Playlist-Builder

Authors: Holzer Tobias, Werlen Nevio

Teacher: Vianin Jeremy

Class: Inf4 App

Inhalt

[1. Backend: 4](#_Toc210122238)

[2. Technologien & Abhängigkeiten 4](#_Toc210122239)

[3. Projektstruktur 4](#_Toc210122240)

[4. Funktionale Beschreibung 5](#_Toc210122241)

[4.1 Konfiguration 5](#_Toc210122242)

[4.2 Endpunkte (Beispiele) 5](#_Toc210122243)

[5. Installation und Ausführung 5](#_Toc210122244)

[5.1 Lokale Ausführung 5](#_Toc210122245)

Frontend:

# Backend:

Das Projekt implementiert ein Backend-System auf Basis von Flask, das als Schnittstelle zwischen einem Client (z. B. einer Web- oder Mobile-App) und der Spotify Web API dient.  
Hauptaufgabe ist die Authentifizierung über OAuth2 und die Weiterleitung von Anfragen an Spotify, um Benutzerdaten oder Musik-Informationen abrufen zu können.

Das Backend wurde so konzipiert, dass es leichtgewichtig, dockerisierbar und einfach testbar ist.

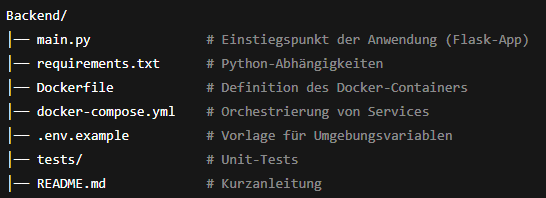
# Technologien & Abhängigkeiten

Die wichtigsten eingesetzten Technologien sind:

* Programmiersprache: Python 3
* Framework: Flask 3.0.3 (Bereitstellung der HTTP-Endpunkte)
* HTTP-Requests: requests – Kommunikation mit der Spotify API
* Konfiguration: python-dotenv – Laden von Umgebungsvariablen aus .env
* Produktionsserver: gunicorn – Bereitstellung des Backends im Container
* Testing: pytest – Unit- und Integrationstests

Alle Abhängigkeiten sind in der Datei requirements.txt dokumentiert.

# Projektstruktur



# Funktionale Beschreibung

## Konfiguration

Über die Datei .env werden zentrale Parameter gesetzt:

* SPOTIFY\_CLIENT\_ID
* SPOTIFY\_CLIENT\_SECRET
* SPOTIFY\_REDIRECT\_URI

Diese Variablen sind notwendig, um den Spotify OAuth2-Flow korrekt auszuführen.

## Endpunkte (Beispiele)

Die Flask-App definiert mehrere Routen, unter anderem:

* Login-Start: Weiterleitung des Benutzers zu Spotify, um die OAuth2-Zustimmung zu erteilen.
* Redirect-Handler:
  + Empfang des Autorisierungscodes von Spotify
  + Austausch des Codes gegen ein Access- und Refresh-Token
* API-Weiterleitungen:
  + Abruf von Benutzerdaten, Playlists oder Musikinformationen über Spotify API
  + Antworten werden im JSON-Format zurückgegeben

# Installation und Ausführung

## Lokale Ausführung

* Repository klonen
* Virtuelle Umgebung erstellen und aktivieren
* Abhängigkeiten installieren:
  + pip install -r requirements.txt
* .env anlegen (auf Basis von .env.example)
* Start der Anwendung:
  + python main.py
  1. Docker-Variante
* Das Projekt ist vollständig dockerisiert:
* .env Datei anlegen
* Container starten:
  + docker-compose up --build
* Das Backend ist standardmäßig unter http://localhost:5000 erreichbar.

1. Tests

Die Tests liegen im Ordner "backend/tests/" und werden mit "pytest" ausgeführt.

1. Erweiterungsmöglichkeiten

* Ergänzung zusätzlicher Spotify-API-Endpunkte (z. B. Playlist-Verwaltung, Suche)
* Integration einer Datenbank (z. B. PostgreSQL) zur Speicherung von Tokens oder User-Informationen
* Deployment auf Cloud-Diensten (z. B. AWS, Heroku, Render)

1. Fazit

Dieses Backend bietet eine skalierbare und containerisierte Lösung, um die Spotify API sicher über OAuth2 einzubinden. Dank Flask bleibt der Code schlank, testbar und leicht erweiterbar.

Jenkins: