TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Entwicklung einer internen Social Media Plattform mit personalisierbarem Dashboard fur Studierende

Esther Kleinhenz

Matrikelnummer: 2649270

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Engineering

Media Engineering

Erstprüfer: Prof. Dr. Oliver Hofmann
Zweitprüfer: Prof. Dr. Matthias Hopf

Nürnberg, 10. Oktober 2018

Hiermit erkläre ich, dass die vorliegende Arbeit von mir selbständig verfasst und nicht

anderweitig für Prüfungszwecke vorgelegt wurde, keine anderen als die angegebenen

Quellen oder Hilfsmittel benutzt sowie wörtliche oder sinngemäSSe Zitate als solche

gekennzeichnet wurden.

Nürnberg, 10. Oktober 2018

Katja Cornelia Hader
E-mail: haderka56442@th-nuernberg.de

Studiengang Wirtschaftsinformatik
Georg Simon Ohm
Technische Hochschule Nürnberg
KeSSlerplatz 12
90489 Nürnberg
Deutschland

Abstract

Content of this Bachelor thesis is ....

Keywords: bla,bla Georg Simon Ohm, Wirtschaftsinformatik

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die vorliegende Arbeit beschäftigt sich mit der wachsenden E-Mail-Flut in den Postfächern der Studierenden und wie man diese reduzieren kann. Schon seit geraumer Zeit ist bekannt, dass das Versenden von Informationen über elektronische Post nicht nur Vorteile mit sich bringt. Wie der Spezialist für Gesundheitsprozessberatung in einem Bericht der Mitteldeutschen Zeitung erwähnt, macht es die stets wachsende E-Mail-Menge unmöglich sich vernünftig mit den Informationen zu befassen"([Ver13]). Nicht nur am Arbeitsplatz sondern auch in Hochschulen wird Gebrauch gemacht, weitere Empfänger oder sogar ganze Verteiler mit in die Kopie einer E-Mail zu integrieren. Um die Prioritäten der Informationen besser bilden zu können, sollen Studierende selbst entscheiden, welche Nachrichten relevant sind. Hierfür wird eine Social Media Plattform mit personalisierbarem Dashboard implementiert.

1.1 Ausgangssituation

Alle Informationen der Fakultat Elektrotechnik Feinwerktechnik Informationstechnik, kurz efi, werden uber die globalen Verteiler des Hochschulinternen Postfaches versendet. Viele dieser Daten sind jedoch nur fur eine geringe Schnittmenge der Empfanger relevant und lassen sich nur schwer priorisieren. Das standig uberlastete Postfach muss somit regelmaßig gepflegt werden. Einen massiven Administrativen Aufwand bedeutet es, E-Mails zu filtern und nach persönlichem Ermessen zu verwalten. Zudem leidet die Nachhaltigkeit der Informationen. Mochten die Empfanger altere E-Mails abrufen, mussten diese meist schon entfernt werden um Platz fur den neuen, eintreffenden E-Mail-Verkehr zu schaffen. Diese Situation fuhrt dazu, dass Empfanger die Informationen meist nicht lesen und sofort entfernen. Die Ersteller haben keinerlei Moglichkeiten zu uberprufen ob und wie viele Studierende und Dozenten eingehende Nachrichten offnen und lesen.

1.2 Ziel der Arbeit

Ziel der Arbeit ist es, durch die Einbindung einer Social Media Plattform den Speicheraufwand des Hochschulpostfaches fur Studierende der Efi-Fakultat zu reduzieren. Die Flut an E-Mails soll durch das Verwenden eines personalisierte Dashboard gedrosselt werden. Hierbei wird zunächst der Fokus auf die grundlegenden Funktionen der Website gelegt. Dazu gehört das Abonnieren, einpflegen von neuen und löschen von alten Nachrichten. Zudem sollen die Autoren benachrichtigt werden, in welchem Umfang die hochgeladenen Informationen bereits abonniert und gelesen wurden.

Kapitel 2

Framework

Um die Website-Erweiterung realisieren zu können, wird zunächst festgelegt welche Programmierschnittstellen verwendet werden. Im Web-Backend fällt die Wahl auf die objektorientierte Sprache Python, die ausschließlich Serverseitig anwendbar ist. Der Programmaufbau Pythons macht den Code leicht lesbar und der einfache Syntax ermöglicht einen strukturierte Implementierung der Website([Ndu17]). Ein entscheidender Vorteil hierbei ist das dazugehörige Framework Django, auf das im folgenden Kapitel genauer eingegangen wird.

2.1 Django

Django ist ein Web-Framework, das auf einer Model-View-Presenter (MVP) Architektur basiert. Ähnlich wie der Model-View-Controller sind die Interaktionen zwischen Model und View die Auswahl und Ausführung von Befehlen und das Auslösen von Ereignissen (vgl. Abbildung 2.1). Da die View aber hier bereits den Großteil des Controllers übernimmt, ist der MVP eine Überarbeitung. Der Teil, der Elemente des Modells auswählt, Operationen durchführt und alle Ereignisse kapselt, ergibt die Presenter-Klasse([She09]). Durch die direkte Bindung von Daten und View, geregelt durch den Presenter, wird die Codemenge der Applikation stark reduziert.

Der Prozess vom Anfragen der URL über den Server, bis hin zur fertig gerenderten Website kann wie folgt vereinfacht darstellen. Der User gibt eine URL im Browser ein und sendet sie an den Web-Server.

Das Interface WSGI am Web-Server verbindet diesen mit dem Web-Framework, indem es den Request zum passenden Objekt weiterleitet. Hier wird der Applikation eine Callback-Funktion zur Verfügung gestellt [Kin17]. Außerdem werden folgende Schritte durchgeführt:

- Die Middleware-Klassen aus der settings.py werden geladen

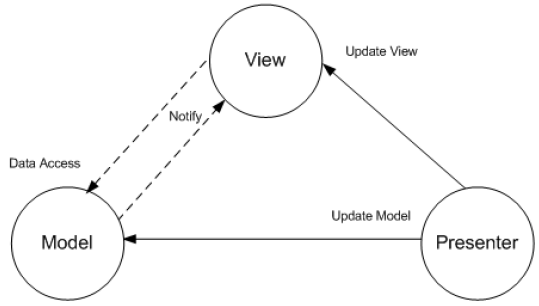

Abbildung 2.1. Vereinfachter MVP

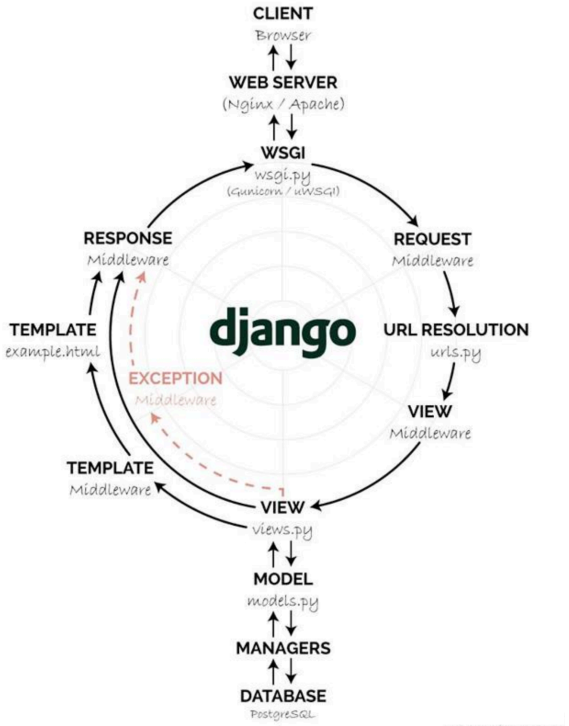

Abbildung 2.2. Request-Response-Kreislauf des Django Frameworks

- Die Methoden der Listen Request, View, Response und Excpetion werden geladen
- Die angeforderte URL wird aufgelöst

Der WSGI-Handler fungiert also als Pförtner und Manager zwischen dem Web-Server und dem Django-Projekt. Um die URL, wie weiter oben erwähnt, aufzulösen, benötigt WSGI einen urlresolver. Durch die explizite Zuweisung der vorhandenen Seiten, kann dieser über die regulären Ausdrücke der url.py-Datei iterieren. Gibt es eine Übereinstimmung, wird die damit verknüpfte Funktion in der View (view.py) aufgerufen. Hier ist die gesamte Logik der Website lokalisiert. Wie bereits erwähnt,

ist es möglich unter Anderem auf die Datenbank der Applikation zuzugreifen und Eingaben des Users über eine Form zu verarbeiten. Nachdem werden die Informationen der View an das Template weitergereicht. Es handelt sich dabei um eine einfache HTML-Seite in der der strukturelle Aufbau im Frontend festgelegt wird. Die Informationen der View können hier zwischen doppelt-geschweiften Klammern eingebunden und, wenn nötig, mit einfachen Python-Befehlen angepasst werden. Nun kann das Template, die vom WSGI-Framework zur Verfügung gestellte Callback-Funktion befüllen und einen Response an den Web-Server schicken. Die fertige Seite ist beim Klienten im Browserfenster zum rendern bereit (vgl. Abbildung 2.1).

2.1.1 Besonderheiten

Das Django-Framework bringt einige Besonderheiten mit sich, die beim implementiern des Prototypen von Bedeutung sind. Diese werden im Folgenden beschrieben.

Die Administratoroberfläche ist eines der hilfreichsten Werkzeugen des gesamten Frameworks. Es stellt die Metadaten der Modelle aus dem Code visuell dar. Verifizierte Benutzer können die Daten nicht nur schnell erfassen, sondern diese auch editieren und verwalten. Das Recht, das Admin-Backend uneingeschränkt zu benutzen, ist dem sogenannten superuser vorenthalten. Dieser kann beim erstmaligen zuweisen nur über die Kommandozeile eingerichtet werden. Ist bereits ein superuser vorhanden, kann dieser im Admin-Backend weiteren Benutzern den gleichen Handlungsfreiraum einräumen. Zudem gibt es noch weitere Stufen der Zugangsberechtigungen, Staff- und Active-Status, die für eine breitere Gruppe von Benutzern geeignet ist. Um die gestaffelten Zugangsberechtigungen auch auf der Website umsetzen zu können, stellt Django verschiedene Decorator zur Verfügung. Soll eine bestimmte Seite nur von eingeloggten Benutzern besucht werden dürfen, so importiert man die Decorator des, von Django zur Verfügung gestellten, Authentifizierungssystems mit

```
from django.contrib.auth.decorators import login_required
```

Direkt über den Beginn der Funktion in view.py, oder auch single-view-function, wird zudem folgende Zeile ergänzt:

```
@login_required
```

Natürlich lassen sich die Decoratoren auch für andere Zwecke vor Funktionen platzieren. Begrenzte Zugänge zu den Views können durch die Art der Anfrage realisiert werden. Der Benutzer muss also zum Beispiel durch GET auf eine Seite zugreifen wollen um Einsicht zu erhalten.

Benutzerdefinierte Decorator sind im Django-Framework möglich, darauf wird hier aber nicht weiter eingegangen.

2.2 Erweiterungen

Durch das hohe Ansehen, dass Django in der Branche genießt ist der Pool an Erweiterungen groß.

pip?

allg Erweiterungen und dann genauer Taggable Manager

2.2.1 Taggable-Manager

Django-taggit ist eine Erweiterung von Alex Gaynor, einem Entwickler aus Washington DC. Das Add-on ermöglicht das Verwenden von Tags die automatisch mit einem eindeutigen Slug versehen werden. Der darin enthaltene Taggable Manager verwendet Django's Contenttype Framework, welches per Default verwendet wird, um die Modelle der Applikation zu verfolgen und diese durch generische Beziehungen zu verknüpfen. Die Felder app_label und model machen die Modelle eindeutig zuweisbar. Instanzen des Contenttypes präsentieren und speichern die Informationen und Erstellen automatisch neue Instanzen, wenn Modelle hinzugefügt werden. Zudem stehen Methoden zur Verfügung, die das Abrufen und Arbeiten mit Instanzen der einzelnen Modelle erleichtern.

Der Taggable-Manager ist jedoch nicht an das Contenttype-Framework gebunden ([Her16]). Durch die Verwendung eines echten Fremdschlüssels, kann zum Beispiel die Performance und Referenzgarantie verwirklicht werden. Dazu ist lediglich die Erstellung einer eigenen lookup-Tabelle notwendig, die die Entitäten zweier Tabellen direkt verlinkt, anstatt diese generische zu verbinden. Weiterführend können Modelle in einem benutzerdefinierten Modell vereint werden, sodass er Zugriff auf diese einheitlich geschieht. AuSSerdem ist es möglich Primary-Keys zu verwenden, die nicht aus ganzen Zahlen bestehen, sondern aus Buchstaben und Wörtern.

Um django-taggit zu installieren wird der folgende Befehl in die Kommandozeile eingefügt([Gay10]):

```
$ pip install django-taggit
```

Im model.py wird das Feld tag neu erstellt und als Taggable Manager definiert. Außerdem muss taggit in der settings.py unter INSTALLED_APPS ergänzt werden. Um dem Programm zu sagen, dass nun eine neue Liste der Datenbank hinzugefügt werden muss, werden folgende Befehle in die Kommandozeile eingefügt:

```
$ python3 manage.py makemigrations
$ python3 manage.py migrate
```

Im Admin-Backend kann nun geprüft werden, ob das neue Feld in die Datenbank integriert wurde. Neue Tags können in das Textfeld eingetragen werden. Der Parser

verarbeitet jedes Wort, dass durch ein Komma oder ein Leerzeichen getrennt ist als ein Tag. Soll dieses jedoch aus mehreren Wörtern bestehen so müssen diese mit Anführungszeichen umfasst werden. StandardmäSSig unterscheidet der Taggable Manager zwischen GroSS- und Kleinschreibung, Tags sind also case sensitive. Ändern kann man das, indem der Boolean TAGGIT_CASE_INSENSITIVE in der settings.py auf True gestellt wird.

2.3 Bootstrap

Eine umfangreiche Website einheitlich zu gestalten ist oft sehr komplex und zeitaufwendig. Die Entwickler von Twitter haben deshalb, zunächst Firmenintern, an einem neuen Verwaltungswerkzeug gearbeitet, das mehrere Bibliotheken zusammenführen sollte. Sie merkten, das die neue Bibliothek, die daraus entstand, nicht nur auf Ihre eigene Website anwendbar, sondern so flexible ist, dass jede Art von Website davon profitieren könnte. 2011 entschieden Sie sich Bootstrap für die Öffentlichkeit frei zugänglich zu machen. Die Open-Source-Bibliothek, die auf GitHub abrufbar ist, wird seitdem von vielen interessierten Programmierern weiterentwickelt und ist somit stark gewachsen. Version 2.0 verfügt außerdem über die Fähigkeit Websites responsive auf verschiedenste mobile Endgeräte anzupassen.

Das Bootstrap-Paket beinhaltet vorgefertigte Cascading Stylesheets, kurz CSS, die Farben, Schriftarten und viele weitere Stildefinitionen. Zudem befinden sich auch Erweiterungen des JavaScript-Frameworks jQuery in der Bibliothek, die weiterführende Funktionen beinhalten wie zum Beispiel Filter oder Dropdown-Menüs. Das Paket kann einfach eingebunden werden im head-tag einer HTML-Datei (vgl. Abbildung 2.3).

```
<head>
    <link href="{% static 'bootstrap/css/bootstrap.css' %}" rel="stylesheet">
</head>
```

Abbildung 2.3. Einbindung von Bootstrap in einer HTML-Datei

Durch das Einbinden von Bootstrap in einer HTML-Datei werden einige Styles bereits automatisch auf die darin vorkommenden Tags, wie Links und Überschriften, angewandt. Dies ist jedoch nur ein sehr kleiner Teil den die Bibliothek zur Verfügung stellt. Möchte man Bootstrap umfangreich nutzen so lassen sich die Stildefinitionen mit Klassen oder ID's in diverse HTML-Tags eintragen (vlg. Abbildung).

Möchte man bestimmte gestalterische Eigenschaften von Bootstrap überschreiben muss eine eigens verfasste CSS-Datei nach der Verlinkung von Bootstrap in die

```
<div class="content container">
    <div class="row">
        <div class="col-md-8">
            {% block content %} {% endblock %}
        </div>
    </div>
</div>
```

Abbildung 2.4. Bootstrap-Klassen in HTML-Tag

Website eingebunden werden. Der Parser liest die Datei von oben nach unten, Links nach Rechts. Liest dieser also zu erst die Bootstrap Bibliothek und speichert diese, so überschreiben die Styles die danach kommen, die bereits gelesenen Eingaben. Die Styles, die inline auf ein Tag angewendet werden sind somit die bestimmenden Eigenschaften. Natürlich sollten Stildefinitionen niemals inline eingepflegt werden, da dies zu einem sehr unübersichtlichen und wartungsintensiven Code führen.

Kapitel 3

Prototyp

3.1 Organisation

3.1.1 Verwaltung im Administrator-Backend

3.1.2 Berechtigung der User

3.2 Funktion

3.2.1 Abonnieren

3.2.2 Filtern

3.2.3 Benachrichtigung

Kapitel 4

Ergebnis

4.0.1 Evaluierung

Kapitel 5

Zusammenfassung und Ausblick

Zusammenfassung...

Referenzen

[Gay10] Alex Gaynor. Exploring django-taggits data model. 2010. https://django-taggit.readthedocs.io/en/latest/getting_started.

[Her16] Stephan Herzog. Model view controller, model view presenter, and model view viewmodel design patterns. 2016. https://medium.com/sthzg/a-short-exploration-of-django-taggit-bb869ea5051f.

[Kin17] Adam King. Django middlewares and the request/response cycle. 2017. https://medium.com/zeitcode/django-middlewares-and-the-request-response-cycle-fcbf8efb903f.

[Lei13] Ingo Leipner. Stress fur beschaftigte: Wie kann man die e-mail-flut bekampfen. 2013. http://www.mz-web.de/wirtschaft/e-mail-flut-mails-bearbeiten-kommunikation-stress-zeit-sparen.

[Ndu17] Nnenna Ndukwe. Python is the back-end programming language of the future and heres why. 2017. https://medium.com/@nnennahacks/https-medium-com-nnennandukwe-python-is-the-back-end-programming-language-of-the-future-heres-why.

[She09] Alexy Shelest. Model view controller, model view presenter, and model view viewmodel design patterns. 2009. https://www.codeproject.com/Articles/42830/Model-View-Controller-Model-View-Presenter-and-Mod.