

JAIFLIX PROJECT

Submitted by

JAYASHRI M

Reg No:1P23MC024

In partial fulfillment of the requirement for the award of the Degree of

Master of Computer Applications

Bharathiar University

Coimbatore

Under the Internal Supervision of

Ms.K.Narmatha MCA.,

Assistant Professor

School of Computer Studies-MCA



Master of Computer Applications,

RVS College of Arts and Science (Autonomous),

Sulur, Coimbatore – 641 402

Nov 2024

RVS COLLEGE OF ARTS AND SCIENCE(AUTONOMOUS)
AFFILIATED TO BHARATHIAR UNIVERSITY
NAAC Re-accredited & ISO
9000:2008 Certified SULUR,
COIMBATORE-641402.



Nov 2024

Register Number – 1P23MC024

Certified *bonafide* original record of work done by **JAYASHRI M**

Internal Supervisor

Director

Submitted for the project Evaluation and *Viva-Voce* held on.....

Internal Examiner

External Examiner

DECLARATION

I, **JAYASHRI M**, hereby declare that the project entitled **JAIFLIX**, submitted to the School of Computer Studies, RVS College of Arts and Science, in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications is a record of original project work done by me during the period Jun 2024 to Nov 2024 under the internal supervision of **Ms.K.NARMATHA MCA., ASSISTANT PROFESSOR, RVS COLLEGE OF ARTS AND SCIENCE (AUTONOMOUS)** From Bharathiar University, Coimbatore.

JAYASHRI M

Signature of the Candidate

ACKNOWLEDGEMENTS

I express my sincere thanks to our Managing trustee **Dr.K.Senthil Ganesh MBA (USA), MS (UK), Ph.D.**, for providing us with the adequate faculty and laboratory resources for completing my project successfully.

I take this as a fine opportunity to express my sincere thanks to **Dr.T.Sivakumar M.Sc., M. Phil., Ph.D., Principal**, RVS College of Arts And Science (Autonomous) for giving me the opportunity to undertake this project.

I express my sincere thanks to **Dr.P.Navaneetham M.Sc., M.Phil., Ph.D., Director (Administration), Department of Computer Science** for the help and advice throughout the project.

I express my sincere thanks to **Dr.S.Yamini M.Sc.,(CC), M. Phil., Ph.D., Director (Academic), Department of Computer Science** for her valuable guidance and prompt correspondence throughout the curriculum to complete the project.

I express my gratitude to **Ms.K.Narmatha MCA., Assistant Professor** for his valuable guidance, support, encouragement and motivation rendered by her throughout this project.

Finally, I express my sincere thanks to all other staff members and my dear friends, and all dear and near for helping me to complete this project.

JAYASHRI M

ABSTRACT

Jaiflix is a mobile application developed using React Native, complemented by an admin panel built with Node.js. This platform offers users a seamless streaming experience for movies and TV shows, featuring user authentication, content lists, and a fully functional video player equipped with advanced controls for audio, subtitles, and video progress tracking.

The admin panel, developed using Node.js, works with MongoDB and TMDb (The Movie Database) to manage content. Administrators can easily add, update, or delete movies and TV shows, track user activity, and organize content by genres and providers.

In the future, Jaiflix could include features like personalized recommendations, social features such as watch parties, offline viewing, and analytics to track app performance. Expanding support to smart TVs and gaming consoles will make the platform more accessible to a wider audience.

CONTENTS

- I. Declaration
- II. Acknowledgements
- III. Abstract

CHAPTER 1

1. Introduction	01
1.1 An overview of the project	01
1.2 Mission of the project	01
1.3 Background study	02
1.3.1 A study of the existing system	02

CHAPTER 2

2. System Analysis	03
2.1 A study of the proposed system	03
2.2 User requirement specification	03
2.2.1 Major modules	03
2.2.2 Sub modules	04
2.3 Software requirement specification	04
2.4 System specification	05
2.4.1 React Native	05
2.4.2 React.js	05
2.4.3 Node.js	05
2.4.4 Express.js	05
2.4.5 MongoDB	05
2.4.6 Hardware configuration	06
2.4.7 Software configuration	06

CHAPTER 3

3. System design and development	07
3.1 Fundamentals of design concepts	07
3.1.1 Abstraction	07
3.1.2 Refinement	07
3.1.3 Modularity	07
3.2 Design Notations	08
3.2.1 System structure chart	08
3.2.2 System flow diagram	09
3.2.3 Data flow diagram / UML	10
3.2.4 Software Engineering Model	11
3.3 Design process	11
3.3.1 Software Architecture	11
3.3.2 Control Hierarchy	12
3.3.3 Structural Partitioning	12
3.3.4 Data structure	12
3.3.5 Software procedure	13
3.4 Database design	14
3.5 Input design	15
3.6 Output design	15
3.7 Development approach	15

CHAPTER 4

4. Testing and implementation	16
4.1 Testing	16
4.1.1 System testing	16
4.1.2 White box testing	17
4.1.3 Unit testing	17
4.2 System implementation	17
4.2.1 System maintenance	18

CHAPTER 5

5. Conclusion	20
5.1 Directions for future enhancements references	20

CHAPTER 6

6. References	
6.1 BIBLIOGRAPHY	21
6.2 Book References	21

ANNEXURE

ANNEXURE - A - Output design	22-32
ANNEXURE - B - Source code	33-46

CHAPTER-1

1. Introduction

This document outlines the development and architecture of **Jaiflix**, a mobile streaming application designed to deliver a seamless entertainment experience for users. Built using the React Native framework for the frontend and Node.js for the backend, Jaiflix offers rich features like movie and TV show browsing, user account management, video playback, and content management via an admin panel. The application integrates with TMDB (The Movie Database) to provide accurate and up-to-date movie information and implements MongoDB for data storage.

1.1 An Overview of the Project

The goal of Jaiflix is to create an engaging and functional movie streaming platform that allows users to browse and watch content effortlessly while giving administrators full control over content management. The application features user authentication, personalized movie lists, and a customizable video player. On the admin side, the panel allows adding, updating, deleting, and organizing movies and TV shows. The Node.js powered API manages the communication between the database and the frontend, ensuring smooth content delivery.

1.2 Mission of the Project

The mission of Jaiflix is to provide an immersive and personalized streaming experience that meets user expectations for convenience, content variety, and ease of use. By building a robust admin panel, the project also seeks to provide administrators with a simplified content management process. This application aims to streamline the entertainment process for users while offering an efficient way for content providers to manage and distribute their media.

Key objectives include:

- Delivering a high-quality streaming experience.
- Providing flexible content management tools for administrators.
- Ensuring security and privacy for users during content access and playback.

1.3 Background Study

Before developing Jaiflix, an analysis of existing streaming platforms was conducted to understand key features, user expectations, and the technical challenges involved in building a large-scale video streaming service. The study also explored user preferences for browsing, searching, and interacting with streaming content, as well as the backend architecture necessary to support real-time content updates, video streaming, and user authentication.

1.3.1 A Study of the Existing System

- 1. Documenting Existing Streaming Platforms:** We examined popular streaming platforms like Netflix, Amazon Prime Video, and Hulu to understand their UI/UX design, video player functionality, and content management processes. This study covered the user registration process, movie browsing, and video playback experience. We diagrammed the flow of data from the API to the user interface.
- 2. User Feedback and Preferences:** Surveys and interviews with users of streaming platforms provided insights into user preferences regarding search functionality, personalized recommendations, and video quality. Common user complaints, such as buffering issues and limited content availability, were also noted.
- 3. Performance and Scalability Analysis:** The technical infrastructure of existing streaming systems was studied to understand how these platforms handle large volumes of users and simultaneous video playback. The study identified the need for efficient database querying, real-time content synchronization, and scalability to prevent slowdowns or outages during peak usage periods.

This analysis provided the foundation for designing Jaiflix, ensuring that it incorporates best practices from established systems while addressing common user pain points.

CHAPTER-2

2. System Analysis

This section outlines the in-depth analysis of Jaiflix, focusing on its proposed system design, user requirements, software components, and hardware and software specifications.

2.1 A Study of the Proposed System

Jaiflix is a dynamic streaming platform that offers a seamless user experience for streaming movies and TV shows. The system integrates a React Native mobile app for the front end and a Node.js-based admin panel for managing content. The system ensures real-time updates, smooth content delivery, and efficient content management through TMDB integration.

- **User Perspective:** Users can browse, watch, and manage their movie and TV show lists, while also having access to a personalized viewing experience.
- **Admin Perspective:** Admins have full control over content management, including adding, updating, and deleting movies or TV shows, and retrieving details from TMDB.

2.2 User Requirement Specification

The user requirements are focused on two types of users: administrators and end-users. Administrators need an intuitive interface to manage the media library, including the ability to add, update, and delete movies and TV shows. End-users require a simple, interactive interface to browse, search, and stream content, while having access to personalized features like user profiles, watchlists, and movie recommendations.

2.2.1 Major Modules

1. **User Authentication Module:** Includes user registration, login, and session management functionalities.
2. **Movie/TV Show Browsing Module:** Allows users to browse available content by category, genre, and popularity.
3. **Video Playback Module:** Includes video streaming with volume, brightness controls, and advanced features like gesture support.

4. **My List Module:** Enables users to create personalized lists of movies and shows.
5. **Admin Content Management Module:** Facilitates adding, updating, and deleting movies and TV shows, with the ability to fetch data from TMDB.
6. **User Preferences Module:** Stores user settings like favorite genres and viewing history.

2.2.2 Sub Modules

1. **Search and Filter Sub-Module:** Allows users to search content by movie/TV show name, genre, and release year.
2. **Movie Details Sub-Module:** Displays detailed information for each movie or TV show, including description, cast, and genres.
3. **Watch Progress Sub-Module:** Tracks and saves the user's progress in watched content, enabling easy resumption.
4. **Admin Dashboard Sub-Module:** Provides an overview of content and user activity, along with CRUD operations for movies and shows.
5. **Genre and Provider Sub-Module:** Admin can assign genres and providers for each movie or show.

2.3 Software Requirement Specification

Jaiflix requires a combination of both frontend and backend technologies to provide a smooth and scalable experience for both users and admins.

- **Frontend (Mobile App):**
 - **Framework:** React Native
 - **Components:** Custom components for UI, FlatList for displaying movie banners, and a video player for content streaming.
 - **Libraries:** React Navigation for routing and React Native Video for the video player.
- **Backend (Admin Panel & API):**
 - **Platform:** Node.js
 - **Database:** MongoDB for storing user data and content information.
 - **Middleware:** Express.js for handling API requests.
 - **Authentication:** Passport.js for session management and authentication.

2.4 System Specification

2.4.1 React Native

- **Framework:** React Native enables the development of cross-platform mobile applications, ensuring a consistent user experience on both iOS and Android.
- **Components** Utilizes functional components leveraging hooks to promote modularity and reusability, enhancing the maintainability of the codebase.

2.4.2 React.js

- **Framework:** React.js powers the admin panel's user interface, allowing dynamic, responsive, and scalable admin tools for managing the platform.
- **Components:** Functional components using hooks, with a focus on modularity and reuse across the application.
- **State Management:** Redux or Context API for managing the application's global state.

2.4.3 Node.js

- **Platform:** Node.js acts as the backend for the admin panel and API endpoints. It ensures efficient server-side processing and communication with MongoDB.
- **Performance:** Asynchronous and non-blocking I/O for handling high user traffic efficiently.

2.4.4 Express.js

- **Middleware:** Express.js serves as the backbone for API creation, handling all HTTP requests, routing, and connecting to MongoDB for CRUD operations.
- **Endpoints:** Secure endpoints for user authentication, movie management, and fetching movie data from TMDB.

2.4.5 MongoDB

- **Database:** MongoDB stores data related to users, movies, and TV shows. It is a NoSQL database, which ensures scalability and flexibility in handling the large volume of data.
- **Schema Design:** The data schema includes users, movies, genres, providers, and session tracking for users' watched progress.

2.4.6 Hardware Configuration

The hardware configuration needed for Jaiflix includes a standard development machine with at least:

- **Processor:** Intel Core i5 or above
- **RAM:** 8GB or more
- **Storage:** 500GB SSD or more

2.4.7 Software Configuration

- **Operating Systems:** Windows 10/11 for development.
- **Development Tools:**
 - **Android Studio** for building and testing mobile applications.
 - **Visual Studio Code** for code development.
- **Version Control:** Git for code management and collaboration.
- **Package Managers:** npm (Node Package Manager) for handling dependencies such as Express, Mongoose, and React Native libraries.
- **Database:** MongoDB installed locally or remotely via MongoDB Atlas

These specifications ensure that the Jaiflix system is capable of delivering high performance, scalability, and ease of management for both users and admins.

CHAPTER-3

3. System Design and Development

This section explains the system design concepts and methodologies used in the Jaiflix project, including design principles, notations, architecture, and data structures. It focuses on structuring the system for efficiency, scalability, and maintainability.

3.1 Fundamentals of Design Concepts

3.1.1 Abstraction

- **Definition:** Abstraction involves simplifying complex systems by modeling classes based on essential properties and behaviors while hiding unnecessary details.
- **Application in Jaiflix:** In Jaiflix, abstraction is used to create models for users, movies, and genres. The complexity of data interactions is hidden from the user, who interacts with a straightforward interface to manage and view content.

3.1.2 Refinement

- **Definition:** Refinement refers to the iterative process of enhancing the design and functionality of the system.
- **Application in Jaiflix:** Throughout the development of Jaiflix, feedback is collected from users and administrators, leading to iterative enhancements. This process ensures that features such as the video player, user authentication, and content management evolve based on user experience and needs.

3.1.3 Modularity

- **Definition:** Modularity involves dividing the system into smaller, manageable modules that can be developed, tested, and maintained independently.
- **Application in Jaiflix:** The system is modularized into distinct components, including the admin panel, user authentication, movie management, and video playback. Each module can be updated independently without affecting the overall system.

3.2 Design Notations

3.2.1 System Structure Chart

A system structure chart outlines the hierarchical organization of the different modules within Jaiflix. It shows how modules such as Admin Panel, User Login, and Movie API interact with each other, emphasizing the dependencies between them.

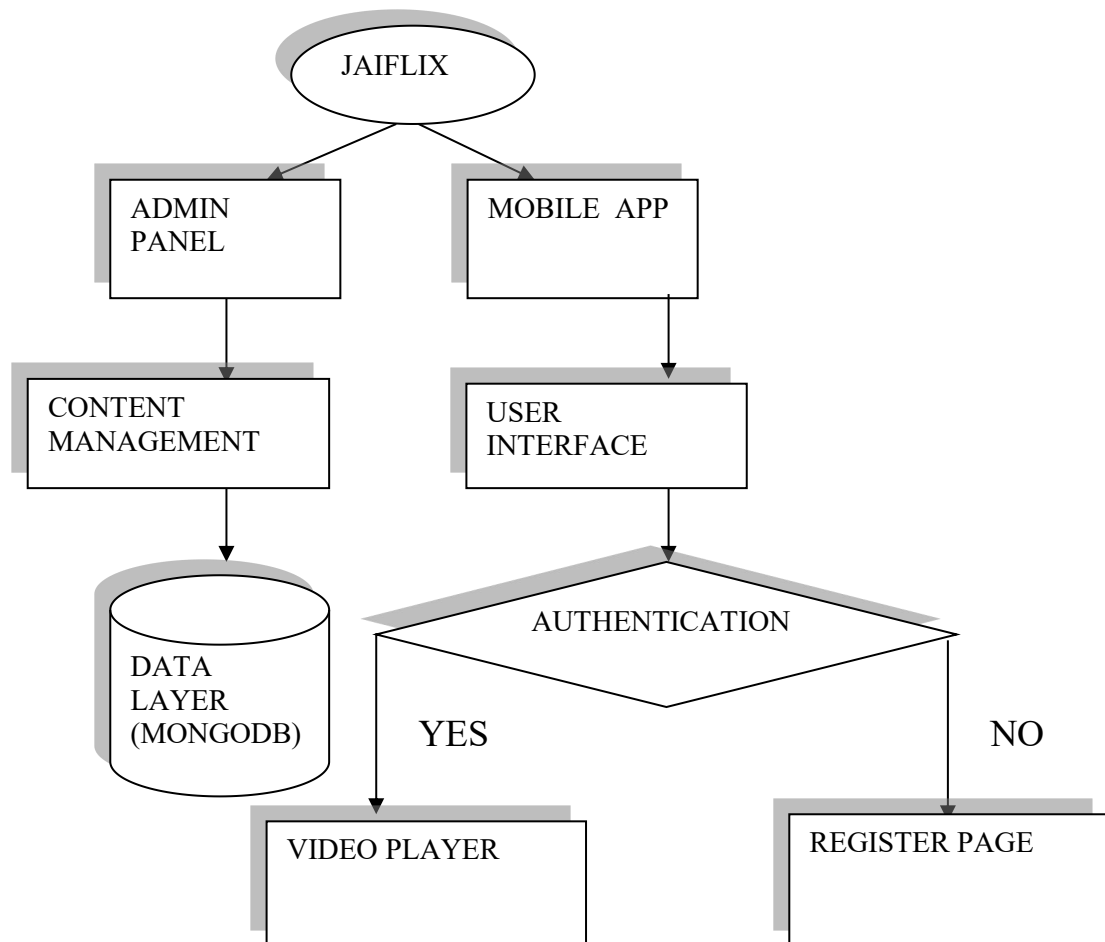


Figure 3.1 System Structure Chart

3.2.2 System Flow Diagram

The system flow diagram illustrates the flow of information and the sequence of processes that occur within the Jaiflix system, including how user requests are handled by the backend server and how data is retrieved from TMDB and MongoDB.

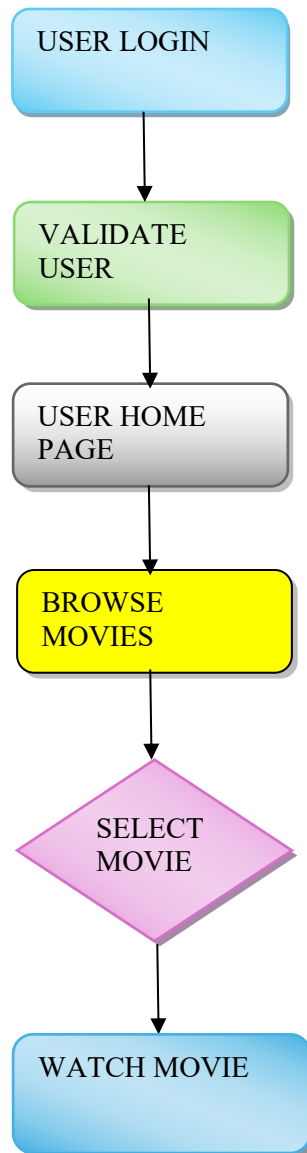


Figure 3.2 System Flow Diagram

3.2.3 Data Flow Diagram / UML

The Jaiflix application involves several external entities, processes, data stores, and data flows that create an integrated system for both users and administrators. External entities include the User, represented by a circle, who interacts with the Jaiflix app, and the Admin, also represented by a circle, who manages the platform through the Admin Panel. Key processes within the system are depicted as rounded rectangles, including User Login & Authentication, which handles the validation of user credentials; Browse & Select Movie, allowing users to explore and choose movies; Watch Movie, which facilitates the streaming of video content to users; and Admin Content Management, where administrators add, update, or remove movie details.

The system utilizes several data stores, represented by open rectangles, to manage information. The User Data Store holds user profiles, login credentials, and preferences; the Movie Data Store contains all movies and TV shows fetched from TMDB and local sources; and MongoDB serves as the primary database for storing everything from user profiles to movie information and watch history. Data flows through the system via arrows that indicate the movement of information: Users submit their login credentials to the Login & Authentication process, which accesses the User Data Store for validation. Users browse available movies, prompting the Browse process to fetch details from the Movie Data Store. Upon selecting a movie, the Watch Movie process retrieves the media file from the database. Additionally, the Admin Panel allows administrators to update movie information, with changes reflected in the Movie Data Store through the Admin Content Management process. This cohesive structure ensures a streamlined user experience and effective content management.

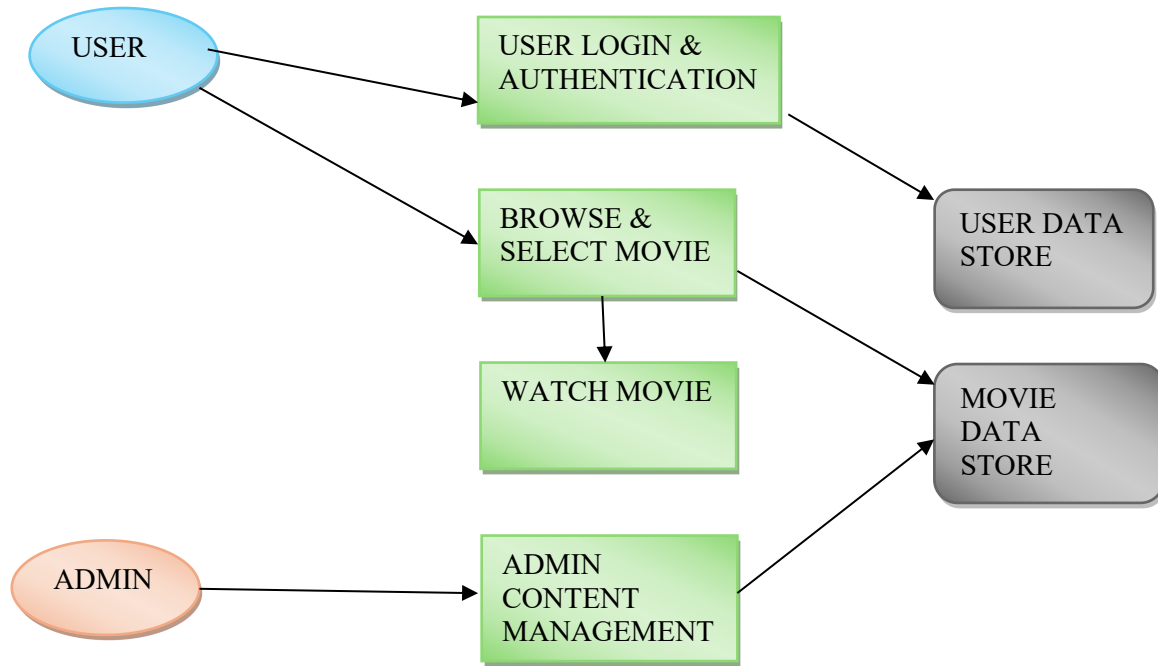


Figure 3.3 Data Flow Diagram

3.2.4 Software Engineering Model

The software engineering model adopted for Jaiflix is the **MVC (Model-View-Controller)** architecture.

- **Model:** Contains the data layer, defining data structures and handling database interactions via MongoDB.
- **View:** Consists of the user interface built with React Native for the mobile app and React.js for the admin panel.
- **Controller:** Acts as the intermediary, processing user requests, interacting with the model, and updating the view as needed.

3.3 Design Process

3.3.1 Software Architecture

- **Architecture Pattern:** Use a **Microservices Architecture** that separates the application into distinct services for the admin panel and mobile app, allowing for scalability and maintainability.
- **Frontend:** Developed using **React Native**, providing a responsive and user-friendly interface.

- **Backend:** Built with **Node.js** and **Express**, responsible for handling API requests and managing server-side logic.
- **Database:** Utilize **MongoDB** for data storage, ensuring flexibility in data management and retrieval.

3.3.2 Control Hierarchy

- **User Level:** Basic users (viewing content, managing their profile).
- **Admin Level:** Admins (content management, user management).
- **System Level:** System processes (handling API requests, data synchronization, etc.).
- **Flow:** The control flow should follow a top-down approach, where requests from users are handled by the admin or system layers.

3.3.3 Structural Partitioning

- **Frontend Components:**
 - **Authentication:** Login and registration forms.
 - **Content Browsing:** Movie listing, search functionality.
 - **User Management:** Profile settings, favorites list (MyList).
 - **Video Player:** Playback controls and settings.
- **Backend Services:**
 - **User Service:** Handles user authentication and profile management.
 - **Movie Service:** Manages movie data (CRUD operations).
 - **Admin Service:** Admin functionalities for content management.

3.3.4 Data Structure

- **User Schema:**
 - `userId`: String (unique identifier)
 - `username`: String
 - `password`: String (hashed)
 - `email`: String
 - `favorites`: [MovieId] (Array of movie IDs)
- **Movie Schema:**
 - `movieId`: String (unique identifier)

- title: String
- description: String
- genres: [String] (Array of genre names)
- provider: String
- videoUrl: String (URL to the video file)

3.3.5 Software Procedure

- **User Registration:**
 1. User fills out the registration form.
 2. Data is validated and hashed.
 3. User is saved to the database.
- **User Login:**
 1. User submits login credentials.
 2. Credentials are validated against the database.
 3. If valid, create a session for the user.
- **Movie Browsing:**
 1. User accesses the home page.
 2. All available movies are fetched from the database.
 3. Movies are displayed in a scrollable format.
- **Video Playback:**
 1. User selects a movie.
 2. The video player initializes and streams the content.
 3. User interacts with playback controls.

3.4 Database Design

- **User/ Admin Table:** Contains user and Admin information (id, username, password, email, favorites).

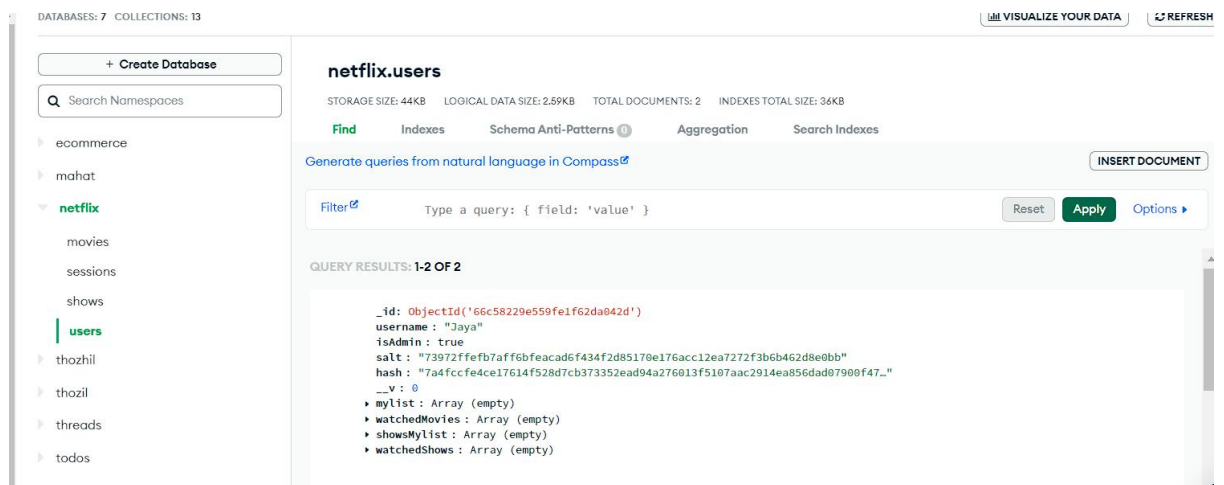


Figure 3.4 User/Admin Database

- **Movie Table:** Contains movie details (id, title, description, genres, video URL).

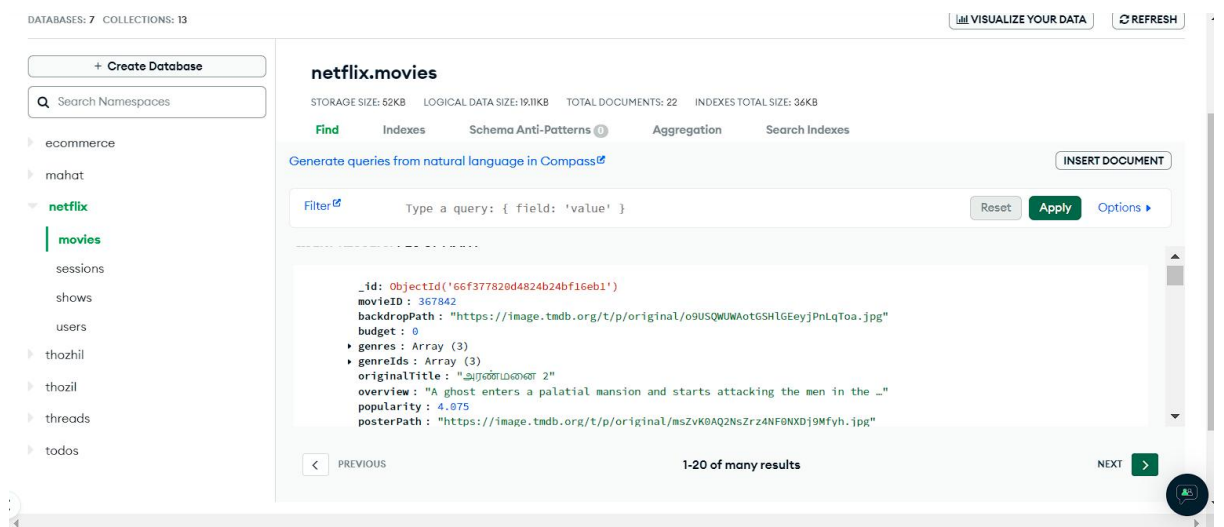


Figure 3.5 Movie Database

- **Show Table:** Contains TV Show details (id, title, description, genres, video URL).

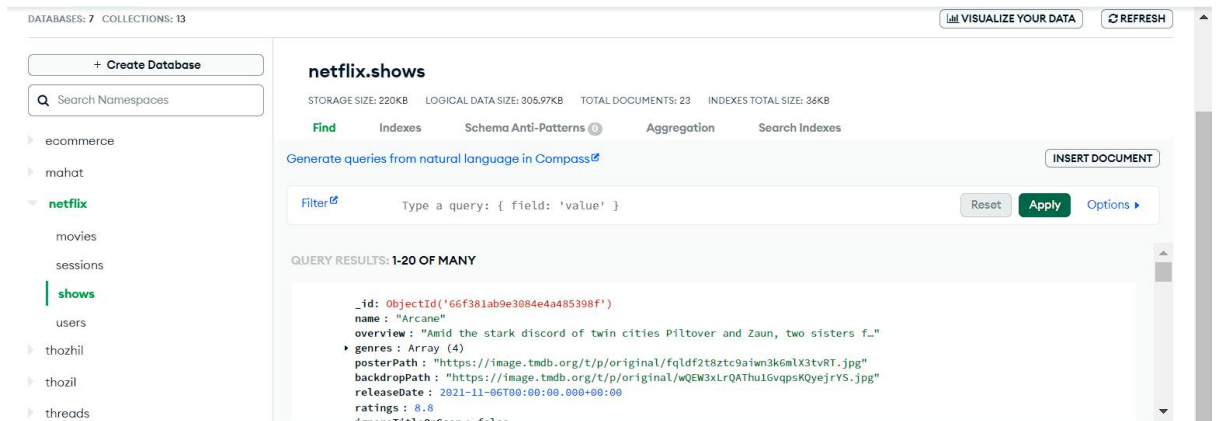


Figure 3.6 Show Database

3.5 Input Design

- **Forms:**
 - Registration and login forms should include fields for username, email, and password.
 - Movie submission forms in the admin panel for adding new movies.

3.6 Output Design

- **UI Components:** Use cards for movie display, including title, poster, and action buttons (play, add to favorites).
- **Dashboard:** Admin dashboard should summarize user and movie statistics, with charts for visual representation.

3.7 Development Approach

- **Agile Methodology:** Use iterative development to allow for flexibility and responsiveness to changes.
- **Version Control:** Implement Git for tracking changes and collaborating with team members.
- **Testing:** Establish a testing framework using Jest for unit testing and Postman for API testing.

CHAPTER-4

4. Testing and Implementation

4.1 Testing

Software testing involves the process of evaluating and verifying that a software product or system meets the specified requirements. It includes the act of examining the artifacts and behavior of the software through validation and verification. Testing offers an independent assessment, allowing businesses to understand the risks involved in deploying the software. The primary goal of testing is to identify errors and ensure that the system operates as intended.

Testing is closely intertwined with the implementation process. As the system is implemented, tests ensure that it functions without errors, validating that the implementation was correctly executed. In practice, these two phases work in tandem to deliver a successful product.

Implementation, in this context, refers to executing a plan or process and ensuring that the system runs as expected. Each step of the implementation must be tested, a process known as implementation testing.

4.1.1 System Testing

System testing is a comprehensive software testing approach that evaluates the complete and fully integrated system to ensure it meets the specified requirements. This type of testing occurs after integration testing and before acceptance testing, confirming that the system is suitable for release to end-users.

Key characteristics of system testing include:

- **Scope:** It tests both functional and non-functional aspects of the software.
- **Objective:** To detect defects in integrated units and the overall system.
- **Process:** Takes input from passed integration tests and checks for irregularities across the system components.
- **Method:** System testing is usually a black-box testing method, where the tester does not need to know the internal workings of the application. It focuses on validating the

system against the Software Requirement Specifications (SRS) and functional requirement specifications.

System testing ensures that the entire software system operates as designed. The goal is to ensure that the system performs tasks as required, aligning with the expectations of both developers and end-users. This stage of testing is typically performed by an independent testing team to maintain objectivity and impartiality in evaluating system quality.

4.1.2 White Box Testing

White box testing, also known as clear box or glass box testing, is a method where the internal structure of the software is known to the tester. Unlike black box testing, it focuses on validating internal processes and logic by examining the codebase .

- **Goal:** To test the internal workings of the software, including paths, conditions, loops, and logic.
- **Scope:** Requires the tester to have programming skills and knowledge of the internal code to test the functionality accurately.

4.1.3 Unit Testing

- **Definition:** Unit testing involves testing individual components or modules of the application to ensure they perform as expected.
- **Objectives:**
 - Isolate each part of the program and show that the individual parts are correct.
 - Facilitate easier debugging and code refactoring.
- **Approach:**
 - Write test cases for every function, method, or class to validate their correctness.
 - Utilize mocks and stubs for dependent components to focus on the unit being tested.

4.2 System Implementation

System implementation refers to executing a well-formulated plan for deploying a system or process. The success of implementation hinges on clear objectives, predefined actions, and thorough testing to ensure the system operates correctly. Each action in the implementation

process undergoes testing, known as implementation testing, to confirm that the system performs as intended.

Key Aspects of Implementation:

1. **Testing Technological Specifications:** It is critical to verify the implementation of specific technological requirements to ensure they align with user and system expectations.
2. **Improving Interfaces:** Through testing, implementation can improve user interfaces to make them more understandable and user-friendly for developers and users alike.
3. **Action Confirmation:** Implementation testing ensures that identified actions can be successfully executed without issues.
4. **Identifying Issues:** Ambiguous or conflicting actions are easily identified during implementation testing, allowing for quick resolution.
5. **Quality Control:** Testing during implementation acts as a quality check, ensuring that the system's features are correctly implemented and operational.

During this phase, the test manager collaborates with the IT infrastructure team to ensure the availability of the appropriate test environment and its configuration. Implementation also involves the realization of system components, such as:

- **System Classes:** The blueprint for how the system behaves.
- **User Interface:** Ensuring users interact with the system intuitively and effectively.
- **Database Structures:** Ensuring the data storage and retrieval system operates as designed.

4.2.1 System Maintenance

System maintenance involves ensuring the system remains operational and efficient over time. Maintenance includes two primary activities:

1. **System Maintenance:** The process of restoring the system to its original functionality by fixing bugs, improving performance, and maintaining operational consistency.
2. **System Enhancement:** The addition or modification of features based on evolving user needs or updated specifications. Enhancement expands the system's capabilities, making it adaptable to new requirements.

Types of Maintenance:

- **Corrective Maintenance:** Fixing errors found in the system after it has been deployed.

- **Adaptive Maintenance:** Modifying the system to cope with changes in the environment (e.g., new hardware or software platforms).
- **Perfective Maintenance:** Improving system performance or maintainability, even when no defects are present.
- **Preventive Maintenance:** Identifying and correcting potential issues before they occur, ensuring the system remains stable over time.

Importance of System Maintenance:

- Keeps the system functioning at an optimal level by addressing potential faults before they impact operations.
- Ensures system security and compliance with current technological standards.
- Enhances the system to meet new user requirements, extending its lifecycle and value.
- **Documentation:** Maintain comprehensive documentation that includes installation procedures, user guides, and troubleshooting instructions to assist users and future developers.

For businesses, investing in proper maintenance results in long-term cost savings . Regular updates and fixes prevent the need for expensive overhauls or replacements, keeping the system efficient and relevant.

CHAPTER-5

5. Conclusion

In conclusion, the Jaiflix project successfully replicates the essential features of a streaming platform like Netflix, providing users with a seamless experience for browsing, streaming, and managing their favorite movies and TV shows. Through a robust backend built with Node.js and MongoDB, along with a user-friendly mobile app developed in React Native, Jaiflix is well-equipped to meet user demands for content consumption. The integration with TMDB ensures that users have access to a vast library of media, while the admin panel allows for efficient content management and user administration.

5.1 Directions for Future Enhancements:-

For future enhancements to Jaiflix . These may include:

- **Multi-Language Support:** Implementing localization to cater to a diverse audience by offering multiple language options for the user interface and subtitles.
- **Advanced Recommendation System:** Utilizing machine learning algorithms to analyze user preferences and viewing habits to provide personalized content recommendations.
- **Social Features:** Adding social functionalities such as user reviews, ratings, and the ability to share content with friends could enhance user engagement and community building.
- **Enhanced Playback Features:** Incorporating features like picture-in-picture mode, enhanced audio settings, and additional subtitle options for a more customizable viewing experience.
- **Content Partnerships:** Exploring partnerships with content creators and distributors to expand the media library and provide exclusive content to users.

CHAPTER-6

6. References

6.1 BIBLIOGRAPHY

- **Official React Native Documentation:** <https://reactnative.dev/>
- **Node.js Documentation:** <https://nodejs.org/en>
- **All about npm:** <http://www.npmjs.com>
- **MongoDB Documentation:** <https://www.mongodb.com/>
- **TMDB API Documentation:** <https://www.themoviedb.org/>
- **Wikipedia for various diagrams & testing methods:** <http://www.wikipedia.org/>

6.2 Book References

1. "Learning React: A Hands-On Guide to Building Web Applications Using React and Redux", Author: Kirupa Chinnathambi, Publisher: Addison-Wesley Professional, Year: 2020, Edition: 2nd Edition
2. "Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node", Author: Vasam Subramanian, Publisher: Apress, Year: 2019, Edition: 2nd Edition
3. "Full-Stack React Projects: Learn MERN Stack Development by Building Modern Web Apps Using MongoDB, Express, React, and Node.js", Author: Shama Hoque. Publisher : Packt Publishing, Year: 2020, Edition: 2nd Edition
4. "The Road to React: Your Journey to Master React.js in JavaScript", Author: Robin Wieruch , Publisher: Independently Published, Year: 2020, Edition: 4th Edition
5. "MongoDB: The Definitive Guide: Powerful and Scalable Data Storage", Author: Shannon Bradshaw, Eoin Brazil, Kristina Chodorow, Publisher: O'Reilly Media, Year: 2019, Edition: 3rd Edition
6. These books provide in-depth knowledge on MERN stack development, from React front-end programming to server-side scripting with Node.js and managing databases using MongoDB

ANNEXURE

ANNEXURE –A-OUTPUT DESIGN

ADMIN PANEL:-

REGISTER

Admin Register

[Already an admin. Login!](#)

LOGIN

Admin Login

[Not an admin. Register as Admin.](#)

DASHBOARD

Dashboard

Add Movie

Add TV Shows

Logout

Update Movie

Update TV Shows

Delete Movie


Delete TV Shows

ADD MOVIE

List of Movies

Aranmanai


Fetch Details



Aranmanai 2

A ghost enters a palatial mansion and starts attacking the men in the family. What is its back story?


Release Date: 2016-01-29



Aranmanai 4


After his sister's suspicious death, a man decides to discover the hidden truth, setting off a chasm for chaos and terror.

Release Date: 2024-04-11



[Go to Dashboard](#)

Add Movie



Aranmanai 2

A ghost enters a palatial mansion and starts attacking the men in the family. What is its back story?

2016-01-29 • Action,Horror,Romance • 136mins

MovieID:
367842

Title:
Aranmanai 2

Original Title:
அரண்மனை 2

Overview:
A ghost enters a palatial mansion and starts attacking the men in the family. What is its back story?


Ratings:
4.417

Backdrop Path:
/o9USQWUWAotGSHIGeeyPnLqToa.jpg

UPDATE MOVIE

[Go to Dashboard](#)


Edit Movie List



Aranmanai 2

A ghost enters a palatial mansion and starts attacking the men in the family. What is its back story?


Release Date: Fri Jan 29 2016 05:30:00 GMT+0530 (India Standard Time)



Aranmanai 4

After his sister's suspicious death, a man decides to discover the hidden truth, setting off a chasm for chaos and terror.

Release Date: Thu Apr 11 2024 05:30:00 GMT+0530 (India Standard Time)




Aranmanai

Aranmanai is about a family that wants to sell their ancestral home. The supernatural elements they discover there in the aranmanai (palace) lead to some bone-chilling moments.

Release Date: Fri Sep 19 2014 05:30:00 GMT+0530 (India Standard Time)

[Go to Dashboard](#)

Update Movie Details



Aranmanai 2

A ghost enters a palatial mansion and starts attacking the men in the family. What is its back story?

Fri Jan 29 2016 05:30:00 GMT+0530 (India Standard Time) • Action,Horror,Romance • 136mins

ID:
66f377820d4824b24bf16eb1

MovieID:
367842

Title:
Aranmanai 2

Original Title:
அரண்மனை 2

Overview:
A ghost enters a palatial mansion and starts attacking the men in the family. What is its back story?


Ratings:
4.5

24

DELETE MOVIE

[Go to Dashboard](#)

Delete Movie




Aranmanai 2

A ghost enters a palatial mansion and starts attacking the men in the family. What is its back story?

Release Date: Fri Jan 29 2016 05:30:00 GMT+0530 (India Standard Time)

Delete




Aranmanai 4

After his sister's suspicious death, a man decides to discover the hidden truth, setting off a chasm for chaos and terror.

Release Date: Thu Apr 11 2024 05:30:00 GMT+0530 (India Standard Time)

Delete



Aranmanai

Aranmanai is about a family that wants to sell their ancestral home. The supernatural elements they discover there in the aranmanai (palace) lead to some bone-chilling moments.

Release Date: Fri Sep 19 2014 05:30:00 GMT+0530 (India Standard Time)


Delete

ADD SHOW

UPDATE SHOW

[Go to Dashboard](#)


Edit TV Shows List



Arcane

Amid the stark discord of twin cities Piltover and Zaun, two sisters fight on rival sides of a war between magic technologies and clashing convictions.


Release Date: Sat Nov 06 2021 05:30:00 GMT+0530 (India Standard Time)



Breaking Bad

Walter White, a New Mexico chemistry teacher, is diagnosed with Stage III cancer and given a prognosis of only two years left to live. He becomes filled with a sense of fearlessness and an unrelenting desire to secure his family's financial future at any cost as he enters the dangerous world of drugs and crime.

Release Date: Sun Jan 20 2008 05:30:00 GMT+0530 (India Standard Time)



Breaking Bad Fortune Teller

During the Ming Dynasty, the emperor sets out for Xinjiang to advocate peace. Shortly after his absence, mysterious occurrences plague the palace and Daoist Tian Mu Tong is called to cast out demons. Together with eunuch Xiao Mo Gu and Jin Yi Wei leader Yun Xiang Rong, Mu Tong uncovers deep dark secrets and conspiracies hidden within conspiracies.

Release Date: Mon Apr 11 2016 05:30:00 GMT+0530 (India Standard Time)

TV Show Details

[Go to Dashboard](#)



Arcane

Amid the stark discord of twin cities Piltover and Zaun, two sisters fight on rival sides of a war between magic technologies and clashing convictions.

Sat Nov 06 2021 05:30:00 GMT+0530 (India Standard Time) • Animation, Sci-Fi & Fantasy, Action & Adventure, Mystery • 8.8/10

Season 1



Episode 1: Welcome to the Playground (44 mins)

Orphaned sisters Vi and Powder bring trouble to Zaun's underground streets in the wake of a heist in posh Piltover.

Download Link: <http://commondatastorage.gc>



Episode 2: Some Mysteries Are Better Left Unsolved (41 mins)

Idealistic inventor Jayce attempts to harness magic through science—despite his mentor's warnings. Criminal kingpin Silco tests a powerful substance.

Download Link: <http://commondatastorage.gc>



Episode 3: The Base Violence Necessary for Change (45 mins)

An epic showdown between old rivals results in a fateful moment for Zaun. Jayce and Viktor risk it all for their research.

DELETE SHOW

[Go to Dashboard](#)

Delete Show



Arcane

Amid the stark discord of twin cities Piltover and Zaun, two sisters fight on rival sides of a war between magic technologies and clashing convictions.

Release Date: Sat Nov 06 2021 05:30:00 GMT+0530 (India Standard Time)

Delete



Breaking Bad

Walter White, a New Mexico chemistry teacher, is diagnosed with Stage III cancer and given a prognosis of only two years left to live. He becomes filled with a sense of fearlessness and an unrelenting desire to secure his family's financial future at any cost as he enters the dangerous world of drugs and crime.

Release Date: Sun Jan 20 2008 05:30:00 GMT+0530 (India Standard Time)

Delete



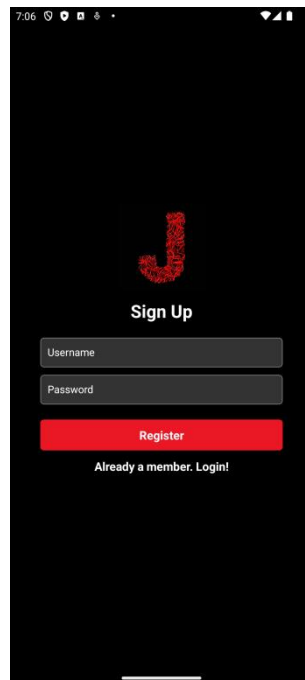
Breaking Bad Fortune Teller

During the Ming Dynasty, the emperor sets out for Xinjiang to advocate peace. Shortly after his absence, mysterious occurrences plague the palace and Daoist Tian Mu Tong is called to cast out demons. Together with eunuch Xiao Mo Gu and Jin Yi Wei leader Yun Xiang Rong, Mu Tong uncovers deep dark secrets and conspiracies hidden within conspiracies.


Release Date: Mon Apr 11 2016 05:30:00 GMT+0530 (India Standard Time)

USER - REACT NATIVE APP:-

REGISTER



7:06



Sign Up

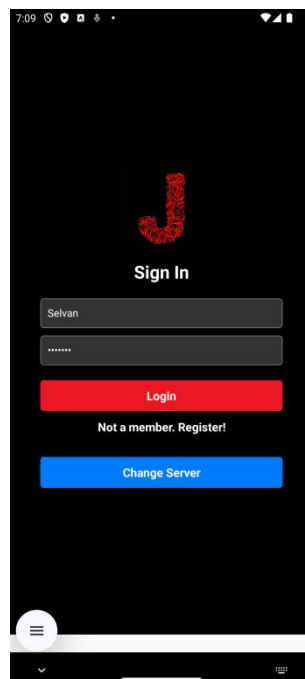
Username

Password


Register

Already a member. Login!

LOGIN



7:09




Sign In

Selvan

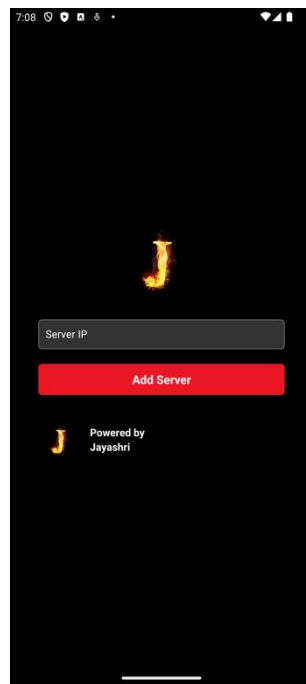
Login

Not a member. Register!

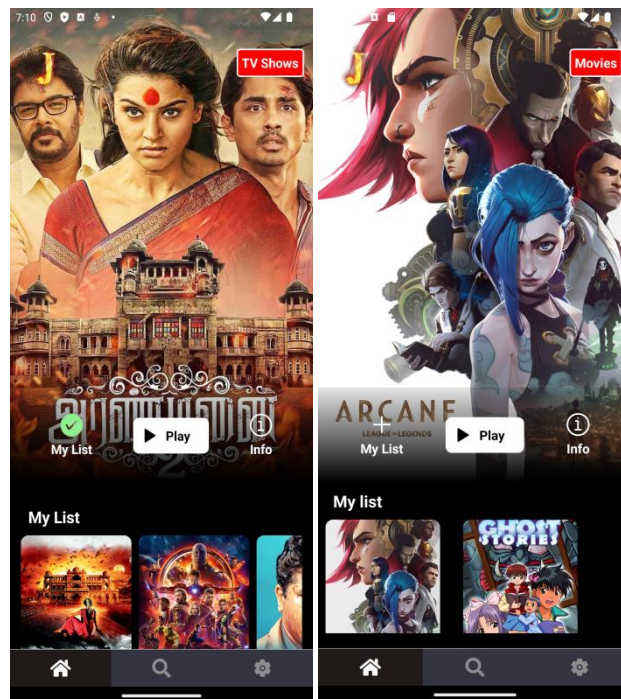
Change Server



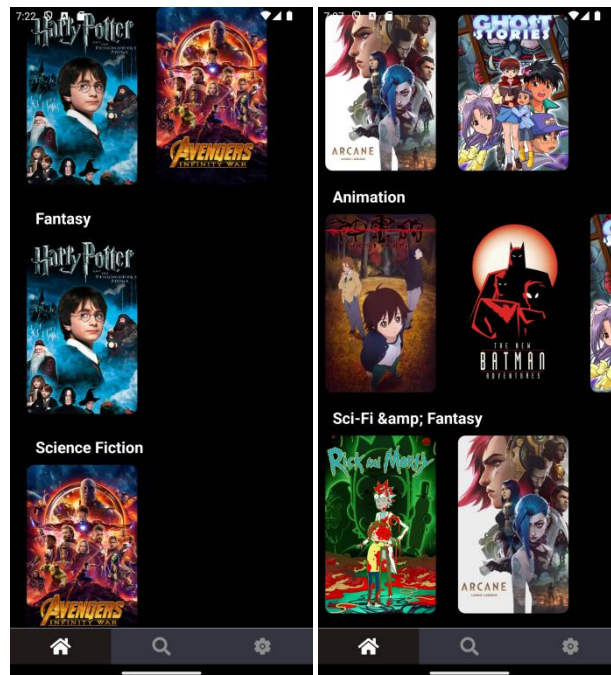
ADD SERVER



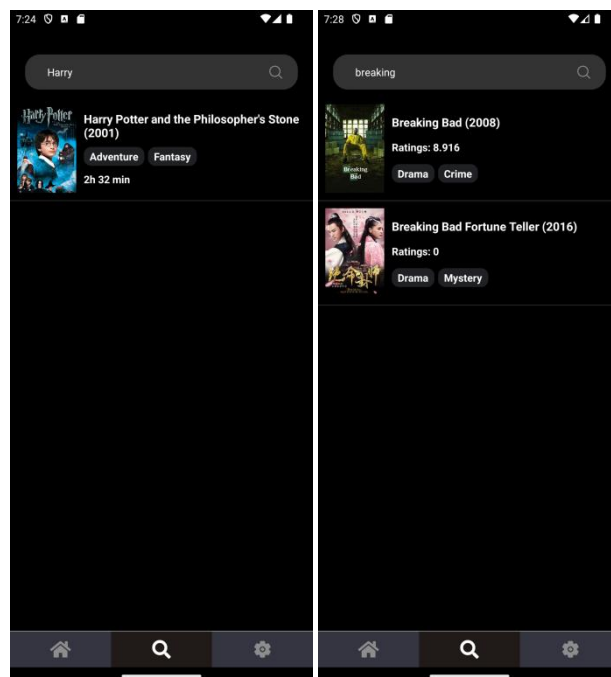
HOME_PAGE (MOVIE AND TV SHOW)



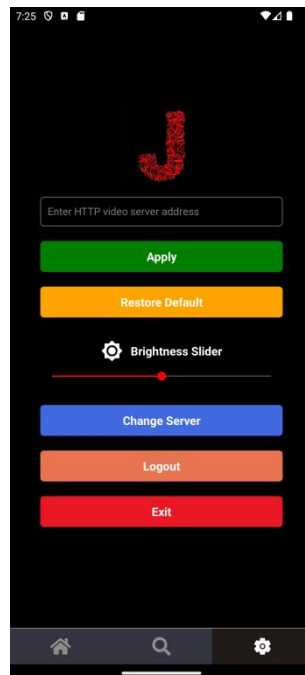
GENRES CLASSIFICATION (TV SHOW AND MOVIE)



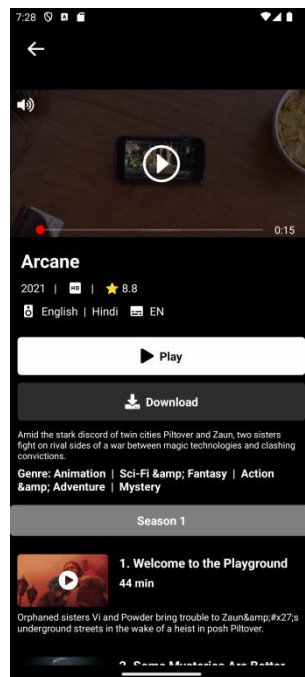
SEARCH BAR (MOVIE AND TV SHOW)



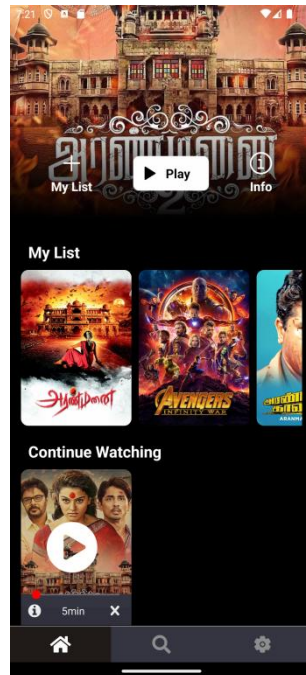
SETTINGS PAGE



MOVIES AND TV SHOW DETAILS DISPLAY



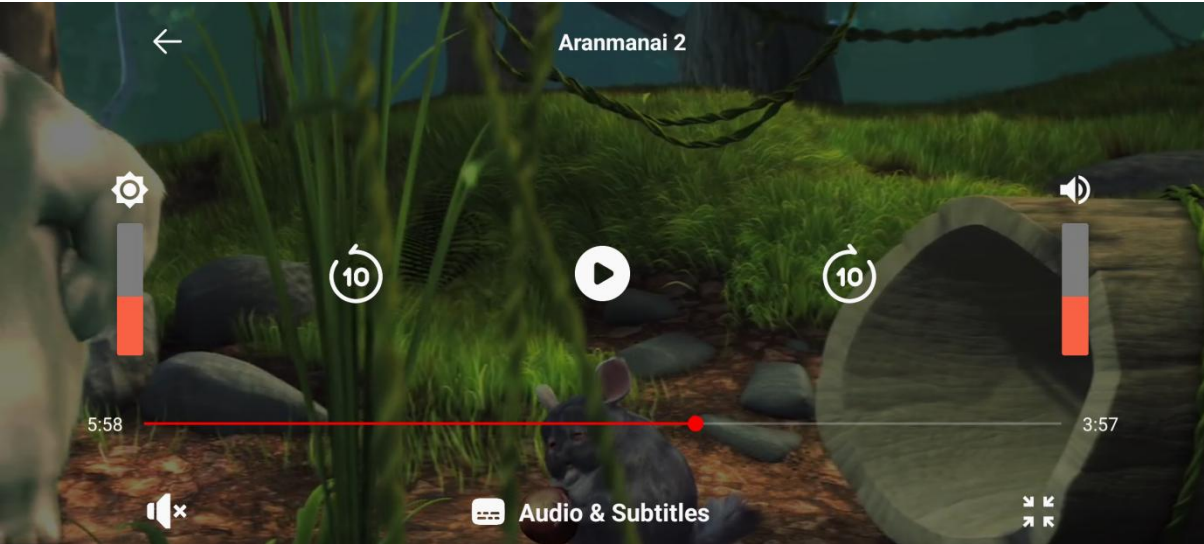
MY LIST AND LATEST WATCHED VIDEO



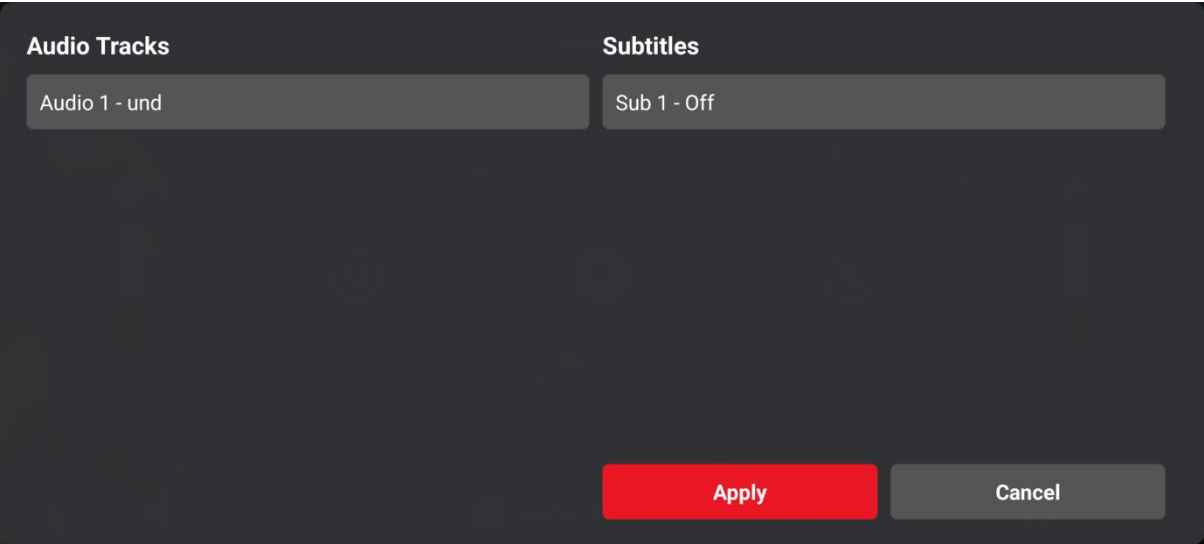
VIDEO STREAMING:-

LANDSCAPE





AUDIO SUBTITLE SETTINGS PAGE



ANNEXURE –B-SOURCE CODE:

REACT NATIVE APP – JAIFLIX

APP.JS

```
import { View, StyleSheet } from 'react-native'
import React from 'react'
import AppNavigation from './src/navigation/AppNavigation'
export default function App() {
  return (
    <View style={styles.container}>
      <AppNavigation/>
    </View>
  )
}
const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
});
```

LOGINSCREEN.JS

```
import React, { useEffect, useState } from 'react';
import { View, Text, TextInput, TouchableOpacity, StyleSheet, Alert, StatusBar, Image }
from 'react-native';
import { userloginAPI, checkAuthAPI } from '../api/userloginAPI';
import { useNavigation } from '@react-navigation/native';
import AsyncStorage from '@react-native-async-storage/async-storage';

const LoginScreen = () => {
  const navigation = useNavigation();
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  useEffect(()=>{
    const checkauthuser=async ()=>{
```

```

const response= await checkAuthAPI();
console.log("check auth response",response)
if(response.authenticated){
  navigation.navigate('BottomTabNavigator', {
    screen: 'HomeScreen',
    params: { mylist: response.user.mylist, watchedMovies: response.user.watchedMovies },
  });
}
}
checkauthuser()
},[])
const handleLogin = async () => {
  // console.log('Logging in with:', { username, password });
  const responseData = await userloginAPI(username, password);
  // console.log("Login response are",responseData)
  if (responseData.success === false) {
    // console.warn("Wrong username or password!")
    Alert.alert("Authentication failed", "Wrong username or password!")
  } else if (responseData.success === true) {
    navigation.navigate('BottomTabNavigator', {
      screen: 'HomeScreen',
      params: { mylist: responseData.user.mylist, watchedMovies:
responseData.user.watchedMovies },
    });
  }
};
const handleRegister = () => {
  navigation.navigate("RegisterScreen")
}
const changeServer = async () => {
  await AsyncStorage.removeItem('serverIp');
  navigation.navigate("AddServerScreen")
}
return (
<View style={styles.container}>

```

```

<StatusBar translucent backgroundColor="transparent" />
<Image source={require('../assets/logo.jpg')} style={styles.logo} />
<Text style={styles.title}>Sign In</Text>
<TextInput
  style={styles.input}
  placeholder="Username"
  placeholderTextColor={"white"}
  onChangeText={(text) => setUsername(text)}
/>
<TextInput
  style={styles.input}
  placeholder="Password"
  placeholderTextColor={"white"}
  secureTextEntry
  onChangeText={(text) => setPassword(text)}
/>
<TouchableOpacity style={styles.loginButton} onPress={handleLogin}>
  <Text style={styles.buttonText}>Login</Text>
</TouchableOpacity>
<TouchableOpacity onPress={handleRegister}>
  <Text style={styles.buttonText}>Not a member. Register!</Text>
</TouchableOpacity>
<TouchableOpacity style={styles.changeServer} onPress={changeServer}>
  <Text style={styles.buttonText}>Change Server</Text>
</TouchableOpacity>
</View>
);
};
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#000000', // Dark background color

```

```

},
logo: {
width: 120,
height: 120,
marginBottom: 10,
},
title: {
fontSize: 24,
color: 'white',
marginBottom: 20,
fontWeight:'bold'
},
input: {
height: 40,
width: '80%',
borderColor: 'gray',
borderWidth: 1,
backgroundColor: '#333333',
marginBottom: 10,
paddingHorizontal: 10,
borderRadius: 5,
color: 'white',
},
loginButton: {
backgroundColor: '#EB1825',
paddingVertical: 10,
paddingHorizontal: 20,
borderRadius: 5,
marginVertical:10,
width:'80%'
},
buttonText: {
color: 'white',
fontSize: 16,

```

```

fontWeight:'bold',
textAlign:'center'
},
changeServer: {
top:20,
width:'80%',
backgroundColor: '#007BFF',
paddingVertical: 10,
paddingHorizontal: 20,
borderRadius: 5,
marginVertical: 10
}
});
export default LoginScreen;

```

REGISTERSCREEN.JS

```

import React, { useState } from 'react';
import { View, Text, TextInput, TouchableOpacity, StyleSheet, StatusBar, Image } from
'react-native';
import { useNavigation } from '@react-navigation/native';
import { userRegisterAPI } from '../api/userRegisterAPI';
const RegisterScreen = () => {
const navigation = useNavigation();
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");
const handleLogin = async () => {
navigation.navigate('LoginScreen');
};
const handleRegister = async () => {
const responseData = await userRegisterAPI(username, password);
if (responseData.success === false) {
console.warn("Username and password already exists!")
} else if (responseData.success === true) {

```

```

navigation.navigate('LoginScreen');
}}
return (
<View style={styles.container}>
<StatusBar translucent backgroundColor="transparent" />
<Image source={require('../assets/logo.jpg')} style={styles.logo} />
<Text style={styles.title}>Sign Up</Text>
<TextInput
style={styles.input}
placeholder="Username"
placeholderTextColor={"white"}
onChangeText={(text) => setUsername(text)}
/>
<TextInput
style={styles.input}
placeholder="Password"
placeholderTextColor={"white"}
secureTextEntry
onChangeText={(text) => setPassword(text)}
/>
<TouchableOpacity style={styles.registerButton} onPress={handleRegister}>
<Text style={styles.buttonText}>Register</Text>
</TouchableOpacity>
<TouchableOpacity onPress={handleLogin}>
<Text style={styles.buttonText}>Already a member. Login!</Text>
</TouchableOpacity>
</View>
);};
const styles = StyleSheet.create({
container: {
flex: 1,
justifyContent: 'center',
alignItems: 'center',
backgroundColor: '#000000', // Dark background color

```

```

},
logo: {
width: 120,
height: 120,
marginBottom: 10,
},
title: {
fontSize: 24,
color: 'white',
marginBottom: 20,
fontWeight:'bold'
},
input: {
height: 40,
width: '80%',
borderColor: 'gray',
borderWidth: 1,
backgroundColor: '#333333',
marginBottom: 10,
paddingHorizontal: 10,
borderRadius: 5,
color: 'white',
},
registerButton: {
backgroundColor: '#EB1825',
paddingVertical: 10,
paddingHorizontal: 20,
borderRadius: 5,
marginVertical:10,
width:'80%'
},
buttonText: {
color: 'white',
fontSize: 16,

```

```
fontWeight:'bold',
textAlign:'center'
},
});
export default RegisterScreen;
```

ADMIN PANEL SOURCE CODE:-

APP.JS

```
require('dotenv').config()
const express = require('express')
const app = express()
const cors = require('cors');
const bodyParser = require('body-parser');
app.set('view engine', 'hbs');
const port = process.env.PORT
const mongoose = require('mongoose');
mongoose.connect(process.env.MONGO_DB_URL, { useNewUrlParser: true,
useUnifiedTopology: true });
const db = mongoose.connection;
const session = require('express-session');
const User = require('./models/User')
const MongoStore = require('connect-mongo')
app.use(express.json({ limit: '50mb' }));
app.use(express.urlencoded({ limit: '50mb', extended: true }));
app.use((err, req, res, next) => {
console.error(err.stack);
res.status(500).send('Something went wrong!');
});
app.use(session({
secret: 'abcd1234',
resave: false,
saveUninitialized: false,
store: new MongoStore({ mongoUrl: process.env.MONGO_DB_URL}),
```



```

cookie: {
  maxAge: 1000 * 60 * 60 * 24 * 7, // 1 week (adjust as needed)
},
});

const passport = require('passport'); // Import Passport.js library for authentication
const LocalStrategy = require('passport-local').Strategy; // Import Passport Local Strategy for
username/password authentication
app.use(passport.initialize()); // Initialize Passport middleware
app.use(passport.session()); // Use Passport middleware for session management
passport.use(new LocalStrategy(User.authenticate())); // Use local strategy for authentication
passport.serializeUser(User.serializeUser()); // Serialize user data for storage in session
passport.deserializeUser(User.deserializeUser()); // Deserialize user data from session
const path = require('path'); // Import path module
app.use(express.static(path.join(__dirname, 'public')));

app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
const Movie = require('./models/movie')
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});
const dashboard = require('./routes/dashboard')
const addMovie = require('./routes/addMovie')
const updateMovieRoute = require('./routes/updateMovie')
const deleteMovie = require('./routes/deleteMovie')
const getMovies = require('./routes/getMovies')
const authRoutes = require('./routes/authRoutes')
const myList = require('./routes/mylist')
const watcheMovie = require('./routes/watchedMovie')
const scanAllMovies = require('./routes/scanAllMovies')
const addShows = require('./routes/addShows')
const updateShows = require('./routes/updateShows')

```

```

const deleteShow = require('./routes/deleteShow')
const scanAllShows = require('./routes/scanAllShows')
const getShows = require('./routes/getShows')
const watchedShows = require('./routes/watchedShows')
const showsMylist = require('./routes/showsMylist')
const managePosters = require('./routes/managePosters')
const checkCon = require('./routes/checkcon')
app.use('/', checkCon)
app.use('/', dashboard)
app.use('/', addMovie)
app.use('/', updateMovieRoute)
app.use('/', deleteMovie)
app.use('/', getMovies)
app.use('/', authRoutes)
app.use('/', myList)
app.use('/', watcheMovie)
app.use('/', scanAllMovies)
app.use('/', addShows)
app.use('/', updateShows)
app.use('/', deleteShow)
app.use('/', scanAllShows)
app.use('/', getShows)
app.use('/', watchedShows)
app.use('/', showsMylist)
app.use('/', managePosters)
app.listen(port, () => {
  console.log(`API is running on port ${port}`)
})

```

DASHBOARD.JS

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Admin Dashboard</title>
</head>
<body>
<div class="dashboard-box">
<h1>Dashboard</h1>
<div class="dashboard-buttons">
<div class="row">
<a class="add-button" href="/addMovieRoute">Add Movie</a>
<a class="update-button" href="/updateMovieRoute">Update Movie</a>
<a class="delete-button" href="/deleteMovieRoute">Delete Movie</a>
{{!-- <a class="scanAll-button" href="/scan-all-movie-page">Scan All Movies</a> --}}
</div>
<div class="row">
<a class="add-button" href="/addShowsRoute">Add TV Shows</a>
<a class="update-button" href="/updateShowsRoute">Update TV Shows</a>
<a class="delete-button" href="/deleteShowsRoute">Delete TV Shows</a>
{{!-- <a class="scanAll-button" href="/scan-all-shows-page">Scan All TV Shows</a> --}}
</div>
<div class="row">
<a class="logout-button" href="/admin/logout">Logout</a>
</div>
</div>
</div>
</body>
<style>
body {
font-family: 'Arial', sans-serif;
background-color: #f4f4f4;
margin: 0;
padding: 0;
display: flex;
flex-direction: column;
align-items: center;

```

```
justify-content: center;
height: 100vh;
}
.dashboard-box {
background-color: #fff;
border-radius: 10px;
padding: 20px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
display: flex;
flex-direction: column;
align-items: center;
}
h1 {
color: #333;
}
.dashboard-buttons {
display: flex;
flex-wrap: wrap;
justify-content: space-around;
width: 100%;
margin-top: 20px;
}
.row {
display: flex;
justify-content: space-around;
width: 100%;
margin-bottom: 10px;
}
.add-button {
flex: 1;
margin: 0 10px;
padding: 15px;
background-color: #28a745;
color: #fff;
```

```

text-decoration: none;
text-align: center;
font-weight: bold;
border-radius: 5px;
transition: background-color 0.3s ease;
}
.update-button {
flex: 1;
margin: 0 10px;
padding: 15px;
background-color: #007bff;
color: #fff;
text-decoration: none;
text-align: center;
font-weight: bold;
border-radius: 5px;
transition: background-color 0.3s ease;
}
.delete-button {
flex: 1;
margin: 0 10px;
padding: 15px;
background-color: #dc3545;
color: #fff;
text-decoration: none;
text-align: center;
font-weight: bold;
border-radius: 5px;
transition: background-color 0.3s ease;
}
.scanAll-button {
flex: 1;
margin: 0 10px;
padding: 15px;

```

```

background-color: #f87744;
color: #fff;
text-decoration: none;
text-align: center;
font-weight: bold;
border-radius: 5px;
transition: background-color 0.3s ease;
}

.logout-button {
flex: 1;
margin: 0 10px;
padding: 15px;
background-color: #6c757d;
color: #fff;
text-decoration: none;
text-align: center;
font-weight: bold;
border-radius: 5px;
transition: background-color 0.3s ease;
}

.add-movie-button:hover {
background-color: #276837;
}

.update-movie-button:hover {
background-color: #2866a8;
}

.delete-movie-button:hover {
background-color: #921e2a;
}

.logout-button:hover {
background-color: #313437;
}
</style>
</html>

```