Melodify

Клиент-сървър приложение за слушане на музика

СОФИЙСКИ УНИВЕРСИТЕТ

„СВ. КЛИМЕНТ ОХРИДСКИ“

ОСНОВАН 1888 Г.

Стефан Велев, фак. №: **62537**

Даниел Халачев, фак. №: **62547**

спец. **Софтуерно инженерство**, **IV** курс

Факултет по математика и информатика

Съдържание

1. Въведение

Познавате ли човек, който не обича да слуша музика? Ние не! Независимо дали слушаме музика в свободното си време, или я използваме за концентрация, всекидневно прибягваме до платформи за слушане на музика като Spotify или YouTube Music. Макар и много удобни, безплатните им лицензи за употреба винаги вървят с щедро количество реклами, които провалят потребителското ни изживяване. Поради тези причини, екипът ни избра именно темата **Melodify: клиент-сървър приложение за слушане на музика**, като предизвикателство, което ще ни позволи да създадем мечтаната от нас платформа, и ще ни даде опит в изграждането на клиент-сървър приложения на **Java**.

2. Проектиране

2.1. Общо описание

Приложението **Melodify** е базирано на **клиент-сървър архитектура**. Сървърът имплементира `REST API` за работа с ресурсите на платформата чрез `GET`, `PUT`, `PATCH`, `POST` и `DELETE` заявки.

Приложението съхранява информация за различни видове единици - песни, които притежават жанр и множество изпълнители; албуми, които включват тези песни; потребители, които притежават множество плейлисти, и опашка, в която се нареждат желаните към момента песни за слушане.

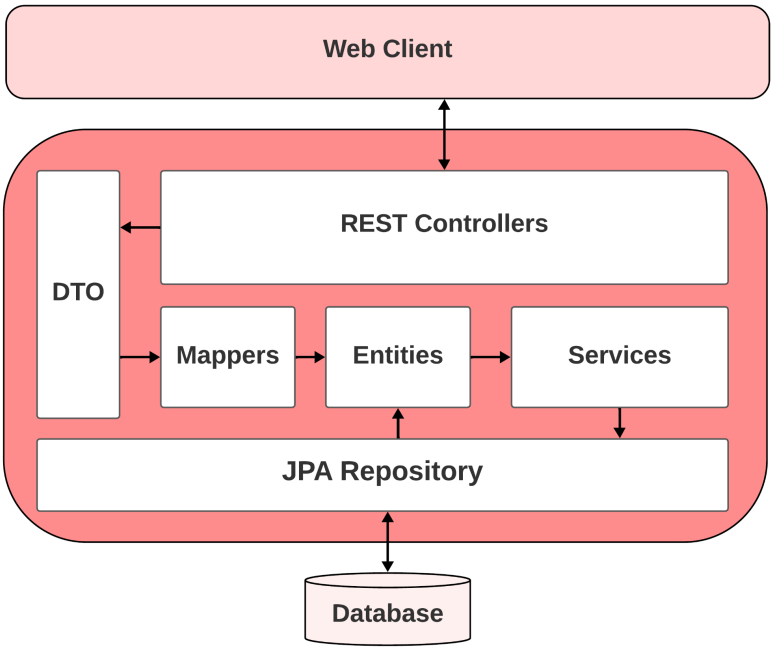
В приложението има два типа роли - на обикновени потребители и администратори. Допълнителните възможности на администраторите на системата са, че те могат да добавят и извършват операции с песни, албуми, жанрове, изпълнители, неща, които обикновените клиенти на системата не могат. За да се постигне това по отношение на оторизацията се използва т.нар. JWT токен (`JSON Web Token`), който в себе си съхранява информация за уникалния идентификатор на потребителя и това дали има администраторски права.

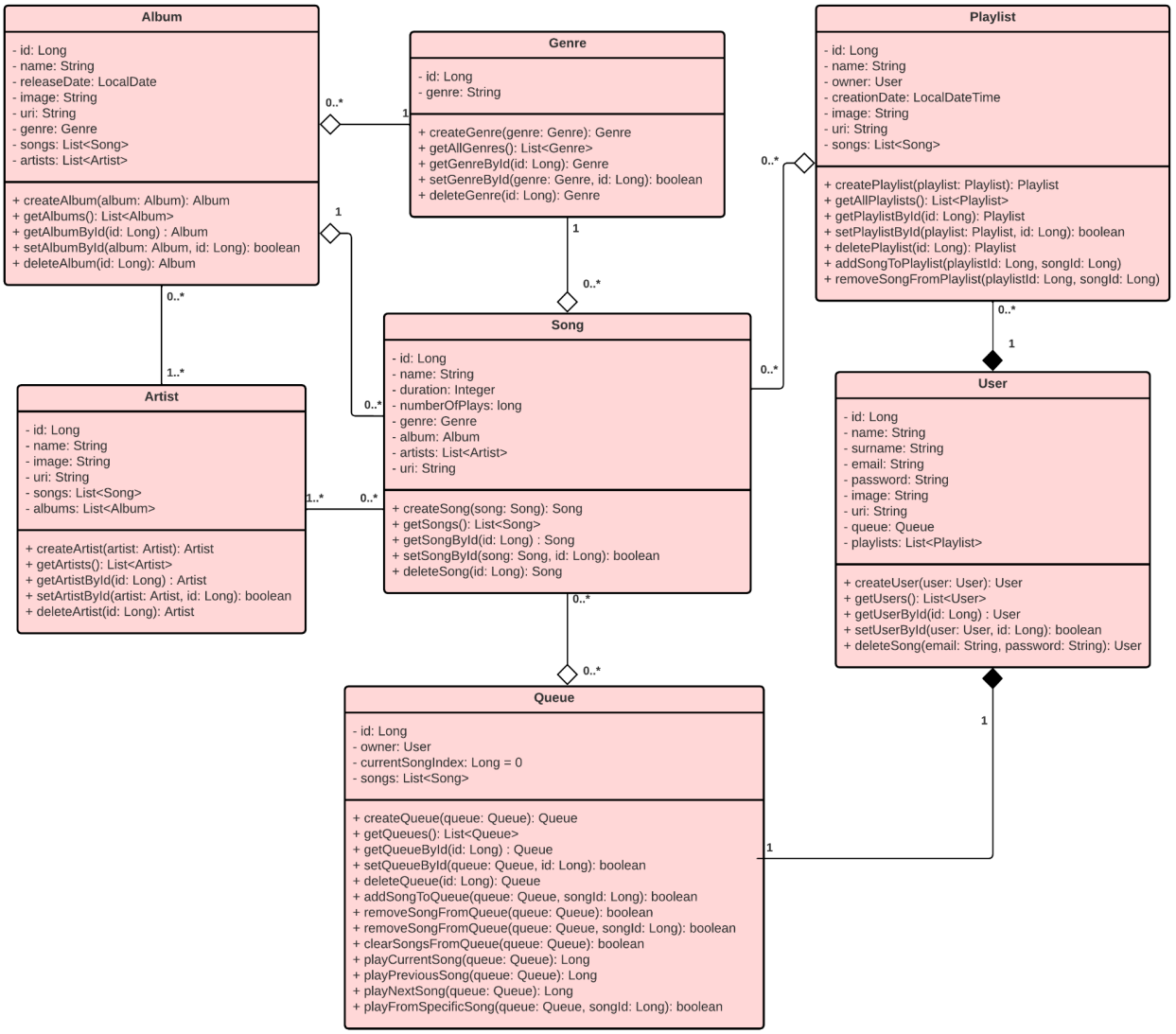
Потребителската информация се пази в `PostgreSQL` база данни. Медийните единици (като изображения и аудио файлове) се съхраняват като файлове на сървъра.

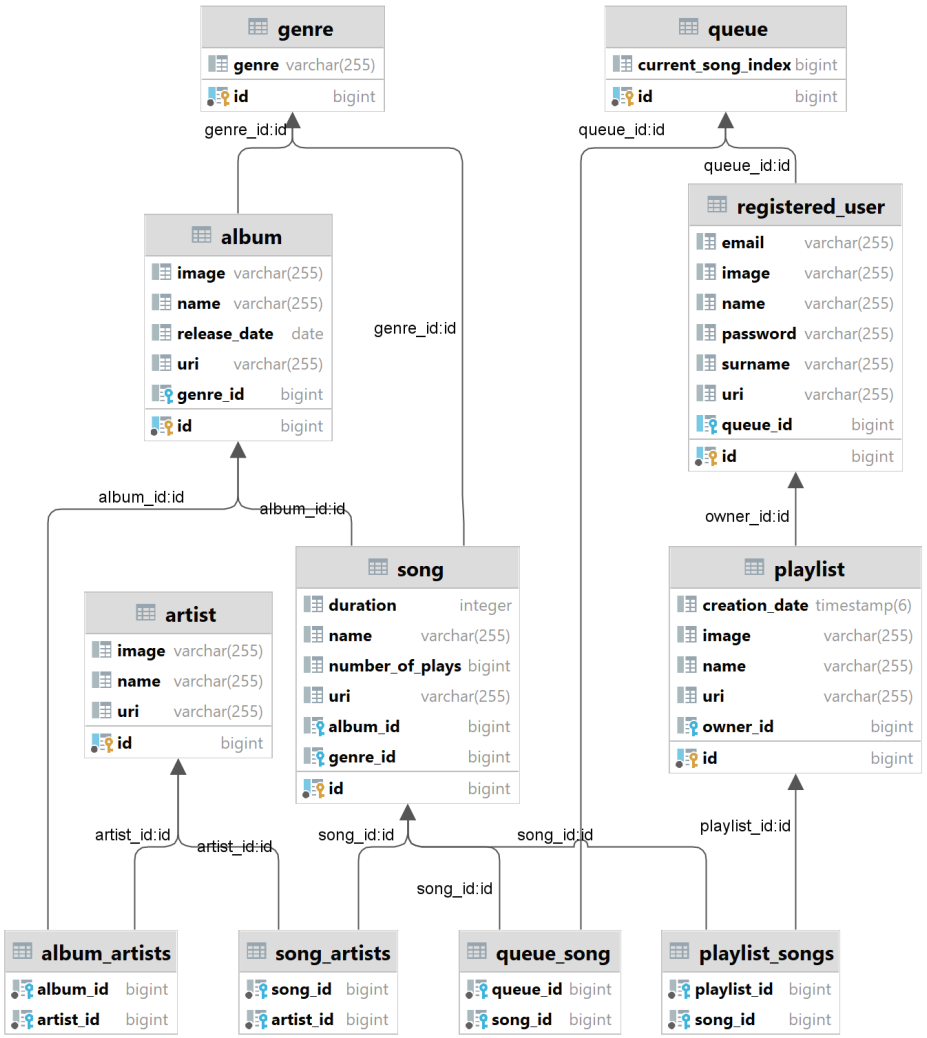
Потребителят взаимодейства със сървъра чрез **thin client**, състоящ се от страници, имплементирани с технологиите `HTML`, `CSS` и `JavaScript`, с което се постига ниско потребление на ресурси, като се избягват ненужно сложни за целта работни рамки като React и Angular.

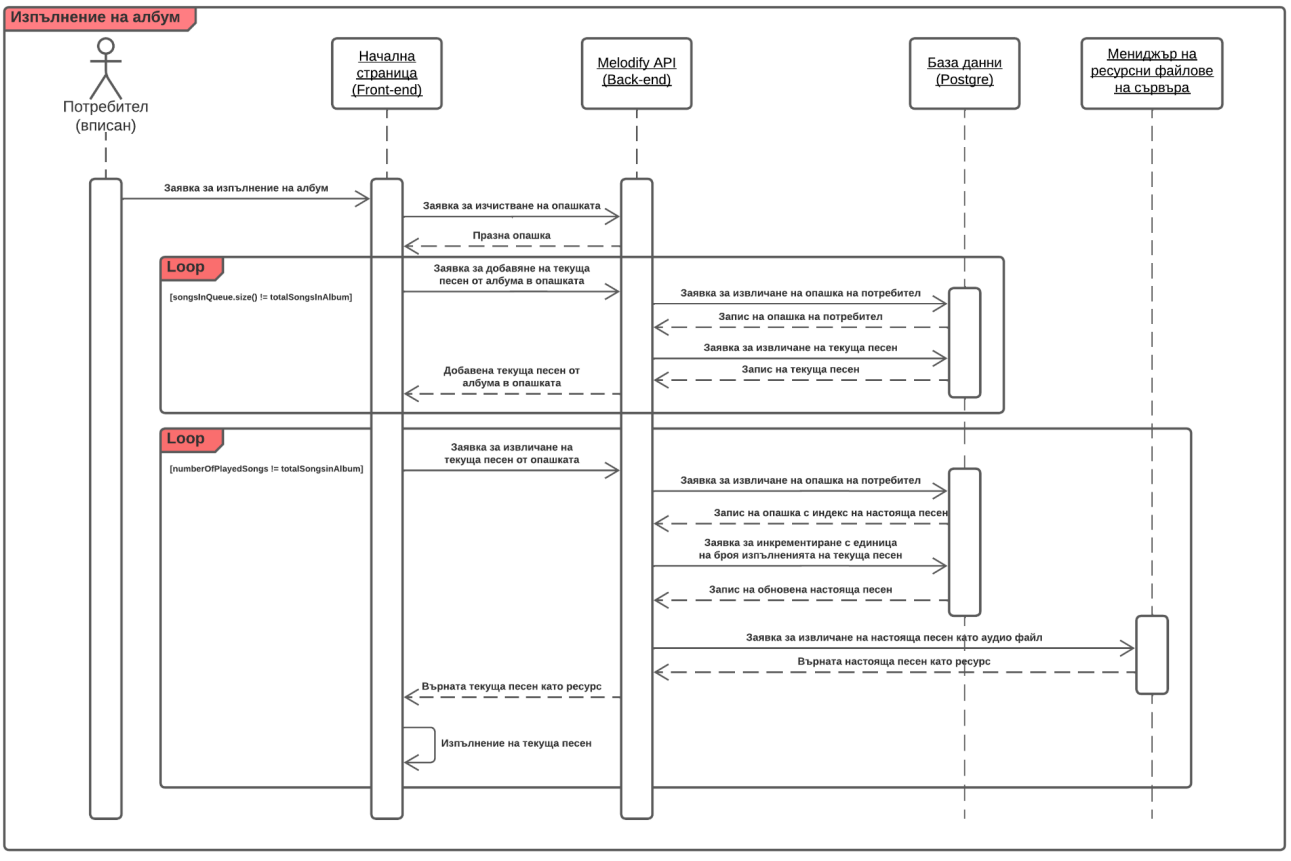
Клиентът обменя информация със сървъра чрез заявки и протокола **`HTTP`**. За работа чрез заявки може да бъде използван `Swagger Interface`, който може да бъде достъпен на `localhost:8080/swagger.html`.

2.2. Диаграми, представящи архитектурата и функционалностите на системата






genre
genre varchar(255)
id bigint
queue
current_song_index bigint
id bigint
genre_id:id
queue_id:id
queue_id:id
album
image varchar(255)
name varchar(255)
release_date date
uri varchar(255)
genre_id bigint
id bigint
genre_id:id
registered_user
email varchar(255)
image varchar(255)
name varchar(255)
password varchar(255)
surname varchar(255)
uri varchar(255)
queue_id bigint
id bigint
album_id:id
album_id:id
owner_id:id
song
duration integer
name varchar(255)
number_of_plays bigint
uri varchar(255)
album_id bigint
genre_id bigint
id bigint
playlist
creation_date timestamp(6)
image varchar(255)
name varchar(255)
uri varchar(255)
owner_id bigint
id bigint
artist
image varchar(255)
name varchar(255)
uri varchar(255)
id bigint
artist_id:id
artist_id:id
song_id:id
song_id:id
playlist_id:id
song_id:id
album_artists
album_id bigint
artist_id bigint
song_artists
song_id bigint
artist_id bigint
queue_song
queue_id bigint
song_id bigint
playlist_songs
playlist_id bigint
song_id bigint


Изпълнение на албум
Потребител (вписан)
Начална страница (Front-end)
Melodify API (Back-end)
База данни (Postgre)
Мениджър на ресурсни файлове на сървъра
Заявка за изпълнение на албум
Заявка за изчистване на опашката
Празна опашка
Loop
[songsInQueue.size() != totalSongsInAlbum]
Заявка за добавяне на текуща песен от албума в опашката
Заявка за извличане на опашка на потребител
Запис на опашка на потребител
Заявка за извличане на текуща песен
Запис на текуща песен
Добавена текуща песен от албума в опашката
Loop
[numberOfPlayedSongs != totalSongsinAlbum]
Заявка за извличане на текуща песен от опашката
Заявка за извличане на опашка на потребител
Запис на опашка с индекс на настояща песен
Заявка за инкрементиране с единица на броя изпълненията на текуща песен
Запис на обновена настояща песен
Заявка за извличане на настояща песен като аудио файл
Върната настояща песен като ресурс
Върната текуща песен като ресурс
Изпълнение на текуща песен

3. Разработване

3.1. Сървър

Сървърът е базиран на работната рамка за **Spring Application Server** - Tomcat. **Spring** е фреймуърк на **Java**, който предоставя инфраструктурата за изграждане на корпоративни приложения. Той разчита на т.нар. “инверсия на контрола”, дизайн принцип, който гласи, че не кода на приложението е отговорен за създаването и управлението на обектите, а отговорния за тази задача е т.нар. **Spring контейнер** (**IoC container**). Контейнерът създава, конфигурира и свързва обектите, дефинирани в контекста на приложението. Той също така управлява обхвата, жизнения цикъл и унищожаването на тези обекти.

Бизнес логиката се основава на:

- модели (Models)
- обекти за пренос на данни (Data Transfer Objects)
- класове за свеждане на модели до обекти за пренос на данни и обратно (Mappers)
- хранилища (Repositories)
- услуги (Services)
- контролери (Controllers)
- конфигурации за автентикация, оторизация и валидация

Основните същности в приложението са дефинирани като модели, представляващи Entity Beans. Класовете се дефинират като същности чрез Java Persistence API и неговата имплементация - Hibernate.

Така представени, същностите/моделите не могат да бъдат изпратени по мрежата. За целта, един клас се свежда до Data Transfer Object, който съхранява само необходимата за клиента информация. Обратно, информация получена от клиента, например в тялото на заявката, също се съхранява като Data Transfer Object до момента на нейното трансформиране.

Model се свежда до Data Transfer Object и обратно чрез Mapper - клас, който се генерира динамично чрез библиотеката MapStruct.

Интерфейси хранилища от тип Repository, които наследяват интерфейс JpaRepository позволяват CRUD операции с базата данни и моделите на системата, без изричното изпълнение на SQL заявки.

Услугите (Services) представляват бизнес логиката на приложението. За тяхното разработване сме следвали принципа за единствената отговорност (**Single Responsibility Principle**), т.е. една услуга да използва само един интерфейс хранилище (repository). В случаите, когато се налагат съставни операции (например при създаване на потребител се създава и плейлист за него), сме използвали шаблона за дизайн “фасада” (Facade), като създаваме нова фасадна услуга, която е съставена от повече от една други услуги. Някои от тези по-сложни операции изпълняваме като транзакции, за да може да разчитаме на техните ACID свойства в случай, че тя не се изпълни успешно.

Контролерите (`Controllers`) са специални класове, които дефинират методи, улавящи `HTTP` заявки. В нашия случай `HTTP` методите, които използваме, са `GET` (за достъпване на ресурс), `POST` (за създаване на нов ресурс), `DELETE` (за изтриване на ресурс), `PUT` (за замяна на ресурс), `PATCH` (за изменение на части от ресус).

3.2. Клиент

Графичният интерфейс използва **Single-Page Application** стратегия, която позволява да се възпроизвежда музика без прекъсване дори когато потребителят навигира в различни изгледи за албуми, изпълнители, плейлисти и търсене. Основната страница съдържа контейнер, в който се възпроизвежда съдържанието на други изгледи чрез `JavaScript Fetch API` и `AJAX`.

Всеки изглед представлява отделна `HTML` страница, която динамично зарежда данните спрямо подадения като параметър идентификатор на ресурса.

4. Внедряване

Приложението може да бъде внедрено по няколко начина. За всеки от тях се изисква на нивото на корена на проекта да бъдат дефинирани следните папки, които да съдържат медийните ресурси:

```
.
├── images
│   ├── albums
│   │   ├── 1.png
│   │   └── ...
│   ├── artists
│   │   ├── 1.png
│   │   └── ...
│   ├── playlists
│   └── users
│       ├── ...
│       └── guest-user.png
├── Melodify
│   └── Melodify.jar
└── tracks
    ├── 1.mp3
    └── ...
```

4.1. Внедряване на място

Приложението може да се изпълнява на машина, използвана за разработката, като сървър за изпълнение. За целта директно се извиква `shell` скрипта `mvnw.sh`/`mvnw.cmd` от сорс кода спрямо операционната система, или всички файлове и зависимости се компилират и изпълняват чрез `javac` и `java`.

4.2. Пакетиране в jar файл

Чрез плъгина spring-boot-maven-plugin и командата `mvn package` се създава jar файл, който съдържа необходимия `MANIFEST.MF` файл, посочващ основния клас на приложението. Резултатният `jar` файл съдържа в себе си вграден `Tomcat server` и може да бъде използван за изпълнение на приложението на различни машини, както е описано по-долу.

4.3. Хостване в облака

Приложението, разпространено като `Melodify.jar` файл, може да бъде хостнато в `Azure Web Services`, `Google Cloud` или друга платформа за cloud услуги.

4.4. Изпълнение в контейнер

Приложението, разпространено като `Melodify.jar` файл, може да бъде изпълнено в контейнер за виртуализация, например чрез `Docker` и `Kubernetes`.

4.5. Директно изпълнение

Разбира се, приложението може да бъде стартирано на локална машина, предназначена за сървър, чрез командата `java -classpath . -jar Melodify.jar [--spring.config.location=file:/…/application.properties]`. Към командата може да бъде подаден и параметър - файл `application.properties`, който променя ключовете за връзка с базата данни по подразбиране.

4.6. Стартиране като systemd услуга

Когато се хостне на машина, използваща операционната система `Linux` и демонът за инициализиране `systemd`, приложението може да бъде зададено като systemd услуга, която се инициализира автоматично при стартиране на операционната система. За целта, jar файлът се поставя в папката `var`: `/var/melodify/Melodify.jar`. След това се създава скрипт `melodify.service` в папката `/etc/systemd/system`, чието основно съдържание е следното:

```
[Unit]
Description=Melodify
After=syslog.target network.target

[Service]
User=myapp
Group=myapp

Environment="JAVA_HOME=/path/to/java/home"

ExecStart=${JAVA_HOME}/bin/java -jar /var/melodify/Melodify.jar
ExecStop=/bin/kill -15 $MAINPID
SuccessExitStatus=143
```

```
[Install]
WantedBy=multi-user.target
```

Посочват се точните имена на потребителя, групата му и пътят до изпълнимия файл за командата java. При необходимост се добавя път до конфигурационен файл.

5. Тестване

Основен предмет на тестването бяха класовете контролери, които обвързват ресурсите на REST API с бизнес логиката на приложението и гарантират, че само оторизирани потребители с правилни заявки могат да използват услугите. С цел модуларно тестване на класовете в общи и специфични сценарии, бяха използвани библиотеките за тестване JUnit и Mockito, които позволяват mocking на логиката извън целевите класове. В резултат се постига 100% преминаване на тестовете, 100% покритие на класовете, 93% тестово покритие на методите и 82% тестово покритие на редовете код в контролерите. Класовете, дефиниращи модели, обекти за трансфер на данни, хранилища и mappers не подлежат на тестване.

В допълнение, екипът извърши и тестване на `REST API` и чрез инструмента **`Postman`**. Проведени бяха и проверки на графичния интерфейс, включително на специфични сценарии - например опит пускане на предишна/следваща песен при празна опашка.

6. Интерфейси

Приложението може да се ползва чрез два вида интерфейси - REST API и графичен интерфейс.

6.1. REST API

REST интерфейсът предоставя достъп до албуми, изпълнители, плейлисти, песни и потребителски данни чрез 8 контролера и общо 54 endpoints. За кратка справка на употребата, разгледайте файла `REST API Short.html`, а за по-подробна информация, включително headers и параметри, разгледайте файла `REST API Long.html`.

6.2. Графичен интерфейс

Melodify

Register

Enter name

Enter surname

Enter email

Enter password

Register

Melodify, 2023 Login Register Terms of Service

Melodify

Login

Enter email

Enter password

Login

Melodify, 2023 Login Register Terms of Service

Melodify

Search

Home

Library +

Liked songs
Stefan

Favourites
Stefan

Welcome

Playlists Artists Albums

Your Playlists

Liked songs

Favourites

Your Artists

Shawn Mendes

Ed Sheeran

Your Albums

Illuminate
Shawn Mendes

=
Ed Sheeran

Queue

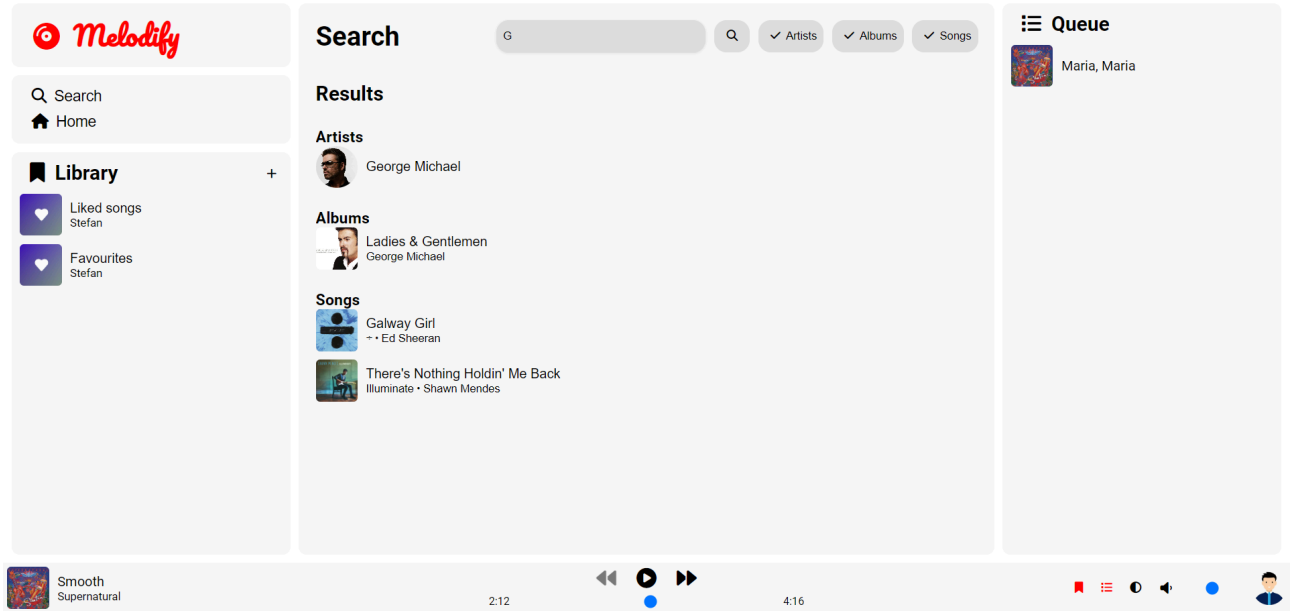
Mercy

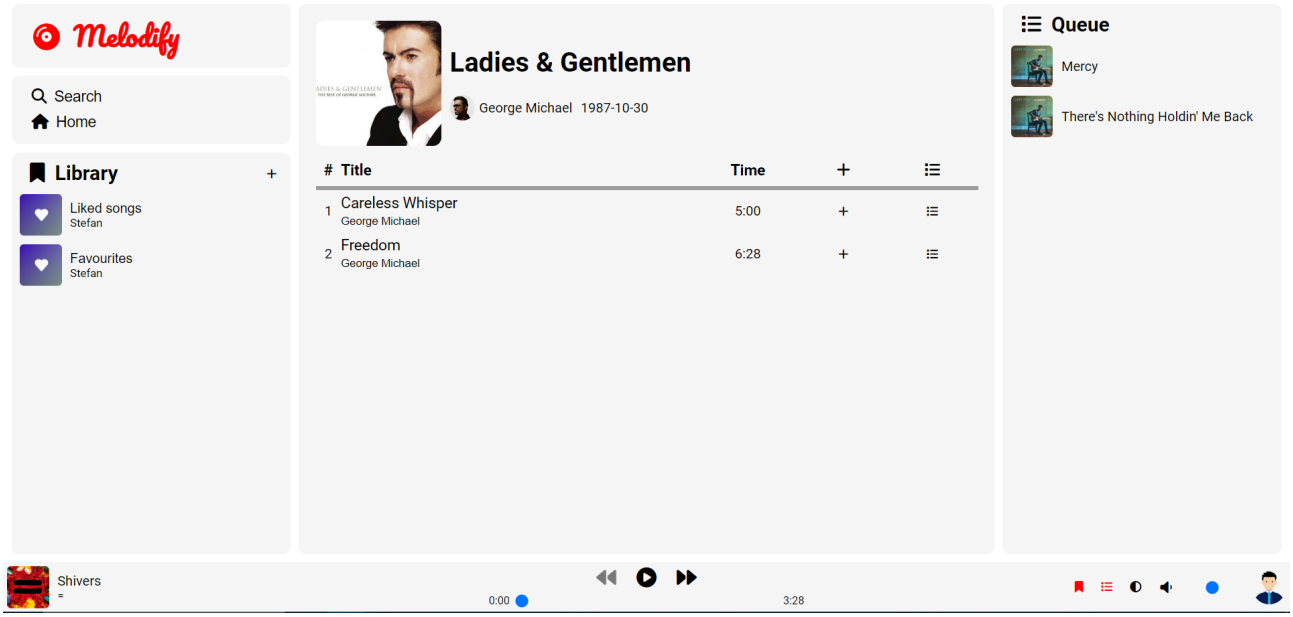
There's Nothing Holdin' Me Back

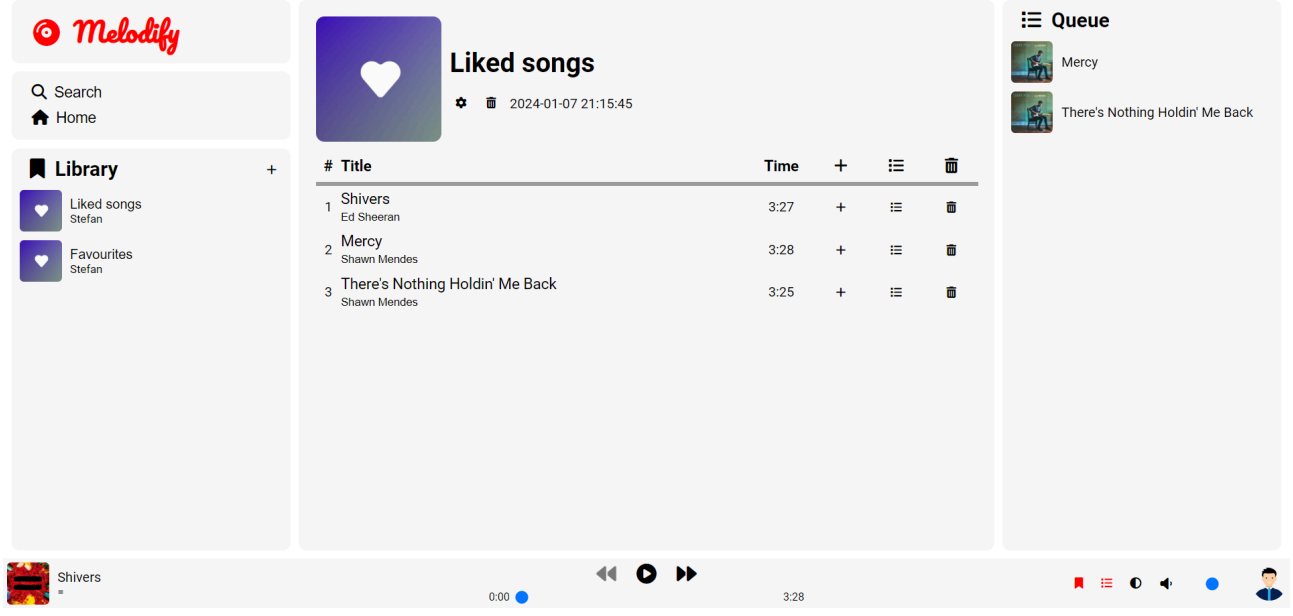
Shivers
=

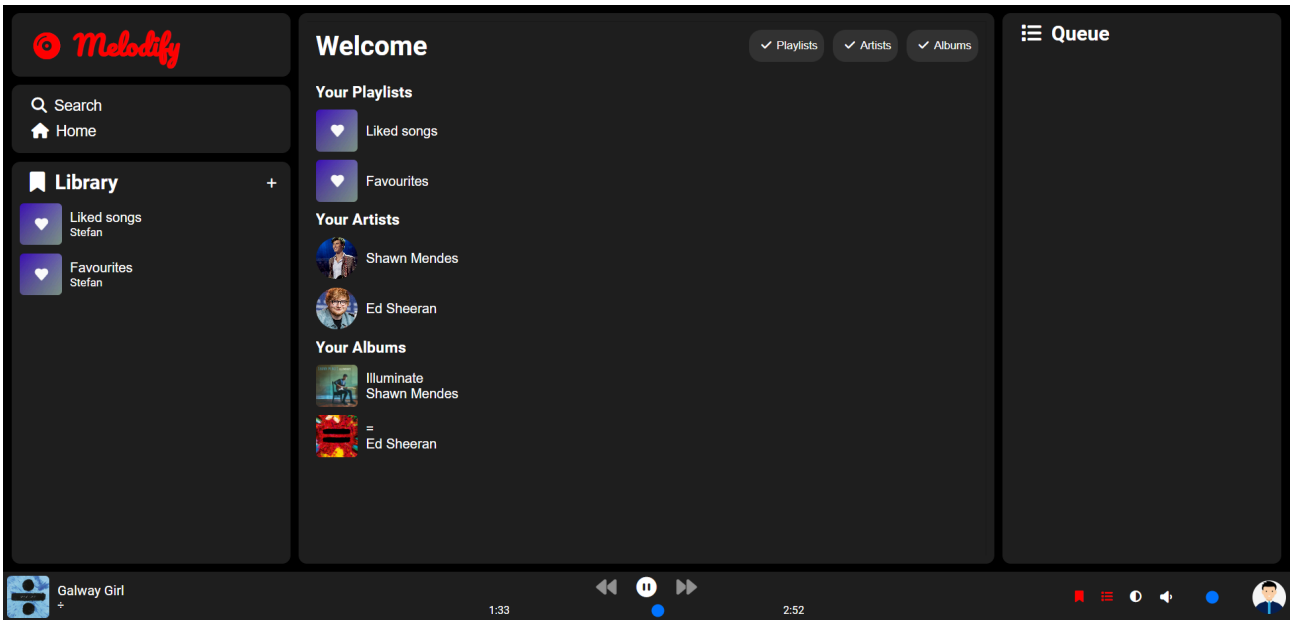
0:00

3:28


Melodify
Search
Home
Library
Liked songs
Stefan
Favourites
Stefan
Search
G
Artists
Albums
Songs
Results
Artists
George Michael
Albums
Ladies & Gentlemen
George Michael
Songs
Galway Girl
÷ • Ed Sheeran
There's Nothing Holdin' Me Back
Illuminate • Shawn Mendes
Queue
Maria, Maria
Smooth
Supernatural
2:12
4:16


Melodify
Search
Home
Library
Liked songs
Stefan
Favourites
Stefan
Ladies & Gentlemen
George Michael 1987-10-30
Title
Time
1 Careless Whisper
George Michael
5:00
2 Freedom
George Michael
6:28
Queue
Mercy
There's Nothing Holdin' Me Back
Shivers
0:00
3:28


Melodify
Search
Home
Library
Liked songs
Stefan
Favourites
Stefan
Liked songs
2024-01-07 21:15:45
Title
Time
1 Shivers
Ed Sheeran
3:27
2 Mercy
Shawn Mendes
3:28
3 There's Nothing Holdin' Me Back
Shawn Mendes
3:25
Queue
Mercy
There's Nothing Holdin' Me Back
Shivers
0:00
3:28



7. Заключение

Екипът ни вложи знания, време, усилия и отношение в изготвянето на този курсов проект. В резултат, ние вярваме, че се получи красив и адекватен продукт спрямо възможностите на екипа от двама човека. За поставената задача беше избрана клиент-сървър архитектура, която най-добре адресира поставения проблем и позволява удовлетворяващо потребителско изживяване. `Backend` инфраструктурата беше базирана на съвременен, стегнат и удобен формат - `REST API`, използващ `JSON` и `HTTP`, разбираем както от хора, така и от машини. Избраното представяне на основните същности - албуми, песни, изпълнители и плейлисти, чрез модели беше съобразено с възможността за поддръжка и бъдещо разширение на софтуера, включително хостване на медийните файлове на външни услуги. Контролерите на сървъра бяха изчерпателно тествани в общи и специфични сценарии, при което беше достигнато задоволително ниво на тестово покритие. Считаме, че потребителският интерфейс, макар и минималистичен, представя удобен достъп до услугите на приложението, без дори да е необходимо ръководство на потребителя. Освен за използване, софтуерът е и лесен за внедряване - за целта е необходимо единствено администраторът да запълни базата данни, да предостави аудио файлове и изображения и да изпълни предоставения jar файл. За създаването на всичко това, екипът използва вече изградени знания, но въпреки това срещна известни трудности, особено при работата с непознати досега външни библиотеки. В процеса на работа ние не само затвърдихме значителна част от знанията, придобити по време на курса - `Beans`, `JDBC`, `Java Persistence API`, `AJAX`, но и изградихме нови умения - за работа с `Maven`, `Lombok`, `Swagger` и дори `Git`.