Ticket Flix

Mini Project Report

Submitted by

TANSYA BABU

Reg. No.: AJC20MCA-I054

In Partial fulfillment for the Award of the Degree of

INTEGRATED MASTER OF COMPUTER APPLICATIONS

(INMCA)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2024-2025

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

CERTIFICATE

This is to certify that the Project report, "TICKETFLIX" is the bona fide work of **TANSYA BABU (Regno: AJC20MCA-I054)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2024-25.

Dr Bijimol T.K.	**Mr.Binumon Joseph**
Internal Guide	**Coordinator**

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

DECLARATION

I hereby declare that the project report **"TICKETFLIX"** is a bona fide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the **Integrated Master of Computer Applications (INMCA)** from **APJ Abdul Kalam Technological University**, during the academic year **2024-2025.**

Date: 04/11/2024 **TANSYA BABU**

KANJIRAPPALLY **Reg: AJC20MCA-I054**

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Director (Administration) **Rev. Fr. Dr. Roy Abraham Pazhzyaparampil** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude towards our Head of the Department **Rev.Fr.Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Mr. Binumon Joseph, Assistant professor** for his valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Dr. Bijimol T.K., Associate professor** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

TANSYA BABU

ABSTRACT

TicketFlix is an innovative online ticket booking platform tailored to bring convenience and excitement to movie enthusiasts. Designed with user experience in mind, TicketFlix provides a seamless and efficient way for users to browse movie listings, view showtimes, and book tickets at preferred theaters. With a sleek and responsive interface, it allows users to make quick bookings, select seats, and manage their reservations from any device, bringing the magic of cinema right to their fingertips.

The platform is more than just a ticketing service; it's an ecosystem that connects users and theaters, offering tools for theaters to manage movie schedules, seat availability, and promotional offers directly through the TicketFlix dashboard. Users can filter movies by genre, language, and viewing format (such as 3D or IMAX), enhancing their selection experience. TicketFlix also enables loyalty rewards, discounts, and special offers, making it a preferred choice for frequent moviegoers who seek added value and flexibility.

TicketFlix aims to elevate the movie-going experience by reducing the hassle of traditional ticket lines and integrating features like notifications on upcoming releases and recommendations based on viewing history. This level of personalization and functionality keeps users engaged and helps theaters maintain steady occupancy rates. With its focus on efficiency, reliability, and user satisfaction, TicketFlix is shaping the future of cinema by making movie ticket booking more accessible, enjoyable, and rewarding for all.

CONTENT

SL. NO	TOPIC	PAGE NO
1	INTRODUCTION	1
1.1	PROJECT OVERVIEW	2
1.2	PROJECT SPECIFICATION	2
2	SYSTEM STUDY	4
2.1	INTRODUCTION	5
2.2	LITERATURE REVIEW	5
2.3	PROPOSED SYSTEM	7
2.4	ADVANTAGES OF PROPOSED SYSTEM	7
3	REQUIREMENT ANALYSIS	8
3.1	FEASIBILITY STUDY	9
3.1.1	ECONOMICAL FEASIBILITY	9
3.1.2	TECHNICAL FEASIBILITY	10
3.1.3	BEHAVIORAL FEASIBILITY	10
3.1.4	FEASIBILITY STUDY QUESTIONNAIRE	10
3.2	SYSTEM SPECIFICATION	12
3.2.1	HARDWARE SPECIFICATION	12
3.2.2	SOFTWARE SPECIFICATION	12
3.3	SOFTWARE DESCRIPTION	12
3.3.1	DJANGO	12
3.3.2	SQLite	13
4	SYSTEM DESIGN	14
4.1	INTRODUCTION	15
4.2	UML DIAGRAM	15
4.2.1	USE CASE DIAGRAM	15
4.2.2	SEQUENCE DIAGRAM	16
4.2.3	ACTIVITY DIAGRAM	18
4.2.5	CLASS DIAGRAM	20
4.2.6	OBJECT DIAGRAM	21
4.3	USER INTERFACE DESIGN USING FIGMA	23
4.4	DATABASE DESIGN	24
4.4.1	RDBMS	24

4.4.2	NORMALIZATION	24
4.4.3	SANITIZATION	27
4.4.4	INDEXING	27
4.5	TABLE DESIGN	27
5	SYSTEM TESTING	35
5.1	INTRODUCTION	36
5.2	TEST PLAN	36
5.2.1	UNIT TESTING	36
5.2.2	INTEGRATION TESTING	37
5.2.3	VALIDATION TESTING	37
5.2.4	USER ACCEPTANCE TESTING	37
5.2.5	AUTOMATION TESTING	38
5.2.6	SELENIUM TESTING	38
6	IMPLEMENTATION	51
6.1	INTRODUCTION	52
6.2	IMPLEMENTATION PROCEDURE	52
6.2.1	USER TRAINING	53
6.2.2	TRAINING ON APPLICATION SOFTWARE	53
6.2.3	SYSTEM MAINTENANCE	54
6.2.4	HOSTING	54
7	CONCLUSION & FUTURE SCOPE	57
7.1	CONCLUSION	58
7.2	FUTURE SCOPE	58
8	BIBLIOGRAPHY	59
9	APPENDIX	61
9.1	SAMPLE CODE	62
9.2	SCREEN SHOTS	77
9.3	GIT LOG	80

List of Abbreviation

IDE	-	Integrated Development Environment
HTML	-	Hyper Text Markup Language
CSS	-	Cascading Style Sheet
UML	-	Unified Modeling Language
MVT	-	Model-View-Template
NLP	-	Natrual Language Processing
JS	-	JavaScript
AI	-	Artificial Intelligence
API	-	Application Programming Interface
SQL	-	Structured Query Language
UI	–	User Interface
UX	-	User Experience
CRUD	-	Create,Reead,Update,Delete
JWT	-	JSON Web Token
ORM	-	Object-Relational Mapping
SEO	-	Search Engine Optimization
SSL	-	Secure Sockets Layer
CDN	-	Content Delivery Network
AJAX	-	Asynchronous JavaScript and XML

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The **Ticket Flix** is a dynamic web-based platform designed to streamline the process of browsing, selecting, and booking movie tickets at various theaters. It provides a seamless user experience with features such as user registration, personalized movie recommendations, showtime listings, interactive seat selection, secure payment options, and booking history tracking. For cinema operators, the system includes robust administrative tools for managing movies, theaters, showtimes, and user bookings, along with advanced reporting and analytics. By leveraging modern web technologies like Python, Django, and machine learning algorithms, the system enhances personalization, offers AI-driven recommendations, and integrates advanced functionalities like in-seat food ordering and chatbots for customer support, making it a comprehensive solution for both users and administrators.

1.2 PROJECT SPECIFICATION

The **Ticket Flix** is designed to deliver a comprehensive and efficient platform for movie-goers and cinema operators. Below are the key specifications of the project:

1. User-Side Functionalities:

- **User Registration and Login**: Users can create an account with secure authentication.
- **Profile Management**: Edit personal details, preferences, and update contact information.
- **Movie Browsing**: View a list of movies, filter by genre, language, or rating, and access detailed information.
- **Showtime Listings and Theater Selection**: Find available showtimes for preferred theaters.
- **Seat Selection**: Choose seats through an interactive layout that displays available and occupied seats.
- **Booking and Payment**: Complete booking through various secure payment gateways.
- **Booking Confirmation**: Receive instant booking confirmation via email or SMS.
- **Booking History**: View past and upcoming bookings.
- **Reviews and Ratings**: Provide feedback on movies and the booking experience.

2. Admin-Side Functionalities:

- **Admin Login**: Secure access to the admin dashboard.

- **Movie Management**: Add, update, or remove movies in the system.
- **Showtime Management**: Configure and manage showtimes for different theaters.
- **Theater Management**: Manage theater details, seating arrangements, and locations.
- **User Management**: Monitor registered users, including activity and profile information.
- **Booking Management**: Oversee all user bookings and cancellations.
- **Reports and Analytics**: Generate reports on user activity, movie trends, revenue, and other metrics.
- **Notifications**: Send updates to users about promotions, upcoming releases, or changes in showtimes.

3. Theatre Owner Functionalities:

- **User Interaction Management:** Respond to user reviews, ratings, and feedback related to the theater.
- **Food and Beverage Management:** Optionally manage in-seat services, including food and beverage pre-booking options.
- **Notifications and Announcements:** Send notifications or updates about upcoming shows, special events, or changes in schedules.
- **Reports and Analytics:** Access insights into booking trends, peak showtimes, and movie popularity specific to the theater.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

The proposed system, Ticket Flix, is a web-based movie ticket booking platform designed to address the challenges identified in traditional booking methods by leveraging the benefits of digital transformation. Ticket Flix aims to provide a seamless, efficient, and user-centered experience by enabling customers to view movie listings, check showtimes, select preferred seats, and complete payments all in one unified platform. For cinema operators, Ticket Flix offers robust administrative tools, including automated scheduling, seat availability tracking, and real-time data analytics for performance monitoring. The system incorporates modern web technologies and machine learning algorithms to personalize recommendations based on user preferences, making the experience more engaging and tailored. By focusing on ease of use, scalability, and smooth interaction between customers and cinema staff, Ticket Flix delivers an end-to-end solution that transforms the traditional movie booking process, reduces wait times, improves seat management, and enhances overall customer satisfaction.

2.2 LITERATURE REVIEW

Aspect-based sentiment analysis (ABSA) has emerged as a powerful tool for understanding and interpreting user sentiments with greater precision, especially in contexts where evaluating individual components or features of an entity—such as products, services, or experiences—adds significant value. Unlike traditional sentiment analysis, which only identifies the overall sentiment of a text or speech, ABSA focuses on extracting and classifying sentiments related to specific aspects of a subject, providing more granular insights into customer feedback [1]. This capability is particularly beneficial in fields like customer reviews, social media analysis, and product feedback, where understanding the sentiment towards specific features or attributes is crucial for improving products or services.

In recent years, deep learning techniques, such as neural networks and transformer-based models, have dramatically improved the performance and accuracy of ABSA systems. These models excel in extracting aspects from text and accurately classifying the sentiment towards each aspect, providing a more detailed sentiment profile [2]. However, traditional ABSA methods have primarily focused on analyzing text data, leaving voice data underutilized despite its potential. With the rise of voice-enabled technologies like virtual assistants and smart devices, integrating voice feedback into ABSA has become an important area of research. Real-time sentiment analysis of spoken language is now increasingly relevant, particularly for customer service applications and voice assistants, where understanding a user’s sentiment based on their voice is essential for improving interactions [4].

Incorporating voice feedback into ABSA presents unique challenges, particularly related to speech recognition, prosody, and emotion detection. These elements—such as tone, pitch, and rhythm—carry important cues that influence sentiment interpretation. For example, the same words can convey positive or negative sentiments depending on the speaker's tone of voice, making it crucial to integrate emotion detection with sentiment analysis to ensure accurate interpretation of the voice data [5][13]. The complexity of speech adds layers to the sentiment classification process, requiring advanced techniques to differentiate between various emotional tones and their corresponding sentiments.

To address these challenges, several hybrid approaches have been proposed that combine both voice and text inputs for a more comprehensive ABSA solution. By leveraging multimodal data—where speech and text are analyzed together—these systems are able to capture a richer context, thus improving sentiment classification accuracy [15]. Hybrid models, for example, may use text analysis to extract aspects and then integrate voice data to assess the emotional tone or sentiment behind each aspect, providing a more holistic understanding of user feedback. This approach has shown promising results in real-time sentiment analysis scenarios, such as in voice assistants, where analyzing the sentiment behind a user's spoken query based on their tone can significantly enhance the assistant's response quality and relevance [9][19].

Moreover, research has demonstrated the benefits of combining speech recognition with context-aware sentiment analysis, which allows systems to better understand and classify sentiments based on the context in which the voice feedback is given. For instance, sentiment towards a product feature may vary depending on the surrounding conversation or external context, and combining speech recognition with contextual sentiment analysis enables systems to refine their understanding and improve sentiment classification [10][20].

The integration of voice feedback into aspect-based sentiment analysis (ABSA) represents a significant advancement in capturing nuanced user sentiments. Unlike text-only methods, voice-enabled ABSA systems can interpret emotional subtleties—such as tone, pitch, and rhythm—that are critical for accurately understanding user intent and sentiment toward specific aspects of a product or service. This is particularly impactful in customer service and support scenarios, where vocal cues such as frustration or enthusiasm can profoundly affect sentiment interpretation [6][18]. For instance, a customer's voice tone when discussing a particular product feature may reveal dissatisfaction or satisfaction more distinctly than words alone. By combining voice and text inputs, voice-based ABSA not only enhances the granularity of sentiment detection but also provides insights that are more responsive to the emotional context of feedback. This capability allows businesses and service providers to identify and address specific areas of improvement more effectively, making voice-enabled ABSA a powerful tool for enhancing customer satisfaction and refining service quality in real-time applications [5][13].

2.3 PROPOSED SYSTEM

The proposed system, **Ticket Flix**, is a comprehensive online movie ticket booking platform designed to address the limitations of the existing manual system. Ticket Flix will enable users to browse movie listings, view showtimes, select preferred theaters, and book tickets in real-time from the convenience of their devices. By incorporating features such as interactive seat selection, secure online payments, and personalized movie recommendations, the system aims to provide a seamless and efficient booking experience.

For cinema operators, Ticket Flix offers advanced management tools, allowing them to automate movie scheduling, manage seat availability, and track bookings with real-time analytics. The system will also include features for tracking booking history, customer feedback, and promoting upcoming movies through notifications. By leveraging modern web technologies and machine learning, Ticket Flix will deliver a user-friendly, scalable, and secure solution, ensuring a significant improvement in both customer satisfaction and operational efficiency compared to the traditional system.

2.4 ADVANTAGES OF PROPOSED SYSTEM

- **Convenient Online Booking**: Users can book tickets from anywhere at any time, eliminating the need to visit the theater in person.
- **Real-Time Seat Selection:** The interactive seat selection feature allows users to choose available seats in real-time, enhancing the movie-going experience.
- **Secure Online Payments**: Ticket Flix integrates multiple secure payment gateways, offering users a wide range of payment options, including credit cards, debit cards, and digital wallets.
- **Personalized Recommendations**: Machine learning algorithms will provide users with personalized movie suggestions based on their viewing history and preferences.
- **Reduced Wait Times**: By allowing users to book tickets online, Ticket Flix reduces congestion at the theater and shortens wait times for in-person bookings.
- **Booking History and Management**: Users can easily access and manage their booking history, view upcoming shows, and rebook favorite movies with just a few clicks.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

Feasibility study is the test of the system proposal made to identify whether the user needs may be satisfied using the current software and hardware technologies, whether the system will be cost effective from a business point of view and whether it can be developed with the given budgetary constraints. Depending on the study, the decision is made whether to go ahead with a more detailed analysis. When a new project is proposed, it normally goes through feasibility assessment. Feasibility study is carried out to determine whether the proposed system is possible to develop with available resources and what should be the cost consideration.

The document provides the feasibility of the project that is being designed and lists various areas that were considered very carefully during the feasibility study of this project such as Technical, Economic and Operational feasibilities. The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standard.

Various feasibility studies are:

- Economic Feasibility
- Technical Feasibility
- Operational Feasibility

3.1.1 Economical Feasibility

The economic feasibility of developing an online movie ticket booking system is promising given the rising demand for digital solutions that offer convenience and efficiency. The initial investment includes development costs, infrastructure setup, and marketing expenses, but these are offset by various revenue streams and long-term operational savings. The system is expected to generate revenue through service fees on each ticket booking, advertising, and partnerships with theatres. Additionally, the automation of ticket sales can reduce the need for manual labor, leading to cost savings for theatre operators. The system's scalability ensures it can handle increasing user traffic without proportional increases in costs.

Benefits:

- o Increased Convenience:
- o Wider Reach:

- Reduced Operational Costs:
- Enhanced Customer Experience:
- Data Collection and Analysis:

3.1.2 Technical Feasibility

The technical feasibility of developing a movie ticket booking system in Python is promising due to Python's robust ecosystem and versatile frameworks. Leveraging Python's Django or Flask frameworks will enable the development of a scalable and secure backend, providing essential functionalities such as user authentication, movie listings, seat selection, and payment processing.

The frontend can be built using HTML, CSS, and JavaScript frameworks such as React or Vue.js, allowing for a responsive and interactive user interface that enhances the user experience. Python's rich set of tools for testing and deployment, combined with cloud services like AWS, Google Cloud, or Azure, ensure the application can scale efficiently to meet varying traffic demands.

3.1.3 Behavioral Feasibility

The behavioral feasibility of TicketFlix appears favorable, as the system is designed to align with current user trends toward digital solutions. By prioritizing user acceptance, providing adequate training, and engaging with the community, TicketFlix can effectively overcome potential resistance and foster a positive user experience. The successful adoption of the system will ultimately lead to enhanced customer satisfaction and operational efficiency for theaters.

3.1.4FEASIBILITY STUDY QUESTIONNAIRE

1.What features would you like to see added to improve the booking experience?

Firstly, an intuitive seat selection interface with real-time updates on seat availability would streamline the booking process. Secondly, personalized recommendations based on past preferences or genres of interest would enrich the user experience.

2.Have you encountered any issues during the booking process? If yes, please describe.

Yes, there are some instances during the booking process. One time, the website experienced a significant lag during peak hours, causing delays in selecting seats and completing the payment.

3.Have you ever experienced any payment-related issues (e.g., payment failure, unauthorized charges)?

Yes, have experienced payment-related issues while booking tickets. On one occasion, the payment failed to process despite having sufficient funds in someone’s account, and had to retry multiple times before it finally went through.

4.Would you prefer options for ticket cancellation or rescheduling? Why or why not?

Yes, I would prefer options for ticket cancellation or rescheduling. These features provide flexibility in case of unexpected changes in plans, such as personal emergencies or scheduling conflicts.

5.What improvements would you suggest for enhancing customer support services?

- 24/7 Support Availability:
- Live Chat and Instant Messaging:
- Comprehensive FAQ and Help Center:
- Multichannel Support:
- Quick Response Time:

6.What additional features would encourage you to use the platform more frequently?

Additional features that would encourage me to use the platform more frequently include personalized recommendations based on my movie preferences, exclusive deals or discounts for frequent users, and a loyalty program with rewards for booking tickets.

7.Have you experienced any downtime or performance issues while using the platform?

Users are encouraged to briefly describe any instances where they encountered issues such as website downtime, slow loading times, or errors during the booking process.

8.What do you think are the platform's strongest features?

- Ease of Use:
- Real-Time Updates:
- Secure Payment Processing:
- Customer Support:
- Personalization:
- Notification System:

9.How customers are satisfied with the variety of food and drink options available through the booking system?

Yes, all of the customers are very much satisfied with the food and drinks offered from counter.

10.Is there any other feedback or suggestions you would like to provide regarding the movie ticket booking system?

Users might offer insights on areas for improvement, new features they would like to see, or specific issues they've encountered that haven't been covered in previous questions. It also demonstrates a commitment to continuous improvement based on user feedback.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

Processor - Intel Core i5 or higher

RAM - 8GB or higher

Hard disk - 500GB solid-state drive (SSD) or higher

3.2.2 Software Specification

Front End - HTML, CSS, JavaScript,

Bootstrap, JQuery, AJAX

Back End - Django, SQLite for database

Client on PC -Windows 10

Technologies used - HTML, CSS, JavaScript, Django, SQLite, AJAX, NLP

3.3 SOFTWARE DESCRIPTION

3.3.1 Django

Django is a high-level Python web framework that enables rapid development of secure and scalable web applications. It follows the Model-View-Template (MVT) architectural pattern and provides built-in features for authentication, database handling, and URL routing, making development efficient. Django promotes reusability, less code repetition, and emphasizes the principle of "Don't Repeat Yourself" (DRY). It’s widely used for creating robust web applications and supports integration with various databases.

3.3.2 SQLite

SQLite is a lightweight, serverless, and self-contained relational database management system (RDBMS) that is widely used for its simplicity and efficiency. It is an embedded database that allows developers to integrate database functionality directly into their applications without the need for a separate server process. SQLite is designed to be highly portable, with a single file database structure that makes it easy to deploy and share. It supports most of the SQL standards, including transactions, subqueries, and triggers, providing robust data management capabilities. Due to its minimal setup and low overhead, SQLite is commonly used in mobile applications, web browsers, and small to medium-sized projects where a full-fledged database server may be unnecessary. Its simplicity, reliability, and cross-platform compatibility make it an excellent choice for developers looking for an effective database solution with minimal configuration requirements.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

The system design phase is a critical stage in the development of the Ticket Flix online movie ticket booking platform, as it outlines the architecture, components, modules, interfaces, and data necessary for the system's implementation. This phase transforms the project requirements identified during the feasibility study into a blueprint for development, ensuring that the system is both functional and user-friendly. By employing a modular design approach, Ticket Flix will facilitate seamless interactions between users and theater operators while incorporating essential features such as user registration, movie browsing, seat selection, and payment processing. The design will emphasize scalability, security, and performance to accommodate varying user loads and ensure a smooth booking experience. Additionally, user interface design will focus on creating an intuitive and visually appealing platform that enhances user engagement. Overall, the system design aims to deliver a robust and efficient online ticketing solution that meets the needs of both customers and theater operators, ultimately enhancing the movie-going experience.

4.2 UML DIAGRAM

Unified Modeling Language (UML) diagrams are essential tools in the design and development of the Ticket Flix online movie ticket booking system, as they provide a visual representation of the system's architecture, processes, and interactions between different components. UML diagrams serve various purposes, including illustrating system requirements, modeling workflows, and detailing object interactions. Key UML diagrams for Ticket Flix include use case diagrams, which depict the functional requirements from the user's perspective, highlighting the interactions between users and the system; class diagrams, which represent the structure of the system by showing classes, attributes, and relationships; and sequence diagrams, which illustrate the flow of messages between objects over time during critical processes, such as booking tickets or processing payments. By utilizing these diagrams, the development team can ensure a comprehensive understanding of the system's functionalities and workflows, facilitating better communication among stakeholders and providing a clear roadmap for implementation. Ultimately, the UML diagrams will help ensure that Ticket Flix is designed effectively to meet user needs and operational requirements.

4.2.1 USE CASE DIAGRAM

A use case diagram visually represents the system's functional requirements, showing interactions between users (actors) and the system to achieve specific goals (use cases). It identifies the system's boundaries, actors, and use cases, highlighting what the system must accomplish. This diagram is commonly used in the early stages of development to clarify user needs, define system functionality, and ensure the system meets requirements.

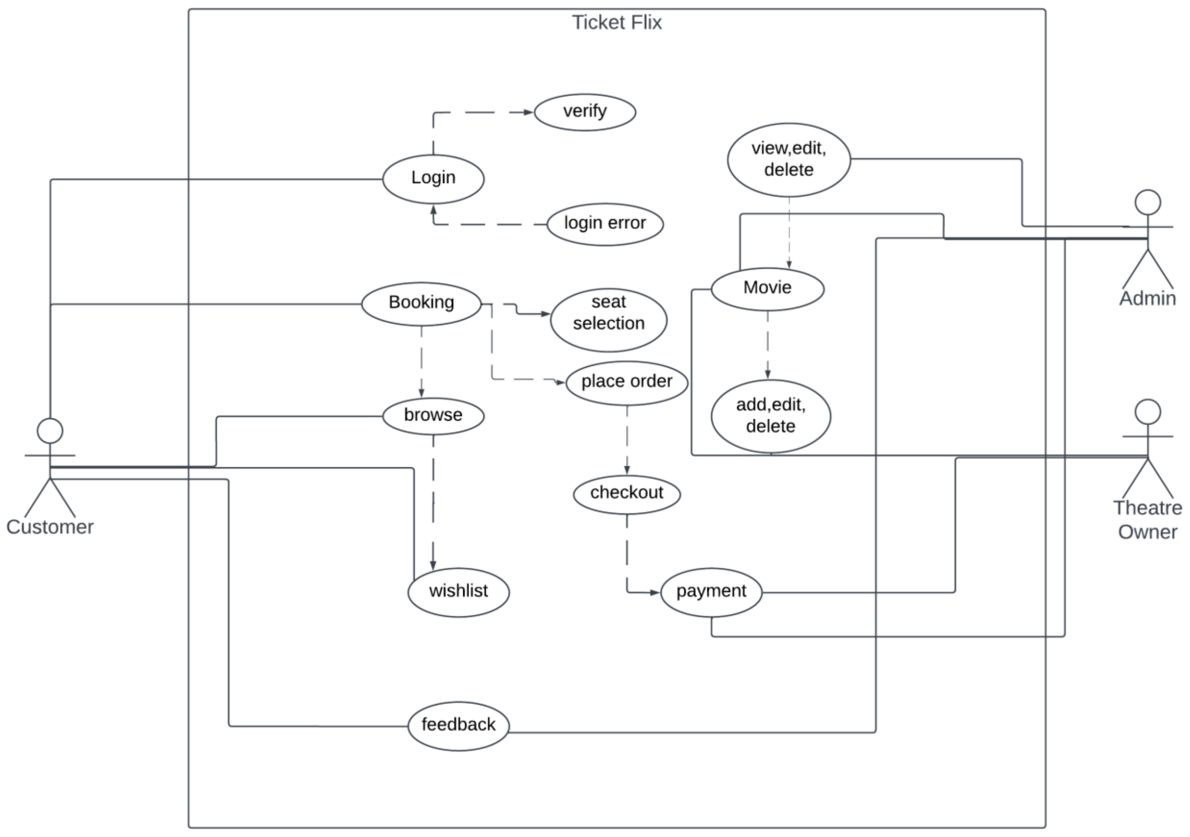


Figure 1: Use Case Diagram

4.2.2 SEQUENCE DIAGRAM

A sequence diagram is an interaction diagram that shows how objects or components in a system interact over time. It represents the sequence of events in a scenario, using vertical lifelines for objects and horizontal arrows for messages exchanged between them. These diagrams help visualize the flow of control, making it easier to design, analyze, and communicate complex systems. Sequence diagrams are commonly used in software development to model system behavior and document interactions between components.

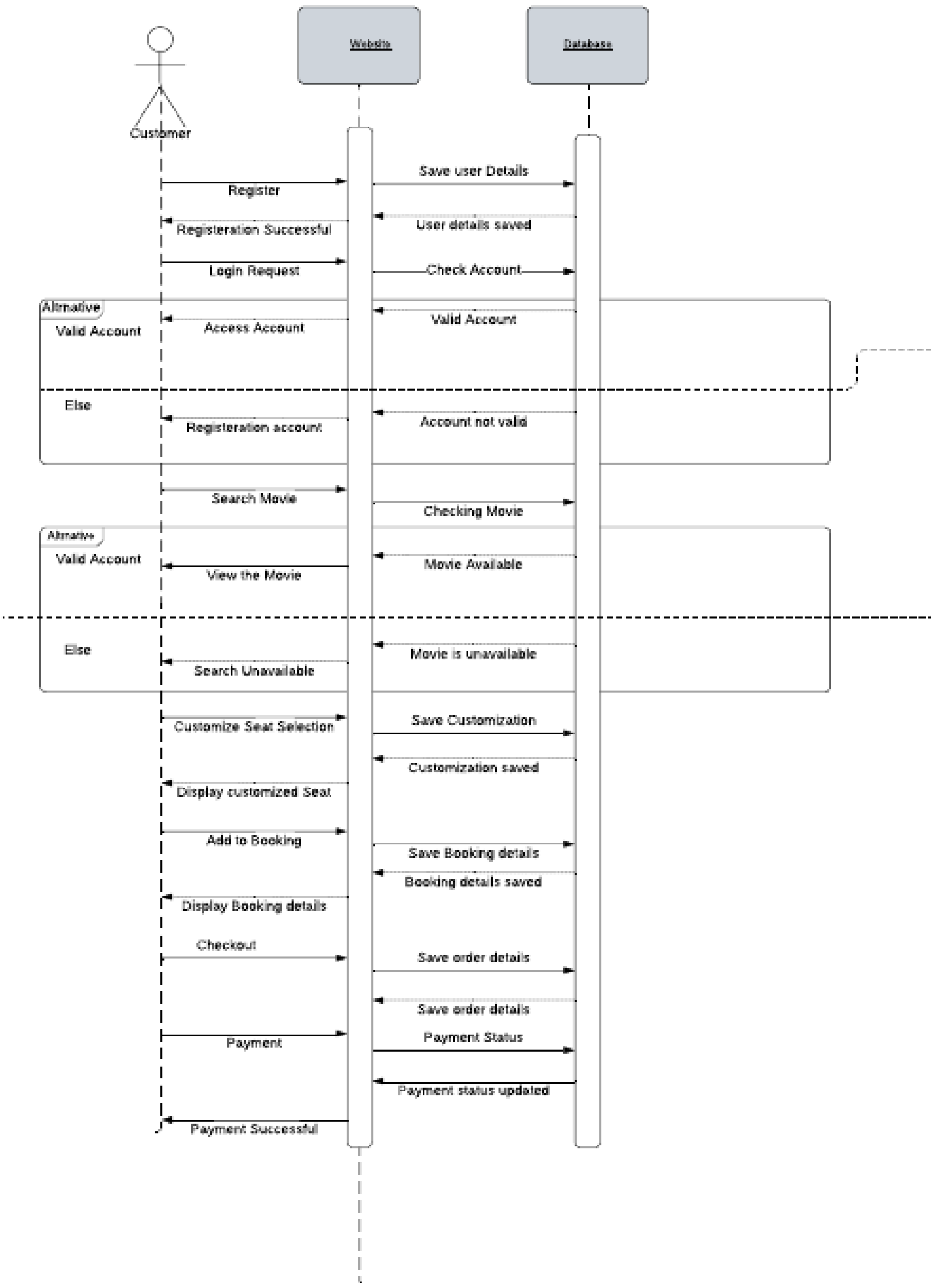


Figure 2: Sequence Diagram

4.2.3 Activity Diagram

An activity diagram is a graphical representation of workflows and processes that shows the flow of control or object flow. It visually depicts the steps, actions, and decision points involved in a process or system. It is used to model, visualize, and analyze the behavior of a system, and to capture the sequence of activities or steps that occur during the execution of a use case or business process. The activity diagram is an important tool in software development, business process modeling, and system analysis, as it helps stakeholders to understand complex processes and workflows in a clear and concise manner.

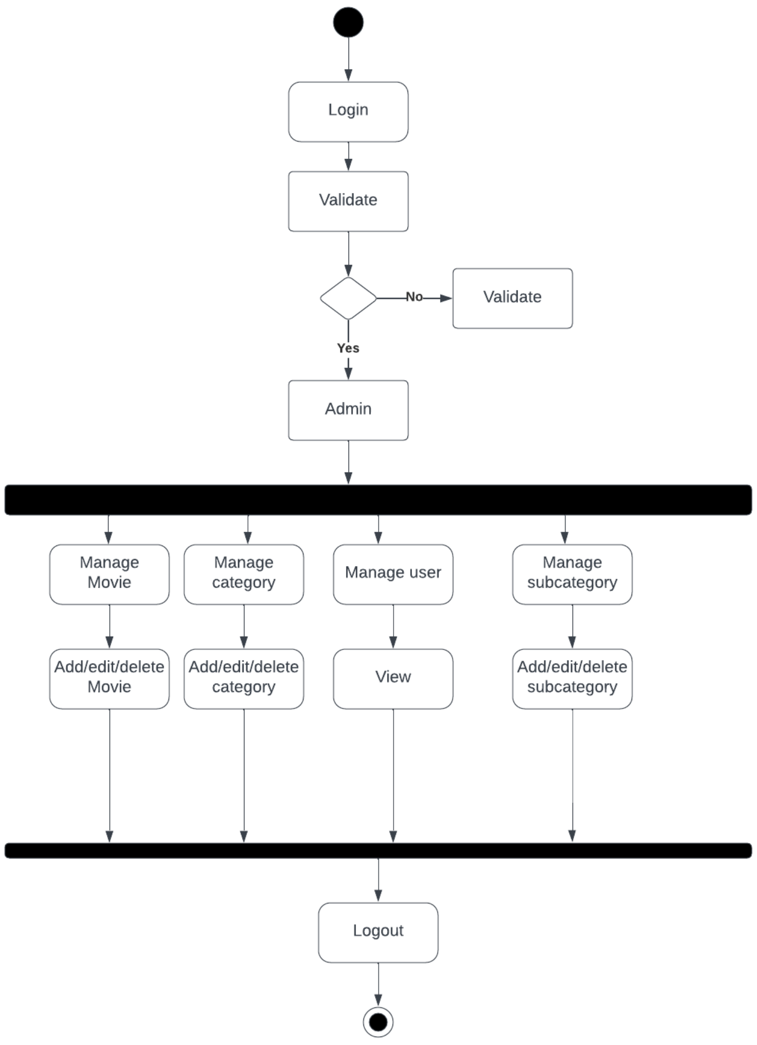


Figure 3: Activity Diagram Admin

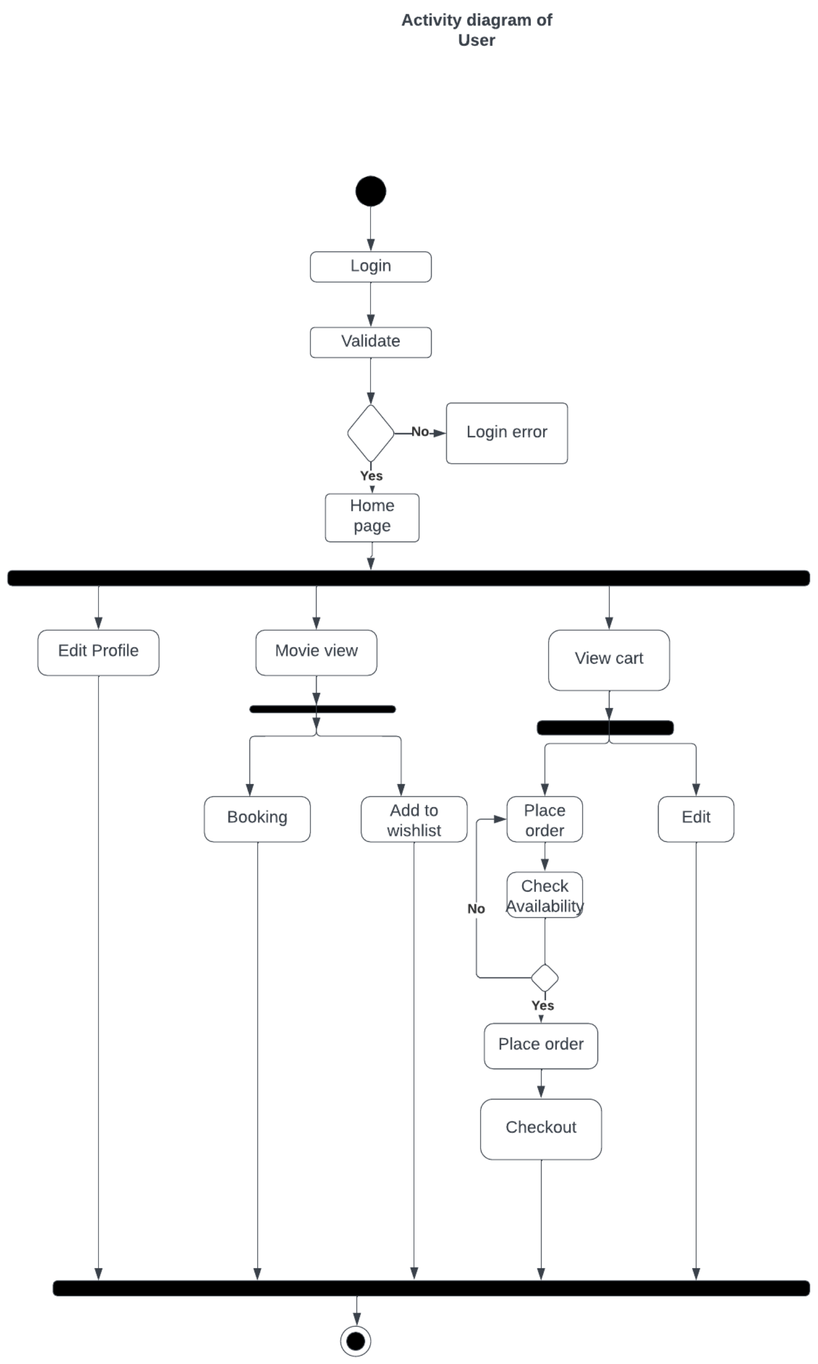


Figure 4 : Activity Diagram of User

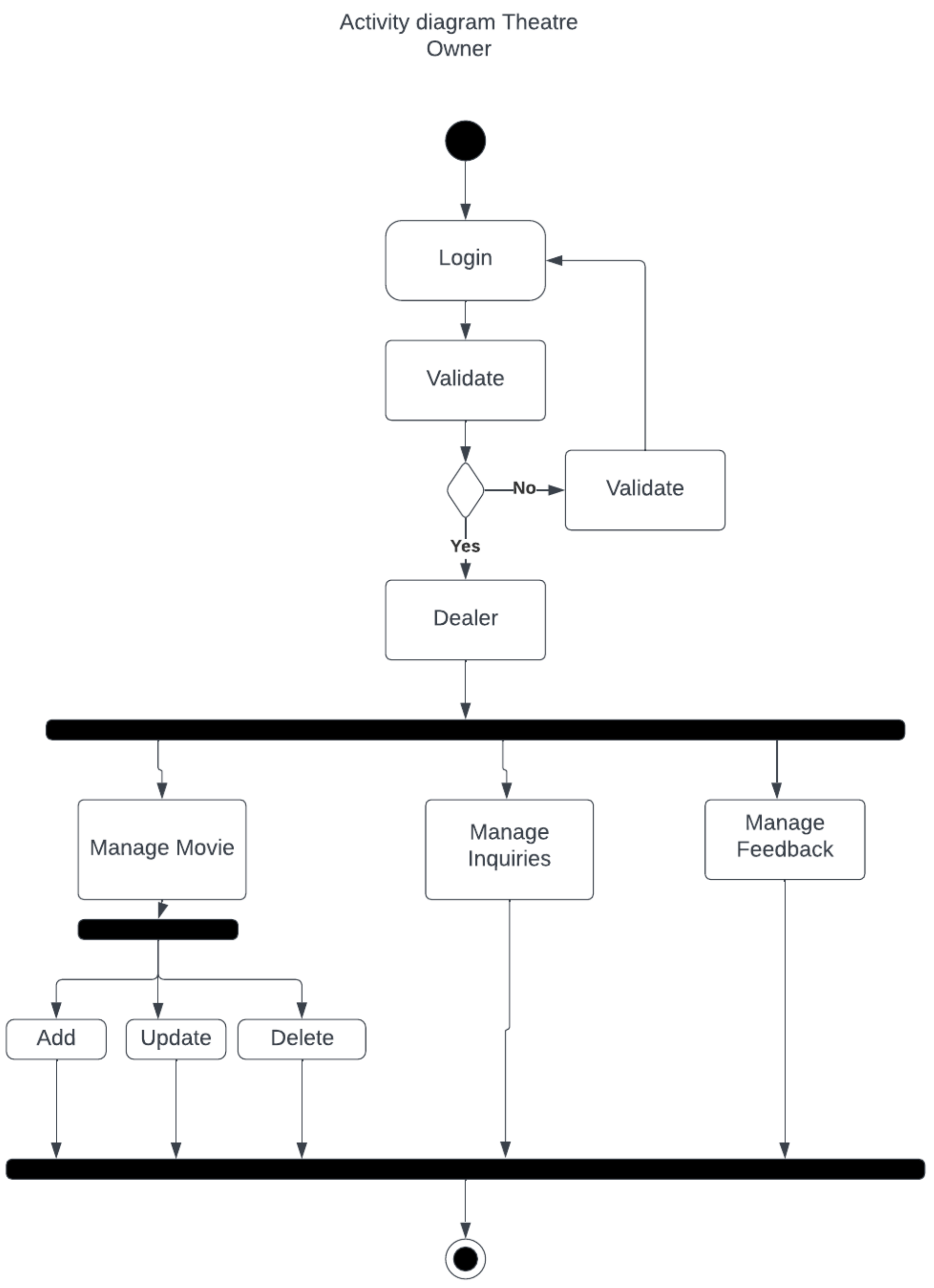


Figure 5: Activity Diagram of Theatre Owner

4.2.4 Class Diagram

A class diagram is a type of diagram used in object-oriented programming to illustrate the structure of a system by showing the classes and their relationships to each other. It provides a visual representation of the system's classes, their attributes, and the methods or operations that can be performed on them. The class diagram allows developers to design and plan the system's architecture, define the relationships between classes, and identify the attributes and methods that will be used in the system. It is an essential tool for software developers to communicate and collaborate with stakeholders and other team members involved in the project.

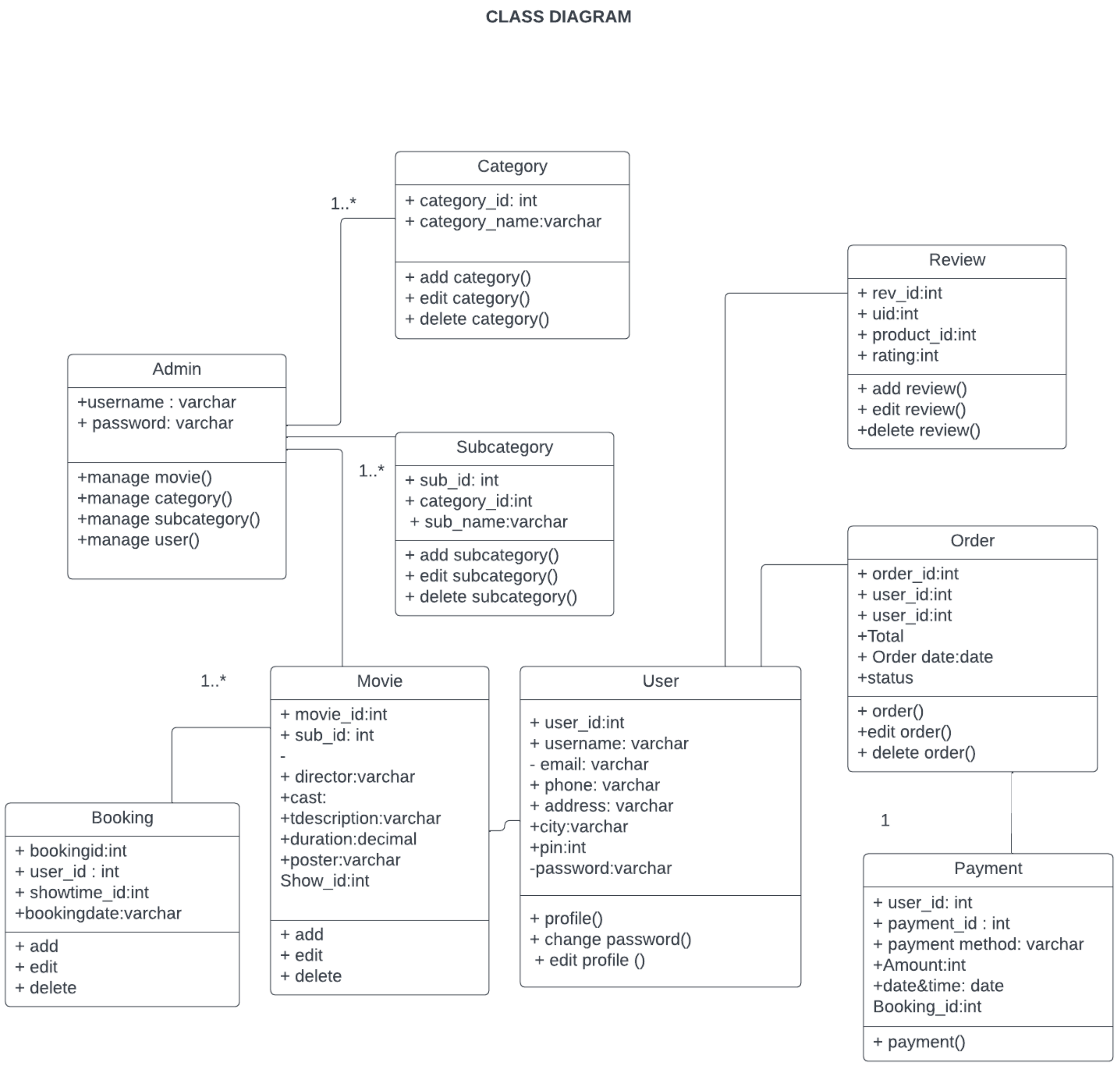


Figure 6: Class Diagram

4.2.5 Object Diagram

An object diagram is a type of UML diagram that represents a system or a part of a system by depicting objects and their relationships with one another. It provides a static view of the system, showing the objects and their attributes, as well as the links or associations between them. An object diagram is useful for visualizing complex systems and understanding the relationships between objects. It can also be used to validate and verify the design of a system or to communicate it to stakeholders. Overall, an object diagram is an effective tool for representing the structure and behavior of a system in a clear and concise manner.

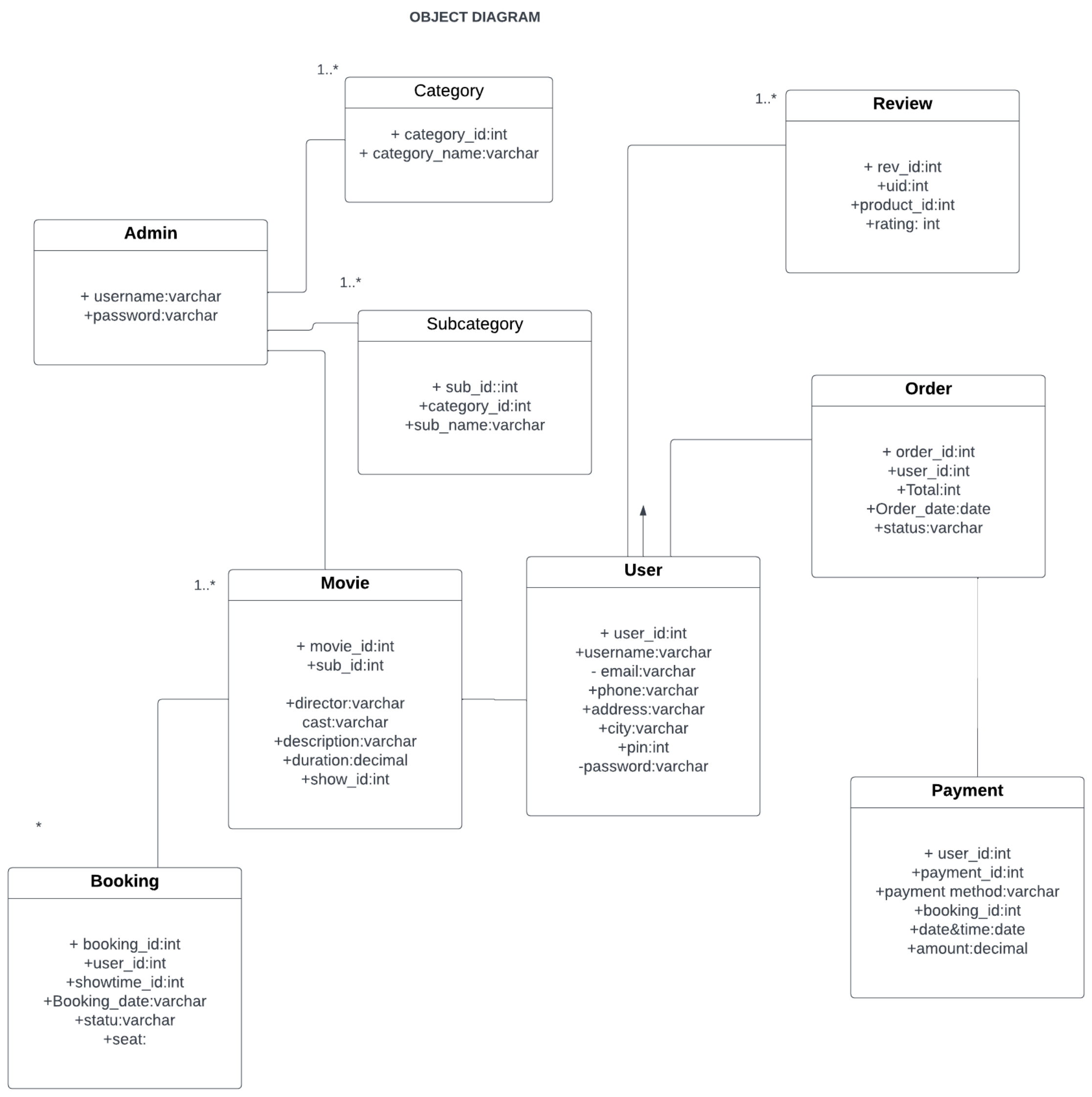


Figure 7: Object Diagram

4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Login

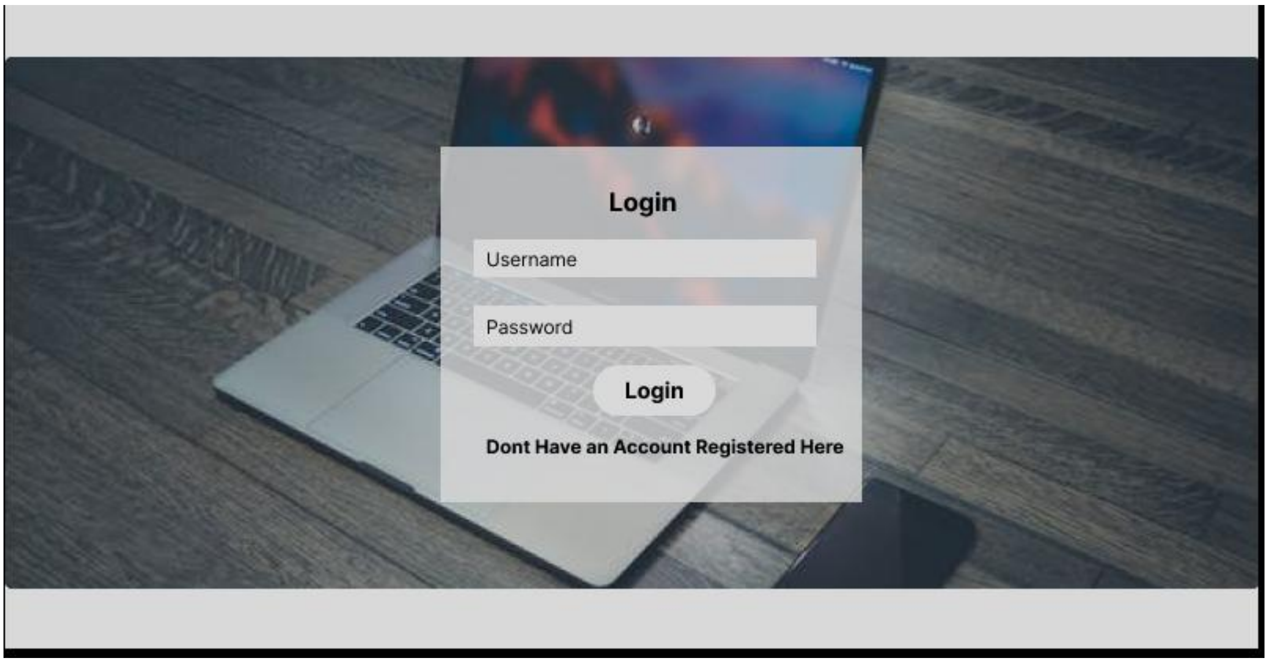


Figure 8: Login Figma

Form Name: Register

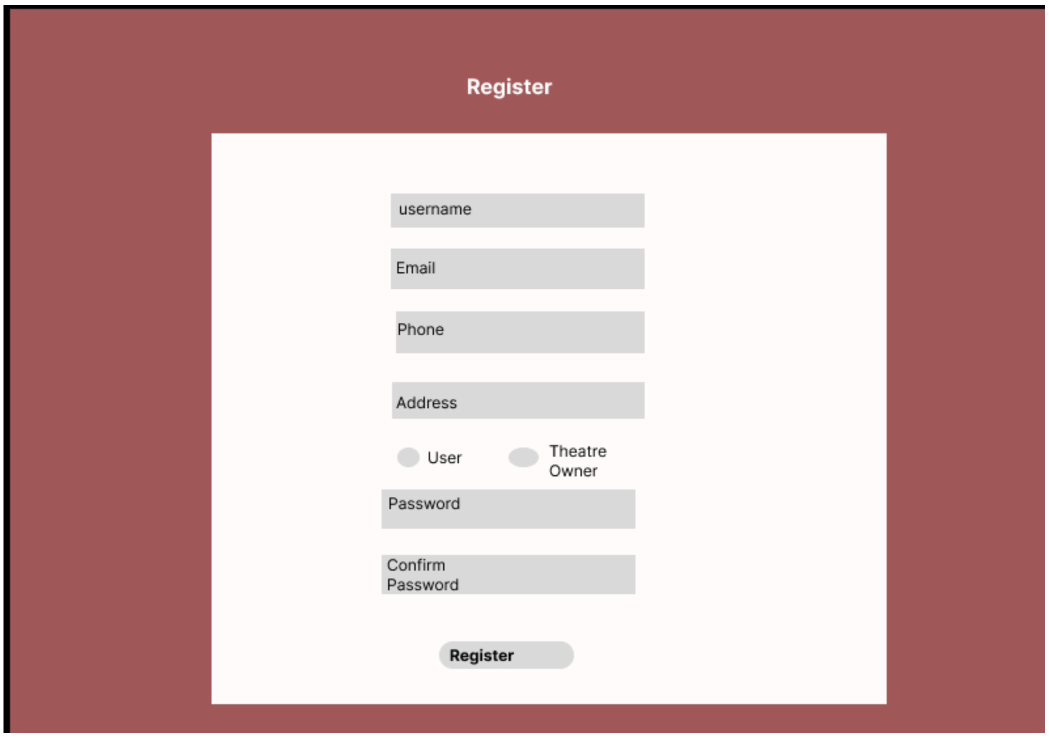


Figure 9: Register Figma

4.4 DATABASE DESIGN

4.4.1 Relational Database Management System (RDBMS)

A Relational Database Management System (RDBMS) is a type of software used to manage and store data in a structured way. It relies on a relational model to organize data into tables, each of which contains related data that can be queried and manipulated using SQL (Structured Query Language). RDBMSs are used to manage large volumes of data and are commonly used in businesses, organizations, and governments to store and access important information.

One of the main advantages of using an RDBMS is its ability to ensure data consistency and accuracy. By enforcing constraints and relationships between tables, an RDBMS can prevent data inconsistencies and errors that can arise when data is stored in a non-standardized way.

4.4.2 Normalization

Normalization is the process of organizing data in a relational database in order to minimize redundancy and dependency. It involves breaking down a larger table into smaller tables and establishing relationships between them. Normalization is important because it helps to reduce data redundancy, which can lead to inconsistencies and errors in the database.

There are several levels of normalization, referred to as normal forms. The most commonly used are first normal form (1NF), second normal form (2NF), and third normal form (3NF). Each level of normalization has specific requirements that must be met in order to achieve that level. Normalization helps to ensure that data is stored in a consistent and logical manner, which makes it easier to retrieve and manipulate the data. It also reduces the amount of storage space required and improves database performance by reducing the amount of data that needs to be processed. However, over-normalization can also lead to performance issues, so it is important to strike a balance between normalization and efficient data retrieval.

First Normal Form

First Normal Form (1NF) is a fundamental concept in relational database design that establishes a set of rules for organizing data in a tabular format. According to 1NF, a table is in its first normal form if it contains no repeating groups or arrays. This means that every field in a table must contain atomic (indivisible) values, and there should be no repeating groups of columns.

Achieving first normal form is important because it allows for data to be efficiently and accurately queried, updated, and maintained. It also eliminates data redundancy and inconsistencies, which can lead to errors and inefficiencies in the system. To ensure that a table is in 1NF, one should consider decomposing it into smaller tables that satisfy the atomicity and uniqueness requirements, and establishing relationships between those tables through primary and foreign keys. This process of normalization is essential for ensuring the integrity and efficiency of the database schema.

Second Normal Form

Second Normal Form (2NF) is a concept in relational database design that builds upon the principles of First Normal Form (1NF) by eliminating data redundancy and dependency issues in the database. According to 2NF, a table is in its second normal form if it is in 1NF and every non-key attribute is functionally dependent on the table's primary key. In other words, 2NF ensures that each table column is uniquely determined by the primary key, and there is no partial dependency between columns.

To achieve 2NF, one must identify and separate the columns that depend on a subset of the primary key and group them into a new table with a new primary key. This process is known as decomposition, and it helps eliminate data redundancy and inconsistency issues by breaking down tables into smaller, more manageable units. By decomposing the table into smaller units, it is possible to achieve higher levels of normalization, leading to more efficient and reliable database operations.

Third Normal Form

Third Normal Form (3NF) is another concept in relational database design that builds upon the principles of First Normal Form (1NF) and Second Normal Form (2NF) by further eliminating data redundancy and dependency issues. According to 3NF, a table is in its third normal form if it is in 2NF and there is no transitive dependency between non-key attributes. This means that every non-key attribute must be dependent only on the primary key, and not on any other non-key attribute.

To achieve 3NF, one must identify and remove any transitive dependencies that exist in the table. This can be achieved by decomposing the table into smaller units based on the functional dependencies of the attributes, and establishing relationships between these smaller units through foreign keys. By removing transitive dependencies, it is possible to achieve higher levels of normalization, leading to better data management and more efficient database operations.

Boyce Codd Normal Form

Boyce-Codd Normal Form (BCNF) is a database normalization technique that helps to ensure that the database tables are well-structured and free from anomalies. The BCNF is a higher level of normalization compared to other normal forms such as first normal form (1NF), second normal form (2NF), and third normal form (3NF). BCNF is achieved by decomposing tables that contain functional dependencies that violate the Boyce-Codd normal form.

A table is considered to be in BCNF if it satisfies the following criteria:

- Every determinant in the table is a candidate key, which means that there are no non-trivial dependencies between attributes.
- The table must not contain any non-trivial functional dependencies between attributes that are not part of any candidate key.

Fourth Normal Form

Fourth Normal Form (4NF) is a further level of database normalization that addresses multi-valued dependencies between attributes. It is an extension of the third normal form (3NF) and aims to eliminate redundancy in multi-valued dependencies. A multi-valued dependency occurs when there is a relationship between non-key attributes that is not fully dependent on the primary key. In other words, there are non-key attributes that depend on a combination of other non-key attributes, rather than on the primary key alone.

To achieve 4NF, we decompose tables that contain multi-valued dependencies into smaller tables, each of which contains a single multi-valued dependency. In this way, we can eliminate redundancies and ensure that the database is well-structured and free from anomalies.It's important to note that 4NF is not always necessary for every database, and it may not always be feasible to decompose tables into smaller tables due to performance or other practical considerations. However, in cases where multi-valued dependencies exist, 4NF can be a useful technique to improve data quality and consistency.

Fifth Normal Form

Fifth normal form Fifth Normal Form (5NF), or Project-Join Normal Form, is the highest level of normalization that addresses complex relationships between entities, especially in many-to-many relationships. It requires decomposing the database into smaller tables, each with its own primary key, focusing on single attributes or relationships to be "fact-oriented." Achieving 5NF involves using join dependency techniques, ensuring that tables can be reconstructed through join operations. This reduces redundancy and improves data consistency while enhancing flexibility. However, achieving 5NF can be complex and may not be practical for all databases.

4.4.3 Sanitization

Sanitization in the context of ensuring that no residual data can be recovered even after a comprehensive forensic investigation is known as secure data wiping or secure data erasure. This process involves permanently deleting data from a storage device, ensuring that it cannot be recovered even through advanced forensic techniques. To ensure that data is securely wiped, various techniques and tools can be used, such as overwriting the data with random patterns multiple times or physically destroying the storage device. The exact method used depends on the sensitivity of the data and the level of security required.

Secure data wiping is essential in cases where sensitive or confidential data needs to be permanently erased, such as when disposing of old hardware, transferring ownership of a device, or closing down a business. Failure to properly sanitize data can result in the unintentional exposure of sensitive information, which can have serious legal, financial, or reputational consequences.

4.4.4 Indexing

In SQL, indexing is the process of creating data structures that allow for faster retrieval of data from a database. Indexes are essentially lists of keys that point to specific locations within a table, making it possible to quickly locate and retrieve data that matches specific search criteria.

Creating indexes in SQL can significantly improve query performance, especially when working with large databases. By creating indexes on frequently accessed columns, such as primary keys or foreign keys, SQL can quickly locate the data that matches specific search criteria, without having to scan the entire table.

It's important to note that indexing can also have a negative impact on performance if done incorrectly. Over-indexing, or creating too many indexes, can slow down the insertion and updating of data, as well as increase the size of the database.

4.5 TABLE DESIGN

1.tbl_user

Primary key: **id**

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	user_id	Int	Primary Key	Primary Key of tbl_login
3	username	Varchar (30)	Not Null	To store name of users
4	email	varchar	Not Null	To store email of users
5	password	Varchar (30)	Not Null	To store password of users
6	Phone number	Int		To store phone number of Users

2.tbl_register

Primary Key: user_id

Foreign Key: user_id references table tbl_login

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	user id	Int (11)	Primary Key	Primary Key of tbl_login
3	name	varchar (30)	Not Null	To store the name of the users
4	phone	varchar (10)	Not Null	To store the mobile number of the users
5	address	varchar (30)	Not Null	To store the address of the users
6	status	Int (3)	Not Null	To store the status of the users who are active

3.tbl_category

Primary Key: id

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (11)	Primary Key	Primary Key of table
2	name	varchar (50)	Not Null	To store the category name of the Movie
3	description	Varchar (50)	Not Null	To store the description of the movie

4.tbl_movie

Primary Key: id

Foreign Key: user_id references table tbl_login

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field

1	id	Int (Auto)	Primary Key	Primary Key of table
2	Movie id	Int (11)	Primary key	Primary key of tbl_movie
3	User id	Int (11)	Foreign key	Primary key of tbl_login
4	title	Int (11)	Not null	To store title of the movie
5	director	Varchar (100)	Not null	To store the director of the movie
6	cast	Varchar (100)	Not null	To store the cast of the movie
7	description	Varchar (50)	Not null	To store the description of the movie
8	duration	Int (50)	Not null	To store the duration of the movie
9	poster	Varchar (50)	Not null	To store the poster of the movie
10	price	Int (50)	Not null	To store the price of the movie
11	Show id	Varchar (50)	Not null	To store the showtime

5.tbl_theatre

Primary Key: id

Foreign Key: owner id references table tbl_theatreowner

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	Theatre id	Int (11)	Primary Key	Primary Key of tbl_theatre
3	Owner id	Int (11)	Foreign key	To store the Owner
4	name	Int (100)	Not null	To store the name of Theatre
5	location	varchar	Not null	To store the location of Theatre
6	Total seats	Int	Not null	To store the Total seats of Theatre

6.tbl_showtime

Primary Key: id

Foreign Key: movie_id references table tbl_movie

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (11)	Primary Key	Primary Key of table
2	movie	varchar (50)	Foreign key	To store the movie of showtime

3	theatre	varchar (50)	Foreign key	To store the theatre of showtime
4	Start_date	date	Not null	To store the start date of the showtime
5	End_date	date	Not null	To store the end date of the showtime

7.tbl_booking

Primary Key: id

Foreign Key: movieid references table tbl_movie

Theatre_id references table tbl_theatre

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	user	Int (11)	Primary Key	Primary Key of tbl_booking
3	Movie id	Int (11)	Foreign key	Primary key of the tbl_login
4	Theatre id	Int (100)	Foreign Key	To store the showtime
5	Showtime id	Varchar (50)	Not Null	To store the booking date
6	selected_seats	Int	Not Null	To store the status
7	booking_date	date	Not Null	To store the seat
8	total_amount	Int	Not Null	To store total amount
9	status	Varchar	Not Null	To store the status
10	payment	Int	Not Null	To store the payment
11	Refund_status	Varchar	Not Null	To store the Refund status

8.tbl_seating

Primary Key: id

Foreign Key: row references table tbl_seating

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	Theatre id	Int (11)		
3	row	Int (11)	Foreign key	To store the showtime

4	seat_number	Int (11)	Foreign key	To store the seat number
5	Is_booked	Int (11)	Foreign key	To store the status

9.tbl_rating

Primary Key: id

Foreign Key: user_id references table tbl_login, movie_id references table tbl_movie

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	rating id	Int (11)	Primary Key	Primary Key of tbl_review
3	User id	Int (11)	Foreign key	Primary key of the tbl_login
4	Movie id	varchar (50)	Foreign key	To store the movie
5	score	Varchar (50)	Not Null	To store the rating

10.tbl_payment

Primary Key: id

Foreign Key: user_id references table tbl_login

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	Payment id	Int (11)	Primary Key	Primary Key of tbl_payment
3	User id	Int (11)	Foreign key	Primary key of the tbl_login
4	Booking id	varchar (50)	Not Null	To store the Booking
5	amount	Varchar (50)	Not Null	To store the amount
6	Payment date	date	Not Null	To store the payment date
7	Payment method	Int	Not Null	To store the payment method

11.tbl_feedback

Primary Key: id

Foreign Key: feedback_id references table tbl_feedback

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	user	Int (11)	Primary Key	Primary Key of tbl_payment
3	booking	Int (11)	Foreign key	Primary key of the tbl_login
4	Feedback_text	varchar (50)	Not Null	To store the Booking
5	sentiment	Varchar (50)	Not Null	To store the amount
6	Created_at	varchar	Not Null	To store the payment date
7	Payment method	Int	Not Null	To store the payment method

12.tbl_sentiment

Primary Key: feedback

Foreign Key: aspect references table tbl_sentiment

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	feedback	Int (11)	Primary Key	Primary Key of tbl_payment
3	aspect	Int (11)	Foreign key	Primary key of the tbl_login
4	sentiment	varchar (50)	Not Null	To store the Booking
5	confidence	Varchar (50)	Not Null	To store the amount

13.tbl_giftvoucher

Primary Key: code

Foreign Key: discount_value references table tbl_giftvoucher

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	code	Int (11)	Primary Key	Primary Key of tbl_payment
3	Discount_value	Int (11)	Not Null	To store the discount value
4	Expiration_date	date	Not Null	To store the expiry date

5	Is_active	Varchar (50)	Not Null	To store the active
6	user	Varchar	Not Null	To store the user

14.tbl_voucherusage

Primary Key: voucher

Foreign Key: user_id references table tbl_login

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary key of table
1	voucher	Int (11)	Primary Key	Primary Key of tbl_voucherusage
2	user	Int (11)	Foreign key	Foreign key of the tbl_login
3	Used_at	varchar (50)	Not Null	To store the usage

15.tbl_wishlist

Primary Key: id

Foreign Key: movie references table tbl_movie

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary key of table
2	user	Int (11)	Primary Key	Primary Key of tbl_login
3	movie	Int (11)	Foreign key	Primary key of the tbl_movie

16.tbl_notification

Primary Key: id

Foreign Key: message_type references table tbl_notification

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	meassge	Int (11)	Primary Key	Primary Key of tbl_payment
3	Message_type	Int (11)	Foreign key	Primary key of the tbl_login

4	Created_at			

17.tbl_profile

Primary Key: user_id

Foreign Key: phone references table tbl_login

No:	Fieldname	Datatype (Size)	Key Constraints	Description of the Field
1	id	Int (Auto)	Primary Key	Primary Key of table
2	user	Int (11)	Primary Key	Primary Key of tbl_payment
3	phone	Int (11)	Foreign key	Primary key of the tbl_login
4	address	Varchar	Not Null	To store the address
5	role	varchar	Not Null	To store the role

CHAPTER 5
SYSTEM TESTING

5.1 INTRODUCTION

System testing is an essential aspect of the software development process that ensures that the entire system or application meets the specified requirements and works as intended. In the project report, the section on system testing should provide an in-depth description of the testing process, including the scope of testing, the testing methodology used, and the tools employed.

It should also cover the different types of system testing performed, such as functional testing, performance testing, security testing, usability testing, and compatibility testing, along with the test scenarios and test cases used. Additionally, the report should include a summary of the test results, highlighting any issues or defects detected and how they were resolved.

5.2 TEST PLAN

A test plan in system testing typically involves testing the entire software system as a whole, including all its features and functions. The objective of system testing is to verify that the software meets its specified requirements and is functioning as expected in its intended environment.

The test plan typically includes details such as the testing environment, the test strategy, the test objectives, the test schedule, and the test team responsibilities. It may also include the testing techniques, tools, and metrics to be used. The test plan should be reviewed and approved by all stakeholders, including the project manager, development team, and testing team, to ensure that everyone is on the same page regarding the testing approach.

The test plan should also provide information on the different levels of testing. These typically include:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

5.2.1 Unit Testing

Unit testing is a type of testing that is focused on verifying the functionality of individual code units or modules. A well-designed test plan for unit testing should outline the specific objectives of unit testing, the approach to be taken, and the resources that will be required to carry out the testing. Unit testing is typically carried out by the development team and is an essential part of the software development lifecycle.

Unit testing is an important part of the overall testing process, as it can help to identify defects early in the software development lifecycle when they are easier and less expensive to fix. By creating a detailed test plan for unit testing, the development team can ensure that all code units have been thoroughly tested and that the software meets the desired quality standards. This can ultimately help to improve the reliability, performance, and maintainability of the software.

5.2.2 Integration Testing

Integration testing is a type of testing that focuses on testing the interaction between different modules or components of the software system to ensure that they are working together as expected. In system testing, integration testing is typically performed after unit testing, and it is focused on testing the entire system as a whole. The goal of integration testing is to detect and resolve any issues that arise from the interaction between different modules before the system is released to users.

Integration testing can help to identify defects that may not have been caught during unit testing, as it focuses on the interaction between modules rather than on individual code units. By testing the software system as a whole, integration testing can help to ensure that the system meets the desired quality standards, performs as expected, and meets the needs of its users.

5.2.3 Validation Testing or System Testing

Validation testing, also known as system testing, is a critical part of software testing that is conducted to evaluate the entire system or application under test. The goal of this type of testing is to validate that the software system meets its intended requirements and works as expected.

Validation testing involves a series of tests designed to check the software system's functionality, reliability, usability, performance, and security. It is typically performed after integration testing and prior to user acceptance testing. The objective of system testing is to ensure that the software system meets the customer's expectations and requirements before it is released to production.

5.2.4 Output Testing or User Acceptance Testing

Output testing, also known as user acceptance testing (UAT), is a type of testing that focuses on ensuring that the software system meets the needs and expectations of the end-users. The purpose of user acceptance testing is to validate that the software system is ready for release and to ensure that it meets all the functional and non-functional requirements specified by the customer.

User acceptance testing is typically performed by end-users, business stakeholders, or subject matter experts who validate the system's functionality, usability, and overall user experience. The test cases for UAT are designed to simulate real-world scenarios and to verify that the system behaves as expected in different use cases.

5.2.5 Automation Testing

Automation testing is the process of using software tools to execute pre-scripted tests on a software application or system. The goal of automation testing is to improve the efficiency and accuracy of the testing process while reducing the time and cost involved in manual testing. Automation testing can be used for functional testing, regression testing, performance testing, and other types of software testing.

One of the main benefits of automation testing is that it can help to reduce the time and effort required for repetitive testing tasks. Automated tests can be executed more quickly and consistently than manual tests, and they can be run repeatedly without the need for human intervention. Automation testing can also help to identify defects and issues more quickly and accurately than manual testing, as it can perform a large number of tests in a short period of time.

5.2.6 Selenium Testing

Selenium is an open-source automation testing tool used to automate web browsers. It supports a wide range of programming languages, including Java, Python, and Ruby, and can be used to test web applications on different platforms and browsers. Selenium is widely used for functional testing, regression testing, and other types of testing, and it can help to improve the efficiency and accuracy of the testing process.

One of the key features of Selenium is its ability to simulate user interactions with web applications. Selenium can automate actions such as clicking buttons, filling out forms, and navigating through web pages, allowing testers to validate the functionality and usability of web applications. Selenium can also be used to test the responsiveness and performance of web applications by simulating multiple users accessing the application simultaneously.

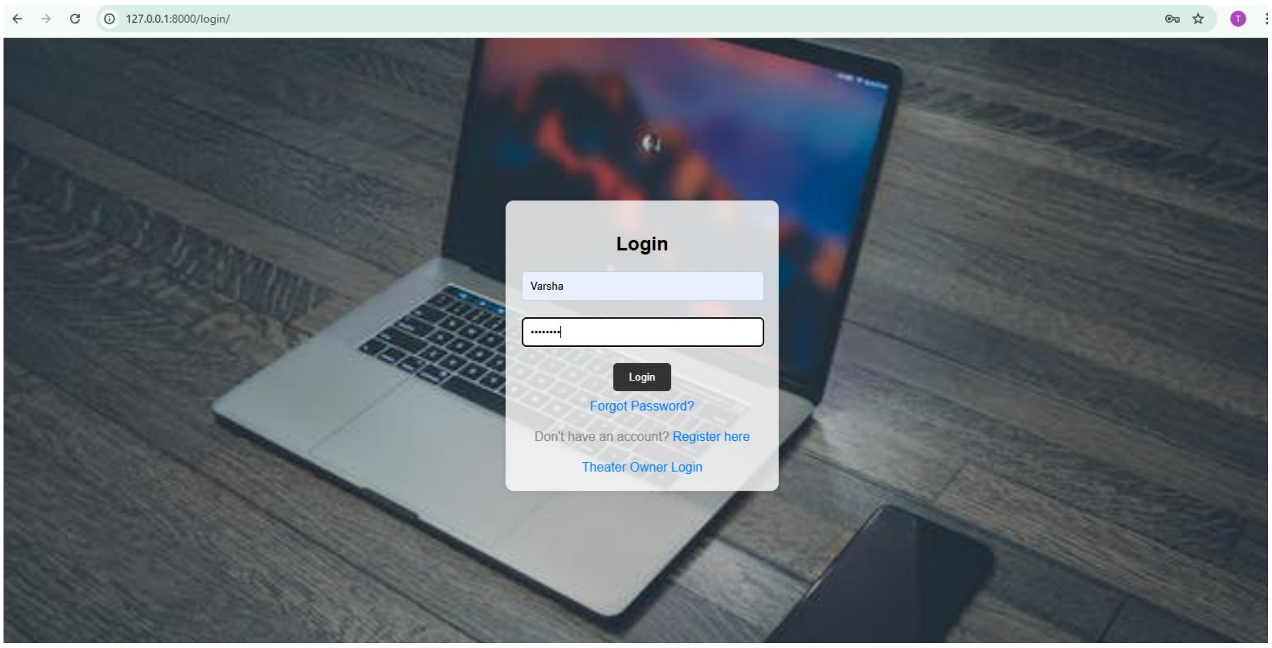
Test Case 1

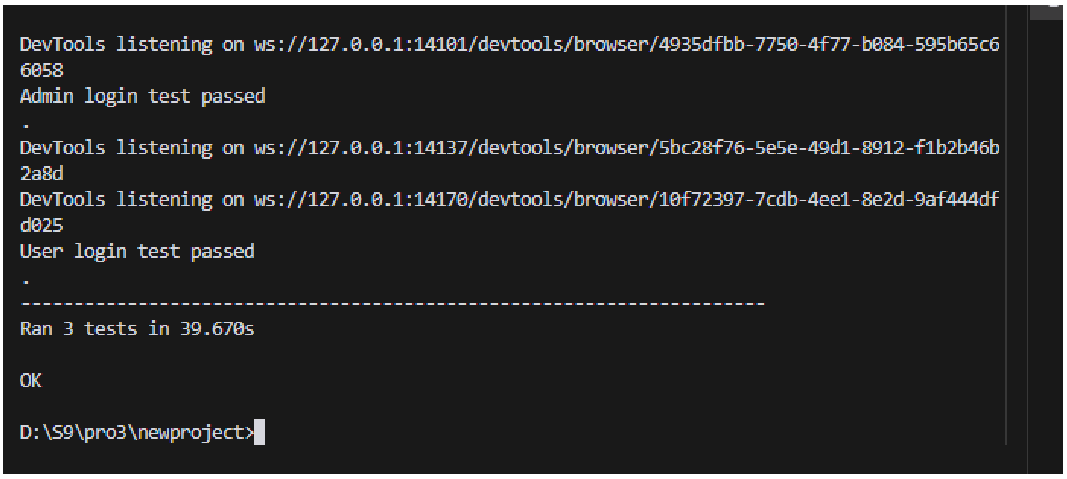
Code

```
import unittest
```

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
class TestLogin(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.get("http://localhost:8000/login")
    def test_admin_login(self):
        driver = self.driver
        driver.find_element(By.NAME, "username").send_keys("tansya")
        driver.find_element(By.NAME, "password").send_keys("tansya@23")
        driver.find_element(By.NAME, "password").send_keys(Keys.RETURN)
        time.sleep(2)
        try:
            self.assertIn("Admin Dashboard", driver.title)
            print("Admin login test passed")
        except AssertionError:
            print("Admin login test failed")
    def test_theatre_owner_login(self):
        driver = self.driver
        driver.get("http://localhost:8000/login")
        driver.find_element(By.NAME, "username").send_keys("Megha")
        driver.find_element(By.NAME, "password").send_keys("megha12")
        driver.find_element(By.NAME, "password").send_keys(Keys.RETURN)
        time.sleep(2)
        try:
            self.assertIn("Theatre Owner Dashboard", driver.title)
            print("Theatre owner login test passed")
        except AssertionError:
            print("Theatre owner login test failed")

    def test_user_login(self):
        driver = self.driver
        driver.get("http://localhost:8000/login")
        driver.find_element(By.NAME, "username").send_keys("Varsha")
        driver.find_element(By.NAME, "password").send_keys("varsha12")
        driver.find_element(By.NAME, "password").send_keys(Keys.RETURN)
        time.sleep(2)
        try:
            self.assertIn("User Dashboard", driver.title)
            print("User login test passed")
        except AssertionError:
            print("User login test failed")
    def tearDown(self):
        self.driver.quit()
if __name__ == "__main__":
    unittest.main()
```

Screenshot





```
DevTools listening on ws://127.0.0.1:14101/devtools/browser/4935dfbb-7750-4f77-b084-595b65c6
6058
Admin login test passed
.
DevTools listening on ws://127.0.0.1:14137/devtools/browser/5bc28f76-5e5e-49d1-8912-f1b2b46b
2a8d
DevTools listening on ws://127.0.0.1:14170/devtools/browser/10f72397-7cdb-4ee1-8e2d-9af444df
d025
User login test passed
.
----------------------------------------------------------------------
Ran 3 tests in 39.670s

OK

D:\S9\pro3\newproject>
```

Test Report

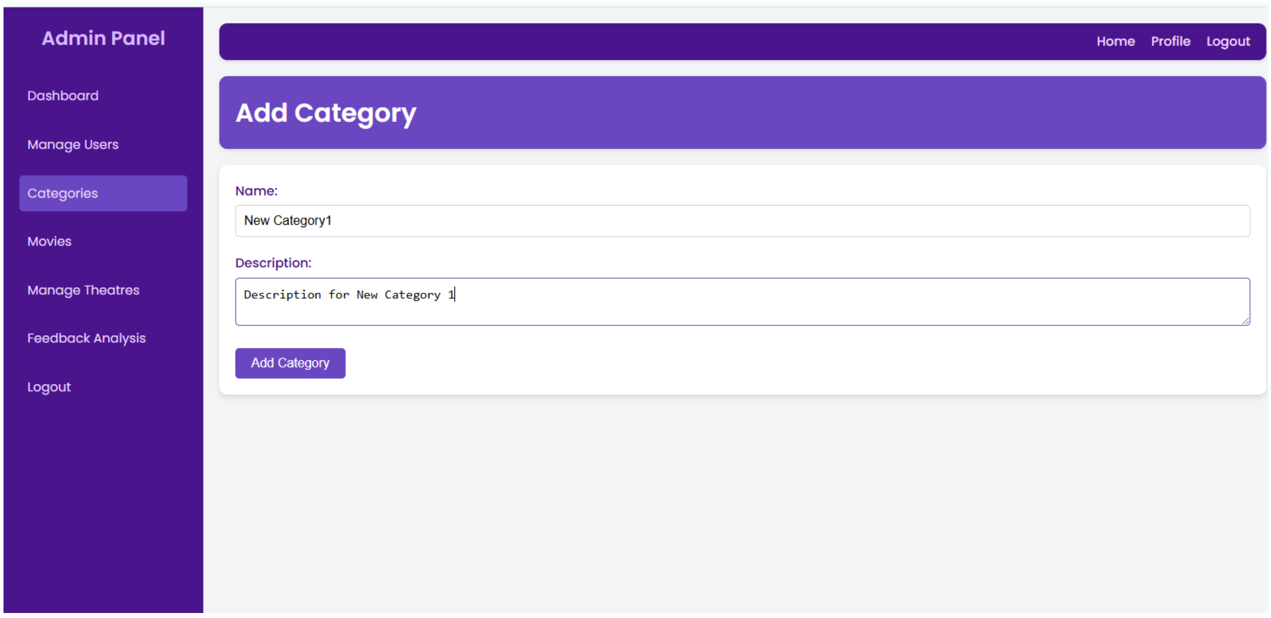
Test Case 1					
Project Name: Ticket Flix					
Login Test Case					
Test Case ID: Test_1			**Test Designed By: Tansya Babu**		
Test Priority(Low/Medium/High):			**Test Designed Date: 25/09/2024**		
Module Name: Login Page			**Test Executed By: Dr. Bijimol T.K.**		
Test Title: Verify Login with valid username and password			**Test Execution Date: 27/09/2024**		
Description: Login page Testing					
Pre-Condition: User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status (Pass/ Fail)**
1	Navigate to login page		Display Login Page	Login page displayed	Pass
2	Valid username entered	Username: Varsha	User should be able to login	User logged in and navigate to the dashboard	Pass
3	Valid Password entered	varsha@12			
4	Click on Submit Button				
Post-Condition: Email id and password is checked with database values for successful Login					

Test Case 2:

Code

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
def test_add_category():
    driver = webdriver.Chrome()
    driver.get('http://127.0.0.1:8000/categories/add/')

    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.XPATH,
'//h1[contains(text(), "Add Category")]')))

    driver.find_element(By.ID, 'name').send_keys('New Category1')
    driver.find_element(By.ID, 'description').send_keys('Description for new
category1')
    driver.find_element(By.XPATH, '//button[@type="submit"]').click()
    try:
        success_message = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.XPATH, '//div[contains(text(), "Category
added successfully")]'))
        )
        print("Test passed: Category added successfully.")
    except:
        print("Test passed: Category was  added successfully.")
    finally:
        driver.quit()
test_add_category()
```

Screenshot

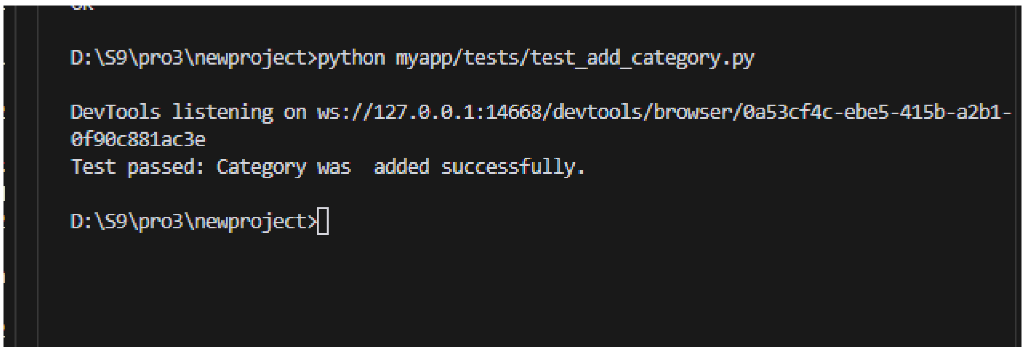


Add Category

Name:
New Category1

Description:
Description for New Category 1

Add Category



```
D:\S9\pro3\newproject>python myapp/tests/test_add_category.py

DevTools listening on ws://127.0.0.1:14668/devtools/browser/0a53cf4c-ebe5-415b-a2b1-
0f90c881ac3e
Test passed: Category was  added successfully.

D:\S9\pro3\newproject>
```

Test report

Test Case 2

Project Name: Ticket Flix	
Category Test Case	
Test Case ID: Test_2	**Test Designed By: Tansya Babu**
Test Priority (Low/Medium/High):	**Test Designed Date: 25/09/2024**
Module Name: Category Page	**Test Executed By: Dr. Bijimol T.K.**
Test Title: Add Category with valid name and description	**Test Execution Date: 27/09/2024**
Description: Category Page Testing	

Pre-Condition: Category has valid name and description

Step	Test Step	Test Data	Expected Result	Actual Result	Status (Pass/ Fail)
1	Navigate to add category page		Display Add Category Page	Add Category page displayed	Pass
2	Valid name entered	name: New Category1	Admin should be able to login	Admin logged in and navigate to the dashboard	Pass
3	Valid Description entered	Description name: Description for New Category 1			
4	Click on Add Button				

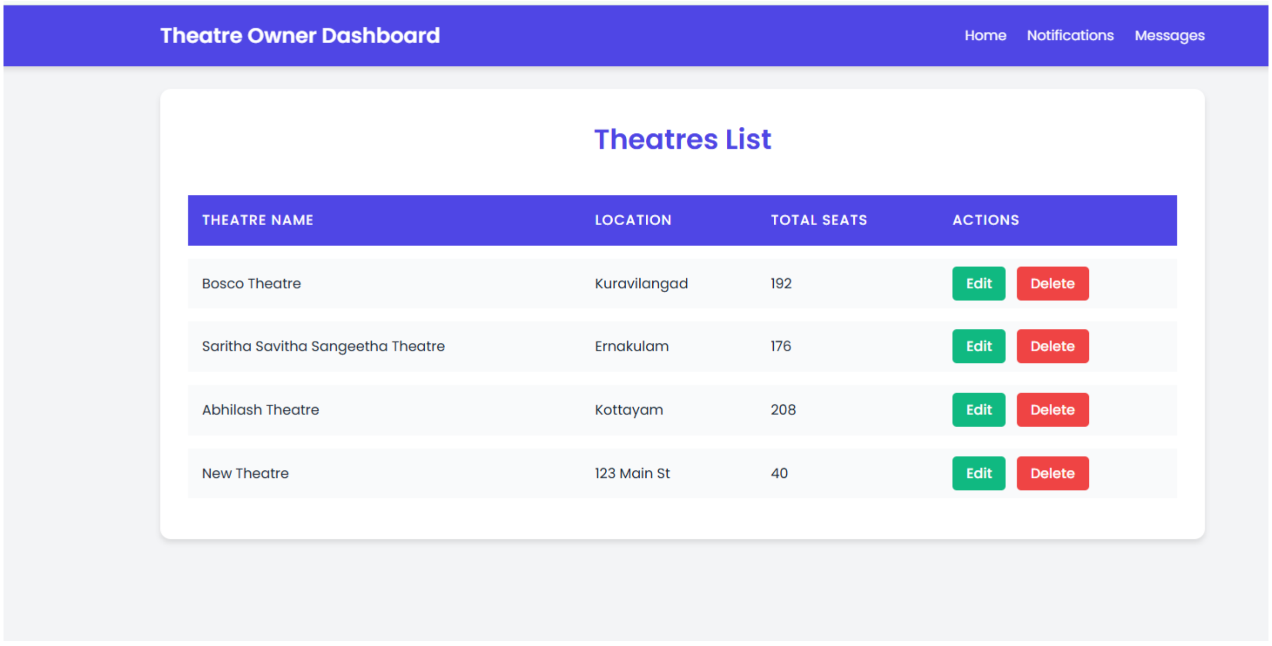
Post-Condition: Email id and password is checked with database values for successful Login

Test Case 3:

Code

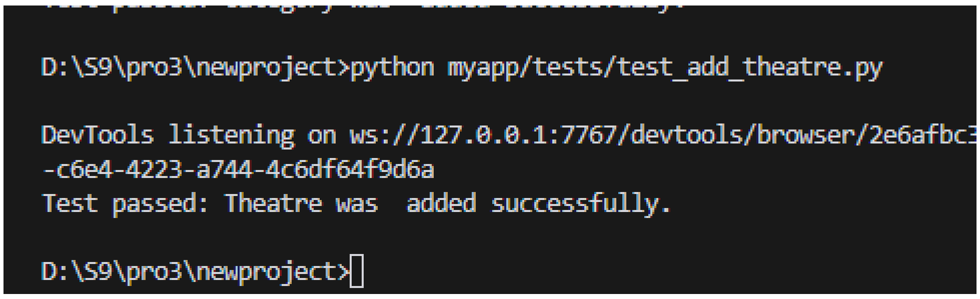
```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def test_add_theatre():
    driver = webdriver.Chrome()  # Make sure to have the ChromeDriver installed and
in your PATH
    driver.get("http://127.0.0.1:8000/add_theatre/")
      WebDriverWait(driver,   10).until(EC.presence_of_element_located((By.XPATH,
'//h1[contains(text(), "Add New Theatre")]')))

    driver.find_element(By.ID, "name").send_keys("New Theatre")
    driver.find_element(By.ID, "location").send_keys("123 Main St")
    driver.find_element(By.ID, "rows").send_keys("A,B,C,D")
    driver.find_element(By.ID, "seats_per_row").send_keys("10")
      driver.find_element(By.ID,   "food_and_beverage_options").send_keys("Popcorn,
Soda")
        driver.find_element(By.ID,   "accessibility_options").send_keys("Wheelchair
Accessible")
    parking_checkbox = driver.find_element(By.ID, "parking_available")
    if not parking_checkbox.is_selected():
        parking_checkbox.click()

    driver.find_element(By.XPATH, "//button[@type='submit']").click()
    try:
        success_message = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.XPATH, '//div[contains(text(), "Theatre
added successfully")]'))
        )
        print("Test passed: Theatre added successfully.")
    except:
        print("Test passed: Theatre was  added successfully.")
    finally:
        driver.quit()

test_add_theatre()
```

Screenshot



Theatres List

THEATRE NAME	LOCATION	TOTAL SEATS	ACTIONS
Bosco Theatre	Kuravilangad	192	Edit Delete
Saritha Savitha Sangeetha Theatre	Ernakulam	176	Edit Delete
Abhilash Theatre	Kottayam	208	Edit Delete
New Theatre	123 Main St	40	Edit Delete



```
D:\S9\pro3\newproject>python myapp/tests/test_add_theatre.py

DevTools listening on ws://127.0.0.1:7767/devtools/browser/2e6afbc3
-c6e4-4223-a744-4c6df64f9d6a
Test passed: Theatre was  added successfully.

D:\S9\pro3\newproject>
```

Test report

Test Case 3

Project Name: Ticket Flix					
Theatre Test Case					
Test Case ID: Test_1			**Test Designed By: Tansya Babu**		
Test Priority(Low/Medium/High):			**Test Designed Date: 25/09/2024**		
Module Name: Theatre Page			**Test Executed By: Dr. Bijimol T.K.**		
Test Title : Verify Theatre with valid name and location			**Test Execution Date: 27/09/2024**		
Description: Theatre page Testing					
Pre-Condition: User has valid username and password					
Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to add theatre page		Display Theatre Page	Theatre page displayed	Pass
2	Valid name entered	name: New Theatre	Theatre owner should be able to add theatre	Theatre Owner logged in and navigate to the dashboard	Pass
3	Valid Location entered	Location: 123 Main St.			
4	Click on Add Button				
Post-Condition: Theatre is added by theatre owner based on Location that the user can view					

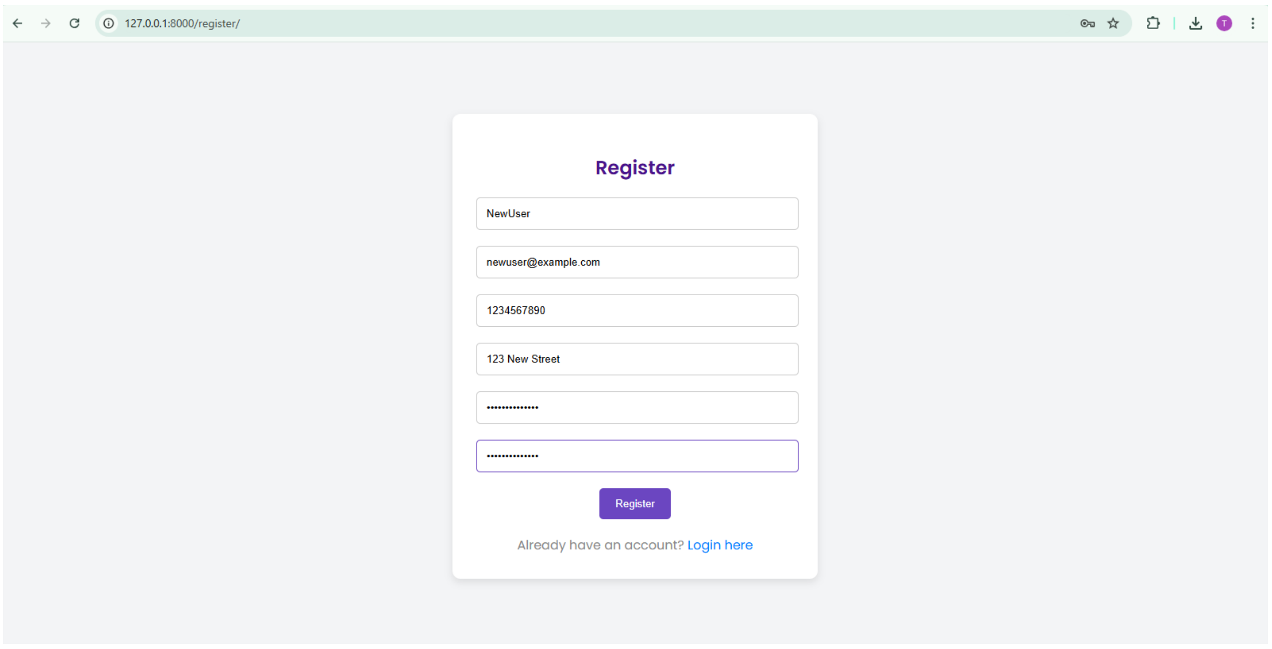
Test Case 4:

Code

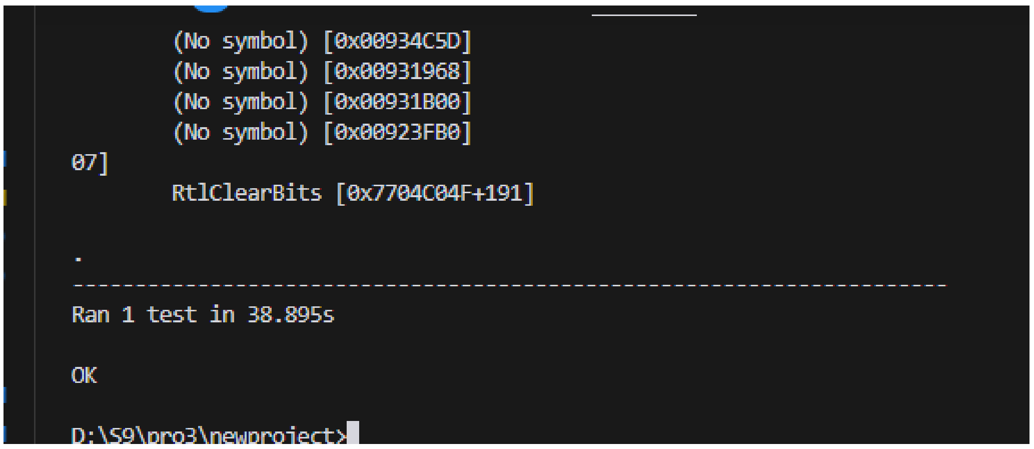
import unittest

```
import logging
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
class RegisterTest(unittest.TestCase):

    def setUp(self):
        service = Service(ChromeDriverManager().install())
        self.driver = webdriver.Chrome(service=service)
        self.driver.maximize_window()
        self.base_url = "http://localhost:8000"  # Ensure this is correct

    def test_successful_registration(self):
        driver = self.driver
        driver.get(f"{self.base_url}/register/")
        try:
            driver.find_element(By.NAME, "username").send_keys("NewUser")
           driver.find_element(By.NAME, "email").send_keys("newuser@example.com")
            driver.find_element(By.NAME, "phone").send_keys("1234567890")
            driver.find_element(By.NAME, "address").send_keys("123 New Street")
            driver.find_element(By.NAME, "password").send_keys("NewUserPass123")
                                                            driver.find_element(By.NAME,
"confirm_password").send_keys("NewUserPass123")
            driver.find_element(By.XPATH, "//button[@type='submit']").click()
            # Wait for successful registration confirmation
            WebDriverWait(driver, 20).until(
                        EC.presence_of_element_located((By.XPATH,  "//h1[contains(text(),
'Registration Successful')]"))  # Adjust this based on your success message
            )
            self.assertIn("Registration Successful", driver.page_source)
            logger.info("Successful registration test passed.")
        except Exception as e:
            logger.error(f"Error in test_successful_registration: {str(e)}")

    def tearDown(self):
        if self.driver:
            self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

Screenshot



NewUser
newuser@example.com
1234567890
123 New Street
••••••••••••••
••••••••••••••



```
        (No symbol) [0x00934C5D]
        (No symbol) [0x00931968]
        (No symbol) [0x00931B00]
        (No symbol) [0x00923FB0]
07]
        RtlClearBits [0x7704C04F+191]

.
----------------------------------------------------------------------
Ran 1 test in 38.895s

OK

D:\S9\pro3\newproject>
```

Test report

Test Case 4

Project Name: Ticket Flix	
Register Test Case	
Test Case ID: Test_4	**Test Designed By: Tansya Babu**
Test Priority (Low/Medium/High):	**Test Designed Date: 25/09/2024**
Module Name: Registration Page	**Test Executed By: Dr. Bijimol T.K.**
Test Title: Verify registration with valid credentials	**Test Execution Date: 27/09/2024**
Description: Registration page Testing	
Pre-Condition: User has valid credentials	

Step	**Test Step**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/ Fail)**
1	Navigate to register page	Username: New User Email: newuser@example.com Phone: 7306313815 Address:123 Main Street	Display registration Page	Registration page displayed	Pass
2	Click on register button		Add User type page displayed	Add user type page displayed	Pass
3	Valid details entered	Display add user type page			
4	Click on Submit Button				

Post-Condition: Details are validated with database values for successful registration

CHAPTER 6
IMPLEMENTATION

6.1 INTRODUCTION

Implementation is the process of turning a plan or idea into action. In the context of a project, implementation refers to the stage where the plan is put into action, and the project is executed. This stage involves the use of resources and the coordination of activities to achieve project goals. During the implementation stage, project managers must ensure that the project is executed efficiently and effectively. They need to monitor progress and make adjustments to the plan as needed. It is essential to have clear communication with team members and stakeholders to ensure that everyone is working towards the same goal.

The success of a project depends heavily on the implementation stage. A well-planned project may not succeed if the implementation is poorly executed. Therefore, it is crucial to have a detailed plan, adequate resources, and a skilled team to ensure a successful implementation. Additionally, effective monitoring and control mechanisms should be put in place to detect any issues and address them promptly.

6.2 IMPLEMENTATION PROCEDURES

The following are the typical procedures for implementing software: Planning and Preparation: Before starting the implementation process, it is essential to plan and prepare thoroughly. This includes developing a detailed implementation plan, identifying project requirements, assigning responsibilities, and ensuring that all necessary resources are available.

- **Development and Testing:** The development team writes and tests code, ensuring it meets functional requirements and integrates all components seamlessly, identifying and fixing bugs along the way.
- **User Acceptance Testing:** End-users test the software to confirm it aligns with their requirements, is user-friendly, and functions as expected in real-world scenarios.
- **Deployment:** After passing tests, the final product is deployed into the production environment, making it available for actual use by all intended users.
- **Post-Deployment Testing:** Testing is conducted after deployment to ensure the software operates correctly in the live environment and continues to meet functional requirements.
- **Maintenance and Support:** Ongoing maintenance ensures the software remains efficient, secure, and adaptable to evolving business needs while addressing any arising issues.
- **Evaluation:** Feedback from users and stakeholders is collected to evaluate the software's performance and identify areas for potential improvements or enhancements.

6.2.1User Training

User training is an essential part of software implementation, as it enables end-users to use the new software effectively and efficiently. The following are the typical steps involved in user training:

• **Training Needs Assessment:** This initial step involves identifying the training requirements of end-users by determining which software features they will use and the tasks they need to perform.

• **Training Plan Development:** A training plan is created based on the assessment, outlining objectives, training methods, and required materials to guide the training process.

• **Training Delivery:** The training is conducted for end-users using various methods, such as classroom sessions, online courses, or one-on-one training to suit different learning preferences.

• **Training Evaluation:** After training, its effectiveness is assessed by gathering feedback from end-users to determine if training objectives were met and identify areas for improvement.

• Follow-Up Training: Additional training sessions are provided based on feedback to fill any knowledge or skill gaps identified during the evaluation process.

Effective user training ensures that end-users can use the new software with confidence, leading to increased productivity and improved business outcomes. It is important to ensure that end-users receive adequate training and support during the implementation phase to ensure a smooth transition to the new software.

6.2.2 Training on the Application Software

Training on application software is crucial for ensuring that end-users can use the software effectively and efficiently. The following are the typical steps involved in training on application software:

• Orientation: An overview of the software is provided, highlighting its features, functionalities, and benefits, to generate interest and enthusiasm among end-users.

• **Hands-on Training:** End-users engage in practical training through live demonstrations or by accessing a test environment, allowing them to practice using the software effectively.

• **Role-Based Training:** Training is tailored to different roles and responsibilities, ensuring that each end-user receives relevant instruction aligned with their job functions.

• **Customized Training Materials:** Relevant and comprehensive training materials are developed, making them easy to understand and ensuring they meet the specific needs of end users.

• Evaluation: The effectiveness of the training is assessed to identify areas for improvement, confirming that end-users have gained the necessary knowledge and skills for software usage.

Effective training on application software ensures that end-users can use the software with confidence, leading to increased productivity and improved business outcomes. It is important to provide adequate training and support to end-users during the implementation phase to ensure a smooth transition to the new software.

6.2.3System Maintenance

System maintenance of project refers to the ongoing process of ensuring that the system functions optimally and meets the business needs. The following are some of the essential activities involved in system maintenance:

• **Regular Updates**: The system should be updated regularly with bug fixes, security patches, and new features to ensure that it remains efficient and secure.

• **Backup and Recovery:** Regular backups of data should be taken to prevent data loss in case of system failure. A disaster recovery plan should also be in place to ensure business continuity in case of unexpected system failures.

• **Performance Monitoring:** Regular monitoring of the system's performance should be carried out to identify any bottlenecks, potential issues, or areas for improvement.

• **User Support:** A help desk or support team should be available to respond to user queries and issues related to the system.

• **Security Measures:** The system should be protected with appropriate security measures, such as firewalls, antivirus software, and intrusion detection systems.

• **Training and Documentation:** End-users should receive ongoing training and support to ensure that they can use the system effectively. User manuals and documentation should also be regularly updated to reflect any changes or new features in the system.

6.2.4 Hosting

Hosting refers to the service that allows individuals and organizations to make their websites or web applications accessible on the internet. Web hosting providers offer the necessary technologies, infrastructure, and support to store website files and ensure they're available online. When a user types a website's URL, the web host's server delivers the content to the browser, enabling access to the site. There are different types of hosting services, including shared hosting, where resources are shared among multiple websites; dedicated hosting, which provides a server solely for one website; and cloud hosting, which uses virtual servers for scalability and reliability. Hosting plans vary in terms of performance, storage, and bandwidth, allowing users to choose services that match their specific needs and traffic demands.

Render

A web host is a company that provides the technologies and services required to make a website accessible on the internet. It stores a website's data—such as files, images, and code—on servers, allowing users to access the site by entering its web address in a browser. Web hosting services can

range from **shared hosting**, where multiple sites share server resources, to **dedicated hosting**, where one site has an entire server to itself for higher performance and security.
Other options include **VPS hosting**, which divides a server into virtual segments for more control, and **cloud hosting**, which uses a network of virtual servers to ensure scalability and reliability. Web hosting providers also offer various resources, including storage, bandwidth, and security features, allowing site owners to choose a plan that meets their website's performance, traffic, and budget needs.

Procedure for hosting a website on Render:

Step1:

Sign up for a Render account by visiting render.com and creating an account using your email or GitHub credentials.

Step2:

Once logged in, go to the Render dashboard and select **"New +"** at the top left, then choose **"Web Service"** to begin setting up your website.

Step3:

Connect Render to your code repository. You can deploy from platforms like GitHub or GitLab. After selecting your repository, Render will pull in your website's code.

Step4:

Configure your web service by entering a name, selecting the branch you want to deploy, and setting up the build and start commands (if applicable) for your website.

Step5:

Choose a plan. Render offers both free and paid plans, so pick one that meets your website's expected traffic and resource needs.

Step6:

Click **"Create Web Service"** to deploy your site. Render will start building and deploying your website from the selected repository.

Step7:

Once the deployment is complete, you'll see a live URL where your site is hosted on Render. You can use this URL to view and test your website online.

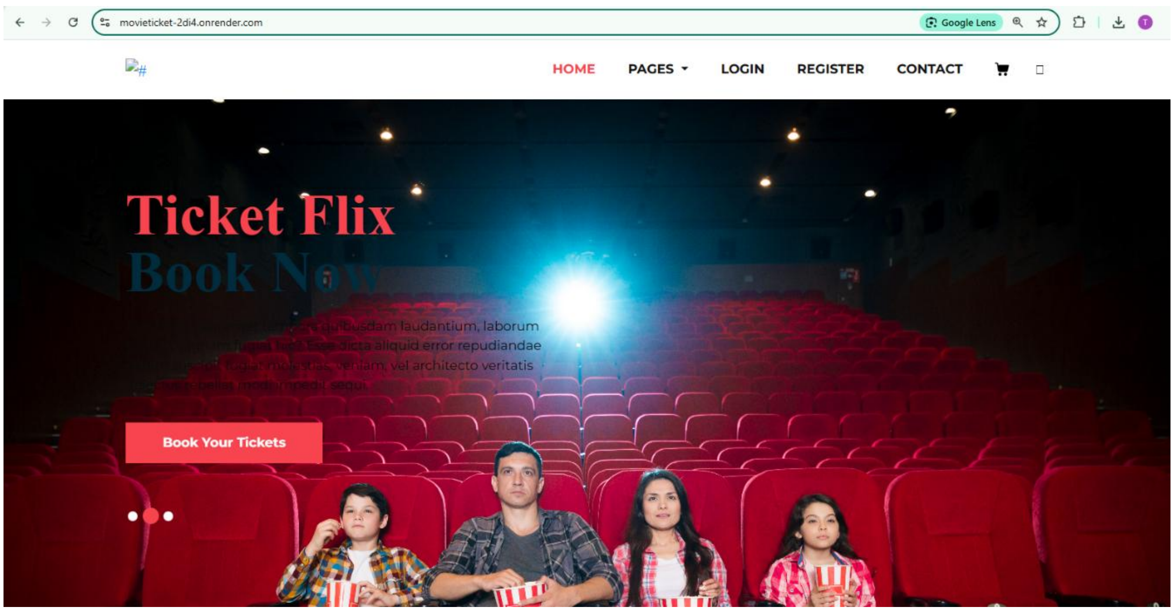
Step8:

(Optional) Configure a custom domain by going to the web service settings and following Render's instructions to link your domain.

Your site is now live and hosted on Render!

Hosted Website: Render
Hosted Link: https://movieticket-2di4.onrender.com/
Hosted Link QR Code

Screenshot



CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In conclusion, **TicketFlix**, an online movie ticket booking system, provides a seamless and efficient solution for users to browse, select, and book movie tickets at their preferred theatres. The platform significantly enhances the user experience by offering features such as personalized recommendations, real-time seat selection, and integrated payment systems. By streamlining the ticket booking process, TicketFlix reduces the hassle of manual bookings, improves the accessibility of showtimes, and boosts customer satisfaction. The system also benefits cinema operators by automating ticket sales, managing theatres efficiently, and providing detailed insights through analytics. With its scalable and flexible architecture built using modern web technologies, TicketFlix can adapt to future enhancements, ensuring it continues to meet the evolving needs of both users and theatre owners.

7.2 FUTURE SCOPE

The future scope of TicketFlix holds promising potential for expanding its functionality and improving user engagement. Advanced personalization can be further enhanced through AI-driven recommendations based on user preferences and viewing history. Integrating augmented reality (AR) features, such as virtual theatre tours and interactive seat selection, could elevate the user experience. Additionally, incorporating loyalty programs, real-time notifications for movie releases, and special offers could increase user retention. TicketFlix can also expand its platform by collaborating with food and beverage services to offer seamless in-seat ordering, enhancing the convenience of moviegoers. With further advancements in machine learning and predictive analytics, the system could provide more precise movie recommendations based on user moods and preferences. Scalability to international markets with support for multiple languages and currencies is another area of growth. These developments will ensure that TicketFlix stays ahead in the competitive entertainment industry, catering to evolving user demands and technology trends.

.

CHAPTER 8
BIBLIOGRAPHY

REFERENCES:

[1] "Aspect-Based Sentiment Analysis: A Survey of Techniques and Applications"

[2] "Deep Learning for Aspect-Based Sentiment Analysis: A Comprehensive Review"

[3] "Voice Feedback Integration in Aspect-Based Sentiment Analysis Systems"

[4] "Utilizing Voice Recognition in Aspect-Based Sentiment Analysis"

[5] "Sentiment Analysis of Spoken Language: Challenges and Opportunities"

[6] "A Hybrid Approach to Aspect-Based Sentiment Analysis with Voice Feedback"

[7] "Exploring User Sentiment through Voice Feedback: An Aspect-Based Method"

[8] "Real-Time Aspect-Based Sentiment Analysis for Voice Assistants"

[9] "Harnessing Voice Data for Enhanced Aspect-Based Sentiment Analysis"

[10] "Contextual Aspect-Based Sentiment Analysis Using Speech Recognition"

[11]"Emotion Detection in Voice Feedback: Implications for Aspect-Based Sentiment Analysis"

[12] "Leveraging Voice Feedback for Fine-Grained Aspect-Based Sentiment Analysis"

[13] "The Role of Prosody in Aspect-Based Sentiment Analysis of Spoken Language"

[14] "Adaptive Aspect-Based Sentiment Analysis with Voice Feedback"

[15] "Multimodal Sentiment Analysis: Integrating Voice and Text for Aspect Recognition"

[16] "Aspect-Based Sentiment Analysis of Customer Reviews Using Voice Input"

[17] "Exploring Voice Feedback in Aspect-Based Sentiment Analysis for E-Commerce"

[18] "Using Voice Feedback to Enhance Aspect-Based Sentiment Classification"

[19] "Towards Real-Time Aspect-Based Sentiment Analysis in Voice Assistants"

[20] "Aspect Extraction from Voice Feedback in Customer Service Applications"

[21]"Sentiment Analysis of Audio Feedback: An Aspect-Based Approach"

[22]"Combining Speech and Text for Aspect-Based Sentiment Analysis"

[23]"Voice-Based Customer Feedback: An Aspect-Oriented Sentiment Analysis Framework"

[24]"Improving Aspect-Based Sentiment Analysis Accuracy with Voice Data"

[25]"The Impact of Voice Tone on Aspect-Based Sentiment Analysis Outcomes"

CHAPTER 9

APPENDIX

9.1 Sample Code

Login and Register

Login.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - Movie Ticket Booking</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background: url('{% static "images/movie1.jpeg" %}') no-repeat center center fixed;
            background-size: cover;
        }
        .login-container {
            background-color: rgba(255, 255, 255, 0.8);
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            width: 300px;
            text-align: center;
        }
        .login-container h2 {
}
.login-container input[type="text"],
.login-container input[type="password"] {
   width: calc(100% - 20px);
   padding: 10px;
   margin-bottom: 20px;
   border: 1px solid #ccc;
   border-radius: 5px;
}
.login-container button {
   background-color: #333;
   color: #fff;
   padding: 10px 20px;
   border: none;
   border-radius: 5px;
   cursor: pointer;
}
.login-container button:hover {
   background-color: #555;
}
.login-container .message {
   margin-top: 20px;
   color: #888;
}
.login-container a {
   color: #007bff;
   text-decoration: none;
}
.login-container a:hover {
   text-decoration: underline;
}
.login-container .forgot-password {
   margin-top: 10px;
```

```
        }
    </style>
    <script>
        // JavaScript to disable the back button
        window.history.pushState(null, "", window.location.href);
        window.onpopstate = function () {
            window.history.pushState(null, "", window.location.href);
        };
    </script>
</head>
<body>
    <div class="login-container">
        <h2>Login</h2>
        <form method="post" action="{% url 'login' %}">
            {% csrf_token %}
            <input type="text" name="username" placeholder="Username" required>
            <input type="password" name="password" placeholder="Password" required>
            <button type="submit">Login</button>
            {% if error %}
            <p style="color: red;">Invalid credentials</p>

            {% endif %}
        </form>

        <div class="forgot-password">
            <a href="#">Forgot Password?</a>
        </div>

        <div class="message">
            Don't have an account? <a href="{% url 'register' %}">Register here</a>
        </div>
        <div class="message">
            <a href="{% url 'theatre_owner_dashboard' %}">Theater Owner Login</a>
        </div>
    </div>
</body>
</html>
```

Register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register - Movie Ticket Booking</title>
    <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Poppins', sans-serif;
            margin: 0;
            padding: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #f3f4f6;
        }
        .register-container {
            background-color: #fff;
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
            width: 400px;
    text-align: center;
}
.register-container h2 {
    margin-bottom: 20px;
    color: #4a148c;
}
.register-container input[type="text"],
.register-container input[type="password"],
.register-container input[type="email"],
.register-container input[type="tel"] {
    width: calc(100% - 20px);
    padding: 12px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    transition: border-color 0.3s;
}
.register-container input:focus {
    border-color: #6b46c1;
    outline: none;
}
.register-container button {
    background-color: #6b46c1;
    color: #fff;
    padding: 12px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}
.register-container button:hover {
    background-color: #4a148c;
}
```

```
    .register-container .message {
        margin-top: 20px;
        color: #888;
    }
    .register-container form {
        margin-top: 20px;
    }
    .register-container a {
        color: #007bff;
        text-decoration: none;
    }
    .register-container a:hover {
        text-decoration: underline;
    }
    .register-container .role-container {
        text-align: left;
        margin-bottom: 20px;
    }
    .role-container label {
        display: inline-block;
        margin-right: 10px;
    }
    .error {
        color: red;
        font-size: 14px;
    }
</style>
<script>
    function validateForm() {
        var username = document.forms["registerForm"]["username"].value;
        var email = document.forms["registerForm"]["email"].value;
        var phone = document.forms["registerForm"]["phone"].value;
        var address = document.forms["registerForm"]["address"].value;
var password = document.forms["registerForm"]["password"].value;
var confirmPassword = document.forms["registerForm"]["confirm_password"].value;
var role = document.forms["registerForm"]["role"].value;

var errorMessage = "";

if (username == "") {
   errorMessage += "Username must be filled out.\n";
} else {
   var usernamePattern = /^[a-zA-Z]+$/;
   if (!usernamePattern.test(username)) {
      errorMessage += "Username can only contain alphabets.\n";
   }
}
if (email == "") {
   errorMessage += "Email must be filled out.\n";
} else {
   var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
   if (!emailPattern.test(email)) {
      errorMessage += "Invalid email format.\n";
   }
}
if (phone == "") {
   errorMessage += "Phone number must be filled out.\n";
} else {
   var phonePattern = /^\d{10}$/;
   if (!phonePattern.test(phone)) {
      errorMessage += "Invalid phone number format. It should be 10 digits.\n";
   }
}
if (address == "") {
   errorMessage += "Address must be filled out.\n";
}
```

```
            if (password == "") {
                errorMessage += "Password must be filled out.\n";
            } else {
                var passwordPattern = /^[a-zA-Z0-9]+$/;
                if (!passwordPattern.test(password)) {
                    errorMessage += "Password must be alphanumeric.\n";
                } else if (password.length < 6) {
                    errorMessage += "Password must be at least 6 characters long.\n";
                }
            }
            if (confirmPassword == "") {
                errorMessage += "Confirm password must be filled out.\n";
            } else if (password != confirmPassword) {
                errorMessage += "Passwords do not match.\n";
            }
            if (!role) {
                errorMessage += "Role must be selected.\n";
            }

            if (errorMessage != "") {
                document.getElementById("error").innerText = errorMessage;
                return false;
            }
            return true;
        }
    </script>
</head>
<body>
    <div class="register-container">
        <h2>Register</h2>
        <form name="registerForm" method="post" action="{% url 'register' %}" onsubmit="return
validateForm()">
            {% csrf_token %}
            <input type="text" name="username" placeholder="Username" required><br>
            <input type="email" name="email" placeholder="Email" required><br>
            <input type="tel" name="phone" placeholder="Phone" required><br>
            <input type="text" name="address" placeholder="Address" required><br>

            {% comment %} <div class="role-container">
                <input type="radio" id="user" name="role" value="user" checked>
                <label for="user">User</label>

                <input type="radio" id="theatre_owner" name="role" value="theatre_owner">
                <label for="theatre_owner">Theatre Owner</label>
            </div> {% endcomment %}

            <input type="password" name="password" placeholder="Password" required><br>
            <input type="password" name="confirm_password" placeholder="Confirm Password"
required><br>
            <button type="submit">Register</button>
        </form>
        <p id="error" class="error"></p>
        {% if error %}
            <p style="color: red;">{{ error }}</p>
        {% endif %}
        <div class="message">
            Already have an account? <a href="{% url 'login' %}">Login here</a>
        </div>
    </div>
</body>
</html>
```

Movies Adding from Admin Side

```
{% load static %}
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add Movie - Admin Dashboard</title>
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600&display=swap"
rel="stylesheet">
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
    body {
      font-family: 'Poppins', sans-serif;
      background-color: #f3f4f6;
      color: #1f2937;
      line-height: 1.6;
    }
    .container {
      display: flex;
    }
    .sidebar {
      width: 250px;
      height: 100vh;
      background-color: #4a148c;
      color: #e9d8fd;
      padding: 20px;
      position: fixed;
    }
    .sidebar h2 {
      color: #d6bcfa;
    margin-bottom: 30px;
    font-size: 1.5em;
    text-align: center;
}
.sidebar ul {
    list-style: none;
}
.sidebar ul li {
    margin-bottom: 15px;
}
.sidebar ul li a {
    color: #e9d8fd;
    text-decoration: none;
    display: block;
    padding: 10px;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}
.sidebar ul li a:hover, .sidebar ul li.active a {
    background-color: #6b46c1;
}
.main-content {
    flex: 1;
    margin-left: 250px;
    padding: 20px;
}
.header {
    background-color: #6b46c1;
    color: #fff;
    padding: 20px;
    text-align: center;
    margin-bottom: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
.header h1 {
    font-size: 2em;
}
.navbar {
    background-color: #4a148c;
    padding: 10px 20px;
    margin-bottom: 20px;
    border-radius: 10px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}
.navbar ul {
    list-style-type: none;
    display: flex;
    justify-content: flex-end;
}
.navbar ul li {
    margin-left: 20px;
}
.navbar ul li a {
    color: #e9d8fd;
    text-decoration: none;
    font-weight: 500;
    transition: color 0.3s ease;
}
.navbar ul li a:hover {
    color: #fff;
}
form {
    background-color: #fff;
    border-radius: 10px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    padding: 20px;
    max-width: 600px;
    margin: 0 auto;
}
.form-group {
    margin-bottom: 20px;
}
label {
    display: block;
    margin-bottom: 5px;
    color: #4a148c;
    font-weight: 500;
}
input, select, textarea {
    width: 100%;
    padding: 10px;
    border: 1px solid #d1d5db;
    border-radius: 5px;
    font-size: 16px;
    transition: border-color 0.3s ease;
}
input:focus, select:focus, textarea:focus {
    outline: none;
    border-color: #6b46c1;
}
button {
    display: inline-block;
    padding: 10px 20px;
    background-color: #6b46c1;
    color: #fff;
    text-decoration: none;
    border-radius: 5px;
    transition: background-color 0.3s ease;
```

```
            border: none;
            cursor: pointer;
            font-size: 16px;
            font-weight: 500;
        }
        button:hover {
            background-color: #4a148c;
        }
        @media (max-width: 768px) {
            .container {
                flex-direction: column;
            }
            .sidebar {
                width: 100%;
                height: auto;
                position: static;
            }
            .main-content {
                margin-left: 0;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="sidebar">
            <h2>Admin Panel</h2>
            <ul>
                <li><a href="{% url 'admin_dashboard' %}">Dashboard</a></li>
                <li><a href="{% url 'view_customers' %}">Manage Users</a></li>
                <li><a href="{% url 'list_categories' %}">Categories</a></li>
                <li class="active"><a href="{% url 'list_movies' %}">Movies</a></li>
                <li><a href="{% url 'admin_theatre_list' %}">Manage Theatres</a></li>
            <li><a href="{% url 'sentiment_analysis' %}">Feedback Analysis</a></li>
            <li><a href="{% url 'logout' %}">Logout</a></li>
        </ul>
    </div>
    <div class="main-content">
        <div class="navbar">
            <ul>
                <li><a href="{% url 'admin_dashboard' %}">Home</a></li>
                <li><a href="#">Profile</a></li>
                <li><a href="{% url 'logout' %}">Logout</a></li>
            </ul>
        </div>
        <div class="header">
            <h1>Add Movie</h1>
        </div>
        <form method="post" action="{% url 'add_movie' %}" enctype="multipart/form-data"
onsubmit="return validateForm(event)">
            {% csrf_token %}
            <div class="form-group">
                <label for="title">Title:</label>
                <input type="text" id="title" name="title" required>
            </div>
            <div class="form-group">
                <label for="category">Category:</label>
                <select id="category" name="category" required>
                    {% for category in categories %}
                    <option value="{{ category.id }}">{{ category.name }}</option>
                    {% endfor %}
                </select>
            </div>
            <div class="form-group">
                <label for="release_date">Release Date:</label>
                <input type="date" id="release_date" name="release_date" required>
            </div>
            <div class="form-group">
                <label for="description">Description:</label>
                <textarea id="description" name="description" required></textarea>
            </div>
            <div class="form-group">
                <label for="poster">Poster:</label>
                <input type="file" id="poster" name="poster" accept=".jpg,.jpeg,.png" required>
            </div>
            <div class="form-group">
                <label for="director">Director:</label>
                <input type="text" id="director" name="director" required>
            </div>
            <div class="form-group">
                <label for="duration">Duration:</label>
                <input type="time" id="duration" name="duration" required>
            </div>
            <div class="form-group">
                <label for="price">Price:</label>
                <input type="number" id="price" name="price" step="0.01" required>
            </div>
            <button type="submit">Add Movie</button>
        </form>
    </div>
</div>
<script>

    function validateForm(event) {

    }

    function checkTitleExists(title) {
        }
    </script>
</body>
</html>
```

9.2 Screenshots

Landing Page

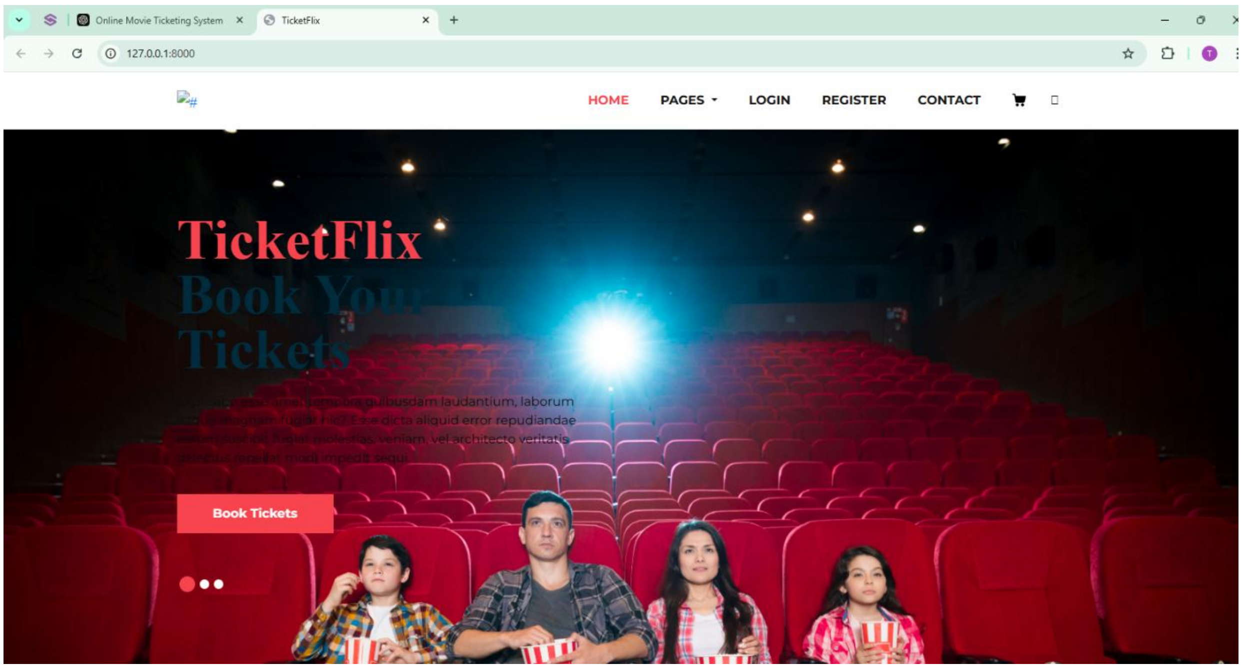
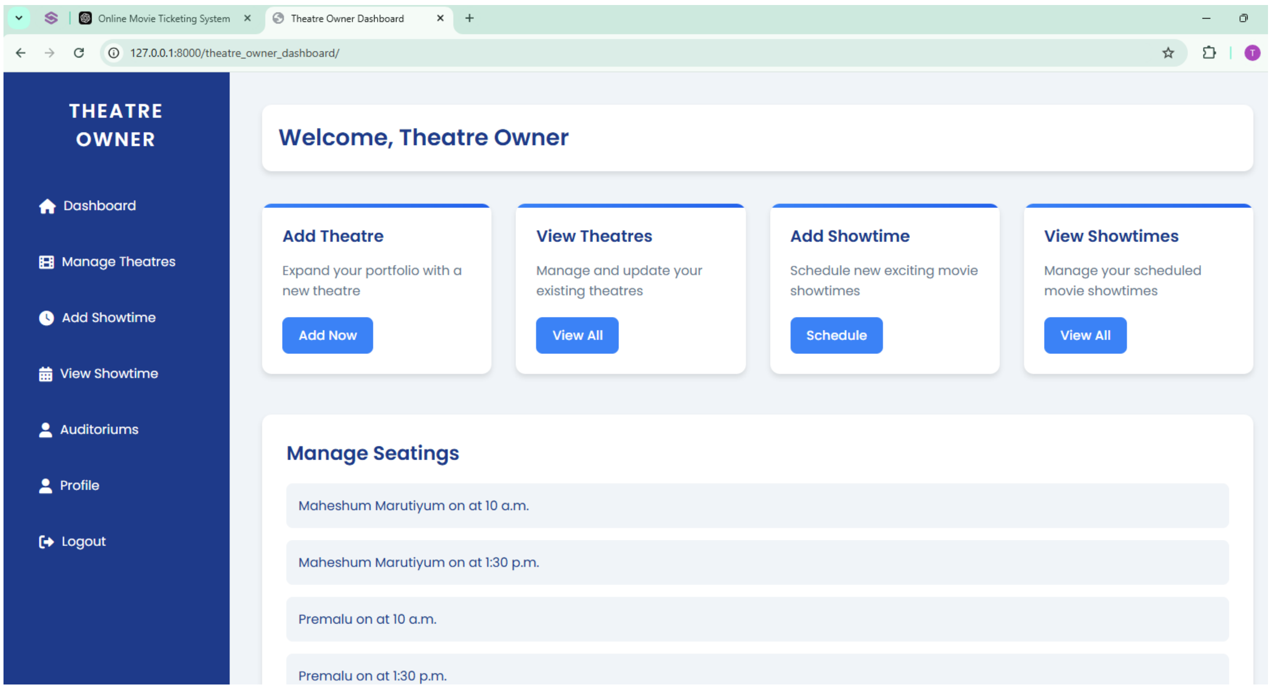


Figure 10: index page

Theatre Owner Dashboard



Welcome, Theatre Owner

Add Theatre
Expand your portfolio with a new theatre

View Theatres
Manage and update your existing theatres

Add Showtime
Schedule new exciting movie showtimes

View Showtimes
Manage your scheduled movie showtimes

Manage Seatings

Maheshum Marutiyum on at 10 a.m.

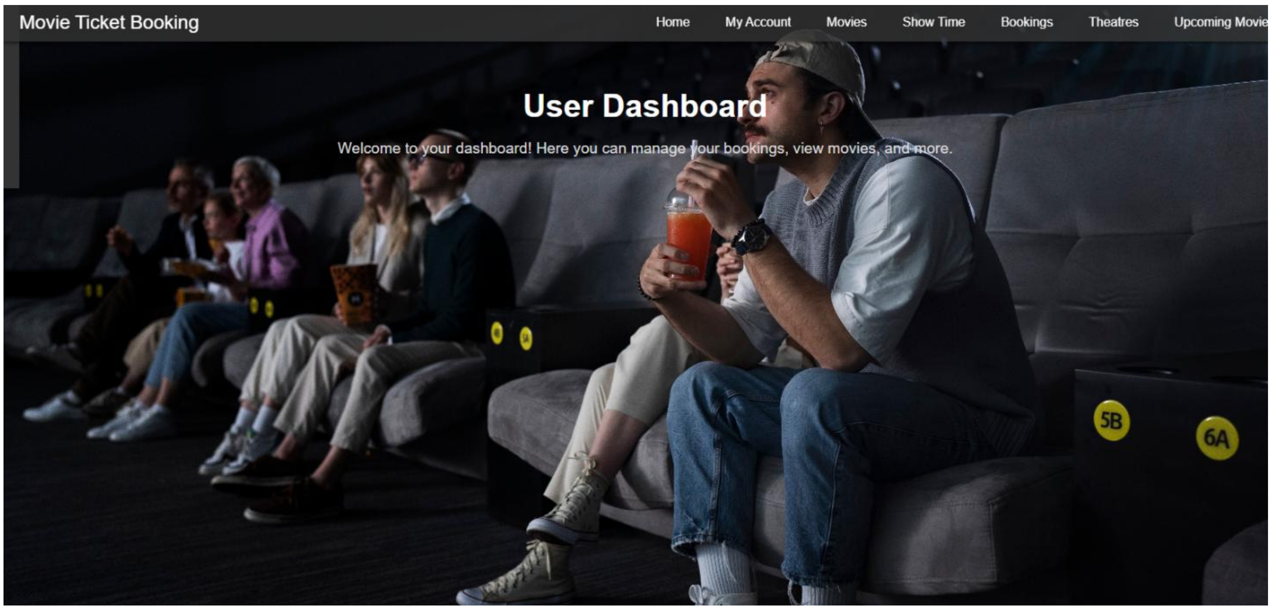
Maheshum Marutiyum on at 1:30 p.m.

Premalu on at 10 a.m.

Premalu on at 1:30 p.m.

Figure 11: Theatre Owner dashboard

User Dashboard

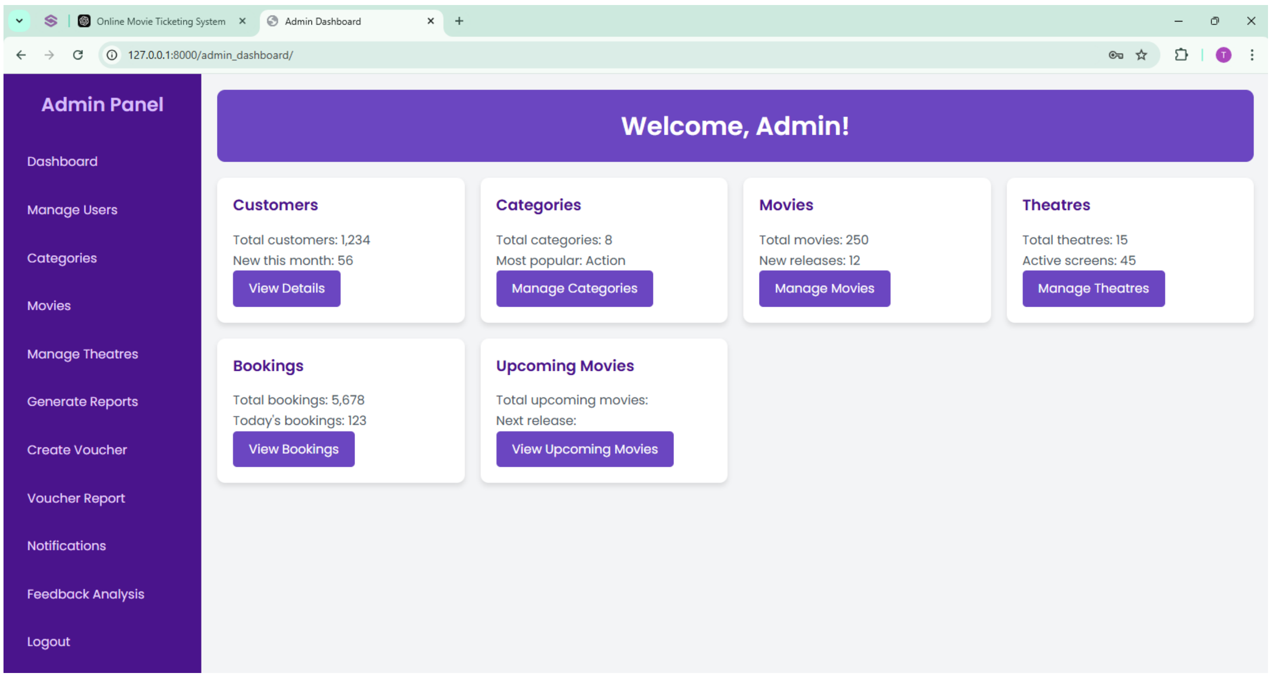


User Dashboard

Welcome to your dashboard! Here you can manage your bookings, view movies, and more.

Figure 12:User Dashboard

Admin Dashboard



Welcome, Admin!

Customers
Total customers: 1,234
New this month: 56

Categories
Total categories: 8
Most popular: Action

Movies
Total movies: 250
New releases: 12

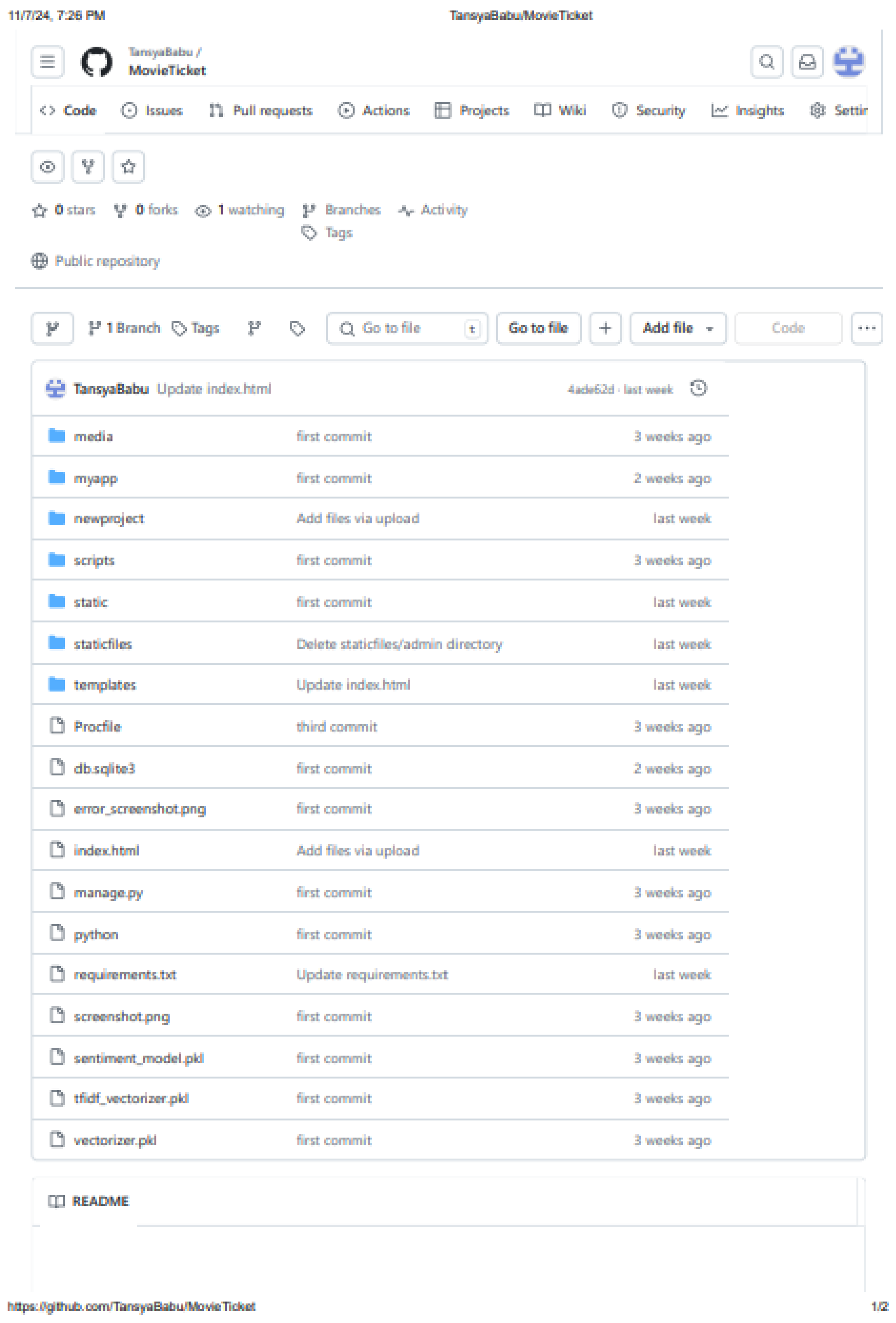
Theatres
Total theatres: 15
Active screens: 45

Bookings
Total bookings: 5,678
Today's bookings: 123

Upcoming Movies
Total upcoming movies:
Next release:

Figure 13: Admin Dashboard

9.3 Gitlog



11/7/24, 7:26 PM TansyaBabu/MovieTicket

TansyaBabu / MovieTicket

Code Issues Pull requests Actions Projects Wiki Security Insights Settir

0 stars 0 forks 1 watching Branches Activity Tags

Public repository

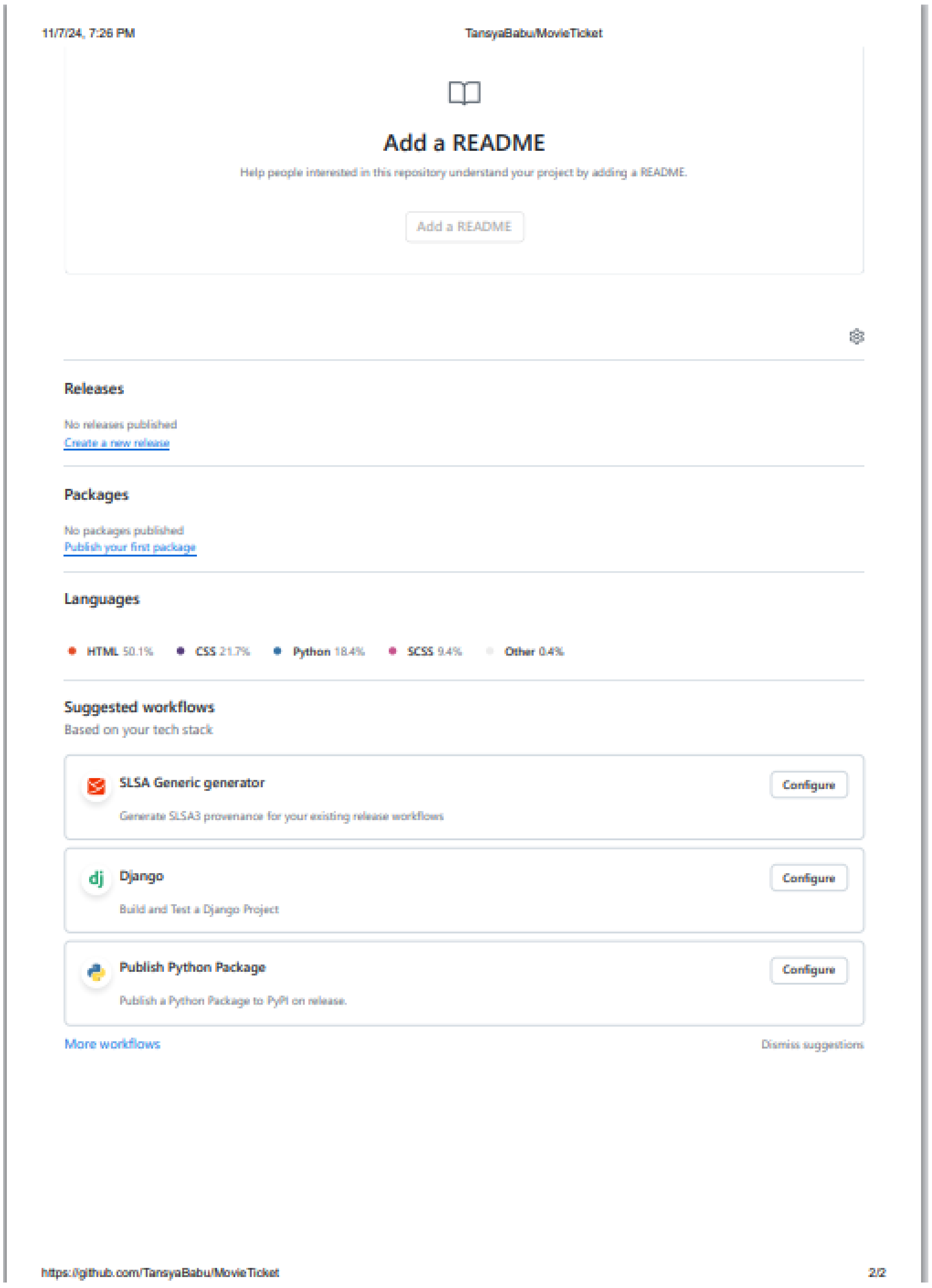
1 Branch Tags Go to file t Go to file + Add file Code ...

TansyaBabu Update index.html 4ade62d · last week

media	first commit	3 weeks ago
myapp	first commit	2 weeks ago
newproject	Add files via upload	last week
scripts	first commit	3 weeks ago
static	first commit	last week
staticfiles	Delete staticfiles/admin directory	last week
templates	Update index.html	last week
Procfile	third commit	3 weeks ago
db.sqlite3	first commit	2 weeks ago
error_screenshot.png	first commit	3 weeks ago
index.html	Add files via upload	last week
manage.py	first commit	3 weeks ago
python	first commit	3 weeks ago
requirements.txt	Update requirements.txt	last week
screenshot.png	first commit	3 weeks ago
sentiment_model.pkl	first commit	3 weeks ago
tfidf_vectorizer.pkl	first commit	3 weeks ago
vectorizer.pkl	first commit	3 weeks ago

README

https://github.com/TansyaBabu/MovieTicket 1/2



11/7/24, 7:26 PM TansyaBabu/MovieTicket

Add a README

Help people interested in this repository understand your project by adding a README.

Add a README

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● HTML 50.1% ● CSS 21.7% ● Python 18.4% ● SCSS 9.4% ● Other 0.4%

Suggested workflows
Based on your tech stack

SLSA Generic generator
Generate SLSA3 provenance for your existing release workflows
Configure

Django
Build and Test a Django Project
Configure

Publish Python Package
Publish a Python Package to PyPI on release.
Configure

More workflows Dismiss suggestions

https://github.com/TansyaBabu/MovieTicket 2/2