Ticket System

Software Requirements Specification

Version 2

03/06/25

Group 2
Malaika Joiner
Julie Tong
Aditya Ujawane

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

Revision History

Date	Description	Author	Comments
<date>	<Version 1>	<Your Name>	<First Revision>

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date
	<Your Name>	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed description of the requirements for the Movie Ticketing App. This document will outline the application's functionalities, constraints, and dependencies to help software engineers design and implement the software efficiently. The document follows the IEEE Guide to SRS and serves as a reference for all project stakeholders, including developers, testers, and business analysts.

1.1 Purpose

The Movie Ticketing App aims to provide a seamless platform for users to browse, book, and manage movie tickets online. The intended audience includes:

- **End-users (Customers)**: Users who want to book movie tickets for themselves and others.
- **Cinema Administrators**: Responsible for managing movie schedules, seat availability, pricing, and promotions.
- **Developers and Testers**: Software engineers and QA testers are responsible for building, testing, and maintaining the app.
- **Business Stakeholders**: Decision-makers aiming to improve the efficiency and profitability of the movie ticketing business.

1.2 Scope

The Movie Ticketing App will:

- Allow users to **search** for movies, view showtimes, and book tickets.
- Enable secure **online payment** and provide digital tickets with QR codes.
- Offer **seat selection** based on real-time seat availability.
- Provide **user authentication** and profile management for personalized recommendations.
- Allow cinemas to **manage movie listings, schedules, pricing, and discounts.**
- Send **notifications and reminders** for upcoming bookings.
- Integrate a **loyalty and rewards system** to incentivize frequent bookings.

The app will **not** include:

- In-app streaming of movies.
- User-generated content such as reviews or forums in the initial release.

The application will provide:

- A **mobile-friendly interface** for both Android and iOS.
- **Fast and secure ticket booking** with minimal user effort.
- **Integration with third-party payment gateways** (e.g., Stripe, PayPal, Apple Pay, Google Pay).
- **Real-time updates** on seat availability and movie schedules.
- **Support for multiple languages and currencies.**

1.3 Definitions, Acronyms, and Abbreviations

- **SRS** – Software Requirements Specification
- **UI** – User Interface
- **API** – Application Programming Interface
- **OTP** – One-Time Password
- **DBMS** – Database Management System

- **Admin** – Cinema Administrator managing movie listings
- **PCI-DSS** – Payment Card Industry Data Security Standard
- **JWT** – JSON Web Token (used for secure authentication)

1.4 References

- IEEE Std 830-1998 – Recommended Practice for Software Requirements Specifications. (University of Alaska System)
- Payment Gateway API Documentation (Stripe Documentation).
- OAuth 2.0 and JWT authentication standards (Frontegg).
- Database design principles for high-performance applications (Red Gate Software).

1.5 Overview

- **Section 2** provides a general description of the product.
- **Section 3** details specific software requirements, including functional and non-functional requirements.
- **Section 4** lists external interfaces and system constraints.
- **Section 5** describes system features and use cases in more detail.

2. General Description

This section provides an overview of the product and its functionalities.

2.1 Product Perspective

The Movie Ticketing App is a standalone product but integrates with:

- **Third-party payment gateways** for processing transactions.
- **External movie databases** for automated scheduling updates.
- **Theater management systems** to synchronize movie listings and seat availability.
- **Push notification services** to send real-time updates to users.

2.2 Product Functions

The app will include the following functionalities:

- **User Registration/Login**: Secure authentication via email, phone number, or social media accounts.
- **Movie Browsing & Filtering**: Users can search for movies by title, genre, language, location, and time.
- **Seat Selection**: Interactive seat maps showing real-time availability and pricing tiers.
- **Online Payment Processing**: Secure transactions using credit/debit cards, digital wallets, and UPI.
- **E-Tickets and QR Code Generation**: Generate digital tickets for easy check-in and verification at theaters.
- **User Profiles**: Allow users to manage bookings, track purchase history, and set preferences.
- **Loyalty Rewards System**: Offer discounts, cashback, and reward points for frequent bookings.
- **Cinema Management Dashboard**: Admin panel for movie theater owners to manage schedules, ticket prices, and promotions.

- **Push Notifications & Email Alerts**: Send booking confirmations, reminders, and special offers.

2.3 User Characteristics

- **General Users**: Non-technical users who want a simple interface for booking movie tickets.
- **Cinema Administrators**: Users who need advanced features for managing schedules, pricing, and reports.
- **System Administrators**: IT personnel responsible for maintaining and securing the app's backend and database.

2.4 General Constraints

- The app must comply with local **data protection laws** (e.g., GDPR, CCPA).
- **Payment transactions** must be processed via PCI-DSS-compliant gateways.
- System downtime should not exceed **1% monthly** for reliability.
- Mobile app size should not exceed **50MB** to ensure faster downloads.
- The app should be **scalable** to support high traffic during peak hours (e.g., movie releases, and holidays).

2.5 Assumptions and Dependencies

- The app assumes **stable internet connectivity** for seamless operation.
- It relies on **third-party payment gateways** for transactions and refunds.
- It depends on **external APIs** to fetch real-time movie schedules and seat availability.
- The operating system requirements are **iOS 13+** and **Android 8+** for compatibility.
- Customer support will be available via **chatbots and live agents** during business hours.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- The UI needs to be easy to use
- Users should be able to access via a digital kiosk in the theater or online via a website

3.1.2 Hardware Interfaces

- UI needs to support both physical/printable tickets and digital tickets

3.1.3 Software Interfaces

- The system has to interface with the database of showtimes and tickets available
- It needs to be able to scrape online review websites to display reviews and critic quotes of movies

3.1.4 Communications Interfaces

- Tickets can be provided via email or in person at the box office

3.2 Functional Requirements

3.2.1 <Feature #1: Ticket Purchase and Management>

3.2.1.1 Introduction

- This feature enables customers to purchase movie showing tickets and manage them. It ensures a seamless transition process while also adhering to security and business rules.

3.2.1.2 Inputs

- User account information
- Showtime selection
- Seat selection
- Payment information—may take credit card, PayPal, or Bitcoin
- Number of Tickets—up to 20 per transaction
- Discount codes—student, veteran/military, and senior

3.2.1.3 Processing

- Verify user account and login
- Check showtime and seat availability
- Apply discounts if applicable
- Process payment and currency conversion
- Generate unique and non-replicable tickets
- Update database with transaction details and remaining tickets
- Record transactions in daily logs

3.2.1.4 Outputs

- Digital or printable tickets with unique identification
- Confirmation email with ticket details
- Updated user account information—purchase history, loyalty points

3.2.1.5 Error Handling

- Invalid user account or login attempt
- Unavailable showtime or seats
- Invalid payment information
- Failed transactions
- Exceeding the maximum ticket limit
- Discount code errors—invalid or expired

3.2.2 <Feature #2: User Account Management>

3.2.2.1 Introductions

- This feature allows customers to create and manage their accounts within the theater ticketing system
- Creating an account is optional but provides benefits such as storing personal and payment information, loyalty points, and purchase history
- Allows for ticket returns and exchanges

3.2.2.2 Inputs

- **User Registration**
  - Personal information—name, contact information
  - Login credentials—email/username/phone number, password
- **Payment Information**
  - Credit card details
  - PayPal account information
  - Bitcoin wallet information
- **Account Login**
  - Email/username/phone number
  - Password
- **Account Updates**

    - Modified personal information
    - Updated payment information

3.2.2.3 Processing

- Account creation: The system must validate the provided information
    - Checks for uniqueness of email/username
    - Encrypts password
    - Stores account information in the database
- Login Authentication: System compares and verifies that the provided credentials are the same as the stored credentials.

Only one account can concurrently log in to a user account.

- Payment Processing: System securely stores and manages payment information for a faster transaction
- Loyalty Point Calculation: System automatically awards loyalty points based on ticket purchases and updates the user's loyalty points balance
- Ticket Returns/Exchanges: System verifies if the ticket was purchased with a registered account and processes returns/exchanges according to the theater's policies

3.2.2.4 Outputs

- Account Confirmation: A confirmation email/message is sent to the user given a successful account creation
- Login Success: User is granted access to their account dashboard
- Purchase History: A detailed record of past purchases is displayed
- Loyalty Points Balance: Current loyalty point balance is shown to user
- Updated Account Information: Confirmation that personal information or payment information has been successfully updated

3.2.2.5 Error Handling

- Invalid Input—incorrect or missing information during registration or account updates
- Duplicate Account—email or username is already associated with an existing account
    - Incorrect Login Credentials
    - Payment Failure
- Session Management—if a user attempts to log in from a second device, the system terminates the previous session

3.3 Use Cases

3.3.1 Use Case #1

Buy Ticket

Actor(s): TicketBuyer

Flow of events:

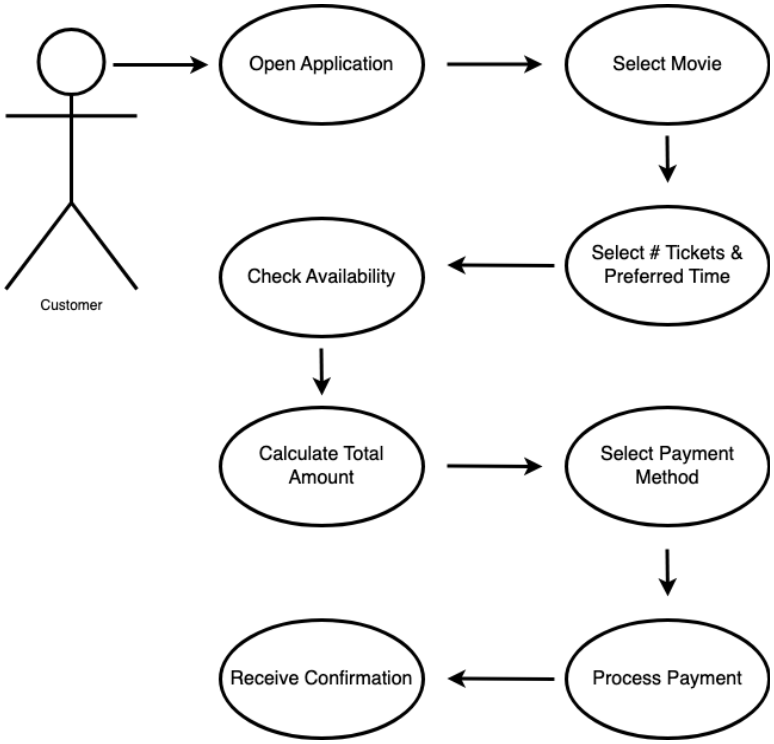
1. TicketBuyer opens the application or website on the device.
2. TicketBuyer chooses which movie they want to buy tickets for.
3. TicketBuyer selects how many tickets to buy and for which seats.
4. The amount of money owed is calculated, and the TicketBuyer selects their preferred payment option and gives the required amount.
5. The TicketBuyer receives a confirmation of purchase to show at the movie site.

Entry Conditions

1. TicketBuyer must be connected to the internet.

2. Computing requirements: supported browser if on the website

Use Case Diagram:



3.3.2 Use Case #2

Search for Movies

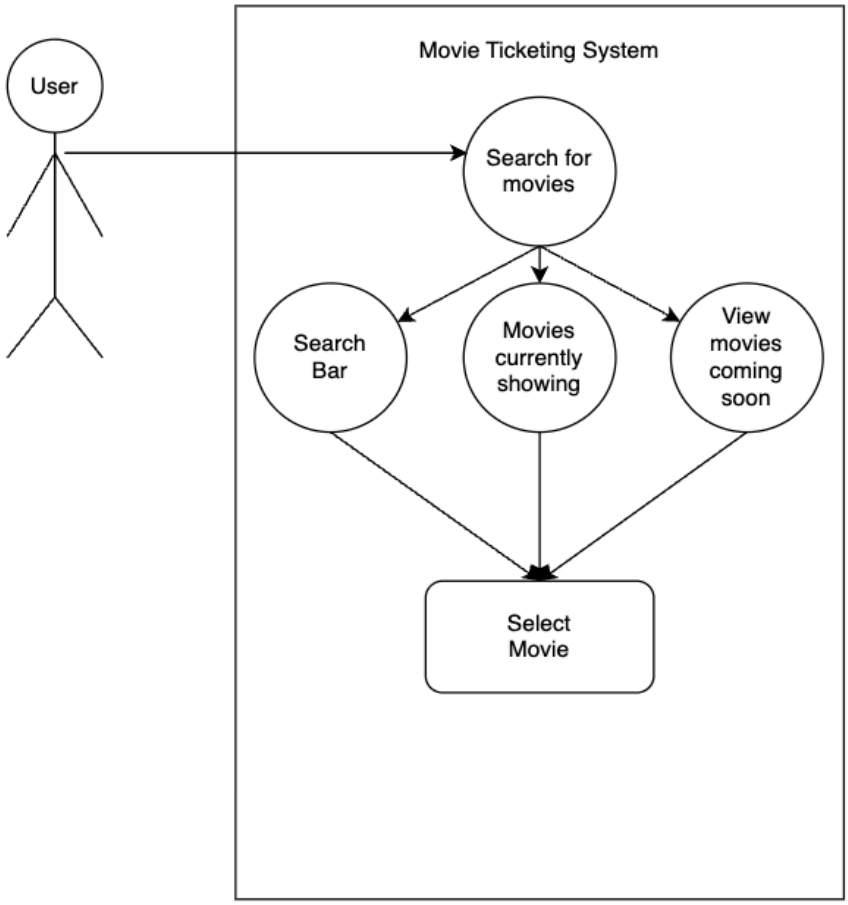
Actor(s): MovieSearcher

Flow of events:

1. MovieSearcher opens the application or website on the device.
2. MovieSearcher can click on the “Movies Currently Showing” or “Movies Coming Soon” page to redirect them to their desired movie list or choose to use the search button to filter movies with certain keywords. .
3. MovieSearcher scrolls through their options of movies until they decide to click on a specific movie or exit the page.
4. If a movie is chosen, MovieSearcher will be directed to ticket buying.

Entry Conditions

3. MovieSearcher must be connected to the internet.
4. Computing requirements: supported browser if on website



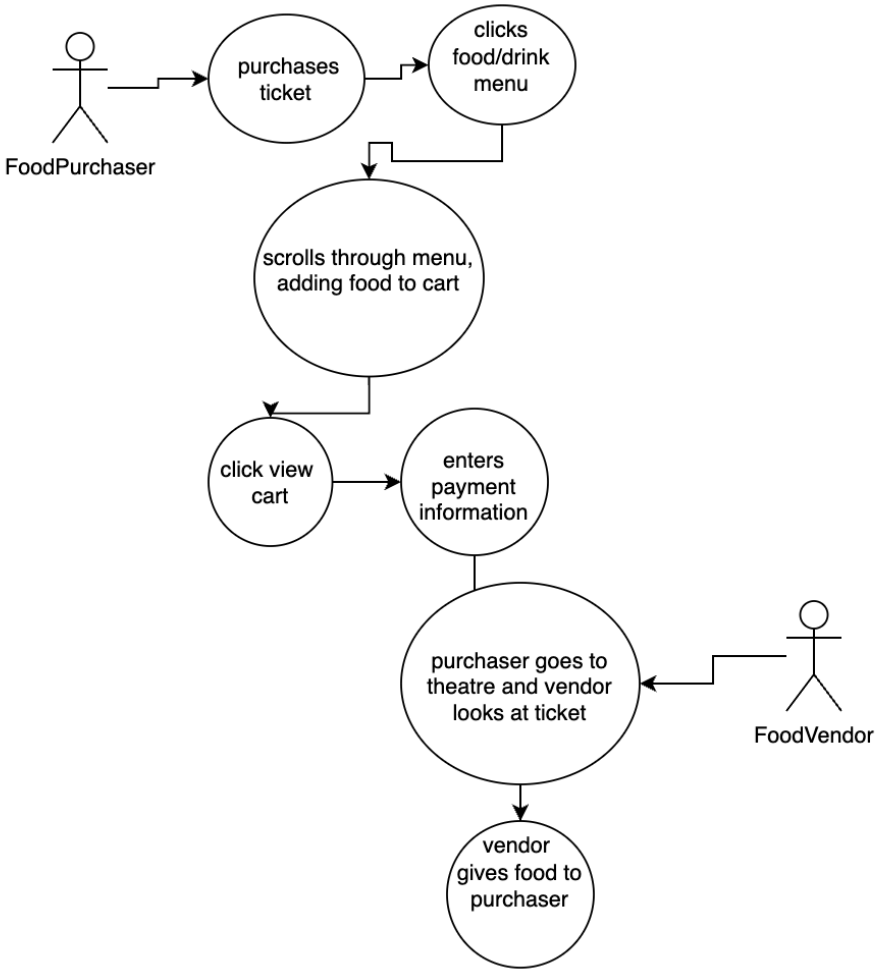
3.3.3 Use Case #3

Purchase Food and Drink

Actor(s): FoodPurchaser, FoodVendor

Flow of events:

1. FoodPurchaser successfully purchased a ticket to a movie.
2. FoodPurchaser has the option to click on the “Add Food or Drink” menu.
3. After clicking the menu, FoodPurchaser is redirected to the list of food and drinks they can add to their purchase.
4. FoodPurchaser looks through a list of options and may choose to add any items to cart.
5. FoodPurchaser clicks “View Cart” when finished ordering.
6. FoodPurchaser is directed to the payment page. When buying, FoodPurchaser is asked which movie ticket it is related to.
7. When FoodPurchasers arrive at the movie theatre, they show the confirmation of purchase to FoodVendor, and they receive their food quickly without needing to pay for it in the theatre.



3.4 Classes / Objects

3.4.1 <Class / Object #1-3>

TicketBuyer
Username: String Payment: Payment Information Login : Log-in Information
PurchaseTicket(ticket : movie) PurchaseFood(food : String) InputUsername(username: String) InputPassword(password: String) InputPaymentInfo(info : PaymentInformation)

Log-InInformation
UserName: String Password: String
ValidatePassword()

PaymentInformation
Balance: Dollars = 0
GiveMoney(amount : dollars)

MovieList
movie1: movie movie2: movie movie3: movie
addMovie(Movie: movie) deleteMovie(Movie: movie) listMovies()

movie
movieName: String rating: String ticketPrice: (amount : dollars = 0) movieDesc: Sting
giveMovieInformation()

3.5 Non-Functional Requirements

3.5.1 Performance

The product should be able to run on both a website browser and from a downloadable application for mobile devices. It will rely on a good connection to the internet to get access to information that is always updating. Application should be fully functioning with an internet speed at 2 Mbps.

3.5.2 Reliability
System will experience little to no crashes, even during times with increased usage and demand. When a purchase is stopped mid-transaction, it is guaranteed that the user will receive their money back and will be able to try again. There will be less than 6 minutes of downtime per year for the system.

3.5.3 Availability
The application is available to use and make purchases 24/7 at all times. It will still be functional even during the theatre's closing hours, allowing users to make purchases during the night or holidays.

3.5.4 Security
All data for accounts and purchases made through the application are secure and protected against data breaches. Payment information will never be given to unauthorized users and will be kept confidential. Security system follows the ISO 27001 standard.

3.5.5 Maintainability
General system failures will be addressed and fixed within 5 minutes with an 85% maintainability rate. Users are also able to personally submit help requests to customer service, which are closely monitored for 13 hours on weekdays from 8:00 AM to 9:00 PM. When emailed during customer service hours, the user will generally receive a reply within 10 minutes. However, if emailed outside of customer service hours, they will receive a reply promptly the next business day.

3.5.6 Portability
The application should be available to most devices released in the past 15 years. This includes but is not limited to a variety of smartphones, laptops, desktop computers, and tablets; regardless of branding. This does not include certain devices such as playstation consoles, flip-phones, and smartwatches.

3.6 Inverse Requirements
- Users cannot purchase a ticket without being already logged into an account.
- Users cannot purchase food or drink without being logged in and already buying a movie ticket.
- Users cannot purchase a movie ticket to an R-rated movie without verifying age as 17+ on their account.
- Users cannot purchase an alcoholic drink without verifying as 21+ on their account.
- Users cannot view or purchase tickets from a movie that has stopped showing.
- If payment is not successful, the purchase of the item does not go through.

3.7 Design Constraints
- The app will not be guaranteed to run on older devices. It will be constrained to mostly modern devices from the last 10 years.
- The devices needed to run the application need at least 1GB of RAM, an internet connection, and a 2.0 GHz clock speed.
- Movie tickets will only be available to participating theatres and the movies those theatres are offering.
- Age verification for alcoholic drinks and certain rated movies must fit with the local legal guidelines

- The application's interface must be kept simple to control the loading speeds (should all load within 2 seconds) for browsing and purchasing on mobile devices
- The overall budget for this project is minimal with only a few software developers hired, so the design should be restricted to necessary elements.
- The application must be compatible with screen readers, text magnification, captioned text and other accommodations for users with preferences and/or disabilities.
- Language support and currency exchange must be available for the regions where the application is available

3.8 Logical Database Requirements

- Uses an MySQL database to manage customer accounts and purchase details.
- The database must be able to store individual accounts that contain a username, password, payment information, and all their purchase history.
- Each purchase must be documented with a unique purchase ID and date.
- Each username created must be unique to one account.
- Database is backed up every 2 hours to prevent data loss.
- All data should be kept in an archive for 2 years.
- Properly documents and monitors audit trails.
- The system should be able to handle and smoothly operate a large amount of accounts stored in the database.

3.9 Other Requirements

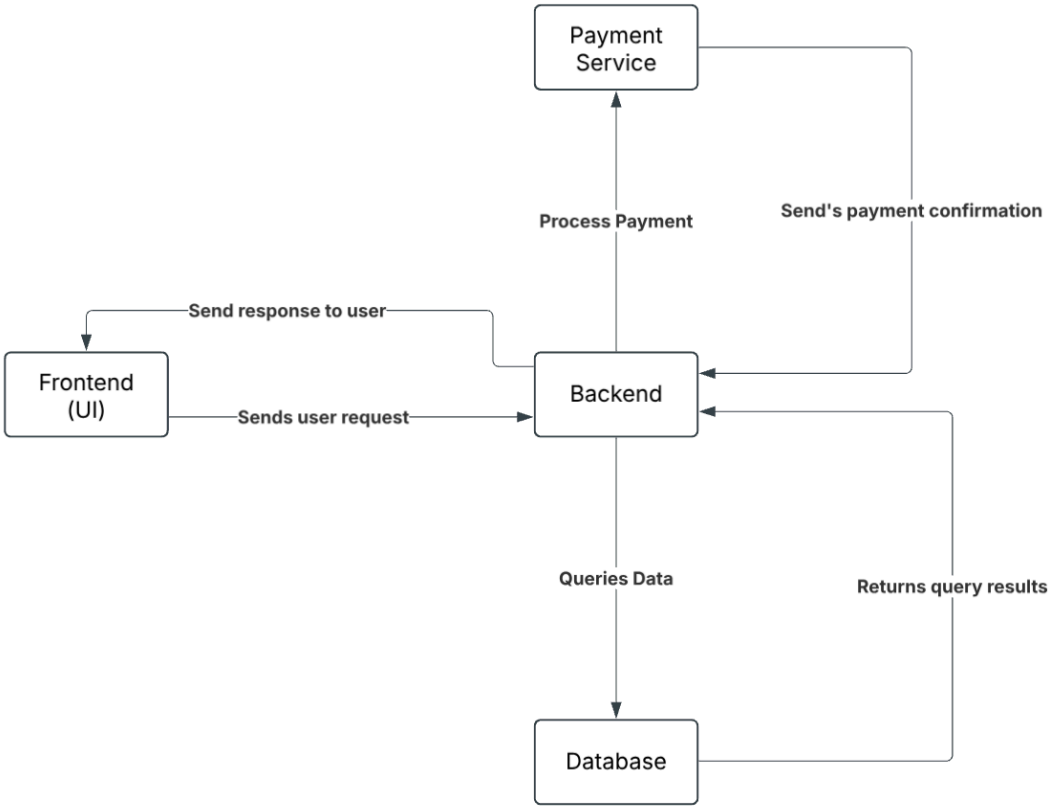
- The website's accessibility accommodations must comply with WCAG guidelines
- Both web-accessed and application user interfaces must be kept simple and user-friendly
- Application must well adapt to different screen sizes and formats
- Payments must comply with local taxes and other regulations
- The website must be well search engine optimized
- Purchases must follow proper PCI SSC security standards for card transactions and data.

4. Software Design Specification (SDS)

4.1 System Description

- The Movie Ticket Theater System is designed to facilitate online movie ticket and food purchases for customers. From a development perspective, the system follows a modular, object-oriented design to ensure maintainability and scalability.
- The user interface will communicate with the backend to access and change data in the database. The database will store user information such as accounts, purchase history, and other information. The payment services will receive requests with a payment gateway to complete and process transactions.

4.2 Software Architecture Overview



The movie ticketing system follows a client-server architecture with multiple components handling different responsibilities:

1. **Frontend (UI):**
   The primary interface through which users interact with the system.
   - Allows users to browse available movies, select showtimes, book tickets, and make payments.
   - Sends user requests to the backend and displays responses.
2. **Backend:**
   - Processes all incoming requests from the frontend.
   - Retrieves and updates data from the database.
   - Manages business logic such as seat availability, user authentication, and booking validation.
   - Sends booking confirmation and ticket details to the frontend.
3. **Database:**

   - Stores all relevant information, including user profiles, movie listings, showtimes, bookings, and payment transactions.

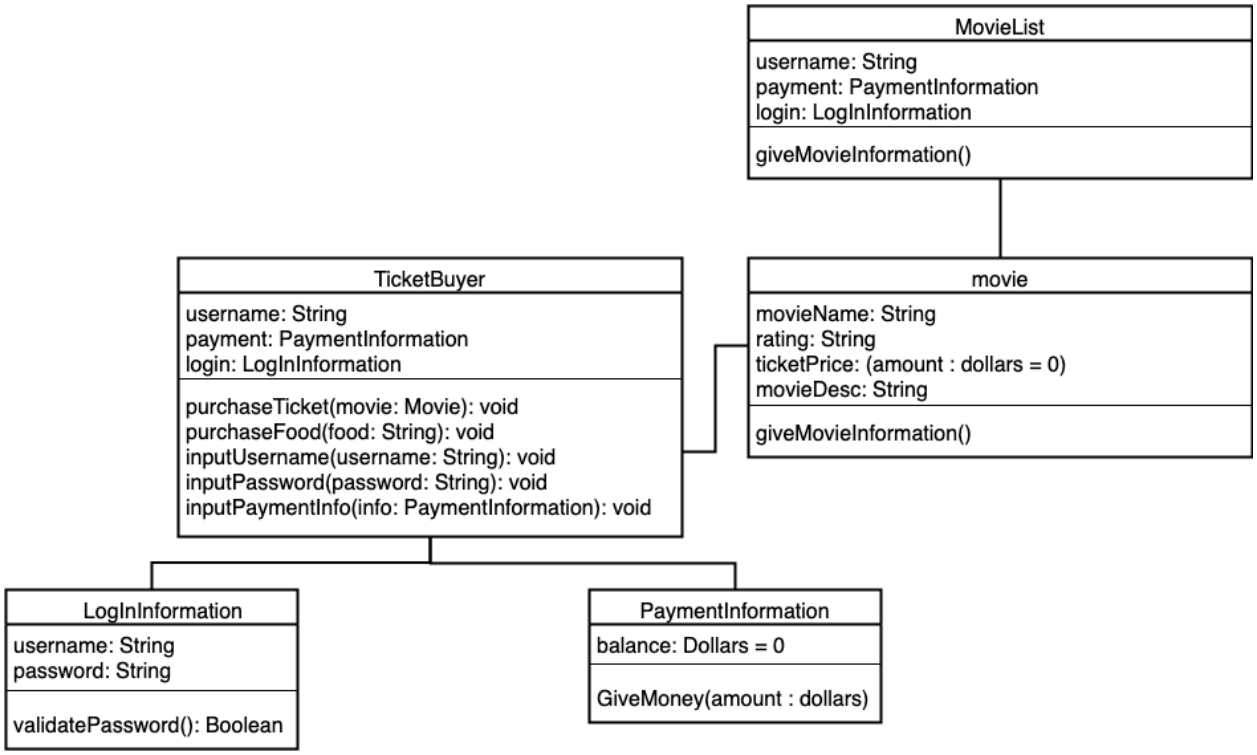
- Ensures data consistency and integrity.

4. **Payment Service:**
    - Handles secure transactions and payment processing.
    - Interfaces with third-party payment gateways if necessary.
    - Confirms successful or failed payments to the backend.

Workflow Example:

- The user selects a movie and requests available seats.
- The frontend sends the request to the backend.
- The backend queries the database for available seats and returns the data.
- The user selects a seat and proceeds with the booking.
- The backend initiates the payment process through the payment service.
- Once payment is confirmed, the backend updates the database and sends confirmation to the frontend.

4.3 UML Class Diagram & Descriptions



Class Descriptions:

- TicketBuyer: Class to represent the user who is using the application. Uses the classes Log-in Information and payment information objects as attributes. It has functions to put in information as string inputs as parameters (username, password, payment information) and also has a function to purchase items using a String.
- Log-in Information: Can be created as an instance object for TicketBuyer to store usernames and passwords as strings in the database. Its validate function is used to see if the imputed usernames and passwords strings as parameters are correct and match the strings already inside the database.

- Payment information: Can be created as an instance object for TicketBuyer to purchase items. It is able to transfer money from the linked payment account to the movie ticket site using the GiveMoney() function. It has Dollars as a double data type attribute.
- MovieList: Class to represent the entire list of movies currently available to view on the application. Its attributes are the movie objects it holds, and it has functions to add or remove the movies in the list, which both take a movie object as input parameters. There is also a function to display the entire list of movies available to the user.
- Movie: Can be created as objects for the MovieList class. Movie instances can contain the movie title, rating, and description as strings and price as a double data type. It has a function to list all its attributes to the user and its attributes can be accessed by the system.

5. Test Plan

5.1 Test Strategy Overview

The purpose of the test plan is to verify that the movie theater ticket software system is functional and to ensure that its components work as intended. The test plan will cover unit, functional, and system testing. The test cases are designed to validate the software's correctness, reliability, and usability.

5.2 Features to be Tested

The primary features of the Movie Ticket System that are to be tested include:

- **Security Features:** Fraud prevention and role-based access
- **User Authentication:** Registration, login, and session management
- **Seat Selection:** Choosing available seats for a selected movie
- **Movie Listings:** Display available movies, showtimes, and seat availability
- **Payment Processing:** Handles various payment methods and confirms transactions, ensuring payment
- **Ticket Generation:** Issuing digital tickets with QR codes
- **Cancellation and Refunds**: Process ticket cancellations and issue refunds.
- **Admin Functions**: Adding/editing/deleting movies, showtimes, and pricing.
- **System Performance**: Load testing, response time, and scalability.
- **Security Features**: Data encryption, role-based access, and fraud prevention.

5.3 Test Strategy

The testing strategy includes unit, functional, and system tests:

- **Unit Testing**: Verifies individual modules for correctness.
- **Functional Testing**: Ensures the system functions according to requirements.
- **System Testing**: Evaluates the end-to-end system behavior under real-world conditions.

5.4 Test Cases

This can be found in this link

5.5 Coverage and Failure Scenarios

5.5.1 Test Coverage

This part describes the extent of testing across different system functionalities and requirements. It ensures that critical features are tested adequately.

- **Functional Testing:** Covers the core functions of software. Includes user authentication, movie listing, seat selection, payment processing, and ticket generation.
- **Non-Functional Testing:** Includes performance, security, and usability testing, aiming to ensure that system is efficient.
- **Extreme Case Testing:** Ensures that the system can handle/does handle extreme cases. Including maximum seat selections, peak-hour bookings, and large payment transactions.
- **Compatibility Testing:** Verifies that the system can perform across different browsers, devices, and operating systems.
- **Error Handling and Recovery Testing:** Checks how the system reacts to errors/failures. Includes network outages and incorrect payment details.

5.5.2 Failure Scenarios

- Failure to Log in: If an account with the entered username and password is not valid, give an error message to the user prompting them to re-enter their user information.
- Username/Email already in use when attempting to create a new account: Gives an error message about the restriction of duplicate accounts and prompts the user to choose another username or email.
- Ticket is unable to be purchased failure: In the case of a showing not currently being available, the system should prevent the purchase of the ticket and display an error message instead.
- Failure to access internet connection: The website will not load properly with poor internet connection and will prompt the user to re-check their internet stability.
- Payment failure: When the payment does not successfully transfer due to incorrect payment information or an imbalance of funds, the ticket is not purchased and instead prompts the user to reattempt the purchase.
- Attempt to purchase an empty cart: If the cart is currently empty, if attempted to continue with a purchase it should redirect the user to add more items to the cart before purchasing.
- Failure to search for products: when searching for a name of a product/movie, if it does not successfully display relevant results, it should prompt the user to search again in a few seconds.

5.6 Test Set/Vectors:

5.6.1 Test Case 1: Login

- Test Vector:
    - Username: valid_user
    - Password: secure_password123
- Expected Result: The user is logged in successfully and redirected to the homepage.

5.6.2 Test Case 2: Registration

- Test Vector:
    - Username: valid_user
    - Email: valid_address@email.com
- Expected Result: The username and email is not currently in the database under a different account

5.6.3 Test Case 3: Search

- Test Vector:
    - SearchItem: movieName
- Expected Result: the display should list related items to the intended searched item.

5.6.4 Test Case 4: Shopping Cart

- Test Vector:
    - AddedItem: movieTicket
- Expected Result: the entered item(s) should be displayed to the user under their cart and should be purchased when on the payment page.

5.6.5 Test Case 5: Checkout

- Test Vector:
    - Payment information: Credit card number, Paypal account, etc
    - Balance: currentBalance
    - CartValue: Value
- Expected Result: The amount of balance associated with the given payment information should be greater than or equal to the value of the cart to successfully purchase items.

5.6.6 Test Case 6: Performance

- Test Vector:
- userAmount: 1000
- Expected Result: The website should simulate having a large amount of users currently on the website to track response time. The system should successfully function without crashing or lowering speeds.

5.6.7 Test Case 7: Password Reset

- Test Vector:
    - Registered Email: testuser@gmail.com
- Expected Result: A password reset email is sent.

5.6.8 Test Case 8: User Profile

- Test Vector:

    -Username: valid_user

    - NewPhoneNumber: 123-456-7891
- Expected Result: The new phone number should be valid and will successfully update the profile with a new phone number

5.6.9 Test Case 9: Security

- Test Vector: Verify that multiple failed login attempts lock the account

- Step 1: Enter an incorrect password five times, using the correct username.
    - Username: test_user, Password: wrong_password_1
    - Username: test_user, Password: wrong_password_2
    - Username: test_user, Password: wrong_password_3
    - Username: test_user, Password: wrong_password_4
    - Username: test_user, Password: wrong_password_5
- Step 2: Attempt to log in again after the failed attempts (using either the correct or another incorrect password). This will check if the account is locked after repeated failed attempts.
    - Username: test_user, Password: any_password

- Expected Result:
    - **Step 1:** After the fifth failed attempt, the system should respond with an error message indicating that the login attempt failed. The system should not lock the account yet but track the failed attempts.
    - **Step 2:** After the 5 failed attempts, when the user attempts to log in again, the system should:
        1. Prevent further attempts
        2. Display a message such as "Your account has been locked due to too many failed login attempts. Please try again later or contact support."
        3. The user should not be able to log in, regardless of whether they enter the correct password or another incorrect one.

5.6.10 Test Case 10: Logout

- Test Vector:
    - Action: Click the “Logout” button located in the user interface
- Expected Result: Upon clicking the "Logout" button, the system should log the user out and redirect them to the login page or the homepage, with a confirmation message or visual cue indicating the user is logged out.

5.7 Github Link

https://github.com/aujawane/CS-250-SRS-Document

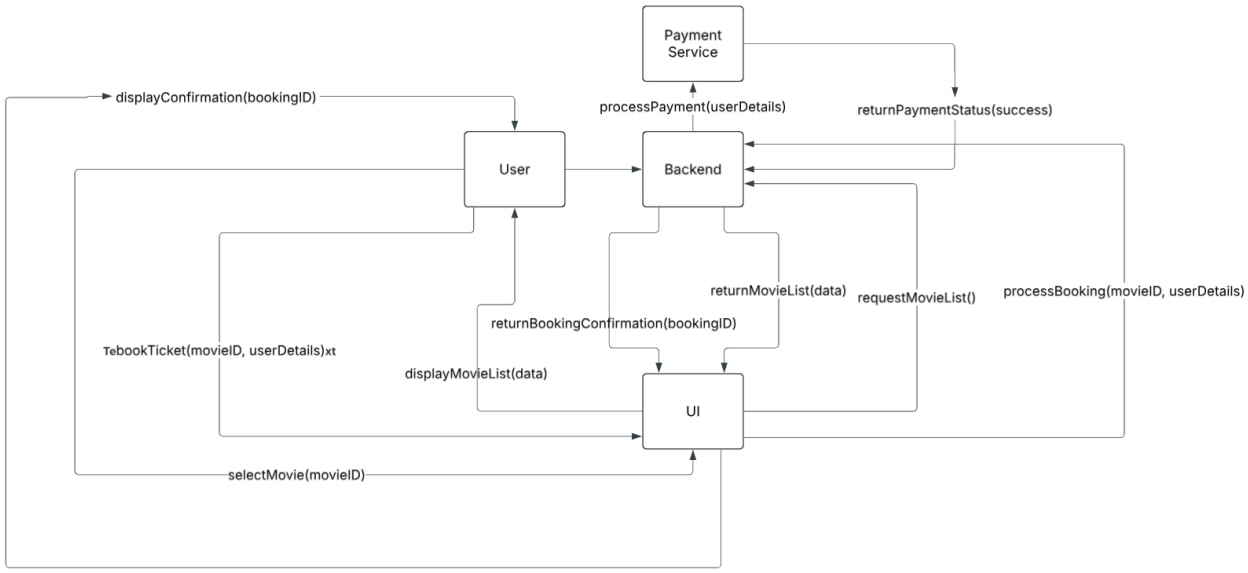
6. Development Plan & Timeline

Team Member	Task Breakdown	Estimated Time
Julie	UML Diagram System Description	1 hour
Aditya	Software Architecture overview Data Flow Diagrams (DFD) State Transition Diagrams (STD)	2 hours
Malaika	UML Diagram Descriptions Use Cases	3 hours

	Specific Requirements	

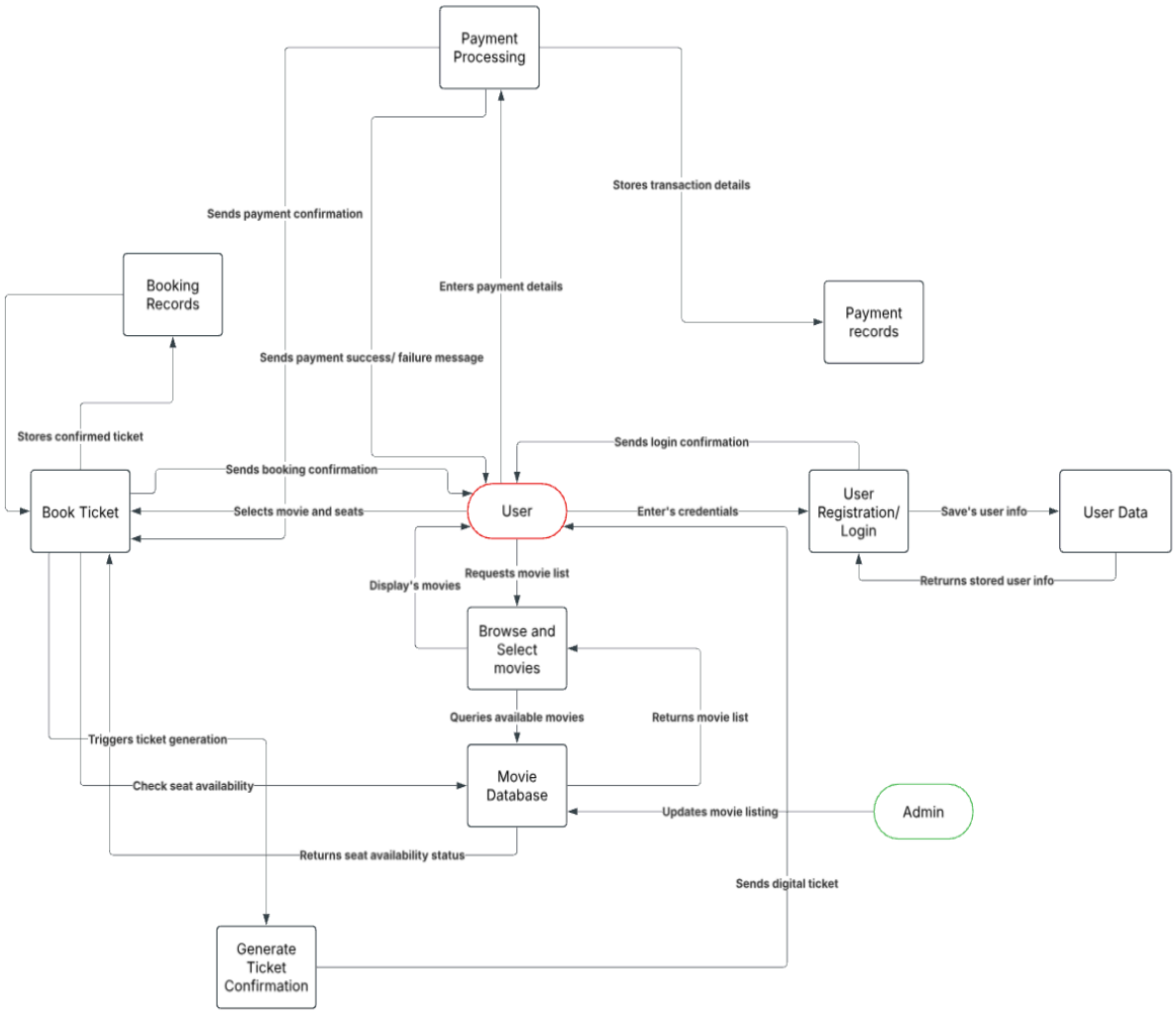
7. Analysis Models

7.1 Sequence Diagrams



The Sequence Diagram outlines the interactions between the User, UI, Backend, and Payment Service during the ticket booking process. The user selects a movie through the UI, which requests the movie list from the Backend. The Backend returns the movie list to the UI, which displays it to the user. Once the user chooses a movie and enters their details, the UI sends a booking request to the Backend. The Backend processes the booking and initiates payment processing with the Payment Service. After the payment is processed, the Payment Service sends the payment status back to the Backend, which then sends a booking confirmation to the UI. Finally, the UI displays the booking confirmation to the user. This diagram captures the flow of data and actions throughout the ticket booking process.

7.3 Data Flow Diagrams (DFD)



The Data Flow Diagram illustrates how data moves through the system, including user interactions and system processes.

Entities, Processes, and Data Stores

1. **User (External Entity):**
    - Provides input, such as registration details, movie selection, and payment information.
    - Receives output, such as available movies, booking confirmation, and payment status.
2. **Processes:**
   User Registration: Captures user details and stores them in the database.
    - **Movie Selection:** Fetches available movies and showtimes.
    - **Booking Ticket:** Handles seat selection and reservation.

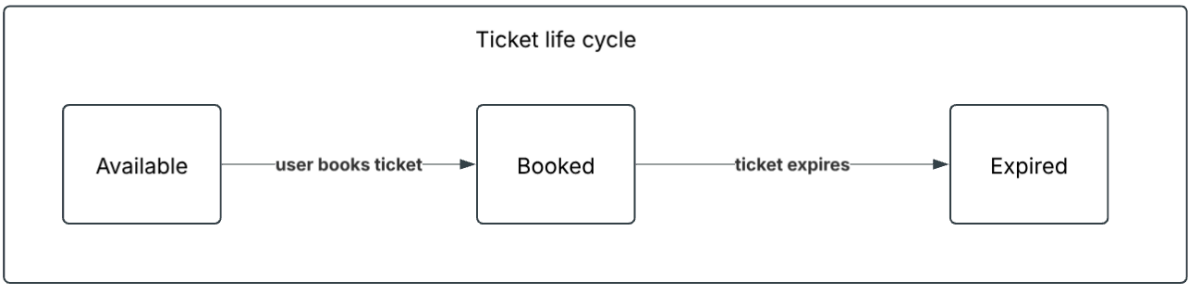
- **Payment Processing:** Validates and processes the payment.

3. **Data Stores:**
   User Data: Stores registered user details.
   - **Movie Data:** Stores movie listings, showtimes, and available seats.
   - **Ticket Data:** Maintains booking details and seat reservations.
   - **Payment Data:** Records payment transactions and statuses.

Data Flow (Connections)

- **User Registration → User Data:** Stores user details upon registration.
- **Movie Selection → Movie Data:** Retrieves and displays available movies.
- **Booking Ticket → Ticket Data:** Reserves the selected seat.
- **Payment Processing → Payment Data:** Stores transaction details and status.
- **Payment Data → Payment Processing:** Returns success/failure status to confirm or reject the booking.

7.2 State-Transition Diagrams (STD)



The State-Transition Diagram represents the different states a ticket can be in and the transitions between them.

States:

1. **Available:**
   - The default state where the ticket is not yet booked and can be reserved by any user.

2. **Booked:**
   - The ticket is reserved by a user after a successful booking.
   - Payment has been processed, and the booking details are recorded.
3. **Cancelled:**
   - The user cancels the booking before the showtime.
   - The seat returns to the available pool.
4. **Expired:**
   - The ticket automatically expires after the showtime has passed.
   - No longer available for use or modification.

Transitions (Events That Change State):

1. **Available → Booked:**
   - The user selects a seat and successfully books the ticket.
2. **Booked → Cancelled:**

    - The user cancels their booking before the showtime, making the seat available again.
3. **Booked → Expired:**
    - The movie showtime has passed, and the ticket is no longer valid.
4. **Expired → Available (Optional):**
    - If a system rule allows, expired tickets might be reset for rebooking.

8. Change Management Process

A team member can propose changes to the SRS by submitting a GitHub issue detailing the modification, its rationale, and its impact. Proposed changes will be reviewed through team discussions, and if approved by majority vote, they will be documented in the Change Log. Implementation will follow a structured process where changes are made in a separate Git branch, submitted via a pull request, and reviewed by at least one other team member before merging. The updated SRS will be pushed to the group's GitHub repository as a PDF or TXT file

9. Test Execution Process

9.1 Test Environment Setup

The testing environment is a critical component to ensure the accuracy and effectiveness of all test cases. The environment should closely mirror the production system to detect potential issues early. The following configurations will be maintained:

- **Operating System**: The test system will support multiple operating systems, including Windows, macOS, Linux for desktop environments, and Android/iOS for mobile applications. This ensures compatibility across different user platforms.
- **Browsers**: Testing will be conducted across various web browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. The objective is to ensure that the web-based interface functions consistently across all supported browsers.
- **Databases**: The system's backend will be tested using MySQL, ensuring that data storage, retrieval, and transactional integrity are validated under various scenarios.
- **Payment Gateway**: Transactions will be tested through Stripe and PayPal sandbox environments to verify payment processing, refunds, and security compliance.

9.2 Execution Phases

A structured testing approach will be followed, ensuring each phase validates different levels of the system functionality.

9.2.1 Unit Testing

Unit testing focuses on verifying the smallest testable components of the application, such as individual functions, classes, or methods. Each module will be tested in isolation to ensure that:

- Input/output behavior is correct.
- Error handling works as expected.
- Modules function independently without dependencies on other components.

Test cases will be written using automated frameworks such as **JUnit (for Java-based modules)** and **pytest (for Python-based modules)** to ensure test coverage and repeatability.

Unit Test Cases:

- **Login Module:** Verify that valid user credentials allow access and incorrect credentials deny access.
- **Registration Module:** Ensure duplicate emails are not allowed during account creation.
- **Search Module:** Confirm that searching for a movie returns relevant results.
- **Payment Module:** Validate successful transactions with different payment methods.

9.2.2 Integration Testing

Integration testing validates the interaction between different modules to ensure seamless functionality. It will include:

- **Data Flow Validation:** Ensuring that data correctly passes between modules (e.g., from booking to payment processing).
- **API Communication Testing:** Verifying that external APIs (payment gateways, authentication services) respond correctly to requests.
- **Third-Party Services Testing:** Ensuring compatibility with third-party integrations, such as notification services and external movie databases.

Integration Test Cases:

- **Booking Module & Payment Gateway:** Validate the ticket purchase process from selection to payment confirmation.
- **User Profile & Authentication System:** Ensure profile updates reflect correctly and persist after logout/login.
- **Movie Search & Recommendation Engine:** Verify that user history influences suggested movies correctly.

9.2.3 System Testing

System testing evaluates the complete system to confirm that all components function together as intended. It will include:

- **End-to-End Workflow Testing:** Validating the entire ticket booking journey from movie search to payment and confirmation.
- **Error and Exception Handling:** Ensuring appropriate error messages are displayed when users enter incorrect data or encounter system issues.
- **Edge Case Analysis:** Testing scenarios such as booking the last available seat, simultaneous payments, and failed transactions.

System Test Cases:

- **Peak Load Handling:** Test system behavior when 1000+ users attempt bookings simultaneously.

- **Ticket Availability Sync:** Validate seat selection updates in real-time across multiple users.
- **Failed Payment Handling:** Ensure transactions revert if payment fails midway.

9.2.4 User Acceptance Testing (UAT)

User acceptance testing ensures the system meets business and user expectations. It will include:

- **Beta Testing with Real Users:** Conducting usability tests with real users to gather feedback on interface design and workflow.
- **Scenario-Based Testing:** Validating use cases such as first-time user booking, frequent customer experience, and administrator panel operations.
- **Business Rule Validation:** Ensuring compliance with business logic, such as age restrictions for R-rated movies and loyalty point calculations.

UAT Test Cases:

- **First-Time User Experience:** Verify that new users can complete registration and book a ticket seamlessly.
- **Refund Processing:** Ensure ticket cancellations and refunds are handled properly.
- **Promo Code & Discounts:** Validate correct application of promotional discounts during checkout.

9.2.5 Performance & Security Testing

Performance and security tests will be conducted to ensure system reliability and protection.

- **Load Testing:** Simulating high traffic conditions to test server scalability and response time under peak loads.
- **Stress Testing:** Testing the system beyond expected limits to identify breaking points and ensure graceful recovery.
- **Security Testing:** Conducting penetration tests to identify vulnerabilities, ensuring data encryption and protection against common threats (e.g., SQL injection, cross-site scripting).

Performance & Security Test Cases:

- **System Stability Under Heavy Load:** Ensure the system doesn't crash with 5000 concurrent users.
- **Multiple Failed Login Attempts:** Validate account lockout after repeated incorrect password attempts.
- **Data Encryption:** Ensure sensitive user information (e.g., payment details) is stored securely.

9.3 Test Schedule

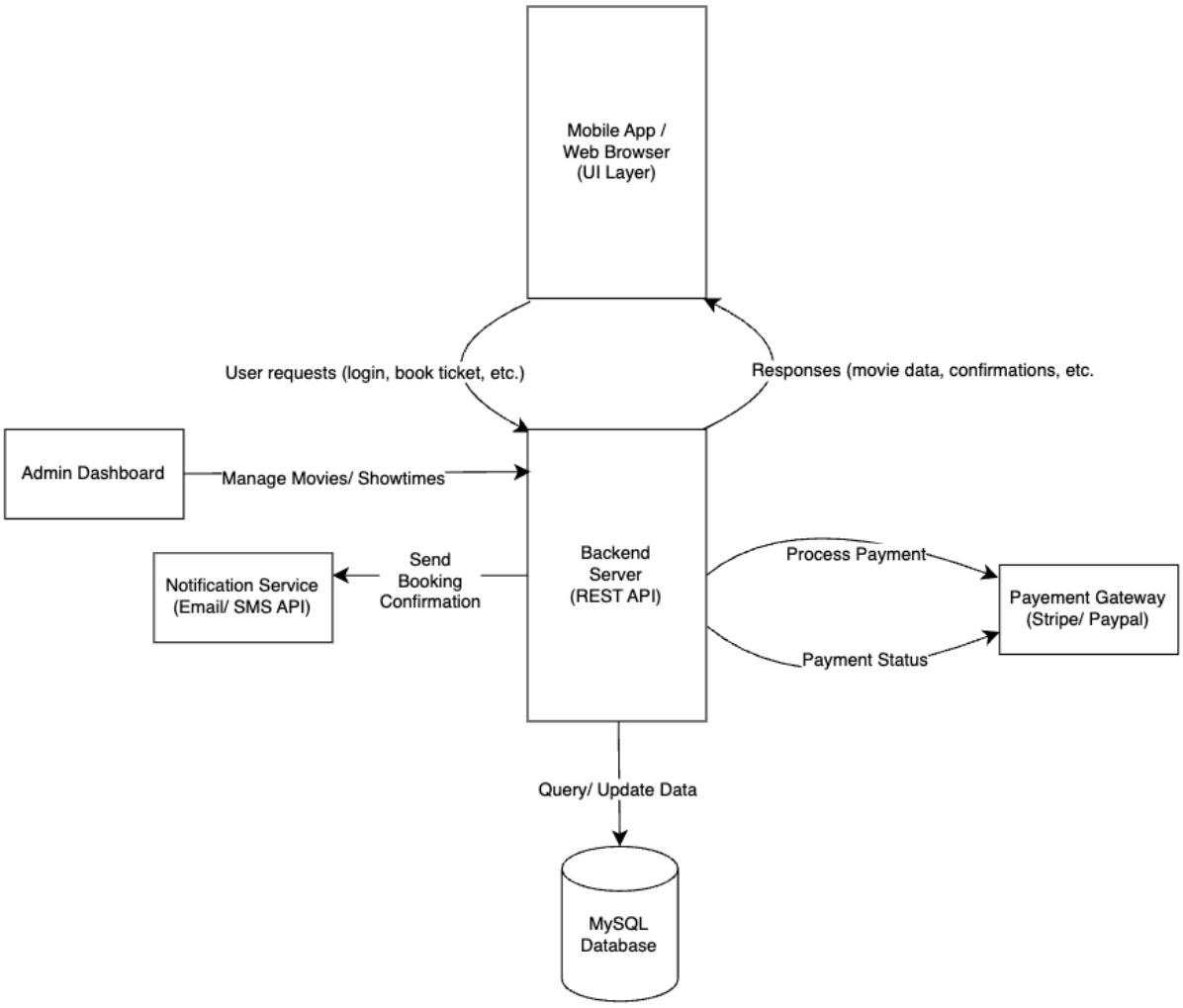
Testing will be conducted in structured phases, allowing sufficient time for identifying and resolving defects before deployment.

Phase	Duration	Team Members
Unit Testing	2 Weeks	Developers
Integration Testing	1 Weeks	QA Engineers
System Testing	2 Weeks	QA Team
Performance Testing	1 Weeks	QA Engineers
Security Testing	1 Weeks	Security Analysts
UAT	1 Weeks	End Users

10. Architecture Design and Data Management

10.1 Updated Software Architecture Diagram

The software architecture diagram represents a multi-layered client-server model for the Movie Ticketing System. At the top layer, users interact with the system through either a mobile app or a web browser, initiating actions like browsing movies, selecting seats, and making payments. These requests are sent to the backend server, which processes logic such as validating seat availability, managing user sessions, and coordinating ticket bookings. The backend connects to a centralized MySQL database to retrieve or store structured information like user profiles, showtimes, bookings, and payment history. It also communicates with third-party services such as Stripe or PayPal for secure payment processing and an external notification service for sending email or SMS confirmations. An admin dashboard is also connected to the backend, providing theater staff the ability to manage schedules, movie listings, pricing, and promotional content in real time.



10.2 Data Management Strategy

The data management strategy diagram illustrates the logical structure of the system's MySQL database using a simplified entity-relationship model. Each rectangle represents a table, with fields listed to show how data is organized. The Users table stores customer account information, while the Movies table contains metadata about each movie. Showtimes are tied to movies and specify when and where each film is playing. When users make bookings, the Bookings table captures the transaction, linking the customer and selected showtime along with seat information. Associated with each booking is a corresponding record in the Payments table, which stores payment method, amount, and status. Additionally, if a user purchases food or drinks, the FoodOrders table records those items and links them back to the booking. These relationships enforce data consistency through foreign keys, ensuring that each booking, payment, and food order is logically tied to a user and showtime.

10.2.1 Database Approach

Our system will utilize a centralized database approach, ensuring a shared pool of related data accessed by multiple components of the application (e.g., user registration, ticket booking, food and drink ordering). This architecture significantly reduces data redundancy and enhances data integrity, in contrast to using isolated data silos.

10.2.2 Database Management System (DBMS)

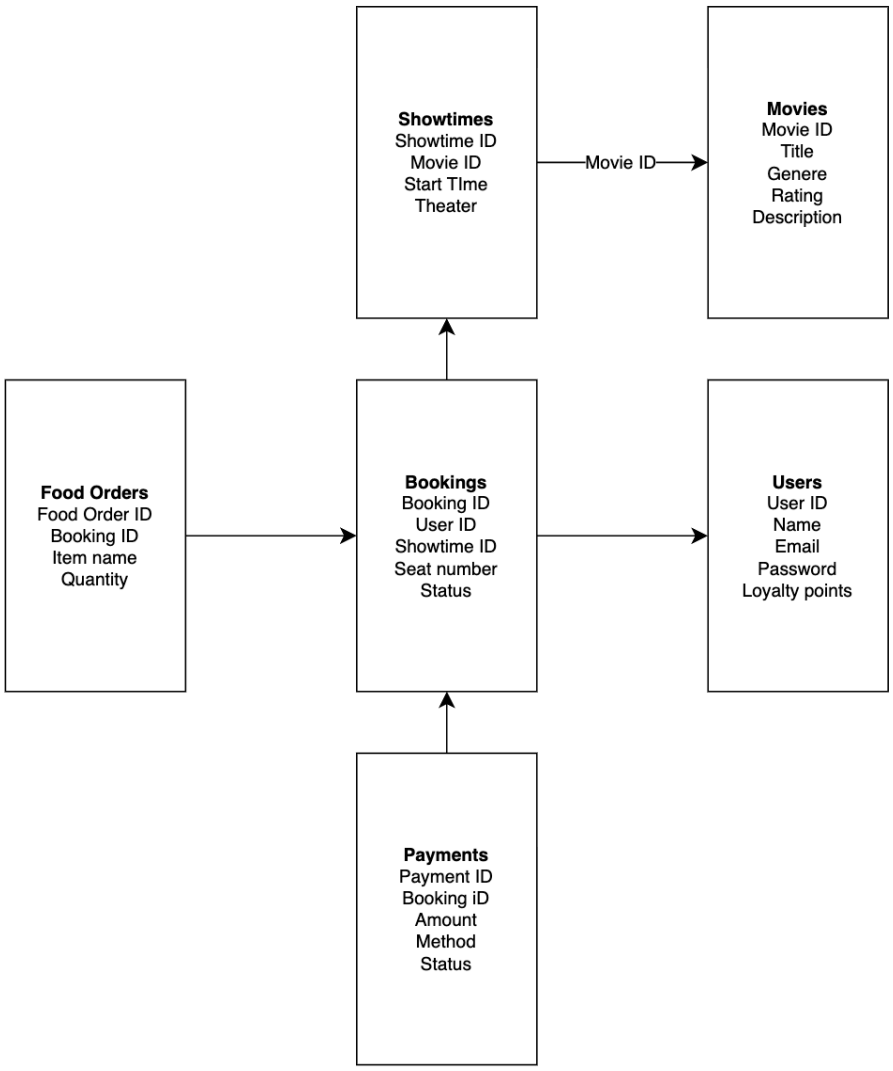
The system will employ a Database Management System (DBMS) as an intermediary between application logic and the physical database. It serves as the interface between the database and the application/backend logic. It is responsible for:

- Managing and manipulating stored data
- Ensuring support for all critical business activities (ticketing, user management, payments)
- Providing concurrency control, data recovery, and security enforcement

10.2.3 Relational Database Model (MySQL)

- The system uses MySQL for relational data management.
- Tables represent entities such as Users, Movies, Showtimes, Tickets, and Transactions.
    - Rows store individual records (e.g., a specific ticket)
    - Columns represent attributes (e.g., movie_title, showtime, seat_number)
- The MySQL schema supports foreign key constraints to ensure referential integrity between tables

10.2.4 Data Modeling



10.2.5 Database Schema and State

- **Database Schema:** Represents the full design of the database (e.g., tables, columns, indexes, keys).
    - The schema changes infrequently.
- **Database State:** Represents the actual content (data) at a given point in time.
    - The state is updated frequently (e.g., after bookings or cancellations).

10.2.6 Structured Query Language (SQL)

- SQL is the primary language used for:
    - Data Definition (DDL): Creating and modifying schema elements (tables, constraints)
    - Data Manipulation (DML): Performing operations like INSERT, UPDATE, DELETE, and SELECT
- SQL is essential for integration with backend logic and automated reporting.

A. Appendices

Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.

Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.

A.1 Appendix 1

A.2 Appendix 2