TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Entwicklung einer internen Social Media Plattform mit personalisierbarem Dashboard fur Studierende

Esther Beate 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, 18. November 2018

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

Content of this Bachelor thesis is ....

Keywords: bla,bla Georg Simon Ohm, Wirtschaftsinformatik

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

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 den Einzelnen aufweisen können. Dadurch sind die Postfäch 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 veröffentlichten 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 der einfache Syntax ermöglicht einen 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 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.

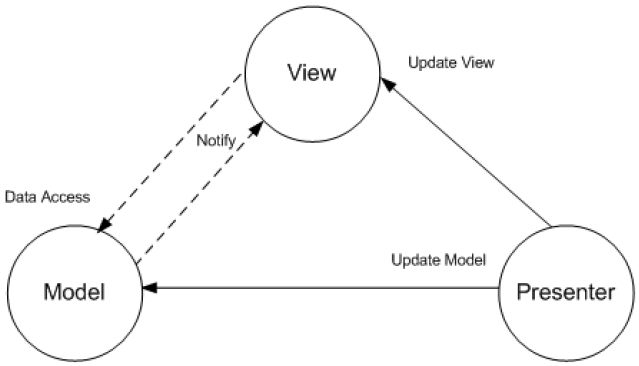

Abbildung 2.1. Vereinfachter MVP [She09]

Der User gibt eine URL im Browser ein und sendet sie an den Web-Server. Das Interface WSGI 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 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*(vgl.). 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. Es ist 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. [Kin17], Abbildung 2.2.).

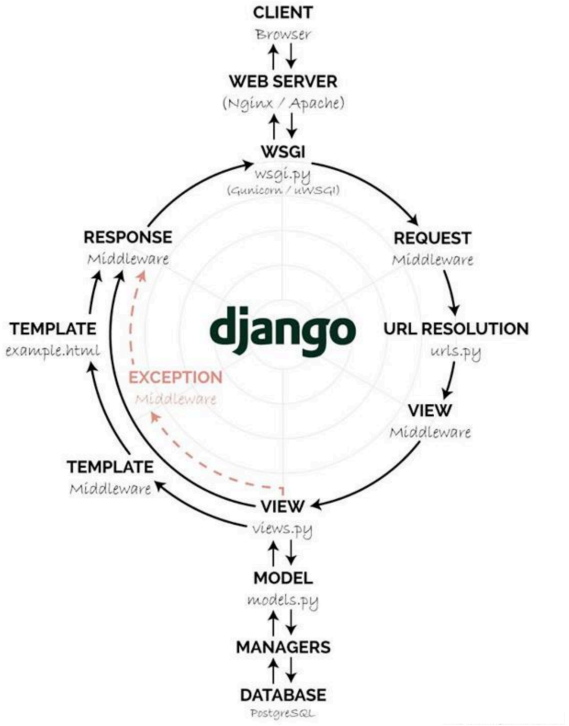

Abbildung 2.2. Request-Response-Kreislauf des Django Frameworks [Nev15]

2.1.1 Besonderheiten Djangos

Das Django-Framework bringt einige Besonderheiten mit sich, die beim 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, so importiert man die Decorator des, von Django zur Verfügung gestellten, Authentifizierungssystems mit

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

Vor der Definition der Funktion wird dann folgende Zeile ergänzt:

@login_required

Natürlich lassen sich Decorator auch für andere Zwecke vor Funktionen platzieren. Unter Anderem werden so die Views vor möglichen Angriffen, wie Cross-Site-Scripting, geschützt.

Durch den einfachen Aufbau ist es außerdem möglich diese 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, verschiedensten Module zu verwenden und zu testen. Die Versionen dieser spielen hierbei eine entscheidende Rolle, um Konflikte zu vermeiden [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. Die folgende Abbildung (2.4) zeigt das Erstellen eines neuen Ordners, das Erstellen der virtuellen Umgebung und den Aktivierungsbefehl. Ist der Name des Environment in Klammern am Kommandozeilenanfang, bedeutet das, diese ist jetzt aktiv.

```
[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, „rekursives Akronym für pip Install Packages"(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.

2.1.3 Lightweight Directory Access Protocol

Das ldap, Lightweight Directory Access Protocol, muss als Erweiterung in die hier bearbeitende Bachelor-Arbeit eingebunden werden, um später die Login-Daten im

Hochschulinternen Netz abfragen zu können. Dies ist ein Internetprotokoll, welches die Kommunikation mit dem Active Directory verwaltet. Es wird eingesetzt um Benutzer so schnell und effizient wie möglich durch eine bereits existierende Datenbank abzufragen und zu authentifizieren. Der Aufbau ist mit einem Baum zu vergleichen (vgl. Abbildung 2.4.). Die Wurzel besteht aus sehr allgemeinen Informationen, umso näher man den Blättern kommt, umso spezifischer werden diese. Ein Objekt in der Struktur wird durch einen einmaligen Namen identifiziert, der aus den gesamten hinterlegten Informationen besteht. Der Name für den in der Abbildung 2.4 dargestellten Baum wäre „cn=John Doe, ou=People, dc=sun.com“(vgl. [Sch17]).

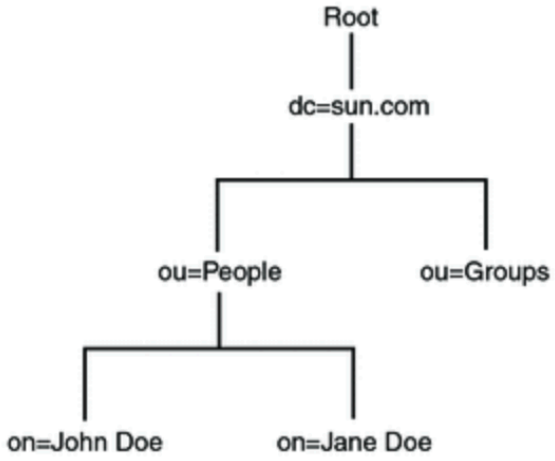

Abbildung 2.4. Beispiel eines LDAP-Trees [Orc10]

2.1.4 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*, *Clickjacking* und die Sicherstellung der *Session-Security*.

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 dem eingebettetem JavaScript-Code die Daten im Server verändert werden sollen. Hierbei nutzt der Angreifer die Rechte eines eingeloggten Benutzers und kann somit Informationen im Back-end verfälschen. Um dies zu verhindern wird im Template des Prototypen, zwischen den Form-Tags der *crsf-Token* 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*, ein generierter Zusatzwert, der das entschlüsseln dieser um ein vielfaches erschwert. Wird ein Request gesendet, vergleicht eine von Django initzialisierte *Middleware* beide Zahlen. Sind diese nicht gleich, hat ein Dritter also die Informationen manipuliert, wird der 403 HTTP-Standart-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 Bewertung an. Eine Vielzahl an Bibliotheken wurden für diese Arbeit getestet, aber wegen mangelnder Kompatibilität oder Funktionalität als nicht hilfreich erachtet und werden deshalb hier nicht weiter erwähnt. Die im Folgenden aufgeführten Bibliotheken sind im Prototyp zur Anwendung gelangt.

2.2.1 Taggable-Manager

Django-taggit ist eine Erweiterung, die das Verwenden von Tags vereinfacht. 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 (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 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 (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 beschreiben werden. 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.

2.2.2 Hilfsbibliotheken

—warum habe ich diese bibs gealden (beschreibung evtl in prototyp) 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 Tags öfters vor als andere, so werden sie entsprechend größer dargestellt.

Django-hitcount dient zum zählen der Besucher einer Seite (vgl. [Tim15]). Dies 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 weiter 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 in der View und im 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.

Um das Versenden und Verwalten von E-Mails in Django zu realisieren eignet sich django-post-office (vgl. [Ong18]). Nach der Installation kann im Admin-Backend ein E-Mail-Templates angelegt werden, Anhänge verwaltet und das Senden dieser im

Log überprüft werden. Die Benachrichtigungen können asynchron versendet werden mit einem integrierten Planungsmanager. Der Inhalt kann Text oder HTML-basiert sein und in mehreren Sprachen hinterlegt werden.

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 (vgl. [Ott11]). Die Open-Source-Bibliothek, die auf GitHub abrufbar ist, wird seitdem von vielen Programmierern weiterentwickelt und ist somit stark gewachsen. Version 2.0 verfügt außerdem über die Fähigkeit Websites responsive auf verschiedenste mobile Endgeräte anzupassen (vgl. [Ott12]).

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). Das bedeutet, dass Media-Queries 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 2.5. 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. Möchte man Bootstrap umfangreich nutzen so lassen sich die Stildefinitionen mit Klassen oder ID's in diverse HTML-Tags eintragen (vlg. Abbildung 2.4.).

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

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

Abbildung 2.6. Bootstrap-Klassen in HTML-Tag

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

Um die wissenschaftliche Frage, nicht nur zu beantworten, sondern zu beweisen, wird in dieser Arbeit die Methode des Prototypings genutzt. Der Prototyp dient zum experimentellen Arbeiten und sichert eine strukturell fundierte Umsetzung des darauf folgenden 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 aufbauend auf Diesem ein Produkt realisiert werden, das als Erweiterung in das Netzwerk der Hochschule integriert werden soll.

3.1 Forschungsdesign

Das Kapitel zeigt 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, des sich aus der Forschungsfrage ergebenden Problems analysiert und 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. Das Implementieren der Applikation kann nun auf Basis der Recherche durchgeführt werden. Dazu gehört das Testen verschiedener Bibliotheken und Erweiterungen um die 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.

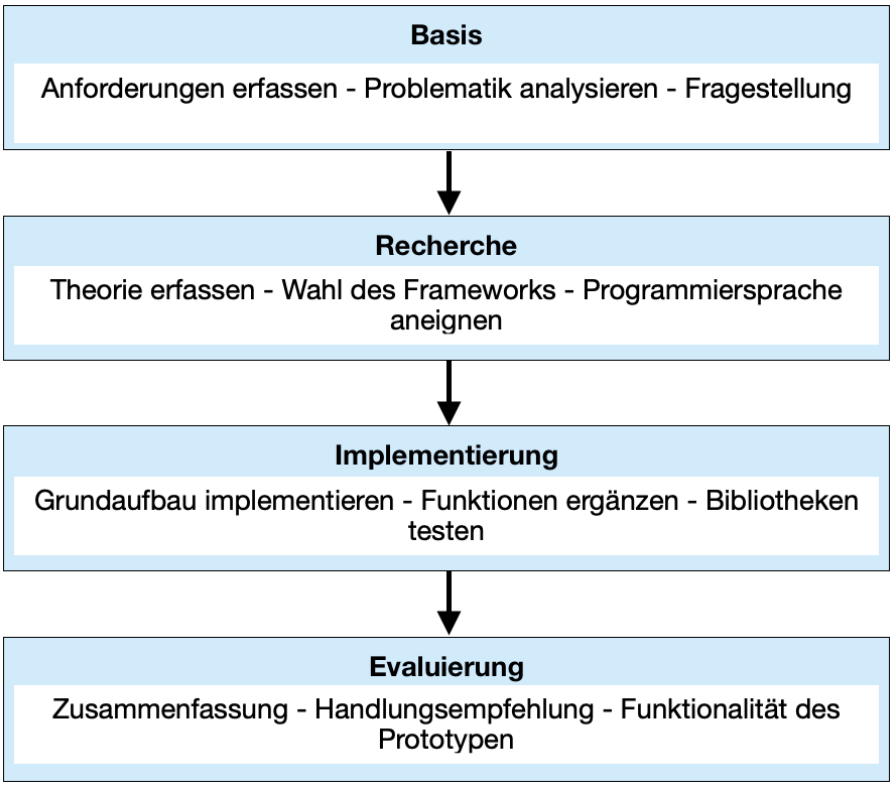

Abbildung 3.1. Forschungsdesign

3.2 Organisation

Zum Entwickeln des Prototypen wird die Open-Source- Entwicklungsumgebung Visual Studio Code von Microsoft verwendet. Die dynamische Oberfläche ist sehr performant, da sie wenig Speicherplatz bedarf und zunächst ohne größere Ergänzungen sofort einsetzbar ist. Um den Code mit Shortcuts übersichtlicher zu gestalten und auf Python und Django abstimmen zu können werden aus der üppigen Extension-Bibliothek kostenlose Erweiterungen eingebunden. Unter Anderem sind die folgenden Add-on's während der Arbeit zum Einsatz gekommen (vgl. [Mic18]):

- Python von Microsoft (Linting und Debugging)
- MagicPython von MagicStack Inc. (Syntax-Highlighter)
- Python Extension Pack (Django Code-Vervollständigung)

Der entwickelte Syntax wird regelmäßig auf das hochschulinterne Git-Repository geladen. Hier kann man mit Hilfe der verwendeten IDE bei der Überarbeitung im sogenannten `Working-Tree` die Änderungen visuell einsehen und wenn nötig bearbeiten. Es wird ausschließlich auf den Master-Branch gepusht, da nur ein Entwickler

an dem Prototyp tätig ist. Lediglich zum Testen experimenteller Bibliotheken werden neue Branches angelegt und diese, falls Sie sich als hilfreich erweisen, mit dem Master-Branch fusionieren.

Um einen Einblick in den Aufbau eines Django-Projektes zu erlangen wird dies im folgenden genauer beschrieben. Die unterste Projektebene wird durch die manage.py-Datei 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. Die `__init__.py` und wsgi.py sind trivial und deshalb wird hier nicht weiter darauf eingegangen. Im Ordner `thisisenv` sind alle Bibliotheken und Add-on's der virtuellen Umgebung hinterlegt. Der Fokus 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 dessen werden hier genauer erläutert. Zunächst wird auf die Ergänzung des bestehenden `UserModel` eingegangen, nachdem veranschaulicht der 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 diese integriert werden müssen (vgl. [Dja18]). Mit der folgenden Eingabe

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

werden die neun Tabellen der Modelle erstellt. Um diese dann auch anwenden zu können, muss der Befehl

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

darauffolgend ebenso in die Kommandozeile eingegeben werden.

UserModel:

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

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

Das Feld `groups` wird in dieser Arbeit nicht verwendet und deshalb im Folgenden ignoriert.

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 nur aus einem Objekt besteht (vgl. [Fou18a]). Das heißt, dass hier keine Rekursiven, also 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.

```
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

PostModel:

Das `PostModel` beschreibt alle Felder die ein Post enthalten kann. Basierend auf der Blog-Lösung von Djangogirls.com gehören dazu folgende:

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

Der Autor ist durch einen `ForeignKey` mit dem `UserModel` verbunden. Diese sogenannte `ManyToOne` Verbindung reicht hier aus um einem Post den 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 wir deshalb als `TextField` deklariert. Erstellungsdatum und Publikation sind beides `DateTimeField`s. Ersteres muss vom Ersteller angegeben werden, Zweiteres kann zunächst offen gelassen werden durch die Zusatzangabe `null=True`. Ein weiteres Feld tags wird hinzugefügt um den Posts

unabhängig von den 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 angelegtem CustomUser. Die im User vorkommenden booleschen Felder werden im Kapitel Berechtigung der User genauer erörtert.

3.2.2 Verwaltung im Administrator-Back-end

In diesem Kapitel wird beschrieben wie das Administrations-back-end genutzt werden kann. Es ist jedoch zu beachten, dass die Applikation vorwiegend von Dozenten und Angestellten der Hochschule ohne Administratorrechte verwendet werden soll. Die gestaffelten Berechtigungen werden im Kapitel Berechtigung der User genauer beschreiben.

Ein Django-Projekt bildet bereits beim Einrichten, `per Default`, eine Administrator-Oberfläche um die Inhalte der Website kontrollieren zu können. Nach der Migration von den oben genannten Modellen wird diese erweitert. Nich zu vergessen sind die externen Tabellen der installierten Add-on's, die nach der Migration das Back-end expandieren.

3.2.3 Berechtigung der User

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

Das gestaffeltes 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 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 fokussiert. Das Hinzufügen von Tags um die damit verbunden Posts auf dem persönlichen Dashboard zu sehen wird ihnen gewährleistet. Dies soll verhindern, dass Informationen nicht zu leichtfertig geändert oder gelöscht werden.

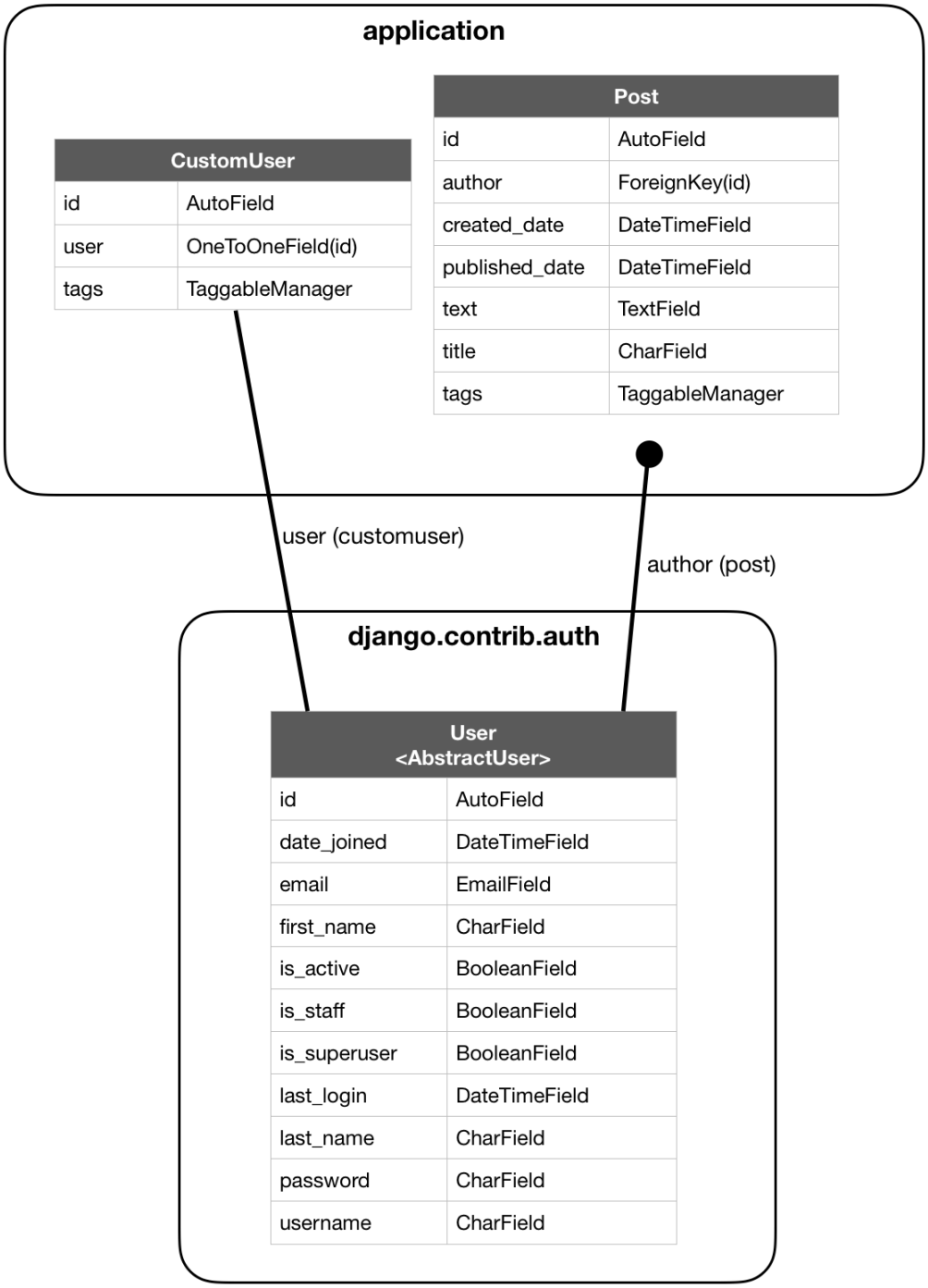

Abbildung 3.3. Datenmodellierung von `User` und `Post`

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 somit ebenso das persönliche Dashboard gestalten. Das Einloggen in die Administratoroberfläche kann vorgenommen werden, jedoch sind der Gruppe noch keinerlei Rechte zugewiesen. Möchte man dies ändern, kann man das von Django

bereitgestellte Feld `User Permissions` im Admin-back-end unter Users, und dem Namen der Person, die gewünschte Berechtigung zuteilen. Diese sind von Django vorgegeben und betreffen alle vorhandenen Modelle der Applikation. Soll ein User zum Beispiel erlaubt sein einen Tag aus dem Archiv zu löschen, so wählt er das Feld `taggit |Tag |can delete Tag` und schiebt es von der Auswahl zu den Berechtigungen. Durch das Setzen des booleschen Wert `is_staff` auf `True` beim Erstellen der Benutzer, ist es möglich im Code der Applikation Abfragen durchzuführen. Dadurch lassen sich bestimmte Views an die eingeloggte Personengruppe anpassen. So ist das Menü für Dozenten und Angestellte ein umfangreicheres als das, der Studenten.

3.3 Funktionen

Um die wichtigsten Funktionen des Prototypen festlegen zu können werden User Stories erstellt (vgl. Abbildung 3.4.). 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, Orang wichtige Funktionen sind aber nicht unbedingt notwendig und Grün kaum relevant sind.

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	🟡

Abbildung 3.4. User Stories

3.3.1 Verwalten

Das Verwalten der Artikel soll von berechtigten Nutzern im Front-end stattfinden, hingegen die prozessuale Logik im Code-Back-end passiert. Der Vorgang des Er-

stellens, 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 festgelegt 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 Post automatisch der Liste der Entwürfe beigefügt.

Speichert der Benutzer nun den Artikel, so werden im Back-end die Daten wir folgt verarbeitet. In der View `post_new` wird zunächst die Validität aller Eingaben geprüft. Falls dies der Fall ist, wird der Post als Objekt zurückgegeben, jedoch durch das optionale Keywort `commit=false` noch nicht in der Datenbank abgelegt. 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 nachdem explizit das von Django zur Verfügung gestellte `form.save_m2m()` aufrufen zu können. Dieser 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 solchen Schlüssel zu einem Feld zugewiesen. Ist das der Fall, erstellt Django automatisiert beim Speichern der Tabelle diesen als `AutoField` im Feld `Id` und zählt bei jedem neu Erstellen eines Objekts hoch. Somit sind alle Objekte eindeutig zuordenbar und können mit dem Kommando `post.pk` jederzeit aufgerufen werden.

Einen bereits vorhandenen Artikel löschen:

In der Detailansicht eines Artikels ist es möglich diesen durch klicken auf den Button Löschen zu entfernen. Die View `post_remove` selektiert über den im Template mitgegebenen `Primary Key` das Objekt und speichert dies in der Variable `post`. Um dem Benutzer nochmals mitteilen zu können, welchen Artikel er entfernt hat,

wird vor dem Löschen eine Nachricht mit dem Titel dessen generiert. Nachdem wird dieser gelöscht (`post.remove()`) und eine Umleitung am Ende der View-Definition schickt den Benutzer auf die Seite der Artikelliste. Hier wird die vorher erstellte Nachricht aus informellen Gründen eingebunden.

Einen bereits vorhandenen Artikel bearbeiten:

Ähnlich wie beim Löschen eines Artikel, kann man diesen in der Detailansicht bearbeiten. Mit dem Betätigen des Bearbeiten-Buttons wird der User auf den Artikel-Editor weitergeleitet. Die Seite zeigt die Form, wie beim Erstellen eines neuen Artikels, nur dass hier die Felder bereits den Inhalt des zu Editierenden Textes enthalten. Dazu wird in der View über den `Primary Key` der Artikel einer Variable `post` zugeordnet. Die bedingte Anweisung fragt in Zeile 91 der Abbildung 3.5., ob der Benutzer die Eingaben bereits zum Speichern veranlasst hat. Wird die Bedingung zunächst nicht erfüllt, rendert die Funktion lediglich die `PostForm`, mit dem bereits eingepflegten Inhalt durch eine GET-Abfrage in der die Daten des Artikels als Instanz übermittelt werden (vgl. Abbildung 3.5. Zeile 100).

Betätigt der Benutzer den Speichern-Button im Front-end wird die bedingte Abfrage 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 diese 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 erwähnten neu Erstellen eines Artikels und muss nicht weiter beschreiben.

3.3.2 Abonnieren

Das Abonnieren bestimmter Themengebiete ist eines der wichtigsten Funktionen im Prototyp.

Nach längerer Recherche im Netz wird unter Berücksichtigung aller Vor- und Nachteile ein Tag-Modell zur Umsetzung hierfür gewählt. Wie bereits in der Datenmodellierung angedeutet, besitzt jeder Artikel beschreibende Tags. Hierbei handelt es sich um kurze stichwortartige Beschreibungen, die den diesen 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 gleich Menge der Schlagwörter verwenden,

```
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 3.5. Funktion post_edit, Auszug aus views.py.

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.

Im Prototyp findet man die Abonnier-Funktion in der Menüleiste unter Suche. Hier erscheint ein zwei-geteiltes 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`, die diese 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 aber hier nochmals 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 3.6. wird der eingeloggte Benutzer der Variable `user_instance` übergeben. Beim Erstellen der `Model-Instanz` (vgl. Abbildung 3.6., 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 3.6., Zeile 163) und ebenfalls dem aktuellen Benutzer zugeordnet. Die Eingabe der `form` wird in ein `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

```
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 3.6. Funktion search_add, Auszug aus views.py.

Tags abgeglichen (vgl. Abbildung 3.6., Zeile 168). Wird Bedingung erfüllt, speichert die Funktion die Tags. In beiden möglichen Fällen, wird der Benutzer benachrichtigt ob der Vorgang erfolgreich oder die Eingabe nicht valide ist.

Nun werden auf dem Dashboard Artikel der neu hinzugefügten Tags angezeigt (vgl Abbildung 3.7.).

3.3.3 Filtern

Für eine effiziente Nutzbarkeit des Prototypen ist es wichtig, dass Benutzer intuitiv nach Tags suchen und diese selektieren können. Hierfür werden verschiedene Möglichkeiten zur Verfügung gestellt, die die Usability der Website verbessern.

Im persönlichen Newsfeed des Dashboards sind, die zu den Artikeln zugewiesenen Schlagwörter jeweils mit Verlinkungen versehen. Möchte ein Benutzer weitere Artikel zu einem bestimmten Thema lesen, so muss er lediglich auf den entsprechenden Tag klicken und erhält somit eine Liste aller Posts, die diesen 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* dessen mitgegeben. Außerdem

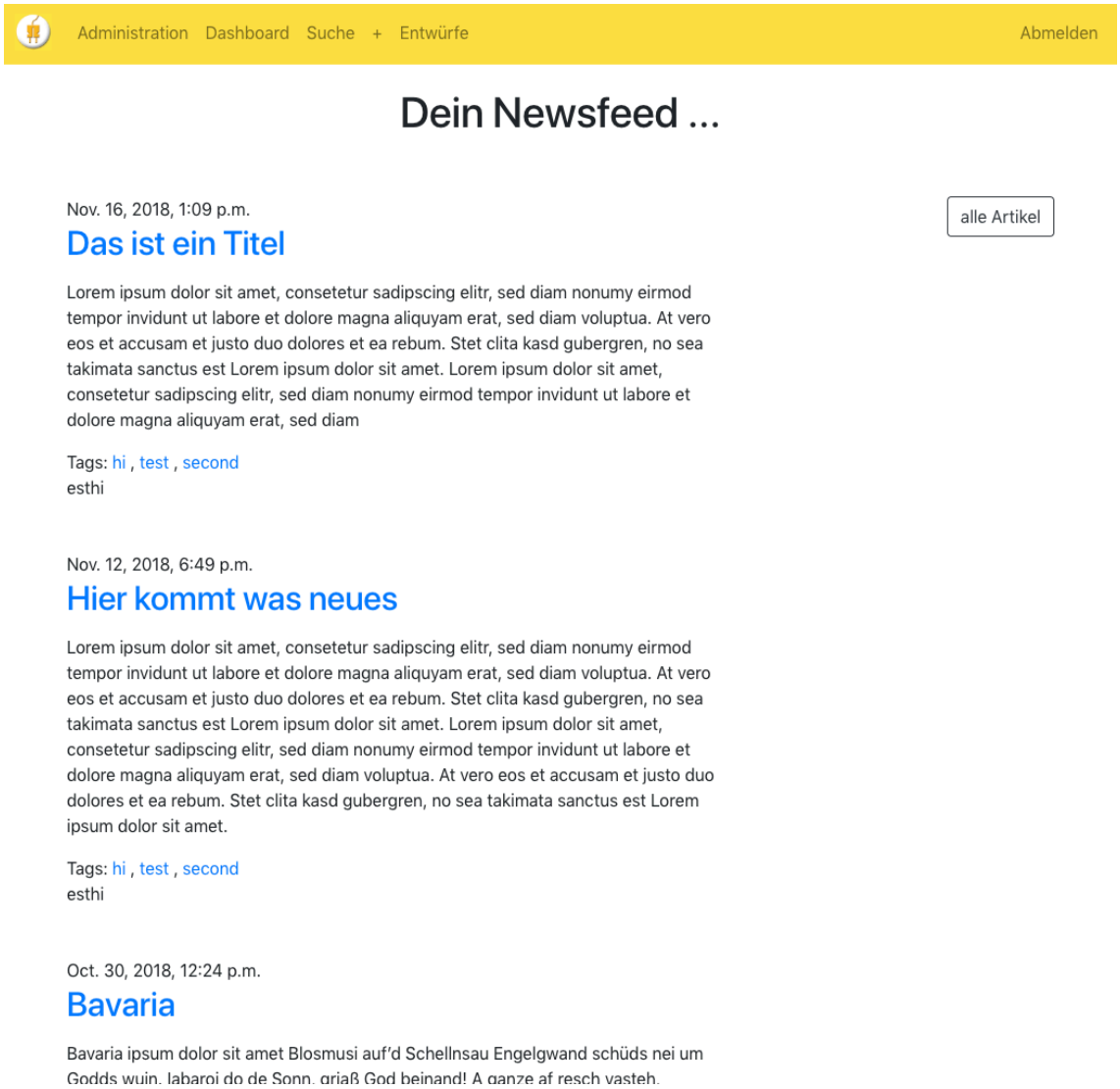

Abbildung 3.7. Prototyp Newsfeed Seite

wird nun die Url `post_list_by_tag` aufgerufen, die auf eine neue Seite verweist

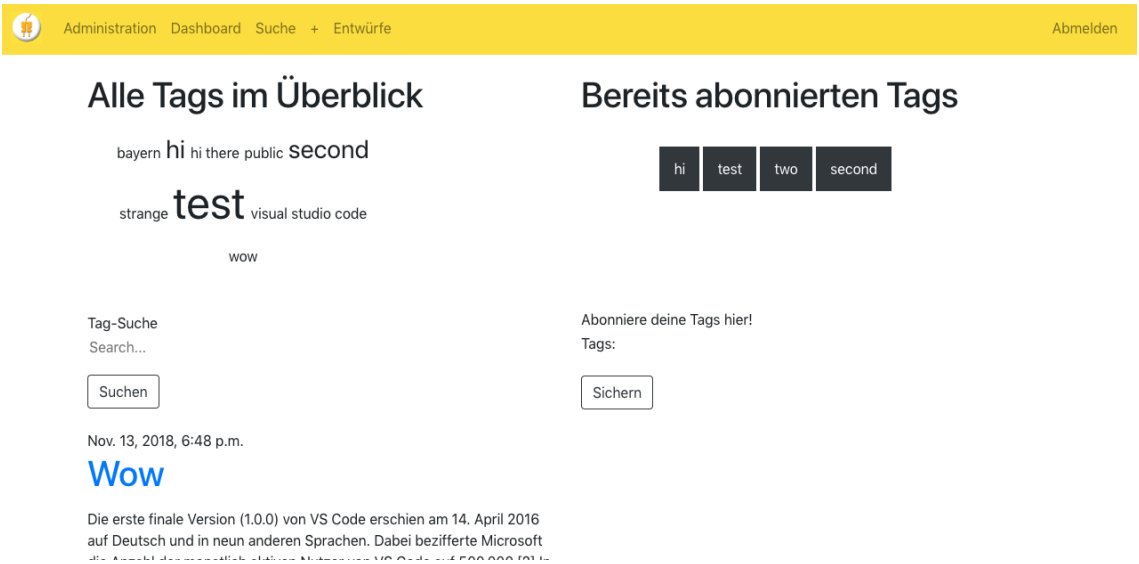

Abbildung 3.8. Prototyp Suche- und Abonnier-Seite

Kapitel 4

Ergebnis

4.0.1 Evaluierung

Eine weitere hilfreiche Erweiterung ist pylint. Das Tool sucht nicht nur nach Fehlern im Code, sondern versucht diesen sauber und einheitlich zu gestalten. Hierbei wird auf den Code-Standard PEP-8 geprüft [Dix18]. Die folgende Liste zeigt eine Kurzfassung der wichtigsten Regeln:

- Einrückung, meist 4 Leerzeichen
- Maximale Zeichenanzahl pro Zeile
- Zwei Leerzeile zwischen Klassen und Funktionen
- Eine Leerzeile zwischen Methoden innerhalb einer Klasse
- Leerzeichen in Ausdrücke und Anweisungen vermeiden
- Die Reihenfolge der Importe: Standartbibliotheken, Drittanbieterbibliotheken, Lokale Anwendungen
- Konventionen der Namensgebung von Funktionen, Modulen usw.

Natürlich sind dies Vorgaben, die eingehalten werden können, aber nicht notwendig sind um den Code fertig kompilieren und ausgeben zu lassen.

Kapitel 5

Zusammenfassung und Ausblick

Zusammenfassung...

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.

[Dja18] Djangogirls. Creating a blog post model. 2018. https://tutorial.djangogirls.org/en/django_models/.

[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 (light-weight 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.

[Ong18] Selwin Ong. django-post_office git repository. 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.