Software Requirements Specification Template

Software Engineering

The following annotated template shall be used to complete the Software Requirements Specification (SRS) assignment.

Template Usage:
Text contained within angle brackets ('<', '>') shall be replaced by your project-specific information and/or details. For example, <Project Name> will be replaced with either 'Smart Home' or 'Sensor Network'.

Italicized text is included to briefly annotate the purpose of each section within this template. This text should not appear in the final version of your submitted SRS.

This cover page is not a part of the final template and should be removed before your SRS is submitted.

<WAD Theater>

Software Requirements Specification

<Version 1.0>

<9/25/25>

<Group 2>
<Michael Williams,
Isaac Afram,
Gershom Delgado>

Prepared for
CS 250- Introduction to Software Systems

Instructor: Gus Hanna, Ph.D.
Fall 2025

Revision History

Date	Description	Author	Comments
<9/25>	<Version 1.0>	<Isaac Afram, Gershom Delgado, Michael Williams>	<SRS on movie ticketing system>

Document Approval

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

Signature	Printed Name	Title	Date
	<Isaac Afram, Gershom Delgado, Michael Williams>	Software Eng.	9/25/25
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

Revision History ii

Document Approval ii

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 1

2. General Description 2

2.1 Product Perspective 2
2.2 Product Functions 2
2.3 User Characteristics 2
2.4 General Constraints 2
2.5 Assumptions and Dependencies 2

3. Specific Requirements 2

3.1 External Interface Requirements 3
3.1.1 User Interfaces 3
3.1.2 Hardware Interfaces 3
3.1.3 Software Interfaces 3
3.1.4 Communications Interfaces 3
3.2 Functional Requirements 3
3.2.1 <Functional Requirement or Feature #1> 3
3.2.2 <Functional Requirement or Feature #2> 3
3.3 Use Cases 3
3.3.1 Use Case #1 3
3.3.2 Use Case #2 3
3.4 Classes / Objects 3
3.4.1 <Class / Object #1> 3
3.4.2 <Class / Object #2> 3
3.5 Non-Functional Requirements 4
3.5.1 Performance 4
3.5.2 Reliability 4
3.5.3 Availability 4
3.5.4 Security 4
3.5.5 Maintainability 4
3.5.6 Portability 4
3.6 Inverse Requirements 4
3.7 Design Constraints 4
3.8 Logical Database Requirements 4
3.9 Other Requirements 4

4. Analysis Models 4

4.1 Sequence Diagrams 5
4.3 Data Flow Diagrams (DFD) 5
4.2 State-Transition Diagrams (STD) 5

5. Change Management Process 5

A. Appendices 5

A.1 Appendix 1 5
A.2 Appendix 2 5

1. Introduction

The introduction to the Software Requirement Specification (SRS) provides an overview of the movie ticketing system, including its purpose, scope, definitions, references, and lastly its organization. This section seeks to establish a common understanding of the system's objectives and functionality, before dwelling on the functional/non-functional requirements in further sections.

1.1 Purpose

The purpose of this SRS is to provide a foundational understanding of the features, functionality, goals, and parameters of the movie ticketing system. It will describe the system's intended user interface and how it is designed to meet the needs and wants of the target audience.

This document ensures that all members, including developers, designers, testers, sponsors/investors share a common understanding of the system's objectives. It will act as a reference throughout the entirety of the SDLC, helping support design, implementation, and approvals of changes leading to the final validation of the system.

1.2 Scope

The scope of this project is to design and implement a web based movie ticketing system that will overall enhance the moviegoing experience users. The system will enable users to browse available movies across different movie theaters; as well as the ability to view showtimes up to two weeks in advance. It will provide detailed information for each movie, including title, genre, duration, rating and a synopsis of the movie, as well as an interactive map for selecting seats to allow guests to reserve their preferred seats. The system will support secure payment processing, after which the system will deliver the tickets via email and/or mobile number as a barcode digital ticket. To encourage engagement and loyal customers, we will promote account creation by providing personalized promotions, and a concise description of our free rewards program. This rewards system will be based on the dollar amount spent. These points will be able to be redeemed in stores for discounted, or free concessions, or free admission. The system will allow access to rated movies, however it will not handle age verification; identification will be checked on-site at the theater. The system will include small print stating the movie is only for certain ages. Overall, this system's goal is to streamline purchasing tickets by creating a user friendly platform that supports loyalty, and most importantly is secure.

1.3 Definitions, Acronyms, and Abbreviations

Reward Points	A loyalty system where users earn points based on dollar amount spent that can be redeemed for discounts or free items. 1 dollar spent = 100 points, equates to about ten cents back per dollar.

UI	User Interface
API	Application Programming Interface
HTTPS	Hyper Text Transfer Protocol Secure
PCI DSS	Payment Card Industry Data Security Standard
CCPA	California Consumer Privacy Act

1.4 References

- PCI DSS/CCPA - regulations by governing bodies

1.5 Overview

The purpose of the remaining sections is to provide a general description, including the characteristics of the users, the hardware requirements. As well as the data requirements needed for this product. Section two discusses the general description of the project, section 3 gives the functional requirements, data requirements, as well as the constraints and assumptions made when designing the ticketing system. Section 4 provides analysis models, sequence and data flow diagrams. Section 5 is for managing changes and additional documentation as required.

2. General Description

The system will be designed to enhance the overall movie going experience for our users, and will further enhance loyal clients through rewards programs. Everything will be on the platform, from reviews, showtimes, seat selection; this way the system drastically simplifies the task of planning going to the theater, increasing the customer traffic. In addition the system will have a secure way of storing payment information, accepting Mastercard, Visa, and Applepay.

(added a bit here to include other features from 1.2 scope - GD)

2.1 Product Perspective

The movie ticketing system is an independent, and self-contained software that is designed to function as a web-based application. This application will be accessible from both desktop and mobile browsers. It combines features that may normally be spread across different platforms into a single unified system. Though the system will be primarily standalone, it will access external systems such as a secure payment gateway, email and SMS functions for ticket delivery, a theater database for accurate showtimes, pricing and seating information. Lastly it will access review sites with an API.

In the perspective of the system, the software will interface with the users through a graphical and detailed interface that supports intuitive navigation. As well as interactive seat selection, showing a clear map of the seats, and how close and centered they are in accordance to the screen. The software will rely on standard hardware and require an internet connection, requiring only a device and any modern web browser. The software will also integrate with outside communication programs for delivery of tickets, and allow the user to allow real time updates. Security is important to users, the system will HTTPS to protect sensitive information such as card/personal information.

In summary, this product is designed as user friendly, centralized, and importantly a secure solution to streamline the movie going experience.

2.2 Product Functions

Our product will filter movies based on genre, relevance, and price. Users will be able to see the showtimes, seat availability, and price based on different nearby movie theaters. Along with the base product there will be an option to pay for a monthly subscription to our site which will provide cheaper prices for movies, food, drinks, etc. The subscription will also provide rewards weekly. However for all users there will be a point reward system based on dollars spent which will gift users free items based on points accumulated.

2.3 User Characteristics

The users will be able to open the app and see the showtimes of certain movies in nearby movie theaters. On top of that they will also be able to see ticket prices and seats available at the nearby theaters. We also can't forget about the subscription that the user has the option to pay for, this monthly subscription will provide the user with cheaper movie tickets, and on top of that will get cheaper food and drinks. The subscription will also provide rewards, such as a weekly reward for being a member. The user will also get more rewards thanks to our rewards system which will be based on dollars spent that the user can use to get free items such as movie tickets, food, drinks, etc.

2.4 General Constraints

The movie ticketing system will be subject to many different constraints that will influence its design and implementation. The system will rely on the users hardware and software, the system requires any modern browser, a stable internet connection, and basic hardware. Additionally, updates will need to be made rarely, and quickly to prevent downtime; downtime can lead to discrepancies in the database in terms of available tickets. Reliability is important to us and our users. Lastly the system depends on an external database that must be in sync with the movie ticketing system, a secure payment gateway, and email/SMS services for delivery of tickets.

In addition the system must meet many regulatory requirements in regards to the security of the system. It must meet laws such as PCI DSS when handling sensitive card information for users, and CCPA for guarding the users information; also allowing the user to see what of their information we store. These constraints limit and define the boundaries this system must operate within. These constraints will ensure the final product is scalable, reliable, and most importantly compliant with laws and secure.

2.5 Assumptions and Dependencies

For the system to have a successful development and deployment, the system depends on several assumptions and factors outside of the system ecosystem. It's assumed that each of the theaters participating will have accurate and up to date information that the system relies on such as movies, showtimes, pricing, and seat availability through their databases. It is assumed that third party payment gateways will remain compliant and reliable to ensure a safe, secure and easy transaction process.

The system also depends on the stability of email and SMS services to deliver the user their tickets in a timely manner. Should these services experience outages or delays, our users will be impacted, impacting the company's image and reputation. In addition, it relies on the user having a stable internet connection, and a modern web browser that is capable of running our application effectively. Lastly the success of the subscription model is dependent on the participation of theaters in promoting it.

3. Specific Requirements

3.1 External Interface Requirements

The movie ticketing system will interact with users, communication services and external applications through a list of defined interfaces.

3.1.1 User Interfaces

The system's user interface will be web-based and be accessible from both desktop and mobile browsers. In this UI users will be able to browse through different movies, allowing them the option to filter by genre or theater. The UI is to be designed with accessibility in mind, it should be clean and simple. The UI will have three main 'tabs' at the top, 'movies', 'theaters', 'My Rewards'. Movies will show every movie currently across all of the theaters, along with their showtimes. Theaters will allow the user to pick a specific theater and choose a movie from there. 'My Rewards' allows the user to view and redeem their points.

3.1.2 Hardware Interfaces

The system will require no special hardware beyond a device that is able to run a modern web browser and able to hold an internet connection. The system will rely on database infrastructure provided by the theater to have reliable data on movies, showtimes, pricing, and available seats.

3.1.3 Software Interfaces

The system will rely on third party payment gateways for processing payments. As well as API's to communicate with the database, retrieving and updating data such as available seats. In addition to that an API, or scraping the web is required to efficiently include movies reviews and synopsis from review sites. Lastly email/SMS services will be integrated for simple delivery of tickets.

3.1.4 Communications Interfaces

For all communication between the user, the system, and outside services will happen using HTTPS to ensure secure transmission of information. With these outside services, the system must have reliable API communication to ensure a smooth experience for the user. If a service is down, the user will be notified with a message informing them an error has occurred.

3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.2.1 <Movie Browsing and Filtering>

3.2.1.1 Introduction

This function will allow users to browse all of the available movies at all of our theaters. Users can filter the movies by genre, theater, price, showtime to find the perfect movie for them.

3.2.1.2 Inputs

- The user will access the 'Movies' tab from the main UI
- The user can also access the 'Theaters' tab to view all theaters, from there selecting a movie from their chosen theater.
- Users will see provided filters of genre, showtime, price and possible theaters if they first put their input in the movies tab.

3.2.1.3 Processing

The system will receive the users request to filter the movies, by the requested method. The system then accesses the theaters database through APIs. The system then will format the data(movies) into the order requested on the UI.

3.2.1.4 Outputs

- A list of movies by popularity by default across all theaters.
- For each movie, genre, duration, rating, synopsis, title, and showtimes available varying by theater
- Ratings and synopsis will be integrated from third party APIs.
- Reward progress and feedback

3.2.1.5 Error Handling

- If no movies fit the criteria of the filter, the system will display a message stating "No movies are available within the filter(s)"
- If a critical error occurs and the database cannot be reached, the system will display a message stating " Movie data cannot be retrieved, please try again at a later time"

- If a third party review site fails, the movie will simply not display reviews/synopsis rather than displaying an error message.

3.2.2 <Theater, Showtime, and Seat Selection>

3.2.2.1 Introduction

This function allows users to select a theater, view and select a showtime for their choice of movie, and lastly select their preferred seating choice, assuming it's available. Ensuring their visit goes as planned.

3.2.2.2 Inputs

- User accesses theater, two ways the user can access a specific theater.
  - After selecting a movie the user is prompted to select a theater that is showing the movie
  - The user first inputs their theater choice using the ‘theater’ tab. Then inputting their choice of movie.
- The system receives the user input for movies and shows showtimes, the user then selects a showtime.
- User selection of one or more(max 20) available seats from our interactive map.

3.2.2.3 Processing

The system will process the user’s selection of theater by accessing the theaters database to retrieve the pricing, schedule, and seating information. After the showtime is selected, the system will generate an interactive map that shows reserved, and available seats in real time. In order to prevent double booking, the moment a user selects a(or more) seat they are temporarily reserved for 10 minutes. Payment must be completed within the 10 minutes, or the ticket will be forfeited.

3.2.2.4 Outputs

- List of all the movies current showing at the selected theater, including the showtimes and prices.
- An interactive map to show the user the available seats left in the selected theater, movie, showtime.
- Confirmation of the user's selected seat for ten minutes before payment.

3.2.2.5 Error Handling

- If no movies are scheduled at a selected theater, a message should display stating “ no movies currently available at this location, please try again at a later date”.
- If the seat map fails to load/update the system should display a message indicating “Seat availability currently unavailable, please try again.”.
- 10 minute grace period the moment a seat(s) is selected to not allow double booking, handing the error before it happens.

3.2.3 <Ticket Purchasing>

3.2.3.1 Introduction

This function allows the user to confirm their selected seat(s), then the system transfers them to the secure payment gateway.

3.2.3.2 Inputs

The site will require various inputs including location, filters, payment method, subscription offers, email and/or phone number.

3.2.3.3 Processing

Once the user gets transferred to the payment gateway, and their payment is successful, their confirmation code along with their ticket will be sent to their email/SMS. The system will have the receipt and barcode sent to the user's email.

3.2.3.4 Outputs

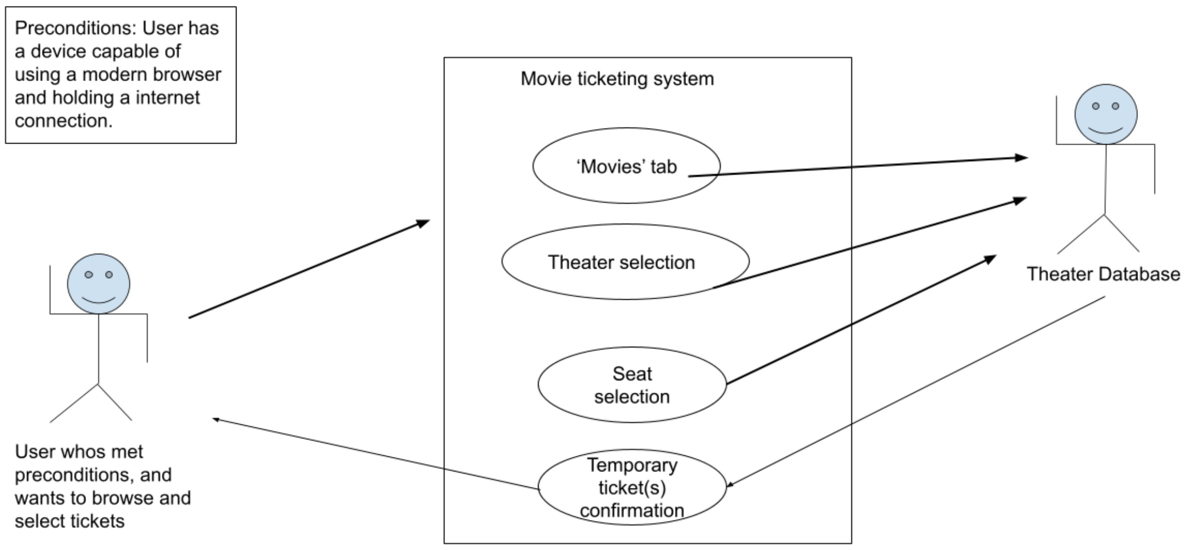
- If the ticket purchase is successful the system will display a message stating ”Your ticket purchase was successful! Your ticket will be available under “My Tickets”. The system will also send an email or text confirmation based on the information given by the user.
- If the user purchases a subscription successfully the system will display a message saying “Your purchase was successful, thank you for subscribing!”

3.2.3.5 Error Handling

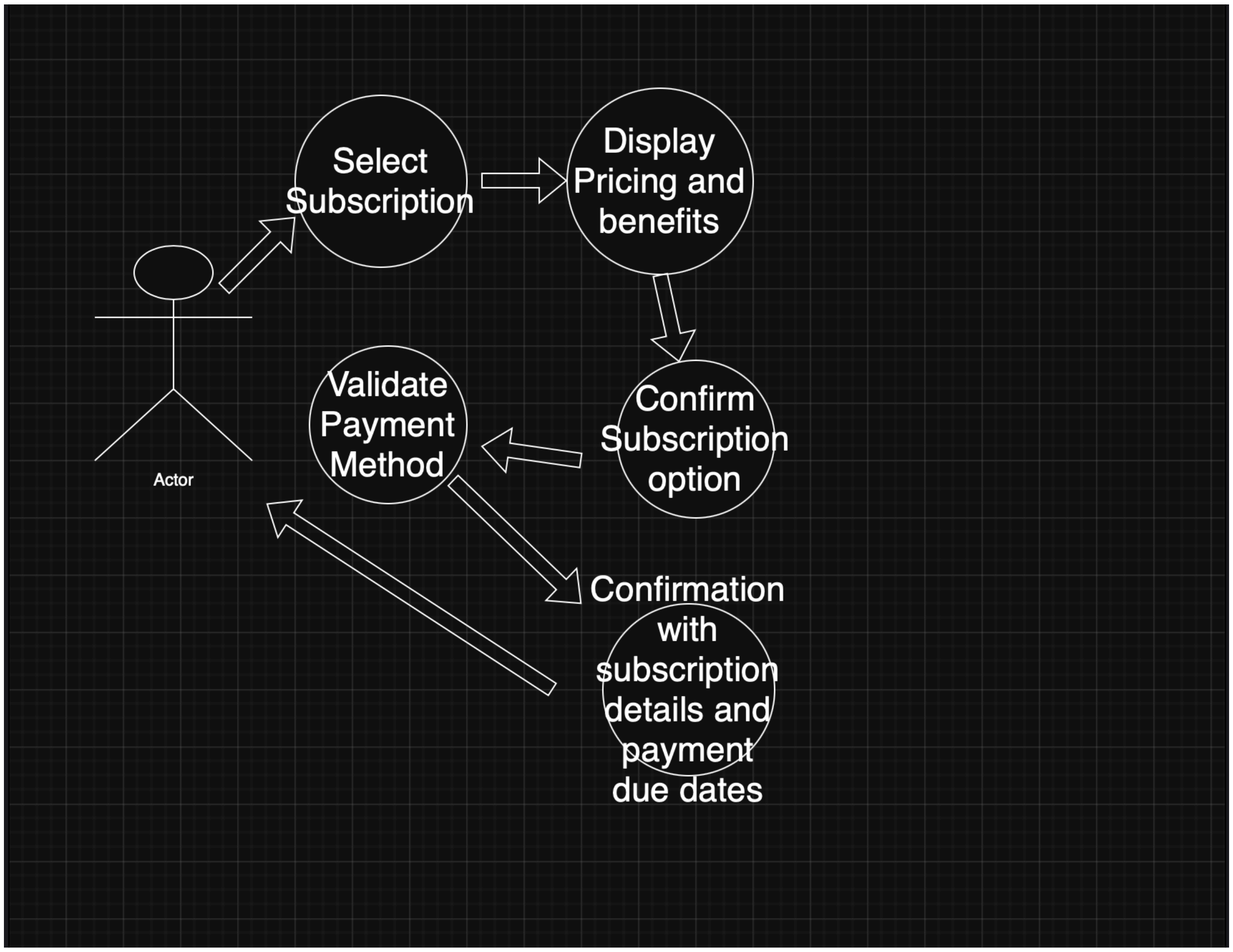
- If the payment method is invalid a message will be displayed saying “Payment method is invalid/incorrect, please try a different method”
- If no movies are showing at the time the system will display a message stating “There are no movies currently available near your location, please try again later.”
- If no movies fit the filters set by the user the system will display a message saying “ There are no movies available given the criteria”

3.3 Use Cases

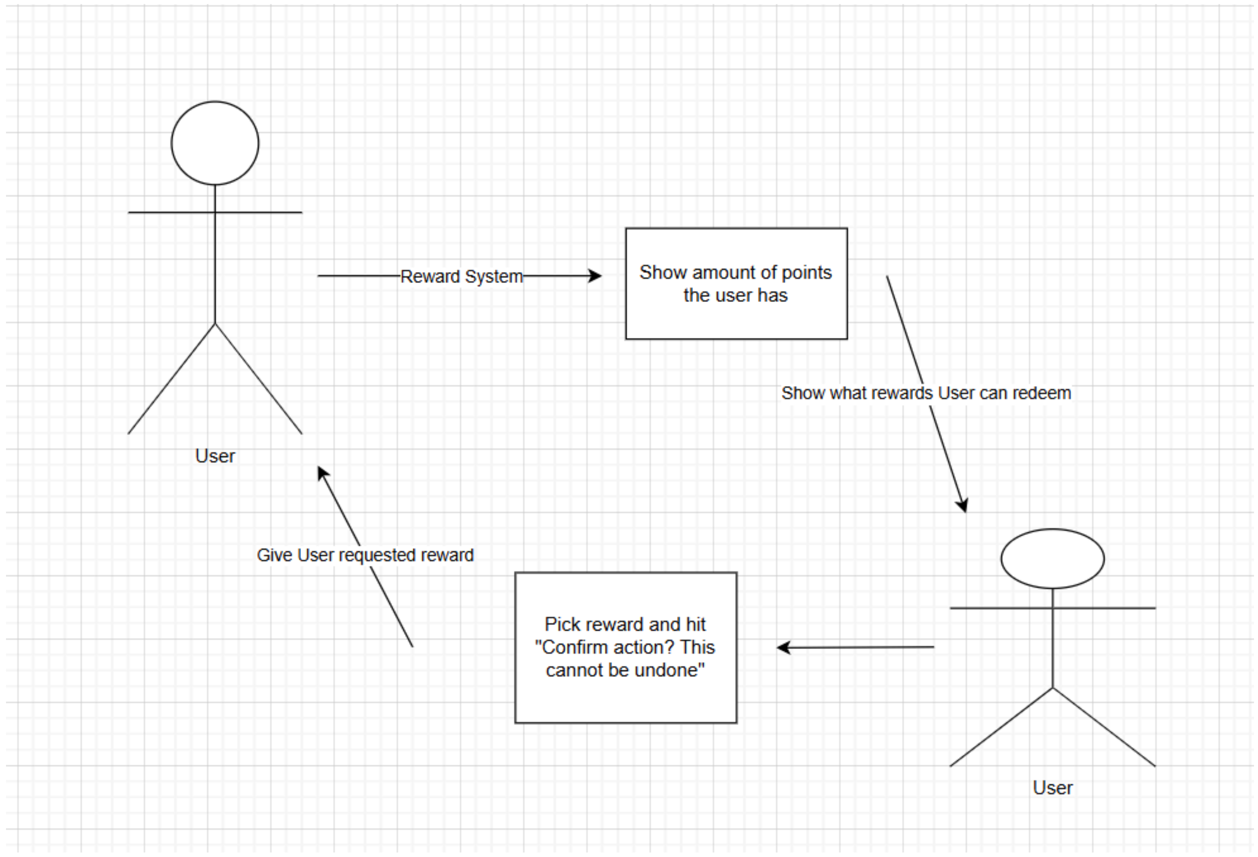
3.3.1 User browses and selects tickets


Preconditions: User has a device capable of using a modern browser and holding a internet connection.
Movie ticketing system
‘Movies’ tab
Theater selection
Seat selection
Temporary ticket(s) confirmation
Theater Database
User whos met preconditions, and wants to browse and select tickets

3.3.2 User case #2


Actor
Select Subscription
Display Pricing and benefits
Confirm Subscription option
Validate Payment Method
Confirmation with subscription details and payment due dates

3.3.2 Use Case #3


Reward System
Show amount of points the user has
Show what rewards User can redeem
User
Give User requested reward
Pick reward and hit "Confirm action? This cannot be undone"
User

3.4 Classes / Objects

3.4.1 <User>

3.4.1.1 Attributes

- UserId
- Name
- Email
- PhoneNumber
- Password
- AccounType
- PointBalance
- Location

3.4.1.2 Functions

- CreateAccount()
- Login()
- ViewRewards()
- Redeemrewards()
- UpdateProfile()
- ViewTickets()

References:

- Use case 3.3.1

3.4.2 <Movie>

3.4.2.1 Attributes

- MovieID16
- Title
- Genre
- Duration
- Synopsis
- Reviews

3.4.2.2 Functions

- DisplayMovieDetails()
- FilterMovies()
- GetReviews()
- GetSynopsis()

References:

- 3.2.1 Functional Requirements
- Use case 3.3.1

3.4.3 <Theater>

3.4.3.1 Attributes

- TheaterID16
- Location
- Name
- ShowTime
- SeatMap
- Movies

3.4.3.2 Functions

- DisplayMovies()
- GetShowTimes()
- GetSeatMap()
- ValidateTicket()

References:

- 3.2.2 Functional Requirements
- Use Case 3.3.1

3.4.4 <Ticket>

3.4.4.1 Attributes

- TicketID16

- TheaterID16
- MovieID16
- SeatNumber
- Price
- ShowTime
- RewardsPointsEarned
- Barcode
- DeliveryMethod

3.4.4.2 Functions

- CreateBarcode()
- DeliverTicket()
- UpdatePoints()
- ConfirmPurchase()
- VoidTicket()
- DeliverReceipt()

References:

- 3.2.3 - Functional Requirements
-

3.5 Non-Functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transactions shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).

3.5.1 Performance

- The system will process at least 95% of user requests in under 2 seconds. These requests can be things such as browsing movies, or selecting showtimes.
- Payment confirmation process will complete in under 5 seconds under normal conditions.
- Email containing payment receipt and possible barcode shall be delivered in under 20 seconds.
- The system needs to constantly update the seal available in real time with a max delay of 1 second.

3.5.2 Reliability

- The system will be able to recover on its own from minor faults such as communication delays without the user doing anything.

3.5.3 Availability

- The system will be running at optimal operations 99.5% of the time, this will exclude scheduled updates and maintenance.
- Confirmed transactions will be stored in a secure and outside of the system in case of system failure.

3.5.4 Security

- All communication between user, system and external services will be encrypted using HTTPS.
- Password storing will be up to industry standard. Passwords will be encrypted using 256 bit encryption.
- The system and payment gateway shall comply with PCI DSS and CCPA guidelines.
- Email verification will be implemented when a user wants to sign in for an extra layer of security.

3.5.5 Maintainability

- Routine updates and bug fixes will be applied and run, while minimizing the down time.
- Source code shall have sufficient documentation at industry standards. This allows easy onboarding and helps with debugging.
- The system will be created using OOP to compartmentalize classes and methods so developers can work and update a section at a time.

3.5.6 Portability

- The system will be fully operational on all major browsers.
- The system will support both mobile and desktop browsers. Also supports a variety of screen sizes from 5in - 27in.
- No requirement of specialized software, easy access, only internet connection and basic hardware is required.

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 to 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.

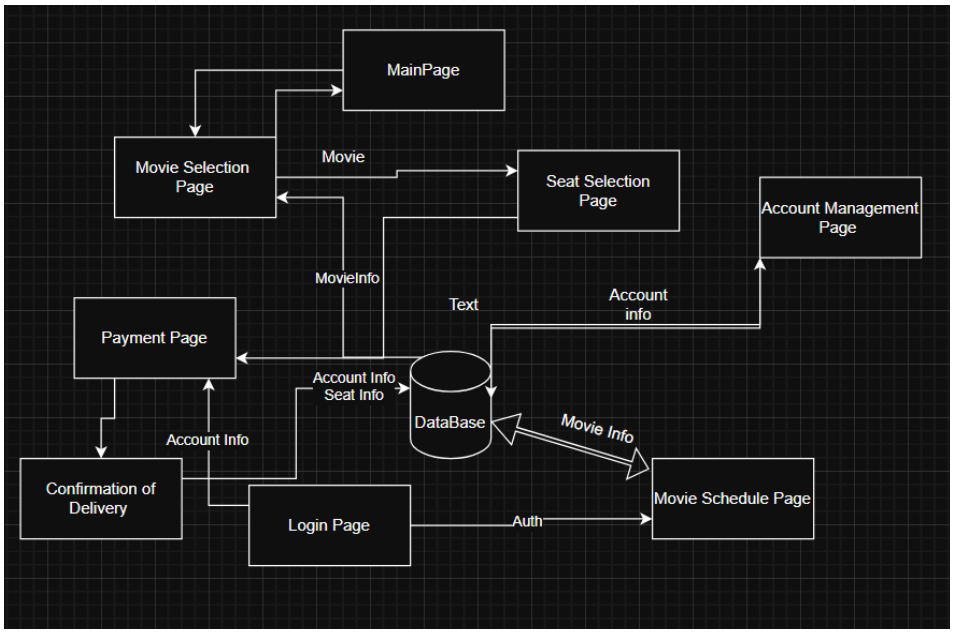
6. Software Design Specification

6.1 System Description

- In the user class, the system will allow the user to update the profile, view the tickets, view and redeem awards, create and log in to his/her account, and update their profile
- In the movie section, the user can display the movie details, filter the movies, get the movie reviews, and get a synopsis from the API
- In the theater section, the user can display the movies, get the movie show times, access the seat map to see which seats are available, and even see if there are any promotions

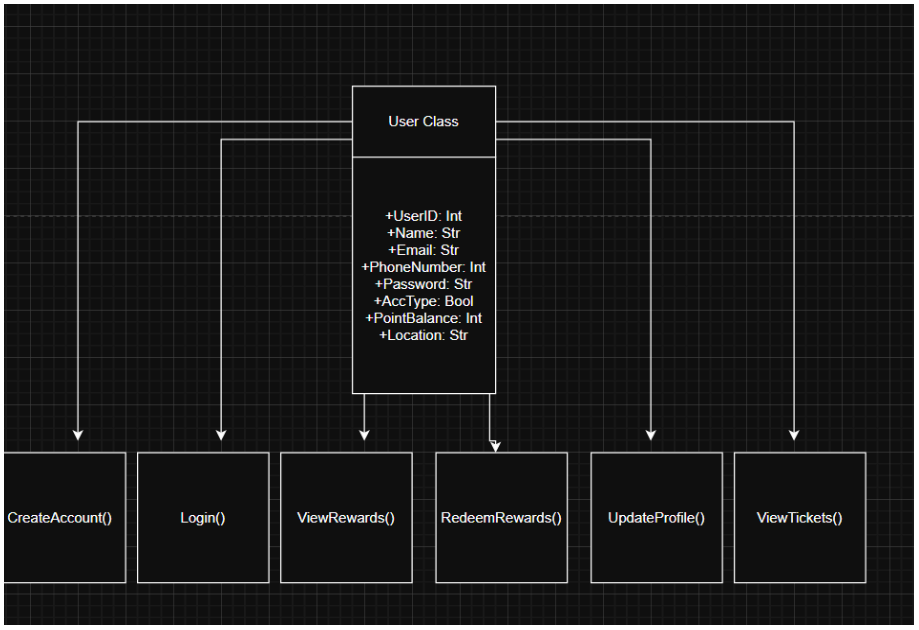
6.2 Software Architecture Overview

6.2.1 Architectural Diagram of All Major Components

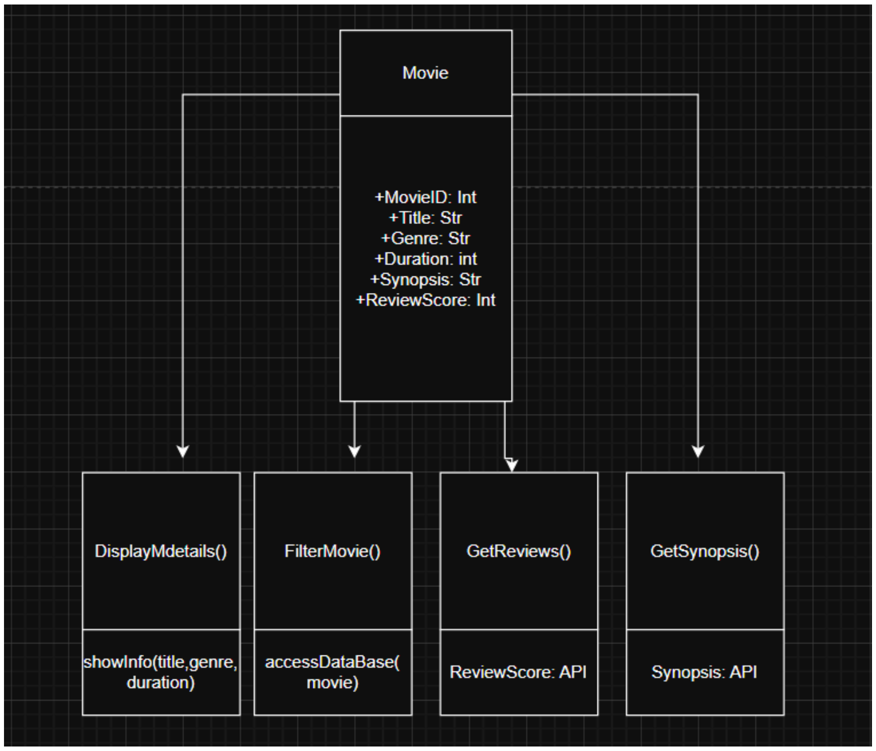


6.2.2 UML Class Diagram

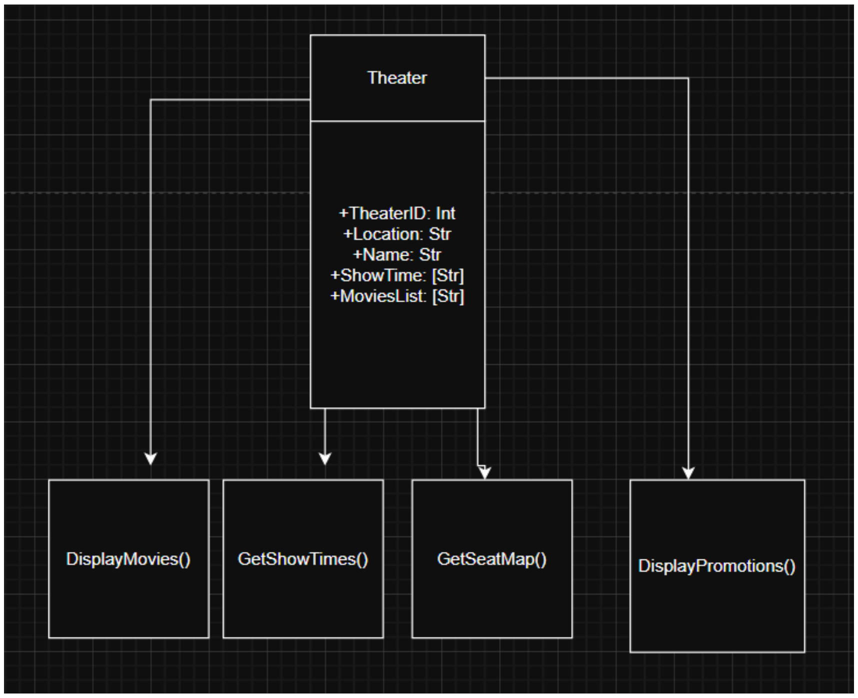
6.2.2.1 User



6.2.2.2 Movie



6.2.2.3 Theater



6.2.3 Description of Classes, Attributes and Operations

6.2.3.1 User

The User class represents an individual account holder within the SR system. Each User object stores personal and authentication information, allowing the system to manage user registration, login, rewards, and ticket history. The class maintains core attributes such as identification details, contact information, and loyalty status. It also provides methods to handle essential user operations like creating accounts, logging in, updating profiles, redeeming rewards, and viewing tickets. Through this class, the system can securely authenticate users, track their activity, and personalize their experience.

The User class contains several attributes that define each account. The UserId attribute is a unique identifier of type UUID that distinguishes each user in the database. The Name is of type String, stores the user's full name and must contain between one and one hundred characters. The Email attribute, also a String, serves as the user's primary contact and login credential; it must be unique and follow valid email formatting standards. The optional PhoneNumber attribute, of type String, stores a contact number.

The Password attribute is a sensitive field of type Secret that contains the user's authentication credential, which must be stored securely. The AccountType signals if a user has upgraded to a paid membership. The PointBalance attribute, an Integer, represents the user's accumulated loyalty points and must always be greater than or equal to zero. Lastly, the optional Location attribute, of type String, contains the user's city or regional information for personalization.

The User class supports several operations that allow for account management and user interaction within the system. The CreateAccount() function allows a new user to register by providing parameters such as name, email, password, phoneNumber, accountType, and location. The method validates that the email is unique and that the password meets security standards. Upon success, it generates a unique UserId, initializes the PointBalance to zero, and stores the password securely.

The Login() function authenticates a user using their email (or phone number, if supported) and password. If the credentials are correct, the method returns an AuthToken that grants session access. Invalid credentials or inactive accounts cause the operation to fail with an appropriate error.

The ViewRewards() operation retrieves the user's reward information, returning details such as their current PointBalance, membership tier, and a list of available rewards. This is a read-only operation and does not alter the system's state. The Redeemrewards() function allows users to redeem a specific reward by providing their userId, the rewardId, and an optional quantity. It ensures the user has sufficient points before deducting the appropriate amount and issuing a redemption result that may include a voucher or confirmation message.

The UpdateProfile() operation enables a user to modify their personal details such as *name*, *email*, *phoneNumber*, *location*, or *password*. It enforces validation rules and ensures that sensitive updates, such as email or password changes, meet security policies. Lastly, the ViewTickets() operation retrieves a list of tickets associated with the user's account, with optional filters for date range or ticket status. This method allows users to view their past and upcoming bookings without altering any stored data.

6.2.3.2 Movie

The Movie class represents films that can be displayed, filtered, or reviewed within the system. It stores essential details about each movie, allowing users to view its description and feedback, and enabling other system components (such as theaters or schedules) to reference it. Each movie instance serves as a container for descriptive and review information related to a single film.

The Movie class contains several core attributes. The MovieID attribute is a unique integer (16-bit) identifier assigned to each movie. The Title attribute, of type String, holds the movie's name and must be unique and non-empty. The Genre attribute, also a String, categorizes the movie into something like 'action' 'adventure' 'romance'. The Duration attribute, of type Int, represents the movie's length in minutes and must be greater than zero. The Synopsis attribute, of type String, provides a short summary of the movie's plot for display to users. Finally, the Reviews attribute will be a list/array that stores user-generated comments and/or scraped ratings related to the movie.

The DisplayMovieDetails() function retrieves and displays all major attributes like title, genre, duration, and synopsis for a selected movie. The FilterMovies() function allows movies to be searched or filtered by attributes such as genre, duration range, or keyword in the title. The GetReviews() function returns the list of stored reviews or ratings for a movie, which may be aggregated for average scores. Lastly, the GetSynopsis() function retrieves and displays the movie's synopsis in a concise format for user viewing.

6.2.3.3 Theater

The Theater class represents a physical cinema location or screening venue within the SR system. It manages information about where movies are shown, including their schedules, seating arrangements, and available films. Each theater instance links movies to their showtimes and provides access to seating and ticket validation features used during booking or entry.

The Theater class contains several attributes that describe a theater's essential details. The TheaterID attribute is a unique integer (16-bit) identifier that distinguishes each theater. The Location attribute, String, specifies the theater's address. The Name attribute, also a String, stores the official name of the theater branch/number. The ShowTime attribute, of list/array, contains scheduled times for movie screenings. The SeatMap attribute, represented as a 2D array or matrix, defines the seating layout, including seat numbers and availability. Finally, the Movies attribute, of list/array, holds the collection of movies currently playing or scheduled at the theater.

The Theater class includes several key operations for managing and displaying theater information. The DisplayMovies() function lists all movies currently showing or scheduled at the theater, using data from the Movies attribute. The GetShowTimes() function retrieves the available screening times for a selected movie or for the entire theater schedule. The GetSeatMap() function returns the current seating arrangement, including booked and available seats, for a particular showtime. Lastly, the ValidateTicket() function verifies the authenticity and validity of a user's ticket by checking the showtime, seat, and movie information against theater records.

6.3 Development Plan and Timeline

The project will be split into 4 developmental phases, totaling a total of 10 weeks.

Phase 1: Week 1-2 will cover requirements gathering, database and UI design.
(Michael)
Phase 2: Week 3-6 will cover module coding and unit testing.
(Isaac, Michael, Gershom)
Phase 3: Week 7-9 will cover integration and system testing.
(Isaac and Michael)
Phase 4: Week 10 will cover deployment and documentation
(Gershom and Isaac)
The work will be divided into 'modules'

User Management Module: Handles account creation, login, profile updates, and reward tracking.

Movie and Theater Management Module: Manages movie details, genres, showtimes, and theater information.

Ticket Booking and Payment Module: Enables users to select movies, view seat maps, and book or cancel tickets.

Database Design and Integration: Defines the schema, relationships, and queries to ensure data integrity.

Testing and Quality Assurance: Involves functional testing, usability evaluation, and verification of user scenarios.

7. Test Plan

7.1 Purpose

This section describes the verification and validation plan for the WAD Movie ticketing system. The goal of this testing plan is to confirm that all the functional and non-functional requirements defined in earlier sections have been implemented correctly. Also confirming the system performs as intended under normal conditions. Testing will also comprise of ensuring that the system also behaves as expected with realistic user inputs and actions in the form of test cases,

7.2 Scope of the Test

- **In Scope**
  - Core classes, user, theater, movies, ticket.
  - Use of APIs and the data interfaces, payment gateway, ticket confirmation and delivery. Also SMS delivery.
  - Behavior of the system such as timer for holding seats, redemption and updates to user points. Also secure login for users.

- **Out of Scope**
  - Ads on our page, marketing is out of scope

7.3 Verification and Validation Outline

Testing begins with verification. The verification of the program ensures that each part of the created system was built correctly and meets its respective design specifications. Validation will ensure that the system as a whole fulfills the requirements set by the stakeholders and by users.

The tests will verify the functionality of the defined classes in section 6 of the SRS. Verify that the integration and communication of the subsystems are working as intended; in addition to the non-functional requirements stated earlier in this SRS.

7.4 Testing Strategy

7.4.1 - Unit Testing(verification)

Objective: Verify that each class and method defined in section 6 (User, Movie, Theater, Ticket) functions correctly and adheres to its design contract. Unit testing confirms that all core operations behave as intended in isolation before integration.

Focus: The examples below illustrate representative unit targets. Each class and method described in section 6.2.3 will include similar verification to ensure overall coverage across the system.

- CreateAccount() (User class): Validate email uniqueness and password-strength enforcement.

- RedeemRewards() (User class): Confirm that the user's PointBalance never drops below zero.
- GetSeatMap() (Theater class): Ensure seat availability is returned accurately using mocked database data.
- ConfirmPurchase() (Ticket class): Verify that payment callbacks atomically update ticket state and prevent duplication.
- Additional Examples: Login(), ViewRewards(), UpdateProfile(), FilterMovies(), GetShowTimes(), ValidateTicket(), DeliverReceipt() will get the equivalent unit-level tests following the same objective.

Test Vectors:
Valid vs. invalid input sets, boundary values, and intentional error conditions (API timeouts, null inputs) will be used for both expected and exception paths.

7.4.2 - Functional Testing

Objective: Validate that the modules interact with each other correctly. Also ensure that the features visible to the user are accurate when modules are interacting with one another.

Test Sets:
- Checkout flow: hold seats -> payment -> confirmation -> ticket delivery
- Browsing flow: browse movies -> filter -> select showtime -> show seat map

These test sets cover many possible issues with the system; covering possible missing API retrievals, payment declines, ticket hold timer expiration and session timeouts. Also ensuring the database is accessed when needed.

7.4.3 - System Testing(validation)

Objective: Confirmation that the system meets all requirements, both functional and non-functional in the required environment.

Test Sets:
- Performance and load: verify system functionality is normal with >200 concurrent users performing actions such as browsing movies. Also validating the latency, ensuring the response time for users is valid.
- Security and Reliability: Ensure HTTPS is working, system recovers automatically from failures from APIs. Also ensure passwords are secured.

7.5 Coverage of Tests and Failure reports

Each test set is to a specific class and interface within the UML diagram in section 6 of this document.

- **Unit Tests**: Internal methods within each class.
- **System Tests**: Operations within the system, for example user load, and processing of user requests.
- **Functional Tests**: Connections between the modules in the classes.

The types of failure covered are; input validation errors, bottlenecks within performance, security(HTTPS). In addition to encryption of passwords.

7.6 Test Cases

10 test cases have been documented on the spreadsheet located in the shared github repository. These cases cover unit, functional, and system testing. Each test case includes a testCaseId, Component, Priority, Description/Test Summary, Test Steps, Expected Result, Actual Result, Status(Pass/noPass), and lastly the name of the team member who executed the test.

This proper documentation allows for dev's to properly and efficiently debug problems and faults in the system. As well as accountability and traceability to tests that have been run.

7.7 Criteria for Acceptance

The system is considered verified and validated when it has met the following requirements:

- All planned tests have been executed, and passed with no critical errors.
- Acceptance of functional tests on first execution is >=95%
- All non functional requirements from section three in the SRS have been met under normal conditions.

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