Entwurfsbeschreibung

Projektleitung: Otto Bittner
30.01.17
GBAW-17

1 Allgemeines

1.1 Docker Umgebung

Da an der Entwicklung verschiedene Entwickler mit verschiedenen Arbeitsumgebungen beteiligt sind, wird eine Docker Umgebung genutzt, in der alle nötigen Werkzeuge und Konfigurationen zusammengefasst sind.
Aktuell werden drei Container benutzt: ‘backend’ für Django und seine Abhängigkeiten, ‘db’ für PostgreSQL und ‘apache’ für einen Apache Webserver. Die Einrichtung und Konfiguration unterscheidet sich innerhalb der Container nicht von der auf einer Linux Distribution. Die Datei ‘docker-compose.yml’ koordiniert hier die Erstellung der Container und verweist auf die jeweiligen docker-Dateien, welche konkrete Build Anweisungen enthalten oder vorgefertigte Container aus dem offiziellen docker-Repository laden. Sofern ‘docker-compose.yml’ dies nicht tut, verweisen die docker-Dateien auf evtl. vorgefertigte Konfigurationsdateien.
Im Anhang befindet sich eine kurze Aufschlüsselung der notwendigen Schritte um die Umgebung lokal zu starten.

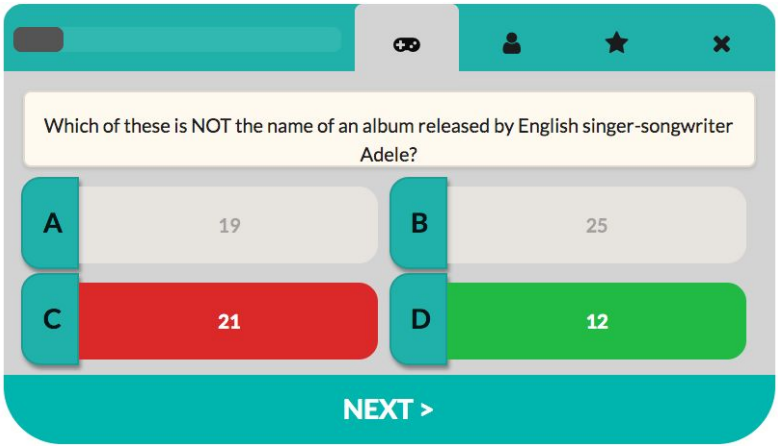
1.2 Dokumentation

Eine Dokumentation des JavaScript Code ist im HTML Format beigelegt. Die Dokumentation des Python Code kann im Admin Interface (s. 7.1 Einrichtung Dockerumgebung) unter “Documentation” eingesehen werden.

2 Produktübersicht

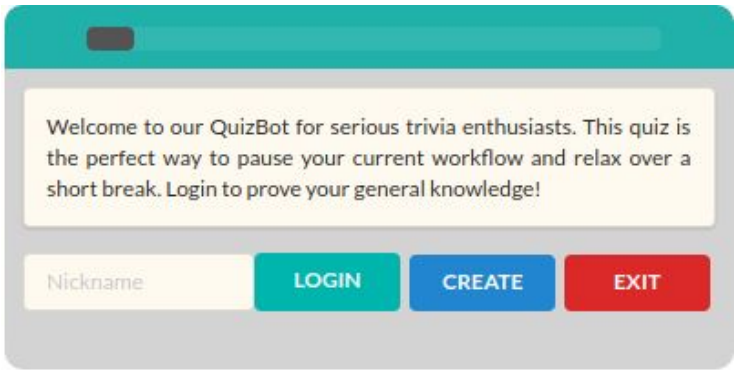
Im Folgenden werden die Funktionen der Software anhand von Mockups geschildert. Die aktuell implementierten Funktionen sind unter 2.6 zu finden.

2.1 Stellung & Beantwortung von Fragen



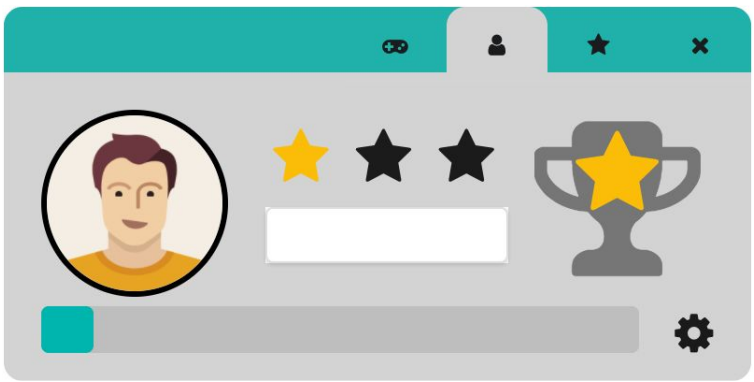
Der Benutzer bekommt Fragen aus der Datenbank gestellt. Bevor der Benutzer eine Quizrunde starten kann, wird dieser in einem Dimmer-Dialog aufgefordert den Timer für die erste Frage zu starten. Bei Bestätigung startet im linken oberen Bereich des Fensters ein Ladebalken, welcher ein Zeitlimit für die Beantwortung der gestellten Frage signalisiert. Nach dessen Ablauf wird die Frage als falsch gewertet. Mit einem Klick auf eine der vier Antwortmöglichkeiten wird die Antwort durch den Server verifiziert. Ziel ist es so zu verhindern, dass der Nutzer die korrekte Frage aus der Browserkonsole ausliest. Bei einer falschen Auswahl wird die Richtige durch eine grüne Markierung gekennzeichnet. Die eigene Antwort wird rot unterlegt. Mit einem Klick auf den “NEXT >”-Button wird die nächste Frage geladen.

2.2 Benutzeridentifizierung



Bei jedem Start der Applikation wird ein Willkommens-Fenster angezeigt. Der Benutzer hat die Möglichkeit sich in ein vorhandenes Profil einzuloggen oder ein neues anzulegen. Der Server verifiziert den eingetragenen Benutzernamen und leitet bei Erfolg weiter zum unter 2.1 erläuterten Fragen-Fenster. Am oberen Rand des Fensters läuft ein Timer. Sofern in dieser Zeit keine Reaktion des Benutzers registriert wird, schließt die Applikation automatisch. Mit einem Klick auf den “EXIT”-Button wird die Applikation ebenso beendet.[1]

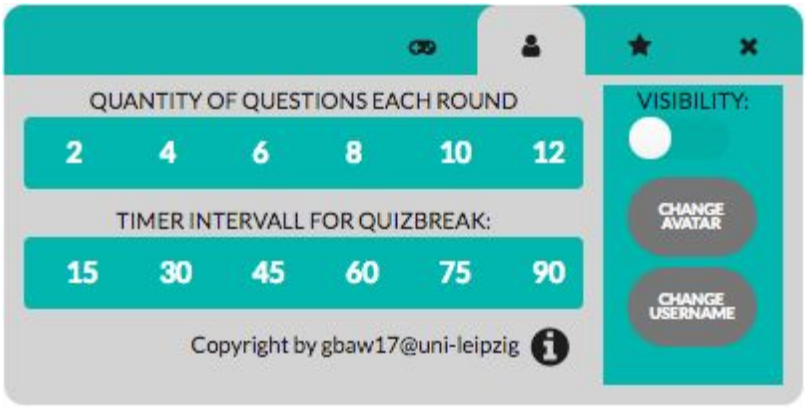
2.3 Benutzerprofile



Im Benutzerprofil wird nach einer abgeschlossenen Quizrunde der Name des aktuell angemeldeten Benutzers angezeigt. Statistiken zu den bereits beantwortenden Fragen, Erfolge und Belohnungen werden außerdem im Benutzerprofil eingesehen.

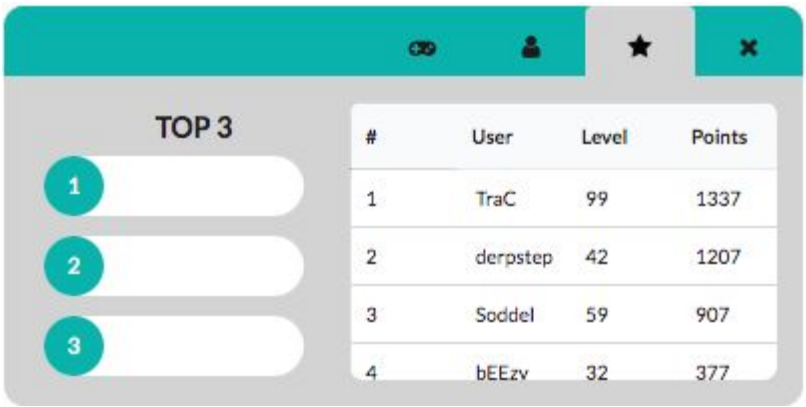
[1] Mit dem finalen Release muss sich der Nutzer nun schon auf der Rahmenwebseite einloggen und nicht mehr im Bot selbst. Er kann sich im Startbildschirm entscheiden ob er spielen möchte oder zu seinem Profil.

2.4 Einstellungen



Hier sind Einstellungen für die Anzahl von Fragen, die Arbeitszeit zwischen Fragerunden und die persönliche Konfigurationen des Benutzerprofils verfügbar.

2.5 Ranking



Das Ranking-Fenster stellt zu allen registrierten Profilen eine Übersicht zu Verfügung. Hierbei werden gesammelte Punkte durch beantwortete Fragen als Metrik benutzt, sodass die Benutzer ihre Leistungen vergleichen können.

2.6 Changelog

Im Prototyp enthalten:

- 2.1 - Zeitlimit noch nicht implementiert
- 2.2 - Zeitlimit und Exit-Button noch nicht implementiert
- 2.3 - Lediglich der Name des aktuellen Benutzers wird angezeigt

In Release 3 hinzugekommen:

- Bugfix für die Auswahl der Fragen
- Zeitlimit für 2.1
- Grundstruktur für Timer um Arbeitsintervalle zu messen

In Release 4 hinzugekommen:

- Timer um Arbeitsintervall zu messen.
  Hier sind noch einige Fehler vorhanden. Diese sind als Issues im GitLab vermerkt und die Behebung für das nächste Release vorgesehen.
- Einbettung des Bot in eine Rahmenseite mit Hilfe eines iFrames.
  Eine Spezifikation für die Einbettung ist im nächste Release geplant, da mit den Fehlerbehebungen evtl. noch Änderungen stattfinden.
- Anpassungen im Backend um Timer zu realisieren

In Release 5 hinzugekommen:

- Zahlreiche Bugfixes bei der Messung des Zeitintervall (Timer)
- Testcode für Frontend Timer
- Zeitlimit im Willkommensbildschirm wird nun durch Fortschrittsleiste visualisiert.
- Benutzereinstellungen können nun geändert werden
- Mokup für Rankingansicht

Im Final Release hinzugekommen:

- Feld um den Benutzernamen zu ändern
- Punktemultiplikator falls mehrere Fragen in Folge richtig beantwortet werden
- Ansicht um Statistiken zu betrachten
- “Schließen”-Knopf um den Bot auch während dem Spielen beenden zu können
- Ansicht um den Nutzer auf die abgelaufene Pause hinzuweisen
- Auswertung der Testabdeckung für Front- und Backend Tests
- Ausführliche Dokumentation von Front- und Backend

3 Grundsätzliche Struktur- und Entwurfsprinzipien

Das System ist in die Bereiche Frontend und Backend aufgeteilt. Diese Unterteilung hat sich in der Praxis bewährt und erlaubt es die Bereiche Darstellung und Datenverwaltung sauber zu trennen. Mithilfe einer fest definierten Schnittstelle zw. beiden Komponenten können Bereiche unabhängig voneinander entwickelt werden.

3.1 Frontend

Pro Ansicht bündelt sich der Code in jeweils eine HTML, eine CSS und eine JavaScript Datei. Bisher ist in den Skripten noch keine Logik enthalten. Das Skript beschränkt sich darauf Ajax Anfragen an den Server zu senden und Weiterleitungen zwischen den verschiedenen Ansichten zu tätigen.
Das mit R3 eingeführte Zeitlimit ist eine periodische DOM-Manipulation mit zeitlicher Limitierung. Die enge Verflechtung mit Funktionen für Benutzerinteraktionen verhindert es, dass diese Funktionalität sinnvoll ausgelagert werden kann.

3.2 Backend

Der serverseitige Code folgt den Vorgaben des benutzten Webframework Django[2] bzw. Django REST Framework[3] (kurz: DRF). Die Architektur dieser Frameworks ist in der jeweiligen Dokumentation ausführlich beschrieben.
Vorallem durch die schnell zu implementierende Benutzerverwaltung und Serialisierung wurde das DRF gewählt.
Das Datenbankschema wird in Django durch Models abgebildet. Um diese in die jeweilige Datenbank einzufügen werden durch Django Migrations angelegt, diese müssen bei Änderungen an den Models erzeugt und dann in die Datenbank migriert werden.
Um Anfragen und Antworten in einem einheitlichen Format zu halten werden Serializer genutzt, welche JSON-Objekt vom Client erhalten bzw. für diesen erzeugen.
Die Logik wird in den Views abgebildet. Eine View Klasse stellt einen Endpunkte der API dar.
Pro View Klasse können die HTTP-Anfragemethoden (GET, POST, PATCH, DELETE) definiert werden. Geschäftslogik wird an dieser Stelle definiert. Zusätzlich benötigte Methoden werden vorerst in der Datei helpers.py gesammelt.
Unter welcher Adresse die Endpunkte angesprochen werden wird an einer zentralen Stelle, in der urls.py, abgespeichert.
Um Funktionalitäten bereitzustellen, die durch Entwickler manuell ausgelöst werden, oder täglich zu bestimmten Uhrzeiten ausgeführt werden müssen, können Management Befehle im Ordner “management” angelegt werden. Bisher wird ein Befehl zum Laden von Fragen aus der Open Trivia API genutzt.

[2] https://www.djangoproject.com/t
[3] http://www.django-rest-framework.org/

4 Struktur- und Entwurfsprinzipien einzelner Pakete

4.1 Benutzerauthentifizierung durch JWT

Um dem Bennutzer passende Fragen anzuzeigen bzw. ihm sein eigenes Benutzerprofil oder seine Einstellungen zu zeigen, muss er nach dem Login vom Server identifiziert werden.
Hierzu wird der JSON Web Token (JWT) Standard genutzt. Django REST unterstützt diesen Prozess mithilfe einer zusätzlichen Library.
Beim Login wird mithilfe des Benutzernamens und Passworts ein Token erzeugt. Der Token wird außerdem mit dem RSA (Rivest, Shamir und Adleman)[4] Private Key des Server signiert. Der Client kann dann mithilfe des Public Keys die Herkunft des Tokens überprüfen.
Zur Anfrage von geschützten Endpunkten muss der Client bei jeder Anfrage im HTTP Authorization Header den Token, den er beim Login erhalten hat, mitliefern. Bei einem gültigen Token wird der Request beantwortet. Nach 8 Stunden oder bei einem neuen Login wird der alte Token ungültig.
DRF kann aus dem Token den Benutzer serialisieren.

4.2 Allgemeines zum Zeitlimit

Ursprünglich sollte der Timer mithilfe von Django-Channels realisiert werden. Diese ermöglichen durch WebSocket eine bidirektionale Verbindung zwischen Client und dem Server. Somit kann das Request-Response Schema überwunden werden.
Während der Gestaltung des Zeitlimit pro Frage bzw. dem Timer, der die Arbeitszeit/Pausenzeit misst stellte sich heraus, dass die Struktur von Django für diesen Zweck nicht optimal ist. Es wäre eine Infrastruktur notwendig, welche im Falle eines Zeitablaufs Events feuert und verarbeitet.
Die nun implementierte Alternative weist eine niedrigere Komplexität auf und ist für die Problemdomäne deutlich passender.

[4] https://de.wikipedia.org/wiki/RSA-Kryptosystem

4.3 Zeitlimit für Fragestellungen

Pro Frage existiert nun ein Zeitlimit. Nach 15 Sekunden wird eine gestellte Frage automatisch mit einer falschen Antwort gewertet und die richtige Antwort angezeigt. Anhand einer Fortschrittsleiste kann der Benutzer die noch verbleibende Zeit zur Beantwortung der Frage einsehen.
Die notwendigen Funktionalitäten bieten JavaScripts setTimeout() und setInterval(). Durch eine periodische Erneuerung der Prozessleiste wird der Timer simuliert. Dieser Prozess wird nach dem eingestellten Zeitlimit automatisch abgebrochen. Eine Benutzerinteraktion vor Ablauf der Zeit stoppt ebenso die Veränderung der Prozessleiste. Reagiert der Benutzer nicht wird eine leere Antwort ans Backend geschickt, was entsprechend als falsche Antwort gewertet wird. Die richtige Antwort wird farbig markiert.

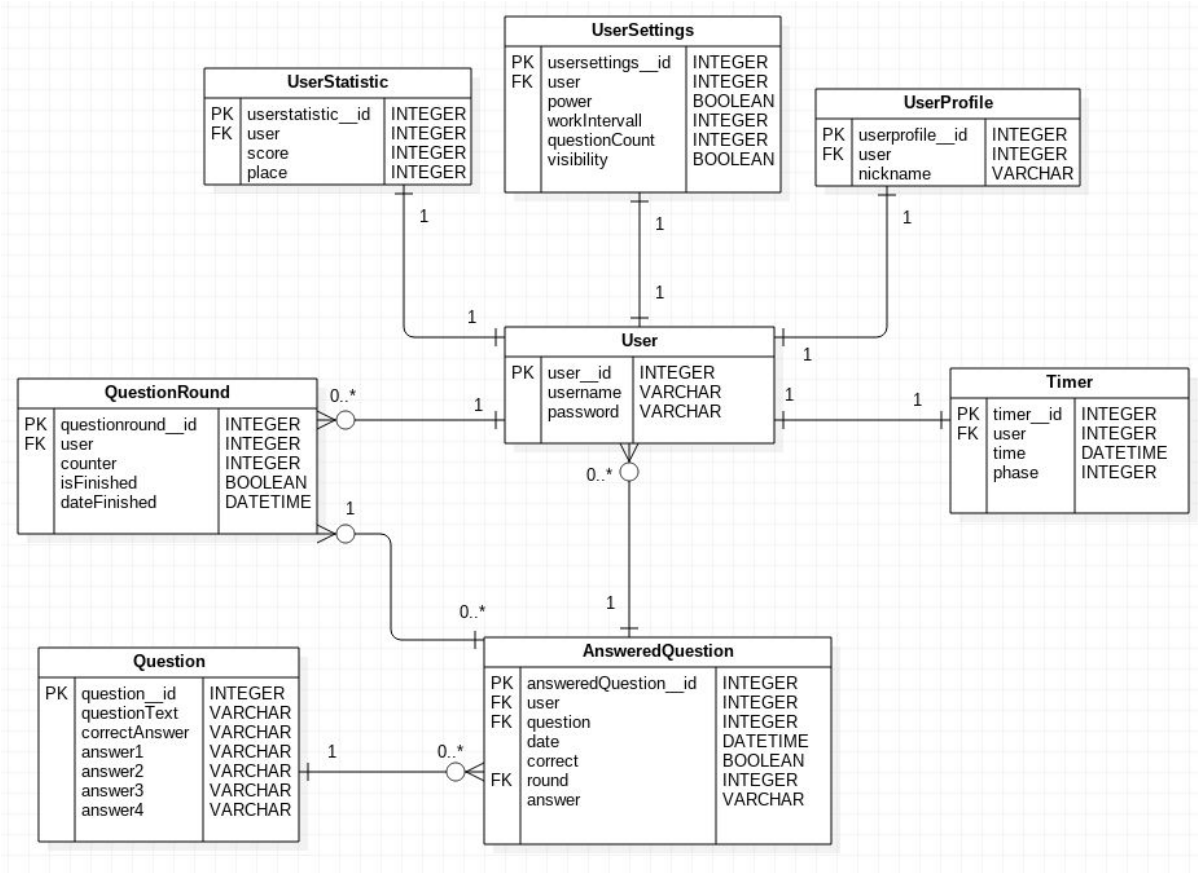
4.4 Zeitmessung der Arbeitsintervalle

Um das Pomodoro Prinzip mit dem Bot umsetzen zu können ist eine Zeitmessung notwendig. Diese wird realisiert indem die Zeit des Clients zu bestimmten Zeitpunkten (Login, Interaktion im startView, Änderungen in den Einstellungen) auf dem Server abgespeichert wird. Durch das Laden dieser Zeit und einer zusätzlich abgespeicherten Phase, die wiedergibt ob der Benutzer gerade arbeitet, pausiert oder im startView des Bots ist, kann genau bestimmt werden, wann die nächste Phase für den Benutzer beginnt.
Auf dem Server wird der Beginn einer neuen Phase als String mit Datum und Uhrzeit abgelegt. Anhand des Phasennamen wird eine feste Länge des aktuellen Intervallabschnitts bestimmt und daraus Uhrzeit und Datum für das Ende der Phase ausgerechnet. Durch periodische Bestimmung der Zeit bis zum Endzeitpunkt wird der Timer realisiert.

4.5 Einbettung in Rahmenwebseite

Die Applikation ist mit Hilfe eines iFrames in eine Rahmenwebseite eingebettet.
Diese Variante bietet sich an, da der Quellcode des Bot und der einbettenden Webseite hier vergleichsweise klar getrennt sind. Somit kann besser vorgeführt werden wie die Applikation zu nutzen ist.

5 Datenmodell



Obige Modellierung stellt das aktuelle Schema der Datenbank dar. Im Verlaufe des Praktikums werden hier etwaige Erweiterungen und Änderungen stattfinden.
Die User Klasse ist durch Django vorgegeben und besitzt noch einige weitere Attribute, jedoch werden diese durch den Bot nicht genutzt, deshalb sind sie hier nicht aufgeführt.
Die Primary Keys der Klassen werden automatisch durch Django erzeugt.
AnsweredQuestions wird genutzt, um zu speichern welche Fragen ein Users bereits beantwortet hat.
QuestionRound stellt die gespielten Runden des Users dar, wobei normalerweise eine Runde pro Arbeitsintervall gespielt wird und somit aus 3-8 Fragen bestehen wird.
Mit der Question Klasse werden Fragen in der Datenbank abgespeichert.
Zu einem User Objekt werden auch je ein UserStatistic, UserSettings und UserProfile Objekt erzeugt. In diesen werden die zum User gehörenden Statistiken und Einstellungen gespeichert, wie die Platzierung, der aktuelle Punktestand bzw. die gewünschte Anzahl an Fragen pro Runde, die Arbeitszeit zwischen den Runden oder die Sichtbarkeit in der Wertungstabelle.
In der Time Klasse wird eine Uhrzeit und eine Phase gespeichert. Als Uhrzeit wird die Zeit des Clients zu Beginn einer neuen Phase abgespeichert. Ausgehend von dieser Zeit, den Benutzereinstellungen und der aktuellen Phase (Arbeiten, Pause, Login) kann dann verbleibende Zeit berechnet werden.

6 Glossar

6.1 WebApp

Eine Webanwendung (auch Online-Anwendung, Webapplikation oder kurz Web-App) ist ein Anwendungsprogramm nach dem Client-Server-Modell. Anders als klassische Desktopanwendungen werden Webanwendungen also nicht lokal auf dem Rechner des Benutzers installiert. Die Datenverarbeitung und -auswertung findet hauptsächlich auf einem entfernten Webserver statt. Das Ergebnis der Datenverarbeitung wird zur Anzeige oder Ausgabe an den lokalen Client Rechner des Benutzers übertragen (Thin Client). Genutzt wird eine Webanwendung dabei in der Regel über einen Webbrowser.[5]

6.2 Frontend/ Backend

Die Begriffe Frontend und Backend werden in der Informationstechnik an verschiedenen Stellen in Verbindung mit einer Schichteinteilung verwendet. Dabei ist typischerweise das Frontend näher am Benutzer, das Backend näher am System. Es gilt aber prinzipiell, dass das Frontend näher an der Eingabe und das Backend näher an der Verarbeitung ist. Bei Client-Server Anwendungen wird das auf dem Client laufende Programm als Frontend, das auf dem Server laufende als Backend bezeichnet.[6]

6.3 Application Programming Interface (API)

Eine Programmierschnittstelle, ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. Neben dem Zugriff auf Datenbanken oder Hardware wie Festplatte oder Grafikkarte kann eine Programmierschnittstelle auch das Erstellen von Komponenten der grafischen Benutzeroberfläche ermöglichen oder vereinfachen.[7]

[5] https://de.wikipedia.org/wiki/Webanwendung
[6] https://de.wikipedia.org/wiki/Front-End_und_Back-End
[7] https://de.wikipedia.org/wiki/Programmierschnittstelle

6.4 Framework

Ein Framework (Rahmenstruktur) ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird. Im allgemeineren Sinne bezeichnet man mit Framework auch einen Ordnungsrahmen. Ein Framework ist selbst noch kein fertiges Programm, sondern stellt den Rahmen zur Verfügung, innerhalb dessen der Programmierer eine Anwendung erstellt, wobei u. a. durch die in dem Framework verwendeten Entwurfsmuster auch die Struktur der individuellen Anwendung beeinflusst wird.[8]

6.5 Asynchronous JavaScript and XML (Ajax)

Im Gegensatz zu einem klassischen Modell einer Webanwendung, bei der nach einer Serveranfrage die komplette Seite neu erstellt wird, kann im Ajax Modell eine asynchrone Datenübertragung zwischen dem Browser und Server stattfinden. So wird nur ein bestimmter Teil der Website nach einer Abfrage geändert während der restliche Teil unverändert angezeigt wird.[9]

6.6 Document Object Model (DOM)

Document Object Model ist eine Spezifikation für HTML und XML die eine Struktur im Sinne der objektorientierten Programmierung darstellt. Ziel ist dabei auf die strukturierten Daten einfach und einheitlich zugreifen zu können.[10]

6.7 Docker

Docker ist eine Software, welche es erlaubt Anwendungen in Containern zu isolieren und mit hilfe von Betriebssystemvirtualisierung eine Trennung zwischen den Ressourcen des Rechnern erzielt. Somit wird auch die Möglichkeit geboten notwendige Konfigurationen über verschiedene Systeme hinweg beizubehalten. [11]

[8] https://de.wikipedia.org/wiki/Framework
[9] https://de.wikipedia.org/wiki/Ajax_(Programmierung)
[10] https://de.wikipedia.org/wiki/Document_Object_Model
[11] https://de.wikipedia.org/wiki/Docker_(Software)

7 Anhang

7.1 Einrichtung Dockerumgebung

Um die Docker Umgebung zu nutzen sind einige wenige Schritte notwendig.
Zum einen muss die Ordnerstruktur des Git Repositories bewahrt werden. Die Konfigurationsdatei für docker-compose muss also im Root Verzeichnis des Repos liegen, genauso wie der Ordner “docker”, welche alle Konfigurationen für die verschiedenen Container enthält.
Man führt nun im Repository den Befehl “docker-compose build” aus. Hierfür sind root-Rechte notwendig. Danach kann die Umgebung mit “docker-compose up” gestartet werden.
Im Anschluss muss noch die Datenbank erstellt und befüllt werden. Ersteres passiert indem im Container mit dem Namen db die SQL-Anweisung “create database gbaw17;” ausgeführt wird. Dieser Befehl muss als Postgres Nutzer ausgeführt werden.
Befüllt wird die Datenbank indem man im Container backend den Befehl “python manage.py migrate” ausführt. Um das Admininterface nutzen zu können ist ein Admin Nutzer notwendig. Dieser wird mit “python manage.py createsuperuser” erzeugt. [12]
Hiernach ist die Software auf localhost verfügbar. Das Admininterface ist unter “localhost:8000/admin” zu finden.

[12] https://docs.docker.com/engine/reference/commandline/exec/