Implementierungsdokument zu

Evoks

Erarbeitet von:

Yannis Preker, Jan Maly, Klara Eckhardt, Umut Ulas Cin, Jonas Scholz

PSE im Sommersemester 2021

Steinbuch Centre for Computing (SCC)
Betreut von: Danah Tonne, Felix Ernst, Andreas Pfeil

Inhaltsverzeichnis

1 Einleitung

Im Rahmen dieses PSE-Praktikums wurde die Webanwendung Evoks entwickelt. Sie soll dazu dienen, Vokabulare zu bearbeiten, zu verwalten und in einem bereits vorhandenen Vokabularbrowser anzuzeigen. Nach den Vorgaben der Gruppe Data Exploitation Methods am Steinbuch Center of Computing wurde das Softwareprojekt in Python mit dem Web-Framework Django umgesetzt. Als Webbrowser für Vokabulare ist die frei verfügbare Software Skosmos integriert und die Vokabulare werden in Jena-Fuseki Triple Stores verwaltet.

Dieses Dokument dient der Zusammenfassung der Implementierungsphase des PSE Praktikums. Im Folgenden werden die umgesetzten Kriterien vorgestellt und Änderungen am Entwurf sowie Verzögerungen im Zeitplan dokumentiert. Im letzten Kapitel werden einige, für das Praktikum interessante Metriken vorgestellt.

2 Implementierte Kriterien

Im folgenden sind alle Muss-, Soll- und Wunschkriterien des Pflichtenheftes aufgelistet. Ein ✔ Symbol zeigt an, ob ein Kriterium vollständig umgesetzt wurde, ein ✘ Symbol zeigt an, dass ein Kriterium nicht oder nicht vollständig umgesetzt wurde.

Alle Muss- und Sollkriterien wurden erfolgreich implementiert.

2.1 Musskriterien

Kriterium	Implementiert
[MK 10] Die Möglichkeit Accouts für NutzerInnen zu erstellen	✔
[MK 20] NutzerInnen können beliebig vielen Gruppen zugeordnet werden	✔
[MK 30] Es gibt für alle NutzerInnen eine Übersichtseite der für sie einsehbaren Vokabulare	✔
[MK 40] Vokabulare können erstellt oder importiert werden	✔
[MK 50] Vokabulare und Terme können angezeigt werden	✔
[MK 60] Vokabulare und Terme können bearbeitet werden	✔
[MK 70] NutzerInnen und Gruppen haben verschiedene Zugriffsrechte auf Vokabulare	✔
[MK 80] Vokabulare können sich in den Zuständen Entwicklung, Review oder veröffentlicht befinden	✔
[K 90] Es gibt eine Live- und Testumgebung, jeweils mit eigener Jena Fuseki Datenbanken	✔
[MK 100] Vokabulare können durch Veröffentlichen von der Testumgebung in die Liveumgebung migriert werden	✔
[K 110] Eine Suchfunktion für Terme innerhalb eines Vokabular oder aller Vokabulare eines Nutzers mit Autovervollständigung	✔
[MK 120] Die Webapplikation Evoks kann mehrfach auf unterschiedlichen Servern instanziiert werden	✔
[MK 130] Auf dem Sonderforschungsbereich 980 oder KIT angepasstes Design der Benutzeroberfläche	✔

2.2 Sollkriterien

Kriterium	Implementiert
[SK 10] Ein Feedback-System um Kommentare zu erstellen	✔
[SK 20] Ein Tag-System um Tags zu vergeben	✔
[SK 30] Vokabulare bekommen bei der Migration in die Liveumgebung eine Versionsnummer	✔
[SK 40] - Eine einfache Assistenzfunktion zur Unterstützung der NutzerInnen	✔

2.3 Wunschkriterien

Kriterium	Implementiert
[WK 10] Ein Versionierungssystem um die Historie der Vokabulare einzusehen	✘
[WK 20] Personalisiertes Profil für NutzerInnen	✔
[WK 30] Eine erweiterte Assistenzfunktion zur Unterstützung der NutzerInnen	✘
[WK 40] Ein Konverter um CSV Dateien in SKOS zu konvertieren	✘
[WK 50] Verlinkung von Begriffen mit Enzyklopädien	✘
[WK 60] Eine AAI Anbindung	✘

3 Änderungen am Entwurf

Im Folgenden werden die Änderungen zum Entwurfsheft beschrieben. Die Änderungen sind nach Themengebiet gruppiert.

3.1 Datentype der Attribute

Die Attribute einiger Modelle werden nicht als einfacher Datentyp, sondern als Django-Models-Feld gespeichert. Dieser Django-interne Datentyp ist notwendig, da nur Daten, die in diesem Datentyp gespeichert sind, mit der Django-internen Datenbank verbunden werden und nur dann auf der Admin-Seite angezeigt und geändert werden können. Ein weiterer Vorteil ist, dass Primär- und Fremdschlüssel direkt für die Datenbank markiert werden können oder Grenzen für die Anzahl der Zeichen einer Zeichenkette angegeben werden können.

3.2 Comment

Im Kommentar-Modell wurde das zusätzliche Attribut Vokabular hinzugefügt, das notwendig ist, um den Kommentar auch einem bestimmten Vokabular zuordnen zu können.

3.3 Delete in Profile und GroupProfile

Sowohl bei der Klasse Profile als auch bei der Klasse GroupProfile wird die delete()-Methode weggelassen, da die delete()-Funktion der Django-Klassen User und Group die Möglichkeit des kaskadierenden Löschens bietet.

3.4 Permissions

Bei der Implementierung haben wir festgestellt, dass die Berechtigungen in Django nicht objektorientiert funktionieren, wie wir dachten. Deshalb haben wir die externe Bibliothek Guardian [1] verwendet. Diese bietet die Möglichkeit, objektbasierte Berechtigungen zu definieren. In unserem Projekt ist nun der Ordner guardian zu finden, dieser enthält den Code der Bibliothek und ist nicht von uns geschrieben worden. Das Vocabulary Model importiert die Funktionen dieser Bibliothek.

3.5 Searchfunction

Wir haben festgestellt, dass es eine Suchoption über SPARQL-Abfragen gibt. Wir nuten diese in der entsprechenden View der Suchleiste. Die Suche über SPARQL-Abfragen ist schnell, das ist wichtig, weil sonst die Autovervollständigung nicht reibungslos funktionieren würde. Daher wird auf die Search-Klasse, die Criteria-Klasse sowie die zugehörigen Schnittstellen verzichtet.

3.6 Fuseki und Skosmos

Diese Klassen sind weitestgehend unverändert geblieben. Es sind in den Klassen Fuseki und SkosmosConfig Attribute hinzugekommen die sich als nötig herausgestellt haben. In Fuseki wird das gesamte Vokabular und nicht nur der Vokabularname, in der SkosmosConfig wird die Fuseki Instanz nicht nur der url übergeben. Weiter wurde die Fuseki Klasse um einige Hilfsfunktionen erweitert.

3.7 Notifications

Notifications wurden nicht implementiert, diese werden durch das Pflichtenheft aber auch nicht vorgegeben.

3.8 Vocabulary

Beim erstellen eines Vokabulars muss dessen urispace mit angegeben werden. Ein Vokabular hat ein Array an Prefixes, dadurch können benutzerdefinierte Prefixes hinzugefügt werden.

3.9 Interfaces

Es gibt keine Schnittstellen, wie sie aus der objektorientierten Programmierung mit Java bekannt sind. In Python sind Schnittstellen ein Spezialfall der abstrakten Klasse. Sie sind nicht bindend und um sie zu verwenden, muss man die Bibliothek ABC (Abstract Base Classes) importieren. Das ist so, weil Interfaces von den Entwicklern nicht vorgesehen waren (wie auch der switch-case). In diesem Projekt führte dies jedoch zu Metaklassen-Konflikten, die die Funktionalität beeinträchtigten. Um dies zu beheben, mussten wir die Schnittstellenklassen anders einbinden, damit konnten die Vorgaben durch das Interface ignoriert werden. Wir haben sie entfernt.

3.10 Views

Anstelle von klassenbasierten Views wurden methodenbasierte Views implementiert. Bei klassenbasierten Views handelt es sich um interne Django-Klassen, die die Vorlagen für bestimmte Funktionen bereitstellen. Aufgrund der Standardfunktionalität innerhalb dieser Klassen ist es schwierig, sie zu ändern oder zu erweitern. Methodenbasierte Ansichten hingegen sind einfacher zu lesen, zu implementieren und bieten die Möglichkeit, spezielle Funktionen gut zu integrieren. Durch den Verzicht auf die klassenbasierten Views wurden auch keine Mixins verwendet, da diese nicht mit methodenbasierten Ansichten kombinierbar sind. Durch die Möglichkeit, HTML-Komponenten dynamisch in die Webseiten zu laden, konnten Komponenten wie die Sidebar oder die Suchleiste dennoch entkoppelt und Doppelungen im Code vermieden werden. Dies war das Ursprüngliche Ziel der Mixins. Jedes Django-Modell enthält nun eine einzige view.py-Datei. Diese Datei verwaltet die für dieses Modell relevanten HTML-Seiten.

4 Zeitplan

Zur Planung der Implementierungsphase haben wir eine Kanban-Tafel verwendet. Eine solche Tafel besteht aus vier Spalten, in denen Aufgaben in verschiedenen Stadien angeordnet werden können. Der Vorteil dieser Methode ist, dass jeder sehen kann, ob und wer an einer Aufgabe arbeitet. Außerdem setzten wir kleinere Fristen und betrachteten abgeschlossene Aufgaben gemeinsam und fügten sie mit dem Hauptprojekt zusammen. Im Folgenden befindet sich eine Übersicht über den Zeitplan und die Gründe für Verzögerungen. Ganz am Ende des Kapitels ist eine detaillierte Übersicht über die Aufgaben und ihre tatsächliche Erledigung angehängt.

Deadline 11.Juli

Das Ziel dieser Deadline war es die Grundlagen des Projektes aufzusetzen, sodass dann ohne Probleme mit der Implementierug begonnen werden konnte. Dies beinhaltete unter anderem automatisierte Unittests einbinden, das Projekt in Docker ausführbar zu machen und die Grundstruktur des Projekts zu erstellen.

Alle Aufgaben wurden pünktlich zur Deadline abgearbeitet.

Deadline 14.Juli

Das Ziel dieser Deadline war es die erste Hälfte des Backends fertig zu machen. Hier war geplant das Profil- und Gruppenmodel, die Authentifizierung des Users, sowie die Fuseki und Skosmos Administration zu implementieren.

Durch eine Klausur am 13.7, sowie Arztterminen konnte die Deadline aus persönlichen Gründen nicht eingehalten werden.

Deadline 17.Juli

Das Ziel dieser Deadline war es das Backend fertig zu implementieren. Hierzu sollten die nicht fertigen Aufgaben der Deadline zuvor sowie zusätzlich die Suchfunktion und das Vocabulary- und Termmodel implementiert werden. Gleichzeitig wurde mit dem Templates Bauen der Gui begonnen.

Mit einem großen Teil des Backends dieser und der vorherigen Deadline konnte am 21.Juli abgeschlossen werden. Das Profile Model hat sich verspätet, da das Verschicken von

E-Mails nicht funktioniert hat. Dieses Problem wurde dann in einen separaten Task verschoben und Profile wurde als abgeschlossen eingetragen.

Deadline 21.Juli

Das Ziel dieser Deadline war es die Templates fertig zu machen, um dann das Backend mit dem Frontend verbinden zu können. Sowie die Migration von Vokabularen zwischen dem Dev- und Live Fuseki Server zu ermöglichen.

Die Templates waren pünktlich fertig. Die Migration der Vokabulare zwischen live und dev Fuseki Server war problematisch, da Dokumentation und tatsächliches Verhalten der Jena Fuseki Datenbank nicht übereinstimmten. Zudem ist die Handhabung des Triple Standards nicht trivial.

Deadline 26.Juli

Ziel dieser Deadline war es, die Views, die das Backend und Frontend verbinden, fertig zu haben, so dass danach bis zur letzten Deadline vor allem Bugs und fehlende Funktionalität im Code behoben werden konnten.

Da niemand im Team etwas über Views wusste und wir es von Grund auf lernen mussten, verzögerte sich das Ende dieser Deadline um drei Tage.

Deadline 31.Juli

Das Ziel dieser Deadline war es das Projekt abzuschließen.

Die Webapplikation wurde mit allen Muss- und Sollkriterien implementiert. Es sind vermutlich noch Fehler zu beheben, dies ist für die Qualitätssicherungsphase vorgesehen.

To-Do

Name	Assign	Deadline	Priority	Status	created by	criteria	donedone
Merge-Only Master branch	Jonas Scholz	@July 8, 2021	Low	Completed	Jonas Scholz		@July 8, 2021
Django Test Struktur aufsetzen	Jan Maly	@July 11, 2021	Medium	Completed	Jonas Scholz	Muss	@July 11, 2021
Docker-Compose (dev) Setup	Jonas Scholz	@July 11, 2021	High	Completed	Jonas Scholz	Muss	@July 11, 2021
Projekt Struktur aufsetzen	Yannis Preker Klara Eckhardt	@July 11, 2021	Low	Completed	Jonas Scholz	Muss	@July 11, 2021
TailwindCSS aufsetzen	Jonas Scholz	@July 11, 2021	High	Completed	Jonas Scholz	Muss	@July 11, 2021
Overleaf Projekt für Implementierung aufsetzen	Klara Eckhardt	@July 14, 2021	Low	Completed	Klara Eckhardt	Muss	@July 14, 2021
Skosmos Administration	Klara Eckhardt Jonas Scholz	@July 14, 2021	High	Completed	Jonas Scholz	Muss	@July 14, 2021
Group Model	Jan Maly	@July 14, 2021	Medium	Completed	Jonas Scholz	Muss	@July 16, 2021
Fuseki Administration	Jonas Scholz	@July 14, 2021	High	Completed	Jonas Scholz	Muss	@July 16, 2021
Custom User Authentifizierung	Yannis Preker	@July 14, 2021	Low	Completed	Jonas Scholz	Muss	@July 17, 2021
Profile Model	Jan Maly	@July 14, 2021	High	Completed	Jonas Scholz	Muss	@July 18, 2021
Comment + Tag Model	Umut Ulaş Cin	@July 14, 2021	Medium	Completed	Jonas Scholz	Muss	@July 21, 2021
Triple Interface	Umut Ulaş Cin	@July 14, 2021	High	Completed	Jonas Scholz	Muss	@July 21, 2021
Search Model	Jan Maly	@July 17, 2021	Medium	Completed	Jan Maly	Muss	@July 17, 2021
Vocabulary Model	Yannis Preker	@July 17, 2021	High	Completed	Jonas Scholz	Muss	@July 21, 2021
Notification Model	Umut Ulaş Cin	@July 17, 2021	Medium	Completed	Jonas Scholz	Muss	@July 21, 2021
Searchable und Criterion Interfaces	Jan Maly	@July 17, 2021	Medium	Completed	Jan Maly	Muss	@July 21, 2021
Suchkriterien - entfallen	Umut Ulaş Cin	@July 17, 2021	Low	Completed	Jan Maly	Muss	
Templates bauen	Klara Eckhardt Jonas Scholz	@July 21, 2021	High	Completed	Jonas Scholz	Muss	@July 21, 2021
Live Migration	Jonas Scholz	@July 21, 2021	Medium	Completed	Jonas Scholz	Muss	@July 23, 2021
Term Model	Yannis Preker	@July 26, 2021	High	Completed	Jonas Scholz	Muss	@July 26, 2021
Vocabulary Settings	Jonas Scholz Yannis Preker	@July 26, 2021	Medium	Completed	Jonas Scholz	Muss	@July 28, 2021
Profile View	Jan Maly Klara Eckhardt	@July 26, 2021	High	Completed	Klara Eckhardt	Muss	@July 28, 2021
Teams List	Jan Maly Klara Eckhardt	@July 26, 2021	High	Completed	Klara Eckhardt	Muss	@July 29, 2021

Name	Assign	Deadline	Priority	Status	created by	criteria	donedone
🤼 Vocabulary Members	Jonas Scholz Yannis Preker	@July 26, 2021	High	Completed	Jonas Scholz	Muss	@July 29, 2021
🏺 Teams Detail	Klara Eckhardt Jan Maly	@July 26, 2021	High	Completed	Jonas Scholz	Muss	@July 29, 2021
🔎 Vocabulary Detail	Jonas Scholz Yannis Preker	@July 26, 2021	High	Completed	Jonas Scholz	Muss	@July 30, 2021
🤺 Term View	Yannis Preker	@July 26, 2021	High	Completed	Jonas Scholz	Muss	@July 31, 2021
📝 Vocabulary List	Jan Maly Klara Eckhardt	@July 26, 2021	High	Completed	Klara Eckhardt	Muss	@August 1, 2021
🤳 Login / Signup / PW Reset View	Umut Ulaş Cin	@July 26, 2021	High	Completed	Jonas Scholz	Muss	@August 1, 2021
Terms of Services Seite	Klara Eckhardt	@August 1, 2021	Low	Completed	Klara Eckhardt	Muss	@August 1, 2021
🧹 Migration Fuseki	Jonas Scholz	@August 1, 2021	High	Completed	Klara Eckhardt	Muss	@August 1, 2021
Vocabulary Terms	Jonas Scholz Yannis Preker	@July 31, 2021	High	Completed	Jonas Scholz	Muss	@August 1, 2021
Daten export/mailing	Umut Ulaş Cin Jan Maly	@August 1, 2021	Medium	Completed	Jan Maly	Muss	@August 1, 2021
Task					Jonas Scholz		

5 Code Metriken

Software Zusammensetzung und Lines of Code

Das Git Repository enthält 21173 Zeilen Code. Hierbei sind die importierte Guardian Bibliothek und die durch Django autogenerierte Datein mit einberechnet. Es setzt sich folgendermaßen zusammen:

Sprache	Anteil in Prozent
Python	62,1
HTML	36,7
JavaScript (TypeScript)	1,0
Dockerfile	0,1
CSS	0,1

Unittests

Die einzelnen Klassen des Backends sind mit Unittests versehen, diese werden erweitert und die Testüberdeckung maximiert. Zusätzlich wird die Integration der einzelnen Komponenten getestet, besonders ob unvorhersehbare Eingaben des Users zu Problemen führen.

Literatur

[1] *django-guardian - per object permissions for Django*. `https://django-guardian.readthedocs.io/en/stable/index.html`. [Online; Version: 2.4.0 accessed 10-July-2021].