Report

Introduction

Filme und Serien schaut man heutzutage vor allem auf Netflix oder Disney+. Beide Plattformen nutzen Recommender Systeme, um den Nutzern passende Filme und Serien zu empfehlen. Einen solchen Recommender und die dazugehörige Webapplikation bauen wir in der Challenge "Tinder for Movies", welche wir dieses Semester besuchen. Um in der Webapplikation auf die relevanten Daten zugreifen zu können, bauen wir eine API, welche die Daten aus der Datenbank abruft und an die Webapplikation weitergibt. Eine API ist eine wichtige und sicherheitskritische Komponente, da sie Zugang zu den erlaubten Operationen im Backend anbietet. Deshalb ist es wichtig, dass die API gut dokumentiert und getestet wird.

Use Case und Motivation

Unsere API soll für die Challenge Tinder for Movies gebraucht werden. Sie stellt den Datenverkehr zwischen der geplanten Webapplikation und der Datenbank sicher.

In der Webapplikation soll auf der Hauptseite eine Liste von allen Filmen angezeigt werden, zu Beginn sortiert nach Popularität. Durch ein Suchfeld können spezifische Filme gefunden werden. Mit einem Klick auf einen Film werden weitere Informationen angezeigt. Weiter soll man sich auf der Webapplikation auch registrieren können. Mit dem eigenen Account kann man dann Filme bewerten. Die Hauptseite sortiert die Filme dann nicht mehr nur nach Popularität, sondern anhand unseres Recommenders. Ausserdem kann der Benutzer seine eigenen Ratings anschauen und auch wieder bearbeiten bzw. löschen. Sicherheitsrelevante Themen für Login und die Registrierung können dabei aber vernachlässigt werden.

Architektur

Um die im Use Case beschriebenen Funktionalitäten zu bedienen, haben wir einen Flask Webserver implementiert. Dieser stellt die Schnittstelle zwischen der Webapplikation und der Datenbank dar. Die Datenbank ist ein Mariadb Server, welchen wir auf einem Ubuntu Server gehostet haben.

Für die Tests wird im Docker Container eine "mariadb-server" Instanz mit installiert. Diese dient als Mock-Datenbank, welche die gleiche Struktur wie die echte Datenbank hat. Sie wird vor jedem Test mit denselben Testdaten von ./sql/setup.sql befüllt.

Die API ist in zwei Teile aufgeteilt:

- User API
- Movie API

User API

Die User API dient zum Verwalten der User. Es können alle User abgefragt werden, ein User mit einer bestimmten ID abgefragt werden, ein User erstellt, gelöscht oder geupdatet werden. Es können auch die Ratings oder alle Filme eines Users abgefragt werden, welche er bewertet hat. Da die Webapplikation lediglich Ratings aus der bestehenden Datenbank verwenden soll und keine neuen Ratings dazu kommen, fallen Kreierung, Bearbeitung und Löschung von Ratings weg.

Movie API

Die Movie API dient zum Verwalten der Movies. Es können alle Movies abgefragt werden, ein Movie mit einer bestimmten ID abgefragt werden, ein Movie erstellt, gelöscht oder geupdatet werden.

Datenbank

Die Datenbank besteht aus 3 Tabellen:

- TMDB_movie_infos
- user
- user_rating

TMDB_movie_infos

Die TMDB_movie_infos Tabelle enthält alle Informationen zu den Filmen. Diese werden von der TMDB API abgefragt und in die Datenbank geschrieben.

user

Die user Tabelle enthält alle User.

user_rating

Die user_rating Tabelle enthält alle Ratings der User. Sie ist die Verbindungstabelle zwischen user und TMDB_movie_infos.

RESTful vs GraphQL

Unsere API ist nicht sehr komplex. Da wir unsere Datenbank nur in einer Webapplikation brauchen, bei welcher die Funktionen und verwendeten Daten von Beginn an klar und im weiteren Verlauf unseres Projekt konsistent sind, können alle Calls genau vordefiniert und auf unsere Problemstellungen zugeschnitten sein. So werden auch immer alle gelieferten Daten gebraucht, wodurch es zu keinem Overfetch kommt. Dadurch könnte GraphQL bei unserem Use Case seine grössten Stärken gar nicht zeigen. Ein Da die Learning Curve und die Komplexität bei GraphQL grösser ist, haben wir uns für RESTful entschieden.

Übersicht

	GraphQL	**RESTful**
Organzied in terms of	Schema & Type System	Endpoints
Community	growing	large
Learning curve	difficult	moderate
Complexity	Higher	Medium
Use cases	multiple microservices, mobile apps	simple apps, resource driven apps