DHBW
Duale Hochschule
Baden-Württemberg
Mosbach

Dokumentation der Projektarbeit

ScriptJockey

von 9495107, 4706893, 9608900

INF20B

21. März 2022

Vorlesung: Web-Services

Inhaltsverzeichnis

1. Organisation und Projektplan

Gruppenmitglieder

Name	Matrikelnummer
Robin Wollenschläger	9495107
Timo Max	4706893
Marco Lappe	9608900

Projektplan

Datum	Tätigkeit
08.02. – 22.02.2022	Ideenfindung
28.02.2022	Themenfestlegung und Aufgabenverteilung
03.03.2022	Initiales Anlegen des Webservers
06.03.2022	Anbindung externer Web-Services
18.03.2022	Fertigstellung Web-Service
19.03.2022	Fertigstellung Web-Oberfläche
20.03.2022	Fertigstellung BPMN und Dokumentation
21.03.2022	Präsentation des Web-Services

Aufgabenverteilung

Aufgabe	Person
Anlegen des Webservers	Robin Wollenschläger
Anbindung Spotify-API an Webservice	Timo Max
Anbindung Genius-API an Webservice	Marco Lappe
Weboberfläche	Robin Wollenschläger
Performance-Tests	Marco Lappe
BPMN	Marco Lappe, Timo Max
Dokumentation	Alle

2. Projektbeschreibung

2.1 Projektidee

Die Idee des Projekts ist es, ein Voting-System zum Steuern der Abspielreihenfolge einer vorgegebenen Playlist zu entwickeln. Die Teilnehmer am Voting-System können für ein Lied der Playlist abstimmen, das Lied mit den meisten Votes wird jeweils als nächstes gespielt und aus der Queue entfernt. Die Lieder werden dabei auf einem zentralen Gerät abgespielt. Außerdem werden den Zuhörern die Lyrics zum aktuell abgespielten Lied angezeigt. Als Streamingdienst zum Abspielen der Lieder wird Spotify verwendet. Die auf der Web-Oberfläche angezeigten Daten liefert der von uns entwickelte Web-Service, der dazu auf die APIs von externen Web-Services zugreift.
Ein Beispiel zur Verwendung des Web-Services ist ein DJ auf einer Party. Der DJ stellt dabei die Playlist und das Gerät zum Abspielen der Lieder zur Verfügung, über die Weboberfläche können die Zuhörer über die Abspielreihenfolge abstimmen. Der Web-Service steuert die Abspielreihenfolge der Lieder und liefert die Lyrics zum aktuellen Lied.

2.2 Verwendete Web-Services

Das Projekt besteht aus 3 Webservices, von welchen einer selbst implementiert ist. Die beiden externen Webservices sind Genius und Spotify, auf deren APIs der selbst entwickelte Webservice ScriptJockey zugreift.
ScriptJockey nutzt diese beiden Webservices, um Playlists aus Spotify und Lyrics von Genius zu laden und Lieder mithilfe von Spotify abzuspielen. Über ScriptJockey kann sich eine Person, z.B. der DJ, bei Spotify anmelden und eine beliebige Playlist auswählen, von welcher anschließend Lieder abgespielt werden. Über die ScriptJockey-Weboberfläche werden die Lieder der Playlist den Zuhörern angezeigt, die jeweils dafür voten können, welches Lied der Playlist als nächstes abgespielt werden soll. Zusätzlich kann der DJ mithilfe von ScriptJockey die Wiedergabe über Spotify steuern, beispielsweise kann er die Wiedergabe pausieren oder das Wiedergabegerät ändern. Diese Steuerung wurde anfangs benötigt, ist aber in der aktuellen Version nicht mehr nötig, da der DJ direkten Zugriff auf den Web-Player von Spotify hat. Die zugehörigen Endpunkte (z.B. /player/play, /player/pause und /switchPlayer) sind dennoch im Web-Service und damit auch in der OpenAPI enthalten, sodass sie für einen zukünftigen automatisierten Ablauf verwendet werden können.

2.2.1 Spotify Web API

Die Spotify Web API ermöglicht es Daten aus dem Spotify-Musikkatalog abzurufen. Die Endpunkte liefern im JSON-Format Informationen wie Interpreten, Alben und Titel direkt aus dem Spotify-Katalog. Abhängig von der Berechtigung des Benutzers bietet die API Zugriff auf benutzerbezogene Daten, das heißt, Playlists, die in der Benutzerbibliothek gespeichert sind.
Die Basis URI für alle API-Anfragen lautet https://api.spotify.com/v1.
Die Spotify Web API basiert auf REST-Prinzipien und unterstützt die üblichen Anfragemethoden GET, PUT, POST und DELETE.
Eine Übersicht aller Endpunkte mit ausführlicher Dokumentation der Requests und anschließenden Response findet man hier: https://developer.spotify.com/documentation/web-api/reference/#/
Um auf die Endpunkte zugreifen zu können, muss man seine Applikation bei Spotify registrieren, erhält dann eine client_id und einen client_secret, mit denen man sich einen access-Token generieren lassen kann. Dieser access-Token wird bei jedem Aufruf der Spotify Web API benötigt.
Wie bereits angedeutet kann man auch auf benutzerbezogene Daten (z.B. Playlists aus der Benutzerbibliothek) zugreifen. Voraussetzung hierfür ist allerdings eine Anmeldung bei Spotify (Premium). Es lassen sich auch verschiedene Scopes definieren, das heißt, welche Berechtigungen unsere Applikation erhält, z.B. Playlists verändern. Diese Zugriffsberechtigungen müssen nach der Anmeldung vom Benutzer gewährt werden. Hier findet man eine Übersicht über die möglichen Scopes: https://developer.spotify.com/documentation/general/guides/authorization/scopes/
Die API bietet mehrere Authentifizierungsmöglichkeiten, wobei wir den „Authorization Code Flow“ verwendet haben. Hierbei muss man sich zuerst über den Spotify Account Service anmelden (client_id aus der Registierung der Applikation muss vorhanden sein) und wenn nötig die angeforderten Zugriffsberechtigungen gewähren. Nach erfolgreicher Anmeldung kann man mithilfe des client_secret-Keys einen access und refresh Token anfordern. Mit diesem access Token können und müssen nun alle API-Endpunkte aufgerufen werden. Der access Token hat eine Gültigkeit von einer Stunde, weshalb noch ein refresh Token mitgeliefert wird, um damit einen neuen access Token zu generieren.

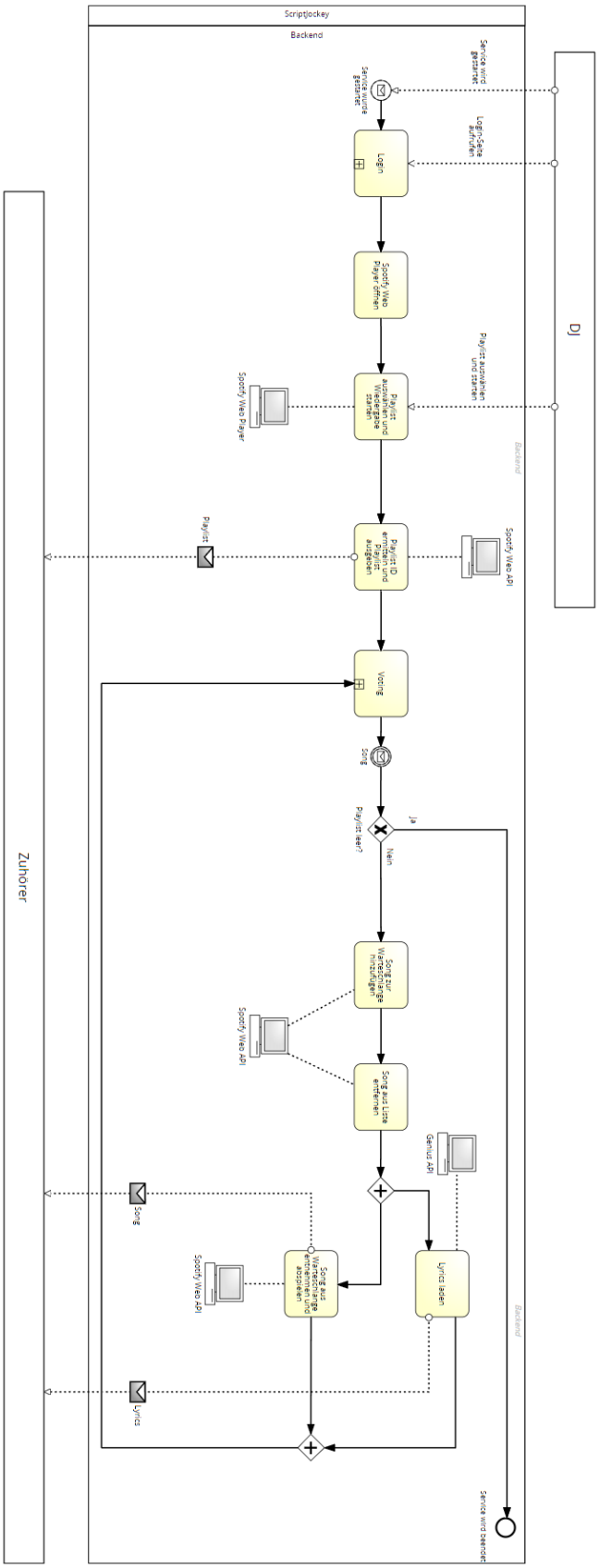
2.2.2 Genius API

Der Webservice von Genius wird dazu verwendet, um die jeweiligen Lyrics zum abgespielten Lied zu erhalten. Die Dokumentation des Genius-Webservices ist hier zu finden: https://docs.genius.com/.

Zum Aufruf der Endpunkte ist außerdem ein access-Token nötig, den man nach der Registrierung der Anwendung bei Genius erhält. Außerdem erhält man dort eine Client-ID und ein Client-Secret. Mit diesen sind authentifizierte API-Aufrufe für benutzerbezogene Daten möglich.

Da in unserem Webservice lediglich das Suchen nach Lyrics benötigt wird, verwenden wir aus der Genius-API nur die GET-Anfrage search. Für diesen Endpoint ist keine Authentifizierung nötig, weswegen wir lediglich den access-Token benötigen. Um die Qualität bei der Suche nach den Lyrics zu erhöhen, werden die von Spotify übergebenen Daten zu Titel und Interpret leicht angepasst, z. B. werden häufige Sonderzeichen und Klammern entfernt. Die search-Anfrage wird mit diesem geänderten Titel und Interpret des Liedes aufgerufen und liefert u. a. die URL einer Unterseite zurück, auf der die Lyrics zu finden sind. Der von uns entwickelte Webservice überprüft, ob Titel und Interpret des Aufrufs und der Antwort übereinstimmen und nutzt dann ein npm-Package, um die Lyrics dieser Webseite zu extrahieren und sendet sie an den Aufrufer zurück.

3. BPMN-Modell des Web-Services

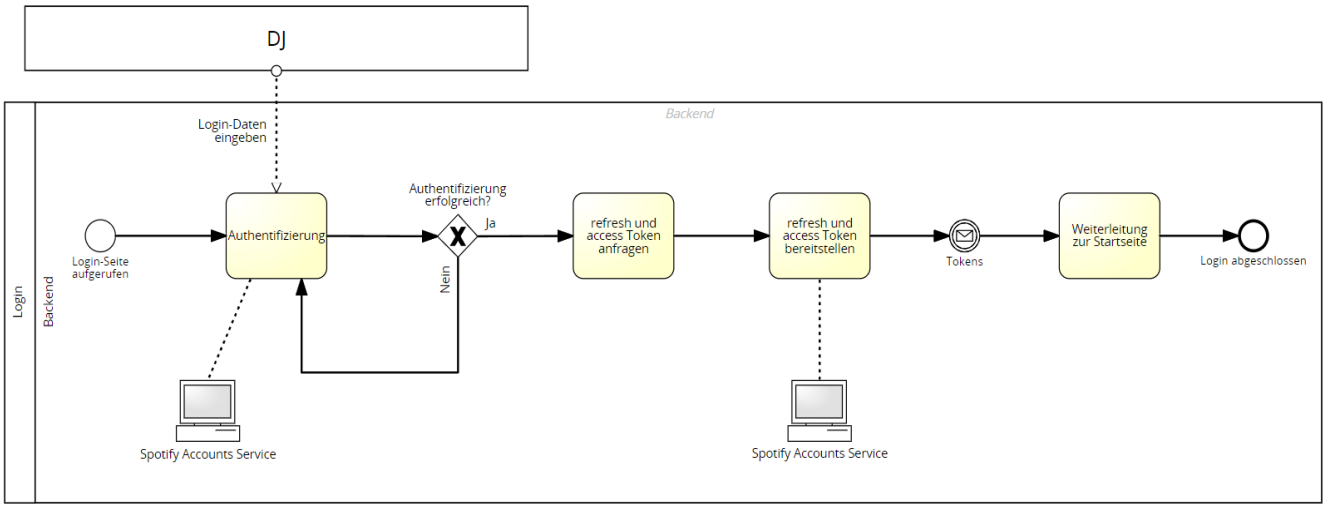


Anmerkung: Die Prozessmodellierung befindet sich zusätzlich noch in der extra PDF-Datei *BPMN_ScriptJockey.*

Zu Beginn startet der DJ den Service. Anschließend ruft er die Login-Seite auf und wird auf die Login-Seite von Spotify weitergeleitet, auf der er sich anmeldet. Der Ablauf des Logins wird im Unterprozess *Login* dargestellt. Nach erfolgreichem Einloggen wird der Spotify Web Player (https://open.spotify.com/) geöffnet. Hier wählt der DJ eine Playlist aus und startet die Wiedergabe. Im Anschluss wird die ID der abgespielten Playlist mithilfe der Spotify Web API ermittelt und der Inhalt an die Zuhörer im Front-End übermittelt.

Daraufhin können die Zuhörer für die Lieder der Playlist voten, dieser Vorgang ist im Unterprozess *Voting* detailliert dargestellt. Nach Abschluss einer Voting-Phase liefert das Voting den Song mit den meisten Votes. Falls die Playlist allerdings leer ist, wird der Prozess beendet, ansonsten wird der Song über die Spotify Web API zur Abspiel-Warteschlange hinzugefügt. Anschließend wird der Song ebenfalls durch die Spotify Web API aus der Liste entfernt, da für ihn nicht mehr gevotet werden darf. Danach werden über die Genius API die Lyrics zum Song geladen und den Zuhörern angezeigt, außerdem wird unter Verwendung der Spotify Web API der Song aus der Warteschlange entfernt, abgespielt und den Zuhörern angezeigt. Danach startet die Voting-Phase für den nächsten Song.

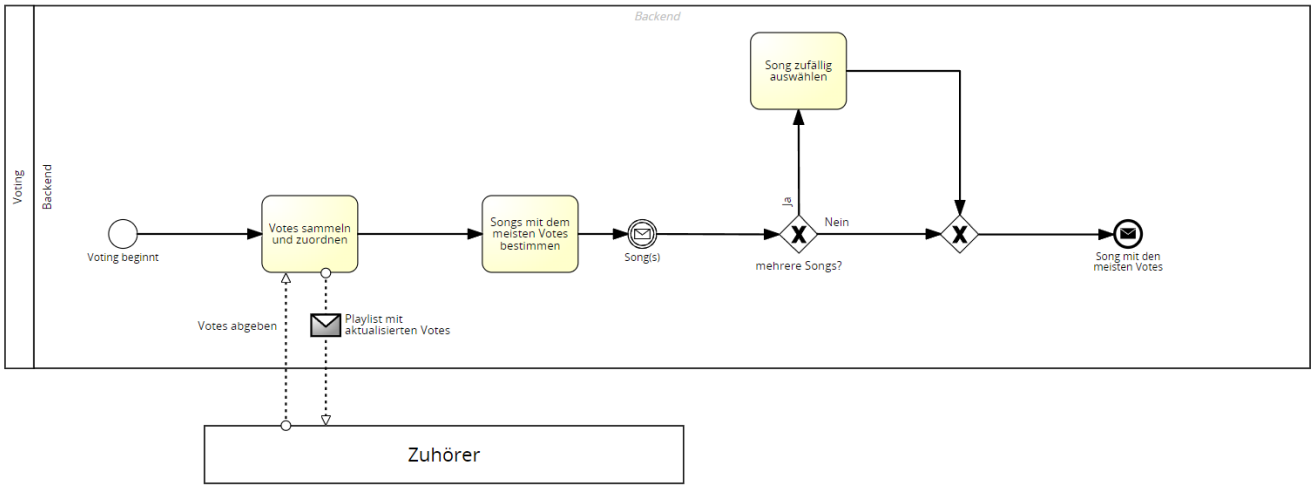
3.1 Login



Sobald die Login-Seite aufgerufen wurde, gibt der DJ seine Anmeldedaten für seinen Spotify Premium Account ein. Wenn die Authentifizierung durch den Spotify Account Service erfolgreich war (ggf. noch Berechtigungen gewähren), dann kann der access- und refresh-Token angefragt werden. Wenn alle übergebenen Parameter stimmen, liefert der Spotify

Account Service einen access- und refresh-Token zurück. Mit dem refresh-Token kann der access- Token, der nach 60 Minuten abläuft, neu generiert werden.
Mit dem access-Token können alle Endpunkte der Spotify Web API aufgerufen werden.
Nach Erhalt der Tokens wird der DJ auf die Startseite weitergeleitet und der Login-Prozess ist erfolgreich abgeschlossen.

3.2 Voting



Sobald das Voting beginnt, können die Zuhörer ihre Votes abgeben. Diese Votes werden gesammelt und den jeweiligen Songs zugeordnet. Die Zuhörer bekommen auf der Web-Oberfläche immer einen aktualisierten Stand der Votes angezeigt. Nach Abschluss einer Voting-Phase wird der Song mit den meisten Votes bestimmt. Eine Voting-Phase endet immer wenige Sekunden bevor der aktuell abgespielte Song zu Ende ist. Wenn mehrere Songs dieselbe Anzahl an Votes erhalten haben, wird aus diesen Songs einer ausgewählt. Nach Abschluss des Votings steht also ein Song fest, der dann zur Warteschlange hinzugefügt werden kann.

4. Implementierung

4.1 Web-Oberfläche

Im Vorfeld haben wir analysiert welche Funktionen das Frontend beinhalten soll und sind zum dem Schluss gekommen, dass die Nutzung eines großen Frameworks durch den großen Overhead eher ein Nachteil ist.
Daher haben wir uns bewusst für den Einsatz von HTML in Kombination mit Bootstrap und das eher leichtgewichtige Framework jQuery zur DOM-Manipulation entschieden.
Um eine hohe Konformität und Responsivität zu gewährleisten haben wir versucht so wenig eigenes css, wie möglich einzubringen und das Design von Bootstrap bestmöglich für unsere Zwecke zu nutzen.
Für Servergesteuerte Live-Updates haben wir über das Paket socket.io eine WebSocket-Verbindung aufgebaut.

Im Vorfeld haben wir uns Gedanken über die im Frontend benötigten Funktionen gemacht:

- Muss:
  - Grundlayout/Navigation
  - Anzeige der Playlist
  - Votingfunktion
  - Verwaltung für DJ
  - Manuelle Spotify Synchronisierung
  - Anzeige aktueller Song
  - Anzeige Lyrics für aktuellen Song
- Soll:
  - Anzeige nächster Song
  - Titel-Suchfunktion

Und anhand dieser Liste die Entwicklung gestartet.

Zum Grundlayout und der Navigation gehören einige Endpoints des nodeJS-Servers, die die benötigten HTML-Dateien liefern. Grundsätzlich wird beim ersten Laden der Seite (URL: „/“) eine index.html-Datei ausgeliefert, die das Grundlayout beinhaltet und weitere benötigte Pakete und Frameworks nachlädt.
Dazu gehören auch eigene JavaScript-Klassen, die beispielsweise für das Routing, als Controller, für die Initialisierung und den Socket verantwortlich sind.

Das Grundlayout enthält eine Navigation-Bar, die fix am oberen Ende steht und dem Benutzer somit alle wichtigen Funktionen auf einen Blick darstellt.
Weiterhin ist ein Container über das HTML-Tag „<main>“ integriert, das für die Hauptinhalte der Seite dynamisch ausgetauscht wird.
Hier kommt das eigene Routing ins Spiel, das je nach Menüpunkt die jeweiligen HTML-Dateien per http-Request vom Server lädt und dann per DOM-Manipulation mittels jQuery in den genannten Container einfügt.
Somit ist die Seite grundsätzlich als SPA (=Single Page Application) ausgeführt und folgt dem MVC-Pattern (=Model View Control). Die Aufteilung und Trennung von View und Control wird über den eigenen Controller gewährleistet, der Daten vom Server per http-Request holt, diese verarbeitet, aufbereitet und in den Kontext der HTML-Struktur einfügt.
Als Model ist grundsätzlich der nodeJS-Server anzusehen.
Alle WebSocket-Events sind in einer separaten Klasse definiert, die steuert, wie auf die definierten Events reagiert werden soll und wie die Daten verarbeitet werden sollen.
Die Klasse „Utils“ enthält zusätzlich noch Logik, um mit Cookies umzugehen.

Beim ersten Laden der Seite zeigt die Navigation den Menüpunkt „I’m the DJ“ an. Da dem Server im Vorfeld nicht bekannt ist, wessen Spotify-Playlist synchronisiert werden soll ist der Menüpunkt für alle sichtbar.
Nach einem Login per Spotify kann Serverseitig auf den zugehörigen Cookie geprüft werdend und damit ein Client als DJ authentifiziert werden.
Clientseitig wird die erste Synchronisierung der Playlist zum Anlass genommen, einen eigenen Cookie zu setzen und daraufhin bei allen anderen Teilnehmern den Menüpunkt „I’m the DJ“ auszublenden und somit Datenmanipulation und Verwirrung zu vermeiden.

Solange das Backend auf die betreffende Anfrage mit dem http-Status 501 antwortet, wird eine Meldung angezeigt, dass die Party noch nicht gestartet ist. Durch Neuladen der Seite aktualisiert sich dieser Status und, wenn synchronisiert, wird eine Playlist geladen. Diese wird in eine HTML-Tabelle geparst und an den main-Container angehängt.
Der Client kann sich die gesamte Liste ansehen und mit Klick auf eine Tabellenzeile kann ein Vote vergeben werden.
Grundsätzlich können nur Votes hinzugefügt und keine abgezogen werden, der Song mit den meisten Klicks bekommt also die höchste Anzahl an Votes.
Diese Votes werden zwischen Server und allen Clients synchronisiert und serverseitig wird

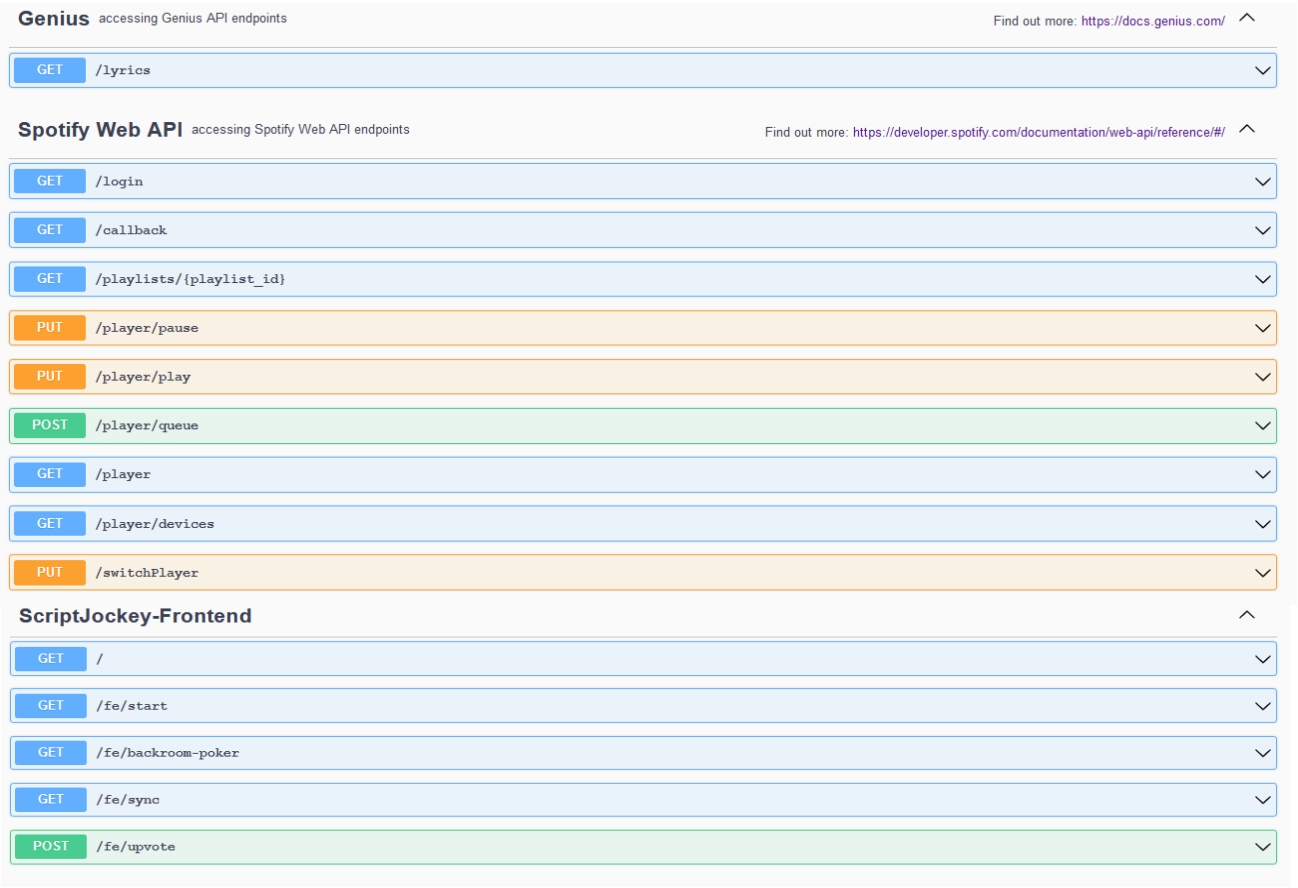
anhand dieser Votes über den nächsten Titel entschieden und automatisiert dieser nach Ende des vorangegangenen abgespielt.
Per Socket-Event wird dieser aktuelle Titel bei einer Änderung an alle Clients verteilt und am unteren Rand der Seite in einem Jumbotron-Container Informationen darüber (Titel, Artist, Album, Cover-Bild) angezeigt.
Über einen Expand-Button im oberen rechten Eck dieses Containers kann dieser nach oben hin aufgeklappt werden und es werden die zugehörigen Lyrics geladen. Falls diese nicht in der Genius-Datenbank gefunden werden, wird stattdessen eine entsprechende Fehlermeldung angezeigt.
Über eine Suchfunktion im rechten Eck der Navigationsleiste am oberen Ende der Seite kann per Volltextsuche noch die gesamte geladene Playlist durchsucht werden, um somit favorisierte Titel leichter zu finden.

4.2 Web-Service

Wir haben unseren eigenen Webservice auf Basis eines nodeJS-Servers realisiert. Dabei nutzten wir das JavaScript Superset Typescript verwendet, um Typsicherheit zu erlangen.
Das npm-Package express half uns dabei einfach und schnell einen Webserver zu implementieren.
Ports und weitere Configurationen nehmen wir über eine .env-Datei vor, die durch das npm-Package dotenv gelesen und als Attribut an die Framework-Variable process angehängt wird. Das gewährleistet uns verschiedene Konfigurationen anhand der Umgebung auszuwählen und somit die Applikation dynamisch auf verschiedene Environments anzupassen.
Da wir keine eigene Authentifizierung nutzen, sondern das über Spotify gesteuert wird und da wir keine Datenmengen dauerhaft speichern müssen haben wir uns gegen eine Datenbank entschieden. Entsprechend der Dynamik der gehandhabten Daten rufe wird diese just in time von den jeweiligen Api’s ab.

Die OpenAPI-Definition unseres Web-Services lässt sich unter */swagger-ui* aufrufen.
Hier folgt eine Übersicht über unsere Endpunkte, teilweise werden diese allerdings nicht in der aktuellen Version verwendet, funktionieren aber *(/player/pause, /player/play, /player/devices und /switchPlayer)*. Außerdem können die Endpunkte */login* und */callback* nicht über die Try-It-Out-Funktion getestet werden.

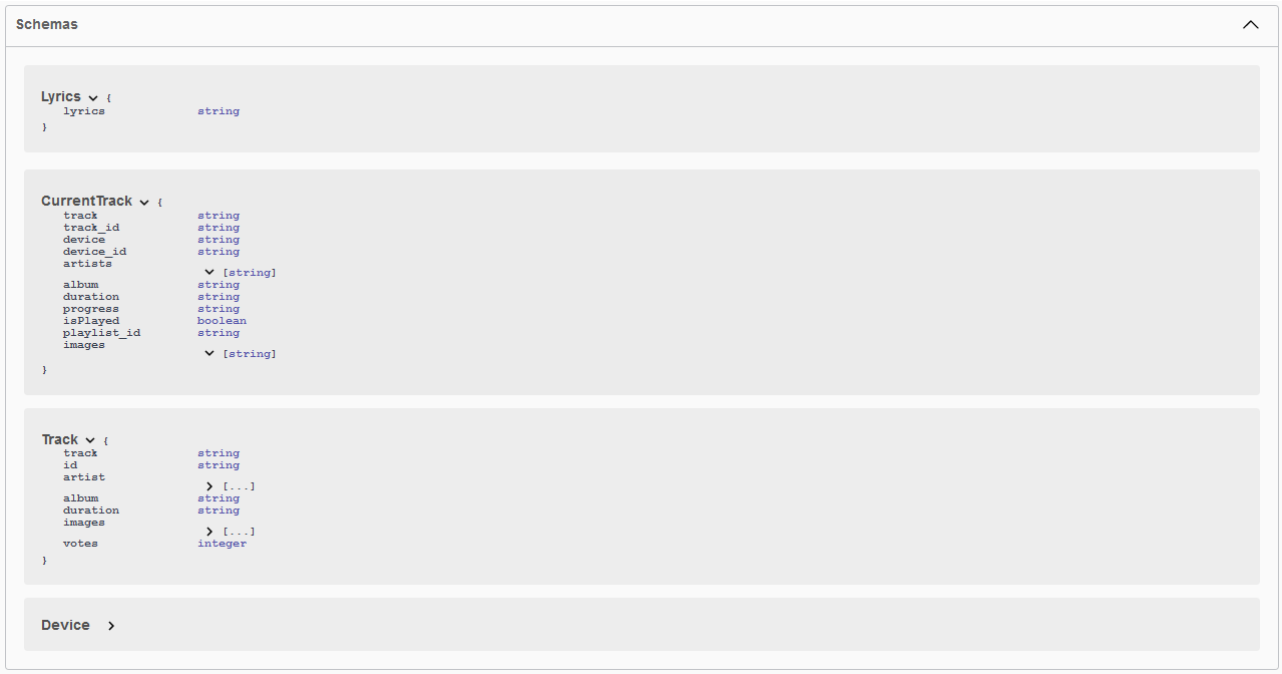
Für alle Spotify Web API-Endpunkte ist eine Anmeldung bei Spotify nötig, da man einen access-Token benötigt.



Der Endpunkt */lyrics* liefert die Lyrics zum angegebenen Titel und Interpreten und gibt als Antwort den Datentyp *Lyrics* zurück.
Der Endpunkt */login* erfragt die Authentifizierung des Nutzers, indem er auf die Spotify-Anmeldeseite weiterleitet.
Der Endpunkt */callback* erfragt refresh- und access-token. Er wird nach erfolgreicher Anmeldung beim */login*-Endpoint aufgerufen.
Der Endpunkt /playlists/{playlist_id} liefert den Inhalt der angegeben Playlist ID, der Datentyp ist ein Array vom Typ *Track*.
Der Endpunkt */player/queue* fügt einen Song anhand seiner ID zur Wiedergabewarteschlange hinzu.
Der Endpunkt */player* liefert den aktuell gespielten Song und zusätzlich Informationen über das Gerät, auf dem abgespielt wird. Als Rückgabedatentyp wird *CurrentTrack* verwendet.

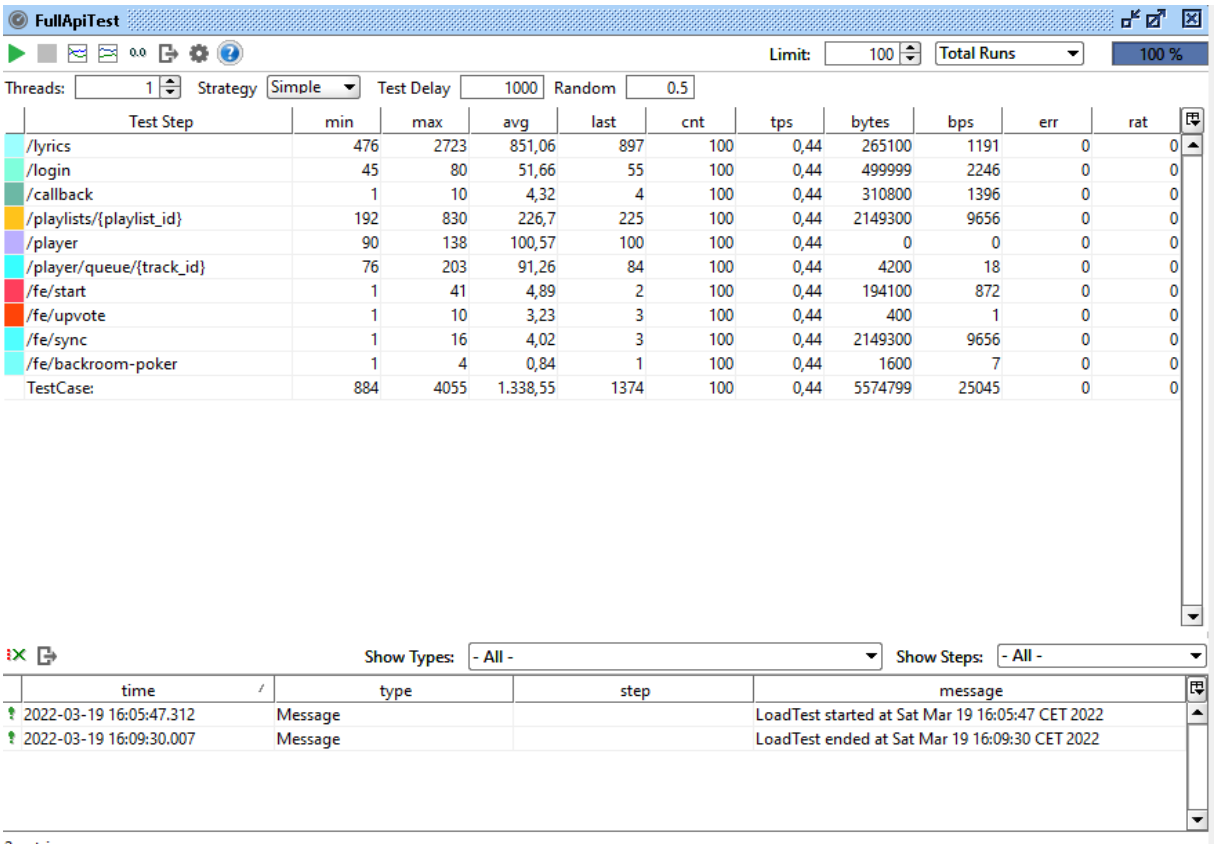
Der Endpunkt / liefert die index.html Datei mit dem Grundgerüst des Frontends aus.

Der Endpunkt */fe/start* liefert die Datei start.html mit dem benötigten Content der Startseite aus.
Der Endpunkt */fe/backroom-poker* prüft, ob bereits ein DJ eingeloggt ist, wenn nicht wird die URL zur Spotify-Anmeldeseite geliefert, wenn doch wird geprüft, ob Request autorisiert ist und ggf. die Datei backrrom.html geliefert, die den Content der Verwaltungsseite für den DJ enthält.
Der Endpunkt */fe/sync* prüft anhand eines mitgesendeten Parameters ob die Playlist neu synchronisiert werden soll und liefert danach die Playlist aus.
Der Endpunkt */fe/upvote* ist als Post-Request realisiert und zählt die Votes des Songs mit der ID des mitgelieferten Parameters um eins nach oben. Dies wird zusätzlich per Socket-Push an alle Clients verteilt.
Abschließend folgt noch die Übersicht über die verwendeten Datentypen:



Der Datentyp *Device* wird in der aktuellen Version nicht verwendet.

5. Performance-Tests mit SOAP-UI



Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
/lyrics	476	2723	851,06	897	100	0,44	265100	1191	0	0
/login	45	80	51,66	55	100	0,44	499999	2246	0	0
/callback	1	10	4,32	4	100	0,44	310800	1396	0	0
/playlists/{playlist_id}	192	830	226,7	225	100	0,44	2149300	9656	0	0
/player	90	138	100,57	100	100	0,44	0	0	0	0
/player/queue/{track_id}	76	203	91,26	84	100	0,44	4200	18	0	0
/fe/start	1	41	4,89	2	100	0,44	194100	872	0	0
/fe/upvote	1	10	3,23	3	100	0,44	400	1	0	0
/fe/sync	1	16	4,02	3	100	0,44	2149300	9656	0	0
/fe/backroom-poker	1	4	0,84	1	100	0,44	1600	7	0	0
TestCase:	884	4055	1.338,55	1374	100	0,44	5574799	25045	0	0

time	type	step	message
2022-03-19 16:05:47.312	Message		LoadTest started at Sat Mar 19 16:05:47 CET 2022
2022-03-19 16:09:30.007	Message		LoadTest ended at Sat Mar 19 16:09:30 CET 2022

Die obige Tabelle zeigt die Performance-Tests unseres Web-Services, in dem alle Endpoints, die der Web-Service bietet und die genutzt werden, getestet wurden. Die Tabelle stammt von der Anwendung Soap UI, mit der die Tests durchgeführt wurden.

Jeder Endpoint wurde 100-mal getestet, für die Performance sind besonders die Spalten „min“, „max“ und „avg“ von Bedeutung.

Insgesamt ist der Web-Service gut und schnell erreichbar, lediglich der /lyrics-Endpoint hat mit durchschnittlich 850 ms eine hohe Antwortzeit. Das liegt daran, dass dort nach Titel und Interpret eines Liedes gesucht wird und die gesamten Lyrics eines Liedes gesucht und zurückgesendet werden. Für diesen Vorgang wird der externe Web-Service Genius aufgerufen, der womöglich längere Antwortzeiten hat als die Spotify API und Endpoints, die keine externen Web-Services verwenden.

Von den Endpoints, die die Spotify API verwenden, ist /playlist/{playlist_id} am langsamsten, da dort aktuelle Playlist mit Lieddaten gesucht und zurückgegeben werden muss.

Die vier Endpoints /fe und /callback sind deutlich am schnellsten, da dort wenige Daten geladen werden und wenige externe Endpoints aufgerufen werden.