Eclipse Entertainment

Movie Ticketing System

Software Requirements Specification

Version 3.0

10/21/2025

Group 8
Ali Nasr, Khalid Noman, Savel Moshi

Prepared for
CS 250- Introduction to Software Systems
Instructor: Dr. Gus Hanna
Fall 2025

Revision History

Date	Description	Author	Comments
9/25/2025	Version 1.0	Group 8	Requirements Specification
10/9/2025	Version 2.0	Group 8	Software Design Specification
10/21/2025	Version 3.0	Group 8	SDS: Test Plan

Document Approval

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

Signature	Printed Name	Title	Date
AN, KN, SM	Group 8	Software Eng.	9/25/2025
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this SRS is to keep this hypothetical project in order and maintain an organized plan for this project. The intended audience is for developers and stakeholders.

1.2 Scope

The MTS lets customers browse movies, select showtimes, choose seats, apply discounts, and purchase tickets. It sends digital confirmations by email with QR codes. It supports payment by cards and digital wallets. Admin tools allow staff to manage movies, showtimes, seating, and pricing. The system does not handle concession ordering or physical access control hardware.

1.3 Definitions, Acronyms, and Abbreviations

MTS - Movie Ticketing System.

Showtime - A scheduled screening (movie, auditorium, start time).

Order - A confirmed purchase containing one or more tickets.

Seat Map - Visual layout of seats for an auditorium, including availability.

QR Code - Scannable code representing a digital ticket.

MFA – Multi-factor authentication.

PCI DSS – Payment Card Industry Data Security Standard

PII - Personally Identifiable Information

TLS 1.2 - Security protocol that encrypts and authenticates communication.

1.4 References

All sources came from the course canvas.
https://sdsu.instructure.com/courses/186650/modules/items/5042755
https://sdsu.instructure.com/courses/186650/modules/items/5042756
https://sdsu.instructure.com/courses/186650/modules/items/5042758
https://sdsu.instructure.com/courses/186650/modules/items/5042768
https://sdsu.instructure.com/courses/186650/modules/items/5042770
https://sdsu.instructure.com/courses/186650/modules/items/5042774

1.5 Overview

Section 2 describes the product context and users. Section 3 specifies requirements. Section 4 presents analysis models. Section 5 outlines change management. Appendices include supporting material.

2. General Description

2.1 Product Perspective

MTS is a web application with mobile support. It integrates with payment gateways, email/SMS providers, and identity providers. Admin features are web-based and require authentication and MFA.

2.2 Product Functions

- Browse movies (along with trailers, description of the movie, ratings of the movies)
- Seating options (include select seating for people wheelchair bound/handicapped)
- Checkout (supporting multiple types of payments (Visa, Mastercard, PayPal))
- Email conformation (sending a ticket with a QR code with ticket information (including seating, and theater number))
- Menu (highlighting snacks, beverages, and meals that we serve)

2.3 User Characteristics

Customers are general moviegoers with basic web literacy using phones or desktops. Admins and managers are theater staff trained to maintain listings and schedules. Some users rely on assistive technologies such as screen readers and keyboard navigation.

2.4 General Constraints

Internet access and modern browser is required and all transactions are over HTTPS. It must comply with PCI-DSS for payments and protect PII, so no card data will be stored on MTS servers. Peak loads are expected around popular releases and evenings.

2.5 Assumptions and Dependencies

Assumptions would be that the customers have a valid for of payment, the user has a device that has the proper hardware/software to access the website, accurate movie times with no delay of showings, costumer providing accurate information for such as email, phone number, and proper name. Dependencies would be things such as third-party payment gateways to allow other forms of payments with good security. Theater seating and show time data base being accurate and providing correct seating and timing. Notifications which would confirm the ticket being booked either through email or text. Hardware for the theater such as ticket scanners, projectors, popcorn machines. Legal regulations such as keeping user data private and secure.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The system provides a responsive web interface that adapts to desktop and mobile devices. The interface guides the customer through the purchase process by presenting movies, showtimes, seat maps, ticket types, and payment options in a clear flow. Customers receive order confirmation on screen and can view past orders through an account or email link. Administrators use a secure portal with login and multi-factor authentication to manage movies, showtimes, seating, and pricing.

3.1.2 Hardware Interfaces

Client devices will be smartphones, tablets, laptops/desktops. There will be lobby kiosk or box-office terminal using the same web UI with printer support as well as existing ticket scanners to read generated QR codes.

3.1.3 Software Interfaces

External API's will be utilized for payments allowing for a secure gate way for transactions and email/text notifications giving confirmation of payment and QR codes for the tickets. Allowing users to provide sensitive information in a secure way.

3.1.4 Communications Interfaces

All communication between clients, the server, and external providers uses HTTPS with TLS 1.2 or higher. The system securely exchanges data between the web interface, the server, and third-party services such as payment gateways and email providers. Payment providers send asynchronous notifications through webhooks to confirm successful payments or refunds. All communication is encrypted and protected against interception or tampering.

3.2 Functional Requirements

3.2.1 Ticket Purchase Flow

3.2.1.1 Introduction

This feature provides customers with the ability to browse movies, select showtimes, choose seats, and complete the purchase of tickets online.

3.2.1.2 Inputs

- Selected movie, date, and showtime.
- Seat choices from the seat map.
- Ticket types and quantities (Adult, Child, Senior, Student, Military).
- Payment information (Visa, Mastercard, Paypal).
- Optional discount or promo codes.

3.2.1.3 Processing

- System verifies showtime availability and retrieves seat map.
- Selected seats are held for up to 5 minutes during checkout.
- Ticket prices and discounts are applied.
- Payment details sent securely to the payment gateway.
- On success, the system finalizes the order and generates ticket QR codes.

3.2.1.4 Outputs

- On-screen confirmation of the order.
- Email with receipt and attached digital tickets (PDF/PKPass).
- QR codes for entry scanning.

3.2.1.5 Error Handling

- If payment fails: display error message and allow retry with different method.
- If seat is no longer available: prompt customer to reselect.
- If discount code is invalid/expired: notify customer and continue checkout.

3.2.2 Administration & Showtime Management

3.2.2.1 Introduction

This feature enables theater administrators to manage movies, showtimes, seating, and pricing through a secure admin portal.

3.2.2.2 Inputs

- Admin credentials and MFA token.
- Movie details (title, rating, runtime, synopsis, poster).
- Showtime details (auditorium, start time, date).
- Auditorium seat maps.
- Pricing and discount rules.

3.2.2.3 Processing

- Authenticate admin credentials with MFA.
- Validate and store movies and showtime data.
- Check for overlapping schedules or capacity issues.
- Apply configured pricing and discount rules.
- Update database records and publish changes to the customer portal.

3.2.2.4 Outputs

- Confirmation message to admin for successful updates.
- Updated listings and showtimes visible to customers.
- Real-time seat availability for managed showtimes.

3.2.2.5 Error Handling

- Invalid login attempts result in account lock after 5 failures.
- Overlapping schedule flagged with error; admin must resolve conflict.
- Incomplete or invalid metadata prompts error and prevents publishing.

3.3 Use Cases

3.3.1 Use Case 1: Purchase Tickets Online

Goal: Complete an online ticket purchase.

Actors: Customer, Payment Gateway, Email Service.

Preconditions: Movies and showtimes are available. The customer has internet access.

Main Flow: The customer browses movies and selects a showtime. The customer chooses seats and ticket types. The system calculates the total cost and applies discounts. The customer provides payment details. The system processes the payment through the payment gateway. On success, the system generates tickets and emails them to the customer.

Postconditions: The customer receives confirmation and digital tickets.

Exceptions: Invalid payment, expired discounts, or unavailable seats.

3.3.2 Use Case 2: Retrieve Past Orders

Goal: Allow a customer to view and access previously purchased tickets.

Actors: Customer, System.

Preconditions: The customer has completed at least one order. The system stores order history.

Main Flow: The customer logs in or uses an email link. The system retrieves past orders. The customer selects an order to view. The system displays ticket details and provides download links for digital tickets.

Postconditions: The customer can access and present tickets for entry.

Exceptions: Invalid login, expired retrieval link, or missing order data.

3.3.3 Use Case 3: Cancel a Ticket

Goal: Cancel an existing ticket within the allowed time window.

Actors: Customer, System, Payment Gateway.

Preconditions: A valid ticket exists and the cancellation policy allows refund.

Main Flow: The customer requests a cancellation. The system verifies policy rules. The system releases the seats. The system processes a refund through the payment gateway. The system emails confirmation of cancellation to the customer.

Postconditions: The ticket is invalidated and the refund is initiated.

Exceptions: Missed cancellation deadline or payment gateway failure.

3.4 Classes / Objects

3.4.1 Movie

The Movie class stores the id, title, rating, runtime, synopsis, and poster URL.

3.4.2 Showtime

The Showtime class stores the id, movie id, auditorium id, start time, format, base prices, and status.

3.4.3 Auditorium

The Auditorium class stores the id, name, and seat map with rows, seat numbers, and accessibility flags.

3.4.4 Ticket

The Ticket class stores the id, order id, showtime id, seat, type, price, and QR code.

3.4.5 Order

The Order class stores the id, customer email, total, status, payment reference, and associated tickets.

3.4.6 DiscountRule

The DiscountRule class stores the id, type of discount such as student, senior, military, child, or promo, and the terms for eligibility.

3.5 Non-Functional Requirements

3.5.1 Performance

Ninety-five percent of pages load in under two seconds with two hundred users. Seat-hold updates complete within three hundred milliseconds. The system supports one thousand concurrent users without issues.

3.5.2 Reliability

Payment and order creation are crucial and occur right away. The system performs daily database backups.

3.5.3 Availability

The purchase flow is available ninety-nine point five percent of the time per month. There is a forty-eight hour notice before any planned downtime.

3.5.4 Security

No card data is stored and the system is PCI-DSS compliant. The admin portal requires MFA.

3.5.5 Maintainability

The code follows style guidelines and is documented. Tests are ran constantly. Critical errors are resolved in less than forty-eight hours.

3.5.6 Portability

The system runs on Linux-based cloud platforms. It works with modern evergreen browsers. The interface is responsive for mobile and desktop screens.

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

3.7 Design Constraints

Specify design constrains imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.

3.8 Logical Database Requirements

Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.

3.9 Other Requirements

Catchall section for any additional requirements.

4. Analysis Models

List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS's requirements.

4.1 Sequence Diagrams

4.3 Data Flow Diagrams (DFD)

4.2 State-Transition Diagrams (STD)

5. Change Management Process

Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.

6. Software Design Specification

6.1 System Description

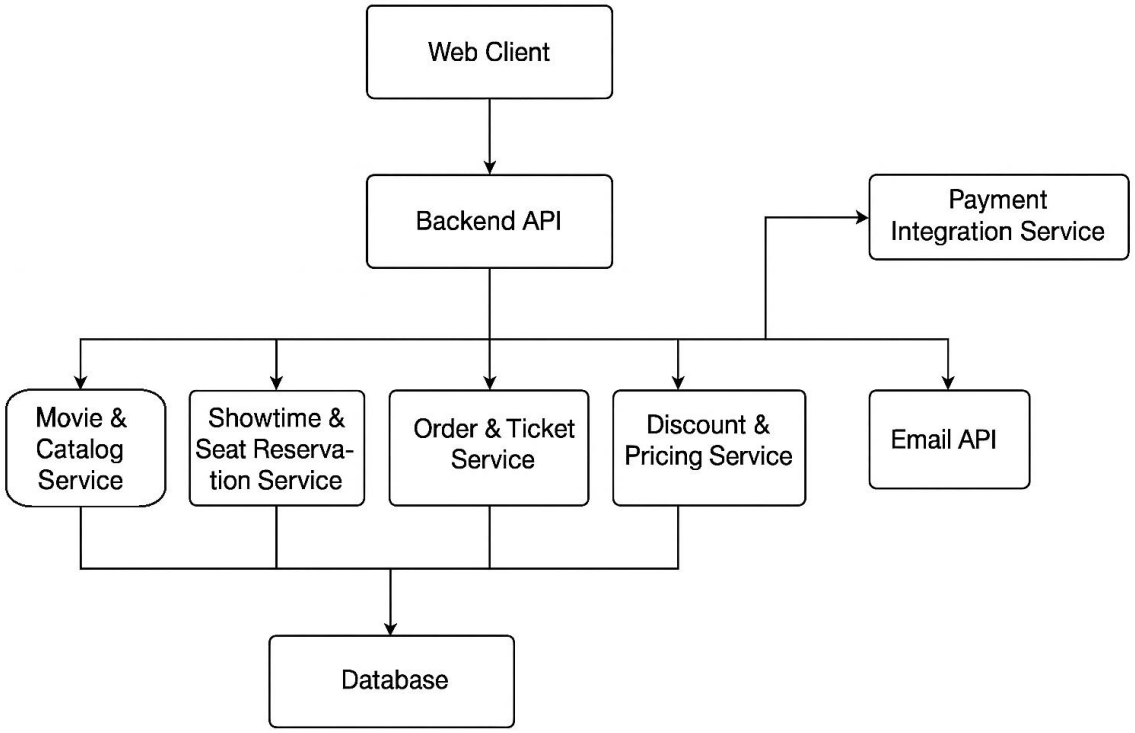
Movie Theater Ticketing system is an online application that allows customers to browse films that are currently playing, look at showtimes, select their seating, and purchase tickets online. Customers can log in and receive a digital confirmation of their orders. Admins can manage the system by entering new films, scheduling showtimes, updating availability for seating, and properly allocating promotional discounts. The architecture is deployed as a client-server web application that pairs an adaptive front-end user interface with a backend RESTful API. The API handles the business logic, interacts with a relational database for persistent data storage, and interfaces with third-party services. This allows for the architecture to be scalable, modular, and have secure data between clients, servers, and third parties.

6.2 Software Architecture Overview

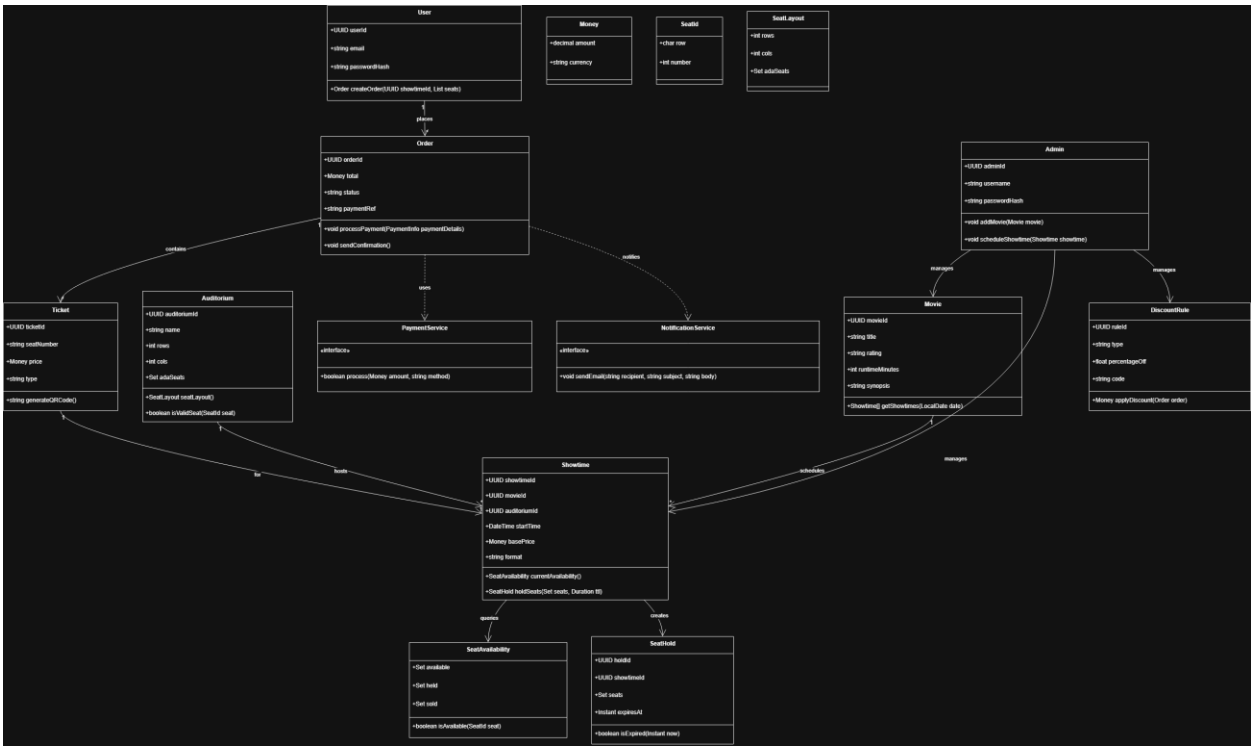
6.2.1 Components

- Web Client (UI): Front-end that allows customers to interact with our platform to look at showtimes, select seating, view movie previews, and purchase tickets. (HTML, JavaScript, CSS, or React)
- Backend API: Connects the user interface to the database to allow for the main functions of the system to be executed. This is divided into different modules for specific tasks
- Movie & Catalog Service: Manages movie data
- Showtime & Seat Reservation Service: Seating arrangements and updating availability
- Order & Ticket Service: Process ticket purchases, providing confirmation and specific information regarding seat and theater number
- Discount & Pricing Service: Calculates the final cost of a ticket while applying promotional discounts
- Payment Integration Service: Third-party payment gateway securing user information
- Notification Service: Communicating order confirmation along with user ticket information
- Database: Holds the information regarding user accounts, seating availability, tracking orders, discounts, and movie details.
- External Services: A Payment Gateway that securely processes payments Email service for confirmations that relays ticketing information back to the customer

6.2.2 Architecture Diagram



6.3 UML Class Diagram



6.4 Class & Operation Description

6.4.1 Movie

Attributes:

- movieId: UUID
- title: string
- rating: string
- runtimeMinutes: int
- synopsis: string
- posterUrl: string

Operations:

- getShowtimes(date: LocalDate): Showtime[]

6.4.2 Auditorium

Attributes:

- auditoriumId: UUID
- name: string
- rows: int
- cols: int

- adaSeats: Set<SeatId> (accessible seats)

Operations:

- seatLayout(): SeatLayout
- isValidSeat(seat: SeatId): boolean

6.4.3 SeatLayout (Value Object)

Attributes:

- rows: int
- cols: int
- adaSeats: Set<SeatId>

Operations:

- allSeats(): Set<SeatId>

6.4.4 Showtime

Attributes:

- showtimeId: UUID
- movieId: UUID
- auditoriumId: UUID
- startTime: DateTime
- format: string (e.g., "IMAX", "3D")
- basePrice: Money
- status: enum { Scheduled, OnSale, Canceled, SoldOut }

Operations:

- currentAvailability(): SeatAvailability
- holdSeats(seats: Set<SeatId>, customerRef: UUID, ttl: Duration): SeatHold
- releaseHold(holdId: UUID): void

6.4.5 SeatAvailability (Value Object)

Attributes:

- available: Set<SeatId>
- held: Set<SeatId>
- sold: Set<SeatId>

Operations:

- isAvailable(seat: SeatId): boolean

6.4.6 SeatHold

Attributes:

- holdId: UUID
- showtimeId: UUID
- seats: Set<SeatId>
- expiresAt: Instant

Operations:

- isExpired(now: Instant): boolean

6.4.7 Order

Attributes:

- orderId: UUID
- email: string
- createdAt: Instant
- status: enum { Pending, Paid, Canceled, Refunded }
- total: Money
- paymentRef: string

Operations:

- addTicket(ticket: Ticket): void
- calculateTotal(discounts: DiscountRule[], ctx: PricingContext): Money
- markPaid(paymentRef: string): void
- cancelAndRefund(reason: string): void

6.4.8 Ticket

Attributes:

- ticketId: UUID
- orderId: UUID
- showtimeId: UUID
- seat: SeatId
- type: enum { Adult, Child, Senior, Student, Military }
- price: Money
- qrCode: string (URL or base64)

Operations:

- renderPdf(): URL
- asPkPass(): URL

6.4.9 DiscountRule

Attributes:

- ruleId: UUID
- type: enum { Student, Senior, Military, Child, Promo }
- percentageOff: decimal(5,2)
- amountOff: Money
- code: string
- eligibility: Json (e.g., minAge, validDays)

Operations:

- applies(ctx: PricingContext): boolean
- apply(subtotal: Money, ctx: PricingContext): Money

6.4.10 User

Attributes:

- userId: UUID
- email: string
- passwordHash: string
- rewardsPoints: int

Operations:

- awardPoints(points: int): void
- redeem(points: int): void
- orders(): Order[]

6.4.11 Admin

Attributes:

- adminId: UUID
- username: string
- passwordHash: string
- mfaSecret: string

Operations:

- createMovie(m: Movie): void
- scheduleShowtime(s: Showtime): void
- createDiscount(rule: DiscountRule): void

6.4.12 PaymentService (Interface)

Operations:

- authorizeAndCapture(amount: Money, tokenizedMethod: string, idempotencyKey: string): PaymentResult
- refund(paymentRef: string, amount: Money, reason: string): RefundResult

6.4.13 NotificationService (Interface)

Operations:

- sendOrderConfirmation(orderId: UUID, email: string): void
- sendTickets(orderId: UUID, email: string): void

6.5 Database Design

6.5.1 Schematics

Movies (Stores movie details.)

- movie_id: INT, Primary Key, AUTO_INCREMENT
- title: VARCHAR(100), NOT NULL
- rating: VARCHAR(10), NULLABLE
- runtime: INT, NOT NULL
- synopsis: TEXT, NULLABLE

Auditoriums (Contains information about each screening room.)

- auditorium_id: INT, Primary Key, AUTO_INCREMENT
- name: VARCHAR(50), NOT NULL
- rows: INT, NOT NULL
- cols: INT, NOT NULL

Showtimes (Represents scheduled showings of a movie.)

- showtime_id: INT, Primary Key, AUTO_INCREMENT
- movie_id: INT, Foreign Key → movies(movie_id)
- auditorium_id: INT, Foreign Key → auditoriums(auditorium_id)
- start_time: DATETIME, NOT NULL
- base_price: DECIMAL(5,2), NOT NULL
- status: VARCHAR(20), DEFAULT ‘active’

Orders (Represents a customer order for one or more tickets.)

- order_id: INT, Primary Key, AUTO_INCREMENT
- email: VARCHAR(100), NOT NULL
- total: DECIMAL(7,2), NOT NULL

- status: VARCHAR(20), DEFAULT ‘pending’
- payment_ref: VARCHAR(100), NULLABLE

Tickets (Each record represents a purchased seat for a showtime.)

- ticket_id: INT, Primary Key, AUTO_INCREMENT
- order_id: INT, Foreign Key → orders(order_id)
- showtime_id: INT, Foreign Key → showtimes(showtime_id)
- seat_row: INT, NOT NULL
- seat_num: INT, NOT NULL
- type: VARCHAR(20), NOT NULL (e.g., adult, child, senior)
- price: DECIMAL(5,2), NOT NULL

Discount_Rules (Defines available promotional discounts.)

- rule_id: INT, Primary Key, AUTO_INCREMENT
- type: VARCHAR(50), NOT NULL
- percentage_off: DECIMAL(4,2), NOT NULL
- code: VARCHAR(50), NULLABLE

Users (Stores registered customer account information.)

- user_id: INT, Primary Key, AUTO_INCREMENT
- email: VARCHAR(100), UNIQUE, NOT NULL
- password_hash: VARCHAR(255), NOT NULL
- rewards_points: INT, DEFAULT 0

Admins (Contains administrator account information.)

- admin_id: INT, Primary Key, AUTO_INCREMENT
- username: VARCHAR(50), UNIQUE, NOT NULL
- password_hash: VARCHAR(255), NOT NULL
- mfa_secret: VARCHAR(255), NULLABLE

6.5.2 Constraints

- UNIQUE(showtime_id, seat_row, seat_num) → Prevents duplicate seat bookings.
- INDEX(showtimes.start_time) → Speeds up queries for upcoming showtimes.
- UNIQUE(users.email) → Ensures each user has a distinct email address.
- UNIQUE(admins.username) → Prevents duplicate admin logins.

6.6 API Endpoints

6.6.1 Public Endpoints

- GET /movies – list movies
- GET /showtimes/{id} – showtime details & seat availability
- POST /showtimes/{id}/holds – hold seats temporarily

- POST /orders – create order & process payment
- GET /orders/{id} – get order/ticket details

6.6.2 Admin Endpoints

- POST /movies – add movie
- POST /showtimes – schedule showtime
- POST /discounts – create discount rules

6.7 Development Plan & Timeline

6.7.1 Partition of Tasks

- All: Push to GitHub
- Khalid: Architecture Diagram, System Description, API Endpoints
- Savel: UML Class Diagram, Class Description, Compile Final PDF
- Ali: Database Design, Development Plan & Timeline

6.7.2 Timeline

- Day 1-2: Draft Diagram, Database Sketch
- Day 3: Write out text
- Day 4: Integrate diagrams and text into SRS
- Day 5: Final Review, export pdf, push to GitHub

7. Test Plan

7.1 Scope and Objectives

The goal of this test plan is to verify that the Movie Ticketing System functions according to the requirements.
Testing will focus on:

- Seat availability and hold system
- Pricing and discounts
- Ticket purchasing process
- Admin showtime scheduling
- Refund and cancellation flows

Three levels of testing will be applied: Unit, Functional, and System.

7.2 Test Sets Overview

ID	Test Level	Description	Feature
U1	Unit	Seat hold and availability logic	Seat Availability / Hold

U2	Unit	Pricing and discounts	Pricing / Discounts
F1	Functional	Seat hold → order commit flow	Ticket Purchase
F2	Functional	Admin showtime scheduling validation	Admin Scheduling
S1	System	Full checkout with payment and email QR	Purchase Flow (E2E)
S2	System	Refund and seat restoration	Refund & Cancellation

7.3 Test Deliverables

Deliverable	Description	Location
SRS Document	Updated with Section 7 (Test Plan)	GitHub
Test Plan Document	Standalone PDF version of this section	GitHub
Excel Test Cases	10 fully detailed test cases	GitHub

7.4 Test Plan Responsibilities

Member	Responsibility	Contribution
Khalid Noman	Functional test sets (F1, F2)	Test Plan / Excel Commit

Savel Moshi	Unit test sets (U1, U2)	Test Plan / Excel Commit
Ali Nasr	System test sets (S1, S2)	Test Plan / Excel Commit

7.5 Exit Criteria

- All P0 and P1 test cases pass successfully.
- No blocking defects remain.
- All required deliverables are pushed to GitHub with recorded commit links.

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