Movie Theater Ticketing Service

Software Requirements Specification

2.4

14 September 2025

Group 7

Matthew Gallenburger, Nikola Milosavljevic, Ryan Schindler

Prepared for

CS 250- Introduction to Software Systems

Instructor: Dr. Gus Hanna

Fall 2024

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
| 9/13 | 1.0 Use Cases & FRs | NM | 3.2-3.4, Section 3 MT and RTC |
| 9/14 | 1.1 NFRs | RS | 2.4, 2.5, 3.5 completed |
| 9/18 | 1.2 User & Context | MG | 1.1-2.3 |
| 9/19 | 1.3 Proofreading | RS | 3.1, Proofreading |
| 10/1 | 2.1 Software Design Specs | RS | SDS made, TOC updated |
| 10/7 | 2.2 System Architecture & UML Class Diagram | MG | Class diagram and system architecture diagram made with their descriptions below. |
| 10/7 | 2.3 Development Plan & Timeline | NM | Sections 6.1, 6.4, 6.5, 6.6 completed |
| 10/21 | |  | | --- | | 3.1 Test Plan Section |  |  | | --- | |  | | NM | Added Section 7 Test Plan (Assignment 3) |
| 10/22 | 3.2 Updated Diagrams | MG | Added the updated UML Class diagram and descriptions Movie and User classes of them below. |

# Document Approval

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
| *Ryan Schindler* | Ryan Schindler | Software Eng. | 9/19/2025 |
|  | Dr. Gus Hanna | Instructor, CS 250 |  |
|  |  |  |  |

**Table of Contents**

Revision History iii

Document Approval iii

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms, and Abbreviations 1

1.4 References 1

1.5 Overview 2

2. General Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Characteristics 2

2.4 General Constraints 3

2.5 Assumptions and Dependencies 3

3. Specific Requirements 1

3.1 External Interface Requirements 4

3.1.1 User Interfaces 4

3.1.2 Hardware Interfaces 4

3.1.3 Software Interfaces 5

3.1.4 Communications Interfaces 5

3.2 Functional Requirements 5

3.2.1 <Functional Requirement or Feature #1> 5

3.2.2 <Functional Requirement or Feature #2> 5

3.3 Use Cases 7

3.3.1 Use Case #1 7

3.3.2 Use Case #2 8

3.4 Classes / Objects 10

3.4.1 <Class / Object #1> 10

3.4.2 <Class / Object #2> 10

3.5 Non-Functional Requirements 12

3.5.1 Performance 12

3.5.2 Reliability 12

3.5.3 Availability 12

3.5.4 Security 12

3.5.5 Maintainability 13

3.5.6 Portability 13

3.6 Inverse Requirements 13

3.7 Design Constraints 13

3.8 Logical Database Requirements 13

3.9 Other Requirements 13

4. Analysis Models 13

4.1 Sequence Diagrams 14

4.3 Data Flow Diagrams (DFD) 14

4.2 State-Transition Diagrams (STD) 14

5. Change Management Process 14

6. DEsign Development Plan 13

6.1 Roles and Responsibilities 13

6.1.1 Team Members 12

6.1.2 Partitioning of Tasks 12

6.2 Architecture 13

6.3 Classes 13

6.3.1 Screening 12

6.3.2 Auditorium 12

6.3.3 Seat 12

6.3.4 Order 12

6.3.5 Ticket 12

6.3.6 Payment 12

6.3.7 User Roles 12

6.3.8 Policy Values 12

6.3.9 User 12

6.3.10 Movie 12

6.4 Timeline 13

6.5 Workflow and Submission 13

6.6 Risks and Mitigations 13

A. Appendices 23

A.1 Appendix 1 24

A.2 Appendix 2 24

# 1. Introduction

## 1.1 Purpose

*The purpose of this Software Requirements Specification (SRS) is to plan the requirements for an Online Movie Ticketing and Screening System. This document is intended for use by system developers, project stakeholders, theater managers and instructors reviewing the project. It will articulate and provide enough detail for the software engineers to design, implement and test the system.*

## 1.2 Scope

*The Online Movie Ticketing and Screening System will allow potential customers to browse movies that the theater is showing, reserve select seats, and purchase tickets securely through an external payment provider. Customers that purchase a ticket through the system will be sent an e-ticket containing an individual barcode, which will be validated upon entry. Customers will also have the ability to cancel or reschedule their reservations that fall in line with the theater’s refund policy: full refund for cancellations that are made at least 24 hours before the time of the showing, with no refunds after that time has expired.*

*The system will provide managers and authorized employees of the theater with tools to schedule screenings and manage theater availability. While at checkout, customers seats will be held for 8 minutes in order to prevent conflicts with other customers selecting those seats. This product will improve customer convenience, streamline ticket processes, and enhance efficiency for theater staff.*

## Definitions, Acronyms, and Abbreviations

* + 1. *E-Ticket: A digital ticket issued by the system that contains an individual barcode for entry.*

*1.3.2 Seat Hold: Temporary reservation of a customer selected seat for 8 minutes while the payment information is being processed.*

* + 1. *Refund Policy: A rule the specifies refund eligibility (full refund if cancelled less than 24 hours before the showing of the movie, otherwise, no refund will be issued).*
    2. *UC: Use Case*
    3. *External Payment Provider: A third-party financial institution that processes transactions.*
    4. *FR: Functional Requirement*

## References

* + 1. *CS250-SRS Template (provided by instructor)*
    2. *Lecture notes on use cases (lecture-use-cases.pdf, provided by instructor)*

*1.4.3 Group assumptions defined during in-class client interviews*

## Overview

*1.5.1 Section 2: Provides a general description of the system which includes product description, product functions, user characteristics, general constraints, assumption, and dependencies.*

* + 1. *Section 3: Provides specific requirements for the system such as external interface requirements, which also includes user interfaces, hardware interfaces, software interfaces, and communication interfaces. This section will also give a description of both functional and non-functional requirements and outline use cases.*
    2. *Section 4: Provides analysis models such as sequence diagrams.*
    3. *Section 5: Describes the change management process.*

# 2. General Description

## 2.1 Product Perspective

*The Online Movie Ticketing and Screening System is a standalone web-based application integrated with a single external payment provider. It will replace the traditional process of an in-person and phone-based system required for purchasing movie tickets. The online system will streamline ticket sales, making the process of facilitating sales more accessible for both customers and employees. The system will integrate with the theater’s infrastructure that will allow the ability to scan and verify customer’s e-tickets.*

## 2.2 Product Functions

*The Online Movie Ticketing and Screening System will provide the following functions:*

* + 1. *Purchase Ticket: Customers can select movies, showtimes, and seats, then pay through a third-party payment provider.*
    2. *Seat Hold: While the customer is checking out, selected seats will be held for 8 minutes while they enter in payment information.*
    3. *Cancel/Reschedule Ticket: Customers will have the ability to cancel or reschedule a reservation which is subject to the theater’s refund policy.*
    4. *E-Ticket Generation: The system will produce a unique and scannable e-ticket.*

*2.2.5 Payment Processing: Securely handle transactions via a third-party payment provider.*

## 2.3 User Characteristics

*2.3.1 Customers: Moviegoers of any age, using a device that could include a smartphone, tablet or traditional desktop computer. The system must be user friendly and scale to a mobile device properly.*

*2.3.2 Theater Managers: Manager and authorized employees that are responsible for scheduling screenings. They require administrative access with simple but flexible management tools.*

*2.3.3 Theater Staff (Scanning Tickets): Employees validating tickets at the entrance. These people will need reliable scanning functionality.*

## General Constraints

*2.4.1 Seats holds while a customer is checking out will expire after 8 minutes if the purchase was not processed.*

* + 1. *Refunds are only issued if the request happens no later than 24 hours before the screening takes place.*
    2. *All payments must be processed through the designated external payment provider.*
    3. *The system must generate a unique e-ticket issued to the customer to prevent duplication or any other form of fraud.*

## Assumptions and Dependencies

* + 1. *Customers have internet-enabled devices that are able to access the system.*
    2. *The theater has the necessary hardware for scanning and processing e-tickets.*
    3. *The external payment provider remains functional and available.*

*2.5.4 Theater staff and managers will be trained and able to use the system effectively.*

**Actor List:**

*Customer (Primary Actor):*

* *Purchases, cancels, and reschedules tickets.*
* *Receives e-ticket for entry to the theater.*

*Theater Manager (Primary Actor):*

* *Schedules and manages movie screenings.*
* *Monitors ticket sales and availability.*

*Theater Staff (Supporting Actor):*

* *Scans customers’ e-tickets prior to entry.*

*External Payment Provider (Supporting Actor):*

* *Facilitates secure processing for customer payments.*

# 3. Specific Requirements

For the movie ticketing website, we select three critical use cases (UCs) that inform the specific requirements that the service needs: purchasing a ticket, adjusting a sold ticket, and scheduling the movie screenings. From these three UCs, we then identify 15 functional requirements (FRs) that are necessary for these UCs to function. In turn, a recommended list of class names and properties are provided for design. A minimum traceability list and a requirement traceability matrix are listed below for quick reference, and a UC diagram is included as figure 1.

Limitations of both the project and the users must be kept in mind to ensure a quick, hassle-free experience for both the customers and the employees. The external interface requirements capture the requirements for the website design and layout, whereas the non-functional requirements give strict, testable parameters for user experience.

Minimal Traceability (per UC)

Quick list of related FRs per UC.

* **UC-1 Purchase Ticket →** FR-001, FR-002, FR-003, FR-004, FR-005, FR-006, FR-007, FR-008, FR-011, FR-012, FR-014
* **UC-2 Cancel/Reschedule →** FR-008, FR-009, FR-004, FR-005, FR-006, FR-013, FR-014
* **UC-3 Schedule Screening →** FR-010, FR-015

Requirement Traceability Matrix (RTM)

| Use Case | Related FRs | Priority |
| --- | --- | --- |
| **UC-1 Purchase Ticket** | FR-001, FR-002, FR-003, FR-004, FR-005, FR-006, FR-007, FR-008, FR-011, FR-012, FR-014 | HIGH |
| **UC-2 Cancel/Reschedule** | FR-008, FR-009, FR-004, FR-005, FR-006, FR-013, FR-014 | HIGH |
| **UC-3 Schedule Screening** | FR-010, FR-015 | MEDIUM |

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

There will be one joint user interface for both the customer and employee for the website. The website will be designed in an easy-to-use and understand website format that minimizes the number of menus needed to navigate to two at most. The employee version will be secure; the employee will have to enter their credentials into the system to log in, and they will be given screening editing privileges depending on whether they are a manager or not.

### 3.1.2 Hardware Interfaces

The website will primarily be used on computer and laptop hardware interfaces; however, the website can be accessed on mobile devices as well. The software will utilize a responsive web browser, freeing the requirement to be designed for one type of screen. However, the interface will be tailored to a 1920 x 1080 pixel resolution, which is commonly used in most computer screens.

### 3.1.3 Software Interfaces

The software interface will be designed primarily for Chrome browsers and the three most popular browsers (3.5.6). All code must be designed and optimized for these browsers.

The system will also interface with the SQL databases housing information on customer accounts and movie screenings (FR 3.2.1-4,7-9). These will be rented through the cloud provider Azure. There must also be a logging interface to an audit log so every transaction and adjustment is recorded on the system (FR 3.2.14).

### 3.1.4 Communications Interfaces

The communication interface will send emails to inform and remind the customers when tickets are created, rescheduled, or cancelled (FR 3.2.7,8). The software will work together with a built-in API for automated emailing.

## 3.2 Functional Requirements

### *Catalog of 15 testable FRs. Each ties back to at least one UC.*

### 3.2.1 Screening Listings

The system shall list available screenings by date, movie, auditorium, and

format for customer browsing.

### 3.2.2 Seat Map

The system shall display a **real-time seat map** for a selected screening,

indicating Available/Held/Sold states.

### 3.2.3 Selection

The system shall allow customers to select seat(s) and ticket type(s) for a

screening and shall enforce seating rules (e.g., accessibility blocks, avoid orphan single

seats).

### 3.2.4 Seat Holding

The system shall **hold** selected seats for **≥ 8 minutes** once checkout begins

and release those seats on payment failure, cancellation, or hold expiry.

### 3.2.5 Summary

The system shall calculate and present an order summary including ticket

prices, fees, and taxes before payment.

### 3.2.6 Payment Processing

The system shall process payments via the external Payment Service and

update order status according to the authorization result.

### 3.2.7 Ticket Creation

The system shall issue an **electronic ticket** with a unique order ID and

scannable code upon successful purchase and shall send confirmations for purchase,

cancellation, and reschedule.

### 3.2.8 Order Retrieval

The system shall allow customers to **retrieve orders** using email and order ID.

### 3.2.9 Cancellation and Reschedule

The system shall allow customers to **cancel or reschedule** within policy

constraints and shall apply the correct **refund/price-difference** outcome.

### 3.2.10 Manager Privileges

The system shall allow authorized managers to **create/edit/publish/unpublish**

screenings and shall detect **scheduling conflicts** for the same auditorium.

### 3.2.11 Ticket Validation and Reuse

The system shall **validate tickets at entry** and prevent **re-use** of the same

ticket code.

### 3.2.12 Sales Moderation

The system shall prevent **overselling** by applying atomic updates to seat and

order states.

### 3.2.13 Cancellation and Reschedule Confirmation

The system shall display **cancellation/reschedule eligibility** and the calculated

refund/credit before the user confirms the action.

### 3.2.14 Audit Log

The system shall record an **audit log** for order state changes and ticket

validations, including timestamp and actor.

### 3.2.15 Policy Configuration

The system shall allow authorized managers to **configure policy values** used

by the system (e.g., seat-hold duration, refund window).

## 3.3 Use Cases

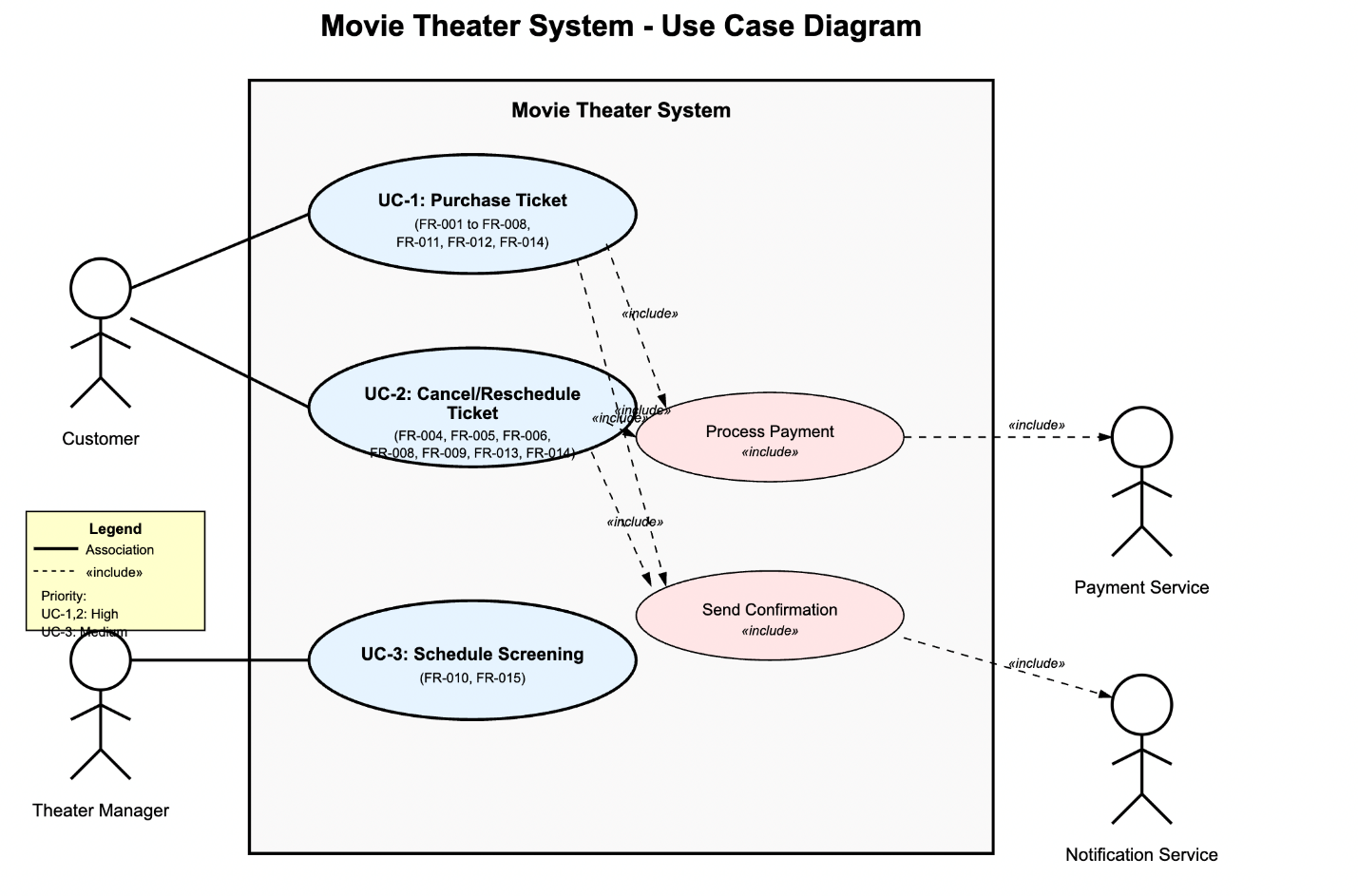


Figure 1. Movie Theater Ticketing System – Use Case Diagram

### 3.3.1 Purchase Ticket

**3.3.1.1 Primary Actor:** Customer  
**3.3.1.2 Supporting Actors:** Payment Service (authorization), Notification Service (confirmation delivery)  
**3.3.1.3 Scope:** Movie Theater Ticketing System  
**3.3.1.4 Level:** User goal

**3.3.1.5 Preconditions**  
1. At least one screening is published and has capacity.  
2. System clock and payment integration are operational.

**3.3.1.6 Trigger**  
Customer decides to buy a ticket for a specific screening.

**3.3.1.7 Main Success Scenario (Basic Flow)**  
1. Customer selects a date/movie to view available screenings.  
2. System displays showtimes and high-level info (time, auditorium, format, price from).  
3. Customer selects a screening; system shows the real-time seat map.  
4. Customer selects seat(s) and ticket type(s).  
5. System places selected seats on **hold** and displays order summary with prices/fees/taxes.  
6. Customer proceeds to payment; system submits the charge to Payment Service.  
7. Payment Service authorizes payment; system marks seats **sold**, generates **e-ticket(s)**, and records the order.  
8. System sends confirmation with ticket(s) to the customer.  
9. Customer can retrieve the order later using email + order ID.

**3.3.1.8 Alternate / Exception Flows**  
A1. **Seat taken mid-process:** If a chosen seat becomes unavailable before hold is applied, system informs the customer and prompts reselection; no seats are held.  
A2. **Hold expiry before payment:** If the 8-minute window elapses before payment, system releases held seats and returns to seat selection.  
A3. **Payment failure/decline:** On failure, system releases holds and shows an error with option to retry.  
A4. **Payment timeout/unknown result:** System shows pending status; if not confirmed within the configured timeout, treat as failure and release seats.  
A5. **Policy/seat-rule violation:** If a selection violates auditorium rules (e.g., leaves orphan single seats, accessibility blocks), system rejects the selection with guidance.

**3.3.1.9 Postconditions**  
- **Success:** Order status = **CONFIRMED**; seats marked **SOLD**; e-ticket issued; confirmation sent; audit log written.  
- **Failure:** No charge captured; any holds released; inventory unchanged.

### 3.3.2 Cancel or Reschedule Ticket

**3.3.2.1 Primary Actor:** Customer  
**3.3.2.2 Supporting Actors:** Payment Service (refund/charge diff), Notification Service (confirmation)  
**3.3.2.3 Scope:** Movie Theater Ticketing System  
**3.3.2.4 Level:** User goal

**3.3.2.5 Preconditions**  
1. A confirmed order exists.  
2. The request is within policy windows defined by the theater.

**3.3.2.6 Trigger**  
Customer chooses to cancel or reschedule an existing order.

**3.3.2.7 Main Success Scenario (Cancel)**  
1. Customer retrieves the order by email + order ID.  
2. System shows cancellation eligibility and refund amount per policy.  
3. Customer confirms cancellation.  
4. System cancels the order, releases seats, and initiates refund via Payment Service.  
5. System sends cancellation & refund confirmation; audit log recorded.

**3.3.2.8 Alternate Success Scenario (Reschedule)**  
RS1. Customer opts to **reschedule** instead of cancel.  
RS2. System lists eligible alternative screenings (same movie or theater policy).  
RS3. Customer selects new screening and seats; system re-applies hold and recalculates totals.  
RS4. If the new price is higher, system collects the difference; if lower, system issues a credit/refund per policy.  
RS5. System finalizes the new order, releases prior seats, and sends updated confirmation; audit log recorded.

**3.3.2.9 Exception Flows**  
E1. **Outside policy:** System declines the cancellation/reschedule and explains the policy window.  
E2. **No seats available for reschedule:** System allows the customer to keep the current booking or cancel (if eligible).  
E3. **Payment failure on price increase:** System keeps the original booking intact and provides retry.

**3.3.2.10 Postconditions**  
- **Success (Cancel):** Order status = **CANCELLED**; seats returned to inventory; refund issued per policy; audit log written.  
- **Success (Reschedule):** Original seats released; new order **CONFIRMED**; refund/charge handled; audit log written.  
- **Failure:** Original order remains unchanged.

### 3.3.3 Schedule Screening (Manager Function)

**3.3.3.1 Primary Actor:** Theater Manager  
**3.3.3.2 Supporting Actors:** —  
**3.3.3.3 Scope:** Movie Theater Ticketing System  
**3.3.3.4 Level:** User goal

**3.3.3.5 Preconditions**  
1. Movie and auditorium configurations exist, including seat map and capacity.  
2. Manager is authenticated and authorized.

**3.3.3.6 Trigger**  
Manager decides to create or edit a screening.

**3.3.3.7 Main Success Scenario**  
1. Manager creates a screening with: movie, auditorium, date/time, capacity/seat map, base price, attributes (e.g., 3D/IMAX).  
2. System validates the screening (time window, capacity) and checks for **auditorium conflicts**.  
3. Manager sets the screening status to **Published** to make it visible for purchase.

**3.3.3.8 Alternate / Exception Flows**  
A1. **Conflict detected:** System prevents saving and explains the conflict (overlap in the same auditorium).  
A2. **Invalid capacity/seat map:** System rejects the change until fixed.  
A3. **Unpublish:** Manager may unpublish a screening; it no longer appears in customer listings but remains editable.

**3.3.3.9 Postconditions**  
- **Success:** Screening exists with validated schedule; visible to customers if **Published**.  
- **Failure:** No changes saved.

## 3.4 Classes / Objects

### 3.4.1 Screening

3.4.1.1 Attributes:

* Publication State: Draft → Published → Unpublished/Cancelled

3.4.1.2 Functions:

* Update Publication

<UC 3.3.1 and FR 3.2.1,2,10>

### 3.4.2 Auditorium

3.4.2.1 Attributes:

* Seat Map
* Capacity
* Accessibility Constraints

3.4.2.2 Functions:

* Create/Edit Auditorium
* Update Seat

<UC 3.3.1 and FR 3.2.1,2>

**3.4.3 Seat**

3.4.3.1 Attributes:

* Availability: Available → Held → Sold

3.4.3.2 Functions:

* Change Availability
* Check for Orphan Seat

<UC 3.3.1 and FR 3.2.2,3,4>

**3.4.4 Order**

3.4.4.1 Attributes:

* Status: Pending → Confirmed → Cancelled/Rescheduled
* Order ID

3.4.4.2 Functions:

* Set Order State
* Link Order

<UC 3.3.1,2 and FR 3.2.8>

**3.4.5 Ticket**

3.4.5.1 Attributes:

* In-use: Issued → Checked-in

3.4.5.2 Functions:

* Issue Ticket
* Check-in Ticket

<UC 3.3.1 and FR 3.2.7,11>

**3.4.6 Payment**

3.4.6.1 Attributes:

* Status: Authorized → Captured/Refunded/Failed

3.4.6.2 Functions:

* Check Payment Information
* Validate with Payment Service

<UC 3.3.1,2 and FR 3.2.6,9>

**3.4.7 User Roles**

3.4.7.1 Attributes:

* Role: Customer, Manager, Payment/Notification Service

3.4.7.2 Functions:

* Set Role Privileges
* Change Role

<UC 3.3.3 and FR 3.2.10>

**3.4.8 Policy Values**

3.4.8.1 Attributes:

* Seat-Hold Duration
* Refund Window
* Orphan Seat Requirements

3.4.8.2 Functions:

* Configure Policy Values

<UC 3.3.3 and FR 3.2.15>

## 3.5 Non-Functional Requirements

The software must meet these 6 non-functional requirements: Performance, Reliability, Availability, Security, Maintainability, and Portability. This will ensure quick, hassle-free, and helpful experiences for both the customer and the management.

### Performance

The website must load quickly and reliably so that users are not frustrated accessing the site.

*3.5.1.1 Quick to load*: 95% of loads complete within 5 seconds.

*3.5.1.2 Quick to transac*t: 95% of transactions complete within 8 seconds.

*3.5.1.3 Quick to update*: 99% of user interactions with the webpage should register within 3 seconds.

*3.5.1.4 Bandwidth*: server can handle 1000 simultaneous requests at once.

### 3.5.2 Reliability

The system must handle any errors that will cripple the service.

*3.5.2.1 Error Handling*: The system will be able to identify, and in certain cases correct, errors.

### 3.5.3 Availability

The service will available as often as possible, allowing for occasional maintenance periods.

*3.5.3.1 Consistent Up-Time*: Server and website is available for public use 99.9% of uptime.

### 3.5.4 Security

The service will protect customers’ data and prevent unauthorized users from accessing and modifying sensitive data.

*3.5.4.1 Authorization*: only managers or higher-level clearances shall be able to access secure information from databases.

*3.5.4.2 Encryption*: customer data should be encrypted with current RSA encryption.

### 3.5.5 Maintainability

The software will be designed to last and work without frequent and drastic revision upon release. There will be small patches to cover unforeseen problems with the system on a continual basis.

*3.5.5.1 Lifespan*: The software should be able to function over the next 3 years with minor revisions.

*3.5.5.2 Frequency of Revisions*: Minor updates will occur every 3 months, while major changes will occur every 3 years.

### Portability

The website should run without error or problem in the most frequently used browser engines.

* + - 1. *Chrome*: Designed for the Chrome browser

*3.5.6.2 Common Compatibility*: Runs without errors in the 3 most common browsers: Edge, Safari, and Firefox

## 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. Design Development Plan

## 6.1 Roles and Responsibilities

**6.1.1 Team members:**

* Matthew Gallenberger– Person A (Architecture & Diagrams)
* Ryan Schindler – Person B (Class & Architecture Descriptions)
* Nikola Milosavljevic – Person C (Plan, Timeline, Repo Setup)

**6.1.2 Partitioning of tasks:**

* *Person A – Architecture & UML Diagrams*
  + Create Software Architecture Diagram (components & connectors: Web UI, App Server, DB, Payment Provider, Notification Service).
  + Create UML Class Diagram (Screening, Auditorium, Seat, Order, Ticket, Payment, User Roles, Policy Values).
* *Person B – Class & Architecture Descriptions*
  + Write detailed class descriptions (attributes with datatypes; operations with

signatures/parameters).

* + Export diagrams as PNG/SVG and deliver assets for integration.
  + Write Architecture explanation (component responsibilities and interactions).
  + Own editorial pass & final merge into master SRS (DOCX/PDF).
* *Person C – Development Plan & Timeline (this section) + Repo Setup*
  + Initialize GitHub repository and upload prior SRS document.
  + Write this Development Plan & Timeline section and keep it updated based on team inputs.
  + Coordinate submission checklist and ensure each member has ≥1 commit.

## 6.2 Architecture

A diagram of a company

AI-generated content may be incorrect.

Architecture Diagram Description:

This diagram illustrates the end-to-end flow of the Online Ticketing Movie System. It shows how the different components in the client, server, database, and other external systems work together.

At the Client Layer, users interact with the system through a web-based or mobile interface where the user can browse different movies, look at seating charts, place reservations and receive their tickets through their email.

The Server will handle and manage the bulk the modules needed to have the system run properly.

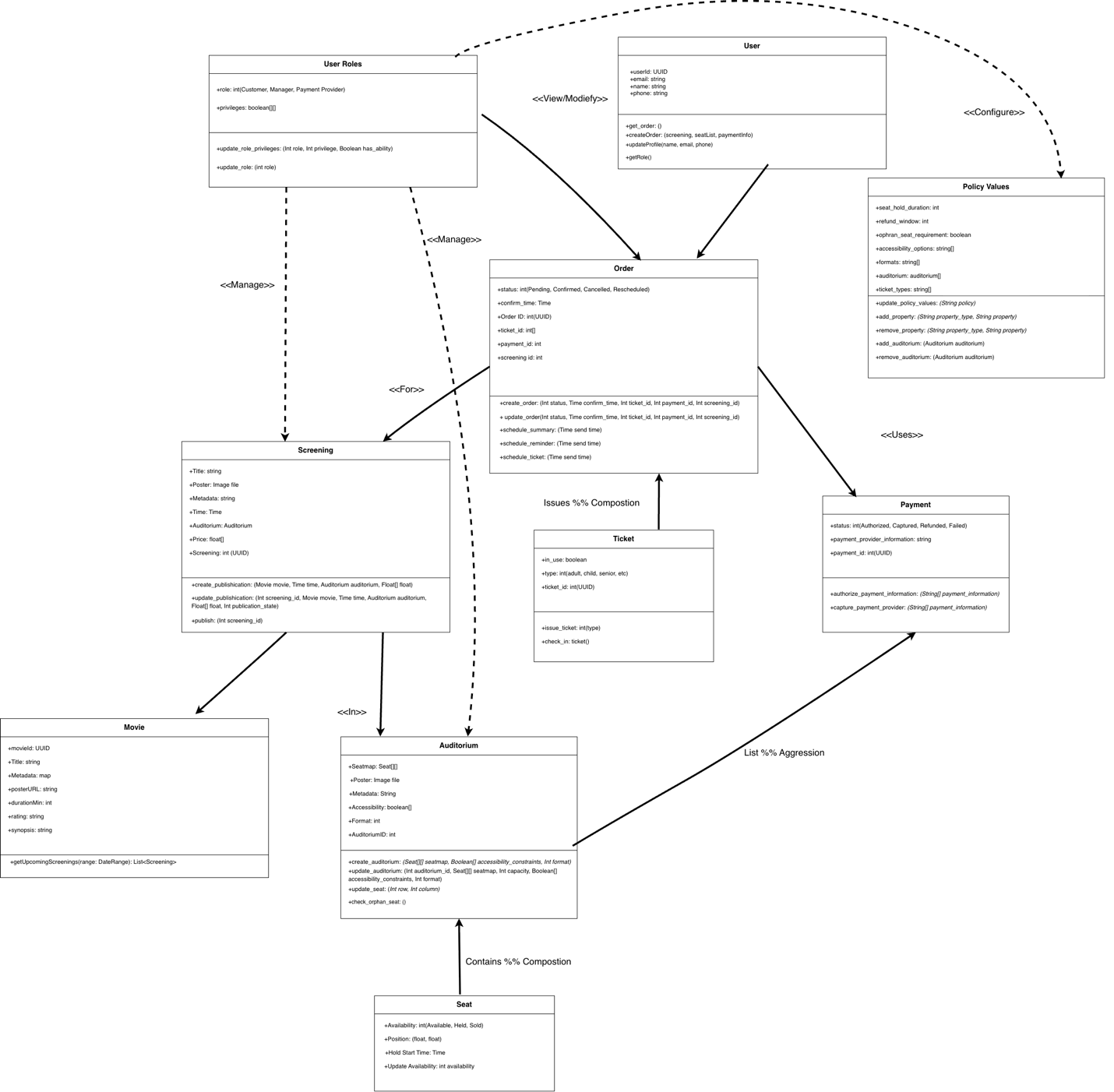
* Order and Checkout
* Manages order creation and hold seats through checkout process.
* Seat Inventory/Hold Manager
* Handles seat availability and movie listings.
* Payment Orchestrator
* Securely handles and facilitates payments with the external payment provider.
* Auth & RBAC, Policy & Audit Logger
* Handles authentication, policy enforcement, and system auditing

The Relational Database will store all the data with respect to users, roles, screenings, auditoriums, orders, tickets, payments, and policy values.

External services include the payment provider module for financial transactions and a notification service for communication with users.

## 6.3 Classes

These are the major classes of the service. In the list, boxes specify attributes of the class while arrows specify methods available to the class.



UML Diagram description:

The UML Diagram shows the main classes and relationships between them that allows the Online Movie Ticketing System function efficiently. The classes work together to manage users, screenings, order, and payments.

The User Roles class defines the different types of users in the system, which are customers, managers and payment providers. This class manages their access level and different privileges.

The Screening class represents individual movie showings, contains attributes like title, release date and auditorium assignment, and schedule times. Each screening is associated with an Auditorium, which defines seat layout, capacity, accessibility, and available seats through the Seat class.

The Order class handles ticket purchases that users place and their reservations. It connects customers to screenings and payments. When an order is placed, the Ticket class will represent individual seats and e-tickets for a certain showing.

They Payment class will process transaction details, including authorization and payment provider integration, ensuring that financial operations are securely processed. The Policy Values class defines configurable policies which include refund policy and other booking constraints.

* **6.3.1 Screening**

The Screening object holds all the information about a particular screening for the movie theater.

* + *Publication State* – Int (Draft, Published, Cancelled, Rescheduled)

Specifies the publication state of the screening. Can be one of four values.

* + *Movie*
    - Title – String
    - Poster – Image file
    - Metadata – String

Structure holding all the information related to the actual movie that is

shown in the screening.

* + *Time* – Time object

The time that the screening is scheduled for.

* + *Auditorium* – Auditorium

The auditorium that this screening is showing in. The auditorium is copied from a list of the auditorium objects that the theater has saved.

* + *Price* – Float[]

The price for the screening. The *Price* object is in array form to specify

different prices for each ticket type.

* + *Screening ID* – Int (UUID)

An identification number to reference a particular screening.

* *Create Publication(Movie movie, Time time, Auditorium auditorium, Float[] float)*

Creates a drafted screening object with the required parameters. Sets the

publication state as draft.

* *Publish(Int screening\_id)*

Publishes the screening so that customers can see it on the website. Sets

the publication state as published.

* *Update Publication(Int screening\_id, Movie movie, Time time, Auditorium auditorium, Float[] float, Int publication\_state)*

Updates one or multiple properties of the screening object. Will

automatically update the publication state. All parameters are optional

except the *Screening Id*.

* **6.3.2 Auditorium**

Each Auditorium object holds the seating and accessibility information for an auditorium theater.

* + *Seatmap* – Seat[][]

A double array of seats to store the seat-specific information related to this auditorium. Seats are ordered first by rows from closest to furthest from the screen, then by column from left to right.

* + *Capacity* – Int

The total number of seats that this auditorium has.

* + *Accessibility Constraints* – Boolean[]

Information on any accessibility constraints this auditorium has, in a Boolean array. The constraints are determined by the *Policy Values*.

* + *Format* – Int

The format of the screen. Each integer value is interpreted by the *Policy Values*.

* + *Auditorium ID* **-** Int

The identification number of the auditorium used to distinguish which

auditorium this virtual object represents.

* *Create Auditorium(Seat[][] seatmap, Boolean[] accessibility\_constraints, Int format)*

Creates a new auditorium object.

* *Update Auditorium(Int auditorium\_id, Seat[][] seatmap, Int capacity, Boolean[] accessibility\_constraints, Int format)*

Updates the properties of the current auditorium.

* *Update Seat(Int row, Int column)*

Updates the properties of the given seat in this auditorium.

* *Check Orphan Seat()*

Checks the selected seat to see if it will create orphan seats.

* **6.3.3 Seat**

The Seat object holds all the information relevant to the selling and positioning of the seats.

* + *Availability* – Int (Available, Held, Sold)

The current status of availability for this seat.

* + *Position* – (Float, Float)

The position that seat shows up in the selection screen for the customer and editor for the manager.

* + *Hold Start Time* – Time

The time when the seat is first put on hold. If the seat has been held longer than the *Seat Hold Duration*, the seat is forfeited and the customer who put it on hold must reapply for the seat.

* *Update Availability(Int availability)*

Changes this seats availability.

* **6.3.4 Order**

The Order object holds all information related to an order and transaction.

* + *Status* – Int (Pending, Confirmed, Cancelled, Rescheduled)

The status of the order. Status tracks of which orders are pending, confirmed, cancelled, or rescheduled.

* + *Confirm Time* **–** Time

The time when this order was completed. The time is used to calculate if the refund window is still valid for a customer.

* + *Order ID* – Int (UUID)

The identification number for this order. This number is used to keep track of the orders in the database for future reference.

* + *Ticket ID* – Int[]

The identification numbers for the tickets associated with this order.

* + *Payment ID* – Int

The identification number for the payment associated with this order.

* + *Screening ID* – Int

The identification number for the screening associated with this order.

* *Create Order(Int status, Time confirm\_time, Int ticket\_id, Int payment\_id, Int screening\_id)*

Creates a new order based on the customer’s selected seats.

* *Update Order(Int status, Time confirm\_time, Int ticket\_id, Int payment\_id, Int screening\_id)*

Updates the order information and sets the order state appropriately if not specified.

* *Schedule Summary(Time send\_time)*

Schedules an email through the email provider service with the summary of the customer’s order. The time that it is scheduled to send is specified with *send\_time*.

* *Schedule Reminder(Time send\_time)*

Schedules an email through the email provider service to remind the customer of their screening. The time that it is scheduled to send is specified with *send\_time*.

* *Schedule Ticket(Time send\_time)*

Schedules an email through the email provider service with the ticket information for the customer to use at the theater. The time that it is scheduled to send is specified with *send\_time*.

* **6.3.5 Ticket**

The Ticket object stores information related to the ticket of a customer. All tickets are stored in a database to be referenced when used at the theater.

* + *In Use* – Boolean

The usage status of the ticket. If the ticket has already been used to enter the theater, then *In Use* will be true.

* + *Type* – Int (Adult, Child, Senior, etc.)

The type of the ticket, as specified by the *Policy Values* of this particular theater.

* + *Ticket ID* – Int (UUID)

The identification number for this ticket. This number is used to keep track of the ticket in the database and for order re-references.

* *Issue Ticket(Int type)*

Creates a new ticket object with the required specifications. The method generates a new UUID for the ticket upon creation.

* *Check-in Ticket()*

Changes the *In Use* property of the ticket to true, thereby making it impossible to be used again.

* **6.3.6 Payment**

The Payment object stores all the information relevant to the customer’s payment. Specific payment information is left to the *Payment Provider* to store and keep track. Payments work in two stages: Authorized and Captured/Refunded/Failed. Payments must first be authorized to be valid payment information through the *Authorize Payment Information* method, and then they are sent to the *Payment Provider* through the *Capture with Payment Provider* method.

* + *Status* – Int (Authorized, Captured, Refunded, Failed)

The status of the payment is stored as an integer encoding these four states.

* + *Payment Provider Information* – String

A reference to the payment provider. Any details necessary to link the transaction to the payment are stored here.

* + *Payment ID* – Int (UUID)

An identification number for this payment. This number is used to keep track of payments in the database.

* *Authorize Payment Information(String[] payment\_information)*

Authorizes payment information and creates a payment object if the payment information is valid.

* *Capture with Payment Provider(String[] payment\_information)*

Uses the payment provider to complete the payment process. The *Status* is updated accordingly.

* **6.3.7 User Roles**

The role of the current user. This determines the user’s access and abilities on the website. The privileges of each role can be changed with the *Update Role Privileges* method. The role is changed with the *Update Role* method.

* + *Role* – Int (Customer, Manager, Payment Provider)

The current role of the user encoded as an integer.

* + *Privileges* **–** Boolean[][]

The set of user privileges according to each user. The double array is sorted by user, then by privilege. Privileges are specified by the theater and are hard-coded in.

* *Update Role Privileges(Int role, Int privilege, Boolean has\_ability)*

Changes the set of privileges to each role. This change is global for all users.

* *Update Role(Int role)*

Changes the role of the current user.

* **6.3.8 Policy Values**

A reference to all theater-specific information and policies. Each policy can be changed according to the theater’s needs.

* + *Seat Hold Duration* – Int

The length of time in seconds for how long seats are held in the system for a customer pending payment. If the seat has been held longer than this amount of time, the seat is forfeited and available to all customers again.

* + *Refund Window* – Int

The length of time in seconds for how long before a completed order can no longer be refunded. If the Screening *Time* has already passed – meaning that the movie has started showing – then the order cannot be refunded.

* + *Orphan Seat Requirement* – Boolean

This value determines whether the system will check for orphan seats. If set to true, then parties cannot purchase a section of seats with a gap of one or two seats between them.

* + *Accessibility Options* – String[]

All of the accessibility options available to an *Auditorium* at this theater. The options can be added or removed through *Add/Remove Property* methods.

* + *Formats* – String[]

The screen formats available to an *Auditorium* at this theater. The formats can be added or removed through *Add/Remove Property* methods.

* + *Auditoriums – Auditorium[]*

The list of available *Auditoriums* for this theater. This list can quickly be referenced to create a new screening. This list can be changed by the *Add/Remove Auditorium* methods.

* + *Ticket Types – String[]*

The ticket types available for purchase at this theater. The ticket types can be added or removed through *Add/Remove Property* methods.

* *Update Policy Values(String policy,*

Updates the *Seat Hold Duration, Refund Window,* and *Orphan Seat Requirement* properties – all the properties that are single values. Policy is specified by *policy* string.

* *Add Property(String property\_type, String property)*

Adds a property to the property lists. The property is specified as *property\_type.*

* *Remove Property(String property\_type, String property)*

Removes a property from the property lists. The property is specified as *property\_type.*

* *Add Auditorium(Auditorium auditorium)*

Adds an *Auditorium* object to the list *Auditoriums* used for quick reference configurations.

* *Remove Auditorium(Auditorium auditorium)*

Removes an *Auditorium* object from the list *Auditoriums* used for quick reference configurations.

* **6.3.9 User**

The User class represents an individual registered within the Online Movie Ticketing System. Each user has a unique indemnity and is associated with one or more orders. Users can view and update their profile information, create new orders, and perform actions according to their assigned roles.

* + *Role*– Int

The current role assigned to the user, determining system permissions and accessible features.

* + *User ID* – UUID

A unique identifier automatically generated for each user.

* + *Name* – String

The user’s full name as displayed on their account.

* + *Email* – String

The user’s email address, used for login and order confirmations.

* + *Phone* – String

The contact phone number for the user.

* *Create Order(createOrder(Screening screening, Seat[] seatList, Payment paymentInfo)*

Creates a new order for the specified screening, seats, and payment details.

* *Get Orders()*

Retrieves all orders that belong to the user*.*

* *Update Profile(String name, String email, String phone)*

Updates the user’s stored contact and account information.

* *Get Role()*

Returns the user’s current role.

* **6.3.10 Movie**

The Movie class represents a film available for viewing and scheduling within the Online Movie Ticketing System. Each movie includes its identifying details and is associated with one or more screenings. The class stores basic metadata such as title, duration, poster, and rating, and provides helper methods to query upcoming screenings.

* + *Movie ID* – UUID

A unique identifier automatically assigned to each movie record.

* + *Title* **–** String

The official title of the movie.

* + *Poster URL* – String

A link or file reference to the movie’s poster image.

* + *Duration Min* **–** Int

The total runtime of the movie in minutes.

* + *Rating* – String

The film’s rating.

* + *Metadata* **–** String / Map

Additional descriptive data such as director, genre, and release year.

* *Get Upcoming Screenings (DataRange range)*

Returns a list of scheduled screenings for this movie within a given date range.

* *Update Poster(String newURL)*

Updates the poster reference for the movie.

* *Add Metadata(String key, String Value)*

Adds or updates a metadata entry for the movie.

## 6.4 Timeline

Target schedule (America/Los\_Angeles). Weeks are guidelines; members may work asynchronously as long as handoff dates are met.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dates | Owner | Focus | Milestone | Handoff Output |
| Sept. 29-30, 2025 | Nikola (C) | Timeline, Roles, and Workflow | Plan ready and Repository available for use | Template for SDD section  Github Repo. |
| Oct. 1-3, 2025 | Matthew (A) | Architecture Diagram + UML Class Diagram | Draft diagrams ready | PNG/SVG exports + source files |
| Oct. 4-7, 2025 | Ryan (B) | Class & Architecture Descriptions | Complete draft text | DOC/MD with class tables & arch description |
| Oct. 8-9, 2025 | Ryan (B), All | Editorial & Merge; light review by all | Final SRS (DOCX/PDF) assembled | Merged DOCX + exported PDF in repo |

## 6.5 Workflow and Submission

1. Repository: GitHub repo with /src (assets), /design (diagrams), /docs (SRS working files), /deliverables (final DOCX/PDF). <https://github.com/KypleeFearz/CS250_Group7>
2. Branching: simple trunk-based; optional feature branches per person (e.g., feature/umldiagrams, feature/class-descriptions).
3. Commits: each member must have ≥1 commit (add diagrams, text, or minor fixes).
4. Integration: Person B merges diagrams + text into the SRS template and exports PDF.
5. Submission: push final DOCX/PDF to /deliverables and submit the GitHub link in the course system.

## 6.6 Risks and Mitigation

* Late handoff of diagrams → Mitigation: placeholders for class names so B can proceed; finalize images before Week 3.
* Format inconsistencies → Mitigation: use the same SRS template; Person B owns one editorial pass in Week 3.
* Missed commit credit → Mitigation: add a small personal change (typo fix or minor edit) if needed before submission.

# 7. Test Plan

## 7.1 Purpose

This section defines the verification and validation plan for the Movie Theater Ticketing System. The goal of the test plan is to ensure that all functional and non-functional requirements specified in this document are correctly implemented, reliable, and fulfill user expectations. The testing process will include unit, integration, and system testing, covering both functional workflows (ticket purchase, cancellation, scheduling) and non-functional qualities (performance, reliability, and security).

## 7.2 Verification Approach

The verification strategy focuses on three granularities of testing, as discussed in class:

1. Unit Testing:  
 - Scope: Individual classes and methods such as Seat, Order, and Payment.  
 - Purpose: Verify that internal logic functions correctly and handles invalid inputs or edge cases.  
 - Example focus: ensuring that Seat.updateAvailability() changes seat status correctly or that Payment.authorizePaymentInformation() rejects invalid payment data.

2. Integration Testing:  
 - Scope: Interactions between subsystems (e.g., Order & Checkout, Payment Orchestrator, Notification Service).  
 - Purpose: Ensure smooth communication between components, such as confirming that a successful payment triggers ticket creation and notification sending.  
 - Example focus: confirming that order status updates correctly propagate to the database and notification system.

3. System Testing:  
 - Scope: The complete application as deployed in the test environment.  
 - Purpose: Validate real-world user workflows across all modules, including UI, server, database, and external integrations.  
 - Example focus: end-to-end purchase of a ticket, cancellation within the refund policy, or manager scheduling a new screening.

## 7.3 Test Coverage

The following major system features will be covered through the test plan:

|  |  |  |
| --- | --- | --- |
| System Component | Feature / Behavior Tested | Test Type |
| Seat Management | Seat selection, hold, release logic | Unit |
| Order Management | Order creation, status transitions | Unit |
| Payment Processing | Payment authorization and refund handling | Unit / Integration |
| Notification System | Sending of email confirmations | Integration |
| User Roles & Policies | Access restrictions, policy enforcement | Integration |
| End-to-End Flow | Full purchase, cancel, reschedule | System |
| Security / Access | Unauthorized actions blocked | System |
| Performance | Seat hold timer, response time | System |

## 7.4 Responsibilities

|  |  |  |
| --- | --- | --- |
| Role | Member | Responsibility |
| Design Verification / Update | Matthew Gallenberger | Verify that design and diagrams reflect all modules tested; update if new classes are added. |
| Test Cases & Coverage | Ryan Schindler | Define and document 10 Excel test cases across all levels (unit, integration, system). |
| Test Plan & Repository Management | Nikola Milosavljevic | Write this Test Plan section, manage GitHub uploads, maintain timeline, ensure all members commit. |

Each team member must contribute at least one commit to the GitHub repository before submission to receive full credit.

## 7.5 Testing Tools and Environment

- Environment: Local web server with mock payment provider (sandbox mode).  
- Database: Azure SQL (test instance).  
- Testing Frameworks:  
 - Unit Testing: JUnit / PyTest  
 - Integration Testing: Postman, Mock APIs  
 - System Testing: Manual workflow validation via browser  
- Automation / CI: GitHub Actions for automatic testing on push events.

## 7.6 Test Deliverables

1. Test Plan Document (this section)  
2. Excel Test Case Sheet – testCasesSample.xlsx, containing:  
 - 2 Unit test cases  
 - 2 Integration test cases  
 - 2 System test cases  
 - 4 additional exploratory tests  
3. GitHub Repository – Updated with /docs/test-plan folder containing:  
 - TestPlan\_Assignment3.pdf  
 - testCases.xlsx  
 - Updated SRS\_v2.3.docx with this new section and GitHub link.

## 7.7 Testing Timeline

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Activity | Responsible | Deliverable |
| Oct 13–14 | Prepare and document test plan | Nikola | Test Plan draft |
| Oct 15–16 | Write and review Excel test cases | Ryan | Excel test sheet |
| Oct 17 | Verify architecture & diagram consistency | Matthew | Diagram verification |
| Oct 18 | Group review and merge into SRS | All | SRS\_v2.3 final version |
| Oct 19 | Push files to GitHub and validate commits | Nikola | Final GitHub folder link |

## 7.8 GitHub Repository Link

All testing materials will be available in the following repository under /testplan/:  
  
https://github.com/KypleeFearz/CS250\_Group7  
  
Each team member must make at least one commit prior to submission. The link above should be included in Canvas as part of the official submission.

## 7.9 Approval

This section and its accompanying files form part of the SRS version 2.3 submission.  
Approved by:  
- Matthew Gallenberger  
- Ryan Schindler  
- Nikola Milosavljevic  
  
Date: October 21, 2025

# 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