Technische Hochschule Nürnberg Georg Simon Ohm
Fakultät Elektrotechnik Feinwerktechnik
Informationstechnik

Studiengang Media Engineering

Bachelorarbeit von
Esther Beate Kleinhenz

Entwicklung einer internen Social Media Plattform mit personalisierbarem Dashboard fur Studierende

WS 2018/2019

Abgabedatum: 7. Dezember 2018
Betreuer: Prof. Dr. Oliver Hofmann

Schlagworte: Python, Django, Informationsplattform, Tagging, E-Mail-Flut

Hinweis: Diese Erklärung ist in alle Exemplare der Abschlussarbeit fest einzubinden. (Keine Spiralbindung)

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Kleinhenz Vorname: Esther Matrikel-Nr.: 2649270

Fakultät: Elektro-, Feinwerk-, Informationstechnik Studiengang: Media Engineering

Semester: Wintersemester

Titel der Abschlussarbeit:

Entwicklung einer internen Social Media Plattform mit personalisierbarem Dashboard für Studierende

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum, Unterschrift Studierende/Studierender

Erklärung zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit ☐ genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

☐ genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Ort, Datum, Unterschrift Studierende/Studierender

Formular drucken

Abstract

Das Ziel der vorliegenden Bachelor Arbeit ist es, zu beweisen, dass die E-Mail-Flut der Hochschule durch den Einsatz einer Social Media Plattform gedrosselt werden kann. Dazu wird ein Prototyp implementiert der ein personalisierbares Dashboard für Studierende und Angestellte bereitstellt.

Das Abonnieren von Artikeln wird mit dem Einsatz von Tags umgesetzt. Benutzer des Systems, versehen Nachrichten mit Schlagwörtern und machen die Informationen dadurch schneller zuweisbar. Durch die einfach Suche von Tags können Studierende Artikel nach eigenem Ermessen zum Dashboard hinzufügen.

Durch den Zähler eines jeden Artikels können Autoren sehen, ob Informationen bereits gelesen wurden. Hierdurch lassen sich Schlüsse über die Nutzung der Applikation und die Relevanz des Artikels ziehen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Eine Vielzahl an ungelesene E-Mails häufen sich täglich in den Postfächern eines Jeden an. Das elektronische Übertragen von Nachrichten ist aus der heutigen Zeit nicht mehr wegzudenken. In der Vergangenheit hat sich jedoch gezeigt, dass das Versenden von Informationen 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"(vgl. [Ver13]). Nicht nur am Arbeitsplatz, sondern auch an Hochschulen wird Gebrauch davon gemacht, weitere Empfänger oder sogar ganze Verteiler mit in die Kopie einer E-Mail zu integrieren. Hierdurch steigen die irrelevanten Informationen unkontrollierbar schnell an. Infolgedessen nimmt der benötigte Speicheraufwand der zahlreichen kommerziellen, aber auch internen E-Mail-Dienste enorm zu (vgl. [Fio14]). Das Dezimieren dieses Kommunikationswegs ist jedoch in vielen Fällen nicht möglich und Beschränkungen jeglicher Art werden als nicht sinnvoll erachtet.

Betrachtet man darüber hinaus den Lebenszyklus einer einzelnen E-Mail, wird deutlich, dass dieser nach dem Erstellen, Senden und Empfangen noch nicht abgeschlossen ist. Nachdem die Information vom Adressat geöffnet wurde, wird sie archiviert, muss aber jederzeit durch eine Suchabfrage sofort angezeigt werden können. Dies verdeutlicht den enormen Aufwand, das das Verwalten elektronischer Post mit sich bringt. Aufbauend auf dieser Problematik wird folgend die Ausgangssituation der Arbeit erläutert.

1.1 Ausgangssituation

Alle Informationen der Fakultat Elektrotechnik Feinwerktechnik Informationstechnik werden über die globalen Verteiler des Hochschulinternen Postfaches versendet.

Viele dieser Daten sind jedoch nur für eine geringe Schnittmenge von Empfängern relevant und enthalten Mitteilungen oder Anhänge, die keinerlei Mehrwert dem Einzelnen aufweisen können. Dadurch sind die Postfächer der Studierenden und Dozenten schnell überlastet und können, ohne regelmäßige Pflege, nicht in vollem Umfang genutzt werden. Zudem lassen sich Informationen schwer priorisieren und der massive administrative Aufwand für den Einzelnen, E-Mails selbstständig zu filtern und nach persönlichem Ermessen zu verwalten, steht in keinem Verhältnis zum Mehrwert eines performanten, aufgeräumten Postfaches.

Die Nachhaltigkeit der Informationen kann meist nicht gewährleistet werden. Grund dafür ist der mangelnde Speicherplatz, verursacht durch die ankommende Nachrichtenflut. Möchten die Empfänger ältere E-Mails abrufen, müssen diese meist entfernt werden um Platz für den neuen, eintreffenden E-Mail-Verkehr zu schaffen. Dies kann das Nichtlesen der Informationen seitens der Empfanger verursachen und führt im schlechtesten Fall zum voreiligen Entfernen von relevanten Nachrichten.

Die Ersteller der Nachrichten haben keinerlei Möglichkeiten zu überprüfen, ob und wie viele Studierende und Dozenten eingehende Nachrichten öffnen und lesen. Eine solche Art der Transparenz wäre jedoch hilfreich, um Informationen inhaltlich zu optimieren und Überschriften treffender zu formulieren. Aus dieser Situation ergibt sich folgende Forschungsfrage: „Kann die E-Mail-Flut der Technischen Hochschule mit Hilfe einer Social Media Plattform gedrosselt und die Nachhaltigkeit der Informationen gewährleistet werden?"

1.2 Ziel der Arbeit

Ziel der Arbeit ist es, durch die Einbindung einer Social Media Plattform in die bereits bestehenden Hochschulwebsite den Speicheraufwand des Postfaches für Studierende der Fakultat zu reduzieren. Durch selbständiges Prüfen der Nachrichtenseite nehmen die Zielgruppen Informationen bewusster wahr und können diese individuell an ihre aktuelle Situation anpassen. Das reduziert den administrativen Aufwand und verhindert Speicherengpässe im E-Mail-Postfach. Broadcast ähnliches Senden von Informationen ist hierdurch nur noch in den seltensten Fällen nötig.

Der Schwerpunkt dieser Arbeit liegt auf der prototypischen Umsetzung der Website-Erweiterung. Hierbei wird zunächst der Fokus auf die grundlegenden Funktionen gelegt. Dazu gehört das Abonnieren, das Einpflegen von neuen und das Löschen von alten Nachrichten. Um den Informationsfluss jedes Einzelnen nicht aus den Augen zu verlieren, soll in regelmäßigen Abständen eine automatisierte E-Mail verschickt werden. Zudem sollen die Autoren einsehen können, in welchem Umfang die ver-

öffentlichten Informationen bereits gelesen wurden. Dadurch lässt sich nach einer gewissen Zeit feststellen, ob die Studierenden und Dozenten die Nachrichten für relevant erachten und die Plattform weiterhin als verlässliches Portal rentabel ist.

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 serverseitig anwendbar ist. Der Programmaufbau Pythons macht den Code leicht lesbar und die einfache Syntax ermöglicht eine strukturierte Implementierung der Website (vgl. [Ndu17]). Durch den modularen Aufbau ist es selbst für unerfahrene Entwickler möglich, die Sprache schnell zu erlernen. Darüber hinaus bringt Python verschiedene Web-Service Tools mit sich, die beim Implementieren einer Website viel Zeit sparen und das Aneignen von komplexen Protokollen redundant machen (vgl. [Sol17]). Das dazugehörige Framework Django reduziert den Entwicklungsaufwand eines Prototypen erheblich und ist daher als zielführendes Framework die beste Wahl.

2.1 Django

Django ist ein Web-Framework, das eine schnelle, strukturierte Entwicklung ermöglicht und dabei ein einfaches Design beibehält. Der darin enthaltene Model-View-Presenter (MVP) kann, ähnlich wie der Model-View-Controller, die Interaktionen zwischen Model und View, die Auswahl und Ausführung von Befehlen und das Auslösen von Ereignissen steuern (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 (vgl. [She09]). Durch die direkte Bindung von Daten und der 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 dargestellt werden.

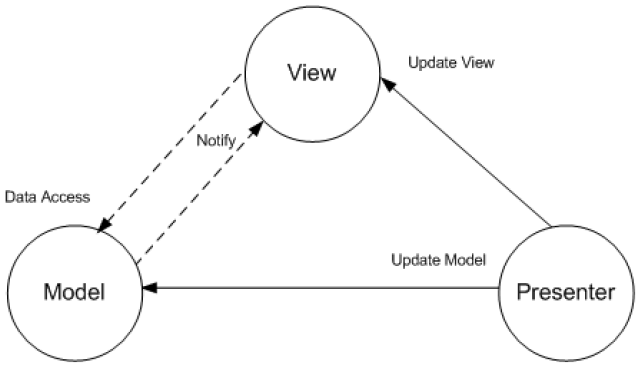

Abbildung 2.1. Vereinfachter MVP [She09]

Der User gibt eine URL im Browser ein und sendet sie an den Web-Server. Das Interface WSGI (Web Server Gateway Interface) 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 (vgl. [Kin17]). Außerdem werden folgende Schritte durchgeführt:

- Die Middleware-Klassen aus der `settings.py` werden geladen
- Die Methoden der Listen Request, View, Response und Excpetion werden geladen
- Die angeforderte URL wird aufgelöst

Der WSGI-Handler fungiert als Pförtner und Manager zwischen dem Web-Server und dem Django-Projekt. Um die URL aufzulösen, benötigt WSGI einen *urlresolver*[1] (vgl. [Dja18b]). 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 (`views.py`) aufgerufen. Hier ist die gesamte Logik der Website lokalisiert. Unter anderem ist es möglich auf die Datenbank der Applikation zuzugreifen und Eingaben des Users über eine Form zu verarbeiten. Anschließend 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. Das Template kann dann die vom WSGI-Framework zur Verfügung gestellte Callback-Funktion befüllen und eine Response an den Web-Server schicken. Die fertige Seite ist beim Klienten im Browserfenster zum Rendern bereit (vgl. [Kin17], Abbildung 2.2.).

[1] Urlsresolver verknüpft Url-Muster mit den passenden Views.

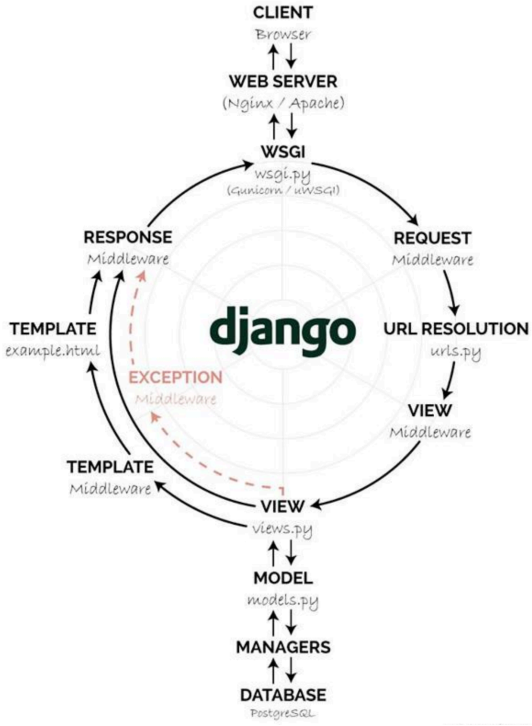

Abbildung 2.2. Request-Response-Kreislauf des Django Frameworks [Nev15]

2.1.1 Besonderheiten Djangos

Das Django-Framework bringt einige Besonderheiten mit sich, die beim Implementieren 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 nutzen, ist dem *Superuser*[2] 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 Dekoratoren zur Verfügung. Soll eine bestimmte Seite nur von authentifizierten Benutzern besucht werden können, kann die Funktion mit einem Dekorator versehen werden:

[2] Superuser ist ein Benutzer, der alle Zugangsrechte im Framework erhält.

```
@login_required
```

Natürlich lassen sich Dekoratoren auch für andere Zwecke vor Funktionen platzieren. Unter anderem werden so die Views vor möglichen Angriffen, wie *Cross-Site-Scripting*[3], geschützt.

Durch den einfachen Aufbau ist es außerdem möglich, Dekoratoren selbst zu implementieren. Ein einfaches Beispiel wäre das Prüfen des an die Funktion übergebenen Parameter. Sollen nur positive Zahlen verarbeitet werden, so kann der Decorator alle anderen Eingaben abfangen.

2.1.2 Virtuelle Umgebung

Wird eine prototypische Anwendung gestartet, ist es notwendig, verschiedenste Module zu verwenden und zu testen. Die Versionen dieser Module spielen hierbei eine entscheidende Rolle, um Konflikte zu vermeiden (vgl. [Fou18]). Um diesem Problem vorzubeugen, wird eine virtuelle Umgebung implementiert. Sie besitzt einen eigenen Projektpfad, beinhaltet alle nötigen Pakete und Bibliotheken, und lässt sich nach dem Einrichten im Terminal starten. Abbildung 2.3. zeigt das Erstellen eines neuen Ordners, das Erstellen der virtuellen Umgebung und den Aktivierungsbefehl. Ist der Name des Environments in Klammern am Kommandozeilenanfang, bedeutet das, dass diese jetzt aktiv ist.

```
[Esthers-MBP:~ Esthi$ mkdir thesis-test
[Esthers-MBP:~ Esthi$ python3 -m venv test
[Esthers-MBP:~ Esthi$ source test/bin/activate
 (test) Esthers-MBP:~ Esthi$
```

Abbildung 2.3. Erstellen der virtuelle Umgebung im Terminal

Um die Pakete und Module kollisionsfrei zu installieren, ist es empfehlenswert, einen Package-Manager zu verwenden. Mit *pip*[4] (vgl. [Wei17, K. 23.1]) können diese installiert, geupdated und gelöscht werden. Außerdem kann der Manager Abhängigkeiten, wenn nötig, überschreiben und optimieren. Falls ein sich von der neuesten Version unterscheidendes Programm installiert werden soll, so ist dies ebenso möglich.

[3] Cross-Site-Scripting kann Webseiten verändern oder Passwörter abgreifen.
[4] pip ist das rekursive Akronym für pip Install Packages.

2.1.3 Sicherheit

Beim Implementieren einer Website ist das Absichern vor schädlichen Attacken heutzutage unabdingbar. Django aktiviert einige Funktionen zum Schutz bereits beim Projektstart automatisch. Dazu gehört das Abwehren von *Cross-Site-Scripting*, *SQL-Injektion*[5], *Clickjacking*[6] und die Sicherstellung der *Session-Security*[7].

Sollen die Formulare des Prototypen gegen *Cross-Site-Request-Forgery* geschützt werden, muss aktiv ein *Token* im Template gesetzt werden.

Ein solcher Angriff tritt auf, wenn über einen bösartigen Link, eine Formularschaltfläche oder einfach den eingebettetem JavaScript-Code die Daten im Server verändert werden sollen. Hierbei nutzt der Angreifer die Rechte eines eingeloggten Benutzers und kann somit Informationen im Backend verfälschen. Um dies zu verhindern, wird im Template des Prototypen zwischen den Form-Tags der *crsf-Token*[8] eingefügt. Der Token setzt einen Cookie mit einer verschlüsselten Zufallszahl. Das Gleiche passiert im Template, wo ein für den Benutzer nicht sichtbares Form-Feld die gleiche verschlüsselte Zufallszahl erhält. Beide Zahlen erhalten zudem einen *Salt*, einen generierten Zusatzwert, der das Entschlüsseln dieser um ein vielfaches erschwert. Wird ein Request gesendet, vergleicht eine von Django initialisierte *Middleware*[9] beide Zahlen. Stimmen diese nicht überein, hat also ein Dritter die Informationen manipuliert, wird der 403 HTTP-Standard-Statuscode gesendet, welcher besagt, dass der Server eine Anfrage erhalten hat, diese aber nicht erfüllen wird. Der `crsf-Token` greift nur, wenn der POST-Request innerhalb der eigenen Website gesendet wird und nicht über URLs, die außerhalb des Frameworks liegen (vgl. [Fou18a]).

2.2 Erweiterungen

Django bringt viele hilfreiche Erweiterungen mit sich, die mit einem Packagemanager einfach in die virtuelle Umgebung geladen werden können. Um das passende Add-on für ein Projekt zu finden, bietet die Plattform djangopackages.org alle Erweiterungen in übersichtlichen Tabellen mit Eigenschaften und Bewertungen an. Eine Vielzahl an Bibliotheken wurden für diese Arbeit getestet, aber wegen mangelnder Kompatibilität oder Funktionalität als nicht hilfreich erachtet und deshalb hier nicht weiter erwähnt. Die im Folgenden aufgeführten Bibliotheken sind im Prototyp zur

[5] SQL-Injection nutzt mangelnde Überprüfung von Metazeichen um Datenbankabfragen zu verändern.

[6] Clickjacking ist eine Überlagerung von Internetseiten um Klicks zu manipulieren.

[7] Session-Security setzen Cookies auch für Subdomäne beim Client.

[8] Token sind Komponente die eine Zugriffskontrolle von Benutzer druchführen können.

[9] Middleware ist ein Plug-in, dass Anfrage- und Antwortverarbeitung durchführt.

Anwendung gelangt.

2.2.1 Taggable-Manager

Um die Artikel besser priorisieren und die Informationsflut für Benutzer reduzieren zu können, wird im Prototyp mit Schlagwörtern gearbeitet. Django-taggit ist eine Erweiterung, die das Verwenden von Tags vereinfacht. Der darin enthaltene Taggable Manager setzt Django's Contenttype Framework ein, 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 Contenttyps 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 (vgl. [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*[10] notwendig, die die Entitäten zweier Tabellen direkt verlinkt, statt sie generisch zu verbinden. Weiterführend können Modelle in einem benutzerdefinierten Modell vereint werden, sodass der Zugriff darauf 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 (vgl. [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 mitzuteilen, dass nun eine neue Liste der Datenbank hinzugefügt werden muss, werden über die Kommandozeile Migrations-Befehle ausgeführt, die im Kapitel Datenmodellierung genauer beschrieben werden. Im Admin-Backend kann nun geprüft werden, ob das neue Feld in die Datenbank integriert wurde und neue Tags können in das Textfeld eingetragen werden. Der Parser verarbeitet jede Eingabe, die durch ein Komma oder ein Leerzeichen getrennt ist als einen Tag. Soll der Tag jedoch aus mehreren Wörtern bestehen, so müssen diese mit Anführungszeichen umfasst werden. Standardmäßig unterscheidet der Taggable Manager zwischen

[10] Eine Lookup-Tabelle speichert Daten statisch im Verhältnis zueinander.

Groß- und Kleinschreibung, Tags sind also case sensitive.

2.2.2 Hilfsbibliotheken

Weitere Add-ons werden geladen, um kleinere Funktionen der Website einfach umsetzen zu können.

Zu diesen gehört `django-taggit-templatetags`, welches durch die Einbindung im HTML-File die Tags der Applikation als Liste ausgibt. Außerdem lassen sich die eingepflegten Tags als Cloud visualisieren. Kommen bestimmte Schlagwörter öfter vor als andere, so werden sie entsprechend größer dargestellt.

`Django-hitcount` dient zum zählen der Besucher einer Seite (vgl. [Tim15]). Es lässt sich auf drei verschiedene Arten in die Applikation einbinden. Der schnellste Weg ist die Darstellung der Besuche mit Hilfe eines Template Tags im HTML-File. Möchte man die Anzeige aber individueller gestalten, so kann durch das integrieren der `HitCountDetailView` in `views.py` die Variable `hitcount` verwenden und im Frontend ausgeben. Eine weitere Möglichkeit ist das Erweitern oder neu Erstellen eines Models im Django Backend. Von dort kann auf das neue Feld im Django-Admin-Backend zugegriffen werden, ebenso wie von der View und vom Template. Die im Add-on integrierten Einstellungen, die in der `settings.py` ergänzt werden müssen, ermöglichen unter anderem das Begrenzen der Lebensdauer des Zählers, bevor dieser zurück gesetzt wird.

Zum Versenden und Verwalten von E-Mails in Django eignet sich `django-post-office` (vgl. [Ong18]). Nach der Installation kann im Admin-Backend ein E-Mail-Template angelegt, Anhänge verwaltet und das Versenden dieser im Log überprüft werden. Es ist möglich die Benachrichtigungen mithilfe eines eines Shell-Skript des Frameworks `Cron`[11] asynchron zu versenden. Der Inhalt kann Text oder HTML-basiert sein und in mehreren Sprachen hinterlegt werden.

2.3 Bootstrap

Um die Usability des Prototyps zu erhöhen, wird das Framework Bootstrap eingebunden.

Eine umfangreiche Website einheitlich zu gestalten ist oft sehr komplex und zeitaufwendig. Die Entwickler von Twitter haben deshalb an einem neuen Verwaltungswerkzeug gearbeitet, das mehrere Bibliotheken zusammenführen sollte (vgl. [Ott11]). Die Open-Source-Bibliothek, die auf GitHub abrufbar ist, wird seitdem

[11] Cron dient zur zeitbasierten Ausführung von definierten Aufgaben.

von vielen Programmierern weiterentwickelt und ist somit stark gewachsen. Version 2.0 verfügt über die Fähigkeit, Websites *responsiv*[12] auf verschiedenste mobile Endgeräte anzupassen (vgl. [Ott12]).

Das Bootstrap-Paket beinhaltet vorgefertigte Cascading Stylesheets, kurz CSS, die Farben, Schriftarten und viele weitere Stildefinitionen implizieren. 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 im head-tag einer HTML-Datei (vgl. Abbildung 2.4.) einfach eingebunden werden. Das bedeutet, dass *Media-Queries*[13] oder ähnliche Methoden nicht mehr nötig sind - nicht nur um eine Website mobilfähig zu machen, sondern auch kompatibel für die verschiedensten Browser (vgl. [Boo12]).

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

Abbildung 2.4. 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, angewendet. Dies ist jedoch nur ein sehr kleiner Teil, den die Bibliothek zur Verfügung stellt. Bootstrap lässt sich umfangreicher nutzen, indem Stildefinitionen mit Klassen oder ID's in diverse HTML-Tags eingetragen werden (vlg. Abbildung 2.5.).

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

Abbildung 2.5. Bootstrap-Klassen in HTML-Tag

Möchte man bestimmte gestalterische Eigenschaften von Bootstrap überschreiben, muss eine eigens verfasste CSS-Datei nach der Verlinkung von Bootstrap in die Website eingebunden werden. Der Parser liest die Datei von oben nach unten, links

[12] Responsive Webseiten sind auf allen Endgeräten angepasst darstellbar.
[13] Media-Qeries setzen statische Umbrüche um Layouts verschiedener Endgeräte anzupassen.

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.

Kapitel 3

Prototyp

Um die Forschungsfrage zu prüfen, wird in dieser Arbeit die Methode des Prototypings genutzt. Der Prototyp dient zum experimentellen Arbeiten und sichert eine strukturell fundierte Umsetzung des Endprodukts. Der Fokus liegt dabei zunächst auf der Funktionalität der Anwendung. Prototyping wird als bevorzugte Methode gewählt, um schnell ein Ergebnis zu erzielen (vgl. [Abr16]). Zudem soll darauf aufbauend ein Produkt realisiert werden, das als Erweiterung in das Netzwerk der Hochschule integriert wird.

3.1 Forschungsdesign

Dieses Kapitel veranschaulicht eine kurze Übersicht der Vorgehensweise und den Leitfaden, an den sich die Implementierung des Prototyps anlehnt (vgl. Abbildung 3.1.). Zu Beginn der Arbeit wird das sich aus der Forschungsfrage ergebenden Problem analysiert und es werden alle wichtigen Anforderungen erfasst. Dies bildet die Basis für alle weitern notwendigen Schritte, um am Ende eine sinnvolle Lösung bereitstellen zu können. Die Recherche dient der Sammlung aller notwendigen Werkzeuge und gibt einen Überblick über verschiedene Hilfsbibliotheken. Die Implementierung der Applikation kann nun auf Basis der Recherche durchgeführt werden. Dazu gehört das Testen verschiedener Bibliotheken und Erweiterungen, um das bestmögliche Ergebnis zu eruieren. Abschließend wird die Funktionalität des Prototypen getestet und evaluiert, ob die Forschungsfrage ausreichend beantwortet wird. Handlungsempfehlungen und mögliche Funktionen zum Erweitern finalisieren die Arbeit.

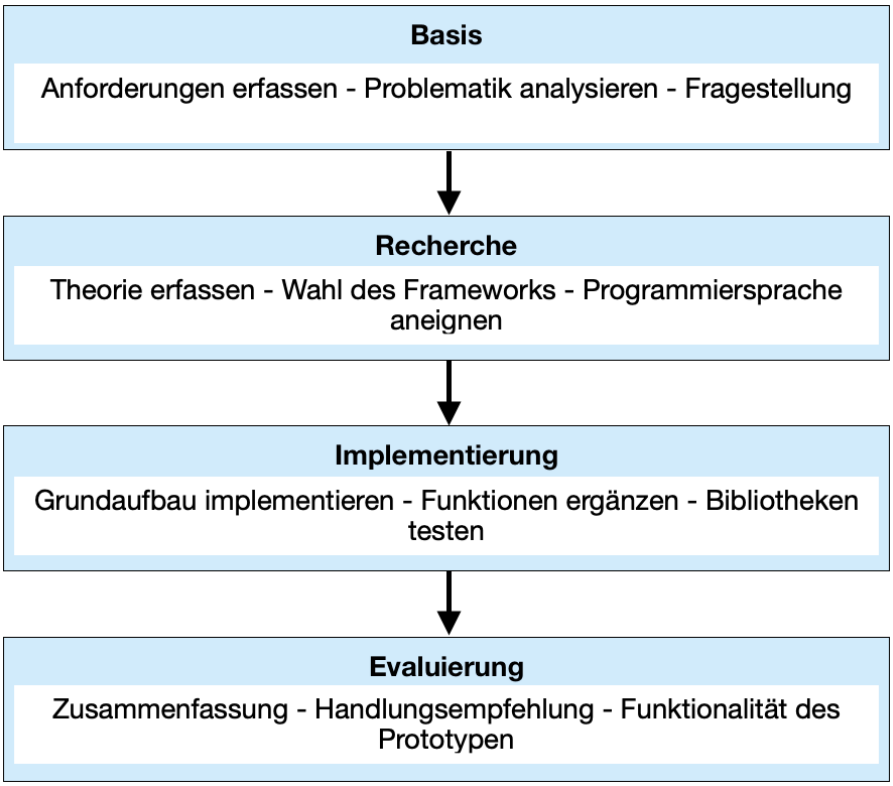

Abbildung 3.1. Forschungsdesign

3.2 Organisation

Um einen Einblick in den Aufbau eines Django-Projektes zu erlangen wird die Struktur im Folgenden genauer beschrieben. Die unterste Projektebene wird durch `manage.py` gebildet. Sie wird unter anderem genutzt, um den lokalen Server starten zu können. In der Ebene darüber findet sich im Ordner `mysite` die Datei `settings.py`. Hier werden die allgemeinen Einstellungen der Website vorgenommen, wie zum Beispiel das Integrieren der Erweiterungen und der Pfad zu den hinterlegten Templates. Außerdem ist die `urls.py` dort zu finden, deren Funktion bereits im Kapitel Django erläutert wurde. Im Ordner `thisisenv` sind alle Bibliotheken und Add-ons der virtuellen Umgebung hinterlegt. Der relevanteste Teil dieser Arbeit liegt im Ordner `application`. Hier sind die Datenbank-Migrationen, die Static-Files wie bootstrap und css, und alle Templates abgelegt. Zudem befindet sich hier die Logik des Prototypen, auf die im Kapitel `Funktionen` weiter eingegangen wird.

3.2.1 Datenmodellierung

Die Struktur der bereits bestehenden Datenbank im Django-Framework und die Erweiterungen desselben werden genauer erläutert. Zunächst wird auf die Ergänzung des bestehenden `UserModel` eingegangen, danach veranschaulicht der zweite Abschnitt das `PostModel` und abschließend werden die Zusammenhänge der Modelle dargestellt.

Alle Modelle werden als Django-Modelle deklariert um beim Kompilieren des Codes dem Compiler mitzuteilen, dass eine Integration stattfinden muss (vgl. [Dja18a]). Mit der Eingabe

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

werden die neun Tabellen der Modelle erstellt. Zur letztendlichen Migration wird

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

ausgeführt.

UserModel:
Hier ist das Authentifizierungssystem von Django mit einem `UserModel` bereits angelegt. Dieses muss für den Prototyp um das Feld `tags` erweitert werden, sodass ein Benutzer folgende Felder besitzt (vgl. [Fou18a]):

- username, fist_name, last_name, email, groups, user_permissions, is_staff, is_active, is_superuser, last_login, date_joined, tags

In models.py ist der `CustomUser` dafür verantwortlich, das neue Feld mit dem `Default-User` zu verknüpfen. Durch das `OneToOneField` (siehe Abbildung 3.2.) wird die Verbindung zum schon bestehenden Modell hergestellt. `OneToOne` bildet eine einzigartige Zuordnung von zwei Objekten, sodass der Rückgabewert eindeutig ist. Das heißt, dass hier keine rekursiven, auf sich selbst verlinkende oder `lazy` Beziehungen möglich sind, um Konflikte bei der Authentifizierung zu vermeiden. Dies ist die übliche Vorgehensweise, um mit einem Primärschlüssel das Default-Model zu erweitern (vgl. [Fou18a]).

PostModel:
Das `PostModel` beschreibt alle Felder, die ein Artikel enthalten kann. Basierend auf der Blog-Lösung von Djangogirls.com (vgl. [Dja18b]) gehören dazu folgende:

```
class CustomUser(models.Model):
    user = models.OneToOneField(User, null=True, on_delete=models.CASCADE)
    tags = TaggableManager(blank=True)
```

Abbildung 3.2. CustomUserModel in models.py

- author, title, text, created_date, published_date, tags

Der Autor ist durch einen `ForeignKey` mit dem `UserModel` verbunden. Diese `ManyToOne` Verbindung reicht hier aus, um einem Post einem Autor, also dem eingeloggten User, zuzuweisen. `Title` ist ein `CharField` und wird mit einer Zeichenbegrenzung festgelegt. Der Text hingegen kann eine beliebige Menge an Zeichen enthalten und wird deshalb als `TextField` deklariert. Erstellungsdatum und Publikation sind `DateTimeFields`. Ersteres muss vom Ersteller angegeben werden, zweiteres kann zunächst durch die Zusatzangabe `null=True` offen gelassen werden. Ein weiteres Feld wird hinzugefügt, um Artikeln unabhängig von Usern Tags zuordnen zu können.

Gesamtmodellierung:

Die Abbildung 3.3. zeigt die Modellierung der Tabelle `User` und `Post`. Außerdem verdeutlicht es die Erweiterung des User-Modells von Django mit dem in der Applikation angelegten `CustomUser`. Die im User vorkommenden *booleschen Felder* werden im Kapitel „Berechtigungen der User"genauer erörtert.

3.2.2 Berechtigungen der User

Im Allgemeinen verwendet man Berechtigungen, um Benutzern Zugang zu bestimmten Resourcen in einem Netzwerk einzuräumen. Außerdem bestimmt eine Staffelung die Art des Zugangs, ob der User die Resourcen nur lesen, verändern oder löschen darf (vgl. [Com18]). Die Rechte werden meist einzelnen Individuen oder einer Gruppe zugeordnet.

Das gestaffelte Berechtigungsmanagement ist im Prototyp notwendig, um den Umgang mit Informationen so sicher wie möglich zu gestalten und um die Nachhaltigkeit dieser zu bewahren. Des Weiteren soll der Prototyp als Vorlage für die Erweiterung der Hochschulwebsite dienen und daher ist eine, zur Hochschule ähnliche Verteilung der Zugangsberechtigungen sinnvoll.

Studenten sollen zunächst Informationen weder einpflegen noch editieren dürfen. Die einzigen Änderungen, die sie vornehmen können, sind auf ihre eigene Datenbank

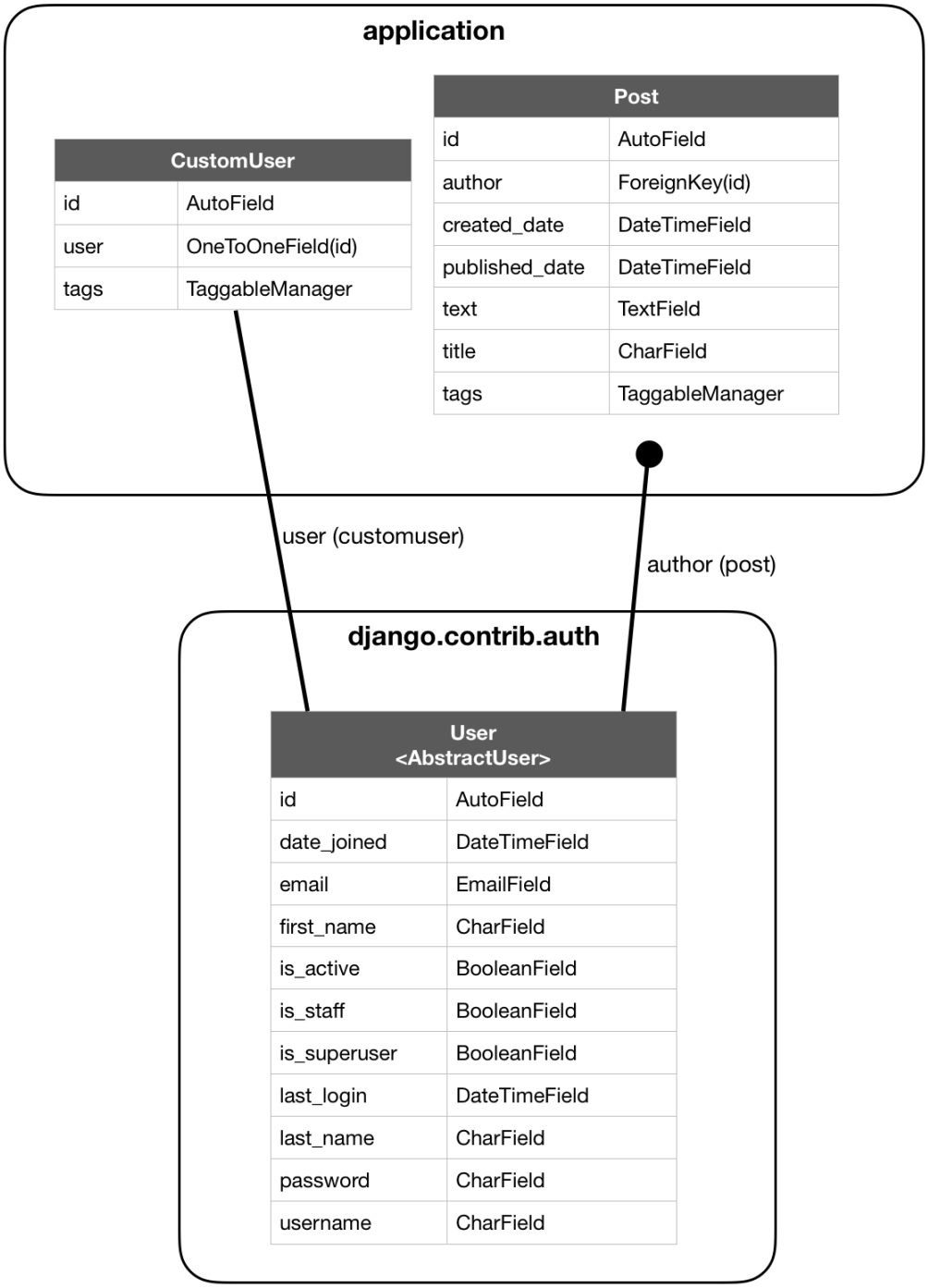

Abbildung 3.3. Datenmodellierung von `User` und `Post`

fokussiert. Dazu gehört das Hinzufügen von Tags, durch die damit verlinkten Beiträge das persönliche Dashboard befüllt wird. Dies soll verhindern, dass Informationen zu leichtfertig geändert oder gelöscht werden.

Dozenten und Angestellte der Hochschule sind dazu berechtigt, Posts zu erstellen, zu editieren und wieder zu löschen. Zudem können sie, wie Studenten, Tags

abonnieren und das persönliche Dashboard gestalten. Das Einloggen in die Administratoroberfläche kann durchgeführt werden, jedoch sind der Gruppe noch keinerlei Rechte zugewiesen. Möchte man das ändern, kann man das von Django bereitgestellte Feld `User Permissions` im Admin-Backend unter `Users` finden, und dem Namen der Person die gewünschte Berechtigung zuteilen. Diese sind von Django vorgegeben und betreffen alle vorhandenen Modelle der Applikation. Durch das Setzen des booleschen Wert `is_staff` auf `True` beim Erstellen der Benutzer ist es möglicha im Code der Applikation Abfragen durchzuführen. Dadurch lassen sich bestimmte Views an die eingeloggte Personengruppe anpassen. So ist unter Anderem das Menü der Dozenten und Angestellten etwas umfangreicherer als das der Studierenden (vgl. Abbildung 3.4.).

Abbildung 3.4. Menü für eingeloggte Benutzer mit Adminrechte

3.3 Funktionen

Um die wichtigsten Funktionen des Prototypen festlegen zu können, werden User Stories erstellt (vgl. Abbildung 3.5.). Diese bestehen aus kurzen Sätzen und beschreiben aus Sicht des Nutzers das Verwenden einer Funktion. Die Priorisierung bezieht sich hierbei auf die Relevanz der Funktion, wobei die Funktionen mit einem rotem Punkt sehr wichtig für den Prototypen sind, Orangene wichtige Funktionen sind, aber nicht unbedingt notwendig, und Grüne kaum Relevanz haben.

3.3.1 Verwaltung der Funktionen

Das Verwalten der Artikel soll von berechtigten Nutzern im Frontend stattfinden, während die prozessuale Logik im Code-Backend realisiert ist. Der Vorgang des Erstellens, des Löschens und des Editierens der einzelnen Einträge wird im Folgenden konkretisiert.

Einen neuen Artikel erstellen:

Das `+` in der Menüleiste leitet den Benutzer zu einer Unterseite. Hier können alle Felder befüllt werden, die im `PostForm`-Formular in der Datei `forms.py` fest-

Wer	Was	Warum	Prio.
Student	möchte nach Einträgen mit dem Tag „Studienbüro“ suchen	um alle Neuigkeiten über das Studienbüro zu sehen	🟠
Student	möchte den Tag „Alumni“ abonnieren	um auf seinem Dashbaord die Posts über Alumni mit aufzunehmen	🔴
Dozent	möchte einen neuen Post anlegen	um die Information zu Veröffentlichen	🔴
Dozent	möchte einen vorhandenen Post ändern	um weiter Tags hinzu zufügen	🟠
Dozent	möchte einen Post löschen	da die Informationen veraltet sind	🔴
Dozent	möchte einen Post anlegen aber erst später veröffentlichen	um mehrere Posts simultan veröffentlichen zu können	🟢
Admin	möchte ins Back-end	um Dozenten neue Rechte zuzuweisen	🟡

Legende der Prioritäten	
🔴	Wichtige Funktionen des Prototypen.
🟠	Wichtige Funktionen des Prototypen aber nicht notwendig.
🟢	Kaum Relevante Funktionen des Protoypen.

Abbildung 3.5. User Stories

gelegt wurden. Dazu gehören der Titel und der Text, die als Pflichtfelder gelten. Das Feld `Tags` muss ebenfalls mindestens einen Wert enthalten, um die Validierung der Eingaben sichern zu können. Eine Ausnahme bildet das Datum der Veröffentlichung. Bleibt das Feld leer, so wird der Beitrag automatisch der Liste der Entwürfe beigefügt.

Speichert der Benutzer den Artikel, so werden im Backend die Daten wie folgt verarbeitet. In der View `post_new` wird zunächst geprüft, ob alle Eingaben valide sind. Falls dies der Fall ist, wird der jeweilige Beitrag als Objekt zurückgegeben, jedoch durch das optionale Keywort `commit=false` noch nicht in der Datenbank gespeichert. Das ist notwendig, um dem Objekt spezifische Informationen mitzugeben. In diesem Kontext wird der aktuell eingeloggte User als Autor hinterlegt. Jedoch birgt die Vorgehensweise eine Problematik im Speichervorgang einer `ManyToMany` Relation zwischen zwei Modellen. Da Informationen nur auf ein bereits in der Datenbank bestehendes Objekt gesichert werden können, ist dies zunächst nicht möglich (vgl. [Fou18b]). Im Prototyp nutzt das `PostModel` die `ManyToMany` Konnektivität mit dem Modell des `TaggabelManagers`. Um die Eingabe des Tag-Felds trotzdem im neuen Artikel speichern zu können, wird zunächst das Objekt gespeichert, um danach explizit das von Django zur Verfügung gestellte `form.save_m2m()` aufrufen zu können. Der Befehl zwingt die Daten der `ManyToMany` Relation zu speichern.

Die eindeutige Zuordnung der Eingabe im Front-end zur Verarbeitung der Artikel im Back-end ist mit einem `Primary Key` realisiert. Das `PostModel` bekommt beim Anlegen keinen Schlüssel zu einem Feld zugewiesen. Django erstellt ihn automatisiert beim Speichern der Tabelle als `AutoField` im Feld `Id` und identifiziert die `Id` automatisch bei jedem neu Erstellen eines Objekts. Somit sind alle Objekte eindeutig zuordenbar und können mit dem Kommando `post.pk` jederzeit abgefragt werden.

Einen bereits vorhandenen Artikel löschen:

In der Detailansicht eines Artikels ist es möglich, denselben zu entfernen. Die View `post_remove` selektiert über den im Template mitgegebenen `Primary Key` das Objekt und speichert es in der Variable `post`. Der Post wird gelöscht mit dem Befehl `post.remove()` und eine Umleitung am Ende der View-Definition schickt den Benutzer auf die Seite der Artikelliste. Hier wird eine zuvor in der View generierte Nachricht visualisiert, sodass der Benutzer sicher sein kann, dass der Vorgang abgeschlossen ist.

Einen bereits vorhandenen Artikel bearbeiten:

Ähnlich wie beim Löschen eines Artikel, kann man diesen in der Detailansicht bearbeiten. Dazu wird in der View über den `Primary Key` der Artikel einer Variable `post` zugeordnet. Die bedingte Anweisung rendert zunächst die `PostForm`, mit dem bereits eingepflegten Inhalt durch eine GET-Abfrage (vgl. Abbildung 3.6.).

Veranlasst der Benutzer die Speicherung des Artikels im Front-end, wird die bedingte Abfrage der Abbildung 3.7. in Zeile 91 erfüllt. Die POST-Abfrage ist hier notwendig, da Django nur so Daten in der Datenbank verändert. Eine Begründung hierfür ist die Art der Übertragung der Daten an den Server. `POST-Requests` bündeln alle Daten, verschlüsseln sie und senden sie dann an der Server (vgl. [Fou18c]). Dadurch ist der Vorgang einfacher kontrollierbar und mit einem `csrf-Token` im Template ebenfalls gegen Cross-Site-Request-Fälschung abgesichert. Die weitere Vorgehensweise der Funktion ist identisch zum bereits dargelegten neu erstellen eines Artikels und muss nicht weiter beschrieben werden.

3.3.2 Artikel abonnieren

Das Abonnieren bestimmter Themengebiete ist eine der wichtigsten Funktionen im Prototyp, um die eingepflegten Informationen zielgerichtet anzeigen zu können.

Unter Berücksichtigung aller Vor- und Nachteile wird ein Tag-Modell zur Um-

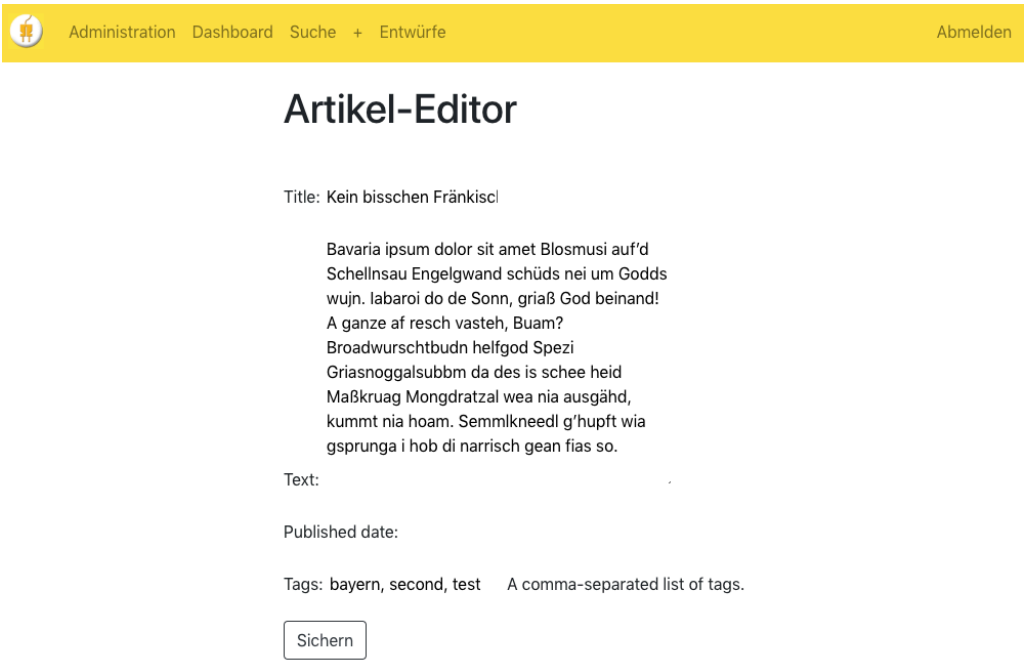

Abbildung 3.6. Prototyp Artikel-Editor.

```
87  @login_required
88  @staff_member_required
89  def post_edit(request, pk):
90      post = get_object_or_404(Post, pk=pk)
91      if request.method == "POST":
92          form = PostForm(request.POST, instance=post)
93          if form.is_valid():
94              post = form.save(commit=False)
95              post.author = request.user
96              post.save()
97              form.save_m2m()
98              return redirect('post_detail', pk=post.pk)
99      else:
100         form = PostForm(instance=post)
101     return render(request, 'post_edit.html', {'form': form})
```

Abbildung 3.7. Funktion post_edit, Auszug aus views.py.

setzung gewählt. Wie bereits in der Datenmodellierung angedeutet (vgl. Datenmodellierung), besitzt jeder Artikel beschreibende Tags. Hierbei handelt es sich um kurze stichwortartige Beschreibungen, die den Artikel so gut wie möglich charakterisieren. Abhängig vom Umfang des Blogsystems, sollte die Anzahl der Tags immer in einem gewissen Rahmen vorhanden sein. Das bedeutet zum einen, dass Ersteller von Artikeln immer die gleiche Menge an Schlagwörtern verwenden, wobei geringe

Abweichungen möglich sind (vgl. [Gmb18]). Hat das System bereits einen größeren Umfang angenommen, sollten zum anderen keine neuen Tags erstellt werden, um die Übersicht für Autoren und Leser zu bewahren.

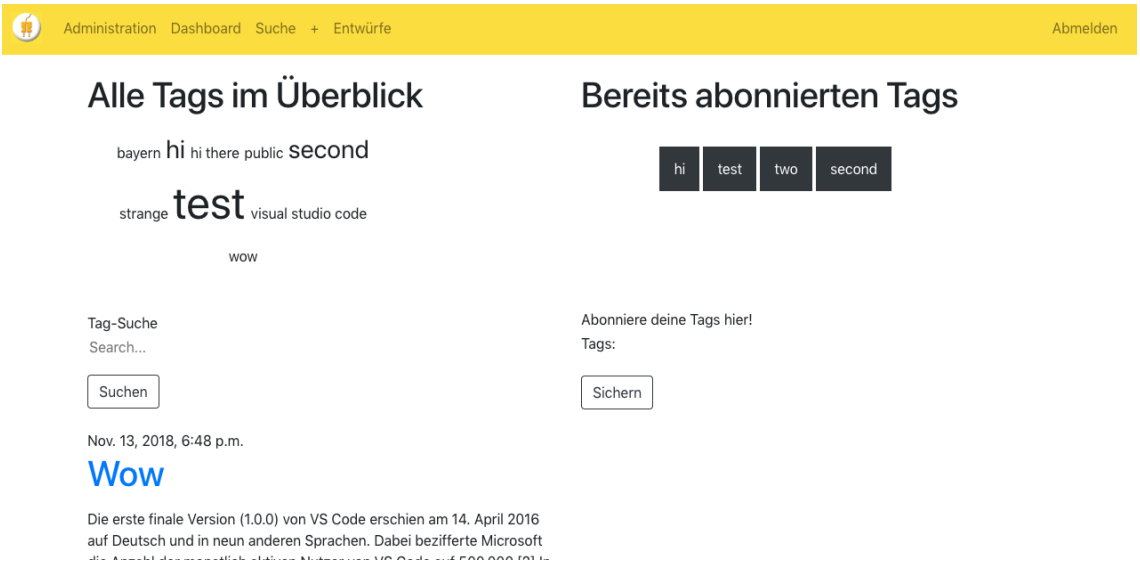

Abbildung 3.8. Prototyp Suche- und Abonnier-Seite

Im Prototyp findet man die Abonnement-Funktion unter dem Menüpunkt `Suche`. Hier erscheint ein zweigeteiltes Layout, welches auf der rechten Seite alle bereits abonnierten Tags auflistet und darunter die Eingabe eines neuen Tags ermöglicht. Um den Benutzer alle bereits existierenden Tags offen zu legen, befindet sich auf der linken Seite des Layouts eine *Tag-Cloud* [1] , die alle Schlagwörter darstellt (vgl. Abbildung 3.8.).

Die Eingabe des zu abonnierenden Tags wird durch ein Formular realisiert. Dieses ist in der `forms.py` Datei konfiguriert und enthält nur ein Eingabefeld. Der Ablauf verläuft gleichartig zum oben dargestellten Erstellen eines Artikels, wird allerdings genauer beschrieben, um die Struktur des `Taggable Managers` zu verdeutlichen.

Gibt der Benutzer einen Tag ein und sendet durch Betätigen des Sichern-Buttons den `Request`, wird dieser in der `views.py` verarbeitet. In Zeile 159 der Abbildung 3.9. wird der eingeloggte Benutzer der Variable `user_instance` übergeben. Beim Erstellen der `Model-Instanz` (vgl. Abbildung 3.9., Zeile 161) wird `user_instance` der Unbekannten `form` zugeteilt, um die Tag-Eingabe im richtigen User-Objekt integrieren zu können. Nach der Abfrage der Formvalidität wird ein neues Objekt angelegt (vgl. Abbildung 3.9., Zeile 163) und ebenfalls dem aktuellen Benutzer zugeordnet.

[1] Tag-Cloud ist eine Visualisierung eines Schlagwortverzeichnisses.

```
151  def search_add(request):
152      posts = Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
153      if request.method == 'GET':
154          search_query = request.GET.get('search_box', None)
155          posts = posts.filter(tags__name__in=[search_query])
156      u = User.objects.get(username=request.user)
157      if u:
158          tagsuser = Tag.objects.filter(customuser__user = u)
159      user_instance = get_object_or_404(CustomUser, user=request.user)
160      if request.method == "POST":
161          form = NewTagForm(request.POST, instance=user_instance)
162          if form.is_valid():
163              obj = form.save(commit=False)
164              obj.user = request.user
165              tag_names = [tag.name for tag in Tag.objects.all()]
166              m_tags = form.cleaned_data['tags']
167              m_tags = ' '.join(str(m_tags) for m_tags in m_tags)
168              if m_tags in tag_names:
169                  obj.tags.add(m_tags)
170                  obj.save()
171                  messages.info(request, 'Der Tag "' + m_tags + '" wurde gespeichert')
172                  return redirect('/search/')
173              else:
174                  messages.info(request, 'Sorry !! Den Tag den du suchst gibt es leider nicht!')
175      else:
176          form = NewTagForm()
177      return render(request, 'search_add.html', locals())
```

Abbildung 3.9. Funktion search_add, Auszug aus views.py.

Die Eingabe der `form` wird in einem `Array` zwischengespeichert und mit dem Attribut `cleaned_data` in ein für Python kompatiblen Datentyp gecastet. Um prüfen zu können, ob die Eingaben der Form tatsächlich im `Tag-Model` enthalten sind, wird diese nochmals in einen String umgewandelt und mit den bereits existierenden Tags abgeglichen (vgl. Abbildung 3.9., Zeile 168). Wird die Bedingung erfüllt, speichert die Funktion die Tags. In beiden möglichen Fällen wird der Benutzer benachrichtigt, ob der Vorgang erfolgreich oder die Eingabe nicht valide war.

Nun werden auf dem Dashboard Artikel der neu hinzugefügten Tags angezeigt (vgl Abbildung 3.10.).

3.3.3 Filtern von Artikeln

Zur Unterstützung der Nutzbarkeit des Prototypen ist es wichtig, dass User intuitiv nach Tags suchen und sie selektieren können. Hierfür werden verschiedene Möglichkeiten zur Verfügung gestellt, die die Usability der Website verbessern sollen.

Im persönlichen Newsfeed des Dashboards sind die zu den Artikeln zugewiesenen Schlagwörter jeweils mit Verlinkungen versehen. Möchte ein Benutzer weitere

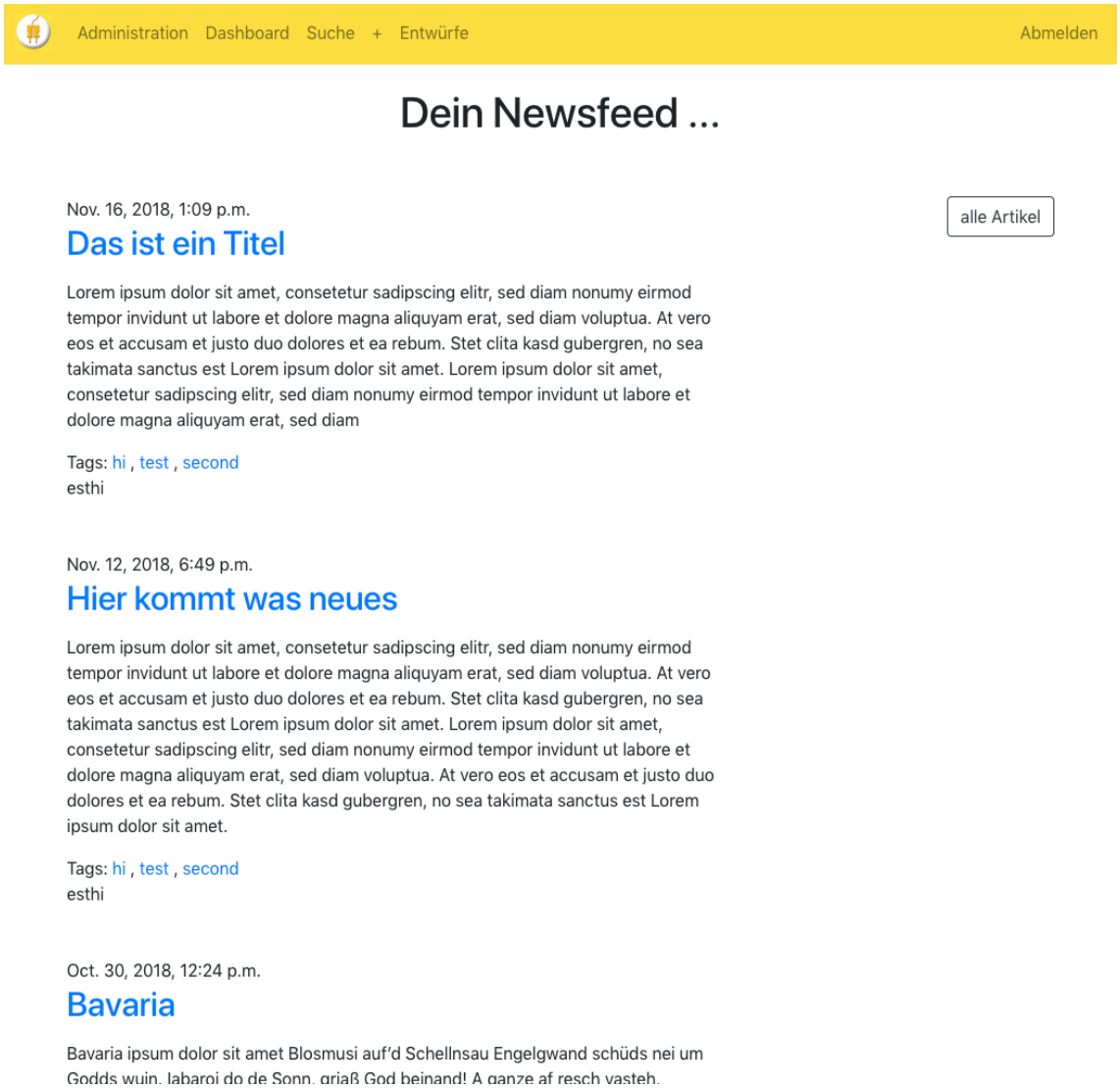

Abbildung 3.10. Prototyp Newsfeed Seite

Artikel zu einem bestimmten Thema lesen, so muss er lediglich auf den entsprechenden Tag klicken und erhält eine Liste aller Beiträge, die den selektierten Tag enthalten. Hierfür wird keine eigene `View` benötigt, denn das Erstellen von Listen mit unterschiedlichem Inhalt kann ebenso über sich unterscheidende Urls realisiert werden. Im Template `post_list` wird beim Klicken auf einen Tag der *Slug*[2] desselben mitgegeben. Außerdem wird nun die Url `post_list_by_tag` aufgerufen, die auf eine neue Seite verweist. Die View `post_list` rendert bei Anfragen mit *Slug* die passende Liste, bei Anfragen ohne werden alle vorhandenen Artikel geladen.

Der Prototyp bietet zudem die Möglichkeit direkt nach Tags zu suchen. Unter dem Menüpunkt `Suche` auf der linken Seite des geteilten Layouts befindet sich ei-

[2] Der Slug dient im Taggable-Manager als eindeutig zuweisbarer Identifikator.

ne *Tag-Cloud*. Darunter ist das Suchfeld, welches durch ein Formular im Template realisiert wird. Die Eingabe fragt jedoch nur Tags aus der Datenbank ab, die Daten werden nicht verändert, da hier lediglich ein `GET-Request` gesendet wird. Mit der Funktion `filter` und dem von der taggit-Bibliothek zur Verfügung gestellten *Lookup* [3] `tags__name__in` können alle Artikel mit dem jeweils enthaltenen Tag dem Template übergeben werden.

[3] Lookup ist eine Art Funktion, die den Ort des darauffolgend Feldes ausgibt.

Kapitel 4

Evaluation

Ziel dieses Kapitels ist es, zu evaluieren, ob der Prototyp dieser Arbeit die folgende Forschungsfrage beantworten kann:

„Kann die E-Mail-Flut der Technischen Hochschule mit Hilfe einer Social Media Plattform gedrosselt und die Nachhaltigkeit der Informationen gewährleistet werden?"

Die Durchführung des Beweises ist realisierbar, indem der E-Mail-Verkehr beispielhaft anhand eines Probanden der Technische Hochschule Nürnberg dargestellt wird. Anschließend wird das Ergebnis veranschaulicht.

Um den Umfang der E-Mail-Flut einordnen zu können, wird das Postfach des Studierenden genauer analysiert. Hierbei handelt es sich um den E-Mail-Verkehr während des Sommersemesters 2018, welches vom 15.03 bis 30.09 andauert. Da die Relevanz der Informationen von den persönlichen Interessen und Aktivitäten des Probanden abhängig ist, kommen diesen Eigenschaften besondere Gewichtung zu.

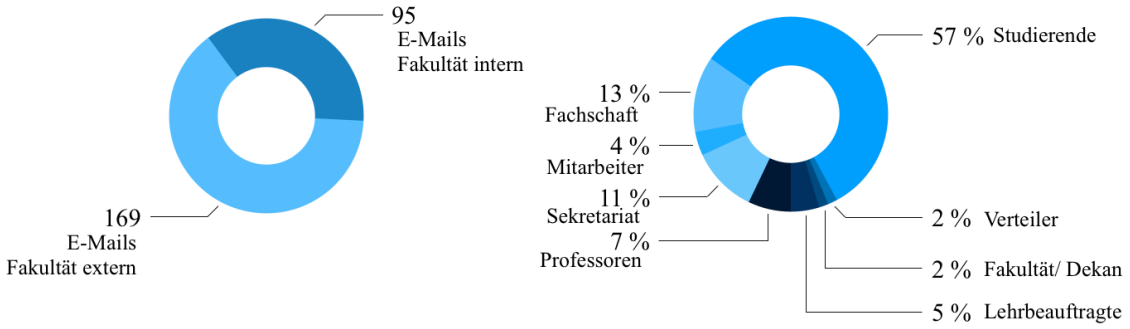

Abbildung 4.1. Vergleich der hochschulinternen E-Mails.

Während des Semesters trafen in Summe 264 Nachrichten im Postfach des Probanden ein. Darunter sind 95 innerhalb der Fakultät befördert worden. Abbildung

4.1. verdeutlicht auf der linken Seite das Verhältnis zwischen den Nachrichten hochschulweit und innerhalb der Fakultät. Diese wurden von diversen Verteilern an den Probanden gesendet. Auf der rechten Seite der Abbildung 4.1. ist eine Übersicht der genutzten Verteiler dargestellt. Dabei wird verdeutlicht, dass über die Mailingliste der Studierenden mehr als die Hälfte aller Mitteilungen versendet wird.

Sortiert der Proband nun den Posteingang nach relevanten Informationen, so zeigt sich folgendes Ergebnis:

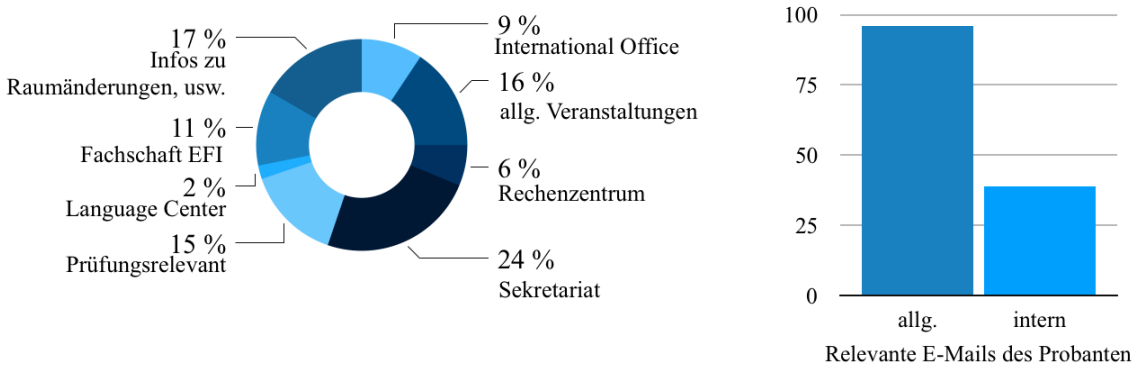

Abbildung 4.2. Details der relevanten E-Mails des Probanden.

Das Balkendiagramm der Abbildung 4.2. zeigt, wie viele Informationen der Gesamtanzahl von E-Mails bedeutsam für den Probanden sind. Detaillierter wird gezeigt, wie viele hiervon innerhalb der Fakultät von Interesse sind. Fokussiert man die Themenübersicht wird deutlich, dass allgemeine Benachrichtigungen, wie Termine von Veranstaltungen, prüfungsrelevante Neuigkeiten oder Updates zu den Systemen der Hochschule den Interessenschwerpunkt bilden. Spezifischere Informationen, wie die des Language Centers, des International Office oder der Fachschaft EFI nehmen zwar einen geringeren Anteil ein, sind für den Probanden aber nicht vernachlässigbar.

Werden die eingehenden E-Mails betrachtet, die der Studierende als nicht relevant aussortiert hat, lassen sich bereits eindeutige Tendenzen erkennen. Wie bereits in Abbildung 4.1. erkennbar ist, sind 169 Informationen, also 64 Prozent irrelevant. Extrahiert man davon die fakulätsinternen Benachrichtigungen, so ergibt sich die Anzahl 53. Prozentual lässt sich daraus berechnen, dass etwa 30 Prozent der überflüssigen E-Mails direkt von der EFI-Fakultät ausgehen.

Bei Sondierung der Detailansicht auf der linken Seite der Abbildung 4.3., ist zu erkennen, dass meist die sehr spezifischen Informationen über Vorlesungen, Interessen oder Freizeitaktivitäten vom Probanden aussortiert werden. Hierbei lässt

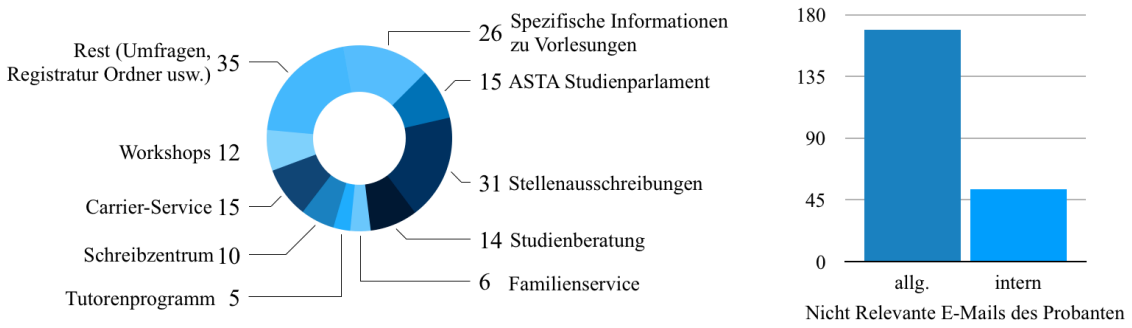

Abbildung 4.3. Details der irrelevanten E-Mails des Probanden.

sich erschließen, wie symptomatische Informationen trotz fehlender Relevanz, das Postfach überfluten.

Die beispielhafte Kalkulation der im Postfach des Probanden befindlichen E-Mails wird nun mit dem durch die Implementierung des Prototyps entstehenden Ergebnis verglichen. Die Aufgabe der entwickelten Anwendung ist es, die Informationen statt per E-Mail, über eine Social Media Plattform zu publizieren. Dennoch sollen die Studierenden und Lehrenden als regelmäßige Erinnerung eine zusammenfassende Benachrichtigung erhalten. Stellt man die Anzahl, der über das Semester verteilten eintreffenden E-Mails des Probanden, den wöchentlichen Mitteilungen gegenüber, so ergibt sich folgendes Diagramm:

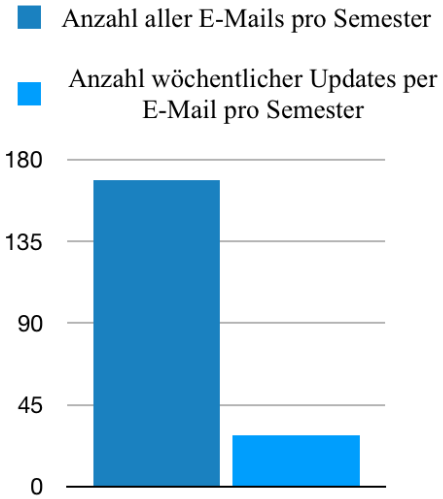

Abbildung 4.4. Vergleich der Anzahl von eintreffenden E-Mails zwischen aktueller Situation und unter Verwendung des Prototyps.

Abbildung 4.4. zeigt deutlich, dass durch den Einsatz des Prototyps die Anzahl der Informationen im Postfach stark reduziert werden können. Der Proband würde

über 80 Prozent seines Speichervolumens im System einsparen. Dadurch bestätigt sich die Hypothese: Die E-Mail-Flut der Hochschule wird durch den Einsatz einer Weberweiterung reduziert.

4.0.1 Ergebnis

Werden alle Auswertungen der Evaluation zusammengefasst und betrachtet, so ist deutlich zu sehen, dass Benachrichtigungen der Hochschule zu ausgedehnt verteilt werden. Fakultätsübergreifende Themengebiete sind häufig über umfangreiche Verteiler an Einzelpersonen weitergegeben worden und erzeugen dabei eine schwer administrierbare Menge.

Der Fokus dieser Arbeit liegt auf dem reduzieren der E-Mail-Flut innerhalb der EFI-Fakultät. Wird der Prototyp auf der Hochschul-Website eingebunden, so kann die Problematik im Idealfall auf ein Kleinstes reduziert werden. In Abbildung 4.5. ist das Verhältnis zwischen irrelevanten und relevanten Informationen der Fakultät visualisiert. Hier wird nochmal deutlich, dass über die Hälfte der Nachrichten keinerlei Bedeutsamkeit für den Probanden haben. Aufgrund dessen, lassen sich folgende Erkenntnisse festhalten. Die Website-Erweiterung vermeidet das Eintreffen der unwichtigen E-Mails und informiert Studierende und Angestellte über alle wichtigen Benachrichtigungen. Somit lässt sich der eintreffende Verkehr bereits um 35 Prozent reduzieren.

Werden die allgemeinen Informationen der gesamten Hochschule ebenfalls in das System eingetragen, so kann das Postfach lediglich für persönliche und organisatorische Absprachen innerhalb der Hochschule genutzt werden und der administrative Aufwand des E-Mail-Speichers kann aufs Kleinste beschränkt werden.

4.0.2 Diskussion

In diesem Kapitel wird das Ergebnis der Arbeit in Bezug auf die Forschungsfrage diskutiert. Außerdem wird der Prototyp mit einem bereits vorhandenen Framework verglichen und in Bezug darauf eingeordnet.

Unter Verwendung der entwickelten Erweiterung kann die E-Mail-Flut der Hochschule unter bestimmten Voraussetzungen gedrosselt werden. Die Evaluation anhand eines Probanden zeigt eindeutig das Potenzial, durch eine optimierte Personalisierbarkeit, die Anzahl von Nachrichten zu reduzieren. Anhand der beispielhaften Zählung der im Postfach vorhandenen E-Mails kann zudem festgehalten werden, dass eine Großzahl dieser als unnötig für Individuen einzustufen ist. Zu beachten ist jedoch, dass es sich bei der Bewertung nur um eine theoretische Annäherung eines

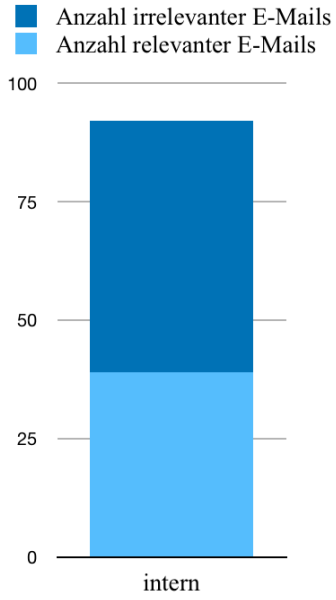

Abbildung 4.5. Vergleich relevanter und nicht relevanter E-Mails.

realen Ergebnisses handelt.

Weitere Schritte, um den Einsatz des Prototypen finalisieren zu können, sind ein ausführliches Testing, was im Rahmen dieser Arbeit nicht untersucht wurde. Unter Beobachtung der einzubindenden Web-Erweiterung kann die Plattform für einen definierten Zeitraum genutzt werden und in Folge dessen eine detaillierte Aussage über die mögliche Reduzierung des Speicheraufwands im Postfach möglich sein.

Das Ergebnis dieser Arbeit wird im Folgenden mit Eigenschaften des Kursmanagementsystems Moodle verglichen. Die Struktur des Prototypen ist, wie in den oberen Kapiteln bereits erläutert, mit verschieden einordenbaren Tags realisiert. Bestimmte Benutzer können Informationen einpflegen und verwalten. In Moodle ist der Vorgang ähnlich realisiert. Hier können Lehrende Material und Informationen in verschiedenen Lernräumen hochladen. Das System ist, im Gegensatz zum Prototyp, sehr umfangreich. Als Benutzer ist es möglich, sich in diese Lernräume einzutragen und durch die Anmeldung aktuelle Benachrichtigungen zu erhalten. Sind manche Informationen in Moodle nur mit einem extra Passwort zugänglich, so ist das in der Erweiterung dieser Arbeit für alle Benutzer gleich verfügbar (vgl. [Dok15]).

Die Menge der Daten einer solchen Plattform sind nicht zu unterschätzen. Moodle verwendet unter anderem Caching-Tools und optimierte Prozesse, um die Datenbanken zu befüllen und schnellstmöglich abfragen zu können. In dieser Arbeit liegt der Schwerpunkt hingegen nicht auf der Optimierung einer Datenbank oder dem Verbessern der Performanz.

Zusammenfassend lässt sich sagen, dass Moodle ein zu umfangreiches Repertoire

an Möglichkeiten bietet, im Gegensatz zum entwickelten Prototyp. Alle Funktionen, die in der Web-Erweiterung enthalten sind, lassen darauf schließen, dass unter Betrachtung der Evaluation, der Prototyp zur Reduzierung der E-Mail-Flut beitragen kann.

Kapitel 5

Schlussbetrachtung

Um die Forschungsfrage, „Kann die E-Mail-Flut der Technischen Hochschule mit Hilfe einer Social Media Plattform gedrosselt und die Nachhaltigkeit der Informationen gewährleistet werden?", zu beantworten, wurde in dieser Arbeit ein Prototyp entwickelt.

Hierbei handelt es sich um eine Erweiterung der Fakultätsinternen Website. Mit Hilfe von Tags können Informationen abonniert und ein personalisiertes Dashboard zusammengestellt werden. Durch differenzierte Berechtigungen ist es realisierbar, bestimmten Benutzern das Einpflegen und Verwalten von Benachrichtigungen zu gewähren.

In der Evaluation wurde durch eine Kalkulation, stichprobenartig anhand eines Probanden dargelegt, in wie weit die eintreffenden Informationen im Postfach der Studierenden irrelevant sind und durch die individualisierbare Website eingespart werden können.

Diese wissenschaftliche Arbeit hat gezeigt, dass durch den Einsatz einer webbasierten Plattform, Informationen an verschiedenste Personen einer Einrichtung verteilt werde können. Durch die individuellen Konfigurationsmöglichkeiten sind die Empfänger nicht überfordert und können mit wenig Aufwand passende Themengebiete abonnieren. Die E-Mail-Flut wird stark verkleinert und der administrative Aufwand des Postfachs reduziert.

Im Folgenden wird beschrieben, wie der Prototyp erweitert und optimiert werden kann.

5.0.1 Ausblick

Betrachtet wird der aktuelle Stand der Anwendung, wie dieser aufbauend optimiert und in Verwendung gebracht werden kann.

Der Umfang der verwendeten Tags wird äquivalent zur Dimension der verschiedenen Themen an Informationen stark ansteigen. Es bietet sich an, diese zukünftig in Gruppen zu unterteilen um die Datenbank logisch und performant aufbauen zu können. Zudem ist das automatische Anlegen von Metadaten der einzelnen Tags bei der Administration hilfreich. Beispiele hierfür wären, wann wurde das Schlagwort erstellt, wie oft taucht es in der Suche auf und wie viele Abonnenten besitzt dieses. In Folge dessen käme eine eigens entwickelte Datenkonstruktion für Tags in Frage, um den Taggable Manager abzulösen.

Eine weitere wichtige Eigenschaft von Informationsplattformen ist das regelmäßige Abrufen der neusten Mitteilungen. Um das Interesse der Studierenden und Lehrenden aufrecht zu erhalten, kann ein Skript integriert werden, bekannt unter dem Name Cron, dass asynchrone Benachrichtigungen ermöglicht. Wird ein `Cron-Job` auf dem systeminternen Server ausgeführt, so wird zyklisch der Benutzer über Neuigkeiten per Mail informiert (vgl. [Dok18]). Im Prototyp wurde das Verfahren der asynchronen E-Mail-Benachrichtigung wie folgt getestet: Mit der Konfiguration des Shell-Skripts werden die Sendezyklen und die Inhalte festgelegt. Die Abbildung 5.1. zeigt den Cron-Tab, der wöchentlich gesendet werden soll (`0 0 * * 0`). Des weiteren werden die Pfade der Entwicklungsumgebung und des Servers mitgegeben. Um das Senden der Mails überwachen zu können, werden die Aktivitäten in ein Log-File geschrieben und sind im Administrator-Backend abrufbar.

```
0 0 * * 0 /Users/Esthi/thesis_ek/thesisenv/bin/python3 /Users/Esthi/thesis_ek/manage.py
send_queued_mail >> send_mail.log 2>&1
```

Abbildung 5.1. Cron-Tab der im Prototyp getesteten Benachrichtigung.

Des weiteren ist zur besseren Benutzung der Web-Erweiterung die Usability optimierbar. Die folgenden Punkte zeigen konkrete Möglichkeiten auf, die Gebrauchstauglichkeit für den User zu steigern:

- Einbindung einer Tagcloud mit Verlinkungen zu Artikeln die diesen Tag enthalten.
- Die Auswahl an Tags für Autoren beim Einpflegen von Artikeln optimieren, zum Beispiel mit Vorschlägen ähnlicher Schlagwörter.
- Das Hinzufügen und Löschen von Tags mit Hilfe einer grafischen Schaltfläche realisieren. Sinnvoll wäre es hierbei die Bibliothek von Bootstrap zu erweitern.

Zusammenfassend lässt sich feststellen, dass die Weiterarbeit am Prototyp unabdingbar ist um diesen später in die Fakultätswebsite einbinden zu können.

Referenzen

[BA11] Twitter Inc Bootstrap Authors. Bootstrap repository. 2011. https://github.com/twbs/bootstrap.

[Com18] The Computer Language Company. Definition of: user permissions. 2018. https://www.pcmag.com/encyclopedia/term/58231/user-permissions.

[Coo10] Oracle Cooperation. About ldap. 2010. https://docs.oracle.com/cd/E19182-01/820-6573/6nht2e5a4/index.html.

[Dix18] Chitrank Dixit. Pep-8 tutorial: Code standards in python. 2018. https://www.datacamp.com/community/tutorials/pep8-tutorial-python-code.

[Dja18a] Djangogirls. Creating a blog post model. 2018. https://tutorial.djangogirls.org/en/django_models/.

[Dja18b] Djangogirls. We inspire women to fall in love with programming. 2018. https://djangogirls.org/.

[Dok15] Moodle Dokumentation. Aufbau einer moodle-site. 2015. https://docs.moodle.org/35/de/Aufbau_einer_Moodle-Site.

[Dok18] Moodle Dokumentation. Cron-job. 2018. https://docs.moodle.org/35/de/Cron-Job.

[Fio14] Marzena Fiok. E-mail-flut sorgt für kostenlawine. 2014. https://www.tecchannel.de/a/e-mail-flut-sorgt-fuer-kostenlawine,402338,3.

[FMS17] Andreas Donner Frank-Michael Schlede, Thomas Bär. Was ist ldap (lightweight directory access protocol)? 2017. https://www.ip-insider.de/was-ist-ldap-lightweight-directory-access-protocol-a-581204/.

[Fou18a] Django Software Foundation. Cross site request forgery protection. 2018. https://docs.djangoproject.com/en/dev/ref/csrf/.

[Fou18b] Django Software Foundation. django.contrib.auth, user model. 2018. https://docs.djangoproject.com/en/2.1/ref/contrib/auth/.

[Fou18c] Django Software Foundation. Modelforms - the save() methode. 2018. https://docs.djangoproject.com/en/dev/topics/forms/modelforms/#the-save-method.

[Fou18d] Django Software Foundation. Working with forms. 2018. https://docs.djangoproject.com/en/dev/topics/forms/#using-a-form-in-a-view.

[Fou18e] Python Software Foundation. Virtual environments and packages. 2018. https://docs.python.org/3/tutorial/venv.html.

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

[Gmb18] Sario Marketing GmbH. Tagging. 2018. https://www.textbroker.de/tagging.

[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.

[Mic18] Microsoft. Extensions for the visual studio family of products. 2018. https://marketplace.visualstudio.com/.

[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.

[Nev15] Ryan Nevius. django-post_office git repository. 2015. https://ryannevius.com.

[Ong18] Selwin Ong. Django request-response cycle. 2018. https://github.com/ui/django-post_office/blob/master/AUTHORS.rst.

[Ott11] Mark Otto. Bootstrap from twitter. 2011. https://blog.twitter.com/developer/en_us/a/2011/bootstrap-twitter.html.

[Ott12] Mark Otto. Say hello to bootstrap 2.0. 2012. https://web.archive.org/web/20120203191214/https://dev.twitter.com/blog/say-hello-to-bootstrap-2.

[Sha09] Shabda. Understanding decorators. 2009. https://www.agiliq.com/blog/2009/06/understanding-decorators/.

[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.

[Sol17] Mindfire Solutions. Advantages and disadvantages of python programming language. 2017. https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121.

[Tim15] Damon Timm. django-hitcount documentation. 2015. https://django-hitcount.readthedocs.io/en/latest/overview.html.

[Wei17] Michael Weigend. *Python GE-PACKT*. 2017. Kapitel 23.1.