eine einfache E-Mail-Adresse sein (z.B. `person@example.org`) oder eine Kombination aus Name der Person und E-Mail-Adresse (entweder in der Form `Vorname`

`Nachame <person@example.org>` oder `person@example.org (Vorname Nachname)`. Außerdem können mehrere E-Mail-Adressen mit Komma getrennt verwendet werden.

Es ist weiterhin möglich das Attribut `to` auf `None` zu setzen. Damit wird die Versendung der E-Mail komplett unterdrückt.

cc
[String oder `None`]
E-Mail-Adressen die die E-Mail als „Kopie“ (d.h. zur Kenntnisnahme) zugesendet bekommen sollen. Es sind die selben Werte wie beim Attribut `to` erlaubt. `None` bedeutet jedoch lediglich, daß niemand diese E-Mail als Kopie erhält, nicht daß sie nicht versendet wird.

bcc
[String oder `None`]
E-Mail-Adressen die die E-Mail als „Blindkopie“ zugesendet bekommen sollen (d.h. zur Kenntnisnahme, jedoch ohne daß andere Empfänger dies sehen können). Es sind die selben Werte wie beim Attribut `cc` erlaubt.

replyto
[String oder `None`]
Eine E-Mail-Adresse an die eine Antwort-E-Mail adressiert werden soll, wenn der Empfänger der E-Mail in seinem E-Mail-Programm die E-Mail beantwortet. Der Standardwert ist hier `None`, was bedeuten würde, daß die Antwort-E-Mail an den automatischen LivingApps-E-Mail-Account gesendet würde. Daher empfielt es sich immer das Attribut `replyto` zu setzen.

subject
[String]
Der Betreff der E-Mail.

attachments
[Liste(*File*)]
Die Attachments, zu dieser E-Mail. Wenn in der Konfiguration ein Attachment konfiguriert ist, so enthält diese Liste dieses Attachment, ansonsten ist die Liste leer. Das E-Mail-Template kann diese Liste auch ändern. Mehrere Attachments in der Liste werden ebenso unterstützt.

Alle diese Attribute sind durch die Konfiguration des E-Mail-Templates vorinitialisiert können jedoch durch das E-Mail-Template geändert werden.

FlashMessage

Ein `FlashMessage`-Objekt stellt eine Meldung dar, die dem Benutzer „bei nächster Gelegenheit“ angezeigt werden soll. Damit kann z.B. ein Anzeige-Template das einen Datensatz ändert eine Meldung hinterlassen und den Benutzer zu einer Übersichtsseite weiterleiten. Die Übersichtsseite kann dann die `FlashMessage` anzeigen, um den Benutzer über das Ergebnis seiner letzten Aktion zu informieren.

Ein `FlashMessage`-Objekt hat folgende Attribute:

timestamp
[Datum]
Der Zeitpunkt zu dem das `FlashMessage`-Objekt angelegt wurde (und damit der Zeitpunkt zu dem die durch diese Meldung beschriebene Aktion stattgefunden hat).

type
[String]
Der Meldungstyp. Mögliche Werte sind:

info
Diese Meldung beschreibt eine erfolgreich durchgeführte Aktion.

notice
Diese Meldung beschreibt eine erfolgreich durchgeführte Aktion, bei der aber Seiteneffekte aufgetreten sind, die dem Benutzer zur Kenntnis gebracht werden sollen. Es sind aber keine weiteren Aktionen des Benutzer erforderlich.

`warning`
: Diese Meldung beschreibt eine erfolgreich durchgeführte Aktion, bei der unvorhergesehene Umstände auftreten sind, die möglicherweise das Eingreifen des Benutzers erfordern.

`error`
: Diese Meldung weist den Benutzer darauf hin, das die von ihm durchgeführte Aktion fehlgeschlagen ist.

`title`
: [String]
Eine Überschrift für die Meldung. Dieser String wird als HTML interpretiert, d.h. er darf HTML-Elemente enthalten (wie z.B. <b> oder etc.) Es sollten nur Inline-Element verwendet werden.

`message`
: [String oder `None`]
Der Meldungstext. Auch dieser String wird als HTML interpretiert, und sollte nur Inline-Element beinhalten.

DataSource

Ein `DataSource`-Objekt bildet eine vom Benutzer konfigurierte Datenquelle ab. Eine solche Datenquelle kann unter *Konfiguration* → *Erweitert* in der *Anzeige-Templates*-Maske für ein Anzeige-Template angelegt werden. Darüber können Sie in der *Dokumentation zur Datenquelle* mehr lesen. Ein `DataSource`-Objekt besitzt folgende Attribute:

`id`
: [String]
Der interne Datenbank-Identifizierer für diese Datenquelle.

`identifier`
: [String]
Der vom Benutzer vergebene Identifizierer für diese Datenquelle.

`app`
: [*App* oder `None`]
Die App die für diese Datenquelle konfiguriert wurde, oder `None` wenn diese Datenquelle für alle Apps konfiguriert wurde (d.h. wenn in der Konfiguration der Datenquelle bei *App* keine App ausgewählt wurde).

`apps`
: [Dictionary(String → *App*)]
Alle Apps die zu dieser Datenquelle gehören. Schlüssel sind die Datenbank-Identifizierer der Apps und Werte die *App*-Objekte.

Ist in der Konfiguration der Datenquelle bei *App* eine App ausgewählt (und *Kopien einbeziehen?* nicht gesetzt), so beinhaltet `apps` nur diese eine App (dies kann natürlich auch die App sein, zu der die Datenquelle selber gehört). Ist *Kopien einbeziehen?* gesetzt, so beinhaltet `apps` dieses Basis-App und alle ihre Kopien und wenn keine App bei *App* ausgewählt wurde, beinhaltet `apps` *alle* Apps auf die der Benutzer Zugriff hat.

App

Ein `App`-Objekt stellt eine Applikation dar und hat folgende Attribute:

`id`
: [String]
Der eindeutige Datenbank-Identifizierer der App.

`globals`
: [*Globals*]
Verweist auf das `Globals`-Objekt, das globale Informationen enthält.

`name`
: [String]
Der Name der App.

`description`
: [String oder `None`]
Die Beschreibung der App.

`lang`
: [String]
Die Sprache in der die App angezeigt wird (beispielsweise `"de"` für Deutsch und `"en"` für Englisch).

`image`
: [*File* oder `None`]
Das App-Icon in Original-Größe.

Wird das Icon in einer anderen Größe benötigt, kann *scaled_url(...)* benutzt werden.

`private_uploads`
: [Bool]
Gibt an, ob auf hochgeladene Dateien dieser App nur von autorisierten Benutzern zugegriffen werden darf.

`createdby`
: [*User*]
Der Besitzer der App.

`createdat`
: [Datum]
Der Zeitpunkt zu dem die App erstellt wurde.

`updatedby`
: [*User* oder `None`]
Der Benutzer, der die App zuletzt geändert hat. (Wurde die App noch nicht geändert, so ist `updatedby` `None`.)

`updatedat`
: [Datum oder `None`]
Der Zeitpunkt, zu dem die App das letzte Mal geändert wurde. (Wurde die App noch nicht geändert, so ist `updatedat` `None`.)

`controls`
: [Dictionary(String → *Control*)]
Die Definition der Felder der App. Dieses Dictionary ist sortiert, d.h. beim Durchlauf werden die Felder in der Reihenfolge durchlaufen in der sie angelegt wurden. Die Schlüssel in diesem Dictionary sind die Feld-Identifizierer und die Werte sind `Control`-Objekte.

Normalerweise beinhaltet `controls` alle Felder des Datensatzes, außer wenn in der Datenquellen-Konfiguration *Nur Listenfelder?* gesetzt ist, dann werden nur diejenigen Felder übernommen, die unter *Konfiguration → Liste* ausgewählt wurden.

`c_<identifier>`
: [*Control*]
`Control`-Objekte stehen auch über sogenannte „Shortcut"-Attribute zur Verfügung. D.h. dass beispielsweise das Feld `vorname` das normalerweise unter `app.controls.vorname` zu finden ist, auch direkt als `app.c_vorname` zur Verfügung steht.

`records`
: [Dictionary(String → *Record*) oder `None`]
Die Datensätze der App. Dies sind evtl. nicht alle Datensätze, wenn der Benutzer nur eingeschränkte Zugriffrechte besitzt. Dieses Dictionary ist sortiert, die Reihenfolge kann in der Konfiguration der Datenquellen konfiguriert werden. Die Schlüssel des Dictionarys sind die Datensatz-Identifizierer und die Werte sind `Record`-Objekte.

Wenn in der Datenquellen-Konfiguration *Nur Anzahl?* gesetzt ist, ist `records` `None`.

`recordcount`
: [Integer oder `None`]
Wenn in der Datenquellen-Konfiguration *Nur Anzahl?* gesetzt ist, beinhaltet `recordcount` die Anzahl der Datensätze (ansonsten ist `recordcount` `None`).

`installation`
: [*Installation* oder `None`]
Wenn die Aplikation durch einen Installationsvorgang erzeugt wurde, ist `installation` ein `Installation`-Objekt. Ansonsten ist `installation` `None`.

`templates`
: [Dictionary(String → UL4-Template)]
Alle internen Templates, die in der App definiert sind. Diese Templates können unter *Konfiguration* → *Erweitert* in der *Interne Templates*-Maske angelegt werden. Die Schlüssel des Dictionarys sind dabei der Identifizierer des Templates und die Werte die UL4-Templates selbst.

Haben Sie als *Datenquelle* nicht die aktuelle, sondern eine andere App ausgewählt, haben Sie hiermit die Möglichkeit, auf die internen Templates dieser in der Datenquelle konfigurierten App mit zuzugreifen.

`t_<identifier>`
: [UL4-Template]
`Template`-Objekte stehen auch über sogenannte „Shortcut"-Attribute zur Verfügung. D. h., dass beispielsweise das Template `gurk`, das normalerweise unter `app.templates.gurk` zu finden ist, auch direkt als `app.t_gurk` zur Verfügung steht. Oder als `datasources.beispiel.app.t_gurk` wenn Sie als Datenquelle eine andere App ausgewählt haben.

`categories`
: [Dictionary(String → *Category*) oder `None`]
Wenn in der Datenquellen-Konfiguration *Kategorien* etwas ausgewählt ist, enthält `categories` die Kategorien, denen diese App zugeordnet ist. Die Schlüssel des Dictionarys sind die internen Datenbank-Identifizierer der Kategorie und die Werte sind `Category`-Objekte.

Ist bei *Kategorien* nichts ausgewählt, ist `categories` `None`.

`params`
: [Dictionary(String → *AppParameter*)]
Die Parameter der Applikation (die unter *Konfiguration* → *Erweitert* in der *Parameter*-Maske angelegt werden können). Die Schlüssel des Dictionarys sind die Identifizierer der Parameter. Die Werte sind `AppParameter`-Objekte.

`p_<identifier>`
: [*AppParameter*]
`AppParameter`-Objekte stehen auch über sogenannte „Shortcut"-Attribute zur Verfügung. D. h., dass beispielsweise der App-Parameter `beispiel`, der normalerweise unter `app.params.beispiel` zu finden ist, auch direkt als `app.p_beispiel` zur Verfügung steht.

`pv_<identifier>`
: [Objekt]
Damit kann direkt auf den Wert eines``AppParameter``-Objekts zugegriffen werden. D. h. `app.pv_beispiel` ist äquivalent zu `app.p_beispiel.value` (was wiederum zu `app.params.beispiel.value` äquivalent ist).

`views`
: [Dictionary(String → *View*)]
Die verschiedenen Formularvarianten der Applikation (die unter *Konfiguration* → *Eingabe* bei *Formularvarianten* angelegt werden können). Die Schlüssel des Dictionarys sind die Identifizierer der Views. Die Werte sind `View`-Objekte.

`active_view`
: [*View*]
Die aktive Formularvariante. Ist `active_view` gesetzt (was durch Ändern dieses Attributs im Template getan werden kann, bzw. bei Formular-Templates automatisch passiert), werden beim Setzen von Feld-Werten die in dieser Variante definierten Feld-Restriktionen berücksichtigt (`minlength` und `maxlength` für String-Felder, `required` für alle).

Wird das Attribut `active_view` gesetzt kann als Wert sowohl ein *View*-Objekt verwendet werden, als auch die `id` eines *View*-Objekts. Dieses *View*-Objekt muß zu dieser App gehören.

datasource
[*DataSource*]
Das *DataSource*-Objekt das diese App beinhaltet, oder `None`, wenn es für diese App kein *DataSource*-Objekt gibt. D.h. es gibt für jede App `a` immer `a.datasource is None or `a.datasource.app is a`.

layout_controls
[Dictionary(String → *LayoutControl*) oder `None`]
Die Layout-Controls des aktiven Views. Dazu muß es einen aktiven *View* geben und in der Datenquelle muß bei *Felder Alle Felder und Layout-Felder* ausgewählt sein. Ansonsten ist `layout_controls` `None`.

lc_<identifier>
[*LayoutControl*]
„Shortcut"-Attribut zum Zugriff auf die Layout-Controls des aktiven Views. `app.lc_beispiel` ist äquivalent zu `app.layout_controls.beispiel`.

favorite
[Bool]
Gibt an, ob der aktuelle eingeloggte Benutzer diese App als Favorit festgelegt hat.

menus
[Dictionary(String → *MenuItem*) oder `None`]
Die zu dieser App konfigurierten Menüs. Die Schlüssel in diesem Dictionary sind die Identifizierer und die Werte sind `MenuItem`-Objekte.

panels
[Dictionary(String → *Panel*) oder `None`]
Die zu dieser App konfigurierten Panels. Die Schlüssel in diesem Dictionary sind die Identifizierer und die Werte sind `Panel`-Objekte.

insert(values)**
[Methode(**Objekt) → *Record*]
Mithilfe von `insert()` kann in der App ein neuer Datensatz angelegt werden. Die Parameter müssen per Schlüsselwort übergeben werden. Der Parametername ist dabei jeweils der Feld-Identifizierer des zu setzenden Feldes. Beispielsweise kann in eine Personen-App mit den Feldern `vorname`, `nachname` und `geburtstag` mittels folgendem Aufruf ein Datensatz eingefügt werden:

```
<?code record = app.insert(
   vorname="Max",
   nachname="Mustermann",
   geburtsdatum=@(2000-02-29),
)?>
```

Der zurückgegebene Datensatz wird mit den übergebenen Feldwerten initialisiert und das Attribut `id` (der eindeutige Datensatz-Identifizierer) ist gesetzt. Jedoch spiegelt der Datensatz keinerlei Änderungen wieder, die evtl. vom System über Datenaktionen durchgeführt wurden.

new_embedded_url(params)**
[Methode(**Objekt) → String]
Gibt die absolute URL zurück für das Eingabe-Formular zum Anlegen neuer Datensätze dieser App. Bei dieser URL ist das Formular in den üblichen LivingApps-Rahmen eingebettet und der Benutzer muß eingeloggt sein um es benutzen zu können.

Mittels `params` können der URL zusätzliche Parameter übergeben werden. Als Werte werden sowohl Strings unterstützt als auch Listen von Strings (in diesem Fall wird der Parameter mehrmals an die URL angefügt). Außerdem wird beim Wert `None` der Parameter ignoriert.

Beispiel 1:

```
<?print app.new_embedded_url()?>
```

erzeugt:

```
https://my.living-apps.de/dateneingabe/1234567890abcdef12345678/new
```

Beispiel 2:

```
<?print app.new_embedded_url(view="abcdef12345678901234")?>
```

erzeugt:

```
https://my.living-apps.de/dateneingabe/1234567890abcdef12345678/new?
↪view=abcdef12345678901234
```

Beispiel 3:

```
<?print app.new_embedded_url(a=["17", 23], b=None, c=[today(), None])?>
```

erzeugt:

```
https://my.living-apps.de/dateneingabe/1234567890abcdef12345678/new?a=17&a=23&
↪c=2022-11-10
```

new_standalone_url(params)**

[Methode(**Objekt) → String]
Gibt die absolute URL zurück für das Eingabe-Formular zum Anlegen neuer Datensätze dieser App. Bei dieser URL ist das Formular nicht in den üblichen LivingApps-Rahmen eingebettet sondern unabhängig und kann auch von nicht eingeloggten Benutzern ausgefüllt werden.

Mittels `params` können der URL zusätzliche Parameter übergeben werden.

Beispiel 1:

```
<?print app.new_standalone_url()?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/new
```

Beispiel 2:

```
<?print app.new_standalone_url(view="abcdef12345678901234")?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/new?
↪view=abcdef12345678901234
```

Beispiel 3:

```
<?print app.new_standalone_url(a=["17", 23], b=None, c=[today(), None])?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/new?a=17&a=23&
↪c=2022-11-10
```

template_url(identifier, record=None, /, **params)

[Methode(String, *Record* oder `None`, **Objekt) → String]
Gibt die absolute URL zurück für ein Anzeige-Template mit dem Identifizierer `identifier`. Ist `record` `None` so wird ein Link zu einem Listen-Template generiert, wird für `record` ein Datensatz übergeben, so wird ein Link zu einem Detail-Template generiert (Die verlinkte App ist dabei die App für die diese Methode aufgerufen wird, nicht die App zu der der Datensatz gehört).

Mittels `params` können der URL zusätzliche Parameter übergeben werden.

Beispiel 1:

```
<?print app.template_url("beispiel")?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678?template=beispiel
```

Beispiel 2:

```
<?print app.template_url("beispiel", r)?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/
↪0987654321fedcba1234?template=beispiel
```

Beispiel 3:

```
<?print app.template_url("beispiel", x=17, y=23)?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678?
↪template=beispiel&x=17&y=23
```

home_url()

[Methode() → String]
Gibt die absolute URL zur Detail-Seite einer App zurück.

Beispiel:

```
<?print app.home_url()?>
```

erzeugt:

```
https://my.living-apps.de/apps/1234567890abcdef12345678.htm
```

datamanagement_url()

[Methode() → String]
Gibt die absolute URL zur Datenmanagement-Seite einer App zurück.

Beispiel:

```
<?print app.datamanagement_url()?>
```

erzeugt:

```
https://my.living-apps.de/_id_36_.htm?uuid=1234567890abcdef12345678&
↪dId=1234567890abcdef12345678&resetInfo=true&templateIdentifier=created_
↪1234567890abcdef12345678
```

import_url()

[Methode() → String]
Gibt die absolute URL zur Import-Seite einer App zurück.

Beispiel:

```
<?print app.import_url()?>
```

erzeugt:

```
https://my.living-apps.de/import-export/1234567890abcdef12345678.htm
```

tasks_url()
[Methode() → String]
Gibt die absolute URL zur Aufgaben-Seite einer App zurück.

Beispiel:

```
<?print app.tasks_url()?>
```

erzeugt:

```
https://my.living-apps.de/_id_1073_.htm?uuid=1234567890abcdef12345678&
↪dId=1234567890abcdef12345678&p_tpl_uuid=1234567890abcdef12345678&
↪resetInfo=true&templateIdentifier=created_task_1234567890abcdef12345678
```

datamanagement_config_url()
[Methode() → String]
Gibt die absolute URL zur Datenmanagement-Konfigurations-Seite einer App zurück.

Beispiel:

```
<?print app.datamanagement_config_url()?>
```

erzeugt:

```
https://my.living-apps.de/datenmanagement-konfigurieren/1234567890abcdef12345678.
↪htm
```

permissions_url()
[Methode() → String]
Gibt die absolute URL zur Berechtigungs-Seite einer App zurück.

Beispiel:

```
<?print app.permissions_url()?>
```

erzeugt:

```
https://my.living-apps.de/_id_833_.htm?uuid=1234567890abcdef12345678&
↪dId=1234567890abcdef12345678&resetInfo=true
```

datamanageview_url(identifier)
[Methode(String) → String]
Gibt die absolute URL zu einer Datenauswertungs-Seite im Datenmanagement einer App zurück.

Beispiel:

```
<?print app.datamanageview_url("beispiel")?>
```

erzeugt:

```
https://my.living-apps.de/_id_36_.htm?uuid=1234567890abcdef12345678&
↪dId=1234567890abcdef12345678&resetInfo=true&templateIdentifier=created_
↪1234567890abcdef12345678_datamanage_master_beispiel
```

seq()
[Methode() → Number]
Gibt einen fortlaufenden Integer-Wert zurück, der innerhalb dieser App eindeutig ist.

custom
: [Objekt]
Dieses Attribut kann vom Benutzer für beliebige zusätzliche Informationen gesetzt werden.

x_<identifier>
: [Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

Neue Datensatzobjekte können angelegt werden, indem das App-Objekt aufgerufen wird. Feld-Werte können als Keyword-Argumente übergeben werden. So erzeugt z. B. `datasources.beispiel.app()` ein leeres Datensatzobjekt zur App „Beispiel“. Besitzt diese App die Felder `vorname`, `nachname` und `geburtstag` kann mittels folgendem Aufruf ein Datensatz-Objekt mit den entsprechenden Feld-Werten angelegt werden:

```
<?code record = app(
   vorname="Liv",
   nachname="Logic",
   geburtsdatum=@(2000-02-29),
)?>
```

Im Gegensatz zu *insert()* wurde der Datensatz nach diesem Aufruf noch nicht gespeichert, dies erfolgt erst durch den Aufruf von *record.save()*. Daher sind vorher die Attribute `id`, `createdby`, `createdat`, `updatedby` und `updatedat` auch `None` und werden erst durch den Aufruf von *record.save()* gesetzt.

Ist eine App nicht explizit in einer Datenquelle konfiguriert, so kann das zugehörige `App`-Objekt trotzdem innerhalb der LivingAPI auftauchen, wenn ein Datensatz über ein Auswahlfeld auf diese App verweist. In einem solchen Fall ist sowohl `records` als auch `recordcount` `None`.

Wird durch Aufruf des App-Objektes eine neues Datensatz-Object angelegt, und die App hat einen aktiven View so werden die in diesem View konfigurierten Standardwerte für die Felder berücksichtigt.

Außerdem werden bei aktivem View zwar alle Felder des Datensatzes gespeichert, aber Fehler in Feldern, die nicht im aktiven Vies sind werden ignoriert.

`Control`

`Control`-Objekte beinhalten die Metadaten eines Feldes einer Applikation. Je nach Feld-Typ ist dies eine spezieller Objekt-Typ (also z.B. `TextControl` für ein normales einzeiliges Text-Feld, `DatetimeMinuteControl` für ein Datumsfeld mit Stunden- und Minuten-Angaben, `LookupSelectControl` für ein Auswahlfeld etc.).

Allgemeine Attribute

Allen Objekte-Typen gemeinsam sind folgende Attribute:

id
: [String]
Der eindeutige Datenbank-Identifizierer des Feldes.

identifier
: [String]
Der menschenlesbare Identifizierer des Feldes (dies ist eine Version der Feld-Beschriftung, die in einen gültigen UL4-Variablennamen ungewandelt wurde und eindeutig ist).

type
: [String]
Der Typ des Feldes. Mögliche Werte sind `"string"`, `"int"`, `"number"`, `"date"`, `"lookup"`, `"applookup"`, `"multiplelookup"`, `"multipleapplookup"`, `"bool"`, `"file"` und `"geo"`. Hierzu können Sie im Kapitel *Übersicht der Feldtypen* mehr lesen.

subtype
: [String oder `None`]
Der Feld-Untertyp.

Typ	Mögliche Untertypen
"string"	"text", "textarea", "email", "url", "password" und "tel".
"date"	"date", "datetimeminute" und "datetimesecond".
"lookup"	"select", "choice" und "radio".
"applookup"	"select" und "choice".
"multiplelookup"	"select", "choice" und "checkbox".
"multipleapplookup"	"select" und "choice".

Bei allen anderen ist `subtype` immer `None`.

Hierzu können Sie im Kapitel *Übersicht der Feldtypen* mehr lesen.

fulltype
[String]
Feld-Typ und Feld-Untertyp (falls vorhanden) getrennt mit `/`, also z.B. `number` oder `lookup/select` usw..

app
[*App*]
Die App zu der dieses Feld gehört.

label
[String]
Die Feld-Beschriftung.

priority
[Bool]
Gibt an, ob das Feld hohe Priorität besitzt, d.h. in der Listenansicht angezeigt werden soll.

order
[Integer]
Die Feld-Reihenfolge, d.h. nach dieser Zahl sind die Felder im `App`-Attribut `controls` (bzw. im `Record`-Attribut `fields`) sortiert.

top
[Integer oder `None`]
Vertikale Position des Eingabefeldes im Eingabeformular in der aktiven Formularvariante. Gibt es keine aktive Formularvariante, ist `top` `None`.

left
[Integer oder `None`]
Horizontale Position des Eingabefeldes im Eingabeformular in der aktiven Formularvariante. Gibt es keine aktive Formularvariante, ist `left` `None`.

width
[Integer oder `None`]
Breite des Eingabefeldes im Eingabeformular in der aktiven Formularvariante. Gibt es keine aktive Formularvariante, ist `width` `None`.

height
[Integer oder `None`]
Höhe des Eingabefeldes im Eingabeformular in der aktiven Formularvariante. Gibt es keine aktive Formularvariante, ist `height` `None`.

z_index
[Integer]
Überlagerungs-Reihenfolge des Eingabefeldes in der aktiven Formularvariante. Gibt es keine aktive Formularvariante, ist `z_index` `None`.

tabindex
[Integer]
Tab-Reihenfolge des Eingabefeldes im Eingabeformular in der aktiven Formularvariante (zum Durch-

schalten der Eingabefelder mit Hilfe der Tab-Taste). Gibt es keine aktive Formularvariante, ist `tabindex` `None`.

required
[Bool]
`True` wenn dieses Feld in der aktiven Formularvariante als Pflichfeld konfiguriert wurde. Gibt es keine aktive Formularvariante, ist `required` `False`.

mode
[`"DISPLAY"` oder `"EDIT"`]
Kann der Wert dieses Feldes im Eingabeformular in der aktiven Formularvariante bearbeitet werden (`"EDIT"`) oder wird der Wert nur angezeigt (`"DISPLAY"`)? (`"DISPLAY"` wird verwendet, wenn das Feld im FormBuilder mit *Inhalt zur zum Lesen anzeigen* konfiguriert wurde. Gibt es keine aktive Formularvariante, ist `mode` `"EDIT"`.

labelpos
[`"TOP"`, `"LEFT"`, `"RIGHT"`, `"BOTTOM"` oder `None`]
Position der Feldbeschriftung relativ zum Eingabefeld in der aktiven Formularvariante. Bei `None` wird die Beschriftung gar nicht angezeigt. Gibt es keine aktive Formularvariante, ist `mode` `"LEFT"`.

labelwidth
[Integer oder `None`.]
Breite des Labels in der aktiven Formularvariante. Gibt es keine aktive Formularvariante, ist `labelwidth` `"LEFT"`.

autoalign
[Bool]
`True` wenn die Breite des Labels automatisch vom FormBuilder berechnet wird. `False` wenn sie der Benutzer selbst angepasst hat. Gibt es keine aktive Formularvariante, ist `autoalign` `True`.

in_active_view()
[Methode() → Bool]
Gibt an, ob es dieses `Control` in der aktiven Formularvariante gibt. Gibt es keine aktive Formularvariante, ist der Return-Wert von `in_active_view()` `False`.

is_focused()
[Methode() → Bool]
Gibt an, ob es dieses `Control` den Eingabefokus hat.

custom
[Objekt]
Dieses Attribut kann vom Benutzer für beliebige zusätzliche Informationen gesetzt werden.

x_<identifier>
[Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

Attribute für String-Felder

Zusätzlich zu obigen Attributen besitzt ein `StringControl` noch folgende Attribute:

placeholder
[String]
Der Platzhalter für das HTML-Eingabefeld.

minlength
[Integer oder `None`]
Die minimale String-Länge (`None` bedeutet ohne Einschränkung).

maxlength
[Integer oder `None`]
Die maximale String-Länge (`None` bedeutet ohne Einschränkung).

`StringControl`s des Untertyps `textarea` besitzen weiterhin noch folgendes Attribut:

encrypted
[String oder None]
Konfiguriert die Verschlüsselung von Feldinhalten. Die möglichen Werte sind:

None
Das Feld wird nicht verschlüsselt.

"force"
Die Feldverschlüsselung wird erzwungen. In einem Formular das eines oder mehrere dieser Felder enthält, werden alle Felder mit dieser Einstellung beim Abschicken verschlüsselt.

"optional"
Der Benutzer kann selbst entscheiden, ob er Verschlüsseln möchte oder nicht.

Attribute für Zahlen-Felder

Zusätzlich zu obigen Attributen besitzt ein `NumberControl` noch folgende Attribute:

precision
[Integer oder None]
Die Anzahl der Nachkommastellen. Wird `precision` im Formular-Template auf eine Zahl gesetzt, so wird außerdem für die Eingabe ein HTML-Element `<input type="number">` statt `<input type="text">` verwendet.

`precision` im Update-Template zu setzen hat keinerlei Einfluß auf die Anzahl der Nachkommastellen und das verwendete HTML-Element.

minimum
[Zahl oder None]
Der minimal erlaubte Wert. Kleinere Werte werde automatisch durch den Wert von `minimum` ersetzt. Ist `minimum` `None` gibt es keine Untergrenze.

maximum
[Zahl oder None]
Der maximal erlaubte Wert. Größere Werte werde automatisch durch den Wert von `maximum` ersetzt. Ist `maximum` `None` gibt es keine Obergrenze.

Attribute für Lookup-Felder

Zusätzlich zu obigen Attributen besitzen `LookupControl`- und `MultipleLookupControl`-Objekte noch folgende Attribute:

lookupdata
[Dictionary(String → *LookupItem*)]
`lookupdata` beinhaltet die Auswahlmöglichkeiten als Dictionary (in der Reihenfolge wie die Optionen vom Benutzer angelegt wurde). Die Schlüssel sind die Identifierer der Option und die Werte sind *LookupItem*-Objekt.

autoexpandable
[Bool]
`True` wenn in FormBuilder für das Feld *Auto-Hinzufügen aktivieren* konfiguriert wurde. In diesem Fall ist es möglich dem zugehörigen *Field*-Objekt einen beliebigen Wert zu geben. Ist dieser Wert weder als *LookupItem* `key`-Attribut noch als `label`-Attribut bekannt, wird ein neues *LookupItem* hinzugefügt.

Attribute für AppLookup-Felder

Zusätzlich zu obigen Attributen besitzen `AppLookupControl`- und `MultipleAppLookupControl`-Objekte noch folgende Attribute:

lookup_app
[*App*]
Die Zielapp aus der Datensätze als Wert für dieses Feld ausgewählt werden können.

lookup_controls
[Dictionary(String → *Control*)]
`lookup_controls` ist ein Dictionary, das die Felder beinhaltet, die der Benutzer zur Anzeige für die Auswahl konfiguriert hat. Die Schlüssel in diesem Dictionary sind die Feld-Identifizierer und die Werte sind *Control*-Objekte.

Attribute für Lookup-und AppLookup-Felder

`LookupControl`-, `MultipleLookupControl`-, `AppLookupControl`- und `MultipleAppLookupControl`-Objekte besitzen zusätzlich noch folgende Attribute:

none_key
[String oder `None`]
Ist `none_key` nicht `None` (was nur bei einem aktiven *View* der Fall sein kann) wird im Eingabeformular eine „Nichts ausgewählt"-Option angeboten. Der Wert von `none_key` wird als Wert dieser Auswahl-Option verwendet. Ist `none_key` `None` wird keine „Nichts ausgewählt"-Option angezeigt.

none_label
[String oder `None`]
Wird als Beschriftung der „Nichts ausgewählt"-Option verwendet, wenn diese „Nichts ausgewählt"-Option angeboten wird. Ist `none_label` `None` sollte eine generische Beschriftung verwendet werden.

Attribute für Datums-Felder

Zusätzlich zu obigen Attibuten haben `DateControl`, `DatetimeMinuteControl` und `DatetimeSecondControl` noch das folgende Attribut:

format
[String]
`format` ist ein für den Datums-Typ passender UL4-Formatstring, der die in `globals.lang` konfigurierte Sprache berücksichtigt.

D.h. das z.B. für ein `DatetimeMinuteControl`-Objekt `format` der Wert `"%d.%m.%Y %H:%M"` hat, wenn `globals.lang` `"de"` ist.

Record

Ein Record-Objekt stellt einen Datensatz einer App dar und hat folgende Attribute:

id
[String oder `None`]
Der eindeutige Datensatz-Identifizierer. Wurde das Datensatz-Objekt neu angelegt (durch Aufruf des App-Objektes) so ist `id` noch `None` und wird erst beim Abspeichern (mittels *save()*) gesetzt.

app
[*App*]
Die App zu der dieser Datensatz gehört.

createdat
[Datum]
Der Zeitpunkt zu dem dieser Datensatz angelegt wurde.

createdby
[*User*]
Der Benutzer der diesen Datensatz angelegt hat.

updatedat
[Datum oder `None`]
Der Zeitpunkt zu dem dieser Datensatz zuletzt geändert wurde. Wurde der Datensatz noch nie geändert, so ist `updatedat` `None`.

updatedby
[*User* oder `None`]
Der Benutzer, der den Datensatz zuletzt geändert hat. Wurde der Datensatz noch nie geändert, so ist `updatedby` `None`.

fields
[Dictionary(String → *Field*)]
`fields` beinhaltet die Feld-Werte des Datensatzes. Einträge in `fields` haben dieselbe Reihenfolge, wie die Einträge im `controls`-Attribut eines `App`-Objekts. Die Schlüssel sind wieder die Feld-Identifizierer und die Werte sind `Field`-Objekte.

f_<identifier>
[*Field*]
Die `Field`-Objekte stehen auch über sogenannte „Shortcut"-Attribute zur Verfügung. `record.fields.vorname` steht auch z.B. direkt als `record.f_vorname` zur Verfügung.

values
[Dictionary(String → Objekt)]
`values` beinhaltet statt *Field*-Objekten direkt die Werte der Felder. Für alle Felder `x` gilt:

```
record.fields.x.value is record.values.x
```

v_<identifier>
[Objekt]
Auch Feld-Werte stehen über „Shortcut"-Attribute zur Verfügung. `record.values.vorname` also z.B. als `record.v_vorname`.

attachments
[Dictionary(String → *Attachment*)]
Die Anhänge zu diesem Datensatz. Die Dictionary-Schlüssel sind die Datenbank-Identifizierer des Anhangs, die Werte sind *Attachment*-Objekte.

children
[Dictionary(String → Dictionary(String → *Record*))]
Die diesem Datensatz zugeordneten Datensätze. Der Dictionary-Schlüssel auf oberster Ebene ist dabei der *Identifizierer*, der dieser Zuordnung in der Konfiguration gegeben wurde. Das Dictionary auf zweiter Ebene beinhaltet die zugeordneten Datensätze. Die Schlüssel sind die Datensatz-Identifizierer und die Werte sind *Record*-Objekte. Als Sortierung wird die in der Konfiguration festgelegte Sortierung verwendet.

c_<identifier>
[Dictionary(String → *Record*)]
Auch für zugeordnete Datensätze gibt es Shortcut-Attribute. `record.children.beispiel`, steht auch direkt als `record.c_beispiel` zur Verfügung.

errors
[Liste(String)]
Fehlermeldungen die den gesamten Datensatz betreffen als Liste von Strings. Diese werden im Augenblick nicht von System gesetzt, können aber vom Benutzer beliebig geändert werden.

add_error(error)
[Methode(String) → `None`]

Fügt die übergebene Fehlermeldung zur Liste der Fehlermeldungen (`errors`) hinzu. Diese Methode erwartet die Fehlermeldung als Argument. Das heißt, der Aufruf sieht beispielsweise so aus: `record.add_error("Die Bearbeitungszeit ist abgelaufen!")`.

`has_errors()`
[Methode() → Bool]
Gibt zurück, ob dieser Datensatz oder eines seiner Felder eine Fehlermeldung hat. Diese Methode erwartet keine Argumente. Das heißt, der Aufruf sieht beispielsweise so aus: `record.has_errors()`.

`clear_errors()`
[Methode() → `None`]
Leert die Fehlermeldungs-Liste zu diesem Datensatz aber nicht die der Felder. Diese Methode erwartet keine Argumente. Das heißt der Aufruf sieht beispielsweise so aus: `record.clear_errors()`.

`clear_all_errors()`
[Methode() → `None`]
Leert die Fehlermeldungs-Listen zu diesem Datensatz und zu den dazugehörigen Feldern. Diese Methode erwartet keine Argumente. Das heißt der Aufruf sieht beispielsweise so aus: `record.clear_all_errors()`.

`is_dirty()`
[Methode() → Bool]
Gibt an, ob Feld-Werte geändert wurden. Ist der Datensatz noch gar nicht gespeichert, gibt `is_dirty()` immer `True` zurück.

`update(**values)`
[Methode(**Objekt) → `None`]
Mithilfe von `update()` kann der Datensatz verändert werden. Die Parameter müssen per Schlüsselwort übergeben werden. Der Parametername ist dabei jeweils der Feld-Identifizierer des zu setzenden Feldes.

D.h. ein Aufruf sieht z.B. so aus:

```
record.update(vorname="Max", nachname="Mustermann")
```

`delete()`
[Methode() → `None`]
Mithilfe von `delete` kann der Datensatz gelöscht werden. Diese Methode benötigt keine Parameter.

`save(force, sync)`
[Methode(Bool, Bool) → Bool]
Mit der Methode `save` kann ein Datensatzobjekt abgespeichert werden. Ist dieses Datensatzobjekt neu angelegt (d.h. `id` ist `None`) wird beim Aufruf von `save()` ein neuer Datensatz angelegt und beim Datensatzobjekt der eindeutige Datensatz-Identifizierer (`id`) vergeben, sowie die Attribute `createdby` auf `globals.user` (eingeloggter Benutzer) und `createdat` auf das aktuelle Datum gesetzt. Ist das Datensatzobjekt nicht neu, werden beim Aufruf von `save()` nur die geänderten Felder gespeichert, sowie das Attribut `updatedby` auf `globals.user` und `updatedat` auf das aktuelle Datum gesetzt.

Ist `force False`, so wird der Datensatz nicht gespeichert, wenn er oder einer seiner Felder eine Fehlermeldung hat (d.h. wenn `has_errors()` `True` zurückgibt), oder wenn Felder von Typ `applookup`, `multipleapplookup` oder `file` auf noch nicht gespeicherte Datensätze oder Dateien verweisen. Statt dessen wird ein Ausnahme-Fehler erzeugt, der die Ausführung des Anzeige-Templates abbricht. Dieser Ausnahme-Fehler kann dann unter *Konfiguration → Erweitert → Anzeige-Templates → (Template) → Meldungen in aktueller Version* angesehen werden.

Ist `force True` so werden Fehlermeldungen am Datensatz oder seinen Feldern ignoriert, für noch nicht gespeicherte Datensätze oder Dateien wird stattdessen `None` als Wert verwendet und der Datensatz wird trotzdem gespeichert.

Wenn für `sync` `False` übergeben wird (der Default) ist das Verhalten wie oben beschrieben, wird `True` übergeben, so wird nach dem Speichern das `Record`-Objekt mit dem Datenbank-Inhalt aktualisiert. Das heißt insbesondere, daß Änderungen die durch Datenaktionen an dem Datensatz durchgeführt wurden, im `Record`-Objekt auftauchen.

Der Rückgabewert von `save()` gibt an, ob der Datensatz tatsächlich in der Datenbank gespeichert wurde, oder ob dabei ein Fehler aufgetreten ist. Die entsprechende Fehlermeldung wird dabei an den Datensatz

gehängt.

Bemerkung: Für noch nicht gespeicherte Datensätze oder Dateien wird beim Speichern mit `force=True` zusätzlich eine Fehlermeldung an das Feld-Objekt gehängt (dies kann erst zum Speicherzeitpunkt gemacht werden, nicht zum Zeitpunkt der Zuweisung des Feld-Wertes, da es ja möglich ist, daß ein referenziertes Objekt erst gespeichert wird, nachdem es dem Feld zugewiesen wurde).

Noch nicht gespeicherte Dateien können dabei bei der Verwendung innerhalb von Anzeige-Templates nicht auftreten, jedoch bei der Verwendung über das *Python-SDK*.

---

executeaction(identifier)

[Methode(String) → None]
Mithilfe von `executeaction` kann eine Daten-Aktion auf dem Datensatz ausgeführt werden. Diese Daten-Aktionen können unter *Konfiguration* → *Erweitert* in der *Aktionen*-Maske definiert werden. Als Parameter muß der Identifizierer der Daten-Aktion übergeben werden.

Gibt es z.B. eine Datenaktion mit dem Identifizierer `freigeben`, so kann für den Datensatz `record` diese Datenaktion folgendermaßen aufgerufen werden:

```
<?code record.executeaction("freigeben")?>
```

edit_embedded_url(params)**

[Methode(**Objekt) → String]
Gibt die absolute URL zurück für das Eingabe-Formular zum Bearbeiten dieses Datensatzes. Bei dieser URL ist das Formular in den üblichen LivingApps-Rahmen eingebettet und der Benutzer muß eingeloggt sein um es benutzen zu können.

Mittels `params` können der URL zusätzliche Parameter übergeben werden. Als Werte werden sowohl Strings unterstützt als auch Listen von Strings (in diesem Fall wird der Parameter mehrmals an die URL angefügt). Außerdem wird beim Wert `None` der Parameter ignoriert.

Beispiel 1:

```
<?print record.edit_embedded_url()?>
```

erzeugt:

```
https://my.living-apps.de/dateneingabe/1234567890abcdef12345678/
↪10293848576afbecd12345678/edit
```

Beispiel 2:

```
<?print record.edit_embedded_url(view="abcdef12345678901234")?>
```

erzeugt:

```
https://my.living-apps.de/dateneingabe/1234567890abcdef12345678/
↪10293848576afbecd12345678/edit?view=abcdef12345678901234
```

Beispiel 3:

```
<?print record.edit_embedded_url(a=["17", 23], b=None, c=[today(), None])?>
```

erzeugt:

```
https://my.living-apps.de/dateneingabe/1234567890abcdef12345678/
↪10293848576afbecd12345678/edit?a=17&a=23&c=2022-11-10
```

edit_standalone_url(params)**

[Methode(**Objekt) → String]
Gibt die absolute URL zurück für das Eingabe-Formular zum Bearbeiten dieses Datensatzes. Bei dieser

URL ist das Formular nicht in den üblichen LivingApps-Rahmen eingebettet sondern unabhängig und kann auch von nicht eingeloggten Benutzern ausgefüllt werden.

Mittels `params` können der URL zusätzliche Parameter übergeben werden.

Beispiel 1:

```
<?print record.edit_standalone_url()?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/
↪10293848576afbecd12345678/edit
```

Beispiel 2:

```
<?print record.edit_standalone_url(view="abcdef12345678901234")?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/
↪10293848576afbecd12345678/edit?view=abcdef12345678901234
```

Beispiel 3:

```
<?print record.edit_standalone_url(a=["17", 23], b=None, c=[today(), None])?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/
↪10293848576afbecd12345678/edit?a=17&a=23&c=2022-11-10
```

template_url(identifier, /, **params)

[Methode(String, **Objekt) → String]
Gibt die absolute URL zurück für ein Detail-Anzeige-Template mit dem Identifizierer `identifier`. Die verlinkte App ist dabei die App zu der Datensatz gehört).

Mittels `params` können der URL zusätzliche Parameter übergeben werden.

Beispiel 1:

```
<?print record.template_url("beispiel")?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/
↪10293848576afbecd12345678?template=beispiel
```

Beispiel 2:

```
<?print record.template_url("beispiel", x=17, y=23)?>
```

erzeugt:

```
https://my.living-apps.de/gateway/apps/1234567890abcdef12345678/
↪10293848576afbecd12345678?template=beispiel&x=17&y=23
```

custom

[Objekt]
Dieses Attribut kann vom Benutzer für beliebige zusätzliche Informationen gesetzt werden.

x_<identifier>
[Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

Field

Ein `Field`-Objekt beinhaltet den Wert eines Feldes für einen Datensatz und besitzt folgende Attribute:

control
[*Control*]
Das `Control`-Objekt für das dieses `Field`-Objekt den Wert beinhaltet.

record
[*Record*]
Das Datensatz-Objekt zu dem dieser Feld-Wert gehört (siehe *Record*).

value
[Objekt]
Der Wert des Feldes. Der Typ des Wertes hängt dabei natürlich vom Typ des Feldes ab.

Für String-Felder ist dies ein String, für Häckchen-Felder ein Bool, für Zahlen-Felder eine Zahl und für Datumsfelder ein Datum. Für Datei-Uploads ist es ein *File*-Objekt.

Ist das Feld ein Lookup-Feld (d.h. eine Auswahlbox oder ein Optionenfeld) so gibt es zwei Möglichkeiten: Werden für die Auswahl die Datensätze einer anderen App benutzt, so ist der Feld-Wert das `Record`-Objekt, das diesen anderen Datensatz beschreibt. Im anderen Fall ist der Wert ein *LookupItem*-Objekt.

Ist das Feld ein MultipleLookup-Feld (d.h. eine Mehrfach-Optionenfeld), so ist der Wert eine Liste von *LookupItem*- bzw. *Record*-Objekten.

Ist das Feld ein Upload-Feld, so ist der Wert ein *File*-Objekt.

Bei einem Geo-Feld ist der Wert ein *Geo*-Objekt.

Ist das Feld kein Pflichtfeld, so ist als Wert bei allen Feld-Typen natürlich immer `None` möglich (außer bei Mehrfachoptionen, wo der „leere“ Wert eine leere Liste ist).

Das `Field`-Attribut `value` kann immer gesetzt werden (ohne daß bei nicht passenden Werten eine Fehlermeldung ausgegeben wird), jedoch kann eine Fehler-Meldung im `errors`-Attribut gesetzt werden, wenn der Wert nicht sinnvoll ist (in diesem Fall wird auch der Wert durch `None` (bzw. `[]`) ersetzt).

Die folgende Tabelle zeigt die unterschiedlichen möglichen Typen für Feld-Werte für jeden Feld-Typ nochmal in der Übersicht:

Typ	Wert
`bool`	`bool` (`True` oder `False`) oder `None`
`string`	`str` oder `None`
`int`	`int` oder `None`
`number`	`int`, `float` oder `None`
`date`	`date` (für Untertyp `date`), `datetime` (für Untertyp `datetimeminute` und `datetimesecond`) oder `None`
`geo`	*Geo* oder `None`
`file`	*File* oder `None`
`lookup`	*LookupItem* oder `None`
`multiplelookup`	Liste von *LookupItem*s (leere Liste wenn nichts ausgewählt)
`applookup`	*Record* oder `None`
`multipleapplool`	Liste von *Record*s (leere Liste wenn nichts ausgewählt)

errors
[Liste(String)]
Fehlermeldungen die dieses Feld betreffen als Liste von Strings.

`add_error(error)`
: [Methode(String) → None]
Fügt die übergebene Fehlermeldung zur Liste der Fehlermeldungen (`errors`) hinzu. Diese Methode erwartet die Fehlermeldung als Argument. Das heißt, der Aufruf sieht beispielsweise so aus: `field.add_error("Bitte geben Sie eine gültige E-Mail-Adresse ein!")`.

`set_error(error)`
: [Methode(String oder None) → None]
Ist `error` None so wird die Liste der Fehlermeldungen geleert. Ist `error` ein String, so besteht die Liste der Fehlermeldungen aus dieser einen Fehlermeldung.

`has_errors()`
: [Methode() → Bool]
Gibt zurück, ob dieses Feld Fehlermeldungen hat (d.h. ob `errors` nicht leer ist). Diese Methode erwartet keine Argumente. Das heißt, der Aufruf sieht beispielsweise so aus: `field.has_errors()`.

`clear_errors()`
: [Methode() → None]
Leert die Fehlermeldungs-Liste zu diesem Feld. Diese Methode erwartet keine Argumente. Das heißt der Aufruf sieht beispielsweise so aus: `field.clear_errors()`.

`lookupdata`
: [Dictionary(String → *LookupItem*), Dictionary(String → *Record*) oder Dictionary(String → String)]
Dieses Attribut ist nur bei `lookup/multiplelookup`- bzw. `applookup/multipleapplookup`-Feldern vorhanden. Es stellt die Auswahlmenge im Eingabeformular dar, und kann daher verwendet werde um diese Auswahlmenge zu ändern. D.h. es kann damit sowohl festgelegt werden, welche *LookupItem*s bzw. *Record*s zur Auswahl stehen, als auch welche Bezeichnungen dafür angezeigt werden.

Bei `lookup/multiplelookup`-Feldern überschreibt es das `lookupdata`-Attribut des *Control*-Objektes. Allerdings dürfen die Werte auch Strings sein. Dieser werden dann im Formular direkt als Bezeichnungen der Optionen verwendet.

Bei `applookup/multipleapplookup`-Feldern müssen die Keys im Dictionary `id`-Werte von Datensätzen aus der Zielapp sein. Die Werte sind entweder die zugehörigen *Record*-Objekte oder Strings, die dann im Formular direkt als Bezeichnungen der Datensätze verwendet werden.

Wird `lookupdata` auf None gesetzt wird wieder die ursprüngliche Auswahlmenge verwendet.

`is_empty()`
: [Methode() → Bool]
Gibt an, ob der Feld-Wert leer ist. Dies ist bei `multiplelookup`- und `multipleapplookup`-Feldern die leere Liste, bei allen anderen Feld-Typen None. Diese Methode erwartet keine Argumente. Das heißt, der Aufruf sieht beispielsweise so aus: `field.is_empty()`.

`is_dirty()`
: [Methode() → Bool]
Gibt an, ob der Feld-Wert geändert wurde. Diese Methode erwartet keine Argumente. Das heißt, der Aufruf sieht beispielsweise so aus: `field.is_dirty()`.

`enabled`
: [Bool]
Für Anzeige-Templates ist dieses Attribut irrelevant. Es gibt an, ob ein Eingabefeld für diesen Wert aktiviert werden soll oder nicht und wird nur in Update-Templates verwendet (d.h. wenn `enabled False` ist, erhält das Eingabefeld das HTML-Attribut `disabled="disabled"`; Ein solches Feld kann vom Benutzer nicht mehr geändert werden und der dort eingetragene Wert wird beim Absenden des Formulars nicht mitgesendet).

`writable`
: [Bool]
Für Anzeige-Templates ist dieses Attribut irrelevant. Es gibt an, ob ein Eingabefeld für diesen Wert beschreibbar angezeigt werden soll oder nicht und wird nur in Update-Templates verwendet (d.h. wenn `writable False` ist, erhält das Eingabefeld das HTML-Attribut `readonly="readonly"`; Ein solches Feld kann vom Benutzer nicht mehr geändert werden, der dort eingetragene Wert wird aber beim Absenden des Formulars mitgesendet).

`visible`
: [Bool]
Für Anzeige-Templates ist dieses Attribut irrelevant. Es gibt an, ob ein Eingabefeld für diesen Wert eingeblendet werden soll oder nicht und wird nur in Update-Templates verwendet (d.h. wenn `visible` `False` ist, wird das Eingabefeld komplett ausgeblendet).

`custom`
: [Objekt]
Dieses Attribut kann vom Benutzer für beliebige zusätzliche Informationen gesetzt werden.

`x_<identifier>`
: [Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

Attribute für Lookup- und Applookup-Felder

`Field`-Object zu Controls vom Typ `LookupControl`, `MultipleLookupControl`, `AppLookupControl` oder `MultipleAppLookupControl` besitzen zusätzlich noch folgendes Attribut:

`lookupdata`
: [Dictionary(String → *LookupItem* oder]
Record oder String) `lookupdata` kann gesetzt werden um die Auswahlmenge für dieses Feld zu ersetzen.

Für Felder zu Controls vom Typ `LookupControl` oder `MultipleLookupControl` müssen die Keys `key`-Attibute von *LookupItem*s sein, die Werte entweder *LookupItem*s oder Strings (bei String wird der Wert 1:1 für die Anzeige der Auswahlmöglichkeiten verwendet).

Für Felder zu Controls vom Typ `AppLookupControl` oder `MultipleAppLookupControl` müssen die Keys `id`-Attibute von *Record*-Objekten sein, die Werte entweder *Record*s oder Strings (bei String wird der Wert 1:1 für die Anzeige der Auswahlmöglichkeiten verwendet).

Wird der Wert von `lookupdata` auf `None` gesetzt, so wird wieder die Auswahl, die durch das *Control* definitiert ist, verwendet (Bei `LookupControl` oder `MultipleLookupControl` `control.lookupdata` und bei `AppLookupControl` oder `MultipleAppLookupControl` `control.lookup_app.records`).

`has_custom_lookupdata()`
: [Methode() → Bool]
Liefert `True` zurück wenn in `lookupdata` die Auswahlmenge überschrieben wurde.

`LayoutControl`

Ein `LayoutControl`-Objekt stellt eine „Dekoration“ in einem Eingabeformular dar, die mit keinem Feld von Datensätzen in Verbindung steht. Es gibt drei Feld-Typen:

- `HTMLLayoutControl` für zusätzliches HTML das im Eingabeformular angezeigt wird (*Formatierter Text* im FormBuilder);
- `ImageLayoutControl` für ein Dekobild;
- `ButtonLayoutControl` für einen „Absenden“-Button;

Ein `LayoutControl`-Objekt kann sowohl einer *App* zugeordnet sein (wenn es einen aktiven *View* gibt) als auch direkt einem *View*.

Allgemeine Attribute

Allen Objekte-Typen gemeinsam sind folgende Attribute:

id
: [String]
Der eindeutige Datenbank-Identifizierer des `LayoutControl`s.

view
: [*View*]
Das `View`-Objekt zu diesem `LayoutControl`.

label
: [String]
Die Beschriftung.

identifier
: [String]
Der menschenlesbare Identifizierer des Feldes (dies ist eine Version der Beschriftung, die in einen gültigen UL4-Variablennamen ungewandelt wurde und eindeutig ist).

type
: [String]
Der Typ des Layoutfelder: `"string"`, `"image"` oder `"button"`.

subtype
: [String oder `None`]
Der Untertyp des Layoutfelder: `"html"` beim Typ `"string"` und `None` beim Typ `"image"` und `"button"`.

top
: [Integer]
Vertikale Position des Feldes im Eingabeformular in dieser bzw. der aktiven Formularvariante.

left
: [Integer]
Horizontale Position des Feldes im Eingabeformular in dieser bzw. der aktiven Formularvariante.

width
: [Integer]
Breite des Feldes im Eingabeformular in dieser bzw. der aktiven Formularvariante.

height
: [Integer]
Höhe des Feldes im Eingabeformular in dieser bzw. der aktiven Formularvariante.

z_index
: [Integer]
Überlagerungs-Reihenfolge des Feldes in dieser bzw. der aktiven Formularvariante.

visible
: [Bool]
Soll dieses Layoutfeld in dieser bzw. der aktiven Formularvariante angezeigt werden?

x_<identifier>
: [Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

Attribute für `HTMLLayoutControl`-Felder

`HTMLLayoutControl`-Felder haben zusätzlich das folgende Attribut:

`value`
: [string]
Der HTML-Quelltext.

Attribute für `ImageLayoutControl`-Felder

`ImageLayoutControl`-Felder haben zusätzlich das folgenden Attribut:

`image`
: [*File*]
Das Original-Bild, das vom Benutzer hochgeladen wurde.

  Zur Anzeige sollte es mittels *scaled_url(...)* auf die durch die Attribute `width` und `height` festgelegte Größe skaliert werden.

`AppParameter`

Ein `AppParameter`-Objekt beinhaltet Informationen über einen Parameter einer App. Es besitzt folgende Attribute:

`id`
: [String]
Der eindeutige Datenbank-Identifizierer der App.

`app`
: [*App* oder `None`]
Die App zu der dieser Parameter gehört (oder `None` wenn dieser Parameter zur Template-Library gehört).

`owner`
: [*App*]
Die App zu der dieser Parameter gehört.

`parent`
: [AppParameter oder `None`]
Der übergeordnete Parameter vom Typ `list` oder `dict` (oder `None` falls dieser Parameter kein Element eines übergeordneten Parameters ist).

`type`
: [String]
Der Parameter-Typ. Mögliche Werte sind:

Wert	Beschreibung
`"bool"`	Ein Wahrheitswert: `True` oder `False`
`"int"`	Eine ganze Zahl
`"number"`	Eine Zahl mit Nachkommastellen
`"str"`	Text
`"html"`	HTML-Text
`"color"`	Eine Farbe
`"date"`	Ein Datum
`"datetime"`	Datum + Uhrzeit
`"datedelta"`	Ein Zeitraum von Tagen
`"datetimedelta"`	Ein Zeitraum von Tagen, Stunden, Minuten und Sekunden
`"monthdelta"`	Ein Zeitraum von Monaten
`"upload"`	Eine hochgeladene Datei
`"app"`	Ein Verweis auf eine App
`"control"`	Ein Verweis auf ein Feld der App zu der dieser Parameter gehört
`"list"`	Eine Liste (bestehend aus Unterparametern)
`"dict"`	Eine Menge an Unterparametern mit Namen (Identifizierer)

Bei allen Typen außer `"list"` und `"dict"` kann der Wert auch `None` sein.

Dieses Attribut kann auch gesetzt werden. Wenn der alte Wert nicht zum neu gesetzten Typ passt, wird der Wert auf `None` gesetzt.

order
[Integer oder `None`]
Die Reihenfolge in der dieser Parameter als Kind eines übergeordneten Parameters aufgeführt ist, wenn der übergeordnete Parameter vom Typ `list` ist (oder `None` sonst). Dieses Attribut kann auch gesetzt werden.

identifier
[String oder `None`]
Der Name des Parameters (oder `None` wenn dieser Parameter einen übergeordneten Parameter vom Typ `list` hat). Dieses Attribut kann auch gesetzt werden.

description
[String]
Die Beschreibung des Parameters. Dieses Attribut kann auch gesetzt werden.

value
[Objekt]
Der Wert des Objekts. Dabei hängt der Typ des Wertes vom Typ des Parameters ab. Dieser Attribut kann auch gesetzt werden. Je nach dem Typ des gesetzten Wertes ändert sich dadurch auch der Parameter-Typ.

createdat
[Datum]
Zeitpunkt, zu dem der Parameter angelegt wurde.

createdby
[*User*]
Benutzer, der den Parameter angelegt hat.

updatedat
[Datum oder `None`]
Der Zeitpunkt, zu dem der Parameter das letzte Mal geändert wurde. (Wurde der Parameter noch nicht geändert, so ist `updatedat` `None`.)

updatedby
[*User* oder `None`]
Der Benutzer, der den Parameter zuletzt geändert hat. (Wurde der Parameter noch nicht geändert, so ist `updatedby` `None`.)

append_param(type, description, value)
[Methode(String, String oder None, Objekt) → AppParameter]
Erzeugt einen neuen Parameter vom Typ `type` mit der Beschreibung `description` und dem Wert `value` und fügt ihn an diesen Parameter vom Typ `list` an. Der neu angelegte Parameter wird zurückgegeben. Diese Methode ist nur bei Parametern vom Typ `list` vorhanden.

add_param(type, identifier, description, value)
[Methode(String, String, String oder None, Objekt) → AppParameter]
Erzeugt einen neuen Parameter vom Typ `type`, mit dem Identifizierer `identifier`, der Beschreibung `description` und dem Wert `value` und fügt ihn an diesen Parameter vom Typ `dict` an. Der neu angelegte Parameter wird zurückgegeben. Diese Methode ist nur bei Parametern vom Typ `dict` vorhanden.

save(sync)
[Methode(Bool) → None]
Speichert einen neuen oder geänderten Parameter in der Datenbank ab.

Wenn für `sync` `True` übergeben, so wird nach dem Speichern das `AppParameter`-Objekt mit dem Datenbank-Inhalt aktualisiert.

delete()
[Methode() → None]
Löscht den Parameter aus der Datenbank.

state
[String]
Der Zustand des Parameters. Mögliche Werte sind:

Wert	Beschreibung
`"new"`	Das Parameter-Objekt wurde neu angelegt, ist aber noch nicht gespeichert.
`"saved"`	Das Parameter-Objekt wurde aus der Datenbank geladen und noch nicht verändert.
`"changed"`	Das Parameter-Objekt wurde verändert.
`"deleted"`	Der Parameter wurde bereits in der Datenbank gelöscht.

is_dirty()
[Methode() → Bool]
Git `True` zurück, wenn das Parameter-Objekt neu ist oder geändert wurde (d.h. wenn `state` `"new"` oder `"changed"` ist).

is_deleted()
[Methode() → Bool]
Git `True` zurück, wenn das Parameter-Objekt in der Datenbank gelöscht wude (d.h. wenn `state` `"deleted"` ist).

MenuItem

`MenuItem`-Objekte stellen zusätzliche von Ihnen konfigurierte Menüeinträge dar, die Sie unter *Konfiguration* → *Erweitert* und dort bei linken Menüleiste unter *App-Menüs* konfigurieren können.

Ein `MenuItem`-Objekt hat folgende Attribute:

id
[String]
Der interne Datenbank-Identifizierer für diesen Link.

app
[*App*]
Die App zu der dieser Link gehört.

identifier
[String]
Der Link-Identifier.

label
[String]
Der Link-Text.

parent
[*MenuItem* oder None]
Das übergeordnete MenuItem.

target_type
[String]
Der Typ auf den der Link verweist. Kann die Werte newform_standalone, newform_embedded, datamanagement, customoverview, evaluation, import_export, tasks, formbuilder, workflow_manager, data_config, permissions, expert, viewtemplate, datamanageview oder custom haben.

icon
[String]
Der Name eines Font-Awesome Icons für den Link. Der Name kein einen der Suffixe -brand, -sharp-solid, -solid, -regular, -light, -thin oder -duotone haben, um den entsprechenden Font Awesome Stil statt des Standards zu erzwingen.

title
[String]
Der Titel des Links, d.h. der Text für das HTML-Attribut title.

target
[String]
Der Wert für das HTML target Attribut des Link s.

cssclass
[String]
Der Wert für das HTML class Attribut des Links.

url
[String]
Die URL auf die der Link zeigt.

order
[Integer]
Legt die Reihenfolge fest, in der Links in einem Menü oder als Liste angezeigt werden.

start_time
[String]
Der Zeitpunkt ab dem der Link angezeigt werden soll. Wird ignoriert, wenn der Wert None ist.

end_time
[String]
Der Zeitpunkt bis zu dem der Link angezeigt werden soll. Wird ignoriert, wenn der Wert None ist.

on_app_overview_page
[Bool]
Soll der Link auf der Übersichtsseite aller Apps angezeigt werden?

on_app_detail_page
[Bool]
Soll der Link auf der Detailseite einer App angezeigt werden?

on_form_page
[Bool]
Soll der Link auf der Formularseite einer App angezeigt werden (sowohl beim Anlegen als auch beim Editieren von Datensätzen)?

on_iframe_page
[Bool]
Soll der Link auf der IFrame-Seite angezeigt werden (momentan ungenutzt)?

on_custom_overview_page
[Bool]
Soll der Link auf der eigenen Übersichtsseite angezeigt werden?

accessible
[Bool]
Ist der Link für den aktuell eingeloggten Benutzer zugänglich?

children
[Dictionary(String → MenuItem)]
Die untergeordneten Kacheln. Die Schlüssel in diesem Dictionary sind die Link-Identifizierer und die Werte sind `MenuItem`-Objekte.

c_<identifier>
[MenuItem]
Untergeordnete `MenuItem`-Objekte stehen auch über sogenannte „Shortcut"-Attribute zur Verfügung.

createdat
[Datum]
Zeitpunkt wann der Link angelegt wurde.

createby
[*User*]
Der Benutzer der diesen Link erstellt hat.

updatedat
[Datum oder `None`]
Zeitpunkt wann der Link zuletzt verändert wurde.

updateby
[*User*]
Der Benutzer der diesen Link zuletzt verändert hat.

x_<identifier>
[Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

Panel

`Panel`-Objekte stellen zusätzliche von Ihnen konfigurierte Navigations- oder Informations-Panel dar, die Sie unter *Konfiguration → Erweitert* und dort bei linken Menüleiste unter *App-Panels* konfigurieren können.

`Panel`-Objekte werden verwendet, um Verweise auf einer Übersichtsseite zu bauen. Ein `Panel`-Objekt hat folgende Attribute:

id
[String]
Der interne Datenbank-Identifizierer für diesen Link.

app
[*App*]
Die App zu der dieser Link gehört.

identifier
[String]
Der Link-Identifier.

label
[String]
Der Link-Text.

parent
[*Panel* oder `None`]
Das übergeordnete Panel.

target_type
[String]
Der Typ auf den der Link verweist. Kann die Werte `newform_standalone`, `newform_embedded`, `datamanagement`, `customoverview`, `evaluation`, `import_export`, `tasks`, `formbuilder`, `workflow_manager`, `data_config`, `permissions`, `expert`, `viewtemplate`, `datamanageview` oder `custom` haben.

description
[String]
Die Beschreibung der Kachel als HTML-Text.

description_url
[String]
Die URL unter der die Panel-Beschreibung abgerufen werden kann.

icon
[String]
Der Name eines Font-Awesome Icons für den Link. Der Name kein einen der Suffixe `-brand`, `-sharp-solid`, `-solid`, `-regular`, `-light`, `-thin` oder `-duotone` haben, um den entsprechenden Font Awesome Stil statt des Standards zu erzwingen.

header_type
[String]
Gibt an wie der Panel-Kopf angezeigt werden soll. Kann die Werte `title` und `card` haben. Bei `card` ist der Kopfbereich höher.

header_background
[String oder `None`]
Gibt an welcher Hintergrund im Panel-Kopf angezeigt werden soll. Kann die Werte `uniformcolor` (d.h. eine Hintergrundfarbe), `lineargradient`, `radialgradient` oder `image` haben.

text_color
[Color oder `None`]
Die Textfarbe.

background_color1
[Color oder `None`]
Die erste der beiden Hintergrundfarben.

background_color2
[Color oder `None`]
Die zweite der beiden Hintergrundfarben.

image
[*File* oder `None`]
Ein Bild, das zu dem Link angezeigt wird.

title
[String]
Der Titel des Links, d.h. der Text für das HTML `title` Attribut.

target
[String]
Der Wert für das HTML `target` Attribut des Link s.

cssclass
[String]
Der Wert für das HTML `class` Attribut des Links.

url
[String]
Die URL auf die der Link zeigt.

order
[Integer]

Legt die Reihenfolge fest, in die Kacheln angezeigt werden. Wird nicht für Panels auf einer eigner Übersichtseite genutzt. Dort werden `row`, `column`, `width` und `height` stattdessen verwendet.

row
[String]
Die Zeile auf einer eignen Übersichtsseite in der die Kachel platziert werden soll.

column
[String]
Die Spalte auf einer eignen Übersichtsseite in der die Kachel platziert werden soll.

width
[String]
Die Breite der Kachel (d.h. die Anzahl der Gitterspalten).

height
[String]
Die Höhe der Kachel (d.h. die Anzahl der Gitterzeilen).

start_time
[String]
Der Zeitpunkt ab dem der Link angezeigt werden soll. Wird ignoriert, wenn der Wert `None` ist.

end_time
[String]
Der Zeitpunkt bis zu dem der Link angezeigt werden soll. Wird ignoriert, wenn der Wert `None` ist.

on_app_overview_page
[Bool]
Soll der Link auf der Übersichtsseite aller Apps angezeigt werden?

on_app_detail_page
[Bool]
Soll der Link auf der Detailseite einer App angezeigt werden?

on_form_page
[Bool]
Soll der Link auf der Formularseite einer App angezeigt werden (sowohl beim Anlegen als auch beim Editieren von Datensätzen)?

on_iframe_page
[Bool]
Soll der Link auf der IFrame-Seite angezeigt werden (momentan ungenutzt)?

on_custom_overview_page
[Bool]
Soll der Link auf der eigenen Übersichtsseite angezeigt werden?

accessible
[Bool]
Ist der Link für den aktuell eingeloggten Benutzer zugänglich?

children
[Dictionary(String → Panel)]
Die untergeordneten Kacheln. Die Schlüssel in diesem Dictionary sind die Link-Identifizierer und die Werte sind `Panel`-Objekte.

c_<identifier>
[Panel]
Untergeordnete `Panel`-Objekte stehen auch über sogenannte „Shortcut"-Attribute zur Verfügung.

createdat
[Datum]
Zeitpunkt wann der Link angelegt wurde.

createby
[*User*]
Der Benutzer der diesen Link erstellt hat.

updatedat
[Datum oder `None`]
Zeitpunkt wann der Link zuletzt verändert wurde.

updateby
[*User*]
Der Benutzer der diesen Link zuletzt verändert hat.

x_<identifier>
[Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

`LookupItem`

Ein `LookupItem` stellt den Wert eines `LookupControl`s dar (d.h. einer Auswahlbox oder eines Optionsfeldes) wenn nicht die Datensätze einer anderen App als Auswahlmöglichkeiten verwendet werden. Ein `LookupItem`-Objekt hat folgende Attribute:

control
[*Control*]
Das *Control*-Objekt zu dem dieses `LookupItem` gehört.

key
[String]
Die eindeutige Kennung der Auswahl-Option.

label
[String]
Die Beschriftung der Auswahl-Option.

visible
[Bool]
`False` wenn diese Option in der aktiven Formularvariante nicht angezeigt werden soll. `True` wenn sie angezeigt werden soll, oder es keine aktive Formularvariante gibt.

`View`

`View`-Objekte stellen eine spezielle Ansicht auf die Felder einer App dar. Im FormBuilder unter *Konfiguration* → *Eingabe* werden sie bei *Formularvarianten* angelegt.

Ein `View`-Objekt hat folgende Attribute:

id
[String]
Der eindeutige Datenbank-Identifizierer der Ansicht.

name
[String]
Der Name der Ansicht.

app
[*App*]
Die App zu der diese Ansicht gehört.

order
[Integer]
Die Reihenfolge der Ansicht, d.h. nach dieser Zahl sind die Ansichten im `App`-Attribut `views` sortiert.

`lang`
: [String]
Die im FormBuilder konfigurierte Sprache für den View.

`width`
: [Integer]
Die von Benutzer festgelegte Breite in Pixeln für diese Ansicht.

`height`
: [Integer]
Die von Benutzer festgelegte Höhe in Pixeln für diese Ansicht.

`start`
: [Datum oder `None`]
Die Anzeige einer Ansicht kann zeitlich beschränkt werden. Ist `start` nicht `None`, so wird Ansicht erst zu diesem Zeitpunkt aktiv, vorher ist sie inaktiv.

`end`
: [Datum oder `None`]
Ist `end` nicht `None`, so wird die Ansicht zu diesem Zeitpunkt deaktiviert.

`login_required`
: [Bool]
`login_required` ist `True` wenn im FormBuilder der Haken bei *Login wird benötigt* gesetzt ist. In dem Falle muß der Benutzer eingeloggt sein und Rechte an der App besitzen, um das Eingabeformular benutzen zu können.

`result_page`
: [Bool]
`result_page` ist `False` wenn im FormBuilder der Haken bei *Keine Ergebnisseite anzeigen* gesetzt ist. In dem Falle wird der Benutzer nach dem Absenden des Formulars zum „Bearbeiten“-Formular weitergeleitet.

`use_geo`
: [String]
`use_geo` legt fest, ob das Update-Template zu dieser Ansicht auf den Standort des Endgerätes zugreifen soll oder nicht. Es gibt drei mögliche Werte:

  `"no"`
  : Es erfolgt kein Zugriff auf die Standort-Daten.

  `"once"`
  : Es erfolgt ein einmaliger Zugriff auf die Standortdaten bei der ersten Benutzerinteraktion, oder wenn *Verwendung des aktuellen Ortes erlauben* am Fuß des Eingabeformulars angeklickt wird.

  `"watch"`
  : Es erfolgt ein anfänglicher Zugriff auf die Standortdaten (wie bei `"once"`) sowie jedes Mal wenn sich der Standort ändert.

`focus_control`
: [*Control* oder `None`]
Dasjenige `Control`-Objekt, das den Eingabefokus hat. Wird nur beim Anlegen neuer Datensätze automatisch anhand der Tab-Reihenfolge gesetzt.

`focus_first_control()`
: [Methode() → `None`]
Setzt den Fokus auf das Eingabefeld mit dem kleinsten Tab-Index.

`controls`
: [Dictionary(String → *ViewControl*) oder `None`]
Die View-Varianten der Controls (und zwar nur die die für diesen View konfiguriert sind). Dazu muß aber in der Datenquelle bei *Felder* mindestens *Alle Felder* gewählt werden.

`c_<identifier>`
: [*Control*]

`Control`-Objekte stehen auch über „Shortcut"-Attribute zur Verfügung. `view.c_beispiel` ist äquivalent zu `view.controls.beispiel`.

`layout_controls`
[Dictionary(String → *LayoutControl*) oder `None`]
Die Layout-Controls dieses Views (falls in der Datenquelle bei *Felder Alle Felder und Layout-Felder* ausgewählt wurde oder `None` falls nicht).

`lc_<identifier>`
[*LayoutControl*]
„Shortcut"-Attribut zum Zugriff auf die Layout-Controls. `view.lc_beispiel` ist äquivalent zu `view.layout_controls.beispiel`.

`ViewControl`

Ein `ViewControl`-Objekt enthält *View*-spezifische Attribute eines *Control*s. Im Gegensatz zu *Control* gibt es von `ViewControl` keine Unterklassen. Daher beinhaltet ein `ViewControl`-Objekt evtl. auch Attribute, die für das zugehörige *Control*-Objekt irrelevant sind. Da über den aktiven *View* diese *View*-spezifischen Attribute auch in den *Control*-Objekten zugänglich sind, können aber prinzipiell die `ViewControl`-Objekte ignoriert werden.

Attribute sind:

`id`
[String]
Der eindeutige Datenbank-Identifizierer des Feldes.

`label`
[String]
View-spezifische Feld-Beschriftung

`identifier`
[String]
Der `identifier` des zugehörigen *Control*-Objekts.

`view`
[*View*]
Das *View*-Objekt zu dem dieses `ViewControl`-Objekt gehört.

`control`
[*Control*]
Das *Control*-Objekt zu dem dieses `ViewControl`-Objekt gehört.

`type`
[String]
Der `type` des zugehörigen *Control*-Objekts.

`subtype`
[String oder `None`]
Der `type` des zugehörigen *Control*-Objekts.

`top`
[Integer]
Vertikale Position des Eingabefeldes in diesem *View*.

`left`
[Integer]
Horizontale Position des Eingabefeldes in diesem *View*.

`width`
[Integer]
Breite des Eingabefeldes in diesem *View*.

height
[Integer]
Höhe des Eingabefeldes in diesem *View*.

z_index
[Integer]
Überlagerungs-Reihenfolge des Eingabefeldes in diesem *View*.

default
[Objekt]
Standardwert für diesen Feld in diesem *View*.

tabindex
[Integer]
Tab-Reihenfolge des Eingabefeldes in diesem *View* (zum Durchschalten der Eingabefelder mit Hilfe der Tab-Taste).

minlength
[Integer oder `None`]
Die minimale String-Länge (`None` bedeutet ohne Einschränkung). (Nur für Felder vom Typ `"string"` relevant). .. index:: maxlength, ViewControl.maxlength

maxlength
Die maximale String-Länge (`None` bedeutet ohne Einschränkung). (Nur für Felder vom Typ `"string"` relevant).

required
[Bool]
`True` wenn dieses Feld in diesem *View* als Pflichfeld konfiguriert wurde.

placeholder
Der Platzhalter für das HTML-Eingabefeld. (Nur für Felder vom Typ `"string"` relevant).

mode
[`"DISPLAY"` oder `"EDIT"`]
Kann der Wert dieses Feldes in diesem *View* bearbeitet werden (`"EDIT"`) oder wird der Wert nur angezeigt (`"DISPLAY"`)? (`"DISPLAY"` wird verwendet, wenn das Feld im FormBuilder mit *Inhalt zur zum Lesen anzeigen* konfiguriert wurde.

labelpos
[`"TOP"`, `"LEFT"`, `"RIGHT"`, `"BOTTOM"` oder `None`]
Position der Feldbeschriftung relativ zum Eingabefeld in diesem *View*. Bei `None` wird die Beschriftung gar nicht angezeigt.

lookup_none_key
[String oder `None`]
Ist `none_key` nicht `None` wird im Eingabeformular eine „Nichts ausgewählt“-Option angeboten. Der Wert von `none_key` wird als Wert dieser Auswahl-Option verwendet. Ist `none_key` `None` wird keine „Nichts ausgewählt“-Option angezeigt.

(Nur für Felder vom Typ `"lookup"`, `"multiplelookup"`, `"applookup"` oder `"multipleapplookup"`).

lookup_none_label
[String oder `None`]
Wird als Beschriftung der „Nichts ausgewählt“-Option verwendet, wenn diese „Nichts ausgewählt“-Option angeboten wird. Ist `none_label` `None` sollte eine generische Beschriftung verwendet werden.

(Nur für Felder vom Typ `"lookup"`, `"multiplelookup"`, `"applookup"` oder `"multipleapplookup"`).

lookupdata
[Dictionary(String → *ViewLookupItem*)]
`lookupdata` beinhaltet die Auswahlmöglichkeiten als Dictionary (in der Reihenfolge wie die Optionen vom Benutzer angelegt wurde). Die Schlüssel sind die Identifierer der Option und die Werte sind *ViewLookupItem*-Objekt.

labelwidth
[Integer]
Breite des Labels in diesem *View*.

autoalign
[Bool]
`True` wenn die Breite des Labels automatisch vom FormBuilder berechnet wird. `False` wenn sie der Benutzer selbst angepasst hat.

autoexpandable
[Bool]
`True` wenn in FormBuilder für das Feld *Auto-Hinzufügen aktivieren* konfiguriert wurde.

x_<identifier>
[Objekt]
Es werden beliebige zusätzliche Attribute unterstützt deren Namen mit `x_` beginnt.

ViewLookupItem

Ein `ViewLookupItem` stellt den *View*-spezifischen Wert eines `LookupControl`s dar (d.h. einer Auswahlbox oder eines Optionsfeldes) wenn nicht die Datensätze einer anderen App als Auswahlmöglichkeiten verwendet werden. Ein `ViewLookupItem`-Objekt hat folgende Attribute:

key
[String]
Die eindeutige Kennung der Auswahl-Option.

label
[String]
Die Beschriftung der Auswahl-Option.

visible
[Bool]
`False` wenn diese Option in diesem *View* nicht angezeigt werden soll. `True` wenn sie angezeigt werden soll.

Bemerkung: `label` und `visible` können sich von *View* zu *View* unterscheiden.)

Attachment

Ein `Attachment`-Objekt beinhaltet die Daten zu einem Anhang. Genauer gesagt gibt es vier verschiedene Typen von Anhängen: `NoteAttachment`, `URLAttachment`, `FileAttachment` und `JSONAttachment`. Gemeinsam sind ihnen alle folgende Attribute:

id
[String]
Der interne Datenbank-Identifizierer für diesen Anhang.

type
[String]
Der Typ des Anhangs, einer der Werte `"note"`, `"url"`, `"file"`, `"image"` oder `"json"`.

record
[*Record*]
Der Datensatz zu dem dieser Anhang gehört.

label
[String]
Die Beschriftung des Anhangs.

active
[Bool]
Ist diser Anhang aktiv oder nicht?

Weiterhin haben `NoteAttachment` und `URLAttachment` ein Attribut `value`, das die Notiz bzw. die URL als String beinhaltet. Bei `JSONAttachment` ist das `value`-Attribut das JSON-Objekt. Bei `FileAttachment` ist das `value`-Attribut ein *File*-Objekt.

User

Ein `User`-Objekt stellt einen LivingApps-Benutzeraccount dar und hat folgende Attribute:

id
[String]
Der eindeutige Datenbank-Identifizierer des Benutzers.

gender
[String]
Das Geschlecht des Benutzers (`"m"` oder `"f"`).

firstname
[String]
Der Vorname des Benutzers.

surname
[String]
Der Nachname des Benutzers.

initials
[String]
Die Initialen des Benutzers.

email
[String]
Die Email-Adresse des Benutzers (gleichzeitig auch der Accountname).

lang
[String]
Die bevorzugte Sprache des Benutzers (beispielsweise `"de"` für Deutsch und `"en"` für Englisch).

image
[*File* oder `None`]
Das Benutzer-Portrait in Original-Größe.

Wird das Portrait in einer anderen Größe benötigt, kann *scaled_url(...)* benutzt werden.

keyviews
[Dictionary(String → *KeyView*) oder `None`]
Dieses Attribut ist nur dann nicht `None`, wenn es sich bei diesem Benutzer um den eingeloggten Benutzer handelt. In dieser Fall ist `keyviews` ein Dictionary das die „Key-Views“ dieses Users enthält, d.h. die Template-Ansichten die für diesen Benutzer ohne Login zur Verfügung stehen. Die Schlüssel sind die Identifizierer der jeweiligen Ansicht und die Werte sind die entsprechenden `KeyView`-Objekte.

streetname
[String]
Der Straßenname.

streetnumber
[String]
Die Hausnummer.

zip
[String]
Die Postleitzahl.

city
[String]
Die Stadt.

phone
[String]
Die Telefonnummer.

fax
[String]
Die Faxnummer.

summary
[String]
Selbstbeschreibung des Benutzers.

interests
[String]
Die Interessen.

personal_website
[String]
Die URL der persönlichen Website.

company_website
[String]
Die URl der Unternehmens-Website.

company
[String]
Der Firmenname.

position
[String]
Die Position innerhalb der Firma.

department
[String]
Die Abteilung innerhalb der Firma.

change(oldpassword, newpassword, newemail)
[Methode(String, String, String) → Liste(String)]
Mit der Methode `change` können Passwort und E-Mail-Adresse geändert werden. Die Angabe von `oldpassword` ist verpflichtend. Wenn ein `new`-Parameter `None` ist, wird der Wert auch nicht geändert. Die zurückgegebene Liste enthält die Meldungen zu den aufgetretenen Fehlern.

File

`File`-Objekte werden verwendet für Icons, Bilder und andere Dateien (also z.B. für die Icons der *App* und für Datei-Upload-Felder). Ein Datei-Objekt hat folgende Attribute:

url
[String]
Die absolute URL unter der die Datei abrufbar ist.

archive_url
[String]
Wenn sich diese Datei in einem ZIP-Archiv befindet kann diese URL benutzt werden, um dann mittels relativer URLs in dieser Datei auf andere Dateien im ZIP-Archiv zuzugreifen. Normalerweise trifft dies nur für Dateien zu, die LivingApps selbst verwendet. Ist diese Datei nicht Bestandteil eines ZIP-Archivs, hat `archive_url` den selben Wert wie `url`.

filename
[String]
Der ursprüngliche Dateiname der Datei.

mimetype
[String]
Der MIME-Typ der Datei (also z.B. `image/jpeg` für JPEG-Bilder oder `application/pdf` für PDF-Dateien)

size
[Integer]
Die Größe der Datei in Bytes.

width
[Integer oder `None`]
Falls es sich bei der Datei um ein Bild handelt, ist `width` die Breite des Bildes in Pixeln (ansonsten `None`).

height
[Integer oder `None`]
Die Höhe der Bilder in Pixeln (oder `None` falls es sich nicht um ein Bild handelt).

duration
[Integer oder `None`]
Wenn ich sich bei der Datei um ein Video handelt, ist `duration` die Dauer des Videos in Millisekunden (und `None` für andere Datei-Typen).

geo
[*Geo* oder `None`]
Enthält die Datei Geo-Daten (z.B. den Ort an dem das Bild oder Video aufgenommen wurde), so werden diese aus der Datei extrahiert und im Attribut `geo` gespeichert.

Damit jedoch Geo-Daten tatsächlich extrahiert werden, muß für die App bei der die Datei hochgeladen wird, dies erst aktiviert werden. Dies ist unter *Konfiguration* → *Erweitert* unter *Uploads* möglich (bzw. unter *Meine LivingApps* → *LivingApps* bei der jeweiligen App).

archive
[`File` oder `None`]
Ist diese Datei Bestandteil eines ZIP-Archivs, verweißt `archive` auf dieses Archiv, ansonsten ist `archive` `None`.

createdat
[Datum]
Der Zeitpunkt zu dem diese Datei angelegt wurde.

Geo

`Geo`-Objekte stellen einen geographische Ort auf der Erdoberfläche dar und werden als Werte von `GeoControl`-Felder verwendet. Ein `Geo`-Objekt hat folgende Attribute:

lat
[Zahl]
Der Breitengrad. Orte mit positivem Breitengrad befinden sich nördlich des Äquators, solche mit negativem Breitengrad südlich.

long
[Zahl]
Der Längengrad. Orte mit positivem Längengrad befinden sich östlich des Nullmeridians, solche mit negativem Längengrad westlich.

info
[String oder `None`]
Eine Beschreibung des durch `lat` und `long` bestimmten Ortes, z.B. eine Adresse.

Bemerkung: Wenn `Geo`-Objekte dargestellt werden, muss Open Streetmap als Datenquelle genannt werden. Mehr Informationen zur Nennung[14] und zur Lizenz[15].

`Installation`

Ein `Installation`-Objekt beschreibt einen Installationsvorgang bei dem eine App installiert wurde. Diese Information ist im *App*-Attribut `installation` zu finden.

Ein `Installation`-Objekt hat folgende Attribute:

`id`
: [String]
Der interne Datenbank-Identifizierer für diese Installation.

`name`
: [String]
Der Name der Installation.

`Category`

Ein `Category`-Objekt stellt eine Kategorie dar, der eine App zugeordnet ist. Dabei kann eine App mehreren Kategorien zugeordnet werden. Diese Zuordnung ist im *App*-Attribut `categories` zu finden.

Ein `Category`-Objekt hat folgende Attribute:

`id`
: [String]
Der interne Datenbank-Identifizierer für diese Kategorie.

`identifier`
: [String]
Der vom Benutzer vergebene Identifizierer für diese Kategorie. Dieser Identifizierer ist eindeutig innerhalb der „Geschwister"-Kategorien dieser Kategorie.

`name`
: [String]
Der Name der Kategorie.

`order`
: [Integer]
Die Reihenfolge, d.h. nach dieser Zahl sind die Kategorien innerhalb der übergeordneten Kategorie sortiert.

`parent`
: [*Category* oder `None`]
Die übergeordnete Kategorie (bzw. `None` falls sich diese Kategorie bereits auf oberster Ebene befindet).

`children`
: [Dictionary(String → *Category*) oder `None`]
Wenn in der Datenquellen-Konfiguration *Kategorien* die Option *Kategorien-Pfade* ausgewählt, so ist `children` `None`. D.h. über die den Apps zugeordneten Kategorien können über das `parent`-Attribut die Pfade zu diesen Kategorie im Kategorien-Baum rekonstriert werden, aber nicht die Bäume selbst.

  Dazu wird das `children`-Attribut benötigt, das aber nur gefüllt wird, wenn in der Datenquellen-Konfiguration *Kategorien* entweder *Kategorien-Bäume* oder *Kategorien-Bäume mit Apps* ausgewählt wurde. Dann ist `children` ein sortiertes Dictionary. Die Schlüssel sind die internen Datenbank-Identifizierer der untergeordneten Kategorie und die Werte sind die entsprechende `Category`-Objekte.

[14] https://www.openstreetmap.de/faq.html#lizenz
[15] https://opendatacommons.org/licenses/odbl/summary/

apps
[Dictionary(String → *App*) oder `None`]
`apps` beinhaltet die dieser Kategorie zugeordneten Apps, wenn in der Datenquellen-Konfiguration *Kategorien* die Option *Kategorien-Bäume mit Apps* ausgewählt ist. `apps` ist ein sortiertes Dictionary. Die Schlüssel sind die internen Datenbank-Identifizierer der jeweiligen App und die Werte sind die entsprechenden `App`-Objekte.

Ist in der Datenquellen-Konfiguration *Kategorien* die Option *Kategorien-Bäume mit Apps* nicht ausgewählt, ist `apps` `None`.

KeyView

`KeyView`-Objekte werden dazu benutzt, um einem Benutzer den Zugriff auf bestimmte Templates zu ermöglichen, ohne das dieser Benutzer sich im LivingApps-System einloggen muß. Die einzige Voraussetzung ist, das der Benutzer den ihm zugewiesenen Schlüssel (`key`) kennt.

`KeyView`-Objekte haben folgende Attribute:

id
[String]
Der interne Datenbank-Identifizierer für diese Kategorie.

identifier
[String]
Der vom Benutzer vergebene Identifizierer für diesen `KeyView`. Dieser Identifizierer ist eindeutig für diesen Benutzer.

name
[String]
Der Name für diesen `KeyView`, der z.B. dem Benutzer zur Navigation angezeigt werden kann.

key
[String]
Der Schlüssel-Wert (eine sechzigstellige Zeichenkette), die als Bestandteil der URL Zugriff auf das Template gewährt. Ist dieser Wert also z.B. `key`, ist unter der URL

http://my.living-apps.de/gateway/keyviews/key

die zugehörige Ansicht verfügbar.

user
[*User*]
Der Benutzer, dem dieser `KeyView` gehört. Mit den Rechten dieses Benutzers wird das Template aufgerufen.

ChainedLibrary

`ChainedLibrary`-Objekte werden erzeugt, wenn auf die Shortcut-Attribute für verkettete Template-Suche (`cl_*` in *Globals* und *App*) zugegriffen wird. Ein `ChainedLibrary`-Objekt hat folgende Attribute:

identifier
[String]
Der Identifier des App-Parameters über den die Verknüpfung hergestellt wurde.

app
[*App*]
Die App zu der diese Verknüpfung gehört. Es gilt beispielsweise für jede *App* `app`:

```
app.cl_gurk.app is app
```

(Dies gilt unabhängig davon, ob der *AppParameter* `gurk` in der *App* `app` existiert, und ob er vom Typ `App` ist).

`templates`
: [Dictionary(String → UL4-Template)]
Alle Templates, die sich aus der Kombination der internen Templates der App `app` und der Basis-Templates ergeben.

  Besitzt die *App* `app` einen App-Parameter mit dem Identifier `identifier` und ist dieser vom Typ `App` so ergeben sich diese Basis-Templates aus den internen Templates der App auf die dieser Parameter verweist (und rekursiv aus dem Templates der App auf die wiederrum dessen App-Parameter `identifier` verweist, usw.).

  Besitzt die *App* `app` keinen App-Parameter mit dem Identifier `identifier` oder ist dieser nicht vom Typ `App` so sind die Basis-Templates die Templates der Template-Library mit dem Identifier `identifier`.

  Gibt es auch diese Template-Library nicht, so bestehen die Basis-Templates aus einem leeren Dictionary (d.h. die Templates sind nur die internen Templates der App selbst).

  Die Schlüssel des Dictionarys sind dabei der Identifizierer des Templates und die Werte die UL4-Templates selbst.

`t_<identifier>`
: [UL4-Template]
`Template`-Objekte stehen auch über sogenannte „Shortcut"-Attribute zur Verfügung. D. h., dass beispielsweise das Template `hurz`, das normalerweise unter `globals.cl_gurk.templates.hurz` zu finden ist, auch direkt als `globals.cl_gurk.t_hurz` zur Verfügung steht.

2.25.2 Variablen

Einem Anzeige- oder E-Mail-Template werden folgende Variablen übergeben:

`globals`
: [*Globals*]
Globale Informationen

`datasources`
: [Dictionary(String → *DataSource*)]
`datasources` beinhaltet die konfigurierten Datenquellen. Die Schlüssel sind die Identifizierer der Datenquellen und die Werte sind `DataSource`-Objekte.

`record`
: [*Record*]
Einen Detail-Template (d.h. einem *Anzeige-Template*, bei dem unter *Typ Detail* ausgewählt wurde), wird zusätzlich noch der Datensatz übergeben.

  Ebenso wird bei einen *E-Mail-Template* der Datensatz, der die E-Mail-Versendung auslöst, übergeben.

  Bei anderen Templates ist diese Variable `None`.

`app`
: [*App*]
Die App zu der dieses Anzeige- oder E-Mail-Template gehört.

2.25.3 HTTP-Request-Parameter

Zur Steuerung der in einem Anzeige-Template zur Verfügung stehenden Daten verwendet die LivingAPI selbst einige HTTP-Request-Parameter. Diese werden in folgendem Kapitel beschrieben:

HTTP-Request-Parameter

Einige HTTP-Request-Parameter werden von der LivingAPI selbst verwendet zur Steuerung welches Anzeige-Template zur Datenanzeige verwendet wird, und welche Daten diesem Anzeige-Template zur Verfügung gestellt werden.

Alle Parameter-Namen die mit `la-` beginnen sind für die LivingAPI reserviert und sollten nicht für andere Zwecke verwendet werden.

Template

Der Request-Parameter `template` bestimmt welches Anzeige-Template verwendet wird. D.h. der Parameter-Wert muß der *Identifizierer* eines Anzeige-Templates sein. Ist der Parameter `template` nicht angegeben wird das Standard-Anzeige-Template verwendet (d.h. das Template dessen *Standard*-Haken gesetzt ist). Gibt es kein Standard-Anzeige-Template erhalten Sie einen 404-HTTP-Fehler.

Paging

Wenn eine Datenquelle so konfiguriert ist, daß die Datensätze dieser App einem Anzeige-Template zur Verfügung gestellt werden, kann es vorkommen, daß die Anzahl der Datensätze zu groß ist um sie auf einer HTML-Seite anzuzeigen. In diesem Fall können Sie durch Request-Parameter festlegen, daß für diese Datenquelle nur ein Teil der Datensätze zur Verfügung gestellt wird. Wenn Sie z.B. eine Datenquelle mit dem Identifizierer `personen` haben, können Sie mit folgendem Request-Parameter festlegen, daß Sie nur die ersten 20 Datensätze haben wollen:

```
la-ds-personen-paging=0_20
```

`0_20` bedeutet dabei, daß beginnend beim ersten Datensatz (also mit dem Index 0) 20 Datensätze ausgeliefert werden sollen.

Dieselbe Funktionalität steht für untergeordnete Datensätze zur Verfügung. Wenn z.B. bei obiger Datenquelle `personen` noch eine Konfiguration für untergeordnete Datensätze mit den Identifizierer `bestellungen` existiert, können Sie folgenden Request-Parameter verwenden:

```
la-dsc-personen-1234567890abcdef12345678-bestellungen-paging=20_10
```

Damit werden dann bei dem Personen-Datensatz mit der Id `1234567890abcdef12345678` aus der Datenquelle `personen` von den untergeordneten `bestellungen`-Datensätzen nur der 20.-29. ausgeliefert. Ist der Parameter für diesen Datensatz nicht angegeben, wird zusätzlich noch der Parameter

```
la-dsc-personen-bestellungen-paging
```

verwendet (d.h. dieser Parameter kann verwendet werden um ein Default-Paging für die untergeordneten BEstellungs-Datensätze *aller* Personen-Datensätze anzugeben).

2.26 Statische Resourcen

Zur Verwendung in Ihren Anzeige-Templates stehen Ihnen in LivingApps einige bekannte CSS- und Javascript-Frameworks zur Verfügung. Diese sind unter der URL `/static` zugänglich.

KAPITEL 3

Programmierschnittstellen

LivingApps bietet einige Programmierschnittstellen, mit deren Hilfe Sie mit Ihren Daten in LivingApps arbeiten können.

3.1 GraphQL LivingAPI

Bemerkung: Diese Seite ist für Entwickler gedacht. Manche Links leiten auf englischsprachige Webseiten weiter und diese Seiten sind meist für Entwickler geschrieben.

Mithilfe der GraphQL LivingAPI können Sie in jeder Sprache, die einen Https Client unterstützt, mit LivingApps interagieren.

Die GraphQL LivingAPI (GQL LAPI) ermöglicht es auf Daten einer in einem Template definierten Datenquelle zuzugreifen und diese Daten zu aktualisieren. Zusätzlich können neue Datensätze angelegt werden oder existierende gelöscht werden.

Die Antworten sind hierbei immer gültiges JSON.

Bemerkung: Die LivingApps Daten werden als JSON zurückgegeben und nicht wie beim Python SDK oder Javascript SDK als UL4on übergeben. Dadurch sind GQL LAPI nutzende Anwendungen unabhängiger von LivingAPI und UL4 Updates als zuvor. Falls es durch ein Update zu einer Veränderung an dem Aufbau der LivingAPI kommt, so werden betroffene Felder mit polyfills versehen und als deprecated markiert. Von da an haben Sie noch 90 Tage Zeit die Anwendung an die neue API anzupassen.

Bemerkung: Auf dieser Seite wird nicht auf die allgemeine Nutzung von GraphQL eingegangen. Falls Sie GraphQL noch nicht kennen, sehen Sie sich folgende Seite[16] an.

[16] https://graphql.org/learn/

3.1.1 Einloggen

Die GQL LAPI unterstützt zwei Einloggmöglichkeiten. Zum einen können der Nutzername und das Passwort mit jedem Request mitgeschickt werden. Zum anderen wird der OpenId-Connect Standard unterstützt.

Ersteres eignet sich besonders gut für Backend oder Entwicklungssetups.

Zweiteres benötigt Zugang zu einem von LivingApps unterstützten OIDC (OpenId-Connect) Provider. Wenden Sie sich dafür bitte an unser Team.

3.1.2 Login mit Nutzernamen und Passwort

Es müssen die Header

- `x-la-gql-user`
- `x-la-gql-pwd`

gesetzt werden.

Bspw.

```
curl --request POST \
    --url https://graphql-pwa-la-prod.living-apps.de/ \
    --header 'Content-Type: application/json' \
    --header 'x-la-gql-pwd: <meinGeheimesPasswort>' \
    --header 'x-la-gql-user: <max@mustermann.de>' \
    --data '{"query":"query {...}"}'
```

3.1.3 Login mit OIDC

Für den Login mit OIDC muss von dem OIDC Provider ein Accesstoken geladen werden. Dieser Accesstoken muss dann im Header „authorization“ mitgeschickt werden.

Bemerkung: Achtung: die Accesstokens haben eine relativ kurze Lebenszeit (bspw. 5 Minuten) und müssen stetig erneuert werden. Bitte machen Sie sich mit OIDCs „Authorization Code Flow“[17] bekannt.

```
curl --request POST \
    --url https://graphql-pwa-la-prod.living-apps.de/ \
    --header 'Content-Type: application/json' \
    --header 'authorization: Bearer <your accesstoken>' \
    --data '{"query":"query {...}"}'
```

Bemerkung: Für Frontendprojekte wie PWAs ist „Login mit OIDC“ zu bevorzugen.

---

[17] https://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth

3.1.4 Dokumentation

Wir bieten für die GQL LAPI einen Playground[18] inklusive kurzer englischsprachiger Typendokumentation an. Falls eine deutschsprachige Dokumentation benötigt wird, können die Typen auch in der LivingAPI Dokumentation nachgelesen werden.

3.1.5 GraphQL und Maps

In der LivingAPI werden häufig Maps genutzt aber in GraphQL gibt es keine Maps. Die Alternative sind Listen die Schlüssel-Wert Objekte enthalten. Falls Mapeinträge den Schlüssel selbst als Feld gespeichert haben wird in GraphQL eine Liste der Werte zurückgegeben.

Bspw.

```
// livingapi
{
    [recordId: string]: Record
}
// graphql
RecordV1[]
```

Wenn Maps benötigt werden, müssen diese auf der Clientseite aus der (Schlüssel-)Wert Liste bebildet werden.

3.1.6 Antworten Caching

Die GQL LAPI wird innerhalb der LivingLogic für die Entwicklung von Progressive Web Apps genutzt. Progressive Webs Apps sind Webseiten die wie native Anwendungen genutzt werden können. So können auch PWAs bei keiner oder sehr schlechter Internetverbindung genutzt werden. Bei einer sehr schlechten Internetverbindung kann es dazu kommen, dass eine Anfrage abgeschickt wird und die Antwort nicht empfangen werden kann. Um zu verhindern, dass dann Mutations doppelt ausgeführt werden können Anfragen mit einer Id versehen werden. Wird eine Anfrage mit der gleichen Id vom gleichen Nutzer mehrfach innerhalb einer Stunde ausgeführt so wird das Ergebnis der ersten Anfrage zurückgegeben und die GQL LAPI führt den Query oder die Mutation nicht mehr aus.

```
curl --request POST \
    --url https://my.living-apps.de/gql/ \
    --header 'Content-Type: application/json' \
    --header 'x-la-gql-pwd: <meinGeheimesPasswort>' \
    --header 'x-la-gql-user: <max@mustermann.de>' \
    --header 'x-la-gql-reqid: <meineReqId>' \
    --data '{"query":"mutation {...}"}'
```

3.2 Python-SDK

3.2.1 Überblick

Mit dem Python-SDK können Sie mithilfe der Programmiersprache Python[19] mit LivingApps interagieren.

Sie können die Daten die Sie in den Datenquellen für eines Ihrer Anzeige-Templates konfiguriert haben, abrufen, neue Datensätze anlegen, sowie existierende ändern und löschen.

[18] https://graphql-pwa-la-prod.living-apps.de/
[19] http://www.python.org/

3.2.2 Installation

Das Python-SDK benötigt mindestens Python 3.6. Zum Anlegen eines virtuellen Environment wird außerdem das `venv`[20]-Modul benötigt. Des weiteren wird zur Installation evtl. ein C-Compiler benötigt. Wenn dies nicht bereits Teil der Standard-Installation auf Ihrem System ist, kann beides mittels:

```
$ apt-get install python3.6-venv python3.6-dev
```

installiert werden (bzw. mit dem äquivalenten Aufruf für Ihr System).

Sodann sollte eine neue virtuelle Umgebung angelegt werden, in der alle benötigten Module installiert werden. Wir nennen diese virtuelle Umgebung `livingapps` und installieren sie im Verzeichnis `~/pyvenvs`:

```
$ python3.6 -mvenv ~/pyvenvs/livingapps
```

Anschließend können wir diese Umgebung aktivieren:

```
$ . ~/pyvenvs/livingapps/bin/activate
```

Um Sie darauf hinzuweisen, daß Sie diese Umgebung aktiviert haben, wird forthin dem Prompt der Name der Umgebung als Präfix vorangestellt. D.h. der Prompt ist ab jetzt `(livingapps) $`.

Um das Python-SDK selbst zu installieren, gehen Sie folgenden Befehl ein:

```
(livingapps) $ pip install ll-la
```

Dabei werden auch die benötigten Module `ll.xist`[21], `requests`[22] und `geocoder` installiert.

Um das interaktive Ausprobieren des Python-SDK zu vereinfachen, empfielt es sich IPython zu installieren mittels:

```
(livingapps) $ pip install ipython
```

(Die folgenden Beispiele zeigen die Verwendung von IPython).

3.2.3 Beispiel-App

Als Beispiel-App benutzen wir eine App, mit der wir Daten zu berühmten Personen verwalten. Wir legen zwei Apps an. Die erste (namens „Persons“) enthält die Personen selbst. Sie hat folgende Felder:

Name	Identifizierer	Typ	Sonstiges
Firstname	`firstname`	`string/text`	
Lastname	`lastname`	`string/text`	
Sex	`sex`	`lookup/select`	Auswahl `male/female`
Field of activity	`field_of_activity`	`multipleapplookup/select`	Mehrfach-Auswahl aus der zweiten App
Date of birth	`date_of_birth`	`date/date`	
Date of death	`date_of_death`	`date/date`	
Grave	`grave`	`geo`	
Portrait	`portrait`	`file`	

Die zweite App (namens „Fields of activity“) verwaltet die Tätigkeitsfelder und hat zwei Felder:

Name	Identifizierer	Typ	Sonstiges
Name	`name`	`string/text`	
Parent	`parent`	`applookup/select`	Auswahl aus derselben App

[20] https://docs.python.org/3.8/library/venv.html#module-venv
[21] https://python.livinglogic.de/XIST.html#module-ll.xist
[22] https://requests.readthedocs.io/en/latest/api/#module-requests

3.2.4 Vorbereitungen in der App

Damit Sie auf die Daten Ihrer App zugreifen können, müssen Sie für jeden Datenexport den Sie benötigen, das passende Anzeige-Template anlegen. Dazu muß Ihr Account vom Administrator als Experten-User freigeschaltet worden sein.

In Kürze: Fügen Sie unter *Konfiguration* → *Erweitert* bei *Anzeige-Templates* ein neues Anzeige-Template hinzu. Verwenden Sie als *Identifizierer* `export`, setzen Sie den *Typ* auf *Liste* und setzen Sie *Standard?*.

Bei *Quelltext* muß nichts eingegeben werden (da wir die Daten ja gar nicht in dem Anzeige-Template anzeigen wollen, sondern mit dem SDK darauf zugreifen wollen). Bei *Berechtigung für* können Sie auswählen, wer auf diese Daten zugreifen darf. (Wenn Sie *alle Benutzer* wählen, muß der Benutzer nicht eingeloggt sein, um zugreifen zu können, d.h. die Daten sind öffentlich.) Klicken Sie auf *Speichern* um Ihr Anzeige-Template abzuspeichern.

Sodann können Sie die Datenquellen die Sie benötigen unter *Datenquellen* anlegen. Wir legen hier zwei Datenquellen für unsere beiden Apps an. Klicken Sie bei Ihrem neuen Anzeige-Template auf den Maskenlink *Datenquellen* und dann auf *Hinzufügen*. Bei *App* wählen Sie die App „Persons“ aus, als *Identifizierer* geben Sie `persons` ein, bei *Felder/Datensätze* wählen Sie *Felder und Datensätze*, sowie bei *Felder Alle Felder*. Klicken Sie dann auf *Speichern* um Ihren neue Datenquelle abzuspeichern.

Legen Sie nun eine zweite Datenquelle für die App „Fields of activity“ an (mit dem *Identifizierer* `fieldsofactivity`).

Eine ausführliche Beschreibung von Anzeige-Templates und Datenquellen findet sich unter *Anzeige-Templates*.

3.2.5 Verwendung des SDKs

Starten Sie IPython mit

```
(livingapps) $ ipython
Python 3.6.4 (default, Jan 15 2018, 09:29:23)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.
In [1]:
```

Anschließend importieren Sie das `la`-Modul:

```
In [1]: from ll import la
```

Nun können Sie sich mit Ihrem Account bei LivingApps einloggen:

```
In [2]: handler = la.HTTPHandler("https://my.living-apps.de/", "username", "password")
```

Benutzername und Passwort kann auch weggelassen werden, dann können Sie nur auf öffentliche Anzeige-Templates zugreifen, d.h. auf Templates, bei denen unter *Berechtigung für alle Benutzer* ausgewählt wurde.

Um nun von Ihrer App und Ihren Anzeige-Template Daten abzuholen, benötigen Sie den Identifizierer Ihrer App. Diesen finden sie wenn Sie sich auf der Startseite der jeweiligen App befinden. Die URL diese Seite lautet in unserem Beispiel:

```
https://my.living-apps.de/apps/5bffc841c26a4b5902b2278c.htm
```

`5bffc841c26a4b5902b2278c` ist dann der gesuchte Identifizierer.

Damit können Sie dann die Daten holen, die Sie für Ihr Template konfiguriert haben:

```
In [3]: data = handler.viewtemplate_data("5bffc841c26a4b5902b2278c")
In [4]: data
Out[4]:
{'globals': <ll.la.Globals version='7' platform='LivingApps' at 0x56d62e8>,
 'app': <ll.la.App id='5bffc841c26a4b5902b2278c' name='LA-Demo: Persons' at 0x56d6080>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
↪,
 'record': None,
 'apps': {'5bffc841c26a4b5902b2278c': <ll.la.App id='5bffc841c26a4b5902b2278c' name=
↪'LA-Demo: Persons' at 0x56d6080>,
  '5bffc44c5be111d74ed79972': <ll.la.App id='5bffc44c5be111d74ed79972' name='LA-Demo:␣
↪Fields of activity' at 0x57d0550>},
 'datasources': {'persons': <ll.la.DataSource id='5bffccdb8cd2298964e2c7b1'␣
↪identifier='persons' at 0x57ca1d0>,
  'fieldsofactivity': <ll.la.DataSource id='5bffdadd22f815dd7480dfcc' identifier=
↪'fieldsofactivity' at 0x57ca208>}}
```

Wenn Sie ein weiteres Anzeige-Template angelegt haben (z.B. mit dem Identifizierer `specialexport`), können Sie die Daten holen, indem Sie diesen Template-Identifizierer als zweites Argument übergeben:

```
In [3]: data = handler.viewtemplate_data("5bffc841c26a4b5902b2278c", "specialexport")
```

(d.h. bei dem Anzeige-Template, das als *Standard?* konfiguriert wurde, muß der Template-Identifizierer nicht angegeben werden).

Das Objekt `data`, das Sie als Ergebnis erhalten, besitzt drei Attribute:

globals
: Globale Informationen zum System und dem eingeloggten Benutzer. Dieses Python-Objekt unterstützt die unter *Globals* dokumentierten Attribute und Methoden der LivingAPI, die Ihnen im Sourcecode des Anzeige-Templates zur Verfügung stehen.

datasources
: Ein Dictionary mit den von Ihnen konfigurierten Datenquellen. Die Keys sind die Identifizierer und die Werte *DataSource*-Objekte. Diese Objekte unterstützen ebenfalls die LivingAPI.

app
: Die App, zur der das Anzeige-Template gehört (d.h. die App, deren Id Sie an den `get()`-Aufruf übergeben haben).

Damit können wir auf unsere App „Fields of activity“ folgendermaßen zugreifen:

```
In [3]: tfapp = data.datasources.fieldsofactivity.app
In [4]: tfapp
Out[4]: <ll.la.App id='5bffc44c5be111d74ed79972' name='LA-Demo: Fields of activity'␣
↪at 0x10b857d30>
```

Die Datensätze sind folgendermaßen zugänglich:

```
In [5]:  for r in tfapp.records.values():
.....:      print(r.v_name)
.....:
Sport
Politics
Literature
Industry
Music
Art
Physics
Maths
Science
Computer science
Film
In [6]:
```

Die Datensätze sind *Record*-Objekte und unterstützen die entsprechenden Attribute und Methoden.

Aus der anderen Datenquelle können wir uns die Daten zu den Personen holen:

```
In [6]: papp = data.datasources.persons.app
In [7]: list(papp.records.values())
Out[7]:
[<ll.la.Record
     id='5bf40f8f11b9ceb4ce7c75ac'
     v_firstname='Albert'
     v_lastname='Einstein'
     v_sex=<ll.la.LookupItem key='male' label='Male' at 0x10fa5bd30>
     v_fieldofactivity=[<ll.la.Record
          id='5bf40f8dc511b06d3be85c7d'
          v_name='Physics'
          at 0x10fa3ea90>]
     v_date_of_birth=datetime.date(1879, 3, 14)
     v_date_of_death=datetime.date(1955, 4, 15)
     at 0x10fa5beb8>,
 <ll.la.Record
     id='5bf40f8f2249662d2b4a8c0b'
     v_firstname='Marie'
     v_lastname='Curie'
     v_sex=<ll.la.LookupItem key='female' label='Female' at 0x10fa5be10>
     v_fieldofactivity=[<ll.la.Record
          id='5bf40f8dc511b06d3be85c7d'
          v_name='Physics'
          at 0x10fa3ea90>]
     v_date_of_birth=datetime.date(1867, 11, 7)
     v_date_of_death=datetime.date(1934, 7, 4)
     at 0x10fa3ef98>,
 <ll.la.Record
     id='5bf40f8f2ecda576c9d524fe'
     v_firstname='Muhammad'
     v_lastname='Ali'
     v_sex=<ll.la.LookupItem key='male' label='Male' at 0x10fa5bd30>
     v_fieldofactivity=[<ll.la.Record
          id='5bf40f8e29069905fe8c71f7'
          v_name='Sport'
          at 0x10fa3e748>]
     v_date_of_birth=datetime.date(1942, 1, 17)
     v_date_of_death=datetime.date(2016, 6, 3)
     at 0x10fa670b8>,
 <ll.la.Record
     id='5bf40f8f3cc3d6e19dbe5c50'
     v_firstname='Marilyn'
     v_lastname='Monroe'
     v_sex=<ll.la.LookupItem key='female' label='Female' at 0x10fa5be10>
     v_fieldofactivity=[<ll.la.Record
          id='5bf40f8de2efb2a43c8185d2'
          v_name='Film'
          at 0x10fa3e8d0>]
     v_date_of_birth=datetime.date(1926, 6, 1)
     v_date_of_death=datetime.date(1962, 8, 4)
     at 0x10fa67160>,
 <ll.la.Record
     id='5bf40f8f493c32a841ade504'
     v_firstname='Elvis'
     v_lastname='Presley'
     v_sex=<ll.la.LookupItem key='male' label='Male' at 0x10fa5bd30>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    v_fieldofactivity=[<ll.la.Record
         id='5bf40f8dee0ef934d6fe34de'
         v_name='Music'
         at 0x10fa3e7f0>]
    v_date_of_birth=datetime.date(1935, 1, 8)
    v_date_of_death=datetime.date(1977, 8, 16)
    at 0x10fa671d0>,
<ll.la.Record
    id='5bf40f8f58b852d36c8dc59c'
    v_firstname='Bernhard'
    v_lastname='Riemann'
    v_sex=<ll.la.LookupItem key='male' label='Male' at 0x10fa5bd30>
    v_fieldofactivity=[<ll.la.Record
         id='5bf40f8dba17a22c136c8f5b'
         v_name='Maths'
         at 0x10fa3e9e8>]
    v_date_of_birth=datetime.date(1826, 6, 17)
    v_date_of_death=datetime.date(1866, 6, 20)
    at 0x10fa67278>,
<ll.la.Record
    id='5bf40f8f692fb0cfbaf2c5b7'
    v_firstname='Carl Friedrich'
    v_lastname='Gauß'
    v_sex=<ll.la.LookupItem key='male' label='Male' at 0x10fa5bd30>
    v_fieldofactivity=[<ll.la.Record
         id='5bf40f8dba17a22c136c8f5b'
         v_name='Maths'
         at 0x10fa3e9e8>]
    v_date_of_birth=datetime.date(1777, 4, 30)
    v_date_of_death=datetime.date(1855, 2, 23)
    at 0x10fa67320>,
<ll.la.Record
    id='5bf40f8f78befe7d55dc40c2'
    v_firstname='Ronald'
    v_lastname='Reagan'
    v_sex=<ll.la.LookupItem key='male' label='Male' at 0x10fa5bd30>
    v_fieldofactivity=[<ll.la.Record
         id='5bf40f8de2efb2a43c8185d2'
         v_name='Film'
         at 0x10fa3e8d0>,
     <ll.la.Record
         id='5bf40f8e1202ba6352c2c3c9'
         v_name='Politics'
         at 0x10fa3e0b8>]
    v_date_of_birth=datetime.date(1911, 2, 6)
    v_date_of_death=datetime.date(2004, 6, 5)
    at 0x10fa673c8>]
```

KAPITEL 4

Javascript Helfer Bibliothek

Die Javascript Helfer Bibliothek ist eine Ansammlung an Javascript Bibliotheken, welche Ihnen zur Nutzung in Ihren Anzeige Templates bereitgestellt wird.

Bemerkung: Die Bibliothek und Dokumentation wendet sich an Personen die bereits Erfahrung mit Javascript haben. Es ist vorteilhaft wenn Begriffe wie Variablen, Funktionen, Paramter und Promises bekannt sind. Viele Beispiele in der Doku können mit minimalen Anpassungen genutzt werden.

Warnung: Die Bibliothek unterstützt nur moderne Browser!

4.1 Funktionen

4.1.1 Teilen

Lassen Sie Ihre Nutzer ganz leicht Links zu Ihren Formularen und Anzeigetemplates via Social Media teilen.

Bemerkung: Auf Mobilgeräten wird die Share Funktion[23] von der Web Share API genutzt. Diese wird auf dem Desktop nicht umfassend unterstützt, weshalb auf dem Desktop ein Modal zum Teilen via gängiger Social Media Kanäle bereitgestellt wird.

[23] https://developer.mozilla.org/en-US/docs/Web/API/Navigator/share

Installation

Importieren Sie den Stylesheet.

```
<link rel="stylesheet" type="text/css" href="/static/ll-form-utils/0.2.4/dist/share.
↪css" />
```

Importieren Sie das Javascript-Modul und verwenden Sie es folgendermaßen:

```
<p>
    <button id="shareUrl">Teile URL</button>
    <button id="shareText">Teile Text</button>
    <button id="shareForm">Teile Formular</button>
</p>

<!-- Importieren Sie das share Skript -->
<script src="/static/ll-form-utils/0.2.4/dist/share.js"></script>
<script>

// Teilen Sie eine URL wenn der "Teile URL" Button angeklickt wurde
document.getElementById('shareUrl').addEventListener('click', () => {
    la_share_module.la_share_url('https://my.living-apps.de');
});

// Teilen Sie einen Text mir URL wenn der "Teile Text" Button angeklickt wurde
document.getElementById('shareText').addEventListener('click', () => {
    la_share_module.la_share_text('Ich habe eine Lösung gefunden auf:', 'https://my.
↪living-apps.de');
});

// Teilen Sie eine URL zu einem Formular wenn der "Teile Formular" Button angeklickt␣
↪wurde
document.getElementById('shareForm').addEventListener('click', () => {
    la_share_module.la_share_form(
        'Bitte füllen Sie folgendes Formular aus: ',
        'https://my.living-apps.de/gateway/apps/646cab89ebc17480353d685d/new?
↪view=646cab894f066018ba59e382'
    );
});
</script>
```

Parameter

Es werden, wie im Installationsbeispiel zu sehen, drei verschiedene Funktionen angeboten. Diese erfüllen den gleichen Zweck und unterscheiden sich nur in dem Titel und Hover Effekten auf dem Desktop.

```
type DefaultSupportedLanguages = 'de' | 'en';
function la_share_url(url: string, lang: DefaultSupportedLanguages = 'de'): Promise
↪<void>;
function la_share_text(text: string, url: string, lang: DefaultSupportedLanguages =
↪'de'): Promise<void>;
function la_share_form(text: string, url: string, lang: DefaultSupportedLanguages =
↪'de'): Promise<void>;
```

Tab. 1: Parameter

Parameter	Defau	Beschreibung
`url`		Geben Sie hier den Link an den Sie teilen möchten. Falls der Link nicht auf `https://my.living-apps/` verweist wird eine Warnung angezeigt.
`text`		Geben Sie einen beschreibenden Text zu Ihrem Link an.
`lang`	`'de'`	Geben Sie an in welcher Sprache die Texte im Desktop Popup angezeigt werden soll. Dieser Parameter hat keinen Einfluss auf den übergebenen Text!

4.1.2 Mail Validieren

Reduzieren Sie die Eingabe von ungültigen E-Mail-Adressen indem Sie die Eingabe vom Nutzer nochmal bestätigen lassen.

Installation

Importieren Sie das Stylesheet.

```
<link rel="stylesheet" type="text/css" href="/static/ll-form-utils/0.2.3/dist/
↪validate_mail.css" />
```

Importieren Sie das Javascript-Modul und verwenden Sie es folgendermaßen:

```
<p>
    <input id="mail-input" value="mail-mit-tppfehler@mi.com" />
    <button id="validate-button">Validieren Sie die E-Mail-Adresse</button>
</p>
<p id="validate-result">Resultat: Klicken Sie auf den "Validieren Sie die E-Mail-
↪Adresse" Button</p>
<!-- Importieren Sie das validate_mail Skript -->
<script src="/static/ll-form-utils/0.2.3/dist/validate_mail.js"></script>
<script>
// Erstellen Sie Referenzen zu den DOM Elementen
const inputElement = document.getElementById('mail-input');
const resultElement = document.getElementById('validate-result');
// Registrieren Sie alle Eventlistener, welche die Funktion auslösen sollen.
document.getElementById('validate-button').addEventListener('click', async () => {
    // Lesen Sie den aktuellen Eingabewert aus.
    const value = inputElement.value;
    // Führen Sie die Funktion aus.
    const result = await la_validate_mail_module.la_validate_mail(value);
    // Geben Sie das Resultat an den Nutzer zurück
    inputElement.value = result;
    resultElement.innerText = `Resultat: ${result}`;
});
</script>
```

Parameter

```
function laValidateMail(maybeMail: string, userMail?: string, target?: HTMLElement):␣
↪Promise<string>
```

Tab. 2: Parameter

Param-ter	Default	Beschreibung
maybeMa		Die String-Eingabe welche vermutlich eine E-Mail-Adresse ist.
userMai	undefir	Eine bereits bekannte E-Mail-Adresse.
target	undefir	Ein DOM Element ohne Kinder, in welches die Oberfläche gemounted wird. Ist der Wert undefined wird ein DOM Element automatisch erzeugt.

Stichwortverzeichnis

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y

Z