# Royal Theater Ticketing System

## Software Requirements Specification

## Version 1

## February 14, 2024

Group 1
Isaac Acosta, Jakob Haddad, Jett Morgan,
Jonathan Van

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

## Revision History

| Date | Description | Author | Comments |
|---|---|---|---|
| 02/14/24 | First Submission of Draft | Isaac Acosta, Jakob Haddad, Jett Morgan, Jonathan Van | Final Draft |
| 2/28/24 | Second Submission of Draft | Isaac Acosta, Jakob Haddad, Jett Morgan | Added SWA and UML Diagrams as well as their respective descriptions |
| 3/13/24 | Third Submission of Draft | Isaac Acosta, Jakob Haddad, Jett Morgan | Added test case excel document with their respective descriptions. |
| | | | |

## Document Approval

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

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

# Table of Contents

# 1. Introduction

## 1.1 Purpose

The intent of this document is to provide a list of the necessary components of the project, as well as outline the system's functionality and limitations.

## 1.2 Scope

The primary scope of this SRS pertains to the Royal Theater Ticketing System project for Royal Theater Ticketing System. The purpose of the mentioned system is to provide a means for customers to view available movie times, purchase tickets and seating accommodations, as well as serve to allow customers to interact with our theater via the internet or our on-site kiosk.

The goal is to allow users both at and outside of our theater to complete transactions with our products, view specified information about a movie they are interested in visiting at our establishment, or find showing times at our location. As a result, money received from such transactions and advertisements on the system will allow for additional revenue to be generated for the theater directly. Our product will enable Royal Theater to reduce in-person wait times thus significantly improving our in-person customer experience. Additionally, digitizing our ticketing system will allow Royal Theater to reduce employee labor costs. In order to reach a wider online market, our online system will support three languages: English, Spanish, and Swedish.

## 1.3 Definitions, Acronyms, and Abbreviations

Largest Contentful Paint - LCP
Cumulative Layout Shift - CLS
First Input Delay - FID
Royal Theater Ticketing System - RTTS
Distributed denial-of-service (cyber attack) - DDoS
*Cloudfare* - Company that provides content delivery network services, cloud cybersecurity, DDoS mitigation, and ICANN-accredited domain registration services.

## 1.4 References

Documents developed through customer interviews to obtain specifications and requirements for the ticketing software.
Customer Interview #1
Customer Interview #2
Customer Interview #3

Customer Interview #4

### 1.5 Overview

The remainder of this SRS provides general information about the product and its requirements, including user and functional requirements, product specifications, constraints, interface conditions and limitations, as well as analysis. The general description of the software can be found in Section 2 of this document, providing information about all questions related to the system itself. Section 3 provides information about specific requirements for features or requests, and Section 4 holds supporting documentation pertaining to this SRS.

## 2. General Description

This subsection serves to describe the general factors of the ticketing system and does not state its specified requirements, which can be found in Section 3. This section serves to only make the mentioned requirements easier to understand.

This document outlines the general list of requirements and features that we intend to have implemented into the system. Such features include handling online ticketing purchases In addition, it also lists the stakeholders and users of our specified system, as well as budget, time constraint and resources allocated to complete the project.

### 2.1 Product Perspective

The Royal Theater Ticketing System is meant to interface directly with the database of Royal Theater itself. Purchases made, movie times and movie info will be given back and forth between the two systems.

### 2.2 Product Functions

Our product will contain a reward system within the website which allows users to accumulate points through the purchase of movie tickets, merchandise, and concession stand purchases. These points will allow our users to claim small rewards to incentivize user account creation.

### 2.3 User Characteristics

Users of our product will need to have access to the internet, as well as a somewhat up-to-date device. In addition, online transactions will request a form of payment such as a credit/debit card or online merchant

### 2.4 General Constraints

Constraints consist of a max concurrent capacity of 1000 users, a 4 month deadline for the software team to develop, in addition to the website needing to be compatible with

most web browsers and theater kiosk hardware. $500,000 is the current budget for this project. Project is based in San Diego (Pacific Time Zone), with 20 local locations.

### 2.5 Assumptions and Dependencies

Our software dependencies for the user is their use of a compatible browser and operating system either on their phone or desktop computer. The operating system's that will be supported are Windows, IOS, and Linux. The user must have a stable internet connection throughout their duration on our website.

## 3. Specific Requirements

This subsection contains the requirements we have gathered for the RTTS. Divided into separate sections to be as clear as possible, with similar features and requirements being grouped together. All of these requirements can be traced through their identifier and checked off individually (eg. *3.1.3.1, 3.2.1.2.)*

1. **Current and Future Movies must be accessible to the User**
    a. Includes live information such as name, runtime, showtimes, critic ratings, actor and movie description, theater number and price of ticket. This includes information about each theater such as how many people are inside their theater of choice
    b. Very high priority
    c. As this is the main feature of the service, this should be as compatible as possible
    d. Only dependency should be on our in-house theater database
2. **User must be able to Purchase Tickets and Select Seating Through the Interface**
    a. Software will securely authorize credit and debit card purchases, including most banks and PayPal
    b. High Priority
    c. Could be expanded to include other methods of payment such as smaller banks or other banking apps
    d. Dependent on our internet connection to authenticate and authorize purchases with banks
    e. Maximum of 20 tickets purchased at one time.
    f. Can book tickets 2 weeks in advance up to 10 minutes past the showtime.
    g. Purchase must be made within 5 minutes of booking the ticket, otherwise the ticket will be abandoned and made publicly available again.
    h. Regular theaters hold 150 seats while deluxe theaters only hold 75. Each theater holds 6-8 showtimes per day.
3. **User must be able to create and access their individual Movie Theater Account**
    a. Servers will securely store information pertaining to a user's account, allowing them to access member perks such as redeemable coupons, or movie tickets with accumulated points distributed with every purchase. In addition, they may edit

personal account information such as saved form of payment, email address or password.

b. High Priority
c. Has the possibility of expansion including a revamp of the rewards system backend or new perks
d. Dependent on our server's up-to-date information on a user's account details

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

*3.1.1.1* The interface for this website is built to be easily navigated by the customer.
*3.1.1.2* There is a help/customer support directory to provide assistance.

### 3.1.2 Hardware Interfaces

*3.1.2.1* The program uses a physical server, which is used to save data such as customer information, invoices, and reward points.

### 3.1.3 Software Interfaces

*3.1.3.1* In order for all up-to-date devices to be able to handle the website resources, the program will be written in JavaScript.

### 3.1.4 Communications Interfaces

*3.1.4.1* The program will use HTTP to connect the user's web browser to the hosting server.
*3.1.4.2* API is used to access a list of movies as well as showtimes.
*3.1.4.3* The program's checkout page provides the option to complete transactions using a PayPal portal.

## 3.2 Functional Requirements

### 3.2.1 Functional Requirement or Feature #1- Online Ticket Orders

3.2.1.1 Introduction
Our website will allow users to purchase tickets through 256-bit encryption secured debit/credit card transactions with the inclusion of support of third-party payment systems such as PayPal.
3.2.1.2 Inputs
Support for various browsers such as Google Chrome, Firefox, and Safari with native support for IOS, Windows, MacOS, and various Linux distributions. Support for American Express, Mastercard, Discover Visa, and PayPal.
3.2.1.3 Processing
The payments will be processed through a secured payment gateway with authorization of the payments success before the distribution of the tickets and invoice.
3.2.1.4 Outputs
After each purchase the user will receive an emailed invoice of their purchase.
3.2.1.5 Error Handling
If an error occurs, an error message will be displayed to the user, along with a code and referral to support.

### 3.2.2 Functional Requirement or Feature #2 - Movie and Theater Seating Info

3.2.2.1 Introduction

Our website allows users to view current and upcoming films as well as additional information fed directly from our in-house database.

3.2.1.2 Interfaces

When browsing our website, a user will be able to click on or tap a film after searching or looking through our feed. Interacting with a film will redirect the user to a new page displaying movie title, runtime, actors, description and critic ratings from *Rotten Tomatoes and IMDb*. In addition, the user can view movie runtimes at our theater.

3.2.1.3 Interaction with Ticket Order System

On the page displaying film information, the user will be prompted with the ability to purchase a ticket(s) for the selected movie. This would redirect the user to our online ticket ordering page, outlines in **3.2.1**

3.2.1.4 Assigned Theater Seating

After purchasing the ticket, the user will be able to view a grid representing all the available seats for their selected theater and time. In this window, the user can select any seat not marked as occupied, and will be emailed confirmation of their choice by our system.

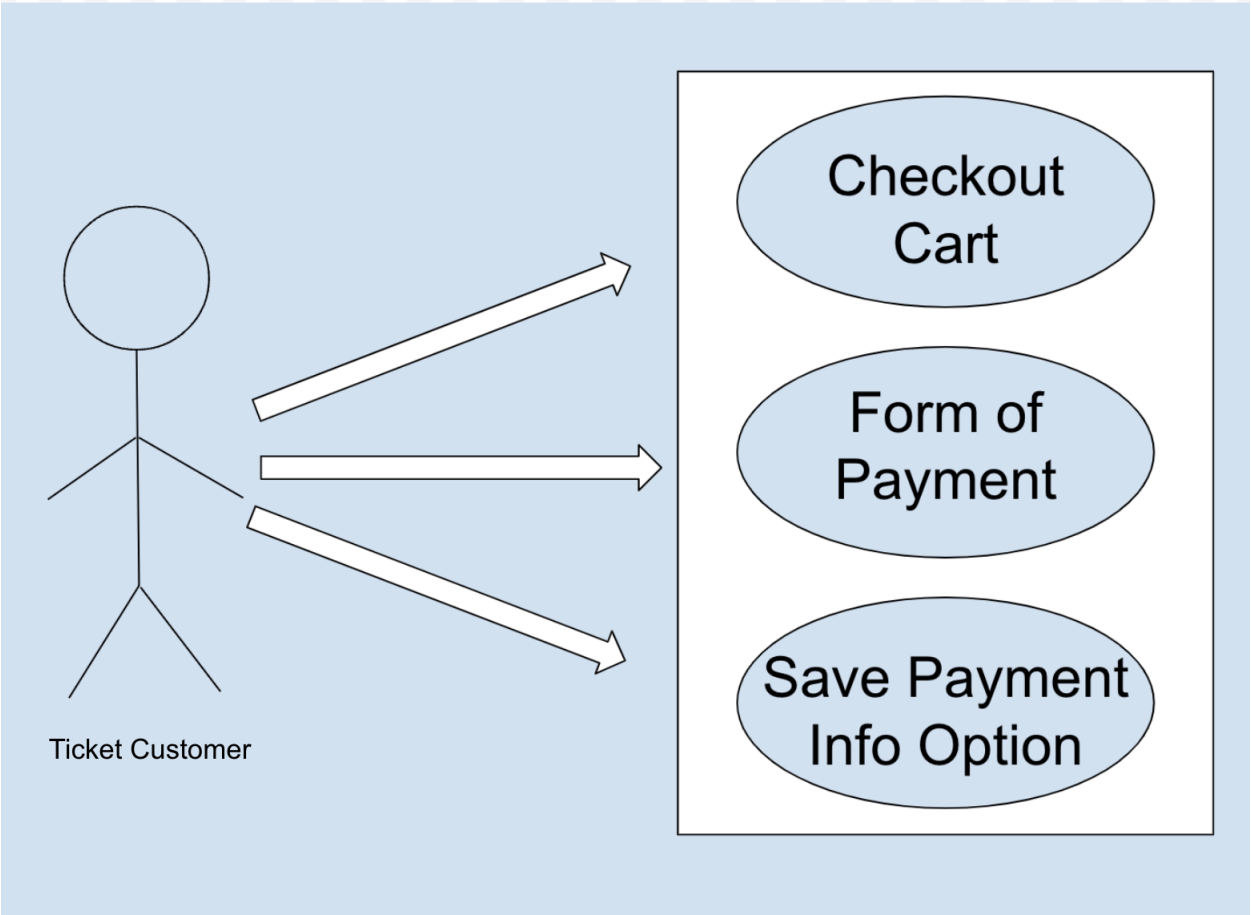
3.2.1.5 Physical Purchase Functionality

After choosing their seat on the website, the user should be prompted with the choice to print their ticket on paper, or be required to show digital verification of their ticket and seating at time at showtime. For in-house kiosks, the user will automatically be given a physically printed ticket. In a case where the ticket cannot print, they will be able to show digital verification instead.

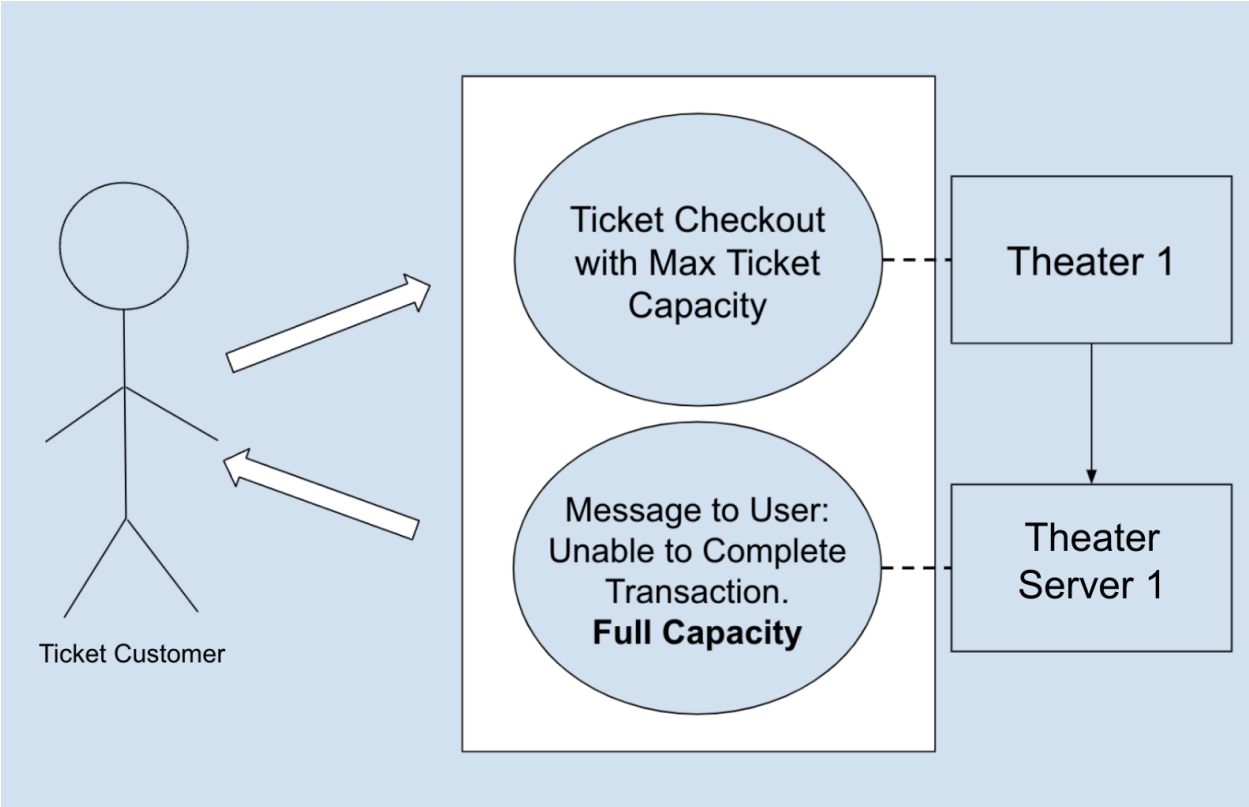
…

## 3.3 Use Cases

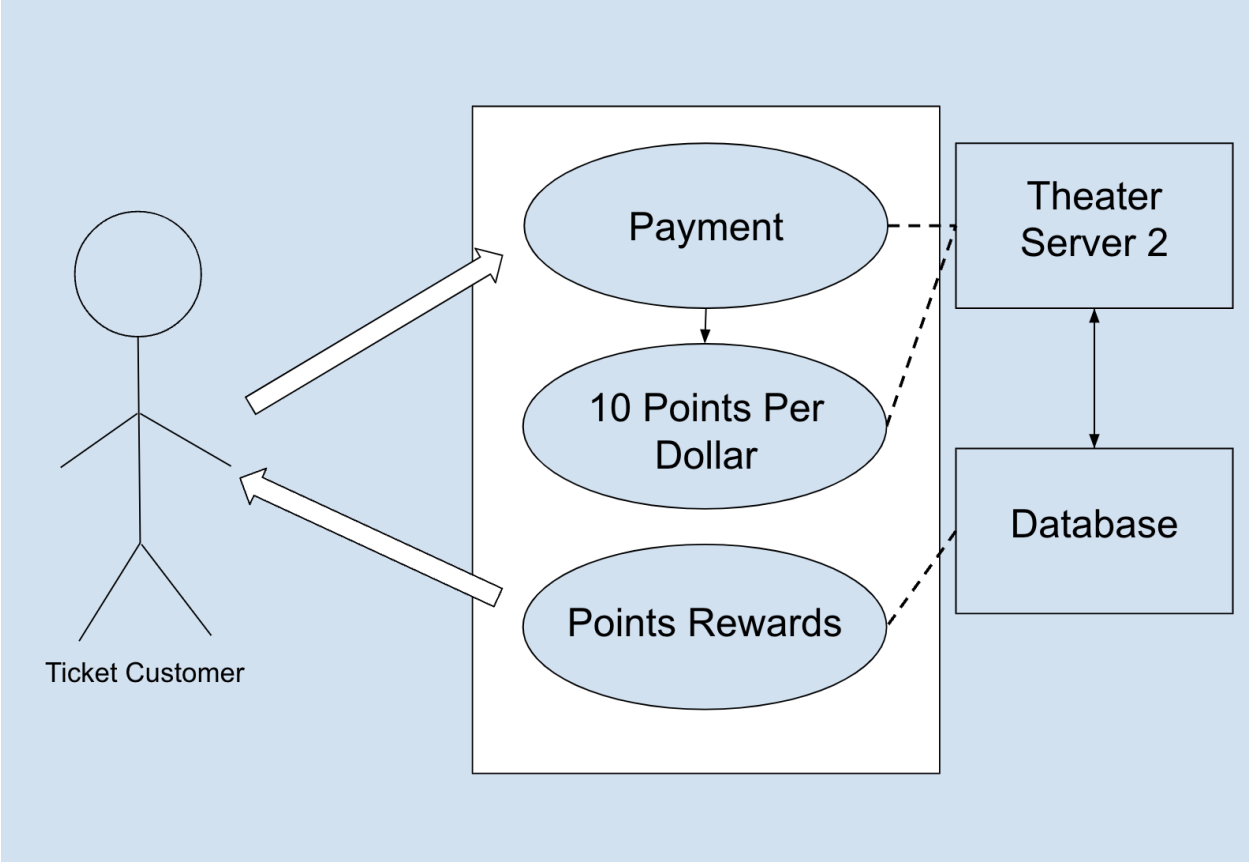
### 3.3.1 Use Case #1 (User options when making a transaction)



### 3.3.2 Use Case #2 (Purchasing a ticket for a movie with max capacity)



### 3.3.3 Use Case #3 (User earning points when making a ticket purchase)




## 3.4 Classes / Objects

### 3.4.1 Class #1: User

Listed will be the permissions granted to the end User when using interacting with our software;

3.4.1.1 Attributes

Users are separate from other classes by their ability to create and edit their accounts and make purchases, which are not necessary for other classes.

3.4.1.2 Functions

**3.4.1.2.1** User Account Management

3.4.1.2.1.1 - User can create an account with our software using an email address, password, age and name.
3.4.1.2.1.2 - User can edit account details after creation of an account
3.4.1.2.1.3 - User can login to an already created account using our software
3.4.1.2.1.4 - User can attach banking account details to their account

3.4.1.2.1.5 - User can use erase banking account details from their account
3.4.1.2.1.6 - User can check their accumulated point balance from their account
3.4.1.2.1.7 - User can opt-in to deletion of their account from our servers
3.4.1.2.1.8 - User can choose their preferred language to change which language the software displays. Comprehensive English, Spanish, Swedish, Tagalog, Japanese, and German.

**3.4.1.2.2** User Interactions with Theater

3.4.1.2.2.1 - User can purchase tickets from our catalog of current and future movies
3.4.1.2.2.2 - User can specify a theater they wish to purchase a ticket for
3.4.1.2.2.3 - User can specify the type of ticket they wish to purchase
3.4.1.2.2.4 - User can input a promotional code or apply for a type of discount based upon user status (military status, student status, age)
3.4.1.2.2.5 - User can view theater information such as how many tickets are already sold for a specific showing
3.4.1.2.2.6 - User can choose a seat for each purchased ticket when applicable (deluxe theater only)
3.4.1.2.2.7 - User can use our catalog browser to search for current and future movies
3.4.1.2.2.8 - User can spend accumulated points on rewards we set
3.4.1.2.2.9 - User can issue a refund for any non-reward point purchase
3.4.1.2.2.10 - User can purchase a Royal Membership card through a prompt
3.4.1.2.2.11 - User can purchase select items from the concession stand to pick-up in theater
3.4.1.2.2.12 - User can leave an 1-5 smilies review after each purchase

<Reference to functional requirements and/or use cases>

### 3.4.2 Class #2: Administrator

3.4.2.1 Attributes

Administrators have access to all permissions operated in the backend of the system. They do not have permission to make purchases or spend points related to the theater system however.

3.4.2.2 Functions

**3.4.2.2.1** - Information Management

3.4.2.2.1.1 - Admins can edit the list of movies available for ticket purchase at each individual theater
3.4.2.2.1.2 - Admins can edit the list of showtimes for each movie and theater
3.4.2.2.1.3 - Admins can cancel or add showtimes as needed
3.4.2.2.1.4 - Admins can edit which sources movie critic ratings are scraped from
3.4.2.2.1.5 - Admins can edit which movie critic quotes are shown next to each movie
3.4.2.2.1.6 - Admins can edit the items purchasable from the concession stands, including their prices

3.4.2.2.1.7 - Admins can edit the ticket status of any purchase
3.4.2.2.1.8 - Admins can add any amount of reward points to a specified account

**3.4.2.2.2** - Error Handling

3.4.2.2.2.1 - Admins can refund any purchase
3.4.2.2.2.2 - Admins can cancel any purchase
3.4.2.2.2.3 - Admins can close any account, deleting their information from the database
3.4.2.2.2.4 - Admins can edit customer information, immediately changing their information in the database

**3.4.2.2.3** - Additional Information Access

3.4.2.2.3.1 - Admins can view all stored user purchases in the database
3.4.2.2.3.2 - Admins can view all showtimes and movies at all theaters
3.4.2.2.3.3 - Admins can view amount of tickets sold and amount of money earned on tickets per movie
3.4.2.2.3.4 - Admins can view the account details of all accounts
3.4.2.2.3.5 - Admins can view a list of all suspected bot/scalper accounts (specified by high ticket purchase in a small amount of time)

## 3.5 Non-Functional Requirements

### 3.5.1 Performance

3.5.1.1 > 1 second to complete the user's HTTP request to the website. This is measured omitting any delay produced by the user's internet speed.
3.5.1.2 LCP (Largest Contentful Paint) < 2.5 seconds - a metric measuring the time is takes for website content to load for the user.
3.5.1.3 CLS (Cumulative Layout Shfit) < 0.1 - a metric to measure the visual stability of a website.
3.5.1.4 FID (First Input Delay) < 150 MS - a metric measuring input delay to the website

### 3.5.2 Reliability

3.5.2.1 We will implement the use of hybrid and local backup servers to preserve data and ensure the functionality of our website in the case of a primary server failure.

### 3.5.3 Availability

3.5.3.1 We will execute our server/website maintenance and updates outside of the hours of operation for our movie theater in order to allow our website to be available during 100% of operation hours under ideal running conditions.
3.5.3.2 Protection against abnormally large purchases to prevent resale of both purchases of tickets.

### 3.5.4 Security

*3.5.4.1* To ensure a secure database, customer information will be encrypted.
*3.5.4.2* A multi factor authentication preference will be provided to prevent account breaching.

3.5.4.3 We will ensure cloud traffic security using *Cloudfare* to regulate abnormal traffic to our website protecting against DDoS attacks.

### 3.5.5 Maintainability

*3.5.5.1* In order to handle a big flow of customers, the program must be scalable to handle all 10 million concurrent users.

### 3.5.6 Portability

*3.5.6.1* Website must be accessible across all web browsers such as Chrome, FireFox, and Edge.
*3.5.6.1* Front-end design must be exhaustively accommodative of all mobile screen variations and layouts.

## 3.6 Inverse Requirements

3.6.1.1 If a user enters a wrong password, we will deny access and provide a message saying "incorrect password" then prompt for the user to re-enter their password.
3.6.1.2 If a user attempts to purchase more seats than are currently available the request will be denied then we will reprompt the user to complete their seat purchase while also displaying the number of available seats.
3.6.1.3 If the user attempts to claim a reward that they do not have the required amount of points for, deny the request and inform the user they have insufficient points.
3.6.1.4 If the user's preferred method of payment declines, we will deny the purchase and display a message saying "Transaction failed: Payment method declined", then we will prompt for a different form of payment.

## 3.7 Design Constraints

### 3.7.1 Development Tools

The quality of our webpage application must conform to Microsoft's "Web, Software, and HTML Consideration" standards.

### 3.7.2 Company Branding Policies

In order to maintain the company's public image, there must be guidelines to follow when developing logo and palette design, as well as displaying promotional material on the website.

### 3.7.3 Website Specifications

Must apply principles that make it as compatible with as many web browsers as possible. In addition, the webpage must be lightweight as to run efficiently on kiosk hardware specifications
Must be able to handle up to 1000 concurrent users without major slowdowns or crashes
All aspects of the website must be intuitive and easy to find on website's homepage
General ability to navigate a webpage is required for use by user

### 3.7.4 Hardware and Software Constraints

To maintain uniformity the website will be coded in JavaScript.

## 3.8 Logical Database Requirements

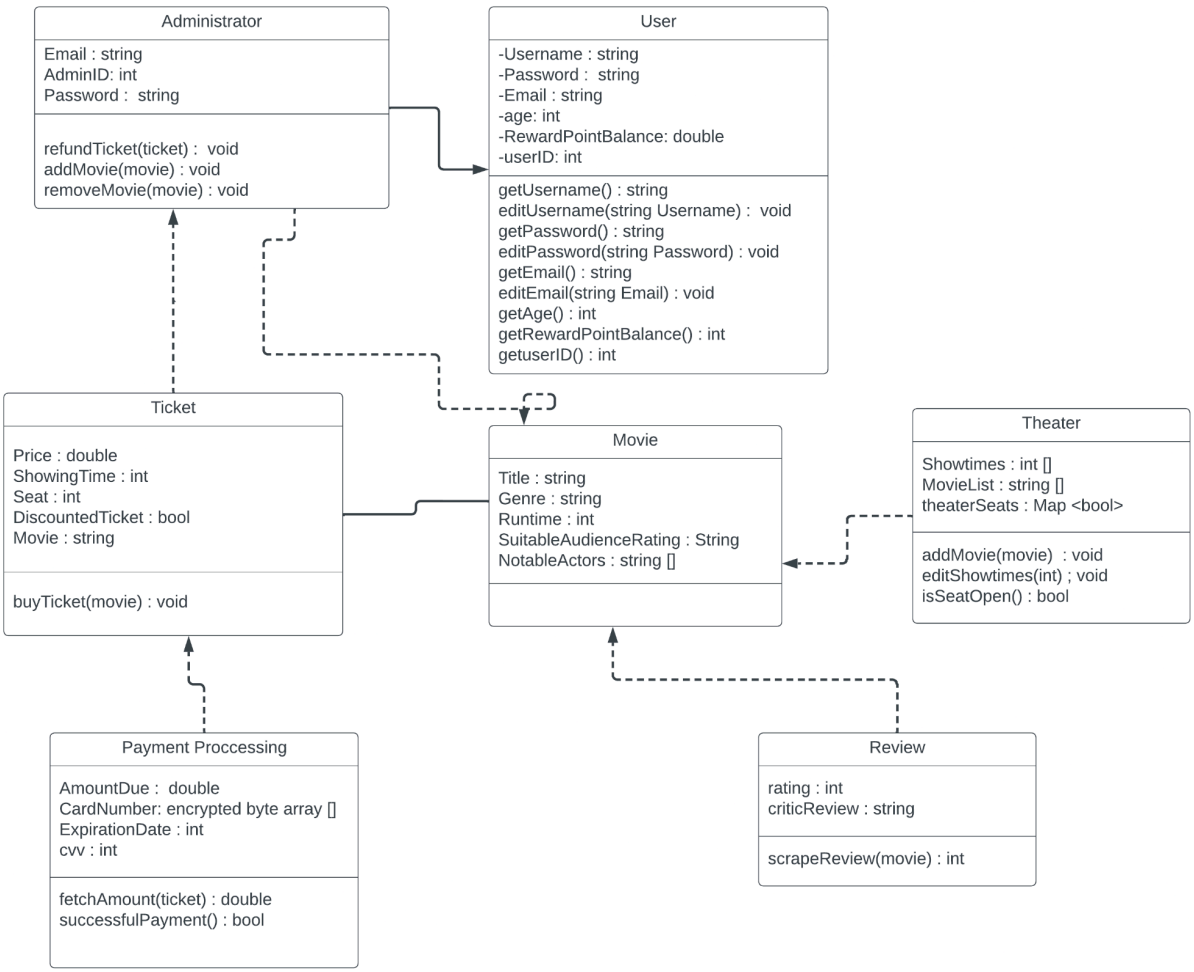
*Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.*

## 3.9 Other Requirements

*Catch all section for any additional requirements.*

# 4. Analysis Models

## 4.1 UML Diagram




*4.1.1* UML Class Diagram Outline

User (Super-class) and Administrator (Sub-class)

The user class contains attributes for basic account information such as string variables for username, password, and email. It also contains the reward integer point balance and unique user ID for each account distinct from their username, as the user ID will be used as a non-verbose identifier within the code to enable ease of access for the programmer. This class also contains "setter" and "getter" functions for each respective attribute to navigate the class's private

encapsulation. The “setter” and “getter” functions take in the parameters of the respective data type that will be changed or return the respective data type of the attribute called. Additionally, this serves as the parent class for the administrator class which inherits attributes and functions of the “User” class with additional admin ID and password as well as auxiliary administrator permissions to refund tickets and add and remove movies by taking movie class objects as parameters.

Movie Class

The movie class is a central class to our program facilitating many interactions with other classes and functions. This class contains the title, genre, and the Motion Picture Association film age ratings, all stored as strings. Runtime is stored as an integer and the notable actors in the movie are stored in string arrays. This serves as a core object parameter for other classes.

Ticket Class

The ticket class contains ticket information and interacts with the movie class. It has a price stored as a floating-point double to allow cent precision, as well as an integer value storing the showing time in 4-digit 24-hour time format, and an integer seat number. It also indicates if the ticket has a discount applied using a boolean true false value and contains the movie name as a string. Lastly, this class has the function to buy a ticket that takes in a movie class object which then interacts with the payment processing class for purchase completion.

Payment Processing

The payment processing class serves as a secure payment gateway to complete user purchases in a safe and secure manner. The amount due is fetched from the ticket class and the users banking information is passed in and encrypted using “SHA2_256” encryption which is then stored as an encrypted byte array. The user’s cvv and expiration date are stored as integers. The functions of this class are the fetch amount function taking in a ticket class object as a parameter and returning a double which is the price of the ticket. There also exists a function to authenticate successful payments using boolean true false return types to indicate the success of the purchase.

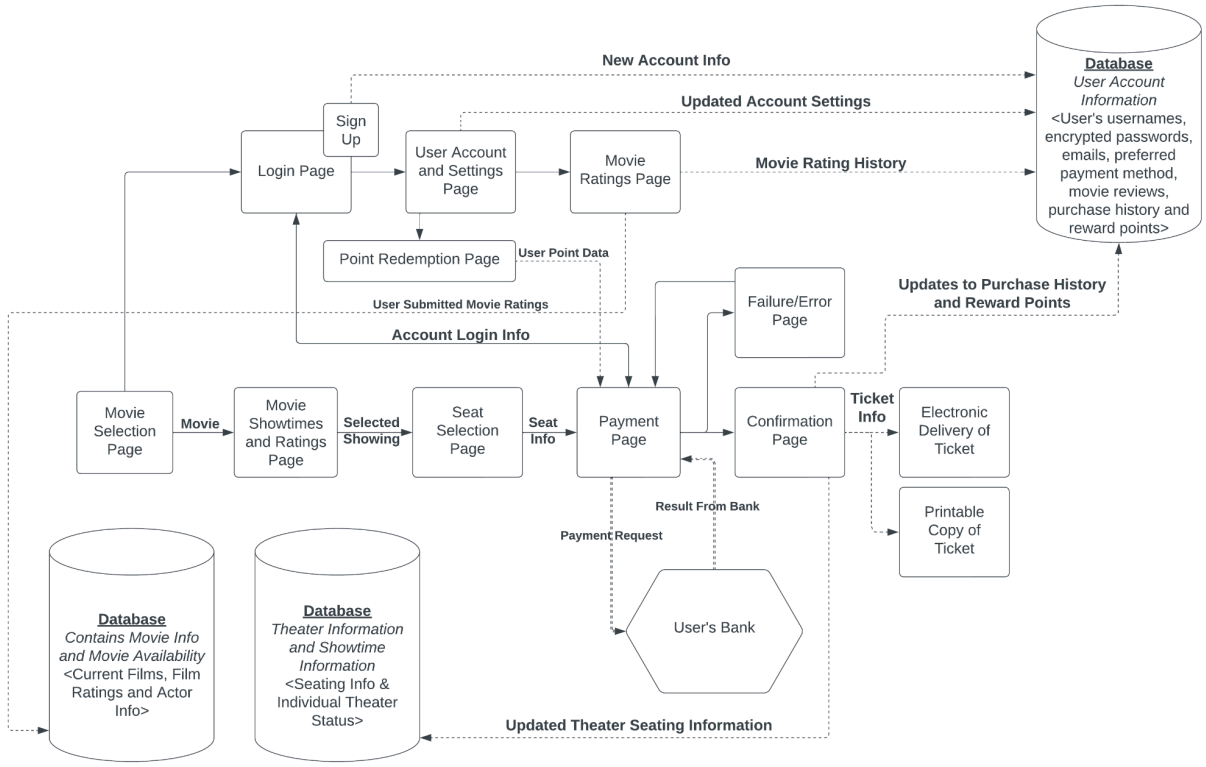
Theater

The theater class is to be used to represent each theater’s showtimes and movie list on a per-day basis as well as the available seats in each theater. The showtimes are stored in an integer array, the movie list is stored in a string array, and the theater’s available seats are stored in a boolean map. The class contains functions to add movies, taking a movie class object as a parameter. Additionally, there is an edit showtimes function which takes in an integer and a function to check if a seat is open with a return type of boolean.

Review

The review class is intended to scrape movie review information and has an integer rating and critic review string and has a function to scrape review taking in movie class objects as parameters.

## 4.2 SWA (Software Architecture) Diagram




*4.2.1* SWA Diagram Outline

The SWA Diagram is a detailed description of the flow of interactions between the user and the movie theater ticketing system. When a user accesses the website, they are presented with a selection of movies which is accessed from the movie database. Once the user has picked a movie to preview, they are provided with specified movie info such as current films, actor information, and ratings, which is also accessed from the movie database. If the user clicks on a showing time, they are asked to login/signup, then they are taken to a seat selection. The seating database is accessed to present a chart of available and unavailable seats, as well as the theater's status. After a seat is selected, the user is taken to the payment screen. If the user has purchased from the website before, the user account info database will automatically provide a filled in form of payment. If the user is new, a billing information template and a PayPal portal preference will be provided. If the user would like to save their billing information for the future, it is stored into the database. Otherwise, if the user has enough rewards points, they can purchase a ticket for free with the redemption of their accumulated points. Unless purchased through rewards points, the transaction is confirmed by the user's bank. If the payment goes through, the user is emailed a payment confirmation as well as a digital ticket and a printable copy. If the payment does not go through, a failure/error message is displayed. After a transaction is completed, rewards points are calculated according to how much money the user spent, and updates are made to their purchase history and reward point status in the user account database. Seating updates are also made to the seating database.

If at any time a user would like to make changes to their account, they can access the user account and settings page to make changes. These changes will then overwrite old information stored in the user account info database.

If a user would like to make a movie rating, they can access the movie rating page. Once a rating is submitted, said movie's ratings will be updated in the movie database.

**Development Plan and Timeline**

Within the four month timespan of development, responsibilities were split up across the team. Isaac Acosta conducted an analysis of web development and the significance of similar websites, in order to connect the client's goals to the website's final product. In addition, Issac also headed the software development work including programming and database building.
Jakob Haddad designed the outline of the development process including project scope, general requirements/responsibilities and diagrams. This information was utilized by in-house engineers to complete individual modules of the website, in addition to connecting each module together.
Jett Morgan performed a series of functional and non-functional testing. Tests conducted ensured the website works as intended, and verified that the website holds up to performance and security requirements. Jett is currently responsible for maintaining the website, and addressing any issues that may arise.

GitHub Repository:
https://github.com/Yaykuby/Software-Design-Specification
Isaac Acosta - Github Commit:
https://github.com/Yaykuby/Software-Design-Specification/blob/main/submitRoyal%20Theater%20Ticketing%20System.pdf
Jett Morgan - GitHub Commit:
https://github.com/Yaykuby/Software-Design-Specification/blob/main/CS250%20-%20Royal%20Theater%20Ticketing%20System%20-%20Jett%20Morgan%20-%20827815411.pdf
Jakob Haddad - GitHub Commit:
https://github.com/Yaykuby/Software-Design-Specification/blob/main/CS250%20-%20Royal%20Theater%20Ticketing%20System%20(1).pdf

# 5. Verification Test Plan

This section lays out test plans for verification, including which features of our software we are testing, what each test set is, how we selected our tests, as well as what the most tested and crucial parts of our testing include.

## 5.1 Test Plan

### 5.1.1 Test Plan Overview

The purpose of our verification and testing phase is to ensure within our own development team before the product is pushed out to the customer, that the product is functional to the extent required by the customer, as well as polished to our own as well as industry standards. The initial range of our testing covers the base functional requirements of the software, moving on to the less foundational features afterwards. We chose to implement tens of tests per functional requirement to ensure that not only does the feature stand up to expected standards, but to check if our caught edge cases and own coding prowess was appropriate for the software.

### 5.1.2 Functional Requirement Testing

To start off our testing, we needed to stress test our main functionality of the software through white box testing. We conducted over 150 tests pertaining to the functionality of the website over our two month window of testing. Firstly, we began with the homepage feature. We tested through observation of the running code and a visual representation of our homepage presented through our IDE, if the expected data was being displayed. After verification, we systematically checked through each page that our website was capable of producing for the user (homepage, search page, selected movie page, payment page, account page, error page, etc.), and checked back with our servers to make sure that data exchanged between the user and the website was being stored and retrieved properly.

In addition to each page, we needed to test our search feature included within the software, as it is one of the main ways that users are expected to interact with our software. Testing included typing certain keywords that were tagged with specific movies being shown at the theater (most commonly denominated by their name). The search bar then led to a page showing all results involved with that keyword. This feature was heavily tested and only failed when searching for movies by actors, which was then quickly fixed and resulted in the search code logic being updated.

As a large part of the software is being able to purchase tickets and choose seats electronically, much of our testing time was spent on ensuring that the website was capable of completing this task without fail. First, we tested if a user could choose a showtime from the selected movie page (the page displayed after clicking on a specific film). After this was ensured, we tested the seat selection functionality of the website. This was difficult as this function required in-real time updates from the database, which was updated directly from the theater's current status. After reworking our database to decrease latency between updates both to and from the database, we were able to successfully select seats and theaters at a speed we were comfortable with. Next was the testing of the payment page, which included retrieving stored banking information, if previously saved by a user, and carrying through with a transaction through a merchant of the user's choice. This feature worked very well in initial testing, and continues to work successfully through alpha and beta testing stages.

### 5.1.3 Unit Requirement Testing

Unit Tests included testing each individual component of the website to make sure it works properly. Upwards of 300 tests were completed and verified to ensure functionality of each individual component the user has access to utilizing. These tests took up a majority of the verification testing time and led to lots of debugging of the individual components to ensure they worked exactly as expected by our team. Every clickable component, searchable movie, enterable information and saved status' was tested by our team multiple times. Logging into the website was first and foremost the most important feature of this testing phase, as being unable to access the website would stop all other functions from working outright. After our URL directed users to the homepage, we branched out and tested each feature as they were presented to the user. For example, from the homepage we tested the "Current Movies" feature, as it is most likely the next feature a user of our site would utilize.

### 5.1.4 System Requirement Testing

Throughout the duration of checking our System Requirement Testing, we ran through two phases of testing; A private Alpha saved for only in-house testing, and an early public Beta, offered to residents who lived near the address of the theater. This began our black-box testing phase. In our Alpha stage, which spanned one week, test users were created and utilized by our developers to mimic real users. They would go through the site, which had (at this point) fully implemented the UI, formatting style and outline chosen by our customer. We tested every possible path that a user could take through our website, after systematically laying out each possible choice a user could make on each part of our software. Our team became very confident in the final product and eagerly awaited beta testing to catch any possible issues we were unable to iron out at this stage in development.

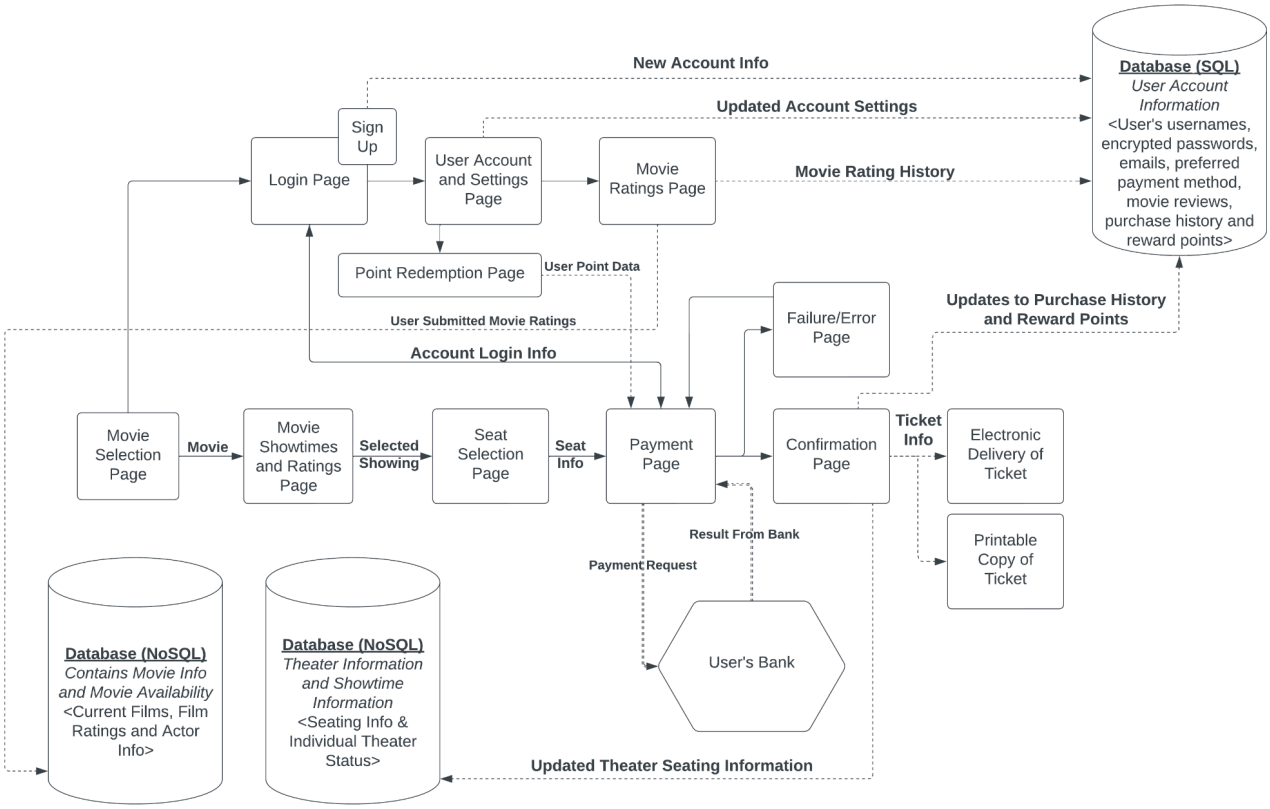
After Alpha testing had concluded, Beta testing started up immediately and ran for three weeks. At the beginning of this stage, we mailed out a link and QR code to a pre-release test version of our website, fully ready for public testing, to all home addresses linked within a 15-mile radius around the movie theater. The Beta branch of our website included all the functionality and dependencies that the release build will have, minus the bug changes made after the beta branch concluded. Shipped in the pre-release build was a feature allowing users to submit any potential bug or website side issues that they had noticed, with the ability to submit images along with the message. The beta branch was accessed by 461 unique users, and resulted in the submission of over 40 bugs or concerns, many of which we were not previously aware of.

# 6. Architecture Design and Data Management

It is imperative that the Royal Theater Ticketing System be equipped with a strategy for data management. Due to the heavy reliance that our software has on information provided outside of the user, we plan to use multiple databases to hold all the on-demand information that is required for a smooth, fully functioning and efficient experience for the user. Each database includes multiple security measures that ensure maximum protection for both the theater's private data as well as that of the user.

## 6.1 Architecture Design Diagram (including databases)

Our Architecture Design Diagram remains largely unchanged from its original iteration. Our fairly significant yet only change was to describe how each database is formatted. We are utilizing the SQL database structure for our “User Account Information” database, and a NoSQL structure for both our “Movie Information” and “Theater/Showtime Information” databases respectively. We go into further detail on why each database is formatted the way they are in out Data Management Strategy, but we chose the more constantly updated and accessed databases to be formatted in NoSQL and our less accessed and more privacy-demanding database to be SQL.




## 6.2 Data Management Strategy

Our data management strategy utilizes three databases, two of which are NoSQL databases since their non-relational structure is more appropriate for our use case of storing a variety of items such as films, ratings, actors, and showtimes that are constantly changing and requiring updates. For our first database, we went with Redis. With this option, we can pull data in a key-value process, which pairs data together. For instance, customers can view seating availability in real time and their sessions on the website can be saved and accessed later. Data handling is important in a system like ours as it must deal with large pieces of information such as movie details, so for our second database, we used MongoDB. MongoDB is responsible for storing information in a JSON document style, which is good for storing several children values for one parent value. Our third database is a SQL database that uses the DBMS MySQL which is responsible for storing sensitive account information. Using an SQL database for this use case is

more advantageous as it allows us to employ SQL's relational structure as the information stored will be associated collectively with a user's account. The three database structures that we opted for were chosen in order to most effectively divide more general theatre info and a more particular and detailed database that will contain movie info. This optimizes database performance by enabling faster access times into each database as each database will contain less info and all associated information will be contained in the same database.

### 6.2.1 Tradeoffs

Both SQL and NoSQL database structures have their own strengths and weaknesses. For our more constantly updated databases, we chose to utilize the NoSQL structure. We favored this type of structure because of its compatibility with flexible, unstructured and dynamic data, non-relational structure and horizontal scalability for cheap. However, due to our favor towards NoSQL, we are missing out on the more structured and private mannerisms of the SQL system. However, we felt that such advantages were not suited to the type of and demand of the data we were storing and accessing in these databases.

We did, however, utilize SQL in one of our databases. This database is responsible for storing user data and info, including saved banking methods, emails, passwords, usernames and purchase history. We felt that an SQL structure was better suited to this type of database because of its relatively unchanging nature (once a user account is created there will be minimal updates to this information) and its more secure reputation. In addition, because the technology and SQL structure is older, it is much more researched and easily fixed if any problems or concerns arise. We are missing out on utilizing the more quickly-updating nature of NoSQL by implementing the database in this way, but it seemed to the team like too much of an expense for not much in return.

# 7. 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