STRANGE-SHOWS

PROJECT REPORT

Submitted to the Department of Computer Applications, Bharathiar University Post Graduate Extension and Research Center in partial fulfillment of requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

Submitted

by

DHINESHKUMAR MM

(23CSEA81)

Under the guidance of

Dr. K. K. SAVITHA, Ph.D.,

Assistant Professor and Head (i/c)

Department of Computer Applications

Bharathiar University Post Graduate Extension and Research Center, Erode

BHARATHIAR UNIVERSITY POST GRADUATE EXTENSION AND RESEARCH CENTER

ERODE

APRIL 2025

DECLARATION

I hereby declare that this project work titled, "**STRANGE-SHOWS"** submitted to Department of Computer Applications, Bharathiar University Post Graduate Extension and Research Center, is a record work done by **DHINESHKUMAR MM (23CSEA81)**, under the supervision and guidance of **Dr. K.K. SAVITHA**, Ph.D., Assistant Professor and Head (i/c), Department of Computer Applications, Bharathiar University Post Graduate Extension and Research Centre, Erode.

Place: ERODE **Signature of the Candidate**

Date: **(DHINESHKUMAR MM)**

Countersigned by

Dr. K. K. SAVITHA, Ph.D.,

Assistant Professor and Head (i/c)

Department of Computer Applications

Bharathiar University Post Graduate Extension and Research Centre

Erode

CERTIFICATE

This is to certify that, this project work entitled, **" STRANGE-SHOWS"** submitted to Department of Computer Applications, Bharathiar University Post Graduate Extension and Research Centre in partial fulfilment of the requirement for the award of the degree of Master of Computer Applications, is a record of original work done by **DHINESHKUMAR MM (23CSEA81)**, during his period of study in the Department of Computer Applications, Bharathiar University Post Graduate Extension and Research Centre, Erode, under my supervision and guidance and that this project work has not formed the basis for the award of any Degree /Diploma /Associateship /Fellowship or similar title to any candidate of any University.

Place:

Date:

Project Guide **Head of the Department**

Submitted for the Project VIVA-VOCE Examination held on_____.

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

I sincerely salute and thank the almighty for the shower of blessing, which has enabled me to attain success in the endeavour.

I would like to express my sincere gratitude to **Dr. N. DHARMARAJ, Ph.D., Director,** BHARATHIAR UNIVERSITY POST GRADUATE EXTENSION AND RESEARCH CENTRE, ERODE for permitting me to do this project

I wish to express my sincere thanks to my internal guide **Dr. K. K. SAVITHA, Ph.D.**, Assistant Professor and Head (i/c), Department of Computer Applications, Bharathiar University Post Graduate Extension and Research Centre, for her motivation, valuable guidance and encouragement provided throughout the project work.

I own special thanks to all staff members of the Department of Computer Applications for their valuable support rendered to me during project work.

I thank everybody whose names are mentioned and not mentioned here, who directly or indirectly contribution in bringing out this project successfully.

I will not forget my deep sense of gratitude to my family for their moral support and encouragement. I think all my friends who helped me completing my project. Finally, I feel great honor to thank each and every one who helped me in one way or the other to complete the project.

DHINESHKUMAR MM

TABLE OF CONTENTS

ABSTRACT

Strange-Shows is an advanced, all-in-one movie ticket-booking software designed to offer users a seamless and interactive cinema experience. The platform allows customers to easily **book movie tickets**, **order food and beverages**, and **share reviews**, all through a unified interface. In addition to these core features, Strange-Shows introduces exclusive sections such as the **Terrace-View**, an OTT-like streaming service, and the **Secret-House**, a Gallery of Honor dedicated to celebrating legendary filmmakers and cinematic history.

The application is developed with a modern technology stack: **React.js** is used for building a fast, responsive, and intuitive frontend interface, while **Express.js** serves as the backend framework to manage API endpoints and server-side operations. **MongoDB** acts as the database, efficiently storing user data, bookings, food orders, reviews, and more.

For deployment and automation, **Microsoft Azure** plays a critical role. Both the frontend and backend are deployed as **Azure Static Web Apps**, ensuring global scalability, high availability, and optimized performance. **Azure DevOps** has been used **YMAL** to utilized and establish CI/CD pipelines and release management, streamlining the development workflow and enabling automated deployments. Pipelines automate code builds and tests, while releases manage staged rollouts and production updates.

Strange-Shows delivers an enhanced movie booking and entertainment platform that not only simplifies the user experience but also introduces new ways to engage with cinema beyond the theatre.

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

The rapid growth of digital platforms has transformed how people experience entertainment, particularly in the cinema and streaming industries. Traditional movie ticket booking systems are evolving to meet modern user demands for convenience, speed, and additional services. In this context, **Strange-Shows** emerges as a comprehensive solution that combines **movie ticket booking**, **food ordering**, **customer reviews**, and **exclusive entertainment sections** into a single, user-friendly platform.

Strange-Shows is designed to offer a seamless experience for cinema enthusiasts by integrating various functionalities into one application. The platform enables users to **book movie tickets**, **order food and beverages** for in-theater delivery, and **share reviews** about their movie and food experiences. Beyond the standard features, Strange-Shows introduces innovative pages like the **Terrace-View**, an OTT-like streaming service providing on-demand content, and the **Secret-House**, a Gallery of Honor showcasing legendary filmmakers, movies, and cinematic trivia.

The project leverages a modern technology stack. The **frontend** is developed using **React.js**, delivering a dynamic, responsive, and intuitive user interface. The **backend** is built on **Express.js**, handling robust API services and server-side logic. **MongoDB** serves as the database solution, offering a flexible and scalable approach to data management.

For deployment and continuous integration/continuous delivery (CI/CD), the platform utilizes **Microsoft Azure**. Both the frontend and backend are hosted as **Azure Static Web Apps**, providing global availability and high performance. **Azure DevOps** manages the CI/CD pipelines and release workflows, automating the build, test, and deployment processes to ensure reliability and efficiency in development and production environments.

1.2 OVERVIEW OF SYSTEM

Strange-Shows is a comprehensive movie ticket-booking software system that integrates various services into a unified digital platform. It is designed to enhance the movie-going experience by offering a seamless interface where users can **book movie tickets**, **order food and beverages**, **post reviews**, and access **exclusive entertainment content**. The system focuses on providing convenience, speed, and user engagement by combining cinema services with modern web technologies.

Key Features

- **Online Movie Ticket Booking**: Users can browse current and upcoming movies.
- **Food & Beverage Ordering:** Food orders can be delivered to the seat during the movie or picked up.
- **Customer Reviews & Ratings**: Users can post reviews and rate movies
- **Terrace-View (OTT Streaming Integration)**: An exclusive OTT-style section where users can watch selected content online.
- **Secret-House (Gallery of Honor)**: Dedicated section showcasing legendary filmmakers, actors, and cinema history.
- **User Authentication & Role-Based Access**: Secure user registration and login using JWT authentication. Separate roles for admin and users.
- **Admin Panel**: Complete control over movie listings, showtimes, and ticket prices. Manage food menu and orders.
- **Responsive UI/UX**: Frontend built with **React.js** for a dynamic and responsive user interface.
- **CI/CD Integration with Azure DevOps**: Automated build, test, and deployment pipelines for continuous delivery. Version control and release management using Azure DevOps Pipelines and Releases.
- **Scalable Deployment on Microsoft Azure:** Frontend and backend deployed on Azure Static Web Apps.
- **Search and Filter Functionality:** Filter showtimes by location, date, and screen type.
- **Multi-Location Theatre Support:** Users can switch between different Strange-Shows theatre locations. Location-based filtering for movies and food menus.

These features create a fully functional and interactive platform, offering an Immense experience

1.3 OBJECTIVES

The primary objective of the **Strange-Shows** Movie Ticket Booking Software is to create a comprehensive and user-friendly platform that streamlines the movie-going experience while offering additional features to enhance user engagement. The system is designed to simplify the process of booking movie tickets, ordering food, and accessing exclusive content, all in one integrated platform. Below are the specific objectives of the system:

Primary Objectives:

- To develop an all-in-one digital platform for movie ticket booking and food ordering.
- To provide an interactive and intuitive user interface for end-users using React.js.
- To offer a seamless and secure user authentication system with role-based access control (User & Admin).
- To integrate exclusive content areas like Terrace-View (OTT content) and Secret-House (Gallery of Honor) to provide additional user engagement.

Technical Objectives:

- Using SCRUM Methodology
- To implement Continuous Integration and Continuous Deployment (CI/CD) pipelines using Azure DevOps(SCRUM) for faster and reliable delivery of features.
- To deploy both frontend and backend applications as Azure Static Web Apps, ensuring global availability and scalability.

Business Objectives:

- To improve the overall cinema experience for users by offering convenience, efficiency, and additional services.
- To increase customer engagement and retention through exclusive features and user-centric design.

By achieving these objectives, the project aims to create a high-performance, secure, and user-friendly movie ticket booking platform that demonstrates the capabilities of the React.js, Express.js, MongoDB, and Microsoft Azure ecosystem in delivering a complete, full-stack web application with CI/CD integration.

CHAPTER-2

SYSTEM STUDY AND ANALYSIS

2.1 INTRODUCTION TO SYSTEM TOOLS

In the development of the **Strange-Shows** Movie Ticket Booking platform, a variety of modern tools and technologies have been utilized to ensure an efficient, scalable, and responsive application. These tools cover both frontend and backend development, database management, deployment, and CI/CD processes. This section provides an overview of the key system tools and technologies used in building the project.

2.2 EXISTING SYSTEM ANALYSIS

- **React.js** – To build a dynamic and responsive frontend.
- **Node.js & Express.js** – To handle server-side logic and API requests efficiently.
- **MongoDB Atlas** – To store user data, posts, likes, comments, and followers.
- **Azure DevOps (CI/CD Pipeline & Project Management)** – Azure DevOps provides a set of cloud-based tools for planning, developing, delivering, and monitoring applications.
- **SCRUM** – Methodology
- **Microsoft Azure Portal (Hosting and Deployment)** – To deploying and hosting the Strange-Shows application.
- **Git & GitHub (Version Control)** – To control and hosts the project repository
- **Visual Studio & VS Code (Code Editor)** – To use for writing and debugging code.

2.3 PROPOSED SYSTEM ANALYSIS

The proposed **Strange-Shows** system offers a complete movie experience, both online and in-theatre. It provides:

- **Ticket Booking:** Users can browse available movies, select showtimes, book seats, and pay online.

- **Food and Beverage Ordering:** Users can order food and beverages along with their tickets or separately from the comfort of their seats.
- **Customer Reviews and Ratings:** Users can share their experiences and rate movies, food, and the overall service.
- **Exclusive Content Pages**:
    - **Terrace-View**: A platform similar to an OTT service, offering exclusive movie content.
    - **Strange-House**: A gallery honouring cinema legends, including biographies, achievements, and contributions.
- **Admin Panel:** A powerful admin interface to manage movie listings, showtimes, food menus, reviews, user activities, and multiple theatre locations.
- **CI/CD Pipelines and Azure Hosting:** Automated build and release pipelines via Azure DevOps for continuous integration and deployment, with scalable hosting on Microsoft Azure.
- **Optimized Performance** – The system is built with **MongoDB Atlas for database scalability**, ensuring high availability.

2.4 SYSTEM WORKFLOW

User Workflow :

- **User Registration & Login** – Users sign up or log in using JWT-based authentication.
- **Dashboard Access**– Once logged in, users are redirected to a personalized dashboard.
- **Movie Ticket Booking**– Users browse movies categorized by location and availability.
- **Food and Beverage Ordering**– Users can place food orders while booking tickets or later through the dashboard.
- **Access Exclusive Content**– Users can explore **Terrace-View**, an OTT-like section for exclusive movie streams.
- **Secret House**– Offers a Gallery of Honor featuring cinema legends, trivia, and multimedia content.
- **Review and Rating Submission**– After watching a movie, users can post reviews and ratings.

Admin Workflow :

- **Admin Login**– Admins log in securely with privileged credentials.
- **Manage Movies and Showtimes**– Admins can add, edit, or remove movie listings.
- **Manage Food Menus**– Admins upload or update food and beverage offerings per theatre location.
- **Multi-Location Theater Management**– Admins have a centralized system to manage multiple Strange-Shows theaters in different locations.
- **CI/CD Deployment via Azure DevOps** – Admins (or DevOps team) deploy system updates through automated pipelines in Azure DevOps.
- **CI/CD**– ensures continuous integration, testing, and deployment of new features and fixes.
- **Azure Static Web Apps**– host the updated frontend and backend securely.

2.5 VERSION CONTROL SYSTEM

2.5.1 GIT & GITHUB

GitHub is used for **version control and collaboration**, allowing developers to track changes and deploy the Project efficiently.

✓ **Why Git & GitHub?**

- ◇ Helps in managing different versions of the project
- ◇ Enables collaborative development with teams
- ◇ Supports automatic deployment

2.6 HARDWARE REQUIREMENTS

The Strange-Show does not require high-end hardware. However, the following are essential:

- **A computer with an internet connection** – To develop, test, and deploy the X cloning
- **Smartphone with any browser installed** – For real-world testing of the website

CHAPTER-3

TECHNOLOGIES USED

3.1 FRONTEND – REACT.JS

React.js is a **JavaScript library** used for building dynamic and interactive user interfaces. It plays a key role in ensuring a seamless user experience in the X Clone project.

Key Features of React.js in This Project:

- **Component-Based Architecture** – The UI is divided into reusable components, making the development process efficient and scalable.
- **Virtual DOM** – Improves performance by minimizing direct updates to the real DOM.
- **React Router** – Enables seamless navigation between pages like Home, Profile, and Notifications.
- **State Management (Redux Toolkit)** – Manages global state efficiently, ensuring smooth interactions and data flow.
- **Responsive UI** – Ensures the platform adapts well to different screen sizes and devices using Tailwind CSS.

3.2 BACKEND – NODE.JS AND EXPRESS.JS

The backend of the X Clone project is developed using **Node.js and Express.js**, providing a robust and scalable API to handle requests from the frontend.

Key Features of Node.js and Express.js in This Project:

- **Event-Driven and Non-Blocking Architecture** – Handles multiple requests efficiently without slowing down.
- **Express.js Framework** – Simplifies API routing, middleware integration, and server-side logic.
- **Middleware Integration** – Implements security features like CORS (Cross-Origin Resource Sharing) and Helmet for protection against cyber threats.
- **RESTful API Design** – Ensures structured communication between the frontend and backend.
- **Error Handling** – Provides proper error responses for failed API calls to enhance reliability.

3.3 DATABASE – MONGODB ATLAS

MongoDB Atlas is a **cloud-based NoSQL database** used for storing and managing data efficiently in this project.

Key Features of MongoDB Atlas in This Project:

- Scalability – Supports large datasets and high-speed queries.
- Mongoose ODM (Object Data Modeling) – Simplifies database interactions and schema definitions.
- Flexible Data Structure – Easily stores users, posts, comments, likes, and notifications in a structured way.
- Automatic Backups – Ensures data safety and prevents data loss.
- Indexing and Query Optimization – Improves performance by allowing faster data retrieval.

3.4 AUTHENTICATION – JSON WEB TOKEN (JWT)

Authentication is a critical component of the X Clone project, ensuring only authorized users can access the system. JWT (JSON Web Token) is used for secure user authentication.

Key Features of JWT in This Project:

- **Token-Based Authentication** – Stores user sessions securely in HTTP-only cookies.
- **Password Encryption** – Uses bcrypt to hash passwords before storing them in the database.
- **Role-Based Access Control** – Ensures only authorized users can perform certain actions.
- **Secure API Requests** – Prevents unauthorized access by requiring a valid token for protected routes.

3.5 CLOUD PLATFORM– MICROSOFT AZURE

It provides a wide range of cloud services, including those for computing, analytics, storage, and networking. Azure enables developers to build, deploy, and manage applications through a global network of Microsoft-managed data centre's.

Key Features of Azure in This Project:

- **Azure Resource Management (ARM)-** provides a unified way to deploy and manage Azure resources for the Strange-Shows project.
- **Azure IAM-** helps control who can access specific resources and what actions they can perform. In the Strange-Shows project, IAM roles ensure secure and role-based access control across Azure services.
- **Azure Static Web Apps**– Hosts the frontend React.js application. Globally distributed static content for fast delivery.
- **Azure App Service** – Hosts the backend Express.js APIs. Supports multiple programming languages, including Node.js (used for Express backend).

3.6 MEDIA STORAGE – AZURE BLOB STORAGE

Azure Blob Storage is a highly scalable, durable, and secure object storage service. It's perfect for storing large amounts of unstructured datas.

Key Features of Cloudinary in This Project:

- **Massive Scalability** – Store petabytes of data. It automatically scales as data grows.
- **High Availability** – Redundancy options like LRS, GRS for durability and disaster recovery.
- **Secure Media Access** – Shared Access Signatures (SAS) and Azure Active Directory (AAD) based access control.
- **Efficient Upload** – Supports multi-part uploads and block blobs for uploading very large files efficiently.

3.7 DEPLOYMENT AND HOSTING

Strange-Shows Movie Ticket-Booking Software, covering React, Express, MongoDB, Azure DevOps Pipelines, and Microsoft Azure Static Web Apps.

Key Deployment Tools in This Project:

- **Microsoft Azure**– Hosting and managing both the frontend and backend applications.
- **Azure DevOps(SCRUM)**– Automating the build, testing, and deployment workflows through CI/CD pipelines.
- **MongoDB Atlas**– Cloud-hosted **NoSQL database service** used for storing all application data, including movie details, bookings, food orders, customer reviews, and user accounts.
- **GitHub (or Azure Repos)** – Version control system to manage the source code of the project.
- **Azure Resource Management**– Provisioning, managing, and monitoring all Azure resources required by the application.

3.8 OTHERS TECHNOLOGIES USED

3.8.1 YAML (YAML Ain't Markup Language):

YAML is a human-readable data serialization standard commonly used for configuration files and data exchange between languages with different data structures. In this project, YAML plays a crucial role in defining the workflows and configuration files, especially in **Azure DevOps CI/CD pipelines**.

3.8.2 Key Features of YAML:

- **Human-readable syntax**: Easy to read and write, making it accessible even for non-developers to understand the pipeline processes.
- **Hierarchy with indentation**: YAML uses indentation to show structure, avoiding the need for complex tags or braces.
- **Widely supported**: YAML is supported by many CI/CD tools, including Azure DevOps, GitHub Actions, and others.

3.8.3 YAML Usage in Azure DevOps Pipelines:

- **Pipeline Definition**: YAML files define the stages, jobs, and tasks of the CI/CD process.
- **Automation**: YAML scripts automate builds, run tests, and deploy the **React.js frontend** and **Express.js backend** to Microsoft Azure services.

3.8.4 Azure DevOps Boards:

- It provides a rich set of capabilities for **planning**, **tracking**, and **discussing work** across development teams.
- In the **Strange-Shows** project, Azure Boards played a crucial role in **managing tasks**, **tracking progress**, and **maintaining transparency** throughout the software development lifecycle.
- **DevOps Board Tools:** boards, backlogs, sprints, and dashboards, enabling teams to visualize work, track deliverables, and collaborate effectively.

3.8.4.1 Workflow in Azure Boards:

- **Backlog Creation -** Broke down Features into User Stories and Tasks.
- **Sprint Planning -** Assigned tasks to team members based on sprint capacity.
- **Monitoring** - Used **Queries** and **Charts** to monitor task completion and identify bottlenecks.

3.8.5 Context API:

React provides a Context API for managing global state in a Website on text allows data to be passed through the component tree without having to explicitly pass props down manually at every level, making it easier to share state between deeply nested components.

CHAPTER-4

SYSTEM DESIGN

The system design of the **Strange-Shows Movie Ticket Booking Software** incorporates DevOps practices to ensure smooth, automated deployments and continuous integration. It uses Microsoft Azure, Azure DevOps Pipelines, and modern technologies like React, Express, and MongoDB to create a robust, scalable, and efficient system.

4.1 HIGH-LEVEL ARCHITECTURE DIAGRAM

Show key components: **Frontend**, **Backend**, **Database**, **CI/CD Pipelines**, **Azure Resources**, **Monitoring Tools**

Frontend (React.js)

- Provides a dynamic and interactive **user interface**.
- Uses **React Router** for seamless navigation between different sections.
- Manages application state using **Redux Toolkit**.
- Ensures **responsive design** using **Tailwind CSS**.

Backend (Node.js & Express.js)

- Handles **API requests, authentication, and business logic**.
- Implements **secure authentication** using **JWT and bcrypt**.
- Uses **Express.js** for efficient routing and request handling.

Backend code is tested and deployed automatically using Azure DevOps pipelines.

Database (MongoDB Atlas)

- Cloud-hosted NoSQL database for handling all structured and unstructured data.
- Stores user profiles, booking history, reviews, menu items, and special content metadata.
- Connected securely to backend APIs.

CI/CD Pipeline Design (Azure DevOps)

- **Trigger**: Code push to GitHub or Azure Repos.
- **CI Steps**: Install dependencies, run tests, build artifacts.
- **CD Steps**: Deploy frontend to Azure Static Web Apps, backend to Azure App Services, configure environment variables, etc.

Azure Blob Storage

- **Massive Scalability** is used for Store petabytes of data and scale on demand without performance degradation.
- **Durability and High Availability** stores multiple copies of your data to ensure durability (with redundancy options like LRS, ZRS, GRS).
- **Global Accessibility** is accessible through HTTP/HTTPS from anywhere in the world, which is ideal for serving media to global users.

4.2 DATABASE DESIGN

The **MongoDB database** is structured into collections, each representing a specific data entity:

1. **User Collection**

   - Stores user details, including email, password (hashed), and profile picture.
   - Manages followers and following lists.

```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  email: {type: String, required: true, unique: true },
  password: {type: String,required: true },
  role: {type: String,enum: ['Admin', 'User'],default: 'User'}
}, { timestamps: true });

module.exports = mongoose.model('User', UserSchema);
```

Fig 4.1

2. **Location Collection**

- Stores the different locations and Address.
- Helps users filter and find showtimes, movies, and services available at their nearest.

```
const mongoose = require('mongoose');

const locationSchema = new mongoose.Schema({
    name: { type: String, required: true },
    address: { type: String, required: true },
}, { timestamps: true });

module.exports = mongoose.model('Location', locationSchema);
```

Fig 4.2

4.3 WORKFLOW EXPLANATION

- **Azure Boards** - Work item tracking and agile project management.
- **Git Repositories** - Source code management (frontend: React.js / backend: Express.js).
- **Pipelines (CI/CD)** - Automated building, testing, and deployment.
- **Azure Services** - Static Web Apps, App Services, Blob Storage, and MongoDB (Atlas or Cosmos DB).

4.3.1 PLANNING AND TASK MANAGEMENT (AZURE BOARDS)

- **Azure Boards** is used to manage the project tasks and ensure transparency and collaboration within the development team.
- The team follows **Agile methodology** with **Scrum** or **Kanban** boards for task tracking.
- **New → Active → Resolved → Closed**
- Tasks move through different states as they are worked on and completed.

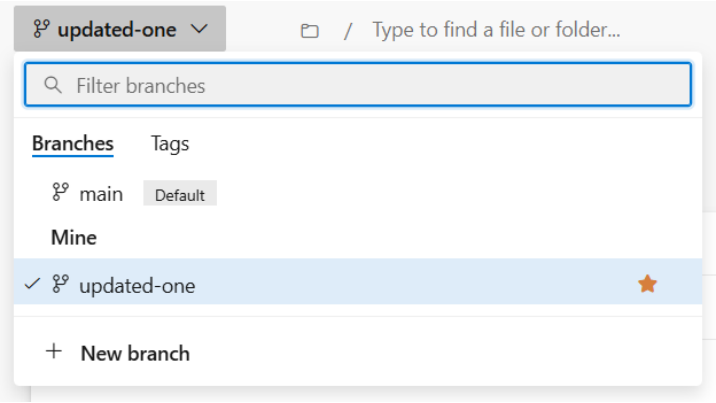
- **Tracking Progress**: Move cards across the Kanban board to show status.

Work Items in Boards:

Type	Description
Epics	Large features like "Movie Booking Module", "Food Ordering Module", "Secret House Gallery".
Features	Sub-features like "User Login", "Admin Add Movies", "Post Reviews".
User Stories	Specific tasks like "Design Login Page", "Implement JWT Authentication".
Tasks	Actionable steps like "Write MongoDB Schema for Movies", "Connect Azure Blob Storage".
Bugs	Identified issues during testing, e.g., "Payment gateway error".

4.3.2 DEVELOPMENT PHASE (GIT REPOS & BRANCH STRATEGY)

- Developers work on different features in **feature branches**.
- Code is pushed to Azure Repos (or GitHub, Visual Studio if linked).
- Pull Requests (PRs) are created and reviewed before merging to **main** or **develop** branches.



4.3.3 BUILD & CONTINUOUS INTEGRATION (AZURE PIPELINES)

- Azure Pipelines automatically build the frontend (React.js) and backend (Express.js) code when PRs are merged.
- Build Steps:
    - Install dependencies
    - Run unit tests

    - Bundle/Compile frontend
    - Lint and verify backend

- Artifacts are generated for deployment.

4.3.4 RELEASE & CONTINUOUS DEPLOYMENT (AZURE PIPELINES RELEASES)

- The Release Pipeline deploys:
    - Frontend to **Azure Static Web Apps**
    - Backend to **Azure App Service**
    - Media files to Azure Blob Storage
- Automatic triggers ensure that new features and fixes are deployed seamlessly after successful builds.

4.3.5 SUMMARY OF WORKFLOW WITH BOARDS INTEGRATION

- **Planning**: Use **Boards** to create and manage work items.
- **Development**: Track progress in **Boards** as developers push code.
- **CI Pipeline**: Auto-build and test on code commits.
- **CD Pipeline**: Deploy to Azure environments automatically.

CHAPTER-5

FEATURE IMPLEMENTATION

The **Strange-Shows** platform was developed with a focus on delivering a seamless user experience for movie ticket booking, food ordering, and user engagement through reviews and secret pages. This chapter outlines the core features that have been implemented in the system, including how each component was designed, developed, and deployed

5.1 MOVIE TICKET BOOKING MODULE

- **User Registration**: Users can sign up using a email, and password. The password is hashed using bcrypt before storing it in the MongoDB database.
- **Login & Logout**: Users log in using their email and password, and an authentication token (JWT) is generated and stored in cookies for session management.
- **Protected Routes**: Certain pages, like profile and post creation, are only accessible if the user is authenticated. Express.js middleware verifies JWT tokens for authorization.
- **Password Encryption**: Bcrypt is used to hash passwords, ensuring security and preventing unauthorized access.
- **User Role Handling:** If the user is an admin, additional links for the Admin Panel appear.

5.1.1 Home Page (For Users)

- **Navigation Bar**: Persistent across the platform, placed at the top of the page for easy access to the main sections of the site.
- **Location Selector:** Users can select their preferred location or theatre from a dropdown menu. Once selected, the content across the site (movies, food options, etc.) dynamically updates based on the chosen location.
- **Navigation Links:**
    - **Movies**
    - **Food**
    - **Reviews**

- **Secret Pages (Terrace View & Secret House)**
- **Logout Button**

5.1.2 Movie Page Section

Accessible from the navigation bar, this section displays:

- A **dynamic** list of Now Playing and Coming Soon movies.
- **Filters** by Location and date
- Interactive components like

  - **Movie posters**
  - **Showtimes**
  - **Seat availability**
  - **Booking buttons**

5.1.3 Now-Playing Section

- The **Now Playing** section displays a curated list of movies that are **currently running in theatres** at the selected location.
- Each movie card offers quick actions

5.1.4 Comin-Soon Section

- The **Coming Soon** section highlights movies that are scheduled for future release at different locations.
- This section builds excitement and anticipation for future releases.

5.1.5 Movie Calendar Section

- The **Movie Calendar** provides a **visual and interactive timeline** of all movies scheduled by date.
- Users can See which movies are playing on a specific date

Backend Support:

- Movies are fetched from the MongoDB Movies collection.
- Showtimes and bookings are handled through secure APIs.
- **Now-Playing Movies Collection:** Stores details of movies currently being shown.
- **Coming-Soon Movies Collection:** Stores upcoming movie details and release schedules.
- **Showtimes Collection:** Holds all showtimes linked to specific movies and dates for the calendar feature.

5.1.6 Food Page Section

This section allows users to:

- Browse and order from an extensive menu of snacks, beverages, and combo deals.
- Add items to the cart and checkout during ticket booking or separately.

Backend Support:

- Menu items are stored in the Foods collection.
- Orders are linked to user accounts and theater locations.

5.1.7 Unified Cart and Checkout

- Combines both movie ticket bookings and food orders into a single cart.
- Users can skip the food ordering step if they want, proceeding directly to checkout.

5.1.8 Order Confirmation and E-Tickets

- Once the payment is complete, users receive **E-Ticket(s)** for movie entry
- **Food Order Confirmation** with details for collection or delivery

5.1.9 Secret Navigation Area

An exclusive feature that sets **Strange-Shows** apart:

Terrace View:

- Acts as an OTT streaming page featuring exclusive or upcoming cinema content.
- Only accessible to registered users.
- Videos are streamed via Azure Blob Storage.

Secret House:

- A gallery of honour showcasing legendary movie directors, actors, and filmmakers.
- Includes profiles, trivia, and behind-the-scenes content.

5.1.10 Season Pass for Movies

- The **Season Pass** in the **Strange-Shows** platform is an exclusive subscription feature that allows users to **book a single seat for unlimited movies** for **one month**.
- Users can book their seats without additional charges during the active pass period.
- Early access to Coming Soon movies (if applicable).

5.2 ADMIN PANEL MODULE

- The **Admin Panel** in the Strange-Shows platform is the **central control hub** for managing the entire system.
- It enables authorized administrators (Show Owners or Location Managers) to **efficiently manage and monitor** the operations of the movie ticketing system, food court, user dashboards, and more across all locations.
- The Admin Panel provides **role-based access**, ensuring only authenticated admins have control over the content and operations.
- Built using **React.js** for the frontend and **Express.js** with **MongoDB** for the backend, the panel offers a **user-friendly interface** with seamless CRUD operations for every major feature.

5.2.1 Movie Management

- The Admin has complete control over the **movie listings** shown on the user dashboard.
- Now Playing Movies **CRUD**
    - Add new movies currently showing in theatres.
    - Edit movie details like title, description, release date, showtimes, cast, director, ticket price, and poster images.
    - Delete outdated or cancelled movies.
    - Manage multiple movie listings for different theatre locations.
- Coming Soon Movies **CRUD**
    - Add upcoming movies and Manage release dates
    - Keep users informed of future shows.
- Movie Calendar Control
    - Maintain a calendar view for all scheduled movies.
    - Easy rescheduling or cancellation of movie slots.

5.2.2 Location Management

- Admins can manage **multiple theatre locations** across cities using the Location Controller.
- **CRUD** Operations for Locations
    - Add new cinema locations with details like city, address, screen capacity, and available amenities.
    - Update location information, showtimes, and services.
    - Delete locations if a theatre is shut down or under maintenance.

5.2.3 Food Management

- Admins can **create and manage** food items available for order at each theater's Food Court.
- **CRUD** Operations for Food Items

- Add new food items with details such as name, category (snack, beverage, combo), description, price, and image.
- Edit existing food item details like price, description, and availability.
- Delete food items no longer available.

- Menu Personalization
  - Customize menus per location based on regional preferences or special promotions.
  - Offer special combos during blockbuster movie releases or events.

CHAPTER-6

API TESTING AND POSTMAN

API testing is a crucial step in ensuring that the backend of the Strange-Show project functions as expected. It helps verify the correctness, security, and performance of API endpoints before integrating them with the frontend. **Postman** is used as the primary tool for testing API requests, responses, and authentication mechanisms.

Key Aspects of API Testing:

- **Validation of API Endpoints**
  - Ensures that GET, POST, PUT, and DELETE requests return expected results.
  - Checks for proper request handling, including query parameters and request bodies.

- **Authentication Testing**
  - Tests user login and registration endpoints using JSON Web Token (JWT) authentication.
  - Validates that only authenticated users can access protected routes.

- **Error Handling and Response Codes**
  - Ensures that appropriate HTTP status codes are returned (200 for success, 400 for bad requests, 401 for unauthorized access, 500 for server errors).
  - Confirms that meaningful error messages are displayed for failed requests.

- **Performance and Load Testing**
  - Measures response times and evaluates API efficiency.
  - Simulates multiple concurrent requests to check server stability.

- **Database Validation**
  - Ensures data consistency by verifying that changes made through API requests are accurately reflected in MongoDB.
  - Confirms that user actions like posting, liking, and commenting update the database correctly.

- **Security Testing**
  - Tests for vulnerabilities such as SQL injection, unauthorized access, and improper data handling.
  - Ensures that sensitive user data remains encrypted and protected.

Using Postman for API Testing:

- **Creating and Sending Requests**
  - Postman allows developers to manually test API endpoints by sending HTTP requests with different payloads.
  - Supports request customization with headers, authentication tokens, and request parameters.

- **Automating API Tests**
  - Postman enables the creation of automated test scripts using JavaScript.
  - Test cases can be saved, reused, and run in bulk to validate multiple endpoints efficiently.

By leveraging Postman, developers can efficiently test and debug the backend, ensuring a smooth and secure integration with the frontend. This improves the overall reliability of the X Clone project while streamlining the development workflow.

CHAPTER-7

FUTURE ENHANCEMENTS

As *Strange-Shows* continues to grow, several exciting enhancements are planned for future development to elevate the platform and expand its reach both technologically and globally.

More Secret Pages & Exclusive Content

- Develop additional **Secret Pages** offering unique experiences, such as:
- **Director's Cut** sections for behind-the-scenes content.
- **Terrace-View+**, an upgraded OTT platform featuring exclusive movies and series.
- A Secret Page that contains More Drama, that seems Dramatic. An Immense experience

Global Expansion & International Recognition

- Expand *Strange-Shows* to multiple **countries**, making it an **internationally recognizable cinema brand**.
- Collaborate with **global movie distributors** and **film festivals** to stream and showcase exclusive global content.

IMAX and Premium Theatre Integration

- Build and integrate **more IMAX screens** and **premium theatre experiences** into the Strange-Shows network.
- Provide **virtual IMAX experiences** through Terrace-View for online users.
- Develop partnerships to bring the **latest technologies in cinema viewing**, including **4DX, Dolby Atmos, and VR theatres**.

Enhanced User Personalization

- Implement AI-powered recommendations for movies, food, and events based on user preferences and viewing history.
- Offer **season passes with customizable packages**, such as family plans or VIP experiences.

Improved DevOps & Infrastructure Scalability

- Optimize **CI/CD pipelines** for faster deployments across multiple regions using **Azure DevOps**.
- Enhance **monitoring and security** for global operations by integrating **Azure Security Centre** and **Application Insights**.

Review & Rating Page Enhancements

- Users will be able to upload **video or audio reviews** along with their text-based feedback, making reviews more engaging and authentic.
- Introduce **verified viewer badges** for users who have actually booked and watched the movie, adding credibility to reviews.
- Allow users to **react (like, love, insightful, etc.)** and comment on other users' reviews, creating a **community-driven discussion space**.
- Implement **spoiler detection** and filters to ensure users can safely read reviews without having the plot spoiled.

User Profile Enhancements

- Introduce **badges** for milestones like "First Review", "Top Foodie", "IMAX Enthusiast", and "Cinema Explorer".
- Provide a **timeline view** showing all movies watched, locations visited, and foods ordered.
- Add options for users to create **personal wishlists** of movies they want to watch or **food items** they plan to try.
- Let users **customize profile pictures**, **cover photos**, and **profile themes** to personalize their experience.

User Insight Tracker Enhancements

- Display **monthly or weekly insights** on movies watched, money spent, locations visited, and reviews posted.
- Provide users with **spending breakdowns** and **budget-setting tools** for movies, food, and events.

- Track and show a **genre preference chart** based on the movies watched, helping users understand their interests.
- Users can set **goals** like “Watch 10 movies this month” and earn **reward points** or **discounts** when achieved.
- Use insight data to offer **personalized movie and food suggestions**, as well as **exclusive offers** based on their habits.

Vision for Enhancements

These enhancements are designed to foster **deeper user engagement**, create a **personalized cinema experience**, and make Strange-Shows a **community-centric platform** where users feel valued and connected.

CHAPTER-8

CONCLUSION

The **Strange-Shows Movie Ticket Booking Software** has been successfully designed and implemented as an all-in-one platform that enhances the cinema-going experience. The system integrates **movie ticket booking**, **food ordering**, **customer reviews**, and **exclusive secret pages** like **Terrace View** (OTT content) and **Secret House** (Gallery of Honor), providing users with an immersive and engaging environment.

This project leverages modern web technologies such as **React.js** for the frontend, **Express.js** for the backend, **MongoDB** for database management, and **Microsoft Azure** for deployment and CI/CD pipeline management through **Azure DevOps**. By combining these technologies, the system ensures **scalability**, **performance**, **security**, and **user-friendly interaction**.

Additionally, **admin features** such as **location management**, **CRUD operations** for movies, foods, and reviews, and **user dashboard controls** enable effective administration of the entire platform. Features like **season passes**, **review and rating modules**, and **user insight tracking** further enhance the platform's capabilities.

Through this project, we demonstrated how cloud technologies and modern development practices like **DevOps pipelines** can streamline the deployment process and improve continuous integration and delivery (CI/CD). The **Azure Static Web Apps** and **Azure Blob Storage** integrations help efficiently manage large media files and static content.

The successful implementation of the **Strange-Shows** project highlights the potential of combining **React.js**, **Express.js**, **MongoDB**, and **Microsoft Azure** to develop high-performance, cloud-based applications. It serves as a foundation for future innovations in the entertainment and cinema industries.

CHAPTER-9

BIBLIOGRAPHY

1) **Banks, A., & Porcello, E. (2017). *Learning React: Modern Patterns for Developing React Apps.* O'Reilly Media.**

   - **URL: https://react.dev**
   - **Relevance: Provided insights into building dynamic user interfaces using React.js for the frontend of the Strange-Show DEVOPS project.**

2) **Casciaro, A., & Mammino, L. (2020). *Node.js Design Patterns.* Packt Publishing.**

   - **URL: https://nodejs.org/en/docs**
   - **Relevance: Helped in designing scalable and efficient backend architecture using Node.js and Express.js.**

3) **MongoDB Inc. (2023). *MongoDB Official Documentation.***

   - **URL: https://www.mongodb.com/docs**
   - **Relevance: Provided knowledge on database management, schema design, and indexing for optimizing query performance in MongoDB Atlas.**

4) **JWT.io (2023). *JSON Web Token Introduction.***

   - **URL: https://jwt.io/introduction**
   - **Relevance: Used for implementing secure authentication mechanisms in the project.**

5) **Postman Team (2023). *Postman API Testing Documentation.***

   - **URL: https://learning.postman.com/docs**
   - **Relevance: Helped in testing and debugging API endpoints before frontend integration.**

6) ***Azure Blob Storage Image & Video Management.***

- **URL: https://learn.microsoft.com/en-us/azure/storage/blobs/**
- **Relevance: Used for handling media uploads, optimizing images, and delivering stored assets.**

7) ***Microsoft Azure Portal. Deployment Documentation.***

- **URL: https://learn.microsoft.com/en-us/azure/azure-portal/**
- **Relevance: Helped in deploying the frontend and backend server securely with scalable infrastructure.**

8) ***Azure DEVOPS Documentation.***

- **URL: https://learn.microsoft.com/en-us/azure/devops/?view=azure-devops**
- **Relevance: Used for modern software development by promoting collaboration between development and operations teams.**

9) ***Azure CI/CD Pipeline Documentation.***

- **URL:https://learn.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops**
- **Relevance: Used for automatically deploys applications to production or staging environments.**

10) ***Azure YAML Documentation.***

- **URL:https://learn.microsoft.com/en-us/azure/devops/pipelines/yaml-schema/?view=azure-pipelines**
- **Relevance: Helped in one or more stages that describe a CI/CD process.**

CHAPTER-10

OUTPUT AND SCREENSHOTS

SIGN-UP

```
const Signup = () => {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [confirmPassword, setConfirmPassword] = useState("");
    const [error, setError] = useState("");

    const navigate = useNavigate();

    const handleSubmit = async (e) => {
        e.preventDefault();

        // Validate password match
        if (password !== confirmPassword) {
            setError("Passwords do not match!");
            return;
        }

        try {
            // Send POST request to your backend
            const response = await axios.post(
                "http://localhost:5000/api/auth/signup",
                {
                  email,
                  password,
                  confirmPassword
                },
                {
                  headers: {
                    "Content-Type": "application/json"
                  }
                }
              );

            console.log("Signup success", response.data);
```

LOG-OUT

```
export const logout = async (req,res)=>{
    try {
        res.cookie("jwt","",{maxAge:0})
        res.status(200).json({message: "Logged out successfully"})
    } catch (error) {
        console.log("Error in logout controller", error.message);
        res.status(500).json({error: "Internal Server Error"});
    }
}
```

LOG-IN

```
const Login = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const navigate = useNavigate();

  // Predefined admin credentials (hardcoded for now)
  const adminEmail = "dine@gmail.com";
  const adminPassword = "123456";

  const handleSubmit = async (e) => {
    e.preventDefault();

    // Check for admin credentials FIRST
    if (email === adminEmail && password === adminPassword) {
      // Admin login successful!
      localStorage.setItem("token", "admin_dummy_token"); // optional
      alert("Logged in as Admin");
      navigate("/admin-ticket-booking");
      return;
    }
```

USER-DASHBOARD

```
const Dashboard = () => {
  const [isAdmin, setIsAdmin] = useState(false);
  const [selectedLocation, setSelectedLocation] = useState(() => {
    // Load from localStorage initially
    return localStorage.getItem("selectedLocation") || "Select Location";
  });

  const navigate = useNavigate();

  useEffect(() => {
    const token = localStorage.getItem("token");
    if (!token) {
      navigate("/login");
      return;
    }

    try {
      const decodedToken = jwtDecode(token);

      if (!decodedToken || !decodedToken.role || decodedToken.exp * 1000 < Date.now()) {
        localStorage.removeItem("token");
        navigate("/login");
        return;
      }
```

NOW-PLAYING MOVIES

```
/ Fetch movies when selectedLocation changes
seEffect(() => {
 if (!selectedLocation) return;

 const fetchMovies = async () => {
   try {
     setLoadingMovies(true);
     const response = await axios.get(`${API_BASE_URL}/api/nowplaying?location=${selectedLocation}`);

     console.log('Fetched movies:', response.data);

     if (response.data.success && Array.isArray(response.data.data)) {
       setMovies(response.data.data);
       setError('');
     } else {
       setMovies([]);
       setError(response.data.message || 'No movies available.');
     }

   } catch (error) {
     console.error('Error fetching movies:', error);
     setError('Error fetching movies.');
   } finally {
     setLoadingMovies(false);
   }
 };
```

NAVIGATION-BAR

```
const Navigation = ({ isAdmin, selectedLocation, setSelectedLocation }) => {
  const navigate = useNavigate();

  const [menuOpen, setMenuOpen] = useState(null); // null, 'movies', 'mobile'
  const [settingsOpen, setSettingsOpen] = useState(false);

  const [locations, setLocations] = useState([]);
  const [loadingLocations, setLoadingLocations] = useState(true);
  const [locationError, setLocationError] = useState("");

  const handleLogout = () => {
    localStorage.removeItem("token");
    navigate("/login");
  };
```

ADMIN PANEL

```
import MoviesManagement from "./MoviesManagement"; // New Movies Controller Page
import FoodManagement from "./FoodManagement";
import InsightsTracker from "./InsightsTracker";
import LocationManagement from "./LocationManagement"; // New component

const AdminTicketBooking = () => {
    const navigate = useNavigate();
    const [activeTab, setActiveTab] = useState("movies");
    const [showSettings, setShowSettings] = useState(false);

    const handleLogout = () => {
        localStorage.removeItem("token");
        navigate("/login");
    };
```

LOCATION MANAGEMENT

```
const LocationManagement = () => {
    const [locations, setLocations] = useState([]);
    const [newLocation, setNewLocation] = useState({ name: "", address: "" });
    const [editingLocation, setEditingLocation] = useState(null);
    const [loading, setLoading] = useState(true); // true on initial load
    const [error, setError] = useState("");

    const BASE_URL = "http://localhost:5000/api/locations";

    useEffect(() => {
        fetchLocations();
    }, []);

    // ✅ Fetch Locations
    const fetchLocations = async () => {
        setLoading(true);
        setError("");

        try {
            const response = await fetch(BASE_URL);
            if (!response.ok) throw new Error("Failed to fetch locations");

            const data = await response.json();
            setLocations(data);
        } catch (err) {
            console.error(err);
            setError(err.message || "An error occurred while fetching locations");
        } finally {
            setLoading(false);
        }
    };
```

MOVIE MANAGEMENT

```
const NowPlayingAdmin = () => {
  const [movies, setMovies] = useState([]);
  const [locations, setLocations] = useState([]);
  const [formData, setFormData] = useState({
    movieTitle: '',
    description: '',
    releaseDate: '',
    showTime: '',
    ticketPrice: '',
    directorName: '',
    cast: '',
    location: '',
    poster: null,
  });
  const [editingId, setEditingId] = useState(null);
  const [existingPoster, setExistingPoster] = useState('');
  const [message, setMessage] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const API_BASE_URL = 'http://localhost:5000/api';
```

FOOD MANAGEMENT

```
const FoodManagement = () => {
  const [foods, setFoods] = useState([]);
  const [form, setForm] = useState({
    id: null,
    name: "",
    price: "",
    description: "",
    image: null,
    imagePreview: null,
  });
  const [isEditing, setIsEditing] = useState(false);
```

FOOD ORDER

```
const Foo  const foodItems: never[]
  const [foodItems, setFoodItems] = useState([]);
  const [cart, setCart] = useState([]);
  const navigate = useNavigate();

  useEffect(() => {
    fetchFoodItems();
  }, []);

  const fetchFoodItems = async () => {
    try {
      const response = await fetch("http://localhost:5000/api/food");
      if (!response.ok) {
        throw new Error("Failed to fetch food items");
      }
      const data = await response.json();
      setFoodItems(data);
    } catch (error) {
      console.error("Error fetching food items:", error);
    }
  };
```

ADMIN-COMINGSOON

```
const AdminComingSoon = () => {
  const [movies, setMovies] = useState([]);
  const [form, setForm] = useState({ title: "", description: "", releaseDate: "", posterUrl: "" });

  useEffect(() => {
    fetchMovies();
  }, []);

  const fetchMovies = async () => {
    const res = await fetch("http://localhost:5000/api/coming-soon");
    const data = await res.json();
    setMovies(data);
  };

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };
```

COMING-SOON

```
const ComingSoon = () => {
  const [movies, setMovies] = useState([]);

  useEffect(() => {
    fetch("http://localhost:5000/api/coming-soon")
      .then((res) => res.json())
      .then((data) => setMovies(data))
      .catch((err) => console.error(err));
  }, []);
```

TERRACE-SHOWS

```
const TerraceShows = () => {
  const [selectedShow, setSelectedShow] = useState(null);
  const [expandedShowId, setExpandedShowId] = useState(null);
  const [playingEpisode, setPlayingEpisode] = useState(null);
  const navigate = useNavigate();
  const bgAudioRef = useRef(null);

  useEffect(() => {
    const audio = new Audio('/audio/bgsound.mp3');
    audio.volume = 0.5;
    audio.loop = true;
    audio.play().catch(err => console.log('Autoplay error:', err));
    bgAudioRef.current = audio;

    return () => {
      if (bgAudioRef.current) {
        bgAudioRef.current.pause();
        bgAudioRef.current = null;
      }
    };
  }, []);
```

SECRET-HOUSE

```
const SecretHouse = () => {
  const [audioPlayed, setAudioPlayed] = useState(false);
  const [selectedDirector, setSelectedDirector] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    if (!audioPlayed) {
      const audio = new Audio('/audio/secret-house.mp3');
      audio.volume = 0.5;
      audio.play().catch((err) => console.log('Autoplay error:', err));
      setAudioPlayed(true);
    }
  }, [audioPlayed]);

  const goBackToTerraceShows = () => {
    const soundEffect = new Audio('/audio/back-sound.mp3');
    soundEffect.play().then(() => {
      navigate('/terrace-shows');
    }).catch(() => {
      navigate('/terrace-shows');
    });
  };
```

SHOW-PREVIEW

```
const backdrop = {
  visible: { opacity: 1 },
  hidden: { opacity: 0 }
};

const modal = {
  hidden: { opacity: 0, y: '-100vh' },
  visible: { opacity: 1, y: '0', transition: { delay: 0.3 } }
};
```

DATABASE CONNECTION

```
const PORT = process.env.PORT || 5000;
const MONGO_URI = process.env.MONGO_URI;

const connectDBAndStartServer = async () => {
    try {
        await mongoose.connect(MONGO_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true
        });
        console.log('✅ MongoDB Connected');

        app.listen(PORT, () => {
            console.log(`🚀 Server running at http://localhost:${PORT}`);
        });
    } catch (error) {
        console.error('❌ MongoDB connection failed:', error.message);
        process.exit(1);
    }
};

connectDBAndStartServer();
```

UPDATE USER

```
exports.signup = async (req, res) => {
    const { email, password, confirmPassword, role } = req.body;

    // Validate matching passwords
    if (password !== confirmPassword) {
      return res.status(400).json({
        success: false,
        message: "Passwords do not match!"
      });
    }

    // Check if the user already exists
    const userExists = await User.findOne({ email });
    if (userExists) {
      return res.status(400).json({
        success: false,
        message: "User already exists!"
      });
    }

    // Hash password
    const hashedPassword = await bcrypt.hash(password, 10);

    // Create new user
    const newUser = new User({
      email,
      password: hashedPassword,
      role: role || 'User'
    });

    // Save to DB
    await newUser.save();

    res.status(201).json({
      success: true,
      message: "User registered successfully!"
    });
```

JWT TOKEN GENERATION

```
// Generate JWT
const token = jwt.sign(
  { id: user._id, role: user.role },
  process.env.JWT_SECRET,
  { expiresIn: '1d' }
);
```

ROUTES

```
// ✅ Auth Routes
app.use('/api/auth', authRoutes);

// ✅ Location Management Routes
app.use('/api/locations', locationRoutes);

// ✅ Coming Soon Movies Routes
app.use('/api/coming-soon', comingSoonRoutes);

// ✅ Now Playing Movies Routes
// Based on your nowPlayingRoutes.js structure, this endpoint serves all CRUD functions
app.use('/api/nowplaying', nowPlayingRoutes);
```

USER SCHEMA (MODEL)

express-backend > models > JS user.js > ...

```
1   const mongoose = require('mongoose');
2
3   const UserSchema = new mongoose.Schema({
4     email: {type: String, required: true, unique: true },
5     password: {type: String,required: true },
6     role: {type: String,enum: ['Admin', 'User'],default: 'User'}
7   }, { timestamps: true });
8
9   module.exports = mongoose.model('User', UserSchema);
10
```

LOCATION SCHEMA

press-backend > models > JS Location.js > ...

```
1   const mongoose = require('mongoose');
2
3   const locationSchema = new mongoose.Schema({
4       name: { type: String, required: true },
5       address: { type: String, required: true },
6   }, { timestamps: true });
7
8   module.exports = mongoose.model('Location', locationSchema);
9
```

MOVIE SCHEMA

```
const mongoose = require('mongoose');

const nowPlayingMovieSchema = new mongoose.Schema({
  movieTitle: {type: String,required: true,trim: true},
  description: {type: String,required: true,trim: true},
  releaseDate: {type: Date,required: true},
  showTime: {type: String,required: true,trim: true},
  ticketPrice: {type: Number,required: true,min: [0, 'Ticket price must be positive']},
  directorName: {type: String,required: true,trim: true},
  cast: [{type: String,trim: true}],
  poster: {type: String,default: '' },
  location: {type: String,required: true,trim: true}
}, {
  timestamps: true, // automatically adds createdAt and updatedAt
  versionKey: false, // ✅ Removes __v field unless you want versioning
});


module.exports = mongoose.model('NowPlayingMovie', nowPlayingMovieSchema);
```

MAIN.JSX

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App.jsx";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

SERVER.JS

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');
const helmet = require('helmet');
const morgan = require('morgan');
const rateLimit = require('express-rate-limit');
const compression = require('compression'); // ✅ Added compression
const fs = require('fs');
const path = require('path');


const uploadDir = './uploads';
if (!fs.existsSync(uploadDir)) {
    fs.mkdirSync(uploadDir);
}


dotenv.config();

const authRoutes = require('./routes/authRoutes');
const locationRoutes = require('./routes/locationRoutes');
const comingSoonRoutes = require('./routes/comingSoonRoutes');
const nowPlayingRoutes = require('./routes/nowPlayingRoutes'); // ✅ Make sure this matches your filename

// App Initialization
const app = express();
```

APP.JSX

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App.jsx";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

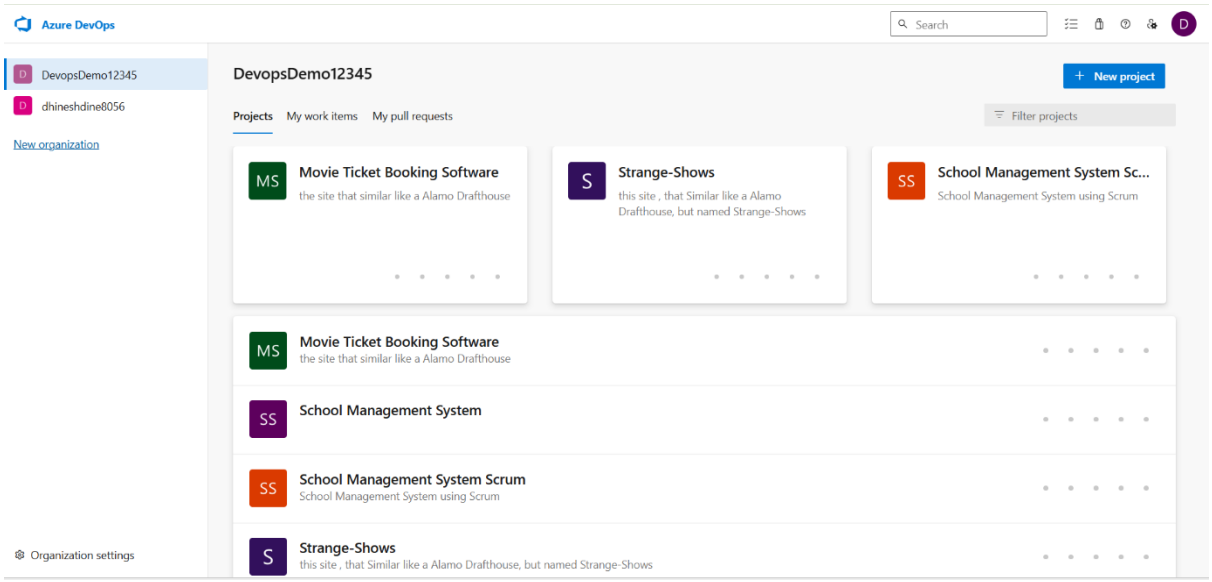
TAILWIND.CONFIG.JS

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./index.html", "./src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {},
  },
  plugins: [require("@tailwindcss/forms")],
};
```

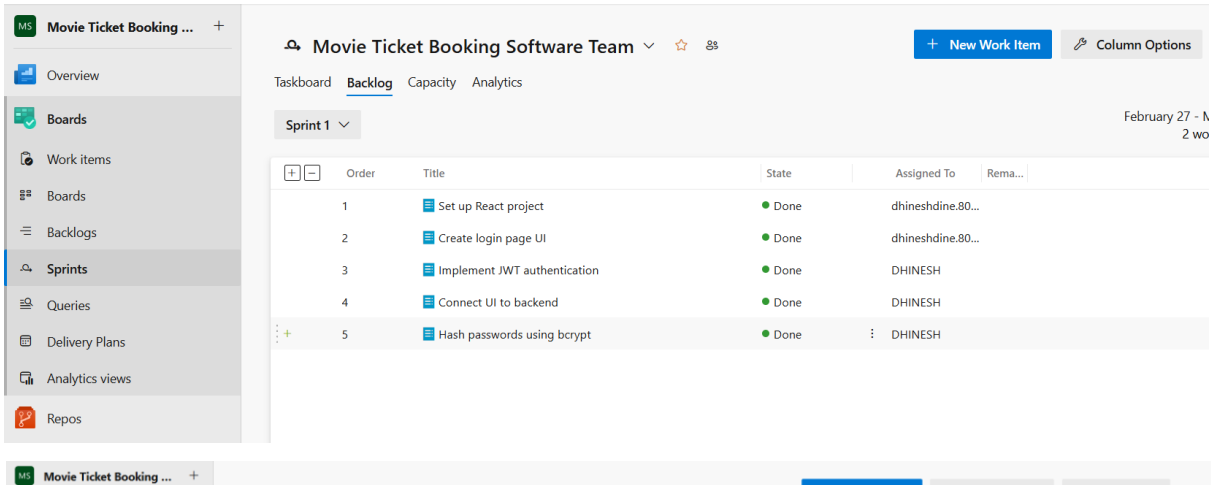
DEPENDENCIES

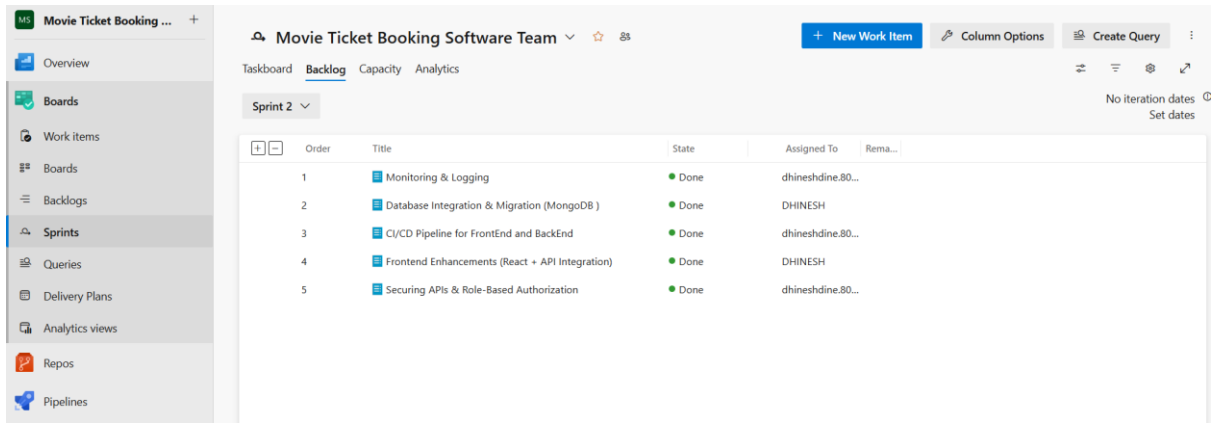
```
"dependencies": {
  "@tailwindcss/forms": "^0.5.10",
  "axios": "^1.8.3",
  "cors": "^2.8.5",
  "framer-motion": "^12.5.0",
  "jwt-decode": "^4.0.0",
  "lucide-react": "^0.479.0",
  "motion": "^12.5.0",
  "react": "^19.0.0",
  "react-dom": "^19.0.0",
  "react-icons": "^5.5.0",
  "react-player": "^2.16.0",
  "react-router-dom": "^7.3.0",
  "react-toastify": "^11.0.5",
  "recharts": "^2.15.1"
},
```

AZRUE DEVOPS ORGANIZATION

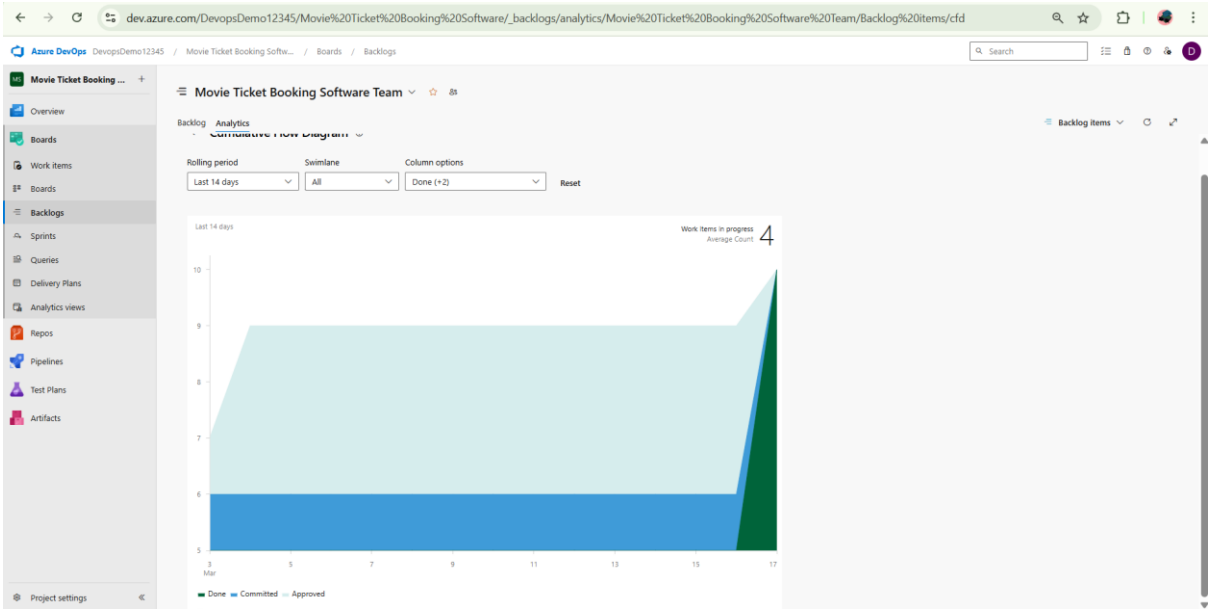


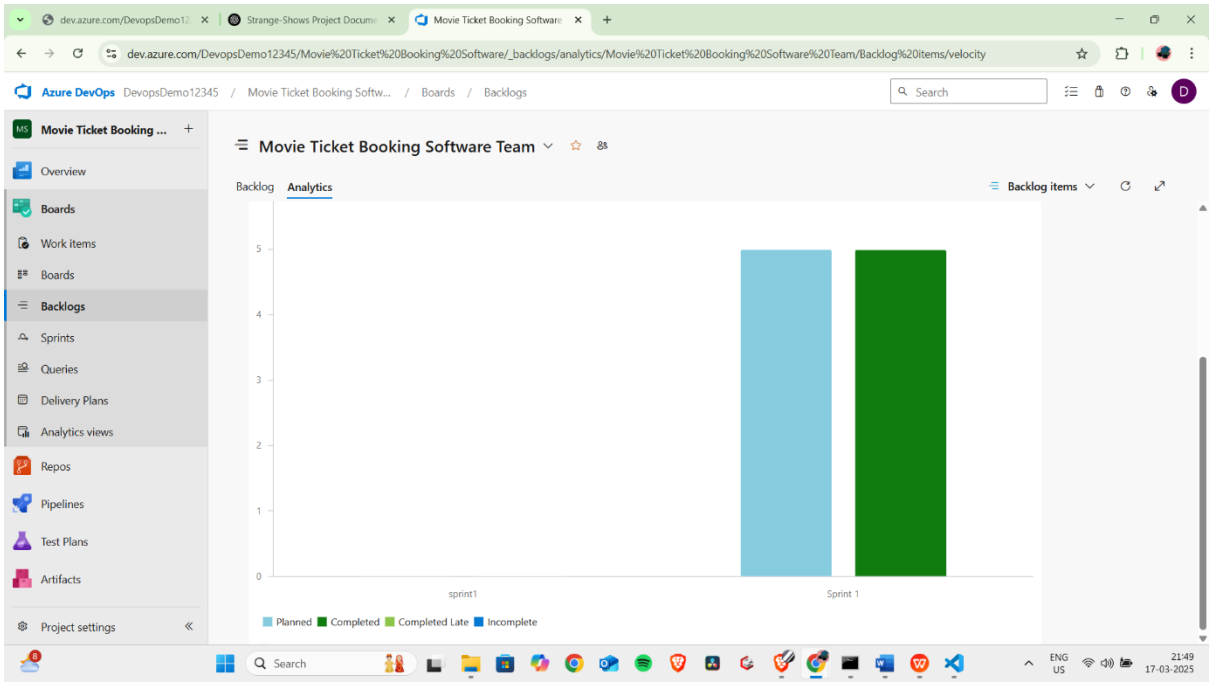
AZURE BOARDS



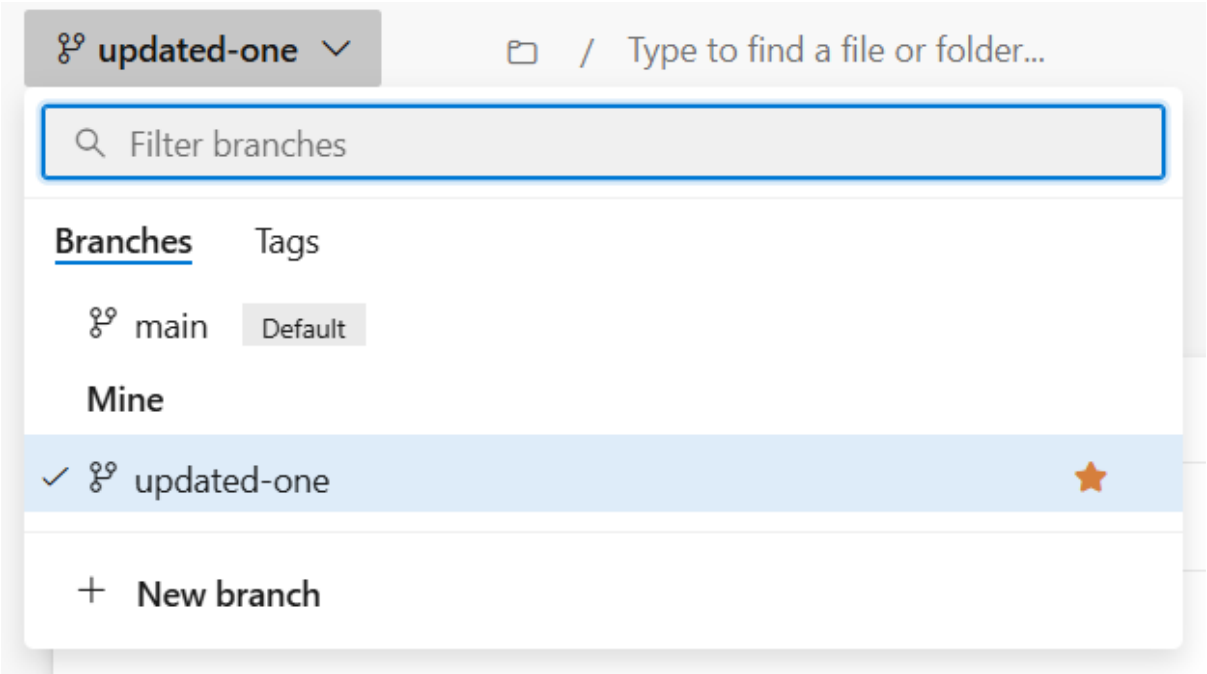


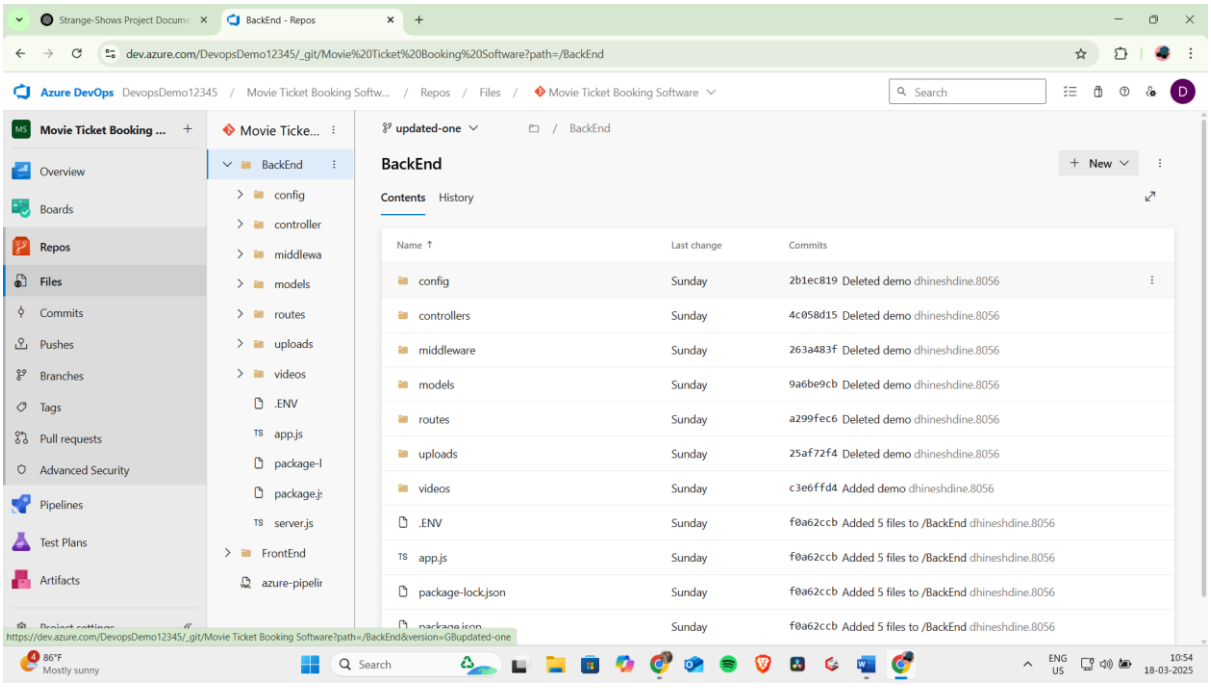
AZURE ANALYTIC DASHBOARD





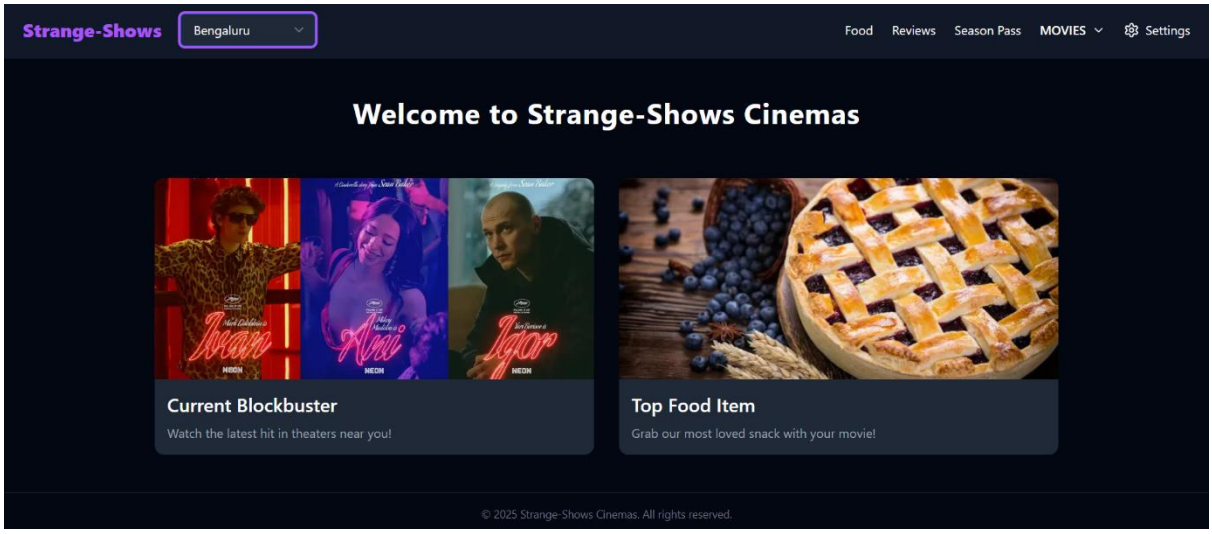
AZURE BRANCHS AND FOLDER STRUCTURE



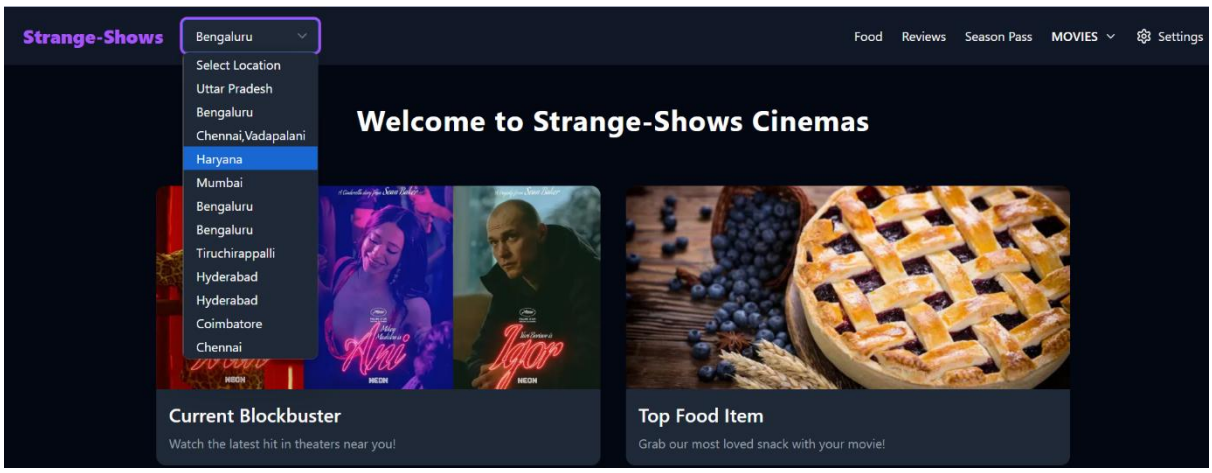


PROJECT OUTPUT SCREENS

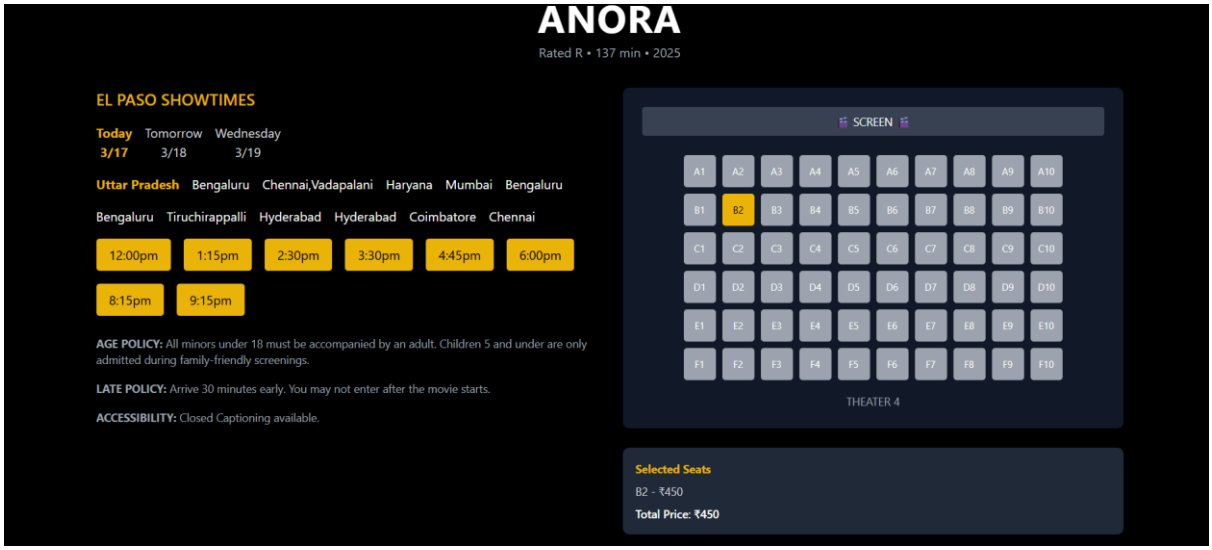
DASHBOARD AND NAVIGATION PAGE



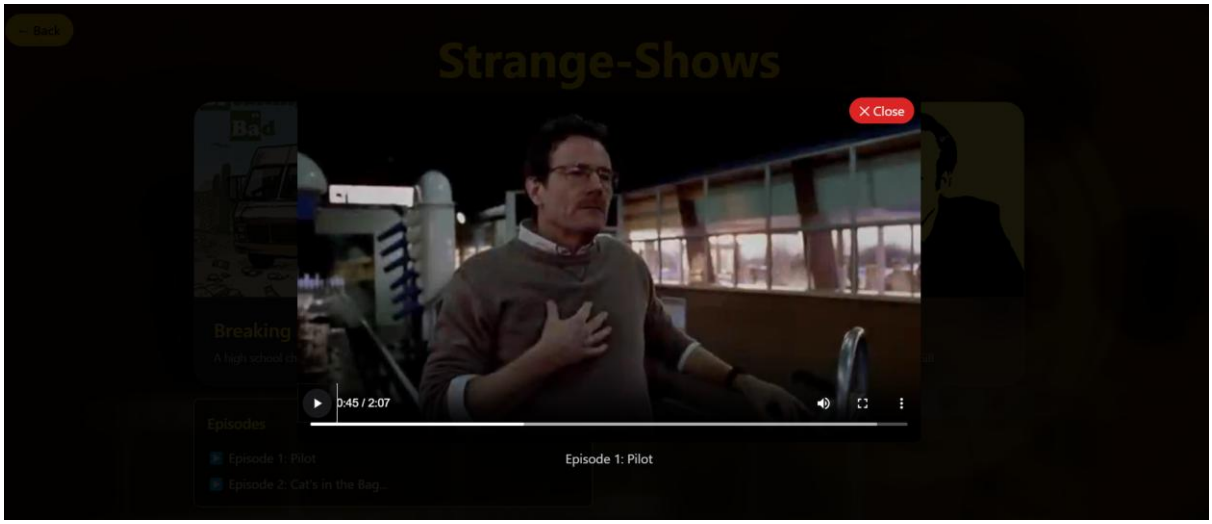
LOCATION SEARCH



BOOKING PAGE



STRANGE SHOWS



SECRET HOUSE