# Movie Theater Ticketing Project

Software Requirements Specification
Version 2.0
October 9, 2025

Group 9
Kristine Alba, Amy Bernal Chavarria, Danna Bundogji

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2023

## Revision History

| Date | Description | Author | Comments |
|---|---|---|---|
| 9/19/2025 | Initial project setup, document created | Amy Bernal C | First revision |
| 9/20/2025 | Added Purpose and Scope | Kristine Alba | First revision |
| 9/21/2025 | Expanded Definitions and References | Kristine Alba | Added technical details |
| 9/22/2025 | Completed General Description section | Danna Bundogji | Wrote system overview |
| 9/23/2025 | Refined Scope, added Assumptions | Amy Bernal C | Improved clarity |
| 9/24/2025 | Added Functional Requirements draft | Kristine Alba | Outlined key features |
| 9/25/2025 | Reviewed and edited grammar, formatting | All members | Polished version for submission |

## Document Approval

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

| Signature | Printed Name | Title | Date |
|---|---|---|---|
|  | Dr. Gus Hanna | Instructor, CS 250 |  |
|  | Kristine Alba | Software Eng. |  |
|  | Danna Bundogji | Software Eng. |  |
|  | Amy Bernal C | Software Eng. |  |

## Table of Contents

# 1. Introduction

## 1.1Purpose

The purpose of this Software Requirements Specification is to define the functional and non-functional requirements of the Movie Ticketing System. This document is primarily intended for the software development team, quality assurance engineers and project managers to design, implement and test the system. Secondary audiences will include managers and administrators who need to understand the system's capabilities for operational purposes.

## 1.2 Scope

The system product is the Movie Ticketing System (MTS), a browser-based ticketing platform supporting both online customers (website) and in-person customers via digital kiosks and box-office terminals. It will manage movie listings, showtimes, seat reservations, ticket sales, discounts, membership handling, feedback collection, and administrative operations for a San Diego theater chain (20 theaters). The initial release focuses on core ticketing features; some features (nearby-theater search, extensive third-party integrations) are deferred to future iterations.

The MTS will improve speed, scalability, and of ticket sales. Goals include providing a secure and user friendly browser-based platform accessible in English and Spanish, support up to 650 concurrent users and prevent scalping though unique NFT-style tickets, deliver consistent pricing, localized currency display, and enforce transaction rules (20-ticket max, 2-week advance window), and reduce downtime and errors found in the legacy system.

## 1.3 Definitions, Acronyms, and Abbreviations

| | |
|---|---|
| SRS | Software Requirements Specification |
| UI | User Interface |
| DBMS | Database Management System |
| Admin | Authorized staff user with elevated privileges |
| Regular Seat / Regular Theater | General admission theater (150 seats) |
| Deluxe Seat / Deluxe Theater | Reserved seating theater (75 seats) |
| NFT Ticket | Non-fungible token representation ensuring ticket uniqueness |

## 1.4 References

- IEEE Std 830-1998, “IEEE Recommended Practice for Software Requirements Specifications,” IEEE, 1998.
- PCI Security Standards Council, “PCI Data Security Standard (PCI DSS) Documentation,” (payment security).
- Perforce Software, “How to Write a Software Requirements Specification (SRS) Document,” https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document
- Relevant Software, “Software Requirements Specification: What It Is, How to Write It,” https://relevant.software/blog/software-requirements-specification-srs-document/
- draw.io (diagrams.net), UML Class Diagrams for MTMS.

## 1.5 Overview

The remaining sections of this document provide a detailed description of the Movie Theater Ticketing System requirements. Section 2 gives a general description of the product including its functions, user characteristics, constraints, and assumptions. Section 3 specifies the detailed functional and non-functional requirements, including interface requirements, business rules, and use cases. Section 4 presents analysis models such as sequence diagrams, data flow diagrams, and state transition diagrams. Section 5 describes the change management process for handling modifications to this SRS.

# 2. General Description

## 2.1 Product Perspective

The Movie Ticketing System (MTS) is designed as a replacement for the legacy ticketing software used by the San Diego theater chain. It will be a browser-based platform that supports access through customer websites, mobile browsers, kiosks, and box office terminals. The system will operate independently but connect with external services such as payment gateways, email and SMS notifications, and blockchain infrastructure for NFT-based ticket verification. Its role is to unify online and in-person ticket sales, provide real-time seat management, and deliver a secure, reliable experience for both customers and theater staff.

## 2.2 Product Functions

The system will allow administrators to manage movie listings, schedules, and pricing across theaters. Customers will be able to browse movies, select seats, reserve tickets, and complete secure payments online or in person. The system will issue digital tickets, either as QR codes or NFT-based tokens, and support validation at the theater. It will also enforce transaction rules, such as ticket limits and advance booking windows. Additional functions include membership management, application of discounts, collection of customer feedback, and generation of sales and occupancy reports for managers.

## 2.3 User Characteristics

The system will serve three main groups of users. Customers will include the general public, with varying levels of technical knowledge, and will require simple, multilingual interfaces in English and Spanish. Theater staff, such as cashiers and ushers, will use the system for ticket sales, refunds, and validation, and will need fast and reliable performance during busy times. Administrators and managers will be technically skilled users responsible for theater setup, monitoring, and reporting. Accessibility features will be included to support users with vision or hearing impairments.

## 2.4 General Constraints

The system must support high performance, handling millions of users during peak demand. All communications will be secured with HTTPS, and the system must comply with PCI-DSS standards for payment data. It should scale easily to support multiple theaters at once and run on modern web browsers, kiosks, and scanning hardware. Interfaces must be available in English and Spanish. Legal and regulatory compliance, such as GDPR and U.S. consumer protection standards, must also be ensured.

## 2.5 Assumptions and Dependencies

The system assumes the theater chain will provide reliable internet connections, kiosk hardware, and scanning devices. A third-party payment processor is expected to always be available and functional. Online access is required for customers, as offline ticketing is not included in the initial release. The NFT ticket feature depends on blockchain services and stable transaction costs. Currency conversion will depend on external APIs, and future integrations with digital wallets or third-party apps will require cooperation from those platforms.

# 3. Specific Requirements

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

Costumer UI

Input: Costumers select movies, showtimes and seats.
Output: System displays available seats, prices and booking confirmation.
Requires for updates to appear within 3 seconds of user action.

Management Dashboard

Input: Managers enter or edit movie schedules, seat availability and pricing.
Output: Dashboard confirms updates and syncs with costumer's UI.
Requires for the changes to be reflected on all interfaces within 10 seconds.

### 3.1.2 Hardware Interfaces

Scanners:

Input: Scans tickets barcodes or QR codes.
Output: System validates tickets.
Requires for the validation to occur within 1 second.

Self Service Kiosks

Input: Costumers are able to select movies, showtimes, seats and payment method.
Output: Printed ticket or digital QR code sent to email.
Requires completing transaction under 3 minutes.

Ticket Printers

Input: Costumer touch selections for movie, showtime, seats and payment methods.
Output: Physical paper ticket printed for costumer.
Requires the transaction to be completed within 5 seconds.

### 3.1.3 Software Interfaces

Input: Store movie schedules, ticket reservations, users accounts and payment records.
Output: Real time data.
Requires supporting concurrent access from at least 650 users without delay.

### 3.1.4 Communications Interfaces

Email and SMS Notifications.

Input: Confirmation message including ticket, movie info and purchase details.
Output: Delivery to costumer email or phone number.
Requires messages to be sent within 3 minutes of purchase.

## 3.2 Functional Requirements

The system must provide customers with the ability to browse movies, select seats, purchase tickets and receive confirmations while also giving administrators tools to manage schedules, costumer support and sales.

### 3.2.1 Concurrency & Scalability

*3.2.1.1 Introduction*
The system must support high volumes of concurrent users.

*3.2.1.2 Inputs*
Multiple simultaneous requests from online and kiosk users.

*3.2.1.3 Processing*
Load balancing, horizontal scaling, and caching.

*3.2.1.4 Outputs*

Stable service availability under heavy load.

*3.2.1.5 Error Handling*
If limits are exceeded, queuing system activates.

### 3.2.2 Platform Accessibility

*3.2.2.1 Introduction*
The system must be browser-based and synchronized.

*3.2.2.2 Inputs*
User login or guest access via browser or kiosk

*3.2.2.3 Processing*
Retrieve current showtimes and seating from central DBMS.

*3.2.2.4 Outputs*
Unified, up-to-date UI across web and kiosks.

*3.2.2.5 Error Handling*
Display connection error if database unavailable.

### 3.2.3 Anti-Bot/Anti-Scalping

*3.2.3.1 Introduction*
Prevent automated ticket scalping.

*3.2.3.2 Inputs*
Ticket purchase attempts.

*3.2.3.3 Processing*
CAPTCHA, device fingerprinting, anomaly detection.

*3.2.3.4 Outputs*
Valid transactions only from verified users.

*3.2.3.5 Error Handling*
Block suspicious transactions, alert admin.

### 3.2.4 Ticketing Rules

*3.2.4.1 Introduction*
Ticket limits and purchase windows.

*3.2.4.2 Inputs*
User selects number of tickets, and showtime

*3.2.4.3 Processing*

Validate against rules (≤20 tickets, within 14 days, before +10 minutes of showtime).

*3.2.4.4 Outputs*
Approved or rejected ticket request.

*3.2.4.5 Error Handling*
Show clear error message for rule violations.

**3.2.5 Account and Session Management**
*3.2.5.1 Introduction*
Support optional accounts and enforce single session.

*3.2.5.2 Inputs*
User login credentials.

*3.2.5.3 Processing*
Authenticate and verify concurrent sessions.

*3.2.5.4 Outputs*
Account dashboard with loyalty, history, and payment info.

*3.2.5.5 Error Handling*
Notify user if account is already logged in elsewhere.

**3.2.6 Payment and Currency**
*3.2.6.1 Introduction*
Handle ticket payments and pricing.

*3.2.6.2 Inputs*
Credit card, PayPal, or Bitcoin payment info.

*3.2.6.3 Processing*
Validate payment, convert currency if needed.

*3.2.6.4 Outputs*
Transaction confirmation.

*3.2.6.5 Error Handling*
Reject invalid or declined payments.

**3.2.7 Ticket Delivery**
*3.2.7.1 Introduction*
Deliver secure, unique tickets.

*3.2.7.2 Inputs*
Purchase confirmation.

*3.2.7.3 Processing*
Generate NFT/unique token, attach to QR code.

*3.2.7.4 Outputs*
E-ticket emailed or printed physical ticket.

*3.2.7.5 Error Handling*
Retry delivery or notify user of failure.

**3.2.8 Feedback System**
*3.2.8.1 Introduction*
Collect quick customer feedback.

*3.2.8.2 Inputs*
User selection of smiley face or rating

*3.2.8.3 Processing*
Store feedback linked to transaction.

*3.2.8.4 Outputs*
Confirmation message.

*3.2.8.5 Error Handling*
If feedback storage fails, discard without blocking purchase flow.

**3.2.9 Discounts**
*3.2.9.1 Introduction*
Apply eligible discounts.

*3.2.9.2 Inputs*
Discount selection (student, veteran/military, senior, child).

*3.2.9.3 Processing*
Verify eligibility and applying pricing rules.

*3.2.9.4 Outputs*
Adjusted ticket price.

*3.2.9.5 Error Handling*
Show error if discount conditions not met.

**3.2.10 Loyalty Accounts**
*3.2.10.1 Introduction*
Store user history and reward points.

*3.2.10.2 Inputs*
Registration info, transactions.

*3.2.10.3 Processing*
Update loyalty points and purchase history.

*3.2.10.4 Outputs*
Loyalty dashboard.

*3.2.10.5 Error Handling*
Notify user of registration or update errors.

### 3.2.11 Administrative Functions

*3.2.11.1 Introduction*
Admin management of system.

*3.2.11.2 Inputs*
Admin login, showtime/movie data.

*3.2.11.3 Processing*
Add, edit, or override system records.

*3.2.11.4 Outputs*
Updated listings, sales reports, override logs.

*3.2.11.5 Error Handling*
Reject unauthorized actions and log attempts.

### 3.2.12 **Logging & Auditing**

*3.2.12.1 Introduction*
Maintain comprehensive daily logs for purchases, sessions, and admin actions.

*3.2.12.2 Inputs*
All relevant system events.

*3.2.12.3 Processing*
Append logs with timestamps, IDs, and metadata to secure storage.

*3.2.12.4 Outputs*
Searchable audit trails for compliance and debugging.

*3.2.12.5 Error Handling*
Failover to secondary log store and alert ops.

### 3.2.13 Queueing System

*3.2.13.1 Introduction*

Virtual waiting room to faily handle spikes for popular showings.

*3.2.13.2 Inputs*
Surge of requests for a single showing.

*3.2.13.3 Processing*
Admit users into a queue, throttle requests, allocate tickets in order.

*3.2.13.4 Outputs*
Queue position and ETA shown to user.

*3.2.13.5 Error Handling*
Show fallback message and attempt requeue if transient error.

#### 3.2.14 Review Integration

*3.2.14.1 Introduction*
Surface critic scores/quotes from external review sources when permitted.

*3.2.14.2 Inputs*
Movie selection; external API/scraper responses.

*3.2.14.3 Processing*
Aggregate and cache scores; respect ToS and rate limits.

*3.2.14.4 Outputs*
Display critic scores/quotes or link to external pages.

*3.2.14.5 Error Handling*
Display "Reviews Unavailable" fallback and log API failures.

## 3.3 Use Cases

### 3.3.1 Purchase Ticket

Primary actor: Customer (guest or logged-in)
Flow of Events:

1. Customer selects movie and showtime.
2. System displays seat map (if Deluxe) or confirms general admission availability (Regular).
3. Customer selects seats/quantity ($\leq 20$) and proceeds.
4. System holds selected seats for up to 5 minutes.
5. Customer enters payment info (or selected saved payment/loyalty points) and confirms.
6. System processes payment and, on success, mints unique ticket token(s), updates seat inventory, and issues e-ticket(s) (email or print).
7. System displays confirmation and prompts optional feedback.

Assumptions/Entry conditions:

1. The showtime exists and has available seats/tickets.

2. Payment gateway is reachable.
3. User is within the 14-day to 10-minute purchase window.

### 3.3.2 Manage Account/Loyalty

Primary actor: Registered Customer, System
Flow of Events:

1. Customer navigates to account page and logs in.
2. System authenticates user and displays account dashboard.
3. Customer views/updates personal info, payment methods, or loyalty settings; views purchase history and points.
4. System saves changes and updates loyalty balances where applicable.

Assumptions/Entry conditions:

1. Customer has an existing account (or completes registration).
2. Authentication services are available.
3. Single-session policy is enforced.

### 3.3.3 Administrative Override/Manage Showtimes

Primary actor: Administrator (staff)
Flow of Events:

1. Admin logs into admin portal with elevated credentials.
2. Admin views dashboard (schedules, sales, flagged issues).
3. For showtime management: Admin adds/edits/removes movie showtimes; system validates conflicts and publishes updates.
4. For override: Admin locates order, executes permitted override (refund, reissue, release seats), and confirms action.
5. System records an audit log for every admin action

Assumptions/Entry conditions:

1. Admin credentials (and MFA if required) are valid.
2. Admin role authorizes requested actions.
3. Audit/logging service is operational.

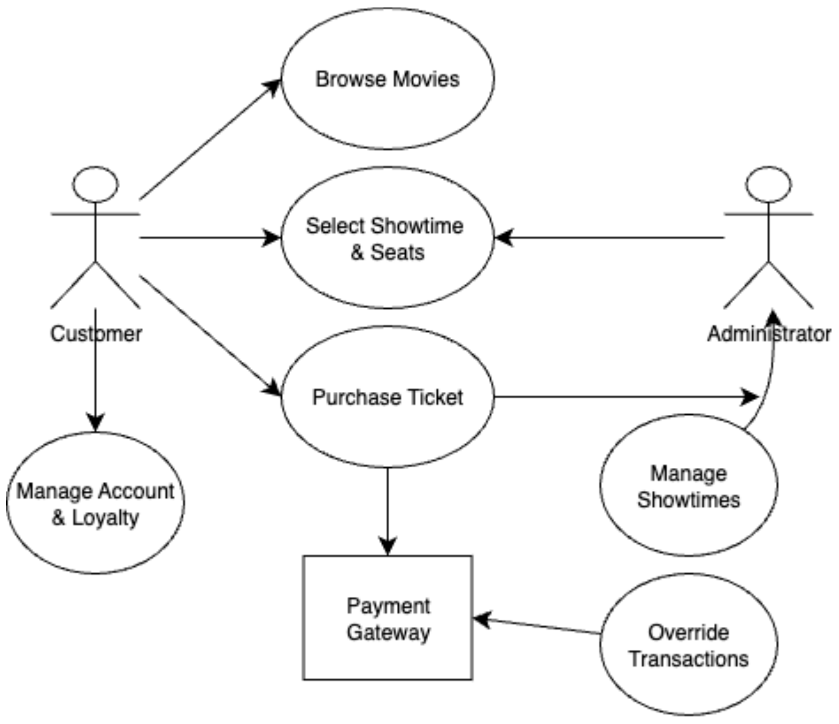

Figure 1: Movie Ticketing System Use Case Diagram

## 3.4 Classes / Objects

### 3.4.1 Movie

3.4.1.1 Attributes

- movieID: integer
- title: string
- genre: string
- duration: float
- rating: string

3.4.1.2 Functions

- getShowtimes(): retrieves all showtimes for a movie
- updateDetails(): modifies movie information
- getAverageRating(): calculates and returns average user rating

### 3.4.2 Showtime

3.4.2.1 Attributes

- showtimeID: integer
- movieID: integer
- theaterID: integer
- startTime: datetime
- availableSeats: integer

3.4.2.2 Functions

- reserveSeat(seatNumber): marks a seat as reserved
- calcelReservation(seatNumber): frees a reserved seat
- getRemainingSeat(): returns remaining available seats

### 3.4.3 Ticket

3.4.3.1 Attributes

- ticketID: string
- showtimeID: integer
- seatNumber: string
- price: decimal
- NFTToken: string

3.4.3.2 Functions

- generateQRCode(): produces a unique QR code for validation
- validateTicket(): verifies ticket authenticity at entry
- resendTicket(email): reissues a lost digital ticket

### 3.4.4 UserAccout

3.4.4.1 Attributes

- userID: integer
- name: string
- email: string
- passwordHash: string
- loyaltyPoints: integer

3.4.4.2 Functions

- login(email, password): authenticates user access
- redeemPoints(points): applies loyalty points to a purchase
- updateProfile(details): modifies user information

### 3.4.5 Payment

3.4.5.1 Attributes

- transactionID: string
- amount: decimal
- paymentMethod: string
- status: string
- timestamp: datetime

3.4.5.2 Functions

- processPayment(method, amount): handles payment through gateway
- issueRefund(ticketID): processes refunds for canceled orders
- getTransactionDetails(): retrieves payment summary

## 3.5 Non-Functional Requirements

The Movie Theater Ticketing System must meet the following non-functional requirements to ensure reliability, performance and usability.

### 3.5.1 Performance

- The system must support up to 650 concurrent users without service degradation
- The system shall respond to 95% of all page requests within 2 seconds under normal load
- The system shall process a completed ticket purchase in ≤ 5 seconds after payment submission

### 3.5.2 Reliability

- The system has a mean time between failures (daMTBF) of at least 500 hours
- The system shall ensure that no more than 0.1% of transactions are lost or corrupted due to system errors

### 3.5.3 Availability

- The system shall provide 99.9% uptime, excluding scheduled maintenance
- The system shall automatically recover from server failure within 60 seconds using failover mechanisms

### 3.5.4 Security

- All communications shall use HTTPS/TLS 1.2 or higher
- Tickets shall be unique and non-replicable (NFT-backed or equivalent)
- The system shall enforce singe-device login per user account
- Payment handling shall comply with PCI-DSS standards

### 3.5.5 Maintainability

- The system shall be modular and documented so that a new developer can resolve defects within 2 hours of investigation
- Updates shall be deployed with < 15 minutes downtime

### 3.5.6 Portability

- The system shall run on the latest two major versions of Chrome, Firefox, Safari, and Edge
- Kiosk terminals shall operate using standards-compliant browsers with no additional plug-ins

## 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

### 4.2 Data Flow Diagrams (DFD)

### 4.3 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

The Movie Theater Management System is designed to automate the management of movie showtimes, ticket booking, payment processing, and customer information. The system will provide both customers and staff an easy-to-use interface for browsing movies, purchasing tickets, and managing schedules.

## 6.2 Software Architecture Overview

### 6.2.1 Software Architecture Diagram

### 6.2.2 Software Architecture Description

## 6.3 UML Class Diagram

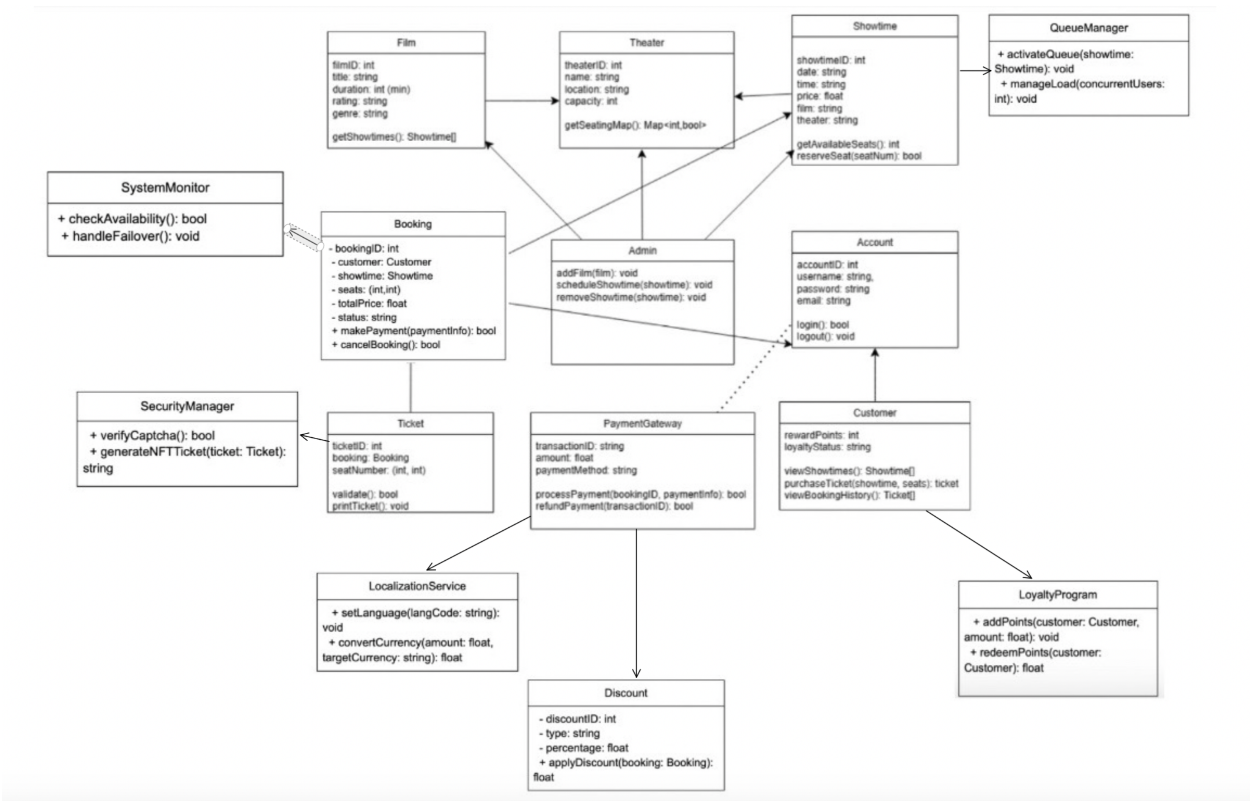

Figure 2: UML Class Diagram for Movie Theater Ticketing System

## 6.4 UML Class Descriptions

*6.4.1 Film*

This class provides info about the film selected.

*6.4.1.1 Attributes*

- filmID: int
- title: string
- duration: float
- rating: string
- genre: string

*6.4.1.2 Operations*

- getShowtimes(): Showtime [] Returns available showtimes for the movie

*6.4.2 Theater*
This class provides info about a specific theater at different locations.

*6.4.2.1 Attributes*
- theaterID: int
- name: string
- location: string
- capacity: int

*6.4.2.2 Operations*
- getSeatingMap(): Map<int,bool> Returns seating availability for the theater

*6.4.3 Showtime*
This class provides info about the film's showtime including the date, time, price, and theater showing it.

*6.4.3.1 Attributes*
- showtime: int
- date: string
- time: string
- price: float
- film: string
- theater: string

*6.4.3.2 Operations*
- getAvailableSeats(): int - Returns number of available seats
- reserveSeat(seatNum): bool - Reserves a seat for a user

*6.4.4 Booking*
This class handles booking a seat with the theater including a specific ID for the booking, the customer that's booking, the showtime of the film, seats, and price.

*6.4.4.1 Attributes*
- bookingID: int
- customer: Customer
- showtime: Showtime

- seats: (int, int)
- totalPrice: float
- status: string
- discountApplied : float
- queuePosition: int

*6.4.4.2 Operations*

- makePayment(paymentInfo): bool - Processes payment for the booking
- cancelBooking(): bool - Cancels booking and updates availability
- applyDiscount(discount: Discount): void - Applies discount to the total price
-

*6.4.5 Ticket*

This class provides info about tickets when booking including the seat number.

*6.4.5.1 Attributes*

- ticketID: int
- booking: Booking
- seatNumber: (int, int)
- nftToken: string

*6.4.5.2 Operations*

- validate(): bool - Confirms ticket
authenticity
- printTicket(): void - Prints physical or digital ticket

*6.4.6 Account*

This class handles the user's info such as username, password, and email to login or logout.

*6.4.6.1 Attributes*

- accountID: int
- username: string
- password: string
- email: string

*6.4.6.2 Operations*

- login(): bool - Authenticates user

- logout(): void - Ends active session

*6.4.7 Customer (inherits Account)*

This class holds info about the user's account such as their loyalty status and if they have reward points.

*6.4.7.1 Attributes*

- rewardPoints: int
- loyaltyStatus: string
- languagePreference: string
- currencyPreference: string

*6.4.7.2 Operations*

- viewShowtimes(): Showtime[]
- purchaseTicket(showtime, seats): ticket
- viewBookingHistory(): Ticket[]
- joinQueue(showtime): void – Adds user to a virtual queue
- redeemLoyaltyPoints(points: int): bool - Applies loyalty rewards to purchase

*6.4.8 Admin (inherits Account)*

This class handles changes that admins can do such as adding films, editing showtime schedules, and removing showtimes.

*6.4.8.1 Operations*

- addFilm(film: Movie): void
- scheduleShowtime(showtime: Showtime): void
- removeShowtime(showtime: Showtime): void
- overrideQueue(queu: QueueManager): void – Adjust queue or priorities
- monitorSystem(status: SystemMonitor): void – Checks system health or failover status

*6.4.9 Payment*

This class handles transactions from users such as generating a transaction ID, the amount that is due and the payment method that the user prefers. It may process a user's payment or refund their payment.

*6.4.9.1 Attributes*

- transactionID: string
- amount: float
- paymentMethod: string
- supportedMethods: string[] (Credit, PayPal, Bitcoin)

- cryptoEnabled: bool

*6.4.9.2 Operations*

- processPayment(bookingID, paymentInfo): bool - Processes user payments
- refundPayment(transactionID): bool - Issues refunds when necessary
- convertCurrency (amount, targetCurrency) : float – Converts currency for localization

**6.4.10 Discounts**
This class defines discount types and rules applied to eligible customers.

*6.4.10.1 Attributes*

- discountID: int
- discountType: string (student, Military, Senior)
- percentage: float

*6.4.10.2 Operations*

- applyDiscount(basePrice: float) : float – Returns adjusted price after discount

**6.4.11 Loyalty Program**
This class tracks customer reward points and manages loyalty tiers.

*6.4.11.1 Attributes*

- pointsBalance: int
- tier: string (silver, gold, platinum)

*6.4.11.2 Operations*

- addPoints(amount: int): void – Adds earned points to balance
- redeemPoints(amount: int): bool – Redeems points for discounts

**6.4.12 QueueManager**
This class manages the virtual queue systems during high-demand events

*6.4.12.1 Attributes*

- queueID: int
- showtime: Showtime
- maxCapacity: int
- currentLoad: int

*6.4.12.2 Operations*

- enqueueUser(customer: Customer): bool – Adds user to the queue
- dequeueUser(): Custumer – Removes next user in line
- monitorLoad(): float – Returns queue load ratio

#### 6.4.13 SecurityManager
This class provides security and anti-bot functionalities including CAPTCHA validation and NFT ticket generation.

*6.4.13.1 Attributes*
- tokenID: string
- captchaEnabled: bool

*6.4.13.2 Operations*
- generateNFTTicket(ticket: Ticket): string – Creates unique NFT for ticket
- validateCaptcha(userInput: string: bool – Validates CAPTCHA for login or purchase)

#### 6.4.14 LocalizationService
This class supports multilingual display and multi-currency operations.

*6.4.14.1 Attributes*
- tokenID: string

*6.4.14.2 Operations*
- generateNFTTicket(ticket: Ticket): string – Creates unique NFT for ticket

#### 6.4.15 SystemMoniter
This class ensures system reliability and automatic failover recovery within defined thresholds.

*6.4.15.1 Attributes*
- nodeStatuus: string
- lastHeartbeat: datetime

*6.4.15.2 Operations*
- checkHealth(): bool – Verifies operational status of services
- triggerFailover(): void – Initiates backup instance on failure

## 6.5 Development Plan and Timeline

### 6.5.1 Task Partitioning

| Task | Description |
|---|---|
| **Database Design** | Create relational schema for movies, users, and bookings |
| **Backend Implementation** | Build server logic for booking, payment, and admin operations |
| **Frontend Development** | Create responsive UIs for web and kiosk users |
| **Integration & APIs** | Connect backend to external payment and email systems |
| **Testing & QA** | Conduct functionality, integration, and load testing |
| **Documentation** | Maintain version control, documentation, and commit logs |

### 6.5.2 Team Member Responsibilities

| Team Member | Responsibilities |
|---|---|
| Amy Bernal | Backend development, database setup, payment integration |
| Danna Bundogji | Frontend development, documentation, and testing |
| Kristine Alba | Architecture design, UML modeling, integration, and deployment |

### 6.5.3 Project Timeline

| Week | Task | Responsible Member |
|---|---|---|
| Week 1 | Database schema and setup | Amy Bernal |
| Week 2 | UI mockups and wireframes | Danna Bundogji |
| Week 3 | Implement booking module | Kristine Alba |
| Week 4 | Integrate payment API | Amy Bernal |
| Week 5 | System testing and fixes | Danna and Kristine |

# 7. Test Plan

***Github link: https://github.com/kalba5618/Group-9-Design-SRS/tree/main/Assignment3_TestPlan***

## 7.1 Purpose

The purpose of this test plan is to define the strategy, scope, resources, and schedule for verification and validation of the Movie Theater Ticketing System (MTTS). It ensures that all functional and non-functional requirements are properly tested, that defects are identified and resolved, and that the final system meets both user and performance expectations.

## 7.2 Test Objectives

The primary testing objectives are:

- Verify that all functional requirements (FR-1 through FR-14) are correctly implemented and traceable to test cases.
- Validate that all non-functional requirements (3.5.1–3.5.6) meet the defined performance, reliability, and security criteria.
- Confirm that the overall architecture and design (as shown in the UML and SWA diagrams) behave correctly under real-world load, concurrency, and user interaction.

## 7.3 Test Scope

The scope of testing includes:

- **In-Scope:** User registration, authentication, ticket browsing and purchase, seat selection, payment processing (credit, PayPal, Bitcoin), discount application, loyalty points, ticket generation (email/print), feedback, queueing for high-demand events, and administrative overrides.
- **Out-of-Scope:** Third-party systems beyond defined APIs (e.g., Rotten Tomatoes or IMDb), hardware malfunctions at kiosks, and theater concession management.

## 7.4 Features to Be Tested

| Feature | Description | Related Requirements |
|---|---|---|
| Ticket Purchase | End-to-end flow for Regular and Deluxe theaters | FR-2, FR-4, FR-6, FR-7 |
| Payment Processing | Credit Card, PayPal, and Bitcoin transactions | FR-6 |
| Discounts | Student, Military, and Senior pricing rules | FR-9 |
| Loyalty Accounts | Account creation, points tracking, and redemption | FR-10 |
| Queue System | Virtual waiting room for high-demand showtimes | FR-13 |
| Administrative Overrides | Refunds and schedule adjustments by admins | FR-11 |
| Feedback Collection | Post-purchase ratings and storage | FR-8 |
| Security / Anti-Bot | CAPTCHA and NFT-based unique tickets | FR-3, NFR-4 |
| Localization | Language and currency support (English, Spanish) | NFR-10 |
| Reliability & Failover | Recovery within 60 seconds of failure | 3.5.2, 3.5.3 |

## 7.5 Test Approach

## 7.6 Test Approach

Testing is performed at three levels of granularity:

1. **Unit Testing** – conducted by developers to verify individual components (e.g., ticket validation rule, NFT token generator).
2. **Integration / Functional Testing** – validates workflows that span multiple modules (e.g., discount application with payment).
3. **System / End-to-End Testing** – validates the entire user experience, including seat selection, payment, email delivery, and logging.

Each test follows these steps:

- Identify the requirement under test.
- Set preconditions (e.g., available seats, user account).
- Execute defined test vectors or data.
- Record results (pass/fail) and screenshots.
- Log defects for review and resolution.

## 7.7 Test Data and Vectors

- **Time-based:** purchase window edge cases (14 days before, 10 min after showtime).
- **Load:** simulate 50 000+ users to test queue activation.
- **Discount:** overlapping eligibilities (student + veteran).
- Software Requirements Specification Page 24Movie Theater Ticketing Project
- **Security:** invalid token and bot attempts.
- **Localization:** browser languages EN/ES with currency conversion.

## 7.8 Test Environment

- **Client:** latest two versions of Chrome, Firefox, Safari, Edge; theater kiosk browsers.
- **Server:** cloud deployment (API Gateway, Ticketing Service, Payment Orchestrator, Central
- DB).
- **External Services:** Payment sandboxes (Credit, PayPal, Bitcoin), mock review APIs, email
- sandbox.
- **Tools:** Postman, Selenium, JMeter, Burp Suite, Excel test matrix.

## 7.9 Entry and Exit Criteria

**Entry Criteria**

- All functional modules developed and build stable.
- Test data and environment configured.

**Exit Criteria**

- 100% of High and Medium priority tests passed.
- No open critical defects.
- Non-functional targets met (≤ 2 s average response, ≥ 99.9% uptime).

### 7.10 Deliverables

This Test Plan section (documented in SRS).

- **Test Case Matrix** (*MTTS_TestCases.xlsx*).
- **Test Execution Log** and Defect Report.
- **Final Test Summary Report.**

### 7.11 Risks and Mitigation

| **Risk** | **Impact** | **Mitigation** |
|---|---|---|
| Payment gateway downtime | High | Use mock endpoints during integration tests. |
| Queue misconfiguration | Medium | Conduct early load tests and adjust parameters. |
| Bot traffic interference | Low | Enable WAF and CAPTCHA during testing. |
| Localization errors | Medium | Conduct manual UI reviews for each language. |

### 7.12 Traceability

All test cases are linked to their respective requirement IDs (FR-x / NFR-x). The traceability matrix in *MTTS_TestCases.xlsx* ensures that every requirement is verified through at least one corresponding test, maintaining alignment between requirements, design, and implementation.

### 7.13 Test Cases

## 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