Movie Theatre Ticketing System

Software Requirements Specification

Version 1.0

7/9/25

Group 5
Kevin Stratton,
Phu Duong,
Ryan Haigh

Prepared for
CS 250- Introduction to Software Systems

Instructor: Gus Hanna, Ph.D.
Summer 2025

Revision History

Date	Description	Author	Comments
7/9/25	Version 1.0	Kevin Stratton, Phu Duong, Ryan Haigh	First version of document

Document Approval

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

Signature	Printed Name	Title	Date
	Kevin Stratton	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

This SRS document will describe how the ticketing system will interact with users and what requirements are needed for its optimal production. Several assumptions will be made of the customers of this software and the possible risks associated with either the users or the scenarios that arise from their demographic (i.e. certain users having incompatible devices), this document will describe our considerations of these possibilities and how to deal with them in our design.

1.1 Purpose

This software aims to deliver appropriate tickets to paying customers in an efficient and secure manner that accommodates them to their needs as much as possible. Potential customers can be in-person or online movie-goers who intend to watch a specific movie; Spanish, English, and/or Swedish versions are provided if necessary.

1.2 Scope

Two softwares will be produced, an online website and an in-person digital kiosk that functions similarly. They will both give customers their movie tickets based on their preferences and movie choice after being purchased from, with the digital kiosk having an option for physical printable tickets. Such tickets will serve as proof of entries into physical movie theaters. Customers will be able to correlate their tickets to specific seats if they choose a ‘deluxe ticket’, otherwise the ticket will only serve as entry for regular tickets. Measures will be implemented to prevent scalping and system overloads during high traffic; a loyalty program will also be implemented to maintain customer relations. Both products will not be able to serve as replacements for customer service if any human error occurs. The software will be scalable up to 10 million concurrent users.

1.3 Definitions, Acronyms, and Abbreviations

Regular Ticket: A proof of entry into a regular theater. Customers with regular tickets may choose to sit anywhere in a regular theater.
Regular Theater: Theaters with 150 seats.
Deluxe Ticket: Reserve specific spots corresponding to the deluxe ticket. Also serves as a proof of entry into a deluxe theater.
Deluxe Theater: Theaters with 75 seats.

1.4 References

- Theatre Ticketing System Qs
- Theatre Ticketing System Requirements

1.5 Overview

The rest of the software will include several other features that allow for consistent uptime as well as safe and efficient maintenance. This SRS will describe those features and how they will work as well as their risks and use-cases.

2. General Description

Customer accessibility will play a major role, as this software will aim to make the process of acquiring movie tickets as simple as possible for several demographics including but not limited to: Spanish speaking/English speaking/Swedish speaking people, people buying tickets in bulk, people in and outside the San Diego area, people who benefit from student and/or military status, people who prefer online transactions, and people with varying payment methods. All in all, the requirements for this software are greatly powered by enhancing user experience.

2.1 Product Perspective

The software will be compatible with many common devices with web-browsers, including a self-service digital kiosk that has an additional print feature for physical tickets. The digital kiosk will most likely be a large screen TV with touchscreen capabilities.

2.2 Product Functions

The software will guide users through a digital ticketing system (depending on their preferred language; English, Spanish, or Swedish), assigning them to a specific ticket that correlates to their chosen movie. During this, the software will perform a transaction with the user, only giving them the ticket once it has been purchased. The digital ticket will serve as a proof -of-entry for physical theater entrance, allowing them to enter and watch the appropriate movies. The software will also be able to print physical tickets for optional carry.

2.3 User Characteristics

Users will be paying customers of the respective theaters that the software will represent. They will only be buying tickets through this software though, other theater services are processed in person. The software will attempt to optimize the interface with the user as much as possible, providing translations of the features in English, Spanish, and Swedish. The user interface will also be as simple as possible, aiding in certain users who may have trouble navigating technological websites.

2.4 General Constraints

The software will be limited by its simplistic nature, possibly offering too little features for users and providing a bare-bones experience. It will also be limited by its budget, possibly making the software not polished and/or bug-free enough by the deadline.

2.5 Assumptions and Dependencies

Making the software compatible with web-browsers and digital kiosks may have a slight chance of increasing software requirements, such as creating a separate version or creating guest modes to keep customers anonymous. The software will be assuming that the most common web-browsers are being used (i.e. Chrome, Edge, Firefox, etc.), and if not, changes will be made accordingly.

3. Specific Requirements

This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project's software design, implementation, and testing.

Each requirement in this section should be:

- *Correct*
- *Traceable (both forward and backward to prior/future artifacts)*
- *Unambiguous*
- *Verifiable (i.e., testable)*
- *Prioritized (with respect to importance and/or stability)*
- *Complete*
- *Consistent*
- *Uniquely identifiable (usually via numbering like 3.4.5.6)*

`

Attention should be paid to the carefuly organize the requirements presented in this section so that they may easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.

3.1 External Interface Requirements

3.1.1 User Interfaces:

3.1.1.1 - The system shall have a web-based interface that is accessible through standard browsers to browse movies, showtimes, and ticket offers.
3.1.1.2 - The system shall have a registration/login system that is secure for both customers and staff/administrators.
3.1.1.3 - The system shall have functionality on both mobile and desktop devices that is usable and responsive.
3.1.1.4 - The system shall display error messages and confirmation messages that are easy to understand.
3.1.1.5 - The system shall comply with the latest WCAG standards for users with disabilities.

3.1.2 Hardware Interfaces:

3.1.2.1 - The system shall support barcode scanning for ticket validation in movie theaters.
3.1.2.2 - The system should be compatible with regular receipt printers for printing tickets and purchase confirmations.
3.1.2.3 - The system shall provide support for kiosks within movie theaters for customers to do on-site ticket purchases.

3.1.3 Software Interfaces:

3.1.3.1 - This system shall integrate a secure payment system to process transactions online.
3.1.3.2 - This system shall interface with a relational database management system such as MySQl to store and manage user information such as their account status, ticket purchases, and movie schedules.
3.1.3.3 - This system shall support the export of sales reports in a format usable for external accounting software, such as CSV format.

3.1.4 Communications Interfaces:

3.1.4.1 - The system shall use HTTPS for any web-based communication so that the privacy and security of user data is maintained
3.1.4.2 - The system shall send email or text message notifications to users about successful ticket purchases
3.1.4.3 - The system shall encrypt all sensitive data transmissions such as payment up to security standards

3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.2.1 <User Registration and Authentication>

3.2.1.1 Introduction - The system shall allow users to create an account, log on with proper security, and manage their account. The authentication shall make sure that users can access personal features like saved payment options and ticket booking history.
3.2.1.2 Inputs - For registration, the system shall take a user-provided name, email, password, and optional phone number. For login, the system shall take a user-provided email and password.
3.2.1.3 Processing - The system shall validate email format and password strength during the registration process, and check if the email is already in the system. It shall also store the password securely and authenticate login credentials against that stored record.
3.2.1.4 Outputs - The system shall output a confirmation message after successfully registering, or an error message if there are issues with the email or password provided. For a successful login, it shall redirect the user to their dashboard, or it shall provide an error if the credentials are invalid.
3.2.1.5 Error Handling - The system shall display errors if the email is already registered, the password does not meet the security requirements, or if the login inputs are incorrect.

3.2.2 <Listing for Movies>

3.2.2.1 Introduction - The system shall display an up-to-date list of currently running and upcoming movies, as well as important information like the title, duration, and showtimes of those movies.
3.2.2.2 Inputs - The user shall send a request to view the movies, such as through the homepage or a listing page. The user shall also be able to apply a filter on their search as mentioned above
3.2.2.3 Processing - The system shall check the movie database for listings that are still active, and apply any filters that the user selects. The system shall then sort the results by release data, availability, or popularity
3.2.2.4 Outputs - The system shall display a readable list of movies with the poster image and title of the movie, as well as the showtimes and a brief description of the movie.
3.2.2.5 Error Handling - The system shall display a “No movies available” message if the filter results in no matches. The system shall also display a general error message if the database is unable to be reached.

3.2.3 <Seat Selection>

3.2.3.1 Introduction - The system shall allow users to view available seats for a selected showtime and pick their ideal seats before moving to payment.
3.2.3.2 Inputs - The system shall allow the user to select a movie, date, and showtime, then offer available seats for the user to pick.
3.2.3.3 Processing - The system shall provide real-time seat availability from the database, and reserve the seats that the user selects temporarily until the user has gone through with payment
3.2.3.4 Outputs - The system shall display a map of all seats, with each being highlighted a different color to indicate either available, booked, or selected. Selecting a seat shall bring up a location name, such as Row 5, Seat H, as well as the total price for all selected seats.
3.2.3.5 Error Handling - The system shall prevent the selection of seats that have already been booked by other users. The system shall also let go of any reserved seats if the session times out, or if the user cancels the selection process.

3.2.4 <Ticket Booking and Payment>

3.2.4.1 Introduction - The system shall let users confirm their booking and make payments through a security payment portal.
3.2.4.2 Inputs - The system shall let the user select a movie, showtime, and seats, then allow the user to input their payment information such as card number and CSV
3.2.4.3 Processing - The system shall calculate the cost of the payment, including taxes and fees. The payment shall be processed through a secure payment gateway, then the booking shall be confirmed and the ticket shall be generated
3.2.4.4 Outputs - The system shall produce a confirmation page with details of the ticket that was purchased by the user, as well as an email confirmation with an attached digital ticket to view and scan.
3.2.4.5 Error Handling - The system shall display an error for a potential payment failure, and automatically let go of any seats that were selected if the payment does not get completed within the specified time

3.2.5 <Managing Tickets and Seats>

3.2.5.1 Introduction - The system shall allow users to view, download, or cancel their tickets, and let administrators manage seat availability.
3.2.5.2 Inputs - The system shall take in the user's login information, and the ticket selections they make. There shall also be separate access for administrators to use management tools
3.2.5.3 Processing - The system shall retrieve the user's history of bookings, and enable the cancellation of tickets within a set time frame. The system shall then update the current seat availability upon cancellation.
3.2.5.4 Outputs - The system shall display a full list of a user's booked tickets with the status of each one. The system shall also display a confirmation message when the user successfully cancels their reservation.
3.2.5.5 Error Handling - The system shall ensure that cancellations after the cutoff time are prevented, and shall display an error if a user's ticket does not exist, or has already been cancelled.

3.2.6 <Administrator Access>

3.2.6.1 Introduction - The system shall provide certain authorized staff with access to administrator features such as creating movie options, managing showtimes, and looking at reports.
3.2.6.2 Inputs - The system shall accept the special login credentials from administrators, and shall allow them to input movie or showtime details to add, update, or delete.
3.2.6.3 Processing - The system shall authenticate users who have administrator privileges, allow CRUD, or Create, Read, Update, and Delete operations on movies and schedules. It shall also generate reports on sales and occupancies of movies
3.2.6.4 Outputs - When an admin makes any changes, the system shall create a confirmation message for successful alterations. It shall also display private dashboards and reports specifically for administrative users.
3.2.6.5 Error Handling - The system shall restrict access to users who are not authorized to access the administrator pages, and display errors for data entries that are invalid.

3.2.7 <Notifications>

3.2.7.1 Introduction - This system shall send out notifications to users for booking confirmations, cancellations, and reminders of their upcoming shows.
3.2.7.2 Inputs - The system shall let events such as successful booking, cancellation, and upcoming shows trigger notifications.
3.2.7.3 Processing - The system shall generate messages that are appropriate for users and related to their trigger events. The system shall also send out text or email notifications through a separate and external service.
3.2.7.4 Outputs - The system shall display a confirmation message through the UI, as well as send out the previously mentioned email or text messages to users.
3.2.7.5 Error Handling - If there are issues with delivery, the system shall try again to send out notifications, and log any notification errors or failures in a report for administrators to review

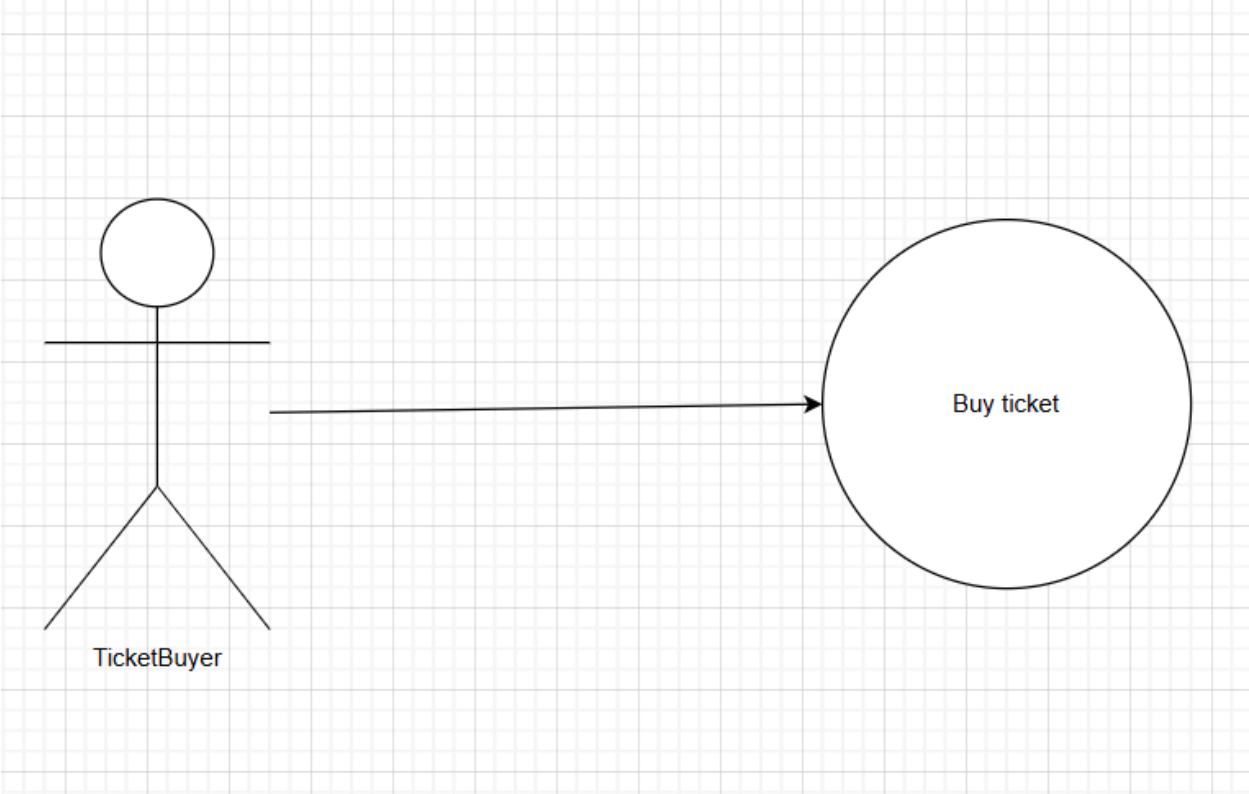
3.2.8 <Accessibility>

3.2.8.1 Introduction - The system shall provide accessibility features to make sure that users with disabilities are able to interact with any public and self-service interfaces, and in compliance with practices such as the Web Content Accessibility Guidelines or any relevant laws.
3.2.8.2 Inputs - The system shall offer assistive technologies for user interactions, such as screen readers, keyboard only navigation, or any visual settings. The system shall also offer accessibility modes or preferences wherever available.
3.2.8.3 Processing - The system shall create UI content with labels and alterations for screen reader compatibility, tab-key navigation for interactive controls, and adjustable color schemes and font sizes for user preferences. The system shall also provide things such as images or buttons for content that might not involve text.
3.2.8.4 Outputs - The system shall create pages and app screens that follow any accessibility criteria. The system shall also create an in-app notification that accessibility options have been enabled.
3.2.8.5 Error Handling - If errors or accessibility features fail to load, the system shall notify the user, and offer suggestions, such as contacting support. The system shall also provide alternatives

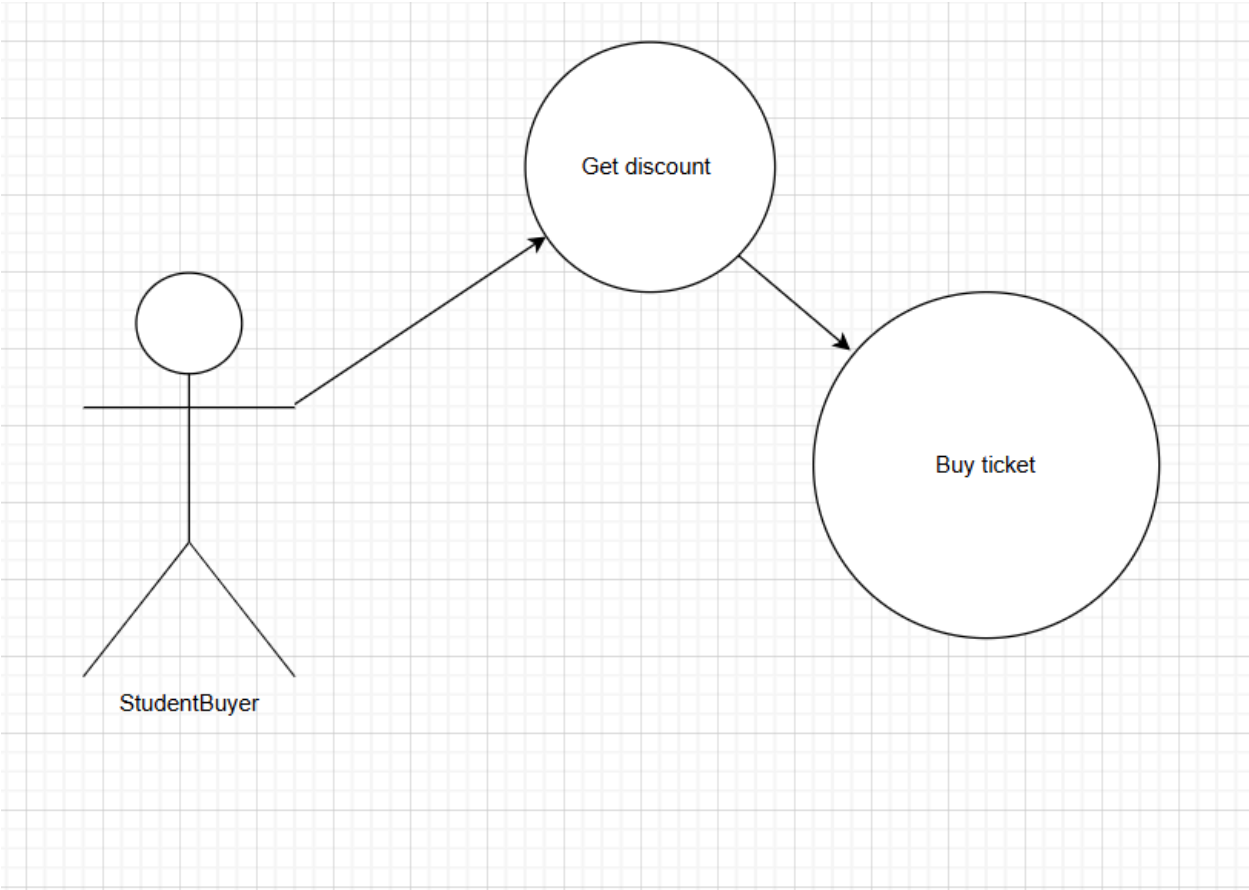
in case essential features run into issues, so that vital functionality can be maintained. Any errors related to accessibility shall be logged in reports to let administrators review.

3.3 Use Cases

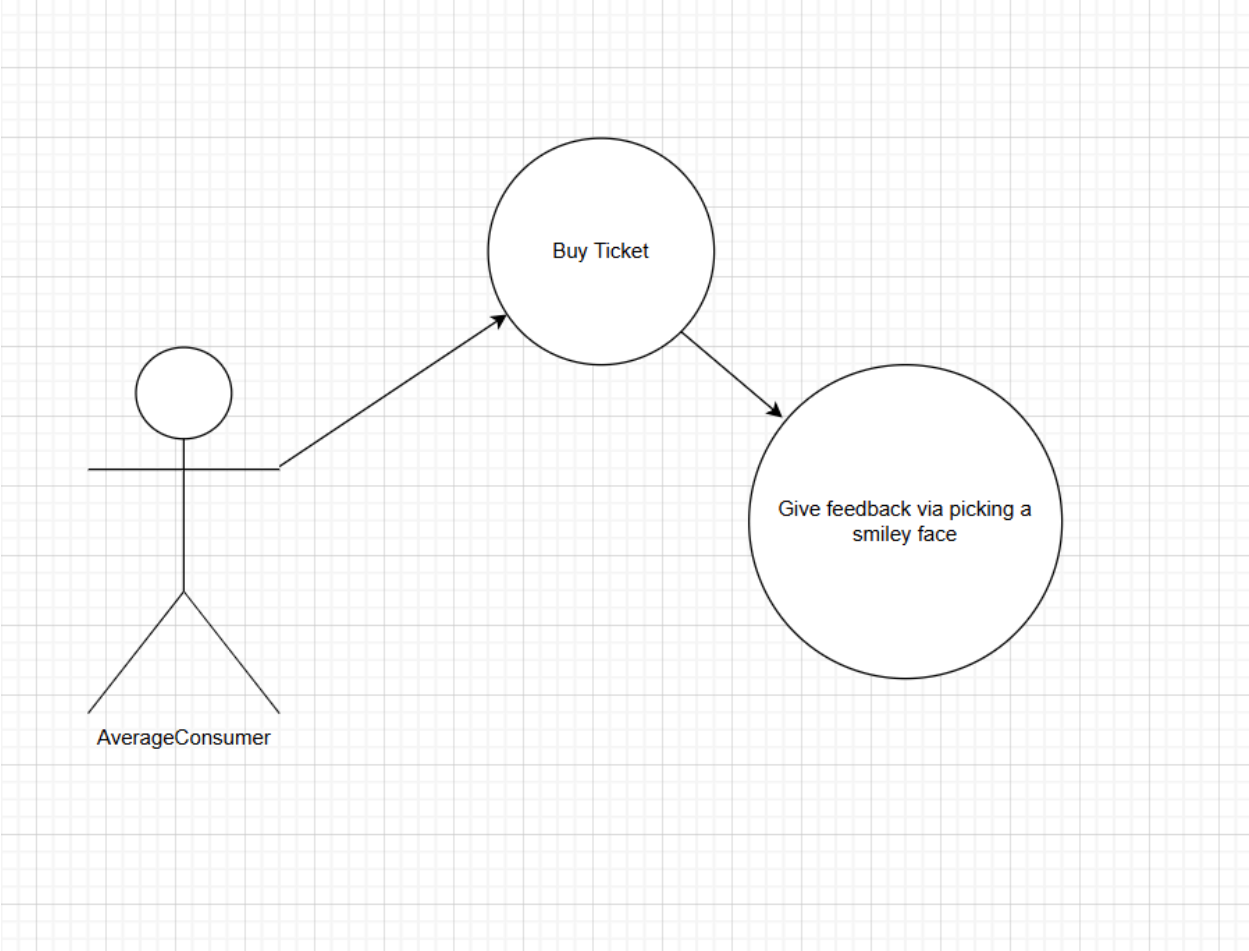
3.3.1 Use Case #1: Ticket Buying



3.3.2 Use Case #2 (Getting a discount)



3.3.3 Use Case #3 (Taking feedback from customers)



3.4 Classes / Objects

3.4.1 Ticket

3.4.1.1 Attributes

- Has name of movie
- Expiration date
- Seat number if deluxe
- Time of movie
- Theatre number

3.4.1.2 Functions

- Acts as proof of entry to a movie theatre
- If its a deluxe ticket, acts as assigned seating in the theatre

3.4.2 Admin

3.4.1.1 Attributes

- Member of the theatre staff
- Employee ID and ranking for further capabilities

3.4.1.2 Functions

- Overrides the system if any issues arise with the customer using the system
- Sets the showtimes of the movies
- Changes the theatre

3.5 Non-Functional Requirements

3.5.1 Performance

- Browser shall be accessed in under 1 second
- Scrolling though the browser shall lag less than 1 second

3.5.2 Reliability

- All information should be present in under one second
- 95% of considered users should be able to navigate the website UI easily

3.5.3 Availability

- There shall at least be 1 update per year, accommodating to feedback from customer surveys

3.5.4 Security

- 95% of hackers that try to hack the website will fail due to the ticketing system noticing the hack
- After ticket is used for the movie, the ticket is removed from the database after 1 second

3.5.5 Maintainability

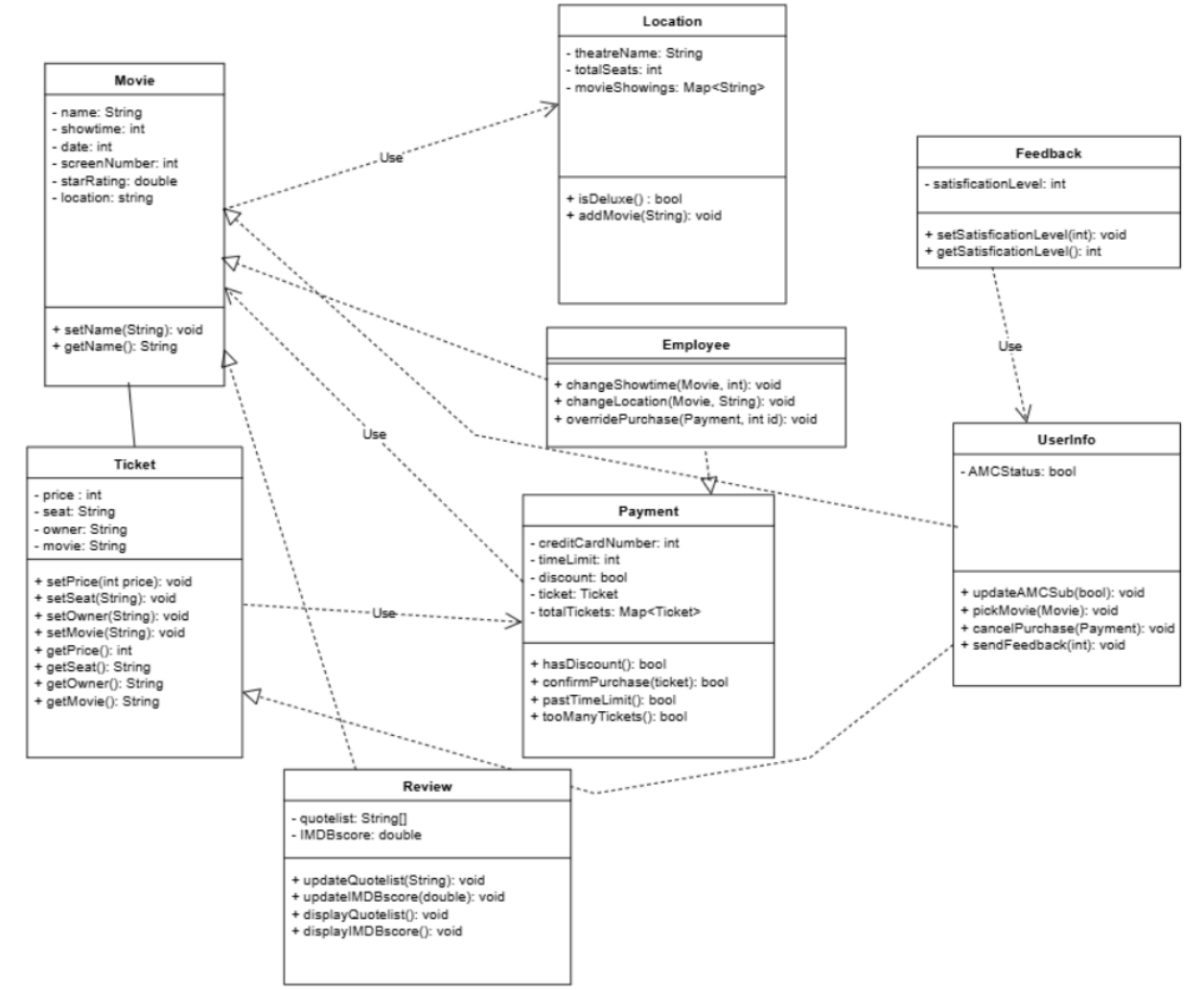
- Servers are maintained at least once a year for 24/7 usage

3.5.6 Portability

- 95% of the computers in various movie theatres should be able to use the browser to sell tickets in under 15 seconds

Design Specifications

UML Diagram



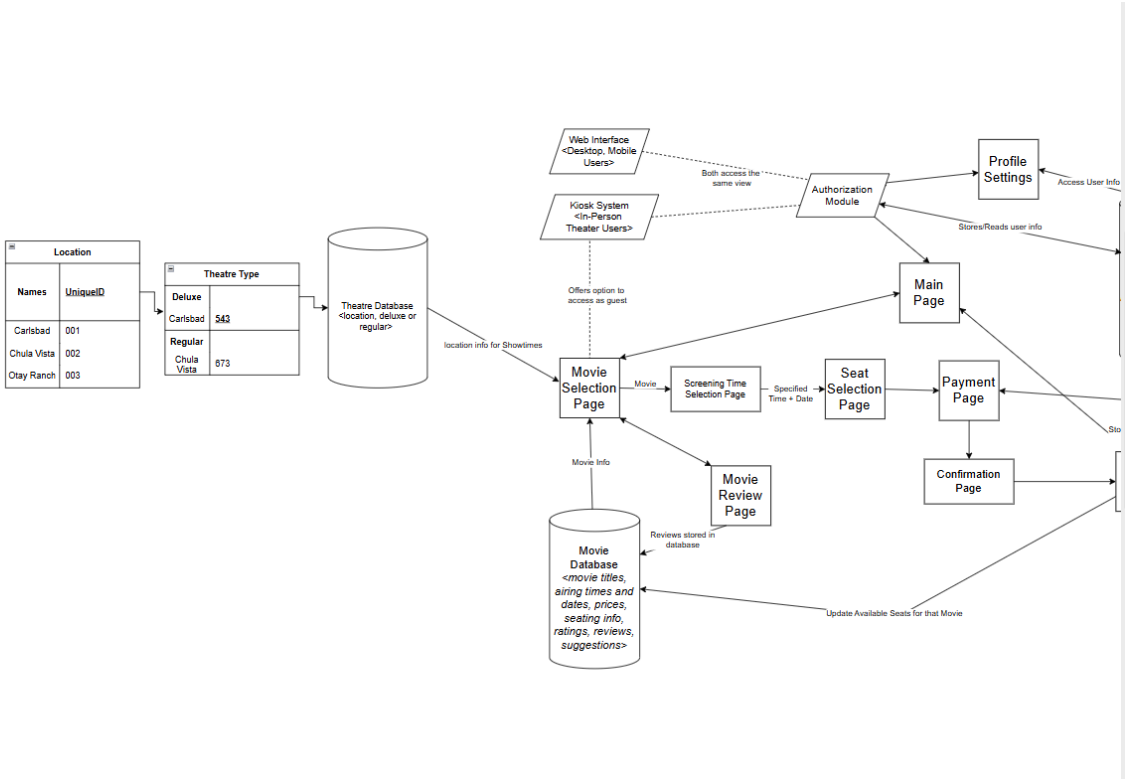
Description

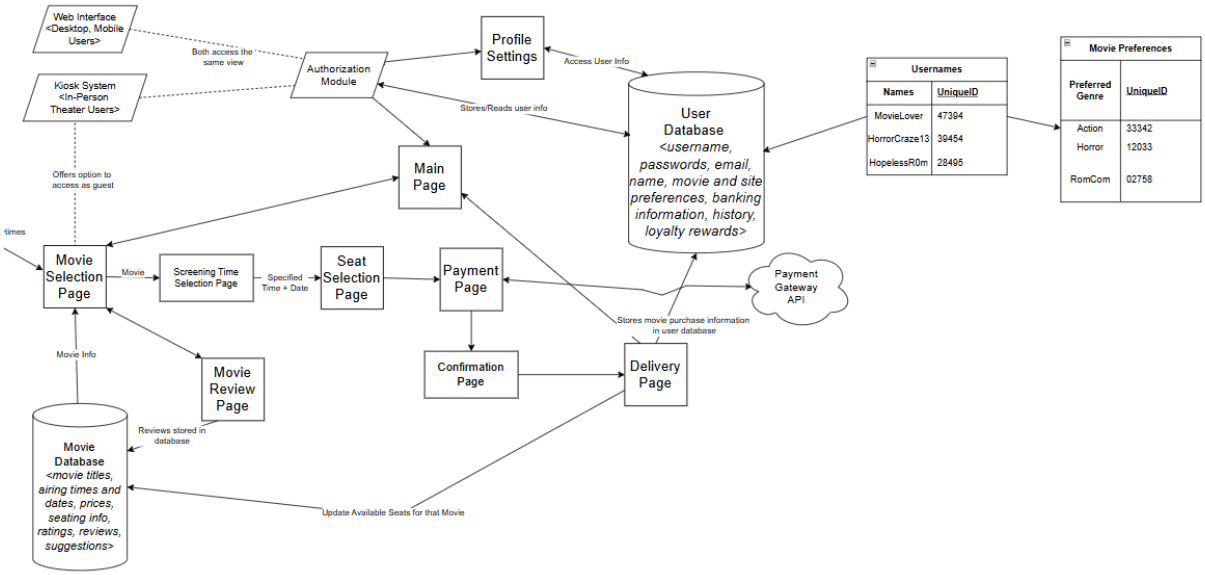
- **Movie Attributes**
  - name: name of the movie
  - showtime: time that the movie starts
  - Date: day that the movie shows
  - screenNumber: where the movie is showing at in the theatre
  - starRating: the rating of the movie extracted from the website, IMDb
  - Location: where the movie is showing at a specific theatre
  - **Methods**

        - setName(String): sets the name of the movie to the given string input
        - getName(): displays the current string name of the movie
- **Ticket Attributes**
    - price: the cost of purchasing the ticket
    - seat: the assigned seat number for the ticket (can be 12A, 7B, etc)
    - owner: who owns the ticket
    - movie: the name of the movie associated with the ticket (taken from Movie Class)
    - **Methods**
        - setPrice, Seat, Owner, Movie: sets the field attributes to a new value according to the desired input
        - getPrice, Seat, Owner, Movie: retrieves the current value of that field variable
- **Review Attributes**
    - quoteList: holds a list of quotes from critics on IMDb
    - IMDBscore: has the score of a movie selected from the Movie Class
    - **Methods**
        - updateQuotelist(String): adds a quote to the list if there's a new quote from IMDb
        - updateIMDBscore(double): changes the current value of IMDBscore to a new value
        - displayQuotelist(): prints all the quotes in the current list to the output
        - displayIMDBscore(): prints the current value in IMDBscore to the output
- **Location Attributes**
    - theatreName: name of the current theatre location in San Diego
    - totalSeats: total amounts of seats in the theatre
    - movieShowings: a list of all the currents movies showing in that location
    - **Methods**
        - isDeluxe: checks if the current location has deluxe seating
        - addMovie(String): adds a movie showing to the movieShowings Map
- **Employee Attributes (none)**
    - **Methods**
        - changeShowtime(Movie, int): changes the showtime of a specified movie
        - changeLocation(Movie, String): changes the location of where the movie will be showing at
        - overridePurchase(Payment, int id): overrides the customer's purchase, given that the employee's id matches the employee database
- **Payment Attributes**
    - creditCardNumber: the credit card info necessary for purchasing a ticket
    - timeLimit: how long the purchasing process should take before it is cancelled due to the showtime
    - discount: if the customer is approved for a discount
    - ticket: the information about the ticket from the Ticket class
    - totalTickets: amounts of tickets the customer is buying (limit is 20)

- **Methods**
  - hasDiscount(): checks the discount variable if it is true or false
  - confirmPurchase(ticket): goes through with the payment and checks if the payment went through
  - pastTimeLimit(): checks if the TimeLimit variable went past 10 minutes
  - tooManyTickets(): checks if the totalTickets amount is past 20
- **Feedback Attributes**
  - satisfactionLevel: the level of satisfaction the customer got from the ticketing system (1 is a sad face, 5 is a happy face)
  - **Methods**
    - setSatisfactionLevel(int) sets the satisfactionLevel variable to a new value
    - getSatisfactionLevel(): retrieves the value of satisfactionLevel
- **UserInfo Attributes**
  - AMCStatus: the current status of a customer's AMC subscription
  - **Methods**
    - updateAMCSub(bool): changes the status of the AMC subscription variable to true or false
    - pickMovie(Movie): picks a movie from the theatre location
    - cancelPurchase(Payment): stops the purchasing process from the customer
    - sendFeedback(int): based on which face is chosen, a number is sent to the Feedback class

SWA Diagram





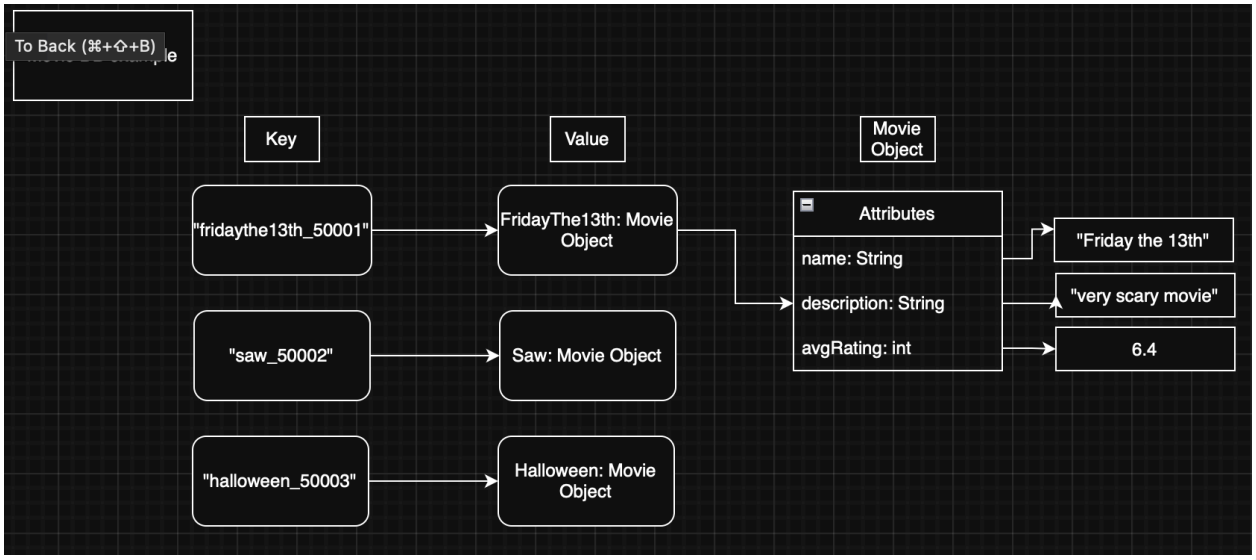
Description

- **Web Interface:** The main access point for customers, enables the user to browse movies, access and register accounts, purchase and book tickets, and manage bookings on both desktop and mobile devices with a responsive and user-friendly website
- **Kiosk System:** A physical self-service terminal within theaters that allows for walk-in ticket purchasings, such as browsing movies and printing tickets, without requiring prior online engagement. Also allows users to physically print tickets depending on the theatre.
- **Login/Signup page:** Accessible only through the online interface; acts as a secure and quick entry point for users to authenticate their already existing accounts, or create new accounts to obtain important information and store it in the User Database for future use.
- **Profile Settings:** User controls for managing their account details such as name, email, password, delivery options, and/or preferred payment methods. Accesses the User Database in order to obtain private information.
- **User Database:** This is a secure repository that stores all of a user's account data, personal information, authentication tokens, and logs of any activity within the entire system, allowing for quick access and retrieval of user information. It uses a noSQL column based database where various factors like usernames and preferences are stored in a table and linked together for easy retrieval.
- **Main Page:** This acts as the homepage that shows off current or upcoming movies, promotional material, and user-friendly navigation for quick access to the ticket purchasing process.
- **Movie Selection Page:** This is a reactive interface used for searching, sorting, or filtering movies by components such as title, date, genre, rating, language, and availability. It is accessible through the main page and provides access to the review page to read

commentary from other users who have viewed the movie, with moderation to remove any unnecessary language.

- **Movie Database:** This is the main storage repository for all information regarding movies, including the titles, descriptions, durations, as well as posters, schedules, and showtimes. This is accessed in the backend and displayed on the movie selection page, holding reviews from the movie review page to give a better understanding of the community response towards certain movies.
- **Theatre Database:** This is a database that lists all the current theatres that use the ticketing system. It is partitioned into different theatres with its own location like Carlsbad or Chula Vista, and whether they are deluxe or regular.
- **Movie Review Page:** This displays user-created movie reviews and ratings, and allows users to submit their feedback on films that they have viewed, providing awareness for other users, as well as data for system analytics. Accessible through the movie selection page, and provides information that is stored in the Movie Database
- **Time & Location Page:** This lets users choose the theater location and showtime for a selected movie, and displays real-time seat availability for each showtime and location.
- **Seat Selection Page:** This acts as an interactive seating map that shows the open and occupied seats for a specified screening, allowing users to pick and reserve their preferred seats, also maintaining real-time updates of seating availability to prevent double booking.
- **Payment Page:** This handles the secure online payments, as well as promotional deals or coupon code entries, and payment methods that were saved within the user's account details. It will launch transaction processes and obtain the success or failure payment responses accordingly.
- **Outside Banking Database:** This manages every financial transaction, payment record, and coupon application through an external banking/payment gateway and processes all payment transactions securely through a secure API call to external providers that handle the authentication, authorization, and fund management. The status of the transaction is then returned to the system, which will update account information, movie information, and allow for the user to continue to the confirmation page. This will be accessed through the backend and is utilized through the ticket payment page.
- **Confirmation Page:** This displays a booking confirmation on successful purchases, and provides a purchase summary with details such as movie showtime, seats, and price details. This will also provide the option for user notifications
- **Delivery Page:** This will deal with the distribution of electronic tickets, booking details, or physical ticket access information, making sure that the user access credentials or a barcode for theater entry through email, text, or a kiosk printout.

Data Management



Description

- For managing our data, we will be implementing the noSQL database approach where three main databases: movie database, user database, and theater database will hold the appropriate data needed to be shared across the entire software. We chose three databases because any less would make the databases hold too much information and any more would unnecessarily complicate the software. noSQL is chosen because these databases will need to hold large amounts of constantly changing data, therefore noSQL's horizontal scalability is highly preferred here. Logically, the data is split up into key-value pairs, possibly a hashmap or dictionary that maps keys to unique objects for further attributes. For readability, the keys will be the names of the movie concatenated with a unique 5 digit number. This is for simplicity, scalability, and efficiency since the software is mainly going to only perform read and write functions. The relationships between the data are already implied based on the database that they are in, so more complicated structures are not needed.

Trade Offs

- While NoSQL offers efficient horizontal scalability, it usually loses strong consistency in favor of eventual consistency, meaning that simultaneous updates might not be immediately updated, and will need brief periods of inconsistency, which will create reservation issues depending on the user size of our application. Additionally, in order to enable efficient access to data without using joins, some information might need to be duplicated across the three databases, which would consume more storage and require cautious update logic to maintain the consistency of the data. The use of our specific key concatenation system would result in more risk if movie names are not unique, such as homonyms, typos, or title changes. Although our simple approach removes the need for

complicated structures to be constructed, it could make future integration efforts more difficult in comparison to explicit, queryable data models.

Development Plan

Kevin Stratton:
He is tasked with developing and managing the backend development of the ticketing system. He makes sure that the system is well secured and can resist penetration from various bots or hackers

Ryan Haigh:
He is tasked with the development and management of the frontend. He will supervise the designs of the UI and make sure the UI is smooth and easy to use for the client and customers eventually.

Phu Duong:

She is tasked with managing the database that the system will use to display the movies and theatres. She will also make sure that the database is accurate based on the data from IMDb and customer feedback

Development Timeline

In a span of 4 months, it is expected that by:

Week 4: The backend system is coded and tested thoroughly for bugs with system testing
Week 6: The design for UI is approved and development for frontend begins
Week 8: The frontend is fully developed and tested via unit and integration tests
Week 12: Established a deal with AMC for using their subscription alongside the ticketing system. Further testing is also done for smoothing out bugs
Week 16: The prototype system is proposed to the client. If approved, yearly updates will be in line for the system

Test Plan

For this software to operate as intended, rigorous testing would be required. This'll be done via formal methods and user testing. First, our backend developer (Kevin) will test the classes individually to see if their methods work on their own. For example, we will test the movie class to see if it can set and get the name of the movie. In another instance, the ticket class will be

tested on its set functions to see if it works correctly. After testing the classes on their own, we will test their functionality when working together. This is where our frontend developer, Ryan will work in tandem with Kevin to test the intersectionality of classes. One example includes sending feedback in the Userinfo class. Then the feedback should be sent to the Feedback class to record and receive to us. Another example would be changing a showtime in the Movie class using the Employee class. After testing intersectionality, the entire system would be tested for its overall functionality through common and expected processes. This would include buying a ticket, overriding a purchase, buying a deluxe seat to test assigned seating, and displaying the IMDb score for a movie on the website. If the testing turns out to be a success, we can move on to user testing for validation. This'll be done either with volunteers, random moviegoers in our theatre system, and/or paid QA testers who'll give us feedback and bug reports. After this, we'll apply the feedback to our own discretion and further polish the software accordingly.

Test Cases

Test Cases Excel Sheet

Github Link

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

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.

3.9 Other Requirements

Catchall section for any additional requirements.

4. Analysis Models

List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS's requirements.

4.1 Sequence Diagrams

4.3 Data Flow Diagrams (DFD)

4.2 State-Transition Diagrams (STD)

5. Change Management Process

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

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