**Spotify Web Application**

* **Siddharth Chhabra**

**Under Supervision of Mr.Hakan Samci**

**August 16th, 2022**

**Contents:**

1. **Software used**
2. **Installation and Setup Processes**
3. **API connections**
4. **Summary**
5. **Use-Case Diagram (User)**
6. **Sequence Diagram (Login Authentication)**
7. **Sequence Diagram (Payment Portal)**
8. **User Interface + Class Diagram**
9. **Backend Functions**

**Software Used**

* Microsoft SQL Server Management Studio 18
* Visual Studio 2022 (Backend - Microsoft EntityFrameworkCore)
* Visual Studio Code (Frontend - Angular 14.0.6)
* Spotify API (Spotify for Developers - [https://developer.spotify.com](https://developer.spotify.com/))
* Stripe Server (For checking Payment - https://stripe.com)
* Used for creation of Class, Sequence and Use case diagrams – <https://online.visual-paradigm.com>

**Installation and Setup Processes**

* **Extraction of File:**

Extract the zip file by right clicking and clicking on the extract here button to unzip the file.

* **Microsoft EntityFrameworkCore:**

Open the project’s backend code using Visual Studio 2022 Enterprise. Once successfully opened, on the right hand side we see a navigation bar called “**Search Solution Explorer**” where we can see our project name. Right click on the project name and go to “**Manage NuGet Packages…**”. Click on browse and search for these four components:

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Design
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Tools

Once all these components are installed you can use Microsoft EntityFrameworkCore on your project. To run this project simply click on the green play button on the navigation bar with the project name on it. This will build, compile and run the backend application on your browser.

Remember to use “update-database” command in the package manager console terminal so that all migrations and databases are up to date and SQL server updates all relevant data beforehand.

* **Angular:**

Once you have extracted the necessary frontend angular, to run this project we use the command “**ng serve –open**” which will compile the project and run it on a local host in your browser. Just for note to create a new angular project we use the command “**npm install *project\_name*”.**

**API connections**

* **Spotify API:**

To start connection with Spotify web API we just need to generate an authentication key by generating a token from the “<https://developer.spotify.com/console>” website. For this you need to be logged in with your official Spotify account. The authentication key has an expiry after 45-50 minutes, so the authentication key will not be supported after that and then we will have to generate a new key.

* **Stripe API:**

For this you need to create an account on “**https://stripe.com**”. The stripe server will provide you with the authentication keys. This is because the public key and secret key are only generated once logged in. After successfully obtaining the keys we can establish connection using our backend code, replace the old public and secret key with the new ones and connect to the Stripe server. Unlike Spotify API these keys will not expire so we do not need to generate them repeatedly.

**Goal / Motivation**

This beta Spotify web application is a software developed for avid music listeners who like to listen to music on a daily basis. The target audience of this application is people who love music and are willing to spend money on the albums made by their favorite artists. A lot of people like listening to music, so our service is the right choice since it is easy to use and is quite straightforward so that people of all ages can understand how this application works.

**Summary**

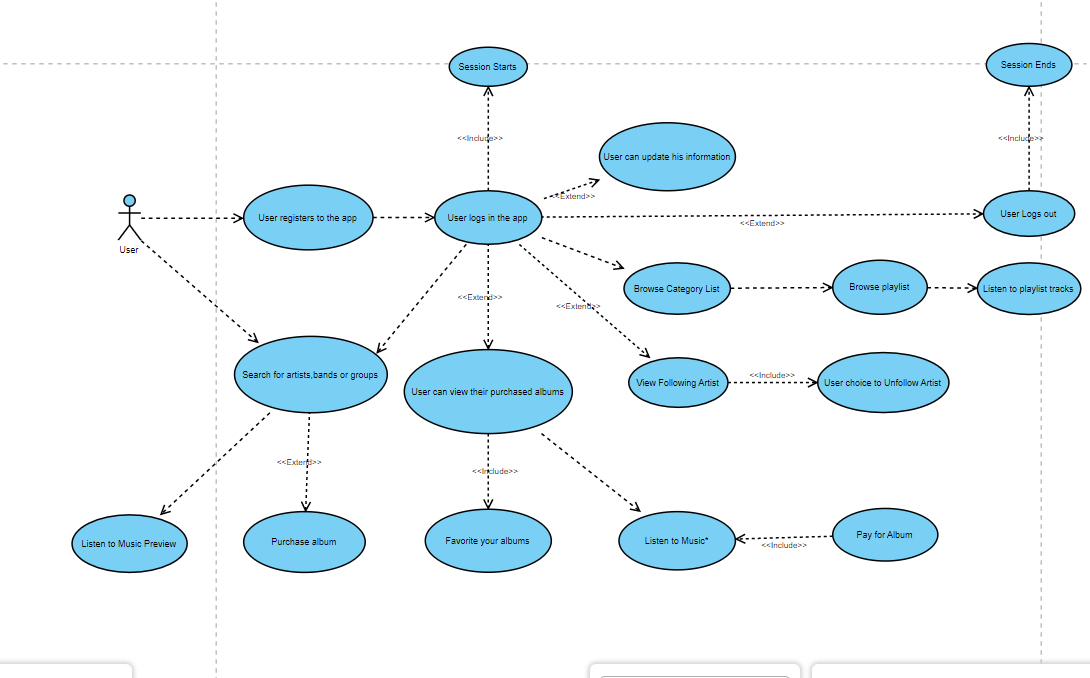
Most tracks that are available on the web application have some preview audio where the music lasts for about 30 seconds. Each user whether registered or not, can listen to this audio. Similarly users can search for an artist, preview their albums and listen to their tracks. If the user likes the tracks he can buy the respective album.

The user has to login to buy the album since only then can the service save the music on the respective account. The card information is never stored on the database so it is impossible for any sensitive payment information to be leaked, therefore it is a safe and secure transaction. The card information is validated with the stripe server and has been tested via Stripe’s mock card information successfully.

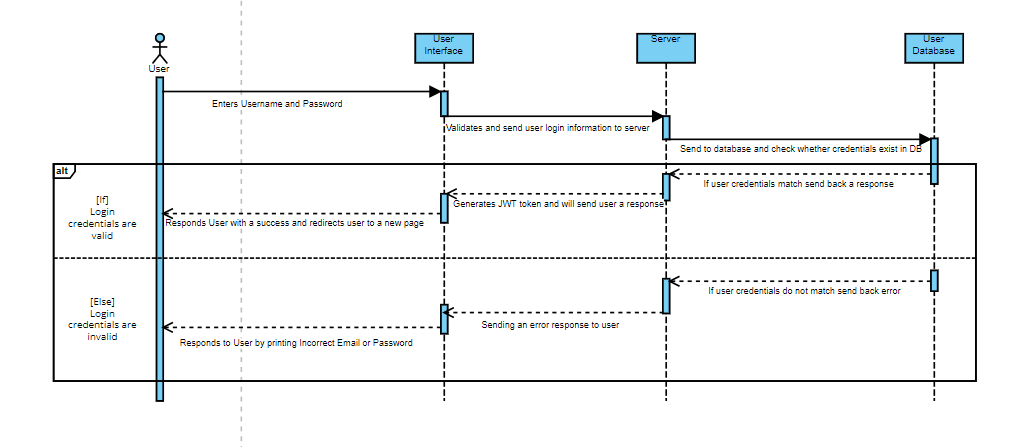
A list of albums will become available to the user once the user purchases his first album. The user after purchasing the album can favorite it and have a separate favorite list where the user can see which albums they have favorited. A registered user who has logged in with their account can follow any artist they want. Once the user has followed an artist they can see a list of artist they follow and have a quick access without having to search for them every time. They can also unfollow an artist if they want sometime later.

The user can also browse through categories where the Spotify provides some categories. Each categories have approximately 20 playlists and each playlists consists of tracks from various artists all at one place so users can get a chance to listen to new artists they do not know about at one place and may connect them to new artists they have never listened to. This feature is also available for all users even if they are not logged in.

**Use Case Diagram (User)**

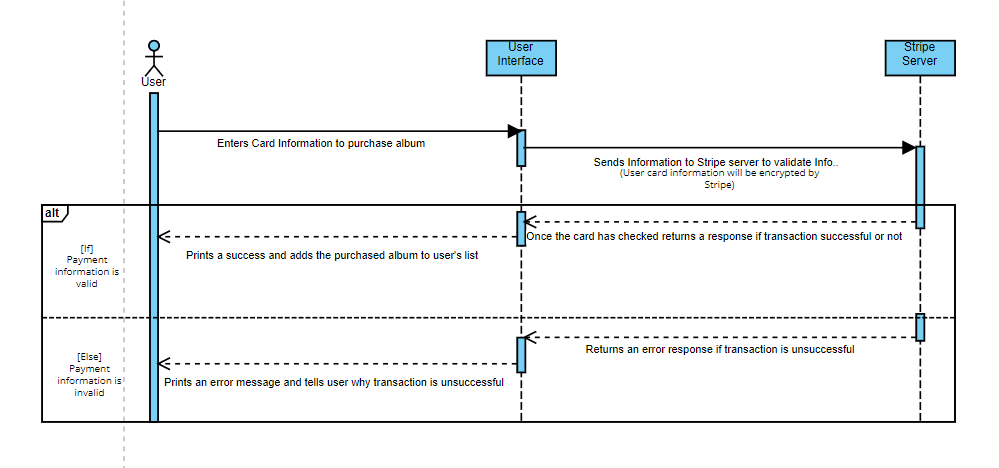


**Sequence Diagram for login authentication**



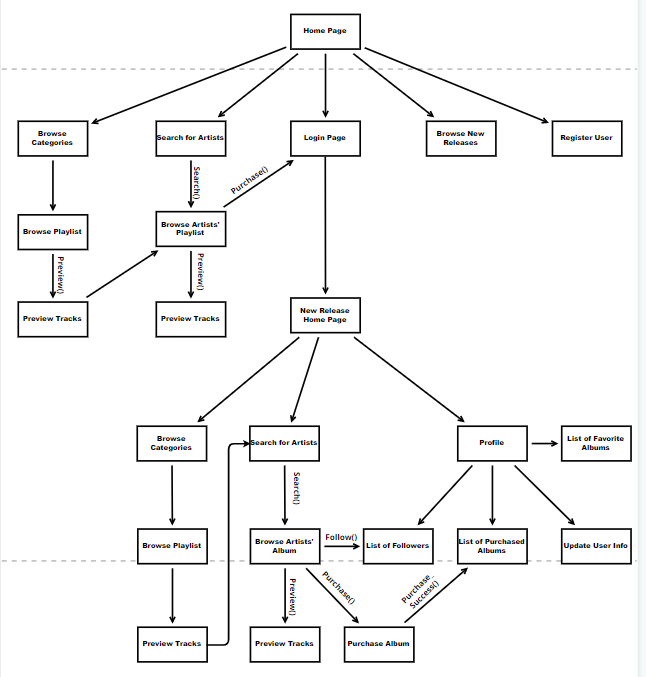
User email and password is verified from the database. As soon as the credentials gets validated it sends a success message to the user interface and session begins. The user is then redirected to the “newrelease” webpage as an authenticated user.

**Sequence Diagram for Payment Portal for User**

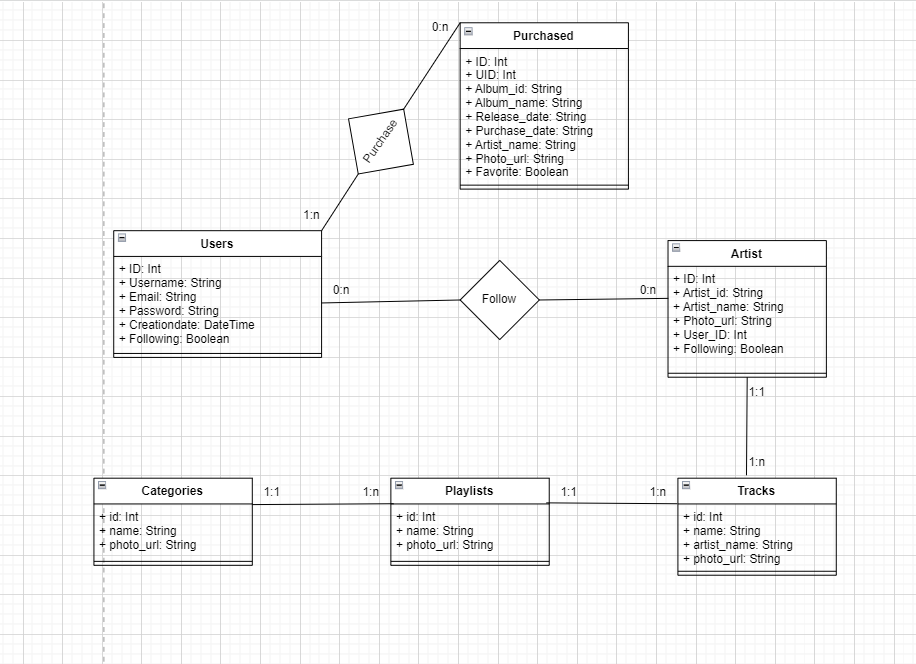


Stripe server encrypts all user credit card information and stores decryption information separately. Therefore, Stripe cannot see any credit card information of any of its users and is safe to use. However in our application testing was done with dummy account provided by stripe itself.

**User Interface**

****

**Class Diagram**

****

**Backend Functions of Project**

**UserController.cs**

In “**UsersController.cs**” we have 7 functions –

* ***Get /api/Users***
* ***Post /api/Users***
* ***Get /api/Users/{id}***
* ***Put /api/Users/{id}***
* ***Delete /api/Users/{id}***
* ***Post /api/Users/Login***
* ***Post /api/Users/CreateUser***

1. The “***Get /api/Users***” function gets all data that is stored in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**” and gets all data from the “**dbo.User**” table. It prints out all different details from the columns including user’s id, username, email, password and creation date.

2. The “***Post /api/Users***” function sends data from an input to the Microsoft SQL Server Management Studio 18 database called “**MusicDb**” in the “**dbo.User**” table.

3. The “***Get /api/Users/{id}*”** function gets all data that stores relevant information to the id(user id) in Microsoft SQL Server Management Studio 18 database called “**MusicDb**” and gets the all data of a specific user from the “**dbo.User**” table.

4. The “***Put /api/Users/{id}****”* function changes specific data or whole data of a user that is stored in Microsoft SQL Server Management Studio 18 database called “**MusicDb**”. It replaces specific or full data from an input by users and it changes the data from the “**dbo.User**” table. In our application we use this function in the “**userinfo.component.ts**” where the user who is logged in can change their information.

5. The ”***Delete /api/Users/{id}*”** function deletes a specific user stored in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**”. It deletes all information about the user from the “**dbo.User**” table.

6. The “***Post /api/Users/Login***” function is a login authenticator which matches user input for email and password fields with the database and then deciding whether the user exists as a registered user in the “**dbo.User**” table or not. This function is also necessary since if login credentials match, JWT token will be generated that last for about 30 minutes by the “**JwtAuthenticationManager.cs**”. The JWT token, expiration, id, email and username will be sent as response to the frontend. They will be added as fields in the local storage with the frontend part. If login credentials do not match there will be no generation of JWT tokens and will basically send an error message. Once login is successful user will be authenticated.

7. The “***Post/api/Users/CreateUsers***” function is a register function in our application which takes input from the user and posts all data in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**”. It adds all the relevant information to the table and now the user can login with these credentials. If same email was registered it will send an error that the account already exists with this email.

**PurchasedsController.cs**

In “**PurchasedsController.cs**” we have 7 functions –

* ***Get /api/Purchaseds***
* ***Post /api/Purchaseds***
* ***Get /api/Purchaseds/{id}***
* ***Put /api/Purchaseds/{id}***
* ***Delete /api/Purchaseds/{id}***
* ***Get /api/Purchaseds/getuid/{uidinp}***
* ***Get /api/Purchaseds/getfav/{uidinp}***

1. The “***Get /api/Purchaseds***” function gets all data from the “**dbo.Purchaseds**” table in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**”. It prints out all different details from the columns such as id, uid, album\_id, album\_name, release\_date, purchase\_date, artist\_name, photo\_url and favorite.

2. The “***Post /api/Purchaseds***” function posts all the data to the “**dbo.Purchaseds**” table in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**”. In our application we use this as soon as a user pays for an album. Once the payment is complete all the relevant data related to user and the purchased album is stored in our database. Important to note here is that **no** payment information is ever stored on our side.

3. The ”**Get /api/Purchaseds/{id}**” function gets specific data of the purchased album by giving the purchase id as input from the “**dbo.Purchaseds**” table in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**. This function is really useful if there might be any problem with client payment all information can be easily obtained by entering the id (Purchase ID) and looking at the details of the purchased item.

4. The “***Put /api/Purchaseds/{id}***” function can change specific data of purchase by entering a purchase id as input, from the “**dbo.Purchaseds**” table in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**.

5. The “***Delete /api/Purchaseds/{id}***” function can delete specific data of purchase by enter a purchase id as input, from the “**dbo.Purchaseds**” table in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**.

6. The “***Get /api/Purchaseds/getuid/{uidinp}”*** function takes uid(User ID) as an input and returns all information related to that specific uid from the “**dbo.Purchaseds**” table in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**. This means it will print out a list of all purchased items of a user from our web application. This is used in our application to print out only the required data to the user and not all purchased information of all users at once.

7. The ”***Get /api/Purchaseds/getfav/{uidinp}***” function is very similar to the previously described function. It takes uid(User ID) as input and returns all information about the albums which the user has purchased and has marked the favorite field as “true” from the “**dbo.Purchaseds**” table in the Microsoft SQL Server Management Studio 18 database called “**MusicDb**. This is again used in our application to print out only relevant information and not everything all at once.

**ArtistsController.cs**

In “**ArtistsController.cs**” we have 6 functions –

* ***Get /api/Artists***
* ***Post /api/Artists***
* ***Get /api/Artists/{id}***
* ***Put /api/Artists/{id}***
* ***Delete /api/Artists/{id}***
* ***Get /api/Artists/getuid/{uidinp}***

1. The “***Get /api/Artists***” function gets all data from the “**dbo.Artists**” table in the Microsoft SQL Server Management Studio 18 database namely “**MusicDb**”. It prints out all different details from the columns such as id, uid, artist\_id, artist\_name, photo\_url, user\_id and following.

2. The “***Post /api/Artists***” function posts all data given as input to the “**dbo.Artists**” table in the Microsoft SQL Server Management Studio 18 database namely “**MusicDb**”.

3. The “***Get /api/Artists/{id***}” function can get specific data by entering the id(following id) and it will show which user has followed which artist from the “**dbo.Artists**” table in the Microsoft SQL Server Management Studio 18 database namely “**MusicDb**”.

4. The “***Put /api/Artists/{id***}” function can change specific data by entering the id(following id) and it will show which user has followed which artist and change any information related to the input id from backend from the “**dbo.Artists**” table in the Microsoft SQL Server Management Studio 18 database namely “**MusicDb**”.

5. The “***Delete /api/Artists/{id}”*** function can delete specific data by entering the id(following id) and it will delete any information about the user following an artist from the backend.

6. The “***Get /api/Artists/getuid/{uidinp}***” functions gets all data that is relevant to a user. This function basically shows all artist info about users who are following an artist. So it shows a list of all artists that a user follows. As soon as a user unfollows the artist will be removed from this list.

**PaymentController.cs**

In “**PaymentController.cs**” we have 1 function –

* ***Post /api/Payment/pay***

1. The “***Post /api/Payment/pay***” function takes all data as input from the user. We also have a “**MakePayment.cs**” file which connects to the stripe server. When this post function is called the inputs by the user are checked with Stripe server. The process to connect to the stripe server has been explained during the installation/startup step before so it is assumed that we already have a safe and secure connection with the private and public keys. If the charge was successful it will return a success message, otherwise it will return a failure.