Softwaredokumentation

Inhaltsverzeichnis

Einleitung

Die vorliegende Webanwendung wurde entwickelt, um Schülern und Studenten einen effektive Plattform zum Lernen mithilfe von digitalen Karteikarten zu bieten. Das Hauptziel besteht darin, den Lernprozess zu optimieren und personalisierte Lernsessions basierend auf der individuellen Selbsteinschätzung des Nutzers zu ermöglichen.

Anforderungen und Anwendungsfälle

1. Benutzerregistrierung- und anmeldung
    - Benutzer können sich mit einem persönlichen Konto einloggen oder ein neues Konto erstellen, um die volle Funktionalität der Anwendung nutzen zu können.
2. Karteikartendeckverwaltung
    - Nach dem Login kann der Nutzer eigene Karteikartendecks anlegen, um thematisch organisierte Lerninhalte zu erstellen.
3. Hinzufügen von Karteikarten
    - Der Nutzer kann Textkarten oder Multiple-Choice-Karten zu einem Deck hinzufügen, um eine vielseitige Lernumgebung zu schaffen.
4. Lernsession durchführen
    - Nutzer können anschließend Lernsessions starten, in denen sie jede Karte bewerten können, indem sie Feedback-Buttons verwenden.
5. Donut-Chart zur Selbsteinschätzung
    - Am Ende einer Lernsession wird dem Nutzer eine Donut-Chart präsentiert, die einen visuellen Überblick über seine Selbsteinschätzung bietet.
6. Adaptive Lernerfahrung
    - Basierend auf der Selbsteinschätzung des Nutzers werden die Karten, die der Nutzer am schwierigsten empfand, in der nächsten Lernsession priorisiert.
7. Schnelle Lernsession mit “Quick Peek”
    - Für Nutzer, die eine schnelle Lernsession ohne Selbsteinschätzung wünschen, steht die Option “Quick Peek” zur Verfügung. Hierbei können alle Karten durchgesehen und mithilfe eines Next-Buttons navigiert werden.
8. Nutzerprofilverwaltung
    - Im Nutzerprofil können Einstellungen wie Benutzernamen, E-Mail-Adresse, Passwort, maximale Anzahl von Karten pro Lernsession, bevorzugte Sprache und Lernbenachrichtigungen angepasst werden.
9. Achievements
    - Durch Achievements wird die Lernerfahrung weiter individualisiert. Nutzer können durch das Erreichen bestimmter Erfolge und Meilensteine Auszeichnungen erhalten, die ihre Motivation steigern.

Wie kann man das Projekt ausführen?

Lokale Ausführung

Datenbank

Die lokale Ausführung beginnt bei der Datenbank. Hierfür muss Docker auf Ihrem PC installiert sein. Als Nächstes muss die ‘docker-compose.yml’-Datei ausgeführt werden. Die Datei liegt im folgenden Pfad: ‘**card-trainer\app\backend\docker-compose.yml’**. Diese Datei erstellt die Postgres-Datenbank auf Port 5433 mit einer Datenbank-Oberfläche (pgadmin) auf Port 16543. Die Anmeldedaten für die lokale Datenbank finden Sie in der .env Datei.

Ausführen des Backends

Wichtig ist, dass das Backend erst ausgeführt wird, wenn die Datenbank vollständig hochgefahren ist. Als Nächstes öffnet man den Pfad **‘card-trainer\app\backend’** in einer beliebigen IDE. Wir bevorzugen IntelliJ IDEA. Zuerst führt man die pom.xml-Maven-Datei aus (via. mvn:install) und lässt durch Maven alle ‘Abhängigkeiten’ installieren. Nachdem Maven fertig ist, führen Sie zuerst den Datenbank-Service aus und warten, bis dieser vollständig hochgefahren ist. Danach können Sie alle anderen Services ausführen.

Wichtig: Der Authentication-Service muss nach dem Database-Service starten, ansonsten hat dieser keinen Zugriff auf die Rainbow-Tabelle, was zu potenziellen Problemen führen kann.

Ausführen des Frontends

Als Nächstes öffnet man den Pfad **`'card-trainer\app\frontend'`** in einer beliebigen IDE. Wir bevorzugen IntelliJ WebStorm.

Schritt 1: Voraussetzungen überprüfen
Stellen Sie sicher, dass die folgenden Tools auf Ihrem System installiert sind:

- Node.js (https://nodejs.org/)
- Angular CLI (Command Line Interface), installiert über NPM (Node Package Manager) mit dem Befehl:
    - Installieren von Angular per commando: `'npm install -g @angular/cli'`

Schritt 2: Abhängigkeiten installieren
Installieren Sie die notwendigen Abhängigkeiten. Dies geschieht über den Befehl:
‘npm install’.
Dieser Befehl liest die package.json-Datei des Projekts und installiert alle notwendigen Pakete.

Schritt 3: Das Projekt starten
Nachdem alle Abhängigkeiten installiert wurden, können Sie das Projekt starten. Dazu verwenden sie den Befehl:
‘ng serve’.

App aufrufen

Die App ist nun deployed und über den Gateway-Service auf Port 80 im Browser erreichbar.

Guidelines

Shared Guidelines

- Git
    - Das Mergen von einer Feature-Branch in den anderen ist strengstens verboten, bspw. 111-feature-xyz -> 112-feature-abc.
      Es ist nur erlaubtvon einem geschützten Branch zu mergen, d.h., richtig wäre: bspw. 111-feature-xyz -> dev || dev -> 112-feature-abc.
    - Alle Bugs, Features etc. müssen immer über Issues durchgeführt werden.
    - Zu einem geschützten Branch wird nur über Merge-Requests gemerged. Hierbei muss ***immer*** mindestens eine andere Person “zustimmen” und mergen. **Niemals** selbst.
    - Generell gilt, Branches immer nur von “dev” abzweigen. Das direkte arbeiten in “dev” ist strengstens verboten.
    - Bugfix Branches werden immer mit: “[ISSUE-ID]-[bugfix]-[Describing Problem]” angelegt.
    - Kleine Hotfixes, die nur wenige Zeilen benötigen, können gegebenenfalls auch in einem anderen Branch integriert werden. Dies muss dementsprechend im Merge-Request angegeben sein!
    - Branches müssen immer auf ein Issue verlinkt sein. Falls in einem Branch mehrere Issues bearbeitet wurden, können auch mehrere Branches durch “Closes #[ISSUE-ID] Closes #[ISSUE-ID]” auch geschlossen werden.
    - Commit-Messages
        - WIP: Bedeutet, dass hier daran noch gearbeitet wird und es mit hoher Wahrscheinlichkeit noch keine lauffähige Version gibt.
        - FIX: Ein Problem wurde behoben. Nach einem Fix sollte immer in wenigen Worten beschrieben werden, was gefixt wurde.
        - ADDED/ADD: Bedeutet, dass neue Dateien hinzugefügt wurden, die für das aktuelle Issue oder für zukünftige Issues benötigt werden.
        - REMOVED: Bedeutet, dass Dateien, die nicht mehr benötigt werden, entfernt wurden.
        - REWORK/REFACTOR: Bedeutet, dass eine Neugestaltung oder Überarbeitung des Codes erfolgt, die über einfache Fehlerbehebungen oder Anpassungen hinausgeht.
        - TESTS/TEST: Hierbei wurden Tests geschrieben, die Funktionalitäten, die davor geschrieben worden sind, getestet haben.

      Sowohl FIX, ADDED/ADD, REMOVED; REWORK/REFACTOR als auch TESTS/TEST sind valide Branchprefixe
- Kommentare, Commits, Issues, Branchnamen, etc. **immer** auf Englisch!
- Issue Guidelines:
    - Beschreibende Titel verwenden
    - Imperativsätze, falls möglich. Bsp. “Revise the main figure” anstatt “main figure”
    - Ziele des Issues klar definieren!

- Pull Request mit Peer-Review initialisieren, damit der Issue-Branch in Dev gemerged wird.

Code Guidelines

Backend

- Controller sind verantwortlich für die Handhabung der HTTP-Layer, nur hier werden Routen exposed!
- Controller sollten wenn möglich keine Business-Logik enthalten!
- Use REST-Conventions[1]
- Service: Sollte um Business-Logik gebaut werden.
- Use Constructor injection!
- Write meaningful Unit-Tests
- An die Richtlinien von Java halten
- Schreib nur Kommentare, wenn es auch sinnig ist!
- Maximal 200 Zeichen pro Zeile
- Verwende CamelCase

Frontend

- Follow Naming Conventions
- Klare Ordnerstruktur
- One File per Object-Class
- Reuse Components if possible
- Use Interfaces
- Vermeide Callback-Hölle
- Vermeide type “any”
- Use Lazy Loading!
- Schreib nur Kommentare, wenn es auch sinnig ist!
- Write meaningful Tests
- An die Richtlinien von Typescript halten
- Maximal 200 Zeichen pro Zeile
- Verwende CamelCase
- Speichere nur Informationen ab, die wirklich benötigt werden

Security Guidelines

- BCrypt - Passwords should always be properly encoded
- JWT for secure api calls
- Hibernate, to prevent SQL Injections
- Preventing broken Authentication
- Preventing XSS (Cross-site-scripting)
- No sensitive data exposure
- Proper Access Control

[1] https://restfulapi.net/resource-naming/

Naming Convention Guidelines

Backend

- CamelCase
- Datentransfer-Objekte enden immer auf “DTO” bspw. LoginDTO
- Controller haben immer [Name]Controller, bspw. AuthController
- Konfigurationsfiles heißen immer [Name]Config bspw. AppConfig
- Services heißen immer [Name]Service bspw. DbQueryService
- Services, die mit anderen Microservices via HTTP kommunizieren, heißen immer [Name]QueryService
- Security related Komponenten müssen immer in einen Ordner namens “security”
- Injektion von anderen Services passiert immer via Konstruktor Injektion!
- In Payload kommen immer die DTOs, niemals Models!
- In Models kommen immer Objekte, die intern verwendet werden und nicht von außen kommen oder hereinkommen. Hinzu sollten in diesem Ordner niemals DTOs sein!
- Application.properties: Default Config für dev, Application-local für (prod),
    - Env Variablen aus den Properties, werden immer über den Konstruktor injected
- Testing Konventionen
    - Es werden lediglich Controller und Services getestet, DTOs werden anhand ihrer indirekten Abhängigkeit getestet.
    - Outbound Traffic zu anderen Microservices muss gemockt werden!
    - TestConfigs, werden immer in dieselbe Klasse geschrieben, außer sie werden in mehreren Klassen benötigt.
    - TestKlassen müssen immer im Ordner Test liegen und ihr Dateiname enden immer mit “.test”
- Ordnernamen werden immer kleingeschrieben!

Frontend

- CamelCase
- Services liegen immer im Ordner Services und haben dort einen eigenen Subordner mit ihrem Servicenamen, bspw.: user-service
- Komponenten sind beschreibend genannt und immer kleingeschrieben.
- Ordnernamen werden immer kleingeschrieben!

Infrastruktur Guidelines

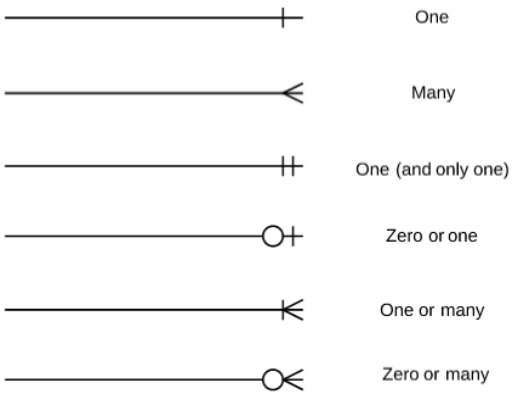
- Nur der Database-Service kommuniziert mit der Datenbank
- Das Frontend kommuniziert nur mit dem Gateway
- Nur eine Handvoll von Routen sind exposed und ohne Authentifizierung erreichbar, für alle anderen Routen wird eine entsprechende Authentifizierung (via E-Mail, Passwort) benötigt.

HTTP-Statuscode Guidelines

- 200 (OK): Falls eine Anfrage akzeptiert und verarbeitet worden ist + Response
- 201(Created): Neue Daten auf einem Service kreieren.
- 202 (Accepted)
- 204 (No Content): Anfrage akzeptiert, verarbeitet, aber keine Response
- 301 (Moved Permanently): Redirect zum Frontend
- 307 (Moved Temporarily): Redirect zum Frontend
- 400 (Bad Request): Daten im falschen Format/malformed oder erfüllt Standards nicht
- 401 (Unauthorized): Anfrage ohne Authentifizierung
- 404 (Not Found): Anfrage von Daten, die nicht existieren
- 409 (Conflict): Informationen existiert bereits
- 500 (Internal Server Error): Service nicht erreichbar oder anderweitige Probleme

UML Guidelines

- SnakeCase
- Fremdschlüssel werden immer als xyz_id# benannt
- Der folgenden Notation folgen:



- Tabellennamen immer groß schreiben
- Attribute immer klein!
- Primärschlüssel unterstreichen
- Sofern Linien überlappen, bitte via Sprünge kennzeichnen

Techstack

Backend

Unser Backend-Techstack besteht aus bewährten Technologien, die sich für die Entwicklung einer stabilen und skalierbaren Anwendungsarchitektur gut eignen.

Für unser Backend wird die Programmiersprache *Java* eingesetzt, da dieses ein großes Ökosystem an Bibliotheken und Frameworks besitzt.
In Kombination dazu werden wir das Framework '*Spring Boot*' verwenden, welches sich für die Entwicklung von Java-basierten Webanwendungen eignet und viele Funktionen für die Entwicklung von RESTful Services bereitstellt.
Für die Übertragung der Objekte wird *JSON* in Zusammenarbeit mit der Objectmapper Bibliothek verwendet.

Für eine reibungslose Kommunikation zwischen Backend und Frontend, insbesondere bei Benachrichtigungen wie "Achievement Unlocked", ist STOMP (Simple Text Oriented Messaging Protocol) die optimale Wahl. Unabhängig von der verwendeten Programmiersprache ermöglicht STOMP eine effiziente Verbindung zwischen den Komponenten einer Anwendung.

Als Datenbank haben wir uns für *PostgreSQL* entschieden. In Kombination dazu werden wir als ORM (Object-Relational Mapping) *Hibernate* einsetzen.
Hibernate ist ein leistungsstarkes Framework für Java, welches die Arbeit mit relationalen Datenbanken vereinfacht, indem es die Kommunikation zwischen Java-Objekten und der Datenbank abstrahiert.
Für das Validieren von Benutzereingaben innerhalb unseres Backends werden wir *Jakarta* verwenden. Dadurch stellen wir sicher, dass die Daten auch den erwarteten Anforderungen entsprechen.

Zum Testen der Software wird *JUnit* verwendet, welches ein weit verbreitetes Framework zur Testautomatisierung von Java-Applikationen ist. Es ermöglicht die einfache Erstellung von Unit-Tests und automatisierten Tests, um die Zuverlässigkeit des Codes sicherzustellen.

Für die Dokumentation unserer API wird *Swagger* eingesetzt. Swagger erleichtert die Erstellung von detaillierten API-Dokumentationen. Die benutzerfreundliche Oberfläche spiegelt die API-Endpunkte wider und erleichtert die Interaktion zwischen den Entwicklern.

Für den Build unserer Applikation wird *Maven* verwendet. Maven ist ein Build- und Dependencies-Management-Tool, welches Abhängigkeiten verwaltet und den Build-Prozess erleichtert.

Für die konsistente Bereitstellung unseres Backends und ihrer Abhängigkeiten in Containern verwenden wir Docker. Docker erleichtert sowohl die Entwicklung als auch die Skalierung signifikant. Zudem ermöglicht es uns, unsere Anwendung leichter zu deployen.

Frontend

Für das Frontend haben wir uns für einen robustes und modernes Technologiestack entschieden, das sich durch Flexibilität und Skalierbarkeit auszeichnet und eine reibungslose Nutzererfahrung gewährleisten soll.

Node.js
Als serverseitige JavaScript-Laufzeitumgebung erlaubt Node.js die Ausführung von JavaScript außerhalb des Webbrowsers. Insbesondere im Frontend ist Node.js die Grundlage für npm, ein Werkzeug zur Verwaltung von externen JavaScript-Bibliotheken und Abhängigkeiten.

npm
npm (Node Package Manager) ist ein leistungsfähiges Paket-Verwaltungstool, das speziell für die Node.js-Umgebung entwickelt wurde. Die Verwendung von npm ermöglicht es uns, auf eine breite Palette von vorgefertigten Modulen zuzugreifen, was die Entwicklung beschleunigt und zugleich sicherstellt, dass wir stets auf aktuelle und stabile Bibliotheken zurückgreifen. Die klare Definition und einfache Integration von Drittanbieterpaketen tragen zu einem gut strukturierten, wartbaren und ressourceneffizienten Code bei.

Angular 17
Angular ist ein leistungsstarkes Frontend-Framework, das von Google entwickelt wurde. Mit der Version 17 haben wir Zugriff auf die neuesten Funktionen und Verbesserungen, die die Entwicklung von komplexen Single-Page-Anwendungen erleichtern. Angular bot uns eine klare Struktur für die Organisation des Codes und ermöglichte die einfache Integration von Komponenten, Diensten und Routen.

Bootstrap 5
Bootstrap ist ein Open-Source-Framework für das Frontend-Design und erleichtert die Entwicklung von responsiven und ästhetisch ansprechenden Benutzeroberflächen maßgeblich. Durch die Integration dieses Frameworks haben wir Zugriff auf eine umfangreiche Sammlung von CSS-Komponenten und -Utilities, die zusätzliche Konsistenz im Design gewährleisten. Insbesondere für die Entwicklung von responsiven Anwendung ist Bootstrap 5 gut geeignet.

Cypress
Cypress ist ein leistungsstarkes End-to-End-Testframework und ermöglicht uns, umfangreiche automatisierte Tests für unsere Benutzeroberfläche zu erstellen. Die leichte Konfiguration und das klare Reporting erleichtern die Wartung der Testfälle.

TypeScript
TypeScript ist eine typisierte Erweiterung von JavaScript, die die Codequalität verbessert, indem sie statische Typisierung ermöglicht. Durch die strikte Typisierung wird der Entwicklungsprozess robuster, indem potenzielle Fehlerquellen frühzeitig erkannt und behoben werden. Somit trägt TypeScript wesentlich zur Verbesserung der Zuverlässigkeit unserer Anwendung bei.

CSS und HTML
CSS (Cascading Style Sheets) und HTML (Hypertext Markup Language) sind grundlegende Bausteine für das Content unserer Anwendung. CSS ermöglicht uns das Styling der Anwendung, während HTML die Struktur und den Inhalt definiert. Die Kombination beider Technologien ermöglicht eine klare Trennung von Design und Inhalt.

FontAwesome
FontAwesome ist eine Bibliothek von vektorbasierten Icons, die wir für eine visuell ansprechende Benutzeroberfläche einsetzen. Durch die Integration von FontAwesome können wir Icons leicht einbinden und das Benutzererlebnis unserer Anwendung verbessern.

Projektstruktur

Backend

Die Backend-Projektstruktur ist systematisch in verschiedene Module unterteilt, um eine klare Organisation und Skalierbarkeit zu gewährleisten. Jedes Modul, angefangen beim Authentication-Service über den Database-Service bis hin zu den weiteren Services, spielt eine entscheidende Rolle im Gesamtsystem. Die Verwendung von Docker-Containern und Docker Compose ermöglicht eine einfache Bereitstellung und Entwicklungsumgebung.

Das Backend beinhaltet ebenso mehrere Microservices, siehe weiter unten. Jeder Service besitzt einen Ordner namens "Configuration". In diesem Ordner werden benötigte Konfigurationen, wie AppConfig bereitgestellt. AppConfig beinhaltet beispielsweise eine Bean namens RestTemplate, welche dafür da ist, HTTP-Calls für andere Services bereitstellen zu können.

Zudem besitzt jeder Microservice immer mindestens einen Controller, bei dem Routen für andere Services bereitgestellt werden. Außerdem gibt es einen optionalen Models-Ordner, welcher entsprechende interne Modelle beinhaltet.

Klassenstruktur

Achievement-Service
Der Achievement-Service beinhaltet den Code für den Achievements-Service, welcher darauf ausgerichtet ist, Erfolge oder Leistungen innerhalb des Gesamtsystems zu verwalten. Dieses Modul stellt Funktionalitäten bereit, die es ermöglichen, Benutzer für das Erreichen bestimmter Meilensteine oder Aufgaben zu belohnen. Durch die klare Trennung dieses Services wird eine saubere Struktur geschaffen, die es erleichtert, Achievements in das Gesamtsystem zu integrieren und sie separat zu verwalten. Dies fördert eine bessere Wartbarkeit und Erweiterbarkeit des Systems.

Authentification-Service
Der Authentication-Service bildet den zentralen Baustein der Backend-Projektstruktur und ist strukturiert in verschiedene Unterverzeichnisse, die eine modulare Entwicklungsumgebung gewährleisten.
Im ".mvn"-Ordner sind Maven-spezifische Dateien für das Build-Management enthalten. Der Hauptcode befindet sich in "src/main/java/com/service/authenticationservice". Hier beherbergt das "configuration"-Unterverzeichnis die Konfigurationsklasse (AppConfig.java), während "controller" den AuthController für die Authentifizierung enthält (siehe auch oben). Im "model"-Unterverzeichnis sind Definitionen für verschiedene Modelle wie MailType zu finden, während "payload" Klassen für Inbound und Outgoing DTOs ist. In "security/jwt" ist die Implementierung für JSON Web Token (JWT) untergebracht, und im "services"-Ordner befinden sich verschiedene Service-Klassen für Datenbankabfragen, E-Mail-Validierung und Sicherheit. Die Hauptklasse für die Spring Boot-Anwendung trägt den Namen "AuthenticationServiceApplication.java". Das "src/main/resources"-Verzeichnis enthält Anwendungs- und lokale Konfigurationsdateien, und im "src/test/java/com/service/authenticationservice"-Ordner sind Testklassen für die einzelnen Komponenten der Anwendung zu finden.

Cards-Service
Der Cards-Service innerhalb der Backend-Klassenstruktur ist zuständig für die Verarbeitung von Karteikartendaten. Dieser Service beinhaltet den Code, der für die Verwaltung und Manipulation von Karteikarten benötigt wird. Das Modul umfasst Funktionen, die das Erstellen, Aktualisieren und Löschen von Karten ermöglichen, sowie die Durchführung von Abfragen und Operationen im Zusammenhang mit Karteikarten. Die Abgrenzung dieses Services ermöglicht Kartenfunktionalitäten separat zu skalieren oder zu aktualisieren, ohne die Integrität anderer Systemkomponenten zu beeinträchtigen.

Database
Die Database ist zentraler Bestandteil des Projekts und enthält wichtige Ressourcen für die Datenbankinfrastruktur. Data wird vordergründig für die persistente Datenspeicherung verwendet. Hier werden beispielsweise Datenbanktabellen, Indizes und andere persistente Datenstrukturen abgelegt.
Die "devdbypass"-Konfigurationsdatei ermöglicht eine lokale Datenbankumgehung während der Entwicklung. Das SQL-Skript “init.sql” wird während der Initialisierungsphase der Anwendung ausgeführt. Die Datei "servers.json" ist eine Konfigurationsdatei, die Informationen über Datenbankserver enthält. Hier werden Parameter wie Hostnamen, Ports oder Zugangsdaten festgelegt, um eine Verbindung zur Datenbank herzustellen. Diese klare Strukturierung ermöglicht eine einfache Konfiguration und Wartung der Datenbankverbindungen im gesamten System.

Database-Service
Das "database-service"-Modul beinhaltet den essenziellen Code für den Database-Service, der eine zentrale Rolle in der Datenverwaltung des Gesamtsystems spielt. Dieser Service abstrahiert Datenbankoperationen, um eine einheitliche Schnittstelle für andere Module bereitzustellen. Durch klare Trennung von Datenbankzugriffsfunktionalitäten in einem dedizierten Service wird die Wartbarkeit und Erweiterbarkeit des Systems verbessert. Das Modul umfasst Funktionalitäten, die das Ausführen von Datenbankabfragen, das Aktualisieren von Datensätzen und das Verwalten von Transaktionen gewährleisten.

Gateway-Service
Der Gateway-Service bildet den zentralen Zugangspunkt für externe Anfragen. Als zentraler Einstiegspunkt ermöglicht das Gateway den Zugriff auf verschiedene Microservices im Backend und sorgt für eine einheitliche und standardisierte Kommunikation zwischen ihnen. Das Gateway-Service ermöglicht eine kohärente und einheitliche Schnittstelle für externe Anfragen, beispielsweise durch das effiziente Routing von Anfragen externer Clients und die Weiterleitung auf die entsprechenden Services.

Mail-Service
Der Mail-Service spielt eine entscheidende Rolle bei der Handhabung von E-Mail-Funktionalitäten innerhalb des Backends. Dieser Service ist verantwortlich für Aufgaben wie die Versendung von E-Mail-Benachrichtigungen, Passwortrücksetzungen und andere mit der Kommunikation per E-Mail verbundene Prozesse. Die Isolierung des Mail-Services ermöglicht Wartbarkeit und Skalierbarkeit sowie die einfache Anpassung an unterschiedliche E-Mail-Protokolle.

Session-Service
Der Session-Service spielt eine zentrale Rolle in der Verwaltung von Nutzersitzungen. Dieser Service stellt Funktionalitäten bereit, um Sitzungen zu erstellen, zu verwalten und zu überwachen, was insbesondere wichtig ist, um den Zustand von Benutzersitzungen über mehrere Anfragen hinweg zu erhalten. Die klare strukturelle Trennung ermöglicht eine fokussierte Entwicklung und Wartung des Session-Services. Der Session-Service selbst beinhaltet Module für die Erstellung von Sitzungen, das Handling von Ablaufzeiten, die Validierung von Sitzungsinformationen und Sicherheitsaspekte wie die Session-Token-Generierung. Seine modulare Struktur ermöglicht es, den Session-Service einfach zu skalieren und zu aktualisieren, ohne das Gesamtsystem zu beeinträchtigen.

User-Service
Der User-Service ist von entscheidender Bedeutung für die Verwaltung von Benutzerinformationen innerhalb des Gesamtsystems. Dieser Service beinhaltet Funktionalitäten, die die Erstellung, Aktualisierung und Löschung von Benutzerprofilen sowie Authentifizierung und Autorisierung bereitstellen. Durch die klare Abgrenzung dieses Services wird eine modulare und skalierbare Struktur geschaffen, die es ermöglicht, benutzerbezogene Operationen effizient zu verwalten. Durch die mögliche Implementierung von Schnittstellen zu anderen Services wie dem Authentication-Service oder dem Database-Service ermöglicht der User-Service eine nahtlose Integration in das Gesamtsystem.

Frontend

Projektstruktur und Standards

Die Architektur Frontends basiert auf Angular 17 und folgt einer klaren Strukturierung von Komponenten, um eine klare Trennung von Verantwortlichkeiten und eine leicht verständliche Codebasis zu gewährleisten. Die modulare Struktur der Komponenten ermöglicht eine einfache Anpassung und Erweiterung der Funktionalitäten.

```
/src
|-- /app
|   |-- /components
|   |   |-- achievement-modal
|   |   |-- base-page
|   |   |-- deck-list-view
|   |   |-- deck-view
|   |   |-- donut-chart
|   |   |-- edit-card-view
|   |   |-- edit-deck-view
|   |   |-- forgot-password
|   |   |-- history-view
|   |   |-- learn-card-view
|   |   |-- login
|   |   |-- navbar
|   |   |-- peek-learn-card-view
|   |   |-- register
|   |   |-- register-successful
|   |   |-- reset-password
|   |   |-- session-renewal-modal
|   |   |-- swagger-view
|   |   |-- toast
|   |   |-- toaster
|   |   |-- tutorial
|   |   |-- user-profile
|   |   |-- verify-successful-view
|   |   |-- /services
|   |   |   |-- achievement.service
|   |   |   |-- authentication.service
|   |   |   |-- card.service
|   |   |   |-- error-handler.service
|   |   |   |-- history.service
|   |   |   |-- toast.service
|   |   |   |-- tutorial.service
|   |   |   |-- user.service
|   |   |   |-- websocket.service
|   |-- app.routes.ts
|-- /assets
|-- /...
```

Komponentenstruktur

Achievement Modal
Die Achievement Modal-Komponente hat den Zweck, dem Nutzer errungene Auszeichnungen oder Erfolge anzuzeigen, sobald er das Achievement erworben hat. Als modales Element wird sie situativ in der Anwendung ausgelöst, um besondere Leistungen hervorzuheben.

Base Page
Die Base Page-Komponente bildet die Grundstruktur für Seiten in der Anwendung. Sie dient als Basis, auf der andere Seitenkomponenten aufbauen und ermöglicht eine konsistente Gestaltung der Webseiten.

Deck List View
Die Deck-List View-Komponente bietet dem Nutzer eine Liste der verfügbaren Karteikartendecks und ist zugleich die Hauptansicht der Anwendung. Sie stellt eine zentrale Schnittstelle innerhalb der Anwendung dar, da sie den nahtlosen Zugang zu anderen Komponenten und Funktionalitäten ermöglicht. Durch die Integration von Funktionen zur Erstellung, Löschung und zum

Importieren von Decks wird zusätzlich ein umfassendes Deck-Management innerhalb der Komponente selbst ermöglicht.

Deck View
Die Deck View ermöglicht einen detaillierten Einblick in ein ausgewähltes Karteikartendeck. Bei Auswahl eines Decks werden relevante Informationen wie der Decktitel (bearbeitbar), die Deckgröße und die noch zu lernenden Karten übersichtlich dargestellt. Durch einen Donut Chart wird die Selbsteinschätzung des Nutzers aus bisherigen Lernsessions visualisiert. Von hier aus kann der Benutzer Lernsessions oder ein Quick Peek beginnen, das Deck exportieren, teilen und Karten im Deck bearbeiten, während der Wechsel zur History View einen Überblick über abgeschlossene Lernsessions bietet. Die Deck View fungiert somit als zentrale Anlaufstelle für die Verwaltung und das Vertiefen des Wissens in einem ausgewählten Karteikartendeck innerhalb der Gesamtapplikation.

Donut Chart
Die Donut-Chart-Komponente visualisiert die Selbsteinschätzung des Nutzers mithilfe eines Donut-Diagramms. Nach einer Lernsession und in der Deck-View-Komponente wird dem Nutzer anhand dieser Grafik ein Überblick über seine Einschätzung präsentiert.

Edit Card View
Die Edit Card View-Komponente spielt eine zentrale Rolle, indem sie dem Nutzer die Möglichkeit gibt, Karten innerhalb eines Decks zu bearbeiten oder eine neue zu erstellen. Er hat innerhalb der Komponente die Wahl zwischen einer Basic- und Mutiple-Choice-Karten. Im Fall einer Multiple Choice-Karte ermöglicht die Komponente dem Nutzer zusätzlich, Antwortoptionen einzugeben. Zusätzlich bietet sie die Funktion, Bilder für die Frage, die Antwort und die Antwortoptionen hochzuladen mithilfe eines Drag-and-Drop-Feature. Ermöglicht wird dieses Feature durch die Funktionen `onDragOver()` und `onDrop()`, welche für die Behandlung von Drag-and-Drop-Ereignissen verantwortlich sind. Sie sorgen dafür, dass der Upload-Bereich korrekt auf das Hineinziehen der Bilddatei reagiert. Die Methode `openFilePicker()` öffnet den Dateiauswahldialog, wenn der Nutzer den entsprechenden Link anklickt. Die Methode `onFileSelected()` wird aufgerufen, wenn der Nutzer eine Bilddatei über den Dateiauswahldialog ausgewählt hat. Die Edit Card View-Komponente fügt sich so nahtlos in den Bearbeitungsfluss innerhalb der Anwendung ein

Edit Deck View
Die Edit Deck View-Komponente zeigt eine Liste von Karten innerhalb eines Karteikartendecks an. Diese kann während des Bearbeitens des Karteikartendecks verwendet werden und bietet eine übersichtliche Darstellung der enthaltenen Karten.

Forgot-Password
Die Forgot Password-Komponente gestattet es Nutzern, ihr vergessenes Passwort zurückzusetzen. Der Forgot Password Screen ist ein integraler Bestandteil der Authentifizierungsfunktionen der Anwendung.

History
Die History-Komponente bietet einen übersichtlichen Überblick über die Lernhistorie des Nutzers. Damit bietet die Komponente eine zentrale Anlaufstelle für den Benutzer, um seinen Fortschritt vergangener Lernsessions und erreichte Meilensteine einzusehen.

Learn Card View
Die Learn Card Screen-Komponente spielt eine zentrale Rolle für den Lernprozess innerhalb der Anwendung. Die Komponente ermöglicht eine interaktive Lernerfahrung durch die Implementierung eines Flip-Mechanismus. Dieser Mechanismus wird durch die Methode flipCard() gesteuert, die durch das Klicken auf einen "Rotate Button" aufgerufen wird, der es dem Nutzer erlaubt, die Karte zu drehen und zwischen der Vorder- und Rückseite hin und her zu wechseln. Diese Methode ändert den Status der Karte (flipped), wodurch die Darstellung zwischen Vorder- und Rückseite umgeschaltet wird. Die CSS-Stile definieren schließlich die Positionierung der Vorder- und Rückseite der Karte sowie die Animation des Flippens. Die Klasse .flipped wird dynamisch hinzugefügt oder entfernt, um den Flip-Effekt zu erzeugen. Der Nutzer hat außerdem die Möglichkeit, eine Selbsteinschätzung abzugeben. Diese unmittelbare Rückmeldung beeinflusst die zukünftige Präsentationsreihenfolge der Karten, wodurch die Lerneffizienz gesteigert wird. Somit ermöglicht die Learn Card Screen-Komponente eine individualisierte und anpassbare Lernerfahrung innerhalb der Gesamtapplikation.

Login
Die Login-Komponente ist zuständig für die Authentifizierung von Benutzern. Durch ihre zentrale Einordnung gewährleistet die Login-Komponente einen Authentifizierungsprozess als Schlüsselkomponente der Anwendung.

Navbar
Die Navbar-Komponente stellt eine Navigationsleiste für die gesamte Anwendung bereit und wird auf jeder Seite angezeigt, um eine einfache Navigation zu gewährleisten.

Peek Learn Card View
Die Peek Learn Card Screen-Komponente fügt sich als integraler Bestandteil in die Anwendung ein, indem sie einen alternativen Lernmodus für Benutzer bereitstellt. Sie ermöglicht ein rasches Durchblättern von Karten ohne die Notwendigkeit einer Selbsteinschätzung.

Register
Die Register-Komponente übernimmt eine zentrale Rolle in der Anwendung, indem sie neuen Nutzern die Möglichkeit bietet, sich zu registrieren. Diese Komponente ist ein integraler Bestandteil des Registrierungsprozesses und nimmt eine Schlüsselrolle in der Benutzerauthentifizierung ein. Nach erfolgreicher Registrierung erhält der Nutzer automatisch eine Verifizierungsmail, die den Registrierungsprozess abschließt. Sie ermöglicht die Erfassung von Benutzerinformationen und trägt maßgeblich zur Sicherheit und Identitätsverwaltung der Gesamtanwendung bei.

Register Successful

Die Register Successful-Komponente bestätigt die erfolgreiche Registrierung eines neuen Benutzers und wird nach einer erfolgreichen Registrierung angezeigt.

Reset Password
Die Reset Password-Komponente bildet eine entscheidende Schnittstelle innerhalb der Anwendung, die es Benutzern ermöglicht, ihr Passwort zurückzusetzen, weshalb sie einen essentiellen Bestandteil der Sicherheits- und Benutzerverwaltungsfunktion der Gesamtanwendung darstellt.

Session Renewal Modal
Das Session Renewal Modal wird aktiviert nach längerer Inaktivität und zeigt dem Benutzer ein Modal an, das die Optionen zum Fortsetzen der Sitzung oder zum Ausloggen präsentiert.

Swagger View
Die Swagger View-Komponente fungiert als Schnittstelle zur API-Dokumentation und ermöglicht eine transparente und effiziente API-Nutzung. Durch die Integration von Swagger erhalten Entwickler einen detaillierten Überblick über die verfügbaren Endpunkte, Parameter und Rückgabewerte, was die Entwicklung und Wartung der Anwendung erleichtert.

User Profile
Die User Profile-Komponente ermöglicht es dem Nutzer, sein Benutzererlebnis individuell anzupassen. Hier kann er nicht nur grundlegende Informationen wie seinen Benutzernamen und seine E-Mail-Adresse ändern, sondern auch sein Passwort aktualisieren. Zudem kann der Nutzer die maximale Anzahl von Karten festlegen, die er während einer Lernsession bewältigen möchte. Die Wahl der präferierten Sprache verbessert die Benutzerfreundlichkeit, während die Option zur Aktivierung von Lernbenachrichtigungen eine personalisierte Lernerfahrung schafft. Durch ihre Integration in die Navigationsleiste fungiert sie als Anlaufstelle für die Anpassung von Benutzerdaten und -einstellungen.

Verify Successful
Die Verify Successful View-Komponente bestätigt die erfolgreiche Überprüfung einer Benutzerregistrierung. Sie ist ein integraler Bestandteil des Registrierungsprozesses innerhalb der Gesamtanwendung. Toast
Die Toast-Komponente ermöglicht die Anzeige von Benachrichtigungen oder Warnungen innerhalb der Anwendung. Sie erscheint kontextsensitiv und kann durch verschiedene Aktionen innerhalb der Anwendung getriggert werden, wie beispielsweise Bestätigungen nach einer erfolgreichen Aktion oder Warnungen bei fehlerhaften Eingaben.

Toaster
Die Toaster-Komponente ist ähnlich wie der Toast, bietet jedoch erweiterte Funktionalitäten und kann für komplexere Benachrichtigungen verwendet werden.

Tutorial
Die Tutorial-Komponente wurde entwickelt, um neuen Benutzern einen interaktiven Leitfaden bereitzustellen und sie durch die wichtigsten Funktionen der Anwendung zu

führen. Diese Komponente ermöglicht eine schnelle Orientierung innerhalb der Gesamtanwendung.

Services

Achievement-Service
Der Achievement-Service kümmert sich um die Interaktion mit dem Backend für alle Achievement-bezogenen Aktionen. Er stellt Methoden bereit, um detaillierte Informationen zu einem bestimmten Achievement abzurufen unter der Route `` `/api/v1/achievements/${id}` ``. Schließlich wird das dazu korrespondierende Bild heruntergeladen unter der Route `` `api/v1/images/${imageId}` ``. Dies erfolgt beispielsweise durch die Nutzung von Angular´s HttpClient-Modul und Observable für eine reaktive Programmierung. Durch die zentrale Verwaltung von Achievement-relevanten Daten und Bildern bietet er anderen Frontend-Komponenten die Möglichkeit, mühelos auf diese Informationen zuzugreifen und sie in der Benutzeroberfläche darzustellen.

Auth-Service
Der Auth-Service übernimmt die Verwaltung der Benutzerauthentifizierung innerhalb der Anwendung. Der Service bietet Funktionen zur Überprüfung des Authentifizierungsstatus unter der Bedingung `'/api/v1/account'`, zum Abmelden von Benutzern unter der POST-Route `'/api/v1/logout'`, zur Verwaltung von JWT-Cookies und zur Initiierung eines Session-Timers. Der AuthService spielt eine zentrale Rolle im Gesamtsystem, indem er sicherstellt, dass Benutzer authentifiziert und autorisiert sind. Er integriert sich nahtlos in andere Frontend-Komponenten, indem er den Authentifizierungsstatus überwacht und bei Bedarf Modalitäten zur Erneuerung der Sitzung bereitstellt.

Cards-Service
Der Cards-Service im Frontend agiert als zentrale Schnittstelle zwischen der Anwendungslogik im Backend und allen kartenbezogenen Aktionen im Frontend. Mit einer Vielzahl von Methoden ermöglicht es das Abrufen von Decks unter der GET-Route `'api/v1/decks'`, das Erstellen von Decks unter der POST-Route `'api/v1/decks'`, das Löschen von Decks unter der DELETE-Route `'api/v1/decks/:id'`, das Abrufen von Karten eines Decks unter der GET-Route `'api/v1/decks/:deckId/cards'`, das Löschen einer Karte eines Decks unter der DELETE-Route `'api/v1/decks/:deckId/cards/:cardId'` sowie das Abrufen detaillierter Informationen zu einem Deck unter der GET-Route `'api/v1/decks/:id'`.
Der Cards-Service nimmt eine Schlüsselrolle im Gesamtsystem ein, indem er sämtliche Interaktionen im Zusammenhang mit Karten und Decks orchestriert. Er ermöglicht anderen Frontend-Komponenten nahtlos auf die Backend-Funktionalitäten zuzugreifen und bietet damit eine effiziente Verwaltung und Präsentation der Lerninhalte.

ErrorHandler-Service
Der ErrorHandler-Service übernimmt die zentrale Verarbeitung von Fehlermeldungen im Frontend. Der ErrorHandlerService spielt eine entscheidende Rolle im Frontend, indem er eine konsistente und lokalisierte Fehlerbehandlung bereitstellt. Durch die Integration von Übersetzungsdiensten und die zentrale Steuerung von Fehlermeldungen erleichtert er

anderen Frontend-Komponenten den Umgang mit unterschiedlichen Fehlerstatus des Backends.

History-Service
Der History-Service im Frontend verwaltet die Kommunikation mit dem Backend für alle historischen Lernsessions und zugehörigen Details. Mithilfe von Angular's HttpClient-Modul ermöglicht er das Abrufen von detaillierten Informationen zu einer bestimmten Lernsession unter der GET-Route `api/v1/decks/:deckId/histories/:sessionId` sowie eine Liste aller bisherigen Lernsessions für ein bestimmtes Karteikartendeck unter der Route `api/v1/decks/:deckId/histories`, um diese anschließend in der Benutzeroberfläche darzustellen. Der Service nutzt Observables für eine reaktive Programmierung und HttpResponse für eine umfassende Verarbeitung von HTTP-Antworten.

Toast-Service
Der Toast-Service ist ein Injectable-Service, der die Anzeige von Toast-Nachrichten innerhalb des Frontends vereinheitlicht. Durch die Verwendung von RxJS-Observable und Subjects ermöglicht er das Auslösen verschiedener Toast-Ereignisse für Erfolge, Informationen, Warnungen und Fehler. Der ToastService fügt sich als integraler Bestandteil in das Gesamtsystem ein, indem es eine zentrale Möglichkeit bietet, Toast-Nachrichten in der Benutzeroberfläche zu steuern. Durch die Nutzung von Angulars Dependency Injection und Observable-Struktur können andere Komponenten und Services leicht auf den ToastService zugreifen und ihn verwenden, um Benachrichtigungen über verschiedene Ereignisse anzuzeigen.

Tutorial-Service
Der Tutorial-Service übernimmt die Interaktion mit dem Backend für Tutorial-bezogene Funktionalitäten. Mithilfe des Angular HttpClient-Moduls bietet er Methoden an, um zu überprüfen, ob der Benutzer ein bestimmtes Tutorial bereits gesehen hat (`'/api/v1/tutorials/:page'`), um anschließend festzuhalten, dass der Benutzer ein Tutorial gesehen hat (`'/api/v1/tutorials/:page'`). Die Bereitstellung des Tutorial-Service ermöglicht eine personalisierte Anpassung der Tutorial-Darstellung durch die Abfrage und das Aktualisieren des Tutorial-Status eines Benutzers.

User-Service
Der User-Service kümmert sich um alle benutzerbezogenen Interaktionen zwischen dem Frontend und dem Backend. Er ermöglicht Funktionalitäten wie das Senden von Passwortzurücksetzungsanfragen unter der POST-Route `'api/v1/password/reset'`, das Aktualisieren von Benutzerinformationen unter der PUT-Route `'api/v1/account'`, das Abrufen von Benutzerdaten unter der GET-Route `'/api/v1/account'`, das Registrieren neuer Benutzer unter der POST-Route `'/api/v1/register'` und die Handhabung von Passwortänderungen unter der PUT-Route `'api/v1/password'`. Zusätzlich kümmert es sich um die Verwaltung von Benutzerprofilen und die Benachrichtigungseinstellungen. Die Verwendung von Diensten wie HttpClient, Auth-Service und ToastService macht den Service zu einem integralen Bestandteil für die Benutzerverwaltung im Frontend. Er lässt sich nahtlos in andere Frontend-Komponenten

integrieren und ermöglicht mühelos die Verwaltung von Benutzerinformationen und trägt damit zu einer konsistenten Benutzererfahrung bei.

Websocket-Service
Der Websocket-Service ermöglicht die Integration von WebSocket-Kommunikation in die Anwendung, um Echtzeitbenachrichtigungen über Errungenschaften zu ermöglichen. Er verwendet den RxStompService für die Kommunikation über STOMP-Protokoll, abonniert ein spezifisches WebSocket-Topic für Errungenschaftsbenachrichtigungen unter der Route `'/user/topic/achievement-notification'` und verarbeitet eingehende Nachrichten. Bei Empfang einer Errungenschaftsnachricht ruft der Service das AchievementService auf, um Details zu der Errungenschaft selbst abzurufen unter der Route `'api/v1/achievements/:achievementId'`, und zeigt sie dann in einem modalen Popup an. Durch die Verbindung mit dem RxStompService und dem Aufrufen des AchievementService ermöglicht der Service die dynamische Anzeige von Errungenschaftsbenachrichtigungen in Echtzeit innerhalb der Gesamtanwendung.

Routing

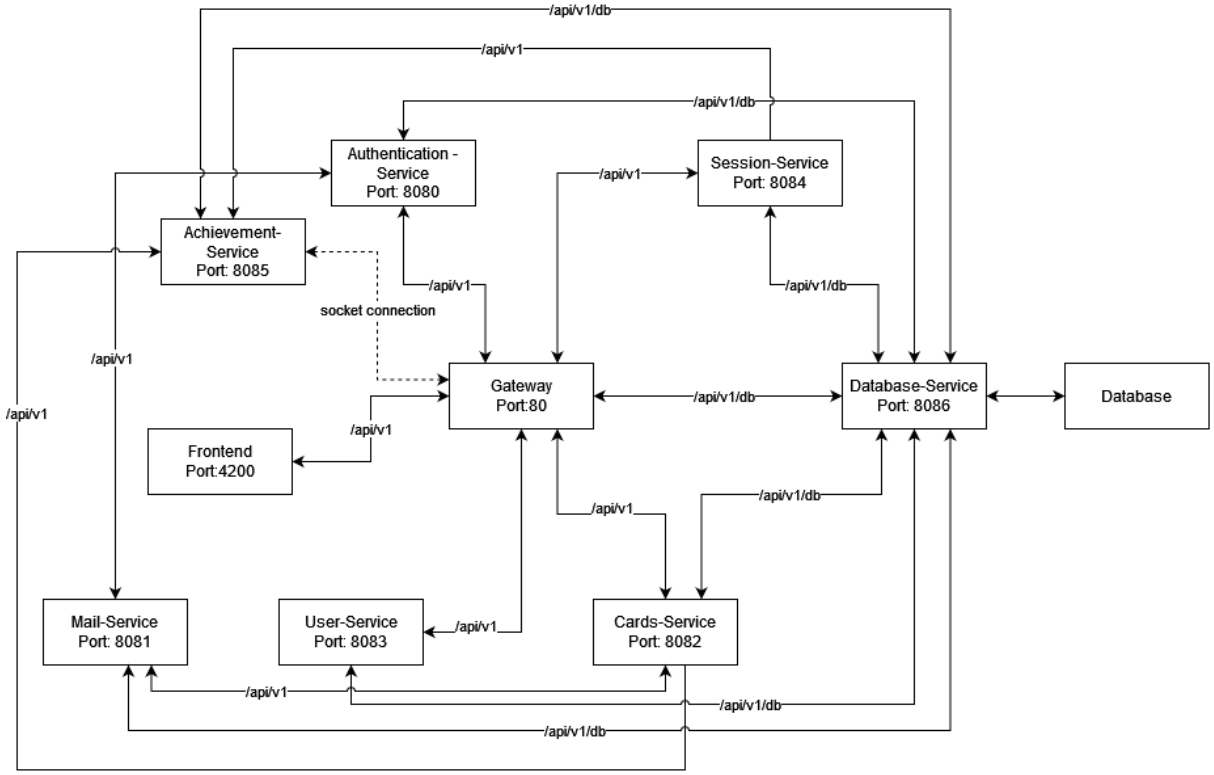
Die app.routes.ts definiert die Routing-Konfiguration für unsere Webanwendung. Hier werden die verschiedenen Pfade der Anwendung mit den korrespondierenden Angular-Komponenten verknüpft. Die Struktur der Routing-Konfiguration spiegelt die Hierarchie und den logischen Ablauf der Anwendung wider.

- `'/login'`:
  Routet zur LoginComponent, die die Anmeldung für registrierte Benutzer ermöglicht.
- `'/register'`:
- Leitet zur Register-Komponente, wo neue Benutzer ihre Registrierung durchführen können.
- `'/register-successful'`:
  Navigiert zur Register-Successful-Komponente, um den Benutzer über eine erfolgreiche Registrierung zu informieren.
- `'/deck/:deck-id'`:
  Routet zur Deck-View-Komponente, die die Detailansicht eines bestimmten Decks ermöglicht.
- `'/deck/:deck-id/edit'`:
  Navigiert zur Edit-Deck-View-Komponente, um die Bearbeitung eines bestimmten Decks zu ermöglichen.
- `'/deck/:deck-id/histories'`:
  Leitet zur History-View-Komponente, die die Historie eines Decks anzeigt.
- `'/deck/:deck-id/card/:card-id/edit'`:
  Routet zur Edit-Card-View-Komponente, die die Bearbeitung einer bestimmten Karte innerhalb eines Decks ermöglicht.
- `'/forgot-password'`:
  Navigiert zur Forgot-Password-Komponente, wo Benutzer ihr Passwort wiederherstellen können.
- `'/reset-password'`:

Leitet zur Reset-Password-Komponente, um das Passwort zurückzusetzen.

- `'/'`:
  Routet zur Deck-List-View-Komponente, die eine Übersicht aller Decks anzeigt. Dies ist die Standardroute, wenn keine spezifische Route angegeben ist.
- `'/deck/:deck-id/learn'`:
  Navigiert zur Learn-Card-View-Komponente, um das Lernen von Karteikarten in einem bestimmten Deck zu ermöglichen. Die `canDeactivate`-Guard-Funktionalität stellt sicher, dass der Benutzer die Lernsession nicht unabsichtlich verlässt.
- `'/deck/:deck-id/peek'`:
  Routet zur Peek-Card-View-Komponente, wo der Benutzer einen schnellen Einblick in alle Karten eines Decks erhalten kann. Die `canDeactivate`-Guard-Funktionalität verhindert, dass der Benutzer die Vorschau unbeabsichtigt verlässt.
- `'/suc'`:
  Leitet zur Register-Successful-Komponente, um zusätzlich eine verkürzte Route für den Erfolgsfall der Registrierung anzubieten.
- `'/profile'`:
  Routet zur User-Profile-Komponente, wo der Benutzer sein Profil anzeigen und bearbeiten kann.
- `'/swagger'`:
  Navigiert zur Swagger-View-Komponente, die eine Schnittstelle für die API-Dokumentation bereitstellt.
- “”:
  Falls eine unbekannte Route aufgerufen wird, wird der Benutzer zur Deck-List-View-Komponente weitergeleitet.

Architektur



Unser Backend besteht wie bereits erwähnt aus mehreren Microservices, hierbei gibt es das Gateway, welches zum einen entsprechende Routen zum Authentication-Service, Session-Service, Cards-Service, User-Service bereithält. Ebenso ist er für das Routing des Sockets zuständig.. Er ist so gesehen der Vermittler, der überprüft, ob der User Zugriff auf diese Route hat etc. Des Weiteren stellt er eine Wildcard fürs Frontend bereit, wodurch unser Frontend über Port 80 erreichbar ist.

Gateway

Ein User muss den folgenden Authentifizierungsprozess durchlaufen, um als authentifiziert zu gelten.
Falls er keinen “card-trainer-user” Cookie besitzen sollte, und keine der freigegebenen Routen aufruft, die keine Authentifizierung benötigt, wird der User ein 401 (Unauthorized) bekommen.
Die freigegebenen Routen sind
"/api/v1/register",
"/api/v1/login",
"/api/v1/email/verify/{token}",
"/api/v1/password/reset",
"/api/v1/decks/share/{token}",
Dementsprechend ist der einzige Weg, wie der User sich autorisieren kann, über die Login Route. Falls er dies nun getan haben sollte, wird er einen Cookie bekommen, mit dem Namen “card-trainer-user”. Dieser enthält Informationen über den User und einen JWT

Token, mit dem der User sich eindeutig identifiziert.
Anschließend besitzt der User Zugriff auf alle anderen Routen, die im Gateway in der GatewayConfig.java definiert sind.

Im Gateway gibt es zwei Filterstufen. Zuerst wird über den AuthTokenFilter der JWT Token, der bei jeder Anfrage ans Backend mitgesendet wird, authentifiziert. Falls dieser Invalide sein sollte, z.B. weil er manipuliert wurde oder abgelaufen ist, dann wird die Abfrage mit einem Statuscode 412 “Precondition failed” abgebrochen.
Falls der Filter nicht in einem 401, weil der Token nicht existiert oder 412 endet, wird der JWTTokenResolveFilter ausgeführt. Dieser löst nun den Token auf und extrahiert die UserId, welche im JWT Token kodiert ist und speichert diese im Header der Anfrage als “userId” ab. Innerhalb des Backends wird die UserId immer angegeben, um zu klarifizieren, dass eine Route als non public - mit Autorisationen - gilt.

Authentication-Service

Der Authentication-Service kümmert sich um jegliche Authentifizierungen von Usern. Er stellt eine Route für die Registrierung bereit, womit der User einen neuen Account anlegen kann. Falls der Account erfolgreich in der Datenbank erstellt werden konnte, dann wird dem User auf die von ihm angegebene Mail eine Verifikationsmail geschickt, mit dem er seinen Account verifizieren kann.
Der Authentication-Service ist der einzige Service, der Passwort-Resets, Registrierungen etc. verwalten darf.

Ab wann ist ein Passwort innerhalb unseres Systems gültig?

- Mindestens ein Großbuchstabe
- Mindestens ein Kleinbuchstabe
- Mindestens ein Sonderzeichen von **~ ` ! @ # $ % ^ & * ( ) _ - + = { [}] | : ; < , > . ? /**
- Mindestens 8 Zeichen lang
- Maximal 72 Zeichen lang
- Darf nicht in der Rainbow-Tabelle sein[2]

Unsere Passwörter werden mit Bcrypt gehasht. Ein Bcrypt-Hash ist wiefolgt aufgebaut.
Beispiel Hash: $2a$10$8rqvZdxBFgVIEqEJrGlcjuwt6lmIugKRYes1eLAac3719D2AmCzvG
Hierbei ist:
$2a die Version des Bcrypt Algorithmus
$10 die Stärke des Algorithmus
$8rqvZdxBFgVIEqEJrGlcj der random Salt
uwt6lmIugKRYes1eLAac3719D2AmCzvG das gehashte Passwort.

Hiermit ergibt sich, dass jeder Passwort-Eintrag in der Datenbank 60 Zeichen lang ist.
Ein erfolgreicher Login kann nur funktionieren, wenn der User seine korrekten Credentials eingibt und der User verifiziert ist.

---

[2] https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-10000.txt - abgerufen am 25.10.2023

Wie funktioniert das Login?

Die Authentifizierung und Autorisierung werden mithilfe der '*Spring Security*' Library realisiert. Ein bereitgestellter *'AuthenticationManager'* ermöglicht die Verwendung der '*authenticate*'-Methode.
Diese Methode erwartet ein *'UsernamePasswordAuthenticationToken'-Objekt*, das aus Benutzername und Passwort besteht. Da der Benutzername in unserem Fall nicht eindeutig ist, verwenden wir stattdessen die E-Mail-Adresse.

Die erfolgreich authentifizierte Benutzerinformation wird im *'SecurityContextHolder'* gespeichert, um die Berechtigungen innerhalb der Anwendung zu unterscheiden. Anschließend implementieren wir die Klasse *'UserDetailsImpl'*, die das Interface *'UserDetails'* implementiert. Hier werden die Benutzerdaten sowie die entsprechenden Berechtigungen festgehalten. Da unsere Applikation sich auf die Benutzerrolle 'User' beschränkt, setzen wir einen entsprechenden Default-Wert für die Berechtigung.

Danach wird aus den Benutzerdetails ein JWT (JSON Web Token) generiert und dem Benutzer in einem Cookie bereitgestellt. Dies ermöglicht es dem Benutzer, sich nicht bei jedem Besuch der Seite erneut anmelden zu müssen. Gleichzeitig fungiert der Token als Datenprovider für Benutzerinformationen wie die User-ID.

Der JWT Token wird vom Frontend bei jeder Request mitgesendet. Dieser wird dann wie bereits erwähnt im Gateway aufgelöst und ab dann wird nur noch die User-ID an die Services mitgesendet.

Wie funktioniert das Erstellen des JWT Token?

```
public ResponseCookie generateJwtCookie(UserDetailsImpl userPrincipal) {
    String jwt = generateTokenFromUserId(userPrincipal.id());
    long maxAgeSecondsCookie = (long) (24 * 60) * 60 * 30;
    return ResponseCookie.from(jwtCookieName, jwt).path("/").maxAge(maxAgeSecondsCookie).httpOnly(false).build();
}
```

```
public String generateTokenFromUserId(Long id) {
    return Jwts.builder()
            .setSubject(String.valueOf(id))
            .setIssuedAt(new Date())
            .setExpiration(Date.from(Instant.now().plusMillis(jwtExpirationMs)))
            .signWith(getKey(), SignatureAlgorithm.HS256)
            .compact();
}
```

Für die Generierung von JWT-Tokens nehmen wir uns die Library io.jsonwebtoken zur Hilfe. Diese stellt uns eine einfache Variante bereit, Tokens zu generieren, welche die geforderten Sicherheitsstandards erfüllt.

Wie funktioniert die E-Mail Verifikation?

Wenn der User die Route `/api/v1/email/verify/{token}` aufruft, dann steht in der URL der Verifikationstoken für den User. Grundsätzlich gilt, dass Verifikationsmails und Password-Reset Tokens nur 24 Stunden gültig sind. Danach gelten diese als abgelaufen.

Hierbei wird zuerst validiert, ob der Token überhaupt noch gültig ist und ob der Tokentyp für diese Operation auch korrekt ist. Hierfür existiert eine interne Tabelle namens "token_type", welche alle verfügbaren Tokentypes enthält. Diese sind (VERIFICATION, PASSWORD_RESET und SHARE_DECK). Falls dies alles gecheckt sein sollte und der User schon verifiziert sein sollte, wird er in Form eines 409 - Conflicts darüber informiert, dass sein Account bereits verifiziert wurde. Andernfalls wird der Account verifiziert und kann nun zum Einloggen in das System verwendet werden. Nach Erfolg wird der User auf eine Frontendkomponente “/verify-successful” geroutet.

Wichtig: Der Token ist eine UUID, welche 36 Zeichen lang ist.

Wie funktioniert der unautorisierte Passwort-Reset - Forgot Password?

Der User kann über die Route `/api/v1/password/reset` eine Passwort-Reset-Mail anfordern. Der Mail-versende Prozess funktioniert identisch mit dem in der Verifikation. Auch hier gilt, das Frontend bekommt immer nur den Statuscode 202, Accepted, um zu verschleiern, ob der Account existiert oder nicht. Mails werden jedoch nur zu existierenden Accounts verschickt.

Anschließend kann der User über das Frontend einen Backendcall via `/api/v1/password/reset` - PUT stellen. Hierbei wird zusätzlich noch das Passwort validiert und entsprechend verschlüsselt (vgl. Ab wann ist ein Passwort innerhalb unseres Systems gültig?). Auch hier wird, wie in der Verifikation entsprechend, der Token wieder validiert.
Wichtig zu erwähnen ist, der Token wird anschließend aus der Datenbank gelöscht, um sicherzustellen, dass dieser Token nicht aus Versehen mehrfach verwendet werden kann.

Wichtig: Der Token ist eine UUID, die 36 Zeichen lang ist.

Wie funktioniert der autorisierte Passwort-Reset - Der User ist eingeloggt?

Die Route `/api/v1/password` ist für einen autorisierten Passwort-Reset. D.h., dass der User bereits in unser System eingeloggt ist und einen gültigen JWT-Token besitzt, über diesen er klar identifizierbar ist.

Hier gilt das gleiche Schema wie beim unautorisierten Passwort Reset. Auch hier wird sich wieder an die Kriterien, die in Ab wann ist ein Passwort innerhalb unseres Systems gültig? Definiert sind gehalten. Bei Erfolg wird das entsprechende Passwort an den DB-Service gesendet und überschrieben.

Mail-Service

Der Mail-Service ist unsere einzige Stelle, an der Mails an den User gesendet werden. Er stellt lediglich eine einzige Route bereit, die unter `/api/v1/mail/{mailType}` aufgerufen werden kann.
Er ist zuständig für das Versenden der Verifikation-Mail, der Passwort-Reset-Mail, die tägliche Lernbenachrichtigung, die täglich um 16:00 Uhr verschickt wird, und für das Versenden der Share-Deckmail.

Daily-Lernmails

Unter Services befindet sich ein Service mit dem Namen “DailyLearnReminderService” dieser hat die Aufgabe, das Scheduling anhand eines Cronjobs zu übernehmen.

Der Ausdruck *cron = "0 0 16 * * ?"* in der Spring *@Scheduled* Annotation plant eine Aufgabe zur Ausführung. In diesem Fall bedeutet es, dass die Aufgabe jeden Tag um 16 Uhr nachmittags startet. Die sechs Felder in dem Cron-Ausdruck repräsentieren Sekunden, Minuten, Stunden, Tag im Monat, Monat und Tag in der Woche, und können angepasst werden, um präzise Zeitpläne zu definieren. Hierbei wird beispielhaft die E-Mail täglich um 16 Uhr versendet.

Wichtig: Die @Scheduled Annotation funktioniert nur, wenn die @SpringBootApplication Komponente zusätzlich mit @EnableScheduling annotiert wurde.

```
@Scheduled(cron = "0 0 16 * * ?")
public void dailyLearnEmail() {
    Optional<List<UserDailyReminderDTO>> userDailyReminderDTOList = dbQueryService.getAllEmailsForDailyLearn();
    userDailyReminderDTOList.ifPresent(
            list -> list.forEach(
                    entry -> mailService.sendDailyLearnReminderMail(entry.username(), entry.language(), entry.email()
                    )
            )
    );
}
```

Hierbei fetcht sich der Mail Service die Liste an Users, die eine tägliche Lernbenachrichtigung erhalten wollten.
Hierbei wird der Username, die Sprache, die der User in seinem Profil ausgewählt hat und die korrespondierende E-Mail per User gequeried.
Die Sprache ist wichtig, um dem User die Mail in seiner individuell ausgewählten Sprache zu präsentieren.

```
public void sendDailyLearnReminderMail(String username, String language, String email) {
    String content = mailContentBuilder.getContent(MailType.DAILY_REMINDER, language, username);
    sendHtmlMail(email, subject: "DailyLearnReminder", content);
}
```

Hierbei wird der Content vom MailContentBuilder geholt, in diesem werden Mail-Placeholder, wie der Username, oder irgendwelche URLs entsprechend ausgetauscht.
Anschließend wird die Mail als HTML Mail versendet.

Alle unsere Mails werden als UTF-8 verwendet, damit sie mit den meisten Mail-Clients kompatibel ist.
Wichtig: Wir können nicht herausfinden, ob der User die Mail auch empfangen hat, da der Java-MailSender über SMTP arbeitet, was kein Feedback über das Absenden einer Mail provided.

Wichtig: Alle unsere Mails werden von unserer Google-Mail "softwaretechnikprojekt@gmail.com" versendet. Hierzu verwenden wir entsprechend die GMAIL SMTP Server zum Versenden der Mails, dieser läuft auf Port 587.
Dies bedeutet im Umkehrschluss, falls die Mailserver von Google nicht erreichbar sein sollten, können sich bei uns keine neuen User mehr verifizieren oder ihr Passwort reseten oder Deck teilen!

Verifikation-Mails & Passwort-Resets

Beide Typen wurden bereits weiter oben beschrieben und funktionieren nach demselben Prinzip wie das Versenden der Lernmails.
Hier ist der einzige Zusatz von Token, der entsprechend in der Datenbank abgespeichert wird.

Share-Deck-Mail

Das Mail versenden funktioniert nach demselben Prinzip, jedoch fungiert hier der Token ein wenig anders.
Ein Token besteht immer aus [DeckId]-UUID.
Dadurch stellen wir sicher, dass wir wissen, welches Deck geteilt werden soll.

User-Service

Der User Service stellt Routen für das Querien und Updaten von User-Daten bereit (User-Table).
Die ausgewählte Sprache muss als Languagecode vom Frontend übergeben werden, um sicherzustellen, dass dies nicht zu späteren Problemen führt. Ein Languagecode für Deutsch wäre beispielhaft "de" oder für Englisch "en".

Hier ist noch wichtig zu erwähnen, dass bei einem Update die AchievementIds und Login-Streak nicht mitgeschickt werden dürfen! Hier unterscheiden sich also das GET und das PUT Objekt.

Der User-Service ist außerdem verantwortlich für das Handeln der Tutorial-Pages.
Falls der User im Frontend eine Tutorialpage gesehen haben sollte, dann kann dies das Frontend über '/api/v1/tutorials/{tutorialPage}' abspeichern. Über die dazugehörige GET Route kann das Frontend dann überprüfen, ob der entsprechende Eintrag schon existiert.

Cards-Service

Der Cards-Service stellt viele unserer Hauptaktionen bereit, er ist verantwortlich für jegliche Deckaktivitäten, von Deckerstellen bis hin zum Erstellen von Karten, Anzeigen der Lernsession-Resultate als auch das Anlegen von Bildern.

Alle Routen im Cards-Service außer `'/api/v1/decks/share/{token}'` benötigen eine entsprechende Authentifizierung!

Das Abspeichern eines Decks funktioniert relativ trivial, und zwar mit der UserId, und zwar mit einem Decknamen, der zwischen 1 - 128 Zeichen lang sein darf.
Selbiges gilt entsprechend für das Updaten eines Decks.
Über die Route `'api/v1/decks'` kann der User sich auch eine Liste von Decks holen. Diese beinhaltet jedoch nur reduzierte Informationen über das Deck, die nötig sind, um zu sortieren und auf die MainDeckView zu gelangen.

Wichtig: Falls dem User das Deck nicht gehören sollte, bekommt er ein 404, Not Found!

Wie funktioniert das Abspeichern von Karten bei uns im System?

Für das Abspeichern einer Karte wird eine UserId, DeckId und ein String namens cardData benötigt. Der erhaltene String wird anschließend zu einer JsonNode geparst, welche dann im Database-Service zu einer TextAnswerCard oder eben MultipleChoiceCard aufgelöst wird.
Anschließend wird bei einer Basic-Card/TextAnswerCard erst das CardDTO abgespeichert und anschließend die Antwort in der Tabelle TextAnswerCard.

Wichtig: Jede Frage und Antwort kann ein Bild besitzen, wie dies im speziellen funktioniert, wird weiter unten definiert. In den DTOs wird das Bild als ImageId angegeben. Welches zuvor vom User abgespeichert worden sein muss.

Das Abspeichern von MultipleChoice-Karten funktioniert vom Prinzip her gleich, die einzige Variation ist, dass wir anstatt nur einer Antwort eine Liste von Antworten (Choice-Answers) haben.

Wichtig: Bei der Erstellung einer Karte, wird automatisch auch eine Repetition initialisiert, mit der entsprechenden Base gewertet.

Wie funktioniert das Updaten einer Karte?

Für TextAnswerCard werden einfach die entsprechenden Daten überschrieben, jedoch ist MultipleChocieCard ein wenig komplizierter.

Wenn zuvor 3 Antworten existieren, mit den IDs [1, 2, 3] und wir schicken nun das neue Multiple-Choice-Card DTO ans Backend, mit nur zwei Antworten [1, 3]. Dann wird anhand einer Exclusion der IDs die ChoiceAnswer mit der ID 2 gelöscht.

Falls eine neue Antwort dazu kommt, dann muss diese in dem Fall die ID “null” haben, falls dies der Fall ist, wird die ChoiceAnswer entsprechend hinzugefügt.
Das Werte überschreiben funktioniert hier jedoch genauso wie in TextAnswerCard.

Wie funktioniert das Sharen eines Decks?

Nachdem eine entsprechende ShareDeckMail gesendet wurde und die Backend-Route ‘/api/v1`/decks/share/{token}`’ aufgerufen wird, wird das Deck, das angegeben wurde als Prefix im Token extrahiert und entsprechend kopiert. Dafür stellen die Models jeweils eine “Copy” Methode Bereit, die ein Object zu einem anderen User kopiert. Dieses Objekt muss dann anschließend lediglich abgespeichert werden.

Wichtig: Der Share Deck Token wird für den User, der es erhalten soll abgespeichert, dadurch kann der Token nicht für andere Zwecke missbraucht werden.

Wie funktioniert das Exportieren eines Decks?

Wenn die Route für das Exportieren eines Decks aufgerufen wird, dann fragt unser Cards-Service zuerst den Database-Service für die entsprechenden CardInformations an. Die werden dem Service dann als JSON bereitgestellt, darin enthalten sind die Bilder als Byte-Array, welche MultipleChoiceKarten es gibt, welche TextAnswerCards es gibt als auch den Decknamen, der exportiert bzw. später wieder importiert werden soll.
Anschließend geht der CardsService hin und speichert erstmal die ganzen Bilder in einem temporären Verzeichnis ab, das mit einer UUID benannt ist. Dadurch stellen wir sicher, dass nicht aus Versehen falsche Informationen zu falschen .zips gelangen.
Anschließend speichern wir auch noch die JSON in das entsprechende Directory ab. Im Anschluss darauf wird der Ordner gezippt und gelöscht. Das Byte-Array, welches nun die gezippten Daten erhält, wird nun als APPLICATION_OCTET_STREAM zurückgegeben. Standardmäßig heißt die Datei, die man herunterlädt, dann ‘card-trainer.zip’ - Im Frontend zum export-‘Decknamen’.zip abgeändert.

Wie funktioniert das Importieren eines Decks?

Beim Importieren wird eine MultipartFile entgegengenommen, diese muss vom Typ ‘’application/x-zip-compressed’ oder vom Typ ‘application/zip’’ sein.
‘application/x-zip-compressed’ ist dafür da, damit die Import Funktion von Firefox, Chrome, Edge unterstützt wird. Hingegen ist ‘application/zip’ dafür da, damit die entsprechende Import Funktion auch auf Safari lauffähig ist.

Auch hier wird wieder ein temporärer Ordner angelegt, in dem wir temporär unsere Dateien ablegen. Zuerst erstellen wir aus der Multipartfile einen ZipInputStream, eine Klasse, die von Java bereitgestellt wird, und parsen nun die Images in das TempDirectory.
Die JSON, sofern vorhanden wird direkt in das ExportDTO/ImportDTO geparst.
Anschließend werden zum ExportDTO jetzt die Bilder, die aktuell im Directory liegen, eingelesen und als Byte-Array abgespeichert.
Anschließend wird das JSON an den DatabaseService gesendet, welcher jetzt wie bei SaveCard die Informationen entsprechend verarbeitet und gespeichert.
Falls ein Bild invalide sein sollte, ist es null und wird einfach nicht abgespeichert. Dadurch können wir gewährleisten, dass nicht das gesamte Deck invalide wird.

Wie funktioniert das Abspeichern von Images/Bildern?

Unser System unterstützt .png und jpeg, alle anderen Datentypen werden entsprechend abgelehnt. Images werden in unserem System in der Datenbank als “Blobs” abgespeichert. Ein Blob wird von java.sql bereitgestellt. Zum Kreieren eines Blobs machen wir uns die Hilfe eines Entitymanagers zunutze, der die Methode ‘getLobHelper().createBlob(data)’ bereitstellt. Dadurch können wir einfach unsere Bilder in die Datenbank abspeichern und mit geringem Aufwand skalierbar machen.

Wichtig: Bilder, die keine UserId besitzen, sind Bilder vom System, z.B. AchievementImages.

Session-Service

Learn und Peek Sessions werden mit einer POST-Route initialisiert, als Return bekommt man die ID, die nötig ist, um anschließend Operationen in der Lernsession machen zu können.
Über die Route-Rating wird auf der Karte der SM-2-Algorithmus ausgeführt und anschließend das Rating, das gegeben wurde, um Eins erhöht.
Anschließend kann über die Route /next-card sich die nächste Karte geholt werden.
Hier muss zwischen Peek-Session und Lern-Session unterschieden werden.

PeekSession:
Hier wird eine Random Karte returned, die innerhalb dieser Peek-Session noch nicht präsentiert wird.

Lernsession:
Hier wird die Karte präsentiert, die am längsten als “abgelaufen” gilt, innerhalb des Decks.

Wichtig: Der HTTP-Statuscode 204, No Content wird zurückgegeben, wenn alle Karten, die gelernt werden müssen, gelernt worden sind.
Der HTTP-Statuscode 409, Konflikt, wird zurückgegeben, falls das Limit an Karten, die pro Lernsession gelernt werden sollen, erreicht wurde. - Dies gilt nur bei der Lernsession!

Anschließend kann eine Lernsession oder PeekSession über die Route ‘/status’ gecancelt oder beendet werden.

Database-Service

Der Database-Service enthält zu allen wichtigen Tabellen einen entsprechenden Controller, der Routen für andere Microservices bereitstellt.
Im Ordner ‘model’ befinden sich die Modelle der Tabellen für unsere Datenbank

```java
@Entity
@Table(name = "deck")
public class Deck {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "d_id")
    private Long id;

    5 usages
    @Column(name = "name", length = 128, nullable = false)
    private String name;

    3 usages
    @ManyToOne
    @JoinColumn(name = "owner_id", nullable = false)
    private User owner;

    Eike Torben Menzel
    public Deck(Long id, String name, User owner) {
        this.id = id;
        this.name = name;
        this.owner = owner;
    }
    Eike Torben Menzel
    public Deck(String name, User owner) {
        this.name = name;
        this.owner = owner;
    }

    Eike Torben Menzel
    public Deck() {
    }
```

Die Annotation @Entity und @Table werden benötigt um klar zu referenzieren um welche Tabelle es sich handelt. Der Name der innerhalb der Tabelle angegeben ist, muss mit der in init.sql angegebenen Tabelle übereinstimmen.

Alle IDs, von einer Tabelle müssen wie oben definiert angegeben werden. Dies gilt als Primärschlüssel.

Über @ManyToOne kann eine 1:n Tabelle entsprechend reingeladen werden.
Falls sinnvoll die Annotation @Lazy hinzufügen, falls das Objekt nicht immer geladen werden muss.

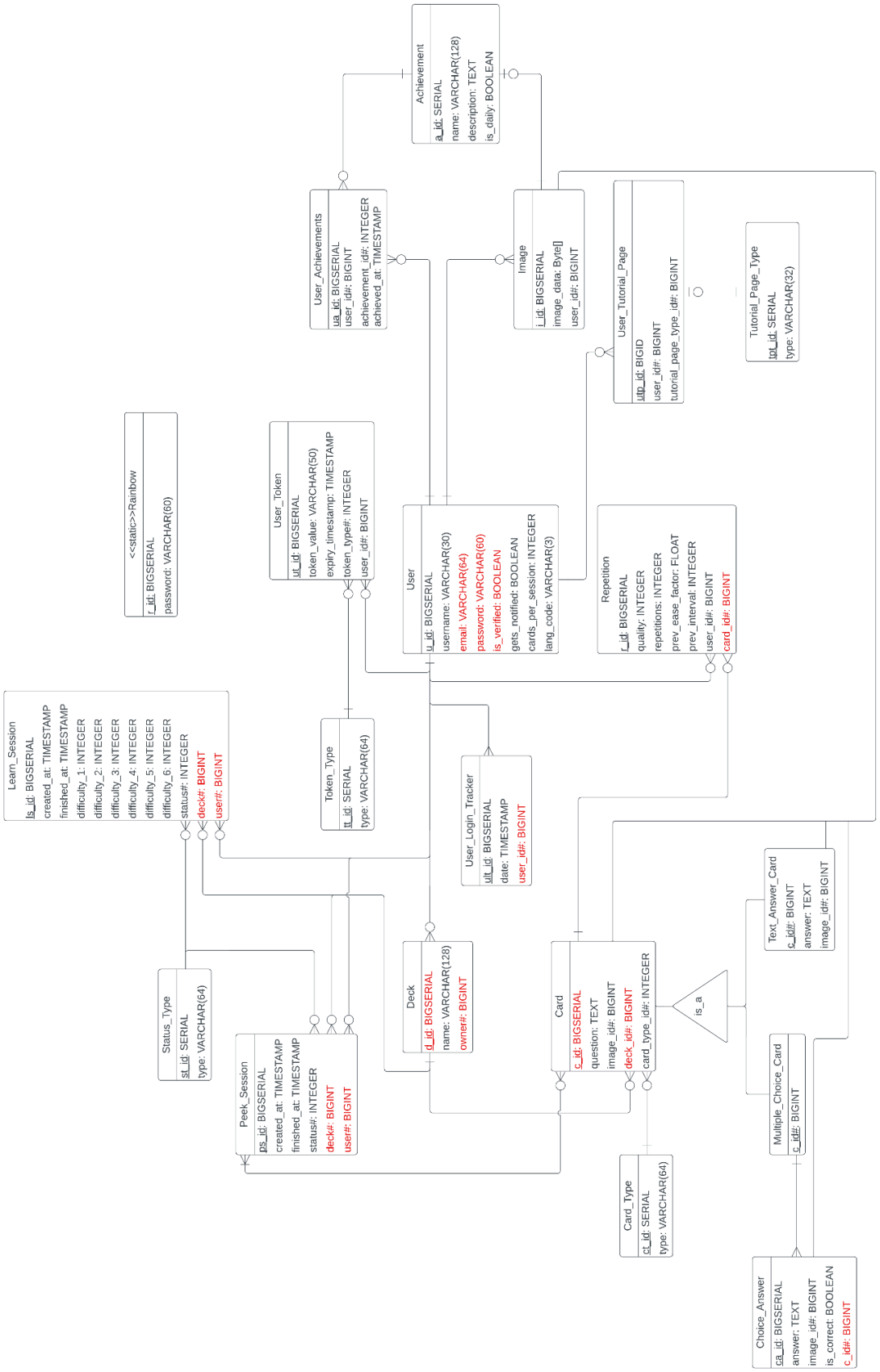
Über @OneToMany kann entsprechend auch andersherum gemappt werden.
Wichtig: CascadeType.ALL sollte nicht vergessen werden, ansonsten kann es zu Problemen bei der Löschung des Objektes führen,

Wichtig: Es wird immer einen leeren Konstruktor für die Realisierung benötigt!

Diese Entitäten werden von JPA verwendet, um vereinfacht queries schreiben zu können.

```java
public interface ImageRepository extends JpaRepository<Image, Long> {
    12 usages  Eike Torben Menzel
    Optional<Image> getImageByIdAndUserId(Long imageId, Long userId);
    1 usage  Eike Torben Menzel
    Optional<Image> getImageById(Long imageId);
}
```

Datenmodell



Tim Martin Süllner, Eike Menzel, Maria Kazim, Gürkan Deniz
Modul: Softwaretechnikprojekt **Professor**: Herr Süß

Datentabellen

User:
Die User Tabelle ist dafür zuständig, die Daten eines Benutzers, nach Registrierungsanfrage, abzuspeichern. Eine E-Mail, welche schon in der Tabelle eingefügt wurde, kann nicht noch einmal eingefügt werden, da sie ‘Unique’ sind. Wiederum kann der Username mehrfach in der Tabelle verfügbar sein. Die Zeile ‘gets_notified’ ist dafür da, um zu wissen, ob der Benutzer eine Erinnerungs-E-Mail zum Lernen erhalten will. Die Zeile ‘is_verified’ bestimmt oder der Benutzer seine angegebene E-Mail verifiziert hat. Wenn nicht, erhält dieser keinen Zugang auf die Hauptseite, bis dieser sich verifiziert hat. Die Zeile ‘cards_per_session’ limitiert den Benutzer beim Lernen, wie viele Karten er lernen will. Zuletzt wird unter der Zeile ‘lang_code’ die ausgewählte Sprache gespeichert, indem der Benutzer die Website angezeigt haben will.

User_Token:
Folgende Tabelle weist dem User einen Token zu. Diese Tokens werden nur für einen bestimmten Zeitraum bereitgestellt und laufen danach ab. Nach Ablauf können diese Tokens nicht mehr benutzt werden. Die Zeile ‘token_value’ speichert den Wert des Tokens ab zur Absicherung, dass Tokens nicht von weiteren Parteien verwendet werden, außer dem User, welcher unter der ID gespeichert wird.

Token_Type:
Die ‘Token_Type’ Tabelle beschreibt, welche Art von Token es sein soll. Ein Token beschreibt, welcher Prozess gerade getätigt wird. Folgende Prozesse befinden sich in dieser Tabelle: ‘SHARE_DECK’, ‘VERIFICATION‘ und ‘PASSWORD_RESET’

- SHARE_DECK: Dieser Token erlaubt es dem Benutzer sein Deck mit einem anderen User zu teilen
- VERIFICATION: Dieser Token beschreibt den Prozess zur Verifikation des Benutzers
- PASSWORD_RESET: Dieser Token beschreibt den Prozess zum zurücksetzen des Passwortes

User_Achievements:
Diese Tabelle ist dafür zuständig, zu speichern, wann und welcher User ein Achievement freigeschaltet hat. Diese werden dann über die IDs des Achievements und des Users, bei Freischaltung, gespeichert.

Achievement:
Ein Achievement ist ein Erfolg, welchen sich der User freischalten kann, durch das Lernen. Unter der Zeile ‘name’ wird der abgekürzte Name des Erfolges abgespeichert. Die darauffolgende Zeile 'description' beschreibt dann detailliert die Bedingung zum Erreichen des Erfolges. Zuletzt kommt dann die Zeile ‘is_daily’ welche bestimmt, ob der Erfolg ein tägliches Achievement ist. Tägliche Achievements bekommt man pro Tag, bspw. das tägliche einloggen.

Image:
Die Tabelle speichert ein Bild als Byte und speichert noch die ID des Users ab, um diese dem User zur Verfügung zu stellen. Auch kann die ID ‘null’ sein, was bedeutet, dass das Bild einen anderen Zweck hat. Dieser Zweck könnte bspw. ein Bild für die Erfolge sein. Sie werden ausschließlich vom System genutzt. Bilder, welche für User eingetragen sind, sind Bilder, welche in ihren Karten gespeichert sind.

User_Login_Tracker:
Diese Tabelle speichert, ab wann der User sich angemeldet hat.

User_Tutorial_Page:
Speichert ab, ob ein User ein bestimmtes Tutorial schon gesehen hat.

Tutorial_Page_Type:
In dieser Tabelle werden die Tutorials und deren Typ gespeichert.

Repetition:
Die ‘Repetition’ Tabelle enthält die Daten, wann eine Karte von einem User das nächste Mal gelernt werden kann. Außerdem speichert sie noch die letzte Wertung ab, falls eine Karte schon einmal gelernt wurde.

Deck:
Die Tabelle ‘Deck’ enthält einen Titel und wird mit einer User-ID abgespeichert, um sicherzustellen, dass dieses Deck dem User gehört.

Rainbow:
Die ‘Rainbow’ Tabelle ist eine abgekapselte Tabelle, welche mit keiner der anderen Tabellen verbunden ist. Sie dient zur Speicherung von Passwörtern, die oft verwendet wurden und sehr unsicher sind.

Learn_Session:
Diese Tabelle hat den Auftrag zu speichern, wann eine Lern-Session von einem User ausgeführt wurde. Dazu speichert sie noch, wann diese beendet wurde und auch welche Bewertungen für die Karten getätigt wurden. Zuletzt wird unter der Zeile ‘status’ gespeichert, welchen aktuellen Stand die Session hat. Sie kann entweder erfolgreich abgeschlossen, abgebrochen oder auch nie fertig gemacht sein. Sollte eine Session nie beendet worden sein und auch nicht abgebrochen, wird diese nach einer bestimmten Zeit als abgebrochen gespeichert. Die Zeilen ‘difficulty’ speichern, wie vorher erwähnt, welche Bewertungen in der Session abgegeben wurden.

Status_Type:
Speichert die Zustände ‘CANCELED’, ‘UNFINISHED’ und ‘FINISHED’. Sie zeigen an, welchen Status die Learn-Session am Ende hat. Folgender Status bedeutet:

- CANCELED: Die Session wurde vom Benutzer abgebrochen
- UNFINISHED: Die Session wurde gestartet, jedoch nie beendet
- FINISHED: Die Session wurde erfolgreich beendet

Peek_Session:
Die Tabelle speichert, wann eine Peek-Session vom User gestartet und beendet wurde. Sie besitzt, wie die Lern-Session eine Zeile ‘status’ um herauszufinden, ob eine Session abgeschlossen, abgebrochen oder auch nie fertig gemacht wurde.

Card:
Die Tabelle ‘Card’ speichert die Frage und das dazugehörige Bild. Eine Karte wird außerdem unter nur einem Deck. Eine Karte selbst hat 2 verschiedene Typen. Ein Typ beschreibt eine einfache Lernkarte mit einer Frage und einer Antwort. Die zweite speichert eine Frage und mehrere Antworten ab.

Card_Type:
Unter der ‘Card_Type’ Tabelle werden die Typen an Karten selbst gespeichert. Hier gibt es den Typ ‘BASIC’ und den Typ ‘MULTIPLE_CHOICE’. Sie beschreiben, ob eine Karte mehrere oder bloß eine Antwort besitzt.

Text_Answer_Card:
Unter dieser Tabelle werden nur Daten gespeichert, wenn der Typ der ‘Card’ Tabelle ein ‘BASIC’ enthält. Wie vorher erwähnt, wird hier bloß eine Antwort und dazu ein Bild gespeichert.

Multiple_Choice_Card:
Diese Tabelle dient als Weiterleitung an die eigentliche ‘Card’ Tabelle. Sie dient außerdem dafür, mehrere Antworten unter einer Karte abzuspeichern.

Choice_Answer:
Unter dieser Tabelle werden nur Daten gespeichert, wenn der Typ der ‘Card’ Tabelle ein ‘MULTIPLE_CHOICE’ enthält. Hier wird eine Antwort und dazu ein Bild gespeichert. Dazu wird hier noch unter ‘is_correct’ gespeichert, ob die Antwort der Karte eine richtige Antwort ist. Eine Karte kann 2 bis n viele Antworten haben.

Connection-zur-Database

```
db.url=jdbc:postgresql://localhost:5433
db.name=swtp
spring.datasource.url=${db.url}/${db.name}
spring.datasource.username=db_controller
spring.datasource.password=postgres
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.database=postgresql

logging.level.com=DEBUG
logging.level.sql=DEBUG
```

Hierüber kann eine Verbindung zur Datenbank aufgewacht.

SM-2-Algorithmus[3]

Unser System richtet sich nach folgenden Werten.
Desto höher das Rating, desto besser hat der User die Response bewertet.
Dafür verwenden wir:
5 - perfect response
4 - correct response after a hesitation
3 - correct response recalled with serious difficulty
2 - incorrect response; where the correct one seemed easy to recall
1 - incorrect response; the correct one remembered
0 - complete blackout.

Jede Karte in unserem System wird am Anfang mit den folgenden Werten initialisiert.

```
repetitionRepository.save(new RepetitionModel( repetition: 0,  quality: -1,  prevEaseFactor: 2.5,  prevInterval: 0, Timestamp.from(Instant.now()), user, card));
```

Der Algorithmus berechnet die Tage, die bis zum nächsten Review gewartet werden muss.
Für genauere Funktionsweise über den Algorithmus, siehe Fußnote 3.

Scheduler-Set-Canceled-after 3 Hours

Sowohl für Lernsession als auch für PeekSession werden Sessions, die nach drei Stunden nicht abgeschlossen sein sollte, da mit sehr hoher Wahrscheinlichkeit der User die Seite geschlossen haben wird. Die scheduled Methode wird alle 3 Stunden ausgeführt. Dadurch stellen wir sicher, dass Jobs, die gecancelt sein sollten, auch immer nach einer bestimmten Zeit gecancelt werden.

[3] https://github.com/thyagoluciano/sm2 - Abgerufen am 22.01.2024

Achievement-Service

Websocket

Der UserHandShakeHandler bildet ein Mapping von UUID zwischen aktiven Verbindungen zu unserm Achievement-Service. Da jeder User sich mehrfach auf der Seite einloggen kann, müssen wir ein entsprechendes Mapping providen, wodurch alle seine aktiven Clients angesprochen werden.
Wenn dann eine Nachricht vom Achievement-Service an das Frontend gestellt werden soll, dann wird an jede Socket ID von diesem User das Achievement gesendet.

Unser Socket basiert auf dem STOMP Protokoll.

Achievements und ihre Requirements

Id	Name	Bedingung
Permanent Achievements		
1	First deck created	1 Deck erstellt
2	handleSingleSession	1x Lernsession abgeschlossen
3	handleFiftySession	50x Lernsession abgeschlossen
4	handleHundredSession	100x Lernsession abgeschlossen
5	handleFiveHundredSession	500x Lernsession abgeschlossen
6	handleHundredCardsLearned	100x Karten gelernt
7	handleFiveHundredCardsLearned	500x Karten gelernt
8	handleTwentyFiveHundredCardsLearned	2500x Karten gelernt
Daily Achievements:		
9	Daily Login	Login after 24h
10	Daily Learnsession	Daily 1x Lernsession completed
11	CardsLearnedDaily: 10	Daily 10 Karten gelernt
12	CardsLearnedDaily: 50	Daily 50 Karten gelernt

Permanente Achievements sind dauerhaft da. Tägliche Achievements hingegen resetten sich jeden Tag.

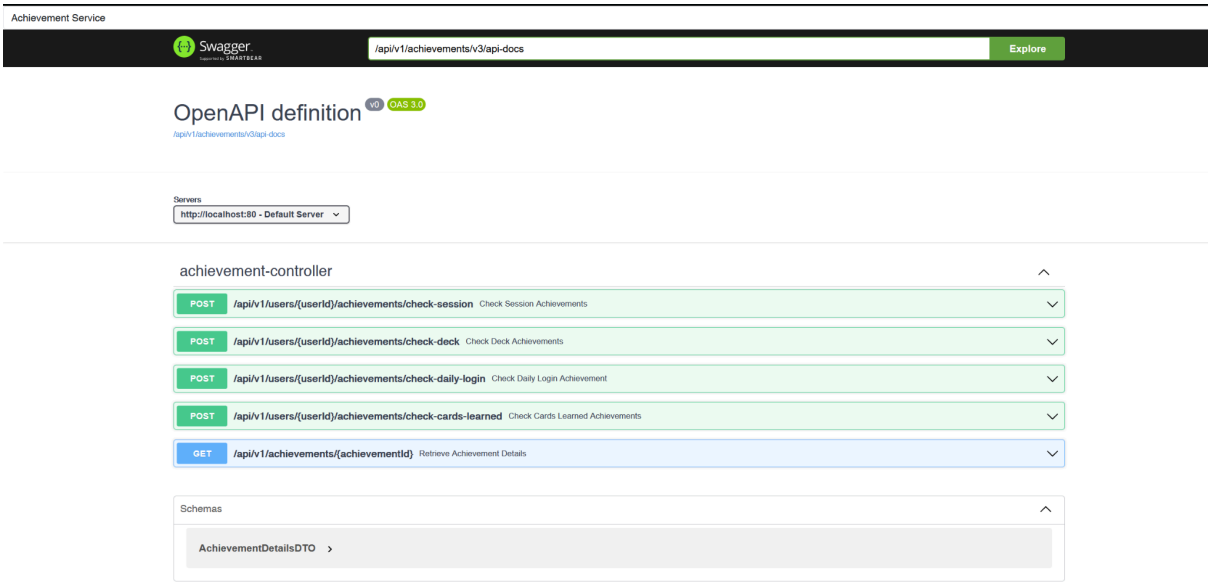
API-Documentation - Swagger

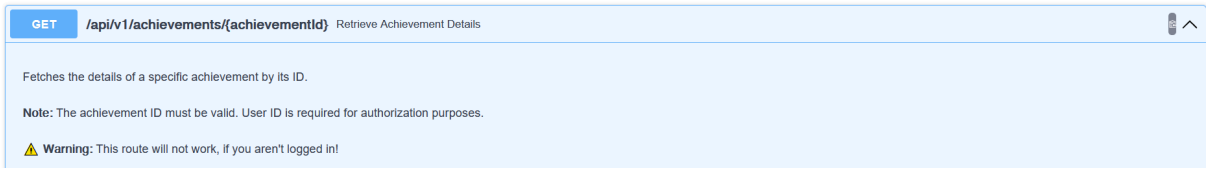
Damit man auf die API-Dokumentation zugreifen kann, muss sowohl das Backend komplett laufen als auch das Frontend gestartet werden.
Anschließend auf http://localhost/swagger navigieren. Dort existiert nun ein Dropdown, mit der Dokumentation für die einzelnen Services.

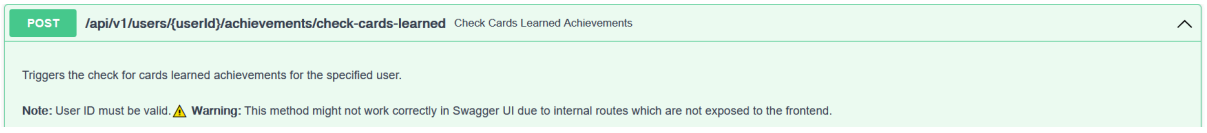
Achtung! Unter jeder Route ist angegeben, was die Requirements für die Route ist. Beispielsweise, dass man eingeloggt sein muss.

Achtung! Da unser Gateway lediglich die Dokus exposed, werden Routen, die vom Gateway nicht freigegeben werden, nicht funktionieren. Dies betrifft u.a. den gesamten Mail-Service, Database-Service und große Teile des Achievement-Service

Beispiel:







Package-Management

Backend

Achievement-Service

spring-boot-starter - latest
spring-boot-starter-web - latest
spring-boot-starter-websocket - latest
lombok - latest

springdoc-openapi-starter-webmvc-ui - 2.2.0

spring-boot-starter-test - latest
junit (5) - latest

Authentication-Service

spring-boot-starter
spring-boot-starter-security
spring-boot-starter-web
jjwt-api, jjwt-impl, jwt-jackson - 0.11.5
jakarta.validation-api
spring-boot-starter-validation - 3.2.1
jackson-databind
jackson-annotations
jackson-core

springdoc-openapi-starter-webmvc-ui - 2.2.0

spring-boot-starter-test - latest
junit (5) - latest
junit-vintage-engine - 5.6.3
spring-security-test - 6.2.1

Cards-Service

spring-boot-starter
spring-boot-starter-web
jakarta.validation-api

springdoc-openapi-starter-webmvc-ui - 2.2.0
spring-boot-starter-test - latest
junit (5) - latest

Database-Service

spring-boot-autoconfigure
spring-boot-starter
spring-boot-starter-web
jakarta.validation-api - 3.0.2

postgresql - 42.6.0
spring-boot-starter-data-jpa - latest

springdoc-openapi-starter-webmvc-ui - 2.2.0
spring-boot-starter-test
assertj-core
h2
junit (5)

Gateway-Service

spring-boot-starter-webflux
spring-cloud-starter-circuitbreaker-resilience4j
spring-cloud-starter-gateway
spring-boot-starter-security
jjwt-api, jjwt-impl, jwt-jackson - 0.11.5

springdoc-openapi-starter-webmvc-ui - 2.2.0

spring-boot-starter-test
spring-cloud-starter-contract-stub-runner

Mail-Service

spring-boot-starter-mail
spring-boot-starter-web
jackson-databind
jackson-annotations
jackson-core

springdoc-openapi-starter-webmvc-ui - 2.2.0

spring-boot-starter-test - latest
junit (5) - latest

Session-Service

spring-boot-starter-web
commons-lang3 - 3.12.0

springdoc-openapi-starter-webmvc-ui - 2.2.0

spring-boot-starter-test - latest

User-Service

spring-boot-starter-web
commons-lang3 - 3.12.0

springdoc-openapi-starter-webmvc-ui - 2.2.0

spring-boot-starter-test - latest

Zusammengefasst ergibt sich:

1. **Database-Service**: Nutzt Spring Boot mit JPA und PostgreSQL zur Datenverwaltung und Jakarta Validation API für die Validierung.
2. **Gateway-Service**: Implementiert ein reaktives Web-Gateway mit WebFlux, Resilience4J für Schaltkreisabsicherung, Spring Security und JWT für Authentifizierung.
3. **Mail-Service**: Bietet E-Mail-Funktionalitäten durch Spring Boot Mail, ergänzt durch Jackson Bibliotheken zur Datenverarbeitung.
4. **Session-Service**: Verwende Spring Boot für Webdienste und Apache Commons Lang für erweiterte Java-Funktionalitäten.
5. **User-Service**: Ähnlich wie der Session-Service nutzt dieser Service Spring Boot für Webdienste und Apache Commons Lang für zusätzliche Java-Features.
6. **Achievement-Service**: Dieser Service fokussiert sich auf die Verwaltung von Achievements und nutzt Spring Boot mit Web, Websocket und Lombok für erweiterte Funktionalität und Codevereinfachung.
7. **Authentication-Service**: Spezialisiert auf Benutzerauthentifizierung, verwendet dieser Service Spring Boot mit Security und Web-Integrationen, JJWT für Token-Management und Jakarta Validation API für Validierungszwecke.
8. **Cards-Service**: Dieser Service bietet Funktionen zur Kartenverwaltung, nutzt Spring Boot und Jakarta Validation API für robuste Webdienste und Datenvalidierung.

Jeder dieser Services verwendet außerdem Spring Boot Starter Test für das Testen und Springdoc OpenAPI (Version 2.2.0) für die API-Dokumentation.

Frontend

Das Paketmanagement für unser Angular-Frontend ist sorgfältig konfiguriert und nutzt eine Vielzahl von Bibliotheken und Technologien, um die Entwicklung, das Testen und die Gestaltung der Benutzeroberfläche zu unterstützen.

Dependencies

Angular

```
"@angular/animations": "^17.0.0",
"@angular/common": "^17.0.0",
"@angular/compiler": "^17.0.0",
"@angular/core": "^17.0.0",
"@angular/forms": "^17.0.0",
"@angular/material": "^17.0.1",
"@angular/platform-browser": "^17.0.0",
"@angular/platform-browser-dynamic": "^17.0.0",
```

@angular/animations, @angular/common, @angular/compiler, @angular/core, @angular/forms, @angular/platform-browser, @angular/platform-browser-dynamic:

Diese Pakete bilden das grundlegende Angular Framework und sind entscheidend für die Struktur, Logik und Benutzeroberfläche der Anwendung. Die Auswahl der Version 17 stellt sicher, dass unser Projekt von den aktuellen Entwicklungen im Angular Ökosystem profitiert. Dies beinhaltet, neue Funktionen, Fehlerkorrekturen und Leistungsverbesserungen des Frameworks zu nutzen, um eine zeitgemäße Entwicklungsumgebung sicherzustellen.

@angular/material:
Das Material Design bietet vordefinierte UI-Komponenten für ein ansprechendes und konsistentes Design. Die Wahl der Version 17.0.1 soll die Integration der neuesten Verbesserungen und Erweiterungen von Angular Material sicherstellen.

@angular/router:
Das Router-Modul ermöglicht die Navigation zwischen verschiedenen Ansichten der Anwendung. Durch die Verwendung der Version 17.0.0 wird sichergestellt, dass die Routing-Funktionalität von den neuesten Entwicklungen und möglichen Optimierungen im Angular-Framework profitiert.

FontAwesome

```
"@fortawesome/angular-fontawesome": "^0.14.0",
"@fortawesome/free-solid-svg-icons": "^6.4.2",
```

Die Integration von "@fortawesome/angular-fontawesome" in der Version 0.14.0 ermöglicht die nahtlose Nutzung von Font Awesome-Symbolen in der Angular-Anwendung. Durch die

Verwendung von "@fortawesome/free-solid-svg-icons" in der Version 6.4.2 werden eine breite Auswahl an vektorbasierten, soliden Icons bereitgestellt, die zur visuellen Gestaltung und Benutzerinteraktion innerhalb der Anwendung genutzt werden können.

ng-bootstrap

```
"@ng-bootstrap/ng-bootstrap": "^16.0.0",
```

Die Integration von "@ng-bootstrap/ng-bootstrap" in der Version ^16.0.0 ermöglicht die Verwendung von Bootstrap-Komponenten in Angular, was eine einfache und effiziente Implementierung von responsiven und benutzerfreundlichen UI-Elementen ermöglicht. Diese Version stellt sicher, dass die Anwendung von den neuesten Funktionen und Optimierungen in der ng-bootstrap-Bibliothek profitiert.

ngx-translate

```
"@ngx-translate/core": "^15.0.0",
"@ngx-translate/http-loader": "^8.0.0",
```

Durch die Integration von "@ngx-translate/core" in Version 15.0.0 wird die Anwendung mit leistungsfähigen Funktionen für die Internationalisierung und Lokalisierung ausgestattet, was eine mehrsprachige Benutzererfahrung ermöglicht. Das zugehörige "@ngx-translate/http-loader" in der Version 8.0.0 erlaubt das Laden von Übersetzungsdateien über HTTP, was eine effiziente Handhabung von sprachspezifischen Inhalten gewährleistet und somit die Anpassungsfähigkeit der Anwendung an unterschiedliche Zielgruppen verbessert.

⚠️ **WICHTIG**: Ein momentan in ngx-translate noch existierender Bug ist, dass ‘translate’ nicht in der *NgOnInit* Funktion verwendet werden kann, da zu dieser Zeit der Translation-Service noch nicht fertig initialisiert wurde. Folglich funktionieren zu dem jetzigen Zeitpunkt Error/Info-Messages nicht, welche zu Zeiten des Seitenaufbaus nicht übersetzt werden. Dieser Bug tritt jedoch nur auf, wenn man gezielt auf eine falsche Seite direkt über die Browser-Suchleiste geht.

STOMP

```
"@stomp/rx-stomp": "^2.0.0",
```

Die Integration von "@stomp/rx-stomp" in Version 2.0.0 ermöglicht die einfache und effiziente Verwendung des STOMP-Protokolls für die Kommunikation zwischen Web Anwendungen und Servern in Echtzeit. Diese Version stellt sicher, dass die Anwendung von den neuesten Funktionen und Verbesserungen in der Bibliothek profitiert, um eine zuverlässige und skalierbare Handhabung von WebSocket-Kommunikation zu gewährleisten.

Swagger UI

```
"swagger-ui": "^5.11.0",
```

```
"swagger-ui-dist": "^5.10.5",
```

Die Einbindung von "swagger-ui" in der Version ^5.11.0 ermöglicht die Integration eines interaktiven Swagger-UIs in die Anwendung, um API-Dokumentationen benutzerfreundlich darzustellen. Mit "swagger-ui-dist" in der Version ^5.10.5 wird die Distribution des Swagger-UIs in der Anwendung vereinfacht und sicherstellt, dass die neuesten Features und Verbesserungen in der Swagger-UI-Bibliothek genutzt werden können.

rxjs

```
"rxjs": "~7.8.0",
```

Durch die Integration von "rxjs" in der Version ~7.8.0 wird die Anwendung mit Reactive Extensions for JavaScript (RxJS) ausgestattet, was eine reaktive Programmierung in Angular ermöglicht. Die gewählte Version stellt sicher, dass die Anwendung von den neuesten Features und Stabilitätsverbesserungen in RxJS profitiert, um komplexe asynchrone Operationen und Datenflüsse effektiv zu handhaben.

Bootstrap

```
"bootstrap": "^5.3.2",
```

Die Integration von "bootstrap" in der Version 5.3.2 bietet ein leistungsstarkes Frontend-Framework für die Gestaltung und das Layout der Angular-Anwendung. Durch die Verwendung dieser Version können Entwickler von den neuesten Funktionen und Verbesserungen in Bootstrap profitieren, um eine moderne und ansprechende Benutzeroberfläche zu gestalten.

DevDependecies

Cypress

```
"cypress": "^13.6.3",
```

Die Integration von "cypress" in der Version 13.6.3 bietet ein leistungsstarkes End-to-End-Testframework für die Angular-Anwendung. Diese Version stellt sicher, dass die Entwickler von den neuesten Funktionen und Verbesserungen in Cypress profitieren, um umfassende automatisierte Tests durchzuführen, die die Anwendungsqualität und Stabilität gewährleisten.

Jasmine

```
"jasmine-core": "~5.1.0",
```

Die Einbindung von "jasmine-core" in der Version 5.1.0 stellt das Kern-Testframework für die Angular-Anwendung dar. Diese spezifische Version gewährleistet, dass die Tests in einer

stabilen Umgebung ausgeführt werden, und ermöglicht Entwicklern, von den neuesten Features und Verbesserungen in Jasmine zu profitieren, um aussagekräftige und zuverlässige Tests zu erstellen.

Karma

```
"karma": "~6.4.0",
"karma-chrome-launcher": "~3.2.0",
"karma-coverage": "~2.2.0",
"karma-jasmine": "~5.1.0",
"karma-jasmine-html-reporter": "~2.1.0",
```

Die Integration von "karma" in der Version ~6.4.0 stellt einen leistungsfähigen Test-Runner für die Angular-Anwendung bereit. Die begleitenden Plugins, darunter "karma-chrome-launcher" in der Version ~3.2.0, "karma-coverage" in der Version ~2.2.0 und "karma-jasmine" in der Version ~5.1.0, erweitern die Funktionalität von Karma. Diese Auswahl gewährleistet, dass Tests effizient in verschiedenen Browsern ausgeführt werden können, die Testabdeckung gemessen wird und aussagekräftige Testberichte generiert werden, um eine umfassende Überprüfung der Anwendungsqualität zu ermöglichen.

TypeScript

```
"typescript": "~5.2.2"
```

Die Integration von "typescript" in der Version ~5.2.2 stellt die Programmiersprache dar, die für die Entwicklung der Angular-Anwendung verwendet wird. Die gewählte Version stellt sicher, dass die Entwickler von den neuesten Funktionen und Optimierungen in TypeScript profitieren, um einen effizienten und gut strukturierten Code zu schreiben.

CI/CD Pipeline

Unsere CI/CD Pipeline besteht aus vier Stages.

- **Build**: In der Build Stage, werden die einzelnen Container aus dem Backend und Frontend gebaut, um sicherzustellen, dass die Container lauffähig sind

- **Test**: Hier werden Unit-Tests und E2E Tests ausgeführt, hierbei werden im Backend Unit-Tests ausgeführt und im Frontend UI-Tests, sogenannte E2E Tests

- **Statische Codeanalyse**: Sowohl für das Backend als auch für das Frontend existiert statische Codeanalyse.
  Beide Stages verwenden das standardmäßig definierte Gateway von der THM.
  Hierbei wird auf Codesmells, Bugs, Security Loopholes etc. geprüft.
  Frontend: https://scm.thm.de/sonar/dashboard?id=swtp-project-cards-frontend
  Backend: https://scm.thm.de/sonar/dashboard?id=swtp-project-cards

- **PushToRegistry**: Hier werden die Container in die Registry der THM gepusht, damit diese anschließend z.B. für eine Docker-Compose FIle verwendet werden können.

Teststrategien

Backend

Im Backend setzen wir auf Unit-Tests, um möglichst alle unsere Funktionen abdecken zu können. Wir haben sowohl die Services als auch die dazugehörigen Controller getestet. Daraus können wir mit einer gewissen Sicherheit davon ausgehen, dass die Funktionen innerhalb ihrer spezifizierten Parameter arbeiten.
Für die Servicetests wurden die externen Unterkomponenten entsprechend gemockt, da die Calls auf anderen Services in der aktuellen Umgebung nicht realisierbar waren.
Für die Controller wurde MockMVC verwendet, dies bietet die Möglichkeit eine Route zu callen und darauf basierende Resultate zu erwarten, die erfüllt werden müssen, damit der Test als erfolgreich gilt.

Insgesamt umfasst das Backend zum derzeitigen Standpunkt 519 Unit-Tests. Siehe Screenshot.

Frontend

Um sicherzustellen, dass sämtliche Frontend-Funktionalitäten unserer Webanwendung fehlerfrei und zuverlässig arbeiten, haben wir eine umfassende Teststrategie entwickelt, die auf dem Cypress Testing Framework basiert. Die Tests werden im Ordner "e2e" im übergeordneten "cypress"-Ordner organisiert. Die Teststrategie umfasst folgende Schlüsselbereiche:

1. Benutzeranmeldung und -registrierung
    - Überprüfung der Registrierung neuer Benutzer.
    - Testen der erfolgreichen Anmeldung von Benutzern.
    - Validierung, dass ungültige Anmeldeinformationen abgelehnt werden.
    - Testen, ob im Falle eines Fehlers eine entsprechende Rückmeldung an den Nutzer erfolgt

2. Zurücksetzen des Passworts
    - Testen der Funktionalität zum Zurücksetzen des Passworts.
    - Testen der Sicherheit und Validierung des neuen Passworts bei der Aktualisierung.

3. Verwaltung von Karteikartendecks
    - Überprüfung der korrekten Erstellung neuer Karteikartendecks.
    - Testen des Hinzufügens von Karteikarten zu Decks.
    - Sicherstellung der korrekten Löschfunktionalität für Karteikartendecks.

4. Karteikarten erstellen und bearbeiten
    - Testen des Hinzufügens verschiedener Karteikartenarten (Text, Multiple Choice).
    - Überprüfung der Bearbeitungsfunktionen für Karteikarten.
    - Sicherstellung der korrekten Löschfunktionalität für Karteikarten.

5. Benutzerprofil verwalten
    - Testen der Bearbeitung von Benutzerinformationen (Username, E-Mail, Passwort).
    - Testen des Speichervorgangs
    - Überprüfung des Ausloggen-Prozesses

Folgendes konnte aufgrund von Problemen auf seitens komplexen JSON Parsings nicht getestet werden:
- Learn-Session
- Peek-Session

Die Routen für den Aufruf in das Backend wurden abgefangen und mit eigens gemockten Daten geprüft, ob diese Daten auch wie erhofft angezeigt werden.

Glossar

User: Ein User ist ein Benutzer, welcher sich innerhalb des Systems registrieren und anmelden kann. Innerhalb der Software kann dieser dann Kartendecks sowie Lernkarten erstellen, lernen, verändern und löschen.

Kartendeck/Deck: Ein Kartendeck besitzt immer einen Namen, zum Beispiel ‘Lineare Algebra’ und ist eine Ansammlung von mehreren Lernkarten.

Lernkarte: Die Lernkarte ist ein Bestandteil eines Kartendecks, welches der Darstellung von Lerninhalten dient und entweder als Multiple-Choice-Frage oder als Text-Antwort-Frage gestaltet sein kann.

Lernsession: Eine Lernsession bezeichnet den Zeitraum, in dem ein User aktiv seinen Lernstoff durcharbeitet. Während einer Lernsession wird dem User eine bestimmte Anzahl von Lernkarten innerhalb des Kartendecks vorgelegt, die er bearbeitet und lernt, um seinen Wissensstand in einem bestimmten Bereich zu erweitern.

Peeksession: Eine Peeksession bezeichnet einen Vorgang, indem der User all seine Karten im entsprechenden Kartendeck lernt.

Spaced-Repetition-Algorithmus: Der Spaced-Repetition-Algorithmus ist eine Lernmethode, bei der Lerninhalte in bestimmten Intervallen wiederholt werden, um das langfristige Erinnerungsvermögen zu verbessern. Dabei werden bereits gut beherrschte Inhalte seltener wiederholt, während wenig vertraute Elemente häufiger abgefragt werden.

Achievement: Ein Achievement ist eine freigeschaltete Belohnung oder Auszeichnung, die User erhalten, wenn sie spezifische Lernziele erreichen, um ihre Motivation und aktive Teilnahme am Lernprozess zu fördern.

Basic-Card/TextAnswerCard: Ein Basic Karte ist eine Lernkarte mit genau einer Frage und einer Antwort. Die Frage sowie die Antwort können als Text und Bild dargestellt werden.

Multiple-Choice-Card: Ein Multiple Choice Karte ist eine Lernkarte mit genau einer Frage und mehreren Antwortmöglichkeiten. Die Frage sowie jede Antwort können sowohl einen Text als auch optional ein Bild besitzen. Man kann außerdem bei jeder Antwort hinterlegen, ob diese wahr oder falsch ist.