TRIBHUVAN UNIVERSITY

St.Lawrence College

Chabahil, Kathmandu, Nepal

Project Report

On

Hamro Cinema:"Online Movie Ticket Booking System"

[CSC-412]

A project report submitted for the partial fulfillment of the requirement for the degree of **Bachelors of Science in Computer Science & Information Technology** awarded by Tribhuvan University.

Submitted by:

Shushovit Jung Rana (28623/078)

Diwash Acharya (28606/078)

Sandip Koirala (28621/078)

Submitted To

St.Lawrence College

Under the Supervision of

Mr. Basanta Gyawali

Department of Computer Science and

Information Technology

Affiliated to Tribhuvan University

Chabahil, Kathmandu

September, 2025

Tribhuvan University

Institute of Science and Technology

St. Lawrence College

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by Shushovit Jung Rana, Diwash Acharya and Sandip Koirala entitled "ONLINE MOVIE TICKET BOOKING SYSTEM" in partial fulfillment of the requirements for the degree of Bachelor of Computer Science and Information Technology is recommended for the final evaluation.

_____

Basanta Gyawali

CSIT Department

St. Lawrence College

Chabahil-7, Kathmandu 44600

Tribhuvan University

Institute of Science and Technology

St. Lawrence College

LETTER OF APPROVAL

This is to certify that this project prepared by Shushovit Jung Rana, Diwash Acharya, and Sandip Koirala entitled "ONLINE MOVIE TICEKT BOOKING SYSTEM" in partial fulfillment of the requirements for the degree of Bachelors in Computer Science and Information Technology has been evaluated. In our opinion it is satisfactory in scope and quality as a project for the required degree.

…… SIGNATURE of Supervisor Basanta Gyawali CSIT Department St. Lawrence College Chabahil-7, Kathmandu 44600	…… SIGNATURE of HOD/Coordinator
…… SIGNATURE of Internal Examiner Internal Examiner	…… SIGNATURE of External Examiner External Examiner

ABSTRACT

This project report presents the development and implementation of a movie booking website using MERN stack. The website is designed to offer users a seamless experience for browsing and booking movie tickets online. The project follows the Agile methodology, ensuring iterative development and continuous improvement based on user feedback. Key outcomes include a robust understanding of full-stack development, effective use of NoSQL databases, and the application of best practices in user experience design and security. The project concludes with recommendations for future enhancements, such as scalability improvements, advanced security features, and mobile optimization. Overall, the movie booking website demonstrates the team's capability to deliver a functional and user-friendly application, laying a solid foundation for future projects in web development.

Keywords: HTML, CSS, React, Javascript, Express, MongoDB

ACKNOWLDGEMENT

This project is prepared in partial fulfilment of the requirement of 7th Semester Project. The satisfaction and success of completion of this task would be incomplete without heartfelt thanks to people whose constant guidance, support, and encouragement made this work successful. In doing this undergraduate project we have been fortunate to have help, support, and encouragement from many people. We would like to acknowledge them for their cooperation.

We would like to thank St. Lawrence College for providing us a huge platform to gain knowledge about different aspects of Computer Science and Information Technology.

We would like to thank our Supervisor Mr. Basanta Gyawali who persuaded and continuously guided us during the whole course of our project. We would also like to thank him for providing us with the necessary content and classes regarding the project. We are also grateful to Mr. Basanta Gyawali for acting as our supervisor and showing immense patience and understanding throughout the project and provided suggestions.

Last, but not least, we also take this opportunity to thank our friends and colleagues for their support and feedback throughout this project.

With Regards,

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

HTML:	HyperText Markup Language
CSS:	Cascading Style Sheet
JS:	Javascript
UI:	User Interface
UX:	User Experience
DFD:	Data Flow Diagram
MERN:	MongoDB, Express js, React js, Node js
SQL:	Structured Query Language
NoSQL:	Non-SQL or Not Only SQL
API:	Application Programming Interface

LIST OF TABLES

TABLE OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Introduction

Gone are the days when customers had to check the movies in the newspaper ads and stand in long queues to purchase tickets. Now, it’s at their fingertip where they can book anytime and anywhere they want [1].

Cinema ticketing has come a long way. Initially, customers had to line up at the box office to check showtimes and purchase tickets. With the advent of online booking systems ,customers can now browse movies and book tickets online with ease, cinemas have access to data to optimize operations and marketing, Mobile apps provide convenience for on-the-go ticket purchases

The digital transformation has opened up opportunities for cinemas to better understand customer preferences and boost ticket sales. As technology continues to advance, so will innovation in cinema ticketing systems.

Whether it is booking or even buying a plane ticket, bus or a movie ticket it has been very easy to do so using the internet. Similarly our group project is a Movie Ticke Booking System (Website) developed using MERN stack. The purpose of this project is to create a user-friendly platform where users can browse movies and book tickets conveniently online. This application leverages modern web technologies to deliver a seamless user experience, ensuring that users can search for their favorite movies, view detailed information, and reserve seats effortlessly. The website's back-end is built with Express, a robust Node.js framework. MongoDB is used as the database to manage and store user information. The front-end is designed to be responsive and intuitive, allowing users to navigate the site with ease on both desktop and mobile devices.

1.2 Problem Statement

Movie enthusiasts often face challenges in booking tickets, especially when relying on traditional methods like in-person box offices or third-party booking services, which may involve extra fees or limited availability. The lack of a centralized and efficient system for browsing movies, selecting preferred showtimes, and reserving seats can lead to inconvenience and frustration for users. Additionally, cinema operators face difficulties in managing seat availability, tracking bookings, and updating showtime information in real-time. Existing solutions often lack seamless integration between the user interface and back-end management, resulting in a suboptimal experience for both users and administrators. To address these issues, our project aims to develop a Movie Ticketing

Website that offers a streamlined and user-friendly platform for both moviegoers and cinema administrators. The solution provides real-time updates, easy navigation, and a straightforward booking process, enhancing the overall movie-going experience. This problem statement outlines the issues your project seeks to solve, emphasizing the need for a more efficient and integrated movie booking system.

1.3 Objectives

- To enable users to select seats, confirm bookings, view show details.
- To enable admin to add, delete, edit movie details and view total bookings, users along with total earnings.
- To implement a recommendation system that recommend movies to users that they prefer according to their booking history.

These objectives clearly define the goals of our project, covering both the technical and user-centric aspects of the movie booking website.

1.4 Scopes

- Implementation of a secure user registration and login system.
- User profiles with basic information and booking history.
- Display of current and upcoming movies.
- Display of available showtimes for each movie.
- Display recommendations to user.
- Interactive seat selection for users to choose preferred seats.
- Integration of a booking system allowing users to reserve and purchase tickets online.
- Utilization of MongoDB to store and manage user profiles

1.5 Limitations

Scalability:

- The design and database structure is suited for small sized theaters.

Security Considerations:

- While basic security measures will be implemented, the system is not fully equipped to handle sophisticated cyber threats without further enhancements.

Single-Language Support:

- The website will support only one language.

Mobile Optimization:

- While the site will be responsive, the mobile experience might lack some of the optimizations found in native apps, such as offline access or push notifications.

1.6 Development Methodology

The Agile development methodology will be employed for this project. Agile is chosen for its iterative and incremental approach, which allows for flexibility, continuous feedback, and improvements throughout the development process.

Project Phases

a. **Requirement Analysis and Planning**:
   - **Requirement Documentation**: Document functional and non-functional requirements.
   - **Project Planning**: Define project scope, timeline, milestones, and resource allocation using MS project.

b. **System Design**:

- **Architectural Design**: Define the overall system architecture, including frontend, backend, and database components.
- **Database Design**: Design the database schema to support the application's data requirements using visio.

c. **Development**:
- **Frontend Development**: Implement the user interface using HTML, CSS, JavaScript, and framework React.
- **Backend Development**: Develop using Node.js with Express with external integration of clerk and inngest for user synchronization and authentication.
- **Database Integration**: Set up and integrate the database MongoDB(MongoDB atlas cloud).

d. **Testing and Quality Assurance**:
- **Unit Testing**: Test individual components and functions manually to ensure they work correctly.
- **System Testing**: Conduct end-to-end testing manually to ensure the entire system works as expected.
- **User Acceptance Testing (UAT)**: Involve end-users in testing to gather feedback and ensure the system meets their needs.

e. **Deployment**:
- **Server Setup**: Set up the hosting environment on services using vercel.

f. **Documentation**:
- **Documentation**: Create comprehensive documentation, including user guides, system architecture details.

1.7 Report Organization

This report is organized into multiple sections to clearly explain the planning, design, development, and implementation of our movie ticket booking website using the MERN (MongoDB, Express, React, Node.js) stack. Below is a brief overview of each section:

- **Chapter 1: Introduction**
  This section introduces the project, including its background, objectives, scope, and the problem it aims to solve.
- **Chapter 2: Background Study and Literature Review**
  Discusses the fundamental theories, general concepts and terminologies

related to the project and literature review for the review of similar projects, theories and results by other researchers.

- **Chapter 3: System Analysis**
  Covers the requirement analysis of the project with feasibility analysis.
- **Chapter 4: System Design**

Contains database design with interface design.

- **Chapter 5: Implementation**
  Describes how the project was developed using the MERN stack, including details of front-end and back-end development, integration, and database management.
- **Chapter 6: Conclusion and Future Enhancements**
  Summarizes the overall development process and highlights key outcomes. It also discusses limitations and possible improvements for future versions.
- **References and Appendices**
  Includes the list of references, code snippets, sample screenshots, and any other supporting documents.

This organization ensures a logical flow and makes it easier for the reader to understand the complete development lifecycle of our project.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

The digital transformation of the entertainment industry has led to a significant shift in how consumers engage with content, particularly movies. With the rise of online streaming services, traditional movie theaters face increased competition, making it crucial to enhance the overall cinema-going experience. One of the key areas where theaters can differentiate themselves is through the convenience and efficiency of their booking systems.

Need for the Project: Despite the availability of advanced online booking platforms, there is a clear need for a customizable, scalable solution that can be tailored to meet the specific needs of small to medium-sized cinemas. Many existing platforms are either too costly or offer features that are irrelevant or overly complex for smaller operations. By building a movie booking website using JavaScript, React, Express, and MongoDB, our project aims to fill this gap, providing an efficient, user-friendly system that enhances the customer experience and streamlines theater operations.

Learning Opportunity: From an educational perspective, this project offers an excellent opportunity to apply theoretical knowledge in a practical setting. It enables us to explore full-stack development, understand RESTful API design, manage a non-relational database, and collaborate effectively as a team.

This background study provides a comprehensive overview of the technological, market, and educational context for your project, demonstrating the relevance and necessity of developing your movie booking website.

2.2 Literature Review

A study highlights that we are witnessing the growing development of web applications that allow users to complete all their tasks with only one click [2]. Several studies highlight the effectiveness of using modern web technologies like JavaScript, Express, and MongoDB for building scalable and responsive web applications. NoSQL databases were developed to fulfill storage requirements in big data environments. In that way, their schema less data structure provides more flexibility to many applications, such as e-mails, documents, and social media content [3]. MongoDB is a cross-platform, document-oriented database that provides high performance and scalability. It works on the concepts of collection and document [4].

The main objective of evaluating user experience is to support and help in selecting the best design, to make sure the development is on right track, or to measure and assess whether the final product meets and comply with the original UX targets [5]. As computers become more powerful, the critical bottleneck in applying computer-based systems to solve problems is more often in the user interface rather than in the computer hardware or software [6]. A study suggests that user interface is arguably the most important element of a computer-based system or product. If the interface is poorly designed, the user's ability to tap the computational power of an application may be severely hindered [7].

MongoDB provides the possibility to store data with a flexible and dynamic schema. This is an advantage over SQL relational databases where you must define and declare the structure of the data prior to inserting it in the database, and where it becomes hard to modify that structure afterwards [8]. Cloud databases have a long list of benefits. Some of their key advantages include scalability, cost, agility, risk reduction, and security [9].

Despite the benefits, online booking systems face challenges such as security concerns, high development costs, and the need for continuous updates to meet user expectations. This underscores the importance of implementing robust security measures, including encryption and secure payment gateways. Additionally, the cost of developing and maintaining such systems can be prohibitive for smaller cinemas, which often lack the resources to invest in complex, feature-rich platforms.

CHAPTER 3: SYSTEM ANALYSIS

3.1 System Analysis

System analysis involves understanding the requirements, breaking them down into manageable parts, and designing a system that meets those needs. This phase includes identifying both functional and non-functional requirements.

3.1.1 Requirement Analysis

i. Functional Requirement

- Users is able to create an account by providing necessary details (e.g., name, password).
- The system provides secure login functionality using email and password.
- The system displays a list of currently available movies along with their details.
- Users are able to select a showtime and view the seating layout for the selected theater.
- Users are able to choose and reserve seats based on availability.
- The system allows users to book tickets for a selected movie, showtime, and seats.
- Users are able to view their upcoming and past bookings within their profile.
- The system encrypts sensitive user data, such as passwords.

Use-Case Diagram

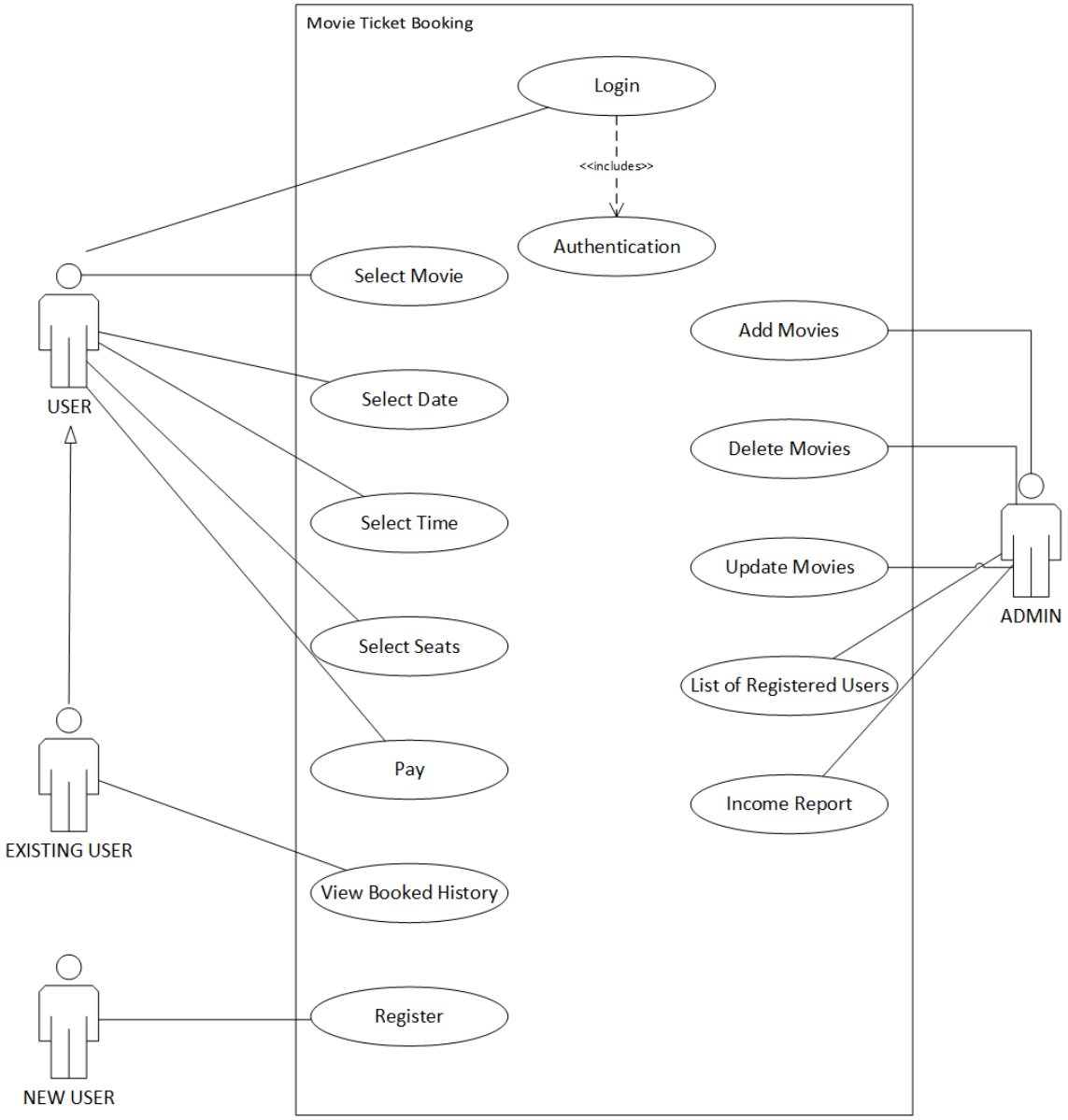

Figure 1 Use Case Diagram

ii. Non-Functional Requirements

- Page load times will not exceed 10 seconds under normal operating conditions.
- The database supports scalability to manage large datasets as the number of users and movies grows.
- The website has a clean, intuitive user interface with a consistent look and feel across all pages.
- The codebase is modular and well-documented, allowing for easy updates and troubleshooting.

- The system uses a version control system (Git) to manage changes and updates.
- The website is compatible with all modern web browsers, including Chrome, Firefox, Safari, and Edge.
- The website is fully responsive, providing a seamless user experience on various devices, including desktops, tablets, and smartphones.
- The mobile interface should be optimized for touch-based navigation, ensuring all features are accessible on smaller screens.

3.1.2 Feasibility Analysis

The feasibility study evaluates the practicality and viability of the developed movie booking website from various perspectives. It examines whether the system meets technical, operational, and economic criteria effectively. Here is how the feasibility study breaks down:

i. Technical Feasibility

- **Technology Stack**:
  - **Designs**: Figma, Visio, Draw.io.
  - **Frontend**: HTML, CSS, JavaScript, and modern frameworks like Tailwind CSS and React.
  - **Backend**: Node.js with Express along with clerk and ingest integration.
  - **Database**: MongoDB
- **Development Tools**:
  - Integrated Development Environments (IDEs) Visual Studio Code.
  - Version control systems like Git with platform GitHub for collaboration.
- **Technical Expertise**:
  - The project team has expertise in web development, database management however the team lacks technical ability in implementing the recommendation system algorithm.

ii. Operational Feasibility

The system operates smoothly, providing seamless experience for both users and administrators. Users find the interface intuitive and easy to navigate, with clear instructions and helpful prompts. Admins manage movie listings, showtimes, and bookings with minimal training, and the system's automation of routine tasks.

iii. Economic Feasibility

Cost Analysis:

- **Infrastructure Costs**: Using free hosting and free domain registration.
- **Software Licenses**: Tools and libraries that are open-source.
- **Marketing Costs**: Promoting the platform to attract users.

iv. Schedule Feasibility

Project Timeline:

- **Requirement Analysis and Planning**: 1 week
- **Design**: 1 week
- **Development and Testing**: 1 month 11 days
- **Total Estimated Time**: Approximately 1 month 25 days

Gantt Chart

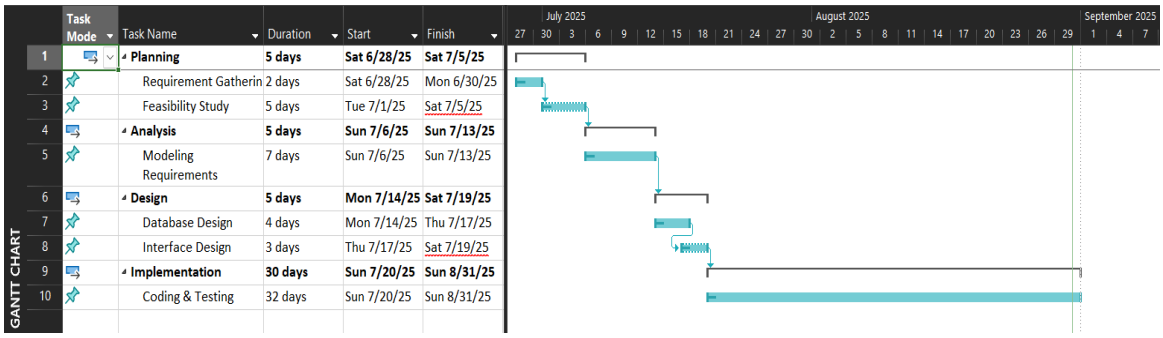

Figure 2 Gantt Chart

3.1.3 Analysis

Data Flow Diagrams

Data Flow Diagram (DFD) of Movie Ticket Booking System consists of level 0 and 1 DFD. Both diagrams show how the data flows inside the system.

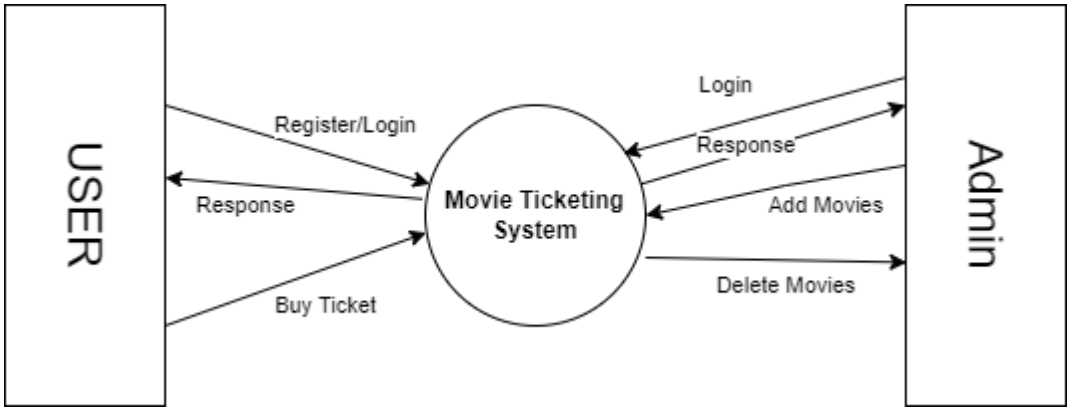

Figure 3 DFD level 0

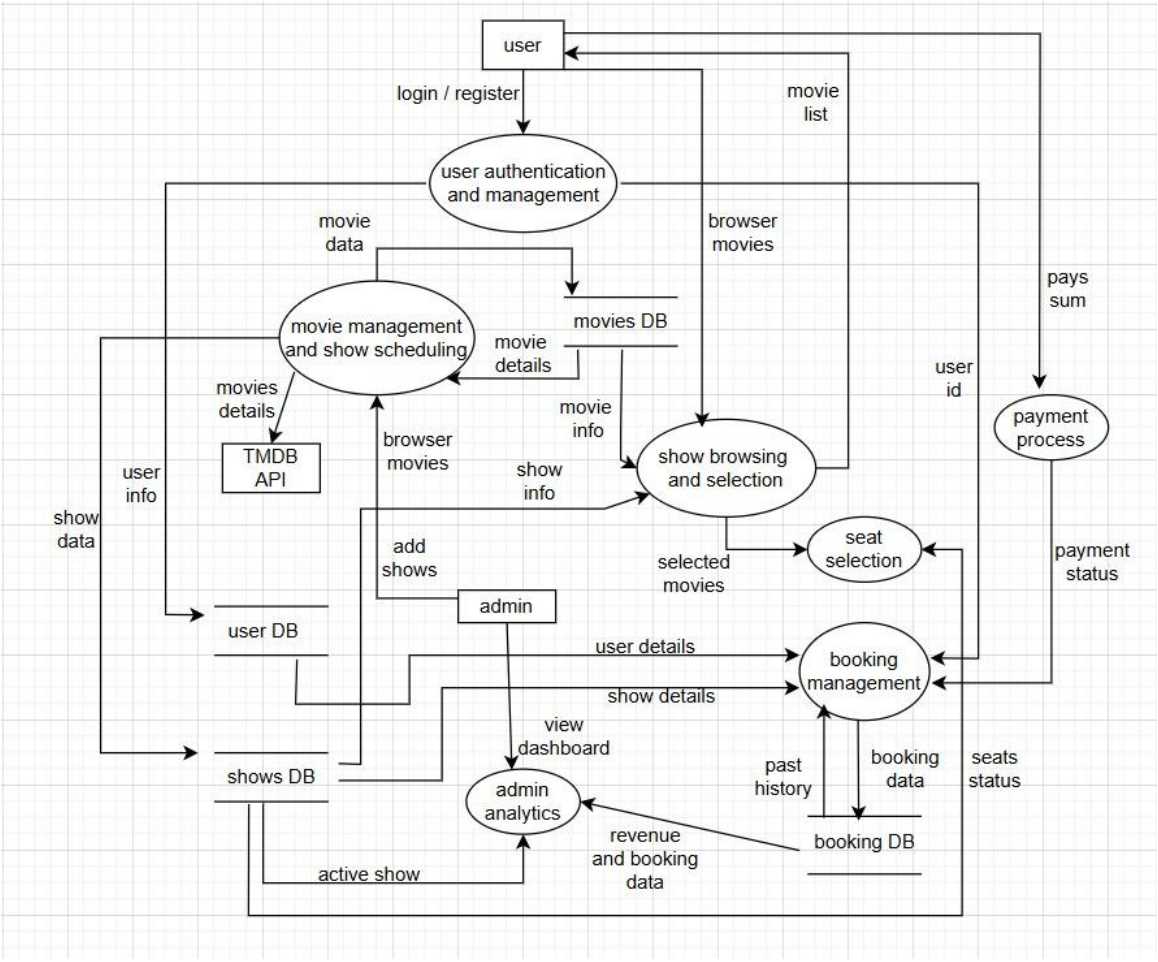

Figure 4 DFD Level 1

Physical DFD

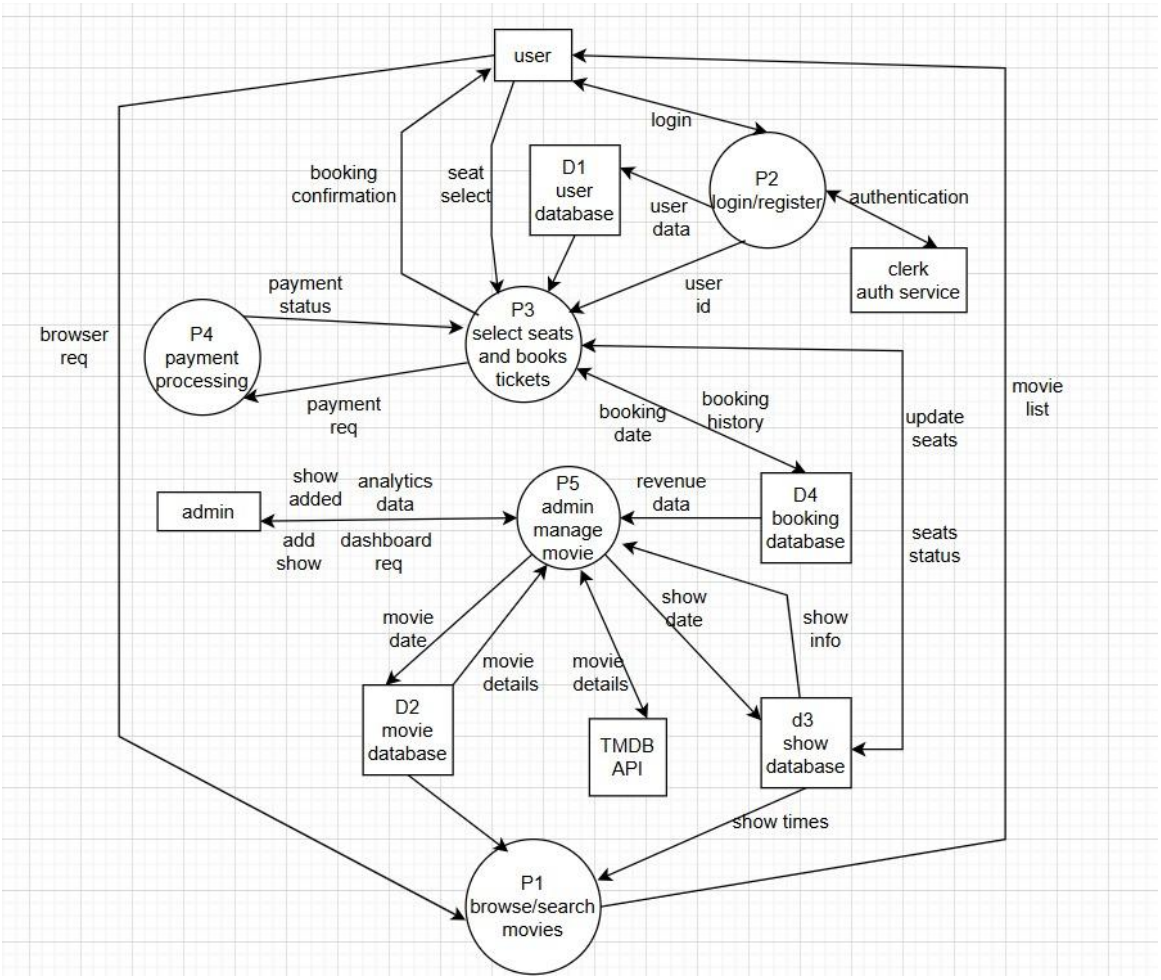

Figure 5 Physical DFD

Class Diagram

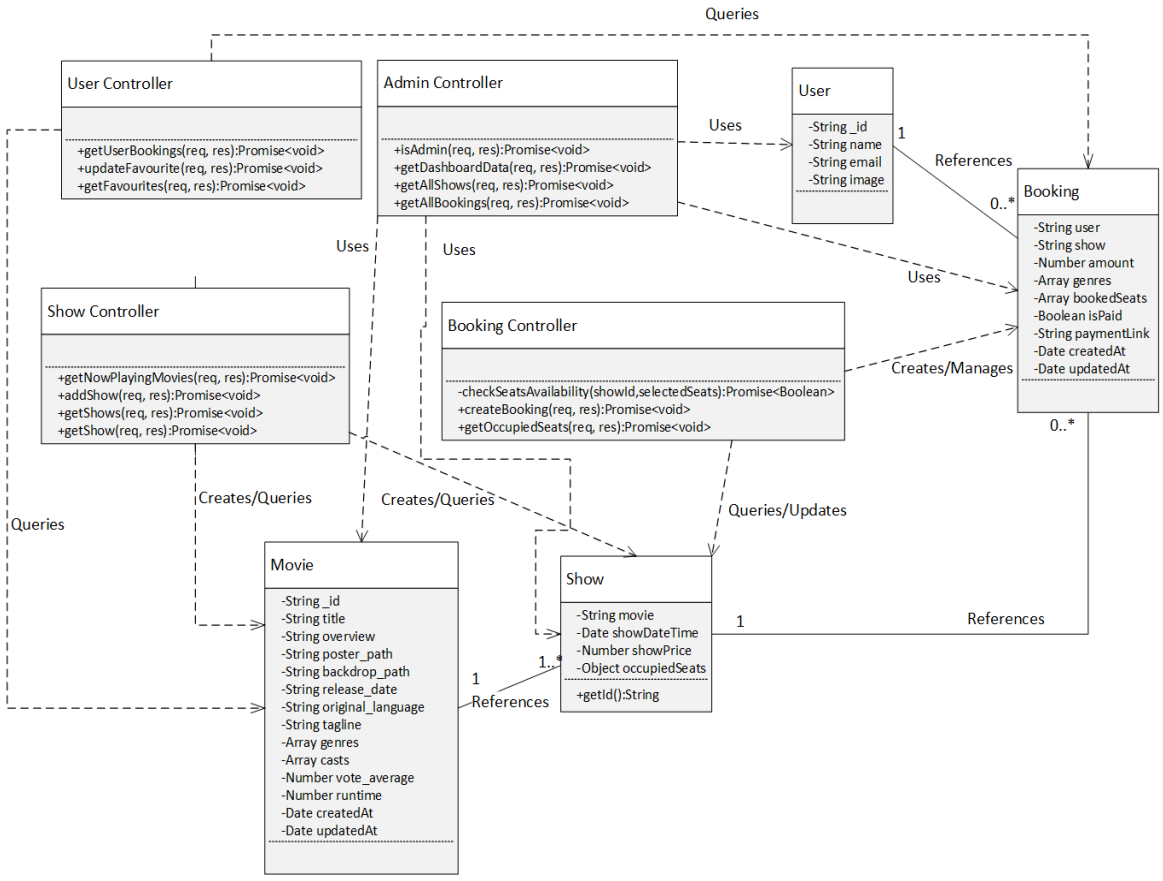

Figure 6 Class Diagram

CHAPTER 4: SYSTEM DESIGN

4.1 Design

Database Design

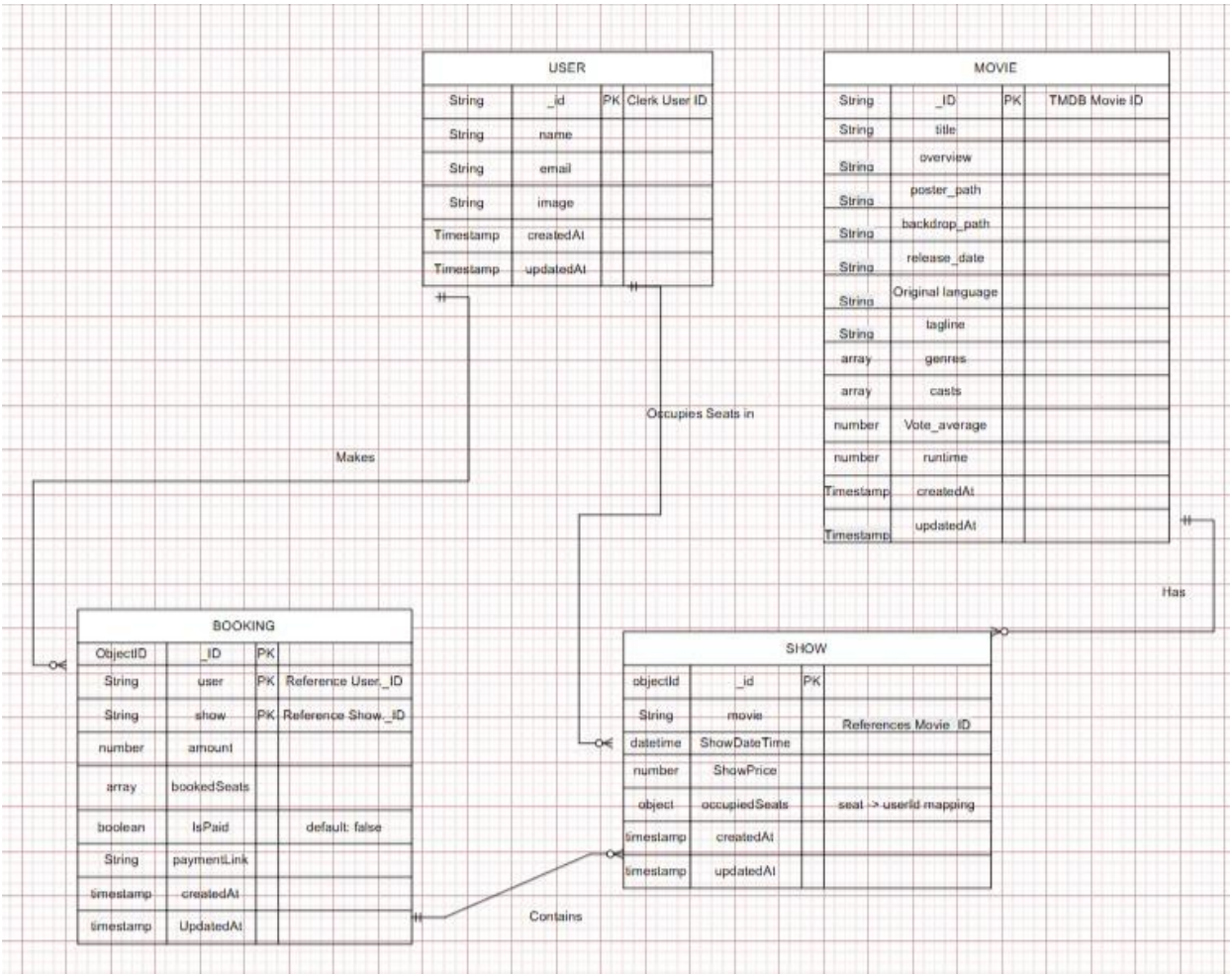

Figure 3 Database Schema

Interface Design

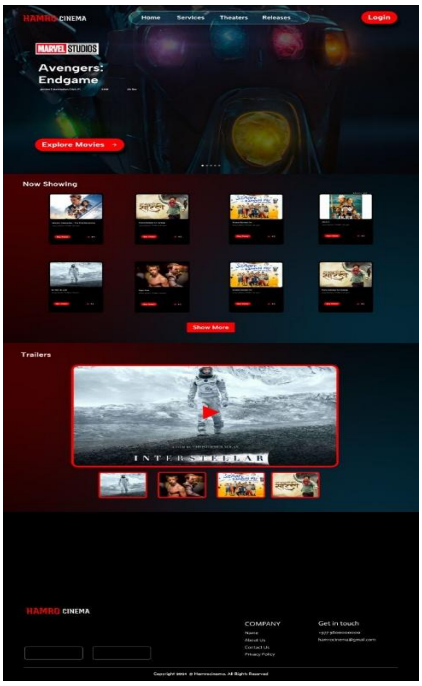

Figure 4 Home Page

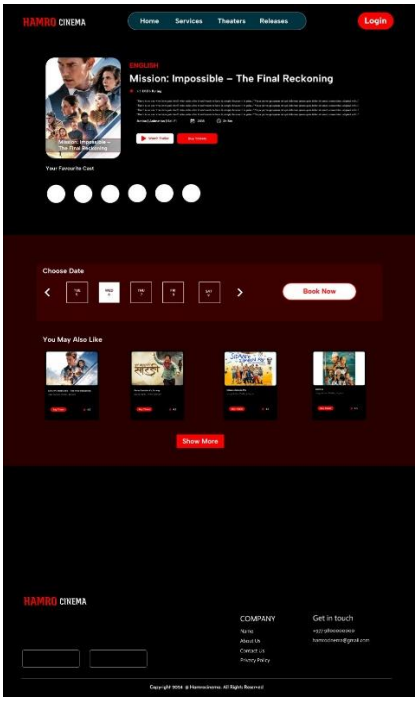

Figure 5 Movie Information

Login and Registration UI

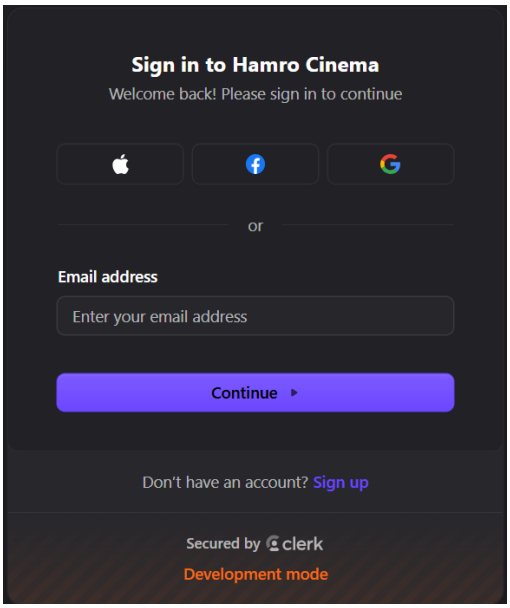

Figure 6 Sign-In Page

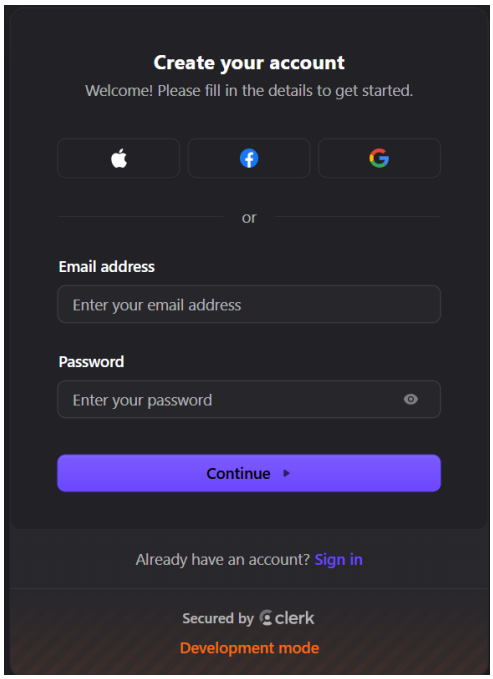

Figure 7 Sign-Up Page

4.2 Algorithm Details

For the movie search feature, we will apply the searching algorithm 'Fuzzy Search Algorithm' to filter movies based on the input query.

Fuzzy Search for Movie Matching

A search technique that uses fuzzy logic that finds matches even when the query isn't exact (e.g., typos, partial matches).

Fuzzy Logic:

A mathematical logic system that deals with **degrees of truth** rather than strict *true/false*. In the real world, things aren't always binary — e.g., temperature isn't just *hot* or *cold*, it can be *slightly warm*, *moderately hot*, etc. Instead of Boolean (0 or 1), fuzzy logic allows values between 0 and 1.

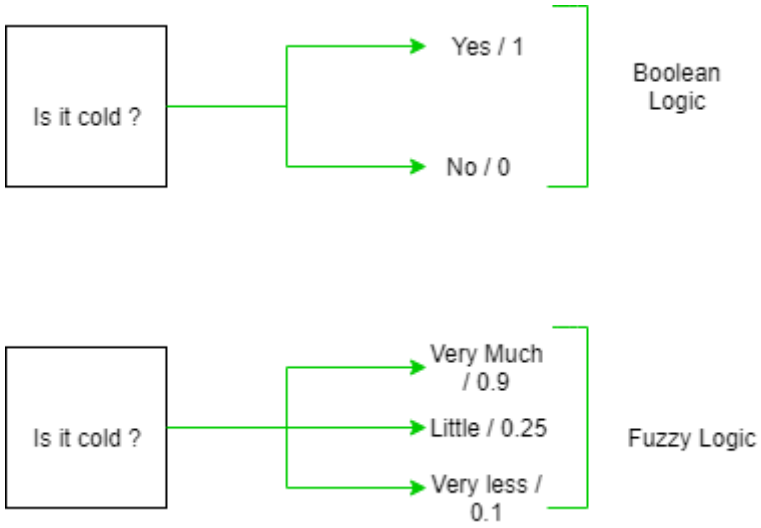

Figure 8 Fuzzy Logic

Working of Fuzzy Search Algorithm

The fuzzy search algorithm compares each character of the user's query against movie titles. A similarity score is calculated based on how many query characters appear in the title. Titles with a score above a certain threshold are considered matches. Matched titles are then sorted by how early the full query appears as a substring in the title.

Example:

Query:"man"

Movie Titles: ["Batman", "Superman", "Ant-Man"]

Final Output:

["Batman", "Ant-Man", "Superman"]

Fusejs

Fuse.js is used for implementation of fuzzy search in this project which is a lightweight JavaScript library for fuzzy search. It lets you search through text,

lists, or objects without requiring exact matches. It’s popular in web apps (like your MERN projects) because it’s fast, dependency-free, and works client-side.

Implementation:

```
import Fuse from 'fuse.js';

const Movies = () => {
  const options = {
    keys: ["title", "description", "genre"], // fields to search in
    threshold: 0.4, // 0.0 = exact match, 1.0 = very fuzzy
    distance: 100,  // how far in the text it can search
    ignoreLocation: true,
  };
  if (searchTerm) {
    const fuse = new Fuse(shows, options);
    const results = fuse.search(searchTerm);
    filteredMovies = results.map(result => result.item); // extract items
  }
```

Keys tells Fuse which fields in shows objects to search (title, description, genre).

threshold → Controls “fuzziness.”

0.0 = very strict (almost exact).

0.4 = allows small typos.

1.0 = very loose (lots of matches).

Distance= How far in the text the match can occur before being ignored. (Here, 100 characters). ignoreLocation if true, matches can be found anywhere in the string, not just near the start.

If the user typed something in the search box it creates a new Fuse instance with shows and options. It runs fuse.search(searchTerm) and returns an array of results like:

[{ item: { title: "Avengers", ... }, score: 0.12 },{ item: { title: "Avengers: Endgame", ... }, score: 0.18]

Extracts just the items (movies) from results. Now filteredMovies contains only the matched movies.

Movie Recommendation Algorithm: K-Nearest Neighbors (KNN)

The K Nearest Neighbors (KNN) classifier is a fundamental supervised machine learning algorithm employed for classification tasks. Operating on the principle of similarity, KNN determines the class of a new data point by identifying the 'k' nearest neighbors to that point within the feature space, typically using a distance metric like Euclidean distance. During training, the algorithm memorizes all

available data points and their associated class labels. When tasked with prediction, KNN calculates distances between the new point and all stored points, selecting the 'k' closest ones. Subsequently, it aggregates their class labels via majority voting, assigning the most common class label among the neighbors to the new data point. In the k-Nearest Neighbours algorithm k is just a number that tells the algorithm how many nearby points or neighbors to look at when it makes a decision.
Example: Imagine you're deciding which fruit it is based on its shape and size. You compare it to fruits you already know.

If k = 3, the algorithm looks at the 3 closest fruits to the new one.

If 2 of those 3 fruits are apples and 1 is a banana, the algorithm says the new fruit is an apple because most of its neighbors are apples.

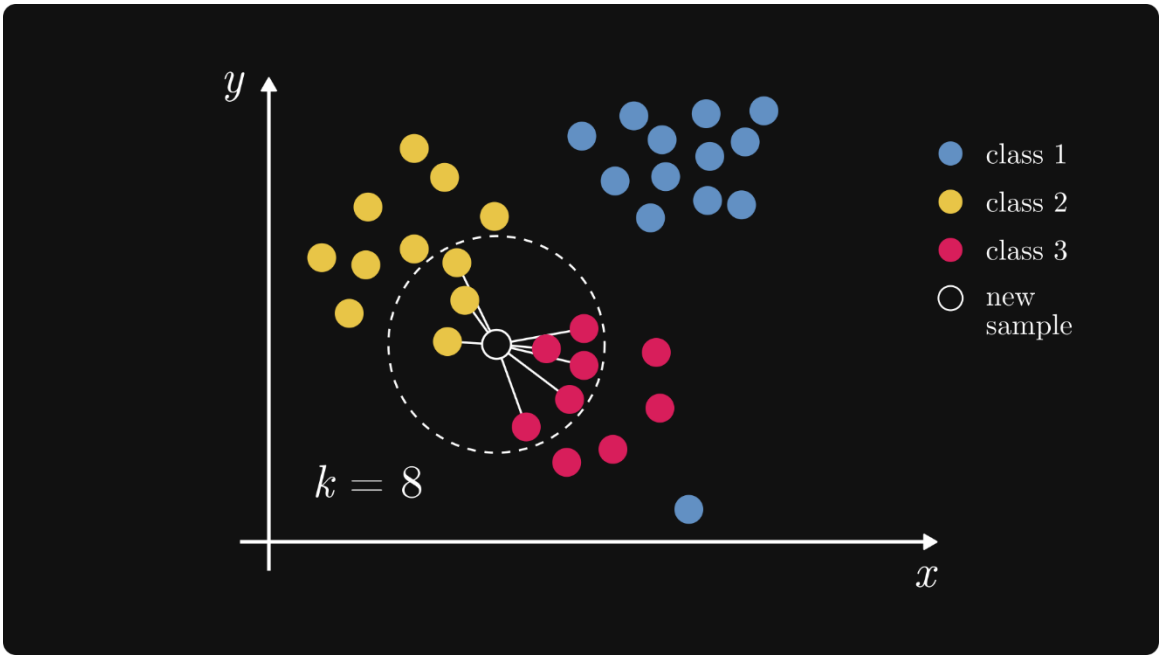

Figure 9 K-Nearest Neighbour

Implementation

```
# Get user ID from command line
if len(sys.argv) < 2:
    print(json.dumps([]))
    sys.exit()
user_id = sys.argv[1]
# Connect to MongoDB
client = MongoClient("mongodb+srv://sjr:root@cluster0.27wr17q.mongodb.net/")
db = client["HamroCinemaMERN"]
bookings = db["bookings"]
movies = db["movies"]
```

```
shows = db["shows"]
# Step 1: Get all booked shows for the user
user_bookings = list(bookings.find({"user": user_id}))
# Step 2: Map booked shows to their movie IDs
watched_movie_ids = []
for b in user_bookings:
    try:
        show = shows.find_one({"_id": ObjectId(b["show"])})
        if show and "movie" in show:
            watched_movie_ids.append(ObjectId(show["movie"]))
    except Exception as e:
        continue  # skip invalid ObjectId
# Step 3: Build user genre preference
user_genres = {}
for movie in movies.find({"_id": {"$in": watched_movie_ids}}):
    for genre in movie.get("genres", []):
        name = genre.get("name")
        if name:
            user_genres[name] = user_genres.get(name, 0) + 1
# Step 4: Score candidate movies (exclude watched)
recommendations = []
for movie in movies.find({"_id": {"$nin": watched_movie_ids}}):
    score = 0
    for genre in movie.get("genres", []):
        name = genre.get("name")
        if name:
            score += user_genres.get(name, 0)
    score *= movie.get("vote_average", 0)
    if score > 0:
        recommendations.append((score, movie["title"]))
# Step 5: Sort descending and take top 5
recommendations.sort(reverse=True)
top_movies = [title for _, title in recommendations[:5]]
print(json.dumps(top_movies))
```

This code Looks at what the user has booked before → figures out their favorite genres. Finds other movies in the database that match those genres. Scores them based on preference + movie rating. Returns the top recommendations.

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 Implementation

The implementation phase involved coding the front-end using JavaScript, React and integrating it with the back-end (Express.js) and database (MongoDB). Each module was tested individually before integration.

5.1.1 Tools Used (Programming Languages, Database Platforms)

- **Frontend:** HTML,CSS(Tailwind), JavaScript.
- **Backend:** Node.js, Express.js.
- **Database:** MongoDB(with Inngest).
- **Design:** Figma, Visio, Draw.io
- **Version Control:** Git and Github.
- **Testing:** Postman

5.1.2 Implementation Details of Modules (Description of Modules/Functions)

a. **User Management Module**

- **User Registration**: Allows new users to create an account by providing their name, email, and password. The password is securely hashed before storing it in the database.
- **User Authentication**: Manages user login and logout processes. It verifies user credentials and creates a session or token for authenticated users.
- **Profile Management**: Enables users to view and update their personal information.

b. **Movie Management Module**

- **Add Movies**: Admins can add new movies ,update existing movie price, timings.
- **Movie Listings**: Displays a list of available movies.

c. **Showtime Management Module**

- **Add/Update/Delete Showtimes**: Admins can schedule new showtimes, modify existing ones. Each showtime is linked to a specific movie and theater.
- **Showtime Display**: Users can view available showtimes for a selected movie and choose one for booking.

d. **Seat Reservation Module**

- **Seat Selection**: Provides a visual representation of the theater's seating layout, showing available and occupied seats. Users can select their desired seats.
- **Seat Reservation**: Once seats are selected, the system temporarily holds them until the booking process is completed to prevent double booking.

e. **Booking Module**

- **Booking Process**: Handles the step-by-step process of booking tickets, from seat selection to final confirmation.

f. **Admin Dashboard Module**

- **Manage Movies and Showtimes**: Provides tools for admins to add, update movies and showtimes.

5.2 Testing

User acceptance testing was conducted manually to ensure the system functions as expected.

5.2.1 Test cases for Unit Testing

Table 1 Test Cases for Unit Testing (User Management Module)

	USER MANAGEMENT MODULE		
	Test Case 1: User Registration	Test Case 2: User Login	Test Case 3: Invalid Login Attempt
Objective	Verify that a user can register with valid input.	Verify that a registered user can log in with valid credentials.	Verify that a user cannot log in with an incorrect password.
Input	Name, valid email, and password.	Registered email and correct password.	Incorrect password.

Expected Output	User account is created, and a confirmation message is displayed.	User is authenticated, and the dashboard is displayed.	Error message is displayed, and access is denied.

Table 2 Test Cases for Unit Testing(Movie Management Module)

	MOVIE MANAGEMENT MODULE		
	Test Case 1: Add New Movie	Test Case 2: Update Movie Information	Test Case 3: Delete Movie
Objective	Verify that an admin can add a new movie to the system.	Verify that an admin can update the details of an existing movie.	Verify that an admin can delete a movie from the system.
Input	Movie title, genre.	Updated movie synopsis.	Movie ID.
Expected Output	Movie is added and appears in the listings.	Movie details are updated and displayed correctly.	Movie is removed and no longer appear in listings.

Table 3 Test Cases for Unit Testing(Seat Reservation Module)

	SEAT RESERVATION MODULE		
	Test Case 1: Display Seat Layout	Test Case 2: Reserve Seat	Test Case 3: Double Booking Prevention
Objective	Verify that the seat layout is displayed correctly for a	Verify that a user can reserve a seat successfully.	Verify that a seat cannot be reserved by two users simultaneously.

	selected showtime.		
Input	Showtime ID.	Selected seat number.	Same seat number selected by two users.
Expected Output	Seat layout with available and reserved seats is displayed.	Seat status is updated to reserved, and confirmation is shown.	The first user reserves the seat, and the second user receives an unavailable notification.

Table 4 Test Case for Unit Testing(Booking Module)

BOOKING MODULE	
Test Case 1: Complete Booking	
Objective	Verify that a user can complete a booking with valid payment.
Input	Selected seats.
Expected Output	Booking is confirmed, and a confirmation message is sent.

5.2.2 Test Cases for System Testing

System testing focuses on validating the complete and integrated software to ensure that the movie booking website meets its specified requirements. It involves testing the interactions between different modules and verifying that the system as a whole functions correctly. Here are some key test cases for system testing:

Table 5 Test Cases for System Testing(User Registration and Authentication)

	USER REGISTRATION AND AUTHENTICATION		
	Test Case 1: Successful User Registration	Test Case 2: Successful User Login	Test Case 3: Login with Invalid Credentials
Objective	Verify that a new user can successfully register.	Verify that a registered user can log in.	Verify that login fails with invalid credentials.
Preconditions	User is not already registered.	User is registered and activated.	User attempts to log in with incorrect credentials.
Steps	• Navigate to the registration page. • Enter valid name and password. • Submit the form.	• Navigate to the login page. • Enter valid email and password. • Submit the form.	• Navigate to the login page. • Enter incorrect password. • Submit the form.
Expected Result	User account is created, and a confirmation email is sent.	User is authenticated and redirected to the dashboard.	An error message is displayed, and access is denied.

Table 6 Test Cases for System Testing(Movie and Showtime Management, Booking Process & Admin Dashboard)

	MOVIE AND SHOWTIME MANAGEMENT	BOOKING PROCESS	ADMIN DASHBOARD AND MANAGEMENT
Test Case	Display Movie Listings	Successful Seat Reservation and Booking	Add New Movie and Showtime
Objective	Verify that the movie listings page displays all available movies.	Verify that a user can successfully reserve seats and complete the booking.	Verify that the admin can add a new movie and schedule a showtime.
Preconditions	Movies are added to the system.	User is logged in, and seats are available.	Admin is logged in.
Steps	Navigate to the movie listings page.	Select a movie and showtime. Choose seats from the seating layout. Confirm the booking.	Add a new movie with required details. Schedule a showtime.
Expected Result	All available movies are displayed with relevant details like title and poster.	Seats are reserved, payment is processed, and booking is confirmed.	The new movie is added, and the showtime appears in the listings.

5.2.3 API Testing with Postman

Api testing was done using Postman along with manual testing to see if operations done from and to the database works or not. There were three tests done.

Test to get now playing movies from tmdb database.

After writing the backend code, in postman the function to get the movies from tmdb database was tested. Using the get method, the test was a success and now playing movies from the tmdb database can be added to the project.

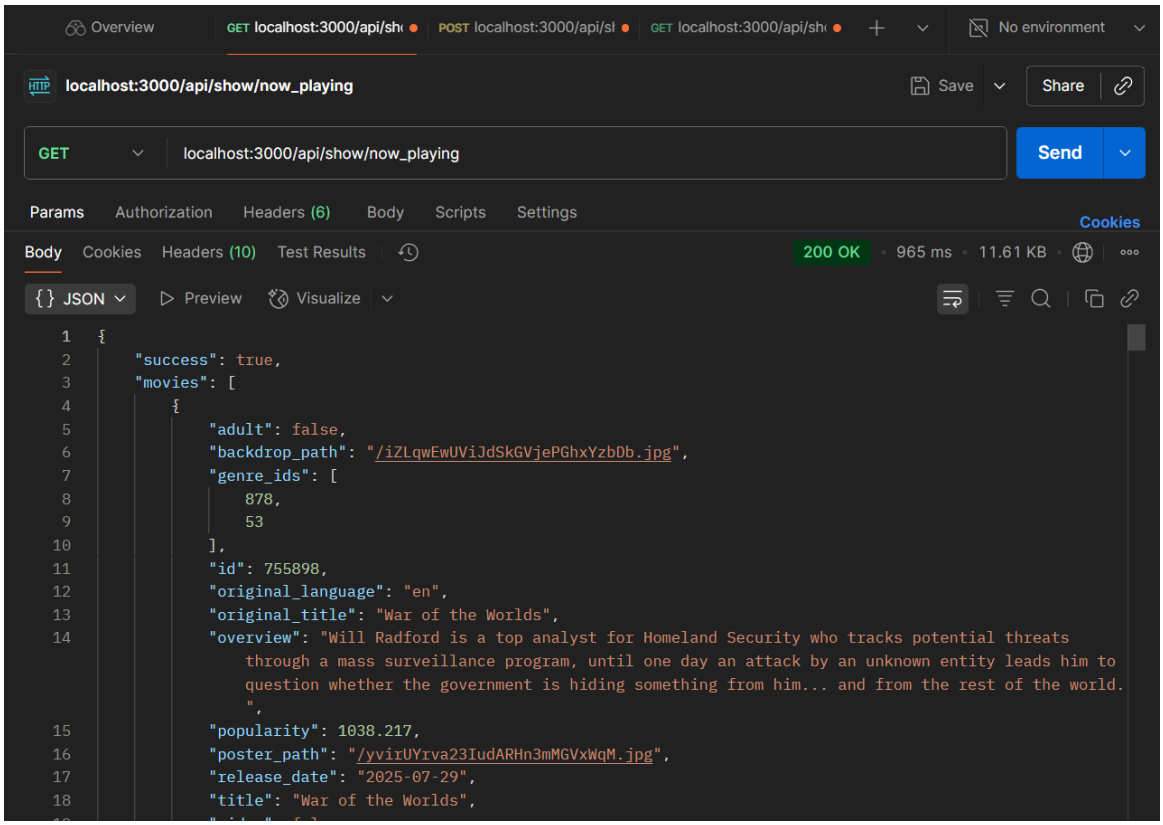

Figure 10 API test 1

Test to add show to the database

Function to add show to the database was tested using Postman by using the POST method. The backend creates shows from the movies database and this test to post a show was success.

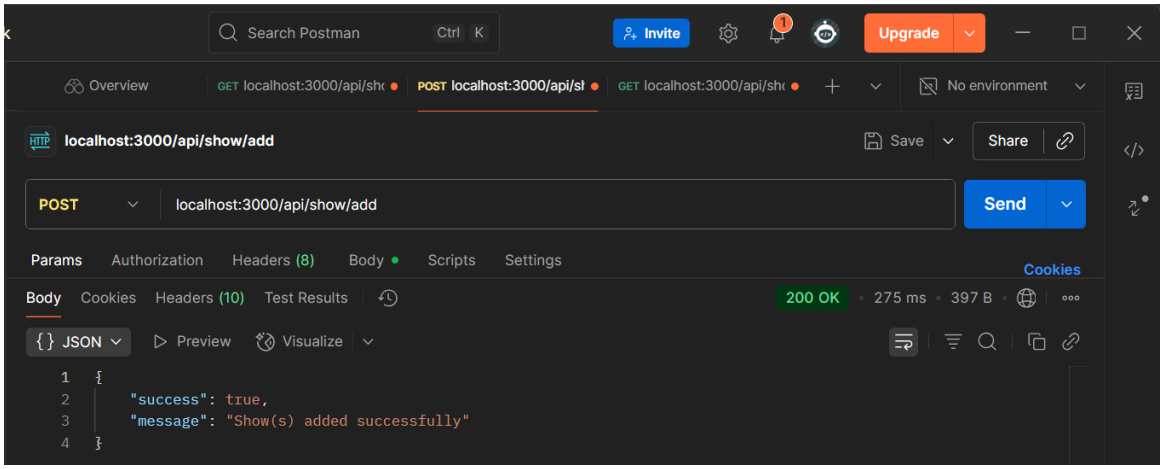

Figure 11 API test 2

Test to get a movie from its id

To test if we can retrieve or fetch a movie from movies database we tested /api/show/movieId route using get method. The test came out success which means the function is working.

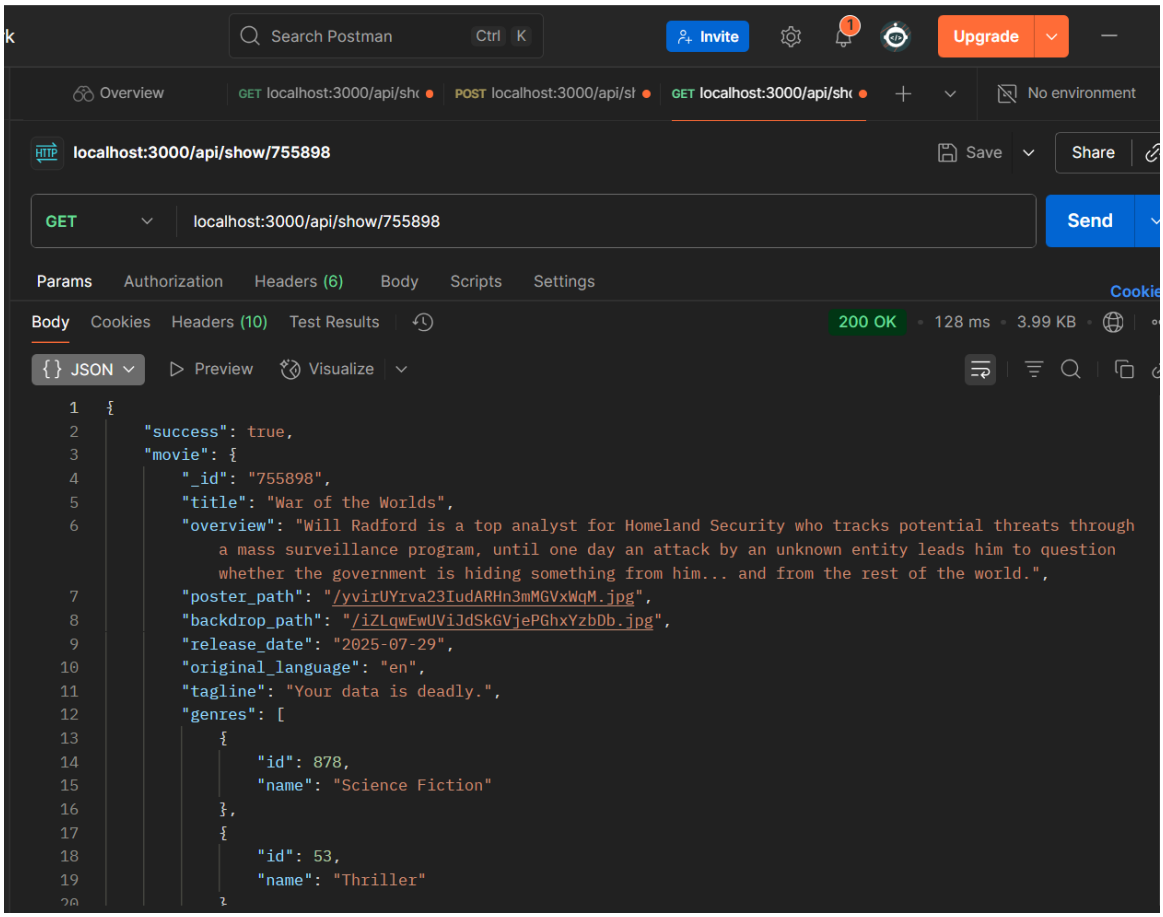

Figure 12 API test 3

5.3 Result Analysis

Outcomes

The main goals are achieved in the system. Users can signup and login while the authentication process also functioning properly. Users can view the movie details like timing, seats, date and can book the seats by paying through a payment gateway. Admin can also access the dashboard where they can add shows by selecting the date, time and price. The database seems to successfully fetch data from TMDB database to show the movies currently being showed in the world. Hence, the goals were achieved for this system. The payment process was successful where after payment user is redirected to their bookings list page where the payment is confirmed when the pay now button doesn't appear.

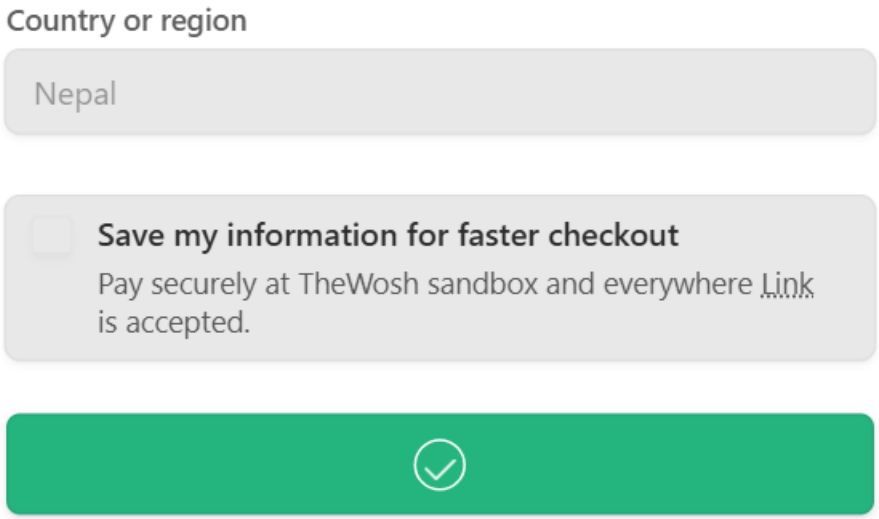

Figure 13 Payment Confirm

Performance Analysis

The speed of movie fetching from TMDB API along with movie fetching from the database has been in less than a second or a second which is a good speed. The movie data from database also fetches at a similar speed to display it to the users in the movie list. Each page loads at a good speed. When user clicks the booking button it takes them to the payment section in approximately 1 second. And when clicking the pay button it takes a little bit more than a second to confirm. Overall the speed and performance is satisfactory.

Test Results

After manually testing the system, all the tests are passed including unit and system testing. The api testing done using postman also is done successfully as the POST and GET

operations work successfully at the tested endpoints. In conclusion all the tests are successful.

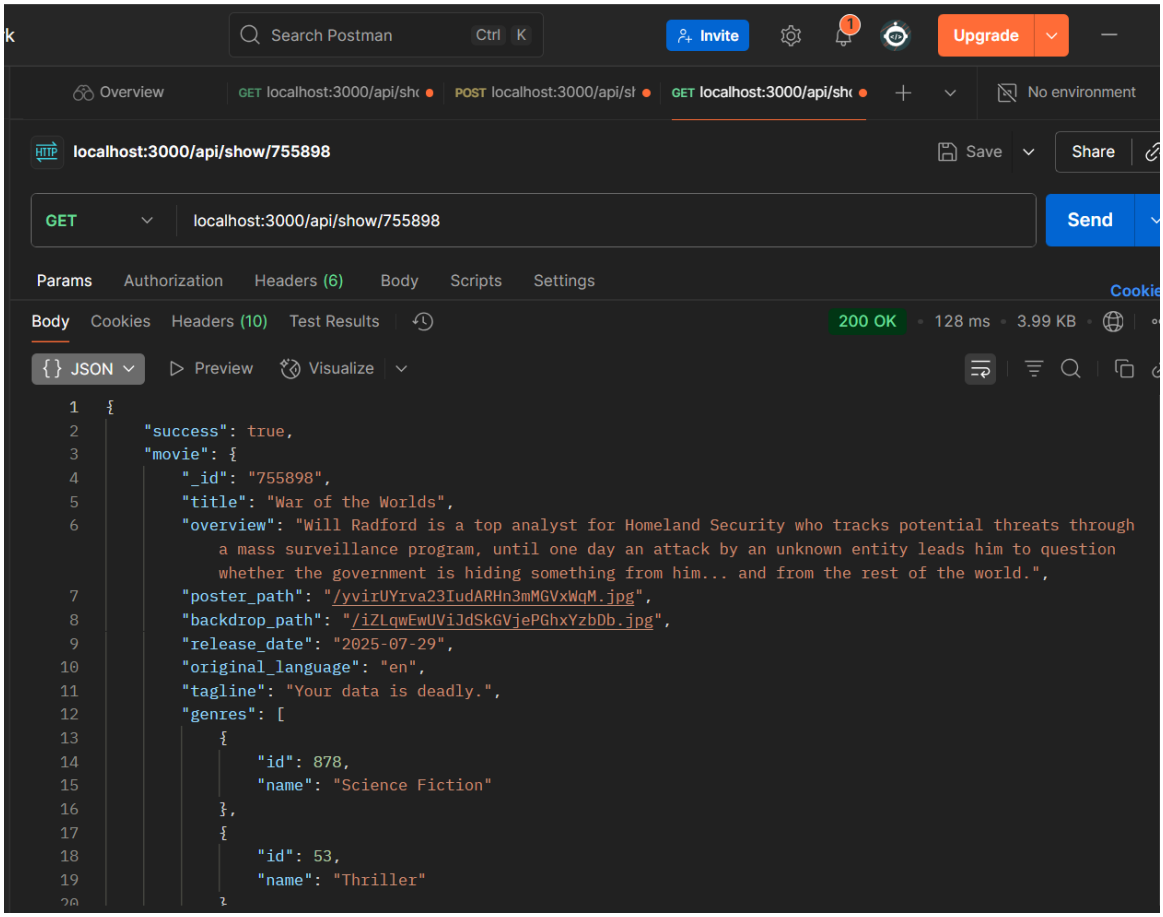

Figure 14 Successful API test

Limitations Noticed

However, there are some limitations in the system. The payment gateway is restricted to test mode and real payment has not been implemented. Also the system has not been tested under heavy traffic and has security vulnerabilities. This means system will most likely not work in high traffic and might have easy security breach.

Final Result Statement

The system successfully met its objectives by enabling users to register, browse movies, book tickets, and complete payments in a secure and user-friendly way. Testing confirmed that the system is functional, efficient however there are some limitations which can be handled in the future.

CHAPTER 6: CONCLUSIONS AND FUTURE RECOMMENDATIONS

6.1 Conclusion

The development of the movie booking website has been a comprehensive learning experience, combining technical expertise, teamwork, and project management. The project successfully delivered a functional and user-friendly platform that meets the requirements for an efficient online movie booking system. The project has not only enhanced the team's technical skills but also improved their ability to collaborate effectively, manage time, and adapt to challenges. The movie booking website stands as a testament to the team's hard work, problem-solving abilities, and commitment to delivering a high-quality software solution. This experience has laid a strong foundation for future projects and professional growth in the field of software development.

6.2 Future Recommendation

In consideration of the future trajectory of our "Movie Ticket Booking System", we propose a set of recommendations aimed at enhancing user satisfaction and elevating platform functionality. This approach not only enhances user engagement but also contributes to a more fulfilling and user-centric platform. Additionally, expanding the social aspect of the platform introducing features like movie discussion forums or user-created groups that could foster a strong sense of community among the users. Implementing a real time notification system to keep users informed about the new movie releases, ticket updates, or responses to their review would add a dynamic element to their interaction with the platform. Lastly, continuous monitoring of user feedback and analytics will be vital in identifying areas for improvement and adapting the platform to evolving user needs, ensuring its long-term relevance and success.

REFERENCES

[1] K. Mudigati and K. , "Kalpita Technologies," 2024. [Online]. Available: https://kalpitatechnologies.com/assets/white-paper/OnlineMovieTicketBooking.pdf.

[2] S. Lazarevic and T. Zuvela, "Design Of Information System To Support Movie Ticket Booking And Cinema Operations," IEEE, Sarajevo,Bosnia, 2022.

[3] L. U. Cayres, B. S. D. Lima, R. E. Garcia and R. C. M. Correia, "Analysis of NodeJS application performance using MongoDB drivers," Faculty of Science and Technology-Sao Paulo State University(UNESP), 2020.

[4] T. Sufiyan, "simplilearn," 11 July 2024. [Online]. Available: https://www.simplilearn.com/tutorials/nodejs-tutorial/nodejs-mongodb#why_connect_nodejs_with_mongodb.

[5] A. H. Allam, A. R. C. Hussin and H. Dahlan, "User Experiences: Challenges and Opportunity," *JISRI.*

[6] R. J. K. Jacob, "User Interface," p. 8, 2003.

[7] S.Sridevi, *User Interface Design,* vol. 2, no. 2, p. 11, 2014.

[8] P. Desmarets and V. V. D. Borght, "Hackolade," [Online]. Available: https://hackolade.com/nosqldb/mongodb-data-modeling.html.

[9] Datastax, 2020. [Online]. Available: https://www.datastax.com/benefits-cloud-database.

APPENDICES

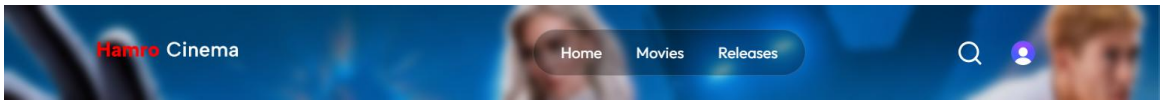

Figure 15 Header

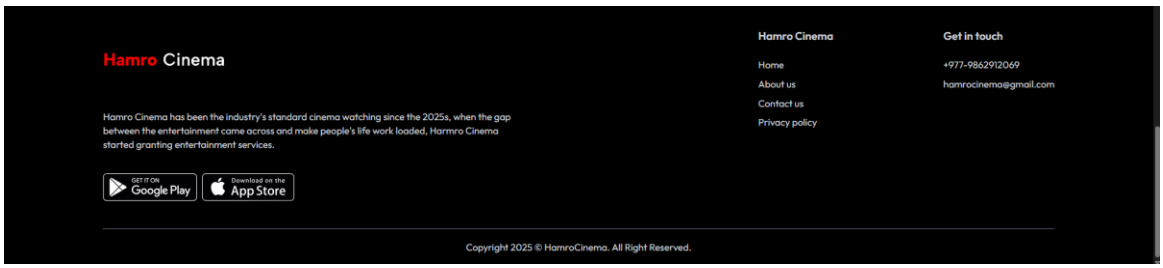

Figure 16 Footer

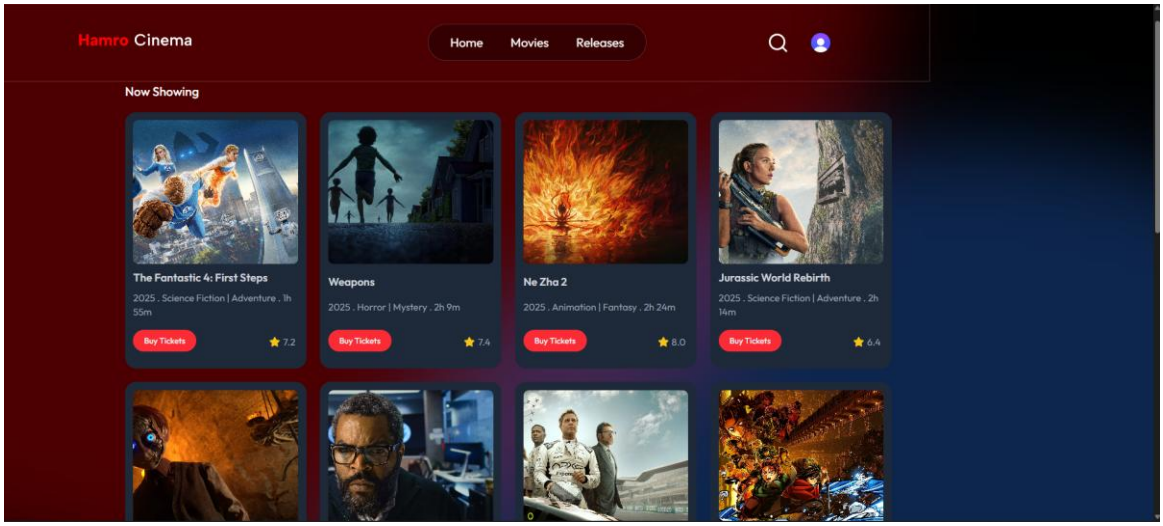

Figure 17 Now Showing Section

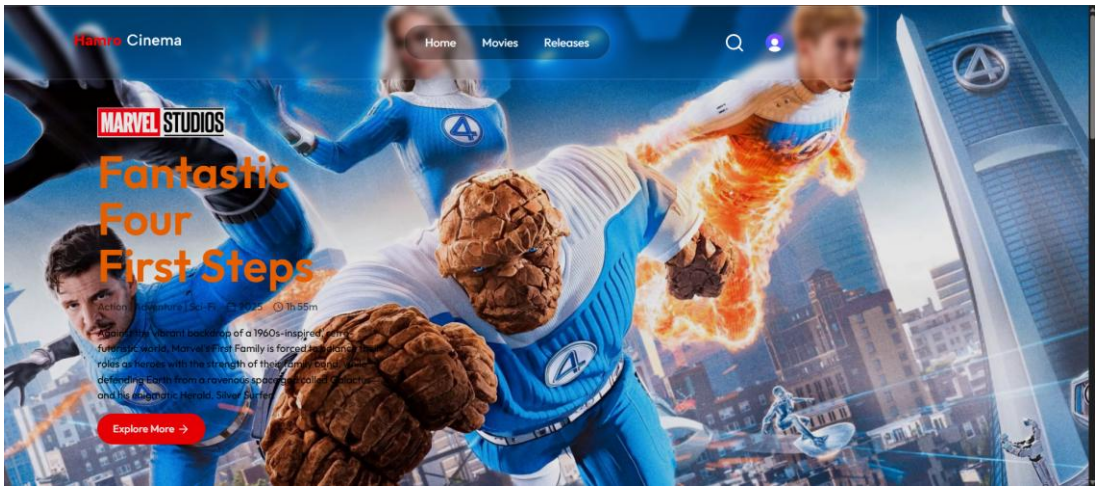

Figure 18 Hero Section

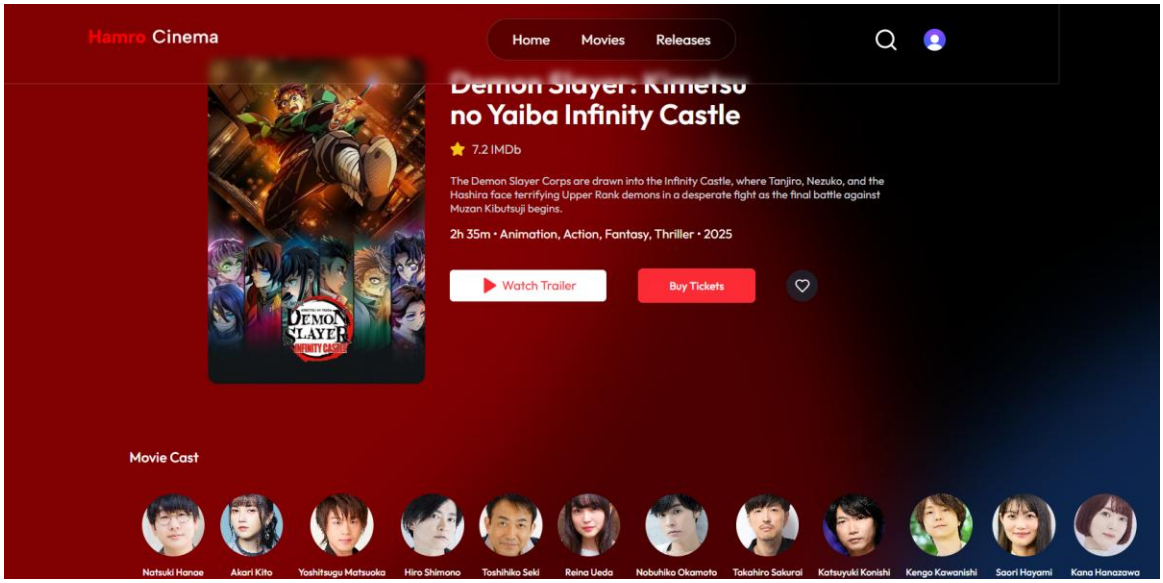

Figure 19 Movie Details

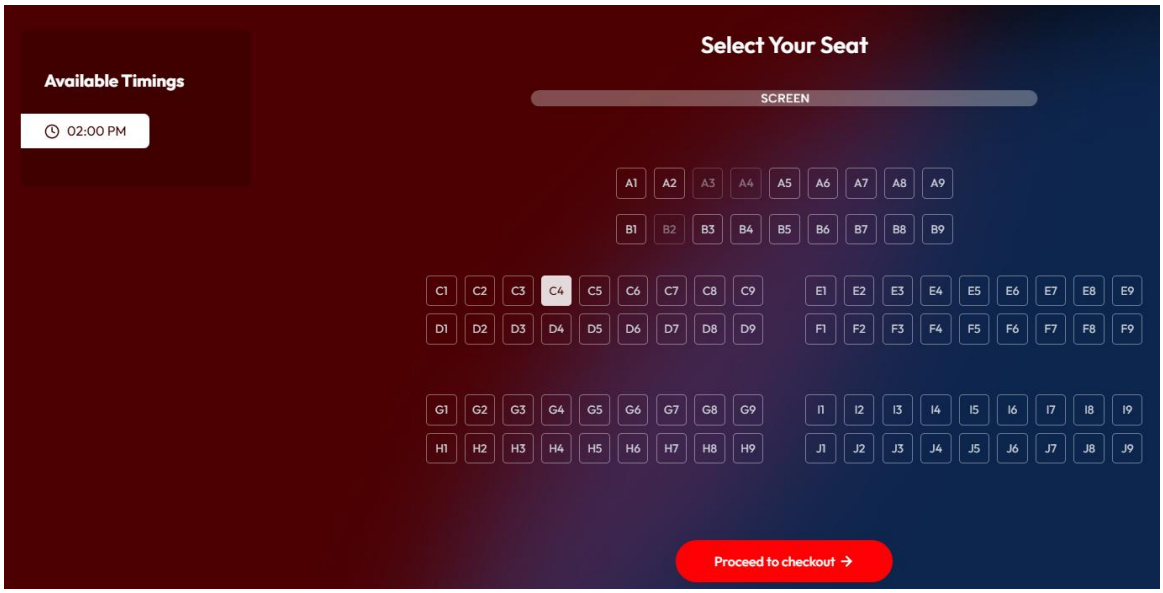

Figure 20 Seat Selection

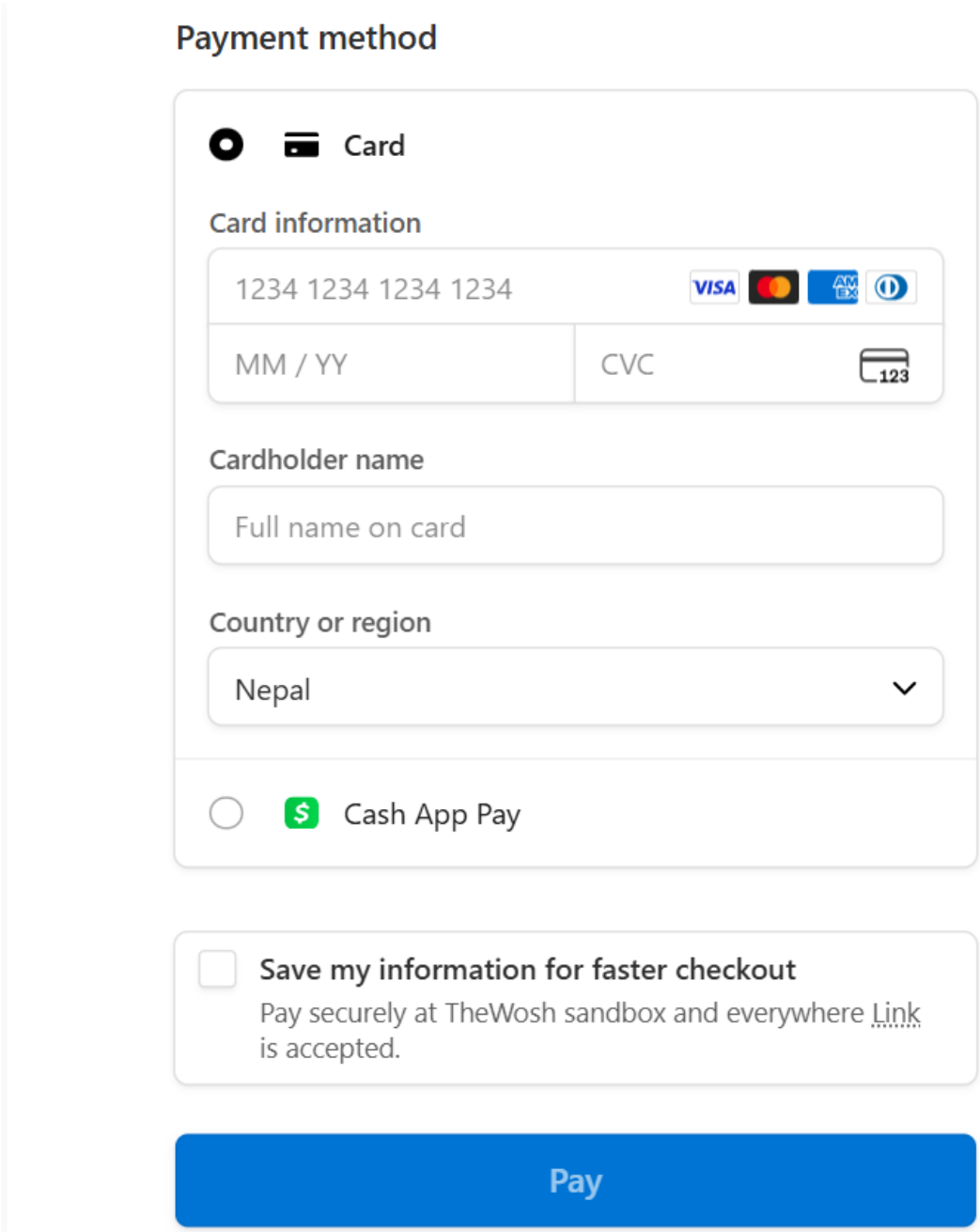

Figure 21 Payment Section

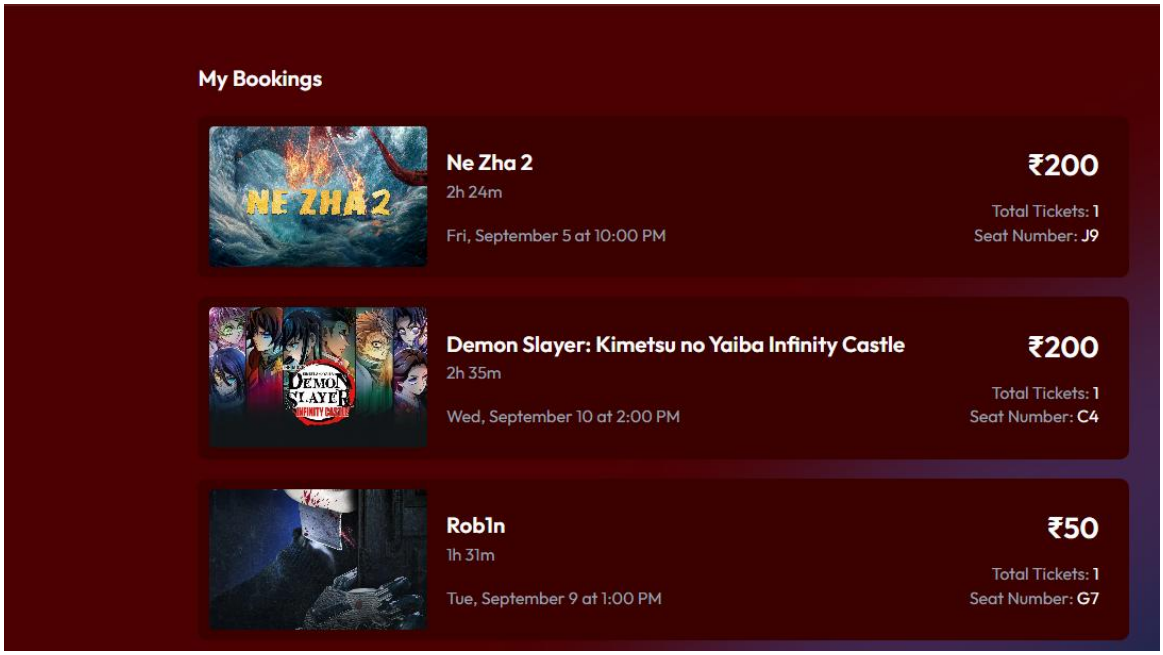

Figure 22 Bookings List

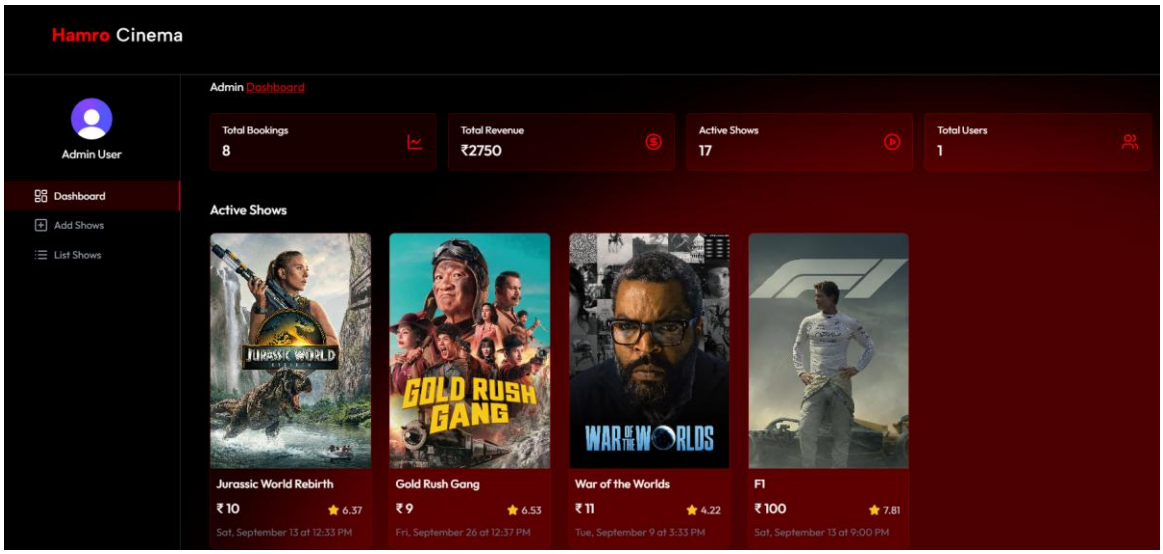

Figure 23 Admin Dashboard

Figure 24 Shows List

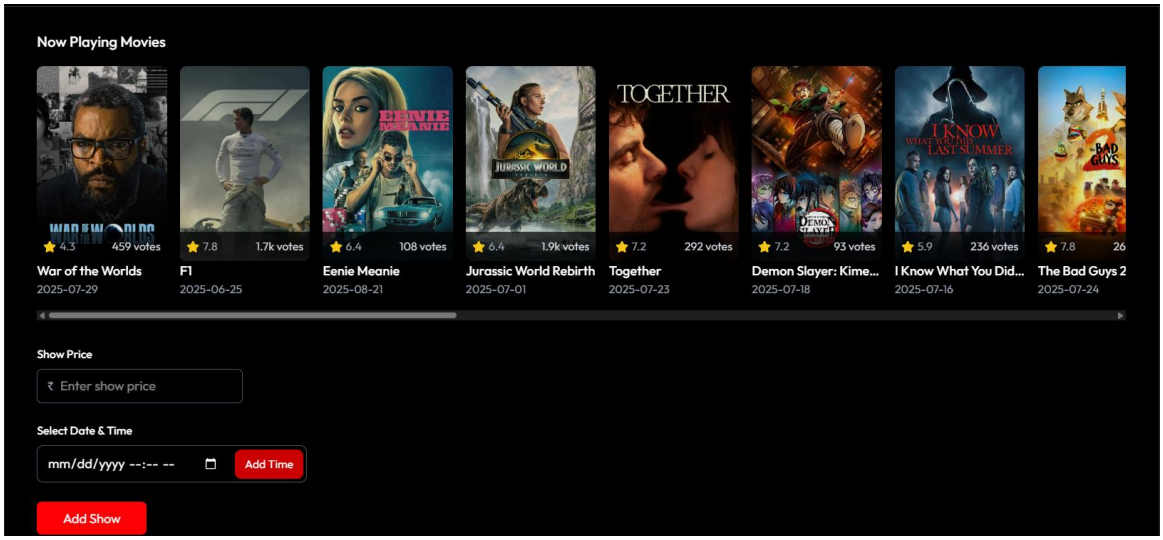

Figure 25 Admin-Add Show Page

Figure 26 Recommendations

Major Source Code Components.

User Controller Class

```
import { clerkClient } from "@clerk/express";
import Booking from "../models/Booking.js";
import Movie from "../models/Movie.js";
//API Controller Function to get User Bookings
export const getUserBookings = async (req, res)=>{
   try{
      const user = req.auth().userId;
      const bookings = await Booking.find({user}).populate({
         path: 'show',
         populate: {path: "movie"}
      }).sort({createdAt: -1});
      res.json({success: true, bookings});
   } catch (error){
      console.error(error);
      res.json({success: false, message: error.message});
   }
}
//API Controller Function to Update Favourite Movie in Clerk User Metadata
export const updateFavourite = async (req, res)=>{
   try{
      const {movieId} = req.body;
      const userId = req.auth().userId;
      const user = await clerkClient.users.getUser(userId);
      if(!user.privateMetadata.favourites){
         user.privateMetadata.favourites = [];
      }
 if(!user.privateMetadata.favourites.includes(movieId)){
         user.privateMetadata.favourites.push(movieId);
      } else{
            user.privateMetadata.favourites = user.privateMetadata.favourites.filter(item=>
item!== movieId);
```

```
            } await clerkClient.users.updateUserMetadata(userId, {privateMetadata:
user.privateMetadata})
        res.json({success: true, message: "Favourite Movies Updated"})
            } catch (error){
        console.error(error);
        res.json({success: false, message: error.message});
    }
}
export const getFavourites = async (req, res)=>{
    try{
        const auth = req.auth();
        if (!auth || !auth.userId) {
            return res.status(401).json({success: false, message: "Unauthorized"});
        }
        const user = await clerkClient.users.getUser(auth.userId);
        const favourites = user.privateMetadata.favourites || [];
        const movies = await Movie.find({_id: {$in: favourites}})
        res.json({success: true, movies})
    } catch (error){
        console.error(error);
        res.json({success: false, message: error.message});
    }
}
```

Booking Controller Class

```
const checkSeatsAvailability = async (showId, selectedSeats)=>{
   try{
      if (!Array.isArray(selectedSeats) || selectedSeats.length === 0) {
         console.log("selectedSeats invalid:", selectedSeats);
         return false;
      }
      const showData = await Show.findById(showId)
      if(!showData) {
         console.log("Show not found for id:", showId);
         return false;
      }
      const occupiedSeats = showData.occupiedSeats;
      const isAnySeatTaken = selectedSeats.some(seat=>occupiedSeats[seat]);
      return !isAnySeatTaken;
   } catch (error){
      console.log(error.message);
      return false;
   }
}
export const createBooking = async (req,res)=>{
   try{
      const {userId} = req.auth();
      const {showId, selectedSeats} = req.body;
      const {origin} = req.headers;
      if (!userId) {
         return res.status(401).json({success: false, message: "User not authenticated"})
      }
      //check if seats are available
      const isAvailable = await checkSeatsAvailability(showId, selectedSeats);
      if(!isAvailable){
         return res.json({success: false, message: "Selected seats are not available"});
      }
      //get the show details
      const showData = await Show.findById(showId).populate('movie');
      //create a new booking
      const booking = await Booking.create({
         user: userId,
         show: showId,
         amount: showData.showPrice * selectedSeats.length,
         genres: showData.movie.genres,
         bookedSeats: selectedSeats
      })

      selectedSeats.map((seat)=>{
         showData.occupiedSeats[seat] = userId;
      })
      showData.markModified('occupiedSeats');
      await showData.save();
      //Stripe Gateway Initialize
```

```
        const stripeInstance = new stripe(process.env.STRIPE_SECRET_KEY)

        const line_items = [{
            price_data: {
                currency: 'inr',
                product_data:{
                    name: showData.movie.title
                },
                unit_amount: Math.floor(booking.amount)*100
            },
            quantity: 1
        }]
        const session = await stripeInstance.checkout.sessions.create({
            success_url: `${origin}/loading/my-bookings`,
            cancel_url: `${origin}/my-bookings`,
            line_items: line_items,
            mode: 'payment',
            metadata: {
                bookingId: booking._id.toString()
            },
            expires_at: Math.floor(Date.now() / 1000)+30*60,
        })
        booking.paymentLink = session.url
        await booking.save()
        res.json({success:true, url: session.url})

    } catch (error){
        console.log(error.message);
        res.json({success: false, message: error.message});
    }
}
export const getOccupiedSeats = async (req, res) => {
    try{
        const {showId} = req.params;
        if (!showId || showId === "undefined") {
            return res.status(400).json({success: false, message: "Show ID is required"});
        }
        const showData = await Show.findById(showId);
        if (!showData) {
            return res.status(404).json({success: false, message: "Show not found"});
        }
        const occupiedSeats = Object.keys(showData.occupiedSeats);
        res.json({success: true, occupiedSeats});
    } catch (error){
        console.log(error.message);
        res.json({success: false, message: error.message});
}
```

Show Controller Class

```
//API to get now playing movies from tmdb API

export const getNowPlayingMovies = async (req, res) => {
   try {
      const { data } = await axios.get('https://api.themoviedb.org/3/movie/now_playing',{
         headers: {Authorization : `Bearer ${process.env.TMDB_API_KEY}`}
      })
      const movies = data.results;
      res.json({success: true, movies: movies})
   } catch (error) {
      console.error(error);
      res.json({success: false, message: error.message}
//API to add a new show to the database
export const addShow = async (req, res) => {
   try{
      const {movieID, showsInput, showPrice} = req.body;
      let movie = await Movie.findById(movieID)
      if (!movie) {
         // Fetch movie details and credits from TMDB API
         const [movieDetailsResponse, movieCreditsResponse] = await Promise.all([
            axios.get(`https://api.themoviedb.org/3/movie/${movieID}`,{
         headers: {Authorization : `Bearer ${process.env.TMDB_API_KEY}`}}),
               axios.get(`https://api.themoviedb.org/3/movie/${movieID}/credits`, {headers:
{Authorization : `Bearer ${process.env.TMDB_API_KEY}`}})
         ]);
         const movieApiData = movieDetailsResponse.data;
         const movieCreditsData = movieCreditsResponse.data;
         const movieDetails = {
            _id: movieID,
            title: movieApiData.title,
            overview: movieApiData.overview,
            poster_path: movieApiData.poster_path,
            backdrop_path: movieApiData.backdrop_path,
            genres: movieApiData.genres,
            casts: movieCreditsData.cast,
            release_date: movieApiData.release_date,
            original_language: movieApiData.original_language,
            tagline: movieApiData.tagline || "",
            vote_average: movieApiData.vote_average,
            runtime: movieApiData.runtime,
         }
         //add movie to the database
         movie = await Movie.create(movieDetails);
      }
      const showsToCreate = [];
      showsInput.forEach(show=>{
         const showDate = show.date;
         show.time.forEach((time)=>{
            const dateTimeString = `${showDate}T${time}`;
```

```
            showsToCreate.push({
                movie: movieID,
                showDateTime: new Date(dateTimeString),
                showPrice,
                occupiedSeats: {}
            })
        })
    });
    if(showsToCreate.length>0){
        await Show.insertMany(showsToCreate);
    }
    res.json({success: true, message: "Show(s) added successfully"})
  } catch (error) {
    console.error(error);
    res.json({success: false, message: error.message})
  }
}
//api to get all shows from the database
export const getShows = async (req, res) => {
  try{
            const shows = await Show.find({showDateTime: {$gte: new Date()}}).populate('movie').sort({showDateTime: 1});
    //filter unique shows
    const uniqueShows= new Set(shows.map(show=> show.movie))
    res.json({success: true, shows: Array.from(uniqueShows)})
  } catch (error){
    console.error(error);
    res.json({success: false, message: error.message});
  }
}
//api to get a single show from the database
export const getShow = async (req, res) => {
  try{
    const {movieID} = req.params;
      const shows = await Show.find({movie: movieID, showDateTime: {$gte: new Date()}})
    const movie = await Movie.findById(movieID);
    const dateTime = {};
    shows.forEach((show)=>{
      const date= show.showDateTime.toISOString().split('T')[0];
      if(!dateTime[date]){
        dateTime[date] = [];
      }
      dateTime[date].push({time: show.showDateTime, showId: show._id})
    })
    res.json({success: true, movie, dateTime})
  } catch (error){
    console.error(error);
    res.json({success: false, message: error.message});
  }
```

```
}
```

Admin Controller Class

```
//API to check if user is admin
export const isAdmin = async (req, res)=>{
   res.json({success: true, isAdmin: true});
}
/api to get dashboard data
export const getDashboardData = async (req, res)=>{
   try{
      const bookings = await Booking.find({isPaid: true});
            const activeShows = await Show.find({showDateTime: {$gte: new
Date()}}).populate('movie');
      const totalUser = await User.countDocuments();
      const dashboardData = {
         totalBookings: bookings.length,
         totalRevenue: bookings.reduce((acc, booking)=> acc + booking.amount, 0),
         activeShows,
         totalUser
      }
      res.json({success: true, dashboardData});
   } catch (error){
      console.error(error);
      res.json({success: false, message: error.message});
   }
}
//API to get all shows
export const getAllShows = async (req, res)=>{
   try{
               const shows = await Show.find({showDateTime: {$gte: new
Date()}}).populate('movie').sort({showDateTime: 1});
      res.json({success: true, shows});
   } catch (error){
      console.error(error);
      res.json({success: false, message: error.message});
   }
}
//API to get all bookings
export const getAllBookings = async (req, res)=>{
   try{
      const bookings = await Booking.find({}).populate('user').populate({
         path: 'show',
         populate: {path: "movie"}
      }).sort({createdAt: -1});
      res.json({success: true, bookings});
   } catch (error){
      console.error(error);
      res.json({success: false, message: error.message});
}
```

Movies Display List Component

```
const Movies = () => {
  const {shows, searchTerm} = useAppContext();
  const options = {
    keys: ["title", "description", "genre"], // fields to search in
    threshold: 0.4, // 0.0 = exact match, 1.0 = very fuzzy
    distance: 100,  // how far in the text it can search
    ignoreLocation: true,
  };
  let filteredMovies = shows;

  if (searchTerm) {
    const fuse = new Fuse(shows, options);
    const results = fuse.search(searchTerm);
    filteredMovies = results.map(result => result.item); // extract items
  }
  return filteredMovies.length > 0 ? (
     <div className='relative px-6 md:px-16 lg:px-40 xl:px-44 min-h-[80vh] overflow-
hidden'>
      <BackGradientRed top='-150vh' right='50vh'/>
      <BackGradientBlue top='30vh' left='50vh'/>
      <h1 className="text-lg font-medium  mt-35 my-4">Now Showing</h1>
      <div className="flex flex-wrap max-sm:justify-center gap-5 mb-20">
       {filteredMovies.map((movie)=>(<MovieCard movie={movie} key={movie._id}/>))}
      </div>
    </div>
  ) : (
    <div>
      <h1 className="text-3xl font-bold text-center">No Movies Found</h1>
    </div>
```

Seat Booking Component

```
return show ? (
    <div className='flex flex-col md:flex-row px-6 md:px-16 lg:px-40 py-30 md:pt-50 '>
      {/* Available timings */}
       <div className='w-60 bg-[#3B0000]/60 rounded-lg py-10 h-max md:sticky md:top-
30 '>
        <p className='text-lg font-bold px-6'>Available Timings</p>
        <div className='mt-5 space-y-1'>
          {show.dateTime[date]?.map(item =>
            (
           <div
             key={item.time}
             onClick={() => setSelectedTime(item)}
                 className={`flex items-center gap-2 px-6 py-2 w-max rounded-r-md cursor-
pointer transition
                  ${selectedTime?.time === item.time ? 'bg-white text-[#3B0000]' : 'hover:bg-
white hover:text-[#3B0000]'}`}
           >
```

```
            <ClockIcon className='w-4 h-4' />
            <p className='text-sm'>{isotimeformat(item.time)}</p>
          </div>
        ))}
      </div>
    </div>
    {/* Seats Layout */}
    <div className='relative flex-1 flex flex-col items-center max-md:mt-16'>
      <BackGradientRed top='-150vh' right='50vh' />
      <BackGradientBlue top='-10vh' left='20vh' />
      <h1 className='text-2xl font-semibold mb-4'>Select Your Seat</h1>
      <img src={assets.screen} alt='screen' className='h-12 w-auto' />
      <div className='flex flex-col items-center mt-10 text-xs text-gray-300'>
        <div className='grid grid-cols-2 md:grid-cols-1 gap-8 md:gap-2 mb-6'>
          {groupRows[0].map(row => renderSeats(row))}
        </div>
         <div className='grid grid-cols-2 gap-11'>
          {groupRows.slice(1).map((group, idx)=>(
          <div key={idx}>
            {group.map(row => renderSeats(row))}
          </div>
          ))}
        </div>
      </div>
     <button onClick={bookTickets}  className='flex items-center gap-1 mt-20 px-10 py-3 text-sm font-bolder bg-primary
      hover:bg-white hover:text-red-500 hover:font-bold transition text-white rounded-full font-medium cursor-pointer active:scale-95'>
        Proceed to checkout
        <ArrowRightIcon strokeWidth={3} className='w-4 h-4' />
        </button>
    </div>
  </div>
 ) : (<Loading />)
}
```

Navbar & Footer Component

```
const Footer = () => {

 return (
  <footer className="px-6 pt-8 md:px-16 lg:px-36 w-full text-gray-300">
        <div className="flex flex-col md:flex-row justify-between w-full gap-10 border-b border-gray-500 pb-10">
          <div className="md:max-w-150">
            <img alt="logo" className="h-25 -mx-11" src={assets.MainLogo} />
            <p className="mt-5 text-sm mb-5">
              Hamro Cinema has been the industry's standard cinema watching since the 2025s, when the gap between the entertainment came across and make people's life work loaded, Harmro Cinema started granting entertainment services.
            </p>
            <div className="flex items-center gap-2 mt-8">
```

```
                <img src={assets.googleplayic} alt="playstore" className="h-10 w-auto"
/>
                <img src={assets.appstoreic} alt="appstore" className="h-10 w-auto" />
              </div>
            </div>
            <div className="flex-1 flex items-start md:justify-end gap-20 md:gap-40">
              <div>
                <h2 className="font-semibold mb-5">Hamro Cinema</h2>
                <ul className="text-sm space-y-2">
                  <li><a href="#">Home</a></li>
                  <li><a href="#">About us</a></li>
                  <li><a href="#">Contact us</a></li>
                  <li><a href="#">Privacy policy</a></li>
                </ul>
              </div>
              <div>
                <h2 className="font-semibold mb-5">Get in touch</h2>
                <div className="text-sm space-y-2">
                  <p>+977-9862912069</p>
                  <p>hamrocinema@gmail.com</p>
                </div>
              </div>
            </div>
          </div>
          <p className="pt-4 text-center text-sm pb-5">
            Copyright {new Date().getFullYear()} © HamroCinema. All Right Reserved.
          </p>
        </footer>
  )
}
const Navbar = () => {
    const [isOpen, setIsOpen] = useState(false)
    const {user} = useUser()
    const {openSignIn} = useClerk()
    const navigate = useNavigate()
    const { setSearchTerm } = useAppContext();
    const [showSearch, setShowSearch] = useState(false);
    const [input, setInput] = useState('');
    const {favouriteMovies} = useAppContext();
    const handleSearch = (e) => {
        e.preventDefault();
        setSearchTerm(input);
        setShowSearch(false);
        navigate('/movies');
    };
    return (
      <div className='fixed  w-full lg:w-[190vh] top-0 left-0 z-50 flex items-center justify-
between px-6 md:px-16 lg:px-36 py-2 -mx-5 md:-mx-20 md:-my-0 lg:backdrop-blur-[5px]
lg:border-white/10 lg:border-[2px] gap-2 '>
        <div>
```

```
        <Link to='/' className='max-md:flex-1'>
            <img onClick={()=>{navigate('/');scrollTo(0,0)}} src={assets.MainLogo}
alt='logo' className='w-70 h-auto cursor-pointer' />
        </Link>
      </div>
      <div/>
        <div className={`max-md:absolute max-md:top-0 max-md:left-0 max-md:font-
medium
        md:text-lg z-50 flex flex-col md:flex-row items-center lg:mx-20
        max-md:justify-center gap-10 min-md:px-8 py-3 max-md:h-screen
        min-md:rounded-full backdrop-blur-[90px] bg-black md:bg-black/30 md:border
         border-gray-300/20 overflow-hidden transition-[width] duration-300 ${isOpen ?
'max-md:w-full' : 'max-md:w-0'}`}>

          <XIcon className ='md:hidden absolute top-6 right-6 w-6 h-6 cursor-pointer '
onClick={() => setIsOpen(false)}/>
          <Link onClick={()=>{scrollTo(0,0); setIsOpen(false)}} className='hover:text-
red-500 hover:scale-105' to='/'>Home</Link>
          <Link onClick={()=>{scrollTo(0,0); setIsOpen(false)}} className='hover:text-
red-500 hover:scale-105' to='/movies'>Movies</Link>
        {/* <Link onClick={()=>{scrollTo(0,0); setIsOpen(false)}} className='hover:text-
red-500 hover:scale-105' to='/'>Theater</Link> */}
          <Link onClick={()=>{scrollTo(0,0); setIsOpen(false)}} className='hover:text-
red-500 hover:scale-105' to='/movies'>Releases</Link>
              {favouriteMovies.length>0 && <Link onClick={()=>{scrollTo(0,0);
setIsOpen(false)}} className='hover:text-red-500 hover:scale-105'
to='/favourite'>Favourite</Link>}
      </div>
      <div className='flex items-center gap-8'>
          <SearchIcon className='hidden lg:block w-8 h-8 cursor-pointer hover:text-red-
500 md:text-white'
        onClick={()=>setShowSearch(!showSearch)} />
        {showSearch && (
          <div className="absolute top-20 right-10 bg-white rounded shadow-lg p-2 flex
text-black">
      <input
        type="text"
        value={input}
        onChange={e => {
          setInput(e.target.value);
          setSearchTerm(e.target.value); // Live search on every keystroke
          navigate('/movies'); // Optional: navigate to movies page on typing
        }}
        placeholder="Search movies..."
        className="px-2 py-1 border rounded"
        autoFocus
      />
    </div>
        )}
        {
```

```
            !user ? (
                <button onClick={openSignIn} className='px-4 py-1 sm:px-7 sm:py-2 bg-
primary md:hover:bg-white
                md:border-1 hover:text-primary border:white border-white transition
                rounded-full lg:font-medium cursor-pointer'>LogIn</button>
            ) : (
                <UserButton>
                    <UserButton.MenuItems>
                            <UserButton.Action label="My Bookings" labelIcon={<TicketPlus
width={15}/>} onClick={() => navigate('/my-bookings')}/>
                    </UserButton.MenuItems>
                </UserButton>
            )
          }
      </div>
            <MenuIcon className='max-md:m1-4 md:hidden w-8 h-8 cursor-pointer'
onClick={() => setIsOpen(!isOpen)}/>
    </div>
  )
}
```

Movie Card Component

```
const MovieCard = ({movie}) =>{

   const navigate = useNavigate()
   const {image_base_url} = useAppContext();
   return (
<div className='flex flex-col justify-between p-3 bg-gray-800 rounded-2xl
hover:-translate-y-1 transition duration-300 w-66'>
          <img     onClick={()=>{navigate(`/movie/${movie._id}`);     scrollTo(0,     0)}}
src={image_base_url + movie.backdrop_path} alt=""
   className='rounded-lg h-52 w-full
   object-cover object-right-bottom cursor-pointer'/>
   <p className='font-semibold mt-2 truncate text-gray-300'>{movie.title}</p>
   <p className='text-sm text-gray-400 mt-2'>
      {new Date(movie.release_date).getFullYear()} . {movie.genres.slice(0, 2).map(genre
=> genre.name).join(" | ")} . {timeFormat(movie.runtime)}
   </p>
   <div className='flex items-center justify-between mt-4 pb-3'>
      <button onClick={()=>{navigate(`/movie/${movie._id}`); scrollTo(0, 0)}}
           className='px-4 py-2 text-xs text-gray-100 bg-red-500 hover:bg-gray-100
hover:text-red-500 transition
      rounded-full font-medium cursor-pointer'>Buy Tickets</button>
      <p className='flex items-center gap-1 text-sm text-gray-400 mt-1 pr-1'>
         <StarIcon className="w-4 h-4 text-[#F5C518] fill-[#F5C518]" />
         {movie.vote_average.toFixed(1)}
      </p>
   </div>
```

Admin Add-Show Component

```
const AddShows = () => {

  const{axios, getToken, user, image_base_url} = useAppContext()
  const currency = import.meta.env.VITE_CURRENCY;
  const [nowPlayingMovies, setNowPlayingMovies] = useState([]);
  const [selectedMovie, setSelectedMovie] = useState(null);
  const [dateTimeSelection, setDateTimeSelection] = useState({});
  const [dateTimeInput, setDateTimeInput] = useState("");
  const [showPrice, setShowPrice] = useState("");
  const [addingShow, setAddingShow] = useState(false);
  // Fetch movies
  const fetchNowPlayingMovies = async () => {
    try{
        const {data} = await axios.get('/api/show/now_playing',{headers: {Authorization:
`Bearer ${await getToken()}`}})
      if(data.success){
        setNowPlayingMovies(data.movies)
      }
    } catch(error){
      console.error('Error Fetching movies:', error);
    }
  };
  // Add datetime to selection
  const handleDateTimeAdd = () => {
    if (!dateTimeInput) return;
    const [date, time] = dateTimeInput.split("T");
    if (!date || !time) return;
    setDateTimeSelection((prev) => {
      const times = prev[date] || [];
      if (!times.includes(time)) {
        return { ...prev, [date]: [...times, time] };
      }
      return prev;
    });
    setDateTimeInput(""); // reset input
  };
  // Remove a specific time
  const handleRemoveTime = (date, time) => {
    setDateTimeSelection((prev) => {
      const filteredTimes = prev[date].filter((t) => t !== time);

      if (filteredTimes.length === 0) {
        const { [date]: _, ...rest } = prev;
        return rest;
      }

      return { ...prev, [date]: filteredTimes };
    });
  };
```

```
  const handleSumbit = async() => {
    try{
      setAddingShow(true)
      if(!selectedMovie || Object.keys(dateTimeSelection).length === 0 || !showPrice){
        return toast('Missing required fields');
      }
        const showsInput = Object.entries(dateTimeSelection).map(([date, time])=>({date,
time}));
      const payload = {
        movieID : selectedMovie,
        showsInput,
        showPrice: Number(showPrice)
      }
        const {data} = await axios.post('/api/show/add', payload, {headers: {Authorization:
`Bearer ${await getToken()}`}})
      if(data.success){
        toast.success('Show added successfully')
        setSelectedMovie(null)
        setDateTimeSelection({})
        setShowPrice('')
      } else {
        toast.error(data.message)
      }
    } catch (error){
      console.error('Submission Error:', error);
      toast.error('An error occured pls try again')
    }
    setAddingShow(false)
  }
  useEffect(() => {
    if(user){
      fetchNowPlayingMovies();
    }
  }, [user]);
  return nowPlayingMovies.length > 0 ? (
    <>
      <Title text1="Add " text2="Shows" />
      {/* Movies Selection */}
      <p className="mt-10 text-lg font-medium">Now Playing Movies</p>
      <div className="overflow-x-auto pb-4">
        <div className="group flex flex-wrap gap-4 mt-4 w-max">
          {nowPlayingMovies.map((movie) => (
            <div
              key={movie.id}
              className={`relative max-w-40 cursor-pointer transition duration-300
                group-hover:opacity-40 hover:opacity-100 hover:-translate-y-1 ${
                  selectedMovie === movie.id ? "ring-2 ring-primary" : ""
                }`}
              onClick={() => setSelectedMovie(movie.id)}
            >
```

```
          <div className="relative rounded-lg overflow-hidden">
            <img
              src={image_base_url + movie.poster_path}
              alt={movie.title}
              className="w-full object-cover brightness-90"
            />
            {/* Rating & Votes */}
             <div className="text-sm flex items-center justify-between p-2 bg-black/70 w-
full absolute bottom-0 left-0">
              <p className="flex items-center gap-1 text-gray-400">
                <StarIcon className="w-4 h-4 text-IMDB fill-IMDB" />
                {movie.vote_average.toFixed(1)}
              </p>
              <p className="text-gray-300">
                {kconverter(movie.vote_count)} votes
              </p>
            </div>
            {/* Selected Check */}
            {selectedMovie === movie.id && (
                <div className="absolute top-2 right-2 flex items-center justify-center bg-
primary h-6 w-6 rounded">
                <CheckIcon className="w-4 h-4 text-white" strokeWidth={2.5} />
              </div>
            )}
          </div>
          <p className="font-medium truncate">{movie.title}</p>
          <p className="text-gray-400 text-sm">{movie.release_date}</p>
        </div>
      ))}
    </div>
  </div>
  {/* Show Price Input */}
  <div className="mt-8">
    <label className="block text-sm font-medium mb-2">Show Price</label>
     <div className="inline-flex items-center gap-2 border border-gray-600 px-3 py-2
rounded-md">
      <p className="text-gray-400 text-sm">{currency}</p>
          <input min={0} type="number" value={showPrice} onChange={(e) =>
setShowPrice(e.target.value)}
        placeholder="Enter show price" className="outline-none bg-transparent" />
    </div>
  </div>
  {/* Date & Time Selection */}
  <div className="mt-6">
    <label className="block text-sm font-medium mb-2">Select Date & Time</label>
    <div className="inline-flex gap-5 border border-gray-600 p-1 pl-3 rounded-lg">
          <input type="datetime-local" value={dateTimeInput} onChange={(e) =>
setDateTimeInput(e.target.value)}
        className="outline-none rounded-md" />
         <button onClick={handleDateTimeAdd} className="bg-primary/80 text-white
```

```
          px-3 py-2 text-sm rounded-lg hover:bg-primary cursor-pointer">
          Add Time
        </button>
      </div>
    </div>
    {/* Display selected times */}
    {Object.keys(dateTimeSelection).length > 0 && (
      <div className="mt-6">
        <h2 className="mb-2">Selected Date-Time</h2>
        <ul className="space-y-3">
          {Object.entries(dateTimeSelection).map(([date, times]) => (
            <li key={date}>
              <div className="font-medium">{date}</div>
              <div className="flex flex-wrap gap-2 mt-1 text-sm">
                {times.map((time) => (
                  <div key={time} className="border border-primary px-2
                  py-1 flex items-center rounded">
                    <span>{time}</span>
                    <DeleteIcon onClick={() => handleRemoveTime(date, time)}
                        width={15} className="ml-2 text-red-500 hover:text-red-700
cursor-pointer"/>
                  </div>
                ))}
              </div>
            </li>
          ))}
        </ul>
      </div>
    )}
      <button onClick={handleSumbit} disabled={addingShow} className="bg-primary
text-white px-8 py-2 mt-6 rounded-md hover:bg-primary/90 transition-all cursor-pointer">
      Add Show
    </button>
  </>
 ) : (
  <Loading />
 );
};
```