MUSIC STREAMING WEBSITE

A document report was submitted to the School of Computer Science and Engineering in partial fulfilment of the requirement for the degree of
Advanced topics in web development Course.

Instructor:

Assoc. Prof. Nguyen Van Sinh

In specialization by

Mr: Le Thanh Phuong Nam

ID (HCMIU) : ITITWE19025_ Student number (UWE) : 23083609

International University - Vietnam National University HCMC

University of the West of England

Project Presesentation Video:

Youtube link: https://youtu.be/FE46XTmiXjw?si=Bk169vZuEQ1MQjly

Google Drive link:

https://drive.google.com/file/d/15SGH0xjXo78taVMQ9wkqQfoVXxDwYrJQ/view?usp=sharing

Project Github link:

https://github.com/NamBobby/Advanced_Topics_in_Web_Development_Project

January 2025

ACKNOWLEDGEMENTS

To complete the project, I really appreciate Assoc. Prof. Nguyen Van Sinh and MSc. Nguyen Trung Nghia, the two primary people who spent their time instructing me to complete the Advanced Topics in Web Development course. Additionally, they also gave me useful advice to improve the project by fixing shortcomings. Regarding resources, University of the West of England as well as International University - Vietnam National University, Ho Chi Minh City have provided what was needed for the development phase. Finally, my classmates and close friends inspired this idea and motivated me to overcome the initial challenging phase of the project.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This is a detailed presentation of how I built this web platform that can play music online for an Advanced Topics in Web Development course. With the requirements of the course, this project was built for me in a 3-month semester with the frontend interface built using ReactJS technology combined with Vite and Node.js for the backend. Regarding the MySQL database, I combined it with Sequelize ORM from the backend. The highlight is that the platform enhances security by supporting the authentication of JSON Web Tokens (JWT) and hashing of user secrets using Bcrypt.js. Finally, I strived to design the interface to respond to user's requests in real time to achieve a seamless user experience.

Chapter One: INTRODUCTION

1. Overview

Music Streaming Website is a special web development project for building an interactive and scalable website for music lovers and independent artists. As a personal project, I used the knowledge I learned to build a website that emphasizes empowering artists to upload and manage their music independently, allowing users to enjoy it. To meet the requirements for this course, we applied ReactJS + Vite for the frontend and Node.js for the backend. (UWE Bristol, n.d.)

2. Background

Nowadays, music streaming services have changed the way users consume music, returning from using music video viewing platforms to audio music and podcasts. These platforms have stepped up in developing user-friendly interfaces and features like a social network that focuses only on providing audio products (UWE Bristol, n.d.). Therefore, in addition to the goal of practising website building, this project also tries to learn how to apply and build a user-friendly and accessible website for artists to introduce their works to the public. The platform adopts album sorting, playlist management, music uploading and other features inspired by present day technology developments, with a clean user experience.

The project also addresses the growing academic need to apply new technology in implementing advanced techniques in web development and complying with the standards of the **Advanced Topics in Web Development** course. Additional complexity includes secure authentication mechanisms, good data handling and scalability for increasing user workload.

3. Scope and Objectives

3.1. Scope:

The project aims to help artists upload and self-manage their music. In addition, allowing users to create and customize playlists, which strives to provide a safe and engaging user experience with a responsive design..

3.2. Objectives:

The project is fully built on the stack of ReactJS (for frontend) and Node.js (backend), using **ORM** (Sequelize) to manage database, with the use of advanced things like real time stream, secure authentication and user access control (UWE Bristol, n.d.).

4. Assumption and Solution

4.1. Assumptions:

- Modern web technologies can be handled by different devices that users have access to.
- Content will be uploaded by primary artists who will abide by copyright rules.
- The backend infrastructure can efficiently handle multiple concurrent users.

4.2. Solutions:

- A secure authentication & session management can be done by using **JWT (JSON Web Tokens)**.
- Using Sequelize ORM to implement robust database designs for music, users, albums and playlists (UWE Bristol, n.d.).
- State management and a responsive user interface using optimized React hooks and Redux.
- Scalable API designed for frontend and backend communication.

5. Structure of report

This report is structured to cover all aspects of the project systematically:

- Chapter Two: Literature Review - An overview of relevant technical concepts and comparison with existing platforms.
- Chapter Three: Methodology - Details method through architecture and UML diagrams; using advanced method.
- Chapter Four: Implementation and Results - Documents implementation, Results achieved.
- Chapter Five: Conclusion and Future Work - Summarizes the findings and proposes directions for future enhancements.
- References - Lists all academic and technical resources consulted during the project.

Chapter Two: LITURATURE PREVIEW

1. Chapter Overview

In this chapter, I focus on listing the technologies, languages, tools and platforms that are applied to the project and compare their strengths and weaknesses to other technical concepts that could be applied but are not used. Finally, I would compare the concept of the project to a common web platform to determine why this project was built and describe the difference in it (if any).

2. Technical Background

This section is about the technologies applied in this project, what programming languages, what tools and platforms and so on. Each technology is chosen to meet the course requirements while ensuring scalability and ease of development, providing a comparison of their strengths and weaknesses against alternatives, explaining the rationale behind each choice

2.1.1. Programming Language

Projects are allowed to use only JavaScript or PHP for both front-end and back-end development.

2.1.1.1. What is JavaScript?

JavaScript is a popular language used in making dynamic and interactive features on a website (UWE Bristol, n.d.). It is an interpreter object-oriented language, available server side and client side.

Reason for Choice: JavaScript has been picked for streamlining development across the entire stack — Node.js for the backend and ReactJS for the frontend.

2.1.1.2. Comparison:

There are so many benefits to the use of javascript such as easy libraries and frameworks like ReactJS and NodeJS used to make development work easier and more efficient. It has strong community support, is documented and is an easy-to-use, capable system. However, JavaScript's single-threaded nature can become an issue in cases where CPU-intensive tasks are required and become complex in larger project development without strict conventions. Smaller server-side applications can be set up with PHP, it is also very good at templated web pages. However, while using PHP as a separate language for rendering backend logic makes separation easy, it also makes the integration with JavaScript-based frontends take more time and introduces the headache of supporting maintenance elements(Stamey and Saunders, 2006).

2.1.2. Backend Platform

The course allowed the use of either Node.js or PHP for backend development.

2.1.2.1. What is Node.js?

Node.js is basically a JavaScript runtime environment on Google Chrome's V8 engine. It is based on an event driven, non blocking I/O model highly efficient for real time applications.

Reason for Choice: Node.js does natively support the asynchronous processing which is good for real time streaming and concurrent requests. It's compatible with javascript and that makes the stack simpler and moreover, faster development (Tilkov and Vinoski, 2010).

2.1.2.2. Comparison

Node.js is based on JavaScript runtime live on Chrome's V8 engine which has been built for efficient and scalable applications (UWE Bristol, n.d.). By moving into a unified JavaScript environment in Node.js, we remove many of the layers of the stack, making development easier. But Node.js is not ideal for CPU-intensive tasks and needs to be managed carefully in regards to asynchronous code otherwise the code can fall prey to Callback hell (Basumatary and Agnihotri, 2022). On the other hand, PHP, a second choice, is easy to deploy and appropriate for templated web pages and small-scale apps. PHP is a simple enough language, but the real processing power it lacks creates additional complexity if paired with JavaScript-based frontends which makes it less suited for modern, real-time applications in this project (Brooks, 2011).

2.1.3. Frontend Framework

2.1.3.1. What is ReactJS?

Facebook made ReactJS a JavaScript library for building user interfaces using reusable components, efficiently rendering a Virtual DOM (Thakkar, 2020).

Reason for Choice: I chose ReactJS because of its component-based architecture enabling scalable and reusable UI design. In addition, Vite brings fast builds + live reload to the developer experience and I integrated it to improve user experience.

2.1.3.2. Comparison

Due to the Reactjs component-based structure, which is reusable and scalable, as well as the help of Virtual DOM that will maintain the UI rendering and updates as fast as possible to the user (Keshari et al., 2023). ReactJS linked with tools like Vite accelerates development speed with quicker builds, live reload capabilities and so on; however, ReactJS depends a lot on third-party libraries. In contrast, Angular provides a full framework with its tools for routing and state management but it's heavier and so leads to slower initial loads and more complexity.

2.1.4. Database

2.1.4.1. What is MySQL?

MySQL is a relational database management system (RDBMS) that uses SQL (Structured Query Language) to store and query the data. For most people, its reliability and performance in managing structured data is widely known. I managed my project's database by using MySQL Workbench.

Reason for Choice: Since the app uses relational data and MySQL is the standard, I decided to use MySQL to avoid scaling as long as possible and invest most time in the design of the app. The database operations were made simpler as well as flexible in query design by integration of Sequelize ORM (Karayiannis, 2019).

2.1.4.2. Comparison

I can manipulate data easily when using MySQL with Sequelize ORM while still maintaining flexibility for query design, but the problem is that it always requires predefined schemas. MongoDB, a NoSQL database, can replace it because it requires no schema and works well for fast-changing applications. But when I use it for my project, it's not very efficient on complex relational queries.

2.1.5. Development Environment

2.1.5.1. What is Visual Studio Code (VS Code)?

Microsoft is a lightweight and versatile code editor version called Visual Studio Code. It has extensions, plays nice with Git and includes awesome debugging tools.

Reason for Choice: JS-based web development is simple, fast and has a great library of extensions thus I opted for VS Code (Freeman, 2016).

2.1.5.2. Comparison

I think Visual Studio Code has some great compelling features to make developing easier and more efficient. But, similarly, Microsoft Visual Studio provides a wider range of features; although it uses more resources, it is better suited for enterprise scale applications.

2.1.6. Additional Libraries and Tools

- **Multer**: Used for file uploads (e.g., music and thumbnails), ensuring efficient handling of large files.
- **JWT (JSON Web Tokens)**: Provides secure user authentication and session management (Jones, Bradley and Sakimura, 2015).
- **Bcrypt.js**: Used to securely hash passwords, ensuring data protection (ARAVINDA A KUMAR and Divya TL, 2024).
- **Nodemailer**: Implements email functionality, such as sending OTPs for user verification.
- **Github**: Allowing me to work on projects together by managing code versions, reviewing changes and resolving conflicts

3. Comparison with existing platform

3.1. What is Spotify?

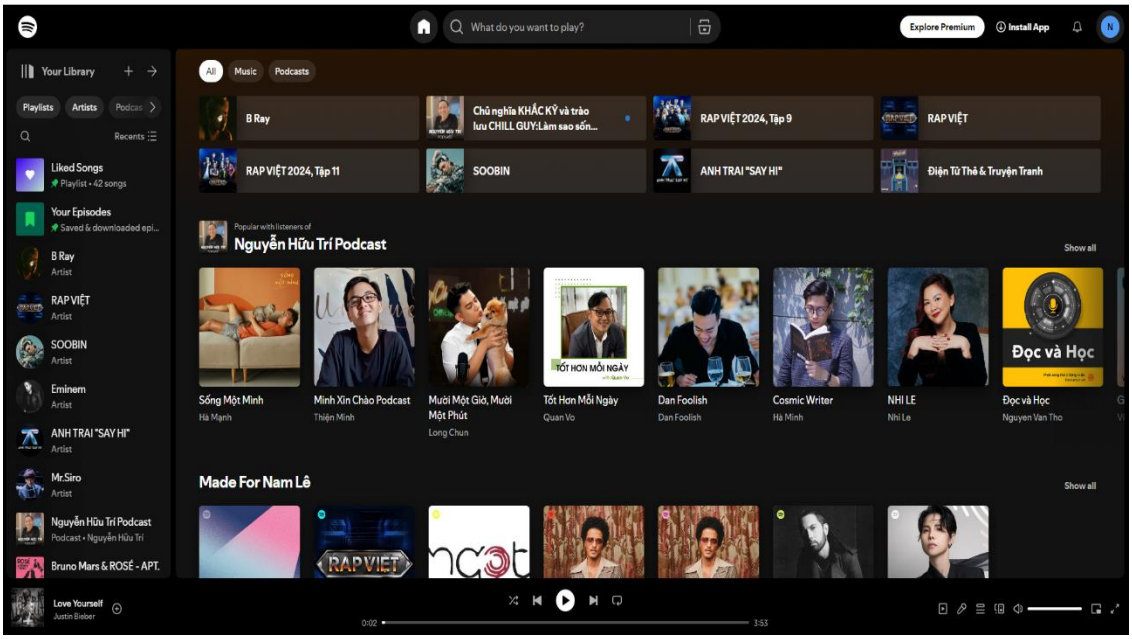

Figure 1. Spotify Platform

Spotify (*Figure 1*) is a globally prestigious music streaming platform accompanied by a great polished experience (real-time streaming, personalized recommendations, large library of licensed music).

3.2. Why Develop This Project Instead of Using Spotify?

1. **Learning Objective**:
   The goal is to understand and practice the technologies and techniques used in such a complex system, by replicating Spotify's core functionalities. With a smaller-scale web development, I think I can gain new knowledge in API design, state management and database optimization.

2. **Project Scale**:
   This project is different from Spotify, applying on a scale of millions of users (UWE Bristol, n.d.). Therefore, I try to reduce the scaling, allowing me to focus on the primary functions without making the development out of scope.

3.3. Unique Features of My Project

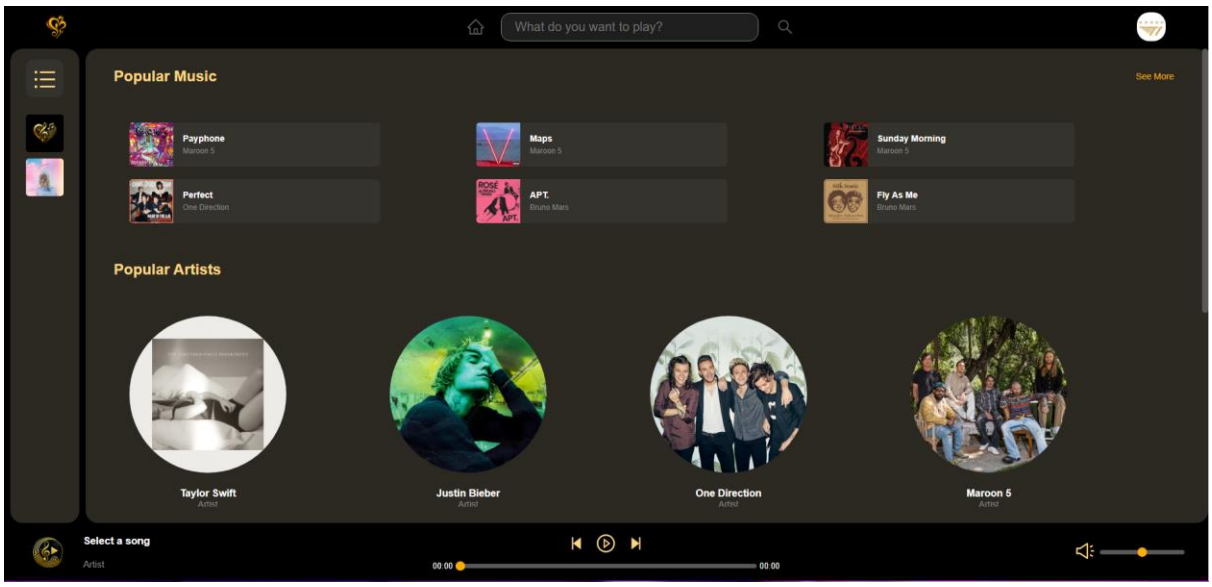

Figure 2.Music Streaming Website

While Spotify is a robust platform, this project (*Figure 2*) has unique features that are consistent with its academic nature:

- **Avoids Copyright Limitations**:
  Unlike Spotify, independent artists are only permitted to upload original content and avoid the complications of licensing and copyright agreements. With this, the process of development becomes so much easier and also creative. On one hand, since the project is small-scale for academic practice purposes, it is not meant to be used publicly.
- **Customized Features for Learning**:
  Features that would not be seen in large-scale services such as Spotify are included in the project in order to demonstrate expertise like advanced authentication processes or manual playlist management.

Chapter Three: METHODOLOGY

1. Chapter Overview

In this chapter, I am concentrating on the methods used through UML diagrams from architecture, database, use cases and so on.

2. Challenges

Because of the challenges I encountered during the development phase, which required adjustments in the planning and execution process, which provided context for decisions about the design and development of the project. When testing and debugging I found it quite difficult to balance the completion time of the project and the same set of parallel courses. In addition, although the project was required to be done individually, it was changed from a three-person team to an individual after three

weeks of disseminating the project requirements from the instructor. This created time constraints and re-planning from scratch so that the scale could be minimized and completed by a single person based on the original idea. Thus, it allowed me to complete a full-stack project from the frontend to the backend for the first time.

3. System Design

3.1. System Architecture

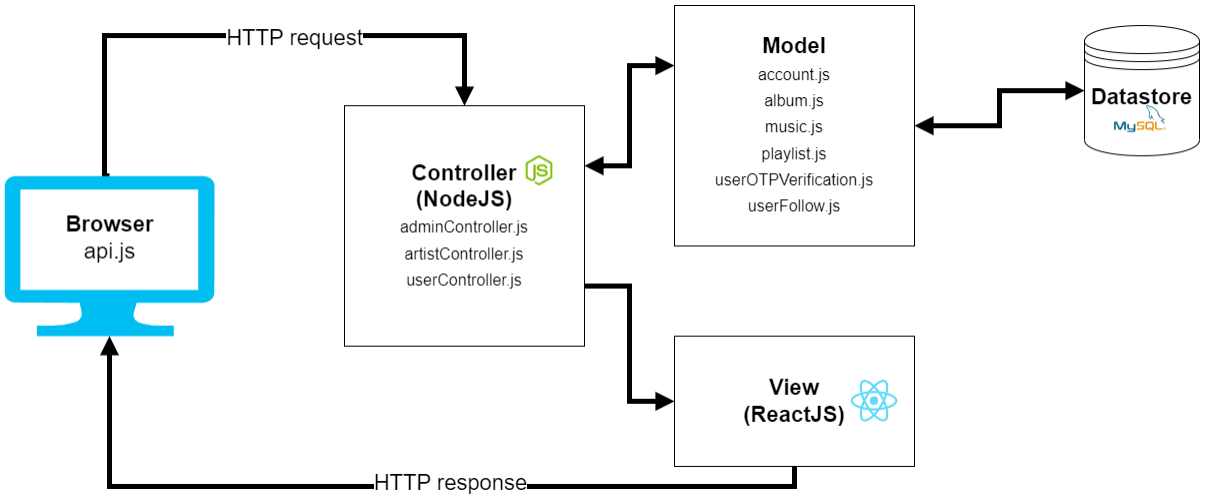

Figure 3. MVC System Architecture Diagram

The System Architecture Diagram (*Figure 3*) contains the logical flow of the system which I used following the Model View Controller (MVC) pattern (UWE Bristol, n.d.). This means that the Controller (Node.js) of the project will communicate with the Models which are Account, Album, or Music and the Browser (View) of the website will communicate with the Controller, as a whole. It uses a data storage layer as a database (MySQL). They allow easy data transfer between the frontend and backend (using ReactJS) using HTTP requests and HTTP responses.

3.2. Use case Diagram

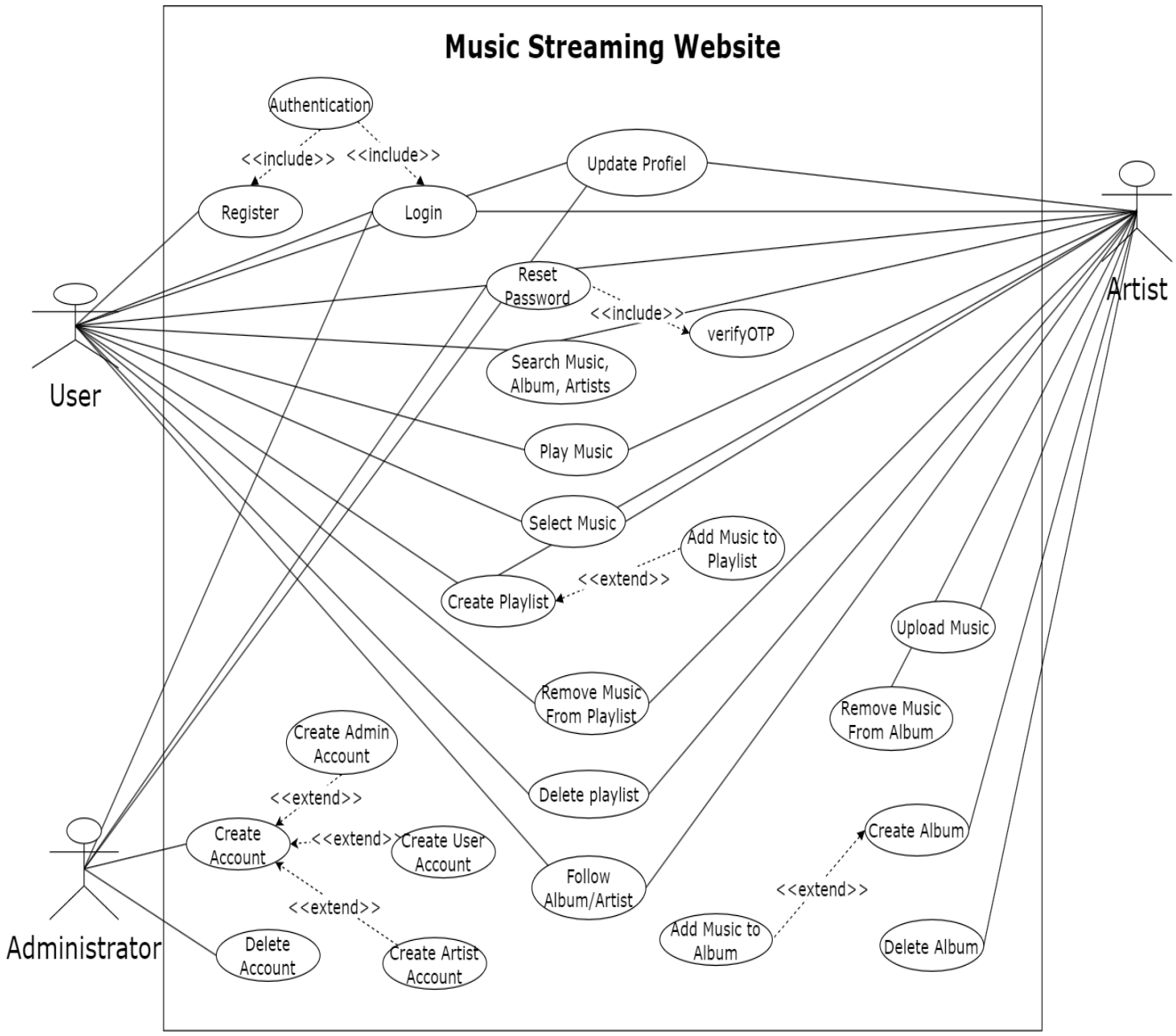

Figure 4.Use Case Diagram

The use case diagram (*Figure 4*) above shows the actors (User, Artist, Admin) and the relations they have with the system. The core use cases in my project include "Login", “Search Music”, “Upload Music”, “Artist/album follow” and so on. While the admins are in charge of accounts, the users and artists deal with the generation, maintenance as well as the relationship with the component. This diagram focuses broadly on the roles of the particular system their relative privileges and their actions.

3.3. Physical Database Diagram (Relational Database Diagram)

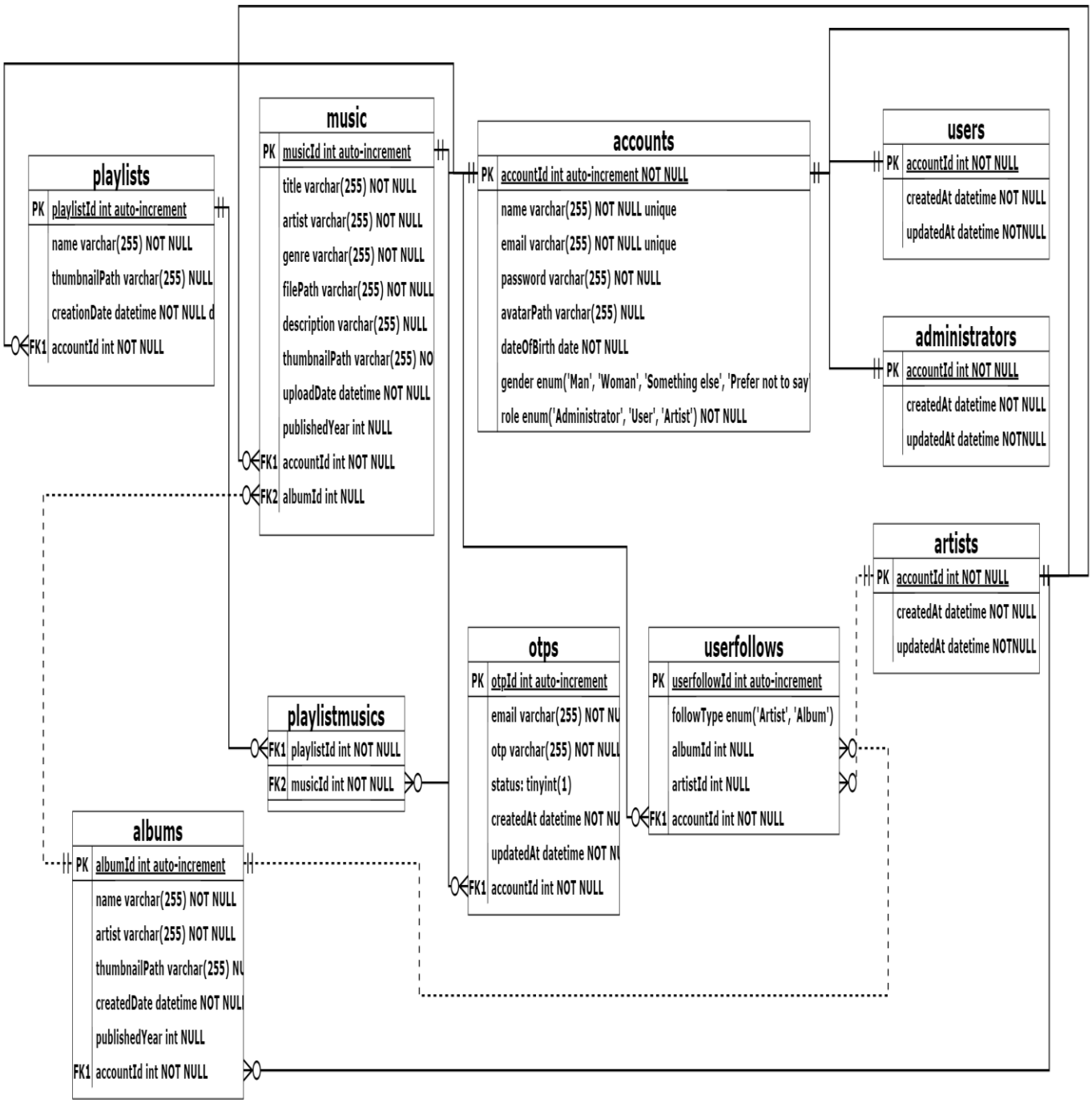

Figure 5.Physical Database Diagram

The Physical Database Diagram (*Figure 5*) shows a detailed view of database schema i.e., table definition, primary key and foreign key and data types. We can connect tables, such as accounts, playlists, albums, music and userfollows, to each other relating them with 'belongs to' and 'includes'. While this diagram guarantees database normalization and data integrity, it is also a perfect blueprint for database implementation (UWE Bristol, n.d.).

3.4. Entity–relationship Diagram

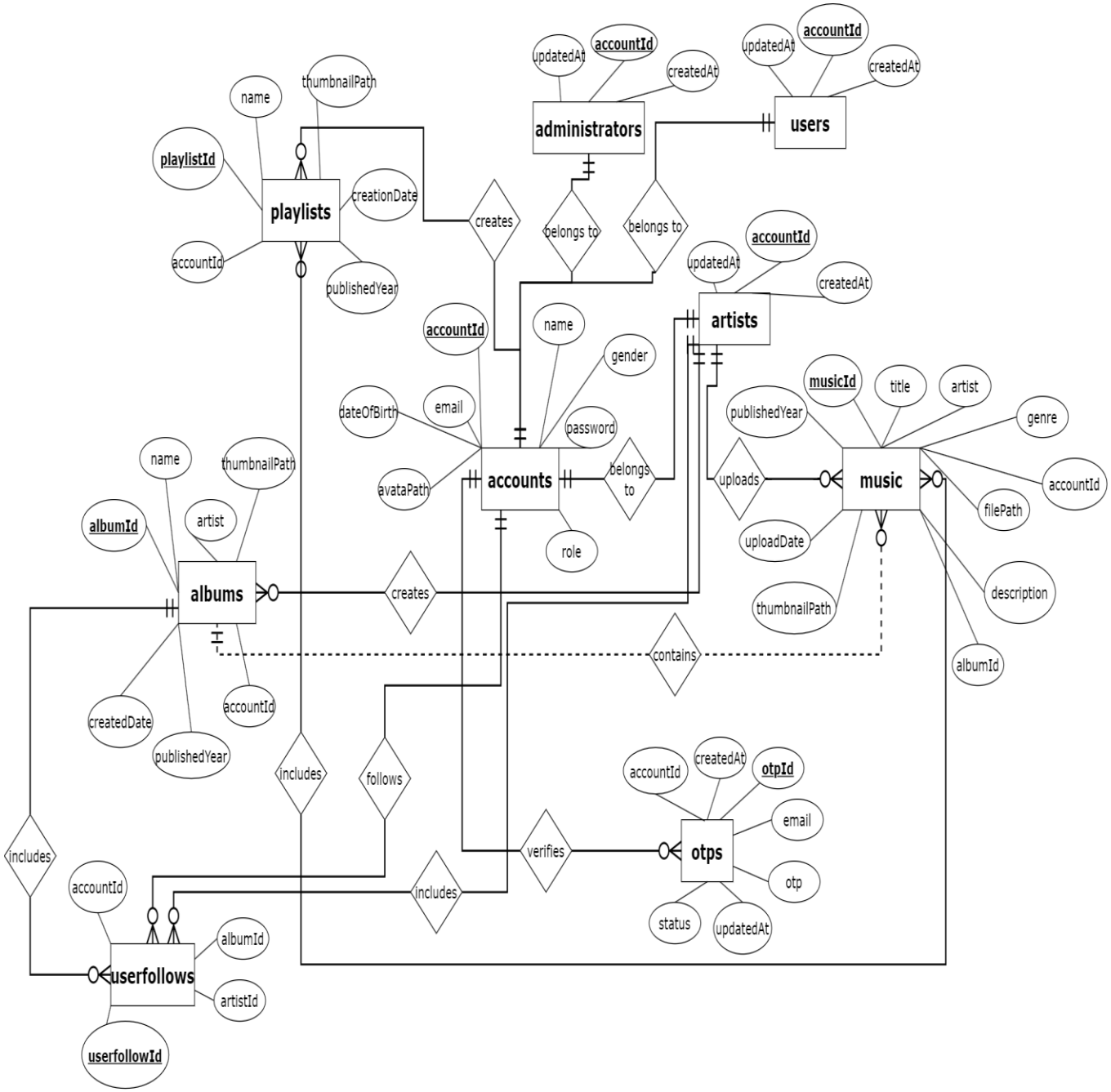

Figure 6.Entity-Relationship Diagram

I represent the database structure of the music streaming website by the Entity-Relationship Diagram (ERD). So it defines accounts, playlists, albums, music, users and their relationships. Suppose the accounts entity is the centre hub to connect with users, artists, administrators and albums entity is related to music and artists. This diagram (*Figure 6*) includes relationships such as "creates", "includes" and "follows" which keeps data modelling around managing playlists, music uploads and user interactions clean.

3.5. Sequence Diagram

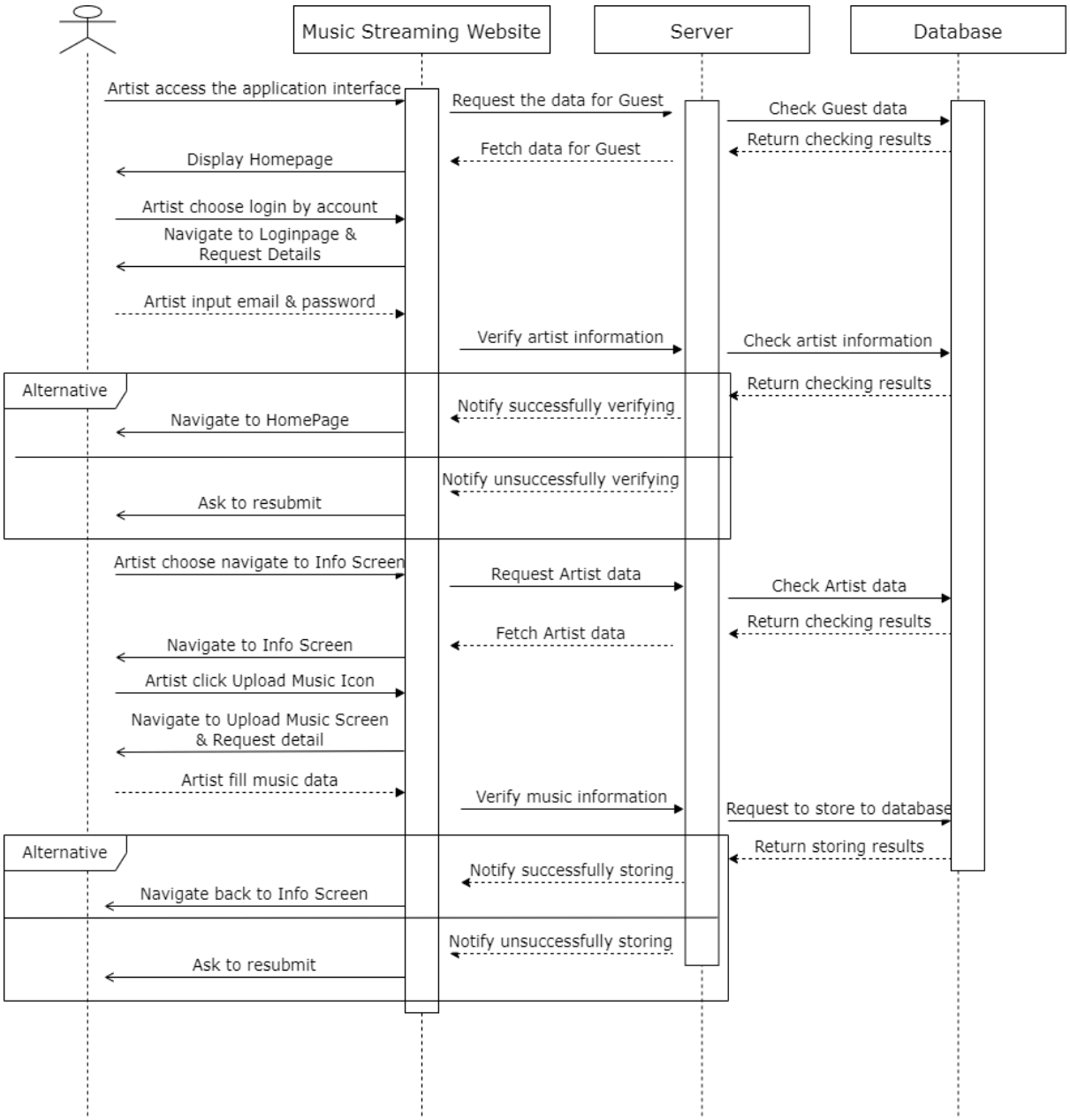

Figure 7.Sequence Diagram

The sequence diagram (*Figure 7*) above describes how the user interface, server and database interact with one another in the process of uploading music by the artist to the website just as it depicts the flow of other features of the project. The diagram above shows how transactions like music upload and data retrieval take place (UWE Bristol , n.d.). At first, an artist logs in and the server looks for the artist's credentials in the database. Then, the artist submits music after authentication, but instead of just saving a file, the artist must first verify the input data before it is stored in the database. When failing to action—wrong credentials, invalid input and so forth—alternate flows are included.

3.6. Class Diagram

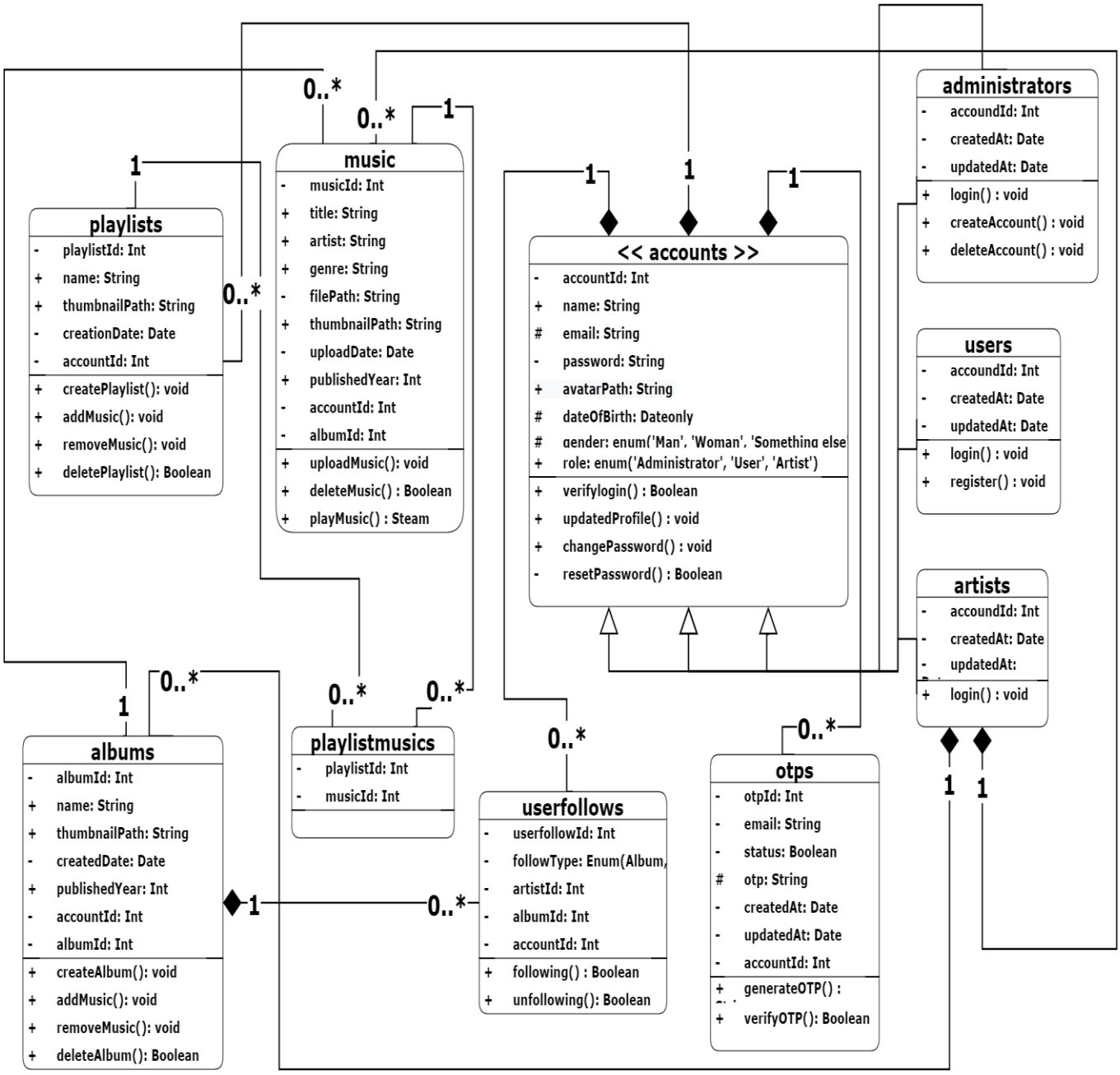

Figure 8.Class Diagram

Class Diagram (*Figure 8*) specifies the structure of system objects and allowed behavior. The class hierarchy is quickly formed with key classes: accounts, users, artists, music, albums and playlists, with attributes and methods that apply to their respective purposes. The Music class has attributes like title, genre, filePath and upload, delete, and play that music. Examples of system interactions are depicted as relationships like inheritance (e.g. Users and Artists of Accounts) and associations.

Chapter Four: IMPLEMENTATION AND RESULT

1. Chapter Overview

This chapter outlines my project's primary functionalities, their complexities, and development results that I spent the most time implementing. The project features a robust authentication system, friendly interface and secure API for the music streaming platform.

2. Workflow

A structured workflow was implemented to make sure the project was finished successfully within the deadline time; it serves as a natural introduction to how the work was structured and provides a clear narrative before diving into the technical results. I created a plan and executed the project step by step while writing the report. I started by building the backend server, with a MongoDB database from which I created the APIs and tested the responses through the Postman application. Then, I changed NoSQL to MySQL due to my familiarity with the tool through prior training and also because the project was being changed to a single person. Next, I designed and built the frontend from the main containers and added the appropriate components. Then, I made calls and used the backend API from the frontend. Finally, I tested and debugged each feature till the backend and frontend were all fixed and running simultaneously and tested the response on different browsers to check for compatibility and performance of the project.

3. Authentication functionality

3.1. Environment setup

Backend Implementation: JWT (JSON Web Tokens) was used to implement authentication for secure session management (UWE Bristol, n.d.). The auth middleware helps me to ensure these private routes are protected. Verification is done through tokens present in the HTTP requests. Passwords are hashed on the database using Bcrypt, meaning they are secure even if compromised or the developer's data management view (ARAVINDA A KUMAR and Divya TL, 2024).

Frontend Implementation: The Ant Design components were used to build the ReactJS forms for user login and registration. In Axios, I customized it to have interceptors to handle authentication tokens and a global loading spinner when handling requests through *auth.context.jsx* component and axios.customize.js service. Everything related to the authentication state is managed via a global context in React so that the user maintains a smooth feel throughout the app (Keshari et al., 2023).

3.2. Complexity

With conditional logic introduced to manage user roles (Admin, User and Artist) was able to segregate access to routes and functionalities, rendering through the *roleBasedRouter.jsx* component used react-router-dom in frontend (Keshari et al., 2023). In terms of backend, I added middleware auth.js (*Figure 9*) to protect

endpoints to secure the APIs ballooned the backend complexity, for example, the middleware that verifies tokens and checks permissions dynamically. For instance, the middleware gets a token from the headers and ascertains it before attaching users details to the request object for further processing.

```
if (req.headers && req.headers.authorization) {
  const token = req.headers.authorization.split(" ")[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    const user = await Account.findOne({
      where: { accountId: decoded.accountId },
    });
    if (!user) {
      return res.status(401).json({ message: "Account not found" });
    }

    req.user = {
      accountId: user.accountId,
      email: user.email,
      name: user.name,
      dateOfBirth: user.dateOfBirth,
      avatarPath: user.avatarPath,
      gender: user.gender,
      role: user.role,
    };

    //console.log(">>> check auth: ", req.user);
    next();
  } catch (error) {
    return res.status(401).json({ message: "TokenExpired/Error" });
  }
} else {
  return res.status(401).json({ message: "Unauthorized access" });
}
```

Figure 9.Authentication Middleware Function

3.3. Result

The authentication system ensures secure and efficient user management:

- Any roles can register, log in, and update their profiles seamlessly.
- JWT ensures that only authorized users to access private routes.
- Access permissions are well segregated through three roles i.e. Admin, Artist & User, keeping data secure (UWE Bristol, n.d.).

For instance, any login response would generate an access token and user details which securely store the state of the session in the browser's local storage. Errors such as an expired token are gracefully handled by the system increasing robustness.

4. Upload functionality

4.1. Environment setup

In the backend, multerConfig.js middleware is used to deal with file uploads and only supported types of files and file sizewise are accepted (UWE Bristol, n.d.). It also stores Music data such as title, or optional it is album to the database using Sequelize ORM. It will keep your music files and thumbnail sorting files in an organized local directory and the path will be stored in the database for easier calling and reducing resources. We have the metadata and audio files in the frontend, and they go together with React and Ant Design to create a form for us. To safely upload the files to the server, I made form-data requests with Axios.

4.2. Complexity

This middleware supports me in validating all the file types (*Figure 11*), paths (*Figure 10*), sizes (*Figure 12*) and so on, which automatically renames validation by Crypto (*Figure 10*) to check if the file format response from the service and return to the frontend is valid too. Moreover, this also adds extra database links and options to select albums to hitch uploaded songs to their related artists (Account role).

- Storage configuration:

```
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    if (file.fieldname === "musicFile") {
      cb(null, "./src/uploads/music/");
    } else if (file.fieldname === "thumbnail") {
      cb(null, "./src/uploads/music/thumbnails/");
    } else if (file.fieldname === "albumThumbnail") {
      cb(null, "./src/uploads/albums");
    } else if (file.fieldname === "playlistThumbnail") {
      cb(null, "./src/uploads/playlists");
    } else if (file.fieldname === "avatar") {
      cb(null, "./src/uploads/avatars/");
    } else {
      cb(new Error("Unknown fieldname"), false);
    }
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = `${Date.now()}-${crypto.randomBytes(6).toString("hex")}`;
    cb(null, `${uniqueSuffix}${path.extname(file.originalname)}`);
  },
});
```

Figure 10.Storage Configuration Function

- File filter

```
const fileFilter = (req, file, cb) => {
  const allowedMusicTypes = ["audio/mp3", "audio/mpeg", "audio/aac"];
  const allowedImageTypes = ["image/jpeg", "image/png", "image/webp"];

  // Check if it's a valid music file
  if (file.fieldname === "musicFile" && !allowedMusicTypes.includes(file.mimetype)) {
    const error = new Error("Incorrect music file type");
    error.status = 400;
    return cb(error, false);
  }

  // Check if it's a valid image file
  if (["avatar", "thumbnail", "albumThumbnail", "playlistThumbnail"].includes(file.fieldname) && !allowedImageTypes.includes(file.mimetype)) {
    const error = new Error("Incorrect image file type");
    error.status = 400;
    return cb(error, false);
  }

  cb(null, true);
};
```

Figure 11.File Filter Function

- Size Checking

```
const checkThumbnailSize = (req, res, next) => {
  if (req.files) {
    if (req.files.albumThumbnail && req.files.albumThumbnail[0].size > 5 * 1024 * 1024) {
      return next(new Error("Album thumbnail file size exceeds 5MB limit"));
    }
    if (req.files.playlistThumbnail && req.files.playlistThumbnail[0].size > 5 * 1024 * 1024) {
      return next(new Error("Playlist thumbnail file size exceeds 5MB limit"));
    }
    if (req.files.musicFile && req.files.musicFile[0].size > 5 * 1024 * 1024) {
      return next(new Error("Music file size exceeds 5MB limit"));
    }
    if (req.files.avatar && req.files.avatar[0].size > 5 * 1024 * 1024) {
      return next(new Error("Avatar file size exceeds 5MB limit"));
    }
  }
  next();
};
```

Figure 12.Check File Size Function

4.3. Result

Uploading music allows artists to safely place their songs along with their metadata and thumbnails. Frontend and backend are integrated seamlessly and uploaded songs are added instantly to play. There's also flexibility for artists to link songs to particular albums, or just keep them as individual tracks.

Chapter Five: CONCLUSION AND FUTURE WORK

1. Chapter Overview

This chapter summarizes the achievements and key outcomes of the music streaming platform development while proposing future enhancements to expand its capabilities and user experience.

2. Conclusion

This project developed a music streaming platform that successfully achieves the core objectives of the music streaming platform, which includes robust, scalable system possessing the requisite features such as authentication, music streaming, playlist creation and the uploading of music.

Key achievements include:

- **Scalability**: With its MVC architecture, ReactJS for frontend and Node.js for backend it creates a scalable and maintainable system. With this structure, the platform can easily grow into more features.
- **Security**: The platform incorporates JWT for secure authentication and Bcrypt.js for password encryption, ensuring strong protection of user data and access.
- **Efficiency**: Using Multer for file uploads and chunk-based streaming significantly improves performance, enabling smooth handling of large music files.
- **Database Management**: Using MySQL with Sequelize ORM, the platform gets good relational mapping and great interactions with the database.

This project gave me practical real life experience in full stack development from a frontend vision to a backend logic through database integration. Future improvements to the platform have been made possible by all those experiences that have been gained.

3. Future work

Future improvements are suggested to improve the functionality, scalability, and user experience of the platform including (UWE Bristol, n.d.)

3.1. Advanced Features

- **Personalized Recommendations**: Based on user preferences, history and trends, using machine learning algorithms to suggest music.
- **Real-Time Collaboration**: Allow users to collaboratively create and manage playlists, promoting social interaction and shared experiences.
- **Offline Mode**: It helps users download music for offline play back for those who have limited internet access.

3.2. UI/UX Improvements

- **Animations and Visualizations**: Introduce good interaction by adding good animations and visual feedback.

3.3. Security Enhancements

- **Two-Factor Authentication**: Install two-factor authentication when a user tries to login to strengthen user account security.
- **HTTPS Integration**: Ensure all client-server communications use HTTPS, encrypting data for safe transmission.

3.4. Analytics and Reporting

- **User Activity Analytics**: Give admin users an analysis of popular songs, albums, and playlists to help find trends and further the content.

3.5. Integration with External Services

- **Platform Integration**: Extend the platform's reach by integrating with services like Spotify and Apple Music.
- **Social Media Login**: Offer users the option to log in via Google, Facebook, or other social platforms, simplifying the registration process and enhancing accessibility.

REFERENCES

UWE Bristol , n.d. *RESTful Web Services,* s.l.: Provided by instructor.

UWE Bristol, n.d. *Advanced HTML,* s.l.: Provided by instructor.

UWE Bristol, n.d. *Intro to NodeJS with MySQL,* s.l.: Provided by instructor.

UWE Bristol, n.d. *JavaScript,* s.l.: Provided by instructor.

UWE Bristol, n.d. *Microservices Architecture,* s.l.: Provided by instructor.

UWE Bristol, n.d. *Software Patterns & Web Oriented Architecture,* s.l.: Provided by instructor.

ARAVINDA A KUMAR and Divya TL, 2024. Security measures implemented in RESTful API Development. *Open Access Research Journal of Engineering and Technology*, 7(1), pp.105–112. https://doi.org/10.53022/oarjet.2024.7.1.0042.

Basumatary, B. and Agnihotri, N., 2022. Benefits and Challenges of Using NodeJS. *International Journal of Innovative Research in Computer Science & Technology*, pp.67–70. https://doi.org/10.55524/ijircst.2022.10.3.13.

Brooks, D.R., 2011. Creating a Server-Side Environment with PHP. In: *Guide to HTML, JavaScript and PHP*. London: Springer London. pp.167–186. https://doi.org/10.1007/978-0-85729-449-4_7.

Freeman, A., 2016. Working with Visual Studio Code. In: *Pro ASP.NET Core MVC*. Berkeley, CA: Apress. pp.343–369. https://doi.org/10.1007/978-1-4842-0397-2_13.

Jones, M., Bradley, J. and Sakimura, N., 2015. *JSON Web Token (JWT)*. https://doi.org/10.17487/RFC7519.

Karayiannis, C., 2019. The MySQL Database Server. In: *Web-Based Projects that Rock the Class*. Berkeley, CA: Apress. pp.219–268. https://doi.org/10.1007/978-1-4842-4463-0_6.

Keshari, P., Maurya, P., Kumar, P. and Katiyar, A., 2023. Web Development Using ReactJS. In: *2023 5th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*. IEEE. pp.1571–1575. https://doi.org/10.1109/ICAC3N60023.2023.10541743.

Stamey, J.W. and Saunders, B.T., 2006. Documenting aspect-oriented PHP (AOPHP). In: *Proceedings of the 24th annual ACM international conference on Design of communication*. New York, NY, USA: ACM. pp.210–213. https://doi.org/10.1145/1166324.1166371.

Thakkar, M., 2020. Introducing React.js. In: *Building React Apps with Server-Side Rendering*. Berkeley, CA: Apress. pp.41–91. https://doi.org/10.1007/978-1-4842-5869-9_2.

Tilkov, S. and Vinoski, S., 2010. Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing*, 14(6), pp.80–83. https://doi.org/10.1109/MIC.2010.145.