PROJECT ON

DESIGN AND DEVELOPMENT OF A MUSIC STREAMING WEB APPLICATION (Ash Player)

A THESIS ON MERN STACK ARCHITECTURE

Submitted by

Raihan Habib Akash

664141

Department of Computer Science and Technology

Faculty of Diploma in Engineering

Daffodil Technical Institute

January 2026

DECLARATION

I hereby declare that the project entitled **“DESIGN AND DEVELOPMENT OF A MUSIC STREAMING WEB APPLICATION USING MERN STACK ARCHITECTURE”**, submitted in partial fulfillment of the requirements for the degree of **Diploma in Computer Science and Technology** at **Daffodil Technical Institute**, is an authentic record of my own work carried out under the supervision of **MD. Noman Jahan**. It is also declared that this project or any part of it has not been submitted elsewhere for the award of any degree or diploma.

_____

Raihan Habib Akash

Approval

The project titled “DESIGN AND DEVELOPMENT OF A MUSIC STREAMING WEB APPLICATION USING MERN STACK ARCHITECTURE” submitted by Raihan Habib Akash (664141) has been accepted satisfactorily in partial fulfillment of the requirement for the degree of Diploma in Computer Science and Technology.

DTI Board of Examiners

_____

MD. Noman Jahan

Department of Computer Science and Technology — Instructor

Daffodil Technical Institute — (Supervisor)

_____

Tasfina Haque — Instructor

Daffodil Technical Institute

Acknowledgement

I would like to express my sincere gratitude and respect to my supervisor, MD. Noman Jahan, for his invaluable guidance, constant encouragement, and insightful feedback throughout the development of this project. His expertise and patience have been instrumental in shaping the technical core of this work. I extend my heartfelt appreciation to the faculty and staff of Daffodil Technical Institute for providing the necessary resources and an outstanding learning environment required to complete my diploma. I am deeply grateful to my teachers, who played a pivotal role in shaping my academic journey, and to the institute's administration for their kind support and encouragement during the execution of this project. My deepest gratitude goes to my parents and family for their unwavering support, sacrifices, and motivation, which have brought me to this present moment. Lastly, I express my sincere thanks to my friends and all well wishers for their moral support and inspiration during this academic journey.

ABSTRACT

In the modern digital era, music streaming platforms have become an integral part of everyday entertainment. Traditional media-based music systems lack accessibility, personalization, and scalability. To overcome these limitations, web-based music streaming applications provide users with on-demand access to audio content anytime and anywhere.

This thesis presents the **design and development of a Spotify-like music streaming web application using the MERN stack**, which consists of **MongoDB, Express.js, React.js, and Node.js**. The system follows a client-server architecture where React.js is used for building an interactive user interface, Node.js and Express.js handle backend logic and API services, and MongoDB is used for storing user, song, and playlist data.

The application supports user authentication, song streaming, playlist management, and role-based access control. The project follows the **System Development Life Cycle (SDLC)**, including requirement analysis, system design, implementation, testing, and documentation. The result is a scalable, secure, and user-friendly music streaming platform that demonstrates the effectiveness of the MERN stack in modern web application development.

TABLE OF CONTENTS

CONTENTS

Chapter 4: Implementation and Testing

Chapter 5: Conclusion and Future Work

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1:

INTRODUCTION

1.1 Introduction

The rapid advancement of information technology has significantly transformed the way digital content is created, distributed, and consumed. In recent years, web-based applications have become the backbone of modern digital services, providing scalable, interactive, and user-centric solutions across various domains. Among these services, **online music streaming platforms** have emerged as one of the most influential digital innovations, reshaping how users access and experience music.

Modern music streaming applications function as complex software systems that integrate user interfaces, backend services, databases, and media delivery mechanisms. These platforms must handle large volumes of data, provide real-time interactions, ensure secure user authentication, and deliver uninterrupted audio playback. Consequently, the design of a robust and scalable web application architecture has become a critical requirement for delivering a high-quality user experience.

This thesis focuses on the **design and development of a music streaming web application (Spotify Clone)** using the **MERN stack**, which consists of **MongoDB, Express.js, React.js, and Node.js**. The application is designed to provide users with functionalities such as user registration, secure login, music browsing, audio playback, and playlist management. The necessity of this project arises from the limitations of traditional music systems, such as offline storage dependency, lack of personalization, and poor scalability. By adopting a modern full-stack JavaScript architecture, this project aims to demonstrate an efficient, modular, and scalable solution for online music streaming.

1.2 Inspiration

The inspiration for this project originates from the widespread adoption of popular music streaming platforms such as Spotify and Apple Music, which have set new standards for digital media consumption. These platforms showcase how modern web technologies can be utilized to deliver seamless user experiences, personalized content, and real-time interactions.

Additionally, the growing demand for **full-stack web developers** and the increasing industry adoption of JavaScript-based technologies motivated the selection of the MERN stack for this project. Developing a Spotify Clone provides a practical opportunity to apply theoretical knowledge of web development, databases, APIs, and authentication mechanisms in a real-world scenario. The goal of simulating a professional, production-level web application environment serves as the driving force behind this project.

1.3 Objectives

The primary objective of this research is to design, develop, and validate a full-stack music streaming web application using the MERN stack. The specific objectives are as follows:

1. **System Architecture Design:**
   To design a scalable client–server architecture using React.js for the frontend, Node.js and Express.js for backend services, and MongoDB for database management.
2. **User Authentication and Authorization:**
   To implement a secure user authentication system using JSON Web Tokens (JWT) to manage user sessions and protect application resources.
3. **Music Streaming and Playlist Management:**
   To enable users to browse songs, play audio content, and create personalized playlists within the application.
4. **RESTful API Development:**
   To develop RESTful APIs that facilitate efficient data communication between the frontend and backend components.
5. **Security and Performance:**
   To ensure secure data handling, role-based access control, and optimized system performance suitable for real-world usage.

1.4 Project Schedule

The successful execution of this project followed the **Project Management Life Cycle**, ensuring a structured and systematic development process from requirement analysis to final testing and

documentation. The timeline below outlines the allocation of time for each phase of the project.

Phase	Activities	Duration (Weeks)	Total Weeks
Initiation	Idea generation, requirement gathering	Week 1	1
Analysis	System analysis, feasibility study	Week 2-3	2
Design	System architecture, database design, UI design	Week 4	1
Development	Frontend and backend development, API implementation	Week 5–12	7
Implementation	Authentication, media handling, playlist features	Week 13	1
Testing	Unit testing, integration testing, system testing	Week 14–15	2
Documentation	Final report writing and result analysis	Week 16-17	2

TABLE 1

1.5 Expected Outcome

Upon completion of this thesis project, the expected outcome is a fully functional and responsive **music streaming web application** that demonstrates reliability, security, and usability. Specifically, the system is expected to deliver:

1. **Seamless User Interaction:**
   Users will be able to register, log in securely, browse music, and play audio content without interruption through an intuitive web interface.
2. **Secure Authentication System:**
   The implementation of JWT-based authentication will ensure secure access control and protection of user data.
3. **Efficient Data Management:**
   MongoDB will store and manage user information, songs, and playlists efficiently, supporting scalability and fast data retrieval.
4. **Smooth Client–Server Communication:**
   RESTful APIs will enable reliable and structured data exchange between the frontend and backend components.
5. **Comprehensive Documentation:**
   The project will include complete documentation, system diagrams, and testing results that can serve as a reference model for future web-based music streaming applications.

CHAPTER 2:

LITERATURE REVIEW

2.1 Introduction

The development of modern web-based applications is grounded in a comprehensive body of knowledge encompassing software engineering principles, web technologies, database systems, and client–server architectures. This chapter presents a critical review of existing literature related to **music streaming platforms**, **full-stack web development**, and **system development methodologies**. It establishes the theoretical foundation for the design decisions adopted in this project by analyzing comparative studies of technology stacks, evaluating feasibility aspects, and reviewing standardized development methodologies applicable to large-scale web applications.

2.2 Comparative Studies

A critical component of web application development is the selection of appropriate technologies and architectural patterns. The choice of the **MERN stack** over alternative stacks such as **LAMP** or **MEAN**, and the adoption of a **client–server architecture** instead of monolithic systems, requires a comparative analysis.

MERN Stack vs Other Technology Stacks

• **MERN Stack (MongoDB, Express.js, React.js, Node.js):**
The MERN stack is a JavaScript-based full-stack framework that allows developers to use a single programming language across both frontend and backend development. React.js enables component-based UI development, Node.js provides non-blocking asynchronous execution, and MongoDB supports flexible document-based data storage. This stack is widely adopted for scalable and real-time applications.

• **LAMP Stack (Linux, Apache, MySQL, PHP):**
The LAMP stack is a traditional web development framework known for its stability. However, it lacks real-time capabilities and requires multiple languages, increasing development complexity.

• **MEAN Stack (MongoDB, Express.js, Angular, Node.js):**
Although similar to MERN, Angular introduces a steeper learning curve compared to React and is less flexible for rapid UI customization.

Conclusion:
For a dynamic and interactive music streaming platform, MERN offers the optimal balance of scalability, performance, and development efficiency.

2.3 Feasibility Studies

A feasibility analysis was conducted to evaluate the practicality of the proposed Spotify Clone system across five dimensions.

2.3.1 Technical Feasibility

• **Frontend:** React.js supports responsive UI design, reusable components, and efficient state management.
• **Backend:** Node.js and Express.js enable scalable RESTful API development.
• **Database:** MongoDB efficiently manages user data, playlists, and song metadata.

Conclusion:
The project is technically feasible as all technologies are mature, widely supported, and suitable for music streaming applications.

2.3.2 Operational Feasibility

• **User Interaction:** Users can register, log in, browse songs, and create playlists without requiring technical knowledge.
• **System Management:** Admins can manage songs and users through backend services.

Conclusion:
The system enhances usability while minimizing operational complexity.

2.3.3 Economic Feasibility

• **Cost Efficiency:** The MERN stack is open-source, eliminating licensing costs.
• **Deployment:** Can be hosted on low-cost cloud platforms (Render, Vercel, Railway).

Conclusion:
The project is economically viable for educational and small-scale commercial deployment.

2.3.4 Legal Feasibility

• **Data Privacy:** User credentials are securely handled using JWT and encrypted passwords.
• **Compliance:** The system design follows standard data protection practices.

Conclusion:
The application complies with general software security and privacy standards.

2.3.5 Schedule Feasibility

• **Timeline:** Modular development allows frontend and backend tasks to proceed in parallel.

Conclusion:
The project can be completed within the academic semester timeframe.

2.4 Proposed Main Characteristics of the Project

The application has authentication system. And user can easily play their favourite songs.

2.4.1 User Characteristics

• **General Users (Listeners):**
Access music streaming, playlist creation, and profile management features.

• **Admin Users:**
Manage songs, playlists, and user content through administrative APIs.

2.4.2 System Characteristics

• **Authentication System:** JWT-based secure login system.
• **Streaming Service:** Audio playback with playlist support.
• **Database Management:** Document-based storage using MongoDB.

2.5 Methodology

The project adopts the **System Development Life Cycle (SDLC)** methodology, which is widely used in software engineering for structured system development. The SDLC approach ensures systematic progression through planning, design, implementation, testing, deployment, and maintenance.

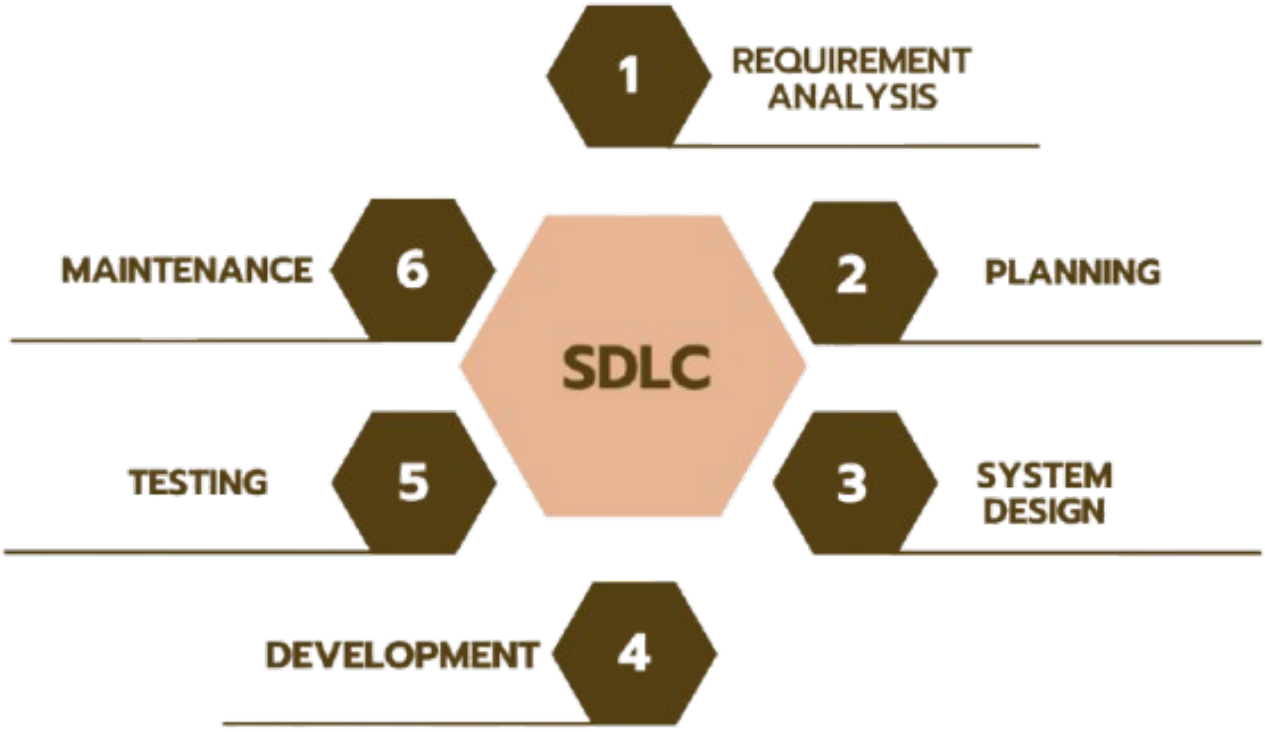


FIGURE 1

2.6 Project Management Life Cycle

The development of this thesis followed the standard Project Management Life Cycle:

- **Initiation:** Defining project scope and requirements.
- **Planning:** Designing system architecture and project timeline.
- **Execution:** Coding frontend and backend components.
- **Monitoring:** Continuous testing and debugging.
- **Closing:** Documentation and final deployment.

2.7 Challenges

Several challenges were encountered during the design and development process:

- **Authentication Security:** Preventing unauthorized access and securing JWT tokens.
- **Media Handling:** Managing audio files efficiently without performance degradation.
- **Scalability:** Ensuring the backend can handle concurrent users.
- **API Performance:** Optimizing database queries for faster response times.

CHAPTER 3:

SYSTEM DESIGN AND ANALYSIS

3.1 Introduction

This chapter presents the system design and analytical overview of the Ash Player web application developed using the MERN stack. It describes how the system is structured, how different users interact with it, and how data flows through the application. Standard software engineering tools such as Use Case Diagrams, System Architecture Diagrams, Data Flow Diagrams (DFD), and Activity Diagrams are used to clearly illustrate the internal working mechanisms of the system. The purpose of this chapter is to provide a clear and logical understanding of the system before moving to implementation and testing.

3.2 Use Case Modelling and Description

Use case modeling is employed to identify the functional requirements of the system from the user's perspective. It defines the interaction between the users (actors) and the system. The Spotify Clone system involves three primary actors: **User (Listener)**, **Admin (Content Manager)**, and **System Administrator**. Each actor interacts with the system in a distinct manner based on assigned responsibilities.

3.2.1 User (Listener) Use Case

The User (Listener) represents the general users of the music streaming platform. These users access the system to listen to music and manage playlists.

Use Case Description:

Attribute	Description
Actor	User

Description	Streams music and manages playlists
Preconditions	User must be registered and logged in
Main Flow	Login → Browse Songs → Play Music → Create Playlist

Postconditions Music is streamed successfully

The listener can browse available songs, play audio content, and create or modify playlists. Authentication ensures that only authorized users can access personalized features.

3.2.2 Admin (Content Manager) Use Case

The Admin (Content Manager) is responsible for managing the music content available in the system.

Use Case Description:

Attribute	**Description**
Actor	Admin (Content Manager)
Description	Manages music content
Preconditions	Admin login required
Main Flow	Login → Upload Songs → Update/Delete Songs

The admin uploads audio files, updates song information, and removes outdated content to maintain the quality of the platform

---

3.3 System Architecture (MERN Stack)

The Ash Player system follows a **client–server architecture** based on the MERN stack. The frontend is developed using React.js, which interacts with backend services built with Node.js and Express.js through RESTful APIs. MongoDB is used as the database to store user data, song metadata, and playlists.

The architecture ensures modularity, scalability, and efficient data communication between system components.

3.3.1 Application Architecture and Technology Stack

The system is composed of three primary layers:

Frontend Layer

- Developed using **React.js**
- Provides user interface for music browsing and playback
- Communicates with backend using HTTP requests

Backend Layer

- Built with **Node.js and Express.js**
- Handles authentication, API logic, and business rules
- Implements Clerk for secure authorization

Database Layer

- Uses **MongoDB**
- Stores users, songs, playlists, and authentication data
- Uses Mongoose for schema management

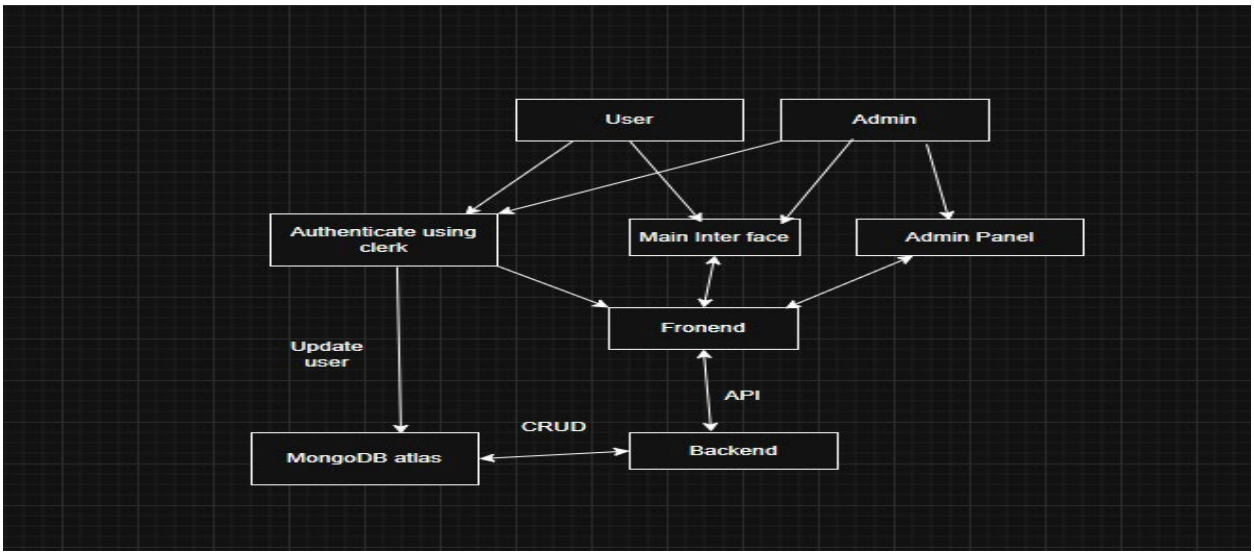
This layered architecture improves maintainability and supports future enhancements.

3.4 Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) illustrates how data moves through the Spotify Clone system. It represents the flow of information between users, processes, and data storage.

DFD Overview:

- The user sends a login or music request to the system.
- The backend verifies credentials using the database.
- Upon validation, music data is retrieved and sent back to the user.
- Playlist updates are stored in the database.

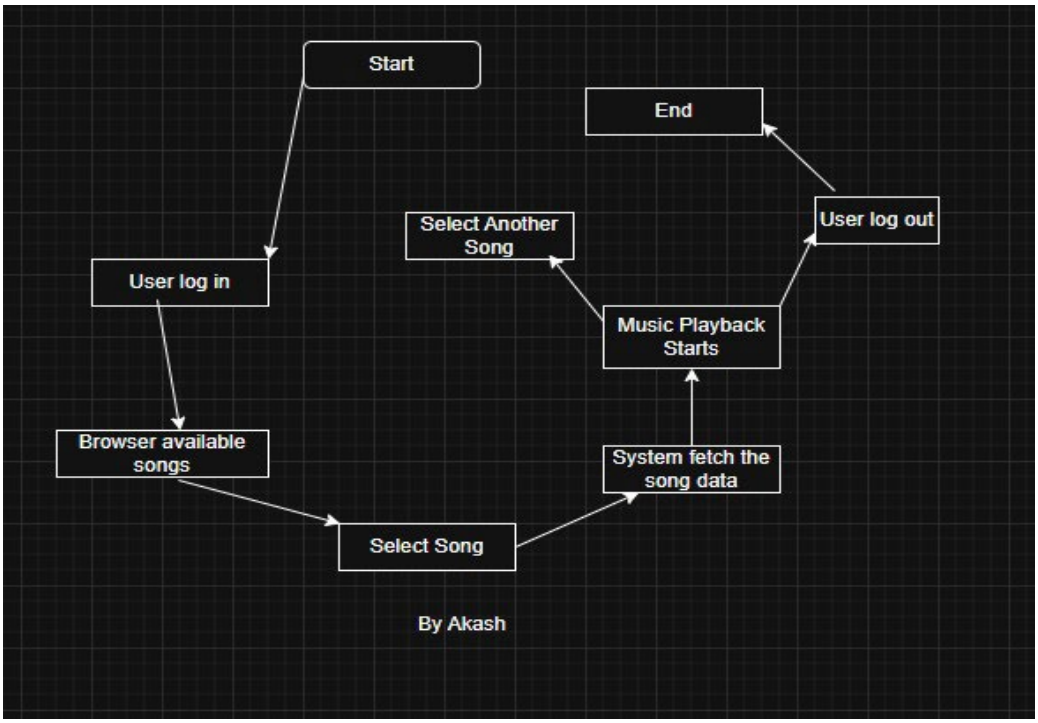


The DFD ensures clarity in understanding system data interactions and processing flow.

3.5 Activity Diagram

The activity diagram demonstrates the step-by-step workflow of user interaction within the system. A typical activity flow for music playback includes:

1. User logs into the system
2. User browses available songs
3. User selects a song
4. System fetches song data
5. Music playback starts
6. User logs out or selects another song



This diagram helps visualize the sequential operations and decision-making process within the application.

CHAPTER 4:

IMPLEMENTATION AND TESTING

4.1 Introduction

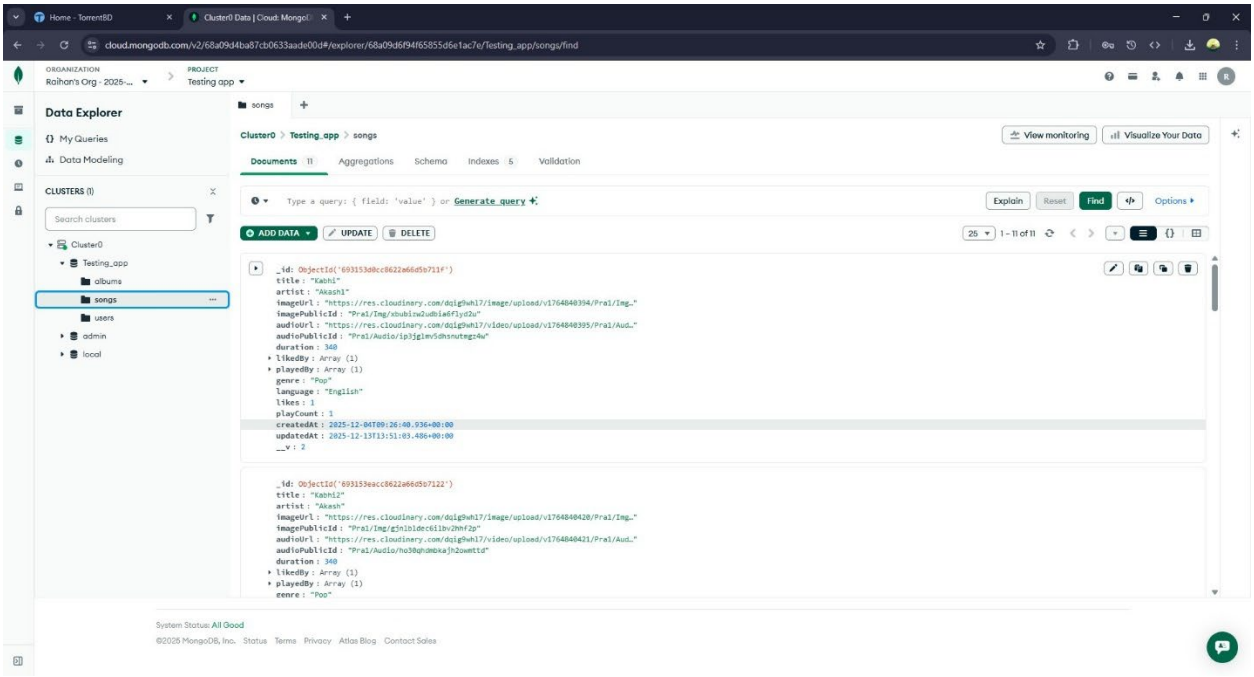
This chapter describes the implementation details and testing procedures of the Spotify Clone web application developed using the MERN stack. It explains how the frontend, backend, database, and authentication mechanisms were implemented and integrated. Additionally, this chapter discusses various testing strategies used to verify system functionality, performance, and reliability.

4.2 Database Implementation (MongoDB)

MongoDB is used as the primary database for storing application data due to its flexibility and scalability. The database stores user information, song metadata, and playlist data in a document-oriented format. Mongoose is used as an Object Data Modeling (ODM) library to define schemas and manage database operations.

Collections Implemented

- **Users Collection:** Stores Clerk User ID and basic profile information.
- **Songs Collection:** Stores song title, artist, duration, and audio URL.
- **Playlists Collection:** Stores user-created playlists and associated songs.



This database structure ensures efficient data retrieval and supports future system expansion.

4.3 Front-end Design (React.js User Interface)

The frontend of the application is developed using **React.js**, providing a dynamic and responsive user interface. The application includes key pages such as Login, Home, Music Player, and Playlist Management.

Clerk's React SDK is integrated to handle user authentication, allowing users to register and log in securely. State management is handled using React hooks, and API communication is performed using Axios.

4.4 Back-end Development (Node.js & Express.js)

The backend server is developed using **Node.js and Express.js**, exposing RESTful APIs for handling application logic. The backend processes requests related to music retrieval, playlist management, and user-specific data.

4.4.1 Authentication and Authorization (Clerk)

Authentication is managed using **Clerk**, which verifies user identity and issues JWT tokens. The backend uses Clerk middleware to validate tokens for protected API routes, ensuring that only authenticated users can access sensitive resources.

4.4.2 REST API Implementation

RESTful APIs are implemented to support communication between the frontend and backend.

Key API Endpoints:

- /api/songs/featured – Fetch random songs
- /api/songs/made-for-you – Fetch Matched genre songs
- /api/songs/trending – Fetch Trending Songs
- /api/albums – Create and manage playlists
- /api/user – Retrieve user-related data

These APIs ensure structured and secure data exchange.

4.4.3 Music Streaming and Playlist Management

Music playback is implemented by streaming audio files using secure URLs. Users can play songs, pause playback, and manage playlists. Playlist data is stored in MongoDB and linked to users using the Clerk User ID.

4.5 Unit Testing

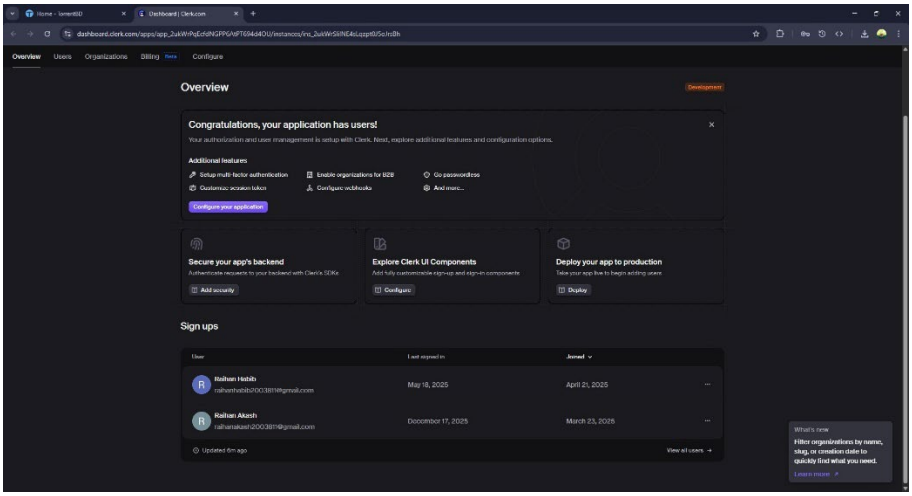
Unit testing is performed to verify the correctness of individual components and functions. Frontend components are tested to ensure proper rendering, while backend API endpoints are tested using tools such as Postman.

Unit tests confirm that each module works independently as expected.

4.6 Integration Testing

Integration testing ensures proper communication between the frontend, backend, authentication service, and database. API requests are tested to verify that authenticated users can successfully retrieve and manipulate data.

This testing phase confirms that Clerk authentication integrates correctly with backend authorization logic.



4.7 System Testing

System testing evaluates the complete application as a whole in a real-world usage scenario.

4.7.1 User Functionality Testing Results

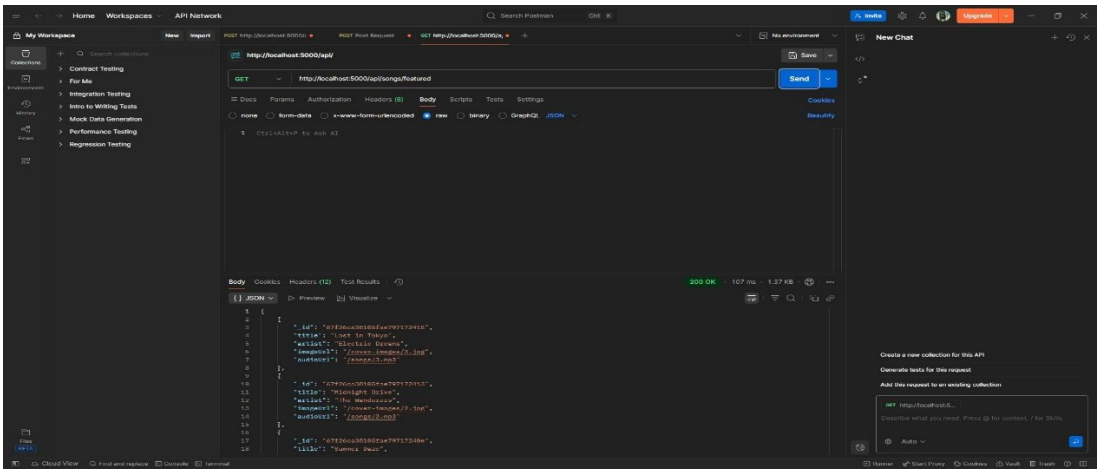
Tests confirm that users can log in, browse songs, play music, and create playlists without errors.

4.7.2 Authentication and Security Testing Results

Authentication testing verifies that unauthorized users cannot access protected routes. Clerk successfully manages session handling and token validation.

4.7.3 Performance and API Testing Results

Performance testing ensures acceptable response times for API requests and smooth music playback under normal load conditions.



4.8 Acceptance Testing

Acceptance testing is conducted to validate that the system meets all specified requirements. The application is tested from an end-user perspective to confirm usability, functionality, and reliability.

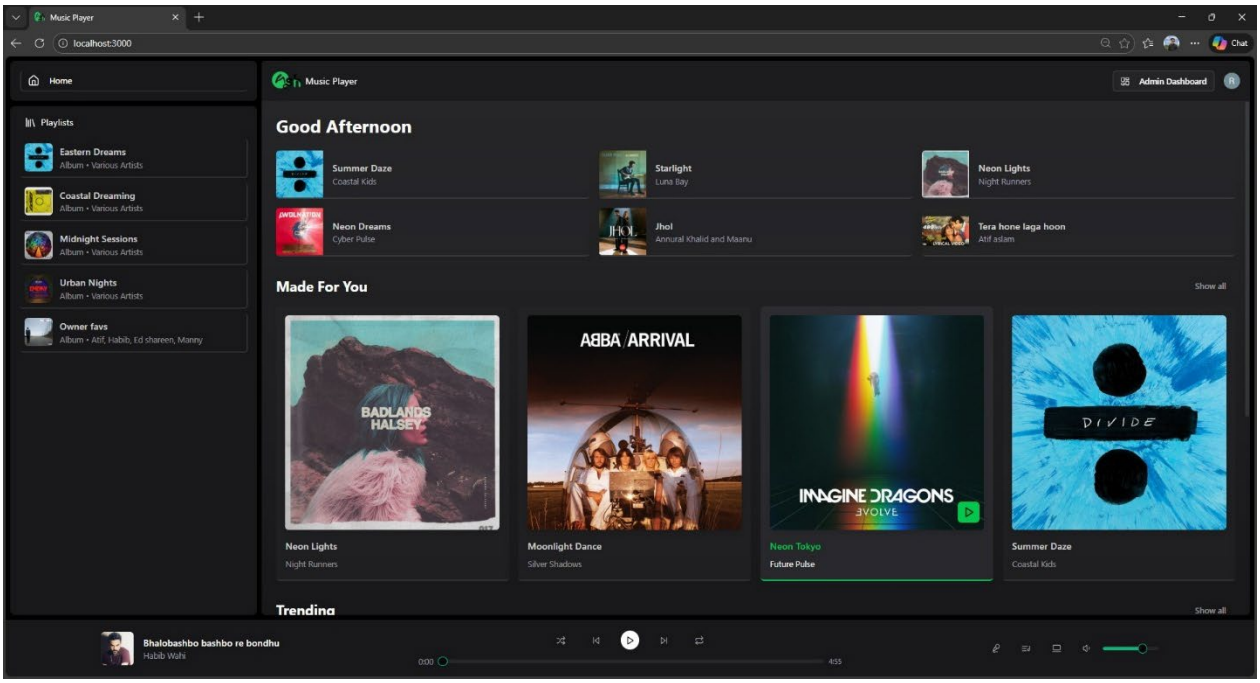
All defined objectives are satisfied, and the system is deemed ready for deployment.

4.9 Snapshot of Final Web Application

This section presents screenshots of the final Ash Player web application, including the Home Page, Music Player Interface, and Playlist Management Screen. These snapshots demonstrate the successful implementation of the system.

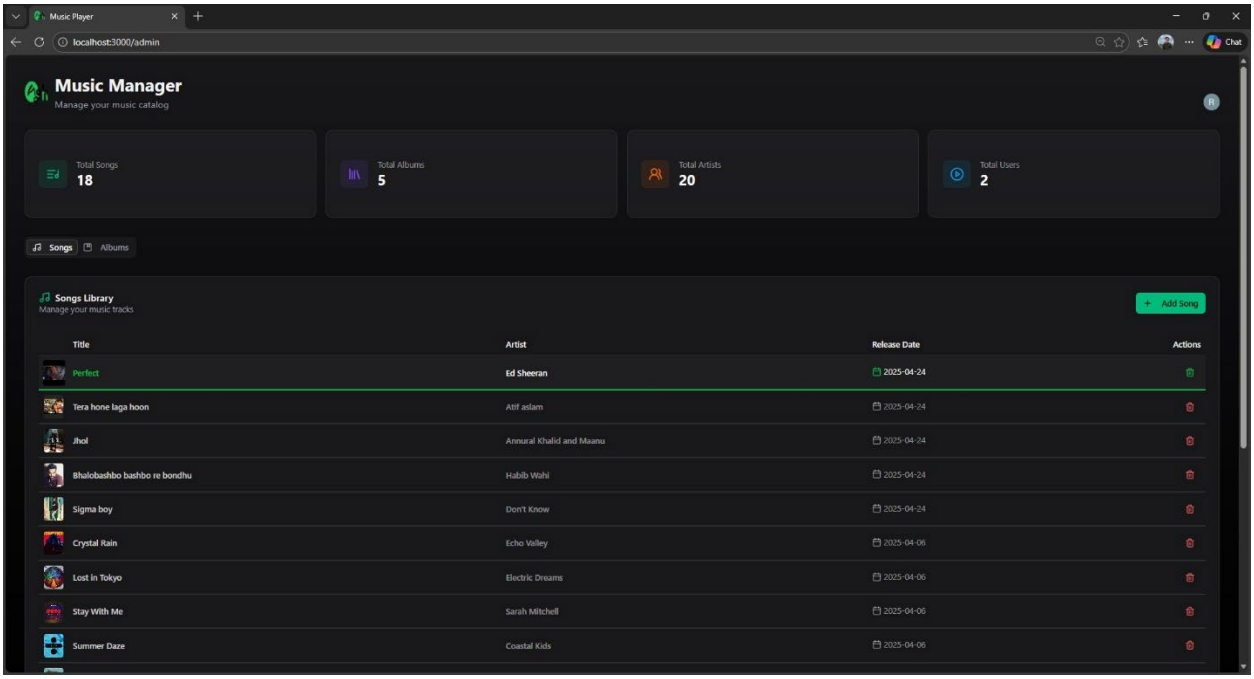
4.9.1 User interface

This part is only for the users. User can authenticate using clerk, player some music, add playlist for their own.



4.9.1 Admin Panel

This part is for admin. Admin can add songs, remove songs, remove user, Check songs list.



CHAPTER 5:

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This thesis presented the design, development, and evaluation of a music streaming web application (Ash Player) using the MERN stack integrated with Clerk authentication. The primary objective of the project was to build a secure, scalable, and user-friendly web-based music streaming platform that demonstrates the practical application of modern full-stack web development technologies.

Throughout the project, the system was designed following standard software engineering principles and methodologies. The frontend was implemented using React.js to ensure an interactive and responsive user experience, while the backend was developed using Node.js and Express.js to handle application logic and API services. MongoDB was used as the database to store user data, song metadata, and playlists efficiently. Clerk authentication successfully handled user registration, login, and session management, ensuring secure access control without manual password handling.

Comprehensive testing, including unit testing, integration testing, system testing, and acceptance testing, confirmed that the application meets all functional and non-functional requirements. Users can securely log in, browse songs, play music, and manage playlists without errors. The system architecture proved to be modular, maintainable, and suitable for future expansion.

Overall, the project successfully demonstrates how a modern music streaming platform can be developed using the MERN stack with third-party authentication services. The outcomes of this thesis meet the stated objectives and provide a solid foundation for further development and real-world deployment.

5.2 Future Work

Although the current implementation of the Spotify Clone fulfills its core objectives, several enhancements can be introduced in future versions to improve functionality, scalability, and user experience:

1. **Recommendation System:**
   A machine learning–based recommendation engine can be integrated to suggest songs based on user preferences and listening history.
2. **Advanced Search and Filters:**
   Features such as artist-based search, genre filtering, and mood-based playlists can enhance content discoverability.

3. **Offline Playback Support:**
   Implementing offline music caching would allow users to listen to songs without an active internet connection.

4. **Scalability and Cloud Deployment:**
   The application can be deployed on cloud platforms with load balancing and CDN support to handle a larger number of concurrent users.

5. **Mobile Application Integration:**
   A mobile version of the application using React Native can be developed to extend accessibility across devices.

6. **Enhanced Admin Dashboard:**
   Advanced analytics and reporting tools can be added for admins to monitor user activity and content performance.

These future enhancements would significantly improve the system's capabilities and align it more closely with commercial-grade music streaming platforms.

REFERENCES

1. **Ash Player Project Documentation (Unpublished Academic Project Report).**
2. **MongoDB Documentation – Data Modeling, accessed December 20, 2025, https://www.mongodb.com/docs/manual/data-modeling/**
3. **Node.js Official Documentation, accessed December 20, 2025, https://nodejs.org/en/docs**
4. **Express.js Guide – Building RESTful APIs, accessed December 20, 2025, https://expressjs.com/en/guide/routing.html**
5. **React.js Official Documentation – Component-Based Architecture, accessed December 20, 2025, https://react.dev/learn**
6. **Clerk Authentication Documentation, accessed December 20, 2025, https://clerk.com/docs**
7. **REST API Design Best Practices – Red Hat, accessed December 20, 2025, https://www.redhat.com/en/topics/api/what-is-a-rest-api**
8. **MERN Stack Explained – GeeksforGeeks, accessed December 20, 2025, https://www.geeksforgeeks.org/mern-stack/**
9. **Music Streaming System Architecture – ResearchGate, accessed December 20, 2025, https://www.researchgate.net/publication/360215789_Music_Streaming_System_Architecture**
10. **Comparative Study of MEAN and MERN Stack, accessed December 20, 2025, https://www.researchgate.net/publication/349879522_Comparative_Study_of_MEAN_and_MERN_Stack**
11. **Software Engineering Methodologies – SDLC Overview, accessed December 20, 2025, https://www.tutorialspoint.com/sdlc/index.htm**
12. **System Design for Web Applications – Educative, accessed December 20, 2025, https://www.educative.io/courses/grokking-the-system-design-interview**
13. **UML Activity Diagrams – GeeksforGeeks, accessed December 20, 2025, https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/**
14. **Data Flow Diagram (DFD) Guide – Lucidchart, accessed December 20, 2025, https://www.lucidchart.com/pages/data-flow-diagram**
15. **Client–Server Architecture Explained – IBM, accessed December 20, 2025, https://www.ibm.com/topics/client-server**

16. **Web Application Security Best Practices – OWASP, accessed December 20, 2025, https://owasp.org/www-project-top-ten/**

17. **Authentication and Authorization in Web Applications – Okta, accessed December 20, 2025, https://www.okta.com/identity-101/authentication-vs-authorization/**

18. **API Testing with Postman – Official Guide, accessed December 20, 2025, https://learning.postman.com/docs/testing-and-scripts/**

19. **Software Testing Techniques – TutorialsPoint, accessed December 20, 2025, https://www.tutorialspoint.com/software_testing/index.htm**

20. **System Testing and Acceptance Testing – GeeksforGeeks, accessed December 20, 2025, https://www.geeksforgeeks.org/software-testing-types/**

21. **Draw.io UML Diagram Tool Documentation, accessed December 20, 2025, https://www.diagrams.net/doc/**

22. **Canva Diagram Creation Guide, accessed December 20, 2025, https://www.canva.com/graphs/**

23. **Performance Optimization in React Applications – React Docs, accessed December 20, 2025, https://react.dev/learn/render-and-commit**

24. **Node.js Performance Best Practices – NodeSource, accessed December 20, 2025, https://nodesource.com/blog/nodejs-best-practices/**