

# Prototyping Projektdokumentation

---

Name: Samuel Sättler

E-Mail: saettsam@students.zhaw.ch

URL der deployten Anwendung: <https://discoverartists-saettsam.netlify.app>

## Inhaltsverzeichnis

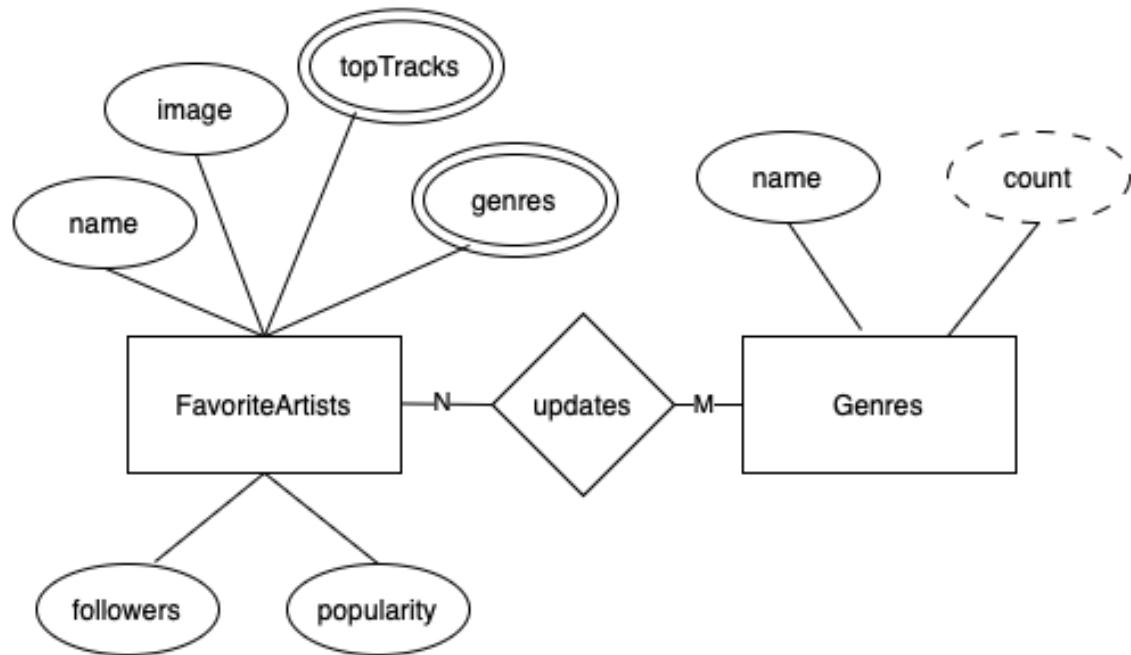
<b>1. Einleitung .....</b>	<b>2</b>
<b>2. Datenmodell .....</b>	<b>2</b>
<b>3. Beschreibung der Anwendung .....</b>	<b>3</b>
<b>3.1. Startseite.....</b>	<b>3</b>
<b>3.2. Künstlersuche .....</b>	<b>4</b>
<b>3.3. Künstlerdetail .....</b>	<b>5</b>
<b>3.4. Favoritenliste.....</b>	<b>6</b>
<b>3.5. Genres.....</b>	<b>7</b>
<b>4. Erweiterungen.....</b>	<b>8</b>
<b>4.1. Spotify API Integration.....</b>	<b>8</b>
<b>4.2. Genre-Auswertung mit Chart.js .....</b>	<b>9</b>
<b>4.3. Top-Track-Preview.....</b>	<b>9</b>
<b>4.4. Dark Mode Styling .....</b>	<b>10</b>

## 1. Einleitung

Die Anwendung «Discover Artists» ermöglicht es Nutzern, Spotify-Künstler zu suchen, anzuzeigen und in einer persönlichen Favoritenliste zu speichern. Grundlage der App ist die Spotify Web API, über welche relevante Künstlerdaten wie Name, Genres, Popularität, Followeranzahl und die beliebtesten Songs (Top-Tracks) bezogen werden. Die gespeicherten Künstler werden in einer Datenbank abgelegt, wobei auch eine genrebasierte Auswertung erfolgt, die auf einer Seite statistisch per Diagramm dargestellt wird.

Zu den Hauptfunktionen gehören eine Künstlersuche mit Live-Zugriff auf Spotify, eine Detailansicht mit Informationen und Vorschau-Player, das Speichern und Entfernen von Favoriten sowie eine grafische Übersicht der häufigsten Genres. Die Anwendung wurde mit SvelteKit entwickelt, nutzt MongoDB zur Datenspeicherung und ist in einem schlichten, dunklen Design gehalten.

## 2. Datenmodell



Die Anwendung arbeitet mit zwei zentralen Collections:

**favoriteArtists**: Enthält die gespeicherten Künstler mit den Basisinformationen und zusätzlich den Top-10-Tracks.

**genres**: Speichert die Häufigkeit von Genres über alle gespeicherten Artists hinweg zur Visualisierung.

Beim Speichern oder Löschen eines Künstlers werden die Zähler für jedes zugehörige Genre entsprechend aktualisiert. Technisch gesehen, besteht eine **implizite** N:M-Beziehung da sie nicht direkt verbunden sind. Ein Artist kann mehrere Genres besitzen und ein Genre kann bei mehreren Artists vorkommen.

### 3. Beschreibung der Anwendung

#### 3.1. Startseite

Route: /

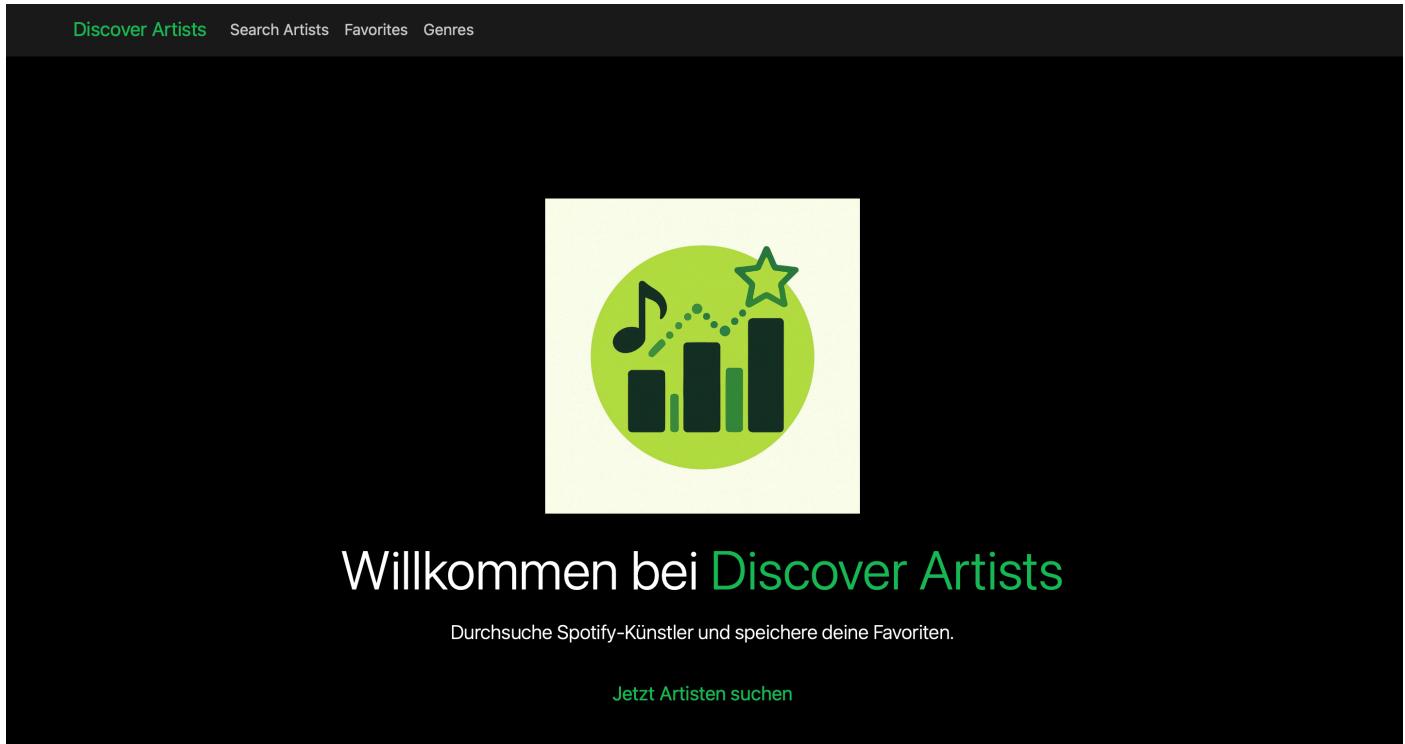


Abbildung 1 - Startseite

Die Startseite zeigt das Logo, einen Titel sowie eine kurze Beschreibung der Anwendung. Per Klick kann zur Künstlersuche navigiert werden. Das Design über die ganze Seite ist schlicht gehalten (möglichste responsiv mit %) und mit Fokus auf dunklem Farbschema.

Dateien:

- routes/+page.svelte
- routes/+layout.svelte
- app.css, app.html

### 3.2. Künstlersuche

Route: /search

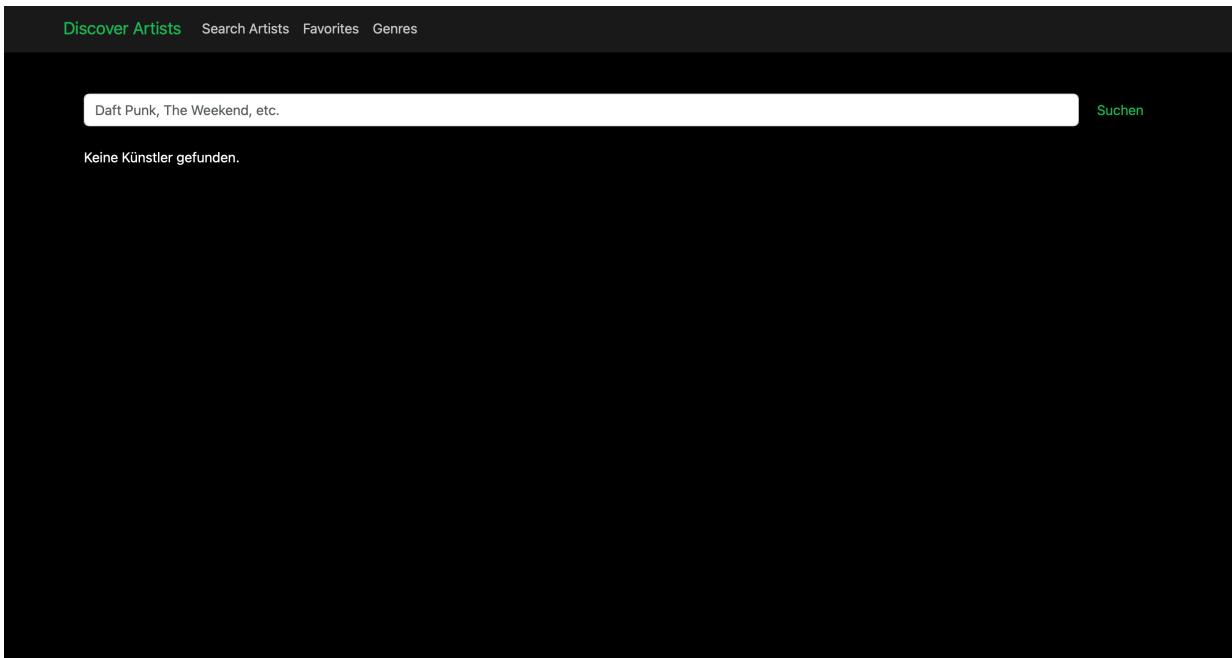


Abbildung 2 - Suchfeld vor Sucheingabe

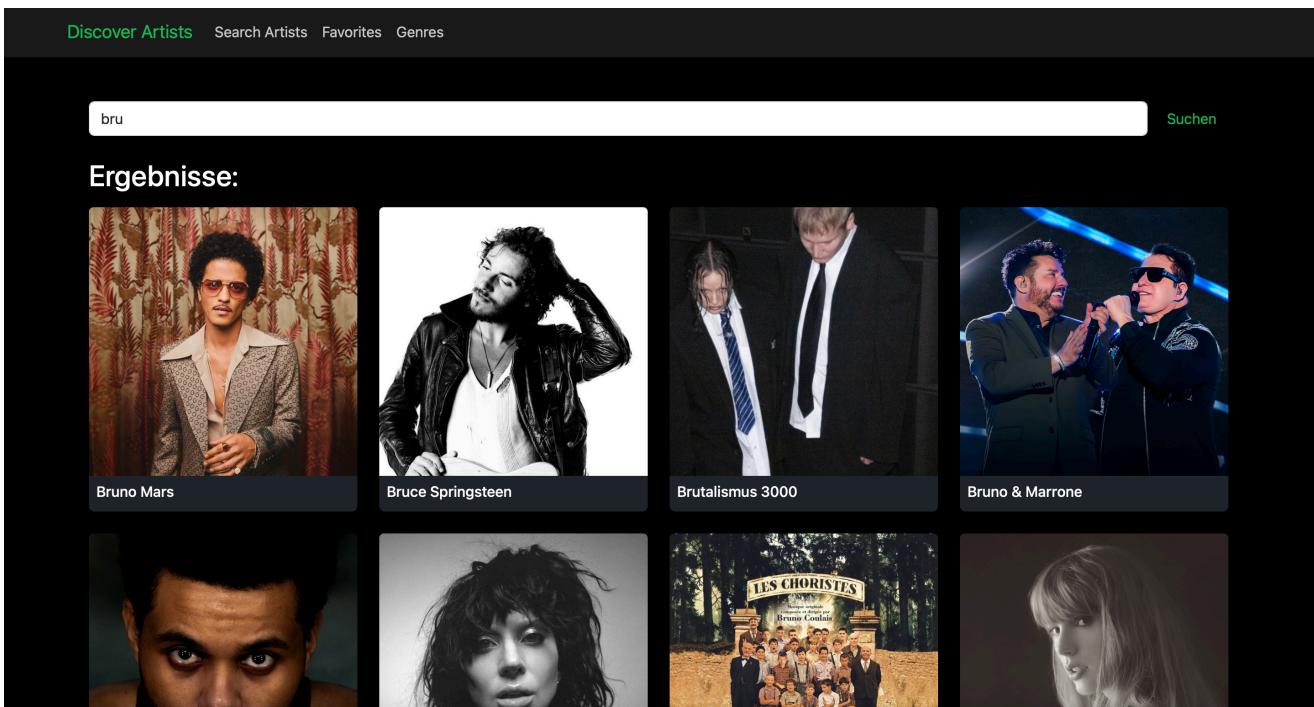


Abbildung 3 - Ergebnisse für Sucheingabe (Default 10 Ergebnisse)

Über ein einfaches Suchfeld können Spotify-Künstler gefunden werden. Die Ergebnisse werden als Raster von ArtistCards mit Bild und Name angezeigt. Die Suche ist direkt verknüpft mit der SpotifyAPI und sucht in dessen Datenbank nach den Informationen zu den Artisten. Per Klick gelangt man zur Detailansicht (/artist/[id]). Der weiterführende Link wird nur mit der SpotifyId ergänzt, die weitere Logik passiert in /artist/[id].

Dateien:

- routes/search/+page.svelte
- routes/search/+page.server.js
- lib/components/ArtistCard.svelte, dazu app.css für styling
- lib/server/spotify.js, dazu .env für SPOTIFY\_CLIENT\_ID und \_SECRET

### 3.3. Künstlerdetail

Route: /artist/[id]

The screenshot shows the artist detail page for Bruno Mars. At the top left is a navigation bar with 'Discover Artists', 'Search Artists', 'Favorites', and 'Genres'. Below the navigation is a back button labeled '← Zurück'. The main area features a large portrait of Bruno Mars wearing a patterned suit. To the right of the portrait is a section titled 'Top-Tracks' listing his most popular songs with their durations: 'Die With A Smile: 4:11', 'APT.: 2:49', 'Just the Way You Are: 3:40', 'Locked out of Heaven: 3:53', 'That's What I Like: 3:26', 'When I Was Your Man: 3:33', 'It Will Rain: 4:17', 'Talking to the Moon: 3:37', 'Marry You: 3:50', and 'Grenade: 3:42'. Below the tracks is a button labeled 'Als Favorit speichern'.

Abbildung 4 - DetailPage für Artisten

This screenshot shows the same artist detail page for Bruno Mars, but the 'Als Favorit speichern' button has been hovered over, causing it to change color. The button is now white with a black outline. The rest of the page content remains the same, including the artist's name, popularity, followers, and the list of top tracks.

Abbildung 5 - Button (hovered) für Artist als Favorit in die DB speichern

Nach Auswahl eines Künstlers, werden via weitergegebene SpotifyId die Daten «gefetched» und die Detailansicht auf der linken Seite via ArtistCard, mit Name, Bild, Genres, Follower-Zahl und Popularität generiert. Die rechte Seite zeigt die Top-10-Tracks inklusive Vorschauplayer (leider von Spotify seit längerer Zeit nicht mehr supported: [Preview URLs Deprecated](#)). Jedoch wie in «Erweiterungen» erwähnt werden, die Songpreviews mittel «spotify-preview-finder» gesucht und zur wiedergabe angezeigt. Die Künstler können mit den TopTracks gespeichert oder entfernt werden.

Zur ArtistCard muss man noch erwähnen, diese wurde so kreiert, dass man sie mehrfach wiederverwenden kann, also dynamisch mit beliebiger Anzahl verschiedenen Attributen. Für die Suche wird nur der Name des Artisten verwendet. Hingegen hier in der DetailPage nutzt man die gleiche Card jedoch mit mehr Information, so wurden Redundanzen vermieden (wird auch in der Favoritenseite zur Auflistung wiederverwendet). Die Datenbankfunktion `getFavoritArtistById()` wird in dieser Applikation verwenden zur Überprüfung ob ein Eintrag schon existiert (für den dynamischen Speichern und Entfernen Button), falls jedoch ein Eintrag schon existiert werden die Daten aus dieser Abfrage verwendet um Ressourcen und Performance zu optimieren.

Dateien:

- routes/artist/[id]/+page.svelte
- routes/artist/[id]/+page.server.js
- lib/components/ArtistCard.svelte, dazu app.css für styling
- lib/components/TopTracks.svelte
- lib/server/spotify.js, dazu .env für SPOTIFY\_CLIENT\_ID und \_SECRET
- lib/server/db.js

### 3.4. Favoritenliste

Route: /favorites

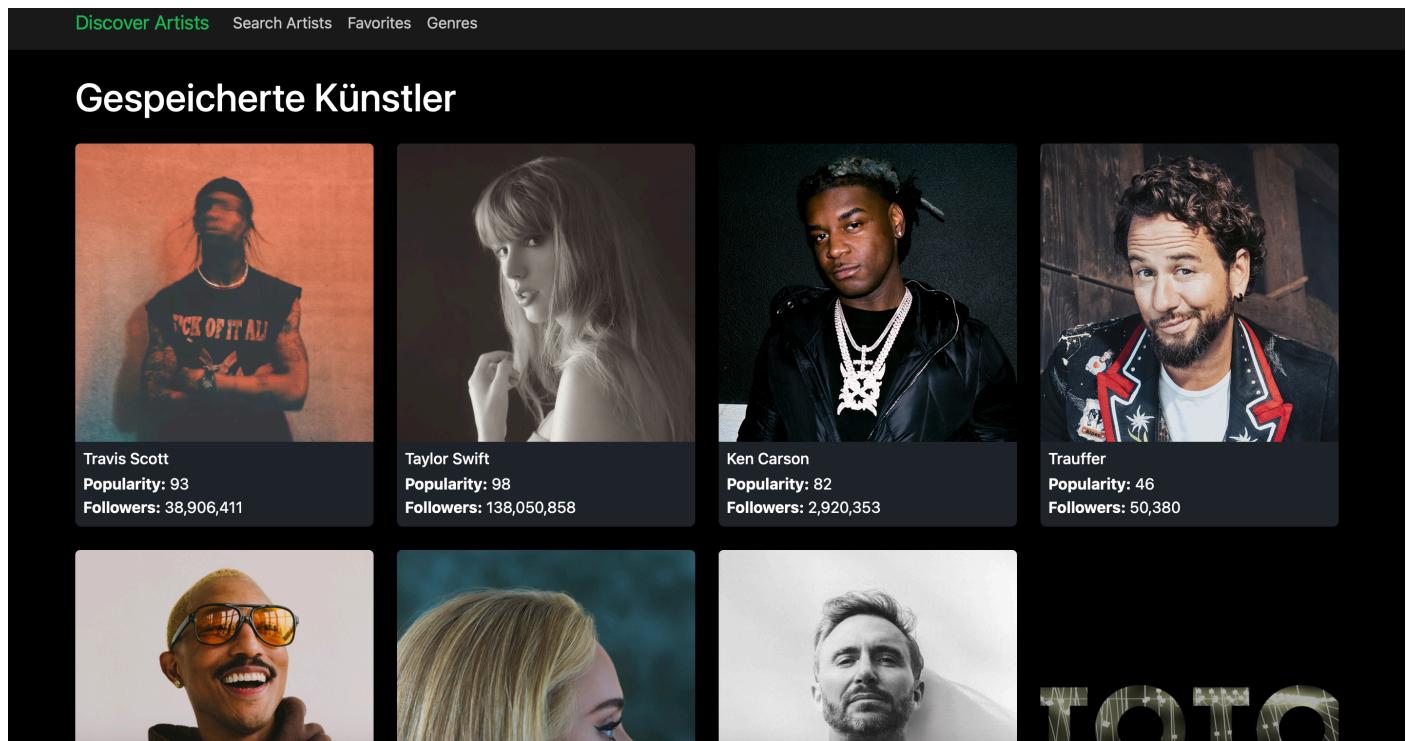


Abbildung 6 - Favoriten page mit einer Liste aller gespeicherten Artists

Zeigt alle gespeicherten Künstler in einer ähnlichen Übersicht wie bei der Suche. Diese Daten stammen aus der MongoDB Datenbank via `getFavoriteArtists()`. Per Klick auf ein Element gelangt man erneut zur Detailansicht zu `/artist/[id]`, für die Wiederverwendbarkeit und von der Logik her wird für das Anzeigen die gleiche Detailansicht verwendet wie für die Detailansicht nach der «Künstlersuche». Also wird ebenfalls im `<a href...` die `artist.id` (SpotifyID) mitgegeben aber dann per ID in der Datenbank abgefragt mit allen Informationen um es möglichst effizient zu gestalten

Dateien:

- routes/favorites/+page.svelte
- routes/favorites/+page.server.js
- lib/server/db.js

### 3.5. Genres

Route: /genres

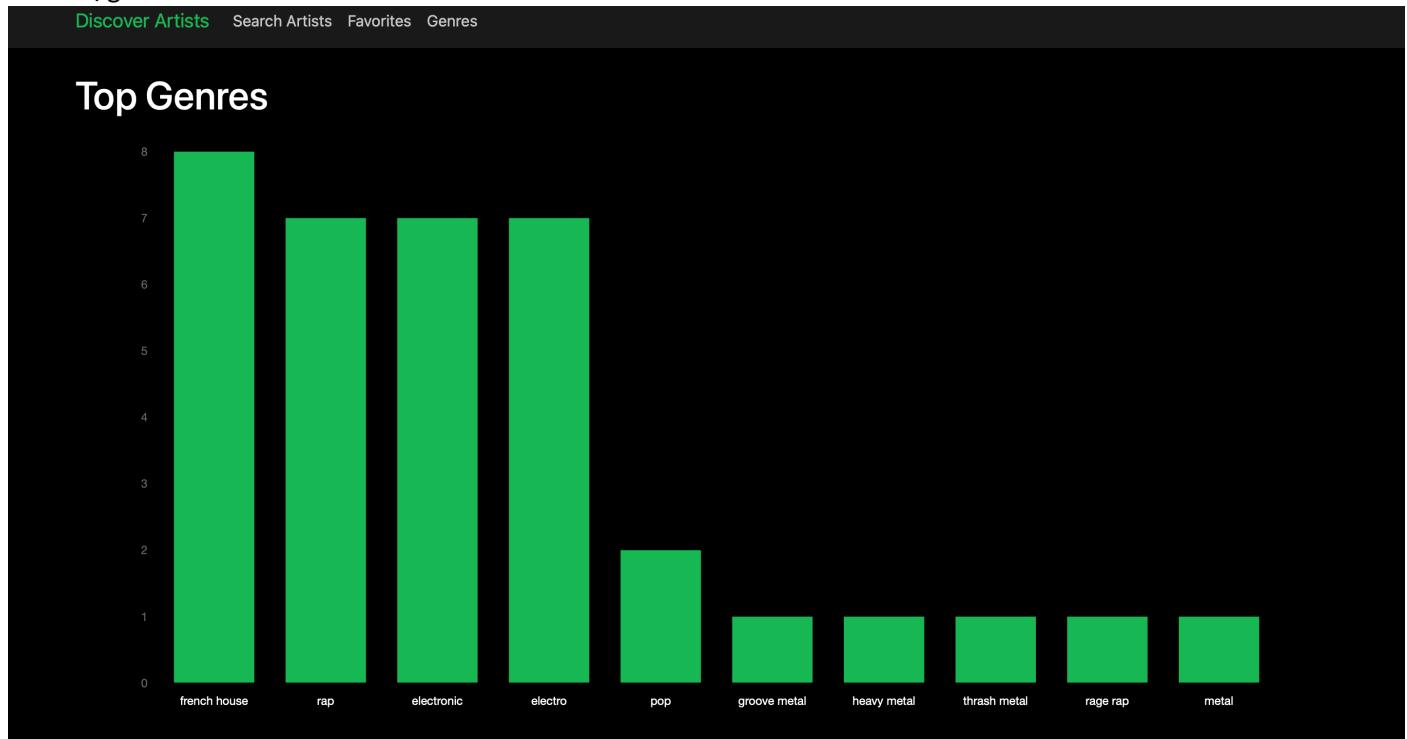


Abbildung 7 - Genre page mit Statistik der genre über alle Arists

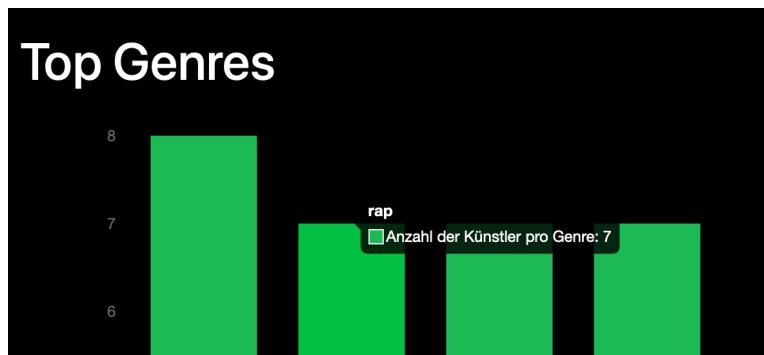


Abbildung 8 - Hovern zeigt den Namen des Genre und die anzahl

Hier wird ein Balkendiagramm angezeigt, das die Häufigkeit gespeicherter Genres auswertet. Die Darstellung erfolgt mit ChartJS. Die Labels und Balkenfarbe wurden angepasst. Bei jeder Speicherung/Lösung eines Künstlers wird die Zählung der Genres aktualisiert (`updateGenres()`). Die Logik für das Erstellen des Charts passiert im «GenreBarChart» File und die `+page.svelte` und `server` werden nur gebraucht für das «fetchen» der Daten und Weiterleiten an die Svelte Card.

Dateien:

- routes/genres/+page.svelte, dazu app.css für styling
- routes/genres/+page.server.js
- lib/components/GenreBarChart.svelte

## 4. Erweiterungen

### 4.1. Spotify API Integration

Ein zentrales Element der Anwendung ist die direkte Anbindung an die [Spotify Web API](#). Dafür wurde ein eigener Zugriffstoken über das Spotify Developer Dashboard generiert. Die Schritte im Überblick:

#### 1. Spotify Dashboard einrichten

Es wurde ein eigener Spotify-Client im Developer Dashboard erstellt. Dabei mussten Redirect-URLs konfiguriert und die `client_id` sowie `client_secret` gespeichert werden (in `.env` und als private SvelteKit-Variablen verfügbar). Wichtig war eigentlich nur das Generieren des Tokens. Webseite und Redirect URLs wären nur relevant, wenn ich eine OAuth eingebaut hätte, zum Beispiel für das Abfragen von Spotifyuser Daten, wie die persönlichen Lieblingskünstler, -Genre oder -Songs.

The screenshot shows the Spotify Developer Dashboard interface. At the top, there's a navigation bar with the Spotify logo, 'Spotify for Developers', 'Documentation', 'Community', and a user profile for 'samuel.saettler'. Below the navigation, the URL 'Dashboard > Discover Artists' is visible, along with a 'Metrics' button. The main content area is titled 'Basic Information' with a large red 'D' icon. It contains several input fields and sections: 'Client ID' (text: 'Oaab036311384d808c5fc1dab243a289'), 'App Status' (button: 'Development mode'), 'App name' (text: 'Discover Artists'), 'App description' (text: 'SvelteKit-Studentenprojekt zur Suche und Analyse von Künstlern'), 'Website' (text: 'https://discoverartists-saettsam.netlify.app'), 'Redirect URIs' (list: 'https://discoverartists-saettsam.netlify.app', 'http://127.0.0.1:5173'), 'Bundle IDs' (empty), 'Android packages' (empty), 'APIs used' (list: 'Web API'), and an 'Edit' button at the bottom left.

Abbildung 9 - SpotifyAPI Developer Dashboard für die Einstellungen und nötig für das generieren des Tokens

## 2. Tokenverwaltung implementieren

In der Datei lib/server/spotify.js wurde eine `getSpotifyAccessToken()` Funktion erstellt, die bei jedem Aufruf über `client_credentials` Flow ein Token abruft.

## 3. Recherche in der Spotify-Dokumentation

Die Struktur der Antworten (z. B. bei `/v1/search` oder `/v1/artists/:id/top-tracks`) musste sorgfältig aus der Dokumentation analysiert werden, um nur relevante Daten aus der SpotifyAPI wie «genres», «followers.total», «popularity», «topTracks» etc. zu extrahieren.

## 4. Top-Track-Auswahl

Die zehn populärsten Tracks pro Künstler wurden mit Marktfilter «CH» geladen. Auch ohne «preview\_url» werden Titel und Dauer dargestellt (feature ist Deprecated).

## 5. Speicherung in MongoDB

Beim Speichern eines Künstlers wurden «topTracks» direkt bei der Collection `favoriteArtist` mitgespeichert, sodass diese auch offline verfügbar sind. Die Genre-Zählung wird dabei jedes Mal bei Kreierung oder Löschung aktualisiert.

Dateien:

- lib/server/spotify.js
- lib/server/db.js
- routes/search/+page.server.js
- routes/artist/[id]/+page.server.js
- .env (lokal, nicht auf Github)

## 4.2. Genre-Auswertung mit Chart.js

Die Seite `/genres` zeigt ein dynamisches Balkendiagramm mit allen gespeicherten Genres aus der Datenbank. Implementiert in `GenreBarChart.svelte`. Die Genres werden serverseitig aggregiert und als JSON an die Komponente übergeben. Das Bar-diagramm wurde mithilfe der [ChartJS](#) Dokumentation gefertigt.

Dateien:

- routes/genres/+page.svelte
- routes/genres/+page.server.js
- lib/components/GenreBarChart.svelte
- lib/db.js

## 4.3. Top-Track-Preview

Unabhängig davon, ob ein Track eine Vorschau hat oder nicht, wird in der `TopTracks`-Komponente ein Bereich angezeigt mit Songdauer und Songnamen. Wie schon erwähnt ist die «preview\_url» nicht mehr richtig verfügbar und meistens «null». Jedoch gibt es einen Workaround, und zwar mit der Library [spotify-preview-finder](#). Diese Library durchsucht mit Hilfe des gegebenen Songtitels die öffentlich zugängliche Spotify-Webseite mittels SpotifyAPI abfragen, um funktionierende MP3-Preview-Links (z. B. p.scdn.co/...) zu extrahieren. Um die Suche zu verbessern und schneller zu machen, übergebe ich ihm meine Spotify Credentials via das Setzen von Umgebungsvariablen für `process.env`

Dateien:

- Lib/server/previewUrlLibaray.js
- lib/components/TopTracks.svelte
- lib/server/spotify.js
- routes/artist/[id]/+page.server.js
- routes/artist/[id]/+page.svelte

#### 4.4. Dark Mode Styling

Die ganze Anwendung wurde im Dark Mode designt. Die Farben sind konsistent und einfach/modern gewählt. Icons werden mit Bootstrap Icons geladen. Die meisten Styling Elemente sind nicht via CSS sondern via den existierenden Bootstrapelementen implementiert worden.

Dateien:

- app.css und diverse Bootstrap Elemente/Icons