



Verwaltungsinterface für eine webbasierte Anwendung

Bachelorarbeit

von

Imane Nazih

geboren in

Ouedzem

eingereicht bei

Technik Sozialer Netzwerke

Jun.-Prof. Dr.-Ing. Kalman Graffi

Heinrich-Heine-Universität Düsseldorf

September 2016

Betreuer:

M.Sc. Tobias Krauthoff

Prof. Dr. Martin Mauve

Zusammenfassung

Im Rahmen von Online-Partizipation kommen für Argumentationen und Diskussionen verschiedene Software-Lösungen wie Foren, Pro- und Kontra-Listen und die Argumentationskarten zur Anwendung. Jeder dieser Ansätze bringt Vorteile mit, allerdings sie weisen mehrere Nachteile auf. Foren skalieren schlecht bei steigender Benutzer Anzahl. Bei Pro- und Kontra-Listen fehlen die Beziehungen zwischen einander. Der Argumentationskarten Ansatz benötigt Expertenwissen. Von daher ist zurzeit ein neuartiger Prototyp im Rahmen des Projekts 'Dialog-basierte Argumentations-Software' entwickelt, der auf Argumentationskarten basiert und gegenüber vorher erwähnten Lösungen Verbesserung verspricht, indem er eine echte Diskussion simuliert, die auf das Wissen der Diskussionsteilnehmer basiert. Da das Wissen in der Datenbank gespeichert und verwaltet wird, besteht die Notwendigkeit für ein manuelles Erstellen und Pflegen der Datenbank unter Berücksichtigung der verschiedenen Abhängigkeiten zwischen Datenbankeinträgen. Mit herkömmlichen Methoden wie per Kommandozeile ist die Wartung der Datenbank sehr anstrengend und kompliziert. Deshalb ist das Ziel dieser Arbeit, ein Admin Interface für die Erstellung und Pflege der Datenbanktabellen des Prototypen zu entwickeln.

Das Admin Interface wird mit dem Pyramid Web Framework implementiert. Es wird dabei das Konzept von Model-View-Controller verwendet, indem die Funktionalitäten des Interface in Backend und Frontend aufgeteilt werden. Während im Backend Anfragen anhand Datenbank Schnittstelle SQLAlchemy verarbeitet und daraus Antworten bereitgestellt werden, beschäftigt sich das Frontend mit der Darstellung der Antworten und der Navigation. Hierbei werden die Frameworks Bootstrap und CSS für die Gestaltung der Admin Webseite angewandt. Darüber hinaus wird das Javascript Framework für die Navigation und die Interaktion des Administrators mit dem Interface eingesetzt. Der Daten Austausch zwischen Frontend und Backend wird mittels AJAX Framework realisiert.

Die Admin Webseite wird mit einem Dashboard und einer Navigationsleiste versehen. Hier wird die Möglichkeit geben zu einem Bereich zu navigieren, Daten anzuzeigen, neue Daten Hinzuzufügen und bestehende Daten zu editieren.

Danksagung

An dieser Stelle möchte ich mich bei M.Sc. Tobias Krauthoff für die Vergabe des interessanten Themas und die hervorragende Betreuung, sowie für seine Anregungen in zahlreichen Diskussionen, Tipps und Hinweise bei der Anfertigung dieser Arbeit bedanken.

Bei Herrn Prof. Martin Mauve bedanke ich mich für die Übernahme des Erstgutachters.

Zudem möchte ich bei Herrn Prof. Kalman Graffi für die Übernahme des Zweitgutachters bedanken.

Ich möchte meinen herzlichen Dank gegenüber meine Familie aussprechen, besonders meinen Mann und meinen Sohn, die mich mit viel Geduld moralisch unterstützt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis	ix
Tabellenverzeichnis	xi
Abkürzungsverzeichnis	xiii
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel der Arbeit	1
1.3 Aufbau der Arbeit	2
2 Verwandte Arbeit	3
2.1 Django	3
2.2 Flask	7
2.3 Tornado	9
2.4 Bottle	10
2.5 Pyramid_sacrud	11
2.6 Websauna	12
2.7 Kotti	13
2.8 SubstanceD	14
3 Grundlagen	16
3.1 Pyramid	16
3.2 SQLAlchemy	19
3.3 Templating mit Chameleon	20
3.4 Javascript	22
3.5 Bootstrap	23
3.6 D-BAS	23
3.6.1 Argumentationskarten in D-BAS	24
3.6.2 Funktionalität von D-BAS	24
3.6.3 Datenmodells von D-BAS	25

3.6.4	Struktur von D-BAS	25
4	Verwaltungsinterface	28
4.1	Einführung	28
4.2	Struktur des Adminmoduls	28
4.2.1	Views	29
4.2.2	Hilfsfunktionen und Zugriff auf Datenbank	31
4.3	Funktionalitäten von Adminmodul aus Frontend Sicht	35
4.3.1	Aufbau der Admin Webseite	36
4.3.2	Interaktion des Admins mit der Webseite	40
5	Fazit und Ausblick	48
5.1	Fazit	48
5.2	Ausblick	49
	Literaturverzeichnis	51

Abbildungsverzeichnis

2.1	Django: Login	5
2.2	Django: Adminansicht	5
2.3	Django: Themenansicht	6
2.4	Django: Thema hinzufügen	6
2.5	Flask: Useransicht	8
2.6	Flask: User hinzufügen.	9
2.7	Flas: Thema hinzufügen	10
2.8	Pyramid_sacrud: Demo Webseite	11
2.9	Kotti: Demo Webseite	14
3.1	D-BAS: Datenmodell	26
3.2	dbas Struktur	27
4.1	Admin-Struktur	29
4.2	D-BAS: Admin Webseite	36
4.3	Admin-Ablaufdiagramm	37
4.4	Admin-Dashboard	38
4.5	Issue Bereich	41
4.6	Add Statement Popup Fenster	43
4.7	Edit Issue Popup Fenster	47

Tabellenverzeichnis

Abkürzungsverzeichnis

WSGI	Web Server Gateway Interface
MVC	Model-View-Controller
MVT	Model-View-Template
ORM	Object-relational Mapping
URL	Uniform Resource Locator
ZODB	Zope Object Database
D-BAS	Dialog-basierte Argumentations-Software
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
SQL	Structured Query Language
AJAX	Asynchronous Javascript And XML
HTML	Hypertext Markup Language
CMS	Content Management System
ACL	Access Control List
DOM	Document Object Model
CRUD	Create-Read-Update-Delete

Kapitel 1

Einleitung

In diesem Kapitel wird zuerst eine Motivation für das Thema der Bachelorarbeit Verwaltungsinterface für eine webbasierte Anwendung gegeben. Anschließend wird auf das Ziel dieser Arbeit eingegangen. Abschließend wird die Kapitelübersicht vorgestellt.

1.1 Motivation

Im Rahmen von Online-Partizipation sind für Argumentationen und Diskussionen verschiedene Software-Lösungen wie Foren, Pro- und Kontra-Listen und die Argumentationskarten eingesetzt. Jeder dieser Ansätze bringt Vorteile mit, allerdings weisen mehrere Nachteile auf. Foren skalieren schlecht, wenn die Anzahl der Benutzer steigt. Bei Pro- und Kontra-Listen fehlen die Beziehungen zwischen einander. Der Argumentationskarten-Ansatz benötigt Expertenwissen. Von daher ist zurzeit ein neuartiger Prototyp im Rahmen des Projekts 'Dialog-basierte Argumentations-Software' entwickelt, der auf Argumentationskarten basiert und die Nachteile der drei Ansätze vermeidet, indem er eine echte Diskussion mit dem Wissen der Teilnehmer simuliert, das in der Datenbank gespeichert wird. Deshalb besteht der Bedarf an einem manuellen Erstellen und Pflegen der Datenbank unter Berücksichtigung der verschiedenen Abhängigkeiten zwischen Datenbankeinträgen.

1.2 Ziel der Arbeit

Dieser Arbeit befasst sich mit der Entwicklung einer Web-Verwaltungsschnittstelle für die Erstellung und Pflege der Datenbanktabellen des Prototypen D-BAS inklusiv die Auflösung der Fremdschlüssel-Abhängigkeiten. Das implementierte Admin-Interface soll es am Ende ermöglichen, die Objekte

des Prototyp-Datenmodells in Tabellen darzustellen, wobei nur die berechtigten Administratoren die Tabellen Inhalte angezeigt bekommen dürfen. Darüber hinaus soll die Auflösung der Fremdschlüssel Abhängigkeiten zwischen Tabellen Einträgen die Komplexität der Verwaltung vereinfachen. Außerdem soll der Administrator in der Lage sein sowohl Daten in der Datenbank hinzuzufügen als auch zu editieren.

1.3 Aufbau der Arbeit

Um die aktuelle mögliche Lösungswege für die Entwicklung von einer Admin-Interface für Python Web Anwendungen darzustellen, werden zuerst im Kapitel Verwandtearbeit einige Python Web-Frameworks im Hinblick auf die Architektur, die Sicherheit und die Administration vorgestellt. Danach wird im Kapitel Grundlagen das Web-Framework Pyramid, auf das sich die vorliegende Arbeit setzt, ausführlicher dargestellt. Anschließend wird ein Überblick über die verwendeten Technologien wie SQLAlchemy, Bootstrap und Javascript gegeben. Am Ende des Kapitels wird der Prototyp D-BAS besprochen. Es wird hier eine Übersicht über die Argumentationskarten, die Funktionalitäten, das Datenmodell und die Struktur von D-BAS gegeben. Im Kapitel Verwaltungsinterface wird ein Überblick über die Struktur des Admin-Interface präsentiert. Danach werden die Funktionalitäten in Bezug auf Backend und Frontend genauer und ausführlicher erläutert. Das letzte Kapitel schließt die Arbeit mit einer Zusammenfassung und einem Ausblick.

Kapitel 2

Verwandte Arbeit

Die Python Web-Frameworks sind vielfältig. Aber nur wenige bieten ein Admininterface zur Verwaltung des Datenmodells an. In diesem Kapitel wird zuerst auf die Frameworks Django, Flask, Tornado und Bottle eingegangen und ihre wichtigen Merkmale gezeigt. Danach werden Frameworks wie pyramid_sacrud, Websauna, Kotti und SubstanceD vorgestellt, die auf Pyramid 3.1 basieren.

2.1 Django

Django ist ein Python Open Source Framework. Es ist das meist eingesetzte und bekannteste Python Framework. Eine der Stärken von Django ist die schnelle und saubere Web Entwicklung. Man kann sogar in Stunden sein Web Projekt aufsetzen. Da Django sich mit den Problemen der Web Entwicklung wie die Sicherheit und die Skalierbarkeit auseinandersetzt, kann sich der Entwickler auf die Logik seiner Web Anwendung konzentrieren [Fou16g]. Der Entwickler spart dadurch viel Zeit. Django basiert auf dem Prinzip D.R.Y(Don't Repeat Yourself) [Fou16a], das besagt, die Wiederholung von Code zu vermeiden indem man eine Anwendung in mehreren Projekten einbinden kann. Django setzt auf das Prinzip Model-View-Controller (MVC) beziehungsweise Model-View-Template (MVT) auf. Bei Django sind die Views Controller, weil die Views die Context-Objekte(Dictionary) berechnen, rendern sie in Templates und geben die gerenderte Templates in Django Response Objekt zurück [Fou16b]. Templating bestimmt wie diese Daten angezeigt werden. Django gibt uns die Möglichkeit ein großes Projekt in einzelnen Anwendungen einzugliedern [Fou16f]. Dies hat den Vorteil, dass eine Anwendung in mehr als einem Projekt verwendet werden kann. In Django ist Templating ein hierarchisches System, es gibt ein Basistemplate base.html [Fou16e], in dem man mehrere zu erweiternden Blöcke definieren kann. Das Basis Template kann eine einfache HTML Datei sein. Dazu werden andere Templates Dateien angelegt um die Basis Datei zu erweitern und die jeweiligen leeren Blöcke zu füllen. Django setzt reguläre Ausdrücke um Uniform Resource Locator (URL) auf Code (View) abzubilden

ein [Fou16f]. Wie im Listing 2.1 gesehen werden kann, werden die URLs für die Anwendung mit regulären Ausdrücken konfiguriert. Mit dem ersten Ausdruck wird die URL (`http://server:port/`) zu der Startseite konfiguriert. Der zweite Ausdruck zeigt auf den Nachrichtbereich.

```
urlpatterns = patterns('',
    (r'^$', 'haupt_seite'),
    (r'^=nachricht/$', 'nachricht_seite')
)
```

Listing 2.1: Beispiel für URL-Konfiguration

Django stellt ein Object-relational Mapping (ORM) für die Datenbanksysteme SQLite, MySQL, PostgreSQL, Oracle, MSSQL zur Verfügung. Mit dem ORM kann man mit einer kleinen Anzahl von Zeilen Code in der Datei `models.py` ein Datenmodell definieren [Fou16c]. Django erzeugt die Datenbankabfragen (SQL) für die Erstellung der Tabellen und legt dann die Tabellen in der Datenbank an. Im Listing 2.2 sieht man, wie das Datenmodell einer Beispiel-Anwendung zu definieren ist. Im Datenmodell sind zwei Modell-Klassen `Sprache(Language)` und `Thema(Issue)` mit Attributen definiert. Die Model-Klasse `Thema` besitzt zusätzlich ein Fremdschlüssel-Attribut, das auf die Modell-Klasse `Sprache` zeigt.

```
from django.db import models
from django.contrib.auth.models import User

class Language(models.Model):
    name = models.CharField(max_length=15)
    ui_locales = models.CharField(max_length=15)

class Issue(models.Model):
    title = models.CharField(max_length=200)
    info = models.CharField(max_length=1000)
    user = models.ForeignKey(User)
    date = models.DateTimeField('date')
    lang = models.ForeignKey(Language)
```

Listing 2.2: `models.py`

Was Django attraktiver und einzigartig macht, ist die automatische Generierung eines Admin-Interface [Fou16d], das sich für die Verwaltung der Web Anwendung als sehr nützlich und unverzichtbar erweist. Es muss keine Zeile Code geschrieben werden und wird am Ende ein tolles Tools geschenkt bekommen. Beim Starten des Projekts wird die Standard Anwendung Admin aktiviert. Dann muss nur das Datenmodell des Projekts im Modells definiert sein (Siehe Listing 2.2). User und Gruppe werden

automatisch generiert. Die URL zur Admin Webseite ist standardmäßig konfiguriert. Nachdem Starten des Django Webservers kann die Webseite mit der URL `http://127.0.0.1:8000/admin` aufgerufen werden. Django stellt ein sicheres Authentifizierungssystem zur Verfügung um den Inhalt einer Webseite vor fremdem Zugriff zu schützen. Bevor ein Benutzer auf den Inhalt einer Webseite zugreifen kann, muss er sich als autorisierter Benutzer einloggen (Siehe Abbildung 2.1).

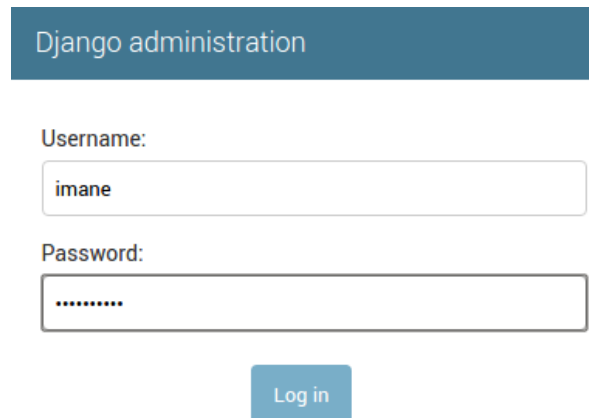


Abbildung 2.1: Django: Login

Nach dem Login gelangt man zur Adminansicht. Hier kann man zu einem Datenmodell der Anwendung navigieren und dieses verwalten (Siehe Abbildung 2.2).

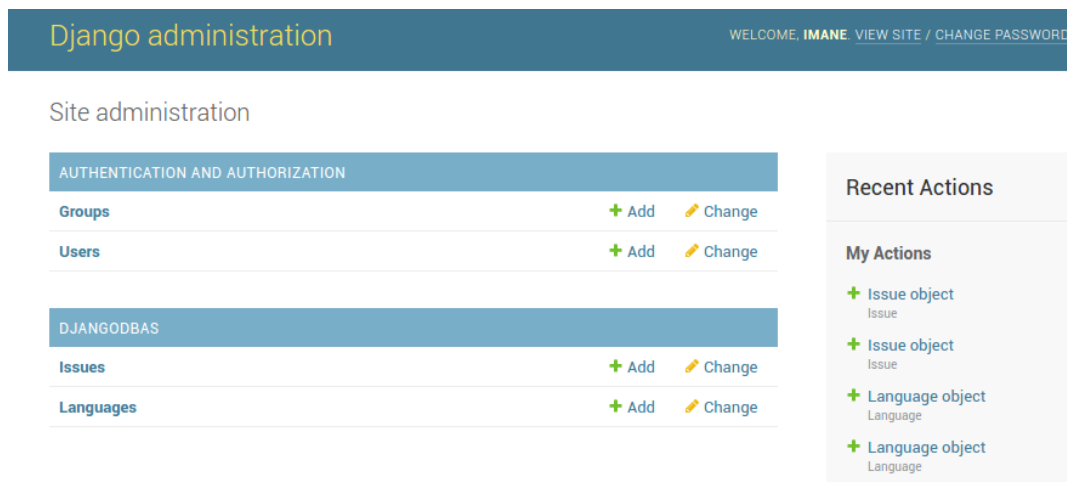


Abbildung 2.2: Django: Adminansicht

Die Abbildung 2.3 zeigt alle in der Datenbank vorhandenen Themen(Issues). Man kann hier ein Thema anzeigen, hinzufügen und editieren.

Wenn auf den ADD Button geklickt wird, wird ein Formular zum Hinzufügen eines neuen Themas

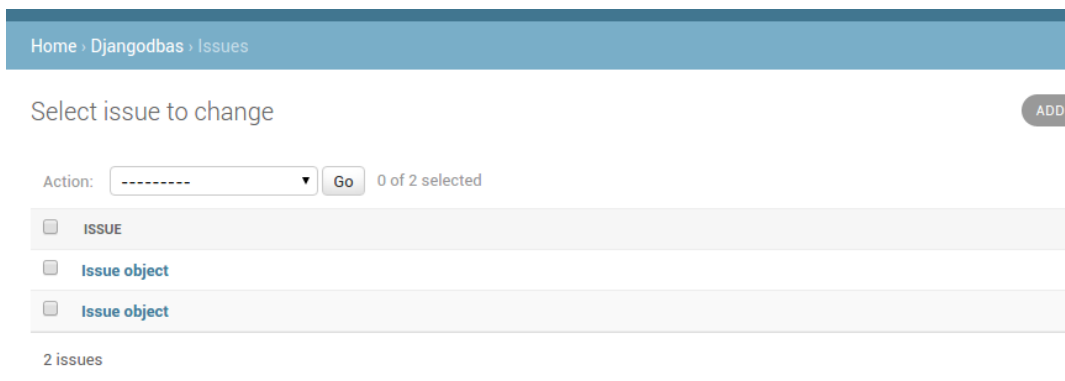


Abbildung 2.3: Django: Themenansicht

angezeigt. Django erkennt die Fremdschlüssel-Abhängigkeit zwischen Thema und User, zwischen Thema und Sprache aus dem Datenmodell und erstellt daraus ein Dropdown-Menü (siehe Abbildung 2.4). Es gibt die Möglichkeit einen bestimmten User und eine Sprache auszuwählen. Wie hier zu sehen ist, generiert Django alles für uns. Wir müssten nur das Datenmodell angeben.

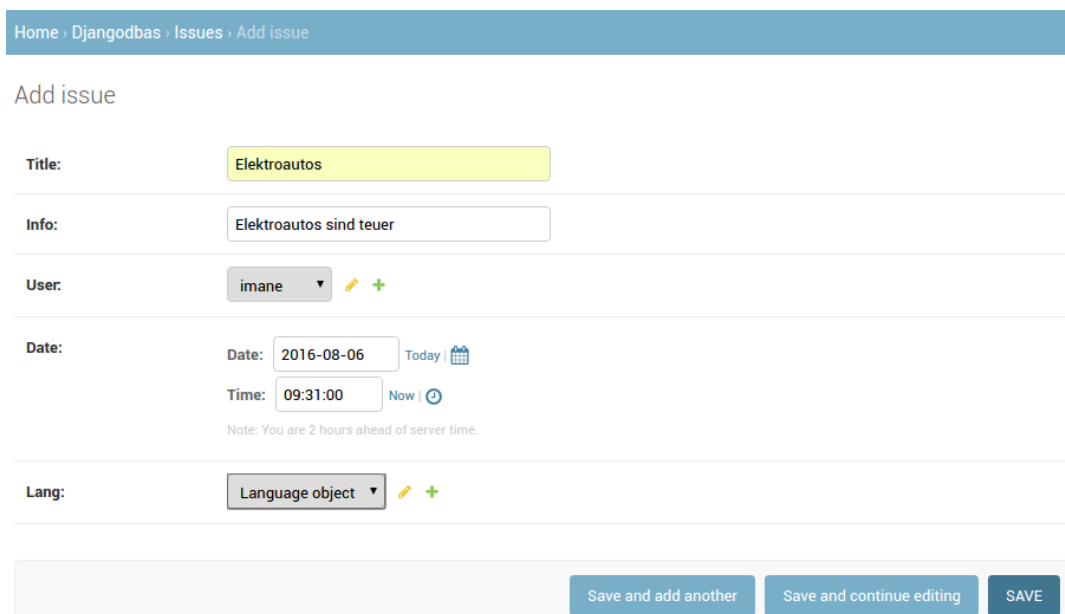


Abbildung 2.4: Django: Thema hinzufügen

2.2 Flask

Flask ist ein Python Mikroframework. Sein Ziel ist Web Anwendungen mit wenigem Aufwand und mehr Freiheit zu entwickeln. Es bietet die am häufigsten verwendeten Kernkomponenten eines Web Applikation Frameworks wie URL Routing, Templating, Anfrage- und Antwort Objekt. Im Gegenteil zu Django verfügt Flask über keine ORM Bibliothek. Es bleibt dem Entwickler überlassen die passende Bibliothek auszuwählen. Flask verwendet zwei externe Bibliotheken, die jinja2 Template-Engine und die WSGI Bibliothek. Die jinja2 Template-Engine kann textbasierte Dateien wie HTML, XML, CSV und LaTeX generieren. Die WSGI Bibliothek dient als Schnittstelle zwischen dem Web Server und der Web Anwendung. Flask ist gut dokumentiert und getestet [Ron16]. Im Gegensatz zu Django wird Flask mit keinem Adminmodul ausgeliefert. Um Flask attraktiver gegenüber anderen Python Web Frameworks zu machen wurde die Flask-Admin Bibliothek eingeführt. Diese ermöglicht das Hinzufügen eines Admin-Interface um die Verwaltung des Datenmodells der Web Anwendung zu erleichtern [Kov16]. Das Listing 2.3 zeigt eine kleine Flask Anwendung:

```
db = SQLAlchemy(app)
admin = Admin(app, name='myapp2', template_mode='bootstrap3')

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    firstname = db.Column(db.String(80))
    lastname = db.Column(db.String(80))
    publicname = db.Column(db.String(80), unique=True)
    email = db.Column(db.String(120), unique=True)

    def __repr__(self):
        return self.publicname

class Issue(db.Model):
    __tablename__ = 'issues'
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.String(80))
    info = db.Column(db.String(80))
    date = db.Column(db.DateTime)

    user_id = db.Column(db.Integer(), db.ForeignKey(User.id))
    user = db.relationship(User, backref='issues')
```

```
def __unicode__(self):
    return self.title

# Add administrative views here
admin.add_view(ModelView(User, db.session))
admin.add_view(ModelView(Issue, db.session))

db.drop_all()
db.create_all()
app.run(debug=True)
```

Listing 2.3: Beispiel einer Flask Anwendung

Zuerst wird ein Admin Interface durch einer Instanziierung der Klasse Admin hinzugefügt. Danach wird das Datenmodell definiert. SQLAlchemy setzt es in Tabellen in der Datenbank um. Zum Schluss werden mit Hilfe von Model Views Admin Webseiten hinzugefügt. Jeder Webseite dient zur Verwaltung einer Datenmodell-Klasse in der Datenbank. Im Vergleich zu Django generiert Flaskadmin automatisch keine User- oder Gruppe-Webseite sondern es generiert nur was man als Datenmodell angegeben hat. Im Listing 2.3 sind User und Thema (Issue) angegeben. Der Flaskadmin generiert neben dem Haupt-Bereich auch zwei andere Bereiche, ein für User und ein für Thema. Abbildung 2.5 zeigt dies. Beim Starten der Anwendung und Aufruf der Webseite `http://localhost:5000/admin/` wird die Admin Webseite mit einer Navigationsbar angezeigt. Dort kann zu User oder zu Thema navigiert werden. Wie in der Abbildung 2.5 zu sehen ist, kann im User-Bereich User angezeigt, editiert, hinzugefügt oder gelöscht werden.

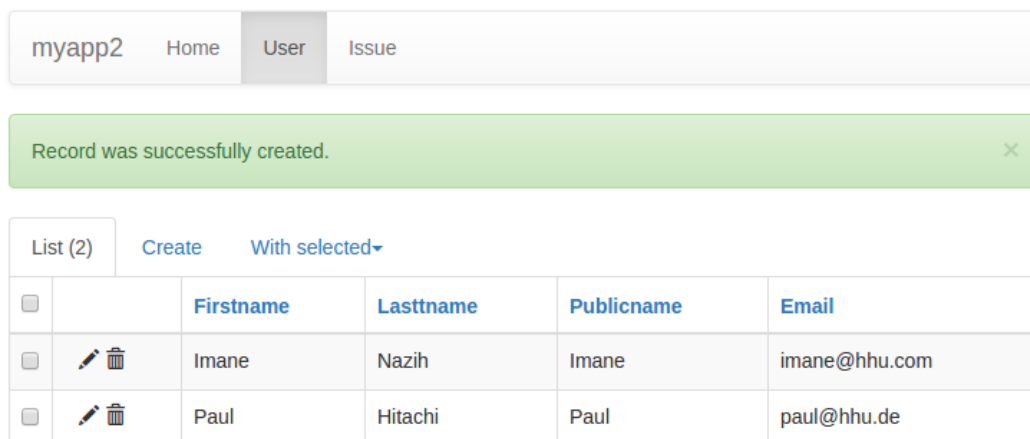


Abbildung 2.5: Flask: Useransicht

Wenn auf dem Link Create geklickt wird, öffnet sich ein Fenster mit einem Formular zum Hinzufügen

eines Users. Siehe Abbildung 2.6.

The screenshot shows a web interface for creating a user. At the top, there is a header with the text 'myapp2' and a menu icon. Below the header, there are two buttons: 'List' and 'Create'. The main form consists of several input fields: 'Firstname' with the value 'Imane', 'Lastname' with the value 'Nazih', 'Publicname' with the value 'Imane', and 'Email' with the value 'imane@hhu.com'. There is also an empty 'Issues' field. At the bottom of the form, there are five buttons: 'Save', 'Save and Add Another', 'Save and Continue Editing', and 'Cancel'.

Abbildung 2.6: Flask: User hinzufügen

Die Abbildung 2.7 zeigt ein Formular zum Hinzufügen eines Themas. Hier gibt es die Möglichkeit über ein Dropdown-Menü einen User auszuwählen, da Thema vom User abhängig ist. Das ist dem Flaskadmin zu verdanken. Er hat automatisch aus der Fremdschlüssel-Abhängigkeit im Datenmodell von Thema zu User ein Dropdown-Menü erstellt. Wie hier zu sehen ist, muss nur Gedanken über Datenmodell gemacht werden und der Rest wird von Flaskadmin erstellt ohne dabei eine einzige View oder HTML Seite zu schreiben.

2.3 Tornado

Tornado ist ein Python Web Framework zur Entwicklung von Web Anwendungen. Es ist auch eine Netzwerk Bibliothek, die entworfen wurde um asynchrone Operationen zu verarbeiten. Dies ermöglicht den Servern viele Verbindungen offen zu halten. Diese Eigenschaft macht Tornado sehr beliebt von Web Anwendungen, die lange offene Verbindungen zum User benötigen. Tornado enthält eine einfache und schnelle Template-Engine. Man kann auch andere Python Template-Engines verwenden.

myapp2 Home User Issue

List Create

User
Imane

Text
Cat or Dog

Info
Cat Or Dog

Date
2016-08-05 22:33:00

Save Save and Add Another Save and Continue Editing Cancel

Abbildung 2.7: Flask: Thema hinzufügen

den. Im Tornado ist kein ORM vorhanden. Tornado läuft auf einem eigenen Server, es braucht kein WSGI. Um Tornado Anwendungen auf einem Server laufen zu lassen, die WSGI benötigen, konvertiert man die Anwendungen mit dem `tornado.wsgi.WSGIAdapter` [Aut16]. Im Gegenteil zu Django und Flask bietet Tornado kein Admin Interface.

2.4 Bottle

Bottle ist ein einfaches Python Mikroframework. Es ermöglicht die schnelle und leichte Entwicklung von WSGI Anwendungen. Es wird mit einem einzigen Dateimodul ausgeliefert. Bottle verfügt über eine einfache Template-Engine und unterstützt andere Templates wie Jinja2 und mako. Es bietet Funktionen wie statische und dynamische Routing, Hochladen von Dateien, den Zugriff auf HTML-Formular Daten und Header an. Bottle Framework ist durch Plugins erweiterbar. Plugins sind externe Bibliotheken, die zusätzliche Funktionen anbieten, die nicht in Bottle enthalten sind. Zum Beispiel das `SQLitePlugin` bietet den Zugriff auf die `SQLite` Datenbank [Hel16]. Es verfügt über kein Admin Interface.

2.5 Pyramid_sacrud

Das Pyramid Framework, dem diese Arbeit zugrunde liegt, bietet kein Admin Interface Out-of-the-box an. Es gibt aber diverse Open Source Pakete, die das Pyramid um ein Admin Interface erweitern, unter anderem pyramid_sacrud, Websauna, Kotti und SubstanceD. Das Paket pyramid_sacrud bietet ein Administratives Webinterface für Pyramid an. Gegenüber dem klassischen Create-Read-Update-Delete (CRUD) erlaubt das Paket eine individuelle und flexible Anpassung des Webinterfaces genauso wie das Django Admin Paket. Eine seiner Stärken ist die neue Architektur ab Version 0.3.0, die die Integration mit verschiedenen Backends ermöglicht um die Ressourcen bereit zu stellen, indem das Paket mehrere Module als Schnittstelle zwischen Views und Ressourcen (Backend) zur Verfügung stellt. Einer diese Module ist ps_alchemy. Es kann SQLAlchemy Ressourcen verarbeiten. Das zweite Modul ist ps_djangoorm, das mit Django ORM umgehen kann. Das Universal Module erlaubt sogar ein eigenes Backend zu schreiben [ura15a]. Für die Arbeit mit diesem Paket wird die Bibliothek pyramid_sacrud benötigt. Die eingesetzte Template Engine ist Jinja2. Das Paket bietet auf seiner Webseite ein Startbeispiel an [ura15b]. Es benutzt das Backend ps_alchemy. Das Beispiel kann um eigene Datenmodelle angepasst werden und stellt in wenigen Minuten ein fertiges Admininterface zur Verfügung. Das Paket bietet die Standard Operationen an. Hier können neue Daten hinzugefügt, aktualisiert, angezeigt oder entfernt werden. Auf der Webseite 'runnable.com' kann eine Demoseite für Pyramid_sacrud gefunden werden. Siehe Abbildung 2.8.

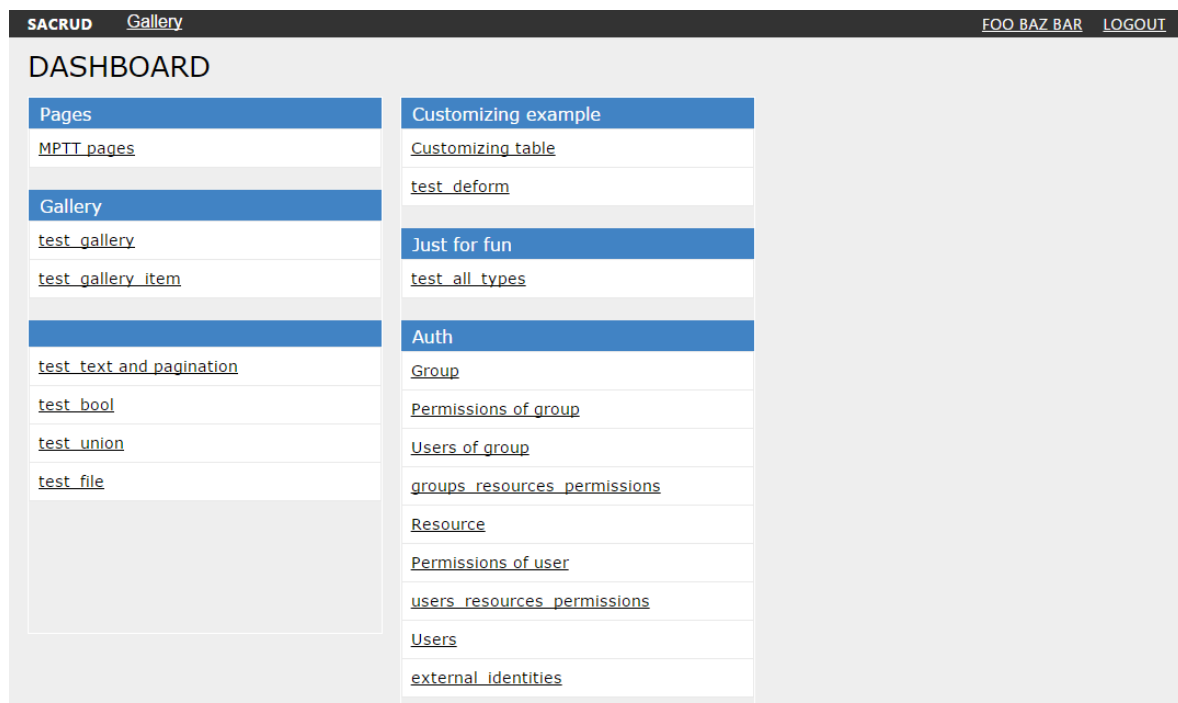


Abbildung 2.8: Pyramid_sacrud: Demo Webseite

2.6 Websauna

Websauna ermöglicht die Entwicklung von skalierbaren und modernen Webanwendungen. Es bietet eine effiziente Herangehensweise an um auftretende Probleme bei der Umsetzung einer Geschäftslogik zu meistern. Infolgedessen können die Entwickler ihre Produktivität schnell steigern und ihre eigenen Ideen einbringen. Websauna verspricht mehr Flexibilität durch Codeminimierung und Verwendung bestehender Software. Es bietet eine verfeinerte Integration vom Pyramid Web Framework, SQLAlchemy ORM, Alembic Datenbank Migration Skripte, IPython Kommandozeile, föderierte Identität und Jinja Templating Engine an. Der Schwerpunkt von Websauna liegt auf Webseiten mit administrativem Backend und Webportale. Außerdem fokussiert es auf die Sicherheit und toleriert menschliche Fehler. Es schützt den Entwickler gegen Schwachstellen wie SQL-Injection und Cross-Site-Scripting. Daher ist das Framework ein guter Kandidat für die online Finanzdienstleistungen und Ecommerce. Websauna bietet Tools für die automatische und manuelle Datenbankmigrationen und Backup an. Websauna kommt mit einem Standard- SQLAlchemy Benutzer- und Gruppenmodell. Die Webseiten Templates basieren auf Bootstrap, das beliebteste HTML, CSS und JS Framework. Websauna lässt auch die Möglichkeit ein eigenes Frontend frei auszuwählen. Es unterstützt verschiedene Benutzerquellen wie Website mit eigenem Anmeldeformular oder Facebook Login. Benutzer können auch intern in einer SQL Datenbank gespeichert werden. Außerdem sind Admin Views erweiterbar und flexibel. Das Framework ist sehr gut dokumentiert so dass keine Frage offen bleibt [Oht16].

Websauna bietet eine automatische Generierung von Admin Webinterface an, das die Manipulation von Datenmodell vereinfacht. Indem es Datenbrowser und Editoren für die Durchführung der Grundoperationen (CRUD) automatisch generiert. Daher könnte es auf die manuelle Entwicklung von Admininterface verzichtet werden. Gegenüber anderen Frameworks können Websauna Webseiten mehrfache Admininterfaces haben. Ein für die Superadmin, ein für die Entwickler und ein für die Kunden. Das Hauptziel von Websauna ist flexibel zu sein, es zwingt kein besonderes URL Muster zu verfolgen. Das Adminmodul basiert auf Traversal. Jeder Admin Endpunkt (Endpoint) kann seine eigene Hierarchie von Pfaden definieren. Jedes Datenmodell wird einer Admin Ressource von der Klasse `websauna.system.admin.ModelAdmin` zugeordnet. Diese Ressource ist zuständig für das Auflisten und Hinzufügen von neuen Objekten. Jede Datenmodell Instanz (eine Zeile in SQL Datenbank) wird einer Admin Ressource von der Klasse `websauna.system.admin.ModelAdmin.Resource` zugewiesen. Diese Ressource ist verantwortlich für das Anzeigen, Editieren und Löschen eines Objekts. Admin Interface stattet Websauna Seiten mit Super Administrator Möglichkeiten aus. Hierbei kann leicht navigiert, Daten hinzugefügt und editiert werden ohne explizit ein Formular und Views zu programmieren oder zusätzliche Software für Datenbank Verwaltung zu installieren. Admin Interface ist zugänglich für Benutzer, die zu Admin Group gehören. Diverse Berechtigungen Schemas können anhand Access Control List (ACL) implementiert werden, so dass Group von Benutzer nur einen Teil der Daten anzeigen können oder gar nicht modifizieren dürfen. Datenmodell müssen explizit im Admin Interface registriert sein. Für jedes Modell im Admin Interface muss die dazugehörige Ressource

Klasse in der 'admins.py' Datei erstellt sein. Die Ressource repräsentiert den Traversal URL Teil und die dazugehörige Views.

Um mit Websauna arbeiten zu können, muss zuerst die folgenden Bibliotheken geschaffen werden, ab Python 3.4, ab PostgreSQL 9.4, libxml2 und Redis. libxml2 ist XML C parser. Redis ist ein In-Memory-Datenstruktur Speicher, er wird als Datenbank oder Cache genutzt. Wenn alle Abhängigkeiten da sind, kann mit der Erstellung des Webanwendung Projekts begonnen werden. Websauna bietet zwei Scaffolds Befehle websauna_app und websauna_addon für das Anlegen eines Projekts an. Wenn das Projekt erstellt ist, wird die Datenbank vorbereitet. Zuerst wird sie angelegt. Deren URL in 'development.ini' konfiguriert. Als nächster Schritt wird ein Migration Skript erstellt und mit dem Migrationstool Alembic ausgeführt. Migration Skript hat zwei Vorteile. Es wird einmal erstellt und kann gegen beliebig viele Datenbanken ausgeführt werden. Außerdem ändert sich die Datenbank ständig. Dabei können die Änderungen als Migration Skript aufgefasst werden. Danach wird das Datenmodell in der Datei 'models.py' definiert. Hier werden Tabellen in Form von Klassen aufgefasst werden. Anschließend wird der initiale Webseite Administrator angelegt. Websauna bietet einen integrierten IPython Notebook Kommandozeile Button auf der Admin Webseite für die Interaktion mit der Datenbank an. Hierbei können neue Daten wie Benutzer in der Datenbank hinzugefügt werden. Abschließend werden Daten und Modellmethoden per IPython Notebook Kommandozeile hinzugefügt. Dies erstellt eine Benutzerschnittstelle für die Manipulation der Daten auf der Webseite.

2.7 Kotti

Kotti ist ein Python Web Framework basiert auf Pyramid und SQLAlchemy. Es verbindet Pyramid mit Content Management System (CMS). Es enthält ein erweiterbares Out of The Box CMS namens Kotti CMS. Das Framework ist benutzerfreundlich und leichtgewichtig. Es ermöglicht eine CMS Anwendung zu entwickeln, die die meist wichtige Funktionalitäten enthält, wie Arbeitsablauf Automatisierung (Workflows) und Sicherheitsschutz dank einer komplexen Sicherheitsanforderungen. Es bietet eine moderne Benutzer und Rechte Verwaltung an, die intuitiv und skalierbar ist. Es arbeitet mit hierarchischen Daten. Das Framework setzt den WYSIWYG Editor ein. Daten können editiert werden wo sie erscheinen. Kotti Oberfläche bauen auf Bootstrap, CSS und JavaScript auf. Daher ist es geeignet für Desktop-PC als auch für mobile Endgeräte. Das Framework setzt Internationalisierung um, so dass die Webseiten komplett übersetzt sind. Es setzt Python 2.6 or 2.7 voraus und setzt Waitress Pyramid WSGI Server als Standard Server ein. Kotti kann mit PostgreSQL, MySQL and SQLite betrieben werden. Das Standard Datenbanksystem ist SQLite. Das erleichtert die initiale Entwicklung mit Kotti. Das Framework bietet den Entwicklern eine solide Basis an um verschiedene Typen von Webanwendungen zu bauen, die entweder die integrierte CMS erweitern oder ersetzen. Entwickler können Views, Vorlagen, Inhaltstypen, Zugriffskontrolle, Arbeitsabläufe anhand klar definierten Pro-

grammierschnittstelle hinzufügen und modifizieren. Hierbei können auch Pyramid und SQLAlchemy Bibliotheken genutzt werden. Komponente wie Colander und Deform zum Arbeiten mit Formularen und Chameleon für Hypertext Markup Language (HTML) Templating werden eingesetzt, sind sie aber nicht eingefordert. Nachdem Kotti per virtualenv installiert wurde, kann wie folgt eine Kotti Webseite erstellt werden: Als erster Schritt wird ein Add-On Paket angelegt und mit der Webseite registriert. Kotti Add-On ist ein Python Paket. Danach wird das Add-On Paket innerhalb der Webseite installiert. Als Zweiter Schritt wird Das Paket mit kotti.configurators konfiguriert. Beim dritten Schritt wird das Datenmodell hinzugefügt. Als vierter Schritt werden Template und Views für die Verwaltung des Datenmodells addiert. Beim letzten Schritt werden die Datenmodell Klassen referenziert [dev14]. Kotti bietet auf seiner Webseite eine Demo Seite an (siehe Abbildung 2.9), wo eingeloggt werden kann, Inhalten hinzugefügt und editiert werden. Benutzer können auch dort verwaltet werden.

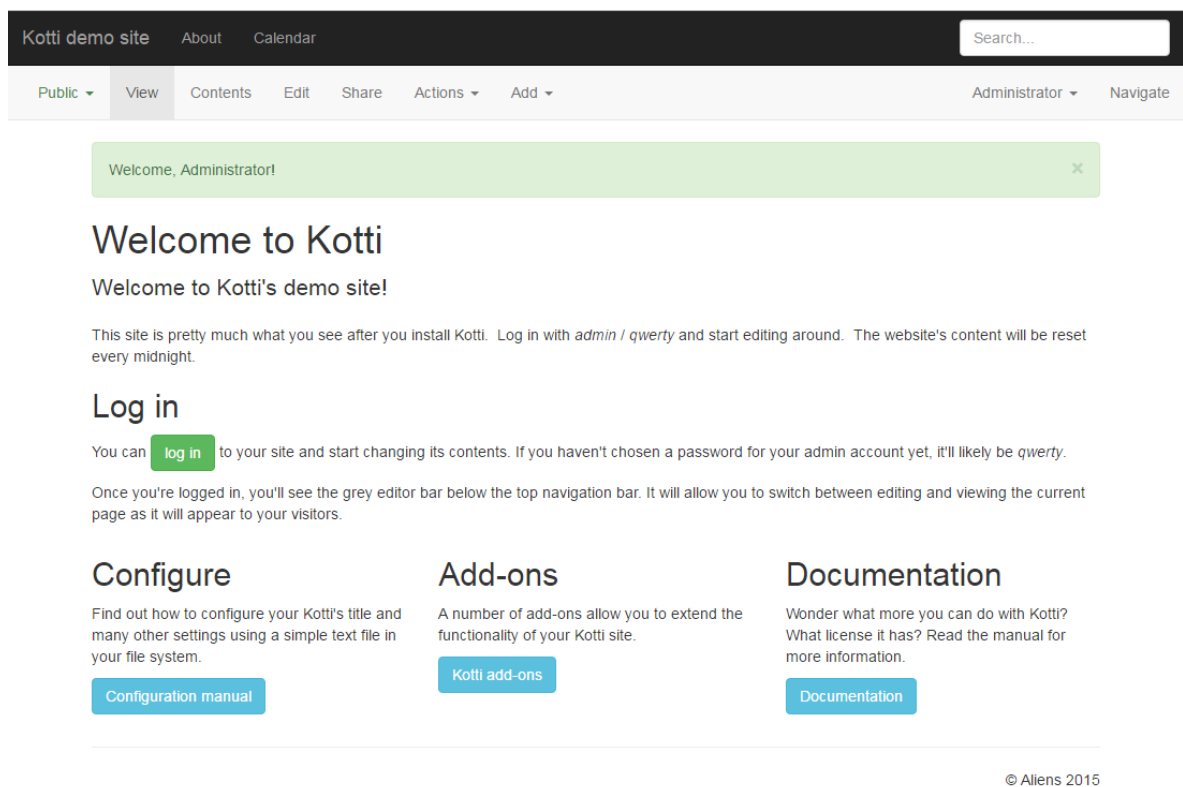


Abbildung 2.9: Kotti: Demo Webseite

2.8 Substanced

Substanced ist ein Anwendung Server basierend auf Pyramid. Er benutzt Zope Object Database (ZODB) um Daten zu speichern. ZODB ist eine Native Object Datenbank für Python [Fou11]. Er

besitzt eine integrierte Benutzer und Gruppen Verwaltung und ein Admin Interface, das Hinzufügen und Editieren von Daten ermöglicht. SubstanceD gibt die Möglichkeit eigenes Datenmodell mit Abhängigkeiten zueinander zu definieren. Er erlaubt sogar die letzten Aktionen im Admin Interface zurückzunehmen. SubstanceD liefert auch ein interessantes Hilfsmittel und zwar den kompletten Inhalt der Webseite auf dem Dateisystem zu speichern und vom Dateisystem in der Webseite hochzuladen. SubstanceD setzt Colander und Deform für die Verarbeitung von Formularen ein [CON16d]. Um ein SubstanceD Admin Interface einzurichten, werden zuerst Python python2.7 und das Paket SubstanceD installiert. Danach wird das Projekt anhand Scaffold SubstanceD erstellt [CON16c]. Anschließend werden die Daten in Form von Klassen definiert und registriert. Daten bei SubstanceD sind Inhalte (Content) [CON16b]. Das gilt auch für Benutzer und Gruppen. Schließlich werden die Management Views definiert. Management Views können nur von Admin angesprochen werden, nicht von anderen normalen Benutzern [CON16e]. SubstanceD führt auf der Webseite <http://demo.substanced.net> eine Demoanwendung vor, die ein echtes Admin Interface simuliert.

Kapitel 3

Grundlagen

Nachdem im vorigen Kapitel die bekanntesten Python Web Frameworks und deren Möglichkeiten ein Admin Interface anzubieten diskutiert wurden, geht dieses Kapitel auf die Grundlagen ein, auf denen diese Arbeit basiert. Zuerst wird das Pyramid Web Framework vorgestellt. Danach wird SQLAlchemy kurz beleuchtet, anschließend wird das Template mit Chameleon erläutert. Javascript und Bootstrap werden ebenfalls hier präsentiert. Abschließend wird eine Einführung über D-BAS gegeben.

3.1 Pyramid

Pyramid ist ein Python Open Source Framework. Es ist minimal, da es sich auf die Lösung der wesentlichen Probleme der Entwicklung einer Web Anwendung konzentriert, wie zum Beispiel die URL Abbildung, das Templating und die Sicherheit. Im Gegenteil zu Django wird bei der Entwicklung von Pyramid Web Anwendungen mit minimalen Bibliotheken gestartet, die den Pyramid-Kern bilden. Wenn weitere Bibliotheken benötigt werden, können diese immer nachträglich hinzugefügt werden. Dies hat den Vorteil, dass sich die Entwickler nicht von Anfang an mit vielen Bibliotheken beschäftigen sondern sich auf die Logik ihrer Anwendung konzentrieren können und neue Bibliotheken schrittweise, nach Wunsch und nach Bedarf verwenden. Um den Entwicklern das nachträgliche Ausrüsten zu vereinfachen, werden Pyramid Add-ons zur Verfügung gestellt. Diese externen Bibliotheken bieten Funktionen an wie das Hinzufügen von ORM, das Senden von Emails und das Verwenden einer Template-Engine. Dazu ist Pyramid ein dokumentiertes, getestetes, stabiles und ein schnelles Framework [pyr16c]. Durch alle diese Vorteile ist Pyramid eine gute Wahl für die Entwicklung von Web Anwendungen. Pyramid ist auch geeignet für die Entwicklung von kleinen Anwendungen, da eine Pyramid Anwendung mit einer einzigen Python Datei geschrieben werden kann(Siehe Listing 3.2).

Pyramid bietet mehrere Gerüste(Scaffolds) zum Generieren eines Projekts für die Entwicklung einer

Web Anwendung an, unter anderen `pyramid_starter`, `pyramid_zodb` und `pyramid_alchemy`. Das `pyramid_starter` Gerüst erstellt ein Pyramid Projekt mit URL Dispatch als URL Abbildung Mechanismus. Das `pyramid_zodb` Gerüst erstellt ein Pyramid Projekt mit Traversal als URL Abbildung Mechanismus und ZODB zum Speichern von Daten. Das `pyramid_alchemy` Gerüst generiert ein Pyramid Projekt mit URL Dispatch als URL Abbildung Mechanismus und SQLAlchemy zum Speichern von Daten. Das generierte Projektverzeichnis enthält mindestens ein Python Paket und andere Dateien zum Ausführen, Testen und zur Beschreibung der Anwendung. eines der Pakete stellt die Anwendung vor. Im Gegenteil zu Django enthält ein Pyramid Projekt eine einzige Anwendung. Pyramid zwingt die Entwickler die Anwendungslogik innerhalb eines Pakets zu schreiben, da es ein Paket leichter mit neuem Code zu erweitern und die Anwendung mühelos zusammenzustellen ist. Das Listing 3.1 stellt ein Beispiel eines Pyramid Projekts namens MyProject dar.

```
MyProject /
|-- CHANGES.txt (beschreibt die Aenderungen, die man vorgenommen hat)
|-- development.ini (Konfigurationsdatei waehrend der Entwicklung)
|-- MANIFEST.in
|-- myproject
|   |-- __init__.py (gibt an dass, myproject ein Python Paket ist)
|   |-- static
|   |   |-- pyramid-16x16.png
|   |   |-- pyramid.png
|   |   |-- theme.css
|   |   |-- theme.min.css
|   |-- templates
|   |   |-- mytemplate.pt
|   |-- tests.py
|   |-- views.py
|-- production.ini (Konfigurationsdatei fuer die Produktion)
|-- README.txt
'-- setup.py (dient zum Testen und Zusammenstellen der Anwendung)
```

Listing 3.1: Beispiel eines Pyramid Projects

Das Beispiel Projekt ist mit dem Pyramid Bootstrapping Werkzeug `pcreate` erzeugt. Alle Pyramid Projekte, die mit `pcreate` erzeugt worden sind, haben die gleiche Struktur. Das Beispiel enthält ein einziges Python Paket namens `myprojekt`, das die Anwendung repräsentiert [pyr16b].

Wie Django und Flask unterstützt Pyramid die WSGI [wsg16]. Das WSGI ist eine Spezifikation, die beschreibt, wie ein Web Server und einer Web Anwendung miteinander kommunizieren [Net15]. Im Listing 3.2 ist ein Beispiel einer einfachen Pyramid WSGI Anwendung vorgestellt [pyr16c].

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello %(name)s!' % request.matchdict)

if __name__ == '__main__':
    config = Configurator()
    config.add_route('hello', '/hello/{name}')
    config.add_view(hello_world, route_name='hello')
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Listing 3.2: Beispiel einer einfachen Pyramid WSGI Anwendung

Die main Methode wird ausgeführt wenn diese Pyramid Anwendung ausgeführt wird. Als erster Schritt wird das Konfigurator Objekt erzeugt, dieses fügt Routing und Views Konfigurationen hinzu. Dann wird eine WSGI Anwendung erzeugt und der Server wird mit ihr gestartet. Die WSGI Anwendung fungiert als Vermittler zwischen dem Server und der Pyramid Anwendung.

Wenn die Anwendung über einen Browser besucht wird, muss dort eine URL eingegeben werden. Nachdem der Server diese URL in Form einer Anfrage bekommen hat, leitet er diese an die WSGI Anwendung weiter. Diese setzt die URL Dispatch als Methode ein um eine passende View zu finden. Diese Methode basiert auf Muster um die URL auf View abzubilden. Aus der URL und dem Muster, das in der Routingkonfiguration definiert ist (siehe Listing 3.2), wird eine Route bestimmt. Jede Route ist mit einer View assoziiert. Eine View ist ein aufrubares (callable) Python Objekt (eine Funktion oder eine Klasse), welches eine Anfrage akzeptiert und eine Antwort zurückgibt. Die WSGI Anwendung übergibt die Antwort an den Server, der sie wiederum an den Benutzer weiterleitet [pyr16d].

In Pyramid gibt es zwei Arten von Anwendungskonfigurationen, die imperative und die deklarative. In dem Beispiel vom Listing 3.2 wird die erste Variante verwendet. Während diese auf die Verwendung von einfachen Python Anweisungen mit dem Konfigurator Objekt basiert, setzt die zweite Variante die Konfiguration mit einem Dekorator wie `View_config` ein [pyr16a].

Um die Anwendung besser zu strukturieren verwendet Pyramid das Templating. Die dynamischen Daten, die von Views bereitgestellt werden, stehen in einer Vorlage (Template) als Variablen sowie spezielle Anweisungen neben den statischen Daten. Pyramid bietet Templating anhand `jinja2`, `Mako` und `Chameleon` an. In Pyramid sind diese Template-Systeme mit Renderer verbunden. Mit dem Ren-

derer wird das Antwort Objekt als Dictionary von der View zurückgegeben. Das Templating System wird die Anweisungen innerhalb des Templates auswerten und mit Daten aus dem Dictionary ersetzen [pyr16c].

Bei Pyramid spielt die Sicherheit eine große Rolle. Es bietet ein Sicherheitssystem, Das in zwei flexible Systeme unterteilt ist, das Authentifizierung- und das Autorisierung-System. Die beiden Systeme kommunizieren mit einander durch Identitätskennung eines Benutzers oder einer Gruppe. Einerseits wird die Identitätskennung über den angemeldeten Benutzer/Gruppe mittels dem Authentifizierung-System mitgeschickt. Andererseits werden Benutzer mittels des Autorisierung-Systems Zugriffsrechte zugewiesen [Con16a].

3.2 SQLAlchemy

Pyramid Anwendungen benötigen ein Datenbanksystem für die dauerhafte Speicherung seiner Daten. Pyramid unterstützt SQLAlchemy und sein ORM um die Anwendungslogik mit der Datenbank zu verbinden [sql16a]. SQLAlchemy ist eine Python Bibliothek, die eine einfache Integration von relationalen Datenbanken ermöglicht. Es verfügt über verschiedene Komponenten wie Engine, Metadata und Session [Fou16h]. Die Engine ist der Startpunkt jeder SQLAlchemy Anwendung [SQL16c], sie verwaltet die Verbindungen zu einer Datenbank. In einer Pyramid Anwendung wird das Datenmodell als Python Klassen definiert. Diese Klassen werden vom SQLAlchemy ORM auf Datenbanktabellen abgebildet. Im listing 3.3 wird ein Beispiel einer Modell-Klasse vorgestellt. Eine Modell-Klasse besteht aus dem Attribut `__tablename__`, das die Tabelle festlegt, auf der die Klasse Issue abgebildet wird. Die Klasse Issue hat andere Attributen vom Typ Column wie `uid`, `title`, `info` und `author_uid`. Diese werden als Spalten in der Tabelle Issue angelegt. Die Klasse enthält ein weiteres Attribut vom Typ Relationship, das eine Fremdschlüssel Beziehung zwischen der Klasse Issue und User definiert [Cen16]. Die `declarative_base()` Funktion, gibt eine neue Base Klasse zurück, die von der Modell-Klasse erweitert wird. Sobald die Modell-Klasse fertig definiert ist, wird eine neue Tabelle generiert [SQL16b].

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, Text, ForeignKey
from sqlalchemy.orm import relationship
```

```
Base = declarative_base()
```

```
class Issue(Base):
```

```
    __tablename__ = 'issues'
```

```
    uid = Column(Integer, primary_key=True)
```

```
title = Column(Text, nullable=False)
info = Column(Text, nullable=False)
author_uid = Column(Integer, ForeignKey('users.uid'))

users = relationship('User', foreign_keys=[author_uid])
```

Listing 3.3: Beispiel einer Modell-Klasse

Für die Kommunikation mit der Datenbank wird eine Session benötigt. Das Listing 3.4 illustriert, wie eine Session erstellt, mit einer Engine verbunden wird und wie die Metadata die Tabellen anlegt. Session verwaltet Objekte von Datenmodell-Klassen. Eine Session kann neue Objekte von einer Datenbank laden, Änderungen an Objekten sichern und neue Objekte als Datensätze in der Datenbank speichern. Metadata bindet Tabellen an eine spezifische Engine und legt die Tabellen in der Datenbank an.

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///')

from sqlalchemy.orm import sessionmaker

# Ein Session objekt erstellen
session = sessionmaker()

# Engine mit einer Session binden
session.configure(bind=engine)

# erstellt alle Tabellen in der Datenbank
# die im Datenmodell definiert sind
Base.metadata.create_all(engine)
```

Listing 3.4: Session mit Engine binden und Tabellen in der Datenbank anlegen

3.3 Templating mit Chameleon

Da diese Bachelorarbeit auf Templating mit Chameleon basiert, wird in diesem Abschnitt ein Überblick über Chameleon gegeben. Chameleon ist eine Python Template Engine. Sie wird eingesetzt um HTML oder XML Dokumente für Webseiten zu generieren. Sie benutzt eine Template Attribute Sprache wie TAL und METAL in Form von Element Attribute innerhalb des HTML Dokuments um den

Dokumentfluss zu kontrollieren, und die Wiederholung von Elementen, das Ersetzen und die Übersetzung von Texten durchzuführen [BRC11].

TAL ist eine Template Attribut Sprache. Die Anweisungen von TAL sind XML Attribute vom TAL Namespace `xmlns:tal="http://xml.zope.org/namespaces/tal"`, diese Attribute haben einen Namen und einen Wert. Im listing 3.5 erstes Beispiel wird das `content` Attribut gezeigt. Hier wird der Inhalt 'Hi' des Elements `<h1>` durch den Wert 'hallo' des `content` Attributs ersetzt. Das Attribut namens `repeat` ermöglicht das Iterieren über eine Liste von Daten. Das listing 3.5 zweites Beispiel demonstriert wie eine Liste von Themen durch das `repeat` Attribut iteriert wird. Mit dem Attribut `condition` wird eine Bedingung geprüft. Hier wird der Paragraph und sein Inhalt angezeigt wenn die Bedingung 'isAdmin' erfüllt ist [Com10].

1. content

```
<h1 tal:content="hallo">Hi</h1>
```

2. repeat

```
<tr tal:repeat="thema themen">
  <td tal:content="thema">text ersetzt durch thema</td>
</tr>
```

3. condition

```
<p tal:condition="isAdmin"> Willkommen im Admin Bereich </p>
```

Listing 3.5: Template Attribut Sprache

I18N ist eine Internationalisierung Sprache. Sie stellt ein Übersetzungssystem dar. Das Listing 3.6 zeigt die Verwendung des Attributs `translate` von I18N. Der Schlüssel `issue` wird benutzt um den Inhalt `Issue` zu übersetzen [BRC11].

```
<html i18n:domain="example">

  <div i18n:translate='issue'>Issue</div>

</html>
```

Listing 3.6: I18N: Attribute

3.4 Javascript

Javascript ist eine Programmiersprache für HTML und Web [w3s16g]. Sie wird eingesetzt um Benutzeraktionen auszuwerten und nach definierten Regeln zu reagieren, so kann beispielsweise ein Dialogfenster angezeigt werden wenn der Benutzer auf eine Taste drückt, ein Bereichswechsel kann gesteuert werden wenn auf einen bestimmten Link geklickt wird. Dies findet statt, ohne dass die Webseite erneut aufgebaut wird. Eine ihrer Stärken ist Asynchronous Javascript And XML (AJAX). Mit AJAX können HTTP Anfragen im Hintergrund an den Web Server geschickt und empfangen werden [w3s16a]. Javascript bindet dann die Antwort auf einem Bereich der Webseite ein.

Um Daten zwischen Webserver und Client mittels AJAX auszutauschen wird JavaScript Object Notation (JSON) verwendet. JSON ist ein leichtes Datenaustausch Textformat. Es ist leicht vom Menschen zu schreiben und zu lesen und von Maschinen zu parsen und zu generieren. Es basiert auf der Standard Javascript Programmierungssprache, Standard ECMA-262 3rd Edition-December 1999. Da JSON von Programmiersprachen unabhängig ist, und benutzt Konventionen die bei Programmierer von C, Java, Python usw. bekannt sind, ist es eine ideale Datenaustausch Sprache. Es ist entweder auf einer Zusammenstellung (Collection) von Paaren von Namen und Werten wie Dictionary und Hash-tabellen oder auf sortierten Listen wie Array, Listen und Vektoren aufgebaut [jso].

Um die Programmierung mit Javascript zu vereinfachen wurde das Javascript Framework JQuery eingeführt. Dank seiner Programmierschnittstelle sind in HTML-Dokumenten die Navigation, die Manipulation, die Ereignisverarbeitung und die Animation leichter geworden [Fou16i]. Mit JQuery wird auf die Document Object Model (DOM) Elemente mit dem Dollarzeichen zugegriffen um ein Element auszuwählen und darauf eine bestimmte Aktion anhand der einfachen Syntax '\$(Selektor).action()' durchzuführen. Wobei Ein Selektor eine Id, eine HTML Klasse oder ein HTML Element ist. Um beispielsweise ein Element Anzuzeigen wird die Aktion 'show()' verwendet. Zur DOM-Manipulation bietet JQuery verschiedene, einfache und nützliche JQuery Methoden wie 'text()', 'html()' und 'val()' zum Setzen oder Zurückgeben des Wertes eines selektierten Elements. Zur Ereignisverarbeitung wie die Auswahl von Radio Button und das Anklicken eines Elements bietet JQuery diverse Methoden, beispielsweise lassen sich mit der folgenden Syntax '\$("p").click()' alle <p> Elemente einer Webseite einem Click Ereignis zuordnen. Um eine Reaktion auf das ausgelöste Click Ereignis zu definieren muss eine Ereignis Händler (Event Handler) Funktion dem Ereignis übergeben. Dies wird im Listing 3.7 gezeigt [w3s16h]. wie im Listing zu sehen, wenn das <p> Element angeklickt wird, wird es verborgen.

```
$( " p " ). click ( function () {  
    $( this ). hide ( );  
});
```

Listing 3.7: JQuery: Click Ereignis

3.5 Bootstrap

Bootstrap ist das meist bekannteste HTML, CSS und JavaScript Front-End Framework für die schnelle und einfache Entwicklung von Webanwendungen, die mit einem Code auf allen Endgeräten funktioniert, von Smartphone über Tablets bis hin zu Desktop-PCs. Das Framework wurde von Twitter als Open Source veröffentlicht. Bootstrap gewährt dem Entwickler eine große Anzahl von UI Elementen beispielsweise Buttons, Dropdown-Menü, Modal-Dialog und Eingabegruppen für die Gestaltung von Web-Seiten [boo16]. In diesem Abschnitt wird auf einige Elemente eingegangen, die in dieser Arbeit verwendet wurden. Das Grid-System von Bootstrap ermöglicht es Layouts mit einfachem Aufwand zu erstellen. Bootstrap bietet ein Raster-Layout mit bis zu 12 Spalten. Wobei die Anzahl der Spalten optional ist. Es ist standardmäßig auf 12 gesetzt, die einzelnen Spalten können gruppiert werden um die gewünschte Anzahl von Spalten zu bekommen. Anhand der vier Klassen 'xs'(extra-klein), 'sm'(klein), 'md' (mittel) und 'lg' (large) des Grid-Systems von Bootstrap, können dynamische und flexible Layouts bereitgestellt werden. Um ein einfaches Layout zu erstellen wird zuerst ein div-Element mit der Klasse 'container' oder 'container-fluid' definiert. Danach können Zeilen mit der Klasse 'row' angelegt werden. Zum Schluss kann jede Zeile in mehrere Spalten, bis zu 12 Spalten, mit der Klasse 'col' aufgeteilt werden. Wird die Raster-Klasse 'lg' zu einer Spalte hinzugefügt (col-lg), wird ein breites Layout herangezogen [w3s16d]. Bootstrap stellt verschiedene Arten von Buttons zur Verfügung. Beispielsweise kann ein Warnungsbutton mittels der Klasse 'btn btn-warning' und ein Link Button anhand der Klasse 'btn btn-link' definiert werden. Um einen Button zu erstellen kann zuerst einfach eine div oder eine Span Klasse erstellt werden, danach wird eine der Button Klassen einem Element wie 'button', 'a' und 'input' zugewiesen [w3s16b]. Außerdem bietet Bootstrap das Erstellen von geordneten 'ol' und ungeordneten Listen 'ul', sie dienen Textinhalte darzustellen. Eine Standard Liste kann mit dem Element erzeugt werden, dem die Klasse 'list-group' zugeordnet wird. Darin werden die Listenelemente mit der Klasse 'list-group-item' eingefügt [w3s16e]. Darüber hinaus stellt Bootstrap ein weiteres wichtiges Element, nämlich das Dropdown-Menü zur Verfügung. Es ist eine umschaltbare Liste aus Einträgen, von denen sich der Benutzer für einen entscheiden kann. Ein Dropdown-Menü besteht aus zwei Elementen, einem Button der Klasse 'dropdown-toggle' und einer ungeordneten Liste der Klasse 'dropdown-menu', die eine Liste von Links enthält [w3s16c].

3.6 D-BAS

Im Internetzeitalter spielen Online-Partizipation beim Lernen, Wissen Austauschen und Beteiligen an der Entscheidungsfindung eine große Rolle. Als Lösungsansätze für Online-Partizipation kommen Foren, Pro- und Kontra-Listen und die Argumentationskarten zum Einsatz. Internet-Foren sind Diskussionsplattformen, die dazu dienen, Diskussionsbeiträge zu schreiben, die andere lesen und beant-

worten können. Sobald ein neues Thema angelegt wird, kann eine neue Diskussion gestartet werden. Im Hinblick auf die Anzahl an Themen haben sich Foren in der Praxis als skalierbar erwiesen, jedoch im Hinblick auf die Anzahl an Teilnehmern skalieren sie schlecht. Der Pro- und Kontra-Listen Ansatz hilft uns eine Entscheidung zu treffen indem man die Pro- und Kontra-Argumente vergleicht. Aber bei diesem Ansatz fehlen die Beziehungen zwischen den Pro- und Kontra-Argumenten. Der dritte Ansatz, der auf Argumentationskarten basiert, löst die Probleme der Anderen dadurch dass er die Argumente strukturierter repräsentiert. Dieser Ansatz weist den Nachteil auf, dass er Expertenwissen benötigt. Von daher ist ein neuartiger Prototyp entwickelt, der auf Argumentationskarten basiert, aber deren Nachteile vermeidet [KBBM16].

3.6.1 Argumentationskarten in D-BAS

Die Argumentationskarten stellen die Argumentationen bei der Diskussion eines Themas graphisch dar. Die wesentlichen Bestandteile einer Argumentationskarte sind Themen, Aussagen, Argumente, Prämissen, Autoren und Quellen für die Aussagen. Eine Aussage kann eine These, eine Frage oder ein Vorschlag sein. Im Rahmen der Diskussion wird die These mit wissenschaftlichen Argumenten bewiesen oder widerlegt. Handelt es sich um einen Vorschlag oder eine Frage werden Pro- und Contra-Argumente abgewogen um die Entscheidung für die Realisierung des Vorschlags zu treffen oder die Frage zu beantworten. Ein Argument oder eine Stellungnahme besteht aus einer Aussage und Prämisse oder einer Gruppe von Prämissen. Prämissen sind Aussagen, die die Validität des Arguments verstärken oder abschwächen.

3.6.2 Funktionalität von D-BAS

Der D-BAS beruht auf der Idee, den Diskussionsteilnehmer mit dem System interagieren zu lassen. Ein Teilnehmer kann sich zuerst im System einloggen. Dann kann er ein Thema auswählen. Danach werden Aussagen zum Thema angezeigt. Nach der Auswahl einer Aussage, kann der Teilnehmer seine Meinung dazu abgeben. Diese Meinung kann er begründen indem er den wichtigsten Grund dafür aussucht. Dann bekommt er die Argumente anderer Teilnehmer, die er annehmen oder ablehnen kann. Außerdem kann der Teilnehmer neue Argumente hinzufügen. Darüber hinaus gibt ihm das System die Anzahl der Benutzer, die sowohl die gleiche Meinung als auch das gleiche Argument teilen.

3.6.3 Datenmodells von D-BAS

D-BAS enthält zwei Datenmodells, Diskussion- und News-Datenmodell. Im Discussion-Datenmodell werden die Klassen wie Issue, Language, Group, User, Setting, Statement, StatementReferences , Textversion, Argument, Premisgroup, Premise, VoteArgument, VoteStatement und Notification definiert. Diese Klassen bilden die Kernelemente der Anwendung. Die Abbildung 3.1 zeigt sowohl die Klassen als auch deren Fremdschlüssel -Beziehungen zueinander. Diese sind mit einem Pfeil dargestellt. Die Richtung des Pfeils besagt dass die Klasse von der der Pfeil startet, besitzt einen Fremdschlüssel, der die Klasse referenziert wo der Pfeil endet. Die Klasse Language definiert die Sprache, in der der angezeigte Webseite-Inhalt der Anwendung geschrieben ist. Die Klasse Group legt fest, welche Arten von User es gibt. Jede Usergroup besitzt bestimmte Rechte. Zum Beispiel die Administratoren können auf dem Adminbereich zugreifen, Themen anzeigen und editieren. Autoren dürfen nicht auf Adminbereich zugreifen, sie können aber Themen anzeigen und editieren. Users Group besitzen wenige Rechte. Sie dürfen nur Themen anzeigen. Die Klasse User repräsentiert den Benutzer der Anwendung, der mit dem System interagiert. Die Klasse Issue definiert Themen. Themen bilden das Wissen der D-BAS Diskussionsplattform. Autoren können Themen hinzufügen und editieren. Die Klasse Statement definiert die Aussagen. Sie gehören zu einem Thema. Da die Aussagen in der Anwendung mit einer Quelle referenziert sind, ist eine StatementReferences Klasse definiert. Eine Aussage hat eine Textversion, die ebenfalls im Datenmodell definiert ist. Die Klasse Premise repräsentiert Prämissen. Prämissen sind Aussagen. Sie dienen zum Begründen von anderen Aussagen. Da ein Benutzer für die Begründung einer Aussage mehr als eine Prämisse auswählen kann, ist ein Premisegroup Klasse definiert. Die Klasse Argument ist eine Kombination von Prämissen und Aussagen. Sie stellt eine Stellungname eines Users mit Begründung dar. Die Klassen VoteArgument und VoteStatement geben den Benutzern die Möglichkeit Stellungnamen der anderen Benutzers anzuzeigen. Dies erleichtert dem Benutzer den Einstieg bei der Eingabe von Argumenten. In der Web Anwendung können die Benutzer Nachrichten senden und empfangen, von daher ist die Klasse Notification definiert.

3.6.4 Struktur von D-BAS

Das Projekt Verzeichnis dbas enthält mehrere Python Module, unter anderem die Module Admin, Graph, API, Docs und das Hauptmodul dbas, die setup.py Datei und die Konfigurationsdateien Development.ini und Production.ini. Die Abbildung 3.2 zeigt die Struktur vom dbas Modul. Dieser enthält die -ini-.py Datei, hier ist die Main Methoden implementiert, diese Methode dient zum Erzeugen Der WSGI Applikation, Abbildung von Routen auf Views, Laden von Daten aus der Datenbank, Setzen der Authentifizierung und Einbinden von den anderen Modulen. Das Modul Admin repräsentiert das

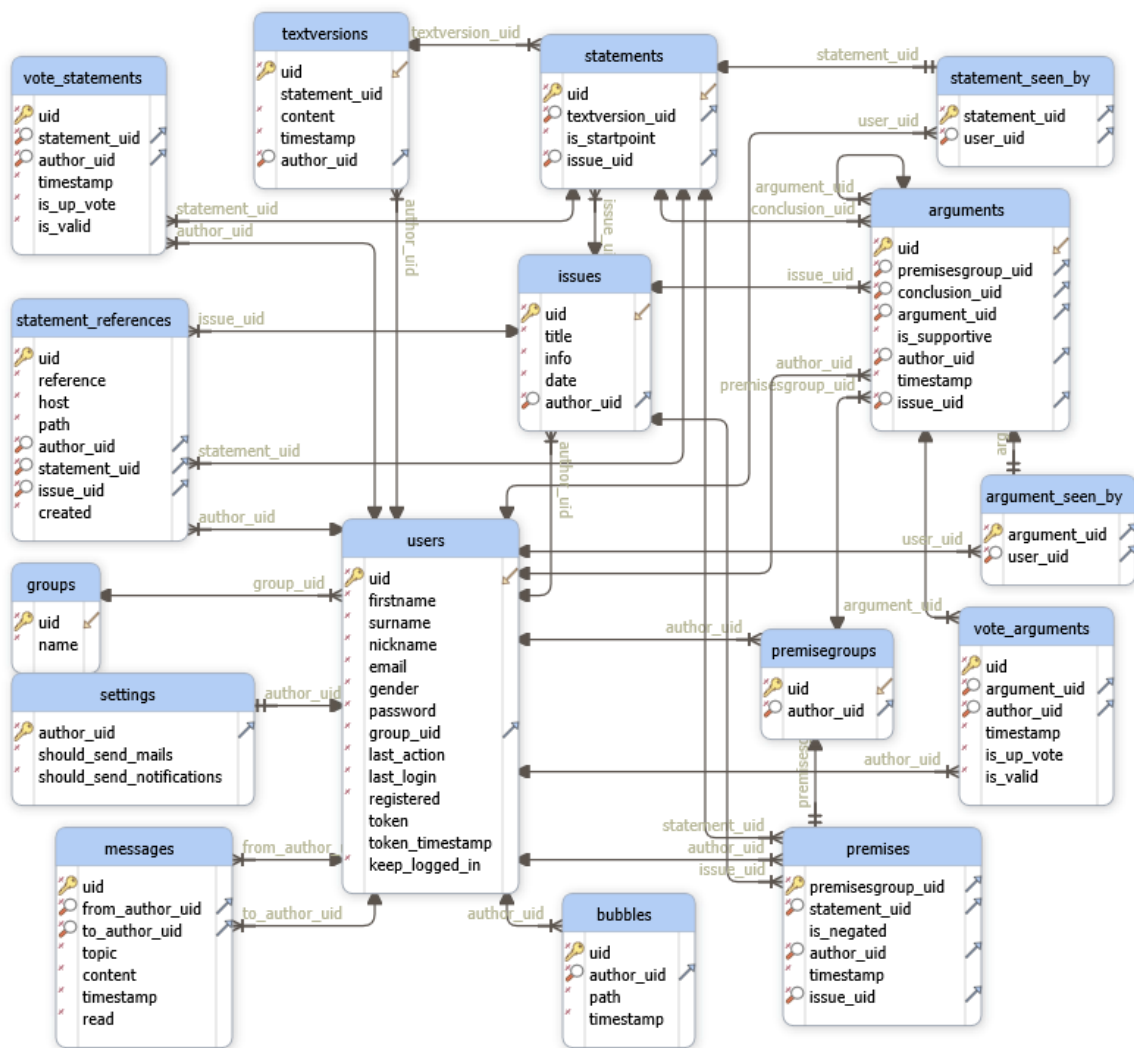


Abbildung 3.1: D-BAS: Datenmodell

Verwaltungsinterface von D-BAS. Das Graph Modul ermöglicht das Exportieren von verschiedenen Daten aus D-BAS. Das Modul API erlaubt die Kommunikation mit der Außenwelt und eine entfernte Diskussion von beliebigen Plätzen. Docs enthält Installationsinformationen.

In der views.py Datei sind alle Views und Ajax Interfaces mit dem View Config Dekorator deklarativ definiert. Wobei der View Config Dekorator benutzt ist, um das @view_config Attribut hinzuzufügen, der eine Methode repräsentiert, diese Methode bekommt eine Anfrage und gibt eine Antwort als Dictionary mit Hilfe des Renders zurück. Die in der Views implementierten Methoden rufen andere Methoden von Hilfsklassen auf. Hilfsklassen beinhalten Methoden, die zum Beispiel Daten aus der Datenbank abrufen oder Daten speichern. Die Kommunikation mit der Datenbank findet über SQLAlchemy statt. Das verwendete Datenbanksystem ist PostgreSQL. Das Modul dbas enthält andere

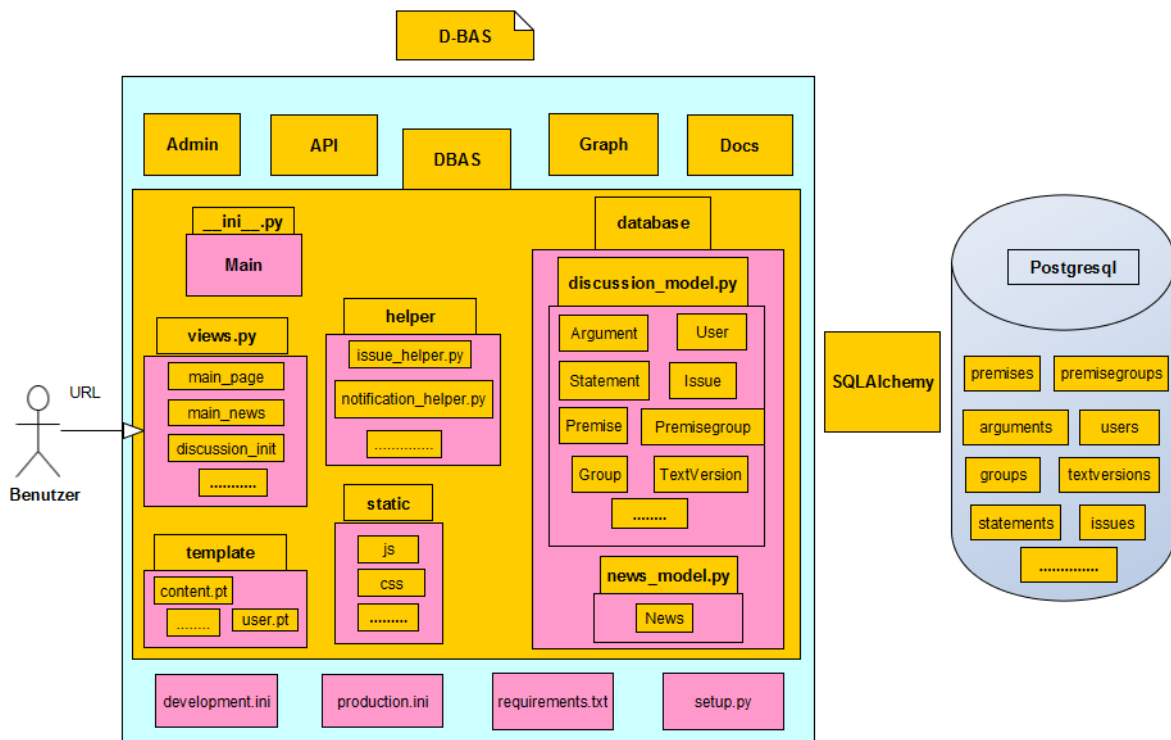


Abbildung 3.2: dbas Struktur

Verzeichnisse wie static und templates. Im Verzeichnis static sind alle statische Dateien wie Cascading Style Sheets (CSS)-Dateien, Javascript-Dateien und Bilder enthalten. CSS ist eine Sprache zur Gestaltung von HTML Dokumente [w3s16f]. Das Verzeichnis templates umfasst alle Templates, die mit dynamischen Informationen gefüllt werden.

Kapitel 4

Verwaltungsinterface

4.1 Einführung

Nachdem im letzten Kapitel eine Grundlage geschaffen wurde, beschäftigt sich dieses Kapitel mit dem Ziel dieser Arbeit, nämlich die Erstellung eines Interface zur Visualisierung und Verwaltung von Datenbanken des Projekts D-BAS. Die Verwaltungstätigkeiten umfassen das Anzeigen von Datenbankinhalten inklusive die Auflösung der Fremdschlüssel, das Hinzufügen und das Editieren von Daten. Das Interface soll den Administrator bei seiner Arbeit wesentlich unterstützen und von der Komplexität der Datenbank fernhalten. Das Verwaltungsinterface ist durch ein Authentifizierungssystem geschützt, nur der autorisierte Administrator hat das Zugriffsrecht auf diesen Bereich. Das Verwaltungsinterface wird in seinem eigenen Modul implementiert und dem D-BAS Prototyp hinzugefügt. Die zu verwaltende Datenbank ist im Modell der Hauptanwendung definiert. Für die Implementierung des Admin Interfaces wird der Entwicklungseditor Pycharm eingesetzt. In diesem Kapitel wird zuerst die Struktur des Adminmoduls vorgestellt. Hier wird sich mit den Funktionalitäten aus dem Backend auseinandergesetzt. Danach werden die Funktionalitäten des Frontends präsentiert.

4.2 Struktur des Adminmoduls

Die Abbildung 4.1 stellt die Struktur des Adminmoduls dar. In diesem Modul kann es sich zwischen Python Dateien, statische Dateien und Templates unterscheiden. Das Modul enthält wie jedes Pyramid Paket eine `__ini__.py` Datei wo die Main Methode definiert ist, `Views.py` und `lib.py` Dateien. Das `js` Verzeichnis enthält die Javascript Dateien. Das `template` Verzeichnis umfasst die HTML Vorlagen. Wenn der Administrator eine Anfrage sendet, landet diese zuerst in einer View und wird von ihr verarbeitet. Wenn dabei Daten in der Datenbank gespeichert oder von dort geholt werden sollen,

kommen die Hilfsfunktionen aus lib.py zum Einsatz. In diesem Abschnitt wird detailliert auf die Funktionen des Backend eingegangen.

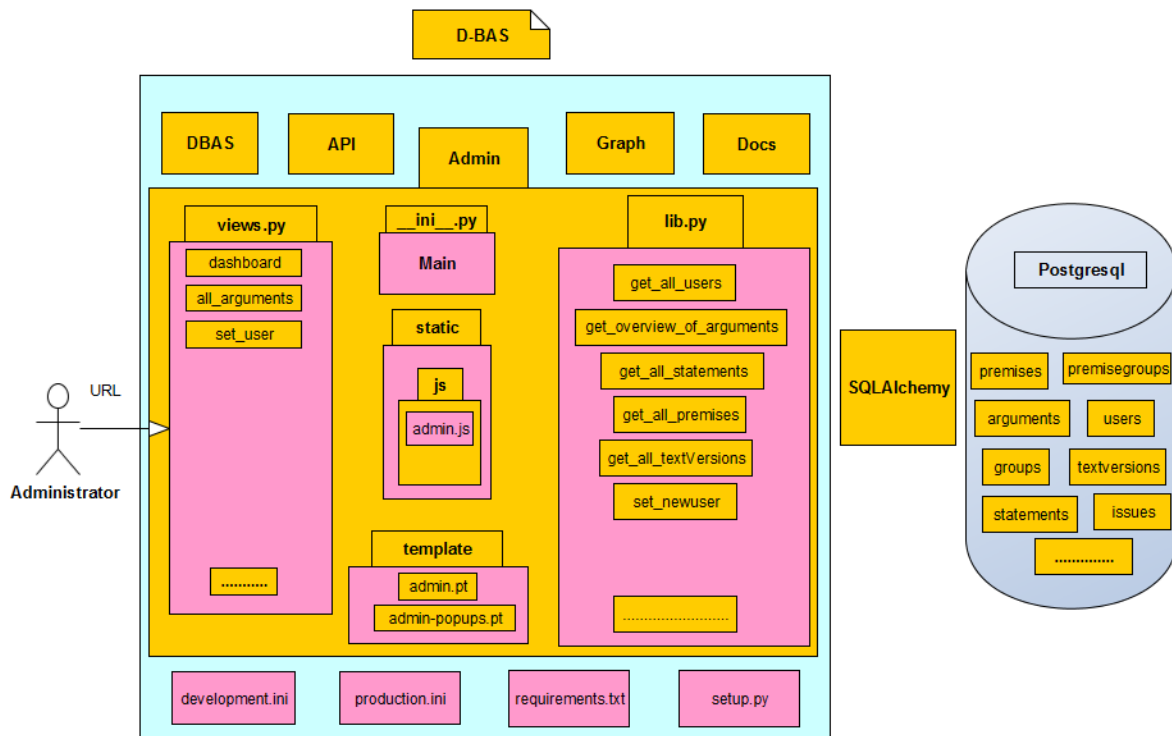


Abbildung 4.1: Admin-Struktur

4.2.1 Views

Die Views.py Datei beinhaltet die zentralen Funktionalitäten des Adminmoduls. Hier sind die Views in Form von Cornice Services implementiert. Die Cornice Bibliothek bietet die Klasse Service um einen Web Service in Pyramid zu definieren. Das Listing 4.1 zeigt einen Code Ausschnitt aus der Views Datei. Am Anfang ist der Haupt Service namens dashboard_page definiert. Dieser Service liefert Antworten für die Haupt-Admin Webseite und ist unter dem Pfad /main zu erreichen. Der Parameter Permission definiert, wer auf diesen Service zugreifen kann. Hier ist der Zugriff für alle Benutzer erlaubt. Der zurückgelieferte Inhalt der Web Seite ist durch Login geschützt. Die Bereitstellung dieses Inhalts ist in der Funktion main_admin des Hauptservices implementiert und nur für die autorisierten Administratoren erhältlich.

```
dashboard = Service(name='dashboard_page', path='/main',
    description="Admin Page", renderer='templates/admin.pt',
```

```
permission='everybody', # or permission='use'
cors_policy=cors_policy)

all_arguments = Service(name='getarguments',
    path='/argument_overview',
    description="Argument Overview",
    cors_policy=cors_policy)

set_user = Service(name='setuser', path='/set-new-user',
    description="set new user", cors_policy=cors_policy)
```

Listing 4.1: Admin Services definieren

Das Listing 4.2 zeigt die Implementierung dieser Funktion. Diese Funktion verarbeitet nur HTTP Get Anfragen. Sie delegiert die Berechnung der Tabelleninhalte an die Hilfsfunktionen aus der lib.py Datei, Z.B. die Funktion `get_all_users` gibt alle gespeicherten User in der Datenbank zurück. Die Funktion `get_dashboard_infos` stellt aggregierten Daten wie die Anzahl der Themen in der Datenbank zur Verfügung. Am Ende werden die berechneten Daten in Form ein Dictionary zur Verfügung gestellt. Das Dictionary wird dem Template `Admin.pt` mitgeteilt, dann wird daraus ein HTTP Antwort Objekt mittels Chameleon generiert und an den Aufrufer zurückgesendet.

```
@dashboard.get()
def main_admin(request):
    users = get_all_users(request.authenticated_userid,
        ui_locales, request.application_url)
    issues = get_all_issues(request.authenticated_userid,
        ui_locales, request.application_url)
    dashboard = get_dashboard_infos()

    return {
        'users': users,
        'issues': issues,
        'dashboard': dashboard,
    }
```

Listing 4.2: main_admin Funktion Implementierung

Wie im Listing 4.1 zu sehen ist, sind weitere Services wie `getarguments` und `setuser` definiert. Das Listing 4.3 zeigt deren Implementierung an. Die beiden Services unterscheiden sich vom Haupt Service darin, dass die Antwort als JSON Datenformat zwischen dem Client und dem Server eingesetzt ist

und unterscheiden sich untereinander darin, dass die Erste Get HTTP Anfragen aufnimmt, akzeptiert die Zweite Post HTTP Anfragen. Die Funktion `get_argument_overview` ruft eine andere Hilfsfunktion aus `lib` Datei auf. Diese Funktion ist für die Bereitstellung von allen Argumenten aus der Datenbank in Form vom Dictionary zuständig. Das Ergebnis wird mittels der Funktion `dumps` aus der JSON Bibliothek von Python Daten in das JSON Datenformat umgewandelt. Die zweite Funktion `set_new_user` speichert mit Hilfe der Funktion `set_newuser` aus `lib` einen neuen User in der Datenbank. Außerdem werden hier die gespeicherten Daten des neuen Users als JSON Datenformat zurückgeliefert.

```
@all_arguments.get()
def get_argument_overview(request):

    ui_locales = get_language(request, get_current_registry())
    return_dict = get_overview_of_arguments(request.authenticated_userid,
        ui_locales)

    return json.dumps(return_dict, True)

@set_user.post()
def set_new_user(request):

    ui_locales = get_language(request, get_current_registry())
    return_dict = set_newuser(request, ui_locales)

    return json.dumps(return_dict, True)
```

Listing 4.3: Services Funktionen mit json Datenformat

4.2.2 Hilfsfunktionen und Zugriff auf Datenbank

Die Hilfsfunktionen aus der `lib` Datei repräsentieren eine Schnittstelle zwischen Views und der Datenbank, indem sie Abfragen mittels SQLAlchemy durchführen um Daten aus der Datenbank zu lesen oder dort zu speichern. Anschließend werden die Abfrageergebnisse an Views zurückgeliefert. Am Anfang jeder Funktion wird sichergestellt, ob der Aufrufer Admin-Rechte hat, um mit weiteren Verarbeitungen fortzusetzen. Wenn nicht gibt die Funktion ein leeres Dictionary zurück. In diesem Abschnitt werden Datenbank Zugriffe anhand Code Ausschnitte gezeigt und erläutert. Hierbei können alle Datenbank Zugriffe in drei Kategorien unterteilt werden: Bereitstellung, Hinzufügen und Ändern von Daten. Das Listing 4.4 stellt einen Ausschnitt aus der Datei `lib` mit der Funktion `get_all_issues` als Beispiel Für die Bereitstellung von Tabellen Inhalten vor. Die erste Anweisung entspricht in der

Structured Query Language (SQL) Welt einer Select Abfrage auf die Tabelle Issue mit einem Join auf die Tabellen User und Language um die Daten aus den einzelnen Tabellen zusammenzuführen. Diese Verknüpfung ermöglicht den Zugriff auf den User und die Sprache, die mit dem jeweiligen Issue in Verbindung stehen. Hier kann schnell festgestellt werden, wie einfach SQLAlchemy Anweisungen gegenüber SQL sind. Während das Ergebnis im SQL eine Tabelle ist, liefert SQLAlchemy ein Python Objekt zurück. Dadurch ist der Zugriff auf dessen Attribute einfacher geworden. Wie Das Listing zeigt, liefert die Datenbank Abfrage ein Array von Themen (Issues) Objekten. Beim Durchlauf der for Schleife wird auf die Attribute jedes Themas zugegriffen und in einem Dictionary gespeichert. All diese Themen Dictionaries werden einem Array hinzugefügt und an den Aufrufer nämlich die Views zurückgegeben.

```
def get_all_issues(user , lang , mainpage):

db_issues = DBDiscussionSession.query(Issue).join(User).
    join(Language).order_by(Issue.uid.asc()).all()
for issue in db_issues:
    tmp_dict = dict()
    tmp_dict['uid'] = str(issue.uid)
    tmp_dict['title'] = issue.get_slug()
    return_array.append(tmp_dict)

return return_array
```

Listing 4.4: Themen aus der Datenbank Bereitstellen

Die zweite Kategorie vom Datenbank Zugriff, das Hinzufügen von Daten in Tabellen, wird zuerst am Beispiel von TextVersion erklärt. Hierbei wird die Funktion set_newtextversion erläutert, die eine neue TextVersion in der Datenbank speichert. Diese Funktion bekommt eine Anfrage mit drei Informationen, Publiknickname, Statement und Textversion. Es wird zuerst eine Prüfung durchgeführt um sicherzustellen, dass der Administrator die benötigten Informationen eingegeben hat. Falls nicht wird die Verarbeitung mit einer Fehlermeldung abgebrochen. Da die TextVersion Datenmodell Klasse die IDs vom Author und Statement zum Hinzufügen eines neuen Datensatzes benötigt, wird zuerst der eingegebene Publiknickname als Filter benutzt um den entsprechenden Author aus der Datenbank zu selektieren. Dieser Filter ist vergleichbar mit der Where Klausel aus der SQL Welt und dient dazu, die Ergebnismenge einer Abfrage zu beschränken. Beim Selektieren von Statement werden die Tabellen Statement und TextVersion über das gemeinsame Spaltenattribut verknüpft. Der Grund dieser Verknüpfung liegt darin, dass das eingegebene Statement eine Textversion ist und die Tabelle Statement einen Fremdschlüssel zu Textversion besitzt. Anschließend wird die Ergebnismenge der Select Abfrage mit dem eingegebenen Statement gefiltert um das entsprechende Statement Object zu erhal-

ten. Bevor ein Hinzufügen durch die Session add Funktion stattfindet, wird kontrolliert, ob es sich um ein Duplikat handelt, indem eine select Abfrage auf die Tabelle TextVersion mit der eingegebenen TextVersion durchgeführt wird. Im Falle eines Duplikats endet die Verarbeitung mit der Duplikat Fehler Meldung. Die add Funktion ähnelt dem Insert in der SQL Welt. Sie nimmt ein Python Objekt auf und gibt nichts zurück. Aus diesem Grund wird eine weitere Select Abfrage benötigt um den gerade gespeicherten Datensatz aus Der Datenbank zu lesen und der View ein Dictionary mit Informationen über der neu hinzugefügten TextVersion zurückzugeben. Die zweite Session Funktion 'flush' in diesem Listing sorgt dafür, dass die zwischen gespeicherten Daten im Cache in die Datenbank geschrieben werden. Erst wenn eine Commit Anweisung erfolgt, werden alle Änderungen der Transaktion bestätigt und in der Datenbank festgeschrieben.

```
def set_newtextversion(request , lang = ''):
    publicnickname = request.params['author']
    text = request.params['text']
    statement = request.params['statement']
    db_user = DBDiscussionSession.query(User)
        .filter_by(public_nickname=publicnickname).first()
    db_statement = DBDiscussionSession.query(Statement)
        .join(TextVersion , TextVersion
            .statement_uid == Statement.uid)
        .filter_by(content=statement).first()
    DBDiscussionSession.add(TextVersion(content=text ,
        author=db_user.uid , statement_uid=db_statement.uid))
    DBDiscussionSession.flush()
    transaction.commit()
    db_textversion = DBDiscussionSession.query(TextVersion)
        .filter_by(content=text).join(User).first()
```

Listing 4.5: TextVersion zur Datenbank Hinzufügen

Als nächster Schritt wird das Hinzufügen von Statement betrachtet. Dies geschieht in der Funktion set_newstatement. Der Administrator kann entweder einen Text aus einer Liste von TextVersionen auswählen, oder eine neue TextVersion eingeben. Beim ersten Fall ähnelt das Hinzufügen eines Statements dem Hinzufügen von TextVersion. Der zweite Fall unterscheidet sich vom Ersten darin, dass zuerst ein Autor der neuen TextVersion eingegeben sein muss um sie in der Datenbank hinzufügen zu können, ansonsten wird eine Fehler Meldung ausgegeben, dass der Autor fehlt. Es wird auch hier eine Prüfung geben, ob das Hinzufügen von TextVersion erfolgreich gewesen wäre. Wenn das der Fall ist, wird die neue TextVersion von der Datenbank geholt und für das Hinzufügen des neuen Statements verwendet. Aber vorher wird die Tabelle Statement auf Vorhandensein von Duplikat gecheckt. Anschließend wird die neu hinzugefügte TextVersion mit dem gerade hinzugefügten Statement aktua-

lisiert. Abschließend wird das neue Statement als dictionary an die View zurückgegeben.

Beim Hinzufügen von Prämisse kommt die Funktion `set_newpremise` zur Anwendung. Um eine Prämisse in der Datenbank zu speichern benötigt diese Funktion die folgenden Informationen, Author, IsNegated, Issue, Statement und IsStartpoint und optional Premisegroup. Es gibt hier vier Fälle zu unterscheiden. Beim ersten Fall kann eine Prämisse mit einer vorhandenen Aussage und Premisegroup hinzugefügt werden. Im zweiten Fall kann eine Prämisse mit einer vorhandenen Aussage und neuer Premisegroup hinzugefügt werden. Beim dritten Fall kann eine Prämisse mit einer neuen Aussage und neuer Premisegroup addiert werden. Im vierten Fall kann eine Prämisse mit einer neuen Aussage und einer vorhandenen Premisegroup addiert werden. Welcher Fall wird durchgeführt, hängt von den eingegebenen Informationen ab. Wenn keine Premisegroup eingegeben ist, wird zuerst eine Neue hinzugefügt und mit dieser die Prämisse angelegt. Wenn das eingegebene Statement neu ist, wird zuerst eine Textversion addiert und mit dieser ein Statement in der Datenbank angelegt. Bevor ein Hinzufügen der Prämisse stattfindet, werden anhand der Eingaben die benötigten Informationen aus der Datenbank selektiert. Das betrifft Author, Issue und Statement. Als Kriterium für ein Duplikat werden die beiden Primärschlüssel der Tabelle Prämisse in Betracht gezogen und damit eine Select Abfrage durchgeführt. Erst bei nicht Vorhandensein von einem Duplikat, kann eine neue Prämisse in der Datenbank hinzugefügt werden. Ansonsten wird eine Duplikat Fehlermeldung zurückgegeben.

Jetzt wird die Funktion `set_newargument` als letzte Funktion für das Hinzufügen von Daten betrachtet. Sie erlaubt ein Argument in der Datenbank zu speichern. Um dies zu realisieren, werden die folgenden Objekte Premisesgroup, IsSupportive, Author und Issue benötigt. Obwohl die Objekte Argument und Statement optional sind, muss einer von beiden eingegeben sein. Wenn ein Argument eingegeben ist, bezieht sich das neue hinzuzufügende Argument auf dessen Aussage und ist in diesem Fall die Eingabe der Aussage nicht erforderlich. Wenn nicht, muss eine Aussage eingegeben werden.

Die dritte und letzte Datenbank Kategorie ist das Aktualisieren von Daten. Sobald ein Änderungswunsch auf der Admin Webseite betätigt wurde, landet dieser als Anfrage bei der entsprechenden Edit View. Wie die bisher betrachteten Fälle leiten die Views diese Anfragen an die jeweilige Edit Hilfsfunktion weiter. Eine Änderungsanfrage könnte Issue, User, Textversion, Statement, Prämisse und Argument betreffen.

Das Editieren wird erstens am Beispiel Issue näher betrachtet. Da eine Edit-Anfrage sowohl unveränderte als auch geänderte Daten umfasst, werden alle angelieferten Daten zunächst gecheckt, ob sie gefüllt sind. Title und Info müssen eingegeben sein. Außerdem muss Info genauso wie beim Addieren mindestens 10 Zeichen enthalten. Author muss auch ausgewählt sein. Da in dem Sprache Radio

Button immer eine Sprache als ausgewählt markiert, wird auf die Prüfung verzichtet ob die Sprache eingegeben ist. Danach werden die benötigten Daten für die Aktualisierung des Issues aus der Datenbank geholt nämlich User und Sprache. Anschließend wird die Aktualisierung über die ID des Issues durchgeführt da die ID konstant bleibt. Siehe Listing 4.6. Abschließend wird ein Dictionary mit den editierten Daten bereitgestellt und an die View zurückgegeben.

Als nächstes wird Das Ändern von User Daten näher beleuchtet. Die Änderung findet in der Funktion `edit_theuser` statt. Dabei werden bestimmte Prüfungen vor der Aktualisierung durchgeführt. Um Duplikate bezogen auf den Nickname zu erkennen, werden alle User mit dem gleichen Nickname und mit anderen Uids als die vom zu aktualisierenden User selektiert. Es wird gleich vorgegangen, um User mit gleichem Publicnickname oder mit gleicher Email zu erkennen. Die Prüfung der Gültigkeit der Email und der Gruppe wird als Letzter durchgeführt. Besteht ein Duplikat, bricht die Aktualisierung mit einer Fehler Meldung ab. Andernfalls wird die Aktualisierung des selektierten Users mit der dazugehörigen Uid fortgesetzt. Dies geschieht anhand der Update Funktion mit einer Liste von den zu editierenden Attributen und den jeweiligen neuen Werten. Ebenfalls endet auch hier eine erfolgreiche Aktualisierung mit Zurückgabe eines Dictionary an die View.

Abschließend folgt die Erläuterung der Aktualisierung vom Statement. Für diese Aktualisierung ist die Funktion `edit_thestatement` verantwortlich. Dabei wird zuerst das entsprechende Statement aktualisiert. Danach muss das dazugehörige TextVersion Objekt aktualisiert werden, da sich das Statement auf eine TextVersion bezieht. Die betroffene TextVersion muss zuerst selektiert und dann mit dem Wert des Statement-Texts aktualisiert werden.

```
DBDiscussionSession.query(Issue).filter(
    Issue.uid == uid).update(
    {'title': title, 'info': info, 'author_uid': db_user.uid,
     'lang_uid': db_lang.uid})
```

Listing 4.6: Issue Aktualisierung

4.3 Funktionalitäten von Adminmodul aus Frontend Sicht

Nach dem Start der Anwendung kann die Admin Webseite unter Eingabe der folgenden URL: <http://localhost:4284/admin/main> erreicht werden. Der Administrator muss sich anmelden um seine Verwaltungstätigkeiten durchführen zu können. Dies erfolgt über den Button Login wie es in der Abbildung 4.2 zu sehen ist. Nach einer erfolgreichen Anmeldung mittels Nickname und Password, bekommt der Administrator die Admin Hauptseite angezeigt. Außerdem bietet die Webseite die Möglichkeit eine Sprache auszuwählen an, in der die Button-Namen, Links und Titels erfasst sind. Wurde

die Anmeldung erfolglos durchgeführt, endet der Prozess mit Anzeige eines 401 Fehlers. In diesem Abschnitt wird sowohl der Aufbau der Admin Webseite als auch deren technische Implementierung vorgestellt. Danach wird die Interaktion des Administrators mit der Webseite aus Frontendsicht besprochen.

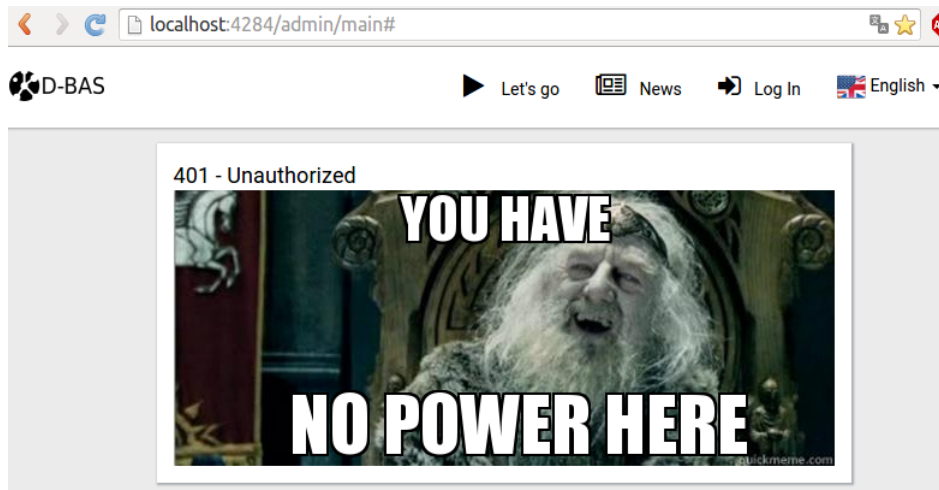


Abbildung 4.2: D-BAS: Admin Webseite

4.3.1 Aufbau der Admin Webseite

Die Abbildung 4.3 stellt die möglichen Abläufe der Administrator Tätigkeiten dar. In der Seitenleiste als auch im Dashboard kann der Administrator zwischen den Bereichen User, Themen, Argumente, Aussagen, Textversionen, Prämissen und Notifikationen auswählen. Im Dashboard sind aggregierte Informationen wie die Anzahl der gespeicherten Elemente pro Tabelle in der Datenbank angezeigt. Dies ist in der Abbildung 4.4 gezeigt. Für die technische Implementierung der Admin Webseite steht das Template `admin.pt` im Mittelpunkt. Diese HTML Vorlage definiert den Aufbau der Webseite. Es besteht aus statischen und dynamischen Daten. Die Ersten sind Header, Footer und die Seitenleiste, die als Navigationsmenü fungiert. Die Letzten sind Tabellen mit Platzhalter. Deren Inhalte werden von Views auf Admin Anfragen zurückgegeben, durch die Template Engine Chameleon ausgewertet und in der Tabellen ersetzt. Das Listing 4.7 stellt einen Code Ausschnitt aus der Datei `admin.pt` für die Implementierung der Seitenleiste dar. Der erste `div` Container sorgt dafür, dass die Seitenleiste anhand der Bootstrap `col-md-4` Klasse in einem eigenen Bereich mit vier Gridspalten eingeschlossen wird. Dieser Bereich ist mit der TAL Anweisung `Condition` geschützt so dass nur der Admin darauf zugreifen kann. Mit der Klasse `navbar navbar-default navbar-left` wird zuerst eine Navigationsleiste an der linken Seite im Bereich platziert. Danach sind Listen mit einem Link auf den entsprechenden Bereich hinzugefügt. Die Klasse `activate` bewirkt, dass das erste Link Element Dashboard am Anfang hervor-

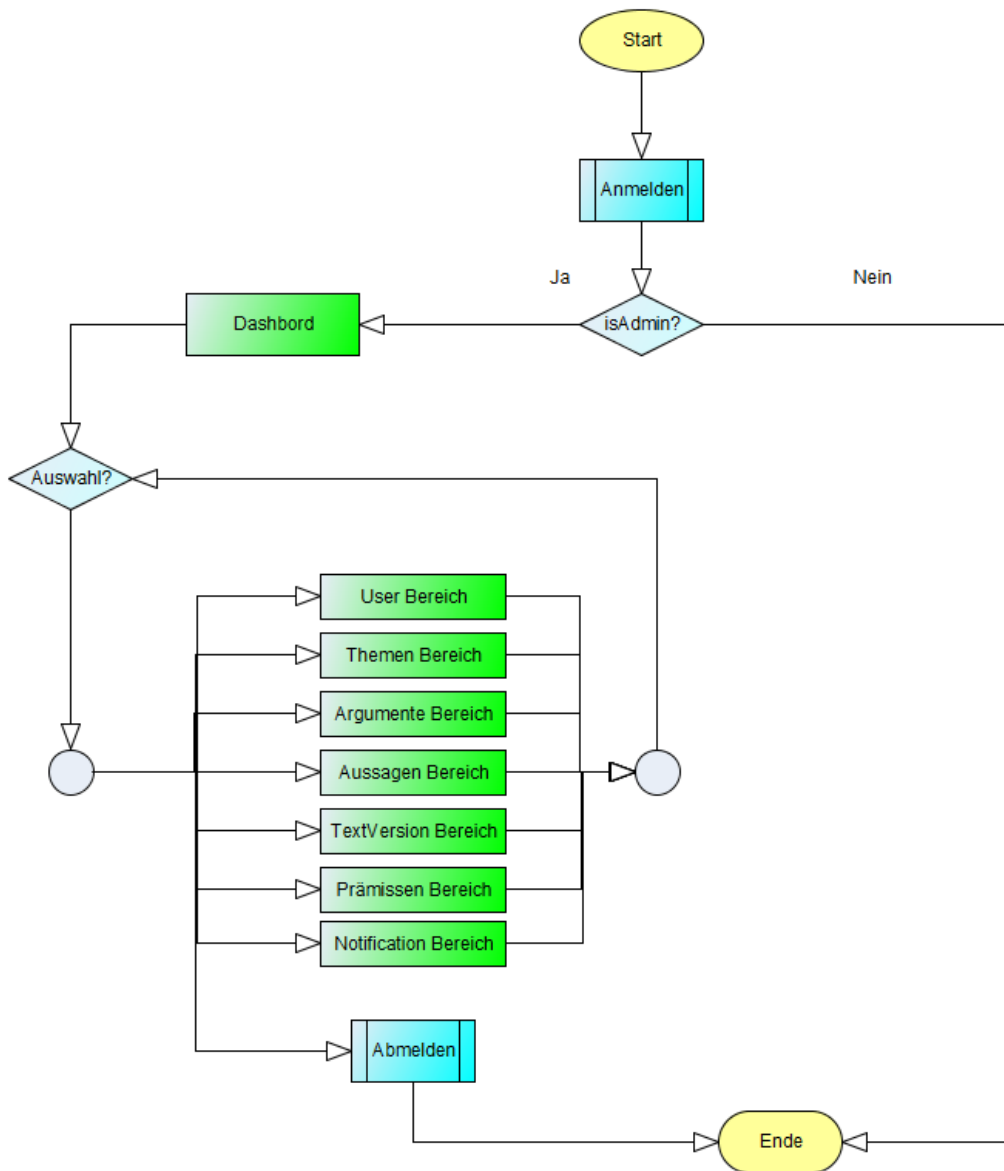


Abbildung 4.3: Admin-Ablaufdiagramm

gehoben ist. Die Seitenleiste enthält weitere Link Elemente wie Users, Arguments und Issues. Diese werden aktiviert sobald darauf geklickt wird. Der Name des Link Elements wird in der ausgewählten Sprache mittels der TAL Anweisung `i18n` übersetzt. Dank der Bootstrap Klasse `glyphicone` ist jedem Link der Seitenleiste mit einem passenden Icon eingebunden. In dem folgenden Listing sind zwei `glyphicons` definiert, `glyphicon-dashboard` und `glyphicon-user`, diese werden neben den Link `dashboard` und `user` angezeigt (Siehe Abbildung 4.4). Dies verleiht der Webseite ein schönes Layout.

```
<div class="col-md-4" id="navbar-wrapper"
```

```

    tal:condition="extras.is_admin">
<nav id="admin-navbar" class="navbar navbar-default navbar-left"
    role="navigation">
<ul class="nav navbar navbar-left">
<li class="active">
    <a href="#" id="admin-dashboards">
        <span class="glyphicon glyphicon-dashboard" aria-hidden="true">
        </span>&thinsp;&thinsp;
        <span i18n:translate="dashboard">Dashboard </span>
    </a>
</li>
<li>
    <a href="#" id="admin-users">
        <span class="glyphicon glyphicon-user" aria-hidden="true">
        </span>&thinsp;&thinsp;
        <span>Users </span>
    </a>
</li>
    .....

```

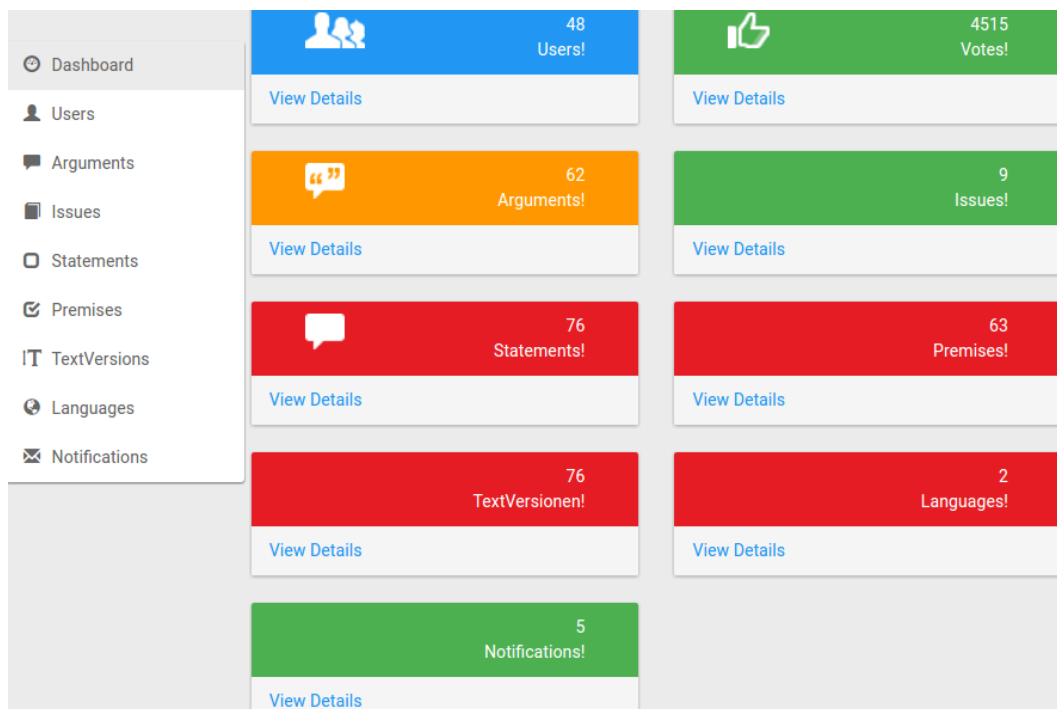


Abbildung 4.4: Admin-Dashboard

```
</div>
```

Listing 4.7: Seitenleiste HTML Teil

Als nächstes wird die Implementierung des Dashboard Bereichs anhand des Listings 4.8 gezeigt. Der Dashboard Container enthält mehrere Bootstrap Panel wie das User Panel. Im Panel-Header sind Informationen wie die Anzahl der Objekte einer Tabelle dargestellt. Diese Informationen sind in Form eines Platzhalters definiert, und durch den berechneten Wert ersetzt, Z.B. der `user_count` des dashboard dictionary, das von Views zurückgegeben wird. Mit dem Tag `img` wird zusätzlich ein Icon hinzugefügt um einen Bereich zu kennzeichnen. Das Panel Footer beinhaltet den Link für den Zugriff auf den dazugehörigen Bereich.

```
<div class="row col-md-12" id="admins-space-dashboard">
  <div class="panel panel-warning">
    <div class="panel-heading">
      <div class="col-xs-9 text-right">
        <div class="huge">${dashboard.user_count}</div>
        <div i18n:translate="users!">Users!</div>
      </div>
    </div>
    <a href="#">
      <div class="panel-footer" id="dashboard-user-count-detail">
        <span i18n:translate="view_details">View Details</span>
      </div>
    </a>
  </div>
</div>
```

Listing 4.8: Dashboard HTML Teil

Das Template `admin.pt` umfasst weitere unsichtbare Bereiche wie den Issue Bereich (Siehe Listing 4.9). Jeder Bereich besteht aus einem Link zum Hinzufügen eines Objekts in der Datenbank und einer Tabelle. Diese ist durch den Tag `table` definiert. Sie umfasst ein Header zum Definieren von Spaltentitel wie Title, Info und Author eines Themas. Außerdem enthält die Tabelle ein Body zum Anzeigen von Bereich Inhalten. Diese Inhalte werden als Dictionaries von den Views zurückgeliefert und durch die TAL Anweisung `repeat` durchlaufen. Hierbei ist das `issues` Dictionary eine Liste von Issue. Bei jeder Iteration wird eine Zeile gefüllt.

```
<table style="border-collapse: separate; border-spacing: 0;"
  class="table table-condensed tablesorter" border="0">
  <thead>
```

```
<tr >
  <th>#</th>
  <th i18n:translate="title">Title </th>
  <th i18n:translate="info">Info </th>
</tr >
</thead>
<tbody id="issue-table">
  <tr tal:repeat="issue issues">
    <td>${issue.title}</td>
    <td>${issue.info}</td>
  </tr >
</tbody >
</table >
```

Listing 4.9: Issue Bereich

4.3.2 Interaktion des Admins mit der Webseite

Für die Interaktion mit der Admin Webseite werden Links, Buttons, Dropdownmenüs und Popup Fenster bereitgestellt. Sobald der Admin ein solches HTML Element betätigt, werden Ereignisse ausgelöst und mittels Javascript abgefangen und verarbeitet. Das Javascript Handling Spektrum deckt die Navigation zu den Bereichen, das Ein- und Ausblenden und das Verändern von deren Inhalten, das Ein- und Ausblenden von Popups Fenster und das Absenden von HTTP Anfragen mittels AJAX ab. Diese Funktionen sind in den Javascript Dateien admin.js, admin-guihandler.js, admin-ajaxhandler und admin-interactionhandler implementiert. In diesem Abschnitt wird zuerst auf das Anzeigen von Bereich-Inhalten eingegangen. Danach wird das Hinzufügen von Daten in der Datenbank betrachtet. Am Ende wird das Editieren von Daten erläutert. Dies wird im Hinblick auf HTML und Javascript behandelt.

Bereich-Inhalt Anzeigen

Nach dem ersten Aufruf der Admin Webseite, bekommt der Admin das Dashbord angezeigt, da es in der Seitenleiste am Anfang als aktiv gesetzt ist. Wie es vorher gezeigt wurde, sind sowohl der anzuzeigende Bereich als auch das zum Navigationslink in der admin.pt Datei definiert, wobei jeder Linksbereich einer ID zugeordnet ist. Das darauf einzutretende Ereignis und deren Reaktion ist in der

Datei admin.js implementiert. Ein Beispiel dazu ist im Listing 4.10 gezeigt. Nach dem Klick auf dem Issue Link wird dessen ID ausgewertet und die Funktion showIssues aufgerufen.

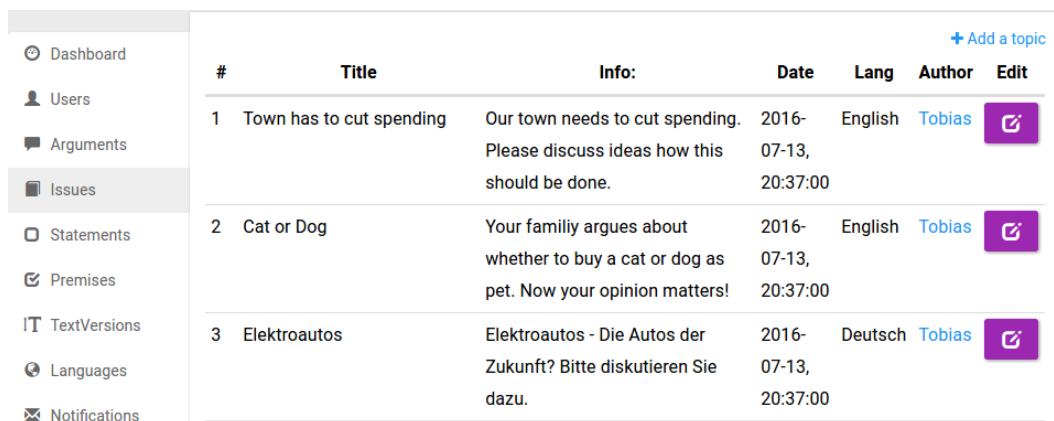
```
this.setUpLinks = function(){
    $('#admin-dashboards').click(function(){_this.showDashboard();});
    $('#admin-issues').click(function(){_this.showIssues();});
    $('#admin-statements').click(function(){_this.showStatements();});
}
```

Listing 4.10: Click Ereigniss auf ein Link in der Seitenleiste

Das folgende Listing 4.11 zeigt, was die Funktion showIssues bewirkt. Erstens wird der gerade angezeigte Bereich ausgeblendet, dies geschieht in der Funktion hideEverything. Zweitens wird der Link hervorgehoben. Zum Schluss wird der Issue Bereich mittels der show Funktion angezeigt(Siehe Abbildung 4.5).

```
this.showIssues = function(){
    this.hideEverything();
    $('#admin-issues').parent().addClass('active');
    $('#admins-space-issue').show();
};
```

Listing 4.11: Issue Bereich anzeigen



#	Title	Info:	Date	Lang	Author	Edit
1	Town has to cut spending	Our town needs to cut spending. Please discuss ideas how this should be done.	2016-07-13, 20:37:00	English	Tobias	
2	Cat or Dog	Your family argues about whether to buy a cat or dog as pet. Now your opinion matters!	2016-07-13, 20:37:00	English	Tobias	
3	Elektroautos	Elektroautos - Die Autos der Zukunft? Bitte diskutieren Sie dazu.	2016-07-13, 20:37:00	Deutsch	Tobias	

Abbildung 4.5: Issue Bereich

Der Inhalt der Bereiche Issue, Statement, TextVersion, User, Prämisse, Language und Notification wird schon beim ersten Aufruf der Admin Webseite bereitgestellt. Daher ist der Ablauf des Anzeigens dieser Bereiche gleich. Die einzige Ausnahme stellt der Bereich Argument dar. Erst wenn auf den Bereich Argument geklickt wird, wird eine Anfrage per AJAX an den Server geschickt. Das Senden der Anfrage und die Verarbeitung der Antwort geschieht in der Funktion getArgumentOverview in

der admin.js Datei. Das Senden der Anfragen per AJAX wird detailliert in der nächsten Abschnitten erläutert. Nachdem die Funktion eine Antwort in Form von Dictionary von Issue Arrays gekriegt hat, wobei jedes Issue Array ein Issue und eine Liste der dazugehörigen Argumente enthält, wird pro Issue eine HTML Tabelle von dessen Argumenten erzeugt. Da der Argument Bereich ein Dropdown-Menü zum Auswählen von Issue enthält, werden Argumenten nach Issue dargestellt.

Daten in der Datenbank Hinzufügen

Um Daten in der Datenbank hinzuzufügen ist jeder Bereich mit dem Add Link versehen. Das Listing 4.12 zeigt das Abfangen des Ereignisses und die Reaktion. Sobald auf dem Add Link geklickt wird, bekommt der Administrator ein Popup Fenster angezeigt um die benötigten Informationen über das hinzufügende Objekt einzugeben. Das Anzeigen und Verarbeiten des Popup Fensters findet in der Datei admin-guihandler.js statt. Zuerst wird die Struktur dieses Popup Fensters vorgestellt.

```
$( '# ' + addStatementButtonId ). click ( function () {  
    guiHandler . showAddStatementPopup ( new AdminInteractionHandler ()  
        . callbackIfDoneForSendNewStatementTable );  
});
```

Listing 4.12: Statement hinzufügen

Alle Popup Fenster sind in der admin_popups.pt Datei definiert, jedem Popup Fenster ist ein ID zugeordnet. Es ist nichts anderes als das modal-dialog Bootstrap Element. Es besteht aus Kopf(Header), Rumpf(Body) und Fuß(Footer). Der Kopf enthält zusätzlich zu dem Button den Titel für das Popup Fenster. Der Button dient zum Schließen des Fensters. Der Rumpf ist der Hauptteil des Popup Elements. Er kann alle Dialog Bedienelemente wie Dropdown-Menüs, Eingabefelder und Radio Buttons umfassen. Der Fuß kann Bestätigungs- und Abbrechenbutton enthalten. Das Add Statement Popup Fenster besteht aus drei Dropdown Buttons und einem Radio Button (Siehe Abbildung 4.6).

Jeder Dropdown Button enthält ein Dropdown-Menü, dem mehrere Einträge zugeordnet sind. Das erste Dropdown-Menü ist mit einem Eingabefeld versehen. Es ermöglicht den Administrator eine Textversion auszuwählen oder eine neue einzugeben. Das zweite ist für die Autor Auswahl der neuen TextVersion. Das letzte sorgt dafür, das Thema der hinzufügenden Aussage auszuwählen. Der Radio Button ermöglicht eine Auswahl für das Attribut isPosition zu treffen. Alle diese Eingaben sind obligatorisch für das Hinzufügen des Statements. Falls eine Eingabe vergessen wurde, wird eine Warnmeldung angezeigt. Dies lässt sich durch die Bootstrap Klasse alert alert-danger realisieren. Siehe das Listing 4.13.

```
<div class="alert alert-danger" id="popup-add-statement-error"
```


Abbildung 4.6: Add Statement Popup Fenster

```

style="margin-top: 1em; display: none;">
  <strong id="popup-add-statement-error-text"></strong >
</div >

```

Listing 4.13: Alarmmeldung bei fehlenden Eingaben

Das erste Dropdown-Menü für die TextVersion Eingabe lässt sich durch die Bootstrap Klasse dropdown-menu bewerkstelligen. Es besteht aus einem Label, das mit der Anweisung extras.tag in der jeweiligen gerade eingestellten Sprache bereitgestellt wird, und einem Input-group Element, das wiederum ein Input-group-btn Element enthält. Dieses Element umschließt ein Dropdown-toggle Button, das Dropdown-Menü selbst und ein Feld für die Eingabe der neuen TextVersion. Beim Dropdown-toggle Button handelt es sich um einen Umschalter für ein ein- und ausklappbares Dropdown-Menü. Die Verwendung vom Input-group Element ermöglicht es, die darin enthaltenen Elemente horizontal auszurichten. Um das Dropdown-Menü mit Textversion Einträgen dynamisch zu befüllen, wird die TAL Anweisung repeat eingesetzt. Alle diese Elemente sind innerhalb dem div Container mit der Form-group Klasse gruppiert (Siehe Listing 4.14).

```

<div class="form-group" id="text-switch">
  <label for="inputtext" control-label">
    ${structure:extras.tag.please_select_textVersion_for_statement}
  </label >
  <div class="input-group">
    <div class="input-group-btn">

```

```

<button class="btn btn-primary dropdown-toggle btn-sm"
  type="button" data-toggle="dropdown">Switch Text
</button>
<ul class="dropdown-menu" id="popup-add-statement-text-input">
  <li tal:repeat="textVersion textVersions"></li>
</ul>
</div>
<input class="form-control" type="text"
  id="add-statement-for-textversion-input">
</div>
</div>

```

Listing 4.14: Dropdown-Menü für die TextVersion Eingabe

Wie vorher im Listing 4.12 erwähnt wurde, übernimmt die Funktion `showAddStatementPopup` aus der Datei `admin-guihandler.js` das Anzeigen und die Verarbeitung des Add Statement Popup Fensters. Diese Funktion bekommt eine Callback Funktion als Parameter eingereicht. Als erster Schritt wird das Modal-dialog anhand der Modal `show` Funktion initialisiert und angezeigt. In der Funktion `showAddStatementPopup` sind fünf Klick Ereignisse und deren Reaktionen implementiert. Alle Klicks sind mit IDs aus dem Popup Fenster verbunden. Der erste Klick verursacht das Schließen vom Popup Fenster sobald auf den Abbrechen Button geklickt wird. Drei von diesen Klicks beziehen sich auf Dropdown-Menüs für die Auswahl vom Autor, Text und Thema. Sobald hier eine Auswahl getroffen wurde, wird der ausgewählte Eintrag als Inhalt des Dropdown-toggle Buttons angezeigt und als selektiert mit der ID des entsprechenden Dropdown-Menüs vermerkt. Der letzte und zentrale Klick ist das Klicken des Sichern Buttons (Listing 4.15). Zuerst werden alle eingegebenen oder selektierten Werte geholt und mit der Methode `trim` Leerzeichen und Zeilenumbrüche am Anfang und am Ende des Eingabestrings entfernt. Abschließend werden diese Eingaben mit der Callback Funktion als Parameter an die `sendNewStatement` Funktion aus der Javascript Datei `admin-ajaxhandler.js` weitergeleitet. Alle Dropdown-Menü Buttons und das Eingabefeld müssten nach dem Sichern der Anfrage oder nach dem Schließen des Popup Fensters zurückgesetzt werden, damit der Administrator beim nächsten Hinzufügen ein Popup Fenster ohne vorher erfassten Daten angezeigt bekommt. Dies wird dadurch erreicht, dass der Dropdown-Menü Button mittels der Funktion `html` mit dem Wert `Switch Issue` zurückgesetzt wird. Die anderen Dropdown-Menü Buttons werden genauso gleich behandelt. Um das Eingabefeld zurückzusetzen wird die Hidden Modal Händler Funktion `On` eingesetzt. Sie bewirkt, dass sobald das Popup Fenster in die Hintergrund verschoben ist, wird anhand der JQuery Funktion `trigger` einen Reset Händler zum Zurücksetzen des Eingabefelds gestartet.

```

$('#popup-add-statement-accept-btn').click(function () {
  var text = $.trim($('#popup-add-statement-text-input
    li a.selected').text()),

```

```
isStartpoint = $('input[name="statement_isStartpoint"]:checked')
    .val();
new AdminAjaxDiscussionHandler().sendNewStatement(text,
    newText, isStartpoint, issue, author, callbackFunctionOnDone);
});
```

Listing 4.15: Das Klicken des Sichern Buttons

In der `admin-ajaxhandler` Datei werden die spezielle Funktionen für den Datenaustausch mittels AJAX und JSON zwischen Client (Administrator Browser) und der Web Anwendung implementiert. Eine dieser Funktionen ist `sendNewStatement`, die hier genauer betrachtet wird. Als erster Schritt wird eine AJAX Anfrage mit vier Informationen (URL, Method, Data und Datatype) erstellt und an die Web Anwendung gesendet. Der URL legt fest zu welchem Service die Anfrage geht. Die Method setzt den Typ der Anfrage. Hier handelt es sich um eine Post Anfrage, dabei werden die Daten in dem Body der Anfrage übermittelt. Der zweite mögliche Typ, der auch in dieser Datei zu finden ist, ist der GET Typ, dabei werden die Daten in der URL verschickt. Die Data kapselt Daten die zum Service gesendet werden ab. Die letzte Information Datatype definiert den Typ der gesendeten Daten, hier geht es um JSON. Nach dem Empfangen der Antwort gibt es zwei Fälle zu unterscheiden. Bei erfolgreicher Antwort wird die Callback Funktion mit den Antwortdaten aufgerufen. Bei erfolgloser Antwort wird der Fehlertext im Popup Fenster angezeigt. Die Callback Funktionen sind in der Javascript Datei `Admin-interactionhandler.js` implementiert. Es geht dabei um die `callbackIfDoneForSendNewStatementTable` Funktion, die die Antwortdaten verarbeitet, indem sie zuerst die Daten mittels der `parseJSON` Methode in ein JavaScript-Objekt konvertiert um komfortabel auf deren Inhalte zuzugreifen. Danach wird eine Prüfung auf Vorhandensein von Fehlermeldungen durchgeführt. Ist dies der Fall, wird ein Fehlertext in Form einer Warnung kurz im Popup Fenster angezeigt. Andernfalls wird anhand der Methode `hide modal` das Popup Fenster geschlossen. Zum Schluss wird eine HTML Zeile erstellt und mit den Antwortdaten befüllt, Hier wird mittels der ID auf die Tabelle des Statement Bereichs zugegriffen, um diese Zeile am Ende der Tabelle anzuhängen.

Das hier beschriebene Vorgehen zum Hinzufügen von Statement ist identisch für alle anderen Objekttypen mit einem kleinen Unterschied zum Argument. Da Argument nach Issue angezeigt wird, wird in `Admin-interactionhandler.js` vor dem Erzeugen der HTML Zeile für das hinzugefügte Argument eine Prüfung durchgeführt. Stimmt das Issue des neuen Arguments mit dem im Argument Bereich aktuell selektierten Issue überein, wird die Argument Zeile am Ende der HTML Tabelle hinzugefügt. Ansonsten wird keine HTML Zeile addiert.

Daten in der Datenbank Editieren

Um Daten in der Datenbank zu editieren ist jede Zeile eines Tabellenbereichs mit einem EditButton versehen (Siehe Abbildung 4.5). Klickt der Administrator darauf oder auf die Zeile, wird das Klick Ereignis ausgelöst. Da jedem Tabelle-Bereich eine ID zugewiesen ist, kann auf eine bestimmte Zeile zugegriffen werden. Dies ermöglicht die Werte jeder Spalte einer Zeile in Variablen zu speichern. Das Listing 4.16 stellt das Ereignis und die Reaktion des Klicks einer Zeile vom Issue Bereich dar. Im ersten Schritt wird die angeklickte Zeile an die Klick Händler Funktion On angehängt. Im zweiten Schritt wird mit dem Schlüsselwort this die dazugehörige Zeile angesprochen, welche die Funktion find aufruft. Der Selektor eq selektiert eine Spalte mit einem bestimmten Index um deren Inhalt mittels der Methode html zurückzugeben und damit die entsprechende Variable zu setzen. Schließlich werden die Variablen an die Funktion showEditIssuesPopup als Parameter übergeben. Ziel ist es, dass der Administrator ein Edit Popup Fenster mit den dazugehörigen Werten angezeigt bekommt um sie editieren zu können.

```
$('#issue-table tr').on('click', function () {
    var uid, title, info, lang, author;
    uid = $(this).find('td').eq(0).html();
    author = $(this).find('td').eq(5).find('a').html();
    .....
    guiHandler.showEditIssuesPopup(uid, title, info, lang, author, new
        AdminInteractionHandler().callbackIfDoneForEditIssueTable);
});
```

Listing 4.16: Issue Editieren

In diesem Abschnitt wird folgendes erklärt, erstens wie die Funktion showEditIssuesPopup die Werte einer angeklickten Zeile im Popup Fenster anzeigt und zweitens wie diese verarbeitet werden. Wie vorher beim Daten Hinzufügen erläutert wurde, ist jedem HTML Element wie Eingabefelder, Radio Buttons und Dropdown-Menüs eine ID zugeordnet um den Zugriff mittels Javascript Anweisungen zu ermöglichen. Die Eingabefelder werden mittels der JQuery Funktion html mit den jeweiligen Eingaben des angeklickten Issues gefüllt. Der Radio Button für die Sprache eines Issues wird mit dem dazugehörigen Wert eingechekkt. Um den Dropdown-Menü Button mit dem dazugehörigen Author des angeklickten Issues anzuzeigen wird zuerst der Button Umschalter gefunden. Danach wird dessen Inhalt mit dem jeweiligen Autor als selektiert im Button gesetzt (Siehe Abbildung 4.7). Sobald der Administrator den Sichern Button anklickt, wird ein Ereignis ausgelöst. Der Button Sichern wird an die zwei Klick Händler Funktionen Off und On angehängt. Während die Off Funktion die ursprüngliche Event Händler Funktion abschaltet, fügt die On Funktion die aktuelle Event Händler Funktion

hinzu. Dies verhindert eine mehrfache Verarbeitung eines Klicks. Anschließend findet die Verarbeitung der zu editierenden Daten statt. Sie ist gleich wie die Verarbeitung der hinzufügenden Daten. Hier werden die Daten des zu editierenden issues mit der Callback Funktion als Parameter an die editIssue Funktion aus der Javascript Datei admin-ajaxhandler.js weitergeleitet.

In der editIssue Funktion wird eine AJAX Anfrage mit der URL edit-issue erstellt und an den jeweili-

Abbildung 4.7: Edit Issue Popup Fenster

gen Service gesendet. Das Vorgehen ähnelt sich dem AJAX Aufruf in sendNewStatement für das Hinzufügen von Statement, das schon hier beschrieben wurde. Ist die Anfrage erfolgreich in der Admin Webanwendung vonstatten gegangen, wird die Antwort an die Funktion callbackIfDoneForEditIssueTable aus der Admin-interactionhandler.js Datei zurückgegeben. Diese Funktion unterscheidet sich von den bisher betrachteten Callback Funktionen darin, dass hier die angeklickte Zeile manipuliert wird. Zuerst wird über die Tabelle des Issue-Bereichs iteriert um die entsprechende Zeile zu finden. Bei jeder Iteration wird geprüft, ob die Uid der jeweiligen Zeile mit der Uid aus der zurückgegebenen Antwort übereinstimmt. Falls ja, werden die Inhalte der zu editierenden Spalten durch die jeweiligen zurückgegebenen Daten ersetzt.

Das Editieren für die anderen Objekttypen läuft genauso ab. Nachdem auf die gewünschte Zeile geklickt wird, wird das entsprechende Edit Popup Fenster angezeigt. Wird im Popup Fenster eine Änderung vorgenommen und der Button Sichern betätigt, wird eine AJAX Edit Anfrage an das Admin Modul gesendet und dort verarbeitet. Ist die Verarbeitung erfolgreich abgelaufen, werden die neuen Werte in der HTML Zeile ersetzt. Ansonsten wird eine Fehlermeldung angezeigt.

Kapitel 5

Fazit und Ausblick

5.1 Fazit

Für Argumentationen und Diskussionen im Rahmen von Online-Partizipation wird ein neuartiger, auf Argumentationskarten basierender Prototyp entwickelt, der Schwächen von verschiedenen Software-Lösungen vermeidet. Dieser Prototyp unterscheidet sich von den anderen Lösungen darin, dass er eine echte Diskussion mit dem Wissen von Teilnehmern simuliert, das in der Datenbank gespeichert wird. Nach jeder Diskussion wächst die Datenmenge an Wissen. Deshalb besteht die Notwendigkeit, dieses Wissen in der Datenbank zu erstellen und zu pflegen. Dadurch, dass Einträge verschiedene Abhängigkeiten untereinander haben, ist die Wartung anstrengend und kompliziert. Daher war das Ziel dieser Arbeit, den Prototyp D-BAS um ein Admin Interface zu erweitern, das die Erstellung und Pflege der Datenbanktabellen des Prototypen vereinfacht. Um einen Überblick über mögliche Lösungswege im Hinblick auf Admin Interface zu gewinnen, wurden zuerst verschieden bestehende Python web Frameworks vorgestellt. Die meisten dieser Frameworks bieten ein integriertes Admin Interface an, das nur einfache CRUD Funktionen erlauben. Danach wurde eine Einführung über die verwendeten Technologien zur Realisierung des Admin Interface gegeben, unter anderem das Python Framework Pyramid und SQLAlchemy im Backend, Javascript und Bootstrap im Frontend. Anschließend wurden die Funktionalitäten, das Datenmodell und die Struktur des Prototyps D-BAS präsentiert. Abschließend wurde das implementierte Admin Interface vorgestellt. Zunächst wurde auf die Komponenten der Lösung eingegangen. Views und Hilfsfunktionen bilden das Backend. HTML Templates und Javascript Funktionen repräsentieren das Frontend. Views stellen den Ausgangspunkt zur Außenwelt dar. Sie verarbeiten die ankommenden Anfragen, indem sie die Hilfsfunktionen zur Kommunikation mit der Datenbank aufrufen. Die Hilfsfunktionen decken Funktionalitäten ab, wie Daten aus der Datenbank Selektieren, Speichern oder Aktualisieren. Hier wird die SQLAlchemy als Schnittstelle zur Datenbank verwendet. Die fertige Antwort wird dann an die Frontend zur Darstellung zurückgeschickt. Für die Gestaltung der Admin Webseite kamen das Bootstrap und CSS Framework zur Anwendung.

Die Admin Webseite ist mit einem Dashboard und einer Navigationsleiste versehen. Das Dashboard zeigt aggregierte Informationen über die Daten in der Datenbank sowie Links zur verschiedenen Bereichen unter anderem User Bereich und Thema Bereich. Die Navigationsleiste dient ebenfalls zum Navigieren zu den Bereichen. In jedem Bereich werden die dazugehörigen Daten sowie ein Link zum Addieren und ein Link zum Editieren angezeigt. Um die Interaktion des Administrators mit der Webseite zu ermöglichen wurde das Javascript Framework eingesetzt. Darüber hinaus wurde das AJAX Framework verwendet um Anfragen und Antworten zwischen dem berechtigten Administrator und dem Adminmodul auszutauschen.

5.2 Ausblick

Das im Rahmen dieser Bachelorarbeit entwickelte Admin Interface kann im D-BAS Projekt für die Verwaltung der Datenbank verwendet werden, wobei es im Laufe dieses Projektes noch zu verbessern und zu erweitern. Eine mögliche Verbesserung wäre zum Beispiel graphische Auswertungen von Änderungen einzubauen, wie das Verhalten der Votes und die Änderungen von Texten. Eine weitere sinnvolle Erweiterung wäre das Hinzufügen von neuen Daten über eine Import Funktion (z.B. CSV) anzubieten. Diese Funktion sollte den Administrator beim Hinzufügen von Daten per Formulare erheblich entlasten. Eine Export Funktion wäre auch gewünscht, wenn Berichte erstellt werden sollen. Mit wachsender Datenmengen in der Datenbank, was zu Unübersichtlichkeit führen könnte, wäre dringend nötig eine Suchfunktion zu implementieren. Ebenfalls sollte auch an Pagination (Seitennummerierung) gedacht werden. Hierbei werden die Inhalten auf mehreren Seiten dargestellt, was zu Übersichtlichkeit und Klarheit führt.

Literaturverzeichnis

- [Aut16] AUTHORS, The T.: *Tornado*. <http://www.tornadoweb.org/en/stable/>, 2016. Accessed: 05.08.2016
- [boo16] *Bootstrap*. <http://getbootstrap.com/>, 2016. Accessed: 05-09-2016
- [BRC11] BORCH, Malthe; REPOZE COMMUNITY the: *Chameleon*. <http://chameleon.readthedocs.io/en/latest/index.html>, 2011. Accessed: 14-08-2016
- [Cen16] CENTRAL, Python: *Understanding Python SQLAlchemy's Session*. <http://pythoncentral.io/understanding-python-sqlalchemy-session/>, 2016. Accessed: 08-08-2016
- [Com10] COMMUNITY, Zope D.: *ZOPE*. <https://docs.zope.org/zope2/zope2book/AppendixC.html>, 2010. Accessed: 14-08-2016
- [Con16a] CONSULTING, Agendaless: *Pyramid Security*. <http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/security.html>, 2016. Accessed: 14-09-2016
- [CON16b] CONSULTING, AGENDALESS: *SubstanceD Defining Content*. <http://docs.pylonsproject.org/projects/substanced/en/latest/content.html>, 2016. Accessed: 10-09-2016
- [CON16c] CONSULTING, AGENDALESS: *SubstanceD Installation*. <http://docs.pylonsproject.org/projects/substanced/en/latest/install.html>, 2016. Accessed: 10-09-2016
- [CON16d] CONSULTING, AGENDALESS: *SubstanceD Introduction*. <http://docs.pylonsproject.org/projects/substanced/en/latest/intro.html>, 2016. Accessed: 10-09-2016

- [CON16e] CONSULTING, AGENDALESS: *SubstanceD Management Views*. <http://docs.pylonsproject.org/projects/substanced/en/latest/mgmtview.html>, 2016. Accessed: 10-09-2016
- [dev14] DEVELOPERS, Kotti: *Kotti Documentation*. <http://kotti.readthedocs.io/en/latest/>, 2014. Accessed: 09-09-2016
- [Fou11] FOUNDATION, Zope: *ZODB*. <http://www.zodb.org/en/latest/>, 2011. Accessed: 10-09-2016
- [Fou16a] FOUNDATION, Django S.: *django documentation: Design philosophies*. <https://docs.djangoproject.com/en/1.10/misc/design-philosophies/>, 2016. Accessed: 08.08.2016
- [Fou16b] FOUNDATION, Django S.: *django documentation: FAQ General*. <https://docs.djangoproject.com/en/1.9/faq/general/>, 2016. Accessed: 08.08.2016
- [Fou16c] FOUNDATION, Django S.: *django documentation: Models*. <https://docs.djangoproject.com/en/1.10/topics/db/models/>, 2016. Accessed: 08.08.2016
- [Fou16d] FOUNDATION, Django S.: *django documentation: The Django admin site*. <https://docs.djangoproject.com/en/1.10/ref/contrib/admin/>, 2016. Accessed: 08.08.2016
- [Fou16e] FOUNDATION, Django S.: *django documentation: The Django template language*. <https://docs.djangoproject.com/en/1.7/topics/templates/>, 2016. Accessed: 08.08.2016
- [Fou16f] FOUNDATION, Django S.: *django documentation: Writing your first Django app, part 1*. <https://docs.djangoproject.com/en/1.10/intro/tutorial01/>, 2016. Accessed: 08.08.2016
- [Fou16g] FOUNDATION, Django S.: *django The web framework for perfectionists with deadlines*. <https://www.djangoproject.com>, 2016. Accessed: 26.07.2016
- [Fou16h] FOUNDATION, Plone: *SQLAlchemy*. <https://www.plone-entwicklerhandbuch.de/relationale-datenbanken/sqlalchemy.html>, 2016. Accessed: 08-08-2016
- [Fou16i] FOUNDATION, The jQuery: *The jQuery*. <http://jquery.com/>, 2016. Accessed: 13-08-2016
- [Hel16] HELLKAMP, Marcel: *Bottle*. <http://bottlepy.org/>, 2016. Accessed: 05-08-2016

- [jso] *Introducing JSON*. <http://www.json.org/>, . Accessed: 14-09-2016
- [KBBM16] KRAUTHOFF, Tobias; BAURMANN, Michael; BETZ, Gregor; MAUVE, Martin: *Dialog-Based Online Argumentation*, 2016.. . Accessed: 18-08-2016
- [Kov16] KOVAL, Serge S.: *Flask Admin*. <https://flask-admin.readthedocs.io/en/latest/>, 2016. Accessed: 05.08.2016
- [Net15] NETO, Clodoaldo P.: *WSGI*. <http://wsgi.tutorial.codepoint.net/>, 2015. Accessed: 04-08-2016
- [Oht16] OHTAMAA, Mikko: *Websauna Introduction*. <https://websauna.org/docs/narrative/background/intro.html>, 2016. Accessed: 08-09-2016
- [pyr16a] *Pyramid: Application Configuration*. <http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/configuration.html>, 2016. Accessed: 05-08-2016
- [pyr16b] *Pyramid: Creating a Pyramid Project*. <http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/project.html>, 2016. Accessed: 05-08-2016
- [pyr16c] *Pyramid Introduction*. <http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/introduction.html>, 2016. Accessed: 05-08-2016
- [pyr16d] *Pyramid: URL Dispatch* . <http://docs.pylonsproject.org/projects/pyramid/en/latest/narr/urldispatch.html>, 2016. Accessed: 05-08-2016
- [Ron16] RONACHER, Armin: *Flask web development one drop at a time*. <http://flask.pocoo.org/>, 2016. Accessed:05.08.2016
- [sql16a] *Pyramid: Databases Using SQLAlchemy*. http://docs.pylonsproject.org/projects/pyramid/en/latest/quick_tutorial/databases.html, 2016. Accessed: 08-08-2016
- [SQL16b] *SQLAlchemy: Basic Use*. http://docs.sqlalchemy.org/en/latest/orm/extensions/declarative/basic_use.html, 2016. Accessed: 08-08-2016
- [SQL16c] *SQLAlchemy: Engine Configuration*. <http://docs.sqlalchemy.org/en/latest/core/engines.html>, 2016. Accessed: 08-08-2016

- [ura15a] URALBASH: *pyramid_sacrud New Architecture*. <http://pyramid-sacrud.readthedocs.io/pages/contribute/architecture.html>, 2015. Accessed: 06-09-2016
- [ura15b] URALBASH: *pyramid_sacrud Overview*. <http://pyramid-sacrud.readthedocs.io/>, 2015. Accessed: 06-09-2016
- [w3s16a] W3SCHOOLS.COM: *AJAX Tutorial*. <http://www.w3schools.com/ajax/>, 2016. Accessed: 13-08-2016
- [w3s16b] W3SCHOOLS.COM: *Bootstrap Buttons*. http://www.w3schools.com/bootstrap/bootstrap_buttons.asp, 2016. Accessed: 05-09-2016
- [w3s16c] W3SCHOOLS.COM: *Bootstrap Dropdowns*. http://www.w3schools.com/bootstrap/bootstrap_dropdowns.asp, 2016. Accessed: 05-09-2016
- [w3s16d] W3SCHOOLS.COM: *Bootstrap Grids*. http://www.w3schools.com/bootstrap/bootstrap_grid_basic.asp, 2016. Accessed: 05-09-2016
- [w3s16e] W3SCHOOLS.COM: *Bootstrap List Groups*. http://www.w3schools.com/bootstrap/bootstrap_list_groups.asp, 2016. Accessed: 05-09-2016
- [w3s16f] W3SCHOOLS.COM: *CSS Tutorial*. <http://www.w3schools.com/css/>, 2016. Accessed: 13-08-2016
- [w3s16g] W3SCHOOLS.COM: *JavaScript Tutorial*. <http://www.w3schools.com/js/>, 2016. Accessed: 13-08-2016
- [w3s16h] W3SCHOOLS.COM: *jQuery Tutorial*. <http://www.w3schools.com/jquery/>, 2016. Accessed: 15-09-2016
- [wsg16] WSGI. <http://wsgi.readthedocs.io/en/latest/frameworks.html>, 2016. Accessed: 04-08-2016

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 22.September 2016

Imane Nazih

Bitte hier

die Hülle mitsamt DVD einkleben

Diese DVD enthält:

- Eine *pdf* Version der Bachelorarbeit
- Alle \LaTeX und Grafik Dateien mitsamt dazugehörigen Skripten, die verwendet wurden
- Der Quellcode, der während der Bachelorarbeit erarbeitet wurde
- Alle während der Evaluation angefallenen Messdaten
- Alle referenzierten Webseiten und wissenschaftlichen Arbeiten