OTH-Wiki

Dominik Smrekar, Johannes Horst, Patrick Sabau, Saniye Ogul und Tobias Schotter

I. EINLEITUNG

Ziel dieses Projektes ist es, eine Web-Anwendung zu entwickeln, die dem Nutzer dabei helfen soll, leichter an relevante Informationen der Technischen Hochschule Amberg-Weiden zu kommen. Da die offizielle Hochschulseite für neue und alte Studenten unübersichtlich ist, gehen viele signifikante Informationen unter. Manche finden die Links zu den Online-Portalen nicht und andere werden mit Informationen bombardiert. Dies führte zur Idee eine Webseite für die OTH-AW zu erstellen, indem Informationen einfach und strukturiert für Studierende von Studierenden angesammelt werden.

II. VERWANDTE ARBEITEN

Als primäre Inspiration dient die freie Enzyklopädie Wikipedia, da sie eine übersichtliche und saubere Möglichkeit der Informationsdarstellung bietet. Hierbei sind das Design und die Struktur höchst vertraut, da jeder schon in irgendeiner Form mit Wikipedia in Kontakt getreten ist.

Darüber hinaus inspiriert uns die Software Confluence, aufgrund der Wikipedia artigen Strukturierung von Nutzer priorisierten Informationen.

Abschließend ist die Internetseite der Hochschule Amberg-Weiden selbst zu erwähnen, da sie unter anderem Informationen für Studierende bereitstellt. Jedoch zeigen sich Schwächen bzw. fehlende Möglichkeiten der Informationsbereitstellung von Studierenden für Studierende.

III. ANFORDERUNGEN

Grundlegend lassen sich die Ziele der Projektarbeit in 5 Abschnitte untergliedern. Hierzu gehört das Frontend (A), Backend + Datenbank (B) und das Deployment (C). Weitere optionale Features sind ebenfalls unter den entsprechenden Punkten zu finden, jedoch sind einige dieser auch in einem separaten Unterpunkt (D) aufgelistet. Die Anforderungen werden durch den Punkt Testabdeckung (E) abgerundet.

Zur Identifikation von Anforderungen werden zuerst die User Stories für die Projektarbeit betrachtet.

Das führt zum Ziel, das Projekt ausführlich anzupassen und den Integrationsaufwand so gering wie möglich zu halten. Für die Anforderungen wurden die Sicht und die Wünsche des Benutzers in Bezug auf OTH-Wiki beschrieben.

A. *Frontend*

Als Nutzer möchte ich, dass mir die Texte der Artikel angezeigt werden, um aus ihnen die gewollten Informationen entnehmen zu können.

Als Nutzer möchte ich neue Texte schreiben, oder bestehende Texte bearbeiten können, um fehlende Infos hinzuzufügen oder falsche Aussagen nachbessern zu können.

- Beim Betätigen des „Edit-Buttons“ gibt es ein Freitextfeld zum Bearbeiten des Textes

Als Nutzer möchte ich mithilfe der Suche passende Artikel finden, wodurch Zeit des selber Suchens gespart wird .

- Schlüsselwörter Suche

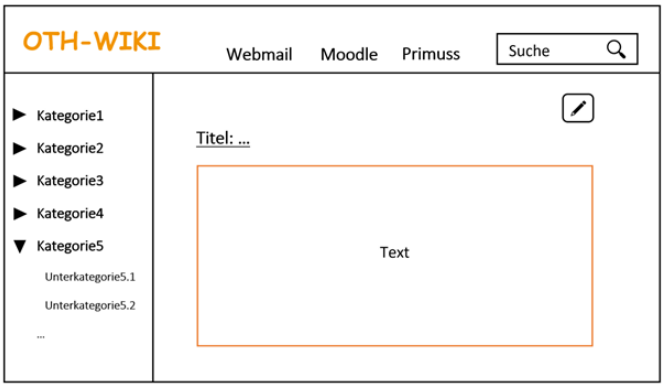

Fig. 1. Konzeptzeichnung Frontend

Abbildung 1 zeigt eine Konzeptzeichnung des Frontends. Diverse Features sind in diesem zu erkennen:

- Kategorien sind aufklappbar
- Webmail, Moodle und Primuss sind Links
- Durch Suche können Artikel mit Schlüsselwörtern gesucht werden
- Titel und Text werden dynamisch angezeigt
- Texte können bearbeitet werden, durch „Edit-Button“

B. *Backend + Datenbank*

Als Entwickler möchte ich eine Datenbank, welche nur vom Backend (welches eine REST-API bereit stellt) angesprochen werden kann. Folgende Features sind gefordert:

- Es soll die Möglichkeit bestehen, Texte als Website-Inhalt zu schreiben (Markdown Formatierung)
- Artikel sollen aus der Datenbank über die REST-API abrufbar sein
- Geänderte Artikel sollen in der Datenbank gespeichert und zusätzlich versioniert werden.
- Es soll eine hierarchische Suche geben, welche in der Datenbank nach Keywords in Texten oder Tags etc. sucht.
- *Optional:* Weiter ist es möglich, die Artikel mit entsprechenden Bildern zu ergänzen

- *Optional:* Eine Kommentarfunktion erlaubt Nutzern das Hinzufügen von Kommentaren und/oder Fragen zu speziellen Inhalten.

C. Deployment

Jede der Teilkomponenten wird in einem eigenen Docker Container gepackt. Mittels docker-compose können diese Container gleichzeitig gestartet und gemanaget werden.

- *Optional:* Der Container werden zusätzlich in einen Kubernetes-Cluster deployed

D. optionale Ziele

- *Optional:* Nutzer haben die Möglichkeit, Fragen an alle aktiven Nutzer zu stellen. Entweder über einen eigenen Fragen-Artikel oder einen dafür spezialisierten Abschnitt der Seite.
- *Optional:* Nutzern wird angeboten, Bilder/Dateien auf der Plattform auszutauschen/hochzuladen.

E. Testabdeckung

Als Entwickler möchte ich eine ausreichende Testabdeckung, damit Fehler frühzeitig erkannt werden. Akzeptanzkriterien sind:

- Die Code-Qualität jeder Komponente wird durch Unit-Tests gewährleistet
- Die Code-Coverage liegt bei mindestens 50%.

IV. ARCHITEKTUR

Teile der Architektur wurden bereits in Abschnitt II. erwähnt, jedoch werden diese nochmal detaillierter spezifiziert.

A. Frontend

Das Frontend wird mittels des Framework **Angular** entwickelt. Dieses setzt auf NodeJS sowie auf die Programmiersprache TypeScript, welche JavaScript um statischen Datentypen ergänzt. Angular kann über einen speziellen *build*-Befehl eine einzelne **.html** Seite generieren, welche anschließend mittels eines Http-Servers geserved werden kann. Diese Aufgabe übernimmt **nginx**. Der zugehörige Docker-Container besteht dabei aus zwei Teilen, einem Build-Container und einem Application-Container. Diese beiden übernehmen dabei den **Build** und den **Serving** Schritt.

Nginx hat neben dem Serving der .html-Seite noch zwei zusätzliche Aufgaben:

- SSL-Verschlüsselung mittels Zertifikate von **Let's Encrypt**
- Proxy für die Backend-Applikation zur Vermeidung von CORS-Fehlern

B. Backend

Das Backend wird mittels FastAPI und Python umgesetzt. Die Hauptaufgabe des Backends besteht es darin, das Datenmodell aus Abbildung 2 in einer MongoDB Datenbank zu verwalten.

Kategorien dienen zur Gruppierung der einzelnen Artikel und können auch ineinander verschachtelt werden. Jeder Artikel ist einer Kategorie zugeordnet und enthält dabei die

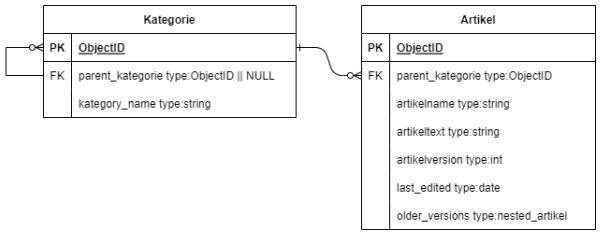

Fig. 2. Datenmodell

Daten, welche den Artikel ausmachen (in Abbildung 2 ist nur ein kleiner Teil der Inhalte dargestellt).

Das Backend nutzt REST für die Kommunikation mit dem User. Die wichtigsten Endpoints sind dabei:

- *GET* /kategorie
- *GET* /artikel/<id>/<version>
- *POST* /artikel

Die beiden **GET** Endpoints dienen dabei der Informationsdarstellung im Frontend, wohingegen der *POST* (oder auch **PUT**) Endpoint entsprechende Änderungen in der Datenbank persistiert.

C. docker-compose

Docker-compose bietet eine Möglichkeit, mehrere Docker Container zu managen und diese gleichzeitig zu starten. Entsprechend werden die einzelnen Teilkomponenten mit diesem Tool orchestriert.

D. Interaktion des Users

Abbildung 3 zeigt, wie ein User mit der Anwendung interagiert.

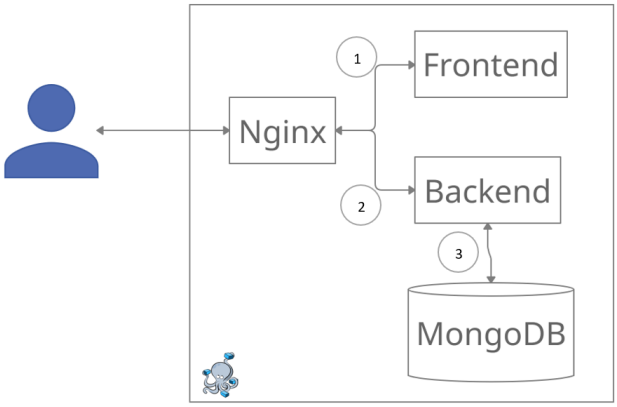

Fig. 3. Konzept Architektur

Der User erhält mittels einer GET-Anfrage das Frontend von dem nginx-Server (Schritt 1). Standardmäßig werden Inhalte von Angular Seiten clientseitig angefragt. Dies bedeutet, es erfolgen weitere REST-Calls des Users (über JavaScript Code der Website) über den nginx-Proxy an den Backend-Service (Schritt 2). Dieser Service, ist die einzige Verbindung zur Datenbank. Nach Verarbeitung der Anfrage (Schritt 3), wird eine entsprechende Antwort (z.B. mit dem Inhalt des Artikel) an den User zurückgesendet.