# <Movie Theater Ticketing System>

# Software Requirements Specification

<Version>

<9/18/2025>

<Group 15>

<Christian De Leon, Jacob Tabalon, Max Caguioa>

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2025

## Revision History

| Date | Description | Author | Comments |
|---|---|---|---|
| <09/18/25> | <Version 1> | <Group 15> | <SRS Requirements 1.0> |
| <10/09/25> | <Version 2> | <Group 15> | <SRS Requirements 2.0> |
| <10/23/25> | <Version 3> | <Group 15> | <SRS Requirements 3.0> |
| <11/6/25 > | <Version 4> | <Group 15> | <SRS Requirements 4.0> |

## Document Approval

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

| Signature | Printed Name | Title | Date |
|---|---|---|---|
| | | | |
| | Dr. Gus Hanna | Instructor, CS 250 | |
| | | | |

# Table of Contents

# 1. Introduction

The introduction of the Software Requirements Specification (SRS) details the requirements for the Movie Theatre Ticketing System(further referred to as MTTS). The MMTS is a web-based application which aims to provide customers a platform to purchase tickets. The application should also grant administrators to manage showtimes, pricing, and discounts, as well as be able to receive customer input and inquiries.

The aim of this document is to thoroughly detail the functions and requirements necessary for the MTTS to be usable, efficient, and reliable for customers, administrators, employees, and all that fall under the Movie Theatre business.

## 1.1 Purpose

The purpose of this SRS is to create a ticketing system for a movie theatre. The audience for which it is written is the general public interested in attending movie theatre showings.

1.2 Scope

The software for Theater Ticketing, will provide a way for customers to purchase tickets for a chain of 20 theaters within San Diego.

One goal is to provide a more convenient experience for buying tickets as they'll be available online, or on a kiosk, essentially eradicating any lines there may be at the theatres while also preventing movie tickets to be botted online ensuring the smoothest possible experience for a regular customer buying tickets for movies.

Another is to provide a more accurate and efficient way for the theatres to keep track of tickets sold/bought, can track/report data of movie sales, and just overall create a better way to track market data.

## 1.3 Definitions, Acronyms, and Abbreviations

| *FAQ* | *Answers* |
|---|---|
| *What is MTTS?* | *Abbreviated term for Movie Theatre Management System.* |
| *What is UI?* | *Abbreviated term for user interface.* |
| *What is SRS?* | *Abbreviated term for System Requirement Specfcations.* |

| | |
|---|---|

### 1.4 References

- CS250 Template(Fall 2023)
- Lecture: Scenario and Use Cases
- Example System Requirement Specifications: Marvel Electronics and Home Entertainment

### 1.5 Overview

The rest of the SRS details the ticketing system itself, functions of the system, and how well it performs. The SRS further dives into the product's interfaces, constraints and various requirements. Our SRS from here on out will detail the product description, the specific requirements followed, and analysis models.

# 2. General Description

The Movie Theatre Ticketing System (MTTS) brings an innovative ticketing system available to customers through a browser-based system. The product aims to serve both the customer base and administration. Customers are able to access the system to view available movie times, tickets, and locations. Whereas administration will be able to manage showtimes, availability, promotions, and their respective details.

### 2.1 Product Perspective

The website will be a standalone application, however, it will be connected with the data/info base to different movie theatres in San Diego to receive new information back and forth without having to input it into different systems. It works in conjunction with other systems to provide a strong user experience.

2.2 Product Functions

This product should handle any customers who want to buy movie tickets at a theater, allowing for the user to easily purchase tickets either online or at a kiosk in the theater. If you are an administrator, you will be able to manage the website and edit as necessary. Furthermore, the software will be able to handle loyalty rewards with the customer, properly storing purchasing data.

### 2.3 User Characteristics

This product targets two groups of users, the average movie watcher, and the employees of the theaters.

Movie goers are not expected to have any prior knowledge to handle the software, and should be able to navigate through the ticket buying process effortlessly.

Employees are expected to have moderate technical skills able to navigate the system properly to ensure a smooth experience for the customers. Higher management employees like managers should be expected to have advanced technical skills able to swiftly navigate the system and be able to manage schedules, update movie listings, track sales, and generate business reports. Of course the different levels of management will be separated by access codes.

### 2.4 General Constraints

The software is expected to be a website that works both at an in-person kiosk and online providing digital tickets that can be printed out but not replicated. It should be consistent and handle easy to interact with no matter the background, while performing on most devices.

### 2.5 Assumptions and Dependencies

The website should be dependent on the different movie theatres database containing all the movies listed, tickets available, etc. If the internal system of the movie theatres database should go down, information pertaining to that theatre will be unavailable but the site should not shut down. It will simply not list information about it or provide a system down message.

## 3. Specific Requirements

### 3.1 External Interface Requirements

#### 3.1.1 User Interfaces

- Website interface
    - Movie listings
    - Different movie theatre locations
    - Checkout/payment screens
    - Maybe “featured” movies

- Seat selections
- Different menus
- Search bars
- Language chooser
- Something that asks for location permissions
- Page for customer input
- Loyalty system

### 3.1.2 Hardware Interfaces

- Theatres provide standard printers for printing tickets and corresponding receipts.
- Theatres provide electronic e-ticket scanners compatible with generated QR codes for online purchases.

### 3.1.3 Software Interfaces

- Showtime Database: Updates available showtimes, deletes expired showtimes, and creates new showtimes for upcoming showings.
- Ticket Database: Stores tickets from verified purchases and  updates seat availability based on purchase.

### 3.1.4 Communications Interfaces

- Created in HTTPS for website
- APIs to communicate between databases or payment systems
- Auto generated emails for sending ticket confirmations
- QR code generators/validations for tickets purchased
- Location services

## 3.2 Functional Requirements

### 3.2.1 <Viewing for available showtimes>

3.2.1.1 Introduction

- This should allow the user to ‘see’ different showtimes for a variety of movies at different theatres within San Diego

3.2.1.2 Inputs

- Clicking on different movie theatres or different movies

3.2.1.3 Processing

- System redirects or shows seating and available tickets derived from database
- Filters out bought out seats and shows accessible ones

3.2.1.4 Outputs

- Displays a screen of available seats if tickets are available.

3.2.1.5 Error Handling

- If no showtimes are unavailable, all options or those that are unavailable will be greyed out and not accessible.

### 3.2.2 <Movie search>

3.2.2.1 Introduction

- This function will allow the users to search for available movies by title, genre, and or date

3.2.2.2 Inputs

- The user entering a movie title in the search bar
- Filtering out genres to see only 1 genre or only some
- Selecting a date on a calendar to see what movie is shown that day

3.2.2.3 Processing

- System connects to the database for movies matching all filters
- System doesn't show or greys out unavailable movies

3.2.2.4 Outputs

- A list of movies matching the filters selected

3.2.2.5 Error Handling

- States no movies found or any sort of message along those lines
- Or if just down, will state system is down.

### 3.2.3 <Pay screen>

3.2.3.1 Introduction

- This is a way for users to pay for movies that they have chosen to buy tickets for

3.2.3.2 Inputs

- Clicking onto "checkout" when selecting seats/tickets
- Inputting payment methods

3.2.3.3 Processing

- Connects to database to mark a seat as "bought"
- Connects to payment systems to process payments

3.2.3.4 Outputs

- Shows a message saying the ticket has been purchased
- Sends an email to provided email for receipt and movie information

3.2.3.5 Error Handling

- Will state payment is failed
- Make sure that payment does not go through
- Gives a message depending on the error reason, payment method failed, system is down, etc.

### 3.2.4 <Sending out QR codes and receipts to emails>

3.2.4.1 Introduction

- This function will send out QR codes that reciprocate to the connection tickets through emails and as well as receipts

3.2.4.2 Inputs

- The user completes their payment for an available ticket
- The user provides an email for receipt/QR code to be sent to

3.2.4.3 Processing

- Connecting to the database to process ticket
- Connecting to an automatic email generator that auto fills for QR code, ticket information, etc.

3.2.4.4 Outputs

- The user will receive an email with the receipt, QR code for their ticket, and relevant information/instructions.

3.2.4.5 Error Handling

- Will not let the user correctly follow through with payment if email could not be sent/error is found. Will state some sort of error message

### 3.2.5 <Seat selection>

3.2.5.1 Introduction

- This function allows users to accurately choose what seat they will occupy during a movie

3.2.5.2 Inputs

- User clicking on non greyed out seats
- User having clicked onto available showtimes

3.2.5.3 Processing

- Connects to movie database to review seats that have already been taken
- Sends out a message to database to state a seat has been reserved

3.2.5.4 Outputs

- Will give a message that a seat has been reserved if clicked on a greyed out seat
- Will give a message saying that seat successfully has been reserved please go to checkout
- Will grey out the seat that they have reserved

3.2.5.5 Error Handling

- If two users click on the same seat but one clicked on it sooner than the other, the later user will receive an error message that the seat has been reserved and refresh their page.

### 3.2.6 <Refund/Cancel booking>

3.2.6.1 Introduction

- This function allows the user to refund their ticket and/or cancel their booking

3.2.6.2 Inputs

- The user opens the site or to view their booking information
- Clicking on the refund options

3.2.6.3 Processing

- Accesses the database to check for ticket
- Sends back to site if ticket there
- Cancels ticket and booking in database
- Access payment method saved/used
- Returns payment

3.2.6.4 Outputs

- Receives email on refunded/canceled ticket/booking
- Gets taken to ticket refunded page

3.2.6.5 Error Handling

- If too late will send show a message
- If cannot find booking will ask to visit in person or email
- If ticket already refunded will let them know

**3.2.7 <Loyalty System>**

3.2.7.1 Introduction

- This function allows the user's purchases to be tracked and allows the user to be rewarded based on their loyalty to the theater.

3.2.7.2 Inputs

- The user accesses their loyalty amount
- The user's name/phone number
- Verify the user identity

3.2.7.3 Processing

- Access the database to check previous purchases
- Calculate loyalty points based on amount of money spent

3.2.7.4 Outputs

- Shows the user possible discounts based on the amount of loyalty points
- Gives the user a discount if they choose to use it

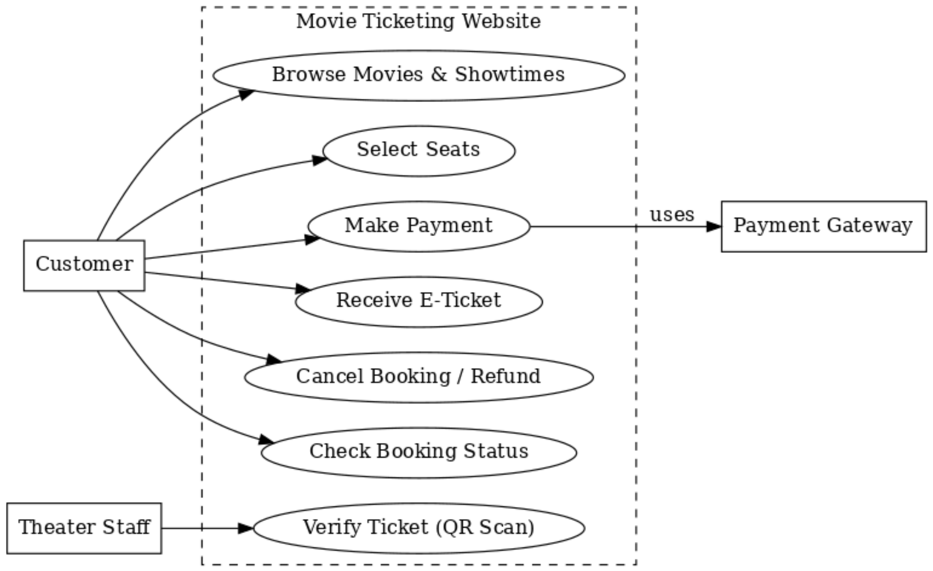
3.2.7.5 Error Handling

- Ensure that the customer has enough points to use the loyalty discount
-

## 3.3 Use Cases

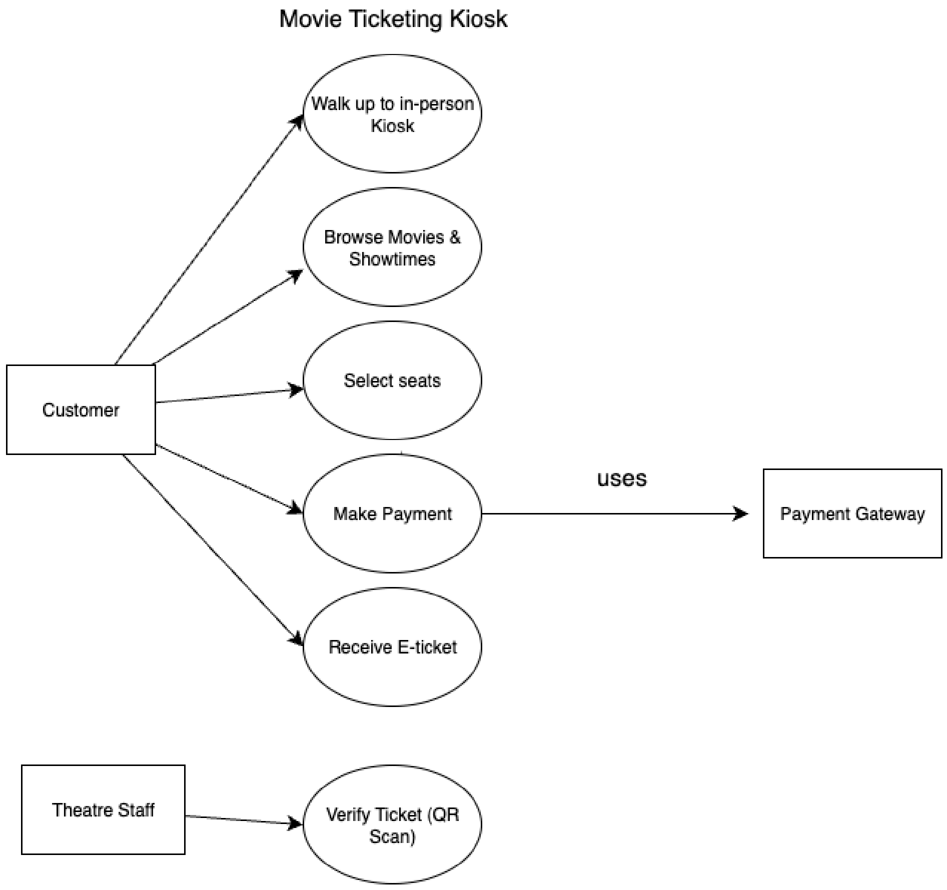
### 3.3.1 Use Case #1

A customer will be able to utilize the website to select movies and buy tickets while being able to check their booking status and request refunds. Payments are processed through the payment gateway and staff must verify tickets.



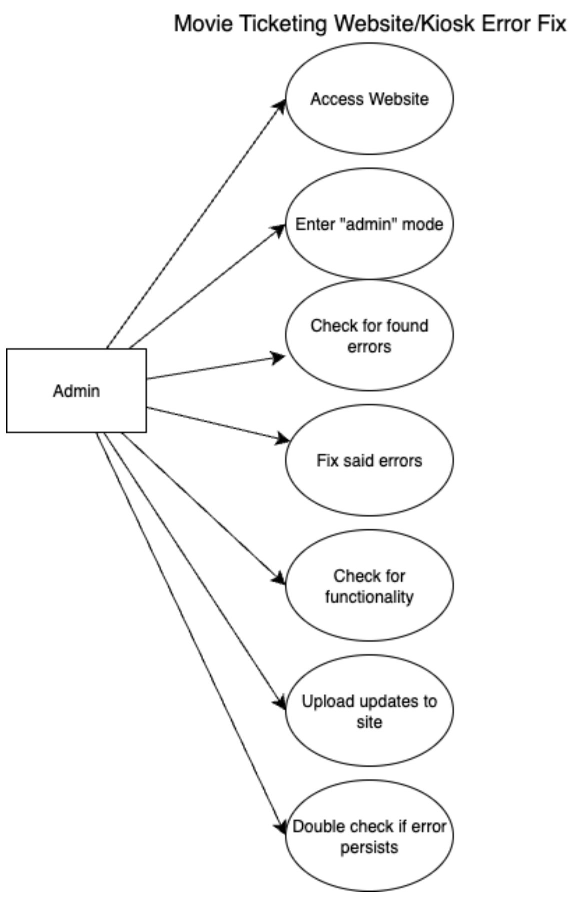
### 3.3.2 Use Case #2

Rather than using the website, a customer can use a kiosk found in the theater also to purchase tickets, browse movies, and receive an e-ticket. The kiosk mainly has ticket purchasing capabilities and payment is also processed through payment gate, tickets are still verified by staff.



### 3.3.2 Use Case #3

The website and kiosk both give higher access to admins, mainly for bugfixing. They are able to enter admin mode and check for errors to fix and ensure functionality and also have access to update features on the site.



## 3.4 Classes / Objects

### 3.4.1 Customer

| **Customer** |
|---|
| **3.4.1.1 Attributes:**<br>- String name<br>- String email<br>- List<Ticket> orderHistory<br>- List<String> feedback<br>- int loyaltyPoints |
| **3.4.1.2 Functions:**<br>- String getName();<br>- void setName(String name);<br>- String getEmail();<br>- void setEmail(String email); |

| |
|---|
| - List<Ticket> getOrderHIstory();<br>- List<String> getFeedback();<br>- void purchaseTickets(Showtime showtime, int quantity);<br>- Void submitFeedback(String comment);<br>- int getLoyalty();<br>- |

### 3.4.2 Administrator

| **Administrator** |
|---|
| **3.4.2.1 Attributes:**<br>- String name<br>- String adminID<br>- String role<br>- String permissions |
| **3.4.2.2 Functions:**<br>- String getName();<br>- void setName(String name);<br>- String getAdminID();<br>- void setAdminID(String adminID);<br>- String getRole();<br>- void setRole(String role);<br>- String getPermissions():<br>- void setPermissions(String permissions);<br>- void manageSeatAvailability(Showtime showtime)<br>- void moderateFeedback(List<String> feedbackList);<br>- void implementDiscounts(List<Discount> discounts);<br>- |

## 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

- The system should be able to handle at least one-thousand people at any given time.
- The system should be able to limit the amount of tickets bought at once to twenty.
    - The system should be able to block bots who are trying to buy tickets in bulk for high-demand movies.

- The system should allow purchase access between two weeks prior to the date of the showing and ten minus before showtime.
- The system should run efficiently; Pages should be 100% loaded within 3 seconds
- Purchases should be processed and completed within 5 seconds

### 3.5.2 Reliability

- Seats are reserved during the process of checkout and payment for up to a ten-minute period.
- Transactions should be made through customers providing card information, or through third-party payment methods.
- Transactions must be immediate and paid in full.
- The system should have nearly no downtime, downtime should be completed within one hour.

### 3.5.3 Availability

- The availability of the system should be all days of the week and twenty-four hours of each day.
- The system should only be unavailable for software updates; alternatively, tickets should still be available for purchase in person at the customers' desired location of attendance.
    - Refer to reliability for more details regarding downtime

### 3.5.4 Security

- The system should prevent bots trying to purchase a large volume of tickets.
- Payment information saved in customer accounts will be encrypted and protected.
- Admin actions and changes will all be documented.
- System will verify if a payment was actually made before providing any form of ticketing

### 3.5.5 Maintainability

- The system will keep updated with frequent maintenance.
- Admin will take in user feedback to increase functionality and usability.
- Updates can be deployed easily during a short downtime, being done in under an hour

### 3.5.6 Portability

- The system will be available on all devices and be able to run on all major browsers with ease.
- The UI will be responsive and adaptive based on the browser and device of either the customer or admin.

## 3.6 Inverse Requirements

- The system will not allow purchases of anything without providing a payment method
- The system will not allow access to the administrator controls without the password and login authentication of an admin
- The system will not allow double booking of the same seat of the same showtime of the same movie
- The system will not allow or show movie listings/showtimes that have already past

- The system will not allow the purchase of tickets for movies that have already past
- The system will not allow the user to keep their ticket after refunding their purchase
- The system will not allow users to log into accounts without the proper credentials needed

## 3.7 Design Constraints

- The system must be compatible with payment programs for secure processing
- The system must be compatible with all different kinds of browsers and computing systems
- The system must be able to handle high amounts of traffic without crashing
- System updates must comply with the company policies and laws of the state and country
- System must be developed under the budget provided by the company
- System must follow data privacy laws

## 3.8 Logical Database Requirements

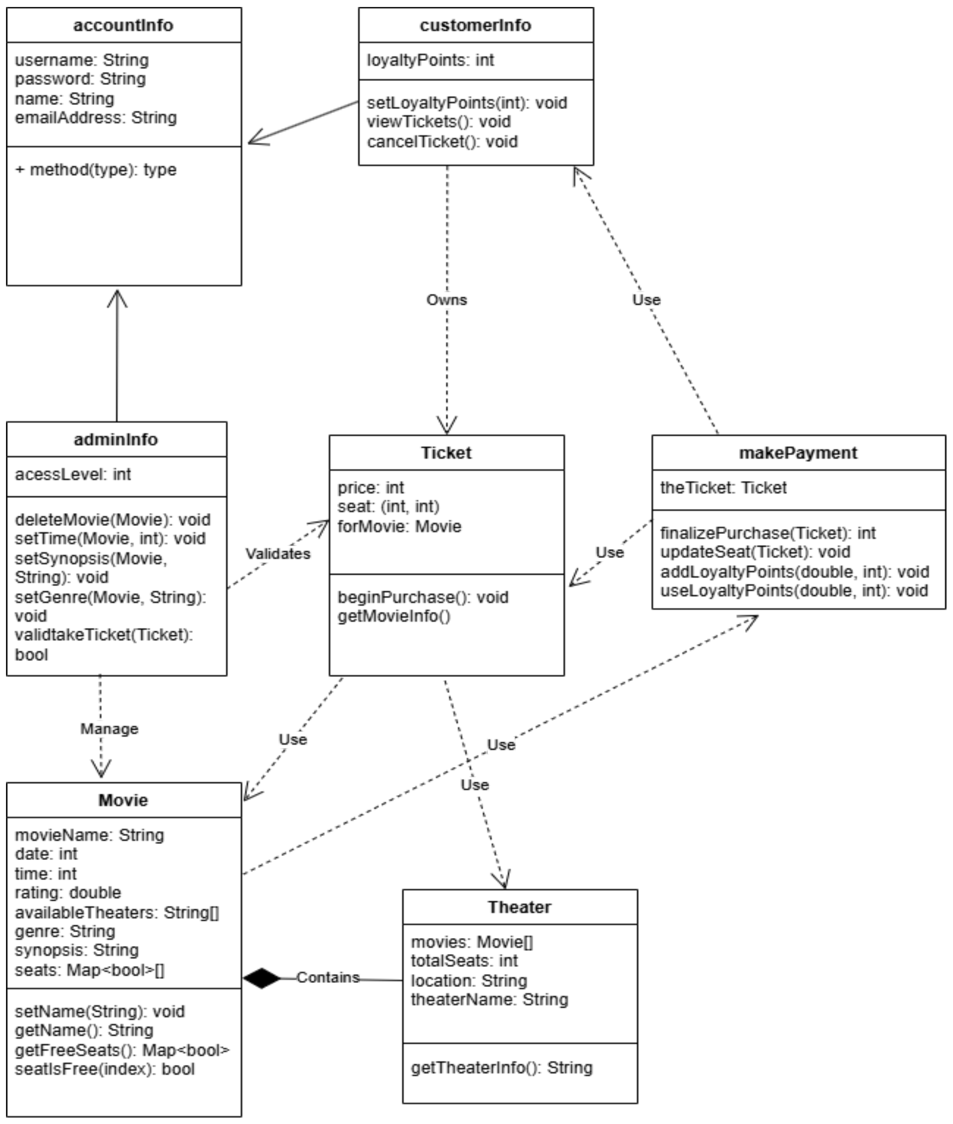
- A database will be used for the Movie Theatre Ticketing System
- As for data formats, prices and payments will be stored in decimal or integer-cents formats to prevent rounding errors.
- Text data will be stored in standard text format.
- Dates and times will be saved in consistent format, and times will be specific to the location of the theatre for the consumer or admin.
- For storage capabilities, the database will back up data frequently to prevent data loss.
- Older data will be archived or deleted.
- Sensitive fields will be encrypted before storage.
- The database must be able to handle large volumes of transactions, both from high-demand showtimes and transactions built up in the database over time.
- In terms of data retention, order and payment records will be retained for a minimum of one year. This is so that records can be audited or reported annually.
- To maintain data integrity, constraints and rules will be in place to prevent double bookings, invalid seat numbers, missing payment records, or fraudulent activity.

## 3.9 Other Requirements

- The system must be able to properly handle any data it receives, adding any new information back into the system
- Any problems with payment should be properly handled and reported to somebody who can deal with it correctly
- The database used by the system should not have any problems with being updated and be easy to upgrade or change

## 4.1 Sequence Diagrams (UML Diagram)

The UML diagram showcases relationships between classes of the MTTS. There is a big web between many of our classes. There are two types of accounts, customers and admins. The account info is a parent to the customer info and admin info. We also have a variety of other classes necessary for the MTTS. This includes tickets, a class to help make payments, theaters, and movies. The ticket is related to every other class, namely the customer, admin, the payment system, the theater, and the movie. It contains important information regarding price, what movie the ticket is for, and seat location, all of which are used in a variety of methods. The payment class contains many methods to help validate payment and ensure that loyalty points are properly awarded to a customer. The theater holds a variety of fields including movies, but has minimal methods. The movie class holds important information regarding the movie, which is used in methods within the ticket. It also has information regarding what seats are available, used indirectly by the payment class through the ticket class.

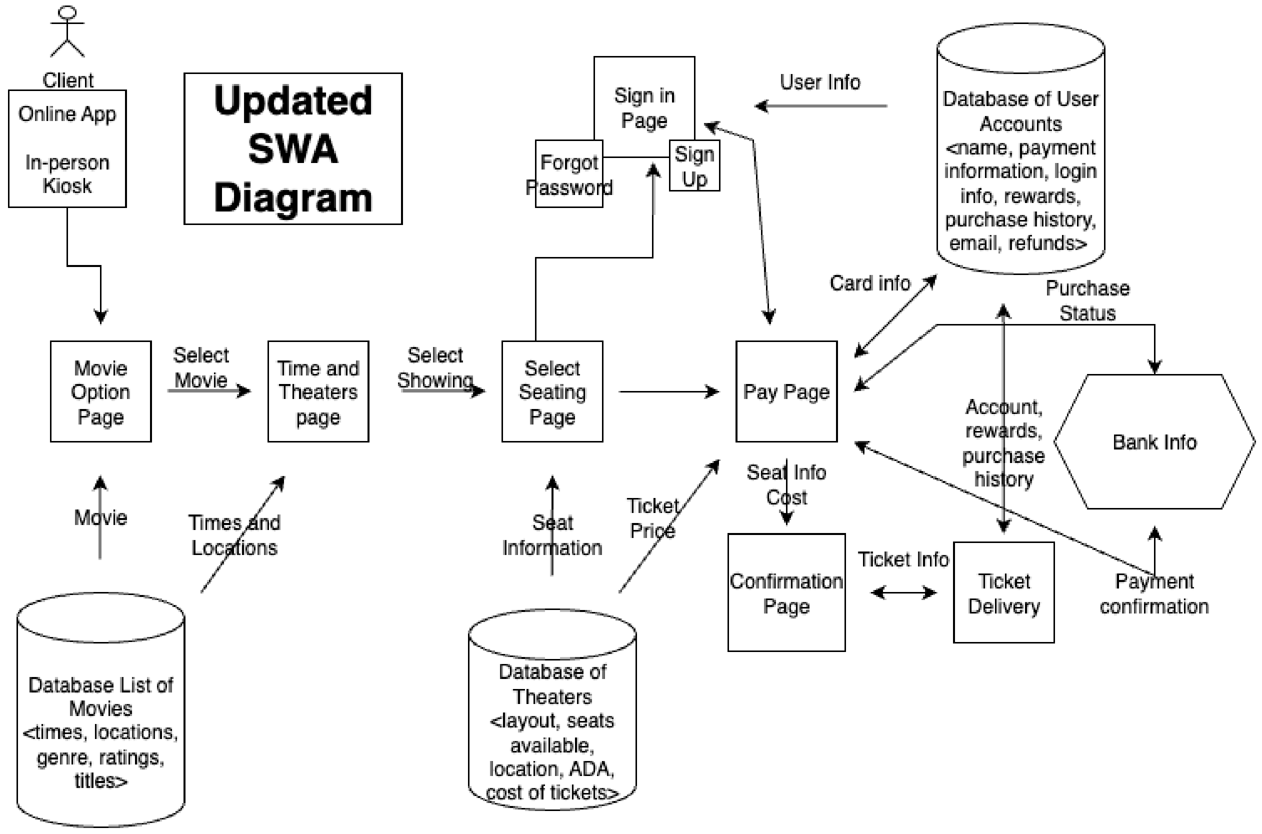


## 4.2 SWA Diagram

There are several main components to this Software Workflow Analysis Diagram. It showcases a user interacting with the site, showcasing how information is received from databases and the path a user must take to reach a certain destination.

The databases are marked as cylinders and each store certain things. The user accounts database stores names, payment info, login info, rewards, purchase history, and email. The theater database stores layout, available seats, location, and cost of tickets. The list of movies database simply holds movies and times and locations. Where certain information of each database is used is showcased by an arrow.

The average user flow begins from the movie option page and their path is showcased by arrows. At the movie option page they select a movie, time, and theater. The user is then directed to sign in if they are not already logged in, and then they are moved to the payment page. The user must then confirm their payment and their payment is processed through the bank. Once their payment is correctly processed, their ticket and confirmation are received.



## 5. Change Management Process

When updating the SRS, as project scope and requirements change, anybody who has worked on the SRS in the past may propose changes to the SRS to fulfill the new scope and requirements. Changes must be approved by anyone involved in the creation of the SRS and version history must be properly documented. Changes should be submitted as a draft that all involved parties can access and edit as necessary.

### 5.1. Development Plan and Timeline

The program will be completed over a time period of six months and the work is split between a team of three people. The workload for the MTTS will split into three sections, one for each person to work on. One person is expected to work on the account system, ensuring that accounts are properly created and stored in the database. Another person is expected to work on properly implementing the ticket system making sure that users can only buy tickets in the correct scenarios. The third person will be working with data and databases, ensuring that all the data is able to be properly read, accessed, and managed.

## 6. Test Cases

A number of test cases will be included in order to test a variety of the functions of the MTTS. There will be three categories of test cases included in this SRS. There are unit tests which are intended to test certain functions of the MTTS. We have a set of functional tests intended to ensure that specific features and functionality meet the required needs of the MTTS described in this SRS without any problems. Finally, we will have a set of system tests ensuring that the system works correctly when performing a full task in the system, i.e. properly purchasing a ticket after launching the website and logging into the account.

### 6.1 Unit Tests

Our unit tests can be found at the excel document found at
https://github.com/jtabalon0518/cs250assignments

The unit tests include verifying important methods essential to running the MTTS. We included tests for the methods getMovieInfo(), deleteMovie(), and getFreeSeats() in order to ensure that key methods worked for admins, the tickets, and movies. These methods are essential to keeping the database clean of unnecessary movies, ensuring the ticket displays the necessary movie information, and that the seat map works correctly.

### 6.2 Functional Tests

Our functional tests can be found at the excel document found at
https://github.com/jtabalon0518/cs250assignments

Our functional tests verify specific requirements of the MTTS, ensuring that certain features work correctly. This includes cases that are uncommon, so we included a test for two users attempting to buy a seat at a similar time. There is also a test for essential functions, the search menu. Another included test regards the refund system, ensuring that a user is unable to refund past a certain time. Our last test is for payment failure, ensuring that a ticket is not awarded to a customer who is unable to pay the fee. These functional tests look into a lot of edge cases, while also covering main features of the MTTS.

## 6.3 System Tests

Our system tests can be found at the excel document found at https://github.com/jtabalon0518/cs250assignments

Our system tests include verifying entire workflows, ensuring that a user is able to specific tasks as they are intended to be completed. There are tests for refunding a ticket, and using loyalty points to buy a ticket, and completing a purchase without using loyalty points. Tests regarding payment are extremely important, so there are several tests here that validate the payment system works correctly and the user receives the necessary feedback regarding their refunds and purchases. The MTTS is a system that mainly focuses on purchasing tickets and ensuring that the user has a good experience, so the system tests will all be centered around a user making a purchase or managing their tickets.

## 6.4 Necessary Changes to Diagrams

Creating the test cases for certain methods involved adding more dependencies for the ticket class, movie class, and payment classes to the UML diagram. We realized that the way our methods were set up, the ticket would require a movie as a field. There was also a need for the make payment class to use the customer class, so that the customer could gain loyalty points and have their loyalty points subtracted during a payment. We also realized there should be a separate seatmap returned for each of the available theaters of a movie. The UML diagram explanation was also updated to help reflect some of these changes alongside extra additions for clarity.

The SWA diagram stayed the same besides a few minor changes like adding more parts dependent on the databases, but there were changes made to the explanation regarding the SWA diagram. We changed our explanation for clarity, and added important details showcasing the MTTS properly.

## 7. Data Management Strategy

The MTTS uses three different databases to run smoothly. This includes a database containing all of the movies including their time and location. There is another that contains user account information, purchase history, and handles payment information. and a database that has all of the theater information including location, layout, and size.

## 7.1 Design Choices

All three of our databases are managed using SQL as it is the best fit for a ticketing system. SQL allows for great consistency and data integrity, helping to keep sensitive information safe and easy to handle. It is very well fit for handling account data, handling transactions, and keeps records of any changes made to user accounts. Our theater and movie databases are also handled using SQL, because SQL covers the specific needs of the MTTS sufficiently. Movies have very structured attributes, with ours including genre, rating, name, location, and title making it easily manageable through SQL. It allows for our users to easily search for movies without needing to get too complex and reducing any overhead. SQL is also fit for our theater system as we use a very simple system for our theaters, with the seat maps being a normal map. It can properly check seat availability and reserve certain seats, which is the main functionality of the MTTS. SQL provides the tools needed for proper user security, and meets all of the needs of the MTTS, so we decided to use it for all of the databases.

We chose to use three databases because it allows for us to separate vastly different types of data. The data needed in the MTTS was categorized based on organization and security which ended up becoming our databases. We wanted to group sensitive account and payment information together because they require a higher level of security and rely on each other, creating the account database. Movie data and theater data are separated into two separate databases mainly for organizational reasons, allowing us to separate different kinds of data and handle them separately.

The main alternative that we considered was NoSQL, and the reason we decided against using it alongside what we lose by not using it is covered more in detail in 7.2.

## 7.2 Tradeoffs

Deciding to use SQL over the NoSQL alternatives for our databases had its advantages and drawbacks for the MTTS. Its upsides included greater consistency and accuracy for data, linked related tables easily, handled transactions safer, and made reporting more simple and direct. Not choosing its counterpart meant giving up accessibility in adding more servers or databases to handle more users and data at the same time and not being flexible to document types; this meant each record could have been stored in different kinds of information instead of being bounded by a strict structure.

A big part in why we chose SQL for all three databases is maintaining consistency between each database. Despite losing certain advantages from mixing databases, using SQL for all three instead of mixing database types depending on what is most technically most fit for each type of database allows for the databases to be handled more easily by a lower amount of people. There is less overhead in needing to train people to handle several different kinds of databases and allows for the MTTS to reduce costs.

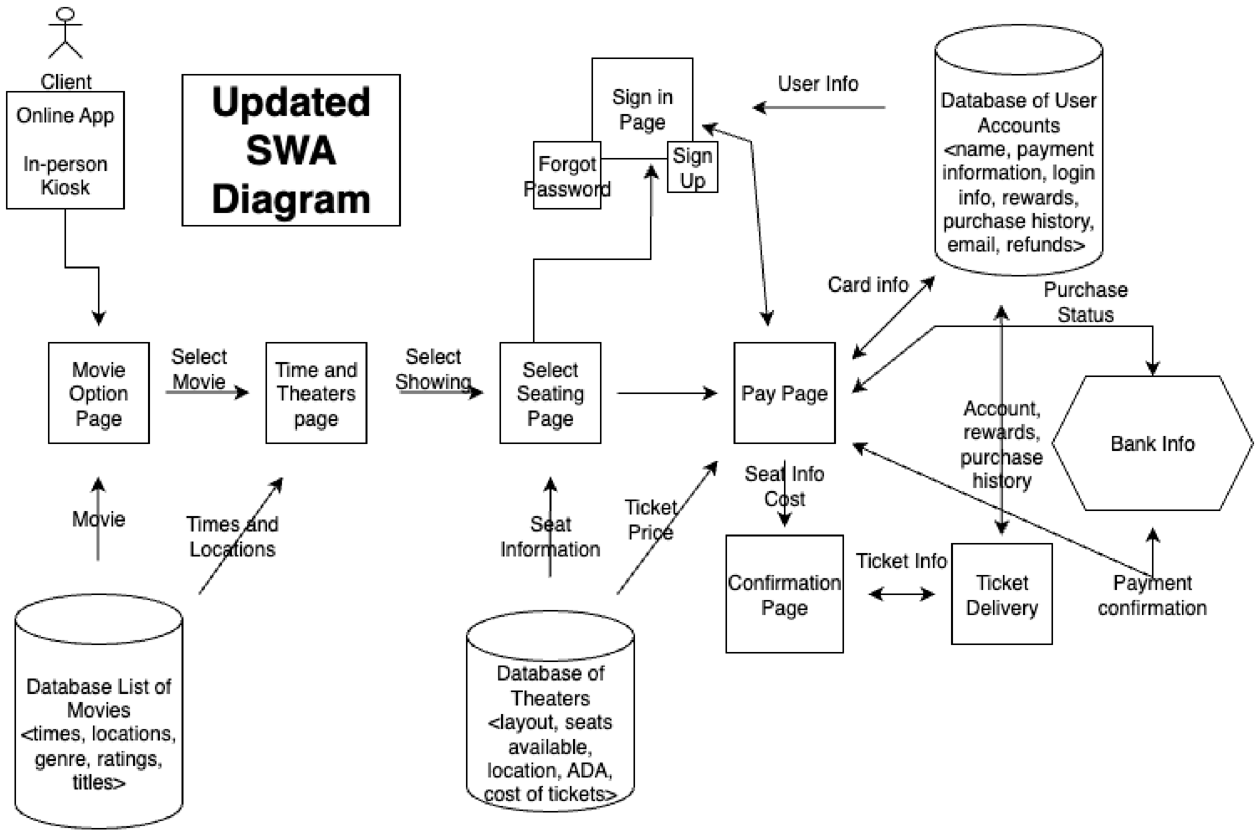
Since we chose to use three databases(Movies, Theatres, User Accounts), it helped separate the data and limited how much sensitive data is exposed or leaked if one database is compromised. The separation also kept each section more organized and focused, making it easier to update and match how different parts of the system used the data and which databases it was necessary to access. On top of these two benefits from the three databases, if we plan on growth for the MTTS, expansion and scaling for the system is far more functional through this method.

Drawbacks from the three databases, although few, include that queries that need access from several databases become more complex and the system in general must be able to perform to handle communication across multiple databases correctly.

## 7.3 Changes Made to Diagrams

We decided to make a very small change to our SWA, showing more attributes that the movie database holds to show a bit more clarity. These attributes include rating, genre, and title.

Reference to SWA.



## 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