DHBW
Duale Hochschule
Baden-Württemberg
Mosbach

JourneyTunes

Web-Services

Studiengang/-richtung:	Angewandte Informatik
Kurs:	INF22A
Dozent:	Hr. Prof. Dr. Alexander Auch
Abgabedatum:	31.03.2024
Mitglieder:	5199998, 6866394, 8085675, 3935706

Inhaltsverzeichnis

1 Projektidee

Die ursprüngliche Vision für JourneyTunes zielte darauf ab, Nutzern einen einzigartigen Reiseservice zu bieten, der Überraschungselemente beinhaltet. Kern der Idee war es, eine intuitive Plattform zu schaffen, auf der Benutzer Hotels auswählen, Routen basierend auf ihrem Standort erstellen und Restaurants entlang dieser Route entdecken können. Ein besonderes Feature sollte die Erstellung personalisierter Playlists sein, die auf den individuellen Musikvorlieben der Nutzer basiert. Für die Nutzung dieses Services war die Anlage eines Benutzerkontos vorgesehen, über das geplante Reisen verwaltet und Hotels registriert werden können. Die Integration externer APIs, insbesondere von Spotify für die Playlists und Google Maps für die Routenplanung, war ein zentraler Bestandteil des Konzepts. Für die Kernfunktionen wie Nutzerverwaltung und Reiseplanung sollten eigene APIs entwickelt werden, während für den Hotelservice zunächst die Integration einer externen API angedacht war, falls eine geeignete gefunden werden könnte.

Das realisierte Projekt JourneyTunes hat sich zu einer umfassenden Reisebuchungsplattform entwickelt. Nach der Registrierung, die durch Eingabe von Nutzername, Passwort und E-Mail-Adresse sowie einer E-Mail-Verifikation abgeschlossen wird, können sich Nutzer anmelden und direkt mit der Reiseplanung beginnen. Die Auswahl eines Hotels leitet die Nutzer zu einem interaktiven Teil der Website, wo sie ihre musikalischen Präferenzen für eine individuell erstellte Playlist angeben können. Am Ende der Reisplanung kommt die übersichtliche Darstellung der ausgewählten Reiseoptionen auf einer Check-out-Seite, auf der alle Details zur geplanten Reise gebündelt sind. Im Profilbereich können Nutzer ihre geplanten und vergangenen Reisen einsehen sowie Hotels registrieren, die sie anbieten möchten. Aufgrund des hohen Entwicklungsaufwands haben wir uns gegen die Implementierung eines Restaurantservices entschieden. Da keine passende externe Hotel-API verfügbar war, haben wir uns für die Entwicklung einer eigenen Lösung entschieden. Für die Routenplanung setzen wir auf den Open Route Service (ORS), eine kostenfreie Alternative zu Google Maps. Insgesamt setzt sich das Projekt also aus sechs Services zusammen. Wir haben zwei Fremd-APIs verwendet, Spotify und ORS, sowie zusätzlich vier eigene, Hotel, User, E-Mail und Trip, implementiert.

2 Teameinteilung und Planung

In diesem Abschnitt werden die Teameinteilung sowie die Planung dargelegt.

Unser strategischer Ansatz für die Entwicklung von JourneyTunes legte den Fokus zunächst auf die Implementierung des Backends. Diese Entscheidung haben mit der Intention getroffen, eine solide und gut durchdachte Grundlage für die Hauptkomponenten unserer Applikation zu schaffen. Indem wir zuerst das Backend entwickelten, stellten wir sicher, dass die Kernfunktionalitäten korrekt und effizient umgesetzt werden, was eine Integration mit dem Frontend in späteren Entwicklungsphasen ermöglicht. Dieser Ansatz erleichterte es erheblich, das Frontend anzubinden, da wir bereits eine starke, zuverlässige und gut dokumentierte Backend-Struktur hatten, auf die wir aufbauen konnten. Wir haben in zweier Teams gearbeitet und bei den wichtigsten Komponenten alle zusammen gearbeitet, um einen Lernerfolg für alle zu gewährleisten.

2.1 Backend

Komponente	Teammitglied
Config-Service	5199998, 6866394
Discovery	5199998, 6866394
Gateway	8085675, 3935706
Email	8085675, 3935706
Hotel	5199998, 6866394, 8085675, 3935706
Routes	5199998, 6866394
Spotify	8085675, 3935706
Trip	8085675, 3935706, 5199998, 6866394
User	8085675, 3935706

Abbildung 1: Einteilung der Backend-Implementierung

2.2 Frontend

Komponente	Teammitglied
Homepage	5199998, 6866394
Login/Registration	8085675, 3935706
Profile	5199998, 6866394
Hotel-Registration	8085675, 3935706
Plan Your Trip	5199998, 6866394, 8085675, 3935706
Design	5199998, 6866394, 8085675, 3935706

Abbildung 2: Einteilung der Frontend-Implementierung

3 Architektur & Aufbau

Dieser Abschnitt befasst sich mit unserer Microservices-Architektur und generellen Aufbau des Projektes, welches zentral aus einem Frontend, mehreren Webservices, einer Datenbank sowie einem Discovery- und Konfiguration-Server besteht.

Der Aufbau ist in Abbildung 3 dargestellt. Die gesamte Anwendung läuft in einem Docker-Container-Ökosystem und ist nur über bestimmte Ports erreichbar. Der Haupteintrittspunkt für Benutzer ist das Frontend, welches über den Port 80 erreichbar ist.

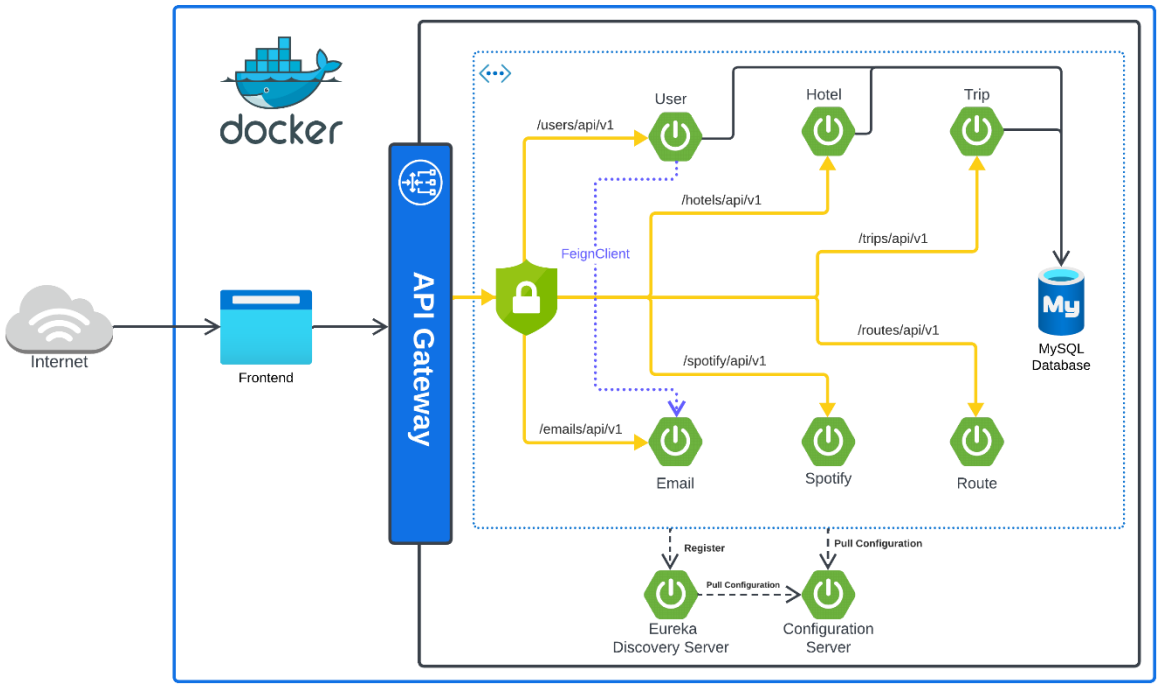

Abbildung 3: JourneyTunes Architektur

Über dieses Frontend, welches mit dem Framework Vue mit Vite und Vuetify erstellt wurde, wird mittels eines AxiosClients mit dem Backend über ein API Gateway kommuniziert. Das Gateway dient als einziger Einsteigspunkt für alle Clients und hat die Aufgabe alle Anfragen an die entsprechenden Microservices weiterzuleiten. Es fungiert zusätzlich noch als Lastenausgleich und regelt für jegliche Anfrage die Authentifizierung ab, falls diese nötig ist. Realisiert wird dieses Gateway mit einer Spring Cloud Gateway Applikation.

Zentral nutzen wir einen Konfiguration-Server, dieser verwaltet und verteilt Konfigurationen an alle Microservices, sowie dem Eureka Discovery und Gateway Server. Dadurch haben wir eine zentrale Konfigurationsstelle, die es ermöglicht leicht Einstellungen an den Konfigurationen der einzelnen Applikationen zu tätigen.

Zusätzlich haben wir uns für einen Eureka Discovery Server entschieden, bei diesem sich alle Microservices registrieren. Dies ermöglicht es uns eine lose gekoppelte Verbindung zu erstellen, die über FeignClient realisiert wird. FeignClients werden durch eine Annotation implementiert und ermöglichen es eine Interface-Defintion anderer Services automatisch während der Laufzeit zu implementieren. Dabei muss keine direkte Adresse angegeben werden, es wird lediglich der Applikationsname benötigt der beim Eureka Discovery Server angegeben ist.

Die einzelnen Webservices sind Spring Boot Web Application mit REST Schnittstellen, die ihre jeweiligen Dienste bereitstellen und ebenfalls mit einer OpenAPI Spezifikationen ausgestattet sind. Hierbei haben wir sechs Services eingebunden, von denen vier eigens erstellte Services sind und zwei externe Services, die in Abschnitt 4.2 genauer darlegt werden.

Zuletzt haben wir noch eine zentrale MySQL Datenbank, die Daten für verschiedene Microservices speichert und verwaltet, die in den Spring Applikationen über Hibernate und der Jakarta Persistence API mittels angelegten Entities und Repositories verwaltet wird.

Abschließend verfügt unsere Architektur über eine zentrale MySQL-Datenbank, die als Datenspeicher für die verschiedenen Microservices dient. Die Interaktion mit dieser Datenbank erfolgt innerhalb der Spring-Anwendungen durch den Einsatz von Hibernate sowie der Jakarta Persistence API. Die Datenmodelle werden dabei als Entities definiert, während Repositories für den Datenzugriff verwendet werden, um die Datenverwaltung und -manipulation zu erleichtern. Diese Struktur ermöglicht es uns, die Datenintegrität zu wahren und komplexe Datenabfragen und -transaktionen effizient zu handhaben.

Unsere endgültige Projektstruktur ist in der Abbildung 4 aufgezeigt. Dabei haben wir für jede Komponente ein eigenes Projekt angelegt. Dort haben wir eine Struktur nach dem klassichen Projektaufbau aufgesetzt, wobei wir alles in Packages aufgeteilt haben. Die einzelnen Services haben ein Package für controllers, modules und clients, sowie ein Package für Datentransferobjekte. Die tatsächlichen Businesslogik-Objekte sind im Package modules zu finden. Dies ist im Quellcode einzusehen.

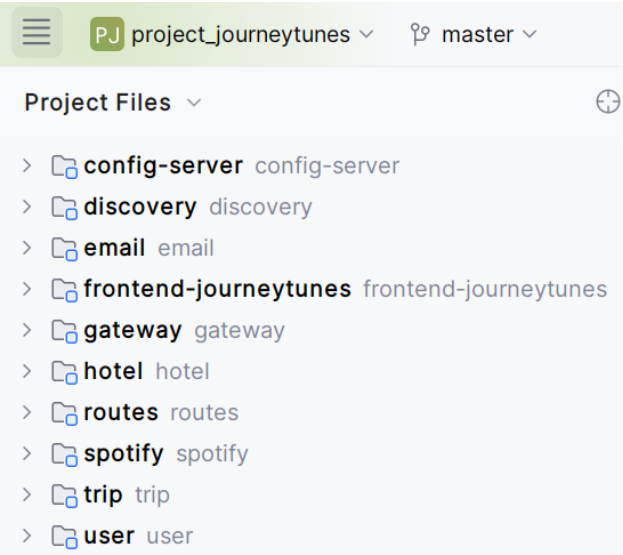

Abbildung 4: Aufbau unseres Projekts im Code

4 Backend

4.1 Datenbank

Für die Speicherung der Daten, wurde eine MySQL Datenbank gewählt. Diese wird mit JPA und Hibernate im Code referenziert und über Entities, Repositories und Services verwaltet.

Zusätzlich gibt es noch ein PhpMyAdmin Frontend, um die Datenbank zu verwalten.

Die Datenbank besteht aus sieben Tabellen, welche die Datenstruktur, die für das JourneyTunes Projekt benötigt wird, bilden. Im ER-Diagramm in Abbildung 5 sind alle Tabellen, einschließlich ihrer Attribute und Abhängigkeiten, dargestellt. Das ER-Diagramm ist zusätzlich noch als PDF im GitLab-Projekt zu finden.

Zusätzlich zu den im ER-Diagramm referenzierten Tabellen, gibt es noch eine Tabelle mit dem Namen „delayed_token_deletion“. Diese wird von einem Datenbank Event verwendet, um die Löschung der Login Token nach einer Stunde zu automatisieren. Befüllt wird diese Tabelle von einem Trigger auf der Tokentabelle, welcher bei jedem neu erstellten Token, das Ablaufdatum und die TokenId in dieser Tabelle speichert.

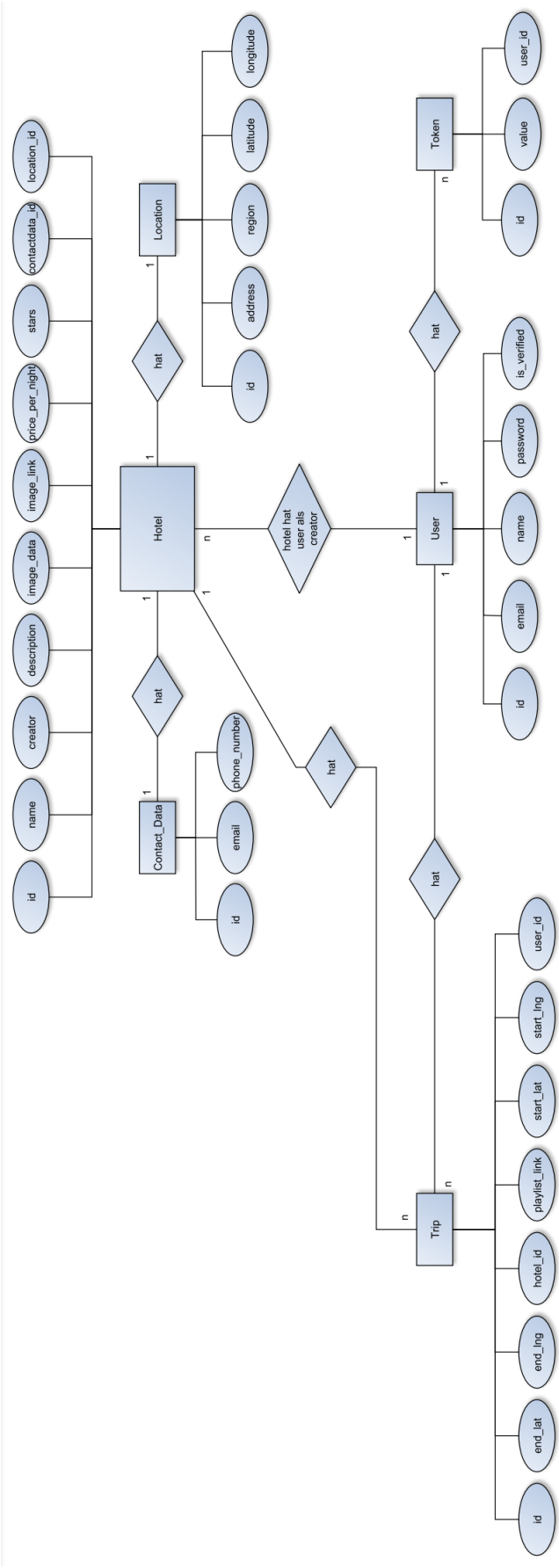

Abbildung 5: ER-Diagramm der MySQL

4.2 Services

4.2.1 Hotel

Schnittstelle: “/hotels/api/v1”

Der Hotel-Service kümmert sich um die Bereitstellung der Hotels für das Frontend und liefert die benötigten Daten für den Trip Service.

Mithilfe verschiedener API-Endpunkte ist es möglich neue Hotels anzulegen, bestehende anzupassen oder auch zu löschen. Es gibt zusätzlich noch einen Endpunkt für das Durchsuchen der Hotels nach den Kriterien: Region, PricePerNight, Stars und Name. Speziell für die Profilseite des Frontends ist noch ein Endpunkt von Nöten, welcher alle Hotels, welche der Benutzer erstellt hat, zurückgibt, dieser Endpunkt ist als BPMN-Modell exemplarisch in Abbildung 6 dargestellt.

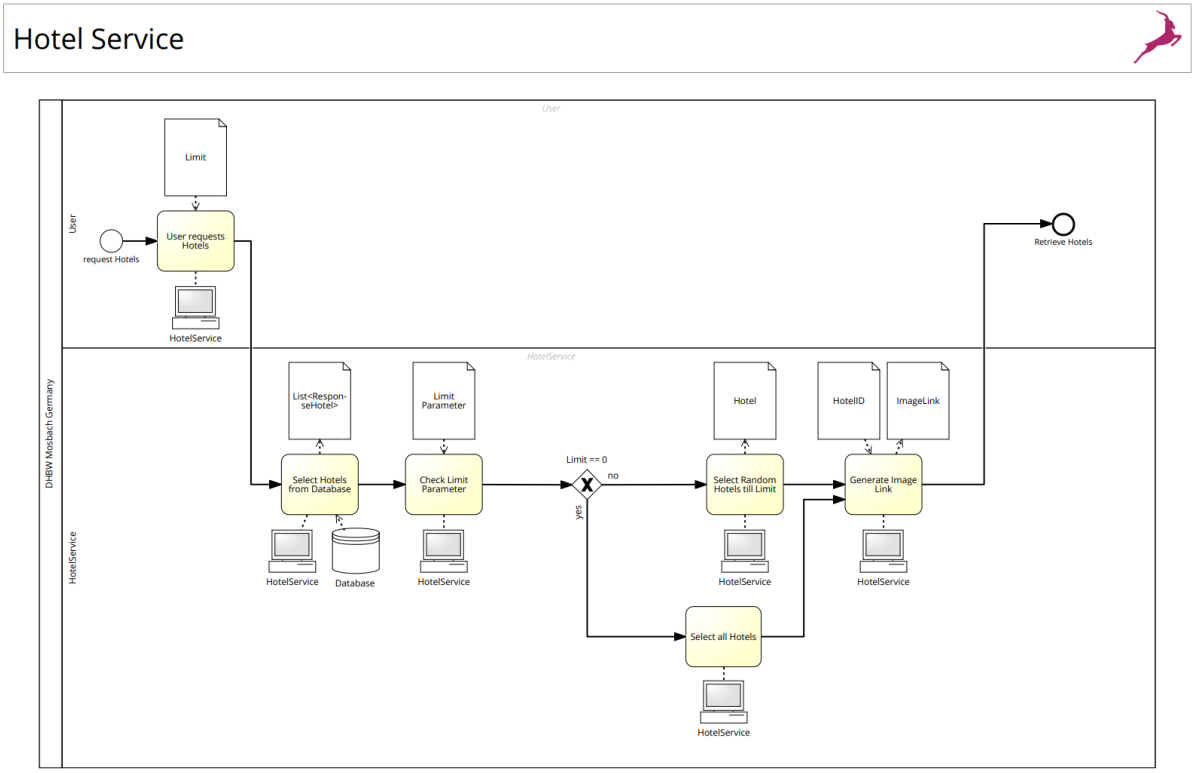

Abbildung 6: BPMN request Hotels

Der Hotel-Service verwendet noch den Trip Endpunkt 4.2.4 um bei Löschung eines Hotels auch die Trips, welche dieses Hotel beinhalten zu löschen.

Um die Geokoordinaten eines neu erstellten Hotels anhand der Adresse abzuspeichern, verwendet der Hotel-Service auch noch einen Endpunkt des Routen-Services 4.2.5, um dies durchzuführen.

Um einen genaueren Überblick über die verschiedenen API-Endpunkte des Services zu erhalten, ist die globale OpenAPI Dokumentation verfügbar.

4.2.2 User

Schnittstelle: „/users/api/v1“

Der User Service regelt alle Belange die Benutzer betreffen. Er ermöglicht es neue Benutzer anzulegen, einen Benutzer an- und abzumelden sowie Informationen über einen Benutzer zu erhalten. Auch die Verifikation des Authentifizierungstokens (Bearer Token) wird vom User Service übernommen.

Das Erstellen eines neuen Users erfolgt über den „/create“ Endpunkt. Hierzu müssen die Benutzerdaten gesendet werden (Name, E-Mail, Passwort). Ein Benutzer muss anschließend verifiziert werden. Zu diesem Zweck wir mittels des E-Mail-Service 4.2.3 eine E-Mail an den Benutzer versendet, die einen verifikationslink beinhaltet. Dieser Ablauf ist in Abbildung 7: BPMN Modell für Create User dargestellt.

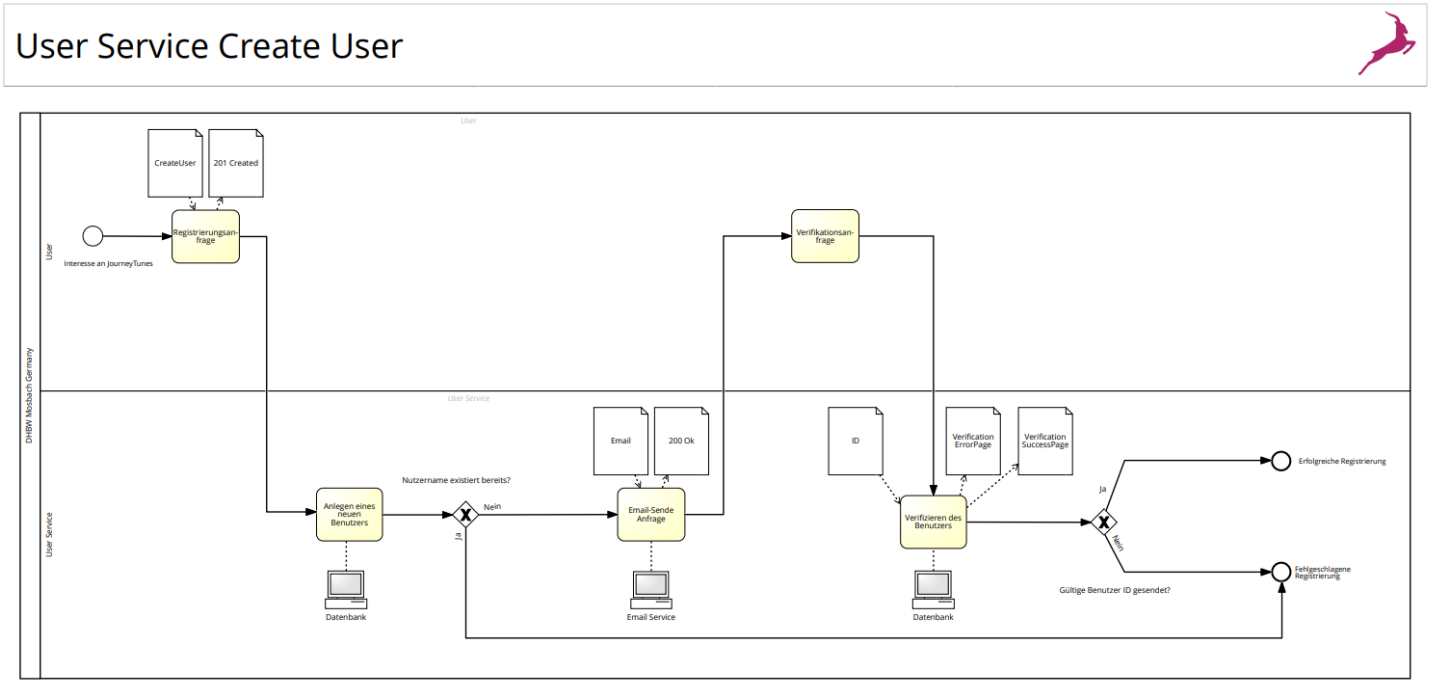

Abbildung 7: BPMN Modell für Create User

Nach erfolgreicher Verifikation kann sich der Benutzer anmelden und erhält ein Bearer Token, welches für eine halbe Stunde gültig ist.

Bei jeder weiteren API anfrage (z.B. an andere Services) wird das Token mitgesendet und vom User Service auf die Gültigkeit überprüft. Hierzu wird vom Gateway eine Anfrage an den User Service getätigt, die das gesendete Bearer Token beinhaltet.

4.2.3 Email

Schnittstelle: „/emails/api/v1“

Der E-Mail-Service kann E-Mails mittels eines SMTP-Servers verschicken. Dieser ist nicht im Service integriert, sondern muss gesondert bereitgestellt werden (lokal oder von einem Drittanbieter).

Um eine E-Mail zu senden, muss ein gültiges E-Mail-Dokument im JSON-Format gesendet werden welches den Empfänger „to“, den Betreff „subject“ und den Text im HTML-Format „body“ enthält.

Der E-Mail-Service fragt nun beim SMTP-Server mit einem Vordefinierten Benutzer und Passwort an die E-Mail zu versenden.

4.2.4 Trip

Schnittstelle: „/trips/api/v1“

Wenn der Benutzer sich für ein Hotel und eine Playlist entschieden hat und die Route berechnet wurde, kann der Nutzer die Auswahl bestätigen. Nun wird ein fertiger Trip erzeugt. Dies übernimmt der Trip Service. Es ist sowohl möglich Trips zu erzeugen als auch Trips zu erhalten.

Trips werden nicht anhand eines Trips oder User ID zurückgegeben, sondern anhand des Authorization Tokens, dass vom Userservice 4.2.2 eindeutig einem Benutzer zugeordnet wird.

Der Trip Service gibt dann einen vollständigen Trip zurück, bei dem die Hotel Referenzen aufgelöst sind. Dies ist dargestellt Abbildung 8.

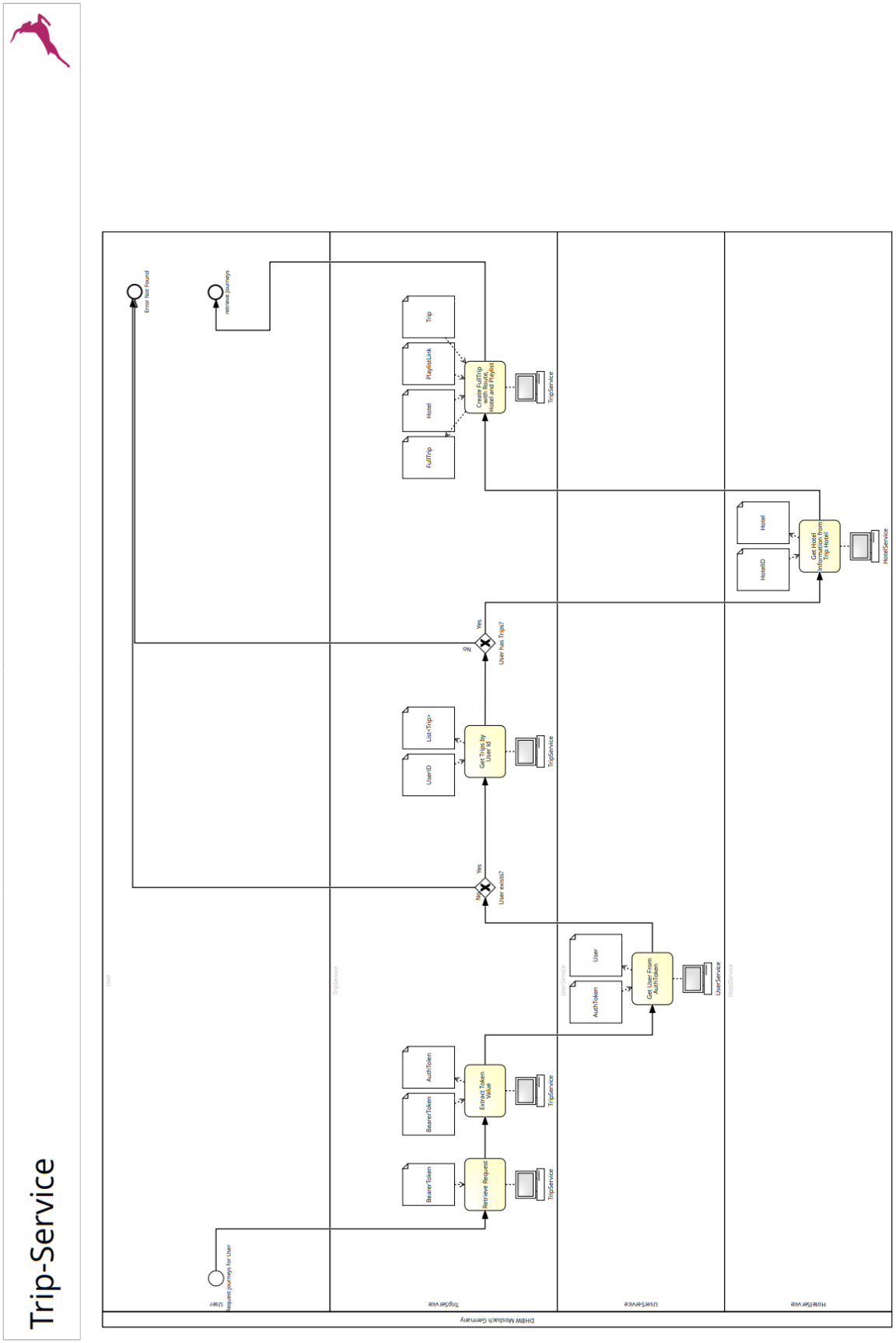

Abbildung 8: BPMN für Trip-Service /journeys

4.2.5 Route

Schnittstelle: „/routes/api/v1“

Der Route Service ist eine Verbindung zwischen einem externen Routenservice (in diesem Fall der Open Route Service) und der JourneyTunes Applikation.

Er bietet die Möglichkeit eine Route zwischen zwei Koordinaten (Längen- und Breitengrad) zu ermitteln „/create“. Zusätzlich lassen sich Längen und Breitengrad von einer gegebenen Adresse erfassen „/coordinates“.

4.2.6 Spotify

Schnittstelle: „/spotify/api/v1“
Um für eine musikalische Unterhaltung während des Trips zu sorgen, haben wir die SpotifyAPI in einen Service eingebunden.

Mithilfe von diesem Service ist es möglich für eine bestimmte Musik Kategorie und Länge, Spotify Playlists zu erhalten. Zusätzlich gibt es noch einen Endpunkt, welcher alle verfügbaren Spotify Musik Kategorien und ihre IDs zurückgibt, um im Frontend ganz einfach eine auswählen zu können.

Um einen genaueren Überblick über die verschiedenen API-Endpunkte des Services zu erhalten, ist die globale OpenAPI Dokumentation verfügbar.

5 Frontend

Dieser Abschnitt gibt einen Überblick über unser Frontend, das auf Vue.js und Vuetify mit Vite aufbaut, und erläutert, wie der AxiosClient die Backend-Integration vereinfacht. Wir stellen zudem unsere Views vor und beschreiben ihre Interaktion mit dem Backend.

5.1 Vue

Vue.js, oder kurz Vue, ist ein progressives JavaScript-Framework, das für die Erstellung von Benutzeroberflächen und Single-Page-Applications (SPA) verwendet wird.

Wir haben uns für dieses Framework entschieden, da Vue einfach in bestehende Projekte integriert werden kann. Außerdem zeichnet sich Vue durch eine reaktive und komponentenbasierte Architektur aus. Dadurch war es uns möglich, einzelne wiederverwendbare Komponenten zu erstellen, welche ihren eigenen Zustand, Layout und Verhalten haben. Beispielkomponenten in unserem Projekt sind „JourneyTunesTitle“ und „Trip“. Dies fördert die Wiederverwendbarkeit und Wartung des Codes. Weiterhin haben wir die reaktive Datenanbindung für automatische Anpassung der Benutzeroberfläche genutzt. Diese automatisierte Anpassung erfolgt, sobald sich die zugrunde liegenden Daten ändern, ohne dass manuelle Eingriffe erforderlich sind. Dies wird in Abbildung 9 gezeigt. Hier werden Hotel-E-Mail sowie Hotel-Telefonnummer dynamisch geladen und passen sich im UI automatisch an, sollten sich die zugehörigen Daten ändern.

```
<p class="pb-2 text-h6">Contact Data</p>
<p class="pb-2">{{hotel.email}}</p>
<p>{{hotel.phoneNumber}}</p>
```

Abbildung 9: Beispiel dynamischer Anpassung in Vue

5.2 Vite + Vuetify

Vite ist ein Frontend-Entwicklungstool, welches stark die ES Modules für das Laden von Modulen im Browser während der Entwicklungsphase nutzt. Vite führt nur minimale Pre-Bundling-Schritte durch und überlässt es dem Browser, den Großteil der Modulauflösung zu handhaben. Des Weiteren bietet Vite bietet auch eine schnelle Hot Module Replacement (HMR)-Funktionalität, die es ermöglicht, Änderungen im Code fast augenblicklich im Browser zu sehen, ohne die Anwendungsstate zu verlieren. Dies hat es uns ermöglicht, das Frontend effizienter zu implementieren.

Vuetify ist ein Material Design Framework für Vue. Es bietet eine umfangreiche Sammlung von vordefinierten Komponenten, die es Entwicklern ermöglichen, schnell und effizient ansprechende Webanwendungen zu erstellen. Wir haben uns insbesondere der Vielzahl von UI-Komponenten wie Buttons, Formulare und Dialoge bedient. Dies ermöglicht es uns, eine konsistente Benutzererfahrung zu gewährleisten. In

Abbildung 10 ist beispielsweise ein Button zu sehen, welcher die Vuetify-Komponenten v-btn beinhaltet.

```
<v-btn
  size="large" @click="selectHotel">
  Select Hotel
</v-btn>
```

Abbildung 10: Beispiel Vuetfiy Komponente Button

5.3 Tailwind CSS

Tailwind CSS ist ein Utility-first CSS-Framework, das Entwicklern ermöglicht, schnell benutzerdefinierte Designs direkt im HTML-Code zu erstellen, ohne separate CSS-Dateien schreiben zu müssen. Es besteht aus einer Vielzahl von Hilfsklassen für gängige CSS-Eigenschaften wie Margin, Padding, Schriftgröße, Farbe und Flexbox. Tailwind fördert einen Ansatz, bei dem Designänderungen durch einfaches Hinzufügen oder Entfernen von Klassen im Markup vorgenommen werden. Es bietet hohe Anpassbarkeit und ermöglicht es, unnötige Styles für die Produktion automatisch zu entfernen, um die Endgröße der Stylesheets zu minimieren. Die beispielhafte Nutzung von Tailwind CSS kann man in Abbildung 11 erkennen. Hier wird dem Paragrafen eine Style-Klasse hinzugefügt, in welcher die Farbe des Textes sowie dessen Schriftstil und Größe.

```
<div class="flex">
  <p class="text-deep-purple font-bold text-3xl">{{hotel.name}}</p>
  <div class="pl-4">
    <span class="text-3xl text-yellow-500" v-for="n in hotel.stars"
:key="n">★</span>
  </div>
```

Abbildung 11: Tailwind CSS beispiel

5.4 AxiosClient

Unsere Frontend verwendet die axios-Bibliothek, um eine Kommunikation mit dem Backend über HTTP-Anfragen zu ermöglichen. Durch die Verwendung von axios können wir sowohl die Anfragen an das Backend senden als auch die empfangenen Antworten verarbeiten.

Um eine konsistente Anfragenkonfiguration zu gewährleisten, haben wir uns für eine zentralisierte Axios-Instanz entschieden, die implementierung ist in Abbildung 12 zusehen. Diese zentrale Instanz ist mit Interzeptoren ausgestattet, diese haben eine wichtige Rolle in unserem Anfrage- und Antwortporzess.

Für jede ausgehende Anfrage fügen die Interzeptoren automatisch den Authorization-Header hinzu. Dieser enthält das Authentifizierungstoken (authToken), dass bei der Benutzeranmeldung im localStorage gespeichert wird und als Bearer-Token übertragen wird. Dieses Vorgehen stellt sicher, dass alle Anfragen authentifiziert sind.

Beim Empfang von Antworten überwacht ein weiterer Interzeptor den HTTP-Statuscode. Sollte der Server einen 401-Statuscode (Unauthorized) zurückliefern, handelt es sich wahrscheinlich um einen Zugriffsversuch durch einen nicht authentifizierten Benutzer. In diesem Fall wird der Benutzer automatisch ausgeloggt, der localStorage wird geleert, und der Nutzer wird auf die Anmeldeseite umgeleitet.

```
import axios from "axios";
import router from "@/router";

const axiosClient = axios.create({
  baseURL: 'http://10.50.15.51:8222/',
});

axiosClient.interceptors.request.use((config) => {
  const authToken = localStorage.getItem('authToken');
  if (authToken) {
    config.headers.Authorization = `Bearer ${authToken}`;
  }
  return config;
}, (error) => {
  return Promise.reject(error);
});

axiosClient.interceptors.response.use(response => {
  return response;
}, async error => {
  if (error.response && (error.response.status === 401)) {
    localStorage.clear();
    router.push('/login');
  }
  return Promise.reject(error);
});

export default axiosClient;
```

Abbildung 12: AxiosClient Code

5.5 Unsere Views

Die Frontend-Applikation ist in fünf Hauptansichten gegliedert: „Login“, „Register“, „Home“, „PlanYourTrip“, „Profile“ und „CreateHotel“. Diese Ansichten bilden die Schnittstellen, die der Benutzer direkt interaktiv nutzen kann. Innerhalb dieser Ansichten integrieren wir verschiedene Komponenten, wie beispielsweise die Darstellung eines Hotels oder die Visualisierung einer Reiseroute, um die Benutzerführung und das Gesamterlebnis der Website zu optimieren. Die Struktur und Funktionsweise der einzelnen Komponenten sind im Quellcode einsehbar.

In den genannten Views sowie teilweise in den Komponenten verarbeiten wir Datenobjekte im JSON-Format, um Inhalte dynamisch im Frontend darzustellen. Ein Beispiel hierfür ist die „HotelDetails“-Seite siehe Abbildung 13, die innerhalb der „Plan Your Trip“-Ansicht aufgerufen wird, sobald ein Nutzer auf ein Hotel klickt. Die Daten für das ausgewählte Hotel werden in Form eines Objekts an die Komponente übergeben und mithilfe eines Referenzobjekts angezeigt. Attribute des Hotelobjekts, wie zum Beispiel die Beschreibung, können direkt über hotel.description abgerufen

und im Frontend eingebunden werden. Diese Methodik ermöglicht es, die Benutzeroberfläche reaktiv zu gestalten und die Datenbindung zu vereinfachen.

```
<template>
  <div class="flex-column d-flex align-center">
    <div class="flex">
      <p class="text-deep-purple font-bold text-3xl">{{ hotel.name }}</p>
      <div class="pl-4">
        <span class="text-3xl text-yellow-500" v-for="n in hotel.stars"
:key="n">★</span>
      </div>
    </div>
    <div class="grid lg:grid-cols-2 sm:grid-cols-1 p-6 justify-center a-
lign-center">
      <div class="flex justify-center">
        <img :src="hotel.imageLink" height="80%" width="80%" class="roun-
ded-2xl" :alt="hotel.name">
      </div>
      <div class=" flex-col d-flex justify-center text-center">
        <p class="text-h5 pb-4 sm:pt-8"><strong>{{ hotel.pricePerNight }}
€</strong> night</p>
        <p class="text-h6">Description</p>
        <p class="pb-4">{{ hotel.description }}</p>
        <div class="grid grid-cols-2">
          <div class="pb-2">
            <p class="text-h6">Address</p>
            <p>{{ hotel.address }}</p>
          </div>
          <div>
            <p class="text-h6">{{ hotel.region }}</p>
          </div>
        </div>
        <p class="pb-2 text-h6">Contact Data</p>
        <p class="pb-2">{{ hotel.email }}</p>
        <p>{{ hotel.phoneNumber }}</p>

        <div class="flex pt-4 justify-center">
          <v-btn
            size="large" @click="selectHotel">
            Select Hotel
          </v-btn>
        </div>
      </div>
    </div>
  </div>
</template>
```

Abbildung 13: Beispiel für die Verwendung von JSON Objekten aus dem Backend in der Komponente HotelDetails

5.6 Verwendung des Backends im Frontend

Innerhalb des Frontends wird für die verschiedenen View Pages der zentraler AxiosClient genutzt, um entsprechend dem Kontext der jeweiligen Ansicht die notwendigen Anfragen an das Backend zu stellen und die erforderlichen Daten abzurufen. Der Datenaufruf erfolgt in der Regel während des initialen Ladeprozesses der Seite, implementiert durch die onMounted() Lifecycle-Hook in Vue.js. Als Beispiel dient die Profile.vue Seite, auf der Nutzerdaten, Reiseinformationen und vom Nutzer registrierte Hotels abgefragt und dargestellt werden.

Der dazugehörige Code in Abbildung 14 illustriert den asynchronen Prozess, der beim Einbinden der Seite ausgelöst wird. Zuerst wird das Authentifizierungstoken aus dem lokalen Speicher abgerufen. Sofern ein Token vorhanden ist, werden asynchrone HTTP-Anfragen gestellt, um Benutzerdaten, Tripdetails und

Hotelinformationen zu erhalten. Jede Antwort aktualisiert den entsprechenden reaktiven Zustand, repräsentiert durch user.value, trips.value und hotels.value. Dieser Ansatz ermöglicht eine klare Trennung zwischen der UI-Logik und den Datenabfragen.

```
onMounted(async () => {
  const authToken = localStorage.getItem('authToken');
  if (authToken) {
    try {
      const userResponse = await axiosClient.post('users/api/v1/get', { to-
ken: authToken });
      user.value = userResponse.data;

      const tripsResponse = await axiosClient.get('trips/api/v1/journeys');
      trips.value = tripsResponse.data;

      const hotelsResponse = await axiosClient.get(`hotels/api/v1/ow-
ner/${localStorage.getItem('userId')}`);
      hotels.value = hotelsResponse.data;
    } catch (error) {
      console.error('An error occurred:', error);
    }
  }
});
```

Abbildung 14: onMounted Hook der Seite Profile.vue

6 OpenAPI Spezifikation

Die OpenAPI Dokumentation ist in den einzelnen Services eingebunden. Das Gateway stellt durch spezielle Angaben in der Konfiguration, den Startpunkt, wie auch das Dropdown Menü mit den einzelnen Services zur Verfügung.

Die OpenAPI Spezifikation befindet sich unter der URL http://localhost:8222/swagger-ui.html oder für den Docker Host der DHBW unter http://10.50.15.51:8222/swagger-ui.html. Nach dem Aufruf einer der URLs landet man auf der Startseite.

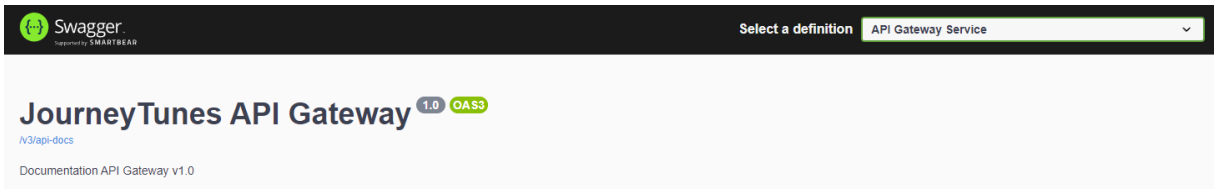

Abbildung 15: Swagger-UI unseres Gateways

Auf dieser Startseite in Abbildung 15 lässt sich durch das Dropdown Menü in der oberen rechten Ecke, die jeweilige Spezifikation für die einzelnen Services auswählen, dies ist dargestellt in Abbildung 16 .

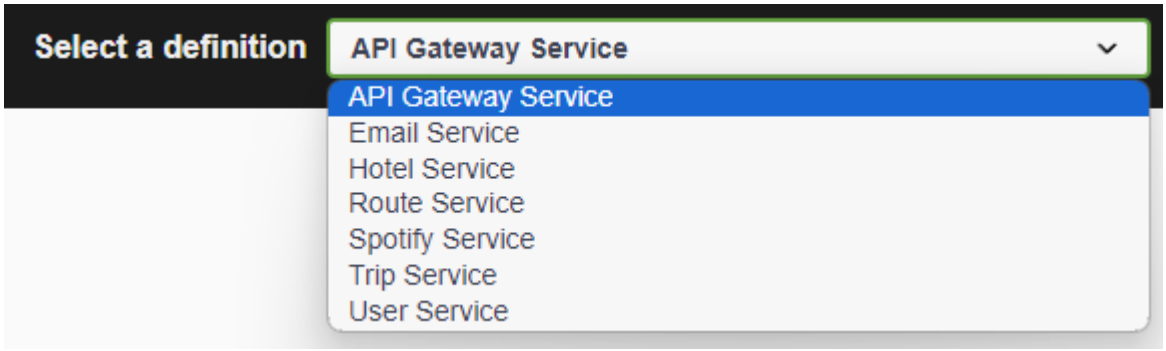

Abbildung 16: Auswahl der Services in der Swagger-UI

Nachdem ein Service, zum Beispiel der E-Mail-Service ausgewählt wurde, befindet man sich auf der Seite der Spezifikation des ausgewählten Services zu sehen in Abbildung 17.

Auf dieser Seite sieht man den Überblick über alle API-Endpunkte.

Um die Verschlüsselten Endpunkte verwenden zu können, muss über den User-Service durch den Login mit einem User Account (entweder einen neuen erstellen oder den auf dem Portainer in den vorbereiteten Daten erstellten User Name: root Pw: root verwenden) ein Token generiert werden, welches für die Authentifizierung verwendet wird.

Um die Endpunkte erfolgreich ausprobieren zu können, kann man zusätzlich entweder der Localhost oder den Portainer Server auswählen je nachdem wo die Services zu diesem Zeitpunkt gehostet werden.

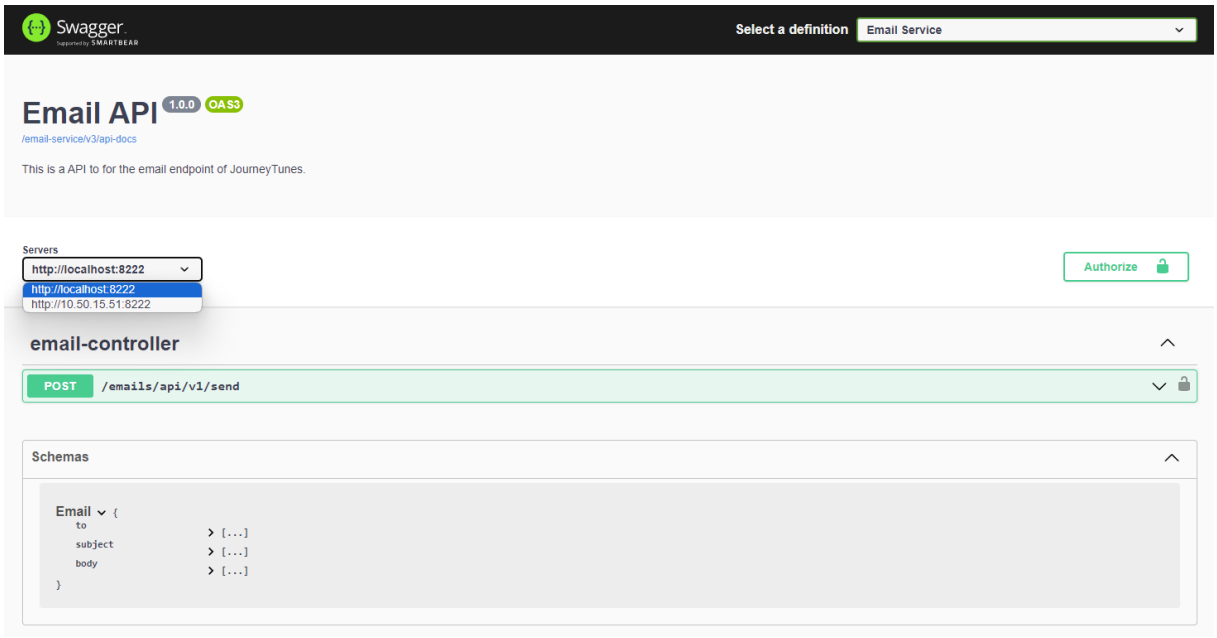

Abbildung 17: E-Mail-Service Swagger-UI

7 Docker

7.1 Dockerfile

Unsere Services sind mit Dockerfiles ausgestattet, die ihre Containerisierung ermöglichen und in der Docker-Compose-Datei referenziert werden. Diese Prozesse beginnen mit dem Herunterladen eines Maven-basierten Java 17 Images, gefolgt von der Kompilierung des Projekts.

Anschließend wird ein schlankes Java 17 Image verwendet, in dem Dockerize installiert ist, um die Abhängigkeitsverwaltung zu automatisieren und das Warten auf andere Services vor dem Start zu ermöglichen.

Die finale .jar-Datei wird in das Image kopiert und mithilfe von Dockerize gestartet, sobald die benötigten Services verfügbar sind, exemplarisch dargestellt in Abbildung 18 der Hotel-Service, der auf den Discovery-Server warten muss, bevor dieser gestartet werden kann.

```
# Builder stage
FROM maven:3.8.1-openjdk-17-slim AS builder

COPY . /usr/src/hotel
WORKDIR /usr/src/hotel

# Build your application
RUN mvn clean package

# Final stage
FROM openjdk:17-slim
VOLUME /tmp

# Install Dockerize
ENV DOCKERIZE_VERSION v0.6.1
RUN apt-get update && apt-get install -y wget \
    && wget https://github.com/jwilder/dockerize/releases/download/$DOCKER-
IZE_VERSION/dockerize-linux-amd64-$DOCKERIZE_VERSION.tar.gz \
    && tar -C /usr/local/bin -xzvf dockerize-linux-amd64-$DOCKERIZE_VER-
SION.tar.gz \
    && rm dockerize-linux-amd64-$DOCKERIZE_VERSION.tar.gz \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

# Copy the built application from the builder stage
ARG JAR_FILE=/usr/src/hotel/target/*.jar
COPY --from=builder ${JAR_FILE} app.jar

# The ENTRYPOINT runs your application
ENTRYPOINT ["dockerize", "-wait", "tcp://discovery:8761", "-timeout",
"60s", "java","-jar","/app.jar"]
```

Abbildung 18: Dockerfile vom Hotel Service

7.2 Docker-compose.yml

Die docker-compose.yml dient als Verwaltungswerkzeug der Microservice-Container. Sie enthält:

- Den Pfad zu den verschiedenen Untercontainern/
- Ob es sich um ein externes Image handelt
- Routing-Informationen (z.B. welche Ports nach Außen offen sind)
- Abhängigkeiten von Untercontainern, die nacheinander gestartet werden müssen
- Informationen zur Ressourcennutzung

Ein Exemplar eines Teils unserer Implementierung ist in Abbildung 19: Docker-compose.yml , zusehen hier ist beschriftet, was die jeweiligen Teilelemente für Funktionen haben. Die gesamte docker-compose.yml ist dem Quellcode zu entnehmen.

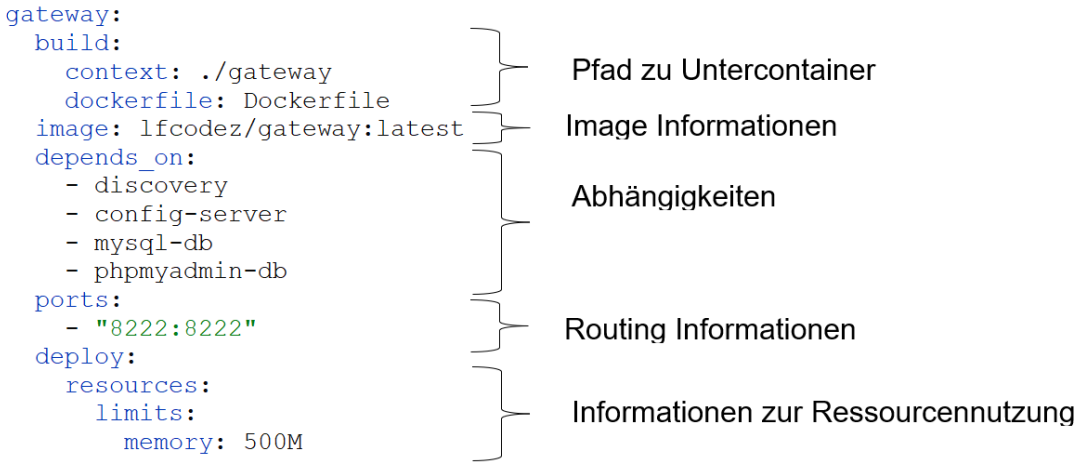

Abbildung 19: Docker-compose.yml

8 Zusatzinformationen

Um unsere Anwendung lokal lauffähig zu machen, kann man die gesamte Anwendung in dem höchsten Directory „project_journeytunes“ einfach über die docker-compose.yml mit dem Befehl: “docker-compose up -d --build” im Terminal bauen und starten.
Zusätzlich muss man jedoch, um alle Funktionen bereitstellen zu können, den Endpunkt der Bilder und der generierten E-Mail, sowie dem AxiosClient im Frontend auf localhost abändern.

Dies wird folgendermaßen gemacht:

1. Bilder-Endpunkt

„hotel/src/main/java/dhbw/mosbach/hotel/controller/HotelController.java“

Das Feld IMAGE_LINK anpassen zu:

```
private static final String IMAGE_LINK = "http://localhost:8222/ho-
tels/api/v1/hotels/";
```

2. E-Mail-Endpunkt

„user/src/main/java/dhbw/mosbach/user/controllers/UserController.java “

im Endpunkt @PostMapping(„/create“) muss der String emailBody zu folgendem abgeändert werden:

```
String emailBody = """…""".formatted("localhost", "8222", user.getId());
```

3. E-Mail-Server

„email/src/main/resources/application.yml“

Mail Parameter host, port, username und password auf gültige Email mit Provider ändern.

```
mail:
  host: smtp.gmail.com
  port: 587
  username: journeytunes3@gmail.com
  password: ahnokobgocciwqca
```

4. Frontend

„frontend-journeytunes/src/clients/axiosClient.js”

```
const axiosClient = axios.create({
  baseURL: 'http://localhost:8222/',
});
```

Sobald die Anwendung vollständig gestartet ist, sind folgende Endpunkte erreichbar:

Frontend	http://localhost:80
API Gateway	http://localhost:8222
Eureka Discovery Server	http://localhost:8761
phpMyAdmin	http://localhost:5050
Standardbenutzer	Username: **„root“**, Passwort: **„root“**