Hochschule Rosenheim

System Design

PlayLiMana

Version: *1.0*
Datum: 27.03.2025
Autoren: *Nico Geis*
Andreas Aigner
Nikolas Lorenz
Goran Matkovic

Änderungsverzeichnis:

Version	Datum	Autor(en)	Änderung
0.1	27.03.25	Nico Geis, Andreas Aigner	Genereller Grundentwurf
0.2	30.03.25	Nico Geis, Andreas Aigner, Goran Matkovic, Nikolas Lorenz	Überblick erweitert, Aufgabenverteilung
0.3	03.04.25	Nico Geis, Andreas Aigner, Goran Matkovic, Nikolas Lorenz	Ideen ausformuliert, Diagramme
1.0	10.04.25	Nico Geis, Andreas Aigner, Goran Matkovic, Nikolas Lorenz	Zusammenführen bzw. Einfügen der Use Cases und Diagramme, Meilensteine
1.1	13.04.25	Nico Geis	Kosmetische Überarbeitungen

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel dieses Dokuments

In diesem System Design soll unsere Projektidee genauer beschrieben werden. Dabei beschreiben wir das Ziel unserer Anwendung, unter anderem dargestellt an einer Auswahl an ausformulierten Use-Cases, und geben mit Hilfe von verschiedenen Diagrammen einen groben Einblick in den technischen und logischen Aufbau der Anwendung. Die uns selbst gesetzten Meilensteine, eine grobe erste Aufgabeneinteilung sowie die Links zu den APIs, deren Verwendung wir in Betracht ziehen, befinden sich am Ende des Dokuments.

1.2 Referenzierte Dokumente

Soundiiz ist eine Anwendung, die eine sehr ähnliche Funktionalität bietet. Viele Anwendungsfälle wie beispielsweise das Übertragen von Playlists über mehrere Musikdienste hinweg überschneiden sich, aber wir werden manche Aspekte weglassen und neue Funktionen einbauen, die es auf *Soundiiz* eventuell noch nicht gibt. (https://soundiiz.com/de/)

2 Überblick

PlayLiMana ist eine Anwendung, die Benutzern ein zentralisiertes Management ihrer Playlists über verschiedene Streamingdienste hinweg ermöglicht. Die Applikation bietet die Möglichkeit, Nutzerkonten zu erstellen und mit Streamingdiensten wie beispielsweise Spotify und Deezer zu verknüpfen, um neue Playlists zu erstellen, bestehende Playlists zu importieren, zu bearbeiten und auf verschiedene Dienste zu übertragen. Es soll auch eine Version ohne Login geben, bei der der Nutzer öffentliche und geteilte Playlists finden kann.

Neben diesen Grundfunktionen haben wir noch Ideen für weitere Features:

- Teilen von Playlists (z.B. über QR-Codes)
- Gemeinsames Bearbeiten von Playlists
- „Mag-ich“-Funktion
- Hör-Verlauf
- Statistiken über das eigene Hörverhalten
- Vorgeschlagene Titel und Playlists
- Automatisch generierte Playlists
- Verbesserte Suchfunktionen mit Filtern

Welche dieser Funktionen tatsachlich umgesetzt werden, legen wir im Laufe des Projekts fest.

3 Funktionales Modell

Im Folgenden wird eine Auswahl von Use-Cases tabellarisch abgebildet. Dargestellt werden die typischen Interaktionen, die der Nutzer mit unserem System durchführen wird.

PRIMÄR	Mediathek durchsuchen	**UC ID**	1.0
		Version	1.0
Ziel im Kontext	Gespeicherte Playlists anzeigen		
Akteure			
Primär	Eingeloggter Nutzer		
Sekundär	Controller, der die Anfragen bearbeitet; Datenbank		
Vorbedingungen	Nutzer muss eingeloggt sein		
Auslösendes Ereignis	Aufrufen der Mediathek-Seite		
Ablaufschritte			
Standard	1. Nutzer sendet Anfrage an Controller 2. Controller ruft alle gespeicherten Playlists aus der Datenbank ab 3. Controller liefert der Benutzeroberfläche die Metadaten (*200 OK*) 4. Nutzer sieht alle seine Playlists auf der Benutzeroberfläche		
Alternativ	- Keine gespeicherten Playlists: wie Standard, aber die Mediathek ist leer (*200 OK*) - Nicht eingeloggt: Weiterleitung zur Login-Seite (*401 Unauthorized* oder *302 Found*)		
Nachbedingungen			
bei Erfolg	Nutzer befindet sich in der Mediathek		
bei Misserfolg	Nutzer muss sich anmelden		
Referenzen	/		
Bearbeitungskommentar	Zu klären: Ist Mediathek Bestandteil der Hauptseite oder separat?		

PRIMÄR	Lokale Playlist erstellen	**UC ID**	2.0
		Version	1.0
Ziel im Kontext	Playlist erstellen, ohne sie automatisch auf Musikdienste zu exportieren		
Akteure			
Primär	(Eingeloggter?) Nutzer		
Sekundär	Controller, der die Anfragen bearbeitet; Datenbank		
Vorbedingungen	/		
Auslösendes Ereignis	Nutzer wählt Option „Playlist erstellen“ aus und setzt diese auf „lokal“		
Ablaufschritte			
Standard	1. Nutzer gibt gültigen Namen und Sichtbarkeit für Playlist; bestätigt 2. Controller leitet Daten an Datenbank weiter 3. Benutzeroberfläche zeigt leere Playlist an		
Alternativ	Playlist kann nicht erstellt oder gespeichert werden		
Nachbedingungen			
bei Erfolg	Leere Playlist erstellt, Nutzer kann nun Titel hinzufügen		
bei Misserfolg	Playlist wird nicht erstellt, Fehlermeldung		
Referenzen	/		
Bearbeitungskommentar	Zu klären: Können auch nicht eingeloggte Nutzer Playlists erstellen? Nice-to-have: Titel direkt hinzufügen bzw. der Nutzer wird direkt auf die Titel-Such-Seite weitergeleitet o. Ä.		

PRIMÄR	Synchronisierte Playlist erstellen	**UC ID**	2.1
		Version	1.0
Ziel im Kontext	Playlist erstellen, und automatisch auf Musikdienste exportieren		
Akteure			
Primär	Eingeloggter Nutzer		
Sekundär	Controller, der die Anfragen bearbeitet; Datenbank		
Vorbedingungen	- Nutzer muss eingeloggt sein - Nutzer muss einen Streamingdienst-Account verknüpft haben		
Auslösendes Ereignis	Nutzer wählt Option „Playlist erstellen“ aus und stellt sie auf „synchron“		
Ablaufschritte			
Standard	1. Nutzer gibt gültigen Namen und Sichtbarkeit für Playlist; bestätigt 2. Nutzer wählt die Streamingdienste aus, auf die die Playlist exportiert werden soll 3. Controller prüft Status des Streamingdienst-Accounts 4. Controller leitet die Anfrage an die externe API weiter 5. Externe API erstellt leere Playlist auf dem Streamingdienst 6. Controller leitet Daten an Datenbank weiter 7. Benutzeroberfläche zeigt leere Playlist an		
Alternativ	- Playlist kann nicht auf allen ausgewählten Streamingdiensten erstellt werden - Playlist kann nur lokal erstellt werden - Playlist kann nicht erstellt oder gespeichert werden		
Nachbedingungen			
bei Erfolg	- Leere Playlist erstellt, Nutzer kann nun Titel hinzufügen - Playlist auf der Seite der Streamingdienste sichtbar		
bei Misserfolg	Playlist wird nicht oder nicht überall erstellt, Fehlermeldung		
Referenzen	/		
Bearbeitungskommentar	Nice-to-have: - Der Nutzer kann bei Ablaufsschritt 2 direkt sein Konto mit einem Streamingdienst-Account verknüpfen - Titel direkt hinzufügen bzw. der Nutzer wird direkt auf die Titel-Such-Seite weitergeleitet o. Ä.		

PRIMÄR		Playlists importieren	**UC ID**	2.2
			Version	1.0
Ziel im Kontext		Ausgewählte Playlists aus Streamingdienst in lokale Mediathek kopieren		
Akteure				
	Primär	Eingeloggter Nutzer		
	Sekundär	Controller, der die Anfragen bearbeitet; Datenbank		
Vorbedingungen		- Nutzer muss eingeloggt sein - Nutzer muss einen Streamingdienst-Account verknüpft haben		
Auslösendes Ereignis		Nutzer wählt Option „Playlist importieren“		
Ablaufschritte				
	Standard	1. Nutzer wählt Streamingdienst aus 2. Controller prüft Status des Streamingdienst-Accounts 3. Controller leitet Anfrage an externe API weiter 4. Externe API fragt Liste von allen Playlists ab 5. Nutzer wählt alle zu importierenden Playlists aus 6. Nutzer wählt aus, ob Playlists synchron bleiben sollen 7. Controller leitet die Daten an die Datenbank weiter 8. Die Datenbank speichert die Playlisten ab (erstellt lokale Kopien) 9. Die Benutzeroberfläche zeigt die Mediathek an		
	Alternativ	- Playlists können nicht aus allen oder aus keinen der ausgewählten Streamingdiensten importiert werden		
Nachbedingungen				
	bei Erfolg	- Lokale Kopien der Playlists erstellt - Playlist auf der Seite der Streamingdienste und in Mediathek sichtbar		
	bei Misserfolg	Playlists nicht importiert, Fehlermeldung		
Referenzen		/		
Bearbeitungskommentar		Nice-to-have: - Der Nutzer kann bei Ablaufsschritt 2 direkt sein Konto mit einem Streamingdienst-Account verknüpfen		

PRIMÄR		Titel auf Deezer suchen	**UC ID**	3.0
			Version	1.0
Ziel im Kontext		Songs auf Deezer finden		
Akteure				
	Primär	Eingeloggter Nutzer		
	Sekundär	Deezer API, Such-Controller		
Vorbedingungen		Nutzer ist eingeloggt, Internetverbindung		
Auslösendes Ereignis		Nutzer gibt Suchbegriff ein und klickt auf "Suchen"		
Ablaufschritte				
	Standard	1. Nutzer gibt Suchbegriff ein 2. Controller sendet Anfrage an Deezer API 3. API liefert Ergebnisse 4. Ergebnisse werden dem Nutzer angezeigt		
	Alternativ	- Keine Treffer: "Keine Ergebnisse" - API-Fehler: Fehlermeldung anzeigen		
Nachbedingungen				
	bei Erfolg	Nutzer sieht Suchergebnisse		
	bei Misserfolg	Fehlermeldung sichtbar		
Referenzen		Deezer API		
Bearbeitungskommentar		/		

PRIMÄR		Titel von Deezer abspielen	**UC ID**	4.0
			Version	1.0
Ziel im Kontext		Musik über Deezer wiedergeben		
Akteure				
	Primär	Eingeloggter Nutzer		
	Sekundär	Deezer API, Player-Controller		
Vorbedingungen		Song ist über API verfügbar		
Auslösendes Ereignis		Nutzer klickt auf "Abspielen"		
Ablaufschritte				
	Standard	1. Nutzer klickt auf Song 2. Anfrage an Controller 3. Controller fragt Deezer API für Stream-URL ab 4. Stream-URL wird an Player weitergegeben 5. Player beginnt Wiedergabe		
	Alternativ	- API-Fehler: Fehlerhinweis - Song nicht verfügbar: "Nicht abspielbar" anzeigen		
Nachbedingungen				
	bei Erfolg	Song wird abgespielt		
	bei Misserfolg	Fehlermeldung sichtbar		
Referenzen		Deezer API		
Bearbeitungskommentar		/		

PRIMÄR	Login mit Drittanbieter oder PlayLiMana	**UC ID**	5.0
		Version	1.0
Ziel im Kontext	Nutzer authentifizieren und Zugriff auf personalisierte Funktionen gewähren		
Akteure			
Primär	Nutzer		
Sekundär	Authentifizierungs-Controller, OAuth-Anbieter (Spotify, Deezer), Datenbank		
Vorbedingungen	Internetverbindung, gültiger Account bei einem der Anbieter		
Auslösendes Ereignis	Nutzer klickt auf "Login" auf der Startseite oder greift auf geschützte Funktion zu		
Ablaufschritte			
Standard	**Variante A: PlayLiMana-Login** 1. Nutzer klickt auf "Mit PlayLiMana einloggen" 2. Eingabe von Benutzername und Passwort 3. Controller überprüft Login-Daten in der Datenbank 4. Session oder Token wird generiert 5. Nutzer wird zu seinen Playlists weitergeleitet **Variante B: Spotify Login** 1. Nutzer klickt auf "Mit Spotify einloggen" 2. Weiterleitung zur Spotify-OAuth-Seite 3. Nutzer autorisiert Zugriff 4. PlayLiMana erhält Auth-Code, tauscht ihn gegen Token 5. Token wird gespeichert, Session beginnt **Variante C: Deezer-Login** 1. Nutzer klickt auf "Mit Deezer einloggen" 2. OAuth-Flow wie bei Spotify 3. Zugriffstoken wird gespeichert und Session gestartet		
Alternativ	- Falsche PlayLiMana-Daten: Fehlermeldung "Benutzername oder Passwort falsch" - Drittanbieter verweigert Zugriff: Fehlermeldung "Autorisierung fehlgeschlagen" - Kein Internet: Login nicht möglich		
Nachbedingungen			
bei Erfolg	Nutzer ist eingeloggt, Zugang zu persönlicher Mediathek & Playlists		
bei Misserfolg	Nutzer bleibt ausgeloggt, Hinweis erscheint		

Referenzen	Spotify OAuth, SoundCloud OAuth, PlayLiMana Auth-System
Bearbeitungskommentar	Token-Speicherung absichern, Logout-Logik und Token-Refresh beachten

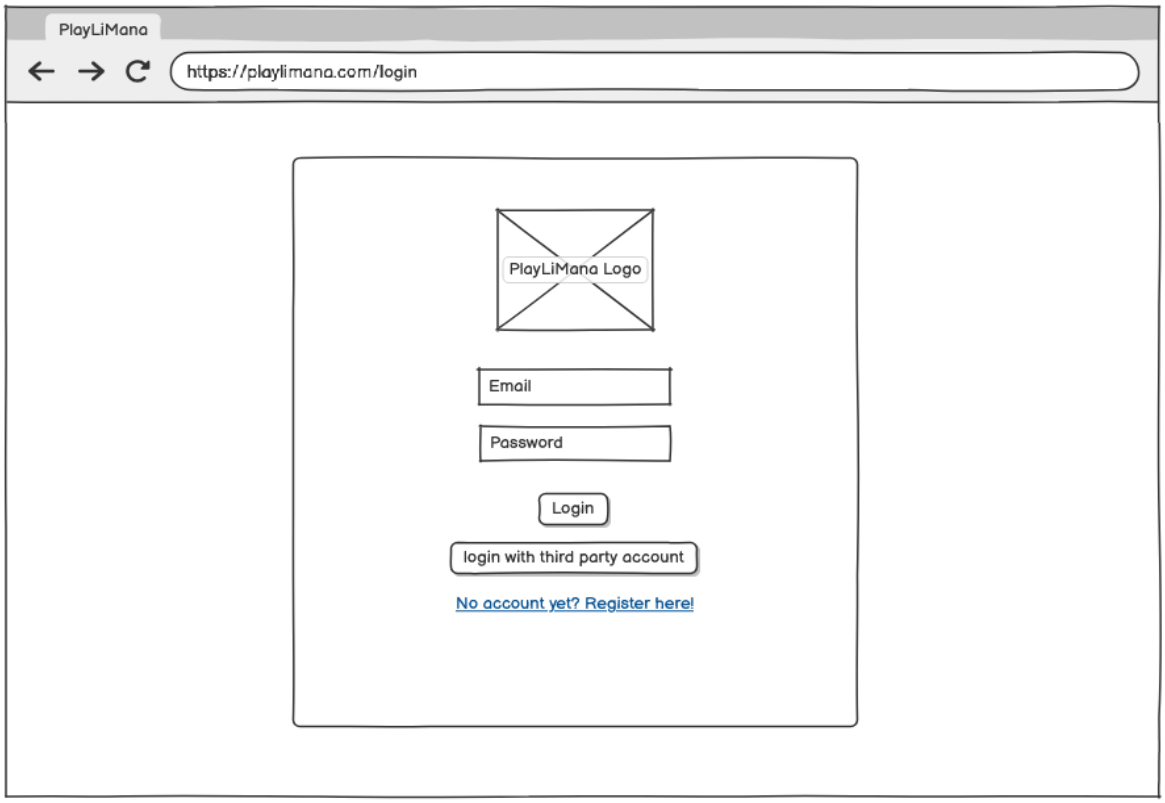
PRIMÄR	User will Anbieter von seinem Konto trennen	**UC ID**	5.1
		Version	1.0
Ziel im Kontext	User hat die Möglichkeit, einen Streamingdienst-Anbieter von seinem PlayLiMana-Konto zu trennen		
Akteure			
Primär	Eingeloggter Nutzer		
Sekundär	Controller, der die Anfragen bearbeitet		
Vorbedingungen	User muss mit unserem PlayLiMana-Konto eingeloggt sein		
Auslösendes Ereignis	User Infos von dem Anbieter (Name, E-Mail, …) werden in dem Frontend nicht mehr angezeigt, API-Token wird aus der DB gelöscht		
Ablaufschritte			
Standard	1. Nutzer drück auf „Unlink“ Knopf 2. Webseite sendet einen „Are you sure“-Pop-up 3. Service API wird aus der DB gelöscht und Verbindung mit der API wird aufgelöst		
Alternativ	- API wird aus der DB gelöscht (*200 OK*) - Nicht eingeloggt: Weiterleitung zur Login-Seite (*401 Unauthorized* oder *302 Found*)		
Nachbedingungen			
bei Erfolg	Nutzer befindet sich auf der Profile Page		
bei Misserfolg	Nutzer muss sich anmelden		
Referenzen	/		
Bearbeitungskommentar	Zu klären: Was passiert, wenn User nur einen Anbieter verknüpft hat, bekommt er einen Prompt, um sein PlayLiMana-Konto zu löschen?		

PRIMÄR	User möchte in der Suchleiste nach Anbietern filtern	**UC ID**	6.0
		Version	1.0
Ziel im Kontext	User hat die Möglichkeit, falls mehrere Streamingdienst-Anbieter mit dem PlayLiMana-Konto verknüpft sind, bei der Titel- und Playlist-Suche zu filtern, auf welchem Anbieter gesucht werden soll.		
Akteure			
Primär	Eingeloggter Nutzer		
Sekundär	Controller, der die Anfragen bearbeitet		
Vorbedingungen	- User muss mit unserem PlayLiMana-Konto eingeloggt sein - User muss mehrere APIs (Spotify, Deezer, …) mit seinem Konto verknüpft haben		
Auslösendes Ereignis	User kann beliebig aussuchen von welchen Service er die Responses bekommt		
Ablaufschritte			
Standard	1. User gibt in der Suchleiste Text ein 2. User filtert nach den Service, den er für die Suche nutzen will (Default all eingebundene)		
Alternativ	- API geben den Inhalt zurück (*200 OK*) - API ist abgelaufen und User muss sich mit dem Service neu anmelden (*401 Unauthorized* oder *302 Found*)		
Nachbedingungen			
bei Erfolg	Nutzer bekommt nur gefilterten Inhalt zurück		
bei Misserfolg	Nutzer wird an die Service Anbieter umgeleitet		
Referenzen	/		
Bearbeitungskommentar	Nutzer muss mindestens einen Filter auswählen		

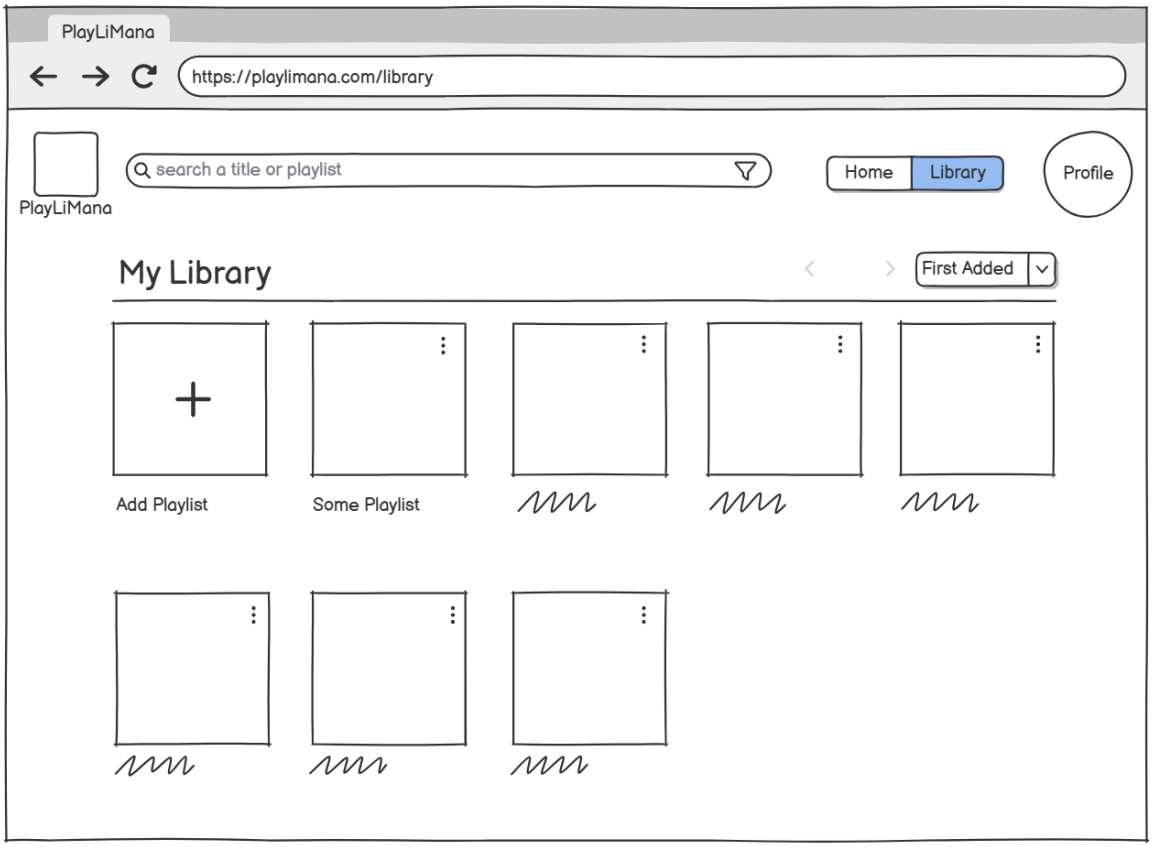
4 Dialoglandkarte

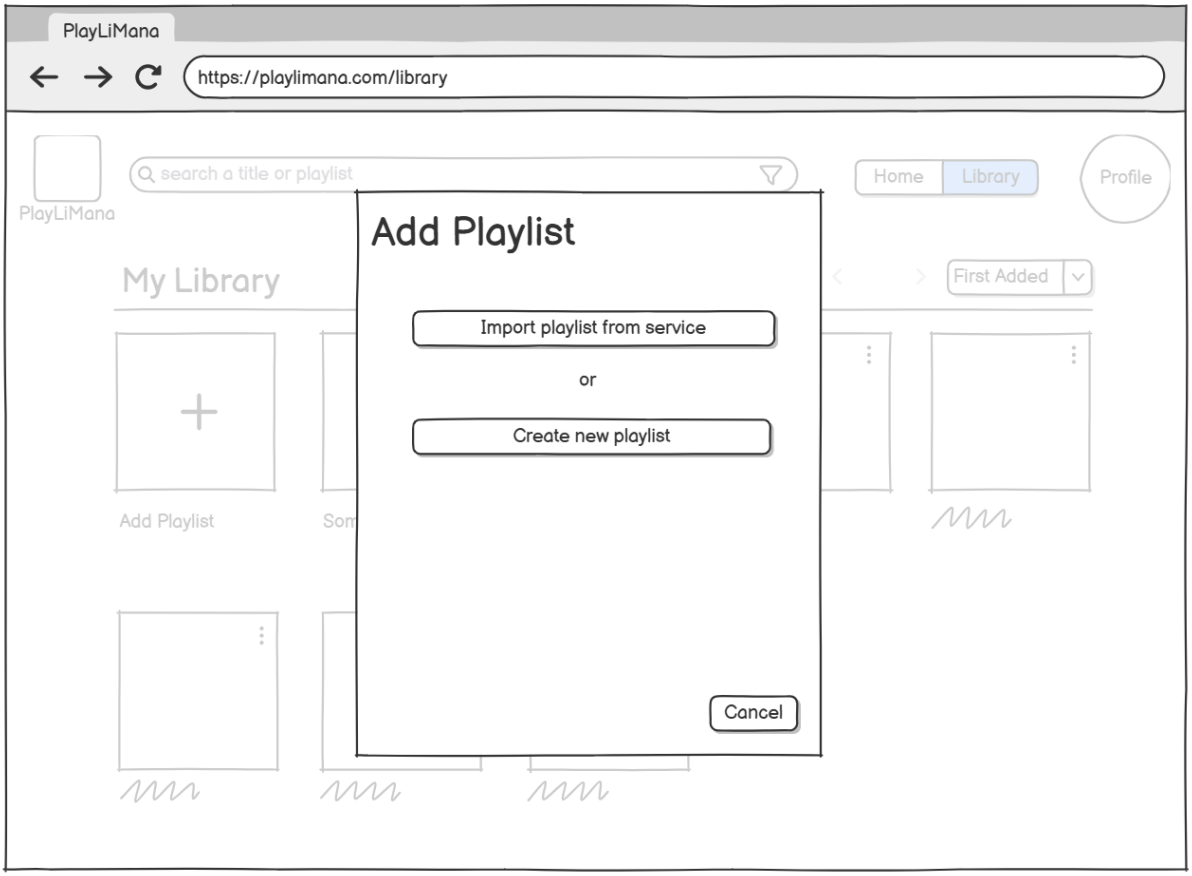
Für einen kurzen Einblick in unser Konzept folgen vier grobe Skizzen von der grafischen Oberfläche.

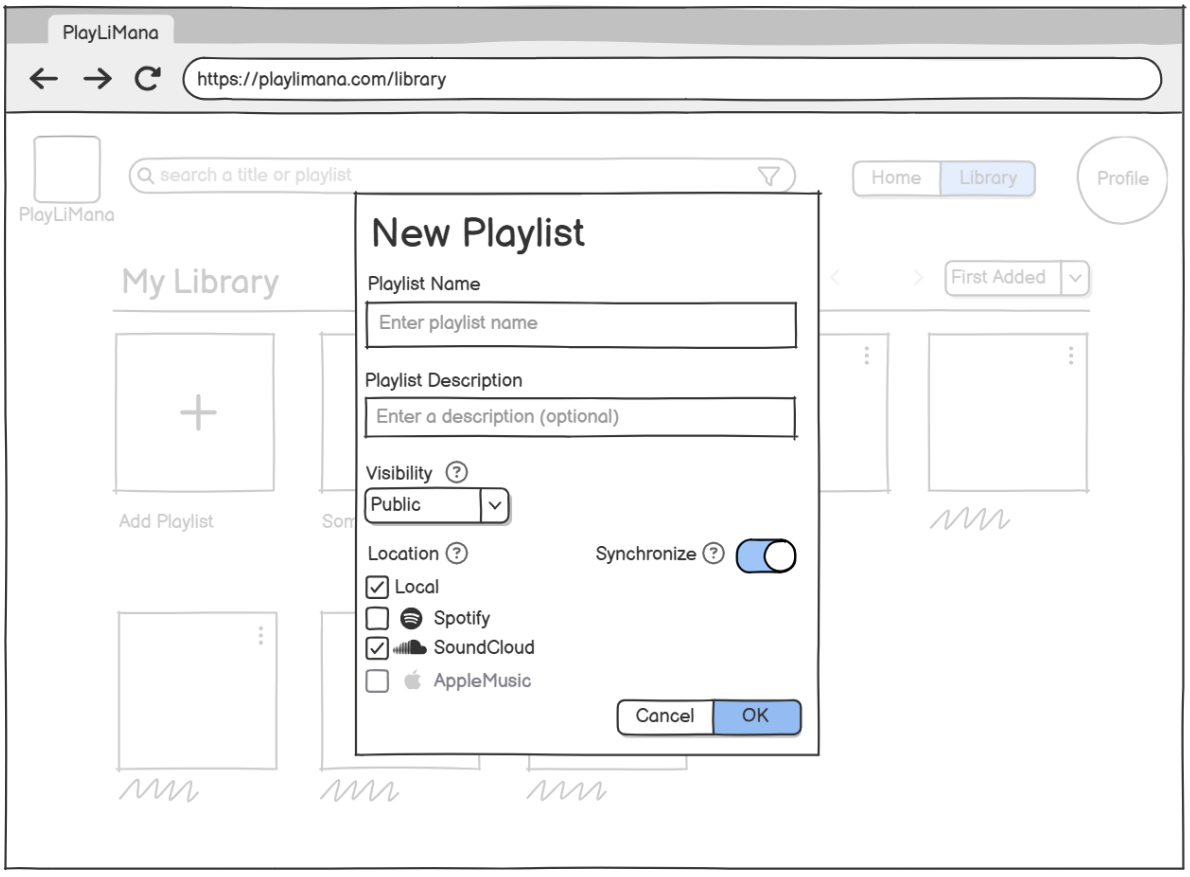
Login:



Mediathek:




PlayLiMana
https://playlimana.com/library
search a title or playlist
Home
Library
Profile
PlayLiMana
My Library
First Added
Add Playlist
Add Playlist
Import playlist from service
or
Create new playlist
Cancel

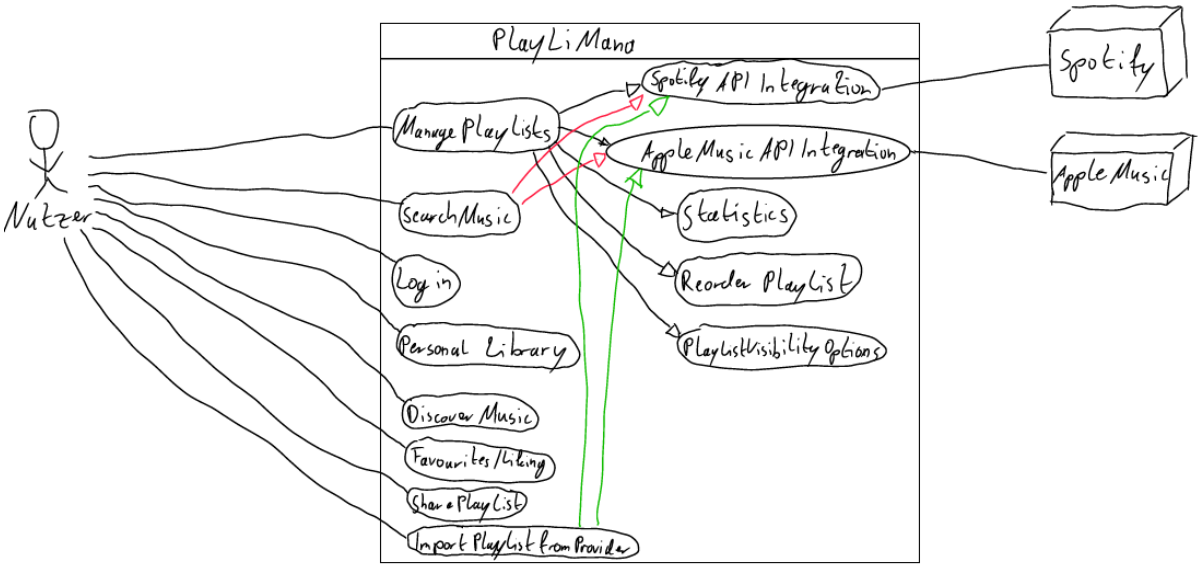

PlayLiMana
https://playlimana.com/library
search a title or playlist
Home
Library
Profile
PlayLiMana
My Library
First Added
Add Playlist
New Playlist
Playlist Name
Enter playlist name
Playlist Description
Enter a description (optional)
Visibility
Public
Location
Synchronize
Local
Spotify
SoundCloud
AppleMusic
Cancel
OK

5 IT-Architektur

Für das Backend wollen wir das *Quarkus-Framework* einsetzen und für das Frontend *Angular*.

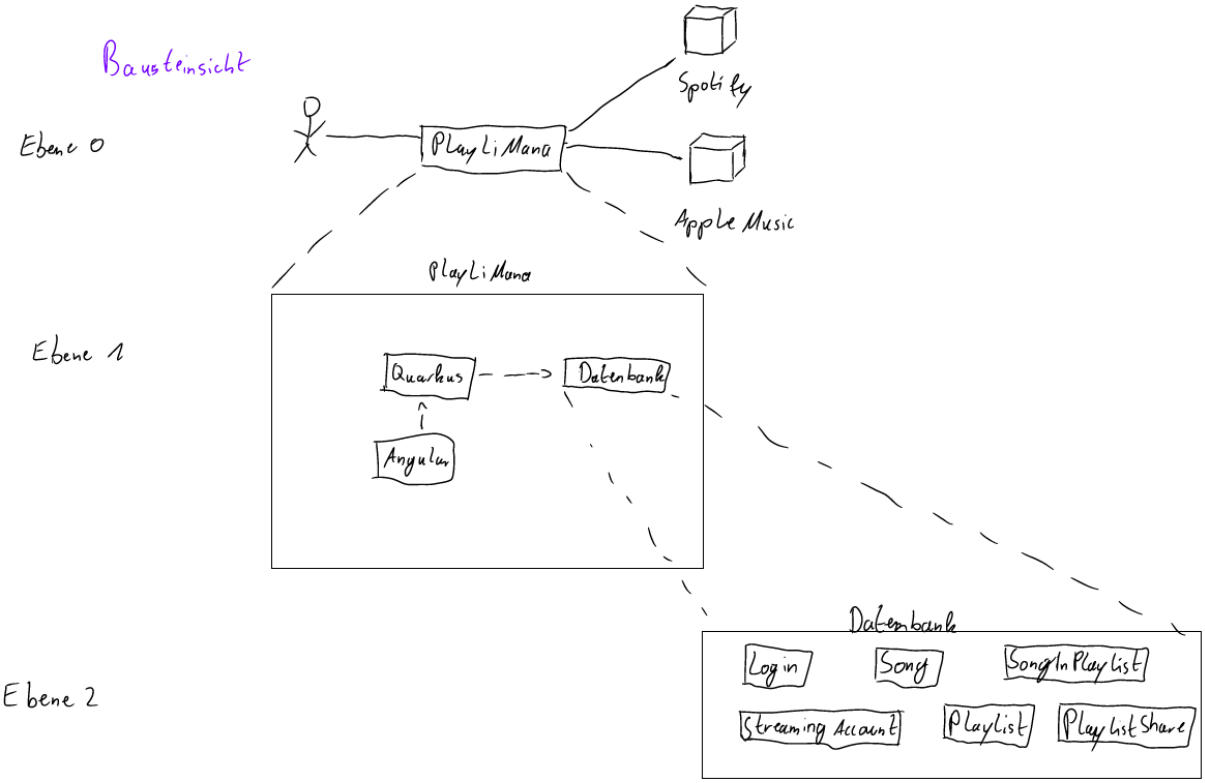
5.1 Kontextabgrenzung

Im folgenden Use-Case-Diagramm sind die verschiedenen Nutzerinteraktionen dargestellt. Es zeigt grob, welche Use-Cases aufeinander aufbauen und wo die Fremd-APIs eingebunden werden.



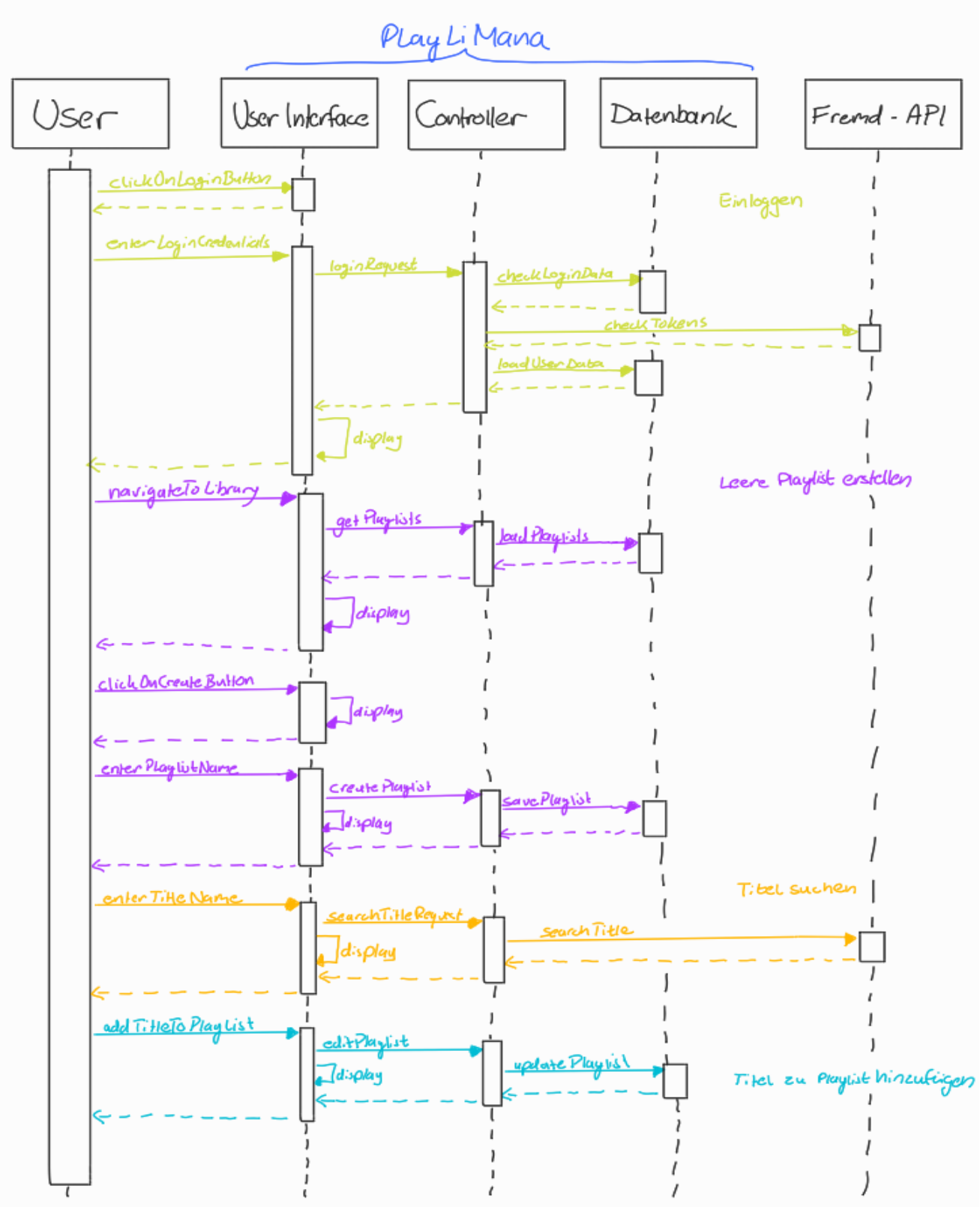
5.2 Bausteinsicht

In diesem Diagramm ist der Kontext um das System auf Ebene 0, das System selbst auf Ebene 1 und die in der Datenbank zu speichernden Entitäten auf Ebene 2 angedeutet:



5.3 Laufzeitsicht

Das folgende Sequenzdiagramm stellt den Ablauf einer typischen Nutzerinteraktion dar: Nachdem sich der Nutzer eingeloggt hat (grün), erstellt er eine neue, leere Playlist (violett), sucht einen Musiktitel (orange) und fügt diesen der erstellten Playlist zu (türkis).

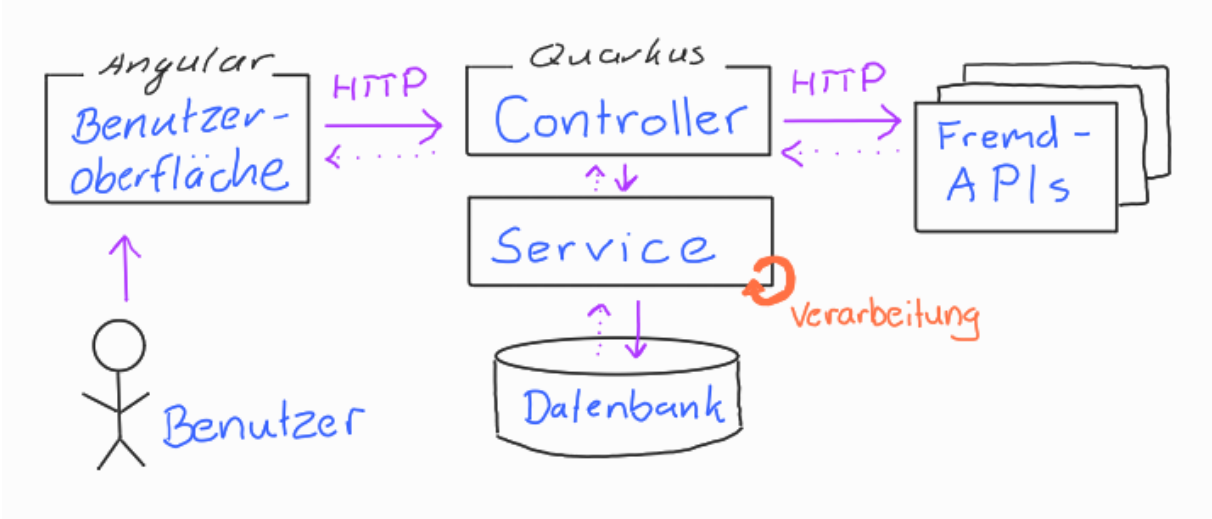


6 Schnittstellen und Integration

6.1 Schnittstellenspezifikation

Die Schnittstelle vom Nutzer zu PlayLiMana ist die Benutzeroberfläche, welche über HTTP-Methoden mit den entsprechenden Controllern kommuniziert. Dieser kann über die jeweiligen Service-Klassen auf das nötige Repository bzw. letztendlich auf die Datenbank zugreifen.

Mit den externen APIs wie beispielsweise von Spotify und Deezer kommunizieren ebenfalls entsprechende Controller- und Service-Klassen. Die Verarbeitung findet in den Service- und weiteren Hilfsklassen statt.



6.2 Schnittstellenformat

Die APIs verwenden JSON für den Datenaustausch. Kommunikation findet über die üblichen HTTP-Methoden – GET, POST, PUT, DELETE – statt und mit den verschiedenen Response Codes wie beispielsweise:

- 200 (OK)
- 201 (Created)
- 400 (Bad Request)
- 401 (Unauthorized)
- 404 (Not Found)
- 500 (Internal Server Error)

7 Datenmodell

7.1 Datenspeicherungskonzept

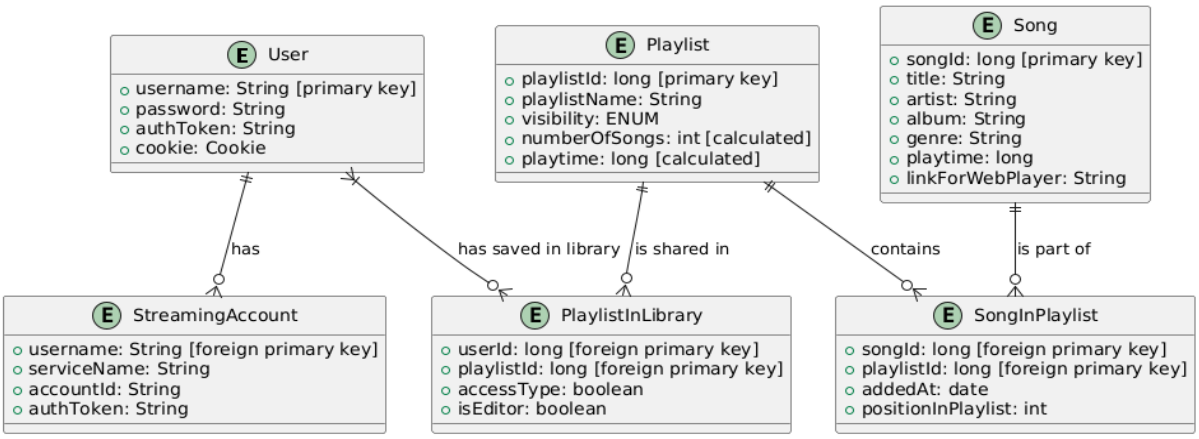
Die Datenspeicherung soll mit *PostgreSQL* erfolgen.

7.2 Physikalische Datenbankstruktur

In unserer relationalen Datenbank müssen die Nutzer und die Playlists gespeichert werden. Darüber hinaus werden für die Relationen noch weitere Tabellen benötigt, z.B. um darzustellen, welcher Nutzer welche Playlists in seiner Mediathek gespeichert hat. Um die Anzahl an API-Aufrufe möglichst gering zu halten, speichern wir auch die nötigen Metadaten der Lieder, die sich in den Playlists befinden. Logischerweise halten wir damit keinen Überblick über alle existierenden Lieder, sondern nur über jene, die sich in einer Playlist auf PlayLiMana befinden.

7.3 Datenmodell

In diesem Diagramm ist die Datenbankstruktur dargestellt. Für die komplexeren Beziehungen zwischen *User* und *Playlist* sowie *Playlist* und *Song* werden eigene Tabellen erstellt:



8 Testspezifikation

8.1 Teststrategie

Unser Hauptziel sind automatisierte Tests mit SonarQube / SonarLint und eine CI/CD-Pipeline, aber wir werden auch manuelle Unit-Tests und End-to-End-Tests durchführen, wo nötig.

8.2 Testfälle

Es folgen einige Beispiele für Testfälle:

- Testen der Benutzerauthentifizierung (Login und Registrierung)
- Testen der Funktionalitäten zur Playlistverwaltung (Erstellen, Löschen, Bearbeiten, Importieren)
- Testen der Suchfunktion für Lieder und Playlists über die verschiedenen Streamingdienst-APIs
- Testen der Mediathek und der Anzeige von Playlists
- Testen der Share-Funktion für Playlists mittels QR-Codes
- Testen der Statistiken-Funktion für Playlists und Benutzerhörverhalten

9 Projektmanagement

9.1 Projektplan

Unsere Meilensteine mit den jeweiligen Fertigstellungsdaten sind in der folgenden Tabelle aufgelistet:

Meilenstein	**Fertigstellungsdatum**	**Beschreibung**
Meilenstein 1	10.04.2025	Ideen für System Design sammeln
Meilenstein 2	10.04.2025	Datenbank und Pipeline aufsetzen
Meilenstein 3	17.04.2025	Dokument „System Design“ fertigstellen
Meilenstein 4	Mitte / Ende Mai	Minimum Viable Product (API-Anbindung, Frontend, …)
Meilenstein 5	Anfang / Mitte Juni	Nice-to-Have-Add-Ons (QR-Codes, Statistiken, …)
Meilenstein 6	26.06.2025	Fertigstellung des Source-Codes und dessen Abgabe
Meilenstein 7	03.07.2025	Endgültige Abgabe

9.2 Aufgabenverteilung

Die Aufgaben für das *Minimum Viable Product* sind aktuell wie folgt auf die Teammitglieder aufgeteilt:

APIs	
Goran	Spotify API
Nikolas	Deezer API
Andreas	wahrscheinlich Tidal API
Nico	wahrscheinlich SoundCloud API

Frontend	
Goran	Profil: - Profilansicht (eigenes Profil vs. Fremdprofil) - Einstellungen
Nikolas	Login: - Anmelden - Neu registrieren
Andreas	Startseite: - eingeloggt - nicht eingeloggt
Nico	Mediathek: - Playlistüberblick - Playlist anzeigen (eigene vs. geteilte Playlist)

Sonstiges (wird im Laufe des Projekts ergänzt)	
Goran	Suchfunktionen etc.
Nikolas	Login-Logik
Andreas	Datenbank
Nico	Playlist-Management-Logik

10 Bibliographie

Wir referenzieren hauptsächlich die Dokumentationen der verschiedenen APIs der Streaminganbieter. Dort steht jeweils beschrieben, mit welchen GET-Anfragen die User-, Playlist- und Liederdaten abgefragt werden können und wie die entsprechende Response aufgebaut ist.

- Spotify: https://developer.spotify.com/documentation/web-api und https://developer.spotify.com/documentation/web-playback-sdk
- Deezer: https://developers.deezer.com/api
- Tidal: https://tidal-music.github.io/tidal-api-reference/
- SoundCloud: https://developers.soundcloud.com/docs/api/guide
- Apple Music: https://developer.apple.com/documentation/applemusicapi/

Wir werden wahrscheinlich nicht alle genannten APIs verwenden, da wir uns zunächst auf Spotify und Deezer fokussieren möchten. Andererseits kommen im Laufe des Projekts gegebenenfalls noch weitere APIs hinzu.