



5. JUNI 2020

PROJEKTDOKUMENTATION

REALISIERUNG EINER WEB APPLIKATION IN VERBINDUNG MIT
EINEM WEBCRAWLER



Inhaltsverzeichnis

1 Projektbeschreibung	2
1.1 Projektziele	2
1.2 Verwandte Projekte	2
2 Projektplanung.....	2
2.1 Zeit- und Aufbauplanung	2
3. Projekt Realisierung	3
3.1 Aufbau	3
3.1.1 Django.....	4
3.1.2 Beautiful Soup.....	8
3.1.3 Frontend - Bootstrap.....	9
3.1.4 PythonAnywhere – Online stellen der Seite	10
3.3 Entstandene Schwierigkeiten	10
4. Projektauswertung.....	11
5. Ausblick.....	11
Verweise	12

1 Projektbeschreibung

1.1 Projektziele

In diesem Projekt werde ich eine einfache accountbasierte Website programmieren, welche von unserer Schulseite die Vertretungen einließt und diese dann individuell registrierten Nutzern, jeden Morgen in der Woche per E-mail verschickt. Dafür können sich die Nutzer auf meiner Internetseite registrieren und sich jederzeit anmelden. Angemeldet können die Nutzer dann ihre Lehrer auswählen bzw. hinzufügen, sowie E-Mail-Benachrichtigungen aktivieren oder deaktivieren.

1.2 Verwandte Projekte

In Betracht auf das Konzept und die Ziele dieses Projektes habe ich keine verwandten Projekte gefunden. Allerdings gibt es im Internet bereits viele große Internetseiten, welche sich genau auf das Webcrawling beziehen. So zum Beispiel die Internetseite Idealo.de [11]. Diese spezialisiert sich auf das Ermitteln von Preisen im Internet, um den Kunden schnellstmöglich den billigsten Preis eines Produktes zu nennen. Eine gewisse Ähnlichkeit gibt es hier zu meinem Projekt, nur dass ich die Informationen von der Schulseite entnehme. Diese werden allerdings nicht wie bei Idealo verwertet und analysiert, sondern an die Nutzer weitergegeben und gespeichert.

2 Projektplanung

2.1 Zeit- und Aufbauplanung

In Bezug auf die Planung hatte ich mir zuvor einen Plan gemacht wie ich das Projekt zeitlich sowie auch inhaltlich plane, um dem ganzen Struktur zu verpassen. Da dies mein erstes größeres Projekt ist, war es nicht so einfach sich die Zeit ordentlich einzuteilen, zumal ich mir dabei auch viel neues beibringen musste.

Mein Plan sah wie folgt aus:

Ziel	Geplant bis:
Erste Schritte mit Django und Bootstrap 4 : Landing Page erstellen	15.10
Models erstellen (z.B. User Model mit Namen, Passwort usw.) + Navigation Bar	25.10
Login und Registration Seite	28.10
Dashboard und MyTeachers Seite	10.11

Vertretungen auslesen und sortiert in der Datenbank speichern + Bugfixes	1.12
E-mails mit Hilfe von Django versenden	15.12
Vertretungen individuell versenden	8.1.20
Klausur-Daten (Raum, Stunden, etc.) verschicken	1.2
Bugfixes, Verbesserungen, Design Optimierung, Dokumentation	1.4.20

Zunächst habe ich begonnen die Basics zu lernen und das Grundgerüst der Seite aufzusetzen. Darunter zum Beispiel eine Landing Page zu erstellen. Nach weiterem Lernen habe ich mich dann an den schwierigeren Teil gesetzt und Nutzer spezifische Teile eingebaut. Beispielsweise eine Login- und eine Registrierungsseite. Folgend habe ich mich dann um den Webcrawler gekümmert und diesen mit der Website verbunden. Als letztes kam dann das Versenden per E-Mail.

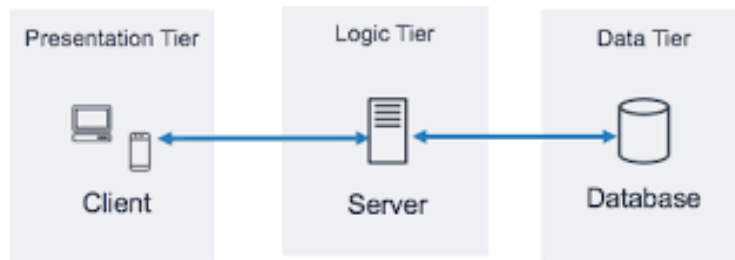
Zeitlich habe ich versucht die einzelnen Meilensteine innerhalb von 2 Wochen erledigt zu haben. Für einige größere Sachen habe ich mir allerdings mehr Zeit gegeben.

3. Projekt Realisierung

3.1 Aufbau

Mein Projekt basiert auf einer Drei-Schichten-Architektur und besteht aus den folgenden drei Teilen:

- Der Datenhaltungsschicht (**Datenbank**), in welcher die Vertretungen sowie alle Daten (Name, Lehrer, E-Mail-Adresse, usw.) von registrierten Nutzern gespeichert werden. Verwendet wird das Datenbanksystem SQLite welches das meist verbreitete Datenbanksystem der Welt ist [I2]
- Der Logikschicht (**Webserver bzw. Webapp**), welche die Daten der Nutzer entgegennimmt, die Vertretungen der Seite ausliest und aufgerufenen Inhalt wiedergibt. Daher kommt auch der Name. Der Begriff Server kommt nämlich von dem Begriff „Diener“, dieser liefert den gewünschten Inhalt [I3]
- Der Präsentationsschicht (**Browser**). Dieser dient dazu dem Nutzer die Daten zu präsentieren



[B1]

3.1.1 Django

Benutzt habe ich das Webframework Django, welches auf die Programmiersprache Python zurückgreift und die Logikschicht der Applikation darstellt (Webapp).

Entschieden habe ich mich dafür, weil es bereits viele wichtige Funktionen beinhaltet. Dadurch muss ich mich nicht im Detail um alles kümmern.

Im Folgenden werde ich näher auf vier dieser integrierten Funktionen von Django eingehen.

1. Sicherheit:

Wie jeder weiß ist die Sicherheit einer Website sehr wichtig, zumal es um die persönlichen Daten der Nutzer geht. Django hilft einem hierbei mit verschiedenen Security Features, um die geläufigsten Sicherheitsfehler zu beheben.

Einer der weit verbreitetsten und wichtigsten Security Features ist dabei die CSRF (cross site request forgery) Protection.

Bei einem CSRF Angriff fälscht der Angreifer eine Anfrage auf das System, hierfür wird von einem Nutzer die autorisierte Session übernommen. Diesem wird dann eine http-Anfrage untergeschoben, welche dann die vom Angreifer gewählte Aktion ausführt. Der Nutzer bekommt dabei nichts mit. [14]

Nun kommt die Protektion von Django ins Spiel. Überall dort wo man POST Anfragen durchführt, kommt das CSRF-Token von Django zu Nutzen.

```

<div class="card-body">
  <form action="" method="POST" class="form-signin">
    {% csrf_token %}
    <div class="form-label-group">{{ form.username }}</div>
    <div class="form-label-group">{{ form.password }}</div>
  </form>
</div>
  
```

[C1]

In diesem Beispiel eines Anmeldeformulars werden Formulardaten zur weiteren

Verarbeitung zum Server gesendet (POST-Method). Um hier Sicherheit zu garantieren wird die CSRF-Protection von Django verwendet (CSRF-Token).

2. Detaillierte Admin Seite:

Django beinhaltet eine detaillierte Admin Seite. Diese lässt sich mit einem Hinzufügen von „/admin“ and die URL auffinden. Nach dem man einen neuen Nutzer mit Admin-Berechtigungen erstellt hat, kann man sich dann auf der Admin Seite anmelden (siehe Abb. 1).

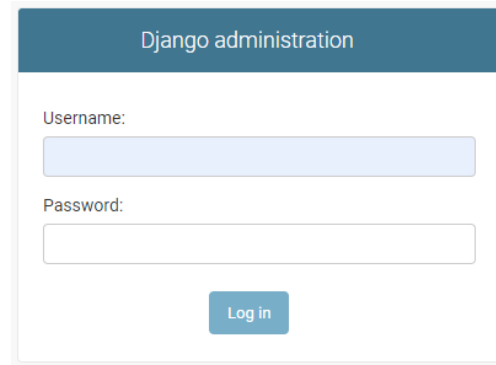
The image shows the Django administration login interface. It has a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". Below the password field is a blue "Log in" button.

Abb. 1

Nach der ersten Anmeldung wird diese zuerst allerdings noch ziemlich leer aussehen (siehe Abb. 2).

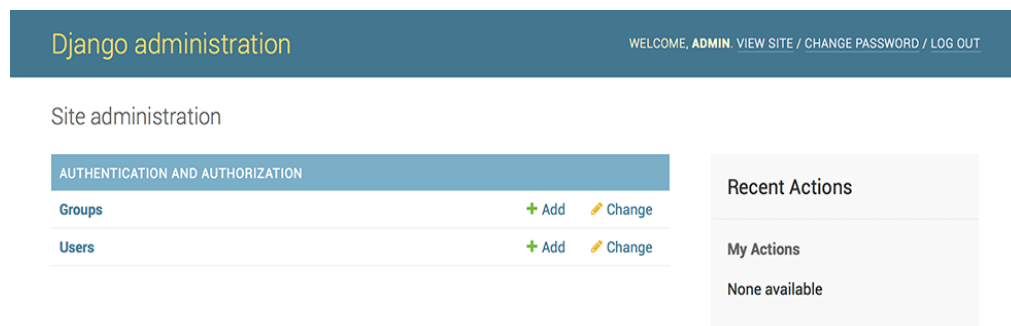
The image shows the Django administration site after a successful login. The header is dark blue with "Django administration" on the left and "WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT" on the right. Below the header, there is a section titled "Site administration". Under this section, there is a table with two rows: "Groups" and "Users". Each row has a blue header bar with "AUTHENTICATION AND AUTHORIZATION" and two links: "+ Add" and "Change". To the right of the table, there is a "Recent Actions" section with a "My Actions" subsection that says "None available".

Abb. 2

Anschließend kann man seine Datenbank Models manuell hinzufügen. Dies sieht wie folgt aus.

```
@admin.register(User)
class UserAdmin(admin.ModelAdmin):
    list_display = [
        'username',
        'first_name',
        'last_name',
        'date_joined',
        'is_subscribing',
    ]
```

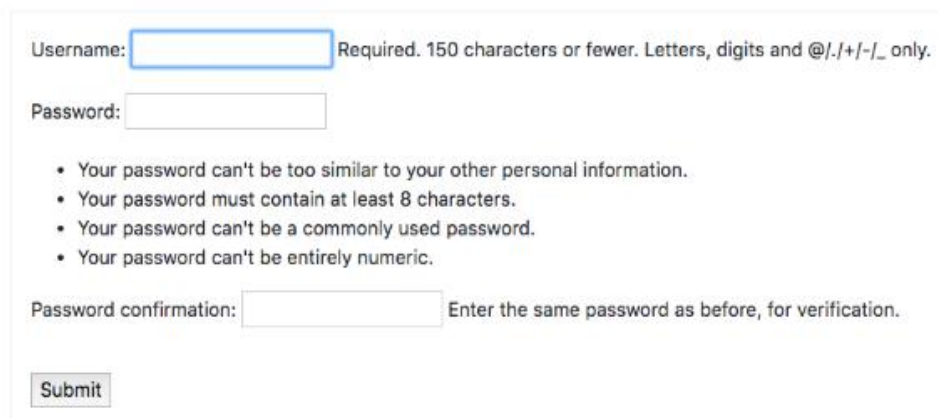
(Ergänzung der User Tabelle zur Admin-Seite) [C2]

Folgend lassen sich diese Datenbank Einträge dann auf der Admin-Seite auslesen.

3. Django Authentifizierungssystem:

Mit wenigen Schritten lassen sich verschiedene Authentifizierungsformulare importieren.

Hier ein Beispiel für ein importiertes Registrationsformular:



[B1]

Mit ein wenig Frontend lassen sich somit schnell und einfach hübsche Seiten zum Einloggen bzw. Registrieren erstellen.

4. E-Mail Schnittstelle:

Django beinhaltet weiterhin eine E-Mail Schnittstelle, die es besonders leicht macht Mails zu verschicken.

Diese Schnittstelle wäre beim Benachrichtigen der Nutzer zu neuen Vertretungen zum Nutzen gekommen. Nun wird sie allerdings nur beim Bestätigen der Registrierung verwendet.

```
# EMAIL SETTINGS
PRIVATE_EMAIL_USE_TLS = True
PRIVATE_EMAIL_HOST = 'smtp.gmail.com'
PRIVATE_EMAIL_HOST_USER = '...@gmail.com'
PRIVATE_EMAIL_HOST_PASSWORD = '...'
PRIVATE_EMAIL_PORT = 587
```

(Einstellungen, um einen Gmail Konto zu verwenden) [C3]

Nach dem Einstellen der Schnittstelle kann man wie folgt E-mails verschicken.

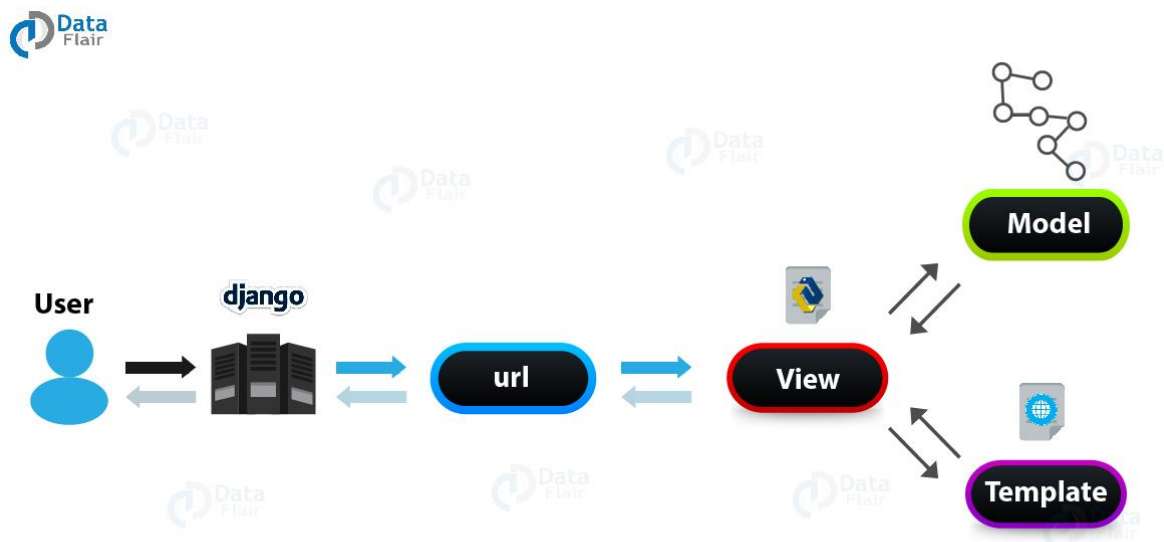
```
# Registration Email Content
subject = 'Thank you for your Registration ;)'
message = 'Hi ' + data.first_name + '! \n Your Registration was Successful! \n Thanks! ;)'
from_email = settings.EMAIL_HOST_USER
to_list = [data.email]

# send mail
send_mail(subject, message, from_email, to_list, fail_silently=False)
```

(Beispiel für eine E-Mail nach dem Registrieren) [C4]

Django Architektur

Django basiert auf einer Model-Template-View (MTV) Architektur.



[B2]

- Model: Dieser Teil beschreibt die Struktur der gespeicherten Daten einschließlich deren Attribute (z.B.: Feld Typ, Standard bzw. Start Wert, maximalen Länge, etc.)
- View: View stellt das Verbindungsstück zwischen Template und Model dar. Des Weiteren werden hier alle logischen Abläufe gesteuert, wie zum Beispiel das Präsentieren von Daten, welche von Model aus kommen und über das Template dem Nutzer vorgestellt werden
- Template: Dieser Teil ist nur für das Darstellen, ein- und ausgeben von Daten zuständig. Zu beachten ist dabei, dass dieser Teil keinerlei Verbindung zum Model Teil eingeht. Alles wird über den View Teil gesteuert.

Erwähnenswert ist auch die **Django Template Language**.

Sie verbindet statischen HTML Code mit spezieller Syntax und beschreibt wie dynamischer Inhalt dargestellt werden soll.

```
<div class="form-label-group">{{ form.username }}</div>
<div class="form-label-group">{{ form.password }}</div>

{% if form.errors %}
    {% for field in form %}
        {% for error in field.errors %}
            <div class="alert alert-danger">
                <strong>{{ error|escape }}</strong>
            </div>
        {% endfor %}
    {% endfor %}
    {% for error in form.non_field_errors %}
        <div class="alert alert-danger">
            <strong>{{ error|escape }}</strong>
        </div>
    {% endfor %}
{% endif %}
```

[C1]

In diesem Beispiel der Login Seite wurde die Django Template Language dafür verwendet Fehlermeldungen darzustellen, falls beim Einloggen etwas schief läuft, sowie um das Username und Passwort Feld Anzeigen zu lassen (ganz oben).

3.1.2 Beautiful Soup

Beautiful Soup ist eine Programmbibliothek, um Daten aus HTML- und XML- Dokumente auszulesen.

Im Folgenden werde ich einen Teil des Code zum Auslesen der Vertretungen näher beschreiben.

```

# Search all elements with class: "subst"
# subst = each day in the week
all_schedule_changes = soup.find_all(attrs={'class': 'subst'})

# Go throw all found Elements
for day, sub in enumerate(all_schedule_changes):
    # Skip day if there are no schedule changes
    if sub.find(text="Vertretungen sind nicht freigegeben"):
        continue
    data = sub.find_all('tr')[1:]
    for i, row in enumerate(data):
        create_schedule_change(row)

```

[C5]

Im obigen Teil des Codes werden alle Tabellen des Vertretungsplans gesucht und gespeichert. Diese stehen für die Vertretungen an den einzelnen Wochentagen und werden anhand der Klasse herausgesucht. (Alle Tabellen haben die Klasse „subst“.) Anschließend wird in jeder dieser Tabelle nach dem Text „Vertretungen sind nicht freigegeben“ gesucht. Wird dieser gefunden heißt das, dass an dem Tag keine Vertretungen vorliegen.

Wird dieser Text allerdings nicht gefunden werden alle Daten mit dem HTML-Tag „tr“ gesucht. Diese stellen die einzelnen Spalten pro Tag auf dem Vertretungsplan dar. Nachfolgend wird mit diesen Daten in der Funktion „create_schedule_change()“ weitergearbeitet bis sie zuletzt in der Datenbank gespeichert werden.

Der Code zum Auslesen der Vertretungen sollte eigentlich im 10 Minuten Takt im Hintergrund laufen, sodass alle neuen Vertretungen in der Datenbank gespeichert werden und bei Neuigkeiten den Nutzern per E-Mail verschickt werden. Unglücklicherweise steht der Vertretungsplan durch die Pandemie nicht mehr zur Verfügung, sodass dieser Teil des Projektes leider entfallen muss.

3.1.3 Frontend - Bootstrap

Bootstrap ist ein Frontend Framework, welches mit Hilfe von HTML, CSS und JavaScript dazu da ist, um anpassende Webseiten und Apps zu entwickeln. Des Weiteren beinhaltet Bootstrap bereits gestaltete Vorlagen, wie zum Beispiel Buttons [I5]. Diese waren besonders nützlich da ich später nur Kleinigkeiten wie die Farbe oder die Form ändern musste.



(Beispiel für bereits gestaltete Buttons [I6])

Weiterhin habe ich ein Bootstrap Template verwendet [17], um schnell ein schönes Design für meine Webapplikation zu entwickeln. Dieses habe ich im Nachhinein nach meinem Geschmack mit CSS angepasst.

Darüber hinaus habe ich mir auf der Internetseite <https://unsplash.com/> Hintergrundbilder für das Frontend heruntergeladen. Diese haben eine überragende Qualität und sind kostenfrei für jeden verfügbar.

3.1.4 PythonAnywhere – Online stellen der Seite

PythonAnywhere [18] ist ein online Web Hosting Service, mit diesem kann man ganz einfach auf WSGI basierte Webapplikationen online stellen. Hiermit habe ich mein Projekt auch online gestellt, damit Sie sich nicht um das lästige Aufsetzen und Migrieren kümmern müssen.

Nach dem Erstellen einer virtuellen Umgebung und dem Kopieren meines GitHub Repositorys, mussten noch paar Einstellungen getroffen werden und die Website war online. Hierbei habe ich jeglichen Teil des Webcrawlers ausgelassen, sodass keine fremden Daten ins Internet gestellt werden.

Die Website ist live unter: <http://fabibixd.pythonanywhere.com/>

3.3 Entstandene Schwierigkeiten

Natürlich entstehen beim Realisieren Schwierigkeiten. Nachfolgend sind einige dieser aufgelistet.

Datenschutz

Ein wichtiger Aspekt beim Erstellen einer Website ist der Datenschutz. Man kann nicht einfach Dinge über andere Personen ins Internet hochladen. Dasselbe Problem liegt auch in diesem Projekt vor. Da ich mit diesem Projekt den Nutzern preisgebe, wann Lehrer nicht unterrichten können entsteht hier wiederum ein Problem bezüglich des Datenschutzes der Lehrer. Um diese Daten hochladen zu dürfen, müsste ich von jedem Lehrer, der auf dem Vertretungsplan erscheinen könnte, eine Einwilligung erhalten. Da ich diese allerdings nicht habe, steht es mir rechtlich nicht zu die Vertretungen den Nutzern zu versenden.

Selenium

Anfangs hatte ich mir überlegt das Framework Selenium, anstatt BeautifulSoup zu verwenden. Zunächst habe ich den Webcrawler, welcher die Vertretungen auslesen soll mit Selenium programmiert. Allerdings funktioniert dieses Framework etwas anders als BeautifulSoup. Selenium ist etwas Einsteiger freundlicher, wobei BeautifulSoup mehr Erfahrung benötigt. Des Weiteren ist Selenium dafür ausgelegt Webseiten zu testen und läuft deswegen auf einer gewohnten Browseroberfläche. BeautifulSoup dagegen ist viel leistungseffizienter und kann im Hintergrund laufen gelassen werden.

Beim Umstieg von Selenium auf BeautifulSoup hatte ich demnach ein paar kleine Schwierigkeiten, da ich mich erneut in ein neues Framework einlesen musste.

Programmiererfahrung

Da ich mich mit all den Frameworks und deren Funktionsweisen noch nicht so auskannte, musste ich mich in jedes der verwendeten Frameworks einarbeiten. Dies hat sehr viel Zeit und Nerven in Anspruch genommen und hat wenig Spaß gemacht. Vor allem beim komplexeren Framework Django hatte ich zunächst viele Schwierigkeiten. Beim Lernen habe ich mich meist an den Dokumentationen der einzelnen Frameworks entlang gehandelt. Weiterhin habe ich von meinem Onkel an den Stellen, an denen ich Hilfe benötigte, welche erhalten.

Datenbank Migration

Eine weitere sehr verwirrende Schwierigkeit, die ich hatte, war die Migration der Datenbank. Nach dem Erstellen oder Löschen eines Models in der Datenbank, muss man diese anschließend migrieren. Da ich dies am Anfang nicht wusste und ahnungslos versucht habe den Server nach dem Erstellen oder Löschen eines Models zu starten, hat mich dies ein wenig Zeit gekostet. Schlimmer war dies noch, wenn man ein Model verändert hat (Beispielsweise von einem Char Field zu einem Integer Field). Dabei kam ich richtig in Verwirrung da man komischerweise die alten Migrationen wieder löschen musste. Dies zu verstehen hat mich viel Zeit gekostet.

4. Projektauswertung

Das Ziel war es in diesem Projekt eine Webapplikation mit der Nutzung eines Webcrawlers für den Datenempfang zu realisieren.

Allen in einem würde ich das Projekt als gelungen bezeichnen. Es musste zwar der Webcrawler des Projektes sowohl durch den Datenschutz der Lehrer als auch durch die Corona-Pandemie entfallen. Nichtsdestotrotz habe ich viele neue Erfahrungen in den Bereichen Webapplikationen, Datenbanken und Frontend sammeln können und eine voll funktionstüchtige accountbasierte Webapplikation aufsetzen können. Obwohl ich anfangs viel neues lernen musste und mit einigen Probleme kämpfen musste, hatte ich dennoch viel Spaß beim Realisieren meiner ersten Webapplikation.

5. Ausblick

Als Ausblick in die Zukunft kann ich mir vorstellen an diesem Projekt weiterzuarbeiten. Eine Idee das Projekt auszubauen wäre zum Beispiel das hinzufügen von Hausaufgaben, sodass die Webapplikation so gesehen als Kalender für Hausaufgaben genutzt werden könnte. Wie es im Endeffekt mit dem Verschicken von Vertretungen aussehen wird, kann man jetzt noch nicht sagen. Weiterhin lässt sich nach dem Fertigstellen rückschließend auf das nächste Projekt noch sagen, dass ich mir die Zeit bei der Planung besser einteilen sollte.

Verweise

School Helper Website: <http://fabibixd.pythonanywhere.com/>

Mein GitHub Repository: <https://github.com/FabibixD/MyProject>

Internetseiten:

- [I1] <https://www.ideal.de/>
- [I2] <https://de.wikipedia.org/wiki/SQLite>
- [I3] <https://www.hosteurope.de/blog/was-ist-ein-webserver/>
- [I4] <https://de.wikipedia.org/wiki/Cross-Site-Request-Forgery>
- [I5] [https://de.wikipedia.org/wiki/Bootstrap_\(Framework\)](https://de.wikipedia.org/wiki/Bootstrap_(Framework))
- [I6] <https://getbootstrap.com/docs/4.0/components/buttons/>
- [I7] <https://startbootstrap.com/themes/stylish-portfolio/>
- [I8] <https://www.pythonanywhere.com/>

Bilder:

- [B1] https://d1.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf
- [B2] <https://medium.com/himachaliprogrammer/django-for-real-developers-2-9e49aaa7eba1>
- [B3] <https://data-flair.training/blogs/django-architecture/>

Code:

- [C1] MyProject/users/templates/users/login.html
- [C2] MyProject/users/admin.py
- [C3] MyProject/school_app/email_info.py
- [C4] MyProject/users/views.py
- [C5] MyProject/courses/scraping/scraping.py