# `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.

# <Movie Theater>

# Software Requirements Specification

<Version 1>

<Date: February 2025>

<Group #12>

<Isaac Blanco and Lachlan Carlson>

Prepared for
CS 250- Introduction to Software Systems
Instructor: Dr. Gus Hanna
Fall 2024

## Revision History

| Date | Description | Author | Comments |
|---|---|---|---|
| <date> | <Version 1> | <Your Name> | <First Revision> |
| 3/6/2025 | Version 2 | Isaac and Lachlan | added the software design specification |
| | | | |
| | | | |

## 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) document should provide an overview of the complete SRS document. While writing this document please remember that this document should contain all of the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document. (Note: the following subsection annotates are largely taken from the IEEE Guide to SRS).*

## 1.1 Purpose

The purpose of this Movie Theater application is to give users the ability to book tickets at their movie theaters. This application will let users reserve and pay for their tickets and snacks in advance. The application will be able to make transactions through any banking services so that no matter the user's banker they will be able to buy tickets.

## 1.2 Scope

*This subsection should:*

*(1)* There will be a Ticket database, snacks and drinks database, and a users database. Report Generators for both tickets and snacks and drinks.

*(2)* The ticket database will keep track of how many tickets have been sold and how many are left. The snack and drink database will do the same keeping track of the inventory of goods. The user database will keep track of users with memberships.The report generator for tickets will let the movie theater know what movies and what genre of movies sell best so that they can do more show times for those movies. The snack and drink report generator will do the same and show which drinks and snacks people prefer when viewing movies to always be freshly stocked on the popular items.

*(3)* The benefits of using a database to keep track of tickets sold allows us to know how many are left for other users to obtain. The same with snacks and drinks, keeping track of what is sold is important for receipts as well in case problems occur with a purchase it will be able to be verified in the database and conflicts can be resolved. The importance of a user database is to keep track of users with memberships so that they can be awarded points based on their purchases. The importance of these report generators is to increase the amount of profit the movie theater can make. It is important to not waste money on unpopular snacks and drinks. It is also important to to not waste resources such as movie theater rooms on movies not many people are interested in seeing.

## 1.3 Definitions, Acronyms, and Abbreviations

- SRS- Software Requirements Specification, the document that outlines all the functional and nonfunctional requirements of a software application
- POS- Point of Scale, a system that is used by the theater in order to process ticket and food purchases
- UI- User Interface, the visual representation of the application that the users will interact with
- DBMS- Database Management System, the software system that manages the application’s database like the tickets, snacks and users.

- API- Application Programming Interface, Functions that let different software components interact with one another
- 2FA- Two-Factor Authentication
- Refusal Policy- the conditions in which a user can get a refund
- Concurrency Limit- The max amount of users the system can handle without running into performance problems

## 1.4 References

IEEE 830-1998 – IEEE Recommended Practice for Software Requirements Specifications

- Description: A standard providing guidelines for writing software requirements specifications (SRS).
- Source: Institute of Electrical and Electronics Engineers (IEEE)
- URL: https://standards.ieee.org/

PCI-DSS (Payment Card Industry Data Security Standard)

- Description: A security standard for handling credit card transactions and protecting cardholder data.
- Source: PCI Security Standards Council
- URL: https://www.pcisecuritystandards.org/

OWASP Top Ten Security Risks

- Description: A list of the most critical security risks for web applications.
- Source: Open Web Application Security Project (OWASP)
- URL: https://owasp.org/www-project-top-ten/

MySQL 8.0 Reference Manual

- Description: Official documentation for MySQL database management system, covering queries, performance, and security.
- Source: Oracle Corporation
- URL: https://dev.mysql.com/doc/refman/8.0/en/

React Official Documentation

- Description: The official documentation for the React JavaScript library, used for building the front-end interface.
- Source: Meta (formerly Facebook)
- URL: https://react.dev/

WCAG 2.1 – Web Content Accessibility Guidelines

- Description: Accessibility guidelines ensuring the system is usable by individuals with disabilities.
- Source: World Wide Web Consortium (W3C)
- URL: https://www.w3.org/TR/WCAG21/

Local Tax Regulations for Online Sales

- Description: Legal requirements regarding taxation for online ticket and snack purchases.
- Source: Local government tax authority
- URL: (To be determined based on theater location)

### 1.5 Overview

The SRS is split into 5 sections

- Section 1: Introduction
  - this describes the purpose, scope, definitions and references of the entire document
- Section 2: General Description
  - This provides a more in depth overview of the movie theater application. This includes its functions, user characteristics and system constraints
- Section 3: Specific Requirements
  - This defines the functional and nonfunctional elements that are required of the application. This also includes the interface requirements and software constraints. The use cases are also involved in this section
- Section 4: Analysis Models
  - This section lists the analysis models used in developing the application. These analysis models should also be traceable in the SRS
- Section 5: Change Management Process
  - This is where the process used to update the SRS is identified and described. This is also where it is defined who can send change requests and how these changes will be handled

## 2. General Description

*This section of the SRS should describe the general factors that affect 'the product and its requirements. It should be made clear that this section does not state specific requirements; it only makes those requirements easier to understand.*

### 2.1 Product Perspective

The movie theatre ticketing system will work with the theatre's existing systems, including the point-of-sale system and payment processing. Customers can buy tickets through a website or mobile app, and it'll sync with in-person ticket sales. The system will also connect with the theatre's rewards program.

## 2.2 Product Functions

The system will:

- Let customers browse movies and showtimes
- Show an interactive seating chart
- Process online payments
- Send tickets by email and phone
- Keep track of available seats
- Handle customer rewards points
- Let staff process refunds
- Create sales reports

## 2.3 User Characteristics

There are three main types of users:

1. Customers: People of all ages buying movie tickets
2. Theatre Staff: Employees who sell tickets and help customers
3. Managers: People who need to check reports and change settings

## 2.4 General Constraints

The system has to:

- Be secure for processing payments
- Stay online during theatre hours
- Handle lots of users during busy times
- Work with the theatre's current equipment
- Follow accessibility rules
- Handle taxes correctly

## 2.5 Assumptions and Dependencies

For the system to work, we assume:

- The theatre has good internet
- Payment processing services work
- Most customers have newer phones
- The theatre database can update in real-time
- Staff computers can run the system
- Movie schedules are provided on time

# 3. Specific Requirements

*This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project's software design, implementation, and testing.*

*Each requirement in this section should be:*

- *Correct*
- *Traceable (both forward and backward to prior/future artifacts)*
- *Unambiguous*
- *Verifiable (i.e., testable)*
- *Prioritized (with respect to importance and/or stability)*
- *Complete*
- *Consistent*
- *Uniquely identifiable (usually via numbering like 3.4.5.6)*

*Attention should be paid to the carefuly organize the requirements presented in this section so that they may easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.*

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

- Web Interface
    - A website that allows a user to browse movies, find available seats, get tickets and snacks, and manage their accounts
- Mobile Application Interface
    - A copy of the website that is in a mobile app, can be used by IOS and Android alike
- Admin Dashboard Interface
    - A web based interface for the movie theater staff to use in order to monitor ticket and snack sales, manage a users account, create reports and process refunds
- POS Interface
    - an interface that allows the staff to sell tickets, check reservation times and scan the tickets (physical and digital)

### 3.1.2 Hardware Interfaces

- Ticket Printers
    - Where the physical tickets can be printed after purchasing
- Ticket Scanners
    - What is needed in order to confirm the user has a valid ticket (works with physical and digital tickets)

### 3.1.3 Software Interfaces

- Payment Gateways
    - This will process payments made with providers like PayPale and Zelle for secure online transactions

- Banking API
    - This will process payments made from different banking institutions
- DBMS
    - The database for storing tickets, users, snacks and transaction data
- Theater Management System
    - The system used to schedule movies, see available seats and their pricing
- Email and SMS services
    - A service that will email or text the tickets to the user, can also be used for confirmations and notifications

#### 3.1.4 Communications Interfaces

- HTTPS Protocol
    - Ensures that data will be transferred between users and the server securely
- OAuth 2.0 Authentication
    - The server used to authenticate through Google, facebook or email
- JSON/XML Data Exchange
    - This is used for exchanging data between the system and external services

## 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 <User Registration and Authentication>

3.2.1.1 Introduction

- Users are able to create an account, log in if there is an existing account and manage their accounts

3.2.1.2 Inputs

- They will input their email and passwords for login

3.2.1.3 Processing

- The user's credentials and encryption keys need to be processed to ensure security and authenticity

3.2.1.4 Outputs

- Confirmations email if registering for the first time
- Email for password reset if password is lost
- Failure messages if user’s credentials are invalid

3.2.1.5 Error Handling

- Display error messages if credentials are invalid
- Prevention of multiple accounts with the same email address or username

#### 3.2.2 <Ticket Booking and Payment Processing>

3.2.2.1 Introduction

- The user will be able to select seats and purchase tickets for the desired seats
- Will also process payments

3.2.2.2 Inputs

- Movie name, showtime and seat

- payment details (credit/debit card, alt payment like paypal, reward points)

3.2.2.3 Processing

- The users seat availability needs to be process
- The payment needs to be processed
- The tickets need to be generated

3.2.2.4 Outputs

- Email of the tickets and a receipt

3.2.2.5 Error Handling

- Displays an error if payment fails
- Displays an error message if seats are invalid for purchase

## 3.3 Use Cases

### 3.3.1 User Login

- Actors
    - Customer
    - Staff
- Preconditions
    - User must have an active account
- Flow of events
    - The user (customer or staff) logs in with email and password (or through google or facebook)
    - System verifies the credentials are valid
    - If valid user is logged in
- Exceptions
    - If credentials are not valid an error message will be displayed

### 3.3.2 Ticket Booking and Payment

- Actors
    - Customer
- Preconditions
    - The user must be logged in
- Flow of events
    - The user finds a movie they like, the show time and seats
    - The system verifies that the seats for the given time are available
    - The user checks out and selects a payment method
    - The system is able to process the users payment securely
    - The transaction goes through and the user is sent their tickets either through email or sms
- Exceptions
    - If the seats are unavailable the user will not be able to proceed to check out
    - If the payment fails the user will be notified

## 3.4 Classes / Objects

### 3.4.1 <Ticket>

3.4.1.1 Attributes

- ticketID (string)
- showTime (datetime)
- seatNumber (string)
- price (decimal)
- customerID (string)
- status (enum: valid, used, refunded)

3.4.1.2 Functions

- generateTicket()
- validateTicket()
- refundTicket()
- sendEmailConfirmation()

### 3.4.2 <Customer>

3.4.1.1 Attributes

- customerID (string)
- name (string)
- email (string)
- rewardPoints (integer)
- purchaseHistory (list)

3.4.1.2 Functions

- register()
- login()
- updateProfile()
- addRewardPoints()

## 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 transaction 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

- Ticket booking process must complete within 3 seconds
- System must handle 500 concurrent users
- Page load time should be under 2 seconds

### 3.5.2 Reliability

- System should have less than 1 error per 1000 transactions
- Database backups must occur daily
- Error messages should be clear and helpful

### 3.5.3 Availability

- System uptime should be 99% during theatre hours
- Maintenance windows only between 2 AM and 5 AM
- Max system recovery time of 10 minutes

### 3.5.4 Security

- All passwords must be hashed
- Payment info can't be stored
- Session timeout after 30 minutes
- Must use HTTPS

### 3.5.5 Maintainability

- Code must have comments
- Use common coding standards
- Keep documentation updated
- Regular system updates

### 3.5.6 Portability

- Works on Chrome, Firefox, Safari
- Mobile-friendly design
- Supports iOS and Android

## 3.6 Inverse Requirements

- System won't process expired credit cards
- Can't book seats that are already taken
- Won't allow ticket sales after showtime
- No refunds after movie starts

### 3.7 Design Constraints

- Must use MySQL database
- Built with React framework
- Maximum file upload size of 10MB
- Must work with theatre's printer system
- Limited to 1GB RAM per instance

### 3.8 Logical Database Requirements

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

### 3.9 Other Requirements

*Catchall section for any additional requirements.*

## 4. Analysis Models

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

### 4.1 Sequence Diagrams

### 4.3 Data Flow Diagrams (DFD)

### 4.2 State-Transition Diagrams (STD)

## 5. Change Management Process

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

## 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

# 5. Software Design Specifications

## System Description

The Movie Theatre Ticketing System handles everything related to selling movie tickets. Customers can use it to find movies, pick showtimes, choose seats, and buy tickets online or at the theatre. Theatre employees can use the system to manage shows and help customers with problems. Managers can also use it to see sales reports and change system settings.

The system connects with other stuff the theatre already uses like their cash registers, payment systems, and rewards program. It keeps track of seats in real-time so people don't accidentally book the same seat through different channels.

## Software Architecture Overview

## Architectural diagram of all major components

Client Layer:

- Web browser interface
- Mobile app
- Kiosk at theatre

Application Layer:

- Ticket management service
- Customer account service
- Movie & showtime management

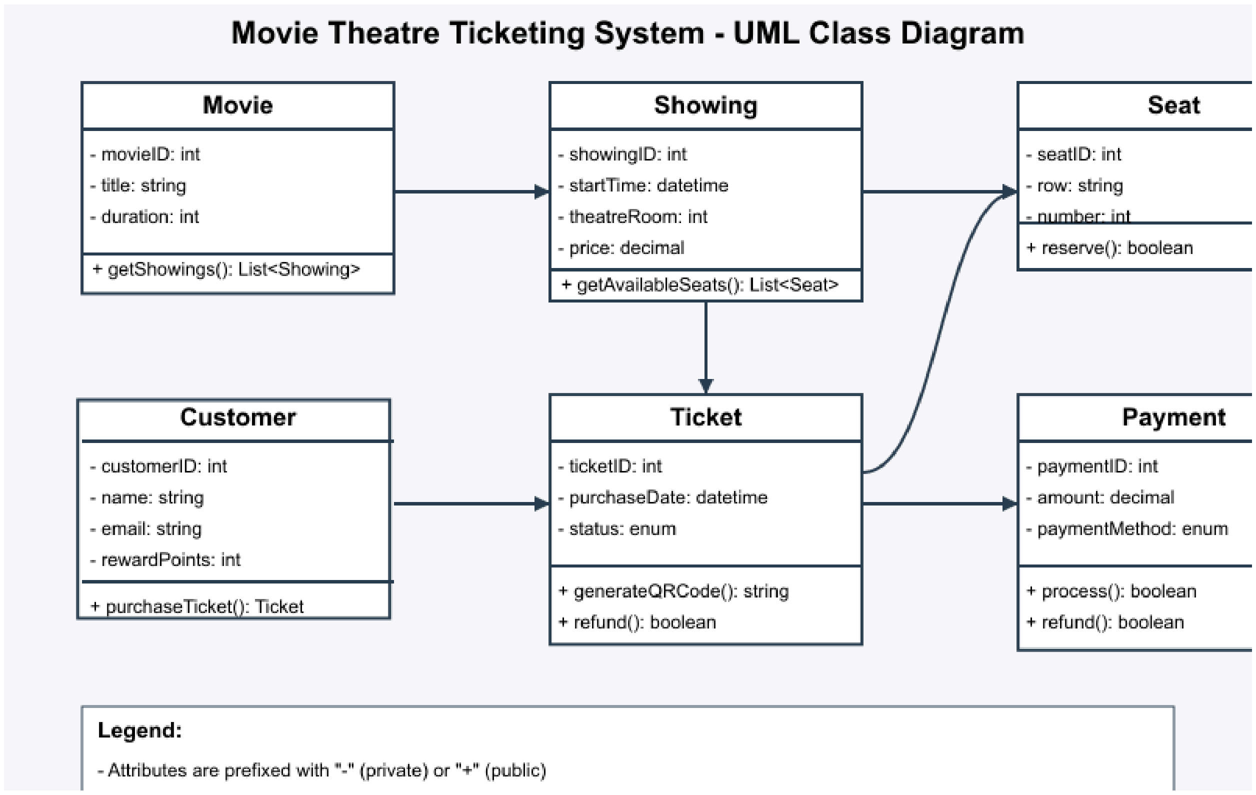
Database Layer:

- Ticket database
- Customer database
- Movie & showtime database

External Systems:

- Payment processing
- Email service
- Theatre POS system

**UML Class Diagram**



## Description of classes

Movie

- Attributes: movieID, title, duration, rating, description, imageURL
- Functions: getMovieInfo(), isCurrentlyShowing(), getAvailableShowings()
- Stores information about each movie showing at the theatre

Showing

- Attributes: showingID, movieID, theatreID, startTime, endTime, price
- Functions: getAvailableSeats(), calculateOccupancy(), cancelShowing()
- Represents a specific screening of a movie at a particular time and theatre

Seat

- Attributes: seatID, row, seatNumber, showingID, status
- Functions: reserve(), release(), markSold()
- Represents individual seats that can be booked for a showing

Customer

- Attributes: customerID, name, email, phone, rewardPoints
- Functions: register(), login(), updateProfile(), getPurchaseHistory()
- Stores information about users who purchase tickets

Ticket

- Attributes: ticketID, showingID, seatID, customerID, price, status
- Functions: generateTicket(), sendConfirmation(), validateTicket(), refundTicket()
- Represents a purchased admission to a specific showing for a specific seat

Payment

- Attributes: paymentID, amount, paymentMethod, status, timestamp, ticketIDs
- Functions: processPayment(), verifyTransaction(), issueRefund()
- Handles financial transactions related to ticket purchases

## Description of Attributes

Movie

- movieID: this will be a unique integer that will represent this specific movie
- Title: This will be a string containing the title of the movie
- Duration: This will be a integer with the length of the movie in minutes rounded up
- Rating: This will be a string that will show what rating the movie is. Ex G or PG-13
- Description: This will be a string with a synapse of the movie for the user to read
- Image URL: this will be a sting that will contain the link to the movie poster so the user can have a visual for the movie they want to see

Showing

- showingID: this will be a unique integer for the specific showing of the movie
- movieID: this will be an integer that is grabbed from the movie class that corresponds to a specific movie
- theaterID: this will be a unique integer that represents the theater the movie will be shown in
- startTime: this will be a dateTime that represents the date and time the movie starts
- endTIme: this will be a dateTime that represents the date and time the movie ends
- Price: this will be a decimal that represents the price of the ticket for this specific movie and showing

Seat

- seatID: this will be a unique integer that represents the specific seat for a specific movie and show time
- Row: this will be a string that represents the letter of the row of the seat
- seatNumber: this will be an integer that represents the number of the seat
- showingID: this will be the integer pulled from the showing class
- Status: this will be a boolean that represents if the seat is free or reserved

Customer

- customerID: this will be a unique integer that represents a specific customer
- Name: this will be a string that contains the users first and last name
- Email: this will be a string that contains the users email address
- Phone: this will be an integer that contains the users phone number
- rewardPoints: this will be an integer that represents how many reward points the user has accumulated

Ticket

- ticketID: this will be a unique integer that represents a specific ticket
- showingID: this will be the integer pulled form the showing class
- seatID: this will be the integer pulled from the seat class
- customerID: this will be the integer pulled from the customer class
- Price: this will be the decimal pulled from the showing class
- Status: this will be an enum that represents the state of the ticket. Ex, purchased, reserved, refunded

Payment

- paymentID: this will a unique integer that represents a specific payment
- Amount: this will be a decimal that represents the total cost of the ticket(s)
- paymentMethod: this will be an enum that represents what the user used to pay for the ticket. Ex, credit, debit, apple pay, reward points.
- Status: this will be an enum that represents the status of the transaction. Ex, accepted, denied, in progress/
- Timestamp: this will be a dateTime that represents the date and time the transaction went through
- ticketIDs: this will be a string arrayList that contains all the tickets in this transaction

## Description of Operations

Movie

- getMovieInfo(): this operation will get and return the values of the title, duration, rating and description attributes
- isCurrentlyShowing(): this operation will return a boolean value if there are currently showtime available for a specific movie
- getAvailableShowings(): this operation will return the showing of the specified movie if it is currently showing

Showing

- getAvailableSeats(): this operation will return all available seats for a specified showtime of a movie
- calculateOccupancy(): this operation will calculate the total number of seats taken to show how many seats are available
- cancelShowing(): this operation will get rid of a specific showtime of a specific movie

Seat

- reserve(): this will set the status attribute of the seat to false representing the seat is unavailable
- release(): this will set the status attribute of the seat to true representing the seat is available
- markSold(): this will show on the admin side that a ticket has been sold and is not just in a queue waiting to be purchased, will also set the status attribute of the seat to false

Customer

- register(): this will create a new user in the database with their own customerID and all of their information
- login(): this will allow the user to log in to their created account and will authorize the user
- updateProfile(): this is where the user will go in order to update any information in their profile like their phone number or email address
- getPurchaseHistory(): this operator will return an arraylist of all the user's purchases utilizing the paymentID attribute from the payment class

Ticket

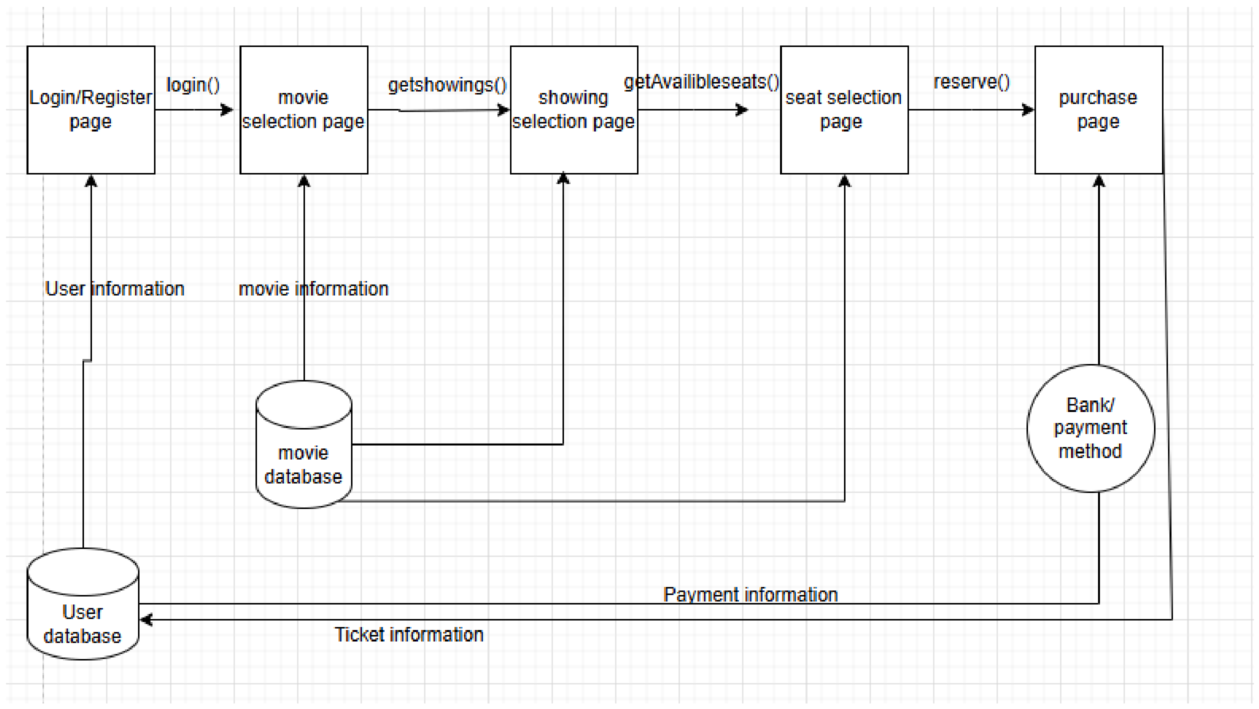
- generateTicket(): this operation will create a Ticket object that proves a user bought a ticket

- sendConfirmation(): this operation will send a message to the user through their email or text by using the corresponding attributes. The message will give the user all the information about the ticket they purchased like a receipt and will show the date and time of the movie
- validateTicket(): this operation will be used on the admin side to make sure that a ticket is real and corresponds to an existing ticketID to verify the user actually bought a ticket for the desired showtime and movie
- refundTicket(): this operation will give the user back the money from the ticket transaction. This operation will also initiate the release() operation in the seat class and will delete the ticketID that was originally created for the initial purchase.

Payment

- processPayment(): this operation will initiate the payment process and will use the payment method specified in order to try to complete the transaction.
- verifyTransaction() : this operation will check to see if the payment was successful, this will display a message to the user saying if their transaction was successful or not
- issueRefund(): this operation will be called when the user uses the refundTicket operation and this is where the user will be compensated for depending on their payment method. Ex if points were used they will get their points back, if a debit card was used their money will be transferred back into their account

## SWA Diagram



## SWA Description

This diagram roughly illustrates how the operations mentioned above will interact with one another and how it will work going from page to page. The idea is that the user starts off by registering for an account if they do not already have one and logging in from there or logging in if they already have an account. From there the user will be able to see the movies that are available for viewing. Once they find a movie they like they will be able to select a showtime of their choice. From there they will be able to pick from available seats that they like. Once that has been done they will be prompted to pay for those seats and will get the users payment information from the user database and will then send the confirmation of the tickets back to the userdatabase, this is also so the user will be able to look at their purchase history. The databases are important because we need a place for the userID to be stored and this is also how we will verify the user actually exists. The movie database will store all of the movie's information including the showtimes and seat availability; this is why it will be needed at the movie selection, showing and seat selection pages.

## Development Plan and Timeline

Partitioning of tasks

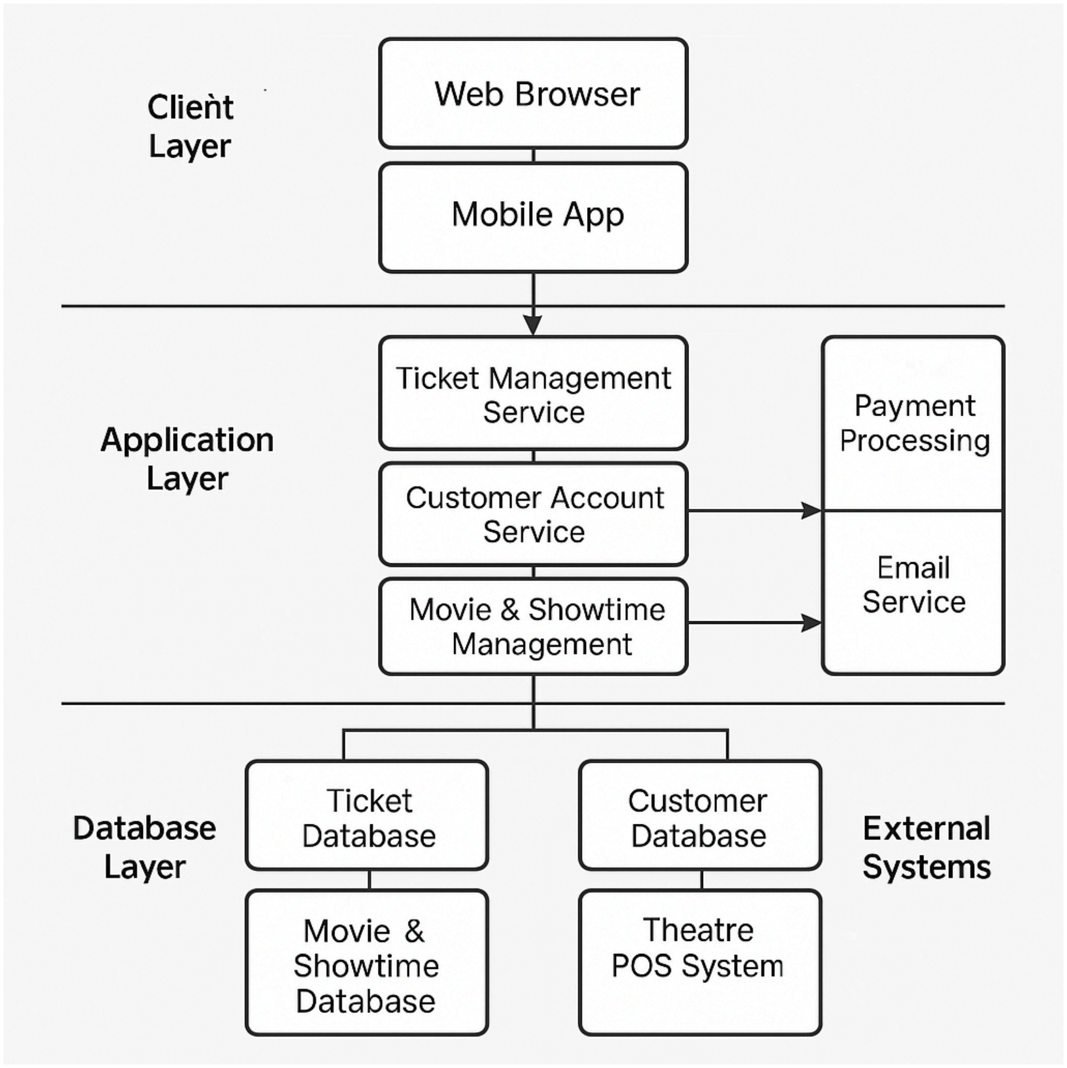
The work shall be split up evenly between the two of us. We will each work on creating one class at a time with all of their attributes and operations. We will work together in order to make sure all of our changes can merge together properly and will not break the entire system. We will also create the databases as we go along and will make sure all the data is being stored properly and that we can retrieve the desired information from each of the databases.

Team members responsibility

Isaac will be responsible for most of the front end development. Making the UI look good for users and admins. Isaac will also be in charge of establishing the connection between the front end and the back end and will help Lachlan with back end development when needed

Lachlan will be responsible for most of the back end development. Making sure all the operations work and have the desired outcome. Lachlan will also be responsible for creating the database and creating the connection to the database so that all the information is stored correctly. He will also assist Isaac with the front end when needed.

## Software Architecture Diagram



## Data Management Strategy

Our system will use a relational database management system (RDBMS). The specific database we will be using is MySQL, this will store and manage all the data related to movies, showings, tickets, users, and payments. The reason why we decided to choose SQL over any other database is because of how the data is structured and related to one another. SQL has strong data integrity and the ability to perform complex joins really efficiently.
We would split out data into three core databases all sharting a common instance but separated for better maintainability. The first database would be the movie management database, this would include tables for movies, showing, theaters, and seats. The purpose of this is to store all the information related to the movies. The second database would be for the customer account. This would consist of tables for the customers, accounts and their reward points. The purpose of this database is to manage the user credentials, contact information, preferences and their loyalty points. The third and last database would be the transaction database. This would consist of

tables for tickets, payments and refunds. The purpose of this is to handle the ticketing and payment processes and to log the transactions.
Our database has been normalized to Third Normal Form, this is to eliminate redundancy and to maintain the integrity of the data. This means that each table would contain the data directly related to its primary key. This would reduce duplication and make the updates a lot more efficient. We would also use referential integrity across the whole system using foreign key constraints. This will ensure consistent relationships between the tables. For example, if we were linking tickets to customers, or seats to showings, this would maintain accuracy and prevent orphaned records in the database. We will also implement a data retention policy to manage long term data efficiently. This will archive transactions and ticket data for 90 days. This will help minimize the load on the database while still letting there be access to the data for analytics or reporting purposes. Our backup strategy will be to do full backups daily and to take snapshots every two hours. This will make sure that the data can be recovered with the least amount of data lost in the event of an error or disruption. Lastly for security all user data (passwords in particular) will all be hashed using bcrypt. We will not store any raw payment information but will use tokenized references from a third party payment processor. This will help ensure our user's data stays as safe as possible.
The possible alternatives we could've used would have been the NoSQL approach. We could've used things like MongoDB or Firebase. These databases are also pretty flexible in their schema design and can be scaled with ease. However the reason why we did not go with these alternatives is because our system relies on the consistent relationship between structured data elements which is why we went with SQL. We could've also used a single combined database. This could have simplified the development but would've made the system harder to maintain and scale in the long run. The last alternative we could've used would have been a cloud native relational database like AWS Aurora or Google cloud SQL. These databases have really good reliability and scalability. The reason why these were not chosen is because of the current scope of our project and the budget constraints so MySQL was the best choice based on our scope and budget.
The trade offs are pretty positive. SQL has a pretty strong consistency while NoSQL does not have a strong consistency however NoSQL does have faster write operations. SQL database has enforced relational integrity while NoSQL offers a more flexible model. SQL supports complex and structured queries with join operation which is really good for relational data while NoSQL is more efficient at handling large volumes of unstructured data. The last trade off is that SQL is good for transaction heavy applications and is really reliable while NoSQL is better at data models that change frequently.