**A**

**Project Report**

**On**

**Ticketly**

**BTech-IT, Sem VI**

Prepared By:

Shah Dhwip Viralbhai (IT-129)

Soni Kunj Kalpeshkumar (IT-138)

Guided By:

Prof. V. H. Shah

Dept. of Information Technology



**Department of Information Technology**

**Faculty of technology,**

**Dharmsinh Desai University**

**College road, Nadiad- 387001**

**March 2025**

**CANDIDATE’S DECLARATION**

We declare that 6th semester report entitled “**Ticketly – A modern way for booking tickets of events**” is our own work conducted under the supervision of the guide **Prof. V.H. Shah.**

We further declare that to the best of our knowledge the report for B.Tech. VI semester does not contain part of the work which has been submitted either in this or any other university without proper citation.

Candidate’s Signature \_\_\_\_\_\_\_\_

Candidate’s Name: Shah Dhwip Viralbhai

Student ID: 22ITUOS151

Candidate’s Signature \_\_\_\_\_\_\_\_

Candidate’s Name: Soni Kunj Kalpeshkumar

Student ID: 22ITUOS133

**DHARMSINH DESAI UNIVERSITY**

**NADIAD-387001, GUJARAT**



**CERTIFICATE**

**This is to certify that the project carried out in the subject of Project-I, entitled** “**Ticketly – A modern way for booking tickets of events**” **and recorded in this report is a bonafide report of work of**

1. **Shah Dhwip Viralbhai: Roll No. IT129 ID No: 22ITUOS151**
2. **Soni Kunj Kalpeshkumar: Roll No. IT138 ID No: 22ITUOS133**

**of Department of Information Technology, semester VI. They were involved in Project work during academic year 2024 -2025.**

Prof. V. H. Shah

(Project Guide),

Department of Information Technology,

Faculty of Technology,

Dharmsinh Desai University, Nadiad

Date:

Prof. (Dr.) V. K. Dabhi

Head, Department of Information Technology,

Faculty of Technology,

Dharmsinh Desai University, Nadiad

Date:

### **ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to everyone who contributed to the successful completion of **Ticketly – A Ticket Booking System**.

First and foremost, I extend my heartfelt thanks to my mentors and faculty members for their invaluable guidance, continuous support, and insightful feedback throughout the development of this project. Their expertise has been instrumental in shaping the core functionalities of the application.

I would also like to thank my peers and colleagues for their constructive suggestions and feedback, which significantly helped in enhancing the user experience and refining the overall features of the platform.

A special appreciation goes to the open-source community and various online learning resources that provided essential knowledge on the **MERN Stack (MongoDB, Express.js, React.js, Node.js), JWT authentication, and payment gateway integration**—technologies that serve as the foundation of this project.

Lastly, I am truly grateful for the opportunity to work on this project, as it has strengthened my understanding of **full-stack development, role-based authentication, and online ticket booking systems**. The experience and skills gained through this project will undoubtedly be invaluable in my future endeavors.

**1. Dhwip Shah 2. Kunj Soni**

**Table of Contents**

**Candidate's Declaration** ................................................. i  
**Certificate** ....................................................................... ii  
**Acknowledgement** ......................................................... iii

**Chapter 1: Introduction** ................................................ 1  
1.1 Project Details ............................................................ 2  
1.2 Purpose ....................................................................... 3  
1.3 Scope .......................................................................... 4  
1.4 Objectives ................................................................... 5  
1.5 Technology and Literature Review ............................ 6

**Chapter 2: Project Management** .................................. 8  
2.1 Feasibility Study ........................................................ 9  
2.2 Project Planning ....................................................... 10  
2.3 Project Scheduling ................................................... 12

**Chapter 3: System Requirements Study** .................... 14  
3.1 Study of Current System .......................................... 15  
3.2 Problems and Weaknesses of Current System ......... 16  
3.3 User Characteristics ..................................................17  
3.4 Hardware and Software Requirements .................... 18

**Chapter 4: System Analysis** ......................................... 20  
4.1 Requirements of New System (SRS) ........................ 21  
4.2 Features of New System ............................................ 22  
4.3 Navigation Chart ....................................................... 24  
4.4 Class Diagram (Analysis Level) ............................... 25  
4.5 System Activity (Use Case Diagram) ....................... 26  
4.6 Sequence Diagram (Analysis Level) ........................ 27  
4.7 Data Modeling .......................................................... 28

**Chapter 5: System Architecture Design** ..................... 30  
5.1 System Architecture ................................................. 31  
5.2 Component Diagram ................................................ 32  
5.3 State Diagram ........................................................... 33

**Chapter 6: Implementation Planning** .......................... 35  
6.1 Implementation Environment .................................... 36  
6.2 Program/Modules Specification ................................ 37  
6.3 Coding Standards ...................................................…38

**Chapter 7: Testing for Ticketly (MERN Stack)** ......... 40  
7.1 Testing Plan ............................................................... 41  
7.2 Testing Strategy ......................................................... 42  
7.3 Testing Methods ........................................................ 43

**Chapter 8: User Manual** ............................................... 45

**Chapter 9: Limitations and Future Enhancements**.... 48

**Chapter 10: Conclusion and Discussion** .................... 50  
10.1 Conclusions and Future Enhancements ................. 51  
10.2 Discussion .............................................................. 52  
10.3 Summary of Project Work ..................................... 53

**Chapter 1 Introduction**

Ticketly is a modern web application designed to facilitate seamless online ticket booking for various events, movies, concerts, and sports activities. Developed using the **MERN (MongoDB, Express.js, React.js, Node.js) stack**, Ticketly offers a user-friendly interface, secure payment options, and a smooth booking experience.

### **1.1 Project Details**

Ticketly is a feature-rich ticket booking platform that allows users to:

* Browse upcoming events, movies, and shows.
* Book tickets based on location, date, and seat availability.
* Securely process payments through integrated gateways.
* View booking history and manage reservations.
* Access event details, including timings, venue, and ticket prices.

The platform is designed to provide a **responsive, scalable, and efficient** system for users and event organizers alike.

### **1.2 Purpose**

The primary purpose of Ticketly is to **digitize the event ticket booking process**, making it convenient, efficient, and accessible from anywhere. Key objectives include:

* Eliminating the need for physical ticket booking.
* Reducing long queues and manual booking errors.
* Offering real-time availability and instant confirmations.
* Providing event organizers with an easy-to-manage ticketing system.

### **1.3 Scope**

Ticketly caters to a wide range of users, including:

* **General users** who wish to book tickets for entertainment and events.
* **Event organizers** who need a platform to manage their shows.
* **Cinemas and theaters** looking for a robust online booking solution.

#### **Scope Limitations**

* The platform currently focuses only on **ticket booking** and does not manage **event execution or live streaming**.
* Initially, the service will be available **in selected cities**, with plans for future expansion.

### **1.4 Objectives**

The key objectives of Ticketly are:

1. **User-friendly Interface** – Ensuring seamless navigation for users across different devices.
2. **Real-time Seat Selection** – Displaying available seats dynamically for better decision-making.
3. **Secure Transactions** – Implementing encrypted payment gateways for secure transactions.
4. **Scalability** – Designing the system to handle high traffic loads efficiently.
5. **Admin Dashboard** – Providing event organizers with analytics, reports, and booking insights.

### **1.5 Technology and Literature Review**

Ticketly is built using the **MERN stack**, leveraging the following technologies:

* **MongoDB** – A NoSQL database used to store user details, bookings, and event data.
* **Express.js** – A lightweight Node.js framework for building the backend API.
* **React.js** – A front-end JavaScript library for creating a dynamic user experience.
* **Node.js** – A runtime environment that handles server-side logic and API requests.

Additionally, Ticketly integrates **third-party APIs** for payment gateways, email notifications, and real-time seat selection. The application follows **modern web development practices** such as RESTful APIs, JWT-based authentication, and cloud deployment for better scalability.

# **2. Project Management**

Ticketly is a comprehensive online ticket booking system designed to simplify event ticketing. Effective project management ensures smooth execution, timely delivery, and high-quality outcomes.

## **2.1 Feasibility Study**

A feasibility study helps determine whether the project is viable in terms of technology, time, operations, and implementation.

### **2.1.1 Technical Feasibility**

Ticketly is built using the **MERN stack**, which is widely adopted for developing modern web applications. The choice of technology ensures:

* **Scalability** – MongoDB allows flexible data storage for managing a growing number of users and events.
* **Performance** – React.js ensures fast rendering and dynamic user interfaces.
* **Security** – Authentication and data protection are implemented using JWT Given these factors, Ticketly is technically feasible and aligned with modern web development standards.

### **2.1.2 Time Schedule Feasibility**

The estimated timeline for developing Ticketly is **4-6 months**, divided into phases:

1. **Requirement Analysis (2 weeks)** – Gathering system requirements and defining scope.
2. **Design Phase (3 weeks)** – UI/UX design, database schema creation, and API planning.
3. **Development Phase (12 weeks)** – Backend and frontend implementation, API integration, and database setup.
4. **Testing & Debugging (4 weeks)** – Functional testing, security checks, and bug fixes. The timeline is feasible based on the complexity of features and available resources.

### **2.1.3 Operational Feasibility**

Ticketly is designed to be **user-friendly and efficient** for both customers and event organizers.

* **For users** – A simple, intuitive UI ensures easy event browsing and booking.
* **For event organizers** – An admin panel helps manage events, bookings, and revenue.
* **For support teams** – A customer support module handles user inquiries.

The operational model ensures smooth adoption by all stakeholders, making it highly feasible.

### **2.1.4 Implementation Feasibility**

Implementation will be carried out in phases:

1. **Prototype Development** – A minimal viable product (MVP) will be created.
2. **Beta Testing** – The platform will be tested with a small user base for feedback.
3. **Full-Scale Deployment** – After successful testing, Ticketly will be made publicly available.

Implementation risks like server downtime, database failures, and security threats will be mitigated using **cloud backups, security patches, and error handling mechanisms**.

## **2.2 Project Planning**

Effective planning ensures that Ticketly is developed efficiently and within the estimated timeframe.

### **2.2.1 Project Development Approach and Justification**

The **Agile Development Methodology** is chosen for Ticketly. Agile allows:

* **Incremental Development** – Features are built and tested in iterations.
* **Flexibility** – Changes can be made based on user feedback.
* **Collaboration** – Regular meetings ensure smooth coordination among developers, designers, and testers.

This approach ensures adaptability and faster delivery of functional components.

**2.2.2 Project Plan**

|  |  |  |
| --- | --- | --- |
| **Phase** | **Duration** | **Tasks** |
| Requirement Analysis | 2 weeks | System study, defining scope |
| UI/UX Design | 3 weeks | Wireframes, prototypes |
| Backend Development | 6 weeks | API creation, database setup |
| Frontend Development | 6 weeks | React.js components, UI integration |
| Testing & Debugging | 4 weeks | Unit testing, security testing |

### **2.2.3 Milestones and Deliverables**

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Expected Timeframe** | **Deliverable** |
| Requirements Finalized | Week 2 | System Requirement Document |
| UI/UX Completed | Week 5 | Wireframes, Design Mockups |
| Backend API Ready | Week 10 | Fully Functional API |
| Frontend Ready | Week 16 | Complete User Interface |
| Testing Completed | Week 20 | Bug-free Platform |

### **2.2.4 Roles and Responsibilities**

|  |  |  |
| --- | --- | --- |
| **Team Member** | **Role** | **Responsibilities** |
| Project Manager | Oversees Development | Ensures timeline adherence |
| Frontend Developer | UI/UX & React.js | Builds user interface |
| Backend Developer | Node.js & Database | Manages APIs and logic |
| QA Tester | Quality Assurance | Tests for bugs and security issues |

### **2.2.5 Group Dependencies**

* **Frontend team depends on backend APIs** for data integration.
* **Testing team depends on frontend and backend completion** for end-to-end testing.

## **2.3 Project Scheduling**

### **Project Scheduling Chart (Gantt Representation)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Task | Week  1-2 | | Week  3-5 | | Week  6-10 | | Week  11-16 | | Week  17-20 |
| |  | | --- | | **Requirement Analysis** |  |  | | --- | |  | | ✅ |  | |  | |  | |  | |
| UI/UX Design |  | ✅ | | ✅ | |  | |  | |
| Backend Development |  |  | | ✅ | |  | | ✅ | |
| Frontend Development   |  | | --- | |  | |  |  | | ✅ | | ✅ | | ✅ | |
| Testing & Debugging |  |  | |  | |  | | ✅ | |

# **3. System Requirements Study**

Ticketly is an online ticket booking platform built using the **MERN stack** to provide a seamless booking experience for events, movies, concerts, and more. This section outlines the system requirements based on the current industry standards and the specific needs of Ticketly.

## **3.1 Study of Current System**

Most existing ticket booking platforms rely on **manual processes or outdated online systems**, leading to inefficiencies such as:

* **Limited real-time seat selection** and booking options.
* **Inconsistent user experiences** across different devices.
* **Slow transaction processing**, causing failed bookings.
* **Security risks** due to lack of proper authentication and encryption.

To address these issues, Ticketly leverages modern technologies, ensuring a **fast, secure, and user-friendly** ticket booking system.

## **3.2 Problems and Weaknesses of Current System**

The traditional ticket booking system faces several challenges:

1. **Manual Booking Hassles** – In-person ticket purchasing leads to long queues and errors.
2. **Limited Availability Information** – Some platforms do not provide real-time seat availability, leading to overbookings.
3. **Payment Failures** – Poor integration with payment gateways results in failed transactions.
4. **Lack of Personalized User Experience** – Many systems do not offer recommendations based on user preferences.
5. **Security Vulnerabilities** – Weak authentication processes expose user data to breaches.

Ticketly overcomes these issues by offering **real-time seat updates, secure transactions, and a user-centric interface**.

## **3.3 User Characteristics**

**Ticketly is designed for multiple user groups, each with specific needs:**

|  |  |
| --- | --- |
| **User Type** | **Description** |
| **General Users** | Customers who book tickets for events, movies, and concerts. They need a simple interface and secure payments. |
| **Event Organizers** | Admins who create, manage, and track event ticket sales. They require a dashboard for analytics. |
| **System Administrators** | Responsible for platform maintenance, database management, and security updates. |

Each user category interacts with the system through tailored interfaces to enhance efficiency and usability.

## **3.4 Hardware and Software Requirements**

To run Ticketly efficiently, the following minimum requirements are needed:

### **Software Requirements**

|  |  |
| --- | --- |
| **Software** | **Requirement** |
| **Operating System** | Windows/MacOS |
| **Backend** | Node.js with Express.js |
| **Frontend** | React.js |
| **Database** | MongoDB |
| **Development Tools** | VS Code, Postman, Git |
| **Browser** | Chrome, Firefox, Edge |
|  |

These specifications ensure smooth operation across different devices and environments.

## **3.5 Constraints**

### **3.5.1 Regulatory Policies**

* Compliance with **data protection laws** (GDPR, IT Act 2000) to protect user data.
* Adherence to **payment gateway security standards** (PCI DSS).
* Local government regulations for **ticketing and entertainment services**.

### **3.5.2 Hardware Limitations**

* Ticketly may **not perform well on outdated systems** with low RAM and weak processors.
* Hosting and **scalability require cloud infrastructure**, making it dependent on internet access.

### **3.5.3 Interfaces to Other Applications**

* Ticketly integrates with **payment gateways** (Razor pay).
* **Email/SMS services** for ticket confirmations and notifications.

### **3.5.4 Parallel Operations**

* The system should allow **multiple users to book tickets simultaneously** without conflicts.
* Parallel processing in the backend ensures **real-time seat availability updates**.

### **3.5.5 Higher Order Language Requirements**

* The platform requires **JavaScript-based frameworks** (Node.js, React.js).
* MongoDB is used instead of traditional SQL databases for **flexibility and scalability**.

### **3.5.6 Reliability Requirements**

* **99.9% uptime** is expected for uninterrupted ticket booking.
* **Load balancing and caching mechanisms** ensure fast response times.

### **3.5.7 Criticality of the Application**

* **High criticality** since payment transactions and seat bookings require **accuracy**.
* Any system failure could lead to **financial losses and poor user experience**.

### **3.5.8 Safety and Security Considerations**

* **Two-factor authentication (2FA)** for user login.
* **Encrypted Authentication transactions** using SSL/TLS.
* **Regular security audits** to prevent cyber threats.

## **3.6 Assumptions and Dependencies**

### **Assumptions**

* Users have access to **stable internet connections** for smooth booking.
* Payment gateways and APIs **function correctly without downtime**.
* Ticketly will be hosted on **a reliable cloud platform** for scalability.

### **Dependencies**

* **Third-party payment providers** (e.g., Razor pay, PayPal) for transactions.
* **External API services** for event details, notifications.

# **System Analysis**

System analysis involves understanding user needs, defining system requirements, and designing an optimized structure for the new system. Ticketly is an online ticket booking platform that enables seamless booking, event management, and secure transactions.

## **4.1 Requirements of New System (SRS)**

The **Software Requirements Specification (SRS)** for Ticketly defines the functional and non-functional requirements to ensure smooth system operation.

### **4.1.1 User Requirements**

The system must provide an intuitive interface for users, ensuring:

* Easy **event browsing and filtering** (by category, date, venue, price).
* **Real-time seat selection** and instant booking confirmation.
* Secure and **multiple payment options** (credit card, UPI, net banking).
* **User account management** (profile, history, wishlist, notifications).
* **Event organizers' dashboard** for creating and managing events.
* **Customer support system** for queries and refunds.

### **4.1.2 System Requirements**

#### **Functional Requirements**

* User authentication using **JWT and OAuth**.
* **Event search and filtering** based on multiple criteria.
* **Real-time seat availability updates** with a reservation timeout.
* **Secure payment processing** with **SSL encryption**.
* **Event creation and management module** for organizers.
* **User review and rating system** for events.
* **Admin panel** for user and event moderation.

#### **Non-Functional Requirements**

* **Scalability:** Support thousands of concurrent users.
* **Performance:** Load time < 3 sec for any action.
* **Security:** Encryption for stored user data, secure API endpoints.
* **Reliability:** 99.9% uptime with cloud-based hosting.
* **Usability:** User-friendly UI with mobile compatibility.

## **4.2 Features of New System**

Ticketly offers several **enhanced features** over traditional systems:

**For Users**

* Intuitive UI for easy ticket booking.
* AI-powered **event recommendations** based on preferences.
* **QR code-based digital tickets** to reduce fraud.
* Wallet system for **easy refunds and cashback offers**.

**For Event Organizers**

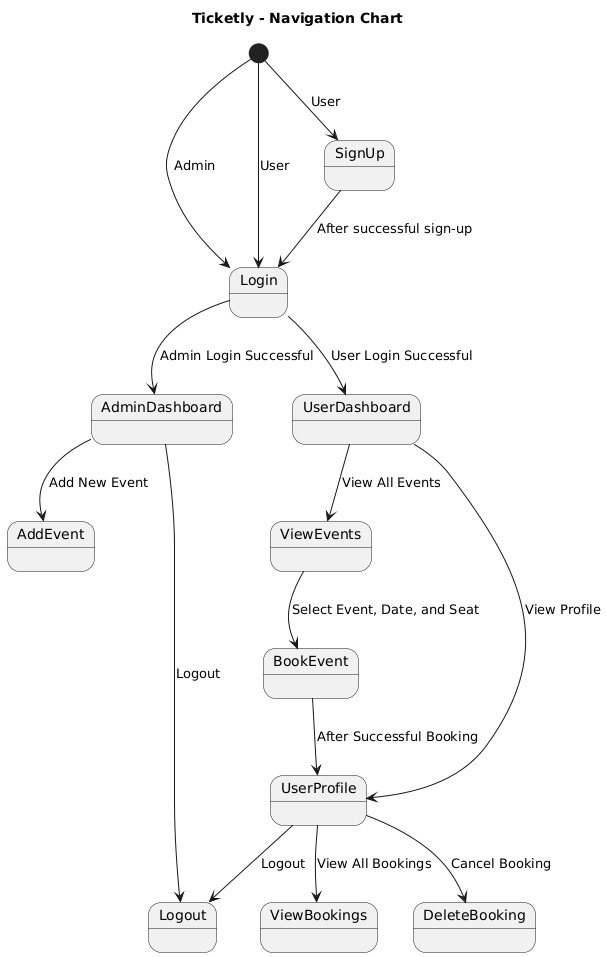
* **Event analytics dashboard** (ticket sales, revenue insights).
* **Automated promotional campaigns** via email/SMS.
* **Dynamic pricing** (early bird discounts, last-minute deals).

### **For Admins**

* **Fraud detection system** for fake bookings.
* **Blacklist feature** for unauthorized event organizer.

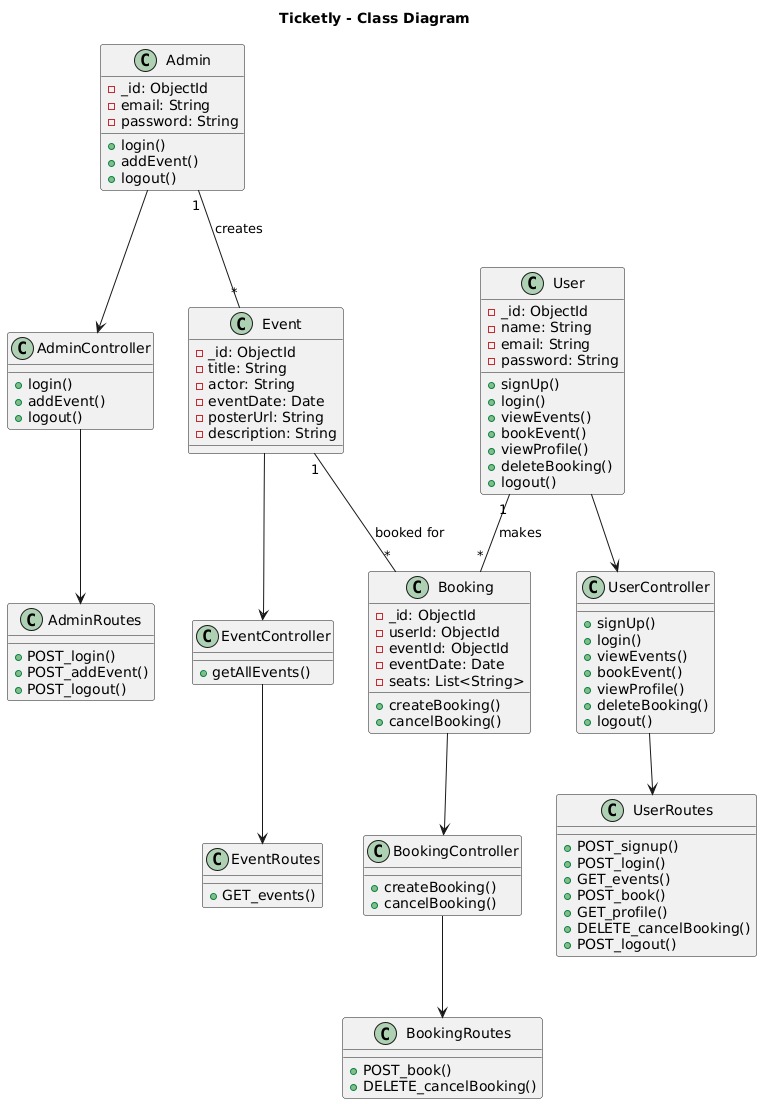
## **4.3 Navigation Chart**

The navigation chart outlines **how users interact** with the system:



## **4.4 Class Diagram (Analysis Level)**

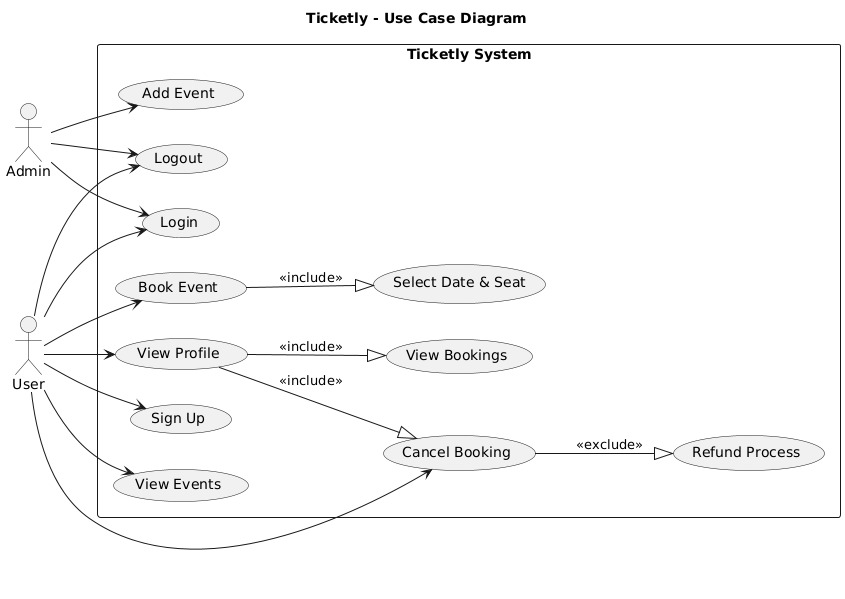
The **class diagram** models the key **entities and their relationships** in Ticketly.



## **4.5 System Activity (Use Case Diagram)**

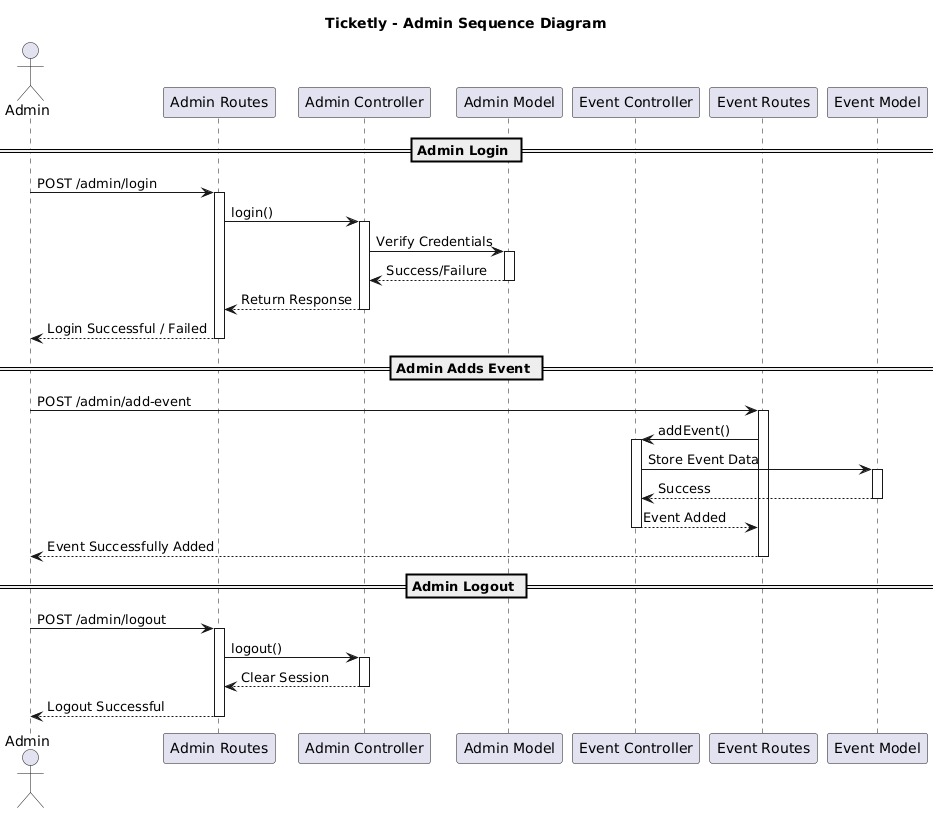
### **Main Use Cases:**

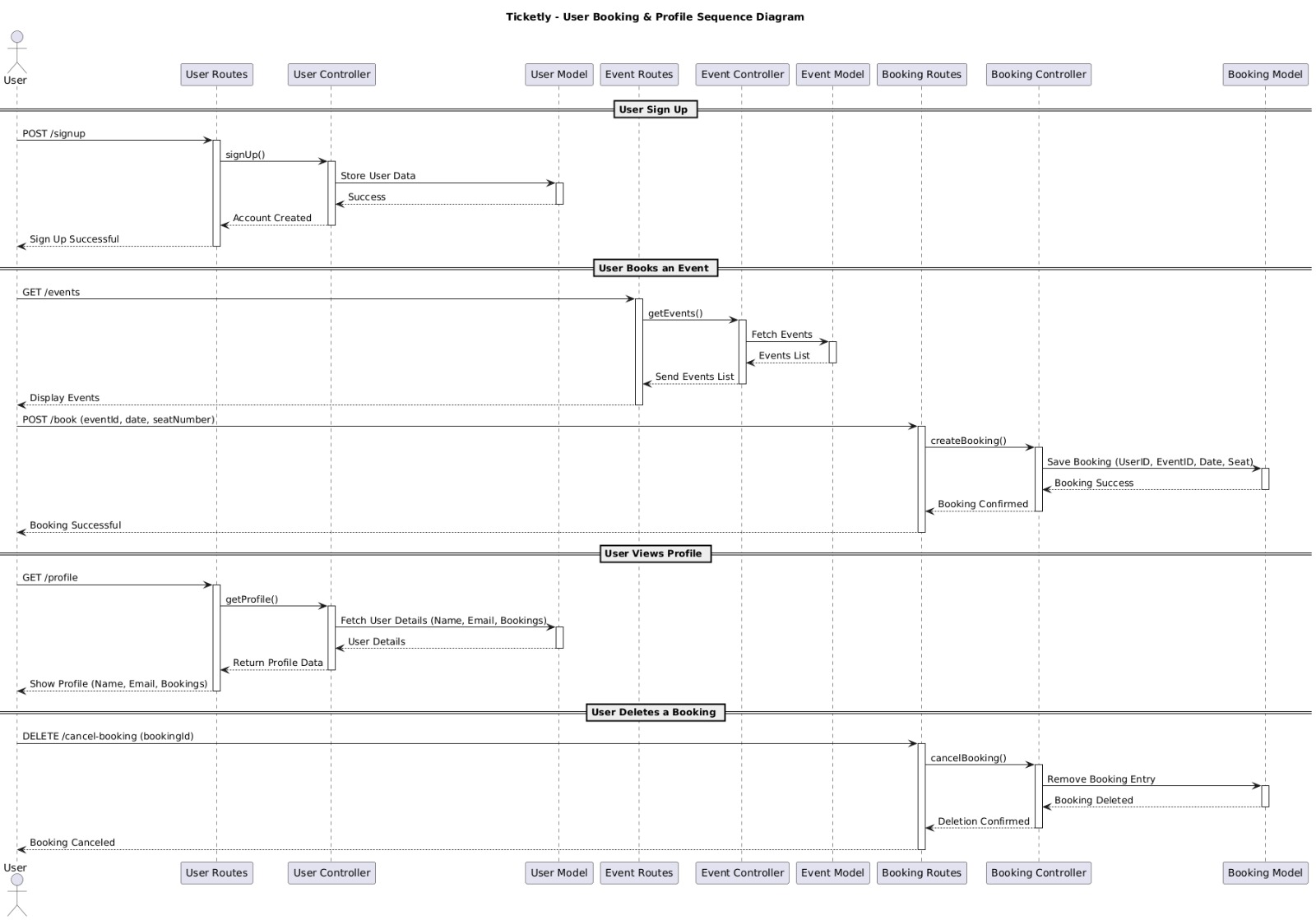
* **User:** Browse events, book tickets, cancel bookings, make payments.
* **Event Organizer:** Create events, manage tickets, track revenue.
* **Admin:** Approve/reject events, monitor transactions.



## **4.6 Sequence Diagram (Analysis Level)**

A **sequence diagram** illustrates the step-by-step flow of booking tickets:





## **4.7 Data Modeling**

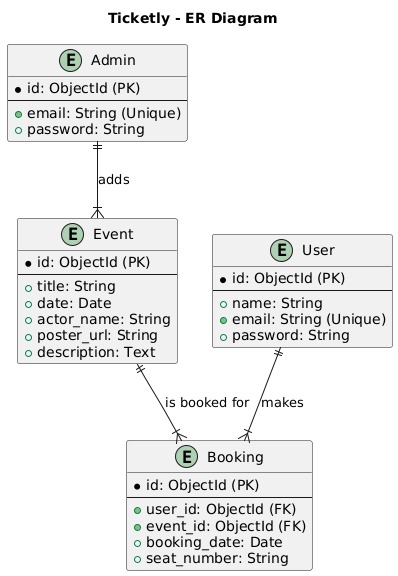
### **4.7.1 Data Dictionary**

|  |  |  |
| --- | --- | --- |
| ****Attribute**** | ****Type**** | ****Description**** |
| userID | INT | Unique identifier for users |
| eventID | INT | Unique identifier for events |
| bookingID | INT | Unique identifier for each booking |
| seatNumber | STRING | Seat assigned to the user |
| amountPaid | FLOAT | Total amount paid for booking |
| paymentStatus | STRING | "Pending", "Successful", "Failed" |

These fields ensure **structured data storage**.

### **4.7.2 ER Diagram**

The **ER (Entity-Relationship) diagram** defines how data entities relate to each other.



**Entities & Relationships:**

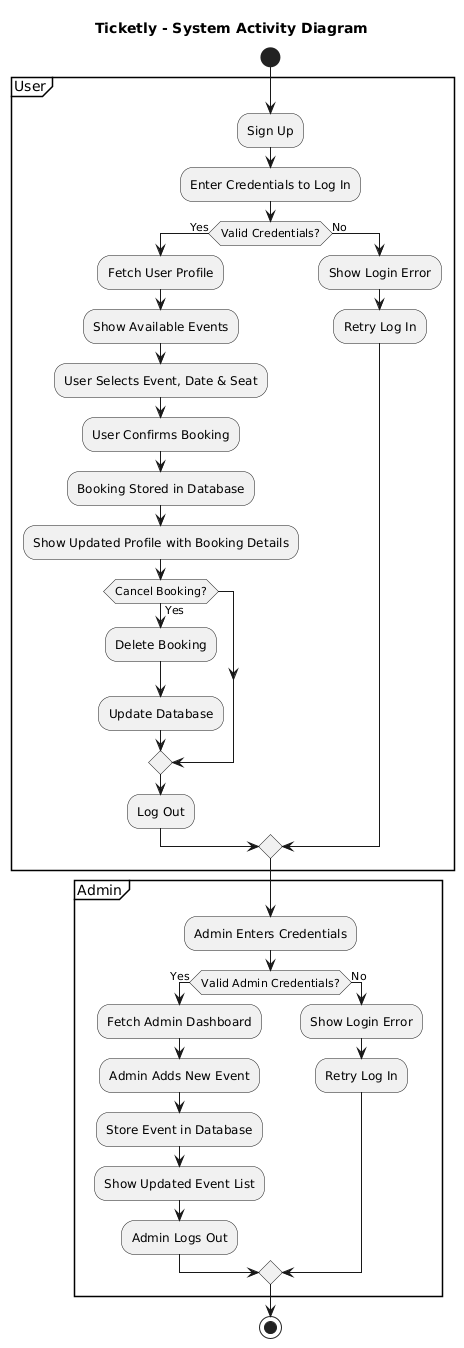
* **User** (1:N) **Booking** (Each user can have multiple bookings).

**Event** (1:N) **Booking** (Each event can have multiple bookings).

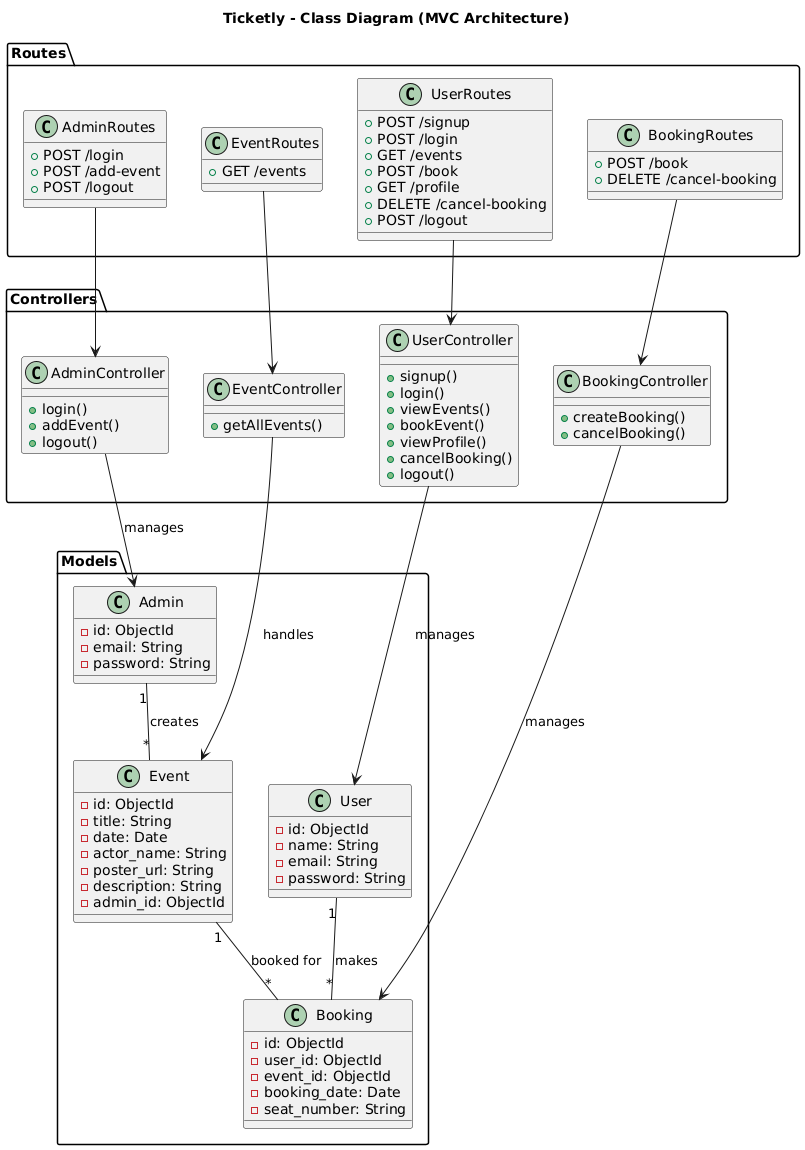
* **Booking** (1:1) **Payment** (Each booking has one payment record).

**Key Features in ER Model:**  
Normalized database structure  
Avoids redundancy  
Ensures data integrity

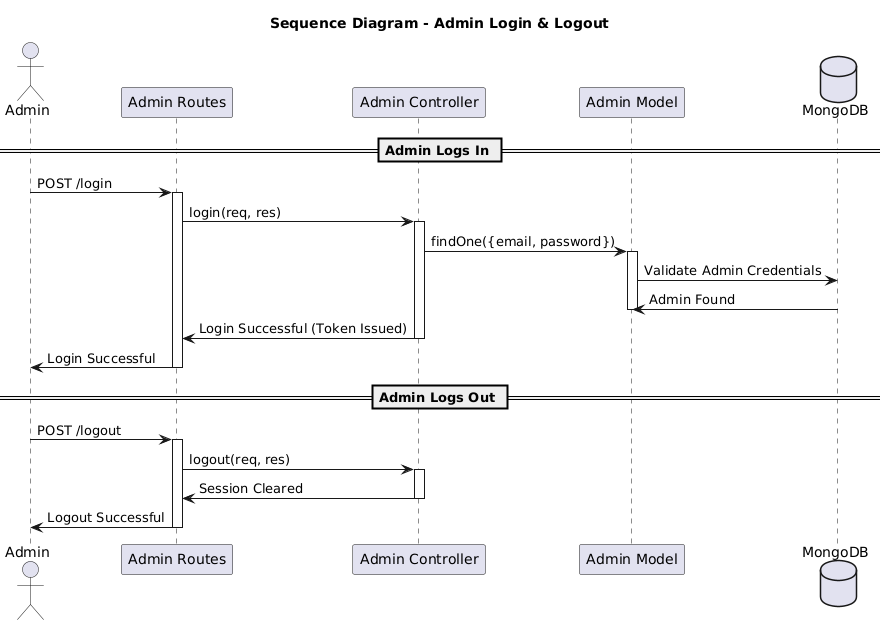
* 1. **System Activity**

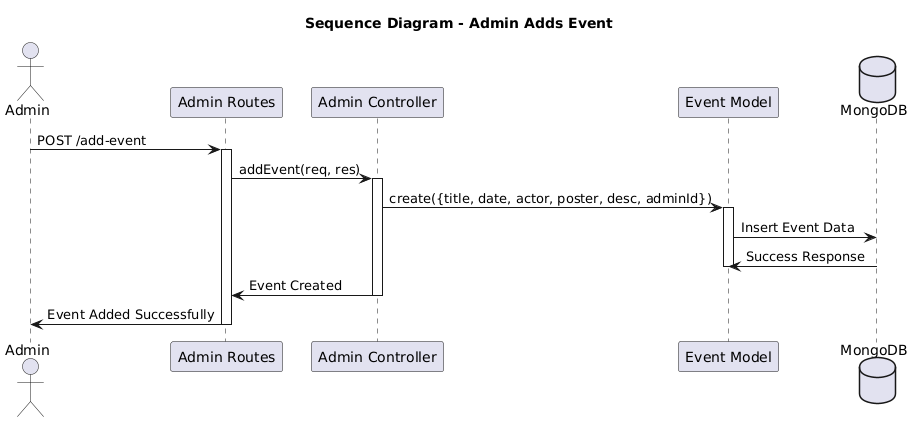
****

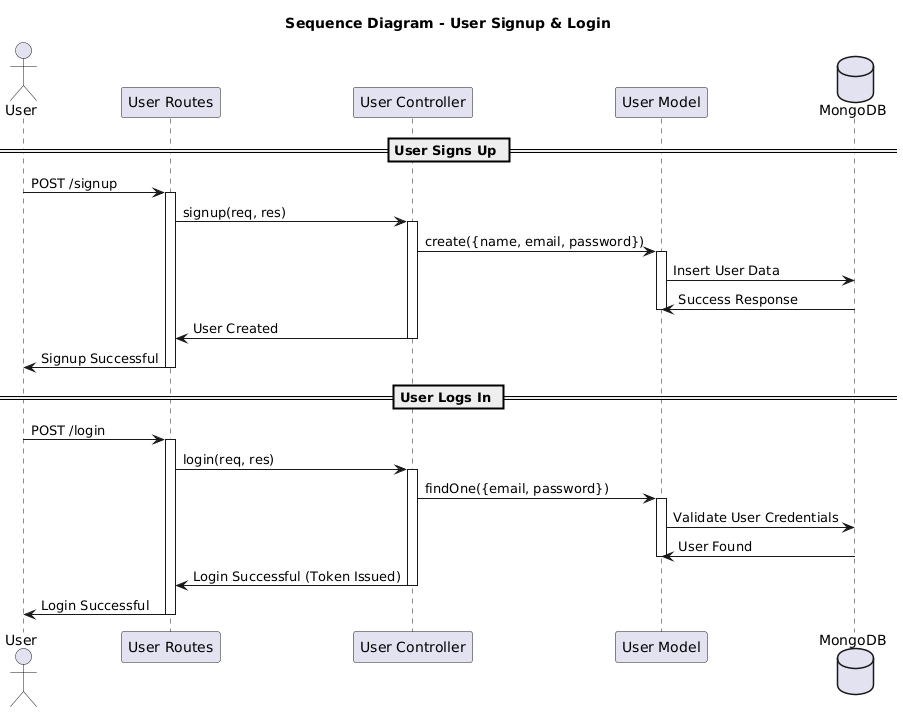
* 1. **System Architecture Design**
     1. **Class Diagram**

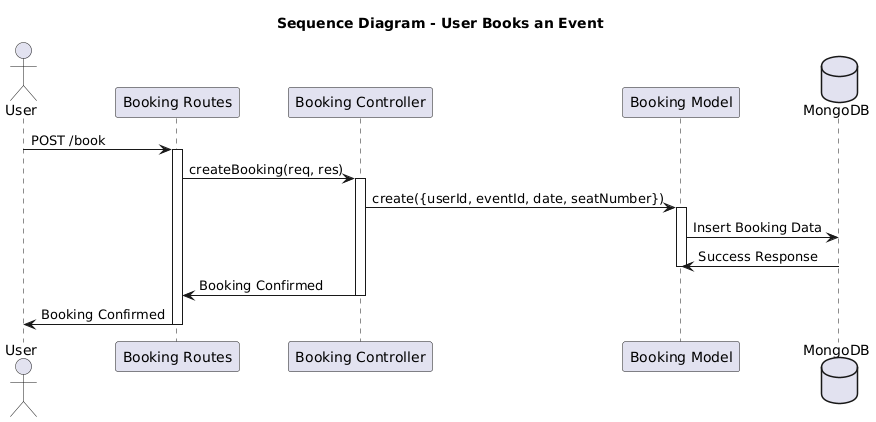
****

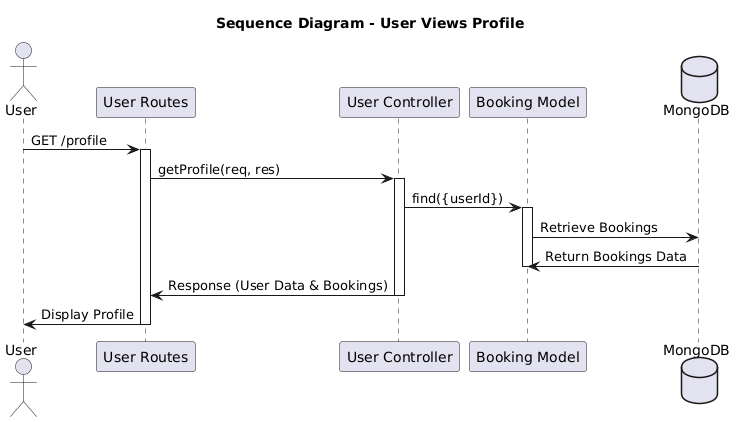
* + 1. **Sequence Diagrams**

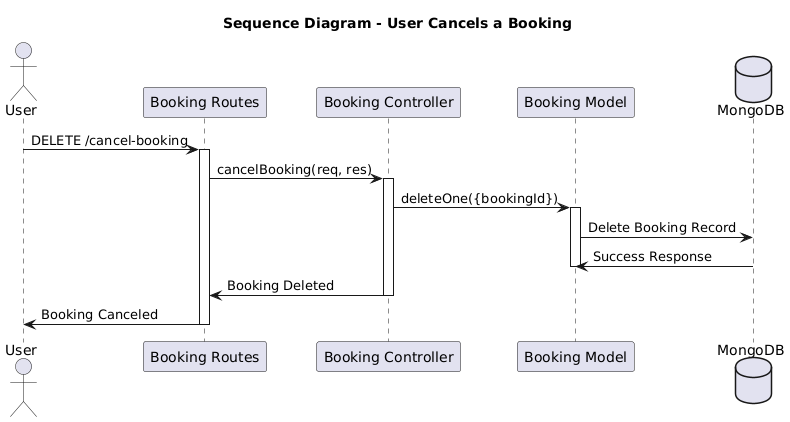
****

****

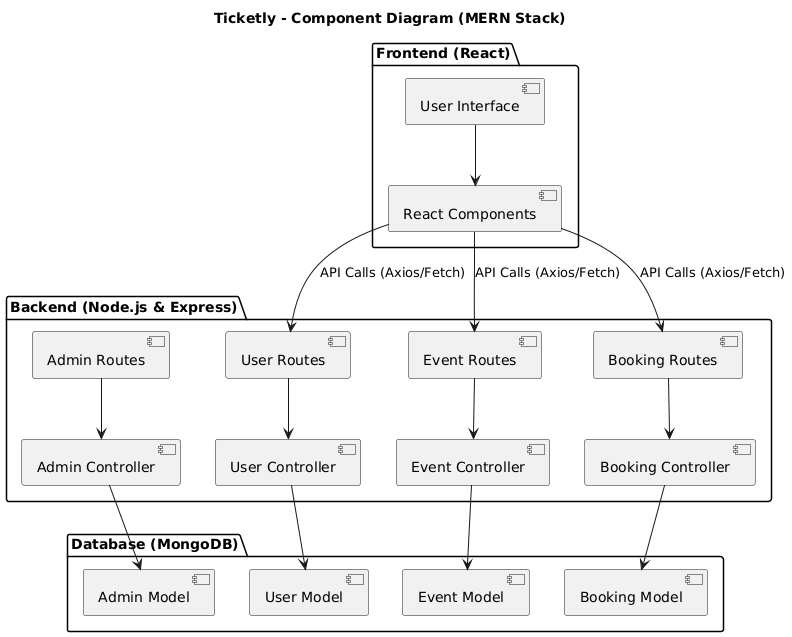
****

****

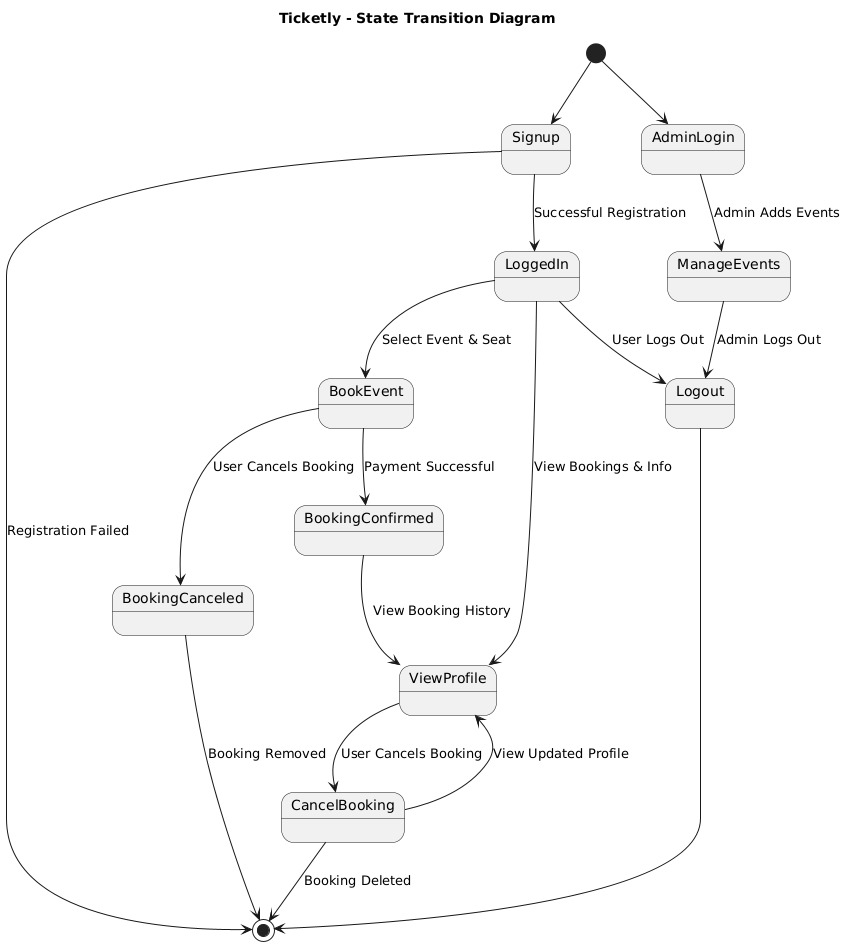
****

****

* + 1. **Component Diagram**

****

* + 1. **State Diagram**

****

## **6.0 Implementation Planning**

### **6.1 Implementation Environment**

* **Single vs Multiuser**: Ticketly is designed as a **multiuser** application where multiple users can interact with the system simultaneously. Users may include customers, ticketing agents, and admins.
* **GUI vs Non-GUI**: The application will have both:
  + **GUI-based Web Application**: Developed using **JSP, Servlets, and JSTL**, ensuring a dynamic web-based interface for users.
  + **Backend Processing (Non-GUI)**: The backend logic, including authentication filters and logging mechanisms, will run as background processes without direct user interaction.

### **6.2 Program/Modules Specification**

Ticketly consists of several key modules:

1. **User Authentication Module**
   * Implements login and registration using Servlets and JSP.
   * Uses **Authentication Filter** to ensure secure access.
2. **Ticket Booking Module**
   * Allows users to search for available tickets.
   * Uses JSTL SQL tags for database interactions to fetch and display results dynamically.
3. **Ticket Management Module**
   * Users can view, modify, or cancel their booked tickets.
   * Admins can manage ticket availability and pricing.
4. **Logging and Security Module**
   * Uses **Log Filter** to track user activities for security and debugging.
   * Implements session handling to ensure secure transactions.
5. **Database Module**
   * Stores user information, ticket details, and transaction history.
   * Uses **JSTL SQL Library** for database queries in the web layer.

### **6.3 Coding Standards**

To maintain code quality and readability, the following coding standards are followed:

### **1. Naming Conventions**

* **JavaScript/TypeScript Best Practices:**
  + Variables & functions: camelCase (e.g., getUserTickets ())
  + Constants: UPPER\_CASE\_SNAKE\_CASE (e.g., MAX\_TICKET\_LIMIT)
  + Component names: PascalCase (e.g., TicketBookingForm)
  + Files & folders: kebab-case (e.g., ticket-controller.js)
* **Database (MongoDB) Naming:**
  + Collection names: **plural & lowercase** (e.g., users, bookings)
  + Schema field names: **camelCase** (e.g., createdAt, ticketPrice)

### **2. Backend (Node.js & Express.js) Best Practices**

* **Project Structure:**
  + Follow **MVC (Model-View-Controller)** architecture for clean separation of concerns.
  + Example folder structure:

/Server

├── controllers/

├── models/

├── routes/

├── middleware/

├── config/

├── utils/

* **Error Handling:**
  + Use try-catch blocks and global error handling middleware.
  + Example:

try {

const ticket = await Ticket.findById(req.params.id);

if (!ticket) throw new Error("Ticket not found");

} catch (error) {

next(error);

}

* **Security Best Practices:**
  + Use **bcrypt.js** for password hashing.
  + Implement **JWT authentication** with expiration and refresh tokens.
  + Sanitize user input with express-validator to prevent **NoSQL injection**.
* **Database Operations:**
  + Use **Mongoose ORM** with schemas and validation.
  + Prefer **async/await** over callbacks for database queries.

### **3. Frontend (React.js) Best Practices**

* **Component Structure:**
  + Follow a **component-based architecture** with reusable UI components.
  + Example folder structure:

/client

├── src/

├── components/

├── pages/

├── context/

├── hooks/

├── utils/

* **State Management:**
  + Use **Context API or Redux** for managing global state.
  + Prefer useState and useEffect for local state management.

### **4. API Standards**

* **RESTful API Design:**
  + Use proper HTTP methods:
    - GET /tickets → Fetch all tickets
    - POST /tickets → Create a new ticket
    - PUT /tickets/:id → Update a ticket
    - DELETE /tickets/:id → Remove a ticket
* **Use Descriptive Status Codes:**
  + 200 OK → Successful request
  + 201 Created → Resource created
  + 400 Bad Request → Invalid input
  + 401 Unauthorized → Authentication failure
  + 500 Internal Server Error → Unexpected error

### **5. Code Documentation & Formatting**

* **Linting & Formatting:**
  + Use **ESLint** and **Prettier** for consistent code style.
  + Enforce **semi-colons, spacing, and indentation**.

# **7.0 Testing for Ticketly (MERN Stack)**

## **7.1 Testing Plan**

### **7.1.1 Objective:**

The primary goal of the testing plan is to ensure that **Ticketly** is stable, secure, and performs well under different conditions. The application will undergo **unit testing, integration testing, system testing, and user acceptance testing (UAT)** to validate its functionality across various modules.

### **7.1.2 Scope:**

* **Frontend (React.js)**: UI responsiveness, component behavior, form validation, and API integration.
* **Backend (Node.js + Express.js + MongoDB)**: API responses, authentication, data integrity, and security.
* **Database (MongoDB + Mongoose)**: Data consistency, query efficiency, and schema validation.
* **Performance & Security**: Load testing, SQL/NoSQL injection prevention, and authentication security.

### **7.1.3 Testing Levels:**

**Unit Testing** - Testing individual components, APIs, and functions.  
**Integration Testing** - Ensuring modules interact properly (e.g., React frontend ↔ Express backend ↔ MongoDB).  
**System Testing** - Verifying overall system behavior.  
**User Acceptance Testing (UAT)** - Real-world testing by users before final deployment.

## **7.2 Testing Strategy**

To ensure Ticketly meets functional and non-functional requirements, the following testing strategy is used:

### **7.2.1. Functional Testing**

* **User Authentication Tests**: Verify login, registration, JWT authentication, and role-based access control.
* **Ticket Booking Tests**: Ensure users can browse events, select seats, and complete the booking process successfully.
* **Payment Gateway Integration**: Validate payment success, failure, and refunds.
* **Admin Dashboard Tests**: Check admin functionalities like adding new events, managing bookings, and viewing analytics.

### **7.2.2. Non-Functional Testing**

* **Performance Testing**:
  + Load testing using **JMeter** to check app stability under high traffic.
  + Response time analysis (API response < 200ms under normal load).
* **Security Testing**:
  + **JWT Authentication Validation** to prevent unauthorized access.
  + **NoSQL Injection Testing** by checking API endpoints with malicious queries.
  + **CORS Policy Testing** to prevent unauthorized cross-origin requests.

### **7.2.3. Automation Testing**

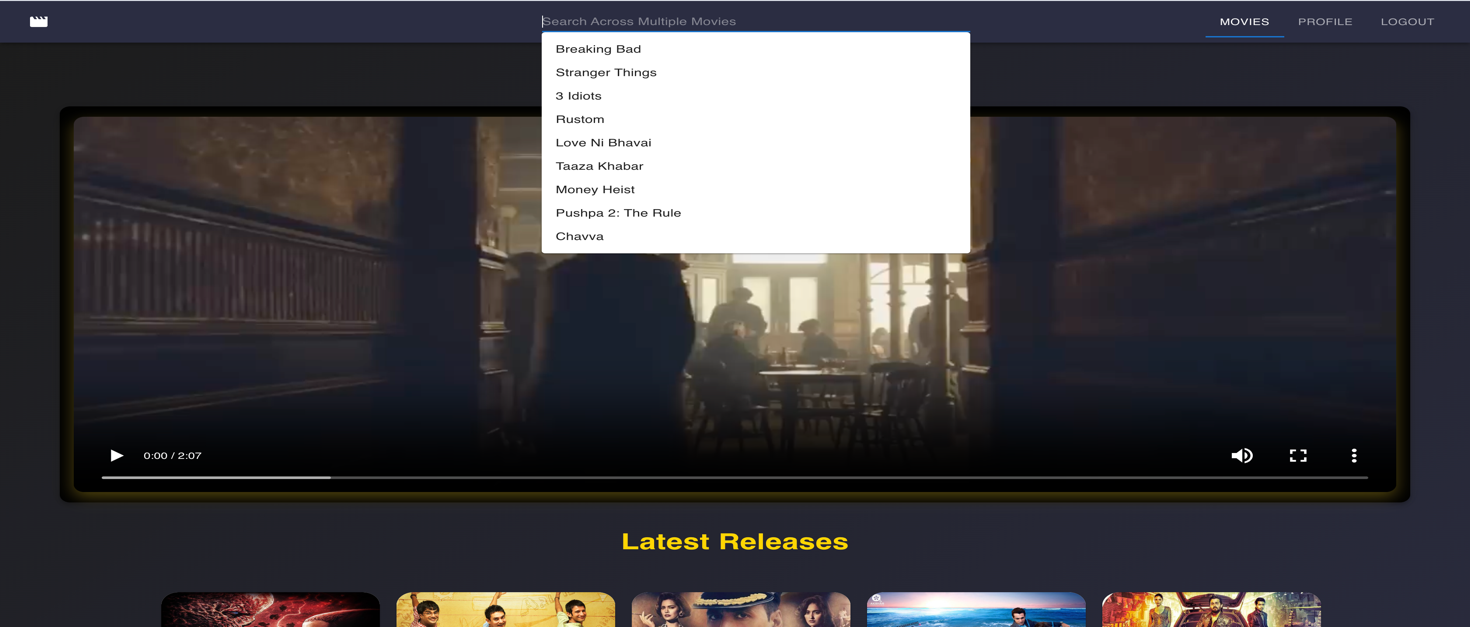
* Use **Jest + React Testing Library** for frontend unit testing.
* Use **Supertest + Postman** for API testing.
* CI/CD pipelines integrated with **GitHub Actions** to automate testing on every push.

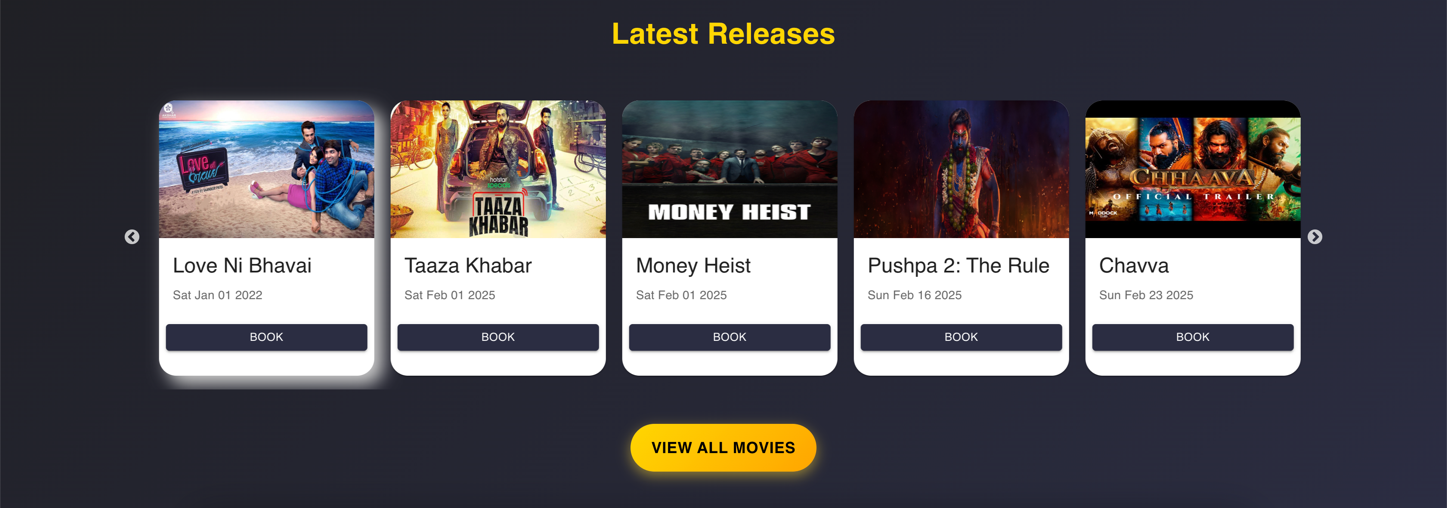
## **7.3 Testing Methods**

|  |  |
| --- | --- |
| **Testing Type** | **Description** |
| **Unit Testing** | Tests individual functions and components using Jest, Mocha, and Supertest. |
| **Integration Testing** | Ensures the frontend (React) and backend (Node.js) communicate correctly. |
| **System Testing** | Validates overall app performance, including UI, backend, and database interaction. |
| **Regression Testing** | Ensures new updates do not break existing features. |
| **Security Testing** | Tests against SQL/NoSQL injection, XSS, and authentication vulnerabilities. |
| **Load Testing** | Simulates high traffic using JMeter to test system stability. |
| **User Acceptance Testing (UAT)** | Final validation by real users before deployment. |

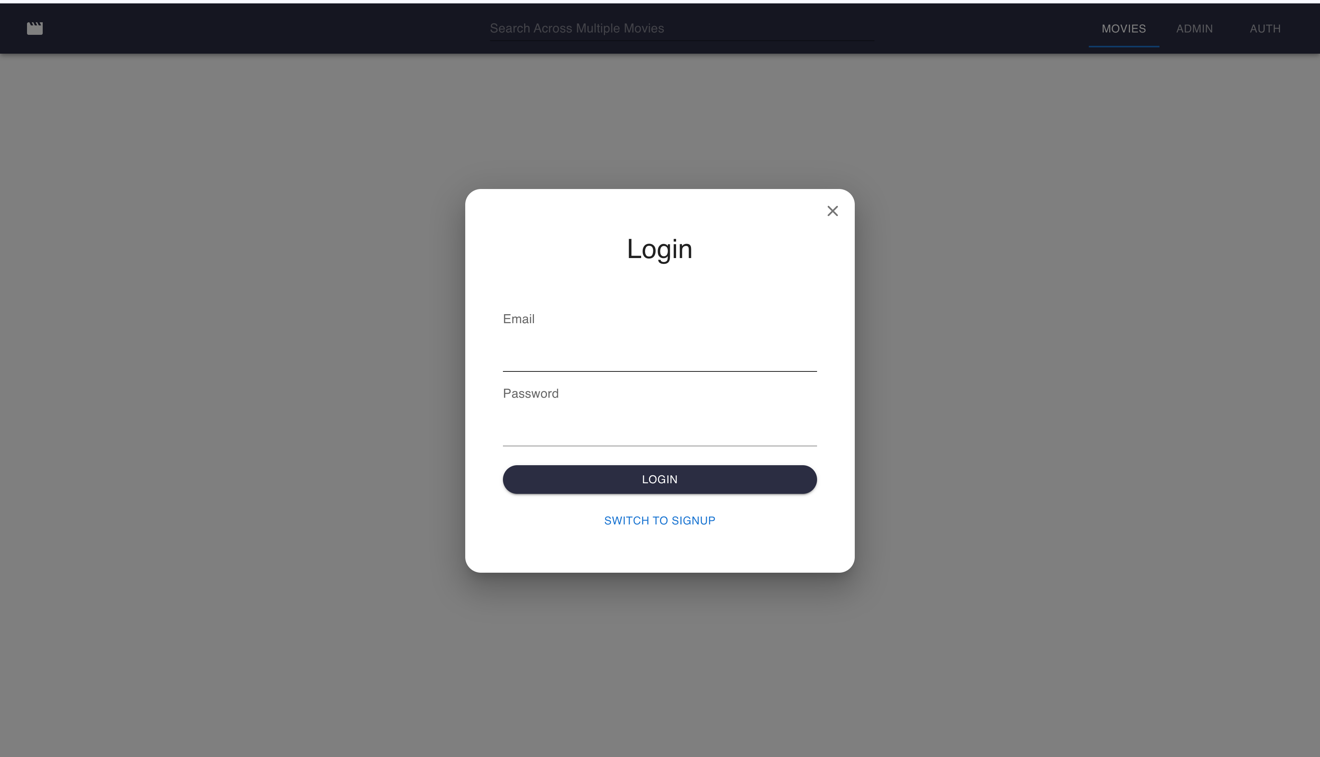
**8.0 User Manual**

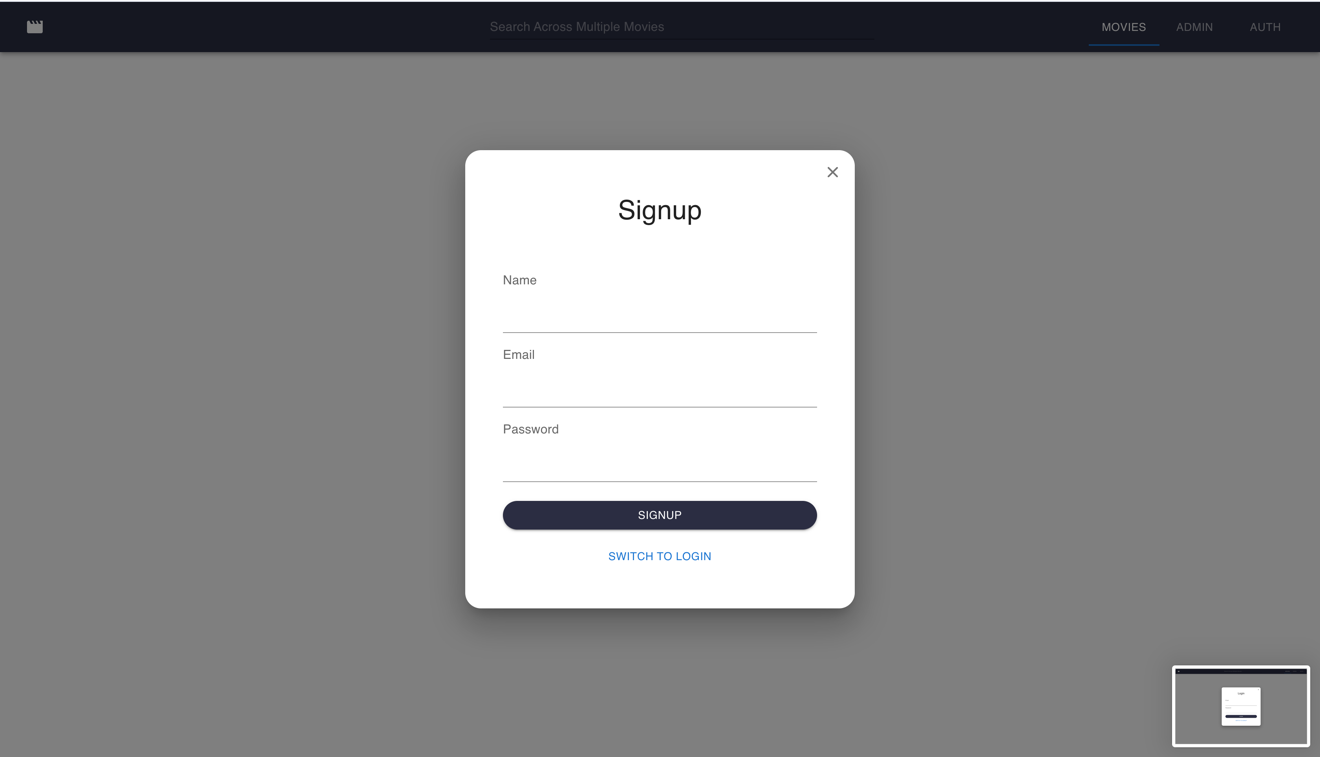
**Home Page:**

****

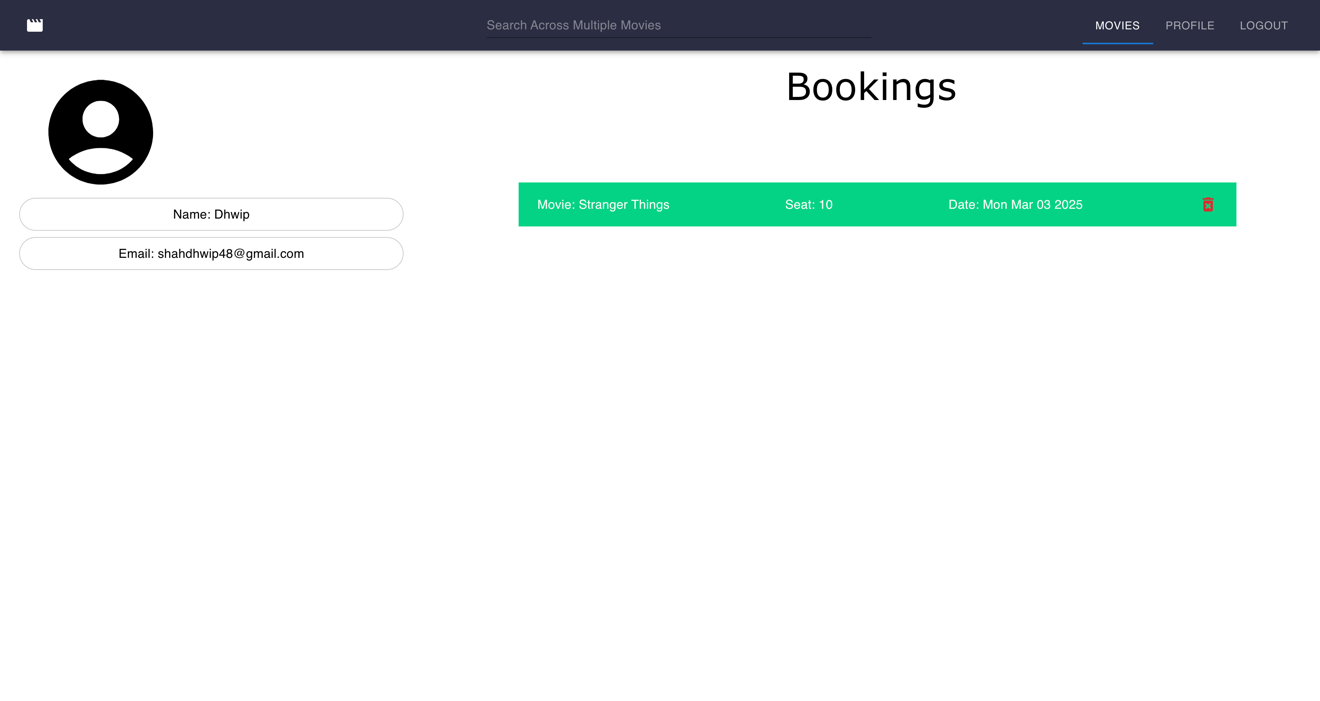
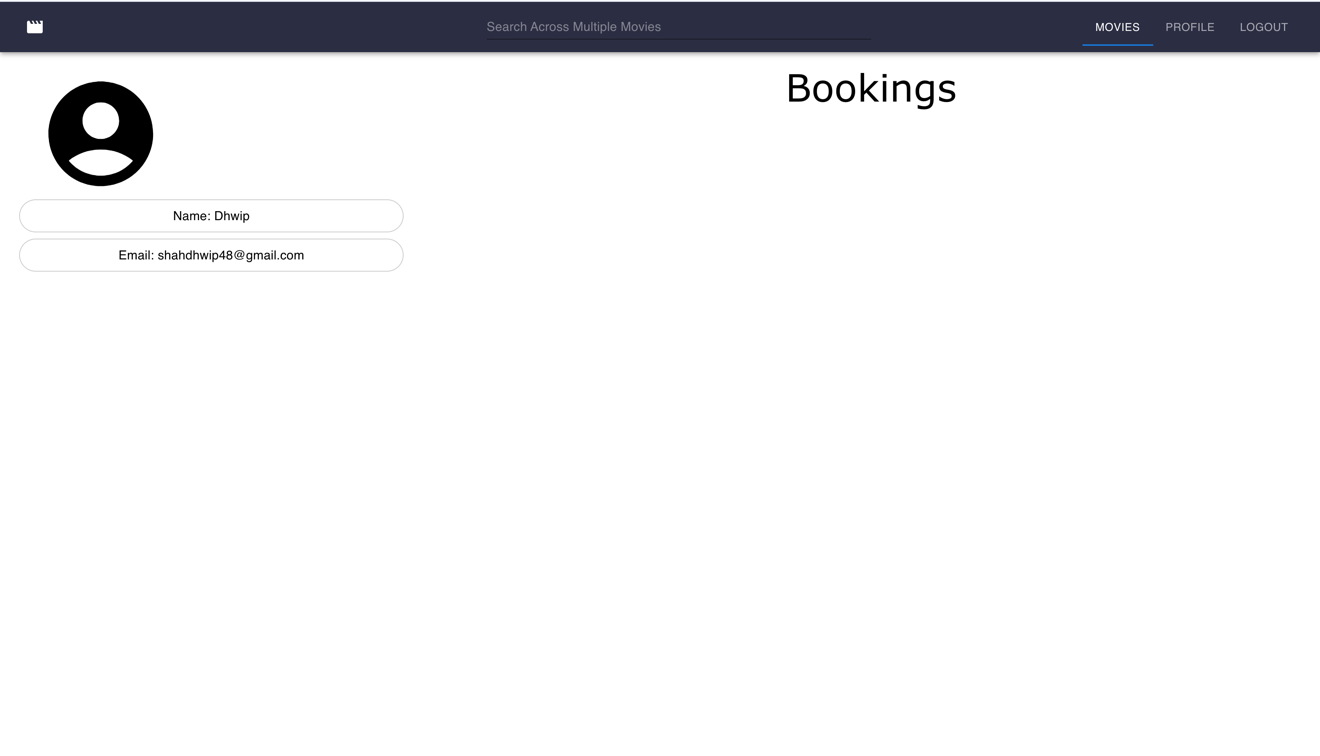
****

**User Login/Signup:**

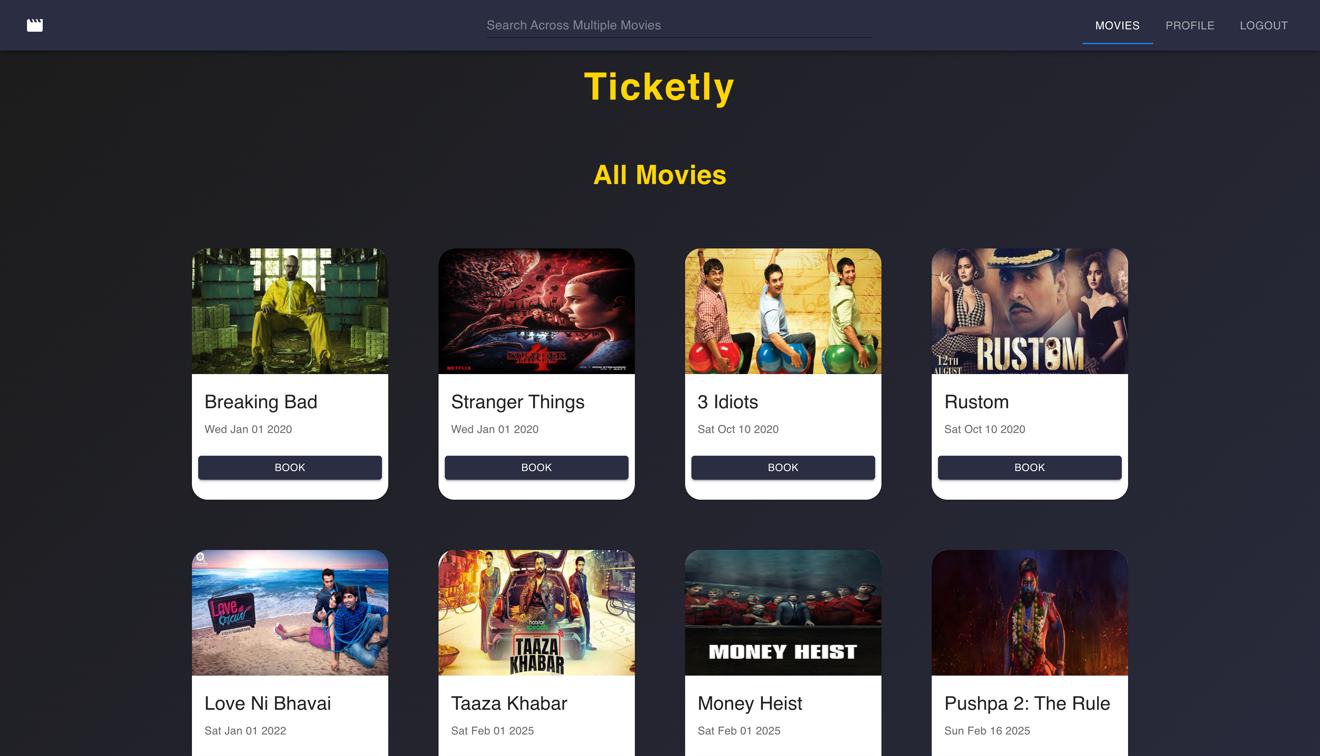




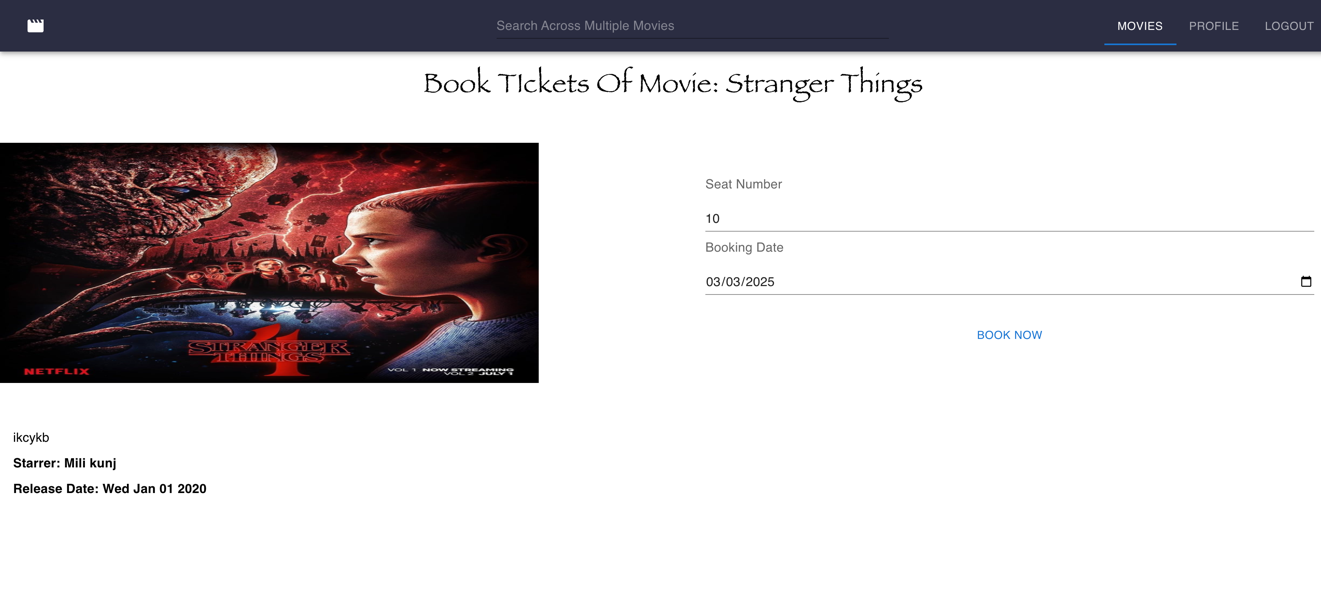
**User Dashboard:**



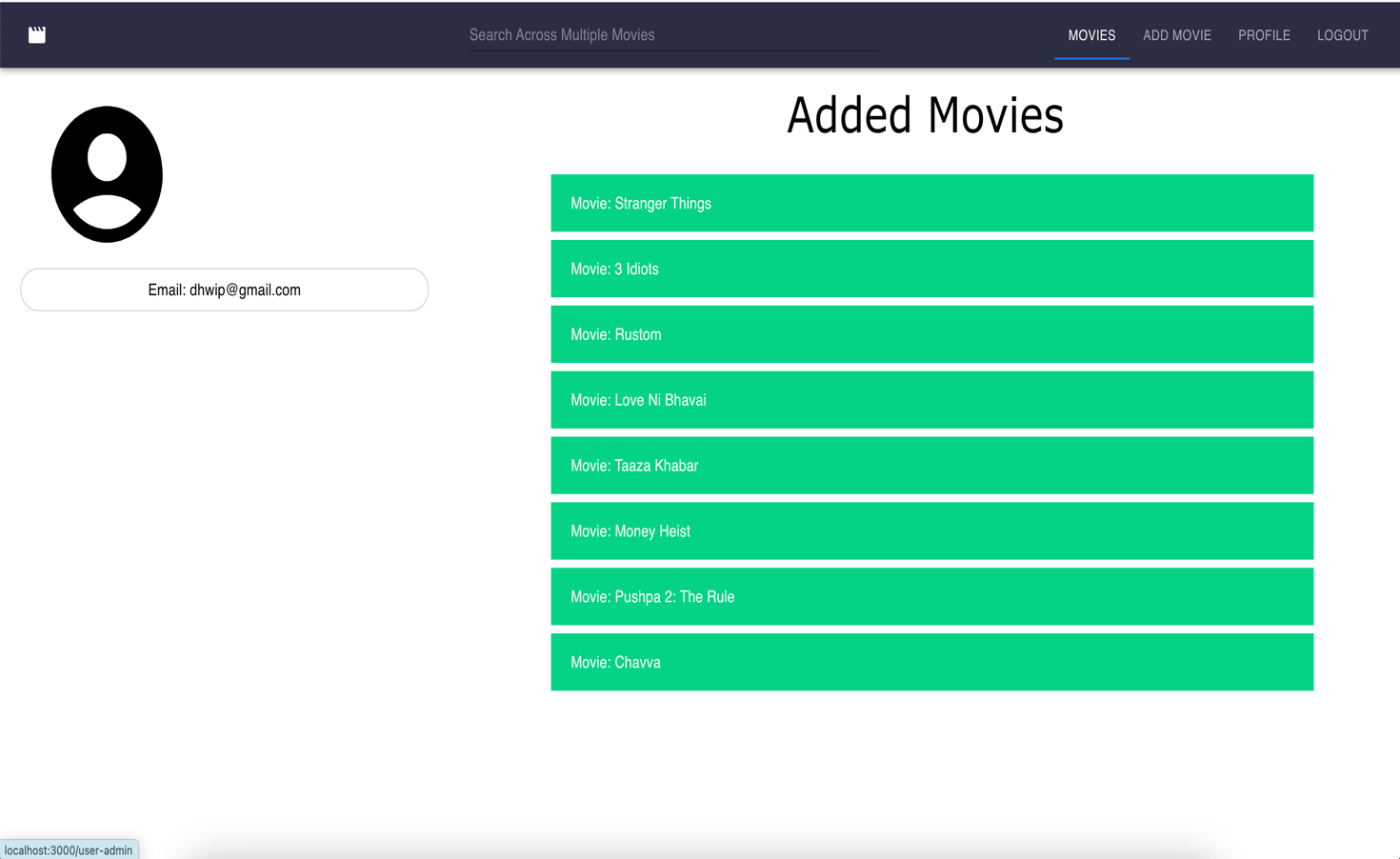
**View All Movies:**



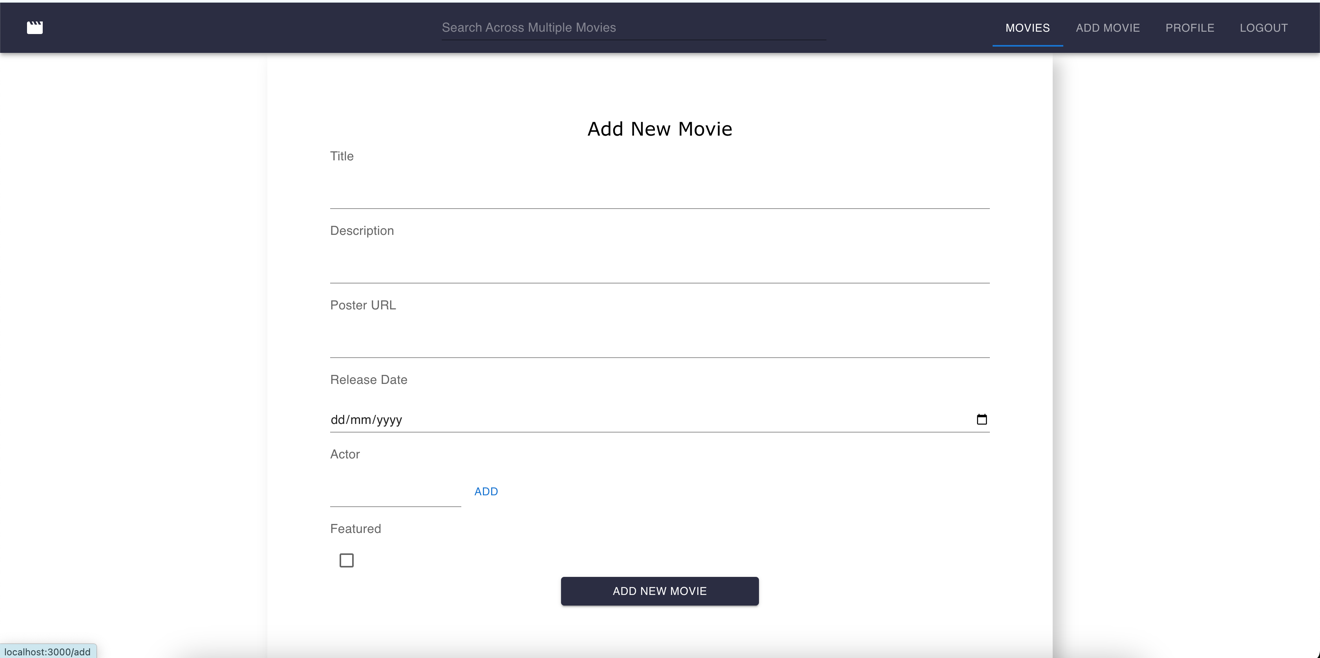
**Booking Page:**



Admin Dashboard:



Add Events/Movies:



## **9.0 Limitation and Future Enhancement**

### **Limitations:**

1. **Scalability Constraints:** The current version of Ticketly can handle a limited number of concurrent users. As the user base grows, performance issues may arise due to increased database load and API request handling.
2. **Limited Payment Options:** At present, Ticketly supports only a few payment gateways. Expanding to more options like digital wallets and UPI transactions would improve accessibility.
3. **Seat Selection Mechanism:** While users can select seats, the real-time seat blocking mechanism could be enhanced to prevent conflicts when multiple users try to book the same seat simultaneously.
4. **Security Aspects:** Basic authentication and authorization are implemented, but additional security layers such as OAuth, two-factor authentication (2FA), and fraud detection are yet to be integrated.
5. **Limited Analytics & Insights:** Ticketly currently lacks an advanced analytics dashboard for tracking user engagement, sales trends, and event popularity.

### **Future Enhancements:**

1. **Improved Performance & Scalability:** Implementing load balancing, caching mechanisms (Redis), and database sharding to optimize performance and handle more users.
2. **AI-Powered Recommendations:** Integrating machine learning to provide personalized movie or event recommendations based on user preferences and booking history.
3. **Enhanced Security Measures:** Adding two-factor authentication (2FA), CAPTCHA verification, and role-based access control to strengthen security.
4. **Blockchain-Based Ticketing:** Using blockchain to prevent ticket fraud and ensure ticket authenticity.
5. **PWA (Progressive Web App) Implementation:** Making Ticketly accessible as a PWA for better mobile experience, even in low network conditions.
6. **Partnership with Local Theaters & Events:** Allowing smaller event organizers and independent theaters to register and sell tickets on Ticketly.

## **10.0 Conclusion and Discussion**

### **10.1 Conclusions and Future Enhancement**

Ticketly has been successfully developed as a feature-rich online ticket booking platform using the MERN stack. The system allows users to browse events, book tickets, and make online payments in a seamless manner. With its user-friendly UI and secure transaction handling, Ticketly provides a convenient platform for moviegoers and event enthusiasts.

The project demonstrates the power of full-stack development using **MongoDB, Express.js, React.js, and Node.js**, ensuring a modern, fast, and scalable web application. However, future enhancements such as **real-time seat booking, AI-powered recommendations, and improved security measures** can further refine the platform and elevate the user experience.

### **10.2 Discussion**

#### **10.2.1 Self-Analysis of Project Viabilities**

Ticketly is a viable project as it addresses the growing demand for online ticket booking. The MERN stack ensures efficient data handling and responsiveness, making it suitable for real-world implementation. The platform's usability and potential scalability make it a strong candidate for further investment and enhancements. However, extensive testing, security audits, and performance optimizations are necessary before launching at a large scale.

#### **10.2.2 Problems Encountered and Possible Solutions**

1. **Handling Concurrent Bookings:**
   * **Problem:** Multiple users selecting the same seat simultaneously led to seat conflicts.
   * **Solution:** Implemented real-time seat reservation using Web Sockets to lock seats temporarily when selected.
2. **Optimizing API Performance:**
   * **Problem:** API response times were slow due to heavy database queries.
   * **Solution:** Used MongoDB indexing, caching mechanisms (Redis), and optimized query structures.
3. **Ensuring Secure Payments:**
   * **Problem:** Handling sensitive payment details securely.
   * **Solution:** Integrated trusted payment gateways with proper encryption and followed PCI DSS compliance guidelines.
4. **Managing State in React:**
   * **Problem:** Complex state management across multiple components.
   * **Solution:** Utilized Redux for centralized state management and efficient data flow.

#### **10.2.3 Summary of Project Work**

Ticketly successfully replicates a real-world ticket booking system with features such as **event browsing, ticket booking, payment integration, and user authentication.** The system is built on the MERN stack, leveraging **MongoDB for storage, Express.js for backend logic, React.js for the frontend UI, and Node.js for handling requests.**

The project has provided valuable insights into **full-stack development, state management, security implementation, and real-time data handling.** While Ticketly is functional, further improvements such as **AI-driven recommendations, blockchain-based security, and enhanced scalability** can make it even more competitive in the market.