Movie Ticketing System

Software Requirements Specification

3.0.1

10/23/25

Group #20
Dee Edwards, Nester Lomeli, Trevor Brown

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

Revision History

Date	Description	Author	Comments
<date>	<Version 1>	<Your Name>	<First Revision>
09/25/25	v 1.0.0	Dee Edwards Nester Lomeli Trevor Brown	First Revision: Introduction, Use Cases, Functional Requirements, Non-Functional Requirements
10/12/25	v 2.0.0	Dee Edwards	Second Revision: some formatting clean-up; added UML class diagram; added Software Architecture Diagram
10/23/25	v 2.1.0	Dee Edwards	Added detailed descriptions for UML class diagram and Software Architecture Diagram
10/23/25	v 3.0.0	Dee Edwards Nester Lomeli Trevor Brown	Test Plan, Test Cases
10/23/25	V 3.0.1	Dee Edwards Nester Lomeli Trevor Brown	Assignment 3 Submission, all group member's commits are linked.

Document Approval

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

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

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this SRS is to outline and document the functional and non-functional requirements to develop a Movie Ticketing system useable by our client Movie Theaters and their customers. This document is intended for an audience of Movie Theater operators to understand and review each major component of the system

1.2 Scope

This document details the specifications for a Movie Ticketing System. The primary purpose of this product is to provide an online marketplace where movie-goers can easily purchase tickets from their local theaters. To achieve this end, the product will also need to have an alternate interface designed for Movie Theater operators to maintain a database of movie showings without technical knowledge.

1.3 Definitions, Acronyms, and Abbreviations

Admin(s)	Business owners and operators of Movie Theaters and their Customer Support; Admins may interface with developers during the design and creation of the system, but must ultimately be able to use the system on their own when the product is ready to ship.
Customer(s)	Patrons of the Movie Theaters which do business with the Theater using this Movie Ticketing System.
User(s)	Comprised of both clients and customers, both groups must be able to access the system in separate ways in order for the system to facilitate commerce between the two parties.

1.4 References

This subsection should:
(1) Provide a complete list of all documents referenced elsewhere in the SRS, or in a separate, specified document.
(2) Identify each document by title, report number - if applicable - date, and publishing organization.
(3) Spec ify the sources from which the references can be obtained.
This information may be provided by reference to an appendix or to another document.

1.5 Overview

This subsection should:
(1) Describe what the rest of the SRS contains
(2) Explain how the SRS is organized.

2. General Description

2.1 Product Perspective

- Customer and in-person hardware
- Chrome web browser to view our product from customer's and in-person hardware
- Stripe for third party payment system

2.2 Product Functions

The product will ask the customer to select movie, showtime, and seat before asking for login information if not already logged in. If all previous steps have been completed, they will go to payment options.

2.3 User Characteristics

Typical movie-going audience; Biggest demographic is likely pre-teens and teens. Customers are looking for entertainment and are primed to be sold on an experience.
Admins are knowledgeable, focused on solving problems and maintaining a streamlined and understandable service.

2.4 General Constraints

Third party payment system, Stripe

2.5 Assumptions and Dependencies

Assumptions

- Hardware should be able to run Chrome web browser

- All cards will be accepted for payment
- Hardware should run on Windows or IOS

Dependencies

- Depends on movie data base
- Depends on admin

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.1.1 Customer-facing User Interface.
3.1.1.2 Client-facing User Interface.

3.1.2 Hardware Interfaces

Should be able to print tickets at in person theater. All other hardware handled by browser and OS

3.1.3 Software Interfaces

3.1.4 Communications Interfaces

Stripe 3rd party payment processing

3.2 Functional Requirements

3.2.1 Browse Movies and Showtimes

3.2.1.1 Introduction

This function allows customers to browse movies, view details, and check available showtimes
3.2.1.2 Inputs
- Customer request (web or kiosk) to view movie listings
- Available filters: data, time, format, language
3.2.1.3 Processing
- system queries movie and showtime database
- Filters are applied to narrow results
- Seat availability is retrieved each showtime
3.2.1.4 Outputs
- A list of movies, with details: title, rating, duration, description, poster image
- Showtimes and seat availability

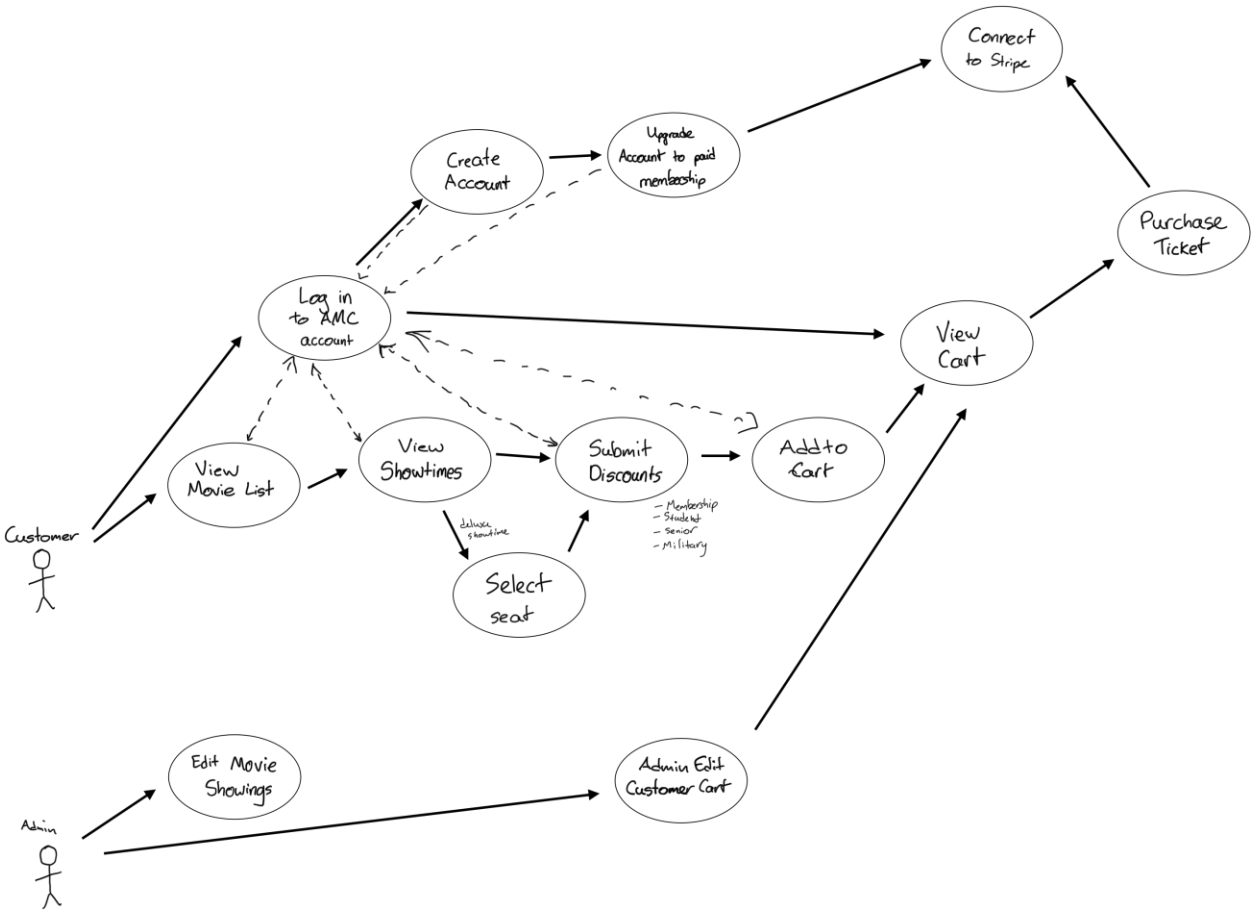
3.2.1.5 Error Handling
- if no showtimes match the criteria, display “No showtimes available”
- if database query fails, display an error message and retry option.

3.2.2 Ticket Purchase

3.2.2.1 introduction

This function allows customers to select seats, purchase tickets, and confirm transactions
3.2.3.3 Processing
- system checks ticket validity (correct showtime, not expired, not already deemed).
- valid tickets are marked as redeemed in the database
3.2.3.4 Outputs
- Confirmation message “Ticket Redeemed”
- Printed stub (if kiosk prints tickets
- Staff view updated attendance record
3.2.3.5 Error Handling
- If ticket is invalid, system displays error “invalid or already used ticket”
- if scanner fails, allow manual entry of confirmation number

3.3 Use Cases



3.4 Classes / Objects

3.4.1 Customer

3.4.1.1 Attributes

- Name
- Email
- Phone Number
- Password
- Stripe Account

3.4.1.2 Functions

- Add to cart
- Select Movie
- Select Showtime
- Login
- Create Account
- Upgrade Account
- Check Out Cart
- Sign in to Stripe

3.4.2 Admin

3.4.2.1 Attributes

- Name
- Password

3.4.2.2 Functions

- Edit Movie Database
- Access & Edit Customer Cart

3.5 Non-Functional Requirements

3.5.1 Performance

95% of transactions shall be processed in less than a second.
Updates as infrequent as possible to reduce system downtime.

3.5.2 Reliability

Service should be able to support at least 1,000 active users simultaneously.

3.5.3 Availability

Accessible on customer devices via Chrome web browser and on-site at theaters using ticketing booths running modified Chrome web browsers.
Service available through 5 minutes of user inactivity (during purchasing process).
purchasing process will be available two weeks prior to movie date and until 10 minutes after showtime start.

3.5.4 Security

Payment methods must be securely encrypted; this can be outsourced to Stripe

Admin interface must be securely password protected in order to prevent tampering with the client's business.
Customer interface must be securely password protected in order to prevent use of customer's saved data, including Stripe login info and any tickets the customer has purchased and not used.
All stored passwords must be securely encrypted to AES-128 standard.

3.5.5 Maintainability

Database should be maintainable by an admin with no technical

3.5.6 Portability

Browser-Based Web App prioritizing compatibility with Chrome, with Firefox added as an additional priority if time and budget allow.

3.6 Inverse Requirements

- The system shall not store raw credit card information in its servers
- The system shall not allow ticket redemption more than 30 minutes after the showtime begins
-The system shall not allow duplicate seat assignments for the same showtime

3.7 Design Constraints

Specify design constrains imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.

3.8 Logical Database Requirements

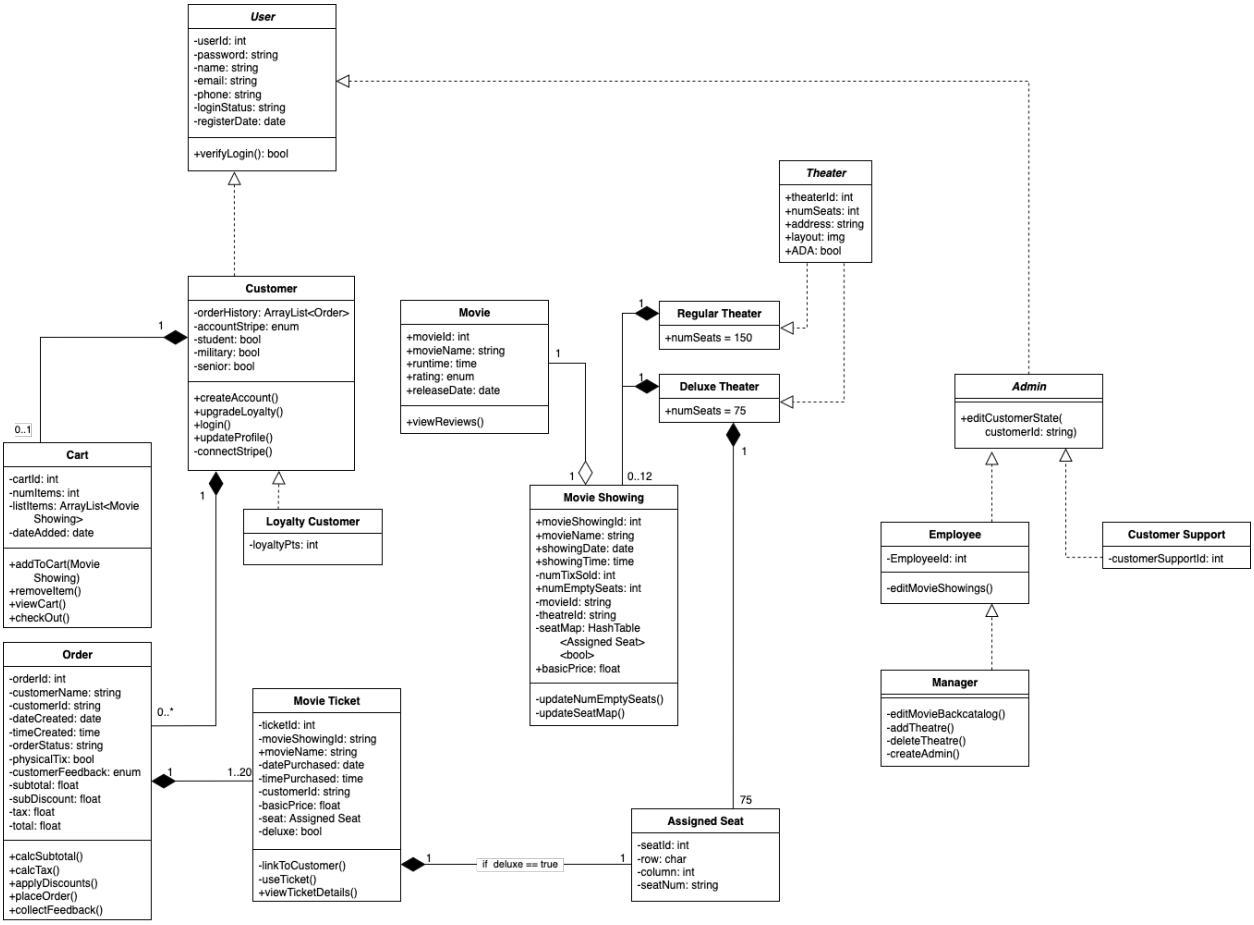
Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.
-Relational database is required to store movies, showtimes, customers, orders, and tickets
-Each ticket record shall include: order ID, showtime ID, seat number, purchase time stamp and redemption status
- Database shall retain transaction history for at least 5 years for auditing and reporting

3.9 Other Requirements

Catchall section for any additional requirements.

4. Analysis Models

4.1 UML Class Diagram



4.1.1 *User Class*

User is an abstract class which provides the general structure of attributes for all system login accounts including customers and admins. It outlines the storage of userIds, passwords, names, email addresses, phone numbers, registration date, and login status. It also provides an operation for verifying credentials at login.

4.1.1.1 Admin Class

Admin is an abstract subclass of *User* which provides the structure for all user accounts which have access to the system's backend. It adds an operation that allows all admins to edit various elements of a customer's cart and checkout process when a userId attribute corresponding to a Customer object is passed into the customerId parameter.

4.1.1.1.1 Employee Class

Employee is a subclass of *Admin* which includes an attribute for storing an EmployeeId (separate from userId inherited from *User*) and an operation for editing movie showings. Objects of this class may correspond to lower-level theater employees.

4.1.1.1.1.1 Manager Class

Manager is a subclass of Employee which adds more operations for the accounts of higher-level theater employees. These operations allow theaters to be added or removed, new admin accounts to be created, and the backlist of movies representing the company's library to be edited.

4.1.1.1.2 Customer Support Class

Customer Support is a subclass of *Admin* intended to give outsourced customer support providers access to *Admin's* operations. It adds the customerSupportId (separate from userId inherited from *User*) to keep track of these outsourced agents.

4.1.1.2 Customer Class

Customer is a subclass of *User* representing customers and making up the majority of user accounts and therefore *User* subclass objects. It includes attributes for keeping track of a customer's order history and Stripe payment account information, as well as military, student, and senior statuses. It includes operations to create the account, edit the account information, upgrade the account to a paid loyalty rewards account, login, and connect to a Stripe payment account. A Customer object (or subclass object) may or may not compose a singular cart object and as many order objects as the customer purchases.

4.1.1.2.1 Loyalty Customer Class

Loyalty Customer is a subclass of Customer representing customers who pay for the loyalty rewards program. The only thing it adds to the Customer class is an attribute for tracking loyalty points, since regular Customer objects already keep track of order history to calculate these points.

4.1.2 *Theater Class*

Theater is the abstract class to create objects of individual theater rooms. It includes attributes for a theaterId to keep track of each theater, the number of seats in the theater, the theater's address (location), the theater's seat layout (in image form), and ADA compliance.

4.1.2.1 Regular Theater Class

Regular Theater is the subclass of theater meant for non-deluxe theaters. Each new object's number of seats is automatically set to 150. Each Regular Theatre object may compose 0-12 Movie Showing objects.

4.1.2.2 Deluxe Theater Class

Deluxe Theater is the subclass of theater meant for deluxe theaters. Each new object's number of seats is automatically set to 75. Each Deluxe Theater object may compose 0-12 Movie Showing objects. Each Deluxe Theatre object composes 75 Assigned Seat objects.

4.1.3 Movie Class

Movie is the class which stores data for each Movie in the company's film catalogue/library. Each object contains attributes detailing the movie's name, runtime, rating (G, PG, R, etc.), and release date, as well as a movieId to keep track of each movie. The class also contains an operation to view the movie's reviews from rotten tomatoes. A copy of a Movie object may be aggregated within a Movie Showing object.

4.1.4 Movie Showing Class

Movie Showing is the class which represents each unique showing of each movie in each individual theater. A movieShowingId attribute keeps track of each unique object. The class also contains attributes storing the name and movieId of the movie being shown, the date of the showing, the time of the showing, the number of tickets sold, the theater's theaterId, and the

basic price (undiscounted) of a ticket. The class also contains a seatMap attribute that is only used for deluxe theaters. Its operations include updating the seatMap and updating the number of empty seats. Each Movie Showing object must be composed within either a Regular Theater object or a Deluxe Theater object. The seatMap attribute and operation are only used in the case of composition within a Deluxe Theater object. Each Movie Showing object also must aggregate a copy of a Movie object.

4.1.5 Cart Class

Cart is the class with which customers (through UI) may find and organize their ticket selections into a list to purchase. The class has attributes storing the number of items, the date the cart was added, a list of movieShowingIds, and a cartId to keep track of each cart object. The class includes operations to add Movie Showings, remove Movie Showings, view the cart, and check out. The checkOut() operation moves the customer through the process of submitting discounts, paying, collecting their tickets (or a receipt to collect a physical ticket), and more. By the end of the operation, all the data stored in the Cart object is finalized and transferred into a new order Object, and the old Cart object is subsequently deleted. Each Car object must be composed within a Customer object.

4.1.6 Order Class

Order is the class which records completed customer purchases by collecting data from a Cart object and its checkOut() operation. The orderId attribute keeps track of each unique order object in the database. It also has attributes detailing the name of the customer who placed the order, the customer's userId, the date and time of the order's placement, the status of the order, whether the ticket is physical or digital, and even customer feedback. Additionally, it has more attributes detailing the costs involved including the subtotal, the subtotal after discounts, the tax, and the total order cost. Among the class's operations are calculating the subtotal, calculating the tax, applying discounts, calculating the order total, collecting feedback, and placing the order (The last two are called by the checkOut() operation of Cart after the new Order object is created but not yet finalized). Each Order object must be composed within a Customer object, and each Order object must compose between 1 and 20 Movie Ticket objects.

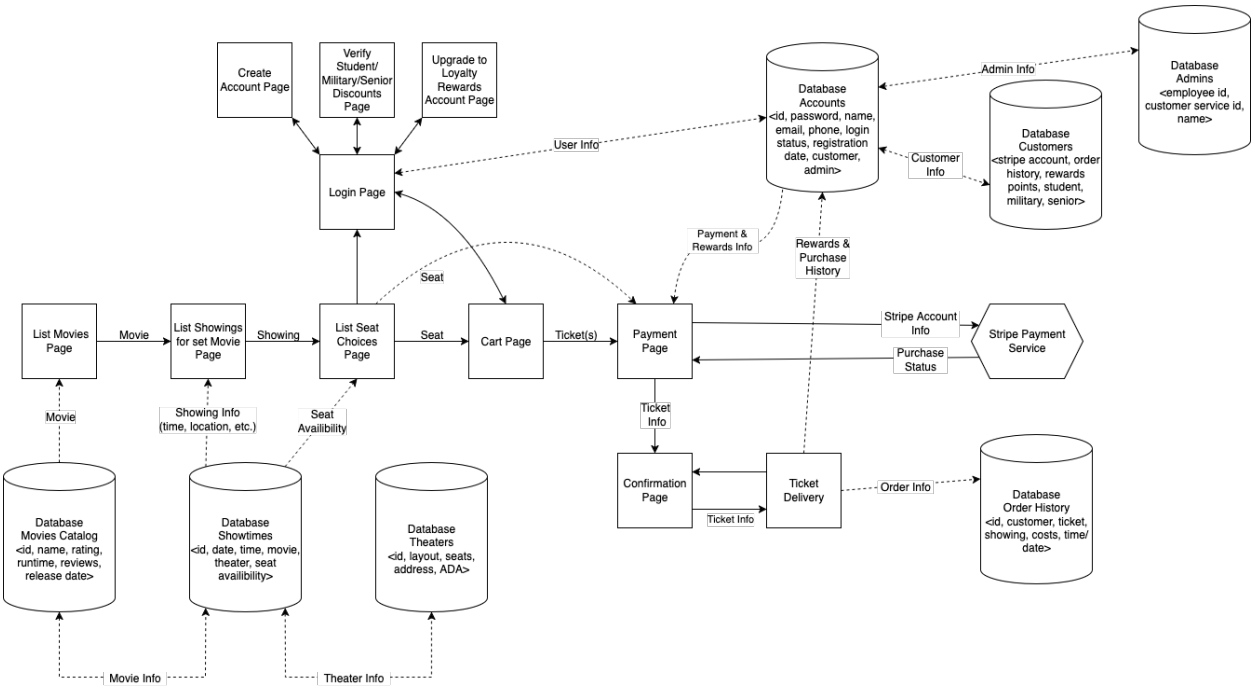
4.1.7 Movie Ticket Class

Movie Ticket is the class whose objects represent each unique digital movie ticket (or receipt for physical ticket). The ticketId attribute keeps track of each unique object of this class. Other attributes in this class include the associated movieShowingId, the name of the movie being shown, the date and time of the ticket's purchase, the basic (undiscounted) price of the ticket, the alphanumeric name of the potentially associated assigned seat, and whether or not the ticket is for a showing in a deluxe theater. The class has operations for linking it to a customer (can only be done once), viewing its details, and using it to enter a theater (can only be done once). Each Movie Ticket object must be composed within an order. If the ticket is for a deluxe showing, then the object must compose an Assigned Seat object.

4.1.8 Assigned Seat Class

Assigned Seat is the class which keeps track of each assigned seat in a deluxe theatre. The class's attributes include a seatId to track each unique object, the row of the seat, the column of the seat, and the complete alphanumeric name of the seat. Each Assigned Seat object is composed within a Deluxe Theater object and a Movie Ticket object.

4.2 Software Architecture Diagram



The List Movies Page is the main introductory page of the system, providing an eye-catching overview of what is offered. It pulls data from the Movies Catalog Database in order to list available movies and their details. When the customer selects a movie, they are moved to the List of Showings Page associated with that movie.

The List of Showings Page pulls data from the Showtimes Database in order to provide the customer with options of when to see the movie they have selected. When they select a specific one of these Showings, they are taken to the List Seat Choices Page associated with that Showing.

The List Seat Choices Page pulls data from the Showtimes Database to display information about the theater and its available seat options. In the case of a Showing in a Regular Theater, this page simply acts as a sparse navigation page, displaying a message informing the customer that no seat choice is necessary. This page will have the option to click to traverse to the Login Page. After selecting seat(s) (or immediately in the case of Regular Theater Showings), the customer can click to continue to the Cart Page with their Seat Information.

The Login Page, accessible via the Seat Selection Page or Cart Page, pulls user info from the Accounts Database to allow customers to login. The Login Page is also associated with three ‘subpages’ to which customers can easily traverse back and forth from this Login Page. The Create Account Page allows customers to create an account. The Verify Discounts Page allows users to provide evidence of status as a student, senior, or military member/veteran in order to add discounts to all purchases with the account. The Upgrade Page allows customers to pay to upgrade their account to a Loyalty Rewards Account. After logging in, besides the three pages already discussed, the customer can also move forward to the Cart Page with a click.

The Cart Page shows the customer's selections (if any) of Movie Showtime(s) and Seat(s), pulling data from the previous page. Upon checkout, the system turns these selections into Movie Ticket(s) for the customer to access once the system moves them into the Payment Page.

The Payment Page combines data from the previous Cart Page with data from the Accounts Database in order to allow the customer to access their saved Stripe Account data to pay for their order (with all discounts applied). This Page uses said saved Account data to access the Stripe Payment Processor servers, which either deny or confirm successful payment. Upon receiving confirmation, the system moves the customer and their Ticket Info to the Confirmation Page.

The Confirmation Page allows customers to see all details of their confirmed order, as well as access the Ticket Delivery Page with a click.

The Ticket Delivery Page allows customers to receive either their digital ticket or a receipt for a physical ticket. This page then sends all relevant Purchase and Rewards Info to the Accounts Database and sends all Order Info to the Order History Database. This Page then allows the customer to return to the Confirmation Page or exit to the main menu.

The Showtimes Database has two-way communication with both the Movies Catalog Database and the Theaters Database in order to have and pass on complete information regarding each Showing.

The Accounts Database has two-way communication with both the Customer Database and Admin Database, which allows the Accounts Database to have access to all information to all types of accounts while specific information related to each of the two types of accounts are kept separated.

4.3 Sequence Diagrams

4.4 Data Flow Diagrams (DFD)

4.5 State-Transition Diagrams (STD)

5. Test Plan

5.1 Purpose

This section describes the approach used to verify and validate the movie ticketing system. The goal is to ensure that all system features operate correctly, efficiently, and in accordance with the requirements defined in this SRS.

5.2 Scope

This section describes the approach used to verify and validate the Movie Ticketing management, checkout and payment, discount verification, and ticket delivery. The testing process includes unit, functional, and system levels to ensure coverage across all components.

5.3 Test Strategy

A bottom-up testing strategy will be used:

Unit **Testing:** Focus on individual classes and methods (e.g, MovieShowing*,* AssignedSeat*,* StripePaymentService*)*
Functional Testing: Validate the integration between modules such as browsing, seat selection, and checkout.
System Testing: Confirm the full user flow from movie selection to payment confirmation and ticket redemption.

Test data will include multiple movie entries, showtimes, deluxe seat maps, and verified student accounts to test discounts.

5.4 Test Environment

Testing will be performed using Google Chrome (version 128+) on Windows 11 and theater kiosk environments.
A seeded relational database will be used with mock or sandbox Stripe keys for payment testing.

5.5 Test Case Overview

A total of ten (10) test cases are provided in testCasesSample.xlsx, covering:
Unit Tests: Seat updates, filtering functions, and payment decline handling.
Functional Tests: Movie browsing, seat map selection, and discount application.
System Tests: End-to-end purchase flow, duplicate seat prevention, and late ticket redemption.

5.6 Exit Criteria

All High-priority tests must pass.
Medium-priority tests must achieve at least 90% success.
No critical defects should remain open upon completion.

5.7 Design Update Summary

Added PaymentGateway interface and StripePaymentService implementation for isolated payment testing.
Clarified that seatMap functionality applies only to Deluxe theaters.
Added a database constraint to prevent duplicate seat assignments.

5.8 Repository Information

All testing materials will be stored in the group GitHub repository under:
(_____)
This folder includes:
TestPlan.md – the written plan.
testCasesSample.xlsx – all detailed test cases.

5.9 Commit and Link Information

Each member must make at least one commit for this assignment.
Provide commit links and repository URL below:

GitHub Link: https://github.com/sunryze28/movie-ticketing-CS250-G20
Member Commits:
Member 1: Dee Edwards: https://github.com/sunryze28/movie-ticketing-CS250-G20/commit/480162c6f47c9ab475f6c10dc9063184aafd65fd
Member 2: Trevor Brown: https://github.com/sunryze28/movie-ticketing-CS250-G20/commit/8ebe68a6236440db9447d82a079200bdf5f24219
Member 3: Nester Lomeli: https://github.com/sunryze28/movie-ticketing-CS250-G20/commit/a6b052f33063aaeb061a16ad25e93c2ce5d2a6da

6. Change Management Process

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

A. Appendices

Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.

Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.

A.1 Appendix 1

A.2 Appendix 2