Cinema-View Ticketing System

Software Design Specification

July 22, 2025

Group (individual submission)

Abdihakim Ahmed

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Summer 2025

Table of Contents

1. Introduction

This section will overview Movie Theater Ticketing System Software Requirement Specification (SRS). It describes the purpose, scopes, definitions, and references in developing the system.

1.1 Purpose

This Software Requirements Specification (SRS) aims to outline the requirements of the Movie Theater Ticketing System that will enable a user to view movies, select the time of their interest, choose a preferred seat, and buy a ticket online. The document shall be used by the development team, project stakeholders, testers, and the course instructor to realize and confirm the system requirements.

1.2 Scope

The Movie Theater Ticketing System is Internet-based software where the users can:
Check the movie schedule and show times.

- Choose the favorable showtimes and seats.
- Book tickets online and make the payments online.
- Get booked confirmation sent by e-mail/SMS.
- Book at the same price as individual customers.

The system is also intended to be convenient, safe, and dynamic enough to support multiple users anytime. Its primary goal is to eliminate the traditional challenge of booking tickets among consumers and administrative expenses incurred by theater managers.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
UI	User Interface
SRS	Software Requirements Specification
DBMS	Database Management System
OTP	One-Time Password
Admin	Theater Administrator with elevated privileges
CRUD	Create, Read, Update, Delete (operations)

1.4 References

- CS250-SRS Template (provided by instructor)
- Lecture Notes: “Scenarios and Use Cases” (William Y. Arms)

1.5 Overview

The remainder of this document includes:

- A **general description** of the system and its environment.
- Detailed **functional and non-functional requirements**.
- **Use case models**, object descriptions, and interface details.
- Specifications for external interfaces, database requirements, and design constraints.

2. General Description

The paragraph describes the factors overall that affect the environment and the Movie Theater Ticketing System. It is intended to develop background of the manner, in which the system functions and what it requires. It does not list some technical conditions, but preconditions them.

2.1 Product Perspective

Movie Theater Ticketing System is a web based and standalone application. Designed as a new product, it will however be able to be combined in future with other systems like digital signage, customer loyalty programs and external payment gateways. User frontend and theater administrator back end portal make up the software system. It will be reposing on a central relational database to store movies, showtime, seats booking status, user reservations and payment history. This application is a closed system and will not require any already existing applications although it is likely to simulate communication with external services e.g. SMS/email conformation, fake payment processors in development and testing.

2.2 Product Functions

The major functional components of the system include:

- **Movie Browsing**: Users can view currently playing and upcoming movies with details like genre, duration, ratings, and synopses.
- **Showtime Selection**: Users can select preferred showtimes by date and cinema hall.
- **Seat Selection**: Users can view a visual seating map and choose available seats.
- **Booking and Payment**: Users can confirm bookings and process payments.
- **Booking Management**: Users can view, cancel, or reschedule bookings.
- **Admin Functions**: Administrators can add/update movie listings, set showtimes, manage seats, and view reports.

2.3User Characteristics

- The primary users of this system are:
  a. General Users (Customers):
     - Basic digital literacy (can browse, fill forms, complete transactions).
     - Access system via desktop or mobile browsers.
     - May or may not have registered accounts.

  b. Administrators (Theater Staff):
     - Familiar with cinema scheduling and operations.

  - Trained to use the backend portal for daily updates.

2.4 General Constraints

- The system must be web-based and mobile-responsive.
- Real-time seat availability must be maintained to prevent double-booking.
- The system must support at least 100 concurrent users without performance degradation.
- Bookings cannot be processed after the movie showtime has started.
- System availability must be maintained at ≥ 99% uptime.

2.5 Assumptions and Dependencies

- The system assumes a stable internet connection for end users.
- A modern web browser (Chrome, Firefox, Safari) is available.
- The theater staff will have access to administrative credentials.
- Email/SMS APIs used for confirmation are functional and available.
- Payment processing will be simulated with placeholder logic during development.

3. Specific Requirements

In this section, the detailed software requirements that will be required in the design, implementation, and the testing of the Movie Theater Ticketing System is described. These requirements are trace, complete and verifiable. They are divided into external interface requirements, functional requirements, use cases and non-functional requirements etc.

3.1 External Interface Requirements

3.1.1 User Interfaces

Customer Web Interface:

- Responsive design compatible with desktops, tablets, and mobile devices.
- Navigation menus for Movies, Showtimes, Seats, Booking History.
- Login and registration forms with password recovery.
- Visual seat selection map with real-time seat availability.
- Confirmation page summarizing transaction details.

Admin Interface:

- Secure login with authentication.
- Dashboards for managing movies, showtimes, and seat maps.
- Admin reporting tools (e.g., total bookings, sales data).
- CRUD functions for movie entries, schedule slots, and system settings.

3.1.2 Hardware Interfaces

- The system will be hosted on cloud servers accessible via any internet-enabled device.
- No direct hardware dependency except for standard user devices (e.g., smartphones, laptops).

3.1.3 Software Interfaces

Database Interface:

- The backend will communicate with a relational database e.g., MySQL or PostgreSQL.
- APIs for CRUD operations related to movies, bookings, users, and payments.

Third-party Service Integration:

- Email/SMS API for sending confirmation.
- Payment API placeholder for transaction handling.

3.1.4 Communications Interfaces

- HTTP/HTTPS protocols for all client-server communication.
- RESTful API architecture for internal and external data exchange.
- Secure communication using TLS encryption (HTTPS).

3.2 Functional Requirements

3.2.1 Functional Requirement: Book Movie Ticket

3.2.1.1 Introduction

This feature allows a customer to book a ticket for a selected movie and showtime, complete with seat selection and payment.

3.2.1.2 Inputs

- Selected movie ID
- Selected date and showtime
- Selected seat numbers
- Payment method
- Customer authentication token (if logged in)

3.2.1.3 Processing

- Validate selected showtime is still available and in the future.
- Check real-time seat availability and lock chosen seats.

- Process payment via the mock gateway.
- Generate booking confirmation record and update database.
- Send confirmation email/SMS to customer.

3.2.1.4 Outputs

- Booking confirmation screen with ticket details (movie name, seat numbers, date/time).
- Confirmation email/SMS.
- Entry created in customer's booking history.

3.2.1.5 Error Handling

- If selected seats are already booked: notify user and prompt reselection.
- If payment fails: cancel booking and release seat lock.
- If session expires: redirect user to login or session start screen.

3.2.2 Functional Requirement or Feature

3.2.2.1 Introduction

This feature allows customers to cancel a previously booked ticket before the showtime and receive a confirmation of cancellation.

3.2.2.2 Inputs

- Booking ID
- User authentication token

3.2.2.3 Processing

- Verify booking exists and is associated with the logged-in user.
- Confirm that the showtime has not yet started.
- Update database to release the booked seats.
- Update booking status to "Cancelled".
- Send cancellation notification via email/SMS.

3.2.2.4 Outputs

- Cancellation confirmation screen.
- Updated booking status in the user's booking history.
- Notification to user (email/SMS).

3.2.2.5 Error Handling

- If the showtime has already passed: notify the user that cancellation is not possible.
- If the booking ID is invalid: prompt user to recheck booking reference.

3.3 Use Cases

This section describes **three core use cases** for the Movie Theater Ticketing System. Each use case includes the name, actors, flow of events, and entry conditions.

3.3.1 Use Case 1: Book Ticket

- **Actor(s):** Customer (Authenticated or Guest)
- **Purpose:** Allows a customer to select a movie, choose a showtime, pick seats, and pay for a booking.

Flow of Events:

1. User browses movies and selects a title.
2. System displays available showtimes.
3. User selects preferred time and seating.
4. System locks selected seats.
5. User provides payment info and confirms.
6. System processes booking and sends confirmation.
7. Seats are marked as booked; user receives email/SMS.

Entry Conditions:

- User is connected to the internet.
- Selected showtime is available.
- Seats chosen are not already booked.

3.3.2 Use Case 2: Cancel Ticket

- **Actor(s):** Customer (Authenticated)
- **Purpose:** Enables a user to cancel a ticket and release reserved seats.

Flow of Events:

1. User logs in and navigates to booking history.
2. Selects a specific booking and chooses "Cancel."
3. System verifies the booking and showtime validity.
4. System releases seats and updates booking record.
5. Cancellation confirmation is shown and emailed to user.

Entry Conditions:

- User is authenticated.
- Cancellation request is made before the movie’s start time.

3.3.3 Use Case 3: Add Movie & Showtime (Admin)

- **Actor(s):** Administrator
- **Purpose:** Allows an admin to update movie listings and assign showtimes.

Flow of Events:

1. Admin logs in to the backend portal.
2. Navigates to “Manage Movies.”
3. Inputs new movie details (title, genre, description, duration).
4. Navigates to “Manage Showtimes.”
5. Selects movie and assigns date/time and hall location.
6. System validates entries and stores the new schedule.
7. Movie and showtime become visible to customers.

Entry Conditions:

- Admin is authenticated.
- Required details (movie info, time slot) are complete.

3.4 Classes / Objects

3.4.1 User

3.4.1.1 Attributes

- userID
- username
- password
- email
- phoneNumber
- role (Customer/Admin)
- registrationDate

3.4.1.2 Functions

- Login()
- Logout()
- ViewProfile()
- UpdateProfile()

3.4.2 Movie

Attributes

- movieID
- title

- genre
- duration
- rating
- description

Functions

- AddMovie()
- UpdateMovie()
- ViewMovieDetails()

3.4.3 Class: `Showtime`

Attributes

- showtimeID
- movieID
- hallNumber
- startTime
- endTime
- availableSeats

Functions

- CreateShowtime()
- UpdateShowtime()
- CheckSeatAvailability()

3.5 Non-Functional Requirements

These are quality-related requirements that define system performance, security, and operational standards.

3.5.1 Performance

- The system must support a minimum of **100 concurrent users** without slowing down response times.
- Page load time should not exceed **2 seconds** under normal conditions.

3.5.2 Reliability

- The system should have **≥ 99% uptime**.
- Scheduled maintenance should be communicated to users at least 24 hours in advance.

3.5.3 Availability

- The system must be available **24/7** except during pre-notified maintenance.
- Any downtime should not exceed **1 hour per week**.

3.5.4 Security

- Passwords must be encrypted using industry-standard hashing algorithms.
- Sensitive transactions (login, payments) must be done over **HTTPS**.
- User roles (Customer vs Admin) must be strictly enforced through access control.

3.5.5 Maintainability

- Codebase must be modular to support updates with minimal system interruption.
- All backend logic should be thoroughly documented.

3.5.6 Portability

- The system must work across all modern web browsers e.g Chrome, Firefox, Safari, Edge).
- Should be deployable on common web servers e.g., Apache, NGINX.

3.6 Inverse Requirements

Inverse requirements specify what the system **must not** do. These help clarify boundaries and avoid ambiguity.

- The system **must not allow** booking of seats that are already reserved.
- The system **must not accept** bookings for showtimes that have already started or expired.
- Users **must not be able** to cancel a ticket after the showtime has begun.
- Admins **must not access** customer payment data directly.
- The system **must not store** plaintext passwords.

3.7 Design Constraints

This section defines technical and organizational constraints that affect the system design.

- The system must use **relational database technology** e.g., MySQL or PostgreSQL.
- The frontend must be built using **standard web technologies** e.g HTML, CSS, JavaScript).
- Backend logic must be developed using **Python (Django/Flask)** or **Java (Spring Boot)**.
- Development must follow **MVC (Model-View-Controller)** architecture.
- The system must be developed using **open-source libraries** only.
- The application must comply with **FERPA and GDPR** standards for data handling.

3.8 Logical Database Requirements

The system will utilize a structured relational database to manage all core data. Key database requirements include:

- **Data Integrity**: All foreign key constraints must be enforced to prevent orphan records e.g., bookings without users.
- **Transactions**: Booking and payment processes must use **atomic transactions** to avoid partial data commits.
- **Normalization**: Database design must follow at least **3rd Normal Form** to reduce redundancy.
- **Data Retention**:
  - Booking history must be stored for **at least 12 months**.
  - Logs related to login attempts and errors must be retained for **6 months**.
- **Data Recovery**: A daily backup process must be in place with at least **7-day retention**.

3.9 Other Requirements

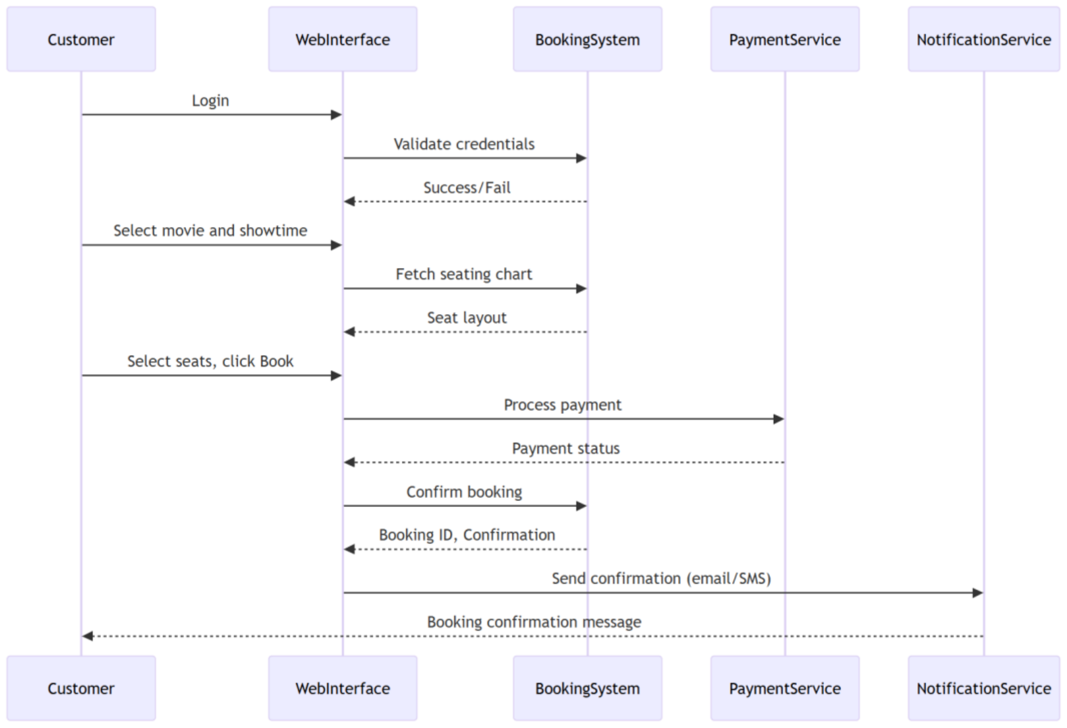
This section captures additional requirements that do not fall neatly into prior categories.

- **Multi-language support** should be considered for scalability (English required; others optional).
- **Accessibility compliance** with WCAG 2.1 guidelines
- System should log:
  - Login attempts that is successful/failed
  - Payment transactions
  - Booking status changes (booked, canceled)
- **Audit Trails** for administrative actions, movie/showtime edits should be maintained.
- The system must include a **Help & Support** section with FAQs and contact forms.

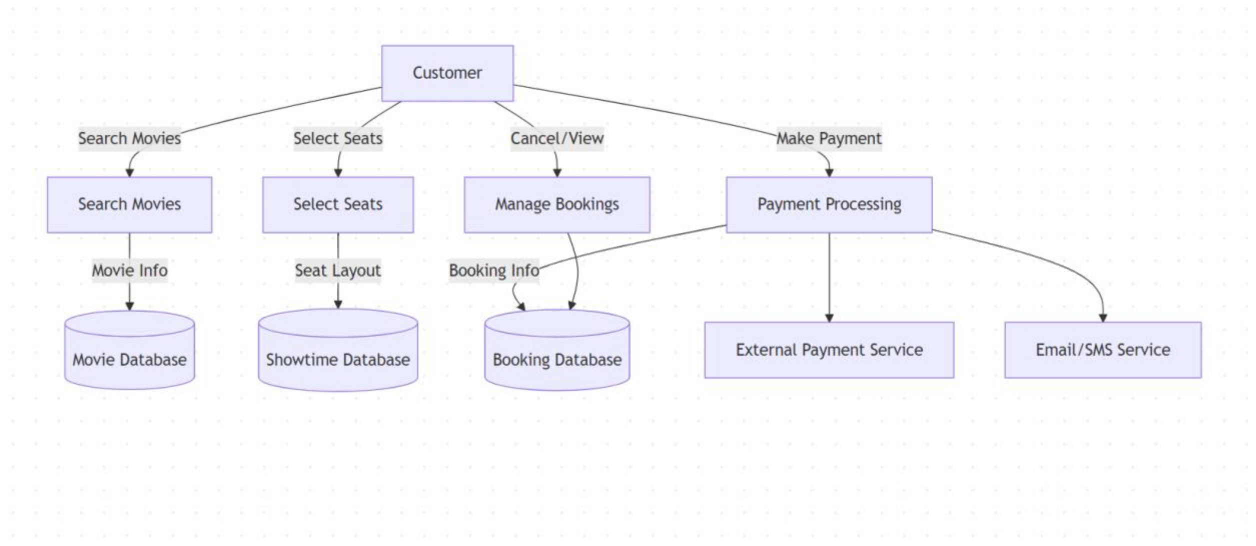
4. Analysis Models

This section includes the primary modeling diagrams used during requirements analysis. Each model is accompanied by a short narrative description and ties directly to the specific requirements detailed in Section 3.

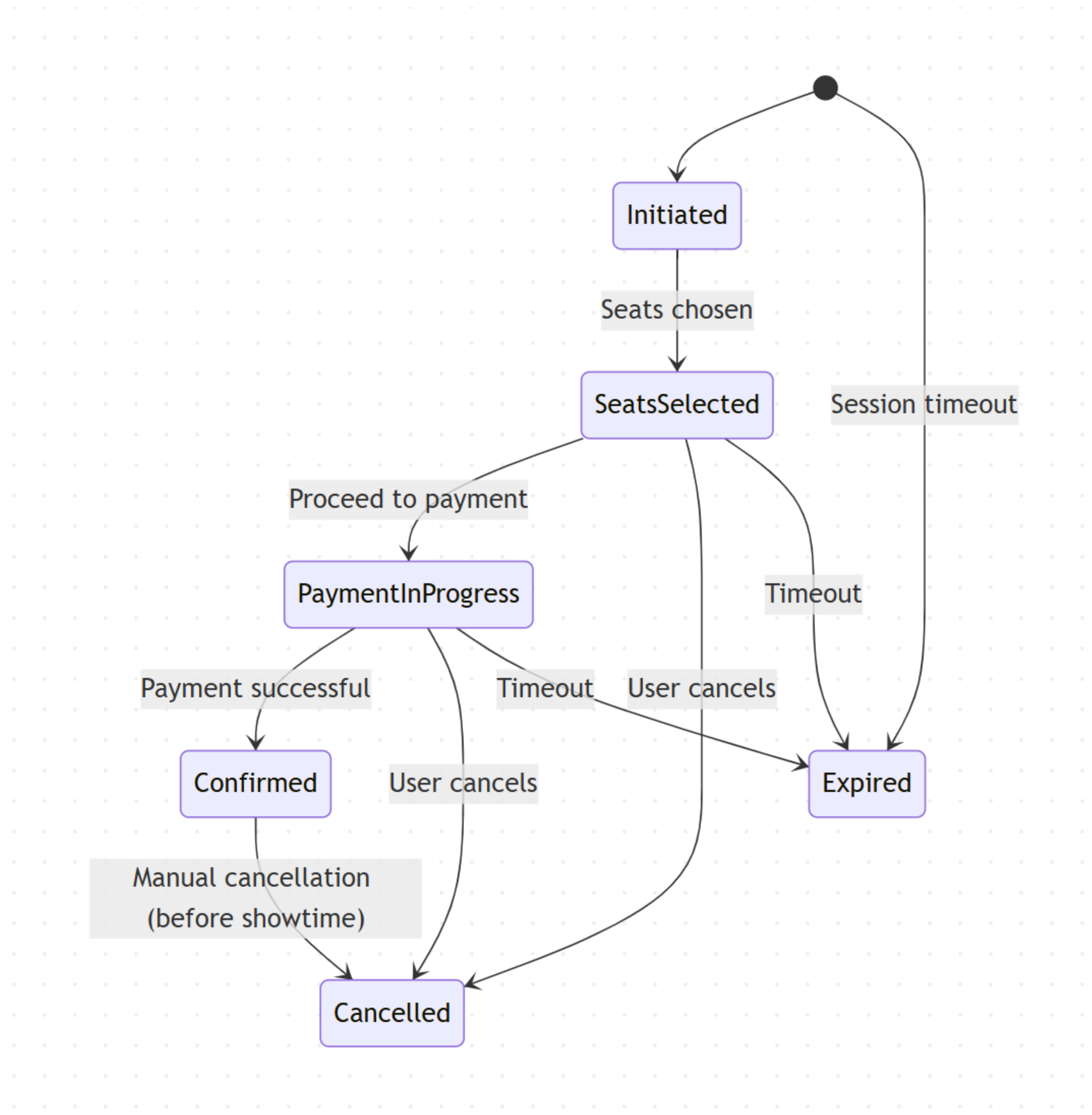
4.1 Sequence Diagrams



4.3 Data Flow Diagrams (DFD)



4.2 State-Transition Diagrams (STD)



5. Development Plan and Timeline

This section outlines the development plan for the Cinema-View Ticketing System, breaking the work into manageable tasks and assigning responsibilities to team members. The timeline details key milestones, ensuring timely delivery of each phase.

5.1 Partitioning of Tasks

- **Phase 1: System Design and Setup (Weeks 1-2)**
    - Task 1.1: System Architecture

  - Task 1.2: Database Design
  - Task 1.3: UI/UX Design
- **Phase 2: Frontend Development (Weeks 3-5)**
  - Task 2.1: UI Development
  - Task 2.2: Seat Selection Interface
  - Task 2.3: Responsive Design
- **Phase 3: Backend Development (Weeks 6-8)**
  - Task 3.1: User Authentication and Management
  - Task 3.2: Booking Logic
  - Task 3.3: Admin Portal
- **Phase 4: Testing and Quality Assurance (Weeks 9-10)**
  - Task 4.1: Functional Testing
  - Task 4.2: Security and Performance Testing
  - Task 4.3: Bug Fixes and Optimization
- **Phase 5: Deployment and Post-Deployment (Weeks 11-12)**
  - Task 5.1: Deployment on Cloud Servers
  - Task 5.2: Post-Deployment Testing
  - Task 5.3: Admin Training

5.2 Team Member Responsibilities

- **Abdihakim Ahmed**: Leading Phase 1 and Phase 3 (System Architecture, Database Design, Backend Development)
- **Sebastian Curran**: Leading Phase 2 and Phase 4 (Frontend Development, Testing)
- **Guled Hussein**: Leading Phase 3 and Phase 5 (Admin Portal Development, Deployment)

5.3 Timeline

- **Weeks 1-2**: System Architecture, Database Design, UI/UX Design
- **Weeks 3-5**: Frontend Development (UI, Seat Selection, Responsive Design)
- **Weeks 6-8**: Backend Development (User Authentication, Booking Logic, Admin Portal)
- **Weeks 9-10**: Functional and Performance Testing, Bug Fixes
- **Weeks 11-12**: Deployment, Post-Deployment Testing, Admin Training

6. Appendices

A.1 Appendix 1: Example User Interaction

1. User browses available movies.
2. User selects a movie and showtime.
3. User chooses seats from the seating chart.
4. User enters payment details and confirms.
5. User receives a confirmation email and a digital ticket.
6. The booking is saved in the user's account dashboard.

A.2 Appendix 2: Glossary of Terms

- **Booking**: A confirmed reservation for selected movie seats.
- **Showtime**: A scheduled time at which a movie is shown.
- **Admin**: A user with elevated privileges to manage movies, showtimes, and reports.
- **Payment Gateway**: The service (mock or real) that processes user payments.
- **Seat Map**: A graphical interface showing available and booked seats.
- **CRUD**: Create, Read, Update, Delete – fundamental operations used to manage data.

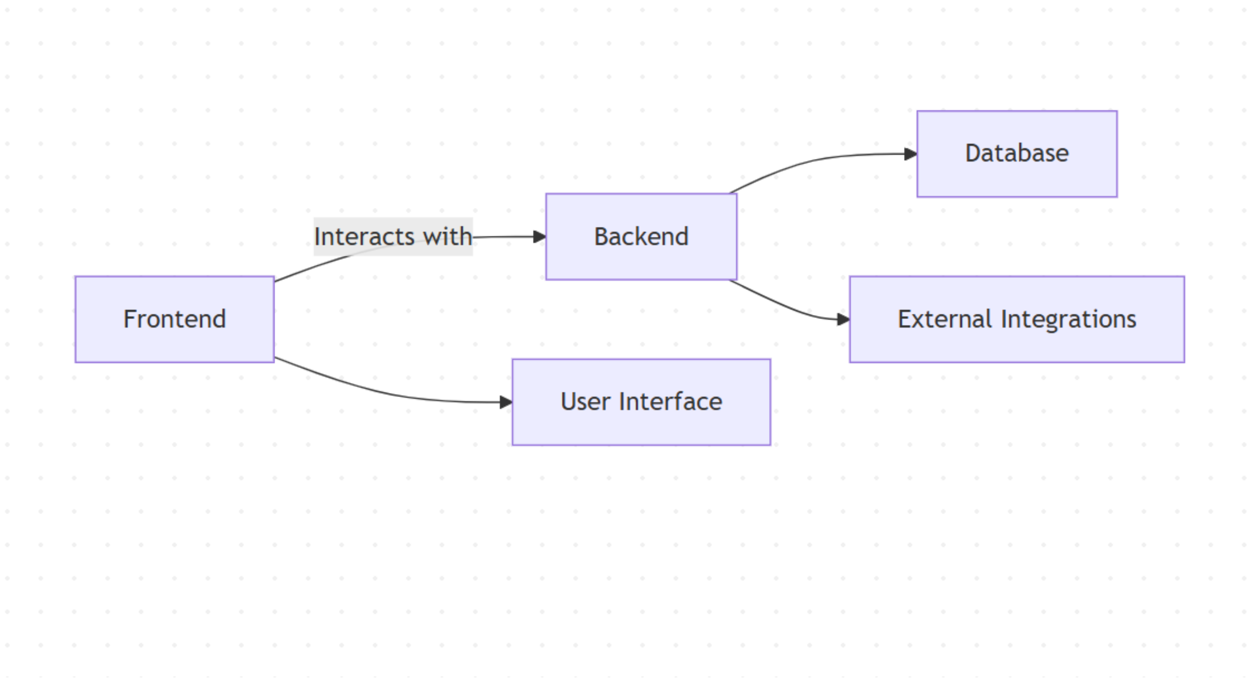
7. Change Management Process

The **Change Management Process** ensures that any changes required during the project lifecycle are systematically handled. Each change is requested using a **Change Request Form (CRF)**, which includes the rationale, impact analysis, and review by a **Change Review Committee (CRC)**. Any accepted changes are added to the **SRS document** with an entry in the revision history.

8. Software Architecture Overview

This section outlines the software architecture of the Cinema-View Ticketing System, which will be built using a multi-tiered architecture to separate the concerns of data management, business logic, and user interface. It will follow a **Model-View-Controller (MVC)** design pattern, ensuring scalability, maintainability, and modularity.

8.1 Architectural Diagram



8.2 Description of Architecture Components

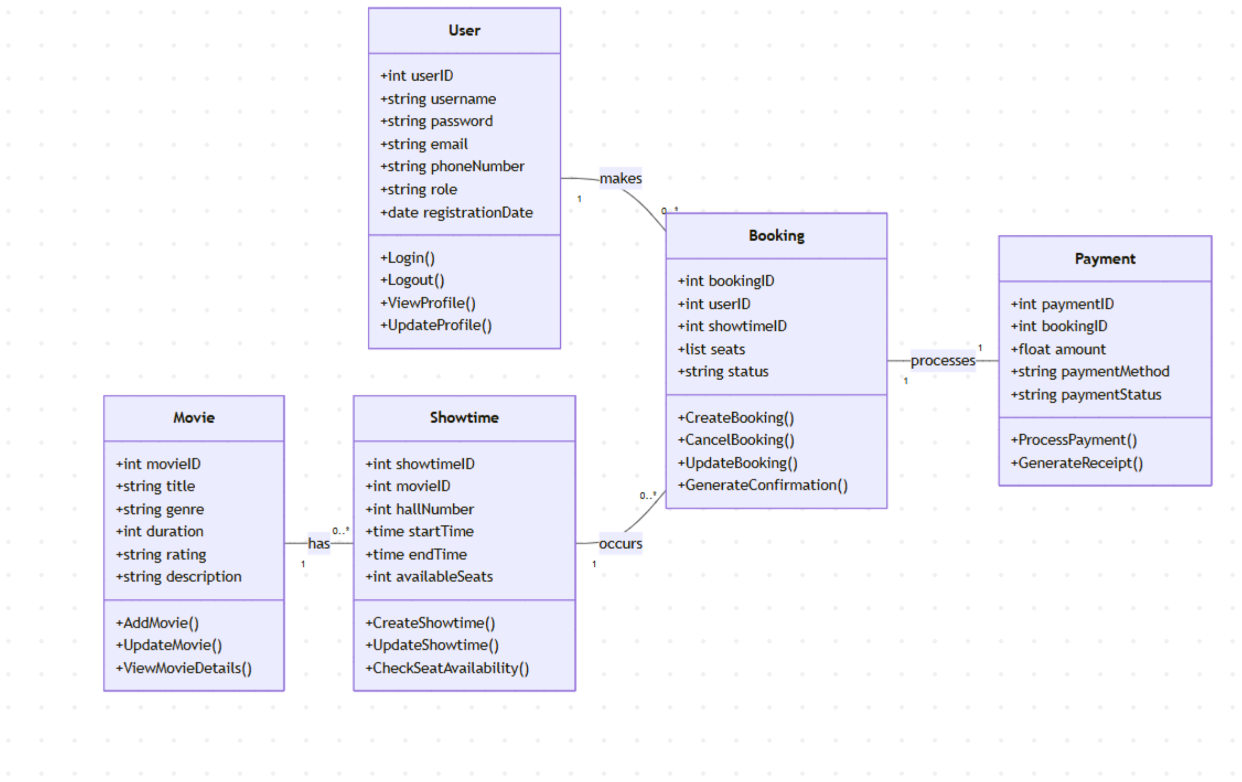
- **Frontend (Client Side)**: Provides the user interface for customers and admins.

- Technologies: React.js/Vue.js, HTML5, CSS3, JavaScript.
- Responsibilities: Display movies, handle seat selection, process bookings.

- **Backend (Server Side)**: Manages business logic, user authentication, and database interaction.
  - Technologies: Node.js with Express.js, Django, or Spring Boot.
  - Responsibilities: Handle booking logic, authentication, and communication with the database.
- **Database**: Stores the system's core data.
  - Technologies: MySQL, PostgreSQL.
  - Responsibilities: Manage data integrity, store user and booking information.
- **External Integrations**:
  - **Email/SMS API**: Sends booking confirmations and reminders.
  - **Payment Gateway**: Processes payments securely.

9. UML Class Diagram

The UML class diagram illustrates the major classes involved in the Cinema-View Ticketing System and their relationships.



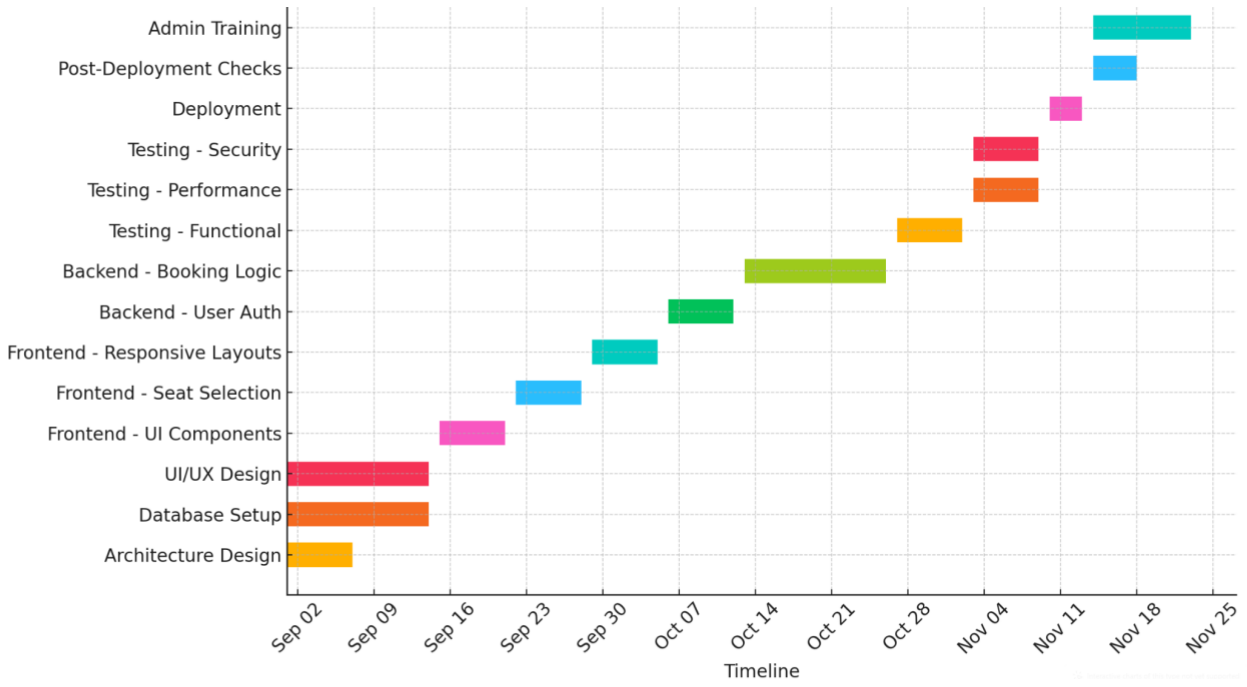
10. Development Plan and Timeline (Continued)

This section continues the detailed planning for the system's design, ensuring that architecture and development follow a systematic and scalable approach.

10.1 Timeline and Phases

- **Weeks 1-2**: Architecture design, database setup, and UI/UX development.
- **Weeks 3-5**: Frontend development (UI, seat selection, responsive design).
- **Weeks 6-8**: Backend development (user authentication, booking logic).
- **Weeks 9-10**: Testing (functional, performance, security).
- **Weeks 11-12**: Deployment, post-deployment checks, and admin training.

Gantt Chart



11. Testing and Quality Assurance (QA)

This section outlines the Testing and Quality Assurance (QA) process to ensure that the Cinema-View Ticketing System operates as expected and meets the specified functional, non-functional, and security requirements.

11.1 Types of Testing

1. **Functional Testing**:
    - **Purpose**: To verify that all functional requirements outlined in the SRS are implemented correctly.
    - **Test Scenarios**:
        - Ensure users can browse movies, select showtimes, and book tickets.
        - Test real-time seat availability.
        - Validate successful booking and payment processing.
2. **Security Testing**:
    - **Purpose**: To identify vulnerabilities and ensure sensitive data is protected.

- **Test Scenarios**:
  - Verify role-based access control (Admin vs. Customer).
  - Ensure passwords are encrypted.
  - Test for vulnerabilities like SQL injection, XSS, and CSRF.

3. **Performance Testing**:
   - **Purpose**: To evaluate system performance under varying loads.
   - **Test Scenarios**:
     - Stress test the system with 100+ concurrent users.
     - Ensure page load times are under 2 seconds.
     - Test database performance for large data sets.
4. **Usability Testing**:
   - **Purpose**: To ensure the system is user-friendly.
   - **Test Scenarios**:
     - Test the navigation flow for ease of use.
     - Ensure seat selection is intuitive and functional on all devices.
5. **Regression Testing**:
   - **Purpose**: To ensure new code changes do not break existing functionality.
   - **Test Scenarios**:
     - Re-run previous tests to confirm the integrity of features after updates.
6. **Acceptance Testing**:
   - **Purpose**: To verify that the system meets the original requirements.
   - **Test Scenarios**:
     - Ensure that all requirements are met.
     - Confirm that the system is ready for deployment.

11.2 Testing Methodologies and Tools

1. **Automated Testing**:
   - **Tools**:
     - **Selenium**: Automates UI tests.
     - **JUnit**: For backend unit tests.
     - **Postman**: For testing RESTful APIs.
2. **Manual Testing**:
   - **Tools**:
     - **Browser Developer Tools**: For debugging and inspecting elements.
     - **Figma**: For reviewing UI/UX designs.
3. **Performance Testing Tools**:
   - **Tools**:
     - **JMeter**: For load testing.
     - **New Relic**: For monitoring system performance.
4. **Security Testing Tools**:
   - **Tools**:
     - **OWASP ZAP**: For vulnerability scanning.
     - **Burp Suite**: For security testing.

11.3 Testing Schedule

Week	Testing Type	Tasks
9-10	Functional Testing	Test core functionalities such as booking and seat selection.
9-10	Security Testing	Perform vulnerability scans and address security issues.
10	Performance Testing	Stress test the system and evaluate database performance.
10	Usability Testing	Conduct usability tests and gather user feedback.
10-11	Regression Testing	Re-test core functionalities after code updates to ensure no new issues.

11.4 Bug Tracking and Reporting

The development team will use a bug tracking system to document and prioritize issues discovered during the testing phases. The following tools will be used:

- **JIRA**: For creating, managing, and tracking bugs and tasks throughout the development process.
- **GitHub Issues**: For smaller, project-specific issues that arise during development or testing.

Once bugs are identified, the development team will prioritize them based on severity (e.g., Critical, High, Medium, Low) and assign them to appropriate team members for resolution.

12. Deployment and Maintenance Plan

The Deployment and Maintenance Plan outlines the steps for deploying the Cinema-View Ticketing System to a live production environment and the ongoing maintenance strategy to ensure its continued performance, security, and user satisfaction.

12.1 Deployment Strategy

The deployment of the Cinema-View Ticketing System will follow a phased approach to ensure minimal disruption and smooth transition from development to production. The plan includes staging environments for testing, and rollback strategies in case of issues.

1. **Development Environment**:
    - **Purpose**: Used for initial development and unit testing.
    - **Tools**: Local environments with Docker or virtual machines to simulate production conditions.
    - **Deployment Process**: Code is pushed to the repository, and initial testing is performed on local servers before moving to staging.
2. **Staging Environment**:
    - **Purpose**: A replica of the production environment used for final testing, including user acceptance testing (UAT).

- **Tools**: The staging environment mirrors the production environment, using the same web server and database configurations.
- **Deployment Process**: Code is deployed to the staging environment, where final testing and bug fixes occur. This is the last chance to test the system before production deployment.

3. **Production Environment**:
    - **Purpose**: The live environment where users interact with the system.
    - **Tools**: The production environment will use cloud infrastructure (e.g., AWS, Google Cloud, or Azure) with CDN for performance and load balancing to ensure high availability.
    - **Deployment Process**: After passing testing and getting approval from stakeholders, the system will be deployed to production using continuous integration/continuous deployment (CI/CD) tools like Jenkins or GitHub Actions.

12.2 Ongoing Maintenance

After the system is deployed to production, ongoing maintenance will be required to ensure the system remains functional, secure, and up-to-date.

12.2.1 Maintenance Activities

1. **System Monitoring**:
    - Continuous monitoring of system performance, uptime, and error rates will be performed using tools like New Relic or Prometheus.
    - Alerts will be set up to notify the development team of any issues such as high CPU usage, slow response times, or downtime.
2. **Bug Fixes and Updates**:
    - Bugs identified after deployment will be tracked in JIRA or GitHub Issues.
    - A patching cycle will be established for minor updates (e.g., security patches, bug fixes) to be deployed at regular intervals (e.g., bi-weekly).
    - Feature updates and enhancements will be deployed periodically, based on customer feedback and system performance.
3. **Security Maintenance**:
    - Regular security audits will be conducted to ensure that the system is protected against new vulnerabilities (e.g., SQL injection, XSS, CSRF).
    - Security patches will be applied promptly to prevent any exploits.
4. **Database Maintenance**:
    - Database optimization will be performed periodically to ensure efficient query performance.
    - Data backups will be taken on a daily basis, with a 7-day retention period for disaster recovery.
5. **User Support and Feedback**:
    - User feedback will be gathered regularly via surveys or feedback forms to assess satisfaction and identify areas for improvement.
    - A Help Desk or FAQ section will be made available for users to get assistance.

12.2.2 Maintenance Schedule

Activity	Frequency	Tools/Methods
System Monitoring	Continuous (24/7)	New Relic, Datadog
Bug Fixes and Updates	Bi-weekly (for minor fixes)	JIRA, GitHub Issues
Security Patches	As required (based on threat)	Manual update or CI/CD pipeline
Database Optimization	Monthly	Query optimization, database indexing
Data Backups	Daily	Cloud backup service
User Support	Ongoing	Help Desk, Chatbot, Email

12.3 Rollback Plan

In case of deployment failures or critical issues, a rollback plan will be executed to revert to the previous stable version of the system. This process will be automated with CI/CD tools to reduce downtime.

1. **Backup**: Ensure that all data (e.g., customer bookings, payment transactions) is backed up regularly to avoid loss during rollback.
2. **Version Control**: Maintain the previous versions of the system in the version control repository, and deploy it again if necessary.
3. **Re-deployment Process**: The previous stable build will be redeployed to the production environment if there are critical issues with the new release.

13. Post-Deployment Support

1. **Customer Support**: A dedicated support team will be available to assist customers with issues after the system is live. The team will use a ticketing system to track and manage customer inquiries.
2. **Performance Reviews**: A quarterly review will be conducted to assess system performance, identify any bottlenecks, and ensure the system is operating efficiently.

14. Risk Management and Mitigation Plan

This section addresses the potential risks associated with the Cinema-View Ticketing System and outlines strategies for mitigating these risks.

14.1 Identified Risks

1. **Technical Risks**:
   - **Integration Issues**: External APIs (e.g., payment gateways) may fail.
     - **Mitigation**: Thoroughly test all integrations in the staging environment and implement fallback mechanisms.
2. **Security Risks**:
   - **Data Breach**: Exposure of sensitive user data.

        - **Mitigation**: Implement strong encryption for sensitive data and perform regular security audits.
3. **Performance Risks**:
    - **Scalability Issues**: The system may not handle high traffic.
        - **Mitigation**: Design the system to scale on cloud platforms with auto-scaling capabilities.
4. **Operational Risks**:
    - **Server Downtime**: Cloud infrastructure failure.
        - **Mitigation**: Use cloud providers with SLAs and ensure redundancy across multiple regions.
5. **Legal and Compliance Risks**:
    - **Non-Compliance**: Failure to comply with data protection regulations (e.g., GDPR).
        - **Mitigation**: Implement privacy-by-design principles and conduct legal reviews before deployment.
6. **Financial Risks**:
    - **Budget Overruns**: Unexpected development complexities or scope creep.
        - **Mitigation**: Set clear milestones and perform regular budget reviews.
7. **Customer Satisfaction Risks**:
    - **Negative User Experience**: Complex user interfaces or system bugs.
        - **Mitigation**: Conduct extensive usability testing and gather user feedback for continuous improvements.
8. **Dependency Risks**:
    - **Reliance on Third-Party Services**: Issues with external payment processors or SMS gateways.
        - **Mitigation**: Establish SLAs with vendors and create fallback mechanisms.

14.2 Risk Assessment Matrix

Risk	Probability	Impact	Mitigation Priority
Integration Issues with APIs	Medium	High	High
Data Breach	High	High	Very High
Scalability Issues	Medium	High	High
Server Downtime	Low	High	Medium
Non-Compliance with Regulations	Medium	High	High
Budget Overruns	Medium	Medium	Medium
Negative User Experience	Medium	High	High
Reliance on Third-Party Services	Medium	Medium	Medium

14.3 Risk Mitigation Strategies

1. **Security Audits**: Regularly perform security scans and implement 2FA for sensitive access.
2. **Scalable Infrastructure**: Use cloud services with auto-scaling to handle traffic spikes.

3. **Legal Compliance**: Ensure adherence to GDPR and other relevant regulations by working with legal experts.
4. **Usability Feedback**: Collect user feedback post-launch for continuous system improvements.

15. Post-Implementation Review and Evaluation

After the Cinema-View Ticketing System is deployed, a Post-Implementation Review (PIR) will be conducted to evaluate the system's performance and identify areas for improvement. The review ensures the system aligns with the requirements outlined in the Software Requirements Specification (SRS) and helps to determine if any additional changes or optimizations are necessary.

15.1 Review Objectives

1. **Evaluate System Performance**:
   - Assess if the system meets performance criteria like handling high traffic, transaction speed, and response times.
2. **Gather Stakeholder Feedback**:
   - Collect feedback from users and administrators to gauge their satisfaction and identify any usability issues.
3. **Ensure Compliance**:
   - Review the system's adherence to security and data protection regulations (e.g., GDPR, FERPA).
4. **Identify System Bugs and Issues**:
   - Detect any unresolved bugs or issues and evaluate the effectiveness of the system’s error handling and resolution process.

15.2 Review Methodology

1. **User Feedback**:
   - Conduct surveys and interviews with users and administrators to gather insights on the system’s usability and functionality.
2. **System Monitoring**:
   - Track performance metrics such as page load times, response times, and system uptime using tools like Google Analytics or New Relic.
3. **Compliance Review**:
   - Perform security audits to ensure continued compliance with data protection regulations and standards.
4. **Key Performance Indicators (KPIs)**:
   - Measure system uptime, customer satisfaction, and conversion rates to determine system effectiveness.

15.3 Evaluation Criteria

The system's success will be evaluated based on:

- **Functional Success**: Did the system meet all functional requirements outlined in the SRS?
- **Performance**: Did the system handle the required load and maintain response time under high usage?
- **Usability**: Was the system user-friendly and easy to navigate for both customers and administrators?
- **Security and Compliance**: Was sensitive data securely handled, and did the system comply with regulations?
- **Stakeholder Satisfaction**: Were users and administrators satisfied with the system's overall performance and usability?

15.4 Continuous Improvement

Based on the Post-Implementation Review, continuous improvements will be made:

1. **Addressing Bugs**: Bugs identified during the review phase will be tracked and fixed in future updates.
2. **Feature Enhancements**: New features like multi-language support and personalized movie recommendations may be added.
3. **Security Enhancements**: Conduct additional security tests to address emerging threats and enhance data protection.
4. **User Support Improvements**: Enhance help desk and customer support based on user feedback regarding response times and issue resolution.

A. Appendices

This section provides supporting information that enhances the understanding of the SRS but is not essential to its main body. It may include conceptual documentation, background research, client meeting notes, references, or other relevant materials used during requirement gathering and analysis.

A.1 Appendix 1

Purpose: Shows an example of an ordinary user response to the system.

- A user visits the website, browses available movies, and selects a showtime for a popular film.
- The user chooses three available seats from the seating chart.
- The system prompts for payment; the user enters valid card details.
- After successful payment, a confirmation message is shown, and a digital ticket is emailed.
- The booking is also saved to the user's account dashboard for future reference.

A.2 Appendix 2

Term	Definition
Booking	A confirmed reservation for selected movie seats.

Term	Definition
Showtime	A scheduled time at which a movie plays.
Admin	A user with privileges to manage movies, showtimes, and reports.
Payment Gateway	The service (mock or real) that processes user payments.
Seat Map	A graphical interface showing available and booked seats.
CRUD	An acronym for Create, Read, Update, Delete operations.