Movie Ticketing System

Software Requirements Specification

v4.0

September 25th, 2025

Group 5
Samuel Jenkins, Gregory Larson, Shawyan Razavi

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

Revision History

Date	Description	Author	Comments
09/22/25	Version 1	Shawyan Razavi, Gregory Larson, Samuel Jenkins	First Revision
10/09/25	Version 2	Shawyan Razavi, Gregory Larson, Samuel Jenkins	Second Revision
10/23/25	Version 3	Shawyan Razavi, Gregory Larson, Samuel Jenkins	Third Revision
11/06/25	Version 3	Shawyan Razavi, Gregory Larson, Samuel Jenkins	Fourth Revision

Document Approval

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

Signature	Printed Name	Title	Date
	Samuel Jenkins	Software Eng.	11/06/2025
	Gregory Larson	Software Eng.	11/06/2025
	Shawyan Razavi	Software Eng.	11/06/2025
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

Revision History **2**
Document Approval **2**
1. Introduction **5**
1.1 Purpose 5
1.2 Scope 5
1.3 Definitions, Acronyms, and Abbreviations 5
1.4 References 5
1.5 Overview 6
2. General Description **6**
2.1 Product Perspective 6
2.2 Product Functions 6
2.3 User Characteristics 7
2.4 General Constraints 7
2.5 Assumptions and Dependencies 7
3. Specific Requirements **7**
3.1 External Interface Requirements 7
3.1.1 User Interfaces 7
3.1.2 Hardware Interfaces 7
3.1.3 Software Interfaces 8
3.1.4 Communications Interfaces 8
3.2 Functional Requirements 8
3.2.1 Purchasing Tickets 8
3.2.2 Account Creation/Management 8
3.2.3 Administrative access 9
3.3 Use Cases 9
3.3.1 Use Case #1: Purchase Tickets 9
3.3.2 Use Case #2: Create Account 10
3.3.3 Use Case #3: Manage Account 10
3.3.4 Use Case #4: Edit Orders 11
3.3.5 Use Case #5: View Sales Data 12
3.3.6 Use Case #6: Manage Theaters 12
3.3.7 Use Case Diagrams 14
3.4 Non-Functional Requirements 16
3.4.1 Performance 16
3.4.2 Reliability 16
3.4.3 Availability 16
3.4.4 Security 16
3.4.5 Maintainability 16
3.4.6 Portability 16
4. Design Specifications **17**
4.1 Architecture 17
4.1.1 Architecture Diagram 17

4.1.2 Architecture Diagram Descriptions Table *17*
4.2 Data Management Strategy *19*
4.2.1 Entity Relationship Diagram *20*
4.2.2 Data Dictionary *20*
4.2.3 Explanation of Architectural Diagram *25*
4.3 Class Specifications *26*
4.3.1 UML Class Diagram *26*
4.3.2 Theater *26*
4.3.3 Showtime *27*
4.3.4 Ticket *27*
4.3.5 Order *28*
4.3.6 UserInfo *28*
4.3.7 Customer *29*
4.3.8 SysAdmin *29*
5. Development Plan ***30***
5.1 Partitioning of Tasks *30*
5.2 Team Member Responsibilities *30*
5.3 Development Timeline *30*
6. Test Plan ***31***
6.1 Testing Procedure *31*
6.2 What We're Testing *31*
6.3 Tying Back to Our Design *31*
6.4 Testing Goals *32*
A. Appendices ***32***

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification document is to detail a list of requirements that the software in question must fulfil, including hardware and software requirements, functional and non-functional requirements, and use-case examples that must be implemented in the final product. This document is to be used by *HyperStudio* developers and employees to aid in the development of the requested software.

1.2 Scope

This SRS document is written with the intent to produce a Movie Ticketing System for our client, San Diego Cinemas, to be used by both employees and management.
The system will:

- Allow customers to purchase tickets online in the customer's browser of choice as well as in-person at a register with the assistance of an employee.
- Provide an up-to-date movie list and showtimes for each theater owned by San Diego Cinemas.
- Allow customers to purchase tickets as early as two weeks before a scheduled showtime and up to ten minutes after the movie has started.
- Generate digital tickets (with QR codes) for online purchases and print tickets at the register.
- Provide management-level users with tools to:
    - Add, edit, and cancel showtimes.
    - Assign movies to specific theaters.
    - Monitor seat availability and sales data in real time.
    - Manage user access (employees vs. managers).

The scope of this software is limited to ticketing and scheduling functions, ensuring the system focuses on delivering secure and efficient ticket purchasing and management capabilities.

1.3 Definitions, Acronyms, and Abbreviations

San Diego Cinemas: The client, abbreviated *SDC*.
Movie Ticketing System: The system, abbreviated *MTS*.
API: Application Programming Interface.
Administrator: User who has elevated privileges in the system and in the client's hierarchy.
HTTPS: Hypertext Transfer Protocol Secure

1.4 References

There are no references in this document.

1.5 Overview

Section 2 describes the system's general purpose and context surrounding its design. It includes the product's general functionality, the system's intended users, and any constraints or assumptions that must be considered in development.

Section 3 details the functional and non-functional requirements of the system. This section also includes information about any external interfaces, intended use-cases, and intended classes that will allow the system to function.

2. General Description

2.1 Product Perspective

The Movie Ticketing System will be an independent and self contained software product to be developed for *San Diego Cinemas*. The MTS will serve as the primary platform for movie ticket sales and movie scheduling across all SDC theater locations in the greater San Diego area.

This system will be a web based application that can be accessed via a compatible browser with an internet connection. The system must also support the kiosk terminals that are physically located inside each theater.

2.2 Product Functions

The MTS will provide the following primary functions:

- **Customer Functions**
    - Browse movies, showtimes, and auditoriums.
    - Select seats and purchase tickets through the web portal or kiosks.
    - Enforce purchase rules (maximum 20 tickets per order, sales open 14 days before showtime and close 10 minutes after start).
    - Receive unique tickets with a QR code digitally (via email) or physically (printed at kiosk).
    - Access discount options such as student, senior, or veteran tickets.
    - Submit/view feedback through a customer feedback interface.
    - View movie reviews and critic quotes pulled
- **Administrative Functions**
    - Add, edit, and/or cancel showtimes.
    - Assign movies to theaters and configure seating layouts.
    - Monitor real time seat availability and ticket sales.
    - Manage user roles (customers, staff, admins).
    - Access reports on sales and system activity.

2.3 User Characteristics

Our intended users include customers who wish to buy tickets from the theater, as well as administrators who need to edit customers orders on request, view ticket sales data, and manage their theaters.

For potential customers, if they are interfacing with the system online, they should understand how to use an internet browser and have a valid email address. These requirements are not needed if the customer is purchasing tickets in-person at a kiosk. In both cases, they must be able to provide valid payment to the theater.

For administrators, they must be able to operate the system on an administrative level, which can be covered by a short training session with our designers.

2.4 General Constraints

The system's customer-facing interface must be easy to use via a touch screen and adaptable to both phone and desktop browsers. The system must also be able to work with the client's existing hardware systems, currently running Windows 11.

2.5 Assumptions and Dependencies

The kiosks and administration computers are assumed to be able to connect to the internet and run a chromium based secure browser for compatibility. Furthermore, it is assumed that all credit card payments are accepted. The seats for each theatre must all be the same, either standard or deluxe

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The GUI for this system will be a responsive browser-based application with a customer-facing website that can be displayed on both mainstream customer browsers as well as touch-screen kiosks in-person. The other section will be an Administrator-facing application that allows direct management of the database after passing a secured login.

3.1.2 Hardware Interfaces

The system must be able to be run on and interface with existing kiosk hardware to facilitate the ticket purchasing process. This includes kiosk-specific cash readers, card readers, and ticket printers. In addition, the administrator side of the system must be able to run on a standard Windows computer, with access to a standalone cash register, card reader, and ticket printer.

3.1.3 Software Interfaces

The system will be able to verify given card payment information with the client's existing connections to payment processors. The system will also handle the automatic distribution of email messages to users of the system via the client's existing email messaging system, with some configuration to meet the new ticketing standard. In addition, the system must handle administrative logins with a third-party two-factor authentication system, like DUO.

3.1.4 Communications Interfaces

The system will communicate via standard internet protocol between the customer's internet browser and the website. Hosting will be handled by the existing hosting architecture owned by the client, SDC. The system will also communicate internally within each theater, with connections between the kiosks being routed to a main administration computer, who then accepts the data and pushes it to the database for record keeping.

3.2 Functional Requirements

3.2.1 Purchasing Tickets

- The system shall allow users to select an available movie and showing, along with having the option to choose a theater with deluxe seating, if one is showing the selected movie
- Upon approval of purchase, a unique ticket is created
- The system shall output either to a printer to give a physical copy of the ticket, or to an email address to give a digital copy.
- The system shall automatically cancel any ticket selection if payment is not received within 5 minutes of being selected
- The system shall always offer a senior discount and a student and veteran discount on weekdays.
- The system shall allow buying a ticket up to 2 weeks in advance of the showing
- The system shall only allow up to 20 tickets to be purchased at once
- The system shall ask for feedback via a smiley face rating system

3.2.2 Account Creation/Management

- The system shall allow customers to create an account by providing a valid email address, password, first and last name, and date of birth.
- The system shall require customers to agree to terms and conditions before an account is created.
- The system shall send a confirmation email upon account creation.
- The system shall allow customers to log in to their account using their email and address.
- The system shall allow customers to view and update their personal information.
- The system shall lock a customer's account after 5 failed login attempts within 10 minutes, and then require a password reset through email verification.
- The system shall accommodate membership features (discounts, rewards, monthly subscriptions)
- The system shall allow the user to permanently delete their account upon request via email confirmation

3.2.3 Administrative access

- The system shall allow the administrators to set show times and change theaters as needed
- The system shall allow administrators to override customer errors in purchasing tickets
- The system shall upload the ticket sales and cost per ticket to the database

3.3 Use Cases

3.3.1 Use Case #1: Purchase Tickets

Goal: This use case allows a customer who wishes to watch a movie to purchase tickets through the website or via an in-person kiosk.
Preconditions: The movie the customer wants to see has available seating, and the customer has a valid payment method. Online customers must have a valid email address.
Success End Condition: The customer receives tickets via email/through the kiosk printer.
Failed End Condition: The tickets are abandoned and available to other customers to purchase.
Primary Actor: TheaterCustomer
Trigger: The TheaterCustomer visits the website or taps the in-person kiosk screen to start the process.
Description:

1. The TheaterCustomer is prompted to log in to their account with the theater company. This step is optional and can be skipped.
2. If the TheaterCustomer is in-person at a kiosk, skip the next two actions and continue with action 3. The in-person TheaterCustomer will be assumed to be buying tickets for the theater at which the kiosk is located.
    a. The online TheaterCustomer is allowed to browse movies being shown in the company's theaters within the next two weeks. The TheaterCustomer must pick a movie to watch.
    b. The online TheaterCustomer must pick a theater that is showing the movie they want to watch.
3. Both types of TheaterCustomers must now pick the showtime for their movie at the theater they chose.
4. The TheaterCustomer will select the number of tickets they wish to purchase, divided into Regular and Senior tickets.
    a. Optionally, Student and Veteran ticket prices are available on weekday showtimes.
5. If the theater that was chosen has deluxe seating, the TheaterCustomer must choose which seats they would like to buy tickets for via a theater floor diagram.
6. At this point, the TheaterCustomer is allowed to enter any other valid discounts before continuing. This step is optional.
7. If the TheaterCustomer is online, they must now provide an email address
8. The TheaterCustomer must confirm their choice on a review page.
9. The TheaterCustomer must provide valid payment.

    a. If the TheaterCustomer is using a kiosk, they may insert cash or use a valid credit card. This step will be handled by the kiosk hardware. The software will wait until the kiosk hardware sends confirmation on a valid purchase.
    b. If the TheaterCustomer is online, they must enter their credit card information, which will be handled by a trusted third party payment processor.
10. The system will wait until payment is confirmed by the payment processor.
11. The system will then distribute tickets in accordance with the TheaterCustomer's method of access. The system has completed its task after this step.
    a. Kiosk TheaterCustomers will receive physical printed tickets, with each ticket being unique to the customer.
    b. Online TheaterCustomers will be sent an email that contains a scannable code to confirm their purchase at the theater.

3.3.2 Use Case #2: Create Account

Goal in Context: This use case allows a customer to create a customer account with the theater company to save their information.
Preconditions: The user has a valid email address.
Success End Condition: The user registers an account with the theater and their information is saved.
Failed End Condition: No account is created.
Primary Actor: TheaterCustomer
Trigger: The TheaterCustomer clicks on a Create Account option on the website.
Description:

1. The TheaterCustomer must enter the following information:
    a. A valid email address
    b. A password that meets our security requirements
    c. Their first and last name
    d. Their birthday
2. The TheaterCustomer must agree to our website's terms and conditions.
3. The system interacts with the database to create the new account according to the user's information.
4. The TheaterCustomer receives confirmation on the website and via email that the account was created.
5. The TheaterCustomer must verify the account via the email that was sent.
    a. If the user does not verify the account, the system will delete the account in 24 hours.

3.3.3 Use Case #3: Manage Account

Goal in Context: This use case allows a theater customer with a valid account to set up a monthly payment to the company for access to membership benefits.
Preconditions: The user must have a registered and verified account with the theater company, as well as a valid credit card.

Success End Condition: The user is billed monthly (30 days) and gains access to membership benefits defined by the company.
Failed End Condition: The membership is not added to the account.
Primary Actor: TheaterCustomer
Trigger: The TheaterCustomer presses the Join Membership button on the website.
Description:

1. The TheaterCustomer must login to their registered account.
2. The system asks the TheaterCustomer if they would like to join the membership program. To continue, the TheaterCustomer must allow the system to continue.
3. The TheaterCustomer confirms the amount to be billed monthly, starting from the sign-up date.
4. The TheaterCustomer must enter their credit card information, which will be handled by a third party payment processor.
5. The payment processor will validate the information given.
6. If successful, the payment processor will start the monthly billing process.
7. The system will give the user's account membership status.
8. The TheaterCustomer will receive confirmation via the website and via email that their account has joined the membership program.

3.3.4 Use Case #4: Edit Orders

Goal in Context: This use case allows a user with administrator access to edit existing orders on the database, with the system being able to refund customers or reprint tickets and having these changes reflect on the ticket sales data.
Preconditions: The user must have administrator access.
Success End Condition: The user's changes to the edited orders are applied to the database. If necessary, refunds are issued and/or tickets are reprinted/sent.
Failed End Condition: The database remains as it is.
Primary Actor: Administrator
Trigger: The Administrator opens the administrator page for the system.
Description:

1. The Administrator must login via a two-factor authentication system.
2. The Administrator goes to the Edit Recent Orders page to facilitate their edits.
3. The Administrator is presented with a list of orders in the past three weeks.
4. The Administrator searches for the order they wish to edit and selects it.
5. The Administrator applies any changes they wish to make to the order. This can include:
    a. Changing given tickets
    b. Changing payment method
    c. Changing the showtime that was ordered
    d. Refunding the order
6. The Administrator confirms the changes to be made to the order.
7. The system verifies the changes, rejecting any invalid changes (such as offering more tickets than there are seats).
8. The system then applies the changes to the order, saving the previous order's state as a record attached to the order.

9. If changes to the tickets were made, such as seating, pricing, or quantity, the system will print out the new tickets at the in-person counter.

3.3.5 Use Case #5: View Sales Data

Goal in Context: This use case allows a user with administrator access to view sales data from the beginning of the system to the time of access.
Preconditions: The user must have administrator access.
Success End Condition: The user is able to view the sales data that the system has accumulated.
Failed End Condition: The user is unable to view sales data.
Primary Actor: Administrator
Trigger: The Administrator opens the administrator page for the system.
Description:

1. The Administrator must login via a two-factor authentication system.
2. The Administrator goes to the View Sales Data page.
3. The system needs to bring up all of the sales data in an easy-to-read page.
4. Optionally, the Administrator can set parameters to the data that the system brings up, repeating step 3 with the specified parameters.

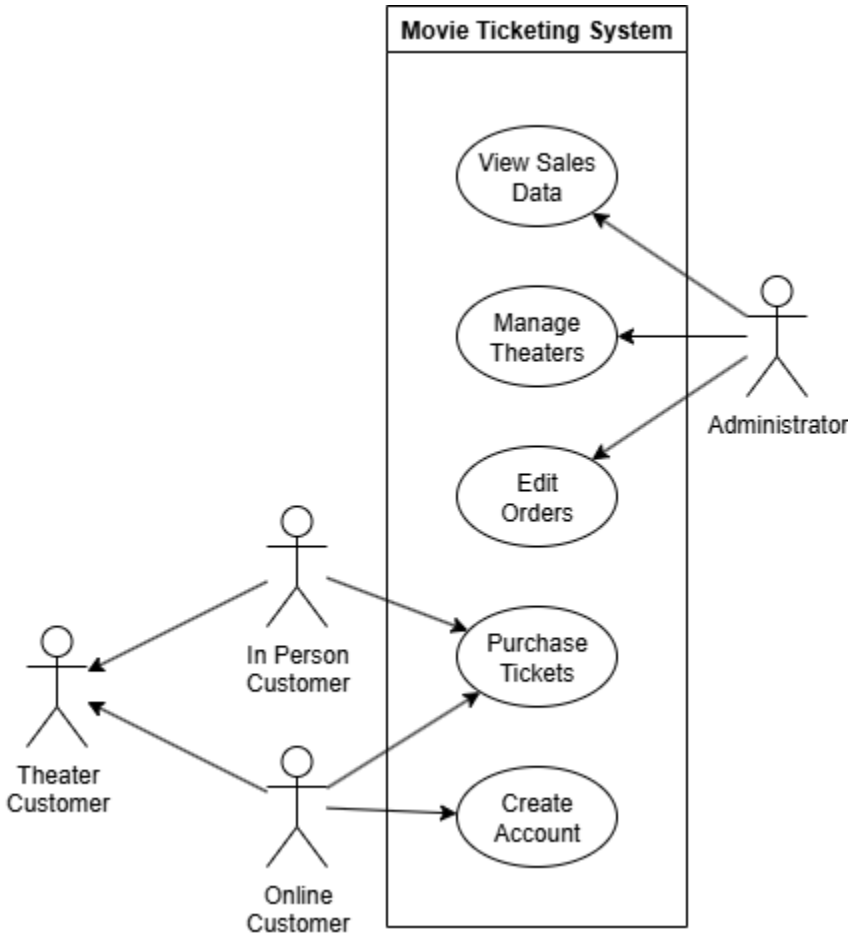
3.3.6 Use Case #6: Manage Theaters

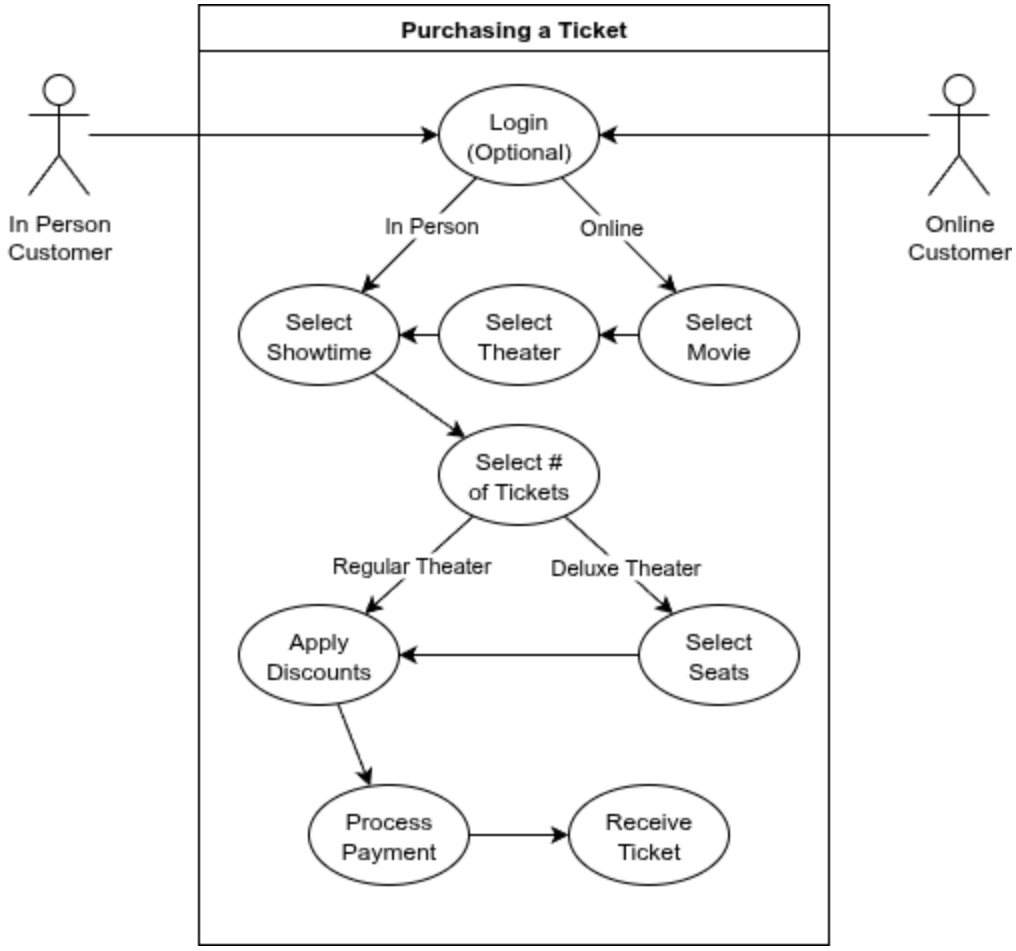
Goal in Context: This use case allows a user with administrator access to set a theater's movie, ticket count, and showtimes on a theater by theater basis.
Preconditions: The user must have administrator access.
Success End Condition: The user's changes to the theater database are pushed, and the changes are shown on the customer end of the website.
Failed End Condition: The database remains as it is.
Primary Actor: Administrator
Trigger: The Administrator opens the administrator page for the system.
Description:

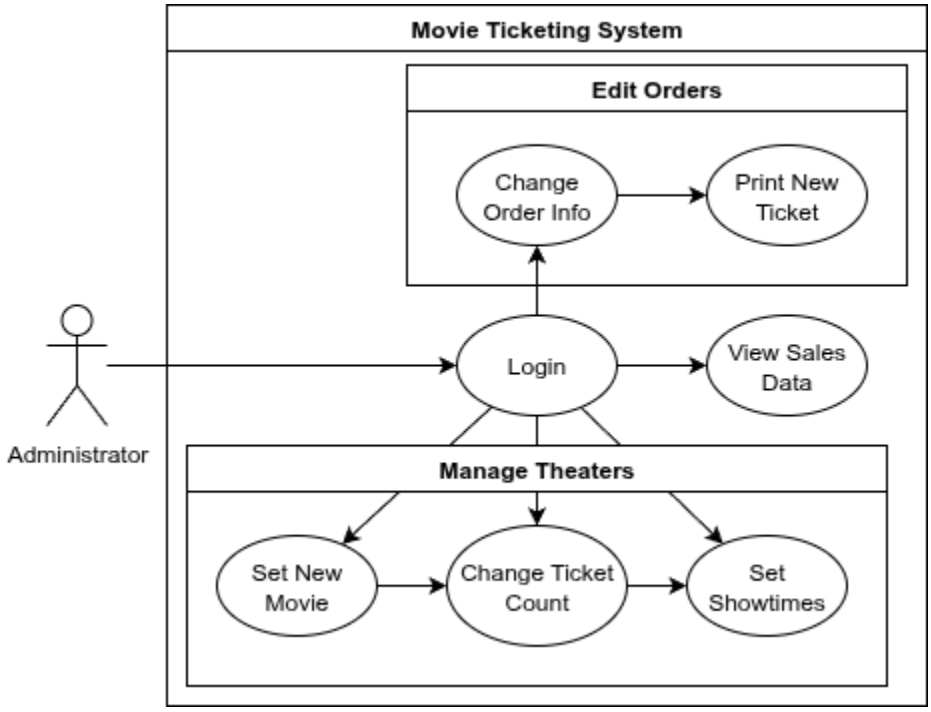
1. The Administrator must login via a two-factor authentication system.
2. The Administrator goes to the Manage Theaters page.
3. The Administrator must select which theater to make changes to.
4. The system brings up the information in that theater's partition of the database.
5. The Administrator may now choose to Set Showtimes, Change Ticket Count, or Set a New Movie for the theater they are managing.
    a. If the Administrator chooses to Set Showtimes, the system requests from the Administrator which showtimes should be sold for the customers.
    b. If the Administrator chooses to Change Ticket Count, the system requests the new ticket count for the theater, then proceeds to the step Set Showtimes.
    c. If the Administrator chooses to Set a New Movie, the system will ask which movie to display to the customers.
        i. The system will then update the database with the new movie, including a review pull from review aggregate websites via an API. The system will then return to the step Change Ticket Count.

6. The above step will repeat until the Administrator chooses a different theater or ends the session.

3.3.7 Use Case Diagrams




Purchasing a Ticket
In Person Customer
Online Customer
Login (Optional)
In Person
Online
Select Showtime
Select Theater
Select Movie
Select # of Tickets
Regular Theater
Deluxe Theater
Apply Discounts
Select Seats
Process Payment
Receive Ticket


Movie Ticketing System
Edit Orders
Change Order Info
Print New Ticket
Administrator
Login
View Sales Data
Manage Theaters
Set New Movie
Change Ticket Count
Set Showtimes

3.4 Non-Functional Requirements

3.4.1 Performance

Up to 1000 users will be able to use MTS simultaneously.

3.4.2 Reliability

MTS will have minimal updates, with the goal of a system uptime rate of 99.9% in a 7-day period.

3.4.3 Availability

Any chromium-based browser will be compatible with MTS, as well as Firefox and Microsoft Edge.

3.4.4 Security

The system will deny access to the database directly, only allowing modifications via the Administrator gateway or via its own system functionality.

3.4.5 Maintainability

An administrator will be able to access and change the calendar of movies, what movies are available, and the showings.

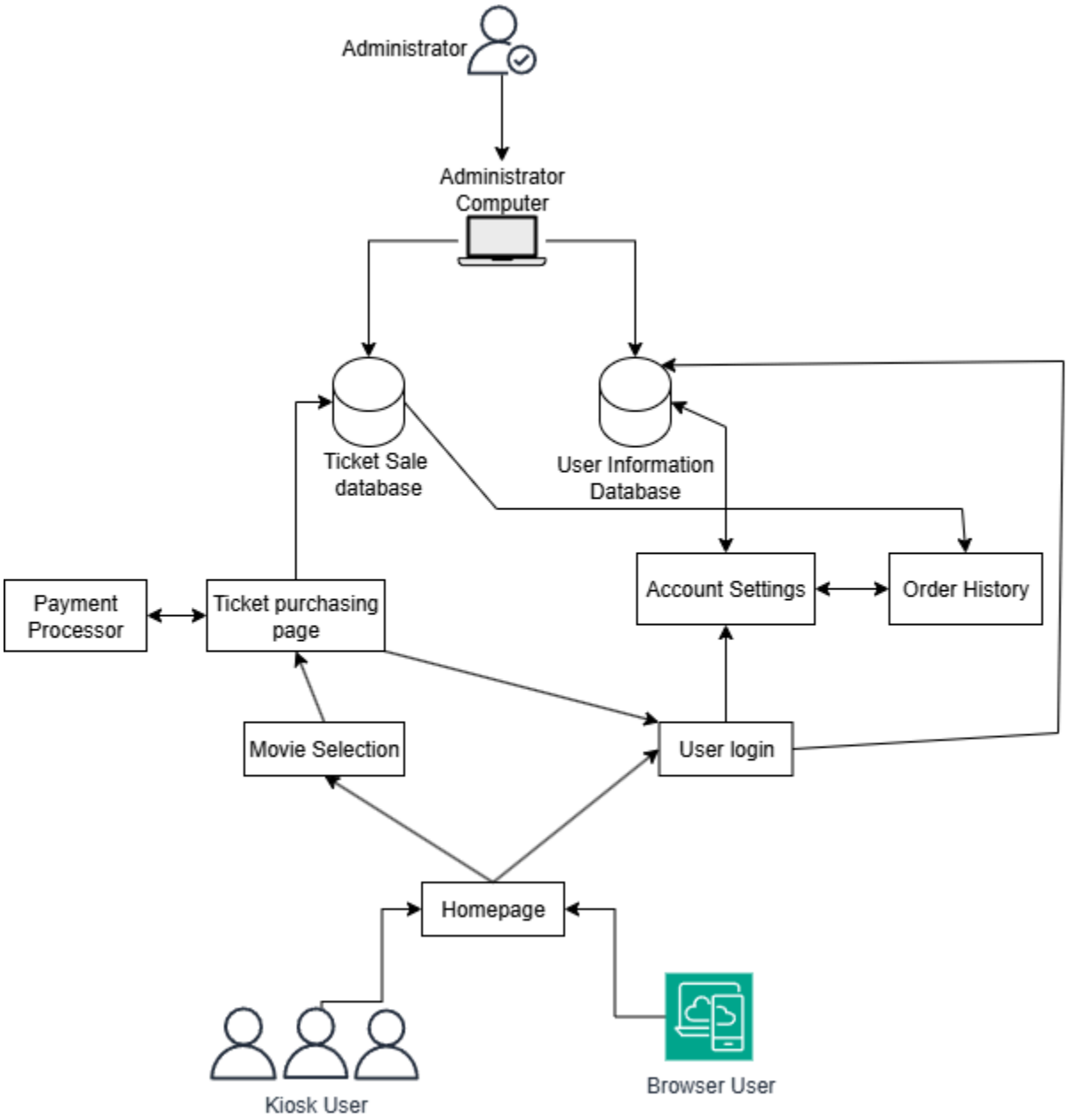
3.4.6 Portability

The MTS customer-facing website will be accessible from any laptop, desktop, or phone that has internet access. We do not plan to allow Administrator access from any system other than verified computer systems inside the theaters themselves.

4. Design Specifications

4.1 Architecture

4.1.1 Architecture Diagram



4.1.2 Architecture Diagram Descriptions Table

Kiosk User	Any person purchasing a ticket from a kiosk inside a theatre. They are taken to the Homepage upon startup.
Browser User	Any person purchasing a ticket online. They are taken to the

	Homepage upon accessing the URL.
Homepage	The initial webpage that is visited upon requesting the website address. Accessible via Kiosk or home browser.
Movie Selection	Webpages related to selecting movies and showtimes. Users are allowed to navigate between the various listings. Listings are pulled from the database as the user navigates.
User login	On navigating to this page, the user is prompted to login to the website to access their account information. The inputs on this site are cross-referenced with the information stored on the User database.
Payment Processor	This represents a third-party payment processor's API. Any payments that are made across the website will be pushed through to this API. A third-party HTML embed may be used to handle payments.
Ticket Purchasing page	Upon selecting a showtime from the Movie Selection webpages, the user is directed to this series of pages to begin the ordering process. Data will be pulled from the database and written back upon confirmation. This page will be timed for 15 minutes before redirecting back to the Movie Selection page.
Account Settings	This page primarily displays items related to the logged-in user's account and allows changes to be made to their personal information, including memberships. Information is pulled from and sent to the User database.
Order History	Shows the history of orders done on an account. Accessed through account settings, and can return to account settings. Information is pulled from the Ticket Sale database.
Ticket Sale Database	Holds all the information stored on purchased tickets.
User Information Database	Holds all the information stored for each unique account. Is accessed by account settings and is updated by account settings
Administrator Computer	The device to access the databases of ticket information and account information.

Administrator	The owner of the chain, or a manager of a theatre. Has access to the databases of ticket information and account information.

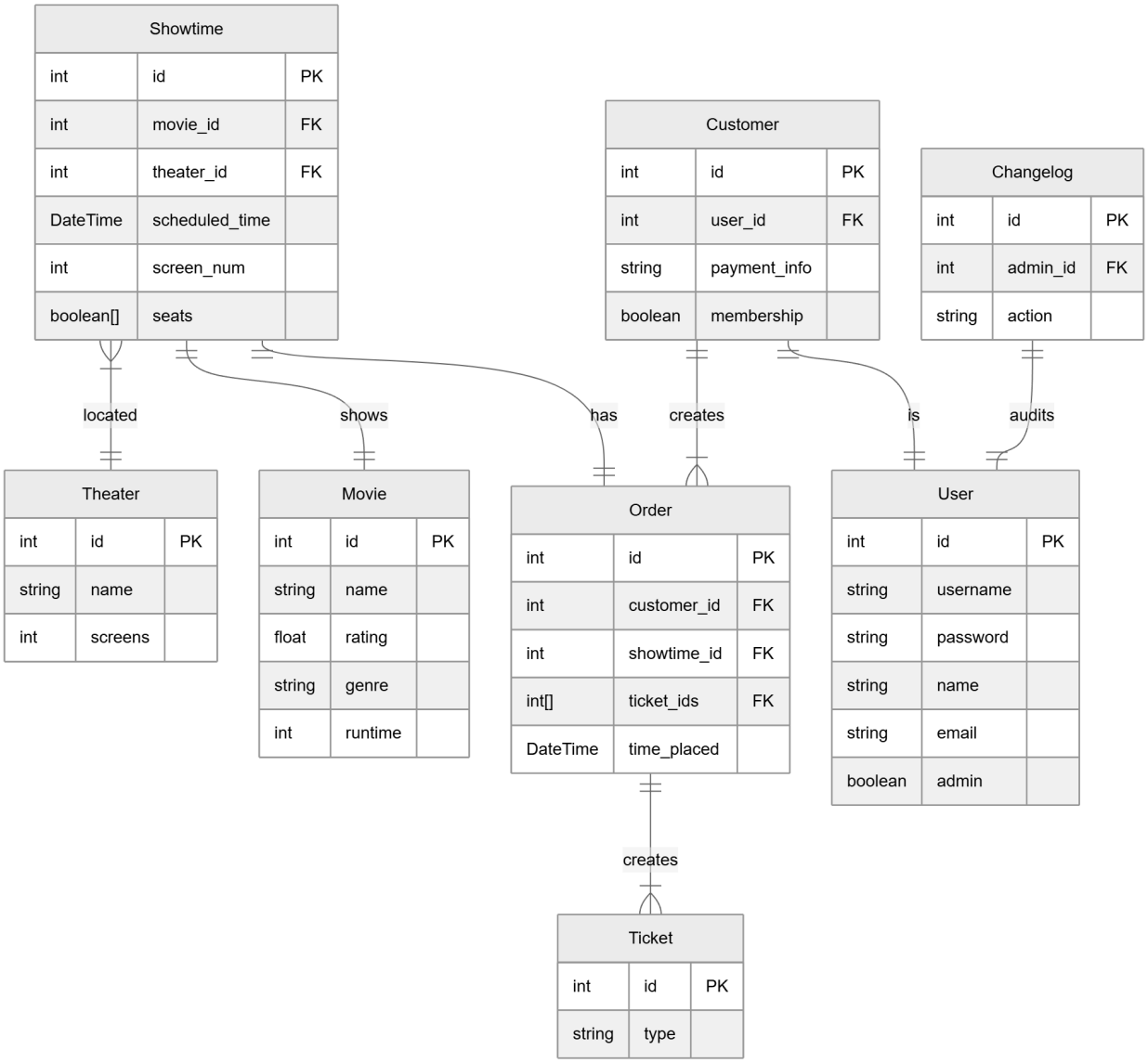
4.2 Data Management Strategy

We will be using two SQL databases linked together as linked servers, written with MySQL. Each database will be stored on separate servers, splitting the data between movie-related data and user-related data.

Our considered NoSQL alternatives were Cassandra and MongoDB. For Cassandra, we considered that while we could utilize the extra headroom on efficiency if the database was large enough, we had reason to believe that the resulting database will not be large enough to override the benefits of using MySQL. For MongoDB, we found that web integration would be an easier task if we used this framework, but due to using sensitive information like payment information, as well as not being as compatible with a transactional method of management, and not expecting a need for flexibility in data volume, we settled on an SQL database.

To handle the data, each entity in the database will be split into two groups. Group 1 will hold all the movie-related information: Showtime, Theater, Movie, Order and Ticket. Group 2 will hold all the user-related information: User, Customer, and the Changelog. The data has been split in this way to allow for minimal downtime for maintenance and to allow space for vertical growth on each database.

4.2.1 Entity Relationship Diagram



4.2.2 Data Dictionary

Showtime	Description: Showtime for a single time and screen Type: Table Primary Key: id Foreign Keys: movie_id, theater_id
id	Description: Unique identifier Type: Attribute/Integer Length: 6 Range: 000000 - 999999

movie_id	Description: ID of movie to be shown Type: Attribute/Integer Length: 6 Range: 000000 - 999999
theater_id	Description: ID of the theater where the showtime is at Type: Attribute/Integer Length: 6 Range: 000000 - 999999
scheduled_time	Description: The time of the specific showing Type: Attribute/DateTime
screen_num	Description: Physical number of screen inside theater Type: Attribute/Integer Length: 2 Range: 00 - 99
steats	Description: Array of flags to mark occupied seating Type: Attribute/Boolean Array Length: 0 - 300 Range: True/False for each index
Theater	Description: A physical theater Type: Table Primary Key: id Foreign Keys: None
id	Description: Unique Identifier Type: Attribute/Integer Length: 6 Range: 000000 - 999999
name	Description: Name of the theater Type: Attribute/String Length: 50 Allowed Characters: Alphanumerical
screens	Description: Number of screens at this theater Type: Attribute/Integer Length: 2 Range: 00 - 99

Movie	Description: A movie that is currently in theaters Type: Table Primary Key: id Foreign Keys: None
id	Description: Unique Identifier Type: Attribute/Integer Length: 6 Range: 000000 - 999999
name	Description: Name of the movie Type: Attribute/String Length: 50 Allowed Characters: Alphanumerical
rating	Description: Critic rating from review aggregate sites Type: Attribute/Float Length: 3 Range: 00.0 - 10.0
genre	Description: The movie's genre Type: Attribute/String Length: 20 Allowed Characters: Alphanumerical
runtime	Description: Movie runtime in minutes Type: int Length: 3 Range: 000 - 999
Order	Description: The order a customer is placing for tickets Type: Table Primary Key: id Foreign Keys: customer_id, showtime_id, ticket_ids
id	Description: Unique Identifier Type: Attribute/Integer Length: 6 Range: 000000 - 999999
customer_id	Description: ID for the customer that placed the order Type: Attribute/Integer Length: 8 Range: 00000000 - 99999999

showtime_id	Description: ID for the requested showtime Type: Attribute/Integer Length: 6 Range: 000000 - 999999
ticket_ids	Description: Array of IDs for the purchased tickets Type: Attribute/Integer Array Length: 0 - 20 Range: 000000 - 999999 for each index
time_placed	Description: Time the order was confirmed by payment processors Type: Attribute/DateTime
Ticket	Description: The uniquely created ticket for an order Type: Table Primary Key: id Foreign Keys: None
id	Description: Unique Identifier Type: Attribute/Integer Length: 6 Range: 000000 - 999999
type	Description: What type the ticket is Type: Attribute/String Length: 20 Allowed Characters: Alphabetic
User	Description: Holds the information for a specific user Type: Table Primary Key: id Foreign Keys: None
id	Description: Unique Id for each registered user Type: Attribute/Integer Length: 8 Range: 00000000 - 99999999
username	Description: Username for an account Type: Attribute/String Length: 50 Allowed Characters: Alphanumeric

password	Description: Password to the account Type: Attribute/String Length: 50 Allowed Characters: Alphanumeric and special characters
name	Description: The legal name of the user Type: Attribute/String Length: 50 Allowed Characters: Alphabet
email	Description: The email for the user Type: Attribute/String Length: 50 Allowed Characters: Alphanumeric and @
admin	Description: Flag for if this user has Admin privileges Type: Attribute/Boolean Length: 1 Range: True/False
Customer	Description: Customer information for Users who are not Admins Type: Table Primary Key: id Foreign Keys: user_id
id	Description: Unique Identifier Type: Attribute/Integer Length: 8 Range: 00000000 - 99999999
user_id	Description: Unique for the user Type: Attribute/Integer Length: 8 Range: 00000000 - 99999999
payment_info	Description: Customer's payment information Type: Attribute/String Length: 16 Allowed Characters: Alphanumeric and special characters
membership	Description: Boolean for if the customer has the membership Type: Attribute/Boolean Range: True/False

Changelog	Description: A history of the changes to the database Type: Table Primary Key: id Foreign Keys: admin_id
id	Description: Unique Identifier Type: Attribute/Integer Length: 6 Range: 000000 - 999999
admin_id	Description: User ID for the admin who made the change Type: Attribute/String Length: 8 Range: 00000000 - 99999999
action	Description: Description of the change Type: Attribute/String Length: 250 Allowed Characters: Alphanumeric, punctuation

4.2.3 Explanation of Architectural Diagram

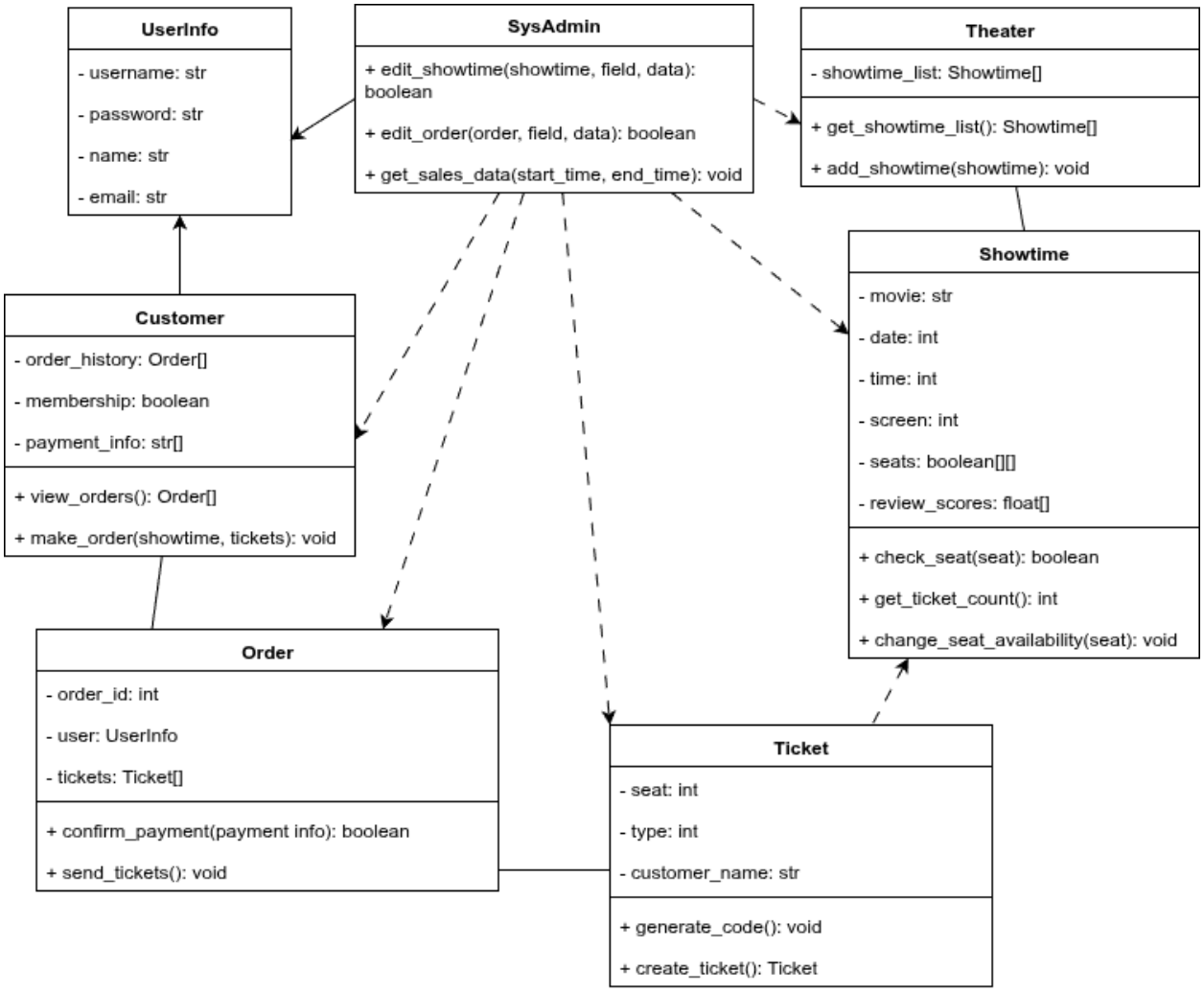
The architectural diagram illustrates the data structure and relationships within the Movie Ticketing System.

- The User entity stores login and role information, while the Customer table extends it with payment and membership data.
- Customers create Orders, which generate Tickets tied to specific Showtimes.
- Each Showtime links a Movie and a Theater, storing details like time, screen number, and seat availability.
- The Movie table holds information such as title, genre, rating, and runtime, and Theater defines each cinema's name and number of screens.
- The Changelog entity tracks administrative actions, linking back to users with admin privileges to ensure system accountability.

Overall, the design enforces data integrity through primary and foreign key relationships, separates user data from transactional data for security, and maintains flexibility for future scalability. The structure supports consistent seat management, secure transactions, and efficient organization of user, movie, and booking information.

4.3 Class Specifications

4.3.1 UML Class Diagram



4.3.2 Theater

Each instance of Theater holds the showtimes for a specific theater location in San Diego. The Theater class also holds functions to get the list of showtimes for that theater, as well as add more showtimes to that specific theater.

Attributes	Private showtime_list: Showtime[] ● Stores a list of the showtimes have been set up by the SysAdmin class.
Operations	Public get_showtime_list(): Showtime[] ● Getter function for the list of showtimes, mainly to be used for displays. Public add_showtime(showtime): void ● Adds a Showtime to the list of showtimes stored by this instance of Theater. Must be pushed to the database.

4.3.3 Showtime

Each Showtime created holds all the details necessary to display to the user: movie name, showtime date, time, and screen, the seats in the theater, and review scores for the movie being shown. Methods include a seat-checking method, a ticket-counting method, and a method to change seat availability status.

Attributes	Private movie: str ● The name of the movie. Private date: int ● The day on which the movie is being held. Store date as YYYYMMDD. Private time: int ● The time that the movie is being shown. Store time as HHMMSS. Private screen: int ● The screen in the theater associated with this showtime. Private seats: boolean[][] ● A 2d array of seats represented as boolean values. True represents an empty seat, false represents a full seat. Private review_scores: float[] ● Review scores retrieved from the review site APIs.
Operations	Public check_seat(seat): boolean ● Returns the specified seat value. Public get_ticket_count(): int ● Returns the number of seats available for this showtime. Public change_seat_availability(seat): void ● Allows Orders and SysAdmin to edit the seat values directly.

4.3.4 Ticket

Each Ticket is made with a seat number, type number, and an associated customer name for front door operations. This class holds a method to create scannable codes for front door operations, as well as its own creation method to be called by Orders.

Attributes	Private seat: int ● The seat number associated with this ticket. Used in print/sending. Private type: int ● The type of ticket, stored as an integer and starting from 0 as a Regular ticket, increasing to define the ticket type. Used to determine price. ○ 0 = Regular ○ 1 = Senior ○ 2 = Child ○ 3 = Student

	○ 4 = Veteran Private customer_name: str ● The name associated with this ticket. Used in print/sending.
Operations	Public generate_code(): void ● Uses an existing library to generate a scannable code. Public create_ticket(): Ticket ● Creates a new ticket, primarily used by the Order class.

4.3.5 Order

Each Order has its own ID for auditing and is associated with a specific user via the UserInfo class as well as a list of Tickets ordered. The Order class has methods to confirm payments (with hooks into our payment API) and send tickets associated with this order, either to a printer or the customer via email.

Attributes	Private order_id: int ● A unique identifier for each order. Private user: UserInfo ● A reference to the UserInfo that started the order. Private tickets: Tickets[] ● A list of tickets that was created by this Order.
Operations	Public confirm_payment(payment info): boolean ● Send a request to the payment API for payment confirmation and processing. Returns true if the payment was successful. Public send_tickets(): void ● Creates a set of files for each ticket, then sends them out via email/prints them at the kiosk if available.

4.3.6 UserInfo

UserInfo serves as a parent class to both Customer and SysAdmin. Its sole purpose is to hold the information to login, as well as the baseline data that every user of the system will have, including username, password, name, and email address.

Attributes	Private username: str ● The username for this user. Private password: str ● The password for this user. Private name: str ● The name of the user. Private email: str ● The email address registered with this user.

Operations	None.

4.3.7 Customer

If a user is a customer, their UserInfo will be a parent to their Customer instance. Each Customer will hold their order history, membership status, and payment information. This class holds methods to call their order history and make new orders by generating the specified amount of Tickets.

Attributes	Private order_history: Order[] ● A list of Orders pulled from the database, associated with this Customer. Private membership: boolean ● A flag for membership; is true when the Customer has a membership. Private payment_info: str[] ● A string array that holds the customer's payment info.
Operations	Public view_orders(): Order[] ● Returns the order history stored on this Customer. Used for displaying on the website. Public make_order(showtime, tickets): void ● Creates an order as specified by the user, associated with this Customer.

4.3.8 SysAdmin

The SysAdmin class will only be usable by certain instances of UserInfo that are associated with managers or higher-ups in the client's company. This class will hold methods for editing showtimes, orders, and retrieving sales data from the database.

Attributes	None.
Operations	Public edit_showtime(showtime, field, data): boolean ● Makes changes to the chosen showtime in the database, returns true if the operation was successful. Public edit_order(order, field, data): boolean ● Makes changes to the chosen order in the database, returns true if the operation was successful. Public get_sales_data(start_time, end_time): void ● Retrieves all orders within the specified time window from the database. Used to display on the website.

5. Development Plan

5.1 Partitioning of Tasks

The project will be divided into four major work areas:

- 1. Database & Backend Services – Designing schemas (users, showtimes, reservations etc.), implementing reservation logic, and connecting to the payment processor.
- 2. User Interfaces (UI) – Building the customer facing website, kiosk screens, and administrator.
- 3. Integration & Middleware – Connecting backend services to external systems (email notifications, payment API, 2FA).
- 4. Testing & Deployment – Unit testing, integration testing, and preparing the final system for deployment .

5.2 Team Member Responsibilities

- Shawyan Razavi – Administrative portal development, integration of external services (payment, 2FA, email), and testing coordination.
- Samuel Jenkins – Database design, SQL integration, and backend service for reservations and orders.
- Gregory Larson – User Interface development (customer web app + kiosk), ensuring efficient design and usability.

All members will contribute to documentation, testing, and GitHub commits.

5.3 Development Timeline

- Database Structuring and Development - 1 month
- Website Development, Frontend and Backend - 2 months
- API and Interface Integration Tests - 2 weeks
- On-Site Testing - 2 weeks
- Pre-Deployment - 1week

This timeline is specified for a maximum of 1 week of testing and debugging for each phase. If more testing is required, a reevaluation of this timeline will be conducted before notifying the client of the required time for testing and debugging.

6. Test Plan

6.1 Testing Procedure

For the Movie Ticketing System (MTS), we are testing the system at three levels. Those being Unit, Functional, and System tests. We confirm that the smaller components can work on their own and together, and that full workflows run the way they're intended to.

- Unit testing: These tests focus on specific system elements. For example, the methods used for determining seat availability or the ticket class's price calculator. Before integrating it with other functions, the main objective is to ensure that each small function can operate as intended.
- Functional Testing: These focus on connections between components. For example, testing how Orders and the Payment Processor work together so that duplicate payment messages don't end up charging or issuing extra tickets.
- System Testing: These cover full workflows, like the refund process. A test here would confirm that when an administrator issues a refund for a ticket order, the status of the payment changes, the ticket is now deemed as invalid, and the theatres QR code scanner can recognize that the ticket is no longer valid.

6.2 What We're Testing

Our test sets are chosen around the main features customers and admins will use:

- Ticket Pricing: Different ticket types (Regular, Student, Veteran, Senior) on different days.
- Payments: Successful payments, duplicate payment notifications, and refunds.
- Reservations: Holding seats during checkout and making sure they release if the checkout is abandoned.
- Ticket Validation: Ensuring only valid tickets scan at the gate and that refunded or invalid ones are rejected.

These are the most important parts of the system, and testing them helps us catch errors that would affect customers or revenue.

6.3 Tying Back to Our Design

We are connecting our tests back to the design diagrams in Section 4:
UML Class Diagram: Unit tests line up with attributes and methods in the Ticket, Order, and Showtime classes.
Architecture Diagram: Functional tests are tied to integrations, like Orders <--> Payments.

ERD (Database): System tests make sure data stays consistent across the database when a full workflow (buying, refunding, validating) runs.
This shows exactly what parts of the system each test is aimed at.

6.4 Testing Goals

With this plan, we're mainly checking that:
The system gives correct results (correct pricing and ticket statuses).
It's reliable (can handle things like duplicate payment messages without messing up).
It protects revenue (refunded or invalid tickets can't still get scanned at the gate).
It works smoothly for users (customers get the right tickets and confirmations).

A. Appendices

End of document.