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 Ticketing System

Software Requirements Specification

Version 4

11/07/2024

Group 5
James Johnson, Seth Blanchard, Marvee Balyos, Jack Roemer

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

Revision History

Date	Description	Author	Comments
9/26/2024	Version 1	Group 5	First Revision
10/10/2024	Version 2	Group 5	Design & Timeline
10/24/2024	Version 3	Group 5	Testing Plan
11/07/2024	Version 4	Group 5	Architecture Redesign w/ Data Management

Document Approval

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

Signature	Printed Name	Title	Date
	James Johnson	Software Eng.	9-26-2024
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a comprehensive description of the Movie Theater Ticketing System (MTTS). This document outlines the system's objectives, functionalities, and constraints to guide the development team in building an effective solution for selling movie tickets both in person and online. The intended audience for this SRS includes the software developers responsible for implementing the system, project managers overseeing the development process, quality assurance teams, and stakeholders from the company that has commissioned this system.

1.2 Scope

The Movie Theater Ticketing System (MTTS) is a comprehensive software solution designed to facilitate the sale of movie tickets both in person at theater locations and online through a user-friendly website. The software product aims to enhance the customer experience and streamline theater operations by providing the following key functionalities:

1. **Ticket Purchasing**: Customers can browse a catalog of available movies, select preferred showtimes, choose specific seats, and purchase tickets seamlessly.
2. **Purchase Modification and Cancellation**: The system allows customers to modify or cancel their ticket purchases within the theater's policy guidelines, offering flexibility and improving customer satisfaction.
3. **Account Creation and Management**: Customers have the option to create personal accounts, enabling them to save their preferences, view purchase history, and expedite future transactions.
4. **Rewards Program Integration**: To encourage repeat patronage, the MTTS incorporates a rewards program where customers earn points for each purchase. These points can be redeemed for discounts on future ticket purchases or special promotions.
5. **Administrative Management**: Theater management can add, modify, and remove movie listings, showtimes, and manage seat availability through an intuitive administrative interface connected to a centralized database.
6. **Information Dissemination**: The system generates detailed reports for accounting and management purposes, providing real-time insights into ticket sales, revenue, and customer behavior.

1.3 Definitions, Acronyms, and Abbreviations

This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret this Software Requirements Specification (SRS). This information may be provided by reference to one or more appendices in the SRS or by reference to other documents.

Definitions:

- **Theater**: One of the individual rooms in which a movie will be shown.
- **Location**: A cinema where multiple theaters are located.
- **Customer User**: A person who accesses the website to purchase tickets and can create, modify, or delete an account with the system.
- **Customer Service User**: An employee who assists Customer Users when they seek assistance, including modifying or refunding purchases and modifying or deleting customer accounts.
- **Management User**: An administrative user who can add, modify, and remove movies from which tickets can be purchased.
- **Movie Showing**: A specific screening of a movie at a designated date, time, theater, and location.
- **Seat Reservation**: The process of selecting and holding a specific seat for a movie showing.
- **Account Verification**: The process by which a Customer User confirms their email address to activate their account.
- **Loyalty Points**: Rewards accumulated by Customer Users through purchases, which can be redeemed for discounts or special offers.
- **Refund Window**: The period before the movie showing during which a customer can cancel their ticket for a refund.

Acronyms:

- Movie Theater Ticketing System - MTTS
- Amazon Web Services - AWS
- Unified Modeling Language - UML
- Software Architecture - SWA

1.4 References

This SRS does not reference outside documentation and works as a standalone document.

1.5 Overview

The rest of this document will provide key information regarding the purpose, scoop, and functionalities of the movie theater ticketing system. The SRS document itself is organized into 4 parts. An introduction section describing the objectives and goals of the system, along with its limitations. It will also contain a list of definitions, references and overview. The second section contains the overall description and gives a high level overview of the ticketing systems functionalities. Describing the systems environment, its main features, its users, constraints and assumptions. Next there will be a subsequent section containing specific requirements. It will contain a list and description of both functional and non-functional requirements. The final

section will contain appendices, providing additional information that supports the main content such as models, and designs.

2. General Description

2.1 Product Perspective

The Movie Theater Ticketing System, although an independent solution used to purchase tickets, serves as a component for several different systems.

The system integrates with payment gateways, such as Stripe and PayPal, to facilitate transactions.

The system interfaces with the theater's scheduling and seat management systems, ensuring that tickets purchased in-person are added to the database for online customers, and real-time updates for seat availability and showtime information are made.

In-theater operations such as ticket printers and card readers must handshake with the database in order to ensure all tickets are read by the system.

2.2 Product Functions

- **Purchase Tickets**
    - Allows a Customer user to purchase tickets for a movie with a given time, theater, and location. Within this function, available rewards from having an account may be applied.
- **Modify Purchases**
    - Allows Customers and Customer Service users to modify a purchase that has been made. Such as reducing the number of tickets or changing the chosen time, theater, and location.
- **Refund Tickets**
    - Allows Customer and Customer Service users to refund the purchase of tickets. It will also update the showing to reflect this change. With the Customer user being limited to purchases that they have made and the Customer Service user being able to refund any purchase.
- **Create Account**
    - Allows a Customer user to make an account with the system. The Customer will provide an email and will set a password for the account. The email should be verified to be owned by the Customer, and the password should meet minimum requirements.
- **Modify Account**
    - Allows a Customer and Customer Service user to modify a given account. Allowing the modification of the email and password along with any other information.
- **Remove Account**

    - Allows a Customer and Customer Service user to delete a given account.
- **Login**
    - Allows a Customer user to log into their account on the website.
- **Add Movie Showing**
    - Allows a Management user to add a new movie showing with the given movie name, time, theater, location, and number of seats. The movies will be added to the "Movie" database table.
- **Modify Movie Showing**
    - Allows a Management user to modify the information of a given showing.
- **Remove Movie Showing**
    - Allows a Management user to remove a given showing. In the case there were already purchases for a given movie, it should also issue refunds and send an email alerting the Customer users.
- **Movie Search**
    - Allows a Customer user to search for a movie. The user can enter in a date range, time range, select available titles, and select locations and this function will return all showings that meet those requirements.

2.3 User Characteristics

The users of the Movie Theater Ticketing System are categorized into three main types:

1. **Customer User**
    - **Education**: No specific educational requirements.
    - **Experience**: Minimal to no prior experience with the system is expected.
    - **Technical Expertise**: Basic internet browsing skills; should be able to navigate a website and complete online transactions without assistance.

    The Customer User accesses the website to purchase tickets and may create, modify, or delete an account with the system. The system is designed to be user-friendly to accommodate users with varying levels of technical proficiency.

2. **Customer Service User**
    The Customer Service User assists Customer Users when they require support. Their responsibilities include modifying or refunding purchases and modifying or deleting Customer accounts.
    - **Education**: High school diploma or equivalent is required.
    - **Experience**: Prior experience in customer service roles is preferred, particularly in retail or hospitality industries where interaction with customers is frequent.
    - **Technical Expertise**:
        - **Basic Computer Skills**: Proficient in using computers and navigating web-based applications.
        - **Software Proficiency**: Ability to learn and efficiently use the Movie Theater Ticketing System after training.

        - **Communication Skills**: Excellent verbal and written communication skills to effectively assist customers.
        - **Problem-Solving Abilities**: Capable of handling customer inquiries and resolving issues promptly.
        - **Data Entry Skills**: Accurate input of customer information and transaction details into the system.

    The Customer Service User will receive training on the operation of the system to ensure they can provide effective assistance to customers.

3. **Management User**
    - **Education**: Bachelor's degree in business administration, management, or a related field is preferred.
    - **Experience**: Experience in managerial roles within the entertainment industry or related fields.
    - **Technical Expertise**:
        - **Advanced Computer Skills**: Proficient in using administrative and database management tools.
        - **Analytical Skills**: Ability to interpret sales data and generate reports.
        - **Software Proficiency**: Skilled in using the administrative functions of the Movie Theater Ticketing System, including adding, modifying, and removing movie showings.
        - **Decision-Making Abilities**: Capable of making informed decisions based on system data and customer feedback.

    Management Users will receive comprehensive training to utilize all administrative features of the system effectively.

2.4 General Constraints

Constraints:

1. **Regulatory Compliance:**
    - **Payment Card Industry Data Security Standard (PCI DSS):**
        - The system must comply with PCI DSS requirements to securely handle and process credit card transactions.
    - **California Consumer Privacy Act (CCPA):**
        - If applicable, the system must adhere to CCPA regulations for California residents.
2. **Technology Stack Constraints:**
    - **Standardization:**
        - The system must be developed using the company-approved technology stack to maintain consistency with existing infrastructure.
    - **Frontend Technology:**
        - Use of Next.js and Tailwind for the frontend.

- **Backend Technology:**
  - Stripe and PayPal for payment processing, NextAuth for user authentication, PostgreSQL for database management.
- **Database Compatibility:**
  - The system must integrate with the existing Database Management System (DBMS) in PostgreSQL.

3. **Hardware Limitations:**
   - **Server Specifications:**
     - The system must operate within the hardware capabilities of the current servers without requiring significant upgrades.
   - **User Devices:**
     - Must support a wide range of customer devices, including desktops, laptops, tablets, and smartphones, without specialized hardware.
4. **Browser and Operating System Support:**
   - **Browser Compatibility:**
     - The system must be compatible with the latest two versions of major web browsers (Chrome, Firefox, Safari, Edge).
   - **Operating Systems:**
     - Support for common operating systems, including Windows, macOS, iOS, and Android.
5. **Integration Constraints:**
   - **Third-Party Services:**
     - The system must integrate with specified third-party payment gateways (e.g., Stripe, PayPal) and email service providers
   - **Legacy Systems:**
     - Must ensure compatibility with any existing legacy systems used by the company for accounting or management purposes.
6. **Security Constraints:**
   - **Data Encryption:**
     - All sensitive data must be encrypted both in transit (using SSL/TLS) and at rest.
   - **Authentication Protocols:**
     - Implementation of secure authentication methods, including multi-factor authentication for administrative access.
   - **Password Policies:**
     - Enforce strong password requirements and secure storage practices (hashing with salting). The same password should not result in the same entry in the database.
7. **Performance Constraints:**
   - **Scalability:**
     - The system must handle peak traffic periods (e.g., during blockbuster releases) without performance degradation.
   - **Response Time:**

        - Page load times and transaction processing should not exceed 3 seconds under normal operating conditions.
8. **Time and Budget Constraints:**
    - **Project Timeline:**
        - Development and deployment must be completed within the stipulated time frame as per the project schedule.
9. **Data Retention:**
    - **Retention Policies:**
        - Keep transaction records for a minimum of five years for audit purposes.
10. **Internationalization and Localization:**
    - **Language Support:**
        - The system must support multiple languages as required by the company's market presence.
    - **Date and Time Formats:**
        - Must handle various international date and time formats based on user settings or location.
11. **Dependency on External Systems:**
    - **Service Availability:**
        - The system's functionality may be limited by the availability and reliability of external services like payment gateways and email providers. Therefore, rely on external APIs only if necessary.
12. **Legal Constraints:**
    - **Intellectual Property Rights:**
        - Ensure that all software components, including third-party libraries, comply with licensing agreements.
    - **Content Restrictions:**
        - Must not display unauthorized content, adhering to copyright laws and regulations.
13. **Environmental Constraints:**
    - **Hosting Environment:**
        - All systems should be deployed through AWS.
14. **Maintenance Constraints:**
    - **Update Schedules and Downtime:**
        - The system should retain an uptime of 99.9% year-round. If necessary, crucial maintenance downtime should occur between the hours of 12 a.m. and 3 a.m. PST.

2.5 Assumptions and Dependencies

Assumptions

- It assumes that payment is processed through either Stripe or Paypal and that no user credit card information is stored in this system.
- It is assumed that theaters have the necessary hardware to talk to the database and complete in-person ticket purchases

- It is assumed that all third party services we use conduct security standards in an appropriate manner (Stripe, SendGrid)
- We are assuming that each theater would update its information on its own, including the showtimes and types of movies that will be available at each theater.

Dependencies

- Payment gateways such as PayPal and Stripe
- SendGrid for email services
- Amazon Web Services AWS for PostgreSQL and server hosting
- Hardware devices located on theater premises are responsible for communicating to the database when a purchase is made offline, in order to update seats and availability.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The MTTS will feature a web-based user interface for a seamless and intuitive experience for customers and other users. Such interfaces include:

- Home Page
    - Displays the movie showtimes, promotions, and login/register options.
- Movie Selection
    - Allows users to filter movies by date, time, genre, and location.
- Ticket Purchase
    - Provides a clear seat map, pricing information, and payment options for the user.
- Account Management
    - Enables users to view past purchases, redeem rewards, and manage account settings (account deletion, password modification)
- Admin Dashboard
    - If necessary, an administrator can remove or add movies in the case of a database failure.
- Locations page
    - Interactive Google map view showing all theater locations.

3.1.2 Hardware Interfaces

- Card Terminals
    - Integration if a ticket is purchased in-person.
- Ticket Printers
    - Provides ticket data to printers to print tickets and receipts.
- Server Infrastructure
    - The system is hosted on servers that support the necessary bandwidth and processing power to handle peak loads.

3.1.3 Software Interfaces

- Payment Gateways
    - The system will need to integrate Stripe and PayPal for all online ticket purchases.
- Emails
    - The system will use SendGrid to send emails to users for the following reasons:
        - Account verification
        - Ticket receipts
        - Discounts and offers
- Database
    - Uses PostgreSQL for storing user data, transaction records, movie listings, and seat availability

3.1.4 Communications Interfaces

The MTTS requires communication through:

- HTTP/HTTPS Protocol
    - Facilitates secure web based communication for the server and browser users.
- API Integrations
    - Utilizes REST APIs for exchange with third party services (payment processors and email services)
- Internal Networks
    - For in-theater purchases, the system will communicate with the local network for printing tickets and accessing the database.

3.2 Functional Requirements

3.2.1 Purchase Ticket

3.2.1.1 Introduction

This function is for Customer users to purchase a ticket for a movie.

3.2.1.2 Inputs

User selection of the movie, date, time, location, theater
User's payment information (Visa, Master Card, PayPal)
User account information (optional for registered users)
If user is logged in, they're given a choice to use reward points for a price discount

3.2.1.3 Processing

Confirm the ticket is still available in the system.
Conduct the user's payment through the payment processor
If the user has opted to use reward points, ensure they have the amount requested in their account, and apply the discount (100 points = $1)
If the user purchased the ticket while registered, apply 25 points of rewards to their account.

3.2.1.4 Outputs

Give receipt to customer via email address on file if registered
If not registered, display the ticket and provide an option to print and send to an email address.
Send receipt to accounting
User's with an account will have award points added to their account

3.2.1.5 Error Handling
The transaction cannot go through because there is no money in the user's account.

3.2.2 Create Account
3.2.2.1 Introduction
This system allows users to optionally create an account. By registering an account, users can receive notifications about early purchases for movie tickets. This feature enables account holders to have priority access to tickets before they are made available to the general public.

3.2.2.2 Inputs
User provides e-mail and password to system
User confirms email to system by clicking verification link

3.2.2.3 Processing
The system verifies the user's email is not already registered.
The system verifies the user's password meets the following criteria:

- Greater than 8 characters
- Less than 64 characters
- Contains at least one number and one special character (!@#$%^&*())

The user's email, and salted, hashed password is stored into the database along with the following information:

- Account verification status
- Account creation date
- Tickets bought
- Loyalty points (default 0)

The user is sent a unique URL which upon visiting will set the user's verification status to true.

3.2.2.4 Outputs
The account has been successful. A message confirming it has been displayed.
The user gets an email confirming their order.
The user is asked to sign up for alerts on movie showings ahead of time.

3.2.2.5 Error Handling
If the email is already in use by another account registered on the platform the user will be notified with an error message and is prompted to either sign in or choose an email address.

3.2.3 Refund Ticket

3.2.3.1 Introduction

This system is for users who wish to refund their ticket prior to the showing.

3.2.3.2 Inputs

Refund the purchase of a ticket at most a day before the showing.

3.2.3.3 Processing

The system verifies if the ticket is able to refunded based on the following condition:

- Ticket date is no less than 1 day in the future

The system processes the cancellation and refunds the user's purchase with their existing credit card on file.

3.2.3.4 Outputs

Display a confirmation of the refund to the user.
E-mail the user and the administrators a confirmation of the refund.

3.2.3.5 Error Handling

If the user isn't eligible for a refund, notify the user with an appropriate message.
If there is an issue with the refund process, inform the user and log the error for review.

3.2.4 Modify Purchase

3.2.4.1 Introduction

This function lets customer service representatives make changes to customer's purchases post sale like adjusting movie showtimes or seat preferences after the initial transaction is completed.

3.2.4.2 Inputs

Purchase ID or customer information (such as an email or phone number) so we can access the Purchase information for you. Recent acquisition specifics including;
New purchase details, such as:

- Update showtime
- Number of tickets
- Seating preferences
- Payment adjustments (if necessary)

3.2.4.3 Processing

The system fetches the purchase information using either the Purchase ID provided or customer details.
The system verifies whether changes are permissible (, for example; within a specified time period, before the movie commences; availability of seats).

3.2.4.4 Outputs

When a customer service representative confirms the information, on the purchase screen a message pops up and the customer also receives an email with the purchase information.
The latest transaction has been saved in the database.

3.2.4.5 Error Handling

If the time for changes has already elapsed (for example if it's too close to the time of the show) or if there are no seats left to book, an error message will appear with an explanation of the problem.

If there's an issue with processing payment adjustments, a notification will be displayed prompting the representative to try another payment method.

3.3 Use Cases

3.3.1 Search For Movie

The client must be able to search various movie theaters for particular movies and showtimes. As well as view information related to the movies they want to watch and the available seating.

3.3.2 Purchase Ticket

The client must be able to purchase movie tickets for a particular time, place, showing, and seating.

3.3.3 Create Account

The client must be able to create an account in order to keep track of their ticket information and rewards points.

3.3.4 Access Account

The client or administrator must then be able to access their account in order to view their information.

3.3.5 Modify Purchases

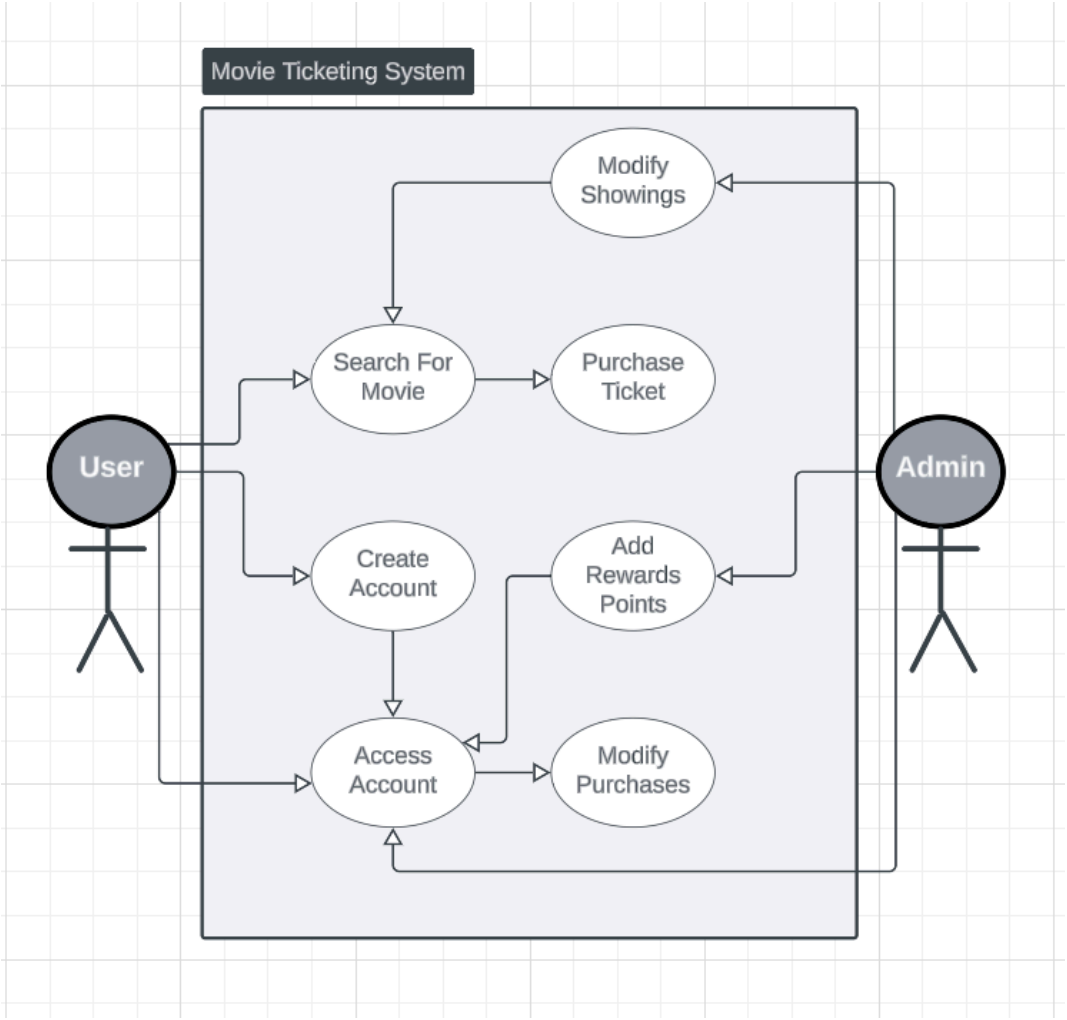
There must also be a way for the clients or administrators to access their tickets in order to make modifications, cancellations or refunds.

3.3.6 Add Rewards Points

The system or administrator must be able to add rewards points to certain accounts for client purchases.

3.3.7 Modify Showings

The administrator must have a way to modify the information related to the movies available, their showtimes, their available locations or any other information necessary.



3.4 Classes / Objects

This section outlines the main classes or objects in the Movie Theater Ticketing System, including their attributes and functions. Each class is directly linked to the functional requirements and use cases specified earlier in this document.

3.4.1 MovieShowing

*Note: While the **MovieShowing** may be handled as a database entity, it is described here as a class for the purpose of this specification.*

3.4.1.1 Attributes

The **MovieShowing** class contains several attributes essential for managing movie showings:

- **movieTitle**: Represents the title of the movie being shown.
- **showingDate**: Indicates the date of the movie showing.

- **showingTime**: Specifies the time when the movie starts.
- **theaterNumber**: Denotes the number of the theater where the movie is being shown.
- **location**: Refers to the physical address of the movie theater.
- **availableSeats**: Represents the number of seats currently available for the showing.
- **totalSeats**: Indicates the total seating capacity of the theater.
- **pricePerTicket**: Specifies the cost of a single ticket for the showing.
- **showingID**: A unique identifier assigned to each movie showing for tracking purposes.

3.4.1.2 Functions

The **MovieShowing** class provides functions to manage and retrieve information about movie showings:

- **getAvailableSeats()**: Returns the current number of available seats for the showing.
- **reserveSeat(seatNumber)**: Reserves a specific seat for a customer, updating the available seats accordingly.
- **cancelReservation(seatNumber)**: Cancels a seat reservation, making the seat available for others.
- **updateSeatAvailability()**: Updates the count of available seats after reservations or cancellations.
- **getShowingDetails()**: Retrieves all pertinent details about the movie showing, including movie title, date, time, theater number, location, and pricing information.

References to Functional Requirements and Use Cases

The ***MovieShowing*** class is associated with the following:

- **Functional Requirements**:
    - 3.2.1 Purchase Ticket
    - 3.2.5 View Available Movies
    - 3.2.11 Movie Search
- **Use Cases**:
    - 3.3.1 Search For Movie
    - 3.3.2 Purchase Ticket
    - 3.3.7 Modify Showings

3.5 Non-Functional Requirements

3.5.1 Performance

The system should be capable of handling up to 500 concurrent users without performance degradation. The system should be prepared to scale in the case that more locations are constructed, or certain movies are high in demand.

95% of transactions shall be processed within 3 seconds under normal operating conditions.

Page load times for all user interfaces shall not exceed 3 seconds for the first load and 1.5 seconds for subsequent loads.

3.5.2 Reliability

The system shall maintain an average of 30 days between any failure or instance of downtime. The system shall ensure data integrity with a rate of 100%.

3.5.3 Availability

The system shall maintain 99.9% uptime, with a maximum of 8.8 hours per year. Scheduled maintenance windows shall not exceed 2 hours per month and will occur during off-peak hours (between 12 a.m. and 3 a.m. PST).

3.5.4 Security

All sensitive data, including payment and personal information, shall be encrypted using industry-standard AES-256 encryption. The system shall implement multi-factor authentication (MFA) for administrative access.

User passwords shall be stored using salted and hashed encryption techniques.

3.5.5 Maintainability

The system's codebase shall adhere to modular design principles, and past versions of the software should be updatable or replaceable without impacting the system.

Routine software updates should be implemented without requiring complete system downtime.

Comprehensive documentation shall be provided for all software components.

3.5.6 Portability

The system shall be compatible with all major web browsers (Chrome, Firefox, Safari, Edge).

The system shall be deployable on Windows and Linux-based servers without requiring, if any, significant changes to the codebase.

The system should be compatible with mobile devices due to responsive CSS practices.

3.6 Inverse Requirements

- The system shall not store user's credit card information in any way
- The system shall not allow more than 10 failed login attempts from an IP address within the span of 5 minutes
- The system shall not allow the refund of tickets 24 hours prior to movie screening

- The system shall not support the streaming of movies
- The system shall not support the sale of concession items

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 to the SRS's requirements.

4.1 Sequence Diagrams

4.3 Data Flow Diagrams (DFD)

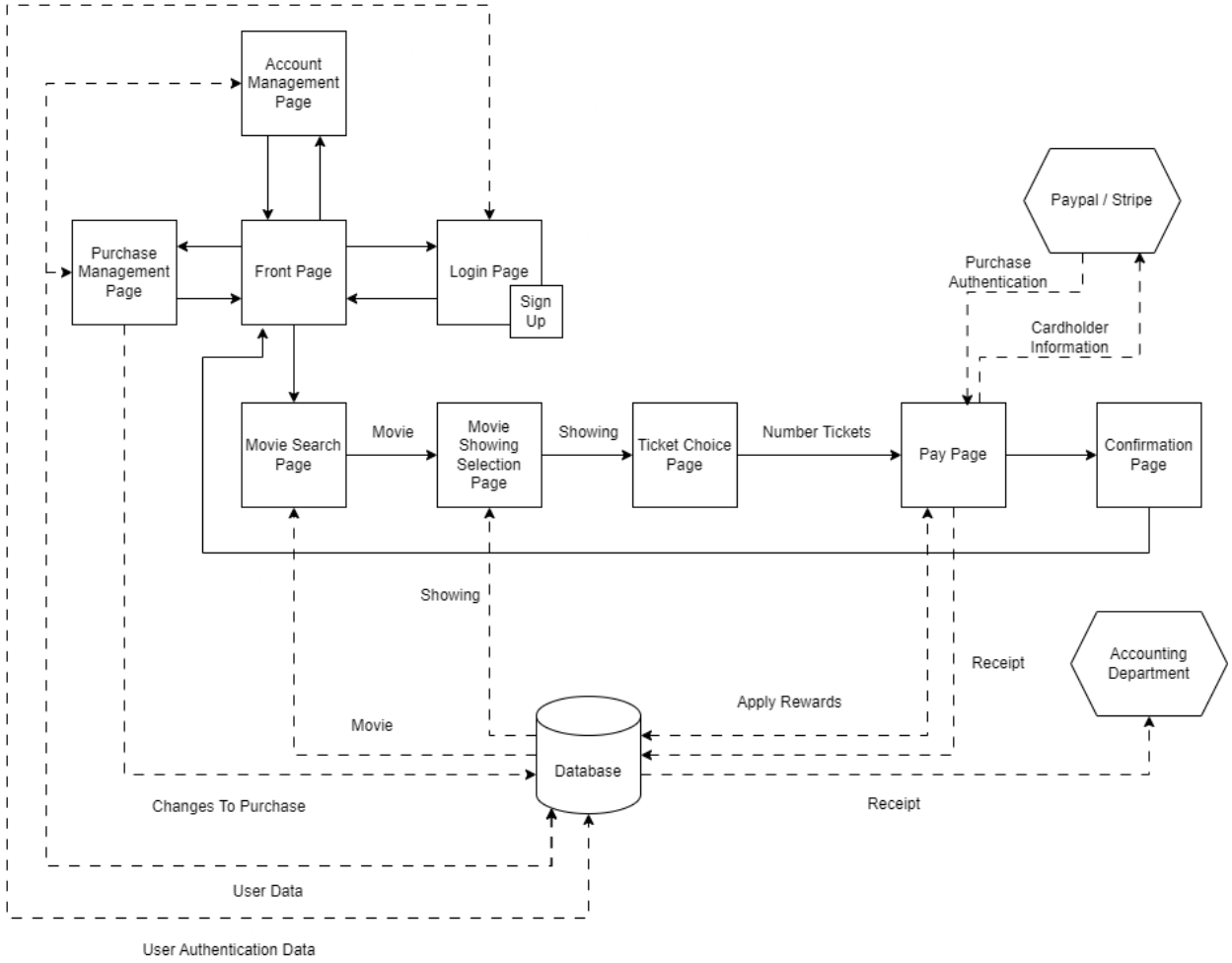
4.2 State-Transition Diagrams (STD)

5. Change Management Process

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

6. Design Specifications

6.1 Software Architecture Diagram



6.1.1 SWA Description

The diagram shows the streamlined process for users to register, sign up, search for their movies, pick showtimes, choose tickets, and pay. The system is well-integrated with external payment solutions to ensure safe transactions. Furthermore, users can manage their own accounts, view purchases, and apply awards. The SWA is described as below:

Pages and Components

1. Front Page
    a. This serves as the central hub of the system. From this page, users can navigate to the Login Page, Movie Search Page, and Account Management Page if authenticated.
2. Login Page
    a. This allows users to log in or sign up for a new account.
    b. Once authenticated, users are redirected back to the front page.

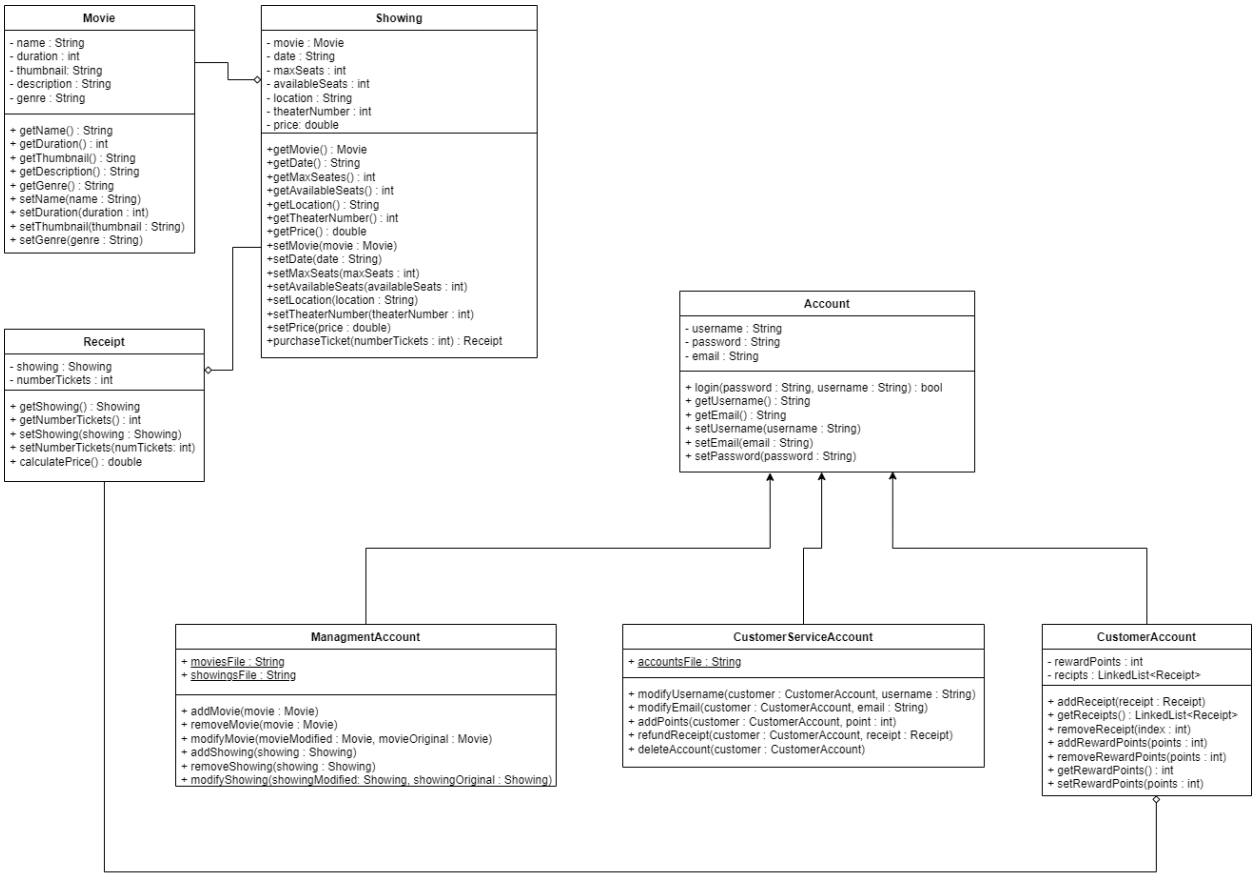
    c. If users are not logged in but want to apply rewards, they are sent to the login page during their payment process, and redirected back afterwards.
3. Account Management Page
    a. Accessible only to authenticated users. This allows users to manage their account information and settings (such as password).
4. Purchase Management Page
    a. From the Front Page, users can access their purchase history and manage past ticket purchases.
5. Movie Search Page
    a. Users can browse through available movies to find one they want to watch.
    b. Upon selecting a movie, they are taken to the Movie Showing Selection Page.
6. Movie Showing Selection Page
    a. This page displays available showtimes for movies. This is viewed after selecting a movie.
7. Ticket Choice Page
    a. Here, users can select the amount of tickets they want to purchase for the chosen show time.
    b. After the user makes their choice, they are moved on to the Pay Page to continue their transaction.
8. Pay Page
    a. This page handles the payment process. It allows users to enter their payment information, and apply rewards if they have any.
    b. If they choose to apply rewards, but aren't logged in, send them to the login page while retaining the purchase session data.
    c. Allow the user to pay with PayPal and Stripe for secure payments.
9. Paypal / Stripe
    a. These external services are used for processing payments securely. After entering cardholder information and authenticating the purchase, the payment gateway sends the user back to the system.
10. Confirmation Page
    a. After a successful payment, the user is directed to this page to confirm their ticket purchase.
    b. The confirmation page includes navigation options to the Front Page, Account Management Page, and Purchase Management Page.

Connectors and Flow

1. Front Page
    a. Serves as the main navigation point.
    b. Users can navigate to:
        i. Login Page
        ii. Movie Search Page
        iii. Purchase Management Page
2. Login Page
    a. Connects users to Account Management Page after successful authentication.

    b. Includes the “Apply Rewards” option that links to the Pay Page for users wishing to use rewards during checkout.
3. Movie Search Page
    a. Allows users to browse movies and select one for more details.
    b. Directs users to the Movie Showing Selection Page once a movie is selected.
4. Movie Showing Selection Page
    a. Displays available showtimes for the selected movie.
    b. Sends users to the Ticket Choice Page after they select a showtime.
5. Ticket Choice Page
    a. Lets users choose the number of tickets for the selected showtime.
    b. Connects to the Pay Page where users proceed to complete their purchase.
6. Pay Page
    a. Handles payment processing and allows users to enter payment details and apply any rewards.
    b. External services require cardholder information and then authenticate the purchase securely.
    c. Once authenticated, the user is sent back to the Pay Page and finally to the confirmation page.
    d. Integrates with external payment services:
        i. PayPal
        ii. Stripe
7. Confirmation Page
    a. Provides a receipt and final confirmation of the ticket purchase.

6.2 UML Diagram



1. Account
    a. Contains the general account information contained by all accounts as well as their general functions.
    b. Attributes:
        i. username: String - Contains an identifier created by the user to access their account.
        ii. password: String - Contains the password related to a specific username in order to access the account. The actual string will be the hashed and salted password used to verify when logging in.
        iii. email: String - Contains the email address of the account in order to send information or notifications to the user.
    c. Operations:
        i. login(password:String, username:String): Bool - Uses the username and password provided to check for and give access to an account.
        ii. getUsername(): String - Returns the username associated with the account as a String.
        iii. getEmail(): String - Retrieves the email address linked to the account as a String.

iv. setUsername(username:String) - Sets the account's username to the specified String.
v. setEmail(email:String) - Updates the account's email address with the provided String.
vi. setPassword(password:String) - Assigns a new password to the account using the specified String.

2. Management Account
    a. This account is used for accessing the Showing and Movie classes in order to change movies or showtimes.
    b. Attributes:
        i. moviesFile: String - Represents the file path or name where movie data is stored.
        ii. showingsFile: String - Contains the file path or name for storing showing data.
    c. Operations:
        i. addMovie(movie:Movie) - Adds a new Movie object to the system.
        ii. removeMovie(movie:Movie) - Removes the specified Movie object from the system.
        iii. modifyMovie(movieModified:Movie, movieOriginal:Movie) - Updates the details of an existing Movie object with the modified version.
        iv. addShowing(showing:Showing) - Adds a new Showing object to the schedule.
        v. removeShowing(showing:Showing) - Removes the specified Showing object from the schedule.
        vi. modifyShowing(showingModified:Showing, showingOriginal:Showing) - Updates the details of an existing Showing object with the modified version.
3. Customer Service Account
    a. This account is used for managing customer accounts.
    b. Attributes:
        i. accountsFile: String - Represents the file path or name where customer account info is stored.
    c. Operations:
        i. modifyUsername(customer:CustomerAccount, username:String) - Updates the username of the specified CustomerAccount to the new value provided.
        ii. modifyEmail(customer:CustomerAccount, email:String) - Changes the email address associated with the specified CustomerAccount to the new value.
        iii. addPoints(customer:CustomerAccount, point:Int) - Awards a specified number of points to the given CustomerAccount.
        iv. refundReceipt(customer:CustomerAccount, receipt:Receipt) - Processes a refund for the specified receipt associated with the given CustomerAccount.

v. deleteAccount(customer:CustomerAccount) - Permanently removes the specified CustomerAccount from the system.

4. Customer Account
    a. Contains information related to customers.
    b. Attributes:
        i. rewardsPoints: Int - The number of rewards points available to be used for purchases.
        ii. receipts: LinkedList<Receipt> - Contains a history of receipts from past purchases.
    c. Operations:
        i. addReceipt(receipt:Receipt) - Adds a new Receipt object to the customer's account.
        ii. get Receipts(): LinkedList<Receipt> - Retrieves a list of all Receipt objects associated with the customer.
        iii. removeReceipt(index:Int) - Removes the Receipt at the specified index from the customer's account.
        iv. addRewardsPoints(points:Int) - Increases the customer's rewards points by the specified amount.
        v. removeRewardsPoints(points:Int) - Decreases the customer's rewards points by the specified amount.
        vi. getRewardsPoints(): Int - Returns the total number of rewards points currently held by the customer.
        vii. setRewardsPoints(points:Int) - Sets the customer's rewards points to the specified value.
5. Showing
    a. Contains information related to the showing of a particular movie.
    b. Attributes:
        i. movie: Movie - Contains the movie information related to a particular showing.
        ii. date: String - Contains the date and time of the movie showing.
        iii. maxSeats: Int - Contains the total number of seats the theater for a particular showing can seat.
        iv. availableSeats: Int - Contains the number of seats available for customers to choose from.
        v. location: String - Contains the location of the movie theater where the showing will be held.
        vi. theaterNumber: Int - Contains the theater number of the current showing.
        vii. price: Double - Shows the price of the movie ticket for the particular showing.
    c. Operations:
        i. getMovie(): Movie - Returns the Movie object currently being shown.
        ii. getDate(): String - Retrieves the date of the movie showing as a String.
        iii. getMaxSeats(): Int - Provides the maximum number of seats available in the theater.

iv. getAvailableSeats(): Int - Returns the number of seats still available for the showing.
v. getLocation(): String - Retrieves the location of the theater as a String.
vi. getTheaterNumber(): Int - Provides the theater number where the movie is being shown.
vii. getPrice(): Double- Returns the ticket price for the movie showing as a Double.
viii. setMovie(movie:Movie) - Sets the Movie object for the current showing.
ix. setDate(date:String) - Assigns a date for the movie showing.
x. setMaxSeats(maxSeats:Int) - Updates the maximum number of seats available.
xi. setAvailableSeats(availableSeats:Int) - Adjusts the count of available seats for the showing.
xii. setLocation(location:String) - Sets the location of the theater.
xiii. setTheaterNumber(theaterNumber:Int) - Updates the theater number for the showing.
xiv. setPrice(price:Double) - Sets the ticket price for the movie showing.
xv. purchaseTicket(numberTickets:Int): Receipt - Processes the purchase of tickets and returns a Receipt.

6. Movie
    a. Contains information related to a particular movie.
    b. Attributes:
        i. name: String - Represents the title of the movie.
        ii. duration: Int - Stores the length of the movie in minutes.
        iii. thumbnail: String - Contains the URL or path to the movie's thumbnail image.
        iv. description: String - Holds a brief summary of the movie's plot or content.
        v. genre: String - Specifies the category or genre of the movie.
    c. Operations:
        i. getName(): String - Returns the name of the movie as a String.
        ii. getDuration(): Int - Retrieves the duration of the movie in minutes as an Integer.
        iii. getThumbnail(): String - Provides the URL or path of the movie's thumbnail image as a String.
        iv. getDescription(): String - Returns a brief description of the movie as a String.
        v. getGenre(): String - Retrieves the genre of the movie as a String.
        vi. setName(name:String) - Sets the name of the movie to the specified String.
        vii. setDuration(duration:Int) - Updates the duration of the movie with the provided Integer value.
        viii. setThumbnail(thumbnail:String) - Assigns a new thumbnail image URL or path for the movie.
        ix. setDescription(description:String) - Sets a new description for the movie using the specified String.

x. setGenre(genre:String) - Updates the genre of the movie with the provided String.

7. Receipt
    a. Contains information about purchases related to a particular showing.
    b. Attributes:
        i. showing: Showing - Represents the Showing object related to the receipt.
        ii. numberTickets: Int - Stores the total number of tickets purchased for the showing.
    c. Operations:
        i. getShowing(): Showing - Returns the Showing object associated with the receipt.
        ii. getNumberTickets(): Int - Retrieves the total number of tickets purchased as an Integer.
        iii. setShowing(showing:Showing) - Assigns a Showing object to the receipt.
        iv. setNumberTickets(numberTickets:Int) - Updates the number of tickets recorded on the receipt.
        v. calculatePrice(): Double - Calculates and returns the total price for the tickets purchased as a Double.

7. Timeline and Tasks

Here we will describe each task being done by each team member, and the estimated time it will take to complete each task.

Front Page

- Estimated 2 months to complete
- Developers:
    - James Johnson

Login Page

- Estimated 1 month to complete
- Developers:
    - Marvee Balyos

Sign Up Page

- Estimated 2 months to complete
- Developers:
    - Marvee Balyos

Movie Search Page

- Estimated 3 months to complete
- Developers:

    - James Johnson

Movie Showing Selection Page

- Estimated 4 months to complete
- Developers:
    - Seth Blanchard

Ticket Choice Page

- Estimated 2 months to complete
- Developers:
    - Marvee Balyos

Pay Page

- Estimated 1 month to complete
- Developers:
    - Marvee

Confirmation Page

- Estimated 1 month to complete
- Developers:
    - Seth Blanchard

Account Management Page

- Estimated 2 month to complete
- Developers:
    - Jack Roemer

Purchase Management Page

- Estimated 1 month to complete
- Developers:
    - Seth Blanchard

Account Class

- Estimated 1 month to complete
- Developers:
    - Jack Roemer

CustomerAccount Class

- Estimated 3 weeks to complete
- Developers:
    - Jack Roemer

CustomerService Class

- Estimated 3 weeks to complete
- Developers:
    - Jack Roemer

ManagmentAccount Class

- Estimated 1 month to complete
- Developers:
    - Jack Romer

Movie Class

- Estimated 2 weeks to complete
- Developers:
    - Marvee Balyos

Showing Class

- Estimated 2 weeks to complete
- Developers:
    - James Johnson

Receipt Class

- Estimated 1 week to complete
- Developers:
    - James Johnson

8. Testing

Link to test cases:
https://github.com/jfallz/MTTS/blob/master/testCasesSample.xlsx%20-%20TestCaseTemplate.pdf

8.1 Unit Tests

8.1.1 Login Method
Verify that the login function of the account class is functioning correctly and that incorrect login is not accepted.

- Instantiate an Account class object with username “User1”, password “Password123!”, email “test@gmail.com”
- Test with correct username “User1” and password “Password123!” should return true
- Test with incorrect username “User2” and correct password “Password123!” should return false
- Test with correct username “User1” and incorrect password “password123” should return false
- Test with incorrect username “User2” and incorrect password “password123” should return false

8.1.2 Set/Get Username

Verify that the set and get method for the username of the Account class are working correctly.

- Instantiate an Account class object with default constructor
- Set the username to “John” with the setUsername(username : String) method.
- Ensure that getUsername() returns “John”.

8.1.3 Set/Get Email

Verify that the set and get method for the email of the Account class are working correctly.

- Instantiate an Account class object with default constructor
- Set the email to “test@gmail.com” with the setEmail(email : String) method
- Ensure that getEmail() returns “test@gmail.com”

8.1.4 Set Password

Verify that the set method for the password of the Account class is working correctly.

- Instantiate an Account class object with default constructor
- Use setUsername(username : String) to set the username to “John”
- Use setPassword(password : String) to set the password to “Password123!”.
- Use the login method with the updated credentials and ensure it returns True.

8.1.5 Set/Get Movie Name

Verify that the set and get method for the movie name of the Movie class are working correctly.

- Instantiate a Movie class object with default constructor
- Set the name to “Movie Name” with setName(name : string)
- Use the getName() method and ensure it returns “Movie Name”

8.1.6 Set/Get Movie Duration

Verify that the set and get method for the movie duration of the Movie class are working correctly. Allowing positive values to be set while negative numbers are rejected.

- Instantiate a Movie class object with the default constructor
- Set the duration to 65 minutes using the setDuration(duration : int) method
- Use the getDuration() method and ensure it returns 65
- Set the duration to 1 minutes using the setDuration(duration : int) method
- Set the duration to -1 minutes using the setDuration(duration : int) method

- Use the getDuration() method and ensure it returns 1

8.1.7 Set/Get Thumbnail

Verify that the set and get method for the thumbnail of the Movie class are working correctly.

- Instantiate a Movie class object with default constructor
- Set the thumbnail using setThumbnail(thumbnail : String), where the String points to the path (or URL) to an image file.
- Use getThumbnail() and ensure the proper thumbnail location is returned.

8.1.8 Set/Get Description

Verify that the set and get method for the description of the Movie class are working correctly.

- Instantiate a Movie class object with default constructor
- Set the movie description to “Movie Description” using setDescription(description : String).
- Ensure that “Movie Description” is returned when calling getDescription()

8.1.9 Set/Get Genre

Verify that the set and get method for the genre of the Movie class are working correctly.

- Instantiate a Movie class object with default constructor.
- Set the movie genre to “Sci-Fi” using the setGenre(genre : String) method
- Ensure that “Sci-Fi” is returned when calling the getGenre() method

8.1.10 Set/Get Date

Verify that the set and get method for the date of the Showing class are working correctly.

- Instantiate a Showing class object with default constructor
- Set the date to “01-01-2000” using the setDate(date : String) function.
- Ensure that getDate() returns “01-01-2000”.

8.1.11 Set/Get Max Seats

Verify that the set and get method for the max number of seats of the Showing class are working correctly. Allowing positive values to be set while negative numbers are rejected.

- Instantiate a Showing class object with default constructor
- Set the max seats to 92 using the setMaxSeats(maxSeats: int)
- Ensure that getMaxSeats() method returns 92
- Set the max seats to 1 using the setMaxSeats(maxSeats: int)
- Set the max seats to -1 using the setMaxSeats(maxSeats: int)
- Ensure that getMaxSeats() method returns 1

8.1.12 Set/Get Available Seats

Verify that the set and get method for the available number of seats of the Showing class are working correctly. Allowing positive values to be set while negative numbers are rejected.

- Instantiate a Showing class object with default constructor
- Set the available seats to 42 using the setAvailableSeats(availableSeats : int)

- Ensure that getAvailableSeats() method returns 42
- Set the max seats to 1 using the setAvailableSeats(availableSeats: int)
- Set the max seats to -1 using the setAvailableSeats(availableSeats: int)
- Ensure that getAvailableSeats() method returns 1

8.1.13 Set/Get Location
Verify that the set and get method for the location of the Showing class are working correctly.
- Instantiate an object of the Showing class with the default constructor.
- Set the location to “Location 1” using the setLocation(location : String) method.
- Ensure that getLocation() returns “Location 1”

8.1.14 Set/Get Theater Number
Verify that the set and get method for the theater number of the Showing class are working correctly. Allowing positive values to be set while negative numbers are rejected.
- Instantiate an object of the Showing class with the default constructor.
- Set the theater number to 12 using the setTheaterNumber(theaterNumber : int) method.
- Ensure that getTheaterNumber() returns 12.
- Set the theater number to -1.
- Ensure that getTheaterNumber() returns 1.

8.1.15 Set/Get Price
Verify that the set and get method for the price of the Showing class are working correctly. Allowing positive values to be set while negative numbers are rejected.
- Instantiate a Showing class object with default constructor
- Set the price to 8.99 using the setPrice(price : double)
- Ensure that getPrice() method returns 8.99
- Set the price to 0.99 using the setPrice(price : double)
- Set the price to -0.99 using the setPrice(price : double)
- Ensure that getPrice() method returns 0.99

8.2 Integration Tests

8.2.1 Set/Get Movie
Verify that the set and get method for the movie object of the Showing class are working correctly.
- Instantiate a Showing and Movie class object with the default constructor.
- Set the Movie objects name to “Movie 1” using the setName(name : String) method
- Add the movie to the showing using the setMovie(movie : Movie) method.
- Use the getMovie() method to return the movie that is stored by the Showing object
- Use the getName() method of the returned movie and ensure that it matches “Movie 1”

8.2.2 Purchase Ticket Method

Verify that the purchase ticket method of the Showing class works correctly returning a ticket with the given information.

- Instantiate a Showing class object with the default constructor.
- Use the setPrice(price : double) method to set the Showing's price to 12.99
- Invoke the purchaseTicket(numberTickets : int) method for the instance of the Showing class with the parameter of 1.
- Use the getShowing() on the Recipt object to get the Showing object
- Use the getPrice() method on the Showing object and ensure that it returns 12.99

8.2.3 Set/Get Showing

Verify that the get and set method for the Showing object in the Receipt class is working correctly.

- Instantiate a Receipt and Showing class object with default constructor.
- Use the setShowing(showing : Showing) method to add the Showing object to the receipt.
- Ensure that the Showing object is returned using the getShowing method.

8.2.4 Customer Service Modify Username

Verify that the customer service class can modify a CustomerAccount's username.

- Instantiate a CustomerAccount with a default constructor
- Instantiate a CustomerServiceAccount with a default constructor
- use the modifyUsername(customer : CustomerAccount, username : String) method passing in the customer account that was created and the username "User2"
- Use the getUsername() method on the customer account to ensure that the username is now "User2"

8.2.5 Customer Service Add Points

Verify that the customer service class can add points to a CustomerAccount's number of rewards points.

- Instantiate a CustomerServiceAccount with the default constructor.
- Instantiate a CustomerAccount with the default constructor.
- Use the addPoints(customer : CustomerAccount, point : int) method to add 10 points to the instantiated customer account.
- Use the getRewardPoints() method on the customer account, and ensure it returns 10.

8.3 System Tests

8.3.1 Page Navigation

Verify that a user can navigate to all pages, and that they function together.

- Once on the website, navigate to the login page from the home page.
- Enter a username and password for the test account and press enter to navigate back to the home page.
- Navigate to the Movie Search Page and enter "Marvel" in the search bar and press enter
- Select the first movie available taking the user to the Movie Showing Selection Page

- Select the first available showing taking the user to the Ticket Choice Page
- Enter a choice of 1 ticket and press enter taking the user to the Confirmation page
- Select confirm taking the user to the Front Page
- Navigate to the Purchase Management Page
- Navigate back to the Front Page
- Navigate to the Account Management Page
- Navigate back to the Front Page
- Confirm all viewed pages were properly displayed.

8.3.2 Ticket Booking

Verify that a user can perform the entire movie booking and refund process, including account management without errors or crashes.

- Launch the movie ticketing system in a web browser.
- Login using valid user credentials.
- Search for a movie (e.g., "Inception").
- Select a showing for the movie (choose date, time, and seat).
- Proceed to payment, select a payment method, and confirm booking.
- Verify that a booking confirmation is displayed.
- Navigate to the "My Bookings" page and view the booked ticket.
- Initiate a refund request for the booked ticket and confirm the request.
- Navigate to the account management page and update the password.
- Log out and attempt to log in using the updated credentials.
- Check if the system reflects the refunded ticket and confirms that no charges remain.

8.3.3 Account Login

Verify that the login method correctly authenticates a user with valid credentials and rejects invalid ones.

- Create an instance of the Account class and set username to "testUser" and password to "securePassword".
- Call the login(String password, String username) method with the correct credentials ("securePassword", "testUser").
- Assert that the method returns true for the correct credentials.
- Call the login(String password, String username) method with incorrect credentials ("wrongPassword", "testUser").
- Assure that the method returns false for the incorrect credentials.

8.3.4 Payment using Stripe and PayPal

Verify that a correct payment method using Stripe or PayPal will result in a ticket purchase

- After confirming you wish to purchase a ticket on the Ticket Choice Page, click on either payment gateway.
- Enter your corresponding payment information into the fields provided.

- Ensure you do not have enough money to pay for the ticket, and click “Confirm Purchase”.
- Ensure that the payment is rejected within 3 seconds, and a notification notifying you to try again or use a different payment method appears.
- Repeat the test but with payment credentials (A random credit card number, etc) and ensure the results are the same.

8.3.5 Account Modification
Verify administrative access to user accounts and information through administrative accounts.
- In the login page create a CustomerServiceAccount.
- In the login page create a default CustomerAccount.
- Using the CustomerServiceAccount, call the modifyCustomerAccount method.
- Check that the information has been changed using the Customer Account.

8.3.6 Declined payment using Stripe or PayPal
Verify that a declined payment using Stripe or PayPal will result in an error
- After confirming you wish to purchase a ticket on the Ticket Choice Page, click on either payment gateway.
- Enter your corresponding payment information into the fields provided.
- Click "Confirm Purchase"
- Ensure that after the payment is authorized, the ticket receipt appears on the screen. The dollar amount specified on the receipt has been withdrawn from the given payment method.

9 Data Management Strategy

Since our system requires storing sensitive user information and transactional data, a robust data management strategy is crucial.

9.1 Database Choice: SQL (PostgreSQL)
- Rationale: PostgreSQL was chosen for its scalability, performance, and consistency.
- Structure:
    - A relational schema with structured data.
    - Tables such as Users, Transactions, Movies, and Showtimes

9.2 Design Decisions:
- Primary Database: For core functionalities like accounts, showtimes, seats, and basic transaction history.
- Secondary Database: A secondary database for logging transaction history would improve the audit trail if necessary.
- User Table: Stores account status (customer service, administrator, customer), sensitive information, points, and links to transactional data.
- Movies and Showtimes: Organized separately to allow querying based on date, time, location, names, genre, etc…
- Transactions: Holds each purchase, linking to both user and showtime tables.

9.3 Alternatives and Trade-offs

- Non SQL alternatives
    - Although SQL is great for handling data transactions, it isn't necessary. A "noSQL" database such as MongoDB could easily handle the schemas our system requires, but may lack consistency which is vital for payments.
- Trade-offs
    - PostgreSQL offers very strong transactional support, and has few flaws. However, it is more complex than other alternatives, so we will need well-trained employees and staff to ensure proper usage of the database.

9.4 Data Security Measures:

- Encryption: All sensitive data is encrypted at rest and in transit, meeting all compliance requirements. Furthermore, this ensures if the database is breached, no sensitive data is obtained.
- Access Control: Administrative data should require proper authentication, preventing bad actors from entering the sensitive 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