PROJEKTARBEIT

Thema:

AI-unterstütze Fitnessberatung

ABGABE BEI:

Herr Zimmermann

Abgabeort:

CSS – Neckarsulm

Abgabetermin:

23.12.23

Autoren: Youssef Sbai & Ramon Grothe

1 Inhaltsverzeichnis

2 Einleitung

2.1 Projektumfeld

Die folgende Projektdokumentation beschreibt die Entwicklung einer Consulting-Website zum Thema Fitnessberatung mit AI-Unterstützung. Dieses Schulprojekt entsteht im Rahmen der BT-S Aufgabenstellung rund um das Thema Frontend-Entwicklung. Es umfasst ein Frontend mit Nutzerinteraktion und ein Backend mit Datenspeicherung und API-Calls.

Im Verlauf dieser Dokumentation wird die Projektplanung und der Entwicklungsprozess der einzelnen Bausteine genauer untersucht.

2.2 Projektziel

Das Ziel dieses Projekts ist es, eine funktionierende Website mit den im nachfolgendem Pflichtenheft aufgeführten Features aufzusetzen. Darüber hinaus wollen wir verschiedene Projektmanagement-Methoden und Metriken kennenlernen, die wir für den Berufsalltag brauchen können.

2.3 Projektbegründung

Die Entscheidung für eine AI-unterstütze Beratungsplattform fiel, da wir unser bereits erlangtes Wissen mit „Large Language Models“ (kurz: LLM‘s) in dieses Projekt einbinden und dabei weiter ausbauen wollen. Auch ist es für uns eine interessante Herausforderung eine funktionierende Chat-Oberfläche zu erstellen.

2.4 Ist-Zustand

Im Rahmen der Aufgabe wird dieses Projekt von Grund auf neu entwickelt. Eine genaue Ist-Analyse ist daher zum Start des Projekts nicht möglich.

2.5 Soll-Zustand

2.5.1 Features

Das Endprodukt soll eine Landingpage mit Informationen zum Produkt und der Verlinkung zum Live-Chat beinhalten. Optional soll die Newsletter-Anmeldung über ein Modul möglich sein. Der Chat selbst soll den Verlauf der letzten Chats speichern. Außerdem ist der Chat bereits mit einem Systemprompt vorbelegt, welches über die Schnittstelle zu „OpenAI“ (*Unternehmen hinter ChatGPT und weiteren AI-Modellen*) direkt mitgegeben wird. So werden die Antworten gezielt auf die Fitnessberatung ausgelegt, ohne eine weitere Prompt-Bearbeitung vom Nutzer zu erfordern.

Der Source-Code und eine Schritt-für-Schritt-Installationsanweisung werden über GitHub bereitgestellt.

2.5.2 Projektrahmen

Zur besseren Eingrenzung haben wir eine Muss-Wunsch-Abgrenzungs-Einordnung erstellt:

- Muss:
    - Landingpage
    - Chat-Page
    - OpenAI-Anbindung zur LLM-Nutzung
    - Datenbank
- Wunsch:
    - Login-Maske mit User-Management
    - Newsletter-Anmeldung
- Abgrenzung:
    - Kein Deployment/Hosting

In der „Muss“-Kategorie sind die Themen, die sowohl von der Aufgabenstellung als auch von unseren eigenen Anforderungen da sein müssen.

Die Login-Maske mit User-Management und die Newsletter-Anmeldung sind mögliche Features, die wir in unsere Anwendung integrieren wollen, falls wir vor der geplanten Abgabe noch genügend Zeit zur Verfügung haben.

Vom Deployment oder sogar Hosting der Seite wollen wir absehen, da für die Nutzung der Anwendung ein OpenAI API-Key erforderlich ist. Durch jeden API-Call entstehen Kosten, welche wir möglichst einschränken wollen. Zur Bewertung nach der Abgabe werden wir jedoch eine passende Lösung finden.

3 Projektplanung

3.1 Technologieauswahl & Begründung

Bevor wir die genauen Arbeitsschritte ermitteln, entscheiden wir uns vorab für die Technologien.

Zur Entwicklung der Frontend-Module entscheiden wir uns für Vue.js, da wir bereits Vorerfahrungen aus einem Arbeitsprojekt mit diesem Framework haben. Für das Styling verwenden wir kein CSS-Framework und stylen die Elemente selbst mit SCSS. Dieses bietet neben dem Standard CSS noch Funktionen und Variablen an, um redundanten Code zu vermeiden.

Im Backend nutzen wir Python, da wir uns sowohl mehr mit dieser Programmiersprache vertraut machen möchten als auch die Entwicklung von AI-Projekten hier mit am besten unterstützt wird.

Zusätzlich dazu verwenden wir LangChain als LLM-Framework. Mit diesem Framework ist es sehr einfach gut strukturierte und wiederverwendbare Prompt-Templates für die Kommunikation mit Sprachmodellen zu verwenden. Auch hier konnten wir bereits erste Erfahrungen in einem Arbeitsprojekt sammeln.

Zuletzt haben wir uns bei der Datenbank für MongoDB entschieden.

3.2 Gantt-Diagramm & Arbeitsaufteilung

Nun widmen wir uns der Planung und zeitlichen Einordnung der Entwicklung. Dafür erstellen wir ein Gantt-Diagramm, um die Planung zu visualisieren.

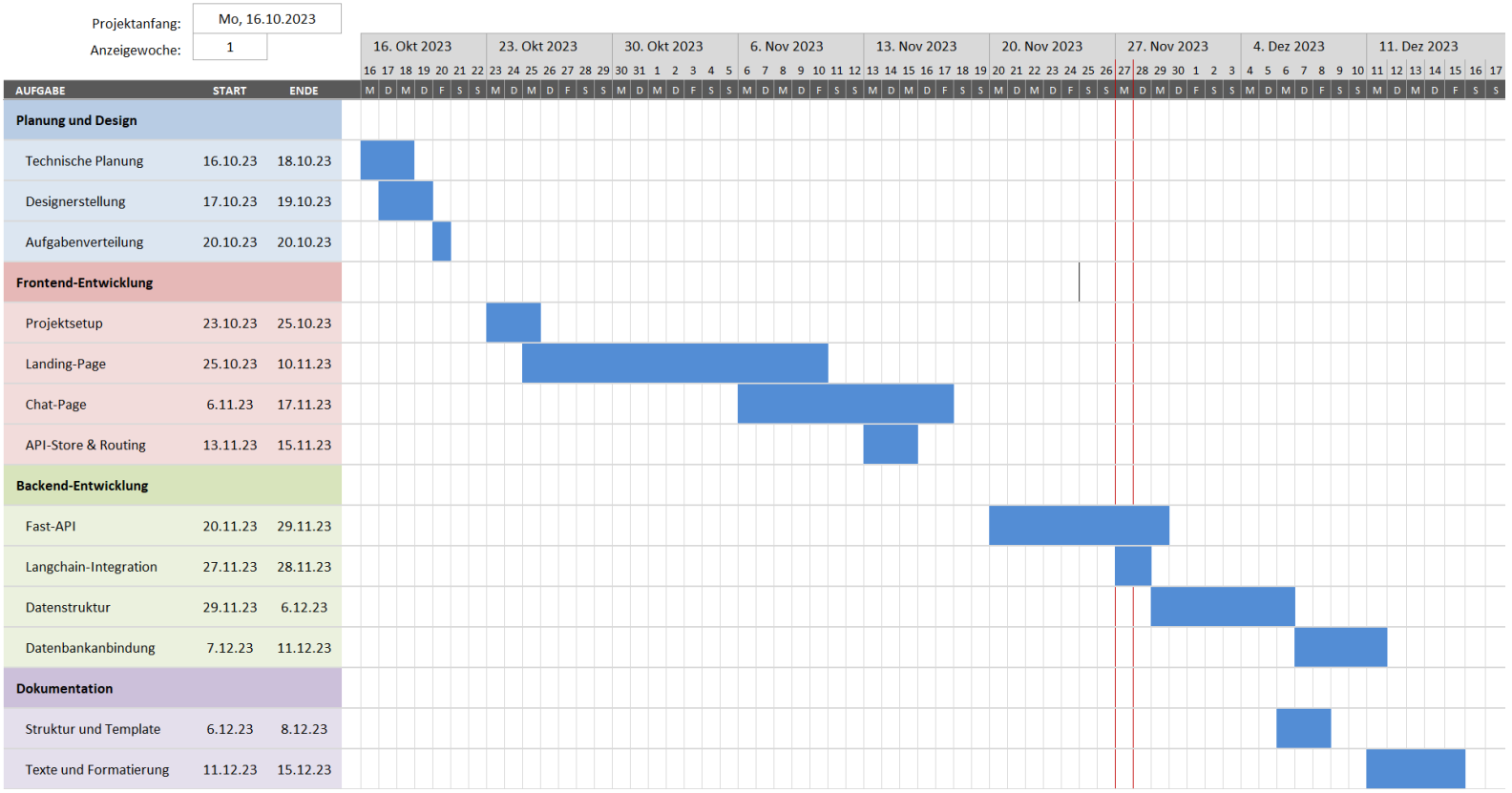

Abbildung 1: Gantt-Diagramm

Mit den einzelnen Arbeitsmodulen teilen wir uns nun auf die jeweiligen Aufgaben auf. Planung und Design übernehmen beide Projektpartner, da dies als Grundlage zur weiteren Arbeit notwendig ist.

Das Projekt-Setup zur Frontendentwicklung wird auch von beiden übernommen. Die Landingpage Module wird hauptsächlich Ramon bauen, während sich Youssef auf die Chat-Page und den API-Store konzentriert.

In der Backend-Entwicklung übernimmt Youssef das Projektsetup und die FastAPI Entwicklung, während Ramon die LangChain-Integration verantwortet. Youssef wird im Anschluss die Datenbankanbindung und Datenstruktur beenden.

Die Dokumentation wird mit Notizen von beiden zusammengetragen und Ramon übernimmt die Texte, Formatierung und Struktur des Dokuments.

3.3 Benötigte Ressourcen

3.3.1 Zeit

Aus dem letzten Projekt nahmen wir mit, dass die Schulzeit für die Entwicklung und Dokumentation nicht ausreicht. Daher können wir in Absprache mit der Firma einen vollen Arbeitstag (auch geteilt auf mehrere Tage) während unserer Arbeitszeit für die Projektentwicklung nutzen. Hier kommt es zugute, dass wir eine ähnliche Thematik als Arbeitsprojekt haben und dadurch unseren Lernprozess unterstützen.

3.3.2 Hard- und Software

Als Hardware stehen uns unsere Firmenlaptops zur Verfügung, welche für die Entwicklung ausreichend sind.

Bei der Software nutzen wir hauptsächlich Visual Studio Code als IDE unserer Wahl und die Office 365 Programme für Dokumentation und Diagrammerstellung.

Die einzig entstehenden Kosten sind die API-Zugriffe zu OpenAI. Hierfür dürfen wir aber den uns zur Verfügung gestellten Firmen-API-Key verwenden, welche wir in einem anderen Projekt bereits genutzt haben. Hierfür sind wir noch in der Lösungsfindung, welche Möglichkeiten wir zur Bereitstellung bei der Projektüberprüfung haben, da wir den Firmen-Key nicht teilen können.

4 Design

4.1 Initiale Konzepte

Bevor wir mit der Designfindung beginnen, erstellen wir einen gemeinsamen Workspace in Figma. So können wir unsere Ideen direkt übertragen und teilen.

Um unser Thema Fitnesscoach in den Vordergrund zu setzen, sollte das Design auch ein sportorientiertes Design haben. Hier haben wir uns sowohl von bestehenden Seiten, beispielsweise von Fitnessstudios, inspirieren lassen. Auch haben wir Designbibliotheken für Websites durchforstet, um etwas passendes zu finden. Unsere ersten Ideen stellen wir zusammen und fügen diese zu einer funktionalen Stage zusammen.

Wir haben mit den jeweiligen Ideen eine farblich passende Chat-Oberfläche erstellt, jedoch noch keine Details hinzugefügt bis wir uns auf ein endgültiges Design einigen.

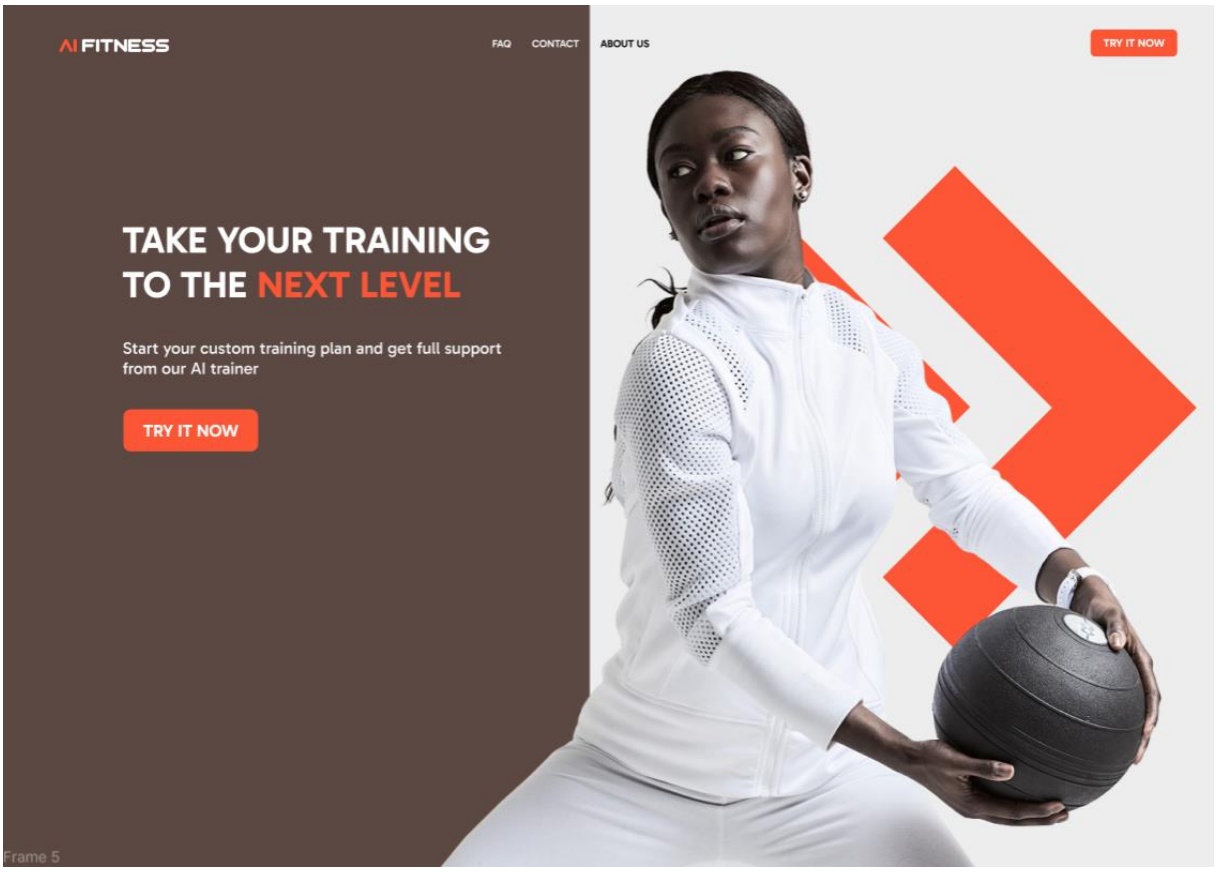

Abbildung 3: Designidee 1

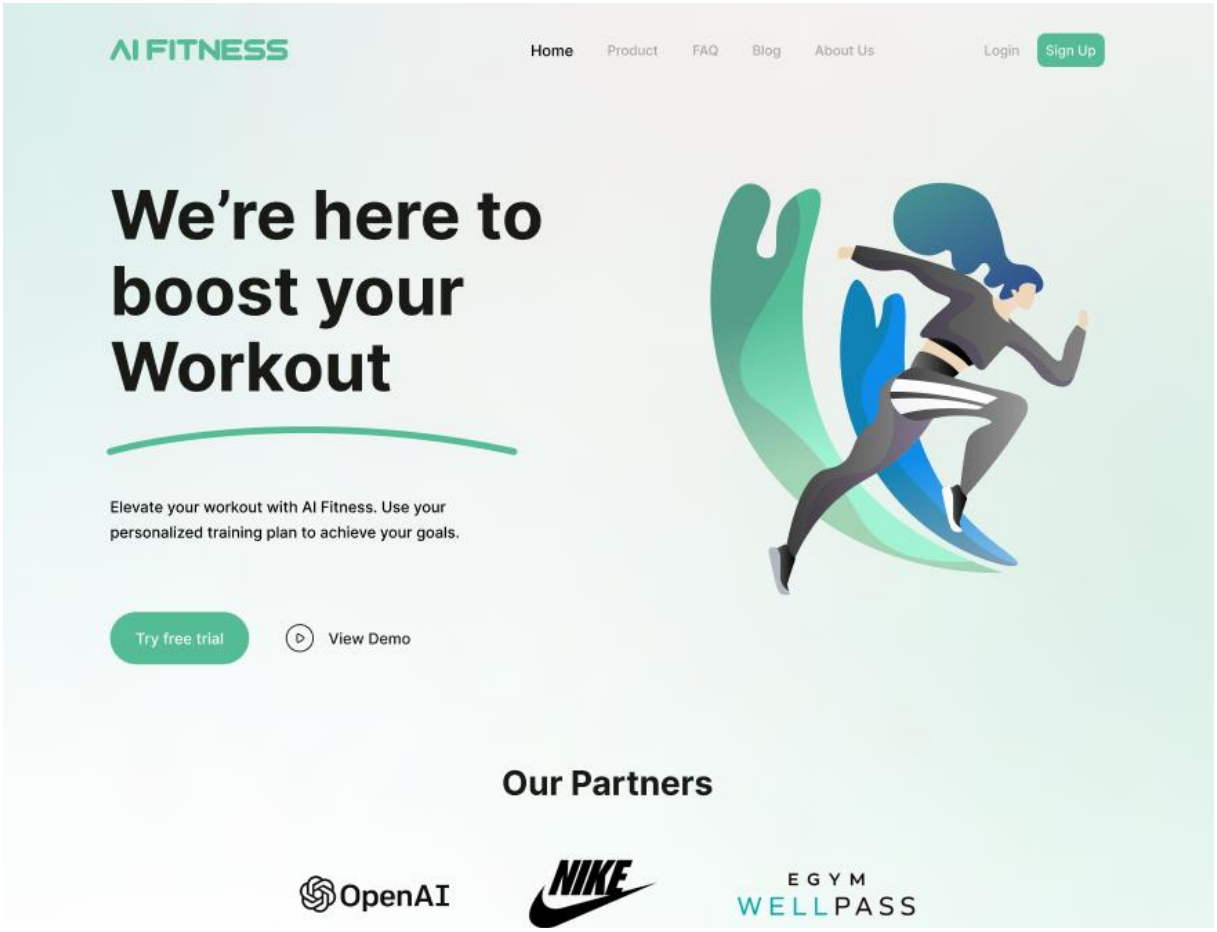

Abbildung 2: Designidee 2

4.2 Finales Design

Nach mehreren Revisionen und weiterer Ideenfindung haben wir unser finales Design und damit die gesamte Landingpage in Figma erstellt. Mit diesem Styleguide als Vorlage können wir deutlich schneller die Module im Frontend erstellen und an den Style der Seite anpassen.

Auch haben wir nun die Chat-Oberfläche als Vorlage erstellt und die genutzten Elemente aus der Landingpage sinnvoll eingebracht.

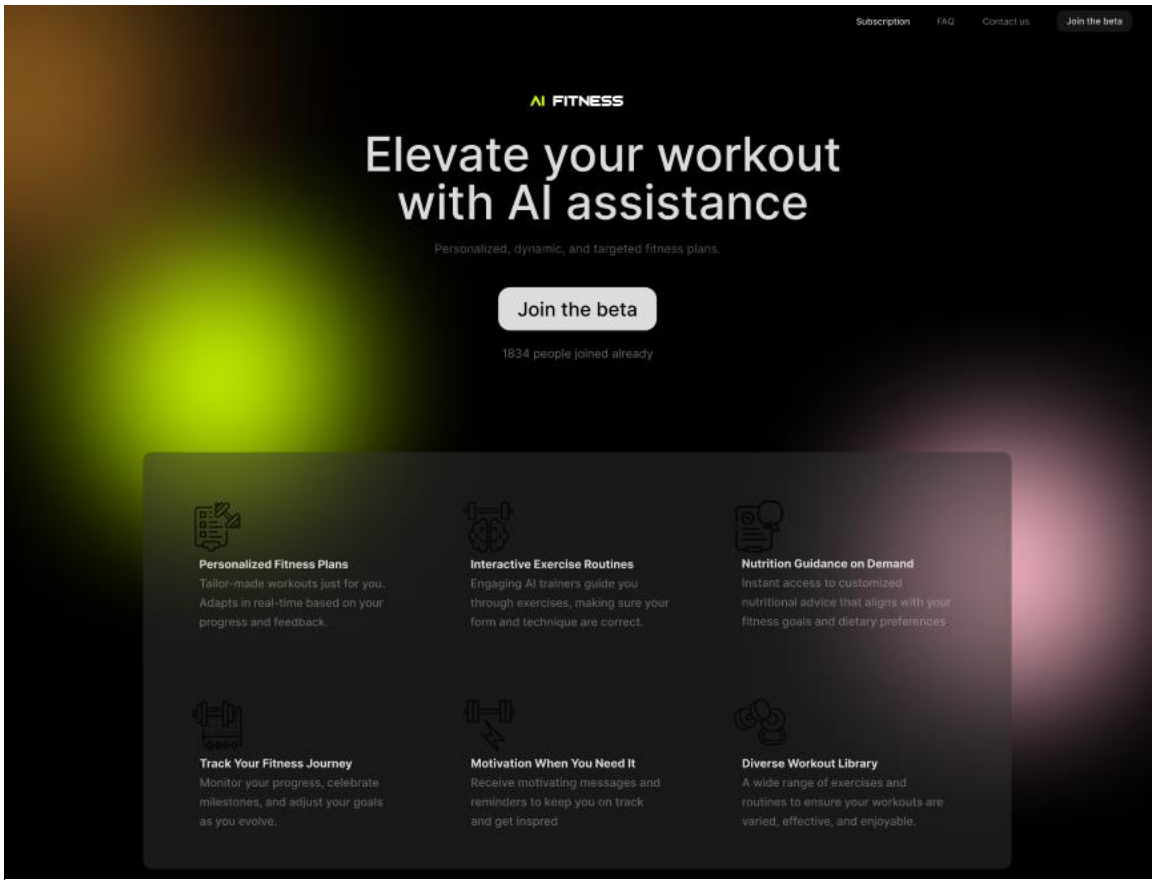

Abbildung 4: finales Konzept - Landingpage

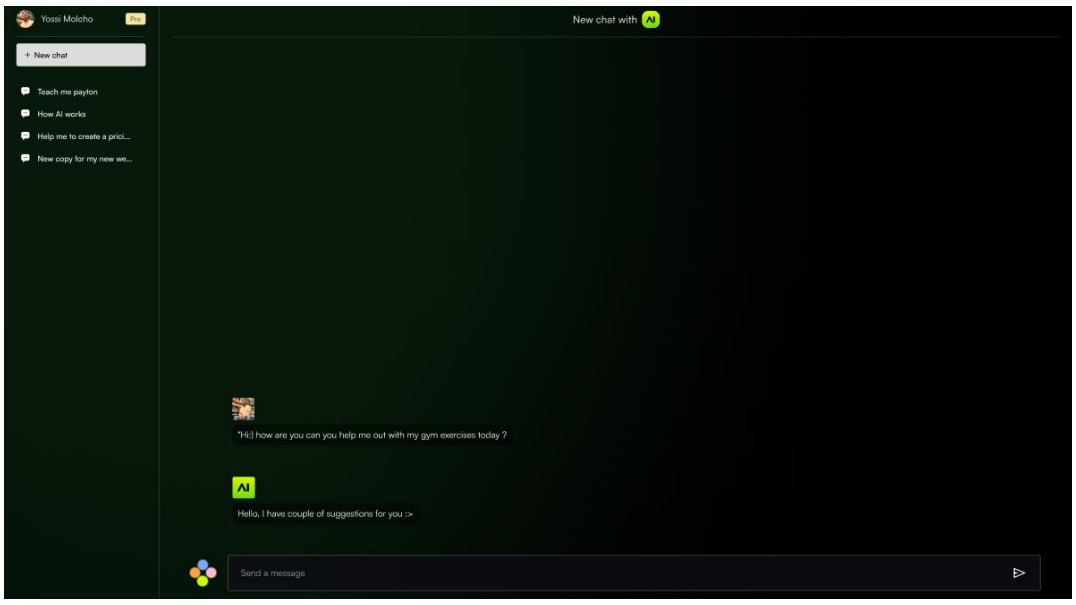

Abbildung 5: finales Konzept - Chatpage

5 Praktische Umsetzung – Frontend

5.1 Technologie & Struktur

Für unser Frontend haben wir uns für Vue.js auf Typescript-Basis als Framework entschieden. Dieses bietet die für uns relevanten Features und Funktionen und ist dabei noch recht leichtgewichtig und übersichtlich als beispielsweise ein Angular-Projekt.

Für eine übersichtliche Struktur trennen wir die Komponenten und Views voneinander, welche wir später in App.vue zusammenbauen. Die View ist dabei die komplette Ansicht mit den einzelnen Komponenten, wie z.B. LandingPage.vue und ChatPage.vue. Eine Komponente ist ein Bauteil dieser View, z.B. Navbar.vue und Stage.vue für die Landingpage.

Die Stylings werden mit der „Composition-API“ von Vue.js direkt in den einzelnen Komponenten platziert. Übergreifendes Styling findet sich entweder in der View oder in App.vue, falls es das gesamte Projekt betrifft, wie z.B. Farbschema.

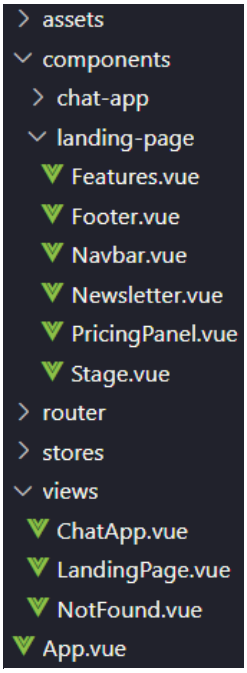

Abbildung 6: Ordnerstruktur - Frontend

5.2 Landingpage

Zu Beginn wollen wir die Landing-Page bauen. Dafür erstellen wir die View „LandingPage.vue“. Hier werden die einzelnen Komponenten importiert und im Template zusammengebaut. Styling und die Platzierung der Einzelteile werden im CSS der Landingpage-View geschrieben.

```
<template>
    <div class="main">
        <Navbar />
        <div class="content">
            <div class="background-layer">
                <div class="neon neon_green" :class="{'neon-moz': isBrowserFirefox}"></div>
                <div class="neon neon_orange" :class="{'neon-moz': isBrowserFirefox}"></div>
                <div class="neon neon_blue" :class="{'neon-moz': isBrowserFirefox}"></div>
                <div class="neon neon_pink" :class="{'neon-moz': isBrowserFirefox}"></div>
            </div>
            <Stage />
            <Features />
        </div>
        <div class="content">
            <PricingPanel />
        </div>
        <div class="content">
            <Newsletter />
        </div>
        <Footer />
    </div>
```

Abbildung 7: HTML-Template Landingpage

5.2.1 Hintergrund-Animation

Die Hintergrund-Animation haben wir als besondere Herausforderung gesehen, da wir auf die Nutzung von CSS-Frameworks verzichtet haben.

Hier haben wir zuerst Animationen von Referenzseiten mithilfe der Entwicklerwerkzeuge in Mozilla Firefox Developer Edition analysiert. Diese basieren jedoch auf Javascript-Funktionen, die im jeweiligen Content Management System integriert wurden. Daher nutzen wir die vorhandenen Keyframes, um die Bewegung der Kugeln nachzustellen und daraus unsere Animation zu bauen.

```css
.neon_green {
    background-color: #cf0;
    top: 0%;
    bottom: auto;
    left: auto;
    right: 0%;
    animation: moveNeonGreen 15s infinite;
}
```

Abbildung 8: Beispiel CSS-Klasse für Hintergrundobjekt

```css
@keyframes moveNeonGreen {

    0%,
    100% {
        transform: translate3d(0, 0, 0);
    }

    33% {
        transform: translate3d(-10vw, 50vh, 50px);
    }

    66% {
        transform: translate3d(-60vw, 30vh, 0);
    }
}
```

Abbildung 9: Beispiel CSS-Keyframes für Hintergrundobjekt

Nun haben wir noch das Problem, dass die Kugel-Objekte in verschiedenen Browsern unterschiedlich aussehen. Letztendlich konnten wir feststellen, dass der „*blur*“-Parameter in Firefox nicht richtig funktioniert. Dafür haben wir eine separate CSS-Klasse erstellt, die die Opazität ändert. Um diese Klasse nur aufzurufen, wenn Firefox als Browser genutzt wird, haben wir eine Prüffunktion hinzugefügt, die einen Boolean ausgibt. Diese haben wir mit dem „:class“-Binding von Vue.js im Template eingebunden, um die CSS-Klasse dynamisch aufzurufen.

5.2.2 Module

Im nächsten Schritt erstellen wir die einzelnen Module. Diese sind hauptsächlich statische Elemente, um den Content zu präsentieren, welcher direkt im Code gepflegt wird.

Eine Besonderheit bietet aber die Navigationsleiste. Hier haben wir uns im Design entschieden, dass diese beim Scrolling auf der Seite oben verankert bleibt. Um das zu erreichen, haben wir die „handleScroll“-Funktion implementiert. Diese prüft den Versatz der Y-Achse und verankert dann ab 80 Pixeln die Navigationsleiste. Auch hier nutzen wir wieder das Binding „:class“, um die CSS-Klasse dynamisch aufzurufen.

Da wir bemerkt haben, dass die Lesbarkeit der Navigationsleistenelemente sich je nach darunter liegendem Modul einschränkt, haben wir noch zwei Animationen hinzugefügt. Zum einen fliegt ein Teil des Logos aus dem sichtbaren Fenster heraus, zum anderen ändert sich die Hintergrundfarbe des rechten Buttons, damit dieser immer sichtbar bleibt.

5.3 Router

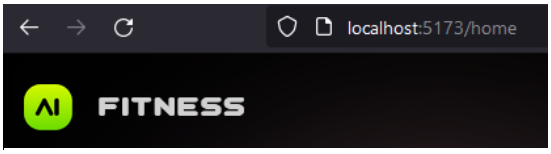

Abbildung 10: URL-Beispiel nach Routing

Wir haben zur Verlinkung aller Unterseiten einen Router hinzugefügt. Vue.js bietet dafür eine leicht implementierbare Lösung mit „vue-router“. Wir hinterlegen hier den Pfadnamen der jeweiligen View und geben den Pfad an, der dann auch in der URL sichtbar ist. Im Router haben wir noch einen regulären Ausdruck hinzugefügt, welcher prüft, ob der angegebene Pfad valide ist. Falls nicht, werden Nutzer zur „NotFound“-View geleitet, welche einen Button zur Landingpage integriert hat.

Die „RouterView“, welche von vue-router importiert wird, haben wir dann im „App.vue“-Template eingebunden. Hier finden sich sonst keine weiteren Komponenten, da diese durch die RouterView automatisch eingebunden werden.

5.4 Chat-Page

Das Designziel der Chat-Page ist eine moderne, schlanke Chatoberfläche, welche wichtige Features wie z.B. Verlauf und Themenvorschläge beinhaltet. Zusätzlich soll sich der Stil an die Landingpage anlehnen, da beide Teile des Gesamtprodukts sind.

Zuerst unterteilen wir das Fenster in zwei Sektionen. Die erste Sektion ist die Ansicht auf der linken Seite, welche den angemeldeten Nutzer, einen Button zum Starten eines neuen Chats und die Historie vergangener Chats anzeigt.

Die zweite Sektion ist das Chat-Interface mit Themenvorschlägen, dem Eingabefeld und den Chat-Blasen des jeweiligen Teilnehmers.

Beide Sektionen werden wir wieder als separate Komponente in die ChatApp-View einfügen.

```
<template>
    <div class="main-chat">
        <div class="chat-neon"></div>
        <div class="chat-content">
            <SidePanel />
            <ChatInterface />
        </div>
    </div>
</template>
```
Abbildung 11: HTML-Template - Chatpage

5.4.1 Hintergrund-Animation

Für eine ansprechende Animation wollten wir ursprünglich einen ähnlichen Stil wie auf der Landingpage nutzen. Da wir aber durch die Chat-Interaktion mit dynamischen Modulen arbeiten, lenkte die erste Animationsidee vom Inhalt ab.

Daher haben wir das Kugelobjekt beibehalten aber anstelle der Bewegungsanimation nur einen pulsierenden Effekt hinzugefügt. Das haben wir mit „transform: scale()“ erreicht, welche die Größe des Objekts ändert.

5.4.2 Chat-Interface

Als Kernstück des Endprodukts wollen wir die Chatoberfläche für Nutzer einfach und übersichtlich gestalten.

Zuerst haben wir eine Übersicht mit unserem Logo und einigen vordefinierten Fragen erstellt. Diese können von Nutzern direkt angewählt werden und wird als initiale Eingabe gesendet. Um eine eigene Frage zu stellen oder auf die letzte Nachricht zu antworten, haben wir ein dynamisches Eingabefeld erstellt, welches die Fenstergröße der Nachrichtenlänge anpasst.

Die Fragen und das Eingabefeld haben wir ein HTML <form> Element gepackt. Die Vorschläge werden durch die „v-if“-Abfrage nur gezeigt, wenn in der aktuellen Chatsession keine Nachrichten vorhanden sind. Die Abfrage mit „v-if“ und „v-else“ verwenden wir wiederholt im Chat-Interface, da so sehr einfach mit der gegebenen Bedingung das Element dynamisch geladen werden kann.

Für die Chatnachrichten haben wir zwei Rollen definiert, User und AI. Je nachdem welche zu welcher Rolle die Nachricht gehört, hat die Chatnachricht die Initialen des Nutzers oder nur AI als Rollen-Icon. Auch ändert sich die Umrandung der Chatnachricht, wenn die Rolle AI wiedergibt. Um die Ladezeit für den API-Aufruf zu visualisieren, haben wir außerdem eine Animation mit CSS hinzugefügt. Damit wir sicherstellen, dass diese immer korrekt angezeigt wird, haben wir für alle gängigen Browser die passende Animations- und Transformationseigenschaft hinzugefügt.

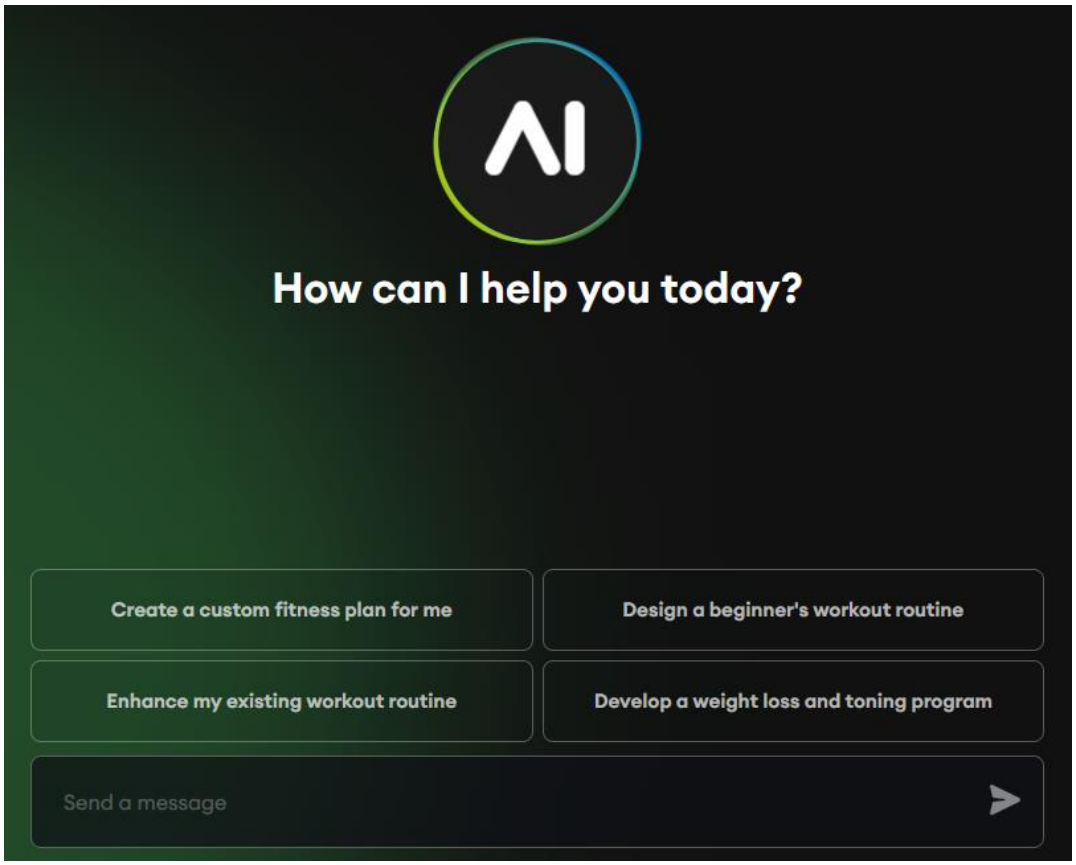

Abbildung 12: Chatpage - Startansicht

5.4.3 Side Panel

Für das Side Panel erstellen wir zunächst die 3 Elemente User, einen „New Chat“-Button und eine Liste der bisherigen Chats.

Für letztere nutzen wir die in Vue.js enthaltene „TransitionGroup“. Diese Komponente ist in der Implementation ähnlich zu einer Liste, stellt aber eine bessere Funktionalität beim Hinzufügen und Löschen von Elementen mit Animationen zur Verfügung.

Zuletzt haben wir noch die Logik für den API-Store Zugriff hinzugefügt. So können vergangene Sessions wieder aufgerufen werden, neue erstellt und nicht mehr benötigte gelöscht werden. Diese müssen jetzt noch in das Template eingebunden werden. Hier nutzen wir die „v-for“-Schleife, um die Listenelemente aus dem Backend mit dem API-Aufruf hinzuzufügen und nach dem Index zu ordnen.

5.5 API-Store

Um den API-Store für die Kommunikation mit dem Backend zu bauen, erstellen wir vorab eine REST API mit unseren Backenddaten. Hierfür nehmen wir NSwag zur Hilfe. Damit erstellen wir eine automatisch generierte API, welche alle wichtigen GET, POST, PUT und DELETE Methoden enthält.

Nun können wir den API-Store erstellen. Dafür nutzen wir die Vue.js-Library „Pinia“. Mit der integrierten „defineStore“-Methode schreiben wir unseren API-Store mit allen nötigen GET- und SET-Methoden. Der Store ermöglicht es den Zustand von unseren Objekten abzurufen und global zu verändern. So wird auch die Kommunikation mit dem AI-Trainer implementiert. Der Nutzer sendet seine Nachricht ab, welche im Backend verarbeitet wird. Das Backend sendet die Antwort der AI dem Client zurück, welche dem Nutzer als Antwort im Frontend ausgegeben wird.

Mit dem fertigen API-Store implementieren wir nun noch die Logik in unsere Chat-Komponenten. Hierfür fügen wir weitere Methoden hinzu, um z.B. die Nutzereingabe zu prüfen und Leerzeichen zu entfernen, damit wir möglichst kleine API-Aufrufe zu senden.

6 Praktische Umsetzung – Backend

6.1 Technologie und Struktur

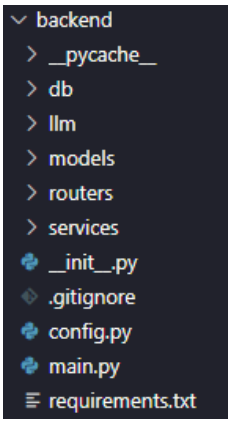

Abbildung 13: Ordnerstruktur Backend

Für unser Backend fiel die Technologiewahl auf Python mit FastAPI und LangChain. Mit diesen Technologien können wir unsere bereits gesammelte Erfahrung nutzen und dabei etwas Neues dazulernen. Python hat einen sehr umfangreichen Support, wenn es um AI-Anwendungen geht. LangChain ist das optimale Framework für „Large Language Model“ (im Folgenden „LLM“) Implementationen. Und zuletzt bietet FastAPI ein schnelles und einfaches Python-Framework für eine robuste RESTful-API.

Bei der Datenbank fiel die Entscheidung auf MongoDB, da wir einen schnellen und einfachen Datenbankzugriff für die Chat-Sessions benötigen. Eine dokumentenorientierte Datenbank bietet sich dafür am besten an.

6.2 LangChain-Implementation

Wir beginnen zuerst mit LangChain und den „Prompt-Templates“, da unsere Services darauf aufbauen werden. LangChain dient für uns als Schnittstelle zwischen dem Backend und dem API-Call zu OpenAI. Für den Aufruf benötigen wir zuerst ein PromptTemplate, welches der AI ihr Verhalten und die Antwortstruktur vorgibt. Da dieser Chat sich nur um das Thema Fitness drehen soll, geben wir das auch dementsprechend im Template an. Fragen außerhalb des Themenbereichs werden vermieden.

Wir benötigen noch ein weiteres Template, um den Titel des Chats aus der Fragestellung herauszufiltern. Hierfür haben wir das sogenannte „FewShotPrompt-Template“. Beim FewShotPrompt gibt man dem LLM mehrere Beispiele vor, wie eine Fragestellung und die dazugehörige Antwort aussehen können. So stellen wir sicher, dass das Ergebnis in diesem Format bleibt.

```
EXAMPLES = [
    {"user": "Who is the president of Gabon?",
        "assistant": "GA President of Gabon"},
    {"user": "Who is Julien Chaumond?",
        "assistant": "🧑 Julien Chaumond"},
    {"user": "what is 1 + 1?",
        "assistant": "🔢 Simple math operation"},
    {"user": "What are the latest news?",
        "assistant": "📰 Latest news"},
    {"user": "How to make a great cheesecake?",
        "assistant": "🍰 Cheesecake recipe"},
    {"user": "what is your favorite movie? do a short answer.",
        "assistant": "🎬 Favorite movie"},
    {"user": "Explain the concept of artificial intelligence in one sentence",
        "assistant": "🤖 AI definition"}]
```

Abbildung 14: FewShotTemplate - Leitbeispiele für KI

6.3 Models

Bevor wir unsere Modelle für die Services erstellen, überlegen wir vorab, wie wir diese speichern und aufrufen können. Da wir MongoDB als auch FastAPI nutzen möchten und mit letzterem asynchrone Aufrufe produzieren, muss der Datenbankzugriff dies auch zulassen. Andernfalls könnte bei synchronen Aufrufen zu einer langsamen und nicht nachvollziehbaren Nutzererfahrung im Frontend führen, wenn einzelne Services und Funktionen auf die vorherigen API-Calls und Berechnungen warten müssen. Daher brauchen wir einen „Object-Document Mapper“ (ODM), der sowohl asynchron arbeitet als auch auf Python ausgelegt ist. Hier haben wir nach kurzer Recherche „Beanie“ gefunden, welcher die für uns relevanten Features mitbringt.

Nun können wir unsere Modelle erstellen. Dabei nutzen wir die Endpoint-Modelle für die API mit Typescript-Support, da eine Typisierung für die grundlegende Funktionalität erforderlich ist.

Das Kernstück bildet die „ChatSession“, welche die Nutzer- und AI-Nachrichten zusammenführt und die Templates anwendet. Die ChatSession wird dann später als Dokument in MongoDB gespeichert.

6.4 Services

Mit den fertigen Models beschäftigen wir uns als nächstes mit den Services. Im ersten Aufruf der Klasse wird das Chatmodell und der API-Key der OpenAI-API initialisiert, welche in „config.py“ hinterlegt wurden. Auch haben wir hier mithilfe von LangChain einen „InMemoryCache“ hinzugefügt. So reduzieren wir die Anzahl der API-Calls, indem wir sich wiederholende Muster im „Cache“ zwischenspeichern und bei Bedarf erneut nutzen.

```
class ChatService:
    def __init__(self):
        self.chat_model = ChatOpenAI(
            api_key=settings.OPENAI_API_KEY, model_name=settings.MODEL_NAME)
        set_llm_cache(InMemoryCache())
        self.title_creator = TitleLlm()
        self.sessions = {}
```

Abbildung 15: ChatService - init-Methode

Die Klasse implementiert außerdem die „CRUD“-Logik (Create, Read, Update, Delete) mit unserem ChatSession-Objekt. Diese Methoden werden dann im nächsten Schritt in FastAPI genutzt.

6.5 FastAPI & MongoDB

6.5.1 Router

Die Implementation von FastAPI bringt unser Backend nun zusammen. Hierfür erstellen wir zunächst einen API-Router. Zu diesem Router fügen wir unsere HTTP-Methoden, welche die vorher definierten Services zugreifen. Damit haben wir alle benötigten Endpunkte hinzugefügt, welche dann von NSwag mit der automatisch generierten API aufgegriffen und für Typescript übersetzt werden.

6.5.2 main.py

Nun müssen wir die letzten Konfigurationen in unserer „main.py“ vornehmen. Hier starten wir FastAPI und fügen die notwendige „Middleware“ hinzu. Dafür nutzen wir „CORSMiddleware“ von FastAPI. Damit können unser Frontend und Backend miteinander kommunizieren, obwohl diese sich auf unterschiedlichen Domänen befinden. Hier setzen wir keine Einschränkungen, da unsere Anwendung vorerst ausschließlich lokal ausgeführt wird. Sollte sich dies ändern, können die Anpassungen zentral in dem „add_middleware()“-Aufruf geändert werden. Auch fügen wir hier den zuvor erstellten Router und unsere Datenbank-Initialisierung hinzu, damit letztere auch beim ersten Aufruf gestartet wird.

```python
app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Allows all origins
    allow_credentials=True,
    allow_methods=["*"],  # Allows all methods
    allow_headers=["*"],  # Allows all headers
)
```

Abbildung 16: Middleware - Implementation

Als Webserver für unser Backend nutzen wir „Uvicorn“, da es auf asynchrone Frameworks wie FastAPI ausgelegt ist und dabei mit einer schnellen Leistung punktet.

6.5.3 MongoDB

Um die Arbeit am Backend abzuschließen, brauchen wir nur noch die Datenbankanbindung. Wir nutzen die in config.py hinterlegten Einstellungen für den MongoDB URI in einem „Motor“-Client, welcher als asynchroner Treiber für MongoDB agiert. Danach setzen wir die Datenbank mit dem Motor-Client auf. Bevor diese aber gestartet wird, initialisieren wir zuerst Beanie, da der ODM essenziell für unsere Datenbanknutzung ist.

```python
async def init_db():
    MOTOR_CLIENT = AsyncIOMotorClient(settings.MONGODB_URI)
    DATABASE = MOTOR_CLIENT[settings.DATABASE_NAME]
    await beanie.init_beanie(
        database=DATABASE,
        document_models=[ChatSession]
    )
```

Abbildung 17: Datenbank - init-Methode

7 Containerisierung & lokale Ausführung

Wir haben uns bei der ursprünglichen Projektplanung gegen ein Deployment entschieden, da zum einen ein kostenpflichtiger API-Key für die vollumfängliche Nutzung erforderlich ist, als auch, dass ein richtiges Deployment zeit- und ressourcenaufwendig ist.

Im Rahmen des Schulprojekts und zur Einsicht für unsere Firma möchten wir aber einen einheitlichen Zustand unserer Anwendung sicherstellen und den Aufwand vom Setzen der Umgebungsvariablen vermeiden. Daher haben wir zu unserem Backend und Frontend jeweils einen „Dockerfile“ hinzugefügt, welcher die Voraussetzungen, wie z.B. „pip install -r“, „npm i“ und die Startbefehle automatisch beim Aufbau ausführt.

Im Anschluss haben wir noch die „docker-compose“-Datei angelegt. Diese führt unsere Anwendung mit mehreren Containern aus. Hier haben wir auch die nötigen Umgebungsvariablen hinterlegt. Lediglich der API-Key muss selbst gesetzt werden. Um unsere gesamte Anwendung nun lokal auszuführen, benötigt man den laufenden „Docker-Daemon“ und kann mit „docker-compose up“ den Prozess starten. Damit werden alle Abhängigkeiten automatisch installiert und die Anwendung läuft nach kurzer Installation.

8 Login-Maske & Authentifizierung

Um die Login-Maske nachträglich hinzuzufügen, mussten wir an der API einige Anpassungen vornehmen. Anstatt einer Liste von ChatSessions direkt zu übergeben, gibt es jetzt eine User-Klasse, welche eine eigene ChatSession-Liste hat.

Die neuen Authentifizierungsendpunkte registrieren den Nutzer und geben ein Zugangstoken zurück, mit welchem dieser Zugriff auf den Chat-Endpunkt hat und Änderungen der ChatSession-Liste vornehmen kann.

Für die Registrierung haben wir eine weitere LLM-Anbindung genutzt, welche die Anmeldeinformation (*Name, Passwort, Email, Sicherheitsfrage*) des Nutzers abfragt im Registrierungsfeld zu einem Objekt konvertiert und bei falschen oder fehlenden Informationen, z.B. ungültiges Email-Adressen-Format, einen Fehler zurückgibt. Sobald das Objekt vollständig ist, wird das Passwort mit „Hashing“ verschlüsselt und die Sicherheitsfrage wird in ein „SecurityQuestionAnswer“ umgewandelt und mit dem User-Objekt in MongoDB gespeichert.

Für den Login bestehender Nutzer, geht dieser durch eine Zwei-Faktor-Authentifizierung. Zuerst werden die E-Mail und das „gehashte“ Passwort abgeglichen und mit einem Zwischentoken versehen. Dann wird die Sicherheitsfrage gestellt. Die Antwort wird mit dem Zwischentoken an das LLM weitergegeben, welche die Antwort mit der Frage überprüft. Bei erfolgreicher Beantwortung wird das reguläre Zugangstoken zurückgegeben. Bei falscher Antwort wirft das LLM eine Fehlermeldung zurück.

9 Fazit

9.1 Soll-/Ist-Vergleich

Rückblickend auf den Soll-Zustand haben wir die Voraussetzungen erfüllt, welche wir vorab definiert haben.

Zusätzlich sind noch die Containerisierung als Feature und viele kleinere Designelemente hinzugekommen, welche in der initialen Planung nicht angedacht waren. Durch die Verschiebung des Abgabetermins konnten wir auch noch die Login-Maske und Logik hinzufügen und damit eine unserer Wunsch-Spezifikationen abhaken.

Insgesamt haben wir damit eine robuste und funktionsreiche Anwendung aufgesetzt, die auch gleichzeitig eine gute Nutzererfahrung bietet.

9.2 Rückblick & Herausforderungen

Die gemeinsame Arbeit am Projekt hat nicht nur Spaß gemacht, sondern auch viele Lerneffekte mitgebracht. Vor der Entwicklung ein durchdachtes Konzept und Design in Figma zu erstellen, hat uns die Arbeit am Endprodukt erheblich erleichtert. So hatten wir ein klares Ziel vor Augen und haben nicht immer weiter Features hinzugefügt. Dadurch ist die Logik der bestehenden Bauteile deutlich robuster ausgefallen.

Auch haben wir unseren Stand in regelmäßigen Meetings abgeglichen und vorgestellt. Damit haben wir uns nicht nur up to date gehalten, sondern konnten unseren Code gegenseitig verbessern und verständlicher machen. Das war für speziell für die Dokumentation hilfreich.

Dennoch sind wir auf einige Herausforderungen gestoßen. Zunächst hätten wir die Aufgaben besser unterteilen und ggf. mit einem Ticketsystem (z.B. Trello oder Jira) arbeiten sollen. Es kam durch unsere Planung vor, dass sich unsere Entwicklung manchmal überschnitten hat und wir Module doppelt bearbeitet haben. Das Gantt-Diagramm war dabei zwar ein guter Grundleitladen aber hindert doch die agile Entwicklungsweise, die wir aus der heutigen Softwareentwicklung kennengelernt haben, wie z.B. der Sprint mit gesteckten kurzfristigen Zielen in Scrum oder Kanban.

Da wir durch die Verschiebung der Abgabefrist noch Zeit gewannen, wollten wir noch die Login-Maske und die zugehörige Authentifizierungslogik mit dem LLM als „wow“-Feature hinzufügen. Dies hat jedoch vor allem unsere Backendlogik an einigen Stellen verändert, wodurch die Dokumentation nicht mehr 100% aktuell ist und aus Zeitmangel nicht mehr vollständig aktualisiert werden konnte.

9.3 Ausblick

Dieses Projekt ist als schulische Hausarbeit entstanden und wird nach der Abgabe nicht weiterentwickelt. Jedoch werden wir dieses häufig als Referenz nehmen, da wir bewusst überschneidende Themen mit unseren aktuellen Arbeitsprojekten gewählt haben. So konnten wir vorbereitend auf dieses dazu lernen.

Auch werden wir die Erfahrungen des gemeinsamen Arbeitens in Zukunft mitnehmen und in unseren Arbeitsalltag einfließen lassen. Das reicht von Projektplanung und -umsetzung bis hin zu kommunikativen Bereichen, wie Feedback und Kritik.

Insgesamt werden wir sehr zufrieden aus dieser Projektphase rausgehen, da wir aus unserer Sicht ein solides Endprodukt geschaffen haben.

10 Abbildungsverzeichnis

11 Quellenverzeichnis

Dokumentationsreferenzen & Troubleshooting für die Entwicklung:

https://vuejs.org/guide/extras/composition-api-faq.html

https://fastapi.tiangolo.com/reference/

https://python.langchain.com/docs/modules/model_io/llms/llm_caching

https://python.langchain.com/docs/modules/model_io/prompts/prompt_templates/few_shot_examples

https://stackoverflow.com/

https://www.uvicorn.org/settings/

https://fastapi.tiangolo.com/tutorial/cors/

https://chat.openai.com/

Erstellungshilfen für die Dokumentation:

https://wiki.induux.de/Pflichtenheft

https://create.microsoft.com/de-de/templates/gantt-diagramme

12 Eidesstattliche Erklärung

Hiermit versichern wir, Youssef Sbai und Ramon Grothe, dass wir diese Hausarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet haben, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Prüfungsleistung war.

Gez. Youssef Sbai – Heilbronn, den 23.12.2023

Gez. Ramon Grothe – Heilbronn, den 23.12.2023