|  |
| --- |
| 12. April 2018, 17:00 Uhr |
| IPA Bericht |
| Jusuf Zera |

**Vorwort**

Inhalt

[1 Obligatorischer Teil 6](#_Toc511307908)

[1.1 Kurzfassung 6](#_Toc511307909)

[1.2 Ausgangslage 7](#_Toc511307910)

[1.3 Detaillierte Aufgabenstellung 7](#_Toc511307911)

[1.3.1 Login 8](#_Toc511307912)

[1.3.2 Benutzerprofil 8](#_Toc511307913)

[1.3.2.1 Stammdaten ändern 8](#_Toc511307914)

[1.3.2.2 Passwort ändern 8](#_Toc511307915)

[1.3.3 Testbenutzer für interne Registrierung erstellen 8](#_Toc511307916)

[1.3.4 Testing 9](#_Toc511307917)

[1.3.5 Dokumentation 9](#_Toc511307918)

[1.3.6 Design 9](#_Toc511307919)

[1.3.7 Versionierung 9](#_Toc511307920)

[1.3.8 Mittel und Methoden 10](#_Toc511307921)

[1.3.9 Vorkenntnisse 10](#_Toc511307922)

[1.3.10 Vorarbeiten 10](#_Toc511307923)

[1.3.11 Neue Lerninhalte 10](#_Toc511307924)

[1.3.12 Arbeiten in den letzten 6 Monaten 11](#_Toc511307925)

[1.4 Projektorganisation 12](#_Toc511307926)

[1.5 Zeitplan 13](#_Toc511307927)

[1.6 Arbeitsjournal 14](#_Toc511307928)

[1.6.1 Dienstag, 03. April 2018 14](#_Toc511307929)

[1.6.2 Mittwoch, 04. April 2018 15](#_Toc511307930)

[1.6.3 Donnerstag, 05. April 2018 16](#_Toc511307931)

[1.6.4 Freitag, 06. April 2018 17](#_Toc511307932)

[1.6.5 Montag, 09. April 2018 18](#_Toc511307933)

[1.6.6 Dienstag, 10. April 2018 19](#_Toc511307934)

[1.6.7 Mittwoch, 11. April 2018 20](#_Toc511307935)

[1.6.8 Donnerstag, 12. April 2018 21](#_Toc511307936)

[1.6.9 Freitag, 13. April 2018 22](#_Toc511307937)

[1.6.10 Montag, 16. April 2018 23](#_Toc511307938)

[2 Projektdokumentation 24](#_Toc511307939)

[2.1 Informieren 24](#_Toc511307940)

[2.1.1 Auftrag 24](#_Toc511307941)

[2.1.2 Vorgaben 25](#_Toc511307942)

[2.1.2.1 Python 25](#_Toc511307943)

[2.1.2.2 Django 25](#_Toc511307944)

[2.2 Planen 26](#_Toc511307945)

[2.2.1 Implementierung der Komponenten 26](#_Toc511307946)

[2.2.1.1 Hauptseite 26](#_Toc511307947)

[2.2.1.2 Login 26](#_Toc511307948)

[2.2.1.3 Profil-Seite 27](#_Toc511307949)

[2.2.1.4 Profil bearbeiten 27](#_Toc511307950)

[2.2.1.5 Passwort ändern 28](#_Toc511307951)

[2.2.1.6 Registrierung 28](#_Toc511307952)

[2.2.2 Vorgehensweise nach MTV-Pattern 28](#_Toc511307953)

[2.2.2.1 Models 29](#_Toc511307954)

[2.2.2.2 Templates 29](#_Toc511307955)

[2.2.2.3 Views 29](#_Toc511307956)

[2.2.3 Vorgehensweise nach Django-Administratoranwendung 30](#_Toc511307957)

[2.2.4 Problemanalyse 30](#_Toc511307958)

[2.2.4.1 Datenverlust 30](#_Toc511307959)

[2.2.4.2 Programmierprobleme 31](#_Toc511307960)

[2.2.4.3 Zeitprobleme 31](#_Toc511307961)

[2.2.4.4 Browserkompatibilität 31](#_Toc511307962)

[2.2.5 Sicherung 32](#_Toc511307963)

[2.2.5.2 Arbeitsplatz 32](#_Toc511307964)

[2.2.5.2 USB-Stick 32](#_Toc511307965)

[2.2.5.3 Github 32](#_Toc511307966)

[2.2.6 Soll/Ist-Vergleich der Systemlandschaft 33](#_Toc511307967)

[2.3 Entscheiden 34](#_Toc511307968)

[2.3.1 Vergleich der Vorgehensweisen 34](#_Toc511307969)

[2.3.2 Entscheidung der Vorgehensweisen 34](#_Toc511307970)

[2.4 Realisieren 35](#_Toc511307971)

[2.4.1 Voraussetzungen 35](#_Toc511307972)

[2.4.1.1 Projekt erstellen 35](#_Toc511307973)

[2.4.1.2 Applikation erstellen 37](#_Toc511307974)

[2.4.2 Gerüst der Webapplikation 38](#_Toc511307975)

[2.4.2.1 Bootstrap-Stylesheet 39](#_Toc511307976)

[2.4.3 Login 41](#_Toc511307977)

[2.4.3.1 Voraussetzungen 41](#_Toc511307978)

[2.4.3.2 Lokale Benutzer 41](#_Toc511307979)

[2.4.3.3 LDAP-Benutzer 41](#_Toc511307980)

[2.4.3.4 urls.py 42](#_Toc511307981)

[2.4.3.5 Login-Template 42](#_Toc511307982)

[2.4.3.6 E-Mail Backend 42](#_Toc511307983)

[2.4.3.7 Plausibilisierung der Eingaben 43](#_Toc511307984)

[2.4.3.8 CSRF-Sicherheiten 43](#_Toc511307985)

[2.4.3.9 Logout 44](#_Toc511307986)

[2.4.4 LDAP-Anbindung 44](#_Toc511307987)

[2.4.4.1 Django-LDAP-Workflow 45](#_Toc511307988)

[2.4.5 Profil anzeigen 45](#_Toc511307989)

[2.3.5.1 URL 45](#_Toc511307990)

[2.3.5.2 Funktion 45](#_Toc511307991)

[2.3.5.4 Profile-Template 46](#_Toc511307992)

[2.4.6 Profil bearbeiten 47](#_Toc511307993)

[2.4.6.1 UserModel erweitern 47](#_Toc511307994)

[2.4.6.2 Django-Migrations 47](#_Toc511307995)

[2.4.6.3 OneToOne-Verknüpfung 48](#_Toc511307996)

[2.4.6.4 URL 48](#_Toc511307997)

[2.4.6.5 Form 49](#_Toc511307998)

[2.4.6.6 Funktion 49](#_Toc511307999)

[2.4.6.7 Profil bearbeiten-Template 50](#_Toc511308000)

[2.4.6.8 E-Mail-Adresse ändern-Workflow 52](#_Toc511308001)

[2.4.7 Django-LDAP-Workflow 52](#_Toc511308002)

[2.4.7.1 Python-Operations 52](#_Toc511308003)

[2.4.7.2 Modify\_Replace 53](#_Toc511308004)

[2.4.7.3 Ursachen 53](#_Toc511308005)

[2.4.7.4 Massnahmen 54](#_Toc511308006)

[2.4.7.5 Folgen 54](#_Toc511308007)

[2.4.8 Passwort ändern 54](#_Toc511308008)

[2.4.8.1 URL 54](#_Toc511308009)

[2.4.8.2 PasswordChangeForm 54](#_Toc511308010)

[2.4.8.3 Funktion 55](#_Toc511308011)

[2.4.8.4 Error-Anzeigen 56](#_Toc511308012)

[2.4.8.5 Passwort-Voraussetzungen 57](#_Toc511308013)

[2.4.9 Registrierung (Weiterentwicklung) 57](#_Toc511308014)

[2.4.9.1 URL 57](#_Toc511308015)

[2.4.9.2 UserCreationForm 58](#_Toc511308016)

[Code: Meta-Daten des Benutzers 58](#_Toc511308017)

[2.4.9.3 Funktion 58](#_Toc511308018)

[2.4.9.4 Error-Anzeigen 59](#_Toc511308019)

[2.5 Kontrollieren 60](#_Toc511308020)

[2.5.1 Testfälle 60](#_Toc511308021)

[2.6 Auswerten 60](#_Toc511308022)

[2.7 Abschlussbericht 60](#_Toc511308023)

[2.8 Quellenverzeichnis 60](#_Toc511308024)

[2.9 Glossar 60](#_Toc511308025)

# 1 Obligatorischer Teil

## Kurzfassung

Die Schweizerische Bankiervereinigung befasst sich intensiv mit der Digitalisierung und hat sich daher entschieden, das Projekt „M17“ (Migration 2017) zu lancieren. Die heutigen Systeme wurden von der Bankiervereinigung vor 14 Jahren selbst entwickelt und entsprechen nicht dem Standard neuster Technologien. Ausserdem sind alle Module miteinander verbunden, sodass die Unabhängigkeit der einzelnen Systeme nicht gewährleistet ist. Daher hat sich die Geschäftsleitung entschieden, die Systemlandschaft von Grund auf neu aufzubauen.

Die Systemlandschaft wird demnach modular und nicht zentral erstellt. Dafür verantwortlich sein, wird die Autorisierungsplattform einer externen Firma, welche die Berechtigungen für die einzelnen Module regelt. Den Benutzerinnen und Benutzer wird es nach wie vor möglich sein, ihre Daten selbst zu mutieren. Das Selbstverwaltungs-Tool der externen Firma entspricht aber nicht den Anforderungen der Schweizerischen Bankiervereinigung. Somit wird dieser Teil des Projekts „M17“ von mir, Jusuf Zera, entwickelt.

Das Ergebnis wird eine Webapplikation sein, die mit der Datenbank (LDAP-Datenbank) der Autorisierungsplattform verbunden ist. Die Benutzerinnen und Benutzer werden sich mit ihrer E-Mail-Adresse und ihrem Passwort anmelden, um anschliessend ihr Profil mit allen Daten zu sehen. Änderbare Daten können auf einer separaten Seite mutiert und gespeichert werden. Bei Fehleingaben wird den Benutzerinnen und Benutzer ersichtlich, was angepasst werden muss, um den Prozess des Speicherns abzuschliessen. Ausserdem kann das Passwort beliebig geändert werden, nachdem der Benutzer die Voraussetzungen erfüllt hat. Die Abmeldung erfolgt über einen einfachen One-Click. Zur Weiterentwicklung nach der IPA wird eine Registrierungsseite erstellt, die Benutzer in einer internen Datenbank erstellt und nicht in der LDAP-Datenbank der externe Firma. Grund dafür ist, dass die Registrierung seitens Geschäftsleitung direkt über die Autorisierungsplattform erfolgen soll.

## 1.2 Ausgangslage

Der Wandel in die Digitalisierung ist ein wichtiger Schritt in der Zukunft der Schweizerischen Bankiervereinigung (SBVg). Aus diesem Grund wurde 2017 das Projekt „M17“ (Migration 2017) initialisiert. Das Projekt beinhaltet die Migration der internen und externen Systemlandschaft. Die gegenwärtig eingesetzten Systeme entsprechen nicht mehr den neuesten Technologien, zudem ist die Usability der Bewirtschaftungssoftware nicht mehr gewährleistet. Die Struktur des Projekts „M17“ ist modular aufgebaut, damit alle Systeme unabhängig voneinander funktionieren oder allenfalls ersetzt werden können. Die jetzige Software wurde im Jahre 2006 von der SBVg selbst entwickelt.   
Die neue Systemlandschaft beinhaltet u.a. eine webbasierte Collaboration-Plattform, auf der Gremienmitglieder als Vertretung seitens Banken interaktiv zusammenarbeiten. Dazu hat die SBVg bereits jetzt eine Portallösung aufgeschaltet, auf der berechtige Personen (vorzugsweise Bankmitarbeitende) Zugriff haben und entsprechende Informationen vorfinden. Die Portallösung soll im Rahmen des Projekts M17 erneuert werden. Eine Komponente davon bildet die Benutzerverwaltung, d.h. den berechtigen Personen werden entsprechende Berechtigungen erteilt. Um inskünftig die Personen, gegenwärtig deren 22‘000, zu verwalten, wird eine Autorisierungsplattform entwickelt, die die Berechtigungen aller Systeme steuert. Bei der von der SBVg verwendeten Autorisierungsplattform entspricht das Selbstadministrations-Tool nicht den Anforderungen der SBVg. Die fehlenden Komponenten werden von der SBVg entwickelt.

## 1.3 Detaillierte Aufgabenstellung

Das Resultat der IPA soll eine webbasierte Selbstverwaltung für Benutzer beinhalten. Die Änderungen werden in der Mitarbeiter-Datenbank (LDAP-DB) gespeichert. Die Selbstverwaltung der Benutzenden muss an diese Datenbank angebunden werden und kommunizieren können. Die Mutationen werden durch das Speichern in die LDAP-DB übernommen.  
  
Ziel des Projekts ist eine benutzerfreundliche Webapplikation, die auf allen Endgeräten wie Smartphones, Tablets und Computer kompatibel ist und auf dem vorgegebenen Design unserer Kommunikationsabteilung basiert.

User-Modell:  
Die Webapplikation ist für Portaluser der Banken gedacht, die Ihre Daten selbständig und jederzeit ändern wollen. Sie sind in unserer LDAP-DB aufgelistet und Mutationen werden dort direkt übernommen. In der Applikation selbst werden keine Berechtigungen gegeben, welche Module etc. genutzt werden können, da dies von der Autorisierungsplattform gemacht wird.   
Es handelt sich dabei um ein Selfservice-Formular, das jedem Benutzer ermöglicht, die gleichen Daten zu ändern (z.B. Name, Vorname, Strasse). Durch das Login wird sichergestellt, dass jeder Benutzer nur seine eigenen Daten ändern kann. Auf der Profil-Seite sind alle Daten ersichtlich, aber auf der Profil Bearbeiten-Seite nur die Daten, die der Benutzer mutieren kann.

Umfang der Arbeit:

* Startseite mit Login
* Profil-Seite, die alle Daten des Benutzers anzeigt
* Profil bearbeiten-Seite auf der die Stammdaten des Benutzers verändert werden können.
* Passwort ändern-Seite auf dem der Benutzer sein altes Passwort durch ein Neues ersetzen kann.
* Registrierungsseite für Testbenutzer und Weiterentwicklung (nur für Administratoren)

Die Benutzerverwaltung besteht aus folgenden Komponenten:

### 1.3.1 Login

Beim Anmelden muss der Benutzer zwei Felder ausfüllen (E-Mail-Adresse und Passwort). Die Felder werden vom System auf ihre Korrektheit und Übereinstimmung verifiziert. Bei einer Falscheingabe, wird eine Fehlermeldung angezeigt. Mit einem einfachen One-Click meldet sich der Benutzer wieder ab.

### 1.3.2 Benutzerprofil

Nach der Anmeldung wird der Benutzer auf sein Profil weiteregeleitet. 

#### 1.3.2.1 Stammdaten ändern

Auf der „Profil bearbeiten“-Seite kann der Benutzer seine Stammdaten wie Name, Adresse etc. beliebig mutieren. Die Änderungen werden beim Speichern in die LDAP-DB übernommen.

#### 1.3.2.2 Passwort ändern

Auf der Seite „Passwort ändern“ kann der Benutzer sein Passwort ändern. Dazu muss er sein altes Passwort und das neue Passwort doppelt eingeben. Nach einer erfolgreichen Überprüfung des alten Passworts wird das neue Passwort beim Speichern in die DB übernommen. 

### 1.3.3 Testbenutzer für interne Registrierung erstellen

Um die Benutzerveränderungen und Validierung der Felder zu testen, wird zusätzlich eine Registrierungsseite erstellt. Der Benutzer fügt seine E-Mai-Adresse, Name, Vorname und Passwort (2x) etc. ein. Es wird ein neuer Benutzer erstellt. Dieser wird jedoch nicht in der LDAP-DB abgespeichert, sondern in einer SQLite3-DB, um Konflikte mit den Benutzern in der LDAP-DB zu vermeiden. Die Portaluser der Banken werden sich nicht über die Webapplikation registrieren. Dafür ist erneut die Autorisierungsplattform zuständig. Die Webapplikation kann aber nach der IPA weiterentwickelt werden, sodass sich zukünftig Benutzer ihr Konto direkt auf der Webseite erstellen können.   
Diese Funktion hilft sowohl den Usern als auch den Administratoren Probleme und Fehler zu beseitigen und eine einwandfreie Anmeldung und Mutation der Daten zu gewährleisten.   
Die Registrierungsseite wird auf der Startseite zu finden sein und ein benutzerfreundliches Formular enthalten.

### 1.3.4 Testing

Das Testen der Funktionen ist ein wichtiger Teil des Projekts. Um die Funktionalität prüfen zu können, werden die Tests folgend ablaufen:  
  
- Eingabe  
- Erwartetes Resultat  
- Tatsächliches Resultat  
  
In der Webapplikation selbst werden Fehleingaben (z.B. nicht übereinstimmende Passwörter) mit rot-markierten Error-Anzeigen hervorgehoben.  
Die Testfälle werden übersichtlich in Tabellen mit Titeln und Nummern dargestellt. Datum, Zeit und Testperson werden ebenfalls eingetragen, um alles korrekt zu protokollieren. Um die Kompatibilität zu gewährleisten, werden die Tests in verschieden Browsern (Google Chrome, Mozilla Firefox, Internet Explorer) ausgeführt. 

### 1.3.5 Dokumentation

Die IPA-Dokumentation wird auf Basis der Projektmanagement-Methode IPERKA (Informieren, Planen, Entscheiden, Realisieren, Kontrollieren, Ausführen) strukturiert. Die Agilität der Methode steuert unter anderem dazu bei, das Projekt erfolgreich umzusetzen. Ausserdem werden die Testfälle der Applikation eingefügt. Zudem geht die Dokumentation darauf ein, mit welchen zusätzlichen Optionen die Applikation nach der Arbeit erweitert werden könnte. 

### 1.3.6 Design

Das Design soll die SBVg repräsentieren und gleich strukturiert sein, wie alle anderen Applikationen. Aus diesem Grund gibt die Kommunikationsabteilung das Grunddesign vor. Um das Design technisch zu implementieren, wird eine Bootstrap-Vorlage verwendet. Damit die Applikation auf allen Endgeräten benutzerfreundlich funktionieren soll, wird sie responsive programmiert.

### 1.3.7 Versionierung

Zur Sicherung des Projekts wird ein GIT-Repository erstellt.

### 1.3.8 Mittel und Methoden

* Python
* Django-Framework
* HTML
* CSS
* Bootstrap
* Atom (Editor)
* UCS - Univention Corporate Server mit LDAP-DB
* GitHub

### 1.3.9 Vorkenntnisse

* HTML & CSS: 3 Jahre
* Python: 3 Monate
* Django: 3 Monate
* LDAP-DB: Basiswissen

### 1.3.10 Vorarbeiten

* Bereitstellung der Serverumgebung (UCS-Server, LDAP-DB, Webserver etc.)
* Vorlage für die Dokumentation der IPA, damit schon ab dem ersten Tag dokumentiert werden kann
* Repository erstellen
* Arbeitsplatz einrichten

### 1.3.11 Neue Lerninhalte

Um Erfahrungen mit der Programmiersprache Python zu sammeln, durfte der Kandidat sich in die Thematik einlesen und vieles durch Üben lernen. Ausserdem wird für die Applikation das Django-Framework verwendet, das neu kennengelernt werden musste. Wie eine LDAP-DB genau aufgebaut ist, gehörte ebenfalls zum Lernprozess, damit das Grundwissen vorhanden ist.

Da die Benutzerverwaltung ein Teil des Projekts „M17“ ist, wurde dem Kandidat ersichtlich, wie man sich bei der Mitarbeit eines solchen Projekts verhält. Dies beinhaltet auch das selbstständige Arbeiten und Evaluieren verschiedener Software.   
Als Quellen standen alle Mitarbeitenden zur Verfügung, die sowohl technisch als auch im Alltag unterstützen. Selbstverständlich wurde auch Hilfe aus dem Internet und der Literatur geholt, um den Radius des Wissens zu vergrössern.

### Arbeiten in den letzten 6 Monaten

Zu den hauptsächlichen Arbeiten gehörten:

* Mitarbeit im Projekt „M17“
* Administration der Datenbank von der SBVg
* Mitarbeit beim Initialisieren der Autorisierungsplattform
* Unterstützung bei diversen Programmierarbeiten
* Evaluation diverser Software
* Unterstützung bei Supportanfragen

Der erste Auftrag war ein 2-tägiger Workshop mit einer externen Firma, die bei uns vor Ort war. Ziel des Workshops war zum einen eine Testumgebung der Autorisierungsplattform aufzusetzen und zum anderen die Anforderungen unsererseits an das LDAP zu definieren, damit die externe Firma das endgültige LDAP-Design erstellen kann.   
  
Der zweite Auftrag ist die Mitarbeit im Projekt „M17“, das laufend weitergeführt wird. Dazu gehört die Strukturdarstellung der neuen Systemlandschaft mit dem Programm Visio, die aufzeigt, wie alle Systeme inskünftig miteinander kommunizieren werden. In diesem Zusammenhang durfte der Kandidat auch das Evaluieren von Software und Anbietern vornehmen.

Tools, die verwendet wurden:

* Word
* Excel
* Powerpoint
* Visio
* UCS – Univention Corporate Server

## Projektorganisation

Als Auftraggeber des Projekts wird die Unternehmensleitung bestimmt, da die Benutzerverwaltung ein Teil des Projekts „M17“ ist. Der Lenkungsausschluss des gesamten Projekts besteht aus Herr Nicolas Hug (Leiter Operations), Herr Lionel Albrecht (Leiter ICT) und Frau Alexandra Arni (Leiterin Online Solutions).

Die technische Projektleitung wird von Herr Detlef Nünninghoff übernommen. Als technischer Mitarbeiter bin ich, Jusuf Zera, für die Planung, Steuerung und Umsetzung der Arbeit verantwortlich, damit keine ungeplanten Komplikationen entstehen.

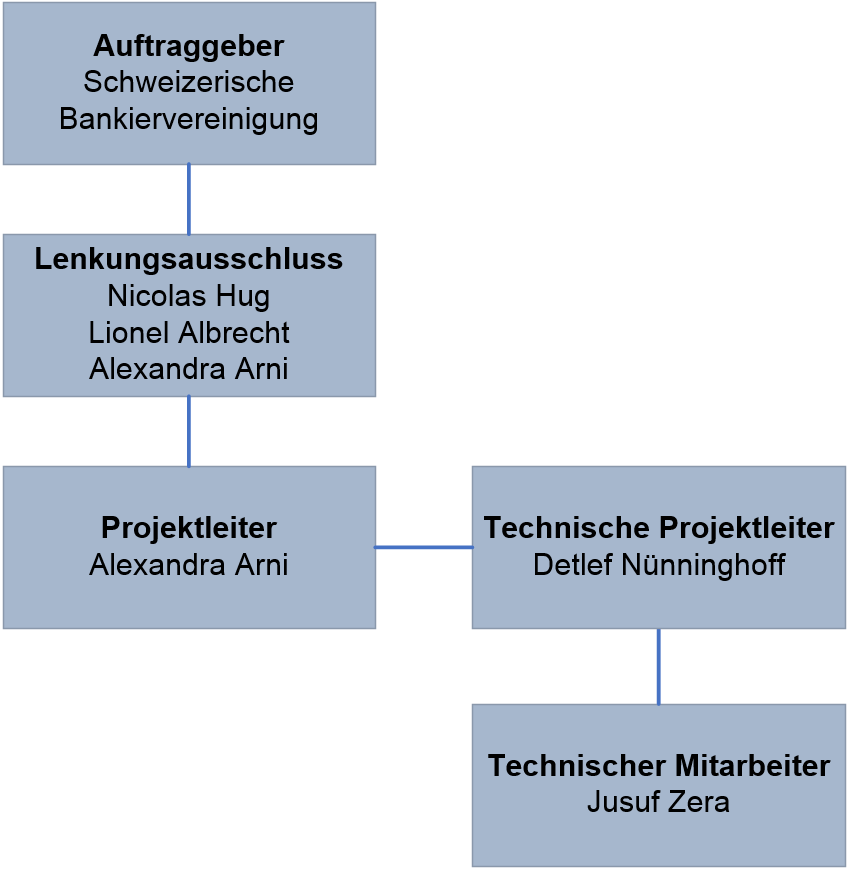


Abbildung: Projektorganisation

Diese Projektorganisation wird auch als „autonome Projektorganisation“ bezeichnet und bietet viele Vorteile unter anderem eindeutig zugewiesene Verantwortlichkeiten, die einfachen Kommunikationswege und das geringe Konfliktpotential der Aufgabenverteilung. Als Kontaktperson kann ich mich ohne Umwege an die Projektleitung wenden, um allfällige Fragen zu klären und im Zeitplan zu bleiben.

Da die Applikation von mir realisiert wird, kann es zu keinen grösseren Problemen innerhalb der Organisation kommen. Durch die gute Zeitplanung beuge ich vor, auf keine Überlastungen während der Arbeitszeit zu stossen.

## 1.5 Zeitplan

Abbildung: Zeitplan der IPA

## 1.6 Arbeitsjournal

### 1.6.1 Dienstag, 03. April 2018

**Tagesziele**

* Erste Dokumentationen erarbeiten
* Informationen bezüglich Anforderungen holen
* Anforderungen festlegen
* Aufbau des Programms erarbeiten

**Nicht erledigte Arbeiten**Keine

**Ungeplante Arbeiten**Keine

**Unterstützung Dritter**Keine

**Zeitplan**  
Der Zeitplan wurde eingehalten

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr | * Erste Dokumentationen erarbeitet * Informationen bezüglich Anforderungen geholt |
| 12:45 Uhr – 14:45 Uhr | * Informationen festgelegt |
| 14:45 Uhr – 16:45 Uhr | * Aufbau des Programms erarbeitet |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.2 Mittwoch, 04. April 2018

**Tagesziele**

* Soll/Ist-Vergleich darstellen
* Planung der Applikation abschliessen
* Vorgehensweise auswählen
* Entscheide festlegen

**Nicht erledigte Arbeiten**Keine

**Ungeplante Arbeiten**Ich habe die Planungsphase mit der „Implementierung der Komponenten“ erweitert.

**Unterstützung Dritter**Herr Nünninghoff hat mir gezeigt, wo man den Ist-Status der Systemlandschaft findet.

**Zeitplan**  
Der Zeitplan wurde eingehalten. Durch überschüssige Zeit in der Entscheidungsphase konnte ich noch die Planungsphase erweitern.

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr | * Soll/Ist-Vergleich darstellen * Planung der Applikation abschliessen |
| 12:45 Uhr – 14:00 Uhr | * Vorgehensweise vergleichen |
| 14:00 Uhr – 15:00 Uhr | * Erster Besuchstag Experten |
| 15:00 Uhr – 17:45 Uhr | * Vorgehensweise auswählen * Entscheide festlegen * Planungsphase: Implementierung der Komponenten |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.3 Donnerstag, 05. April 2018

**Tagesziele**

* Grundstruktur der Webseite erstellen
* Login (mit LDAP-Benutzerdaten) erstellen
* Login dokumentieren

**Nicht erledigte Arbeiten**LDAP-Anbindung wird durch das Erstellen eines lokalen Benutzers auf den zweiten Tag der Realisierungsphase verschoben, da es sinnvoller ist die LDAP-Anbindung mit der Erweiterung des UserModels zu verbinden.

**Ungeplante Arbeiten**Die Voraussetzungen eines Django-Projekts wurden erläutert, bevor ich mit dem Programmieren angefangen habe.

**Unterstützung Dritter**Keine

**Zeitplan**  
Der Zeitplan wurde eingehalten. Jedoch wurden zusätzliche und nötige Arbeiten eingefügt, sodass die Erstellung der Grundstruktur verzögerte. Dies hat bisher keinen Einfluss auf den Zeitplan.

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 10:00 Uhr | * Soll/Ist-Vergleich darstellen |
| 10:00 Uhr – 12:00 Uhr | * Grundstruktur der Webseite erstellen |
| 13:00 Uhr – 17:00 Uhr | * Login erstellen (lokale User) * Login dokumentieren |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.4 Freitag, 06. April 2018

**Tagesziele**

* LDAP-Anbindung herstellen
* Anmeldung mit LDAP-Zugangsdaten
* Benutzerdaten nach Login anzeigen

**Nicht erledigte Arbeiten**Keine

**Ungeplante Arbeiten**Ich habe Recherchen über die Anbindung an einen LDAP-Server in Django durchgeführt, da dieser Schritt erforderlich war, um weiterarbeiten zu können.

**Unterstützung Dritter**Keine

**Zeitplan**  
Durch das Erstellen und der Recherche der LDAP-Anbindung verzögerte sich das Login der LDAP-Benutzer um zwei Stunden. Die Projektleitung wurde informiert und gab vor, dass weiterhin am Projekt gearbeitet werden kann.

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr | * LDAP-Anbindung herstellen * Anmeldung mit LDAP-Zugangsdaten |
| 13:00 Uhr – 17:00 Uhr | * Benutzerdaten nach Login anzeigen |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.5 Montag, 09. April 2018

**Tagesziele**

* Lokaler Benutzer kann Daten mutieren
* Dokumentieren von Profil anzeigen und mutieren

**Nicht erledigte Arbeiten**LDAP-Mutationen werden am 10. April 2018 erstellt und dokumentiert.

**Ungeplante Arbeiten**Keine

**Unterstützung Dritter**Keine

**Zeitplan**  
Durch schnelles Arbeiten wurde die Verzögerung von zwei Stunden wieder aufgeholt. Der Zeitplan wurde eingehalten und das Projekt kann weiterhin realisiert werden. Das Dokumentieren der LDAP-Mutationen wird am 10. April 2018 abgeschlossen.

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr | * Lokaler Benutzer kann Daten mutieren |
| 13:00 Uhr – 17:00 Uhr | * Dokumentieren von Profil anzeigen und mutieren |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.6 Dienstag, 10. April 2018

**Tagesziele**

* Benutzerdaten ändern können
* Dokumentieren von Benutzerdaten anzeigen und mutieren
* Benutzer kann Daten sehen und mutieren
* Erste Realisierung der Passwort ändern-Funktion

**Nicht erledigte Arbeiten**Keine

**Ungeplante Arbeiten**Dokumentieren des Django-LDAP-Workflows. Der Workaround für das Zurückschreiben der LDAP-Datenbank wurde erstellt und dokumentiert. Die Folgen werden durch die begrenzte Zeit auf den 11. April 2018 verlegt. Diese werden mit Herr Nünninghoff besprochen.

**Unterstützung Dritter**Keine

**Zeitplan**  
Der Zeitplan wurde eingehalten. Die Projektleitung wurde darüber informiert, dass Django über die LDAP-Datenbank authentifizieren und die Daten auslesen, aber nicht zurückschreiben kann. Die Applikation kann weiterhin realisiert werden.

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr | * Benutzerdaten ändern können * Dokumentieren von Benutzerdaten anzeigen und mutieren * Benutzer kann Daten sehen und Mutieren |
| 13:00 Uhr – 17:00 Uhr | * Dokumentieren des Django-LDAP-Workflows. * Erste Realisierung der Passwort ändern-Funktion |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.7 Mittwoch, 11. April 2018

**Tagesziele**

* Passwort ändern-Funktion fertig implementieren
* Dokumentieren der Passwort ändern-Funktion
* Erste Realisierung der Registrierung-Funktion

**Nicht erledigte Arbeiten**Keine

**Ungeplante Arbeiten**Die Folgen des Django-LDAP-Workflows wurden dokumentiert.

**Unterstützung Dritter**Da Django nicht in ein LDAP-Verzeichnis zurückschreiben kann, wurden die Folgen mit Herr Nünninghoff besprochen.

**Zeitplan**  
Der Zeitplan wurde eingehalten.

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 10:00 Uhr | * Folgen des Django-LDAP-Workflows dokumentieren * Passwort ändern-Funktion fertig implementieren |
| 10:00 Uhr – 12:00 Uhr | * Dokumentieren der Passwort ändern-Funktion |
| 13:00 Uhr – 15:00 Uhr | * Dokumentieren der Passwort ändern-Funktion |
| 15:00 Uhr – 17:00 Uhr | * Erste Realisierung der Registrierung-Funktion |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.8 Donnerstag, 12. April 2018

**Tagesziele**

* Registrierung für Weiterentwicklung erstellen
* Registrierung dokumentieren

**Nicht erledigte Arbeiten**

* Design anpassen

**Ungeplante Arbeiten**Keine

**Unterstützung Dritter**Keine

**Zeitplan**  
Das Implementieren der Registrierung hat eine Verzögerung von zwei Stunden verursacht. Das Design wird demnach am 13. April 2018 angepasst und dokumentiert.

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr | * Registrierung für Weiterentwicklung erstellen |
| 13:00 Uhr – 15:00 Uhr | * Registrierung für Weiterentwicklung erstellen |
| 15:00 Uhr – 17:00 Uhr | * Registrierung dokumentieren |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.9 Freitag, 13. April 2018

**Tagesziele**



**Nicht erledigte Arbeiten**

**Ungeplante Arbeiten**

**Unterstützung Dritter**

**Zeitplan**

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr |  |
| 13:00 Uhr – 17:00 Uhr |  |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

### 1.6.10 Montag, 16. April 2018

**Tagesziele**



**Nicht erledigte Arbeiten**

**Ungeplante Arbeiten**

**Unterstützung Dritter**

**Zeitplan**

**Zeitaufwand**

|  |  |
| --- | --- |
| Dauer | Arbeit |
| 08:00 Uhr – 12:00 Uhr |  |
| 13:00 Uhr – 17:00 Uhr |  |

Ort, Datum

Unterschrift Fachvorgesetzte Unterschrift Kandidat

# 2 Projektdokumentation

## 2.1 Informieren

### 2.1.1 Auftrag

Der Auftrag besteht darin, das Projekt „M17“ mit meiner Komponente im Rahmen der IPA zu vervollständigen. Die Benutzerverwaltung wird es allen Anwendern möglich machen, ihre Daten selbstständig und beliebig zu mutieren, sobald diese sich über das Login angemeldet haben. Dabei orientiere ich mich an die jetzige Portallösung auf der Hauptseite der Schweizerischen Bankiervereinigung.



Abbildung: Login der SBVg

Das Login wird auch künftig gleich aufgebaut. Da jeder Benutzer nur eine E-Mail-Adresse pro Konto besitzt, muss man sich mit der E-Mail-Adresse und dem dazugehörigen Passwort anmelden. Einen Benutzernamen wird es nach wie vor nicht geben, da die E-Mail-Adresse als solche fungiert.



Abbildung: Mutation der Personendaten im Portal der SBVg

Bei der Benutzerverwaltung sind die einzelnen Felder und die dazu gehörigen Anforderungen je Feld zu beachten. Jedes Eingabefeld wird in der künftigen Webapplikation so definiert, dass die Benutzer ohne Überlegung wissen, was genau eingegeben werden muss.

### 2.1.2 Vorgaben

Dadurch, dass die Systemlandschaft komplett neu aufgebaut wird, müssen an bestimmte Vorgaben eingehalten werden.

#### 2.1.2.1 Python

Die Programmiersprache Python wurde aus logischen Gründen ausgewählt. Der grösste Teil der neuen Systeme basiert auf Python, somit wurde festgelegt, dass die Komponente in der gleichen Programmiersprache in die Systemlandschaft einpflegt werden soll. Da Python schnelle Programmierung möglich macht (wenig Code), kann der Zeitplan gut eingehalten werden.

Vorteile:

* Neue Systeme der SBVg verwenden Python
* Objektorientiert
* Modern
* Lesbar
* Schnelles Programmieren möglich (Zeitplan kann eingehalten werden)
* Ideal für Webentwicklung
* Cross-Plattform (Windows, Linux, MacOS)

#### 2.1.2.2 Django

Die Nutzung eines Frameworks soll die Webentwicklung vereinfachen. Dabei entscheide ich mich für das Django-Framework, da die Benutzerverwaltung eine Webapplikation wird und Django für die Webentwicklung geeignet ist. Django ist ein Python-Framework, was die Kompatibilität mit der Programmiersprache gewährleistet.

Vorteile:

* Viele Pakete
* Aktive Weiterentwicklung
* Sicherheit
* Sehr gut dokumentiert

## 2.2 Planen

In der Planungsphase werden hauptsächlich die Reihenfolge der Implementierung und die verschiedenen Vorgehensweisen erläutert. Dabei geht es nicht um die Erstellung der Webapplikation, sondern um die sukzessive Art, die Applikation zu realisieren.

Um bei überraschenden Situationen während der Realisierungsphase reagieren zu können, werden Problemanalysen erstellt.

Ein Soll/Ist-Vergleich soll darlegen, wo die Benutzerverwaltung im Projekt „M17“ im Vergleich zur jetzigen Situation implementiert wird.

### 2.2.1 Implementierung der Komponenten

Die Benutzerverwaltung gliedert sich aus verschiedenen Komponenten, die in einer bestimmten Reihenfolge implementiert werden müssen, da ansonsten Abläufe nicht richtig funktionieren.

Beispiel:

* Benutzer kann Daten nicht mutieren, ohne sich vorher über ein Login anzumelden.

#### 2.2.1.1 Hauptseite

Die Hauptseite wird als erste Komponente entwickelt. Diese bietet eine Basis für den Inhalt, also den Komponenten, die implementiert werden. Zum Abschluss der Realisierung wird das Design angepasst.

#### 2.2.1.2 Login

Das Login wird als erste Komponente in die Hauptseite eingepflegt. Die Benutzer werden sich über die Login-Maske mit der E-Mail-Adresse und ihrem Passwort anmelden.

Falls ein Benutzer eines der Felder nicht ausfüllt, wird diesem klar gemacht, dass jenes Feld erforderlich ist.

Bei Fehleingaben wird eine einfache Error-Anzeige ersichtlich, dass E-Mail-Adresse oder Passwort falsch eingegeben wurde. Um die Konten zu schützen, wird nicht spezifisch erwähnen, ob E-Mail-Adresse oder Passwort nicht korrekt sind.

#### 2.2.1.3 Profil-Seite

Nach der Anmeldung wird man auf die Profil-Seite weitergeleitet. Diese zeigt alle Daten an, die ausgelesen werden.

* Anrede
* Titel
* Name
* Vorname
* Geburtsdatum
* E-Mail
* Telefon
* Fax
* Mobile
* Abteilung
* Rang
* Tätigkeitsgebiet
* Institut

#### 2.2.1.4 Profil bearbeiten

Über einen Button kommt man auf die Seite, auf welche man die Daten mutieren kann. Wichtig ist dabei, welche Felder nicht mutabel, erforderlich und nicht erforderlich sind:

|  |  |
| --- | --- |
| Feld | Anforderung |
| Anrede | Erforderlich |
| Titel | Nicht erforderlich |
| Name | Erforderlich |
| Vorname | Erforderlich |
| Geburtsdatum | Nicht erforderlich |
| E-Mail | Nicht mutabel |
| Telefon | Nicht erforderlich |
| Fax | Nicht erforderlich |
| Mobile | Erforderlich |
| Abteilung | Nicht erforderlich |
| Rang | Nicht erforderlich |
| Tätigkeitsgebiet | Nicht erforderlich |
| Institut | Nicht erforderlich |

Die erforderlichen Felder werden schon vor der Eingabe durch ein „\*“ gekennzeichnet.

Die Benutzer werden die abgeänderten Daten über einen Button speichern. Wenn alle Felder gültig sind, wird man auf das Profil weirtergeleitet.

Auch beim Bearbeiten des Profils, gilt das gleiche Prinzip, wie beim Login. Erforderliche Felder müssen ausgefüllt sein, um das Speichern zu ermöglichen.

#### 2.2.1.5 Passwort ändern

Die Änderungsfunktion des Passworts wird nach der Anmeldung in der Navigation zu finden sein. Der Benutzer muss sein altes Passwort und zur Authentifizierung das neue Passwort doppelt eingeben. Wenn die Felder gültig sind, wird das neue Passwort gespeichert.

Die Vorgaben des neuen Passworts werden auf der Seite aufgelistet, damit der Benutzer im Vorfeld weiss, wie stark die Anforderungen sind. Somit ist die Benutzerfreundlichkeit gewährleistet.

Bei Fehlangaben wird dem Anwender genau angezeigt, ob der Fehler beim Eingeben des alten oder des neuen Passworts liegt.

Werden die Felder gültig ausgefüllt und bestätigt, wird der Benutzer auf eine Bestätigungsseite weitergeleitet. Über einen Button kann das Profil wieder angezeigt werden.

#### 2.2.1.6 Registrierung

Die Registrierung wird an letzter Stelle implementiert, da die Prioritäten auf die oben genannten Komponenten liegen. Über ein Formular wird der Benutzer seine Daten eintragen und bestätigen. Falls alle Felder gültig sind, wird man auf sein Profil weitergeleitet.

Die erforderlichen Felder werden durch ein „\*“ angezeigt.

Da die Registrierung ein Passwort verlangt (doppelt eingeben), werden bei Fehleingaben oder unzulässigen Passwörtern eine Error-Anzeige dargestellt.

### 2.2.2 Vorgehensweise nach MTV-Pattern

Django basiert auf einen modifizierten Model-View-Controller (MVC). MVC ist ein Software-Design-Pattern, das eine Applikation in drei miteinander verbundenen Teile aufteilen soll.

In Django ist diese folgendermassen aufgebaut:

* Model: Schnittstelle zur Datenbank
* Template: Schnittstelle zwischen Benutzer und der Django-Applikation
* View: Verwaltet Daten, Anwendungslogik und Messaging

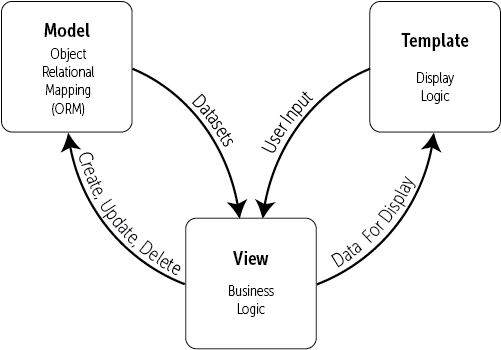
Es handelt sich hierbei also um ein MTV-Pattern:

Abbildung: Model-Template-View (MTV) Pattern in Django

#### 2.2.2.1 Models

Django-Models bieten ein Object Relational Mapping (ORM) für die zugrunde liegende Datenbank. ORM ist eine leistungsfähige Programmiertechnik, die das Arbeiten mit Daten und relationalen Datenbanken erheblich vereinfacht.

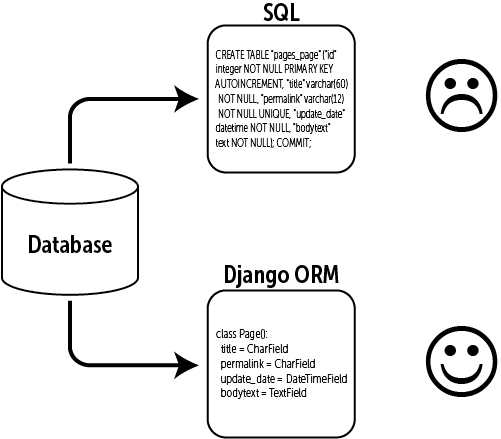


Abbildung: Django-ORM

Basierend auf der heutigen Portallösung müssen die Eingabefelder mit dem Model-Feldern übereinstimmen.

2.2.2.2 Templates

Django-Templates sind üblicherweise HTML-Dateien, um Daten möglichst einfach in einem Webbrowser ersichtlich zu machen. Ich werde mich an folgende Schlüsselargumente halten, damit die Webapplikation einwandfrei läuft.

* Programmlogik von Design trennen
* Redundanzen vermeiden
* Sicherheit gewährleisten

Das Design wird nach Vorgaben der Kommunikationsabteilung der Schweizerischen Bankiervereinigung angepasst.

#### 2.2.2.3 Views

In einem View werden Daten aus einer Datenbank, hier LDAP-Datenbank, abgerufen und an ein Template gesendet. Bei einer Webanwendung liefert das View den Inhalt an die Templates der Webseite.

Die Views treffen Entscheidungen darüber, welche Daten an die Templates geliefert werden. Diese können entweder durch Eingaben des Benutzers oder auf andere Prozesse reagieren.

Jede Django-View führt eine bestimmte Funktion aus und besitzt ein zugeordnetes Template. Views werden entweder durch eine Python-Funktion oder eine Methode einer Python-Klasse dargestellt.

### 2.2.3 Vorgehensweise nach Django-Administratoranwendung

Die Django-Administratoranwendung kann meine Models verwenden, um automatisch eine Webseite zu erstellen, mit der ich Datensätze erstellen, anzeigen, aktualisieren und löschen kann (create, view, update, und delete records). Dies spart viel Zeit bei der Entwicklung, sodass ich meine Models einfach testen kann.

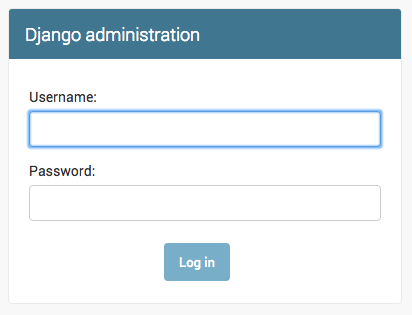
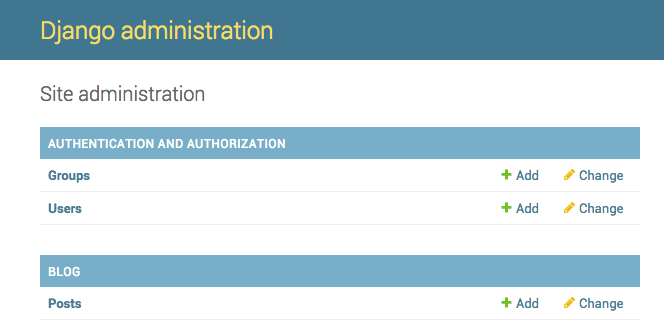
Der Nachteil liegt beim Design der Administratoranwendung. Dieses kann nicht geändert werden. Ich kann lediglich Funktionen und Models hinzufügen.

Abbildung: Design der Django-Administratoranwendung

### 2.2.4 Problemanalyse

#### 2.2.4.1 Datenverlust

Ein Datenverlust während der Arbeit kann viele Ursachen haben und dazu führen, nicht mehr im Zeitplan zu sein.

Ursache von Datenverlust:

* Hardwareprobleme (z.B. PC stürzt ab)
* Softwareproblem (z.B. Editor stürzt ab)
* Unachtsamkeit (z.B. Beim Schliessen von Programmen)

Um dies vorzubeugen, werden ich zwei Mal am Tag je ein USB-Stick-Backups und ein Repository auf Github erstellt. Die Wahrscheinlichkeit einen grösseren Datenverlust einzufangen, ist somit sehr gering.

Nicht nur der Code, sondern auch die Dokumentation muss abgesichert und versioniert werden, damit der Arbeitsablauf bei einem Datenverlust ersichtlich ist.

#### 2.2.4.2 Programmierprobleme

Während der Implementierungsphase ist die Wahrscheinlichkeit hoch, viel Zeit durch Überlegungen und Debuggen des Codes zu verlieren. Da alle Komponenten einen Einfluss auf das Endprodukt haben, muss ich schrittweise arbeiten und nichts überspringen. Der Zeitplan hilft mir dabei, die Übersicht zu behalten, um das Produkt pünktlich zu erstellen.

Bei für mich unlösbaren Probleme oder Sackgassen, kann ich die Python- und Django-Dokumentationen lesen und diese in meinem Projekt anwenden. Die Community von Python wächst täglich und somit wird auch das Wissen mehrerer Personen geteilt. Foren können dazu beitragen, auf Lösungen für bestimmte Probleme zu stossen und diese in mein Projekt anzuwenden.

#### 2.2.4.3 Zeitprobleme

Bei einer begrenzten Zeit von zehn Arbeitstagen kann es zu Problemen während des Projekts kommen, sodass der Zeitplan nicht eingehalten werden kann. Falls es dazu kommen würde, habe ich Vorgehensweisen, die den Schaden so minimieren, dass das Projekt nicht unvollständig abgegeben wird.

* Prioritäten setzten
* Dokumentation hat Vorrang

Ausserdem ist der Zeitplan sowohl in der Realisierung, als auch in der Dokumentation sehr grosszügig geplant, um Zeitprobleme zu vermeiden.

#### 2.2.4.4 Browserkompatibilität

Da die Webapplikation von rund 22‘000 Benutzerinnen und Benutzer künftig gebraucht wird, muss das Design für alle gängigen Browser kompatibel sein. Jeder Browser ist verschieden und zeigt somit verschiedene Designs an.

Das kann ich auch zu meinem Vorteil nutzen, solang die Webseite benutzerfreundlich aussieht und alle Funktionen ausführbar sind.

Die Gegenmassnahme ist die Verwendung eines Frameworks. Das Bootstrap-Framework löst grösstenteils alle Probleme über die verschieden Browser. Ich verwende die Vorlage so, dass das Design der Schweizerischen Bankiervereinigung im Vordergrund steht. Bootstrap wird lediglich die Struktur der Webanwendung darstellen.

Durch die Testfälle in der Kontrollphase wird die Kompatibilität auf allen gängigen Browsern getestet und ersichtlich gemacht, wo noch Anpassungen notwendig sind. Diese können folgend in einer zweiten Version implementiert werden.

### 2.2.5 Sicherung

Die Dokumentation und der Sourcecode müssen gesichert werden, sodass die Dateien jederzeit abrufbar sind. Dabei muss die Struktur der Versionierung für alle Beteiligten klar ist. Die Backups werden zum einen per USB-Stick und zum anderen auf der Plattform Github gespeichert.

#### 2.2.5.2 Arbeitsplatz

Der Arbeitsplatz ist während der gesamten Durchführung des Projekts so gesichert, dass eine konstante Internetverbindung besteht und keine unberechtigte Person Zugriff auf den genutzten Computer hat.

#### 2.2.5.2 USB-Stick

Das Sichern der Dateien auf einen USB-Stick wird von mir zwei Mal am Tag erstellt. Dabei wird die Naming-Convention der Schweizerischen Bankiervereinigung verwendet.

Beispiel:

* 03042018\_1200\_IPA\_Dokumentation\_JZE.pdf
* 03042018\_1645\_IPA\_Dokumentation\_JZE.pdf

Die Naming-Convention besteht aus Datum, Uhrzeit, Titel und dem Kürzel des Erstellers. Die Übersicht und Nachvollziehbarkeit ist gewährleistet.

#### 2.2.5.3 Github

Das Erstellen eines Git-Repositorys ist vor allem bei der Entwicklung eine ideale Lösung den Sourcecode nicht nur zu speichern, sondern auch den Ablauf durch genaue Bezeichnungen nachzuvollziehen.

Das Repository kann bei Github jederzeit wieder lokal hergestellt werden. Auch ältere Versionen können als einzelnes Projekt wiederverwendet werden, um Abläufe bei der Programmierung darzustellen.

Beim Projekt liegt der Vorteil in der Weiterentwicklung. Jeder Entwickler kann dank sogenannte „Commits“ nachvollziehen, wie vorgegangen wurde. Commits sind Versionen, die einzelne Titel der jeweiligen Änderung aufweisen. Sie werden in der Eingabeaufforderung (cmd.exe) ausgeführt.

Bespiel:

* git commit –m „CSS-Datei angepasst“

Der Entwickler muss demnach nicht Zeit mit Suchen verschwenden und kann die angesprochene Datei zum Editieren öffnen.

### 2.2.6 Soll/Ist-Vergleich der Systemlandschaft

Der Soll/Ist-Vergleich zeigt auf, was sich im gesamten Projekt „M17“ verändern wird. Der Hauptfokus liegt dabei auf die „Self-Service-Plattform“, also der Benutzerverwaltung, die im Rahmen der IPA programmiert wird.

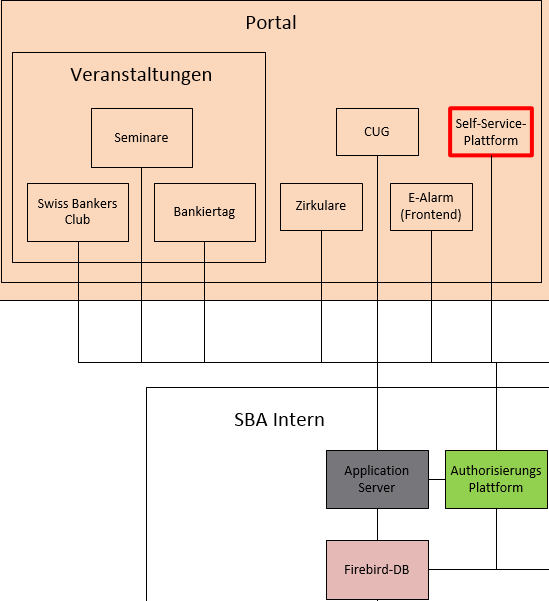


Abbildung: Ist-Situation der Systemlandschaft

Durch die Abbildung wird klar, dass alle Module zentral im Portal der Bankiervereinigung eingepflegt wurden. Dies wird sich so ändern, dass die „Self-Service-Plattform“ als ein eigenständiges Modul ausserhalb des Portals implementiert wird und an der Autorisierungsplattform angebunden ist.

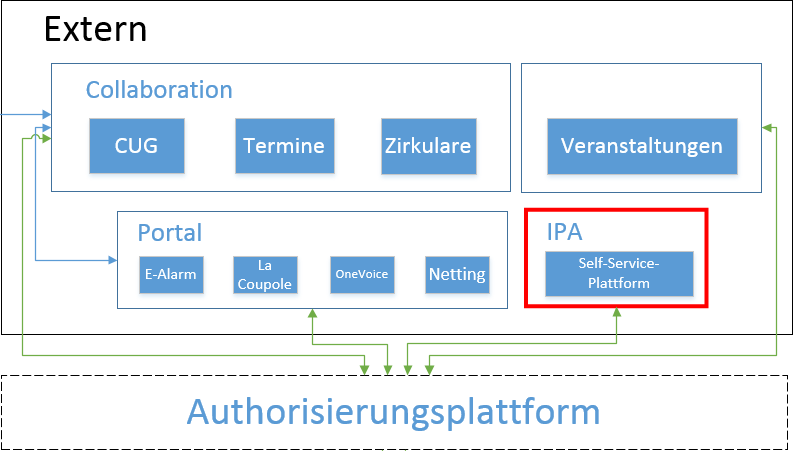


Abbildung: Soll-Situation der Systemlandschaft

Die Schweizerische Bankiervereinigung legt grossen Wert auf die Unabhängigkeit der Module. Künftig soll schneller und günstiger reagiert werden, falls die Geschäftsleitung entscheidet, dass ein Modul ersetzt werden muss.

## 2.3 Entscheiden

Die Entscheidungsphase ist der ausschlaggebende Punkt für die Realisierungsphase. Durch das Vergleichen der verschiedenen Vorgehensweisen, werde ich das zu meinem Projekt passende Vorgehen auswählen und schlussendlich anwenden.

### 2.3.1 Vergleich der Vorgehensweisen

Um die Vorgehensweisen zu Vergleichen, werde ich Kriterien anlegen, die für die ganze Webapplikation von Bedeutung sind.

|  |  |  |
| --- | --- | --- |
| Kriterium | Nach MTV-Pattern | Nach  Administratorenanwendung |
| Anpassbar nach SBVg-Design | Ja, durch selbst erstellt Templates kann das Design beliebig angepasst werden. | Nein, da der Sourcecode der Administratorenanwendung in einem Django-Package abgespeichert und nur begrenzt anpassbar ist. |
| Benutzer-Authentifizierung  (Login) | Ja, durch programmierte Views können Benutzer sich mit Ihren Daten anmelden. | Ja, die Login-Maske ist vorhanden. |
| Benutzerdaten  anzeigen | Ja, durch Templates können alle Daten des angemeldeten Benutzer beliebig angezeigt werden. | Ja, in der Administratorenanwendung können Benutzer ihre Daten sehen. Jedoch werden alle Details des Models angezeigt, was nicht benutzerfreundlich ist und für Verwirrung sorgen kann. |
| Benutzerdaten  mutieren | Ja, durch programmierte Views können Benutzer ihre Daten mutieren und speichern. | Ja, die Administratorenanwendung hat vorgelegte Funktionen. Das Mutieren der Daten ist möglich. |
| Passwort ändern | Ja, das MTV-Pattern macht es möglich das Passwort zu ändern. Dabei kann man verschieden Methoden anwenden. (z.B. altes Passwort und dann das neue etc.) | Ja, der Benutzer kann sein Passwort ändern. Dabei gibt es nur die vorgegebene Methode, die nicht modifiziert werden kann. |
| Benutzer  registrieren | Ja, durch das Programmieren von Views und dem Erstellen der Templates kann ein neuer Benutzer erstellt werden. | Der Benutzer muss vom Admin erstellt werden, während dieser angemeldet ist. |

### 2.3.2 Entscheidung der Vorgehensweisen

Die Django-Administratoranwendung, wie der Name es schon sagt, empfiehlt sich nur für das interne Datenmanagement (d.h. nur für die Verwendung durch Admins oder Personen innerhalb Ihres Unternehmens), da der modellzentrierte Ansatz nicht unbedingt die bestmögliche Schnittstelle für alle Benutzer ist und viele unnötige Details der Models offengelegt werden.

Aus diesem Grund wird die Vorgehensweise mit dem MTV-Pattern eingesetzt, da die Webapplikation modifizierbar ist und die Benutzerfreundlichkeit durch das Corporate Design der Kommunikationsabteilung selbst festgelegt wird.

## 2.4 Realisieren

### 2.4.1 Voraussetzungen

Um das Projekt zu realisieren müssen einige Voraussetzungen erfüllt sein:

* Python installieren
* Django installieren

#### 2.4.1.1 Projekt erstellen

Anschliessend wird das Projekt über Windows-Powershell gestartet. Der Grund dafür ist, das Django automatisch Code generiert. Dieser beinhaltet eine Sammlung von Einstellungen einschliesslich Datenbankkonfiguration, Django-spezifische Optionen und anwendungsspezifischen Einstellungen.



Abbildung: Erstellen der App

Die Ordnerstruktur wird folgendermassen aufgebaut:

Benutzerverwaltung /

manage.py

benutzerverwaltung/

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

|  |  |
| --- | --- |
| **Ordner/Datei** | **Definition** |
| Benutzerverwaltung / | * Verzeichnis des Projekts. * Der Name ist für Django irrelevant und kann abgeändert werden. |
| manage.py | * Wird für das Ausführen von Befehlen verwendet. |
| Benutzerverwaltung /  benutzerverwaltung | * Das eigentliche Python-Package für das Projekt. * Der Name wird für das Importieren von Packages genutzt und kann deshalb nicht geändert werden. * Wird üblicherweise klein geschrieben. |
| Benutzerverwaltung /  \_\_inti\_\_.py | * Eine leere Datei, die Python mitteilt, dass dieses Verzeichnis als Python-Paket betrachtet werden soll. |
| Benutzerverwaltung /  settings.py | * Einstellungen und Konfigurationen des Projekts. |
| Benutzerverwaltung /  urls.py | * Fungiert als „Inhaltsverzeichnis“ für Django-Webseiten. |
| Benutzerverwaltung /  wsgi.py  (für mein Projekt irrelevant) | * Ein Einstiegspunkt für WSGI-kompatible Webserver. |

Um zu verifizieren, dass das erste Projekt erfolgreich erstellt wurde, muss folgenden Befehl eingegeben werden:



Abbildung: Befehl zur Ausführung der Applikation

Bei einer erfolgreichen Ausführung wird das Projekt auf den lokalen Server http://127.0.0.1:8000/ geladen.

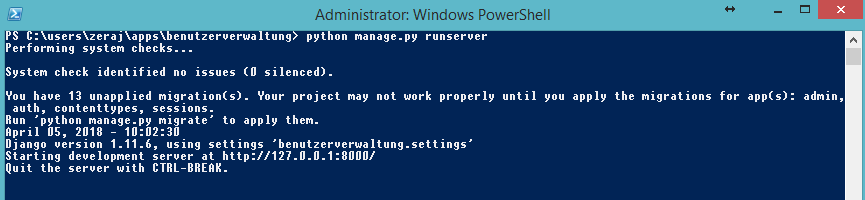


Abbildung: Ausführung der Applikation

Das blanke Projekt wird im Browser angezeigt:

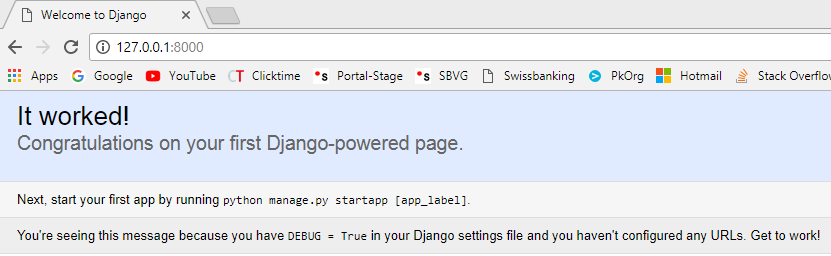


Abbildung: Django-Applikation im Browser

Bisher wurde also der Webserver gestartet, der in Python geschrieben wurde. Es handelt sich hierbei um einen Entwicklungsserver, der nicht für die Produktion gedacht ist. Django befasst sich mit Web-Frameworks und nicht der Erstellung von Webservern.

#### 2.4.1.2 Applikation erstellen

Die erste Applikation muss dem Projekt hinzugefügt werden. Um diesen zu verwenden muss folgender Befehl ausgeführt werden:



Abbildung: Erstellen der App „benutzeraccounts“

Die Ordnerstruktur der Applikation wird folgendermassen aufgebaut:

benutzeraccounts /

\_\_init\_\_.py

admin.py (nicht relevant)

apps.py (nicht relevant)

migrations /

\_\_init\_\_.py

models.py/

tests.py

views.py

urls.py

An der Struktur ist anzusehen, dass die Models und Views erstellt wurden. Die Templates werden nicht automatisch von Django generiert und müssen manuell eingerichtet werden.

Die Standard-Datenbank für Django-Projekte ist die sqlite3-Datenbank. Demnach werden lokal erstellte Benutzer in die interne Datenbank geschrieben.

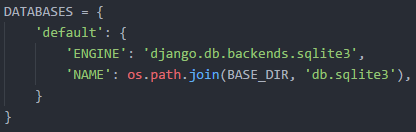


Abbildung: Standard-Datenbank in Django

Der letzte Schritt beinhaltet das Einfügen der App in das Projekt. Dies erfolgt über die setting.py.

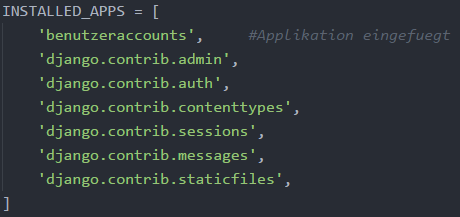


Abbildung: INSTALLED\_APPS in settings.py

### 2.4.2 Gerüst der Webapplikation

Das erste Template wird die Struktur der Webseite. Das Grundgerüst wird nach dem Prinzip DRY (Don’t Repeate Yourself) eingepflegt. Dabei werden Grundelemente jeder Webseite eingebaut und müssen nicht in jedem Template erneut eingefügt werden.

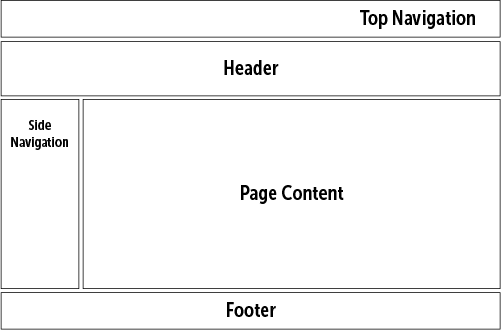


Abbildung: Eine typische Webseite mit Navigation, Header und Footer

Auch bekannt als „Parent-Child-Prinzip“ wird die Website in den Hauptteilen, die Template-übergreifend sind, und in Seiteninhalt aufgeteilt.

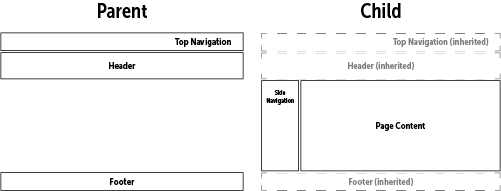


Abbildung: Parent-Child-Prinzip bei Webseiten

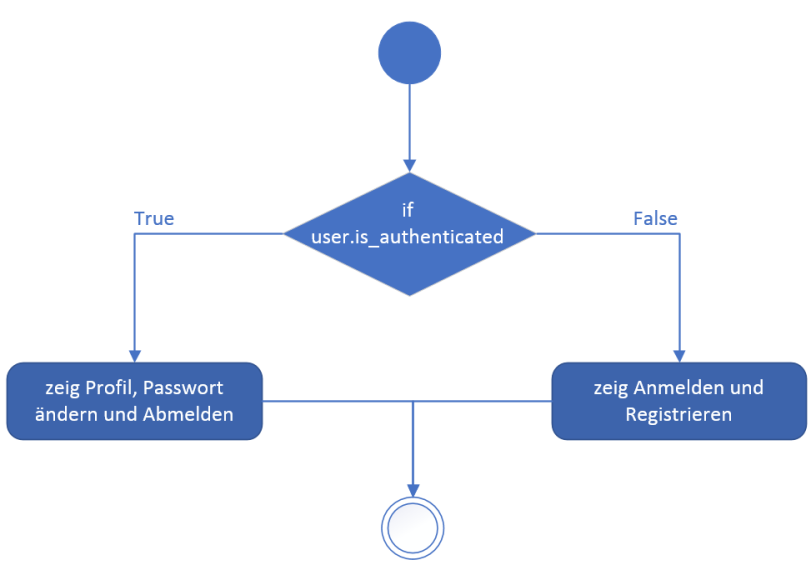
Die Navigation wird ein Teil des Grundgerüsts, deshalb unterscheidet man zwischen angemeldeten Benutzern und nicht angemeldeten Benutzern.

Abbildung: User-Navigationsprozess

#### 2.4.2.1 Bootstrap-Stylesheet

Die Benutzerfreundlichkeit der Webseite ist ein grosser Bestandteil einer Webapplikation. Sowohl für die Benutzerinnen und Benutzer, als auch für mich als Entwickler, wird das Beibehalten der Übersicht durch ein Stylesheet deutlich erleichtert und bietet mir die Möglichkeit meine Vorschritte direkt zu sehen. Es ist sinnvoll das Stylesheet von Anfang an in das Projekt zu implementieren, damit am Ende der Realisierung keine Komplikationen und Aufwände entstehen, das Design von Grund auf anzupassen.

Das Front-End-Framework von Bootstrap eignet sich ideal für die Webentwicklung. Da meine Webapplikation responsive programmiert wird, kann durch wenige Zeilen Code und der Bootstrap-Vorlage ein Grunddesign erstellt werden.

<div class="container">

<nav class="navbar navbar-default" role="navigation">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-dt">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

</div>

<div class="collapse navbar-collapse navbar-dt">

{% if user.is\_authenticated %}

<ul class="nav navbar-nav">

<li><a href="#">Profil</a></li>

<li><a href="#">Passwort ändern</a></li>

<li><a href="{% url 'accounts:logout' %}">Abmelden</a></li>

</ul>

{% else %}

<ul class="nav navbar-nav">

<li><a href="{% url 'accounts:login' %}">Anmelden</a></li>

<li><a href="#">Registrieren</a></li>

</ul>

{% endif %}

</div>

</nav>

</div>

<script src="https://code.jquery.com/jquery-3.1.1.min.js" integrity="sha256-hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8=" crossorigin="anonymous"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>

Die Elemente werden in Listen (ul > li) aufgeführt. Die Bootstrap-Klasse *navbar-toggle* gibt der Applikation vor, ab einer bestimmten Grösse die Listen zusammenzuführen und ein Symbol anzuzeigen. Beim Klicken des Symbols entsteht ein Drop-Down der Listen.

Das vorgegebene Script von Bootstrap sorgt für das Dropdown-Menü.

<div class="panel-footer">

<p style="margin-top: 0.25em;">© SwissBanking 2018. Alle Rechte vorbehalten.</p>

</div>

Code: basesite.html

Der Footer wird am Ende der Seite angefügt, um die Copyright-Rechte der Schweizerischen Bankiervereinigung zu wahren.

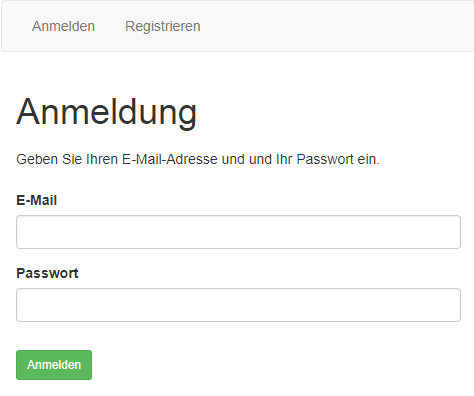
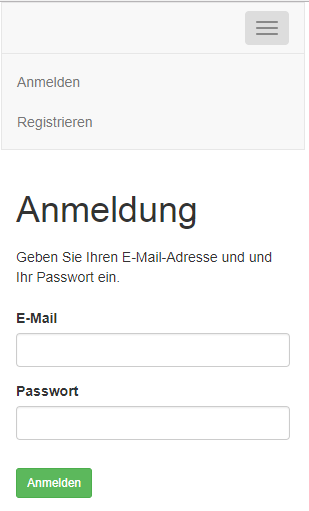


Abbildung: Login der Webseite

Abbildung: Login der Webseite (responisive)

Um die Grundstruktur zu laden, arbeitet Django mit Tags, welche Template-Elemente übergreifend laden.

Jedes Template soll mit *{% extends 'basesite.html' %}* beginnen, damit das Grunddesign vorhanden ist.

Beim Abschluss der Realisierungsphase wird die Bootstrap-Vorlage nach SBVg-Corporate-Design angepasst.

Vorteile:

* Zeit sparen
* Bringt Elemente wie Icons, Buttons, DropDown-Menüs mit
* Mit allen gängigen Browsern kompatibel
* Anpassbar
* Open Source
* Mit Javascript und jQuery erweiterbar
* Responsive-Design

Nachteile:

* Verzögerung beim Laden der Webseite (wegen grossem Inhalt)
* Layout muss sich mit dem Framework anpassen

### 2.4.3 Login

Das Login setzt sich aus der E-Mail-Adresse und dem Passwort des Benutzers zusammen. Django besitzt ein eingebautes Login-System, das für die Benutzerverwaltung in Anspruch genommen wird.

#### 2.4.3.1 Voraussetzungen

Um das eingebaute Login zu verwenden müssen Voraussetzungen erfüllt sein.

* Django.contrib.auth in den INSTALLED\_APPS (settings.py)
* Djang.contrib.middleware.AuthenticationMiddleware in MIDDLEWARE (settings.py)

Standardgemäss sind die Voraussetzungen bereits erfüllt.

#### 2.4.3.2 Lokale Benutzer

Vorerst werden sich lokalen Benutzer anmelden können und die Basis für die LDAP-Benutzer bieten, indem alle Einstellungen und das MTV-Pattern für das Login vorbereitet werden. Grund dafür ist, dass Django primär über die generierte Datenbank Daten ausliest.

Über den Befehl *python manage.py createsuperuser*  wird ein neuer Benutzer kreiert.

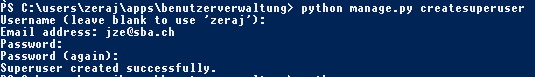


Abbildung: Erstellung eines lokalen Users

Hiermit ist klar, dass das Standard UserModel in Django aus Benutzername, E-Mail-Adresse und Passwort besteht.

Die Erweiterung des Models wird in der Realisierung während des Implementierens der Profil bearbeiten-Seite erläutert.

#### 2.4.3.3 LDAP-Benutzer

Das Projekt wird im Verlauf der Realisierungsphase an den LDAP-Server angebunden, um die Daten der LDAP-User auszulesen. Das Prinzip des Logins bleibt das Gleiche.

#### 2.4.3.4 urls.py

Bevor die Funktion ausgeführt werden kann, müssen alle für das Projekt zugänglichen URLs aufgelistet werden. Ohne die urls.py-Datei kann keine Funktion in einem Browser aufgerufen werden. Es muss der View mitgeteilt werden, was im Browser angezeigt werden soll.

Die Navigation auf einer Django-Website ist dieselbe wie auf jeder anderen Webseite. Inhalte werden über URL aufgerufen. Wenn ein Nutzer auf einen Link auf einer Website klickt, wird eine Anfrage für diese URL an Django gesendet.

from django.conf.urls import url

from django.conf.urls import include

from django.contrib.auth.views import (login, logout)

from django.contrib.auth import views

urlpatterns = [

url(r'^login/$', views.login, {'template\_name': 'benutzeraccounts/login.html'}, name='login'),

url(r'^logout/$', views.logout, {'template\_name': 'benutzeraccounts/logout.html'}, name='logout'),

]

Code: Login/Logout-URL

Die importierten views aus dem *django.contrib.auth.views*-Modul werden durch das *login* und *logout* aufgerufen. Der *template\_name* referiert auf das zu brauchende Template und durch *name=‘login‘* wird definiert, wie die URL angesprochen wird.

#### 2.4.3.5 Login-Template

Durch das Login-Template wird das eingebaute Django-Login in die Webseite eingepflegt. Dabei werden jeweils ein Label und ein Input erstellt.

* Label = Titel des Felds
* Input = Eingabe

Über einen Submit-Button werden die Felder auf ihre Validität geprüft und der User wird daraufhin authentifiziert.

#### 2.4.3.6 E-Mail Backend

Da das eingebaute Login von Django über den Benutzernamen erfolgt, muss ein Backend für die E-Mail-Authentifizierung erstellt werde.

class EmailBackend(ModelBackend):

def authenticate(self, username=None, password=None, \*\*kwargs):

UserModel = get\_user\_model()

try:

user = UserModel.objects.get(email=username)

except UserModel.DoesNotExist:

return None

else:

if user.check\_password(password):

return user

return None

Code: EMailBackend in backends.py

Die *get\_user\_model()*-Methode holt die Daten des UserModels. Die E-Mail-Adresse wird Django als *username* übergeben, damit die Authentifizierung erfolgt.

Das Backend muss in den Settings angefügt werden.

AUTHENTICATION\_BACKENDS = (

'django\_auth\_ldap.backend.LDAPBackend',

'django.contrib.auth.backends.ModelBackend',

'benutzeraccounts.backends.EmailBackend'

)

#### 2.4.3.7 Plausibilisierung der Eingaben

Den beiden Eingabefeldern werden folgende Attribute gegeben, um die Korrektheit des Schreibens zu gewährleisten.

|  |  |  |
| --- | --- | --- |
| Attribut | E-Mail | Passwort |
| type= | email | password |
| maxlength= | 50 | 25 |
| required= | true | true |
| class= | form-controll | form-controll |
| autofocus= | true | - |

Um die Korrektheit der Daten, die ein Benutzer eingibt, zu prüfen, muss eine Error-Validation erstellt werden. In Django wird dies über sogenannte Forms gemacht.

{% if form.errors %}

<p class=" label label-danger">

Ihr Benutzername oder Passwort ist nicht korrekt. Bitte versuchen Sie es erneut.

</p><br><br>

{% endif %}

Code: Die form.error wird über eine einfache If-Schlaufe durchgeführt.

Da es sich um ein eingebautes Modul von Django handelt, erkennt die form.error sofort, ob die Eingaben valid sind und gibt bei Fehleingaben die eingegeben Error-Meldung aus. Bei nicht eingebauten Modulen müssen die *forms* selber in einer Datei (forms.py) erstellt werden. Diese werden Im Verlauf dargestellt.

#### 2.4.3.8 CSRF-Sicherheiten

Das CSRF-Middleware- und -Template-Tag bietet einen einfach zu verwendenden Schutz vor Cross-Site-Request-Forgeries in Django.

<form method="post">

{% csrf\_token %}

MIDDLEWARE = [,

'django.middleware.csrf.CsrfViewMiddleware',

]

Code:CSFR-Middleware in settings.py Code: crfs\_token in einem Template

Es werden Angriffe verhindert, die auf bösartigen Webseiten einen Link, eine Formularschaltfläche oder JavaScript-Code enthält, der bestimmte Aktionen auf der Website ausführen soll. Dabei werden die Anmeldeinformationen eines angemeldeten Benutzers verwendet, der eine schädliche Webseite in seinem Browser besucht.

#### 2.4.3.9 Logout

Die Logout-Funktion wird ebenfalls über die view im django.contrib.auth ausgeführt. Durch eine Einstellung in der settings.py-Datei wird der User auf das Login geleitet. Das bietet den Vorteil, kein Logout-Template zu erstellen und die Applikation schlank zu halten.

*LOGOUT\_REDIRECT\_URL = 'user:login'*

### 2.4.4 LDAP-Anbindung

Eine LDAP-Anbindung ist für die Webapplikation essentiell, da alle Benutzer aufgelistet sind und sich mit ihren Daten anmelden müssen. Django bietet dafür ein Package, das notwendig ist, um eine Verknüpfung herzustellen.

AUTH\_LDAP\_SERVER\_URI = "ldap://192.168.X.XX"

AUTH\_LDAP\_BIND\_DN = "LDAP\_IPA"

AUTH\_LDAP\_BIND\_PASSWORD = "Passwort"

Code: LDAP-Anbindung

Das *django-auth-ldap*-Package wird in der settings.py-Datei ausgeführt und erstellt eine Verbindung über die IP-Adresse des Servers. Verlangt werden noch BIND\_DN, also ein Benutzer (vorzugsweise mit Admin-Rechten) und das Passwort des Benutzers.

AUTH\_LDAP\_USER\_ATTR\_MAP = {

"first\_name": "givenName",

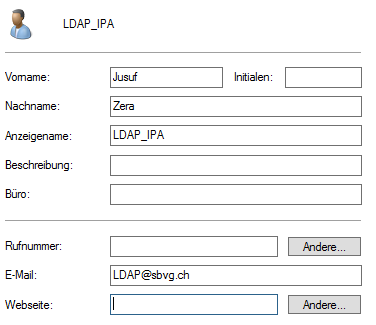
"last\_name": "sn",

"email": "mail",

}

Code: LDAP-Filter

Die AUTH\_LDAP\_USER\_ATTR\_MAP gibt vor, welche Attribute ausgelesen werden sollen. Das django-auth-ldap-Package gibt dabei nur die Attribute givenName (Vorname), sn (Nachname) und mail (E-Mail) vor.

Aus prioritätsgründen wird sowohl das UserModel als auch das Mapping der Attribute beim Implementieren der Profil bearbeiten-Seite erweitert.

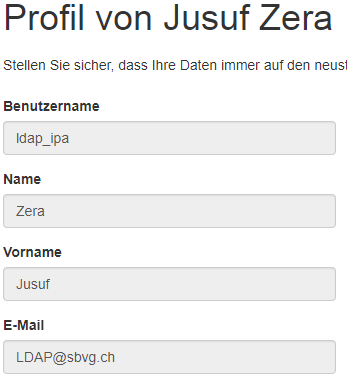


Abbildung: LDAP-Benutzer in Webapplikation

Der Benutzername wird im Endprodukt nicht angezeigt. Die Abbildung soll lediglich darstellen, dass die Daten aus dem LDAP-Verzeichnis gezogen werden.

Damit die Anbindung funktioniert muss das backend des django-auth-ldap-Package eingefügt werden.

AUTHENTICATION\_BACKENDS = (

'django\_auth\_ldap.backend.LDAPBackend',

'django.contrib.auth.backends.ModelBackend',

)

Code: Backend in settings.py

Das LDAPBackend sorgt für die Authentifizierung der LDAP-Benutzer und das ModelBackend (standardgemäss generiert) für die Authentifizierung der internen Benutzer.

#### 2.4.4.1 Django-LDAP-Workflow

Nachdem sich ein LDAP-Benutzer angemeldet hat, wird automatisch ein lokaler User in die sqlite3-Datenbank geschrieben. Die Daten werden somit aus dem LDAP gezogen und in die interne Datenbank eingefügt. Das Django-Framework arbeitet also mit dem eingebauten UserModel und nicht mit dem verknüpften LDAP.

Das hat zur Folge, dass ab jetzt mit dem UserModel gearbeitet wird und die LDAP-Benutzer sich danach richten.

### 2.4.5 Profil anzeigen

#### 2.3.5.1 URL

Der erste Schritt eine Komponente einzufügen ist durch das Einfügen einer URL.

from . import views

urlpatterns = [

url(r'^login/$', login, {'template\_name': 'benutzeraccounts/login.html'}, name='login'),

url(r'^logout/$', logout, name='logout'),

url(r'^profile/$', views.profile, name='profile'),

]

Code: urls.py (Profile)

Die Profil-Komponente ist nicht in Django über ein Modul erreichbar. Daher wird eine eigene View erstellt, welche durch über *views.profile* aufgerufen wird.

#### 2.3.5.2 Funktion

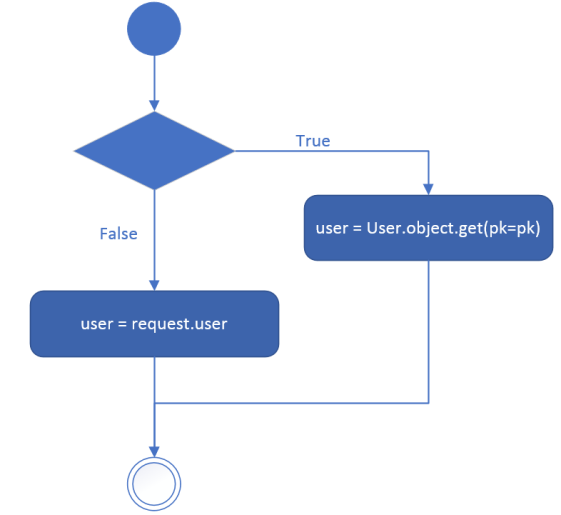
Durch den Import des UserModels ist es möglich die Profil-Funktion als View auszuführen.

Abbildung: Profil-Prozess

from django.contrib.auth.models import User

def profile(request, pk=None):

if pk:

user = User.object.get(pk=pk)

else:

user = request.user

args = {'user': user}

return render(request, ('benutzeraccounts/profile.html'), args)

Code: views.py (Profile)

Pk wird als Parameter vom Query des UserModels verwendet. Pk steht für „primary key“ und könnte auch durch „id“ ersetzt werden. Jedoch ist pk unabhängig vom Primärschlüsselfeld eines Models. Somit spielt es keine Rolle, ob ein eigenes Primärschlüsselfeld definiert wird oder Django eines automatisch erstellt. Es verhält sich wie ein normales Attribut im Modell, ist aber tatsächlich ein Alias für das Attribut, dass das Primärschlüsselfeld für das Modell bildet.

#### 2.3.5.4 Profile-Template

Das Profil-Template ist nicht editierbar und soll nur zum aufzeigen aller Daten dienen. Aus diesem Grund wird den Feldern das Attribut *readonly* gegeben. Die Anwender haben keine Berechtigung das Feld zu ändern.

<input style="margin-bottom: 1em;" class="form-control" readonly value="{{ user.last\_name }}"></input>

Code: Profil-Feld

Durch das übergeben des Tags in Value-Attribut erkennt die Django Applikation, welches Feld aus der Datenbank gezeigt werden muss.

### 2.4.6 Profil bearbeiten

#### 2.4.6.1 UserModel erweitern

Bevor die Daten der Benutzer verändert werden können, müssen die Felder kreierte werden, die nicht standardgemäss in Django implementiert sind. Dabei erweitert man das UserModel, indem man ein weiteres Model erstellt und dieses an das UserModel anknüpft.

Die Models werden in der von Django generierten models.py-Datei erstellt. Die Attribute werden übergeben.

class Profile(models.Model):

user = models.OneToOneField(User)

salutation = models.CharField(max\_length=30, default='')

title = models.CharField(max\_length=30, default='')

phone = PhoneNumberField(default='')

fax = PhoneNumberField(default='')

mobile = PhoneNumberField(default='')

birthday = models.CharField(max\_length=30, default='')

departement = models.CharField(max\_length=50, default='')

rank = models.CharField(max\_length=50, default='')

activity = models.CharField(max\_length=50, default='')

institute = models.CharField(max\_length=50, default='')

Code: ProfileModel in models.py

#### 2.4.6.2 Django-Migrations

Das Model wird durch das Eingeben von Befehlen im Powershell kreiert.

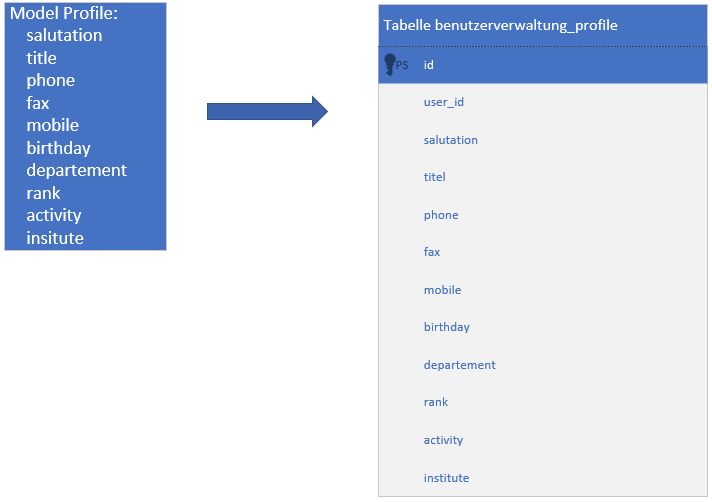
* Python manage.py makemigrations = Kreiert Tabelle „Profile“
* Python manage.py migrate = Migriert die Tabelle in das Projekt

Abbildung: Erstellung der Tabelle „benutzerverwaltung\_profile“

Die Migrationen müssen jeweils durchgeführt werden, wenn ein neues Attribut hinzukommt oder entfernt wird.

#### 2.4.6.3 OneToOne-Verknüpfung

Über ein OneToOneField wird eine One-to-One-Beziehung mit dem bestehenden UserModel hergestellt wird.

Sie wird verwendet, weil authentifizierungsunabhängige Felder kreiert werden müssen, die an das UserModel verknüpft werden.

user = models.OneToOneField(User)

Code: OneToOneField-Verknüpfung mit UserModel

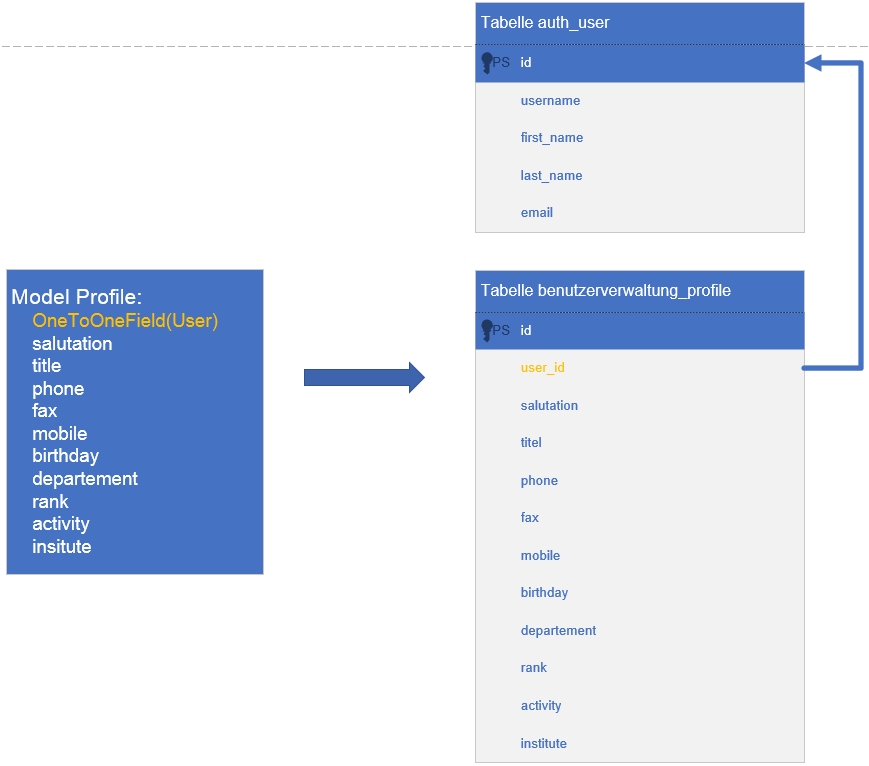


Abbildung: Relation des UserModel und des ProfileModel

Die Tabelle benutzerverwaltung\_profile, also das neu erstellte Model, wird mit dem Primärschlüssel des von Django generierten UserModels (auth\_user-Tabelle) verknüpft.

Da die übrigen Felder in die Applikation eingebaut wurde, kann wieder nach MTV-Pattern vorhergegangen werden.

#### 2.4.6.4 URL

Die Seite zum Bearbeiten des Profils wird über einen Button auf der Profil-Seite des angemeldeten Benutzers zu finden sein.

url(r'^profil/bearbeiten/$', views.edit\_profile, name='edit\_profile')

Code: profil/bearbeiten-URL

#### 2.4.6.5 Form

Dadurch, dass die einzelnen Felder überarbeitet werden, wird das Bearbeiten über Formen erstellt.

from \_\_future\_\_ import unicode\_literals

from django.db import models

from django import forms

from django.contrib.auth.models import User

from benutzeraccounts.models import Profile

class UserForm(forms.ModelForm):

class Meta:

model = User

fields = (

'first\_name',

'last\_name',

'email'

)

class ProfileForm(forms.ModelForm):

class Meta:

model = Profile

fields = (

'salutation',

'title',

'phone',

'fax',

'mobile',

'birthday',

'departement',

'rank',

'activity',

'institute'

)

## 

Code: UserForm und ProfileForm

Es werden jeweils Klassen zu den erstellten Models eingerichtet, die die Meta-Daten beinhalten.

#### 2.4.6.6 Funktion

Da die Formen für die Attribute der Felder zuständig sind, prüft die view-Funktion *edit\_profile* nur, ob alle Eingaben gültig sind. Bei Korrektheit wird der Benutzer auf sein Profil weitergeleitet.

def edit\_profile(request):

if request.method == 'POST':

user\_form = UserForm(request.POST, instance=request.user)

profile\_form = ProfileForm(request.POST, instance=request.user.profile)

if user\_form.is\_valid() and profile\_form.is\_valid():

user\_form.save()

profile\_form.save()

return redirect(reverse('user:profile'))

else:

user\_form = UserForm(instance=request.user)

profile\_form = ProfileForm(instance=request.user.profile)

args = {'user\_form': user\_form, 'profile\_form':profile\_form}

return render(request, 'benutzeraccounts/edit\_profile.html', args)

Code: Profil bearbeiten-Funktion

Die *user\_form* (UserModel) und die *profile\_form* (ProfileModel) müssen einzeln angesprochen werden, weil sie unterschiedliche Attribute aufweisen.

Dafür wird ein *request* der Daten eingebaut, der die Daten aus den Instanzen *instance=request.user* und *instance=request.user.profile* filtert.

*instance=request.****user.profile*** ist nicht *instance=request.****profile***, weil das ProfieModel an das UserModel verknüpft wurde und über die Instanz *user* angesprochen werden muss.

Die Validierung wird durch die *is\_valid()*-Methode geprüft. Dies wird auf beide Models angewendet, sodass man nur auf das Profil weitergeleitet werden kann, wenn alle Attribute gültig eingegeben wurden.

Die *save()-*Methode speichert die eingegebenen Daten. Sind die Daten nicht gültig, wird der Benutzer nicht weitergeleitet. Eine Fehlermeldung zeigt an, was angepasst werden muss.

#### 2.4.6.7 Profil bearbeiten-Template

Das Template muss die Daten über die Form auslesen. Dafür zuständig sind die Tags:

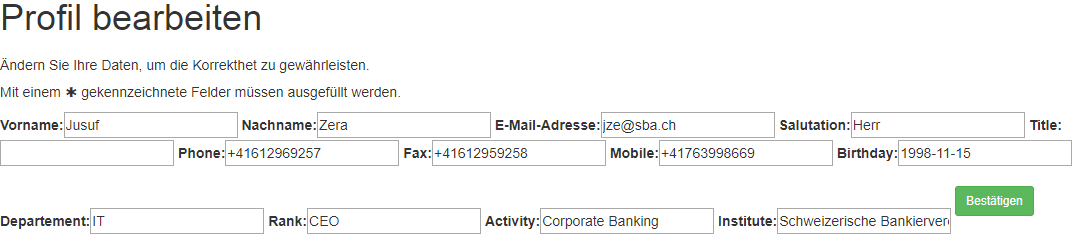
* {{ user\_form }}
* {{ profile\_form }}

Abbildung: Profil-bearbeiten ohne Formatierung

Da die Tags nicht formatiert sind, werden die Felder ohne Formatierung im Browser angezeigt. Die einzelnen Attribute müssen in den forms.py angepasst werden, damit die Benutzerfreundlichkeit gewährleistet ist.

first\_name = forms.CharField(max\_length=30, required=True,

widget=forms.TextInput(

attrs={

'class': 'form-control',

'type': 'text',

'name': 'first\_name',

}

)

)

Code: Bespiel formatiertes Attribut

Dem Attribut *first\_name* werden Style-Attribute übergeben.

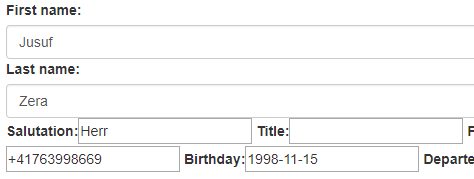
Der Unterschied ist anzusehen:

Abbildung: Unterschied formatiert und nicht formatierter Attribute

Die formatierten Attribute werden nun in Tabellen eingefügt, sodass ein benutzerfreundliches Formular entsteht. Die Platzhalter {{ user\_form }} und {{ profile\_form }} werden durch die Platzhalter der einzelnen Attribute ersetzt:

<tr><td><label style="margin-top: 1em;" for="{{ form.titel.id\_for\_label }}">Titel</label></tr></td>

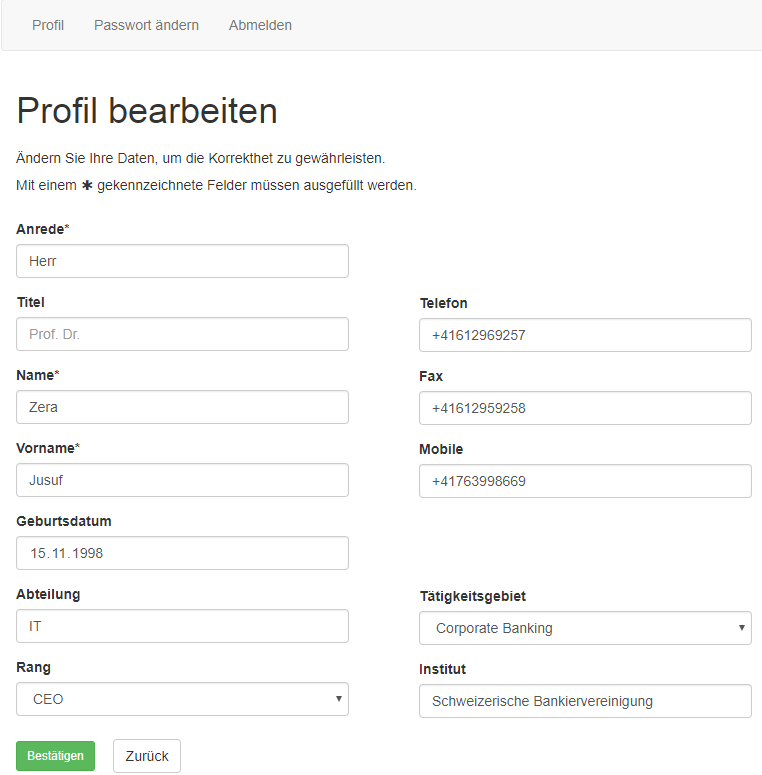
<tr><td>{{ profile\_form.title }}</td></tr>

<tr><td><label style="margin-top: 1em;" for="{{ form.last\_name.id\_for\_label }}">Name\*</label></tr></td>

<tr><td>{{ user\_form.last\_name }}</td></tr>

Code: Attribut-Platzhalter in einer Tabelle

Die Seite wird im Browser benutzerfreundlich dargestellt:

 Abbildung: Profil bearbeiten-Seite

Falls die Anwender ihre Daten ändern und sich doch dazu entscheiden die Anpassungen nicht zu speichern, wird ein Abbrechen-Button erstellt. Dieser leitet den Benutzer ohne zu Speichern auf das Profil zurück.

#### 2.4.6.8 E-Mail-Adresse ändern-Workflow

Damit ein Benutzer die E-Mail-Adresse ändern kann, muss die Schweizerische Bankiervereinigung per Mail kontaktiert werden. Die E-Mail-Adresse ist der ausschlaggebende Punkt für die Verteilung der Berechtigungen der einzelnen Module. Dies wird über die Domäne des Anwenders in der Autorisierungsplattform geregelt.

Bei Ausnahmefällen (z.B. Bundesrat, Nationalrat etc.) kann die private Domäne des Anwenders eingegeben werden. Die Berechtigungen werden manuell übergeben.

### 2.4.7 Django-LDAP-Workflow

Das Django-Framework ist für die Authentifizierung und dem Auslesen der Daten der LDAP-Datenbank geeignet. Das Zurückschreiben funktioniert technisch nicht. Aus diesem Grund wird ein Workaround mit Python-Operations erstellt.

#### 2.4.7.1 Python-Operations

Um die Daten in der LDAP-Datenbank zu ändern, müssen LDAP-Operations angewendet werden. LDAP-Operations sind im ldap3-Package (python-ldap3) gespeichert und haben alle unterschiedlichen Aufgaben:

|  |  |
| --- | --- |
| Operation | Definition |
| BIND | Erstellt eine Verbindung zum LDAP-Server |
| UNBIND | Endet die Session und Kommunikation mit dem LDAP-Server |
| ADD | Hinzufügen eines Eintrags in das LDAP-Verzeichnis.  Die Add-Operation wird nur für neue Einträge verwendet, d.h. der DN muss auf ein nicht existierendes Objekt verweisen, aber die übergeordneten Objekte müssen existieren. |
| DELETE | Löschen eines Eintrag im LDAP-Verzeichnis |
| MODIFY | Die Modify-Operation ermöglicht es, Änderung eines bereits im LDAP-Verzeichnis vorhandenen Eintrags anzufordern. |
| MODIFY-DN | Die ModifyDN-Operation ermöglicht es, den RDN-Wert (Relative Distinguished Name) eines Eintrags zu ändern oder einen Eintrag im LDAP-Verzeichnis zu verschieben. |
| SEARCH | Die Search-Operation wird dazu verwendet, Einträge zurückzugeben, die mit dem Suchfilter übereinstimmen. |
| COMPARE | Vergleicht ein Eintragsattribut mit einem bestimmten Wert. |

Die passende Operation ist die MODIFY-Operation. Dabei gibt es verschiedene Arten sie auszuführen. Die Modify-Operation hat vier:

|  |  |
| --- | --- |
| MODIFY\_ | Definition |
| ADD | Gibt dem angegebenen Attribut Werte |
| DELETE | Löscht Werte aus dem Attribut. Wenn kein oder alle Werte des Attributs aufgelistet sind, wird das ganze Attribut entfernt |
| REPLACE | Ersetzt alle vorhandenen Werte des Attributs durch die aufgelisteten neuen Werte und erstellt das Attribut, wenn es noch nicht vorhanden war. Ein Ersetzen ohne Wert löscht das gesamte Attribut. |
| INCREMENT | Alle vorhandenen Werte des angegebenen Attributs müssen um den aufgelisteten Wert erhöht werden. |

#### 2.4.7.2 Modify\_Replace

Da in Bezug auf die Benutzerverwaltung lediglich das Zurückschreiben nicht funktioniert, is die Modify\_Replace-Operation passend zum Projekt. Die Operation in Python verlangen jedoch einige Voraussetzungen, um ausgeführt zu werden.

* DN (distinguished name) muss spezifiziert werden
* Genauen Änderungen müssen erläutert werden.

Der untenstehende Code wird in einer neuen Datei (modify.py) gesichert und über diese direkt ausgeführt:

* python modify.py runserver

from ldap3 import Server, Connection, ALL, MODIFY\_REPLACE  
from default import settings

def modify(self,dn,changes,controls=None):

s = Server('192.168.X.XX', get\_info=ALL)

c = Connection(s, 'cn=ldap\_ipa,cn=Users,dc=sbvg,dc=ch', 'Passwort', auto\_bind=False)

c.bind()

c.modify('cn=ldap\_ipa,cn=Users,dc=sbvg,dc=ch',

{'givenName': [(MODIFY\_REPLACE, ['Max'])],

'sn': [(MODIFY\_REPLACE, ['Zera'])]})

print(c.result)

c.unbind()

Code: Beispiel Modify\_Replace-Operation

Im Codebeispiel wird eine Verbindung über einen Benutzer mit dem Server hergestellt und durch *get\_info=ALL* werden alle Attribute geholt. Im *c.modify* wird der DN angegeben, der modifiziert wird. Der Vorname wird zu „Max“ und der Nachname zu „Mustermann“ geändert. Zum Schluss wird die Verbindung getrennt.

#### 2.4.7.3 Ursachen

Da dieses Projekt nicht mit einer relationalen Datenbank erstellt werden sollte, sondern mit einem LDAP-Verzeichnis, musste schon in der Planungsphase ersichtlich gemacht werden, dass es technisch nicht möglich ist, mit Django in ein LDAP-Verzeichnis zurückzuschreiben.

Django unterstützt nämlich nur vier Datenbanken:

* PostgreSQL
* MySQL
* SQLite (wird für das Projekt verwendet)
* Oracle

Die LDAP-Datenbank kann durch das Verwenden der django-auth-ldap-Bibliothek (wie im Projekt) angefügt werden. Das Zurückschreiben der LDAP-Datenbank ist nicht in der Bibliothek vorhanden.

#### 2.4.7.4 Massnahmen

Das Zurückschreiben des LDAP-Verzeichnisses wird nicht von Django unterstützt. Das Framework kann über das LDAP authentifizieren und die Daten auslesen.

Das Modifizieren wird über die Django-Models ausgeführt. Somit wird die Authentifizierung und Auslesen weiterhin über das LDAP gehen, das Modifizieren der Benutzerdaten jedoch über die integrierte SQLite3-Datenbank.

#### 2.4.7.5 Folgen

Der Workaround mit der python-ldap3 Bibliothek, wird in der Weiterentwicklung für das Zurückschreiben verwendet. Dabei wird eine Schnittstelle mit der Autorisierungsplattform eingerichtet, dass Daten über das Script in die LDAP-Datenbank einfügen.

Dies bietet den Vorteil, dass zum einen die Webapplikation vollständig funktionieren würde und zum anderen, dass eine erhöhte Sicherheit besteht, weil das Script über die Autorisierungsplattform ausgeführt wird.

### 2.4.8 Passwort ändern

#### 2.4.8.1 URL

Über die in der urls.py erstellten URL kommt der Benutzer auf die Passwort ändern-Seite.

Code: passwort\_wechseln-URL

*url(r'^passwort\_wechseln/$', views.change\_password, name='change\_password'),*

#### 2.4.8.2 PasswordChangeForm

Das Django-Framework beinhaltet neben den eingebauten Views (z.B. Login) auch eingebaute Forms, die für die Passwort ändern-Funktion verwendet werden. Die Formen sind im *django.contrib.auth.forms* gespeichert. Die zwei passenden Forms unterscheiden sich in einem Punkt wesentlich.

|  |  |
| --- | --- |
| Form | Definition |
| PasswordChangeForm | Eine Form, mit dem ein Benutzer sein Kennwort ändern kann, indem das alte Passwort zur Verifizierung eingegeben werden muss. |
| SetPasswordForm | Eine Form, mit dem ein Benutzer sein Kennwort ändern kann, ohne das alte Kennwort einzugeben. |

Um die Sicherheit zu gewährleisten wird die PasswordChangeForm verwendet.

from django.contrib.auth.forms import PasswordChangeForm

class PasswordForm(PasswordChangeForm):

class Meta:

model = User

fields = (

'old\_password',

'new\_password1',

'new\_password2',

)

Code: PasswordChangeForm in forms.py

Da das Passwort im UserModel gespeichert ist, wird *model = User* verwendet. Django erkannt automatisch, dass das *old\_passwort* das momentane Passwort ist.

#### 2.4.8.3 Funktion

Obwohl die PasswordChangeForm in Django eingebaut ist, wird die *change\_password-*Funktion erstellt, da das Passwort über die *update\_session\_auth\_hash()*-Methode aktualisiert werden muss und die neuen Daten über die save()-Methode gespeichert werden müssen.

def change\_password(request):

if request.method == 'POST':

form = PasswordChangeForm(request.user, request.POST)

if form.is\_valid():

user = form.save()

update\_session\_auth\_hash(request, user)

return redirect('user:profile')

else:

form = PasswordChangeForm(request.user)

cont = {'form': form}

return render(request, 'benutzeraccounts/change\_password.html', cont)

Code: change\_password-Funktion

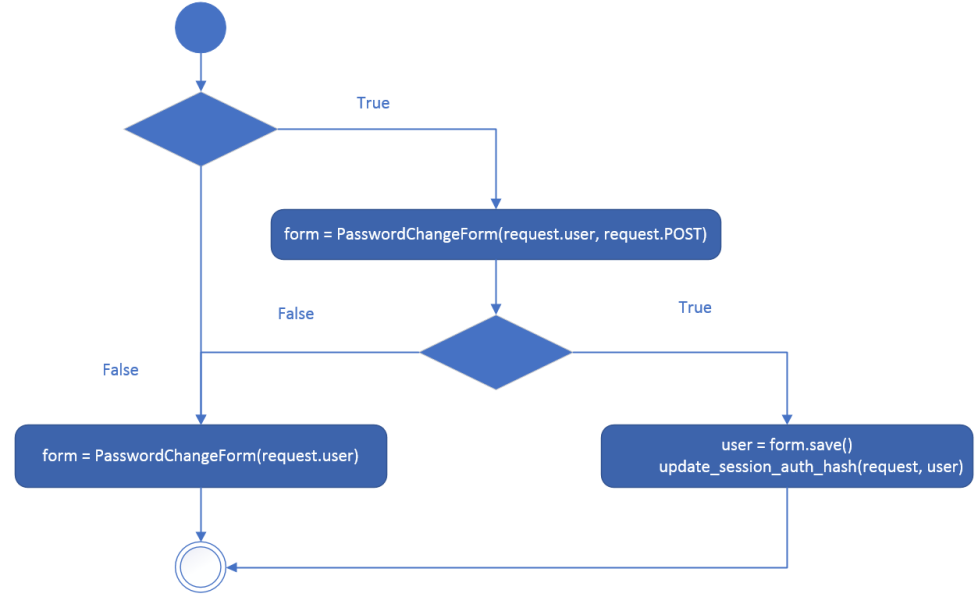


Abbildung: Passwort ändern-Prozess

Durch die is\_valid()-Methode prüft Django ob die vom Benutzer eingegebenen Daten gültig sind. Sind die Daten gültig, wird der Benutzer auf sein Profil weitergeleitet und die neuen Daten werden in die gleiche Session aktualisiert.

#### 2.4.8.4 Error-Anzeigen

Trägt der Benutzer die Daten falsch ein, muss jeweils eine Error-Meldung dargestellt werden, ob das alte oder neue Passwort ungültig eingegeben wurde.

Dafür werden Funktionen in der forms.py erstellt:

def clean\_old\_password(self):

old\_password = self.cleaned\_data.get('old\_password')

cont = User.objects.filter(old\_password=old\_password)

if cont.exists():

raise forms.ValidationError("")

return old\_password

Code: Altes Passwort überprüfen

Die Funktion *clean\_old\_passwort(self)* testet ob das eingegebene Passwort mit dem alten Passwort übereinstimmt, indem die *exists()-*Methode die beiden Vergleicht. Der *ValidationError(““)* wird im Template definiert.

def clean\_password(self)

password1 = self.cleaned\_data.get("password1")

password2 = self.cleaned\_data.get("password2")

if password1 and password2 and password1 != password2:

raise forms.ValidationError("")

return password2

Code: Neues Passwort überprüfen

Die Funktion *clean\_passwort(self)* gibt eine Fehlermeldung aus, wenn die beiden neuen Passwörter nicht identisch sind. Der *ValidationError(““)* wird im Template definiert.

Die Fehlermeldungen werden im Template über Tags eingefügt:

{% if form.old\_password.errors %}

<p class="label label-danger">

Ihre altes Passwort ist nicht korrekt. Bitte versuchen Sie es erneut.

</p>

{% endif %}

{% if form.new\_password2.errors %}

<p class="label label-danger">

Ihr neues Passwort stimmt nicht überein. Bitte versuchen Sie es erneut.

</p>

{% endif %}

Code: Error-meldungen im Template

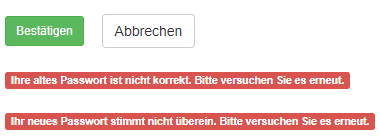


Abbildung: Error-Anzeigen beim Passwort ändern

#### 2.4.8.5 Passwort-Voraussetzungen

Die Validierung des Passworts ist in den Einstellungen zu finden. Unter den AUTH\_PASSWORD\_VALIDATORS sind die Kriterien vordefiniert.

|  |  |
| --- | --- |
| Validator | Definition |
| UserAttributeSimilarityValidator | Überprüft die Ähnlichkeit zwischen dem Passwort und den Attributen des Benutzers |
| MinimumLengthValidator | Prüft, ob das Passwort eine Mindestlänge erreicht.  Standard: 8. |
| CommonPasswordValidator | Überprüft, ob das Passwort in einer Liste von allgemeinen Passwörtern vorkommt.  Standard: Liste von 1000 allgemeinen Passwörtern. |
| NumericPasswordValidator | Überprüft, ob das Passwort nicht vollständig aus Nummern besteht. |

Die Kriterien werden dem Benutzer ohne Error-Anzeige klar gemacht:

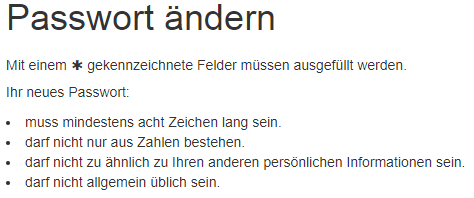


Abbildung: Passwort-Kriterien im Browser

### 2.4.9 Registrierung (Weiterentwicklung)

Da die Registrierung der Benutzer direkt über die Autorisierungsplattform erfolgt, wird das Projekt ohne eine Registrierungsseite veröffentlicht. Jedoch wird die Funktion für die Weiterentwicklung eingebaut, sodass bei einer zweiten Version, die Selfservice-Plattform sowohl Benutzer erstellen als auch mutieren kann.

Das Erstellen von neuen Benutzern hat die Voraussetzung, dass in die LDAP-Datenbank geschrieben werden muss. Da dies über Django nicht möglich ist, werden die Benutzer in die eingebaute SQLite3-Datenbank geschrieben. Die Funktionen bleiben gleiche, sodass in der Kontrollphase Testfälle erstellt werden können.

#### 2.4.9.1 URL

Über die in der urls.py erstellten URL kommt der Benutzer auf die Passwort ändern-Seite.

url(r'^registrieren/$', views.register, name='register'),

Code: Registrieren-URL

#### 2.4.9.2 UserCreationForm

Für die Registrierungsseite wird die eigebaute UserCreationForm (*django.contrib.auth.forms*) verwendet, die über die Meta-Daten der Models einen neuen Benutzer kreiert. Diese setzt voraus, dass ein Benutzername definiert werden muss, damit die Form gültig ist. Der Benutzername wird in der Applikation selbst jedoch nicht angezeigt, da die Anmeldung über die E-Mail-Adresse erfolgt.

class RegistrationForm(UserCreationForm):

username = forms.CharField(max\_length=30, required=True,

widget=forms.TextInput(

attrs={

'class': 'form-control',

'type': 'text',

'name': 'username',

}

)

)

Code: UserCreationForm mit Benutzername

Nachdem alle Felder die zugehörigen Attribute erhalten habe, werden die Meta-Daten des Benutzers in die *class Meta* angefügt.

class Meta:

model = User

fields = (

'username',

'first\_name',

'last\_name',

'email',

'password1',

'password2',

'salutation',

'title',

'phone',

'fax',

'mobile',

'birthday',

'departement',

'institute'

)

## Code: Meta-Daten des Benutzers

#### 2.4.9.3 Funktion

Die register-Funktion wird die UserCreationForm auf die Gültigkeit der Daten prüfen. Sobald die korrekten Daten des Benutzers eingegeben wurden, werden diese in die Datenbank gespeichert. Die Attribute des ProfileModels werden einzeln über eine get()-Funktion geholt und über die save()-Methode gespeichert.

def register(request):

if request.method == 'POST':

form = RegistrationForm(request.POST)

if form.is\_valid():

user = form.save()

user.refresh\_from\_db()

user.profile.salutation = form.cleaned\_data.get('salutation')

user.save()

Code: register-Funktion (verkürzt)

Anschliessend wird über die Variable *password,* welches Eingabefeld als tatsächliches Passwort verwendet werden soll. Die user-Variable definiert, wie die authentifizierung erfolgen soll. Die E-Mail-Adresse wird als username verwendet, da die Anmeldung ebenfalls über diese erfolgt.

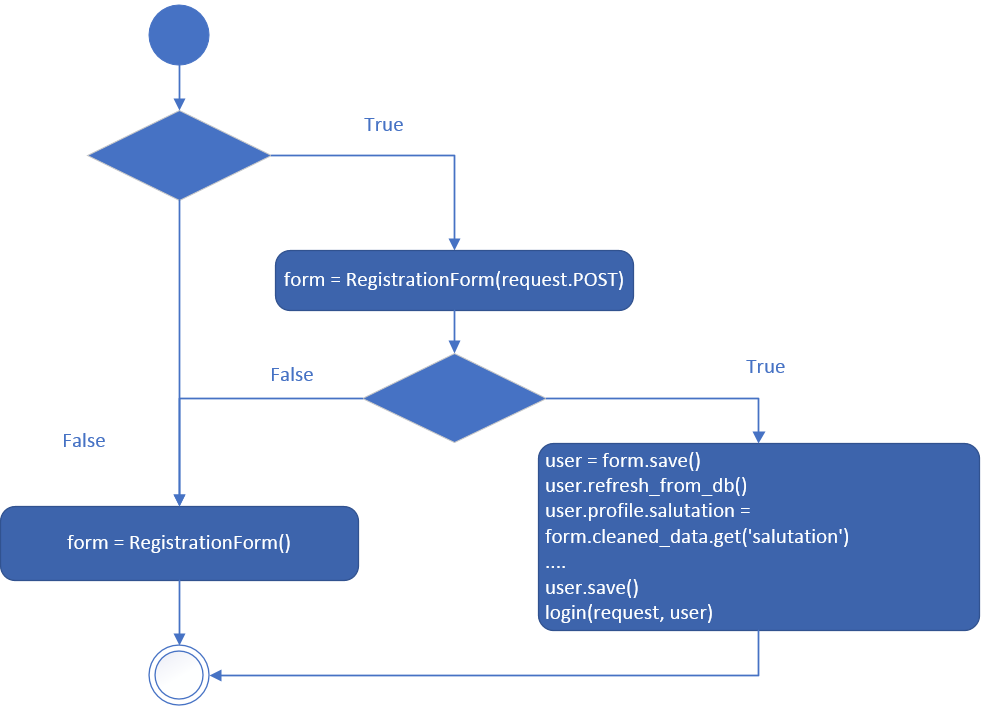
password = form.cleaned\_data.get('password1')

user = authenticate(username=user.email, password=password)

login(request, user)

return redirect('user:profile')

Code: Authentifizierung in def register  
  
Durch *login(request, user)* wird der Benutzer auf die Profilseite weitergeleitet.

  
Abbildung: Registrierungsprozess

#### 2.4.9.4 Error-Anzeigen

Die Error-Meldungen werden für den Benutzernamen, die E-Mail-Adresse und dem Passwort erstellt. Benutzernamen und E-Mail-Adressen dürfen jeweils nur einmal in der Datenbank existieren. Die Passwörter müssen übereinstimmen, damit das Anmelden erfolgt.

def clean\_password(self):

password1 = self.cleaned\_data.get("password1")

password2 = self.cleaned\_data.get("password2")

if password1 and password2 and password1 != password2:

raise forms.ValidationError("")

return password2

def clean\_email(self):

email = self.cleaned\_data.get('email')

cont = User.objects.filter(email=email)

if cont.exists():

raise forms.ValidationError("")

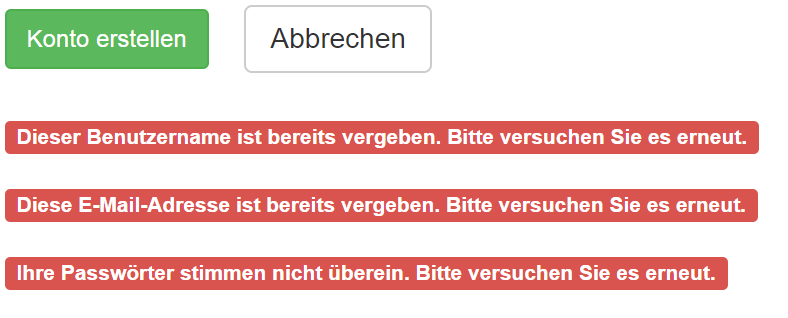
return email

Code: clean\_email Code: clean\_password

Es wird ein Filter erstellt, der die eingegeben E-Mail mit den E-Mail-Adressen in der Datenbank vergleicht. Eine Fehlermeldung wird ausgegeben, wenn die E-Mail bereits existiert. Das Prinzip für den Benutzernamen ist identisch und wird angewendet.

Die Passwörter werden mit dem Ungleichzeichen „!=“ verglichen. Sind sie nicht identisch wird eine Fehlermeldung angezeigt.

Über die Tags *{% if form.username.errors %}, {% if form.email.errors %} und {% if form.password2.errors %}* werden die Error-Meldungen aufgerufen.

  
Abbildung: Error-Meldungen auf Registrierungsseite

## 2.5 Kontrollieren

### 2.5.1 Testfälle

|  |  |
| --- | --- |
| Bezeichnung |  |
| Eingabe |  |
| Erwartetes Resultat |  |
| Tatsächliches Resultat |  |
| Testperson |  |
| Datum und Uhrzeit |  |

## 2.6 Auswerten

## 2.7 Abschlussbericht

## Quellenverzeichnis

* https://erfolgreich-projekte-leiten.de/projektorganisation/
* https://djangobook.com/wp-content/uploads/MTV-Pattern.png
* https://djangobook.com/wp-content/uploads/Django-Models.png
* https://djangobook.com/authentication-forms-templates/
* https://django-auth-ldap.readthedocs.io
* http://ldap3.readthedocs.io
* https://simpleisbetterthancomplex.com/tutorial/2016/06/27/how-to-use-djangos-built-in-login-system.html

## Glossar