

StuRa - Mitgliederdatenbank

version

Team T8

August 13, 2020

Contents

Welcome to StuRa - Mitgliederdatenbank's documentation!	1
User Documentation	1
Working with Members (Mitglieder)	1
Create a Member	1
Alter a Member	2
Delete a Member	2
Working with Organizational units (Organisationseinheiten)	4
Add a new Organizational unit	4
Alter an Organizational unit	5
Delete an Organizational unit	6
Working with Subsections (Unterbereiche)	7
Create a new Subsection	7
First Option via Organizational units	7
Second Option via Subsections	7
Alter a Subsection	8
Delete a Subsection	8
Working with Functions (Funktionen)	9
Add a new Function	9
Create a Function that belongs directly to an Organizational unit	10
Create a Function that belongs to a Subsection of an Organizational unit	10
Alter a Function	11
Delete a Function	11
Working with Checklists (Checklisten)	11
Create a new General Task	11
Create Right for a Function	12
Create a new Checklist	13
Delete a Checklist	14
Working with Users and Admins	15
Create an User	15
Create a User (with Admin rights)	16
Admin Documentation	17
Deployment	17
Sources:	17
Requirements:	17
Basic configuration:	17
Adjust Django Project Settings:	18
Configure Apache2:	18
Solve some Permission Issues:	19
Cronjobs	19
Update the Application	20

Update From GitHub with no changes in migrations:	20
Developer Documentation	20
Introduction Development	20
Clone the Git-Repository, install the requirements and fire up the Webserver for development	20
System Architecture	21
The V in MVC	23
Components	26
Database Structure	28
Activity Diagrams	29
Code Documentation	31
Login	31
Views	31
Templates	32
login.html	32
Ämter	32
Models	32
Views	43
Templates	44
main_screen.html	44
department_row.html	44
Historie	44
Abhängigkeiten	44
django-simple-history	44
Views	44
Templates	45
list.html	45
tabs/*	45
tabs/_pagination.html	45
row.html	45
rowContent.html	45
_titleBuilder.html	45
_noResultsRow.html	46
_modalDataIncludes.html	46
modalData/*	46
Template Tags	46
Mitglieder	46
Abhängigkeiten	47
simplejson	47
Views	47
Templates	50
aemter.html	50

amt_dropdown_list_options.html	50
bereich_dropdown_list_options.html	50
email.html	50
email_input.html	50
mitglieder.html	51
mitglied_erstellen_bearbeiten.html	51
modal.html	51
row.html	51
Checklisten	51
Models	51
Views	60
Templates	61
main_screen.html	61
_createModal.html	62
_deleteModal.html	62
_funktionSelectOptions.html	62
Template Tags	62
Importscripts	62
How to use:	62
CSV Data:	62
Functions:	63
Tests	63
Sources	63
Dependencies	63
selenium	63
Webdriver for Firefox	63
MyTestCase	63
Helper Functions	64
Login Module Tests	65
Testcase 001	65
Testcase 002	65
Mitglieder Module Tests	65
Testcase 003	65
Testcase 004	65
Testcase 005	66
Testcase 006	66
Aemter Module Tests	66
Testcase 007	66
Testcase 008	67
Testcase 009	68
Latest test coverage report	68
Commands	70

delete_old_historie	70
clean_duplicate_history	70
Index	71
Python Module Index	77

Welcome to StuRa - Mitgliederdatenbank's documentation!

User Documentation

Working with Members (Mitglieder)

Create a Member

Pre-conditions:

- login as Admin

To manage Members of the StuRa-Mitgliederdatenbank you first have to navigate to the Member section of the application.



Now you have to click on the Button "+ Hinzufügen"

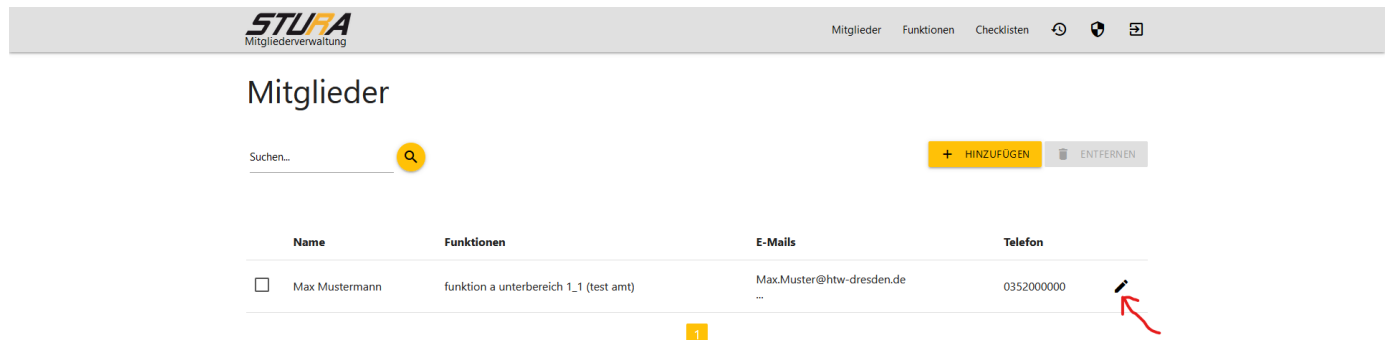
With that example form you see above you can fill in personal data for each member of the StuRa. You can add multiple functions the Member belongs to and the legislative period for that function. You can also add multiple E-Mails for the User. You can save the new Member by clicking on the Button "Speichern". You will then be redirected to the Member-View and your new Member should appear in the table.

Alter a Member

Pre-conditions:

- login as Admin

If you are in the Member section you can simply modify a Member by clicking on the pen on the right in the row of a Member.



Now you can modify all data just like if you would create a Member. To save your changes click the Button "Speichern".


Delete a Member

Pre-conditions:

- login as Admin

If you made a mistake or a Member is leaving the StuRa you can delete the Member in a few steps.

1. Check the Members you want to delete
2. Click on the Button "Entfernen"
3. Check the Dialog and accept it
4. Check success prompt



Mitgliederdatenbank

Mitglieder

Funktionen

Checklisten

Mitglieder

Suchen...

+ HINZUFÜGEN


ENTFERNEN

	Name	Funktionen	E-Mails	Telefon	
1	<input checked="" type="checkbox"/> Max Mustermann	funktion a unterbereich 1_1 (test amt)	Max.Muster@htw-dresden.de	0352000000	

2

© 2020 StuRa HTW Dresden

Datenschutz/Impressum



Mitgliederdatenbank

Mitglieder

Funktionen

Checklisten

Mitglieder

Suchen...

+ HINZUFÜGEN

ENTFERNEN

Mitglieder löschen

Sollen folgendes Mitglied wirklich endgültig gelöscht werden?

Max Mustermann

JA

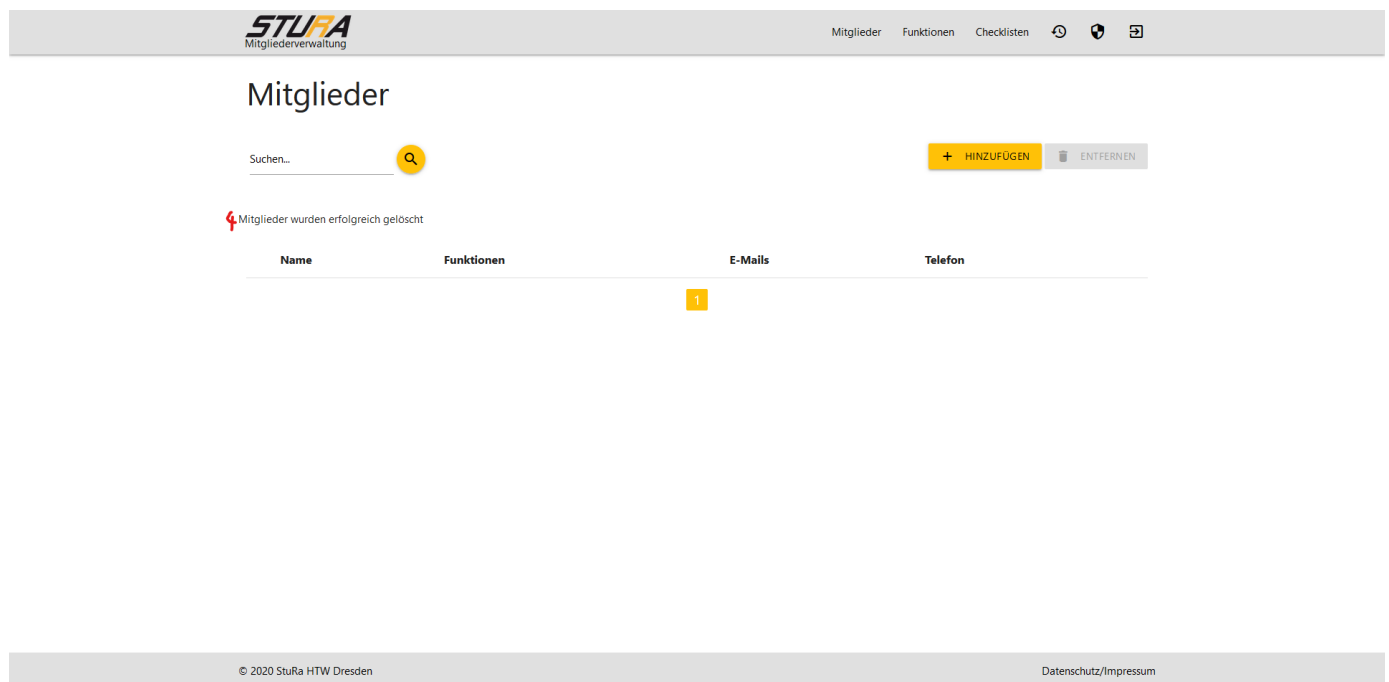
NEIN

	Name	Funktionen	E-Mails	Telefon	
3	<input checked="" type="checkbox"/> Max Mustermann	funktion a unterbereich 1_1 (test amt)	Max.Muster@htw-dresden.de	0352000000	

4

© 2020 StuRa HTW Dresden

Datenschutz/Impressum



Now you successfully deleted the Member/s.

Working with Organizational units (Organisationseinheiten)

Add a new Organizational unit

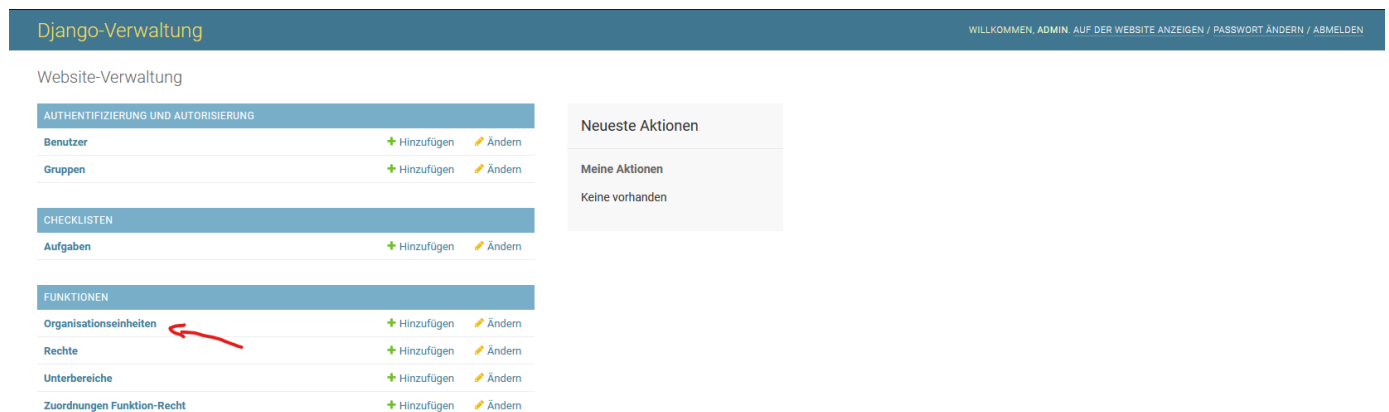
Pre-conditions:

- login as Admin

First step is to get into the admin section of the application.



Then we have to move to the correct section to manage our Organizational units.



Now to add a new Organizational unit we need to click on the Button “Organisationseinheit hinzufügen”.

Django-Verwaltung WILLKOMMEN, ADMIN AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start > Funktionen > Organisationseinheiten > test amt

Organisationseinheit ändern

GESCHICHTE

Bezeichnung:

Funktionen ohne unterbereich count:

FUNKTIONEN			
BEZEICHNUNG	WORKLOAD	MAX MEMBERS	LÖSCHEN?
+ Funktion hinzufügen			

UNTERBEREICHE	
BEZEICHNUNG	LÖSCHEN?
+ Unterbereich hinzufügen	

You can fill in the appearing form and add on new functions and/or subsections of this Organizational unit. At last you need to Save everything with the Button “Sichern” on the lower right of the form.

Django-Verwaltung WILLKOMMEN, ADMIN AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start > Funktionen > Organisationseinheiten

✓ Organisationseinheit „test amt“ wurde erfolgreich hinzugefügt.

Organisationseinheit zur Änderung auswählen

ORGANISATIONSEINHEIT HINZUFÜGEN +

Aktion: 0 von 1 ausgewählt

☐ ORGANISATIONSEINHEIT

☐ test amt

1 Organisationseinheit

If you are Prompted “Organisationseinheit „<Your Organisational unit>“ wurde erfolgreich hinzugefügt.” then everything succeeded and you are finished.

Alter an Organizational unit

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application and then we have to move to the correct section to manage our Organizational units.

Django-Verwaltung WILLKOMMEN, ADMIN AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Website-Verwaltung

AUTHENTIFIZIERUNG UND AUTORISIERUNG	
Benutzer	+ Hinzufügen Ändern
Gruppen	+ Hinzufügen Ändern

CHECKLISTEN	
Aufgaben	+ Hinzufügen Ändern

FUNKTIONEN	
Organisationseinheiten	+ Hinzufügen Ändern
Rechte	+ Hinzufügen Ändern
Unterbereiche	+ Hinzufügen Ändern
Zuordnungen Funktion-Recht	+ Hinzufügen Ändern

Neueste Aktionen

Meine Aktionen

Keine vorhanden

Now we have to click on the Organizational unit we want to alter, in the following form you can modify everything you want corresponding to the Organizational unit.

Django-Verwaltung

WILLKOMMEN, ADMIN. AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start > Funktionen > Organisationseinheiten > test amt

Organisationseinheit ändern

Bezeichnung:

test amt

Funktionen ohne unterbereich count:

0

FUNKTIONEN

BEZEICHNUNG	WORKLOAD	MAX MEMBERS	LÖSCHEN?
+ Funktion hinzufügen			

UNTERBEREICHE

BEZEICHNUNG	LÖSCHEN?
+ Unterbereich hinzufügen	

Löschen

Sichern und neu hinzufügen

Sichern und weiter bearbeiten

SICHERN

To confirm your modifications you have to click on the Button “sichern”. If you are Prompted “Organisationseinheit „<Your Organisational unit>“ wurde erfolgreich geändert.” then everything succeded and you are finished.

Delete an Organizational unit

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application and then we have to move to the correct section to manage our Organizational units.

Django-Verwaltung

WILLKOMMEN, ADMIN. AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Website-Verwaltung

AUTHENTIFIZIERUNG UND AUTORISIERUNG

Benutzer	+ Hinzufügen	Ändern
Gruppen	+ Hinzufügen	Ändern

CHECKLISTEN

Aufgaben	+ Hinzufügen	Ändern
----------	--------------	--------

FUNKTIONEN

Organisationseinheiten	+ Hinzufügen	Ändern
Rechte	+ Hinzufügen	Ändern
Unterbereiche	+ Hinzufügen	Ändern
Zuordnungen Funktion-Recht	+ Hinzufügen	Ändern

Neueste Aktionen

Meine Aktionen

Keine vorhanden

Now we have to check the Checkbox of the Organizational unit we want to delete. After this we need to select in Aktion “Ausgewählte Organisationseinheiten löschen” and click on the Button “Ausführen”.

Django-Verwaltung

Start > Funktionen > Organisationseinheiten

Organisationseinheit zur Änderung auswählen

Aktion:

Ausführen

0 von 1 ausgewählt

☐ ORG

Ausgewählte Organisationseinheiten löschen

☐ test amt

1 Organisationseinheit

To confirm our delete command you have to click on the Button "Ja, ich bin sicher". If you are prompted "Erfolgreich <Number of Organizational units> Organisationseinheit gelöscht." everything succeeded.

Working with Subsections (Unterbereiche)

Create a new Subsection

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application. Now we have 2 Options to create a new Subsection of an Organizational unit.

The screenshot shows the Django-Verwaltung admin interface. The top navigation bar includes 'Django-Verwaltung' and links for 'WILLKOMMEN, ADMIN', 'AUF DER WEBSITE ANZEIGEN', 'PASSWORT ÄNDERN', and 'ABMELDEN'. The left sidebar has 'Website-Verwaltung' and 'FUNKTIONEN'. Under 'FUNKTIONEN', 'Organisationseinheiten' is marked with a red arrow and '1.', and 'Unterbereiche' is marked with a red arrow and '2.'. The main content area shows 'Neueste Aktionen' and 'Meine Aktionen'.

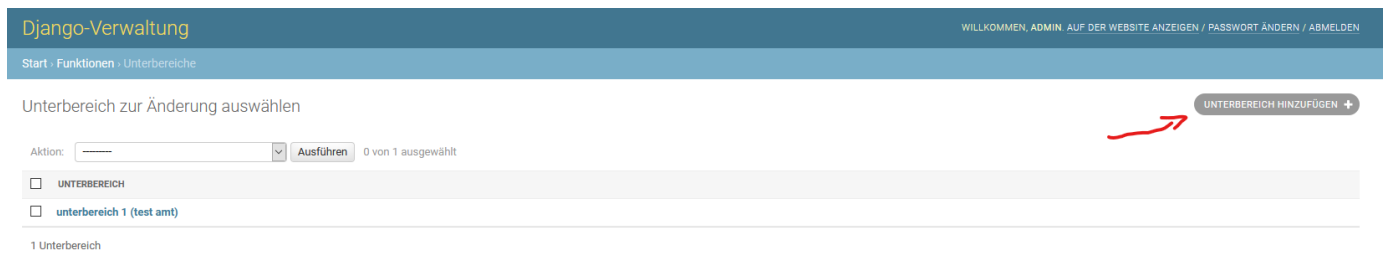
First Option via Organizational units

Navigate and click on the Organizational unit the Subsection belongs to. Now we can start to add a new Subsection, first we click on the Button "Unterbereich hinzufügen". Second we need to name the new Subsection and save our changes with the Button "Sichern und weiter bearbeiten". If you are prompted with a success message everything succeeded.

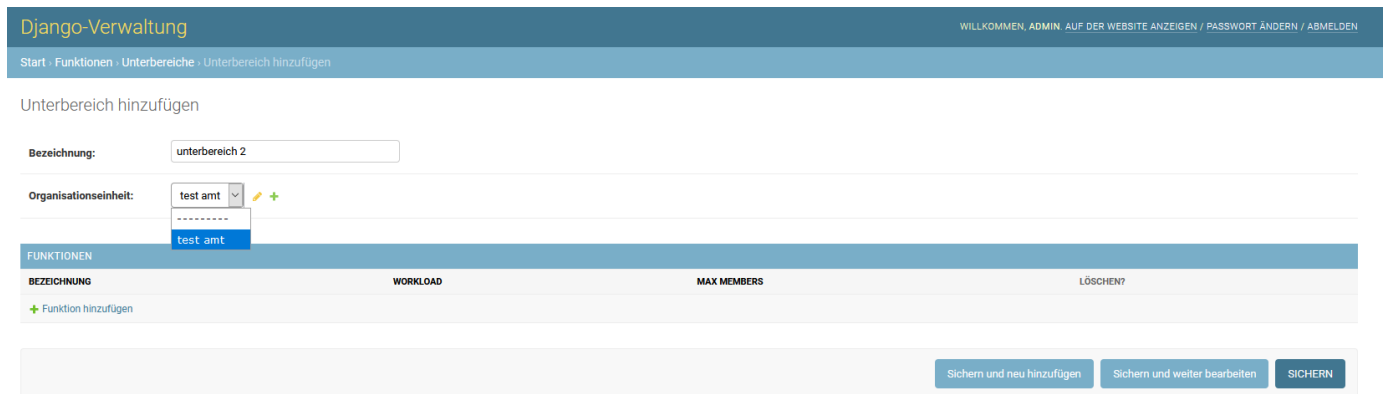
The screenshot shows the 'Organisationseinheit ändern' form in the Django-Verwaltung admin interface. The form has fields for 'Bezeichnung' (test amt) and 'Funktionen ohne unterbereich count' (0). Below the form is a table for 'FUNKTIONEN' and a table for 'UNTERBEREICHE'. The 'UNTERBEREICHE' table has a row for 'unterbereich 1 (test amt)' with a red arrow and '2.' next to the name. Below the table is a red button 'Löschen' and three buttons: 'Sichern und neu hinzufügen', 'Sichern und weiter bearbeiten' (marked with a red arrow and '3.'), and 'SICHERN'.

Second Option via Subsections

Navigate to the Subsection section and then click on the Button "Unterbereich hinzufügen".



Now we can give our new Subsection a name and choose the Organizational unit the Subsection belongs to.



If you are prompted with a success message everything succeeded.

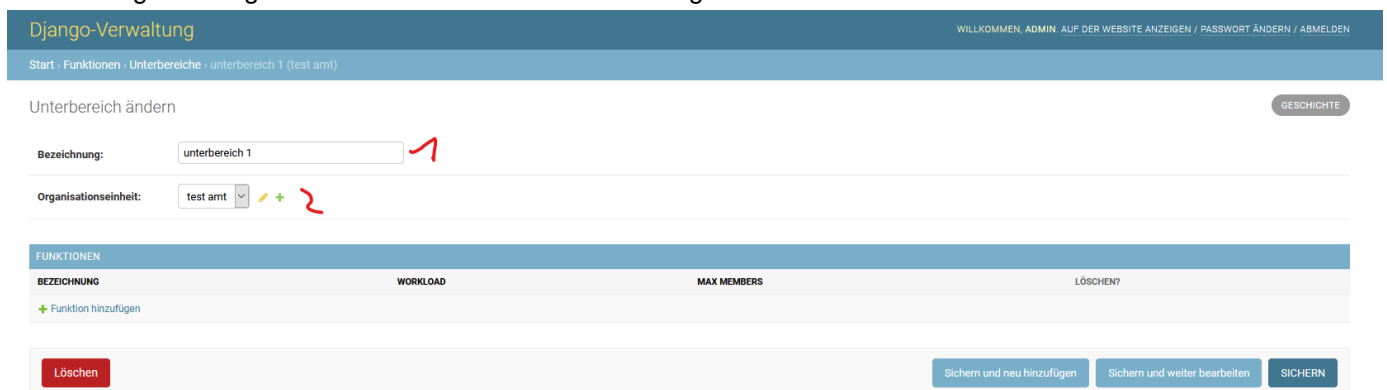
Alter a Subsection

Pre-conditions:

- login as Admin

To Alter a Subsection you can navigate to the Subsection section. There you can choose the Subsection you want to modify. Now you can:

1. Change the name of the subsection
2. Change the Organizational unit the subsection belongs to



To save your modifications click the Button “Sichern und weiter bearbeiten” and don’t forget to check the success message.

Delete a Subsection

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application. Navigate and click on the Organizational unit the Subsection you want to delete belongs to. Now we can check the subsections we want to delete from the organizational unit. At last we can save the modifications with the Button "Sichern und weiter bearbeiten".

Django-Verwaltung WILLKOMMEN, ADMIN. AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start > Funktionen > Organisationseinheiten > test amt

Organisationseinheit ändern GESCHICHTE

Bezeichnung:

Funktionen ohne unterbereich count:

FUNKTIONEN			
BEZEICHNUNG	WORKLOAD	MAX MEMBERS	LÖSCHEN?
+ Funktion hinzufügen			

UNTERBEREICHE	
BEZEICHNUNG	LÖSCHEN?
unterbereich 1 (test amt)	<input type="checkbox"/>
unterbereich 2 (test amt)	<input checked="" type="checkbox"/> 1.
+ Unterbereich hinzufügen	

2.

Check the success message to ensure that the Subsection was deleted successfully.

Django-Verwaltung WILLKOMMEN, ADMIN. AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start > Funktionen > Organisationseinheiten > test amt

✓ Organisationseinheit „test amt“ wurde erfolgreich geändert und kann unten erneut geändert werden. 3

Organisationseinheit ändern GESCHICHTE

Bezeichnung:

Funktionen ohne unterbereich count:

FUNKTIONEN			
BEZEICHNUNG	WORKLOAD	MAX MEMBERS	LÖSCHEN?
+ Funktion hinzufügen			

UNTERBEREICHE	
BEZEICHNUNG	LÖSCHEN?
unterbereich 1 (test amt)	<input type="checkbox"/>
+ Unterbereich hinzufügen	

Working with Functions (Funktionen)

Add a new Function

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application. Now we have 2 different Options to create a new Function.

Create a Function that belongs directly to an Organizational unit

To create a Function of a Organizational unit you first have to navigate to the Organizational unit you want to add the Function to. Then follow these steps to create a new Function:

1. Click on the Button "Funktion hinzufügen"
2. Name the new Function
3. Set a Workload for the Function
4. Set the maximum of allowed Members in the Function
5. Save your changes with the Button "Sichern und weiter bearbeiten"
6. Check the success message

Django-Verwaltung WILLKOMMEN, ADMIN. AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start · Funktionen · Organisationseinheiten · test amt

Organisationseinheit „test amt“ wurde erfolgreich geändert und kann unten erneut geändert werden. **6**

Organisationseinheit ändern GESCHICHTE

Bezeichnung:

Funktionen ohne unterbereich count:

FUNKTIONEN			
BEZEICHNUNG	WORKLOAD	MAX MEMBERS	LÖSCHEN?
<input type="text" value="funktion 1 (test amt)"/> <input type="text" value="funktion 1"/> 2	<input type="text" value="5"/> 3	<input type="text" value="2"/> 4	<input type="checkbox"/>
+ Funktion hinzufügen 1			

UNTERBEREICHE	
BEZEICHNUNG	LÖSCHEN?
<input type="text" value="unterbereich 1_1 (test amt)"/> <input type="text" value="unterbereich 1_1"/>	<input type="checkbox"/>
+ Unterbereich hinzufügen	

5

Create a Function that belongs to a Subsection of an Organizational unit

To create a Function of a Subsection you first have to navigate to the Subsection you want to add the Function to. Then follow these steps to create a new Function:

You can follow the same steps like in `createFunctionOrganizationalUnit`.

Alter a Function

Pre-conditions:

- login as Admin

To alter a Function you first need to know which Organizational unit or Subsection the Function belongs to. If you found it you can navigate to it through the admin section of the application. Now you can:

1. Change the name of the Function
2. Change the workload of a Function or
3. Change the maximum ammount of Members in the Function

To save your modifications click the button "Sichern und weiter bearbeiten" and don't forget to check the success message.

Delete a Function

Pre-conditions:

- login as Admin

To alter a Function you first need to know which Organizational unit or Subsection the Function belongs to. If you found it you can navigate to it through the admin section of the application. Now we can check the Functions we want to delete from the Organizational unit or Subsection. At last we can save the modifications with the Button "Sichern und weiter bearbeiten". Check the success message to see the changes have been made successfully

Working with Checklists (Checklisten)

Create a new General Task

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application.



Now you have to move to the Tasks section of the application.

The screenshot shows the 'Django-Verwaltung' interface. The top navigation bar includes 'Django-Verwaltung' and links: 'WILLKOMMEN, ADMIN', 'AUF DER WEBSITE ANZEIGEN', 'PASSWORT ÄNDERN', and 'ABMELDEN'. The main content area is titled 'Website-Verwaltung'. It contains three sections: 'AUTHENTIFIZIERUNG UND AUTORISIERUNG' with 'Benutzer' and 'Gruppen' (each with '+ Hinzufügen' and 'Ändern' buttons); 'CHECKLISTEN' with 'Aufgaben' (highlighted with a red arrow and '+ Hinzufügen' and 'Ändern' buttons); and 'FUNKTIONEN' with 'Organisationseinheiten', 'Rechte', 'Unterbereiche', and 'Zuordnungen Funktion-Recht' (each with '+ Hinzufügen' and 'Ändern' buttons). On the right, there is a 'Neueste Aktionen' sidebar showing a list of recent actions like 'user Benutzer', 'test_aufgabe Aufgabe', 'test Recht', 'unterbereich 1.1 (test amt) Unterbereich', and several 'test amt Organisationseinheit' entries.

If you now click on the Button “Aufgabe hinzufügen” you can add the new Task. You have to name the new Task and save it with the Button “Sichern”

The screenshot shows the 'Aufgabe hinzufügen' form. The top navigation bar is the same as the previous screenshot. Below it, a breadcrumb trail reads 'Start > Checklisten > Aufgaben > Aufgabe hinzufügen'. The form has a label 'Bezeichnung:' followed by a text input field containing 'test_aufgabe_2' (marked with a red '1'). At the bottom right, there are three buttons: 'Sichern und neu hinzufügen', 'Sichern und weiter bearbeiten', and 'SICHERN' (marked with a red '2').

At last check the success message.

Create Right for a Function

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application.

The screenshot shows the application footer. On the left is the 'STURA' logo with 'Mitgliederverwaltung' underneath. On the right are navigation links: 'Mitglieder', 'Funktionen', 'Checklisten', and a set of icons including a magnifying glass, a plus sign, and a document icon.

Now you have to move to the rights section of the application. You can simply create a new right like in “Create a new General Task”. If you created a right you now need to link it with a Function. To do it, first you need to navigate in the admin section to the Function-Right Subsection.

Django-Verwaltung

WILLKOMMEN, ADMIN / AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Website-Verwaltung

AUTHENTIFIZIERUNG UND AUTORISIERUNG

- Benutzer + Hinzufügen ✎ Ändern
- Gruppen + Hinzufügen ✎ Ändern

CHECKLISTEN

- Aufgaben + Hinzufügen ✎ Ändern

FUNKTIONEN

- Organisationseinheiten + Hinzufügen ✎ Ändern
- Rechte + Hinzufügen ✎ Ändern
- Unterbereiche + Hinzufügen ✎ Ändern
- Zuordnungen Funktion-Recht** + Hinzufügen ✎ Ändern

Neueste Aktionen

Meine Aktionen

- + test_recht Rechte
- + test_aufgabe_2 Aufgabe
- ✎ user Benutzer
- + user Benutzer
- + test_aufgabe Aufgabe
- ✎ test Rechte
- + test Rechte
- ✎ unterbereich_1_1 (test amt) Unterbereich
- ✎ test amt Organisationseinheit
- ✎ test amt Organisationseinheit

If you are there you can add a new link with the Button “Zuordnung Funktion-Recht hinzufügen”.

Django-Verwaltung

WILLKOMMEN, ADMIN / AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start : Funktionen : Zuordnungen Funktion-Recht : Zuordnung Funktion-Recht hinzufügen

Zuordnung Funktion-Recht hinzufügen

Funktion: funktion a unterbereich 1_1 (test amt) 1

Recht: test_recht 2

Sichern und neu hinzufügen Sichern und weiter bearbeiten SICHERN

There you have to select a Function and a right you want to link with and save by clicking the button “Sichern”.

Create a new Checklist

Pre-conditions:

- login as Admin
- Rights for a Function
- __optional:__ General Tasks have been created through the admin panel

To manage Checklisten you have to go to the Checklisten-View of the Application.

STURA
Mitgliederverwaltung

Mitglieder Funktionen **Checklisten** ⌚ 🛡️ 📄

If you are there click on the big panel with the label “+” to add a new checklist.

The screenshot shows the 'Checkliste erstellen' (Create Checklist) dialog box. It has two dropdown menus: 'Mitglied' (Member) with the placeholder 'Bitte auswählen' (Please select) and 'Funktion' (Function) with the placeholder 'Keine' (None). There is a checked checkbox labeled 'Allgemeine Aufgaben zur Liste hinzufügen' (Add general tasks to the list). At the bottom, there are two buttons: 'ERSTELLEN' (Create) in orange and 'ABBRECHEN' (Cancel) in gray. The background shows a blurred view of the 'Checklisten' (Checklists) page.

You have to select the Member which belongs to the Checklist and optionally a Function the Member is associated with. The checkbox General Tasks can only be deselected if you choose a function. Create a new Checklist by clicking on the Button "Erstellen".

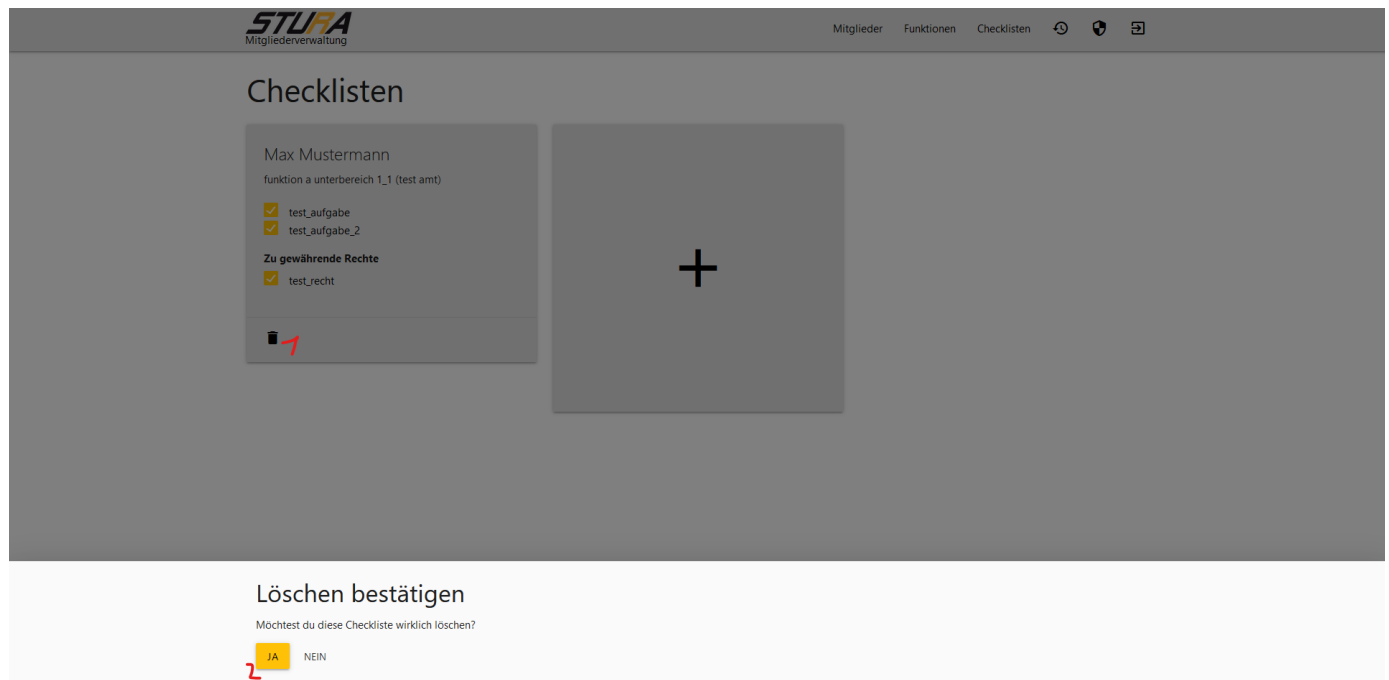
The screenshot shows the 'Checklisten' (Checklists) page. On the left, there is a checklist card for 'Max Mustermann' (function a unterbereich 1_1 (test amt)). It lists two tasks: 'test_aufgabe' and 'test_aufgabe_2', both with unchecked checkboxes. Below the tasks, under the heading 'Zu gewährende Rechte' (Rights to be granted), there is one task: 'test_recht' with an unchecked checkbox. At the bottom left of the card is a trash can icon. To the right of the card is a large gray square with a black plus sign, indicating a button to add a new checklist.

Delete a Checklist

Pre-conditions:

- login as Admin

To delete a Checklist you need to click on the garbage can on the lower left of a checklist and accept the dialog.



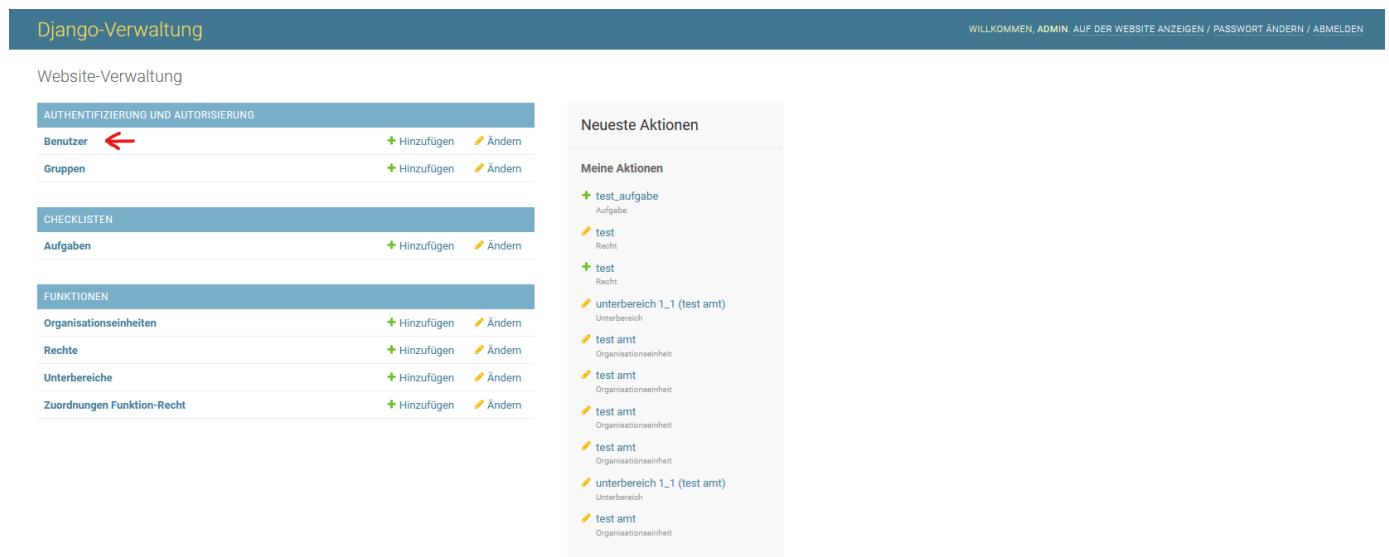
Working with Users and Admins

Create an User

Pre-conditions:

- login as Admin

First step is to get into the admin section of the application. Now you have to go to the User section of the admin page.



Now you see a overview of all Users of the Software. To add a new User you can click on the Button “Benutzer hinzufügen” on the upper right of the page. You have to fill in the form and accept your input with the Button “Sichern und weiter bearbeiten”

Welcome to StuRa - Mitgliederdatenbank's documentation!

Django-Verwaltung

WILLKOMMEN, ADMIN. AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start · Authentifizierung und Autorisierung · Benutzer · Benutzer hinzufügen

Benutzer hinzufügen

Bitte zuerst einen Benutzernamen und ein Passwort eingeben. Danach können weitere Optionen für den Benutzer geändert werden.

Benutzername:

Erforderlich, 150 Zeichen oder weniger. Nur Buchstaben, Ziffern und @/./+/-/_.

Passwort:

Das Passwort darf nicht zu ähnlich zu anderen persönlichen Informationen sein.
Das Passwort muss mindestens 8 Zeichen enthalten.
Das Passwort darf nicht allgemein üblich sein.
Das Passwort darf nicht komplett aus Ziffern bestehen.

Passwort bestätigen:

Bitte das selbe Passwort zur Bestätigung erneut eingeben.

Sichern und neu hinzufügen

Sichern und weiter bearbeiten

SICHERN

Now you can add some more personal information to the user account if you want and you can give the user some more rights for the Page. (More in the section create an admin)

Django-Verwaltung

WILLKOMMEN, ADMIN. AUF DER WEBSITE ANZEIGEN / PASSWORT ÄNDERN / ABMELDEN

Start · Authentifizierung und Autorisierung · Benutzer · user

✔ Benutzer „user“ wurde erfolgreich hinzugefügt. Es kann unten erneut geändert werden.

Benutzer ändern

Benutzername:

user

Erforderlich, 150 Zeichen oder weniger. Nur Buchstaben, Ziffern und @/./+/-/_.

Passwort:

Algorithmus: pbkdf2_sha256 Wiederholungen: 180000 Salt: nCXQCd***** Hash: +VxghL*****
Die Passwörter werden nicht im Klartext gespeichert und können daher nicht dargestellt, sondern nur mit [diesem Formular](#) geändert werden.

Persönliche Informationen

Vorname:

Nachname:

E-Mail-Adresse:

Berechtigungen

☒ Aktiv
Legt fest, ob dieser Benutzer aktiv ist. Kann deaktiviert werden, anstatt Benutzer zu löschen.

☐ Mitarbeiter-Status
Legt fest, ob sich der Benutzer an der Administrationsseite anmelden kann.

☐ Administrator-Status
Legt fest, dass der Benutzer alle Berechtigungen hat, ohne diese einzeln zuweisen zu müssen.

Gruppen:

Verfügbare Gruppen

Ausgewählte Gruppen

Create a User (with Admin rights)

Pre-conditions:

- login as Admin
- successfully created User

If you successfully created a User with the section above you can Upgrade the User account to an Admin account. First you need to navigate to the User you want to give administrative privileges and then you have to simply check the Checkboxes “Mitarbeiter-Status” and “Administrator-Status”

Finally you need to save your changes with a click on the Button “Sichern und weiter bearbeiten”

Admin Documentation

Deployment

This section contains a step-by-step guidance how to deploy this software. It's based on a deployment of Django with Apache2 and mod_wsgi on a raspberry pi 4.

Sources:

- <https://docs.djangoproject.com/en/3.0/howto/deployment/>
- https://wiki.ubuntuusers.de/Apache_2.4/
- https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-ubuntu-16-04

Requirements:

- a web server
- basic knowledge of bash
- Python 3

Basic configuration:

Update your distro

```
sudo apt update && sudo apt upgrade
```

Install Dependencies for the deployment. (We use Python3)

```
sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3
```

You also need to install virtualenv, to separate Python from your system's python. Important is that you use python 3 because we use the Apache mod_wsgi for python 3.

```
sudo pip3 install virtualenv
```

Now we need to clone the Git-Repository and setup the virtualenv for Python. First you need to change to the directory that you want to clone this web application to. Then:

```
git clone https://github.com/mribgrgr/StuRa-Mitgliederdatenbank.git
cd StuRa-Mitgliederdatenbank/
```

Now create a virtual environment and activate it.

```
virtualenv venv
source venv/bin/activate
```

If you have successfully activated your virtual environment then your prompt should look something like this (venv) user@host:StuRa-Mitgliederdatenbank. The last step is to install the requirements.txt in the virtual environment.

```
pip install -r requirements.txt
```

Adjust Django Project Settings:

First we need to configure StuRa-Mitgliederdatenbank/bin/settings.py: We open the file first (based on the previous chapter)

```
nano bin/settings.py
```

For a productive environment set the debug output to false.

```
DEBUG = False
```

Here we need to register our server's public IP address or domain name. Replace "IP_or_DOMAIN" with your personal IP address or domain name.

```
ALLOWED_HOSTS = ["IP_or_DOMAIN"]
```

At the bottom of the settings.py we need to define a static directory for our static html files.

```
STATIC_ROOT = os.path.join(BASE_DIR, 'mystatic/')
```

Now we can close and save the file. After this you need to create the folder static in the directory StuRa-Mitgliederdatenbank with the command.

```
mkdir static
```

The last step is to do initial commands:

```
python ./manage.py makemigrations
python ./manage.py migrate
python ./manage.py collectstatic
python ./manage.py createsuperuser
```

An optional step that can be done is to fill in some functions that are common to the StuRa of the HTW-Dresden.

```
cd importscripts
python main.py
```

Now wait a little moment and then you can change to the parent directory.

```
cd ..
```

And deactivate the virtual environment:

```
deactivate
```

Configure Apache2:

To enable Apache2 as front end we need to configure WSGI pass. To achieve this we need to edit the default virtual host file:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```


We can keep everything that is present in this file and add our config above the last `</VirtualHost>` tag. What we first specify is the static directory and the path to the `wsgi.py`. (In this Example I have cloned the directory in `~/StuRa-Mitgliederdatenbank`) (pi is my username change it to yours)

/etc/apache2/sites-available/000-default.conf

```
<VirtualHost *:80>

. . .

Alias /static /home/pi/StuRa-Mitgliederdatenbank/mystatic
<Directory /home/pi/StuRa-Mitgliederdatenbank/mystatic>
    Require all granted
</Directory>

<Directory /home/pi/StuRa-Mitgliederdatenbank/bin>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>

</VirtualHost>
```

Now we add the recommended daemon mode to the WSGI process. To do it you need to append the following lines to the Apache config.

/etc/apache2/sites-available/000-default.conf

```
<VirtualHost *:80>

. . .

WSGIDaemonProcess StuRa-Mitgliederdatenbank python-home=/home/pi/StuRa-Mitgliederdatenbank
WSGIProcessGroup StuRa-Mitgliederdatenbank
WSGIScriptAlias / /home/pi/StuRa-Mitgliederdatenbank/bin/wsgi.py

</VirtualHost>
```

Solve some Permission Issues:

The first step is to change the permissions of the database, so that group owner can read and write. Then we need to transfer the ownership of some files to Apache2 group and user `www-data`.

```
chmod 664 ~/StuRa-Mitgliederdatenbank/db.sqlite3
sudo chown www-data:www-data ~/StuRa-Mitgliederdatenbank/db.sqlite3
sudo chown www-data:www-data ~/StuRa-Mitgliederdatenbank
```

If you got firewall issues, allow Apache to access the firewall example:

```
sudo ufw allow 'Apache Full'
```

Last but not least check the Apache files if everything is correct:

```
sudo apache2ctl configtest
```

If the output looks like `Syntax OK` you are done and can restart your apache2 service:

```
sudo systemctl restart apache2
```

Cronjobs

In the following lines there is an explanation how to create cronjobs to schedule tasks. These tasks help to cleanup the database of the application.

First make sure you have installed cron then you can add cronjobs with the command:

```
crontab -e
```

Now you can see a file like this:

At the bottom of the file you can add your personal cronjobs. To keep it simple our recommendation is to create a script with all commands you want (described in the commands section of the code documentation). You can easily schedule this script to run once a week with cronjob.

```
0 0 * * 0 bash /path/to/script
```

Update the Application

This section describes how to update the application from an existing deployment.

Update From GitHub with no changes in migrations:

First you need to get the ownership back from www-data. In my example my user is *pi* and the application is located in *pi*'s home directory.

```
sudo chown pi:pi ~/StuRa-Mitgliederdatenbank/db.sqlite3
sudo chown pi:pi ~/StuRa-Mitgliederdatenbank
```

Now we need to stash our changes we have done during our config.

```
git stash
```

We can now pull the latest version from the git Repository

```
git pull
```

To apply our configs back we need to get them back from the stash

```
git stash pop
```

At last you need to give the ownership back to www-data.

```
sudo chown www-data:www-data ~/StuRa-Mitgliederdatenbank/db.sqlite3
sudo chown www-data:www-data ~/StuRa-Mitgliederdatenbank
```

Developer Documentation

Introduction Development

This and the following sections deal with development specific topics.

Clone the Git-Repository, install the requirements and fire up the Webserver for development

Requirements:

- A installation of git
- A Installation of Python (we used 3.8.2)
- (optional) a installation of virtualenviroment
- A Webbrowser

First step is to pull the Git-Repository

```
git clone https://github.com/mribrgr/StuRa-Mitgliederdatenbank.git
```

Next we change the directory to the pulled Git-Repository with:

```
cd StuRa-Mitgliederdatenbank/
```

Now we need to install all python requirements (optional: install it in a virtualenviroment). You can use the requirements.txt, it contains all requirements we used.

```
pip install -r requirements.txt
```

The Basics are done, now we need to create a Database and apply all migrations to it. We use the makemigrations and the migrate command from the manage.py.

```
python3 ./manage.py makemigrations  
python3 ./manage.py migrate
```

If the Database was successfully created there must be a db.sqlite3 in the directory. As last step, before we can fire up the server, we need to create an admin account using the command superuser so that we can login into the application.

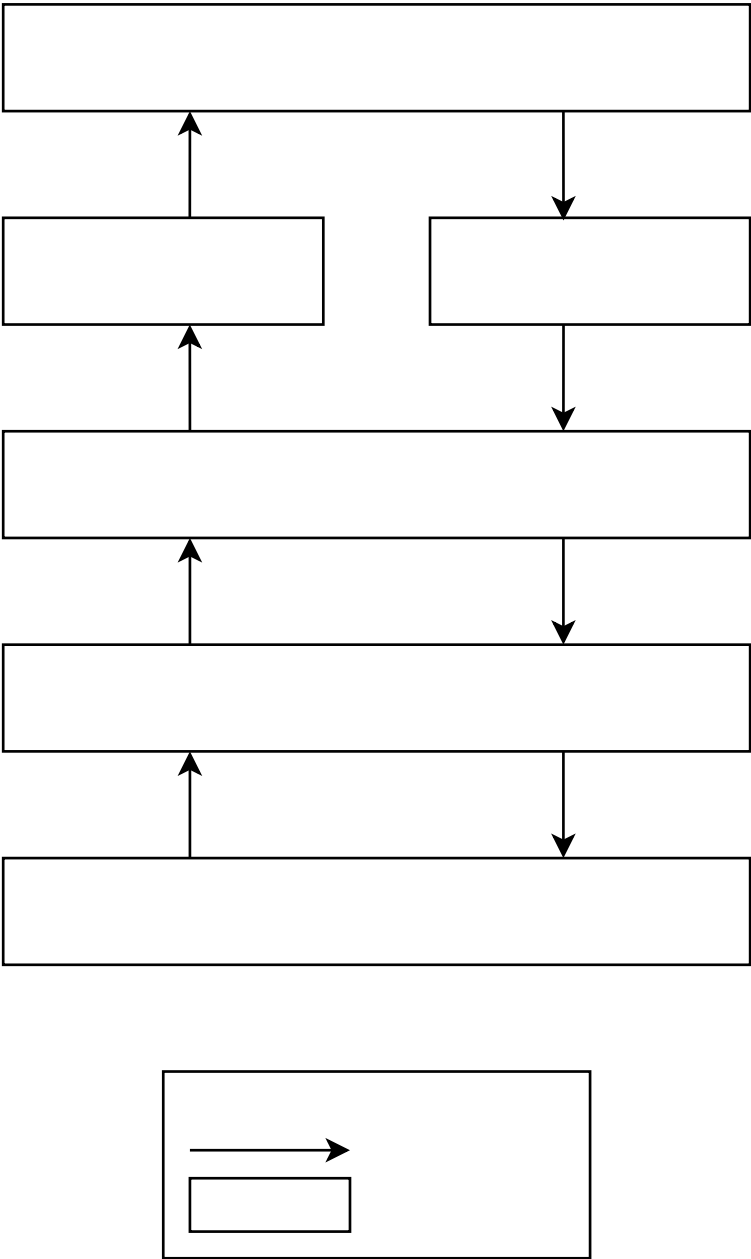
```
python3 ./manage.py createsuperuser
```

Now we can start the server by typing:

```
python3 ./manage.py runserver
```

System Architecture

The architecture of the Django Webframework is a Standard MVC architecture. Below is an example of the basic workflow of the Django Framework:

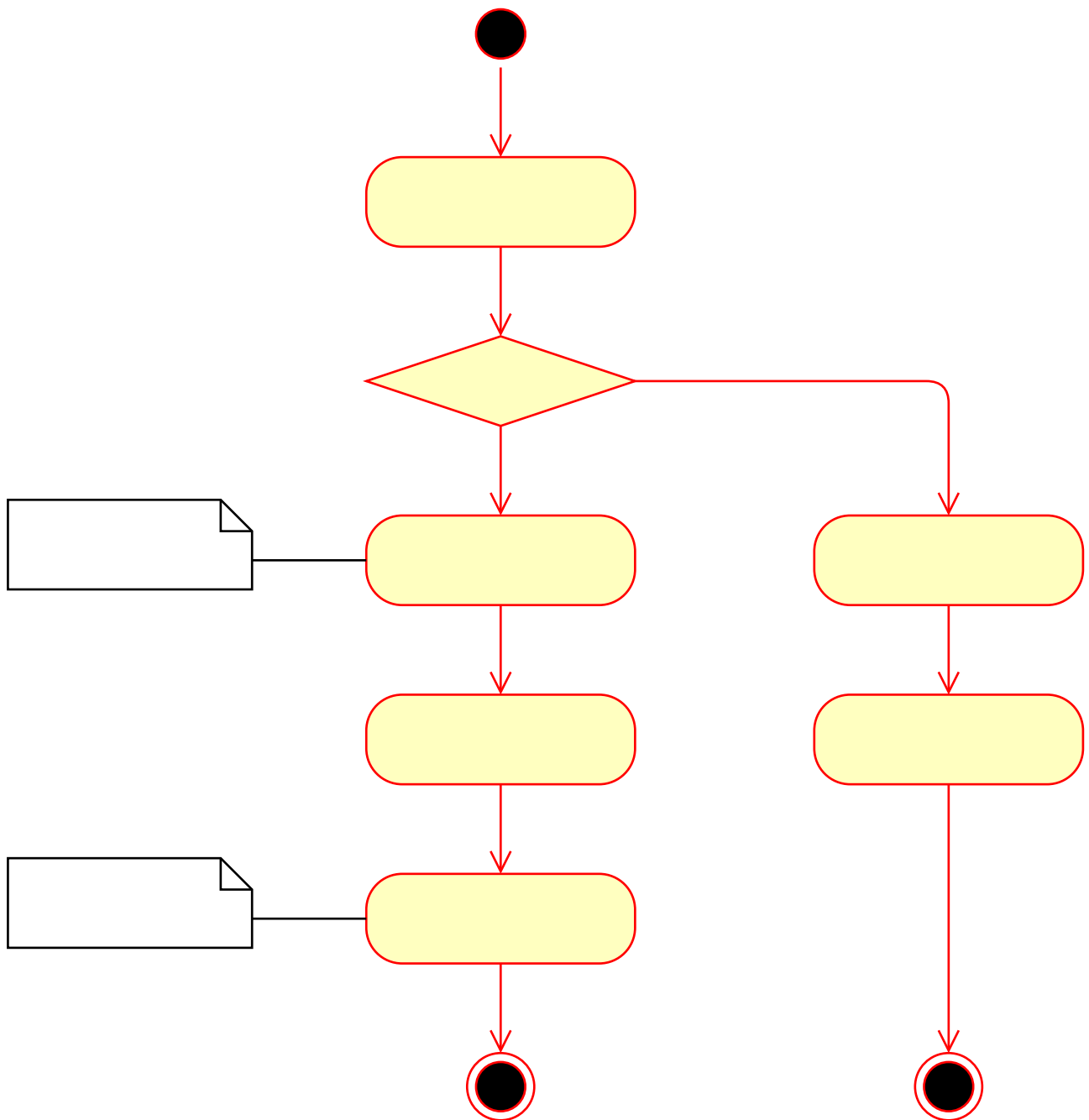


The layers of the Application look like this:

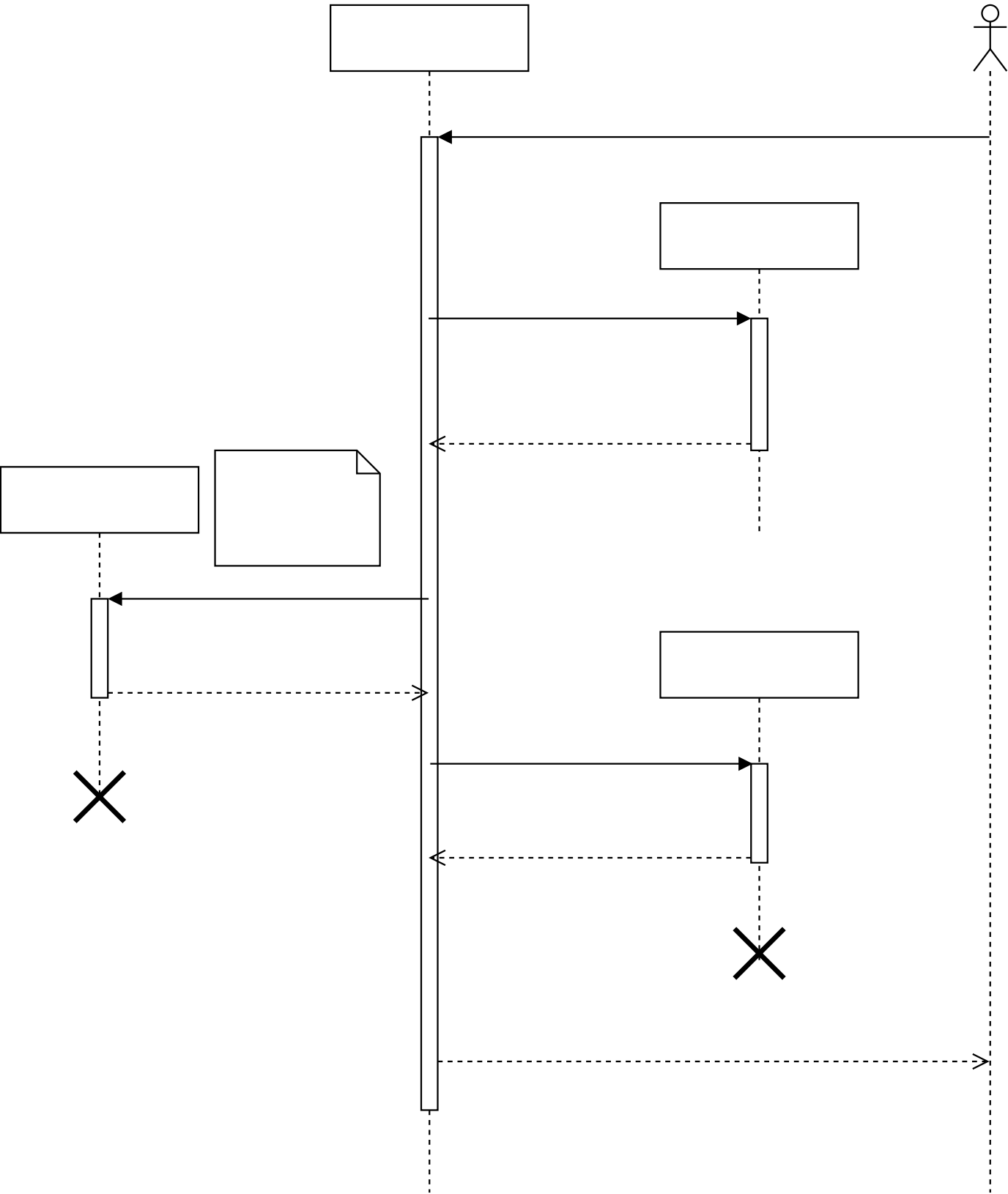
	Python

The V in MVC

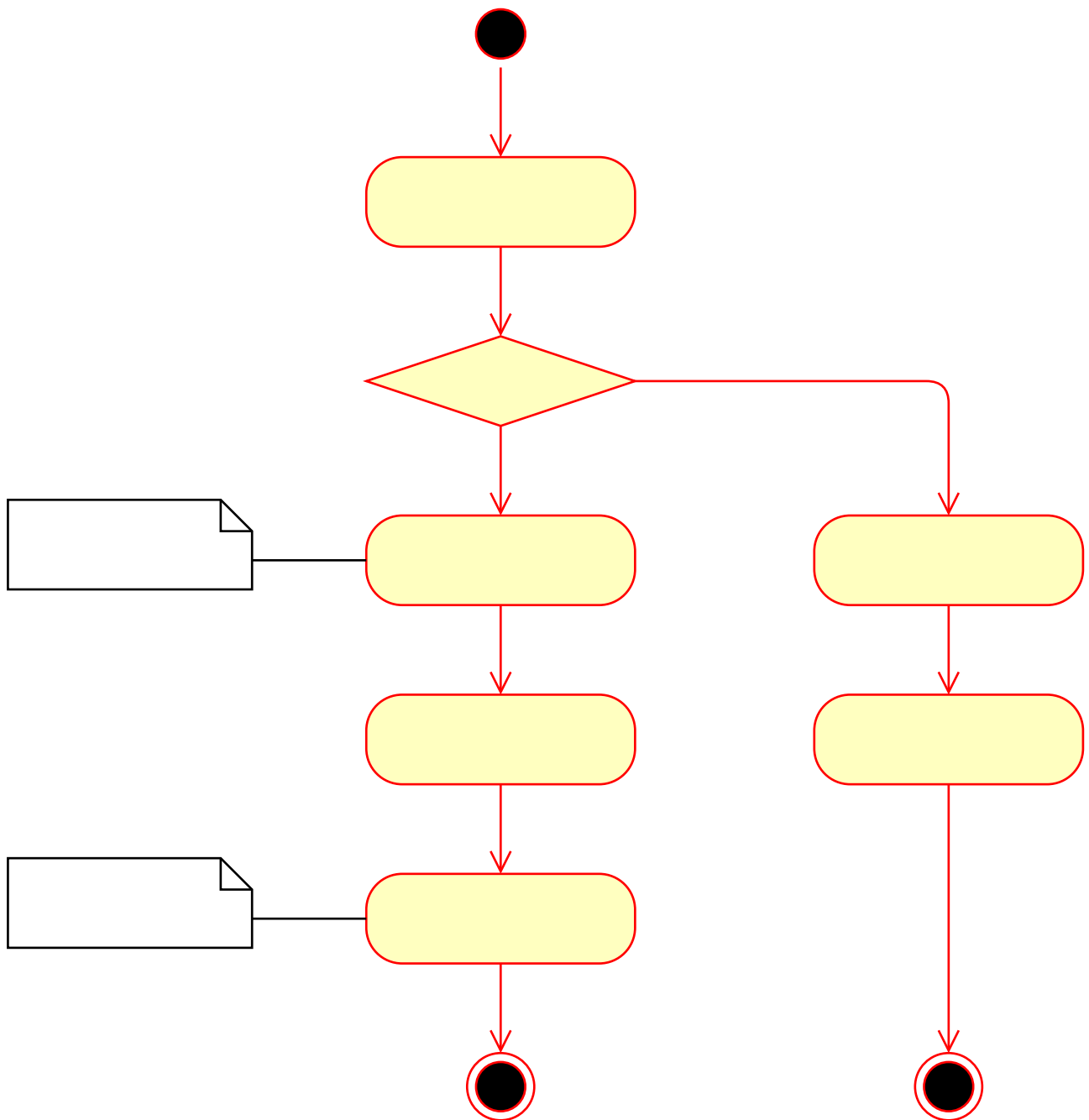
This section gives an overview over the function of the *views.py* in each module.
A generic Example:



The generic sequence diagram from the Example above:

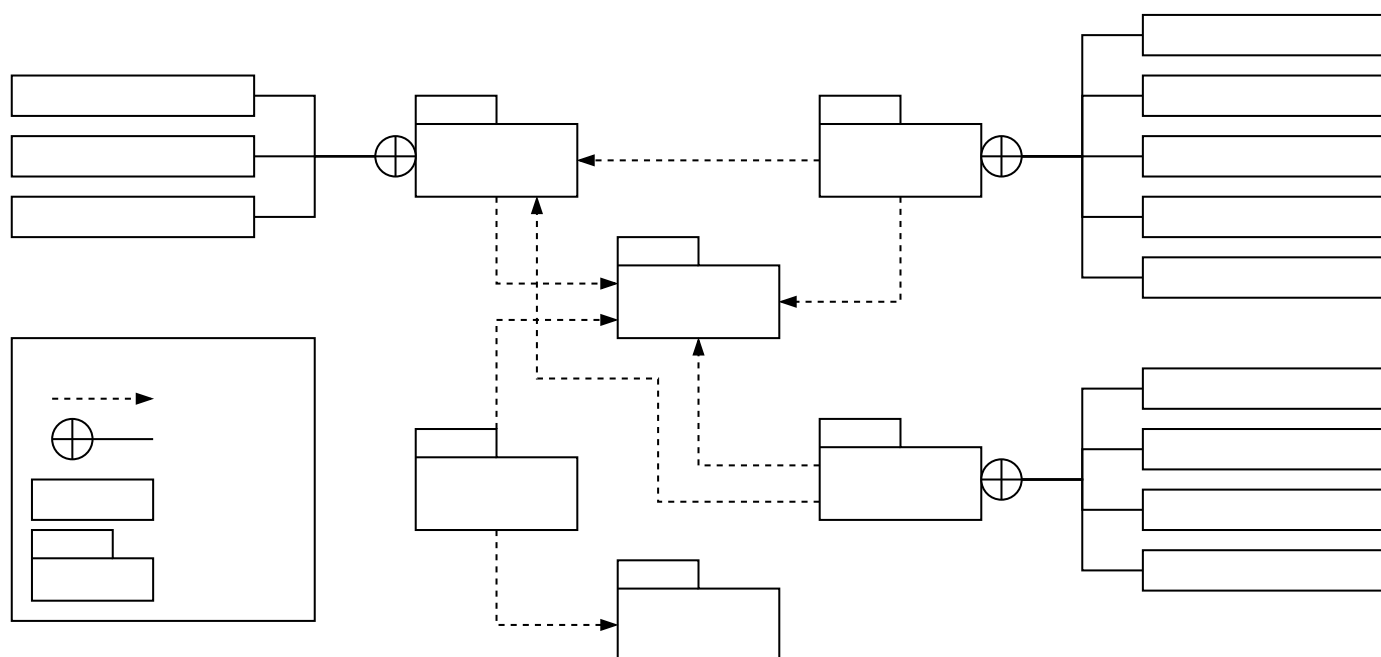


A example for the `mitglieder/views.py`:

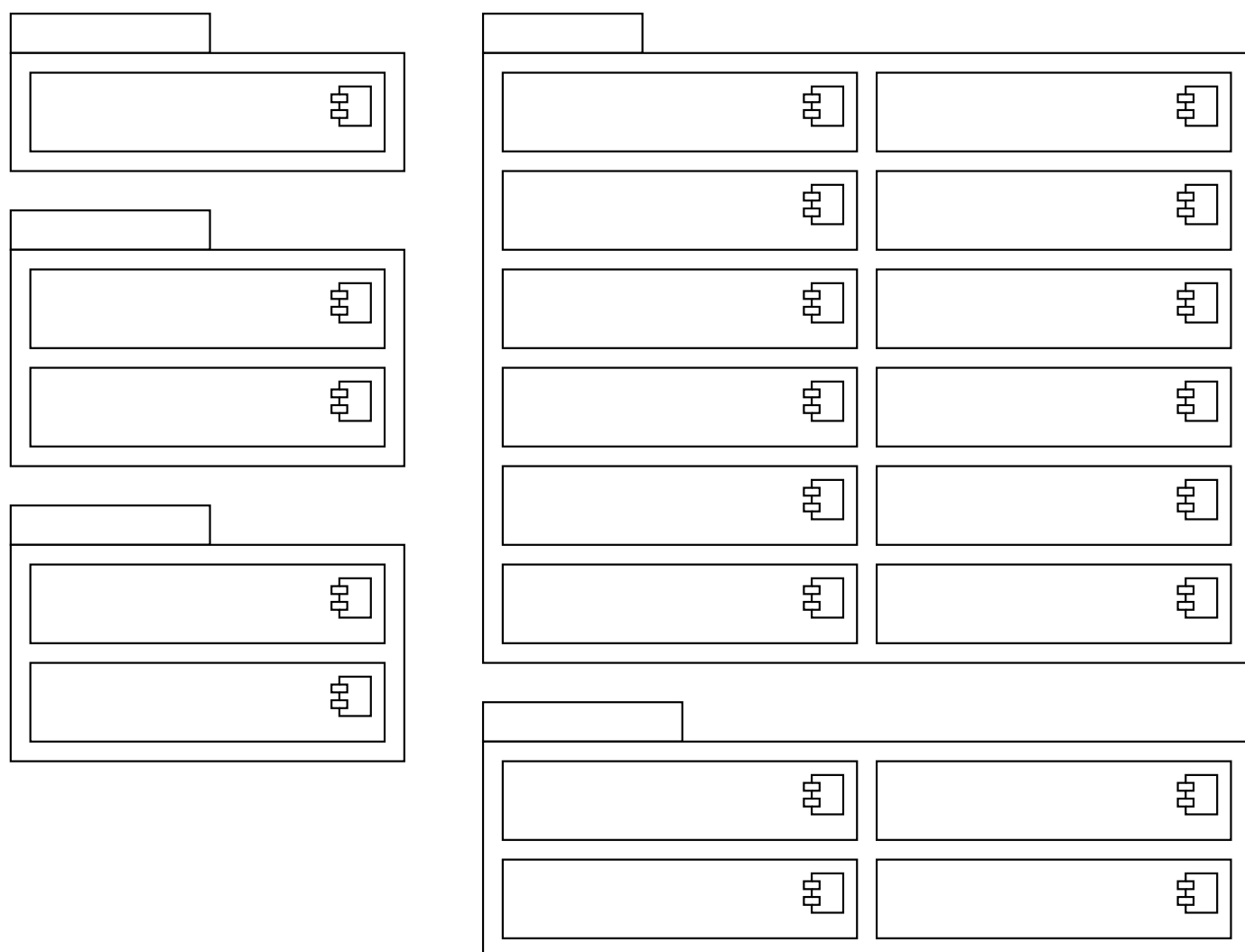


Components

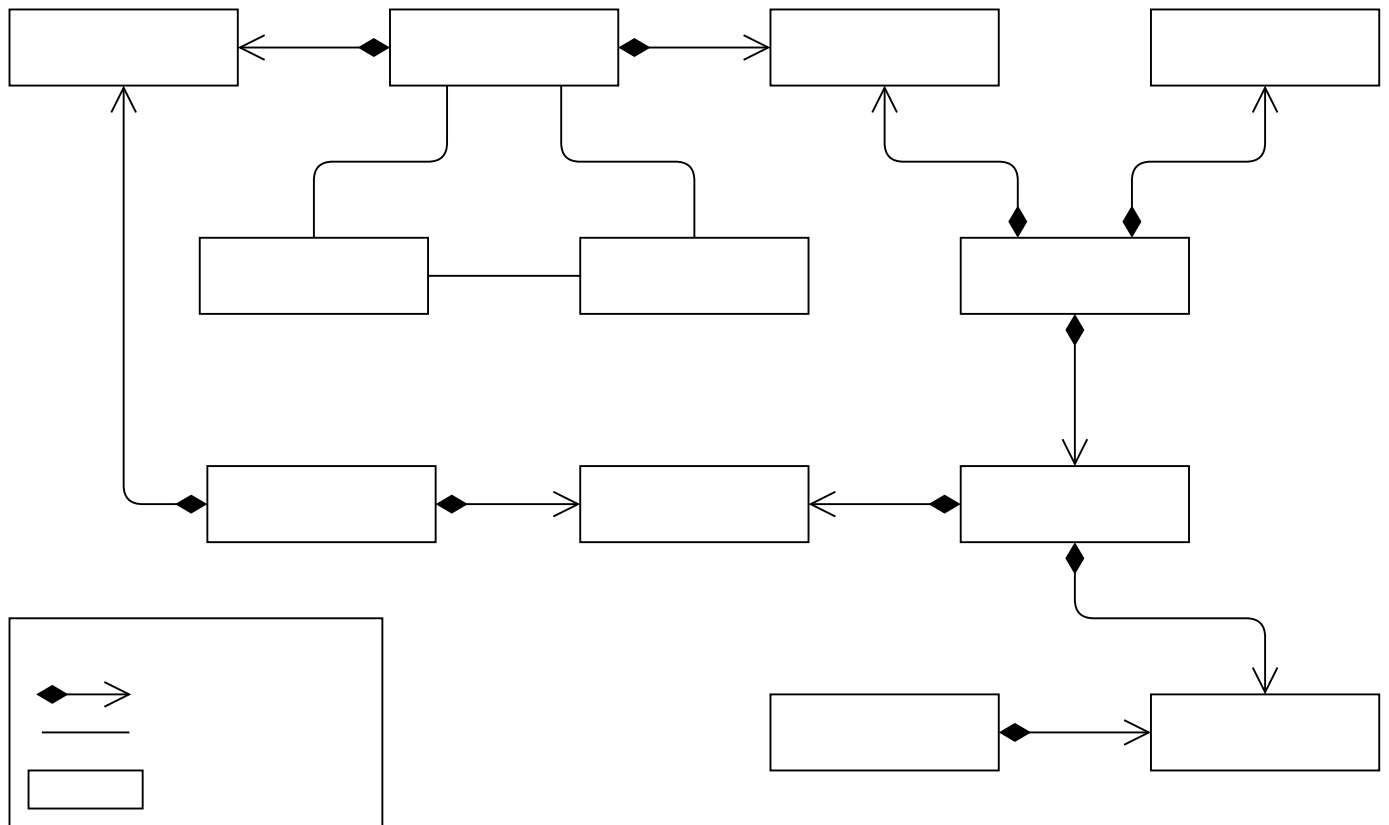
A brief overview over the Packages



A brief overview over the Components of a Package

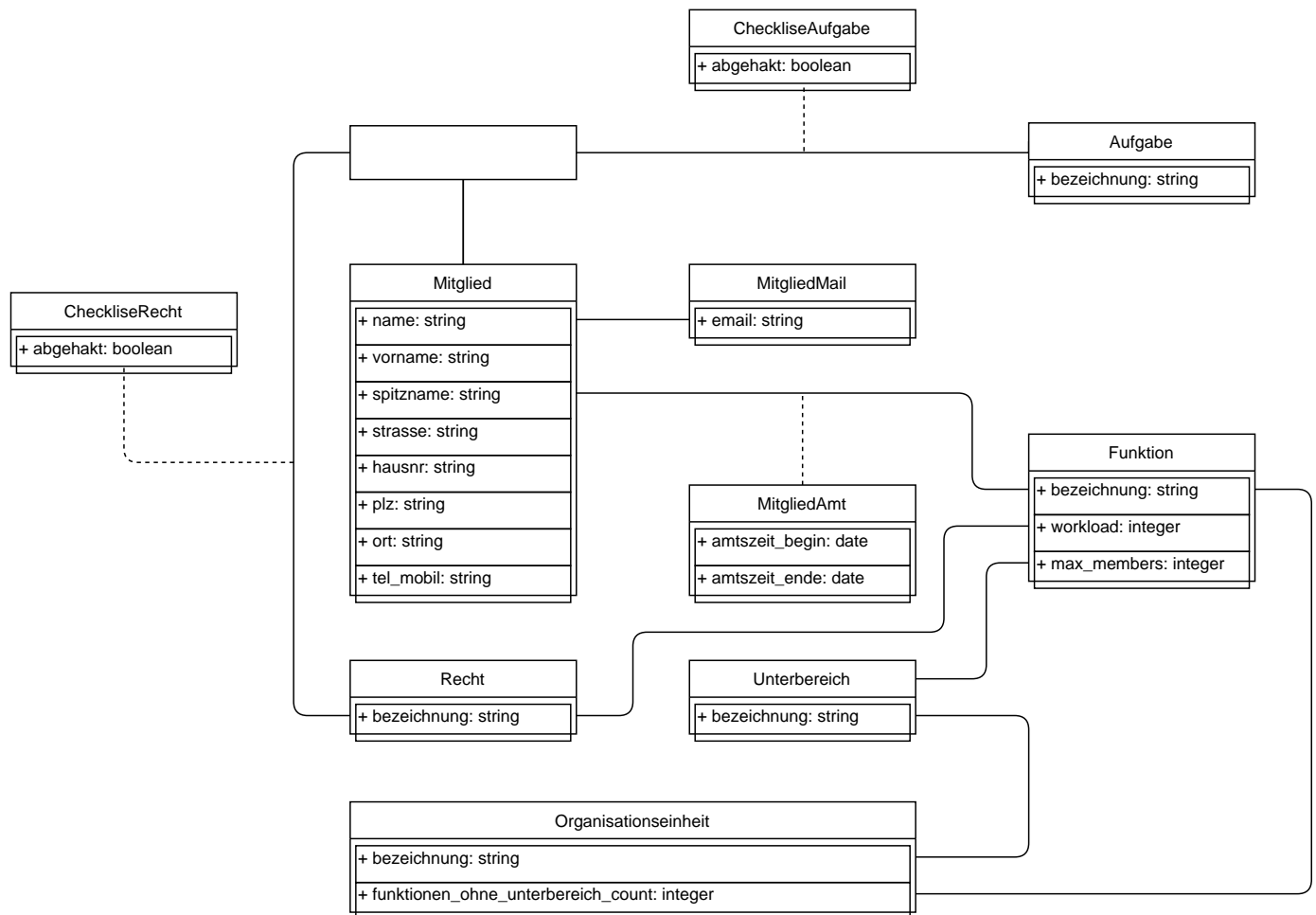


A brief overview over the connections between the Classes



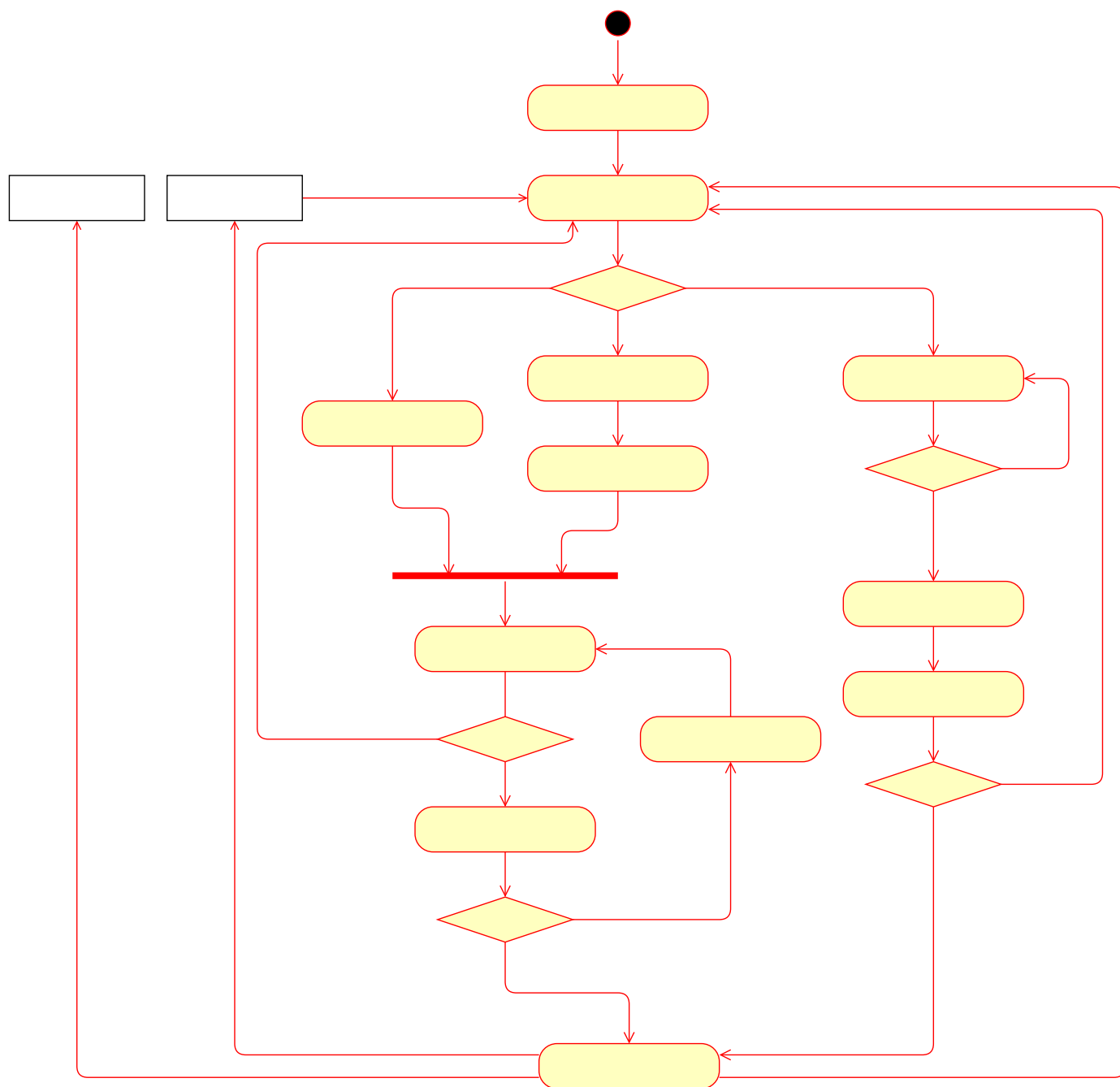
Database Structure

The Database is a SQLite3 Database, the Tables are specified in each Packages model.py and in a scheme it looks like:

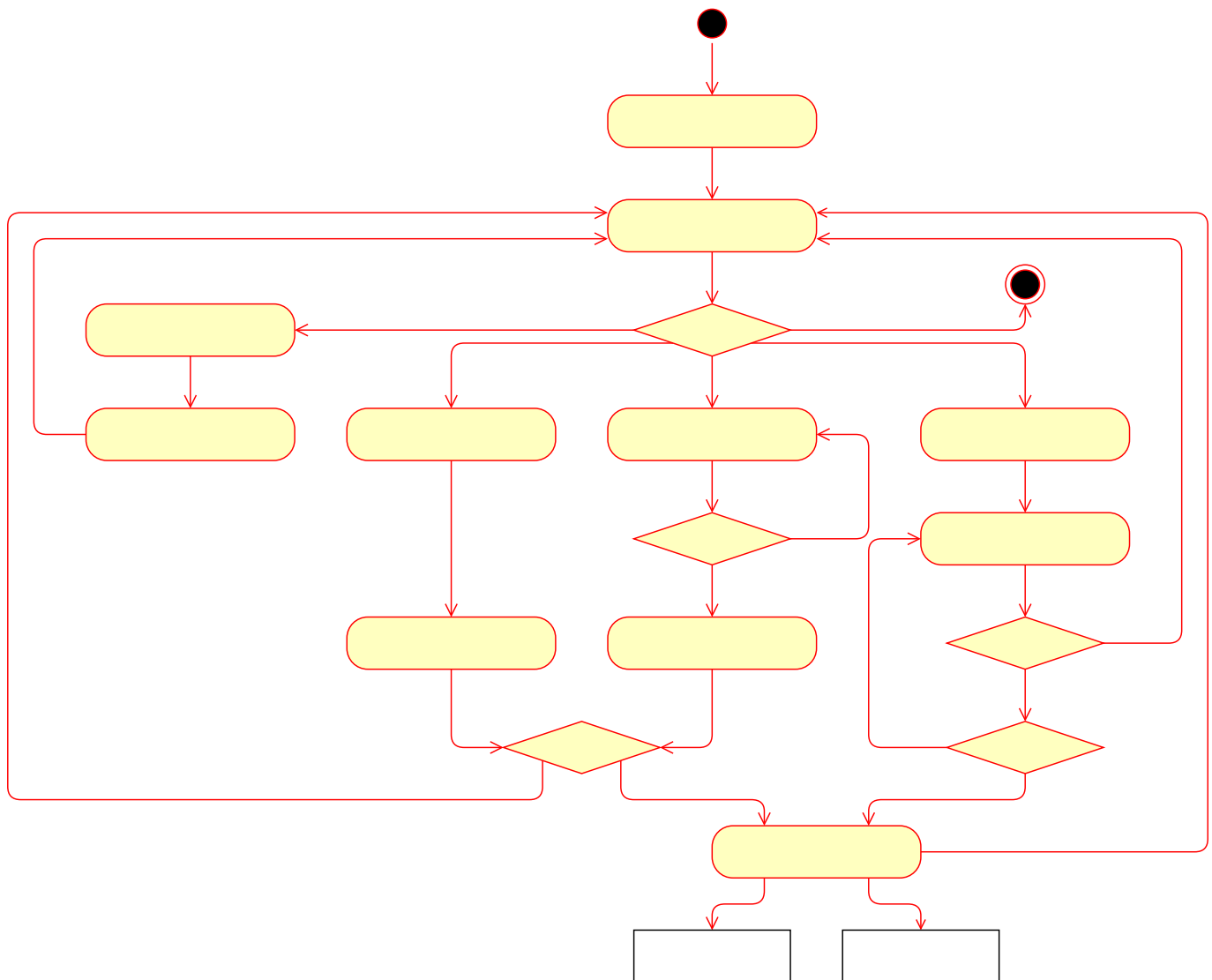


Activity Diagrams

Here you can see two examples for activity diagrams. The first one is based on all workflows we wanted to cover with our software.



The second one describes the workflow Django covers with its admin panel.



Code Documentation

Login

This app is responsible for authenticating users.

Views

`login.views.logout_request` (request)

This view processes all logout requests made by navigating to `/logout`. It logs the user out and displays a goodbye message.

Parameters: `request` – The HTTP request that triggered the view.

Returns: A redirect to the app's root URL (i.e. the login form).

`login.views.main_screen` (request)

This view processes all login requests made via the form found at the app's root URL. If a user who is already logged in tries to access the page, they will automatically be redirected to `/mitglieder`. If the form was submitted, the view gets all data from the submitted form and tries to authenticate the user using that data. If authentication is successful, the user will be logged in and shown an appropriate welcome message. Otherwise, or if the submitted form is invalid, the user will be shown an error message. If the user navigates to the login form (i.e. is not submitting any data), the `AuthenticationForm` provided by Django will be rendered.

Parameters: **request** – The HTTP request that triggered the view.

Returns: The rendered AuthenticationForm if no data was submitted, or a redirect to /mitglieder if the user was logged in successfully.

Templates

All templates can be found under /login/templates/login.

login.html

Contains the login form using Materialize's form components. Variables are defined by the AuthenticationForm supplied by Django.

Ämter

Models

```
class aemter.models.Funktion(*args, **kwargs)
```

Datenbankmodel Funktion

Felder:

- bezeichnung
- workload
- max_members (Maximale Anzahl an Mitgliedern in der Funktion)
- organisationseinheit (Referenziert zugehörige Organisationseinheit)
- **unterbereich (Referenziert zugehörigen Unterbereich)**
Unterbereich kann null sein
- history

exception **DoesNotExist**

exception **MultipleObjectsReturned**

bezeichnung

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

funktionrecht_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

history = <simple_history.manager.HistoryManager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

max_members

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

mitgliedamt_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

objects = <django.db.models.manager.Manager object>

organisationseinheit

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

organisationseinheit_id**save_without_historical_record(*args, **kwargs)**

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

tostring()**unterbereich**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

unterbereich_id**workload**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`class aemter.models.FunktionRecht(*args, **kwargs)`

Datenbankmodell FunktionRecht

Felder:

- funktion
- recht
- history

exception **DoesNotExist**

exception **MultipleObjectsReturned**

funktion

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

funktion_id

history = <simple_history.manager.HistoryManager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

recht

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

recht_id

save_without_historical_record (*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

class aemter.models.HistoricalFunktion (id, bezeichnung, workload, max_members, organisationseinheit, unterbereich, history_id, history_date, history_change_reason, history_type, history_user)

exception DoesNotExist

exception MultipleObjectsReturned

bezeichnung

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

static get_default_history_user (instance)

Returns the user specified by `get_user` method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)

get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)

get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object**history_type**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

history_user_id**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance**instance_type**

alias of **Funktion**

max_members

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

organisationseinheit

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

organisationseinheit_id**property prev_record**

Get the previous history record for the instance. *None* if first.

revert_url ()

URL for this change in the default admin site.

unterbereich

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

unterbereich_id

workload

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

```
class aemter.models.HistoricalFunktionRecht (id, funktion, recht, history_id, history_date,
history_change_reason, history_type, history_user)
```

exception **DoesNotExist**

exception **MultipleObjectsReturned**

funktion

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

funktion_id

static **get_default_history_user** (instance)

Returns the user specified by *get_user* method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)

get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)

get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object

history_type

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

history_user_id

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance

instance_type

alias of **FunktionRecht**

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

property prev_record

Get the previous history record for the instance. *None* if first.

recht

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

recht_id

revert_url ()

URL for this change in the default admin site.

```
class aemter.models.HistoricalOrganisationseinheit (id, bezeichnung,
funktionen_ohne_unterbereich_count, history_id, history_date, history_change_reason, history_type, history_user)
```

exception DoesNotExist

exception MultipleObjectsReturned

bezeichnung

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

funktionen_ohne_unterbereich_count

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

static get_default_history_user (instance)

Returns the user specified by *get_user* method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)

get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)

get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object

history_type

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToManyDescriptor instance.

history_user_id

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance

instance_type

alias of **Organisationseinheit**

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

property prev_record

Get the previous history record for the instance. *None* if first.

revert_url ()

URL for this change in the default admin site.

```
class aemter.models.HistoricalRecht (id, bezeichnung, history_id, history_date, history_change_reason, history_type, history_user)
```

exception DoesNotExist

exception MultipleObjectsReturned**bezeichnung**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

static get_default_history_user (instance)

Returns the user specified by `get_user` method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)**get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)****get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)****history_change_reason**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object**history_type**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

history_user_id**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance**instance_type**

alias of `Recht`

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

property prev_record

Get the previous history record for the instance. *None* if first.

revert_url ()

URL for this change in the default admin site.

```
class aemter.models.HistoricalUnterbereich (id, bezeichnung, organisationseinheit, history_id,
history_date, history_change_reason, history_type, history_user)
```

exception DoesNotExist**exception MultipleObjectsReturned****bezeichnung**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

static get_default_history_user (instance)

Returns the user specified by *get_user* method for manually creating historical objects

```
get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)
```

```
get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField:
history_date>, is_next=True, **kwargs)
```

```
get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField:
history_date>, is_next=False, **kwargs)
```

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object**history_type**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

history_user_id**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance**instance_type**alias of **Unterbereich****property next_record**Get the next history record for the instance. *None* if last.**objects** = <django.db.models.manager.Manager object>**organisationseinheit**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

organisationseinheit_id**property prev_record**Get the previous history record for the instance. *None* if first.**revert_url()**

URL for this change in the default admin site.

class aemter.models.Organisationseinheit(*args, **kwargs)

Datenbankmodel Organisationseinheit

Felder:

- bezeichnung
- history
- funktionen_ohne_unterbereich_count

exception DoesNotExist**exception MultipleObjectsReturned****bezeichnung**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

funktion_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

funktionen_ohne_unterbereich_count

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history = <simple_history.manager.HistoryManager object>**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

save_without_historical_record(*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

unterbereich_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class aemter.models.Recht(*args, **kwargs)

Datenbankmodell Recht

Felder:

- bezeichnung
- history

exception DoesNotExist

exception MultipleObjectsReturned

bezeichnung

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

checklisterecht_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

funktionrecht_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

history = <simple_history.manager.HistoryManager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

save_without_historical_record(*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

class aemter.models.Unterbereich(*args, **kwargs)

Datenbankmodel Unterbereich

Felder:

- bezeichnung
- organisationseinheit (Referenziert zugehörige Organisationseinheit)
- history

exception DoesNotExist

exception MultipleObjectsReturned

bezeichnung

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

funktion_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

history = <simple_history.manager.HistoryManager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

organisationseinheit

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

organisationseinheit_id

save_without_historical_record(*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

Views

aemter.views.main_screen(request)

Displays the Funktionen-screen

Templates

Alle Templates sind unter `aemter/templates/aemter` zu finden.

main_screen.html

department_row.html

Historie

Die Historie zeichnet sämtliche Änderungen (hinzufügen, bearbeiten, löschen) an den folgenden Models des Systems auf:

- `django.contrib.auth.models.User` (Systemnutzer)
- `aemter.*` (alle Models der App `aemter`)
- `mitglieder.*` (alle Models der App `mitglieder`)

Diese Einträge können von Administratoren des Systems auf der entsprechenden Seite unter `./admin/historie` eingesehen werden.

Abhängigkeiten

django-simple-history

- Installation: `pip install django-simple-history`
- Dokumentation: <https://django-simple-history.readthedocs.io/en/latest/>

`django-simple-history` ermöglicht das automatische Aufzeichnen des Zustands eines Models beim Ausführen einer Änderungsoperation (hinzufügen, bearbeiten, löschen).

Views

`historie.views.fetch_entries` (request)

Mit `fetch_entries` kann eine Liste von Historien-Einträgen mitsamt passender Pagination angefordert werden, welche die Einträge enthält, die...

- ...zum angegebenen Tab bzw. Model gehören.
- ...in denen die angegebenen Suchbegriffe vorkommen.
- ...zur angeforderten Seite gehören.

Folgende Aufgaben werden durch diese übernommen:

- Zugriffsbeschränkung: Zugriff wird nur gewährt, wenn der Nutzer angemeldet UND Administrator ist.
- Bereitstellung von Daten: Die View stellt die gewünschte Seite der Historien-Einträge bereit, welche für das gewünschte Model zu den angegebenen Suchbegriffen gefunden wurden.
- Rendern der Liste mit den passenden Historien-Einträgen und der zugehörigen Pagination.

Je nachdem, ob in der `request` Suchbegriffe mitgegeben wurden, werden entweder alle Einträge oder die nach den Suchbegriffen gefilterten Einträge bereitgestellt. Die Filterung funktioniert dabei folgendermaßen:

- Das gewünschte Model wird dahingehend untersucht, ob die wichtigsten Felder eines Eintrags (bei Mitgliedern z.B. ID, Vorname und Name) die Suchbegriffe enthalten.
- Hierfür werden die in Django integrierten `Q Objects` verwendet.
- Alle gefundenen Einträge werden in einem `QuerySet` zusammengefasst, welches anschließend an `render` übergeben wird.

Parameters: **request** – Die HTML-Request, welche den Aufruf der View ausgelöst hat. Enthält stets die gewünschte Seitenzahl, den Namen des Tabs und damit Models, zu dem das Ergebnis geliefert werden soll und optional Suchbegriffe, nach denen das Model durchsucht werden soll.

Returns: Die gerenderte Liste mit den entsprechenden Historien-Einträgen und der zugehörigen Pagination.

`historie.views.list(request)`

Die *list*-View wird aufgerufen, wenn der Nutzer über einen Link erstmalig die Historie aufruft (z.B. aus dem Menü heraus).

Folgende Aufgaben werden von dieser übernommen:

- Bereitstellung von Daten: Es werden alle Historien-Einträge für alle Tabs geholt, anschließend in Seiten à 15 Elemente aufgeteilt und jeweils die erste Seite an die View übergeben.
- Zugriffsbeschränkung: Zugriff wird nur gewährt, wenn der Nutzer angemeldet UND Administrator ist.
- Rendern des Templates der gesamten Seite.

Parameters: **request** – Die HTML-Request, welche den Aufruf der View ausgelöst hat.

Returns: Die gerenderte View.

Templates

Alle Templates sind unter *historie/templates/historie* zu finden.

list.html

Enthält den Grundaufbau der Historie. Die Historie wird hier in die 3 Tabs "Mitglieder", "Ämter" und "Nutzer" unterteilt.

*tabs/**

Unterteilt die drei Tabs "Mitglieder", "Ämter" und "Nutzer" ggf. in weitere Tabs, z.B. bei Mitglieder in "Stammdaten", "E-Mail-Adressen" und "Ämter". Für jedes Model wird für jeden Historien-Eintrag im entsprechenden (Unter-)Tab eine neue Listenzeile samt Modal generiert.

tabs/_pagination.html

Enthält das Template für die Pagination (die Unterteilung der Historien-Einträge in Seiten).

row.html

Je nachdem, zu welchem Model der Eintrag gehört, werden hier die zum Eintrag gehörenden zusätzlichen Daten inkludiert.

Beispiel: Im Model MitgliedMails werden nur die IDs der Mitglieder gespeichert. Damit die Historie aber lesbarer und einfacher verständlich ist, werden z.B. Vor- und Nachname des zugehörigen Mitglieds ermittelt, einmal zum aktuellen und einmal zum Zeitpunkt der Erstellung des Historien-Eintrags.

Falls der Historien-Eintrag zur Änderung eines Datensatzes gehört, werden außerdem die zusätzlichen Daten zum Vorher-Datensatz inkludiert.

rowContent.html

Der eigentliche Inhalt eines Datensatzes. Hier wird der Grundaufbau der Zeile in der angezeigten Liste von Einträgen sowie des zugehörigen Modals beschrieben.

_titleBuilder.html

Beschreibt, wie der Titel, welcher in der Liste von Einträgen sowie auf dem Modal angezeigt wird, zusammengebaut wird.

_noResultsRow.html

Wird inkludiert, falls zu einer Suchanfrage bzw. zu einem Model keine Historien-Einträge vorhanden sind.

_modalDataIncludes.html

Je nachdem, zu welchem Model der Eintrag gehört, wird hier der entsprechende Inhalt des zugehörigen Modals inkludiert.

modalData/*

Für jedes Model wird hier beschrieben, welche Daten wie im zugehörigen Modal präsentiert werden sollen. Falls der Historien-Eintrag zur Änderung eines bestehenden Datensatzes gehört, wird ebenfalls der Datensatz vor der Änderung mitsamt zusätzlichen Daten angezeigt.

Template Tags

`historie.templatetags.t_historie.to_class_name.to_class_name` (value)

Gibt den Namen der Klasse des übergebenen Objekts zurück.

Parameters: **value** (*any*) – Das Objekt, von dem die Klasse ermittelt werden soll.

Returns: Den Namen der Klasse des Objekts.

Return type: str

`historie.templatetags.t_historie.get_associated_data.get_associated_data` (desiredInfo, queryType, primaryKey, timestamp)

Ermittelt zu einem gegebenen Historien-Eintrag gehörige, zusätzliche Daten.

Beispiel: Ein Historien-Eintrag zum Model `mitglied.MitgliedMail` enthält nur die ID des entsprechenden Mitglieds. Mit `get_associated_data` können sämtliche Daten des zur ID gehörenden Mitglieds (z.B. Name oder Anschrift) ermittelt werden, sowohl zum jetzigen Zeitpunkt als auch zu dem Zeitpunkt, zu dem der Historien-Eintrag angelegt wurde.

Parameters:

- **desiredInfo** (*str*) – Der Name des Models, aus welchem die zusätzlichen Daten ermittelt werden sollen. Zulässige Werte: "Mitglied", "Funktion", "Unterbereich", "Organisationseinheit", "Recht"
- **queryType** (*str*) – Gibt an, ob die aktuellen Daten oder die Daten zum Zeitpunkt des Eintrags in die Historie ermittelt werden sollen. Zulässige Werte: "latest", "historical"
- **primaryKey** (*int*) – Die ID des betroffenen Datensatzes, für welchen die Daten ermittelt werden sollen; z.B. die ID des Mitglieds.
- **timestamp** (*datetime*, "semi-optional") – Der Zeitstempel, zu welchem der Historien-Eintrag angelegt wurde. Wird nur benötigt, falls die Daten zum Zeitpunkt des Erstellens des Historien-Eintrags ermittelt werden sollen.

Returns: Eine Instanz des mittels *desiredInfo* angegebenen Models, welche sämtliche Daten des Objekts mit der ID *primaryKey* zum mittels *queryType* und ggf. *timestamp* angegebenen Zeitpunkt enthält.

Mitglieder

Die Django-Applikation "mitglieder" dient zur Verwaltung (Anzeigen, Erstellen, Löschen, Bearbeiten) von Mitgliedern des StuRas. Nutzer, bei denen es sich nicht um Admins handelt, sind dabei nur zum Einsehen der Mitgliederdaten befugt.

Zu jedem Mitglied werden folgende Attribute gespeichert:

- Vorname
- Nachname
- Spitzname (optional)
- Ämter (optional)

- E-Mail-Adresse(n) (optional)
- Anschrift (optional)
- Telefonnummer(n) (optional)

Abhängigkeiten

simplejson

- Installation: `pip install simplejson`
- Dokumentation: <https://simplejson.readthedocs.io/en/latest/>

simplejson ist ein En- und Decoder für JSON.

Views

`mitglieder.views.bereiche_laden` (request)

Rendert ein Dropdown mit allen Bereichen eines bestimmten Referats beim dazugehörigen Amt, nachdem ein Referat bei der Mitgliedererstellung oder -bearbeitung ausgewählt wurde.

Aufgaben:

- Bereitstellung der Daten: Alle Bereiche eines Referats werden aus der Datenbank entnommen.
- Rendern des Templates
- Rechteeinschränkung: Nur angemeldete Nutzer können den Vorgang auslösen

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat. Enthält den Namen des ausgewählten Referats sowie die Nummer des Amts eines Mitglieds.

Returns: Das gerenderte Dropdown.

`mitglieder.views.email_html_laden` (request)

Rendert ein Formular für eine weitere E-Mail, nachdem diese angefordert wurde und inkrementiert die Anzahl der Formulare für eine E-Mail in der View.

Aufgaben:

- Rendern des Formulars
- Erfassen der Anzahl der E-Mails eines Mitglieds
- Rechteeinschränkung: Nur angemeldete Nutzer können den Vorgang auslösen

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat.

Returns: HTTP Response

`mitglieder.views.email_loeschen` (request)

Dekrementiert die Anzahl der Formulare für eine E-Mail in der `mitgliedBearbeitenView` oder `mitgliedErstellenView` nach Löschen eines Formulars.

Aufgaben:

- Erfassen der Anzahl der E-Mails
- Rechteeinschränkung: Nur angemeldete Nutzer können den Vorgang auslösen

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat.

Returns: HTTP Response

`mitglieder.views.erstellen` (request)

Speichert ein neues Mitglied in der Datenbank.

Aufgaben:

- Speichern der Daten: Die Daten werden aus request gelesen und in die Datenbank eingefügt.
- Weiterleitung zur Mitgliederansicht.
- Rechteeinschränkung: Nur Admins können die Funktion auslösen.

Parameters: **request** – Die POST-Request, welche den Aufruf der Funktion ausgelöst hat. Enthält alle Daten zu einem Mitglied.

Returns: Weiterleitung zur Mitgliederansicht.

`mitglieder.views.funktion_loeschen` (request)

Dekrementiert die Anzahl der Formulare für ein Amt in der `mitgliedBearbeitenView` oder `mitgliedErstellenView` nach Löschen eines Formulars.

Aufgaben:

- Erfassen der Anzahl der Ämter
- Rechteeinschränkung: Nur angemeldete Nutzer können den Vorgang auslösen

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat.

Returns: HTTP Response

`mitglieder.views.funktionen_html_laden` (request)

Rendert ein Formular für ein weiteres Amt, nachdem dieses angefordert wurde und inkrementiert die Anzahl der Formulare für ein Amt in der View.

Aufgaben:

- Bereitstellung der Daten: Alle Referate werden aus der Datenbank entnommen.
- Rendern des Templates
- Rechteeinschränkung: Nur angemeldete Nutzer können den Vorgang auslösen

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat.

Returns: Das gerenderte Formular.

`mitglieder.views.funktionen_laden` (request)

Rendert ein Dropdown mit allen Ämtern eines bestimmten Bereich beim dazugehörigen Amt, nachdem ein Bereich bei der Mitgliedererstellung oder -bearbeitung ausgewählt wurde.

Aufgaben:

- Bereitstellung der Daten: Alle Ämter eines Bereichs werden aus der Datenbank entnommen.
- Rendern des Templates
- Rechteeinschränkung: Nur angemeldete Nutzer können den Vorgang auslösen

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat. Enthält den Namen des ausgewählten Bereichs sowie die dazugehörige Nummer des Amts eines Mitglieds.

Returns: Das gerenderte Dropdown.

`mitglieder.views.funktionen_max_member_ueberpruefen` (request, mitglied_id)

`mitglieder.views.main_screen` (request)

Zeigt eine Tabelle mit Mitgliedern an und ermöglicht die Suche nach Mitgliedern mit bestimmten Namen. Admins wird zusätzlich das Löschen von einem oder mehreren Mitgliedern sowie das Wechseln zur View zum Erstellen oder zum Bearbeiten ermöglicht.

Aufgaben:

- Bereitstellung der Daten: Die View holt sämtliche Mitglieder-Einträge aus der Datenbank und stellt diese als Kontext bereit.
- Rendern des Templates
- Rechteeinschränkung: Nur Admins können Mitglieder erstellen, bearbeiten und löschen.

Parameters: **request** – Die HTML-Request, welche den Aufruf der View ausgelöst hat.

Returns: Die gerenderte View.

`mitglieder.views.mitgliedBearbeitenView` (request, mitglied_id)

View zum Bearbeiten eines Mitglieds.

Stellt Textfelder, Dropdowns und Buttons zum Bearbeiten der Attribute bereit, welche mit derzeitigen Attributen des Mitglieds befüllt sind. Über Buttons können weitere Ämter und E-Mail-Adressen hinzugefügt oder bereits bestehende entfernt werden.

Mit Betätigung des Speichern-Buttons wird überprüft, ob Name, Vorname, Ämter und E-Mail-Adressen ausgefüllt wurden und ob alle E-Mail-Adressen gültig sind. Bei erfolgreicher Prüfung wird das Mitglied gespeichert und der Nutzer zu main_screen umgeleitet, ansonsten werden Felder mit fehlenden oder fehlerhaften Eingaben rot markiert.

Aufgaben:

- Zugriffsbeschränkung: Zugriff wird nur gewährt, wenn der Nutzer angemeldet UND Administrator ist.
- Bereitstellung der Daten: Die View holt Attribute eines Mitglieds aus der Datenbank und zeigt diese an.
- Rendern des Templates
- Speichern des Mitglieds in der Datenbank

Parameters:

- **request** – Die HTML-Request, welche den Aufruf der View ausgelöst hat.
- **mitglied_id** – Id des Mitglieds, das bearbeitet werden soll

Returns: Die gerenderte View.

`mitglieder.views.mitgliedErstellenView(request)`

View zum Erstellen eines Mitglieds.

Stellt Textfelder, Dropdowns und Buttons zum Hinzufügen der Attribute bereit. Anfangs steht jeweils genau ein Eingabebereich für ein Amt und eine E-Mail-Adresse zur Verfügung. Über Buttons können weitere dieser hinzugefügt oder bereits bestehende entfernt werden.

Mit Betätigung des Speichern-Buttons wird überprüft, ob Name, Vorname, Ämter und E-Mail-Adressen ausgefüllt wurden und ob alle E-Mail-Adressen gültig sind. Bei erfolgreicher Prüfung wird das Mitglied gespeichert und der Nutzer zu main_screen umgeleitet, ansonsten werden Felder mit fehlenden oder fehlerhaften Eingaben rot markiert.

Aufgaben:

- Zugriffsbeschränkung: Zugriff wird nur gewährt, wenn der Nutzer angemeldet UND Administrator ist.
- Rendern des Templates
- Speichern des Mitglieds in der Datenbank

Parameters: **request** – Die HTML-Request, welche den Aufruf der View ausgelöst hat.

Returns: Die gerenderte View.

`mitglieder.views.mitglied_laden(request)`

Rendert ein Modal mit allen Daten eines aus der Tabelle gewählten Mitglieds.

Aufgaben:

- Bereitstellung der Daten: Die Mitglied-Id wird aus request gelesen und extrahieren aller Daten zum Mitglied mit dieser Id
- Rendern des Templates
- Rechteeinschränkung: Nur angemeldete Nutzer können das gerenderte Template anfordern.

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat. Enthält die Id des Mitglieds, dessen Daten angezeigt werden sollen.

Returns: Das gerenderte Modal, das mit Daten des angeforderten Mitglieds ausgefüllt wurde

`mitglieder.views.mitglieder_loeschen(request)`

Löscht ausgewählte Mitglieder aus der Datenbank.

Aufgaben:

- Entfernen der Daten: Alle Daten der Mitglieder werden aus der Datenbank entfernt.
- Rendern des Templates
- Rechteeinschränkung: Nur angemeldete Nutzer können Löschvorgänge auslösen

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat. Enthält die Ids der Mitglieder, die entfernt werden sollen

Returns: HTTP Response

`mitglieder.views.speichern(request, mitglied_id)`

Speichert ein bearbeitetes Mitglied in der Datenbank.

Aufgaben:

- Speichern der Daten: Die Daten werden aus request gelesen und in der Datenbank gespeichert. Ämter und E-Mails werden gespeichert, indem zunächst alle bereits vorhandenen Instanzen gelöscht werden und anschließend alle Ämter und E-Mails aus request gespeichert werden.
- Weiterleitung zur Mitgliederansicht.
- Rechteeinschränkung: Nur Admins können die Funktion auslösen.

Parameters:

- **request** – Die POST-Request, welche den Aufruf der Funktion ausgelöst hat. Enthält alle Daten zu einem Mitglied.
- **mitglied_id** – Die Id des Mitglieds, das bearbeitet wurde.

Returns: Weiterleitung zur Mitgliederansicht.

`mitglieder.views.suchen` (request)

Anzeige von Mitgliedern, deren Namen auf die Sucheingabe passen.

Aufgaben:

- Bereitstellung der Daten: Die Sucheingabe wird in mehrere Suchbegriffe unterteilt. Bei allen Mitgliedern der Datenbank wird überprüft, ob sie mindestens einen der Suchbegriffe im Vor- oder Nachnamen als Substring enthalten. Diese Mitglieder werden angezeigt und nach der Anzahl der Suchbegriffe, die auf den Vor- oder Nachnamen passen, sortiert.
- Rendern des Templates
- Rechteeinschränkung: Nur angemeldete Nutzer können die Funktion auslösen.

Parameters: **request** – Die Ajax-Request, welche den Aufruf der Funktion ausgelöst hat. Enthält die Sucheingabe.

Returns: Das gerenderte Templates mit den gefunden Mitgliedern.

Templates

Alle Templates sind unter `mitglieder/templates/mitglieder` zu finden.

aemter.html

Enthält die Felder für den Kandidaturzeitraum und Dropdowns für Attribute eines Amts. Initial wird neben den Textfeldern nur das Dropdown für ein Referat angezeigt. Die restlichen Dropdowns werden über die Auswahl eines Referats bzw. Bereichs angezeigt.

amt_dropdown_list_options.html

Dropdown zur Auswahl eines Amts. Die angezeigten Auswahlmöglichkeiten hängen vom gewählten Bereich ab.

bereich_dropdown_list_options.html

Dropdown zur Auswahl eines Bereichs. Die angezeigten Auswahlmöglichkeiten hängen vom gewählten Referat ab.

email.html

Enthält ein Textfeld zur Eingabe einer E-Mail-Adresse sowie einen Löschbutton zum Entfernen der jeweiligen E-Mail-Adresse.

email_input.html

Enthält 0 bis n Kopien von *email.html* sowie einen Button, um genau eine weitere Kopie hinzuzufügen.

mitglieder.html

Ansicht mit allen Mitgliedern, welche über eine Tabelle angezeigt werden. Über das Anklicken einer Zeile wird ein Modal mit allen Daten eines Mitglieds angezeigt. Auch die Suche nach Mitgliedsnamen, das Löschen von Mitgliedern und das Wechseln zur *mitgliedErstellenView* kann hier durchgeführt werden.

mitglied_erstellen_bearbeiten.html

Ermöglicht die Eingabe aller Attribute eines bereits existierenden oder neuen Mitglieds.

modal.html

Ein Modal, das alle Attribute eines Mitglieds mit einer bestimmten Id anzeigt.

row.html

Eine Tabellenzeile, um Name, Vorname, Ämter, E-Mail-Adressen und die Telefonnummer eines Mitglieds anzuzeigen. Admins wird hier zusätzlich eine Checkbox und ein Button zum Bearbeiten des Mitglieds angezeigt. Wird von *mitglieder.html* inkludiert.

Checklisten

Checklisten allow administrators to keep track of which tasks need to be completed in order to welcome a new member (Mitglied) to StuRa or a new Funktion inside of the StuRa. A Checkliste may consist of a set of general tasks (Aufgaben) and/or, if a Funktion was selected, all permissions (Rechte) that have to be granted to the member (Mitglied).

Models

`class checklisten.models.Aufgabe (*args, **kwargs)`

Defines a general task that can be added to a checklist.

- **bezeichnung**: The task's title that will be shown in the UI. May contain up to 50 characters and must not be null.
- **history**: Contains a log of all changes made to the model. Provided by django-simple-history.

`exception DoesNotExist`

`exception MultipleObjectsReturned`

bezeichnung

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

checklisteaufgabe_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

history = `<simple_history.manager.HistoryManager object>`

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

save_without_historical_record(*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

class checklisten.models.**Checkliste**(*args, **kwargs)

Represents a checklist.

- **mitglied**: The Mitglied the checklist was created for. Must not be null.
 - **amt**: The Funktion the checklist was created for. Can be null.
 - **history**: Contains a log of all changes made to the model. Provided by django-simple-history.
- Note that the checklist will be deleted if the associated Mitglied or Funktion is deleted (cascade).

exception DoesNotExist

exception MultipleObjectsReturned

amt

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

amt_id

checklisteaufgabe_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

checklisterecht_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

history = <simple_history.manager.HistoryManager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

mitglied

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

mitglied_id

objects = <django.db.models.manager.Manager object>

save_without_historical_record (*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

class `checklisten.models.ChecklisteAufgabe` (*args, **kwargs)

Represents a general task inside of a specific checklist.

- **checkliste**: The checklist that this task belongs to. Must not be null.
- **aufgabe**: The task that was assigned to the checklist. Must not be null.
- **abgehakt**: Whether or not the task has been completed for this specific checklist. Must not be null and is false by default.
- **history**: Contains a log of all changes made to the model. Provided by `django-simple-history`. Please note that the entry is deleted if the associated `Checkliste` or `Aufgabe` is deleted (cascade).

exception `DoesNotExist`

exception `MultipleObjectsReturned`

abgehakt

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

aufgabe

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

aufgabe_id

checkliste

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

checkliste_id

history = <simple_history.manager.HistoryManager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

save_without_historical_record(*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

class checklisten.models.**ChecklisteRecht**(*args, **kwargs)

Represents a Recht to be given inside of a specific checklist.

- **checkliste**: The checklist that this Recht was assigned to. Must not be null.
- **recht**: The Recht that was assigned to the checklist. Must not be null.
- **abgehakt**: Whether or not the Recht has been given for this specific checklist. Must not be null and is false by default.
- **history**: Contains a log of all changes made to the model. Provided by django-simple-history. Please note that the entry is deleted if the associated Checkliste or Recht is deleted (cascade).

exception DoesNotExist

exception MultipleObjectsReturned

abgehakt

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

checkliste

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToManyDescriptor instance.

checkliste_id

history = <simple_history.manager.HistoryManager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

recht

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToManyDescriptor instance.

recht_id

save_without_historical_record(*args, **kwargs)

Save model without saving a historical record

Make sure you know what you're doing before you use this method.

```
class checklisten.models.HistoricalAufgabe (id, bezeichnung, history_id, history_date,
history_change_reason, history_type, history_user)
```

exception DoesNotExist

exception MultipleObjectsReturned

bezeichnung

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

static get_default_history_user (instance)

Returns the user specified by *get_user* method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)

get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)

get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object

history_type

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

history_user_id

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance

instance_type

alias of **Aufgabe**

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

property prev_record

Get the previous history record for the instance. *None* if first.

revert_url ()

URL for this change in the default admin site.

```
class checklisten.models.HistoricalCheckliste (id, mitglied, amt, history_id, history_date,
history_change_reason, history_type, history_user)
```

exception DoesNotExist

exception MultipleObjectsReturned

amt

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

amt_id**static get_default_history_user (instance)**

Returns the user specified by *get_user* method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)

get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)

get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object**history_type**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

history_user_id

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance

instance_type

alias of **Checkliste**

mitglied

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

mitglied_id

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

property prev_record

Get the previous history record for the instance. *None* if first.

revert_url ()

URL for this change in the default admin site.

class checklisten.models.HistoricalChecklisteAufgabe (id, abgehakt, checkliste, aufgabe, history_id, history_date, history_change_reason, history_type, history_user)

exception DoesNotExist

exception MultipleObjectsReturned

abgehakt

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

aufgabe

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

aufgabe_id**checkliste**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

checkliste_id**static get_default_history_user** (instance)

Returns the user specified by *get_user* method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)

get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)

get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object**history_type**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

history_user_id**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance**instance_type**

alias of **ChecklisteAufgabe**

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

property prev_record

Get the previous history record for the instance. *None* if first.

revert_url ()

URL for this change in the default admin site.

```
class checklisten.models.HistoricalChecklisteRecht (id, abgehakt, checkliste, recht, history_id,
history_date, history_change_reason, history_type, history_user)
```

exception DoesNotExist

exception MultipleObjectsReturned

abgehakt

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

checkliste

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

checkliste_id

static get_default_history_user (instance)

Returns the user specified by *get_user* method for manually creating historical objects

get_history_type_display (*, field=<django.db.models.fields.CharField: history_type>)

get_next_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=True, **kwargs)

get_previous_by_history_date (*, field=<django.db.models.fields.DateTimeField: history_date>, is_next=False, **kwargs)

history_change_reason

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_date

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_object

history_type

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

history_user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

history_user_id**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

property instance**instance_type**

alias of **ChecklisteRecht**

property next_record

Get the next history record for the instance. *None* if last.

objects = <django.db.models.manager.Manager object>

property prev_record

Get the previous history record for the instance. *None* if first.

recht

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

recht_id**revert_url ()**

URL for this change in the default admin site.

Views

checklisten.views.abhaken (request)

This view is responsible for checking or unchecking a task from the checklist. It first checks if the user is allowed to check a task, i.e. is authenticated and superuser. Next, the parameters task_type and task_id are fetched from the request. They are used to determine if an Aufgabe or a Recht was checked, and to get the correct task. Finally, the task is checked or unchecked depending on its current state and the changes are saved.

Parameters: **request** – The HTTP request that triggered the view, including parameters task_type and task_id.

Returns: An empty HttpResponse if the operation was successful.

Returns: An HttpResponse indicating the error if an error has occurred or if the user is not allowed to perform the operation.

`checklisten.views.erstellen (request)`

This view is responsible for creating a new checklist. It first checks if the user is allowed to create a new checklist (i.e. is authenticated and superuser). Next, the Mitglied's and Funktion's IDs as well as whether general tasks shall be included are fetched from the request. The view then tries to find the Mitglied and Funktion with the specified ID, and returns an error message if at least one of them could not be found. After that, the view checks if a checklist for this Mitglied and Funktion already exists. If that is the case, an error message is shown to the user. Finally, the new checklist is created and all Aufgaben and Rechte are added to it according to the specified parameters in the request.

Parameters: **request** – The HTTP request that triggered the view, including parameters `mitgliedSelect`, `funktionSelect` and `generalTasksCheckbox`.

Returns: A `HttpResponse`, if an error has occurred, indicating the error to the user.

Returns: A redirect to `/checklisten` if creating the checklist was successful or a checklist for the same Mitglied and Funktion already exists.

`checklisten.views.get_funktionen (request)`

This view is responsible for getting all Funktionen associated to a Mitglied that was selected in the create checklist modal. First, it checks if the user is allowed to get a list of Funktionen for a Mitglied in this context (i.e. the user is authenticated and superuser). Next, all Funktionen for the specified `mitglied_id` are determined and returned through a rendered template of select options to populate the dropdown in the create checklist modal.

Parameters: **request** – The HTTP request that triggered the view, including the `mitglied_id` to get the Funktionen for.

Returns: An `HttpResponse` indicating the error if an error occurred.

Returns: The rendered select options to populate the dropdown with if everything was successful.

`checklisten.views.loeschen (request)`

This view is responsible for deleting an existing checklist. First, it checks whether the user is allowed to delete the checklist (i.e. is authenticated and superuser). Next, the checkliste specified in the request's parameter `checkliste_id` is deleted. Since all ForeignKey relations in other models are set to cascade if the checklist is deleted (i.e. in `ChecklisteRecht` and `AufgabeRecht`), only the checklist itself needs to be deleted explicitly.

Parameters: **request** – The HTTP request that triggered the view, including parameter `checkliste_id`.

Returns: An empty `HttpResponse` if deleting the checklist was successful.

Returns: An `HttpResponse` indicating an error if one occurred or a user is not allowed to delete a checklist.

`checklisten.views.main_screen (request)`

This view renders all existing checklists. Furthermore, it provides the 20 latest Mitglieder to the modal used for creating a new checklist. If the user is not authenticated, an error message will be displayed and the user is redirected to the login page.

Parameters: **request** – The HTTP request that triggered the view.

Returns: The rendered `main_screen` view, or a redirect to the login page if the user is not authenticated.

Templates

All templates can be found under `checklisten/templates/checklisten`.

`main_screen.html`

This template contains the main page of the app. Each Checkliste is represented by a card and consists of three main elements:

- The header contains the name of the Mitglied and, if specified, the Funktion that the Checkliste was created for.
- The general tasks section only exists if specified while creating the Checkliste and displays all `ChecklisteAufgabe` objects associated with the Checkliste.
- The permissions section only exists if a Funktion was selected for this Checkliste and displays all `ChecklisteRecht` objects associated with the Checkliste.

Tasks can only be checked if the user is administrator; otherwise, the checkboxes are disabled. The same goes for creating a new Checklist; otherwise, the create button is not visible. The template also contains JavaScript methods used to make AJAX requests to the server, e.g. to complete task or delete a Checklist. Refer to the source code comments for more information.

_createModal.html

This template contains the modal that is shown when creating a new Checklist. The modal includes:

- A dropdown to select the Mitglied that the Checklist shall be created for. Please note that only the latest 20 Mitglieder are shown to improve performance.
- A dropdown to select the Funktion that the Checklist shall be created for. It only contains Funktionen belonging to the Mitglied selected beforehand.
- A checkbox to determine whether the set of general tasks (Aufgaben) shall be included in the Checklist. Will be force-checked and disabled if no Funktion is selected to prevent empty checklists from being created.

The template also contains JavaScript methods to populate the Funktionen dropdown via an AJAX request to the server, as well as to correctly disable and check the general tasks checkbox if no Funktion was selected. Refer to the source code comments for more information.

_deleteModal.html

This template only contains a confirmation modal if the user is trying to delete a Checklist to prevent accidental deletions.

_funktionSelectOptions.html

This template is used to generate the Funktion dropdown options for the create modal. It is only used by the `get_funktionen` view and is not referenced directly in other templates.

Template Tags

`checklisten.templatetags.t_checklisten.get_perms.get_perms (checklist_id)`

Gets all ChecklistRecht objects that belong to the Checklist identified by `checklist_id`.

Parameters: `checklist_id (int)` – The ID of the Checklist to fetch the ChecklistRecht objects for.

Returns: A QuerySet containing all ChecklistRecht objects for the specified Checklist.

Return type: QuerySet

`checklisten.templatetags.t_checklisten.get_tasks.get_tasks (checklist_id)`

Gets all ChecklistAufgabe objects that belong to the Checklist identified by `checklist_id`.

Parameters: `checklist_id (int)` – The ID of the Checklist to fetch the ChecklistAufgabe objects for.

Returns: A QuerySet containing all ChecklistAufgabe objects for the specified Checklist.

Return type: QuerySet

Importscripts

You can use the following script/scripts to import some data in your `db.sqlite3`. These script/scripts are used in the functional tests.

How to use:

Change to the directory `cd importscrips` and execute `main.py python3 ./main.py`

CSV Data:

- `ReferateUnterbereicheAemter.csv`

- contains basic data for the tables: “Organisationseinheit, Unterbereich und Funktion”

Functions:

```
importscripts.main.importAemter (file)
```

!WARNING! This function clears following Tables in your Database:

- Organisationseinheit
- Unterbereich
- Funktion

To use this function you need to have a file.csv with following structure:

- delimiter = ‘,’
- organisationseinheit,unterbereich,funktion,max_members
- First line is a heading and will not be imported

Parameters: **file** (*TextIO*) – File containing the data to be imported

Returns: No return Value

Tests

This section contains the description of the functional UI tests created with the Selenium testing framework. The Pagination and the Module historie are passively tested in the other Tests.

Sources

- [django-testing-docs](#)

Dependencies

selenium

- Installation: `pip install selenium`
- Documentation: www.selenium.dev

Webdriver for Firefox

Link to Mozilla’s site of the Gecko-Webdriver [geckodriver](#).

MyTestCase

```
class tests.MyTestCase.MyTestCase (methodName='runTest')
```

Setup and Teardown funktions are specified here. The following Testcases inherit from this class.

All testcases inheriting from this class are testing the User Interface.

setUp ()

This function is called before every testcase.

It sets up the webdriver and creates 1 admin and 1 user. You can adjust the webdriver by changing the *options* parameter. The Importscripsts from the folder *importscripts* are also called here.

The Webdriver Instance is stored in **self.browser**.

Parameters: **self** –

Returns: No return Value

tearDown ()

This function is called after every testcase.

The Webdriver Instance that is stored in **self.browser** will be closed. :param self: :type self: :return: No return Value

Helper Functions

`tests.MyFuncLogin.loginAsLukasAdmin (self)`

Opens a Browser instance and login as admin with the account testlukasadmin.

Parameters: **self** –

Returns: No return Value

`tests.MyFuncLogin.loginAsLukasUser (self)`

Opens a Browser instance and login as user with the account testlukas.

Parameters: **self** –

Returns: No return Value

`tests.MyFuncMitglieder.addMitglied (self)`

Add a member with default parameters over the “mitglieder/erstellen” view. You need to be on the “mitglieder/” site to call this function.

Parameters: **self** –

Returns: No return Value

`tests.MyFuncMitglieder.addMitgliedWithParameters (self, vorname, nachname, spitzname)`

Add a member with default parameters over the “mitglieder/erstellen” view. You can change firstname, lastname and username. You need to be on the “mitglieder/” site to call this function.

Parameters:

- **self** –
- **vorname** (*string*) – Firstname of the Member, that you want to create.
- **nachname** (*string*) – Lastname of the Member, that you want to create.
- **spitzname** (*string*) – Username of the Member, that you want to create.

Returns: No return Value

`tests.MyFuncAemter.createAmt (self, organisationseinheit, unterbereich, funktion)`

Erstellen eines Amtes über die GUI, benötigt ist ein login als AdminPanel Ausgang ist, dass der User Angemeldet ist und sich in der Mitglieder sicht befindet.

Parameters:

- **self** –
- **organisationseinheit** (*string*) – Organisationseinheit, dem das Funktion zugeordnet werden soll
- **unterbereich** (*string*) – Unterbereich des Referats
- **funktion** (*string*) – Angabe des Namens, der das neue Funktion erhalten soll

Returns: No return Value

`tests.MyFuncAemter.createReferat (self, organisationseinheit)`

Erstellen eines Referates über die GUI, benötigt ist ein login als AdminPanel Ausgang ist, dass der User Angemeldet ist und sich in der Mitglieder sicht befindet.

Parameters:

- **self** –
- **organisationseinheit** (*string*) – Name, wie das neue Organisationseinheit heißen soll

Returns: No return Value

`tests.MyFuncAemter.createUnterbereich (self, organisationseinheit, unterbereich)`

Erstellen eines Unterbereiches über die GUI, benötigt ist ein login als AdminPanel Ausgang ist, dass der User Angemeldet ist und sich in der Mitglieder sicht befindet.

Parameters:

- **self** –
- **organisationseinheit** (*string*) – Organisationseinheit, dem der Unterbereich zugeordnet werden soll
- **unterbereich** (*string*) – Name des Unterbereichs

Returns: No return Value

Login Module Tests

The System under test (SUT) is the “login” module.

Testcase 001

```
class tests.test_001_admin.TestAdmin (methodName='runTest')
```

```
test_login_superuser ()
```

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can login as Admin and all sites are displayed correctly.

Testcase 002

```
class tests.test_002_user.TestUser (methodName='runTest')
```

```
test_login_user ()
```

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can login as User and all sites are displayed correctly. But that you can not reach Admin-only content.

Mitglieder Module Tests

The System under test (SUT) is the “mitglieder” module.

Testcase 003

TODO: under Construction

Testcase 004

```
class tests.test_004_mitgliedHinzufuegen.TestMitgliedHinzufuegen (methodName='runTest')
```

test_1MitgliedHinzufügen_AsSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can add a new Member as Admin and if the Member is displayed correctly in the table.

Steps:

- login as Admin
- add a Member

test_50MitgliederHinzufügen_AsSuperuser_lookAsUser ()

This is a complex “positive” Systemtest as Blackboxtest. Here we want to check if you can add a multiple new Members (50) as Admin and if the Member is displayed correctly in the table. We also want to check if the Pagination is working correctly.

Steps:

- login as Admin
- add a Member (as loop)
- check Pagination

Testcase 005

```
class tests.test_005_mitgliedEntfernen.TestMitgliedEntfernen (methodName='runTest')
```

test_1MitgliedEntfernen_AsSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can delete a new Member as Admin and if the Member is deleted correctly in the table.

Steps:

- login as Admin
- add a Member
- delete Member

Testcase 006

```
class tests.test_006_mitgliedAendern.TestMitgliedAendern (methodName='runTest')
```

test_1MitgliedAendern_AsSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can change information of a Member as Admin and if the Member is changed correctly in the table.

Steps:

- login as Admin
- add a Member
- change Member Data
- check if everything is correct displayed

Aemter Module Tests

The System under test (SUT) is the “aemter” module.

Testcase 007


```
class tests.test_007_aemtHinzufuegen.TestAemtHinzufuegen (methodName='runTest')
```

test_1FunktionHinzufuegen_AssSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can add a funktion as Admin.
Steps:

- login as Admin
- add a funktion

test_1OrganisationseinheitHinzufuegen_AssSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can add a “Organisationseinheit” as Admin.
Steps:

- login as Admin
- add a “Organisationseinheit”

test_1UnterbereichHinzufuegen_AssSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can add a “unterbereich” as Admin.
Steps:

- login as Admin
- add a “unterbereich”

test_ReferatUnterbereichAmtHinzufuegen_AssSuperuser ()

This is a complex “positive” Systemtest as Blackboxtest. Here we want to check if you can add a organisationseinheit, unterbereich and funktion as Admin. We also check if we can add a new Member with the new data and if everything is displayed correctly in “/aemter”.
Steps:

- login as Admin
- add a “organisationseinheit”
- add a “unterbereich”
- add a funktion
- create a Member
- navigate to aemteruebersicht “/aemter”

Testcase 008

```
class tests.test_008_aemtEntfernen.TestAemtEntfernen (methodName='runTest')
```

test_1AemtEntfernen_AssSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can delete a funktion as Admin.
Steps:

- login as Admin
- add a funktion
- delete the funktion

test_1ReferatEntfernen_AssSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can delete a “Organisationseinheit” as Admin.
Steps:

- login as Admin

- add a “Organisationseinheit”
- delete the “Organisationseinheit”

test_1UnterbereichEntfernen_AsSuperuser ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can delete a “unterbereich” as Admin.

Steps:

- login as Admin
- add a “unterbereich”
- delete the “unterbereich”

Testcase 009

```
class tests.test_009_aemtAendern.TestAemtAendern (methodName='runTest')
```

test_1AmtBezeichnungAendern ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can change information of a funktion as Admin. We change the name of the funktion by appending a “_1”.

test_1AmtReferatAendern ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can change the “organisationseinheit” of a funktion as Admin.

test_1AmtWorkloadAendern ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can change the workload of a funktion as Admin.

test_1OrganisationseinheitBezeichnungAendern ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can change information of a “organisationseinheit” as Admin. We change the name of the “organisationseinheit” by appending a “_1”.

test_1UnterbereichBezeichnungAendern ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can change information of a “unterbereich” as Admin. We change the name of the “unterbereich” by appending a “_1”.

test_1UnterbereichReferatAendern ()

This is a “positive” Systemtest as Blackboxtest. Here we want to check if you can change the “organisationseinheit” of a “unterbereich” as Admin.

Latest test coverage report

Commands to generate a coverage report:

```
coverage run --source=aemter,bin,checklisten,historie,login,mitglieder,tests --omit=*/
coverage report
```

coverage report

Name	Stmts	Miss	Cover
-----	-----	-----	-----
aemter/__init__.py	0	0	100%
aemter/admin.py	26	0	100%
aemter/apps.py	3	0	100%
aemter/models.py	35	2	94%
aemter/tests/__init__.py	0	0	100%
aemter/tests/test_apps.py	7	0	100%
aemter/tests/test_models.py	17	0	100%

aemter/tests/test_urls.py	8	0	100%
aemter/tests/test_views.py	24	0	100%
aemter/urls.py	7	0	100%
aemter/views.py	22	0	100%
bin/__init__.py	0	0	100%
bin/asgi.py	4	4	0%
bin/settings.py	19	0	100%
bin/urls.py	3	0	100%
bin/wsgi.py	4	4	0%
checklisten/__init__.py	0	0	100%
checklisten/admin.py	1	0	100%
checklisten/apps.py	3	0	100%
checklisten/models.py	10	0	100%
checklisten/tests/__init__.py	0	0	100%
checklisten/tests/test_apps.py	7	0	100%
checklisten/urls.py	7	0	100%
checklisten/views.py	8	5	38%
historie/__init__.py	0	0	100%
historie/apps.py	3	0	100%
historie/models.py	4	0	100%
historie/templatetags/__init__.py	0	0	100%
historie/templatetags/get_associated_data.py	25	19	24%
historie/templatetags/to_class_name.py	5	0	100%
historie/tests/__init__.py	0	0	100%
historie/tests/test_apps.py	7	0	100%
historie/tests/test_urls.py	8	0	100%
historie/tests/test_views.py	22	0	100%
historie/urls.py	4	0	100%
historie/views.py	101	54	47%
login/__init__.py	0	0	100%
login/apps.py	3	0	100%
login/tests/__init__.py	0	0	100%
login/tests/test_apps.py	7	0	100%
login/tests/test_urls.py	11	0	100%
login/tests/test_views.py	35	0	100%
login/urls.py	4	0	100%
login/views.py	27	2	93%
mitglieder/__init__.py	0	0	100%
mitglieder/admin.py	2	0	100%
mitglieder/apps.py	3	0	100%
mitglieder/models.py	42	1	98%
mitglieder/tests/__init__.py	0	0	100%
mitglieder/tests/test_apps.py	7	0	100%
mitglieder/tests/test_models.py	17	0	100%
mitglieder/tests/test_urls.py	47	0	100%
mitglieder/tests/test_views.py	51	0	100%
mitglieder/urls.py	4	0	100%
mitglieder/views.py	205	66	68%
tests/MyFuncAemter.py	34	0	100%
tests/MyFuncLogin.py	33	8	76%
tests/MyFuncMitglieder.py	103	14	86%
tests/MyTestCase.py	31	4	87%
tests/__init__.py	0	0	100%
tests/test_001_admin.py	6	0	100%
tests/test_002_user.py	6	0	100%
tests/test_003_multiuser.py	1	0	100%
tests/test_004_mitgliedHinzufuegen.py	38	4	89%
tests/test_005_mitgliedEntfernen.py	13	0	100%
tests/test_006_mitgliedAendern.py	13	0	100%

tests/test_007_aemtHinzufuegen.py	63	0	100%
tests/test_008_aemtEntfernen.py	47	0	100%
tests/test_009_aemtAendern.py	92	0	100%

TOTAL	1339	187	86%

Commands

delete_old_historie

```
class bin.management.commands.delete_old_historie.Command (stdout=None, stderr=None,
no_color=False, force_color=False)
```

handle (*args, **options)

This command deletes all entries from the Historie of Mitglied, MitgliedAmt and MitgliedMail that

- are at least 1 year old if the referenced Mitglied does not exist in the database anymore
- are at least 5 years old otherwise

Please note that 1 year is equivalent to 365 days, so leap years are not accounted for.

It can be called using `python manage.py delete_old_historie` and can thus be automated, for example by setting up a cronjob.

help = *'Deletes all entries concerning Mitglieder from the Historie older than 5 years. If the associated Mitglied is not in the database anymore, the entry will be deleted if it is older than 1 year.'*

clean_duplicate_history

django-simple-history includes a command that can be used to remove all duplicate entries in the Historie. For example, an entry is created in MitgliedMail and MitgliedAmt when the associated Mitglied is changed, even though no changes have been made to MitgliedMail and MitgliedAmt. To get rid of these entries, you can use:

```
python manage.py clean_duplicate_history [-m ...] --auto
```

The -m flag is optional and is used to specify the amount of minutes to go back when searching for duplicate entries. This command can be automated, e.g. by running a cronjob.

Index

A

[abgehakt](#) ([checklisten.models.ChecklisteAufgabe](#) attribute)
 ([checklisten.models.ChecklisteRecht](#) attribute)
 ([checklisten.models.HistoricalChecklisteAufgabe](#) attribute)
 ([checklisten.models.HistoricalChecklisteRecht](#) attribute)
[abhaken\(\)](#) (in module [checklisten.views](#))
[aemter.models](#)
 module
[aemter.views](#)
 module
[amt](#) ([checklisten.models.Checkliste](#) attribute)
 ([checklisten.models.HistoricalCheckliste](#) attribute)
[amt_id](#) ([checklisten.models.Checkliste](#) attribute)
 ([checklisten.models.HistoricalCheckliste](#) attribute)
[aufgabe](#) ([checklisten.models.ChecklisteAufgabe](#) attribute)
 ([checklisten.models.HistoricalChecklisteAufgabe](#) attribute)
[Aufgabe](#) (class in [checklisten.models](#))
[Aufgabe.DoesNotExist](#)
[Aufgabe.MultipleObjectsReturned](#)
[aufgabe_id](#) ([checklisten.models.ChecklisteAufgabe](#) attribute)
 ([checklisten.models.HistoricalChecklisteAufgabe](#) attribute)

B

[bereiche_laden\(\)](#) (in module [mitglieder.views](#))
[bezeichnung](#) ([aemter.models.Funktion](#) attribute)
 ([aemter.models.HistoricalFunktion](#) attribute)
 ([aemter.models.HistoricalOrganisationseinheit](#) attribute)
 ([aemter.models.HistoricalRecht](#) attribute)
 ([aemter.models.HistoricalUnterbereich](#) attribute)
 ([aemter.models.Organisationseinheit](#) attribute)
 ([aemter.models.Recht](#) attribute)
 ([aemter.models.Unterbereich](#) attribute)
 ([checklisten.models.Aufgabe](#) attribute)
 ([checklisten.models.HistoricalAufgabe](#) attribute)
[bin.management.commands.delete_old_historie](#)

module

C

[checkliste](#) ([checklisten.models.ChecklisteAufgabe](#) attribute)
 ([checklisten.models.ChecklisteRecht](#) attribute)
 ([checklisten.models.HistoricalChecklisteAufgabe](#) attribute)
 ([checklisten.models.HistoricalChecklisteRecht](#) attribute)
[Checkliste](#) (class in [checklisten.models](#))
[Checkliste.DoesNotExist](#)
[Checkliste.MultipleObjectsReturned](#)
[checkliste_id](#) ([checklisten.models.ChecklisteAufgabe](#) attribute)
 ([checklisten.models.ChecklisteRecht](#) attribute)
 ([checklisten.models.HistoricalChecklisteAufgabe](#) attribute)
 ([checklisten.models.HistoricalChecklisteRecht](#) attribute)
[ChecklisteAufgabe](#) (class in [checklisten.models](#))
[ChecklisteAufgabe.DoesNotExist](#)
[ChecklisteAufgabe.MultipleObjectsReturned](#)
[checklisteaufgabe_set](#) ([checklisten.models.Aufgabe](#) attribute)
 ([checklisten.models.Checkliste](#) attribute)
[checklisten.models](#)
 module
[checklisten.templatetags.t_checklisten.get_perms](#)
 module
[checklisten.templatetags.t_checklisten.get_tasks](#)
 module
[checklisten.views](#)
 module
[ChecklisteRecht](#) (class in [checklisten.models](#))
[ChecklisteRecht.DoesNotExist](#)
[ChecklisteRecht.MultipleObjectsReturned](#)
[checklisterecht_set](#) ([aemter.models.Recht](#) attribute)
 ([checklisten.models.Checkliste](#) attribute)
[Command](#) (class in [bin.management.commands.delete_old_historie](#))

E

[email_html_laden\(\)](#) (in module [mitglieder.views](#))
[email_loeschen\(\)](#) (in module [mitglieder.views](#))
[erstellen\(\)](#) (in module [checklisten.views](#))

(in module mitglieder.views)

F

fetch_entries() (in module historie.views)

funktion (aemter.models.FunktionRecht attribute)

(aemter.models.HistoricalFunktionRecht attribute)

Funktion (class in aemter.models)

Funktion.DoesNotExist

Funktion.MultipleObjectsReturned

funktion_id (aemter.models.FunktionRecht attribute)

(aemter.models.HistoricalFunktionRecht attribute)

funktion_loeschen() (in module mitglieder.views)

funktion_set (aemter.models.Organisationseinheit attribute)

(aemter.models.Unterbereich attribute)

funktionen_html_laden() (in module mitglieder.views)

funktionen_laden() (in module mitglieder.views)

funktionen_max_member_ueberpruefen() (in module mitglieder.views)

funktionen_ohne_unterbereich_count
(aemter.models.HistoricalOrganisationseinheit attribute)

(aemter.models.Organisationseinheit attribute)

FunktionRecht (class in aemter.models)

FunktionRecht.DoesNotExist

FunktionRecht.MultipleObjectsReturned

funktionrecht_set (aemter.models.Funktion attribute)

(aemter.models.Recht attribute)

G

get_associated_data() (in module historie templatetags.t_historie.get_associated_data)

get_default_history_user()
(aemter.models.HistoricalFunktion static method)

(aemter.models.HistoricalFunktionRecht static method)

(aemter.models.HistoricalOrganisationseinheit static method)

(aemter.models.HistoricalRecht static method)

(aemter.models.HistoricalUnterbereich static method)

(checklisten.models.HistoricalAufgabe static method)

(checklisten.models.HistoricalCheckliste static method)

(checklisten.models.HistoricalChecklisteAufgabe static method)

(checklisten.models.HistoricalChecklisteRecht static method)

get_funktionen() (in module checklisten.views)

get_history_type_display()
(aemter.models.HistoricalFunktion method)

(aemter.models.HistoricalFunktionRecht method)

(aemter.models.HistoricalOrganisationseinheit method)

(aemter.models.HistoricalRecht method)

(aemter.models.HistoricalUnterbereich method)

(checklisten.models.HistoricalAufgabe method)

(checklisten.models.HistoricalCheckliste method)

(checklisten.models.HistoricalChecklisteAufgabe method)

(checklisten.models.HistoricalChecklisteRecht method)

get_next_by_history_date()
(aemter.models.HistoricalFunktion method)

(aemter.models.HistoricalFunktionRecht method)

(aemter.models.HistoricalOrganisationseinheit method)

(aemter.models.HistoricalRecht method)

(aemter.models.HistoricalUnterbereich method)

(checklisten.models.HistoricalAufgabe method)

(checklisten.models.HistoricalCheckliste method)

(checklisten.models.HistoricalChecklisteAufgabe method)

(checklisten.models.HistoricalChecklisteRecht method)

get_perms() (in module checklisten templatetags.t_checklisten.get_perms)

get_previous_by_history_date()
(aemter.models.HistoricalFunktion method)

(aemter.models.HistoricalFunktionRecht method)

(aemter.models.HistoricalOrganisationseinheit method)

(aemter.models.HistoricalRecht method)

(aemter.models.HistoricalUnterbereich method)

(checklisten.models.HistoricalAufgabe method)

(checklisten.models.HistoricalCheckliste method)

(checklisten.models.HistoricalChecklisteAufgabe method)

(checklisten.models.HistoricalChecklisteRecht method)

`get_tasks()` (in `module`
`checklisten.templatetags.t_checklisten.get_tasks`)

H

`handle()` (`bin.management.commands.delete_old_historie.Command` method)

`help` (`bin.management.commands.delete_old_historie.Command` attribute)

`HistoricalAufgabe` (class in `checklisten.models`)

`HistoricalAufgabe.DoesNotExist`

`HistoricalAufgabe.MultipleObjectsReturned`

`HistoricalCheckliste` (class in `checklisten.models`)

`HistoricalCheckliste.DoesNotExist`

`HistoricalCheckliste.MultipleObjectsReturned`

`HistoricalChecklisteAufgabe` (class in `checklisten.models`)

`HistoricalChecklisteAufgabe.DoesNotExist`

`HistoricalChecklisteAufgabe.MultipleObjectsReturned`

`HistoricalChecklisteRecht` (class in `checklisten.models`)

`HistoricalChecklisteRecht.DoesNotExist`

`HistoricalChecklisteRecht.MultipleObjectsReturned`

`HistoricalFunktion` (class in `aemter.models`)

`HistoricalFunktion.DoesNotExist`

`HistoricalFunktion.MultipleObjectsReturned`

`HistoricalFunktionRecht` (class in `aemter.models`)

`HistoricalFunktionRecht.DoesNotExist`

`HistoricalFunktionRecht.MultipleObjectsReturned`

`HistoricalOrganisationseinheit` (class in `aemter.models`)

`HistoricalOrganisationseinheit.DoesNotExist`

`HistoricalOrganisationseinheit.MultipleObjectsReturned`

`HistoricalRecht` (class in `aemter.models`)

`HistoricalRecht.DoesNotExist`

`HistoricalRecht.MultipleObjectsReturned`

`HistoricalUnterbereich` (class in `aemter.models`)

`HistoricalUnterbereich.DoesNotExist`

`HistoricalUnterbereich.MultipleObjectsReturned`

`historie.templatetags.t_historie.get_associated_data`

`module`

`historie.templatetags.t_historie.to_class_name`
`module`

`historie.views`
`module`

`history` (`aemter.models.Funktion` attribute)

(`aemter.models.FunktionRecht` attribute)

(`aemter.models.Organisationseinheit` attribute)

(`aemter.models.Recht` attribute)

(`aemter.models.Unterbereich` attribute)

(`checklisten.models.Aufgabe` attribute)

(`checklisten.models.Checkliste` attribute)

(`checklisten.models.ChecklisteAufgabe` attribute)

(`checklisten.models.ChecklisteRecht` attribute)

`history_change_reason`

(`aemter.models.HistoricalFunktion` attribute)

(`aemter.models.HistoricalFunktionRecht` attribute)

(`aemter.models.HistoricalOrganisationseinheit` attribute)

(`aemter.models.HistoricalRecht` attribute)

(`aemter.models.HistoricalUnterbereich` attribute)

(`checklisten.models.HistoricalAufgabe` attribute)

(`checklisten.models.HistoricalCheckliste` attribute)

(`checklisten.models.HistoricalChecklisteAufgabe` attribute)

(`checklisten.models.HistoricalChecklisteRecht` attribute)

`history_date` (`aemter.models.HistoricalFunktion` attribute)

(`aemter.models.HistoricalFunktionRecht` attribute)

(`aemter.models.HistoricalOrganisationseinheit` attribute)

(`aemter.models.HistoricalRecht` attribute)

(`aemter.models.HistoricalUnterbereich` attribute)

(`checklisten.models.HistoricalAufgabe` attribute)

(`checklisten.models.HistoricalCheckliste` attribute)

(`checklisten.models.HistoricalChecklisteAufgabe` attribute)

(`checklisten.models.HistoricalChecklisteRecht` attribute)

`history_id` (`aemter.models.HistoricalFunktion` attribute)

(`aemter.models.HistoricalFunktionRecht` attribute)

(`aemter.models.HistoricalOrganisationseinheit` attribute)

(`aemter.models.HistoricalRecht` attribute)

(`aemter.models.HistoricalUnterbereich` attribute)

(`checklisten.models.HistoricalAufgabe` attribute)

(`checklisten.models.HistoricalCheckliste` attribute)

(`checklisten.models.HistoricalChecklisteAufgabe` attribute)

(`checklisten.models.HistoricalChecklisteRecht` attribute)

(checklisten.models.HistoricalAufgabe attribute)

(aemter.models.HistoricalOrganisationseinheit attribute)

(aemter.models.HistoricalRecht attribute)
(aemter.models.HistoricalUnterbereich attribute)
(checklisten.models.HistoricalAufgabe attribute)
(checklisten.models.HistoricalCheckliste attribute)
(checklisten.models.HistoricalChecklisteAufgabe attribute)
(checklisten.models.HistoricalChecklisteRecht attribute)

L

list() (in module historie.views)
loeschen() (in module checklisten.views)
login.views
module
logout_request() (in module login.views)

M

main_screen() (in module aemter.views)
(in module checklisten.views)
(in module login.views)
(in module mitglieder.views)
max_members (aemter.models.Funktion attribute)
(aemter.models.HistoricalFunktion attribute)
mitglied (checklisten.models.Checkliste attribute)
(checklisten.models.HistoricalCheckliste attribute)
mitglied_id (checklisten.models.Checkliste attribute)
(checklisten.models.HistoricalCheckliste attribute)
mitglied_laden() (in module mitglieder.views)
mitgliedamt_set (aemter.models.Funktion attribute)
mitgliedBearbeitenView() (in module mitglieder.views)
mitglieder.views
module
mitglieder_loeschen() (in module mitglieder.views)
mitgliedErstellenView() (in module mitglieder.views)
module
aemter.models
aemter.views
bin.management.commands.delete_old_historie
checklisten.models
checklisten.template_tags.t_checklisten.get_perms
checklisten.template_tags.t_checklisten.get_tasks
checklisten.views
historie.template_tags.t_historie.get_associated_data
historie.template_tags.t_historie.to_class_name

historie.views
login.views
mitglieder.views

N

next_record() (aemter.models.HistoricalFunktion property)
(aemter.models.HistoricalFunktionRecht property)
(aemter.models.HistoricalOrganisationseinheit property)
(aemter.models.HistoricalRecht property)
(aemter.models.HistoricalUnterbereich property)
(checklisten.models.HistoricalAufgabe property)
(checklisten.models.HistoricalCheckliste property)
(checklisten.models.HistoricalChecklisteAufgabe property)
(checklisten.models.HistoricalChecklisteRecht property)

O

objects (aemter.models.Funktion attribute)
(aemter.models.FunktionRecht attribute)
(aemter.models.HistoricalFunktion attribute)
(aemter.models.HistoricalFunktionRecht attribute)
(aemter.models.HistoricalOrganisationseinheit attribute)
(aemter.models.HistoricalRecht attribute)
(aemter.models.HistoricalUnterbereich attribute)
(aemter.models.Organisationseinheit attribute)
(aemter.models.Recht attribute)
(aemter.models.Unterbereich attribute)
(checklisten.models.Aufgabe attribute)
(checklisten.models.Checkliste attribute)
(checklisten.models.ChecklisteAufgabe attribute)
(checklisten.models.ChecklisteRecht attribute)
(checklisten.models.HistoricalAufgabe attribute)
(checklisten.models.HistoricalCheckliste attribute)
(checklisten.models.HistoricalChecklisteAufgabe attribute)
(checklisten.models.HistoricalChecklisteRecht attribute)
organisationseinheit (aemter.models.Funktion attribute)
(aemter.models.HistoricalFunktion attribute)
(aemter.models.HistoricalUnterbereich attribute)
(aemter.models.Unterbereich attribute)

Organisationseinheit (class in aemter.models)
Organisationseinheit.DoesNotExist
Organisationseinheit.MultipleObjectsReturned
organisationseinheit_id (aemter.models.Funktion attribute)
(aemter.models.HistoricalFunktion attribute)
(aemter.models.HistoricalUnterbereich attribute)
(aemter.models.Unterbereich attribute)

P

prev_record() (aemter.models.HistoricalFunktion property)
(aemter.models.HistoricalFunktionRecht property)
(aemter.models.HistoricalOrganisationseinheit property)
(aemter.models.HistoricalRecht property)
(aemter.models.HistoricalUnterbereich property)
(checklisten.models.HistoricalAufgabe property)
(checklisten.models.HistoricalCheckliste property)
(checklisten.models.HistoricalChecklisteAufgabe property)
(checklisten.models.HistoricalChecklisteRecht property)

R

recht (aemter.models.FunktionRecht attribute)
(aemter.models.HistoricalFunktionRecht attribute)
(checklisten.models.ChecklisteRecht attribute)
(checklisten.models.HistoricalChecklisteRecht attribute)
Recht (class in aemter.models)
Recht.DoesNotExist
Recht.MultipleObjectsReturned
recht_id (aemter.models.FunktionRecht attribute)
(aemter.models.HistoricalFunktionRecht attribute)
(checklisten.models.ChecklisteRecht attribute)
(checklisten.models.HistoricalChecklisteRecht attribute)
revert_url() (aemter.models.HistoricalFunktion method)
(aemter.models.HistoricalFunktionRecht method)
(aemter.models.HistoricalOrganisationseinheit method)
(aemter.models.HistoricalRecht method)
(aemter.models.HistoricalUnterbereich method)
(checklisten.models.HistoricalAufgabe method)

(checklisten.models.HistoricalCheckliste method)
(checklisten.models.HistoricalChecklisteAufgabe method)
(checklisten.models.HistoricalChecklisteRecht method)

S

save_without_historical_record()
(aemter.models.Funktion method)
(aemter.models.FunktionRecht method)
(aemter.models.Organisationseinheit method)
(aemter.models.Recht method)
(aemter.models.Unterbereich method)
(checklisten.models.Aufgabe method)
(checklisten.models.Checkliste method)
(checklisten.models.ChecklisteAufgabe method)
(checklisten.models.ChecklisteRecht method)
speichern() (in module mitglieder.views)
suchen() (in module mitglieder.views)

T

to_class_name() (in module
historie.template_tags.t_historie.to_class_name)
tostring() (aemter.models.Funktion method)

U

untbereich (aemter.models.Funktion attribute)
(aemter.models.HistoricalFunktion attribute)
Unterbereich (class in aemter.models)
Unterbereich.DoesNotExist
Unterbereich.MultipleObjectsReturned
untbereich_id (aemter.models.Funktion attribute)
(aemter.models.HistoricalFunktion attribute)
untbereich_set (aemter.models.Organisationseinheit attribute)

W

workload (aemter.models.Funktion attribute)
(aemter.models.HistoricalFunktion attribute)

Python Module Index

a

`aemter`

`aemter.models`

`aemter.views`

b

`bin`

`bin.management.commands.delete_old_historie`

c

`checklisten`

`checklisten.models`

`checklisten.templatetags.t_checklisten.get_perms`

`checklisten.templatetags.t_checklisten.get_tasks`

`checklisten.views`

h

`historie`

`historie.templatetags.t_historie.get_associated_data`

`historie.templatetags.t_historie.to_class_name`

`historie.views`

l

`login`

`login.views`

m

`mitglieder`

`mitglieder.views`