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.

<Theater Ticketing System>

Software Requirements Specification

<Version >

<2/13/2025>

<Group 7>
<Anthony Nguyen, Steven To, Charlie Pham>

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

Revision History

Date	Description	Author	Comments
<date>	<Version 1>	<Your Name>	<First Revision>
<02/13/25>	<1.0>	<Group 7>	Completed all of the Introduction
<02/17/25>	<1.1>	<Group 7>	Completed all of 2
<02/19/25>	<1.2>	<Group 7>	Completed 3.1-3.5

Document Approval

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

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

Table of Contents

1. Introduction

The introduction to the Software Requirement Specification (SRS) provides an overview of the complete SRS including the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the SRS. The objective of this document is to obtain and provide an in-depth look at the complete **theater ticketing system** by addressing the desires of the consumers in detail. The detailed requirements of the **theater ticketing system** are provided in this document.

1.1 Purpose

The purpose of this document is to help guide and serve as a foundation for the various phases of developing a movie theatre ticketing system. This system is designed to facilitate the consumption of tickets and is intended for use by individuals aged 14 and above.

1.2 Scope

The theater ticketing system is a web-based application developed to provide an efficient and seamless process for customers to purchase and manage digital movie tickets online. The system will enable users to browse available movie screenings, select their seats, and make reservations or purchases. The system will also have administrative features such as adding and updating movie schedules, setting ticket prices, and managing seat availability. The theater ticketing system aims to enhance customer convenience and improve operational efficiency. The system is designed to be secure, ensuring a smooth user experience.

1.3 Definitions, Acronyms, and Abbreviations

FAQ’s	Frequently Asked Questions
API	Application Programming Interface
SMS	Short Message Service
ID	Identity or Identification
MySQL	My Structured Query Language
iOS	iPhone Operating System
macOS	Macintosh Operating System

1.4 References

The references are:

- Gerhard Krüger. (2018). How to Write an SRS Document (Software Requirements Specification Document) | Perforce Software. Retrieved February 21, 2025, from Perforce Software website: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document
- admin@relevant.software. (2023, March 23). Your Guide to Writing a Software Requirements Specification (SRS) Document. Retrieved February 21, 2025, from Relevant Software website: https://relevant.software/blog/software-requirements-specification-srs-document/
- IEEE-830-1998.pdf
- Intro to software architecture and design.pdf

1.5 Overview

This document provides a comprehensive specification for the theater ticketing system, which details its requirements, functionalities, and constraints. It is structured to ensure ease of reference for developers, stakeholders, and designers. Section 2 presents a general description of the system, which includes its product perspective, primary functions, user characteristics, constraints, and dependencies. Section 3 gives the specific requirements of the system, defining functional aspects of the system such as external interface requirements, functional requirements, and use cases. It also includes details on non-functional requirements such as performance, security, usability, and system constraints. Section 4 discusses additional factors such as assumptions, dependencies, and design constraints. Section 5 provides an overview of possible future enhancements for the system.

2. General Description

The theater ticketing system is designed to provide an efficient and user-friendly online ticketing experience for users while ensuring efficient management for theater administrators. This section provides an overview of the general factors that influence the system's development and usage, making it easier to understand the specific requirements shown later in the document.

2.1 Product Perspective

The theater ticketing system is a web-based application designed to replace traditional box office ticket tales by offering a digital, real-time booking platform. It will also integrate with third-party payment processors to enhance the user experience. The system provides an interface for managing movie schedules, prices, and sales reports, reducing manual work and improving efficiency. Compared to existing ticketing systems, this system aims to offer a more seamless and user-friendly experience with features such as real-time seat selection and automated notifications. Additionally, the system will incorporate a customer feedback link that leads to a review platform where they can rate and review movies. The system can give personalized movie recommendations to users with an account based on their movie history and reviews on our customer feedback platform.

2.2 Product Functions

Our product offers a multitude of different languages based on region when a user is accessing our website. Our product also has an account feature that the users can access, either login or create an account. This is where the administrators can log in and use the system in administrator mode. In this mode, they can change seats and showings. Once in they can select the movie of their choice. The customers are expected to know what movie they want to watch. Therefore, we have a search option that allows the user to search for the movie they want to watch. However, in the case that they don't, we have a featuring movie list page that shows movies from the most popular to the least popular. The account is linked to a phone number so they will receive a text message when the movie is about to start. The software will count how many seats are available based on the room of the movie at a certain time, and display when the movie is chosen. The system will also have a back button so users can go back if they were to change their minds. However, this feature will be removed once their payment is processed. Moreover, once they reach the payment page, our machine will accept payments in the form of Credit/Debit cards, Apple Pay, and PayPal. The product will also be able to print the receipt and tickets.

2.3 User Characteristics

- Know what movie they want to see beforehand
- Ability to read, and change language if necessary
- Must either have payments in the form of a Credit/Debit card, Apple Pay, or PayPal
- Ticket prices are converted based on their local currency
- The system should accommodate users with disabilities by adhering to accessibility standards
- The system must provide a responsive and consistent user experience across all devices.

2.4 General Constraints

- Payment: Credit/Debit Card, PayPal, Apple Pay.
- limited to buy only 10 tickets at a time
- Ticket prices are converted based on local currency
- Compatible with major operating systems such as Windows, iOS, MacOS, and Android and web browsers such as Safari, Google Chrome, and Firefox
- The database will be MySQL
- The system must be designed to handle a growing number of users and transactions without performance degradation.
- The system should support multiple languages to cater to a diverse user base.
- Must be accessible for individuals with disabilities
- The system must provide real-time updates
- The system must provide data privacy

- The system must allow refunds and cancellation policies
- at most 2 physical machines so the system does not get overloaded with people all trying to claim the same seats at the movies

2.5 Assumptions and Dependencies

The ticketing system relies on several factors that can impact its functionality and requirements. It is assumed that users must be at least 14 years old or accompanied by an adult to purchase tickets. The system also requires stable internet access for seamless transactions and communication with third-party services, such as payment and email/text notifications. The system will support a variety of languages based on user preferences to enhance accessibility. To ensure smooth operation, administrators have to regularly update movie schedules and ticket prices. All payments will be card transactions only, and no cash payments accepted. Also, ticket prices must be automatically converted to local currency based on the user's location.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.1.1 The system shall provide a user-friendly graphical interface for desktop and mobile users.

3.1.1.2 The system shall allow users to browse movie listings, select showtimes, and purchase tickets.

3.1.1.3 The system shall include an interactive seat selection map that updates availability in real time.

3.1.1.4 The system shall provide a user account dashboard for managing bookings and payment history.

3.1.1.5 The system shall support English and Spanish for accessibility.

3.1.1.6 The system shall include a review and rating section for users to share movie feedback.

3.1.2 Hardware Interfaces

3.1.2.1 The system shall be compatible with standard desktop, tablet, and mobile devices.

3.1.2.2 The system shall support barcode and QR code scanning for ticket validation at theater entrances.

3.1.3 Software Interfaces

3.1.3.1 The system shall work with secure third-party payment gateways such as PayPal.

3.1.3.2 The system shall connect with theater management systems to retrieve and update showtimes and seat availability.

3.1.3.3 The system shall work with email and SMS services for sending notifications.

3.1.3.4 The systems shall integrate with MySQL to maintain transaction and user history.

3.1.4 Communications Interfaces

3.1.4.1 The system shall send confirmation emails and SMS notifications after successful ticket purchases.

3.1.4.2 The system shall provide alerts for canceled or rescheduled showtimes via email / SMS.

3.2 Functional Requirements

3.2.1 Ticket Purchase Process

3.2.1.1 The system shall display available movies and corresponding showtimes.

3.2.1.2 The system shall allow users to select a movie and a preferred showtime.

3.2.1.3 The system shall present real-time seat availability and selection options.

3.2.1.4 The system shall provide a detailed order summary before checkout.

3.2.1.5 The system shall process payments through a secure gateway.

3.2.1.6 The system shall generate and distribute digital tickets via email and SMS upon successful payment.

3.2.1.7 The system shall prevent duplicate bookings for the same seat.

3.2.2 User Account Management

3.2.2.1 The system shall allow users to create and manage personal accounts.

3.2.2.2 The system shall store user booking history and payment preferences.

3.2.2.3 The system shall enable users to update account details, including email, phone number, and password.

3.2.2.4 The system shall support account password recovery through email and SMS authentication.

3.2.3 Customer Support

3.2.3.1 The system shall provide a live chat feature for customer inquiries.

3.2.3.2 The system shall include a FAQ section for common troubleshooting issues.

3.2.4.3 The system shall allow users to submit refund requests for canceled showtimes.

3.2.4.4 The system shall provide theater contact details for direct assistance.

3.2.4 Ticket Cancellation & Refund

3.2.4.1 The system shall allow users to request ticket cancellations before the showtime.

3.2.4.2 The system shall provide users with refund policies before cancellation.

3.2.4.3 The system shall process eligible refunds based on theater policies.

3.2.4.4 The system shall notify users of successful cancellations and refunds via email or SMS.

3.2.4.5 The system shall restrict refund eligibility based on the purchase date, showtime, and ticket type.

3.2.5 Movie Review & Rating System

3.2.5.1 The system shall allow users to rate movies after watching them.

3.2.5.2 The system shall enable users to submit written reviews of movies.

3.2.5.3 The system shall allow users to report inappropriate reviews for moderation.

3.2.6 Search & Filter Functionality

3.2.6.1 The system shall allow users to search movies by title, genre, or release date.

3.2.6.2 The systems shall provide filtering options based on rating, language, and availability.

3.2.6.3 The system shall display search results dynamically as the user types.

3.2.7 Payment Processing & Security

3.2.7.1 The system shall support multiple payment methods, including credit/debit cards, PayPal, and Apple Pay.

3.2.7.2 The system shall encrypt all payment transactions to ensure security.

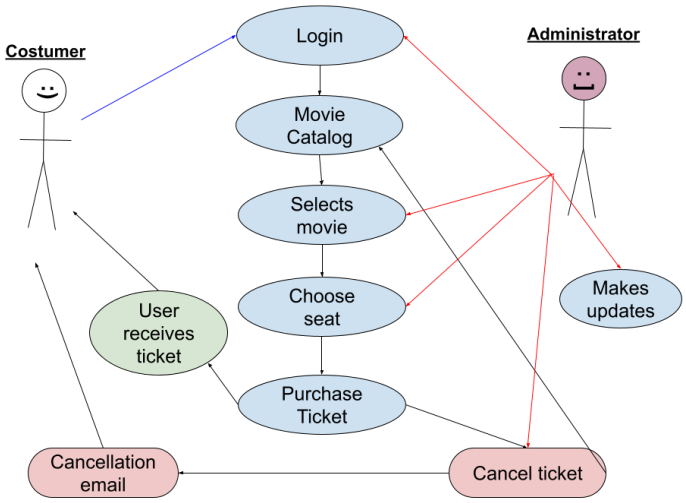
3.2.7.3 The system shall allow users to store payment methods securely for faster checkouts.

3.2.8 Administrative Management

3.2.8.1 The system shall allow administrators to add, update, and remove movie listings.

3.2.8.2 The system shall provide detailed reports on ticket sales, refunds, and revenue trends.

3.3 Use Cases



3.3.1 Use Case #1: Select a Movie

Actor(s): Customer
Preconditions:

- The customer has access to the website.
- The customer has an active internet connection.
- the customer has an account with our website
- The movie selected has available seats.

Flow of Events:

1. The customer logs into the website.
2. The system displays movies to watch in “featuring movies” and a search function.
3. The customer selects a movie through the search function or from the list of available movies.
4. The system displays the available showtimes and dates.
5. The customer selects a preferred date and showtime.
6. The system presents the seating chart with available seats.
7. The customer selects one or more seats.
8. The system updates the ticket price based on the amount of seats.
9. The customer proceeds to checkout.

Postconditions:

- The selected seat is marked as booked in the system.

3.3.2 Use Case #2: Purchase a Movie Ticket

Actor(s): Customer
Preconditions:

- The customer has a valid payment method available.
- The customer completes all of the steps to get to this page

Flow of Events:

1. The system displays the available payment options.
2. The customer enters payment details and confirms the purchase.
3. The system processes the payment and generates a digital ticket.
4. The system sends the digital ticket to the customer’s email and account.

Postconditions:

- The customer successfully receives their digital ticket.

3.3.3 Use Case #3: Cancel a Movie Ticket

Actor(s): Customer
Preconditions:

- The customer has purchased a ticket.
- The cancellation window (determined by theater policy) is still open.
- The customer has an active internet connection.
- The refund policy allows for cancellation.

Flow of Events:

1. The customer logs into their account or checks their email.
2. The system displays the customer’s active bookings.

3. The customer selects the booking they wish to cancel.
4. The system presents the refund policy and confirmation prompt.
5. The customer confirms the cancellation request.
6. The system processes the cancellation.
7. If eligible, the system initiates a refund process.
8. The system sends a cancellation confirmation to the customer.

Postconditions:

- The ticket is invalidated, and the seat is marked as available.
- If applicable, a refund is issued to the customer.

…

3.4 Classes / Objects

3.4.1 <User>

User	
3.4.1.1 Attributes	
UserID: int	A unique identifier for each user.
Name: String	The full name of the user.
Email: String	The user's email address, is used for account creation and communication.
Password: String	A secure password for account authentication.
Age: int	The user's age, is used to enforce the minimum age requirement (14+).
PaymentMethods: (List<string>)	A list of payment methods associated with the user.
3.4.1.2 Operations	
Register(name, email, password, age): void	Allows a new user to create an account.
Login(email, password:): bool	Authenticates the user and grants access to the system.
PurchaseTicket(movieID, showtime, seatNumber): Ticket	Purchases a movie ticket.

Reference to Use Cases:

Use Case #1: Select a Movie
Use Case #2: Purchase a Movie Ticket

3.4.2 <Movie>

Movie	
3.4.2.1 Attributes	
MovieID: int	Unique identifier for the movie.
Title: string	Title of the movie.
Description: string	Brief synopsis of the movie.
Duration: int	Runtime of the movie in minutes.
Genre: string	Category of the movie (e.g., "Action", "Comedy").
Rating: string	Movie rating (e.g., "PG", "PG-13").
Showtimes: List<DateTime>	List of available showtimes.
3.4.2.2 Operations	
AddMovie(title, description, duration, genre, rating): void	Adds a new movie to the system.
UpdateMovie(movieID, title, description, duration, genre, rating): void	Updates movie details.
DeleteMovie(movieID): void	Removes a movie from the system.
SearchMovie(keyword): List<Movie>	Searches for movies by title, genre, or keyword.
DisplayMovieDetails(movieID): Movie	Displays detailed information about a movie.

Reference to Use Cases:

Use Case #1: Select a Movie

3.4.3 <Ticket>

Ticket	
3.4.3.1 Attributes	
TicketID: String	A unique identifier for each ticket.
UserID: int	The ID of the user who purchased the ticket.
MovieID: int	The ID of the movie associated with the ticket.
Showtime: DateTime	The date and time of the movie screening.
SeatNumber: String	The seat number assigned to the ticket.
Price: double	The price of the ticket is converted to the user's local currency.
choseSeat: String	The seat is selected.
3.4.3.2 Operations	
BookTicket(userID, movieID, showtime, seatNumber, price): Ticket	A user can book a ticket for a specific movie and showtime.
CancelTicket(ticketID): void	This enables a user to cancel a booked ticket.
GenerateTicket(ticketID): string	Generates a digital ticket for the user.
CheckAvailability(movieID, showtime): int	Check seat availability for a specific showtime.
ChoseSeat(SeatNumber): void	This allows the user to select their seat and seat number.

Reference to Use Cases:

Use Case #2: Purchase a Movie Ticket

Use Case #3: Cancel Ticket

3.5 Non-Functional Requirements

3.5.1 Performance

3.5.1.1 95% of all user interactions will have a response time of under 20 seconds, assuming it is at normal load conditions.

3.5.1.2 The system will be able to handle 10,000 users on the website simultaneously without a downgrade in performance.

3.5.1.3 The system will process at least 500 transactions during peak hours.

3.5.1.4 The confirmation for seating shall not take more than 1 second to update and reflect for all other users.

3.5.1.5 During peak hours (e.g. the holidays, or the release of a popular movie) the system will be able to handle up to five times the regular transactions.

3.5.1.6 All critical database queries (e.g. searching movies, choosing available seats, booking status) shall execute within 300 milliseconds.

3.5.1.7 The system shall maintain user sessions for a minimum of 15 minutes of inactivity before requiring re-authentication.

3.5.1.8 All external API calls (e.g. payment gateway, third-party movie databases) shall possess a response time of less than 1 second.

3.5.2 Reliability

3.5.2.1 The system must ensure data consistency and accuracy.

3.5.2.2 The system must have an uptime of 99.9% to ensure it is available for users at all times, excluding scheduled maintenance.

3.5.2.3 Ticket purchases must be fully completed or canceled in case of failure.

3.5.2.4 The system shall be tested to identify and resolve potential issues before officialization.

3.5.2.5 The system must automatically back up data daily to prevent data loss in case of hardware or software failures.

3.5.3 Availability

3.5.3.1 The system shall be accessible 24/7 except for scheduled maintenance periods.

3.5.3.2 The system shall provide real-time status monitoring and alert administrators in case of failures.

3.5.3.3 The system shall automatically notify users of any scheduled downtime at least 24 hours in advance.

3.5.3.4 The systems shall support automatic scaling to handle increased traffic during peak hours without losing performance.

3.5.4 Security

3.5.4.1 The system shall encrypt all user data and transactions.

3.5.4.2 Multi-factor authentication shall be required for all administrative accounts.

3.5.4.3 The system shall ask for authentication if inactive for 15 minutes.

3.5.4.4 The system shall conduct monthly security audits to identify and fix vulnerabilities.

3.5.4.5 The system shall make each ticket unique.

3.5.5 Maintainability

3.5.5.1 Regular updates and maintenance schedules must be established to address bugs, security vulnerabilities, and performance improvements.

3.5.5.2 Comprehensive documentation must be provided, such as user manuals, maintenance procedures, and system architecture

3.5.5.3 Comprehensive code documentation, including inline comments and API documentation, must be provided to facilitate understanding and modification.

3.5.5.4 A testing framework must be implemented to ensure that changes or updates do not introduce new bugs or regressions.

3.5.5.5 The system must minimize dependencies on third-party services to reduce the risk of compatibility issues.

3.5.6 Portability

3.5.6.1 The system shall support access via web browsers (Google Chrome, Firefox, Safari).

3.5.6.2 The system shall work across all screen sizes.

3.5.6.3 The system shall be compatible with multiple operating systems, including Windows, iOS, MacOS, and Android.

3.5.6.4 The system shall allow integration with third-party applications, including calendar apps and digital wallets.

3.5.6.5 The system shall support multiple language preferences for accessibility.

3.6 Inverse Requirements

System maintenance shall not be performed during peak hours to minimize downtime.

The system shall not allow booking modifications or cancellations after the showtime has started.

The system shall not allow ticket purchases for past showtimes.

The system shall not allow ticket purchases of more than 10 tickets at a time.

The system shall not allow account login without additional authentication for security purposes.

The system shall not allow booking of the same seat.

The system shall not allow ticket purchases for full theaters.

3.7 Design Constraints

Specify design constraints imposed by other standards, company policies, hardware limitations, 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)

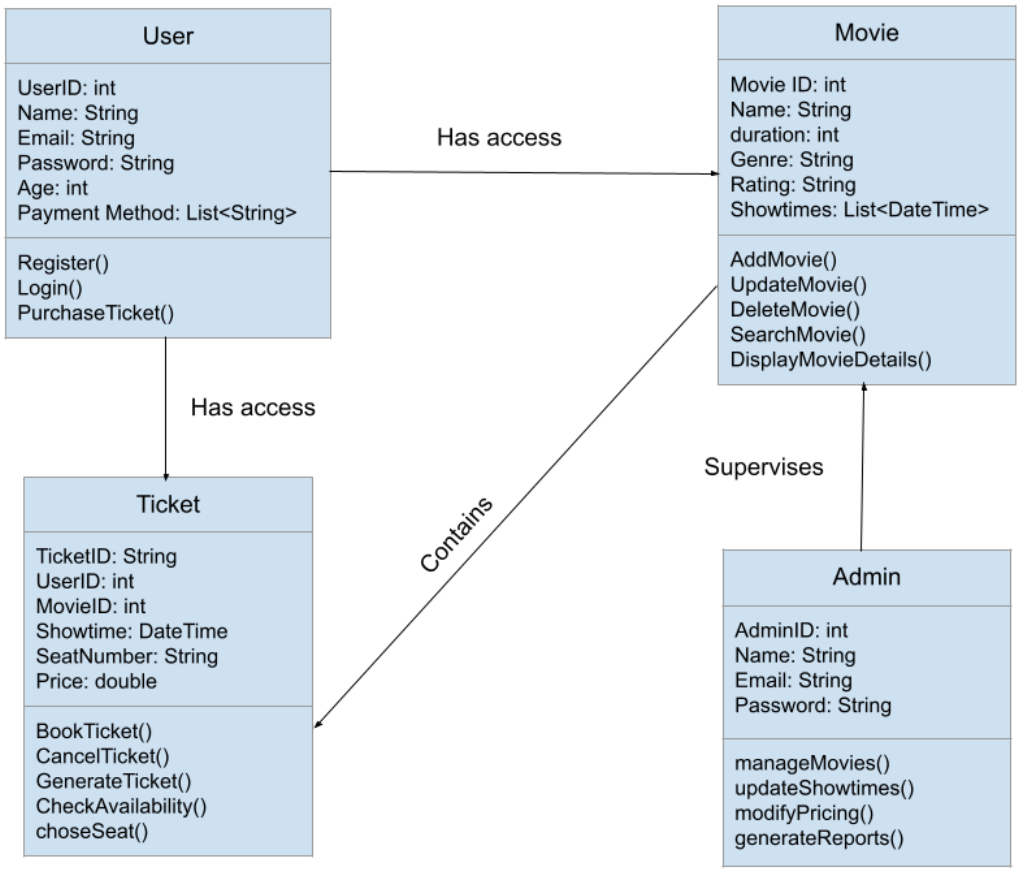
5. Change Management Process

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. Design Specification

6.1 UML

6.1.1 UML Class Diagram



6.1.2 UML Class Description

User

- UserID: int
    - A unique identifier for each user.
- Name: String
    - The full name of the user.
- Email: String
    - The user's email address, and is used for account creation and communication.
- Password: String
    - A secure password for account authentication.
- Age: int
    - The user's age, and is used to enforce the minimum age requirement (14+).
- PaymentMethods: (List<string>)
    - A list of payment methods associated with the user.
- Register(name, email, password, age): void
    - Allows a new user to create an account.
- Login(email, password): bool
    - Authenticates the user and grants access to the system.
- PurchaseTicket(movieID, showtime, seatNumber): Ticket
    - Purchases a movie ticket.

Movie

- MovieID: int
    - Unique identifier for the movie.
- Title: String
    - Title of the movie.
- Description: String
    - Brief synopsis of the movie.
- Duration: int
    - Runtime of the movie in minutes.
- Genre: String
    - Category of the movie (e.g., "Action", "Comedy").
- Rating: String
    - Movie rating (e.g., "PG", "PG-13").
- Showtimes: List<DateTime>
    - List of available showtimes.
- AddMovie(title, description, duration, genre, rating): void
    - Adds a new movie to the system.
- UpdateMovie(movieID, title, description, duration, genre, rating): void
    - Updates movie details.
- DeleteMovie(movieID): void
    - Removes a movie from the system.
- SearchMovie(keyword): List<Movies>
    - Searches for movies by title, genre, or keyword.
- DisplayMovieDetails(movieID): void

    - Movie Displays detailed information about a movie.

Ticket

- TicketID: String
    - A unique identifier for each ticket.
- UserID: int
    - The ID of the user who purchased the ticket.
- MovieID: int
    - The ID of the movie associated with the ticket.
- Showtime: DateTime
    - The date and time of the movie screening.
- SeatNumber: String
    - The seat number assigned to the ticket.
- Price: double
    - The price of the ticket is converted to the user's local currency.
- choseSeat: String
    - The seat is selected.
- BookTicket(userID, movieID, showtime, seatNumber, price): Ticket
    - A user can book a ticket for a specific movie and showtime.
- CancelTicket(ticketID): void
    - This enables a user to cancel a booked ticket.
- GenerateTicket(ticketID): string
    - Generates a digital ticket for the user.
- CheckAvailability(movieID, showtime): int
    - Check seat availability for a specific showtime.
- ChoseSeat(SeatNumber): void
    - This allows the user to select their seat and seat number.

Admin

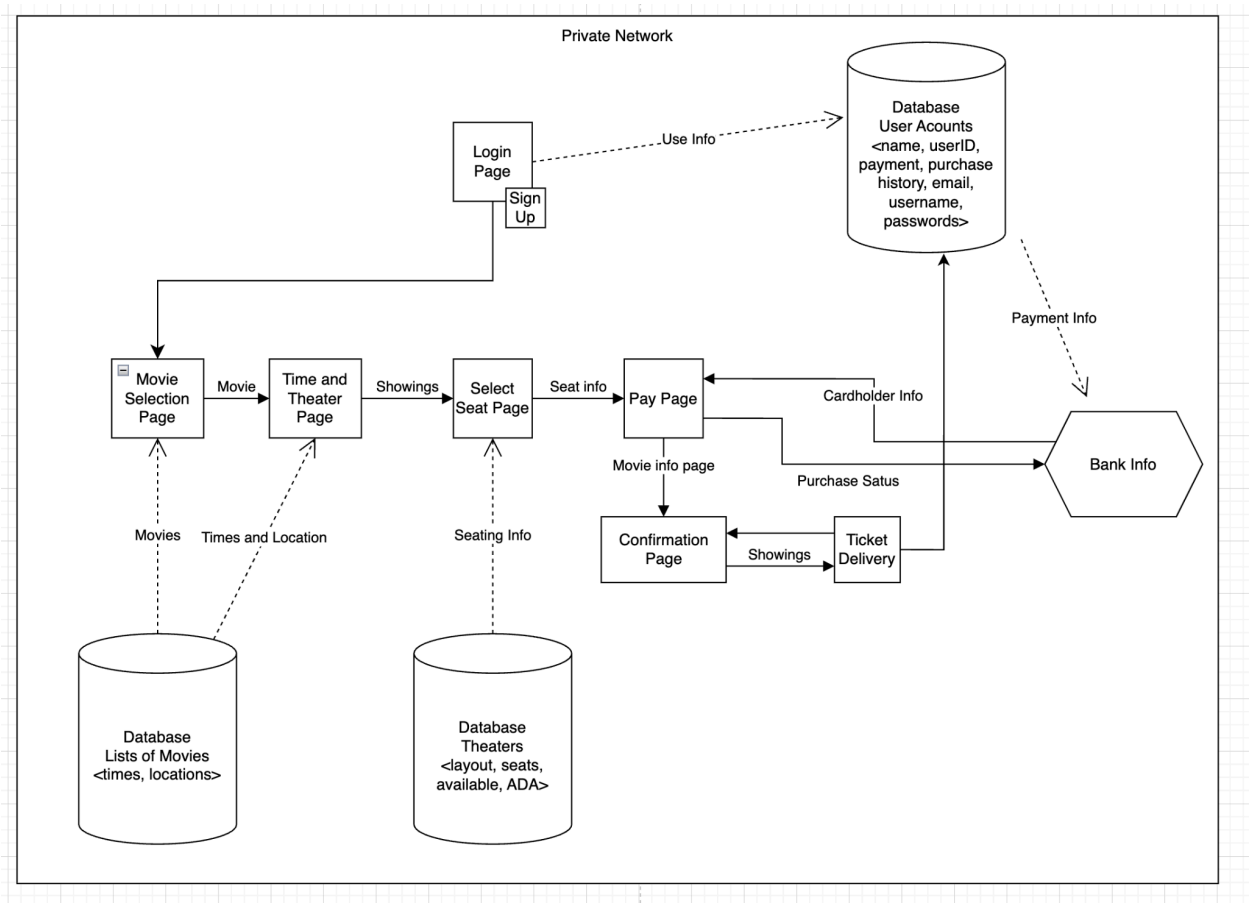
- AdminID: int
    - Unique identifier for administrators.
- Name: String
    - Name of the administrator.
- Email: String
    - Email for authentication.
- Password: int
    - A secure password for account authentication.
- manageMovies(): void
    - Adds, edits, and removes movie listings.
- updateShowtimes(): void
    - Modifies showtimes for movies.
- modifyPricing(): void

- Adjusts ticket pricing.

- generateReports(): List<Report>
  - Produces sales and user activity reports.

6.2 SWA

6.2.1 SWA Diagram



6.2.2 SWA Description

The user starts at the login page, where they access their account, either as a regular customer or an admin from the database. This allows them to begin the program. The data from the database takes them to the movie selection page with certain attributes that correlate to the user type. Then, once in, they are brought to the movie selection page. Here, the user can search for a movie or look through the genre list. After that, they are taken to the time and theater arrangement page. After selecting the time you want to watch the movie, you are taken to the seating arrangement page. Then, after selecting your seat, you are taken to the payment page. Once a payment type is selected and paid for, the system will access the bank account and take you to a confirmation page. From here, you would confirm your purchase, and the receipt of the payment gets sent to the user account database. Your ticket gets delivered via email and is also saved in the account database. With the email, you can then go in and enjoy your movie.

6.2.3 SWA Error Handling

Login:

- Input wrong email or password: When the user gets the email or password wrong, an error message will be displayed; the user can return to the login page to try again.
- Forget password: At the bottom of the login page, there is also a “Forgot Password” link. If the user forgets their password, they can click the “Forgot Password” link, and an email will be sent to them to change their password.
- Once every month, there is a verification email that is sent out, and the user simply must go to their email and click on the link to verify the account is still active and registered to the proper people.

Choosing Seats:

- The system updates seating every 30 seconds. A seat is considered registered once the user is at the payment page
- Two seats are selected at the same time: An error message will be displayed, and the user will be brought back to select a seat again

Payments:

- Card gets declined: While paying, there is about a 10-minute hold on the seat to be purchased. This notice pops up when the user first gets to the payment page. If the card gets declined, then the system will pop up saying “Unable to process payment”.

6.3 Development Plan and Timeline

6.3.1 Partitioning of Tasks

Task	Estimated Completion Date
System Architecture and Design	Week 1-2
Testing and Debugging	Week 2-3
Payment Integration	Week 3-4
Frontend Development and UI Design	Week 4-5
Database and Backend Development	Week 5-6
Documentation and Finalization	Week 6-7
User Authentication and Role Management	Week 7-8
Seat Selection and Booking System Development	Week 8-9
Notification and Confirmation System	Week 9-10

6.3.2 Team Member Responsibilities

	Member
Title Page	Steven To
System Overview	Charlie Pham
Architectural Diagram	Anthony Nguyen
UML Diagram	Steven To
Classes Description	Charlie Pham
Attributes Description	Anthony Nguyen
Operations Description	Steven To

7. Verification Test Plan

UNIT TEST

Test Set 1: login(): boolean - Method

Feature Targeted: The *login()* method is within the user class. This method takes in two parameters: the email address and password. This checks for a user's account and authenticates the user hence granting them access to the system.

Explanation: This test validates the correctness of the *login()* method by checking for accurate inputs for the email and password. This method is split into two partitions, When the correct credentials are inserted, and when the wrong credentials are entered.

1. The correct credentials
2. The incorrect credentials

Test Cases:

- TC001 Correct Credentials
    - Preconditions: The user has an active account
    - Expected Results: The user should be redirected to the dashboard

    - Actual Results: The user is redirected to the dashboard
- TC002 Incorrect Credentials
    - Preconditions: The user account exists
    - Expected Results: An error message should be displayed
    - Actual Results: Error message displayed

Status: Pass

Test Set 2: Register(String, String, String, int) - Method

Feature Targeted: The register(String, String, String, int) method derives from the user class. This method depends on the input from the user to check if there is a preexisting account.

Explanation: While registering, this method asks for a name, email, password, and age. The test cases present are to check whether there is already an account linked to the email address that is being used to register.

Test Cases:

- TC007 Successful Registration
    - Preconditions: The user does not have an account
    - Expected Results: The account was created successfully
    - Actual Results: The account was created successfully
- TC008 Unsuccessful Registration
    - Preconditions: The user already has an account with the email used
    - Expected Results: The account already exists
    - Actual Results: The account already exists

Status: Pass

Test Set 3: SearchMovie(String): Method

Feature Targeted: The SearchMovie(String) method allows the user to input a movie title, genre, or keyword, and the system goes into the Movie database and displays movie's the are similar to the inputs from the user.

Explanation: It takes in the movie chosen as its parameter and goes into said movie's database. Each movie has a genre attached to it, so would be displayed if the user inputs a genre that correlates with the movie. If a movie title in taken, it checks to see if the movie exists and is in the database and displays the movie relating to the search.

Test Cases:

- TC009 Validate Search Functionality
    - Preconditions: Movies are available in the database
    - Expected Results: Matching movies should be displayed
    - Actual Results: Matching movies should be displayed

Status: Pass

INTEGRATION TEST

Test Set 1: choseSeat(String): String - Method

Feature Targeted: The *choseSeat(String)* method is in the ticket class. This allows the user to select their seat, and check if the seat they want is available.

Explanation: It takes in the movie chosen as its parameters and goes into said movie's database, then allows the user to select a seat. It also uses a boolean response as the user is selecting the seat. If the seat is available, then it will allow the user to add it to their ticket, if not, there will be an error message and return to the seat selection page.

Test Cases:

- TC003 Available Seat
    - Preconditions: The movie has available seats
    - Expected Results: Seat should be booked successfully
    - Actual Results: Seat is booked successfully
- TC004 Unavailable Seat
    - Preconditions: Another user already booked the same seat
    - Expected Results: An error message should be displayed
    - Actual Results: Error message displayed

Status: Pass

Test Set 2: PaymentMethods: (List<String>) - Method

Feature Targeted: The *PaymentMethods: (List<String>)* method in the user class. This allows the user to add and save payment methods that are associated with their account.

Explanation: It takes in the payment method as its parameter and is encrypted to ensure security. It also makes sure that the payment method is valid and if it is invalid, then the payment fails.

Test Cases:

- TC005 Successful Payment
    - Preconditions: The user has valid payment details
    - Expected Results: Payment successful, receipt should be generated
    - Actual Results: Payment successful, receipt generated
- TC006 Failed Payment
    - Preconditions: The user enters an expired card or not enough funds
    - Expected Results: Payment should be declined
    - Actual Results: Payment declined

Status: Pass

SYSTEM TEST

Test Set 1: Logging Out

Feature Targeted: Logging Out Functionality

Explanation: The user will be given the option to sign/log out of their account.

Test Cases:

- TC010 Logging Out
    - Preconditions: The user is logged in
    - Expected Results: The user is logged out, and redirected to the home page
    - Actual Results: The user is logged out, and redirected to the home page

Status: Pass

Test Set 2: Email Confirmation

Feature Targeted: Email Confirmation Functionality

Explanation: After ticket(s) is successfully booked, an email confirmation with movie details, seat number(s), etc. will be sent to the email associated with the account used to book the ticket.

Test Cases:

- TC011 Email Confirmation After Booking
    - Preconditions: The user has a valid email
    - Expected Results: A confirmation email is sent
    - Actual Results: A confirmation email is sent

Status: Pass

8. Data Management Strategy

The theater ticketing system is built on a centralized, secure, and scalable SQL-based database system (MySQL).

- It enables fast and secure queries for both customer and admin operations.
- It has a strong transactional integrity; payments are wrapped in atomic transactions to prevent over booking or payment failures.
- It integrates smoothly with backend technologies like Node.js.
- It enforces ACID (Atomicity, Consistency, Isolation, Durability) compliance.

8.1 Database Schema

The database schema is split across six key tables:

- Users: stores account information, credentials, age, and payment methods and places them in a profile.

- Admin: a separate profile from users that can manage authentication and operations.
- Movies: stores metadata such as title, genre, rating, showtimes, duration, and description.
- Showtimes: includes available screening times and corresponding movie IDs.
- Tickets: links users to showtimes and stores seat assignments and prices.
- Payments: handles transaction details including timestamps, user Id, and masked card data.

Sensitive fields such as passwords and card tokens are stored using hashing and encryption practices.

8.2 Alternatives

We evaluated non-SQL options like MongoDB, which offer:

- Schema flexibility: quicker iteration for early development.
- Horizontal scaling: easier to scale when it comes to heavy concurrent users.
- JSON: used like a document storage

However we decided against using non-SQL because:

- Our data is highly relational and structured (seat linking, user bookings, showtimes).
- Due to the possibly of careless design, non-SQL is prone to data duplication and inconsistency.
- ACID compliance in non-SQL systems often requires extra configurations.

8.3 Tradeoffs and Justifications

Options	Pros	Cons
SQL (MySQL)	- Strong data integrity - Ideal for structured data - Slower due to strict	- More rigid schema - Harder to scale horizontally

	schema	
Non-SQL (MongoDB)	- Scalable - Flexible schema - Better for rapid iteration - Faster iteration with flexible models	- Weak relational support - Eventual consistency

Ultimately, MySQL was chosen for its transactional strength, data integrity, and ease of integrating with existing tech stacks. For future scalability, the system can be adapted into a hybrid architecture using non-SQL for analytics, while retaining SQL for transactional data.

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