Entwicklerhandbuch für das Jeopardy Quiz

Gruppe 3 des Capstone Projekts 2022/23

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1 Einleitung

Das nachfolgende Entwicklerhandbuch soll Ihnen den Einstieg in unser Projekt erleichtern. Wir entwickelten in Zusammenarbeit mit Ambient Digital eine Quiz-App, um Team Events zu unterstützen und das Teamgefühl zu stärken.

Unsere Quiz-App ist inspiriert von der Spielshow Jeopardy und die Quiz werden auch in dieser Form gespielt. Es macht also durchaus Sinn, dies an einem großen Bildschirm oder einer Leinwand zu spielen. Mehr zum Spielprozess ist im Benutzerhandbuch zu finden.

Das eigentliche Spielen des Quiz sollte im Vorhinein geplant werden, da für die Erstellung eines Quiz eine gewisse Zeit benötigt wird.

Die folgenden Kapitel dieser Dokumentation sollen eine Grundlage bieten, um an diesem Projekt weiterzuentwickeln. Wir erklären den Grundaufbau unseres Projekts und auch die Strukturen der einzelnen Seiten, die auf der Webseite zu sehen sind. Diese Webseiten sind online erreichbar und es kann sich prinzipiell jeder registrieren.

Es gibt sowohl für unseren Main Branch als auch für unseren Develop Branch eine Webseite, auf der die Applikation getestet werden kann.

Develop Branch: https://quizai-test.vercel.app/ Main Branch: https://quizai.vercel.app/

Die Zusammenarbeit für dieses Projekt mit Ambient Digital ist im Zuge eines Bachelor Moduls zustande gekommen. Dieses Modul belegten wir im Rahmen unseres Bachelors im Bereich Wirtschaftsinformatik am Cologne Institute for Information Systems (CIIS) im Kurs „Capstone Project Information Systems“ im Wintersemester 2022/2023.

1.2 Ziel

Das Hauptziel dieser Quiz-App ist es, die Teamkultur zu fördern und den Zusammenhalt im Team zu stärken. Deswegen sollte bei der Teamzusammensetzung auch darauf geachtet werden, wie die Teams gestaltet werden. Team Kultur ist sehr wichtig beim alltäglichen Zusammenarbeiten. Dadurch, dass sich die Fragen und die Kategorien des Quiz individuell auf das jeweilige Team anpassen lassen, kann so beim gemeinsamen Knobeln der Teamzusammenhalt gestärkt werden.

2. Entwicklung

In diesem Abschnitt wird näher auf die Entwicklung eingegangen.

2.1 Allgemeine Informationen

Die Entwicklung des Projekts fand hauptsächlich auf Entwickler eigenen Windows Geräten statt, allerdings wurde das Projekt erfolgreich auf MacOS Geräten aufgesetzt und getestet. Einer Entwicklung auf MacOS Geräten spricht daher nichts entgegen. Als Integrated Development Environment (IDE) wurde von den Entwicklern Visual Studio Code benutzt, das kostenlos auf der Seite visualstudio.microsoft.com zum Download verfügbar ist. Um das Projekt lokal zu bauen, wurde Docker benutzt, damit beim Starten des Projekts alle benötigten Packages in der gleichen Version installiert werden. Docker kann auf der Website docker.com kostenlos heruntergeladen werden. Als Version Control System (VCS) wurde Git eingesetzt und der Code in ein vom CIIS zur Verfügung gestelltes GitLab Repository gepusht.

Für das Frontend des Projektes wurde die JavaScript Library React.js eingesetzt, mit React Bootstrap für zusätzliche React Komponenten und Bootstrap als CSS-Styling Framework.

Für das Backend haben wir sowohl das Django Framework als auch das Django Rest Framework benutzt.

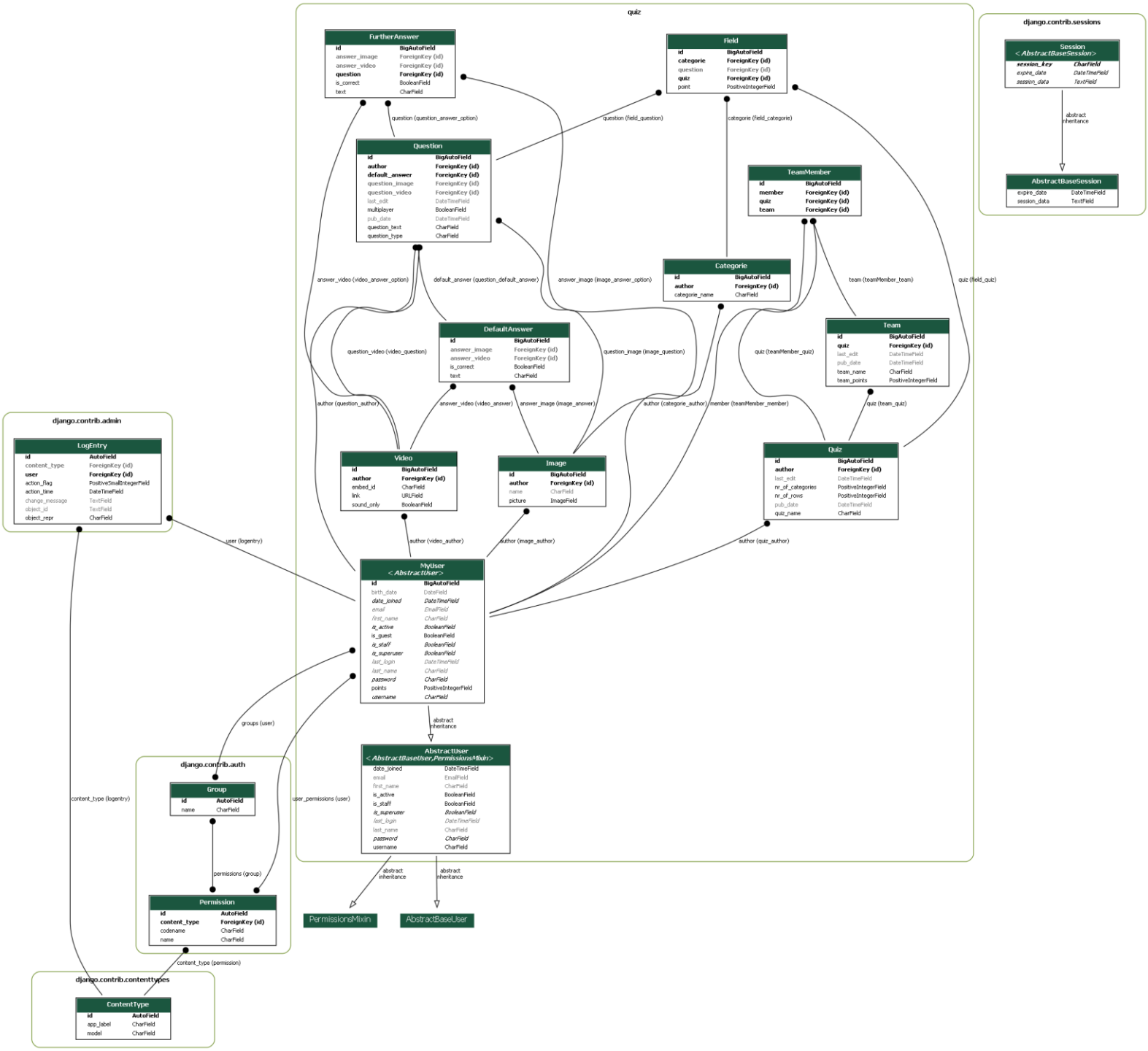

Abbildung 1: Django Datenbank Schema

Zum Testen der Applikation wird das Framework Playwright benutzt. Das End-to-End Testing-Framework Playwright ermöglicht das einfache Testen einer Web Applikation in mehreren Browsern.

2.2 GitLab Pipeline

Das Pushen eines oder mehrerer Commits auf GitLab löst eine CI/CD Pipeline aus. Ein Fail der Pipeline verhindert das Mergen eines Merge Requests. Die Pipeline besteht aus drei Stages, die hier genauer erklärt werden.

Build-Stage

Der Build-Stage besteht aus zwei Jobs: build_frontend und build_backend. Im jeweiligen Job wird das Frontend- und Backend-Image mithilfe von Docker gebaut und zwischengespeichert, um später weiter verwendet zu werden.

Test-Stage

Der Test-Stage besteht aus dem Job e2e_docker. In diesem Job werden zunächst erstmal die drei Docker Container ambient_database, ambient_backend und ambient_frontend gestartet. Anschließend wird das Playwright eigene Docker Image gestartet und die Tests auf diesem ausgeführt.

Deploy-Stage

Der Deploy Stage besteht aus vier einzelnen Jobs. Die Jobs werden abhängig vom Branch ausgeführt, auf den gepusht wurde. Alle Jobs benötigen, dass der Job e2e_docker ohne Fehler durchgeführt wurde.

Die Jobs backend_to_test und frontend_to_test werden nur ausgeführt, wenn auf den Develop Branch gepusht wurde. Im backend_to_test Job wird das Backend mit Hilfe von Heroku auf https://quizai-test.herokuapp.com/admin/login deployed. Im frontend_to_test Job wird das Frontend mit hilfe von Vercel nach https://quizai-test.vercel.app/ deployed.

Die Jobs backend_to_prod und frontend_to_prod werden nur ausgeführt, wenn auf den Main Branch gepusht wurde. Im backend_to_prod Job wird das Backend mit Hilfe von Heroku auf https://quizai-prod.herokuapp.com/admin/login deployed. Im frontend_to_prod Job wird das Frontend mit Hilfe von Vercel nach https://quizai.vercel.app/ deployed.

2.3 Organisation innerhalb des Teams

Anfangs haben wir uns in Backend und Frontend aufgeteilt. Uns ist jedoch schnell klar geworden, dass dies einige Nachteile mit sich bringt. Denn nicht jeder ist immer verfügbar und es sind bessere Lösungsansätze möglich für ein Problem, wenn man sowohl Backend als auch Frontend Änderungen in Betracht zieht. Aufgrund dieser Erkenntnisse haben wir nach ein paar Wochen von der besagten Trennung Abstand genommen und alle in der Funktion von Full Stack Entwicklern gearbeitet.

2.3.1 Merge Requests

Nachdem ein Feature bzw. eine Änderung im Code nach der Meinung des Entwicklers fertiggestellt war, haben wir einen Merge Request auf den Develop Branch gestellt. Danach hat ein anderer Entwickler sich diese Änderungen angeguckt und geprüft, ob dabei ungewollte Änderungen entstanden sind, der Code nachvollziehbar ist und ob die Änderungen die gewünschte Funktionalität liefern. War dies nicht der Fall, so wurde der Merge Request abgelehnt. Gab es nichts zu beanstanden, dann wurde der Feature Branch in den Develop Branch gemerged. Danach wurden die Product Owner (den Ansprechpartnern von Ambient Digital) über den Merge informiert, damit diese ein PO Review durchführen können.

2.3.2 PO Review

Die Product Owner haben dann die neue Funktionalität ausgiebig getestet und überprüft, ob alle Akzeptanzkriterien erfüllt sind. War dies nicht der Fall, so gab es dementsprechend Feedback von den Product Ownern und der Entwickler musste dieses Feedback nun integrieren. Danach wurde wieder ein Merge Request erstellt und dieser Kreislauf ging so lange weiter, bis die Product Owner zufrieden waren. Erst dann wurde das Ticket geschlossen und das Feature war vollständig implementiert.

3. Coding Conventions

3.1 Formatierung

Unser Code war zu Beginn nicht formatiert, dies führte zu sehr unübersichtlichem Code. Wir haben daraufhin in der Retro am 16.12 entschieden, dass wir ab sofort Dateien und Methoden entsprechend kommentieren und mit „Prettier“ und „Black“ den Code formatieren. Die Kommentare haben wir zunächst nur geschrieben, wenn wir an einer Datei sowieso gearbeitet haben. In der letzten Sprint Woche haben wir schließlich den Rest des Codes kommentiert. Dies sollte nochmal das Verständnis und die Orientierung im Code erleichtern.

3.2 Namenskonvention

Wir haben keine feste Namenskonvention festgelegt, sondern uns auf sprechende und selbsterklärende Namen geeinigt. In der Regel ist also anhand der Methoden- und Variablennamen abzuleiten, welche Funktionen diese haben. Besonders bei Methoden, die Daten aus dem Backend ziehen via Axios starten die Methodenname in der Regel mit „getxyz“. Dabei enthält der Name meist die Klasse, die in der folgenden Methode aus dem Backend gezogen wird.

3.3 Ordnerstruktur

Im Allgemeinen haben wir uns an eine Aufteilung in Frontend und Backend gehalten. Dabei haben wir als Backend ein Standard Django Project erstellt, welches Ambient-Digital heißt, mit der Unter-Applikation Quiz. Für das Frontend haben wir eine React-App erstellt und uns auch an die Standard Aufteilung der Ordner gehalten. Alle einzelnen Seitenansichten des Projektes befinden sich im Pages Ordner. Die Komponenten sind im Components Ordner zu finden und die Bilder im Public Ordner. Zusätzlich dazu befinden sich im Styles-Ordner einige CSS-Dateien, die jeweils für eine ganze Seite gelten. Zuletzt befinden sich die Playwright-Tests im Ordner „Tests“, diese sind jeweils entsprechend benannt und vollführen im Vorhinein immer eine Registrierung bzw. einen Login.

4. Einrichtung für Entwickler

4.1 Benötigte Tools

Die schon erwähnte IDE Visual Studio Code bietet sich am besten zur weiteren Entwicklung an, da wir diese auch genutzt haben. In VSC haben wir Extensions genutzt, um einheitlich zu Formatieren und um uns die Arbeit zu erleichtern.

Im Frontend haben wir die Extension „Prettier“ mit der Standardeinstellung und der Funktionalität „Format on save“ verwendet. Im Backend haben wir die erwähnte Extension „Black Formatter“ genutzt, um den Python Code zu formatieren.

Als eine der wichtigsten Tools wird Docker genutzt. Nachdem man Docker passend zu seinem Endgerät installiert hat, muss man drei Container bauen, in denen die einzelnen Teile des Projektes laufen. Nach der initialen Installation muss man einmalig folgenden Befehl in die Konsole eingeben.

$ git config --global core.autocrlf input

Danach kann man Docker mit folgendem Befehl starten:

$ docker-compose up

Der erste Container startet im Wesentlichen die Frontend Applikation, der zweite Container startet das Backend und der letzte Container startet eine lokale Datenbank. Es ist auch in der lokalen Version möglich mehrere Nutzer mit unterschiedlichen Quiz zu verwalten. Das Backend wird minimal vor dem Frontend hochgefahren, dennoch ist das Frontend meist vorher fertig und nutzbar. Damit das Projekt funktioniert, muss gewartet werden, bis alles hochgefahren ist. Dies ist durch die entsprechenden Befehle im Terminal zu sehen.

Gibt es bei der Benutzung von Docker ein Problem, sollte man als erstes versuchen, mit dem Befehl „Docker-compose up --build“ die Container neu zu bauen. Hilft dies nicht, ist der nächste Schritt in Docker selbst sämtliche Images, Volumens oder zwischengespeicherten Dateien zu löschen und dann nochmal die Docker Container neu bauen zu lassen.

Hat man die Container erfolgreich gebaut, so wurden auch alle benötigten Packages mit in diesen Containern installiert. Dies ist ein großer Vorteil von Docker und einer der Hauptgründe, warum wir Docker nutzen. Für jeden, der wissen möchte, welche Packages genau in den Container installiert werden, haben wir eine vollständige Auflistung im Anhang.

Als weiteres Tool sollte man Playwright installiert haben und wir empfehlen auch die Extension „Playwright test for VSCode“. Damit lassen sich Test einfach in VSC ausführen und auch sehr gut debuggen.

Zur Versionierung haben wir – wie schon erwähnt - GitBash genutzt. Dies in der Kombination mit GitLab hat unseren Organisationsaufwand maßgeblich verringert.

4.2 Projekt Setup

Um an unserem Projekt weiter zu arbeiten, muss, nachdem man alle benötigten Tools installiert hat, das Repository geklont werden und anschließend die Container hochgefahren werden. Nun ist auf localhost:3000 das Frontend und unter localhost:8000 das Backend erreichbar. Beim initialen Start sollte man im Backend Container folgenden Befehl eingeben:

$ python manage.py createsuperuser

Dieser Befehl erstellt einen Admin im Backend, der aber auch im Frontend zum Einloggen genutzt werden kann. Dabei wird man nach einer E-Mail-Adresse, einem Nutzernamen und einem Passwort gefragt. Mit diesen Daten kann man sich schließlich im Backend einloggen unter localhost:8000/admin. Im Frontend kann man sich einloggen unter localhost:3000/login .

4.3 Aufbau des Backend

Die Schnittstellen, welche durch das Backend zur Verfügung gestellt werden, sind über localhost:8000/api erreichbar. Auf dieser Seite findet man eine Übersicht über alle vorhandenen Schnittstellen, sowie eine kurze Erklärung zu dem Verwendungszweck der Schnittstellen unter “show

explanations". Diese Erklärungen können im Backend Ordner unter templates/rest_framework/api.html angepasst werden. Weiterhin kann man auch auf die einzelnen Schnittstellen zugreifen und, falls es bei der jeweiligen Schnittstelle möglich ist, GET, POST, PUT oder DELETE Anfragen an das Backend senden. Weitere Informationen zu den Schnittstellen und den Anforderungen an die jeweiligen Anfragen kann man auf localhost:8000/api/<Name der Schnittstelle> unter Optionen anfordern. Dies ist vor allem für Entwickler mit geringeren Erfahrungen mit dem DjangoRestFramework nützlich.

Die Schnittstellen sind im Backend unter ambientDigital/urls.py registriert. An dieser Stelle werden die Urls, mit Hilfe der "api views" (quiz/api_views.py), mit den "Serializern" (quiz/serializers.py) verbunden. Die "api views" beinhalten einen zugehörigen "Serializer", legen fest, welche Tabelle der Datenbank angesprochen wird und ermöglicht die Interaktion mit dem "Serializer" anzupassen. Das beinhaltet zum Beispiel das Einschränken der Art der Anfragen, die eine Schnittstelle entgegennehmen kann oder die Sortierung der Antworten auf GET-Anfragen. In den "Serializern" wird festgelegt, auf welche Felder der Tabelle zugegriffen wird, wie "Foreign Keys" integriert werden und, falls das Standard verhalten nicht gewünscht ist, wie bestimmte Anfragen die Daten in der Datenbank anpassen. Aber auch die Validierung der Registrierung und die für die Authentifizierung notwendigen Tokens werden über die "Serializer" konfiguriert.

Die Struktur der Datenbank wird in der Datei quiz/models.py festgelegt. Bei Veränderung der Struktur müssen neue "Migrations" erstellt werden. Dafür wird zunächst der Befehl "python manage.py makemigrations quiz" im Backend Container ausgeführt, welcher zur Erstellung einer neuen "Migration" führt und dabei das Beheben möglicher Komplikationen erfordert. Wenn diese "Migration" verwendet werden soll, so muss im Backend Container der Befehl "python manage.py migrate" ausgeführt werden.

Des Weiteren gibt es noch die Möglichkeit, das Aussehen und die Bearbeitungsmöglichkeiten des Django-Admins in der Datei quiz/admin.py anzupassen. Der Django-Admin ist über localhost:8000/admin erreichbar und ermöglicht Administratoren z.B. das Managen der Accounts der Nutzer oder Anpassungen an bestimmte Tabellen der Datenbank vorzunehmen.

4.4 Aufbau des Frontend

Das Frontend basiert auf der standardmäßigen Ordnung eines React Projektes. Jede Seite, die in der Applikation zu sehen ist, besitzt eine Datei im Ordner src/pages. Die dort importierten React Komponenten sind im Ordner src/components zu finden. Wir haben noch nicht alle Dateien im Pages Ordner in einzelne Komponenten zerlegt. Im src/context Ordner befindet sich die AuthContext Datei, welche für den Registrierungs- und Einlogprozess nötig ist.

Die einzelnen Dateien für die angezeigten Seiten basieren auf Javascript für den Funktionsteil und auf HTML bzw. CSS für den darstellenden Teil. Die benötigten Variablen und Konstanten werden immer zuerst definiert, gefolgt von den Methoden und schließlich dem HTML-Abschnitt. Modale oder ähnliches, also Code-Abschnitte, die nur nach Bedarf angezeigt werden, befinden sich immer am Ende einer Datei.

Die einzige Datei im src/pages Ordner, für die keine eigene Seite existiert, ist die Datei App.js, in welcher die Pfade definiert werden, welche zulässig sind. Es wird festgelegt, auf welche Seite diese Pfade weiterleiten sollen. Außerdem wird hier auch geprüft, ob diese Seite nur eingeloggten Usern zur Verfügung steht.

Für manche Seiten haben wir mehr individuellen CSS-Code benötigt. Diesen haben wir dann ausgelagert. Die zugehörigen Dateien heißen genauso wie die Javascript Dateien und befinden sich im src/style Ordner.

5. Nächste Schritte

Sollte ein Entwickler dieses Projekt weiterentwickeln, dann sollte im Vorhinein ein Refactoring stattfinden und die React-Seiten in weitere Komponenten zerlegt werden. Dies wäre jetzt ein guter Punkt dafür, da es ein lauffähiges, aber noch nicht zu komplexes Projekt ist.

Dieses Refactoring sollte des Weiteren auch die sorgfältige Überlegung einer einheitlichen Namens-Konvention umfassen, die ab diesem Zeitpunkt verpflichtend sein sollte. Außerdem sollte sorgfältig geprüft werden, ob durch das Hinzufügen neuer Funktionen nicht auch eine zweite Django Unter-Applikation sinnvoll ist, um mehr Struktur in das Projekt zu bringen.

Außerdem sollte eine Systemarchitektur aufgebaut werden, bevor das Projekt noch komplexer wird, und Axios sollte einheitlich in jeder Anfrage ans Backend genutzt werden

Weitere Funktionen wären zum Beispiel eine Profilseite und die Erweiterung zu einer White-Label Lösung. Mehr dazu ist auch auf unserem GitLab Board zu finden.

Weitere Tests würden der Applikation auch noch Mehrwert bringen und dem Programm mehr Sicherheit bieten. Dabei wäre es wichtig, sich nicht nur auf End-to-End-Tests zu beschränken, sondern auch Backend-Tests zu schreiben. Unit-Tests wären eine gute Möglichkeit, das Backend zu testen.

6. Anhang

6.1 Packages aus den Docker Containern

Package	Version	Genutzt für
@testing-library/jest.dom	5.16.5	Testing
@testing-library/react	13.4.0	Testing
@testing-library/user-event	13.5.0	Testing
axios	1.2.0	Promised based http-Client
bootstrap	5.2.3	Frontend
dayjs	1.11.6	Frontend
jquery	3.6.0	Frontend
jwt-dewcoder	3.1.2	Frontend
react	18.2.0	Frontend
react-bootstrap	2.6.0	Frontend
react-dom	18.2.0	Frontend
react-router-dom	6.4.4	Frontend
react-scripts	5.0.1	Frontend
react-select	5.7.0	Frontend
Web-vitals	2.1.4	Frontend

Packages	Version	Genutzt für
Django	4.1.3	Backend
Djangorestframework	3.14.0	Backend
Flake8	5.0.4	Backend
Django-cors-headers	3.13.0	Backend
Whitenoise	6.2.0	Backend
Djangorestframework-simplejwt	5.2.2	Backend
Pillow	9.3	Datenbank
Cloudinary	1.30.0	Datenbank
Django-cloudinary-storage	0.3.0	Datenbank
Psycopg2-binary	3.1.4	Deployment
Gunicorn	20.1.0	Deployment
Heroku		Deployment

6.2 Wichtige Kommandos

$ git config --global core.autocrlf input	Configuriert Linienend
$ docker-compose up --build	Baut alle 3 Docker Container und fährt diese hoch
$ docker-compose up	Fährt alle 3 Docker Container hoch
$ python manage.py createsuperuser	erstellt Admin im Django Backend
$ python manage.py makemigrations quiz	erstellt Migrations für Quiz Unter-Applikation
$ python manage.py migrate	führt erstellte Migrations aus