Software Requirements Specification Template

Software Engineering

The following annotated template shall be used to complete the Software Requirements Specification (SRS) assignment.

Template Usage:
Text contained within angle brackets (‘<’, ‘>’) shall be replaced by your project-specific information and/or details. For example, <Project Name> will be replaced with either ‘Smart Home’ or ‘Sensor Network’.

Italicized text is included to briefly annotate the purpose of each section within this template. This text should not appear in the final version of your submitted SRS.

This cover page is not a part of the final template and should be removed before your SRS is submitted.

Screen Seat Ticketing

Software Requirements Specification

Version 4

03/28/2024

Group #8
Sawyer Jones, Will Ruff, & Alexander Prochazka

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Spring 2024

Revision History

Date	Description	Author	Comments
02/01/2024	Version 1	Sawyer Jones, Will Ruff, Alexander Prochazka	First draft of document
02/29/2024	Version 2	“ ”	Sections 3.7, 4.1, 4.2, 4.3 added to document
03/14/2024	Version 3	“ ”	Section 5 added
03/28/2024	Version 4	“ ”	Section 6 added, architecture diagram updated

Document Approval

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

Signature	Printed Name	Title	Date
	Alexander Procha…	Software Student	2/15/24
	Dr. Gus Hanna	Instructor, CS 250	
	Will Ruff	Software Student	2/15/24
	Sawyer Jones	Software Student	2/15/24

Table of Contents

1. Introduction

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire SRS, outlining the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the SRS. This document aims to comprehensively detail the requirements and functionalities of the movie theater ticket-buying platform. It delves into the problem statement, analyzes stakeholder needs, and defines high-level product features to guide the development process. The in-depth detailed requirements of Screen Seat Ticketing are provided in this document.

1.1 Purpose

This document serves as a gathering and consolidation of system requirements for a movie theater ticket-buying platform. It acts as a source for diverse ideas, predicting usage patterns, and documenting evolving concepts. Moreover, the SRS describes the target audience, user interface intricacies, and hardware/software prerequisites of the platform. It stands as a map guiding the software development lifecycle, aiding designers and developers in steering towards the project's conclusion.

1.2 Scope

The software product to be produced is Screen Seat Ticketing E-Store. The E-Store will facilitate online ticket purchases for moviegoers. This will allow them to browse available movies, select showtimes, choose seating options, and securely complete transactions. Additionally, the platform will provide users with access to relevant movie information. This information will include summaries, and ratings, to aid in their decision-making process. The primary goal of the E-Store is to streamline the ticket purchasing process for both users and theater operators. It aims to enhance user convenience and accessibility by providing a user-friendly interface for browsing and buying tickets online. Specific objectives include:

A. Providing a seamless and intuitive ticket purchasing experience, including options for seat selection, payment processing, and receipt issuance.
B. Enabling users to access real-time information on movie showtimes, availability, and promotions.
C. Enhancing theater revenue and customer engagement through targeted marketing campaigns and personalized offers.
D. Improving operational efficiency for theater administrators by automating ticket sales and reducing manual processes.

The scope of the Screen Seat Ticketing E-Store aligns with higher-level specifications, such as the System Requirement Specification, ensuring consistency and coherence across all project deliverables.

1.3 Definitions, Acronyms, and Abbreviations

FAQ	Frequently Asked Questions
IMDb	Internet Movie Database
API	Application Programming Interface
FIFO	First In First Out
HTTPS	Hypertext Transfer Protocol Secure

1.4 References

The References are:

- ➔ Software Requirements Specification document with example
- ➔ SRS4.0-1.doc
- ➔ Theatre Ticketing Requirements.rtf
- ➔ IEEE-830-1998-1.pdf

1.5 Overview

The rest of this document gives more details about the project, including who will be using it, what hardware and software it will need, and what specific features it will have. It's like zooming in on the different aspects of the project to get a better understanding of how everything will work together. In section 2 the general description of the product will be provided. Section 3 will provide the different types of requirements and use cases we must take into account while designing our E-Store. Section 4 will lay out the supporting information we need and the overall process that needs to be executed. Finally, section 5 will explain how we will update the SRS as a whole.

2. General Description

2.1 Product Perspective

A browser-based ticketing application that stores the following

- Customer description
  It includes user email addresses and passwords, saved delivery methods, and saved payment information.
- Ticket information
  It includes the amount of tickets available and the amount of available seats in the theater.
- Reservation description

It includes the customer details, date of reservation, seat and theater information, and date of purchase.

2.2 Product Functions

Users of the system will have access to the following functions:
2.2.1 Function that allows the user to create an account based on name, email, and password.
2.2.2 Function that allows existing users to log into existing accounts.
2.2.3 Function that allows users to search for available tickets based on data and genre.
2.2.4 Function that will display information about available tickets, like price and seating options.
2.2.5 Function that will allow users to securely purchase tickets and update the internal ticketing database.
2.2.6 Function that will provide users with a confirmation email or receipt upon purchase.
2.2.7 Function that will allow users to switch between supported languages (English, Spanish, Swedish).
2.2.8 Function that will allow users to specify if they are eligible for a discount.
2.2.9 Function that will act as a checkout timer giving the user 5 minutes to hold tickets in their cart.
2.2.10 Function that will act as a feedback system for users after checkout.

2.3 User Characteristics

Users of our system should retrieve available tickets from the database. We will have two classes of users, User and Administrator, as well as a Kiosk class. Users will be given customer functions, while Employees will have theater management functions and customer functions. The customer should have access to the following functions:

- Make a reservation
- Cancel reservations
- View reservations

The employee should have access to these management functions:

- Customer functions
    - Get all customers who have reservations for specified showing
    - View movie schedule
    - Calculate total sales per showing
- Employee functions
    - Add/Delete showings
    - Update ticket pricing
    - Update showing times

2.4 General Constraints

- Regulatory policies;

    - Rotten Tomatoes and IMDb do not allow web scraping for commercial use. An API must be purchased to access movie data
    - Duplication and reselling are not allowed and should be made impossible through proper implementation.
- Hardware limitations;
    - Database storage capacity
        - Up to 10 million accounts, with personal information, purchase history, and loyalty points
        - Daily logs of all tickets purchased
        - All of the theaters in 20 locations
            - Deluxe or regular options
        - Showtimes in each theater for the next two weeks
            - Location info
            - Ticket prices for each showtime
            - Array with seat numbers
- Interfaces to other applications;
    - IMDb Developer: API which allows access to movie names, and a daily updating rating
    - API for Paypal processing, as well as crypto wallets.
    - API to handle credit card transactions
    - CAPTCHA software to prevent bots
- Parallel operation;
    - Only one device per account. If a login occurs on one device, the user will be signed out of all other devices
    - Up to 1000 people should be able to access the site at once. The database should be able to handle multiple queries simultaneously
    - If there is an unusually high number of users, a queue system will be created and handle requests in a FIFO manner.
- Reliability
    - Low to zero downtime. downtime should occur during early morning (3:00 AM)
    - Sales should be confirmed by the bank first, then processed by the application
- Criticality
    - System failure would cause major financial loss for the theaters
    - Data breaches risk users personal information
- Safety and security
    - Credit card information needs to be especially protected
    - Critical information should be backed up on another server, in case of emergency.
    - HTTPS internet protocol for secure connection

2.5 Assumptions and Dependencies

- Browser Assumptions

    - Users will be using one of four common browsers, Chrome, Edge, Firefox, or Safari. Software should be optimized to work on each.
    - There will be no mobile app. Instead, the website will have an optimized layout for mobile browsers.
- In theaters, a self-serve kiosk will be used. It will run Google Chrome in kiosk mode. There will be no login, refund, or digital ticket options for kiosks.
- Online purchases depend on multiple APIs listed in section 2.4.
- There are 20 theaters, and the layout of the theaters does not change. If there is a remodel or new theater built, the list of theaters will need to be updated.
- System would have to be adjusted following a redesign in the theater's business model or ticketing options

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.1.1 The system will have a uniform design between all pages.
3.1.1.2 The system will make use of icons to represent different pages.
3.1.1.3 The system will be responsive to different screen sizes and resolutions.

3.1.2 Hardware Interfaces

3.1.2.1 The system will not have strict hardware specifications. It will be accessible to anyone, on most devices, with access to one of the designated browsers and an internet connection.

3.1.3 Software Interfaces

3.1.3.1 The system will store customer information in a database.
3.1.3.2 The system will display daily showings.
3.1.3.3 The system will allow users to select their seat(s) when applicable.
3.1.3.4 The system will be integrated with email software to deliver user tickets.
3.1.3.5 The system will prevent any transaction involving more than 20 tickets.
3.1.3.6 The system will be integrated with a database to store ticketing information.

3.1.4 Communications Interfaces

3.1.4.1 The system will use HTTPS to ensure a secure connection between the user and the server.
3.1.4.2 The payment system will be encrypted to provide payment security.
3.1.4.3 The system will use APIs for seamless communication between different components.

3.2 Functional Requirements

3.2.1 E-Store must run in a web browser (not as an app)

3.2.1.1 Introduction
The E-Store requirement mandates web browser compatibility for accessibility and user interaction.
3.2.1.2 Inputs

User commands and requests are inputted through web browsers for browsing and purchasing tickets.
3.2.1.3 Processing
The system validates inputs, retrieves movie information, and executes transactions securely.
3.2.1.4 Outputs
Rendered web pages display movie listings, showtimes, and transaction confirmations.
3.2.1.5 Error Handling
Error messages are displayed if issues occur during processing to ensure a smooth user experience.

3.2.2 E-Store must be able to handle at least 1000 people at once

3.2.2.1 Introduction
The E-Store infrastructure must support concurrent access by a minimum of 1000 users.
3.2.2.2 Inputs
Users accessing the platform concurrently generate multiple requests and transactions.
3.2.2.3 Processing
The system manages user sessions and database queries efficiently to handle the load.
3.2.2.4 Outputs
Prompt responses to user queries are delivered, ensuring a seamless user experience.
3.2.2.5 Error Handling
Error handling mechanisms minimize disruption and maintain optimal performance.

3.2.3 E-Store has to block bots trying to buy a lot of tickets to high-demand movies

3.2.3.1 Introduction
The E-Store must implement measures to prevent bots from purchasing an excessive number of tickets for high-demand movies.
3.2.3.2 Inputs
Detection of abnormal ticket purchasing behavior indicating bot activity.
3.2.3.3 Processing
The system identifies and blocks suspicious transactions associated with bots.
3.2.3.4 Outputs
Prevention of unauthorized bulk ticket purchases, ensuring fair access for genuine users.
3.2.3.5 Error Handling
Error messages may be displayed if legitimate users are mistakenly flagged as bots.

3.2.4 E-Store must have an interface with a database of showtimes and tickets available

3.2.4.1 Introduction
The E-Store requires seamless integration with a database containing showtimes and ticket availability information.
3.2.4.2 Inputs
Data queries and updates to retrieve and manage showtime and ticket data.
3.2.4.3 Processing
The system processes database queries and updates to ensure accurate and up-to-date information.
3.2.4.4 Outputs
Display real-time showtime and ticket availability information to users.

3.2.4.5 Error Handling
Error handling mechanisms address database connectivity issues or data inconsistencies.

3.2.5 Tickets are only available 2 weeks prior to showtime and 10 minutes after showtime

3.2.5.1 Introduction
Ticket availability is restricted to a specific timeframe to manage demand and ensure timely purchases.
3.2.5.2 Inputs
User requests for ticket purchases within the designated time frame.
3.2.5.3 Processing
The system validates ticket purchase requests based on the specified timeframe.
3.2.5.4 Outputs
Confirmation or rejection of ticket purchases based on availability windows.
3.2.5.5 Error Handling
Error messages are displayed if users attempt to purchase tickets outside the designated time frame.

3.2.6 User can only buy 20 tickets max at a time

3.2.6.1 Introduction
Limitation on the maximum number of tickets that a user can purchase in a single transaction to prevent abuse and ensure fair access to tickets.
3.2.6.2 Inputs
User input indicating the desired quantity of tickets for purchase.
3.2.6.3 Processing
The system validates the quantity of tickets requested against the maximum limit.
3.2.6.4 Outputs
Confirmation or rejection of the transaction based on the maximum ticket limit.
3.2.6.5 Error Handling
Error messages are displayed if users attempt to purchase tickets exceeding the maximum limit.

3.2.7 E-Store must be able to support administrator mode

3.2.7.1 Introduction
The E-Store must include an administrator mode to enable management and maintenance tasks.
3.2.7.2 Inputs
Administrative commands and requests initiated by authorized users.
3.2.7.3 Processing
The system processes administrative tasks, such as adding or removing movie listings and managing user accounts.
3.2.7.4 Outputs
Confirmation of successful execution of administrative tasks.
3.2.7.5 Error Handling
Error messages are displayed if there are issues with executing administrative tasks.

3.2.8 E-Store must have a customer feedback system

3.2.8.1 Introduction

The E-Store requires a feedback system to gather and analyze user feedback for continuous improvement.
3.2.8.2 Inputs
User feedback is provided through the feedback system.
3.2.8.3 Processing
The system processes and aggregates user feedback for analysis.
3.2.8.4 Outputs
Reports and analysis of user feedback to identify areas for improvement.
3.2.8.5 Error Handling
Error messages may be displayed if there are issues with collecting or processing user feedback.

3.2.9 E-Store must support different discounts for tickets

3.2.9.1 Introduction
The E-Store must accommodate various discount offers for tickets to incentivize purchases.
3.2.9.2 Inputs
Discount codes or promotional offers entered by users during the ticket purchasing process. Must also accommodate discounts for those eligible, like students, veterans, and seniors. Status is stored in the User class.
3.2.9.3 Processing
The system applies discounts to eligible ticket purchases based on predefined rules and conditions.
3.2.9.4 Outputs
Confirmation of discounted ticket prices and total transaction amount.
3.2.9.5 Error Handling
Error messages are displayed if users encounter issues with applying or redeeming discounts.

3.2.10 E-Store must be able to display reviews and critic quotes of movies

3.2.10.1 Introduction
The E-Store requires functionality to collect and showcase movie reviews and critic quotes sourced from online review platforms.
3.2.10.2 Inputs
User requests for movie reviews and critic quotes for specific films.
3.2.10.3 Processing
The system utilizes IMDb Essential Metadata, requesting the title, movie rating, maturity rating, synopsis, thumbnail, and lead actors.
3.2.10.4 Outputs
Displays the requested data on a single webpage, along with a button to begin the purchasing process.
3.2.10.5 Error Handling
If no data exists for a category, a placeholder image/text is used instead.

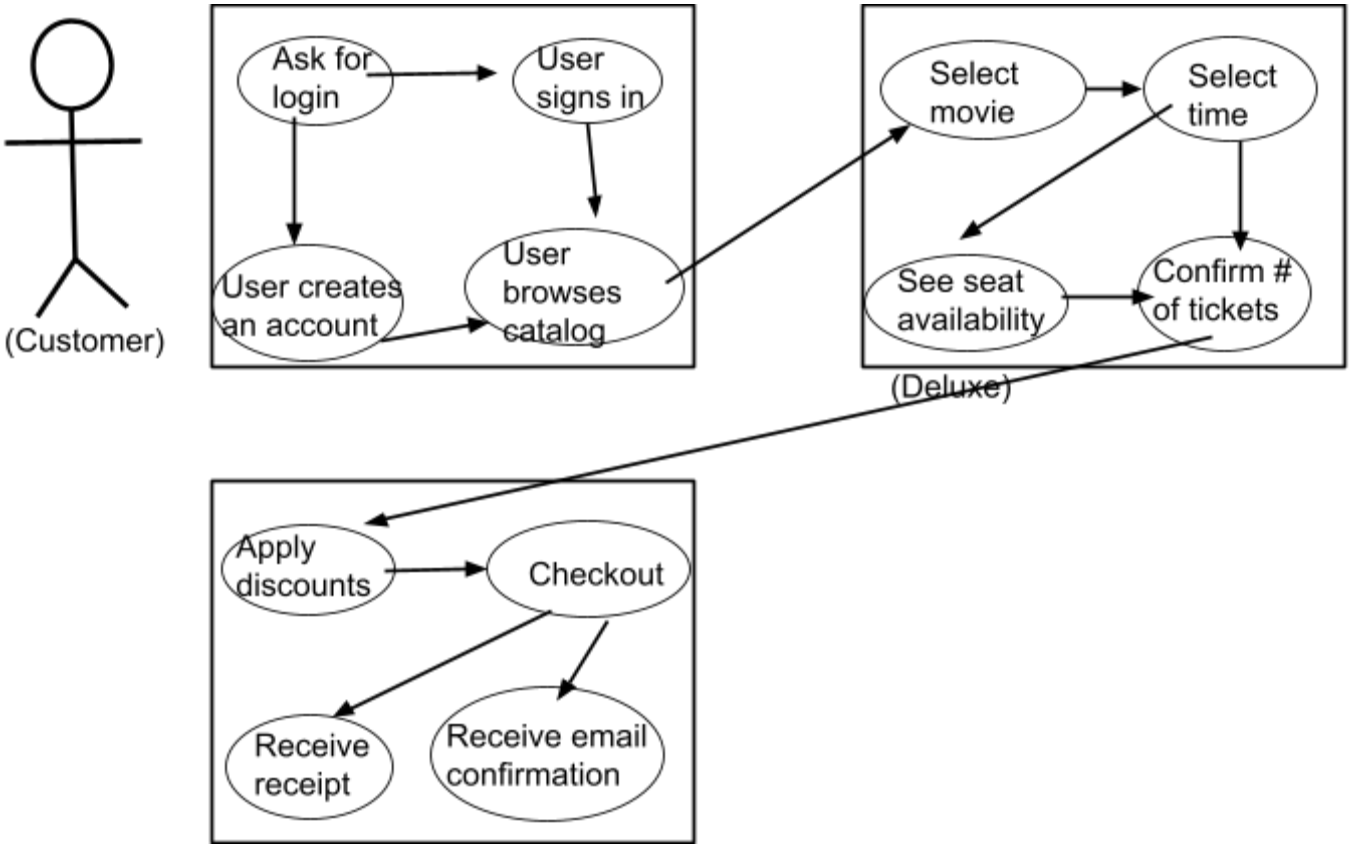
3.2.11 E-Store must provide customer support

3.2.11.1 Introduction
This requirement mandates the inclusion of customer support features, such as an FAQ section, within the E-Store to address user inquiries, issues, and concerns effectively.
3.2.11.2 Inputs

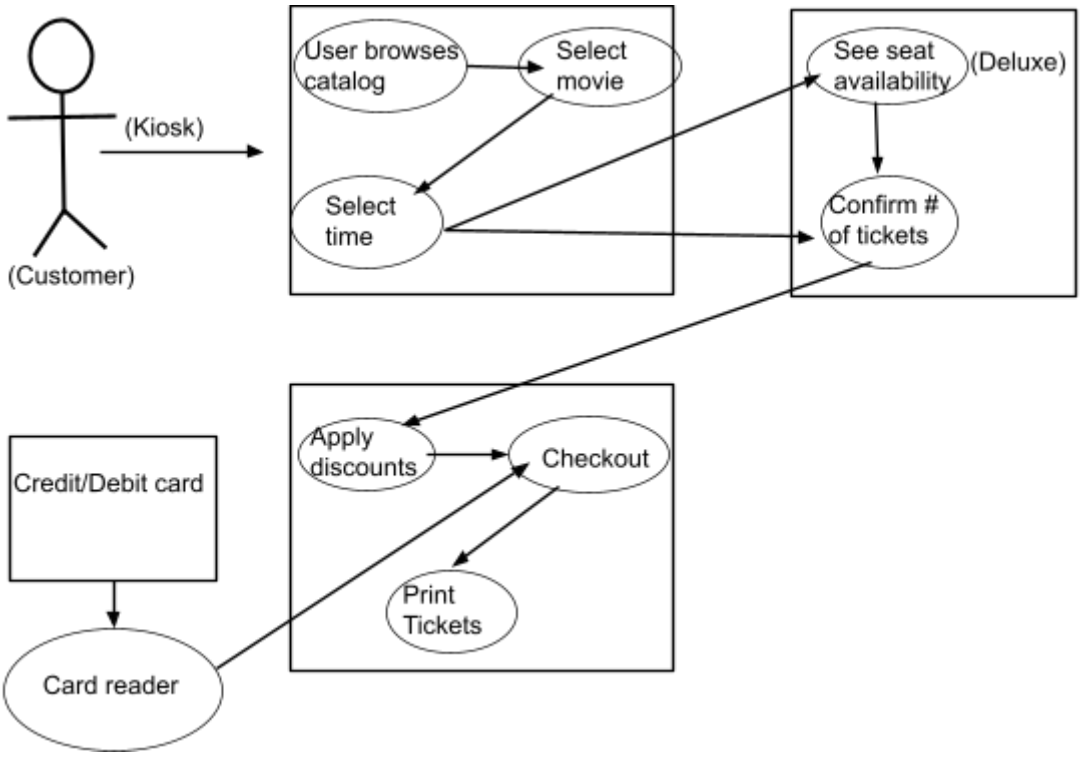
User queries, complaints, and feedback submitted through designated channels, such as email, chat, or phone.
3.2.11.3 Processing
The system processes incoming inquiries, categorizes them in a priority queue based on their nature and urgency, and routes them to appropriate support channels or personnel for resolution.
3.2.11.4 Outputs
Timely and helpful responses provided to users' inquiries, complaints, or feedback, ensuring a positive customer experience.
3.2.11.5 Error Handling
Error handling procedures are in place to address any technical issues or delays in responding to customer queries, ensuring prompt resolution and customer satisfaction.

3.3 Use Cases

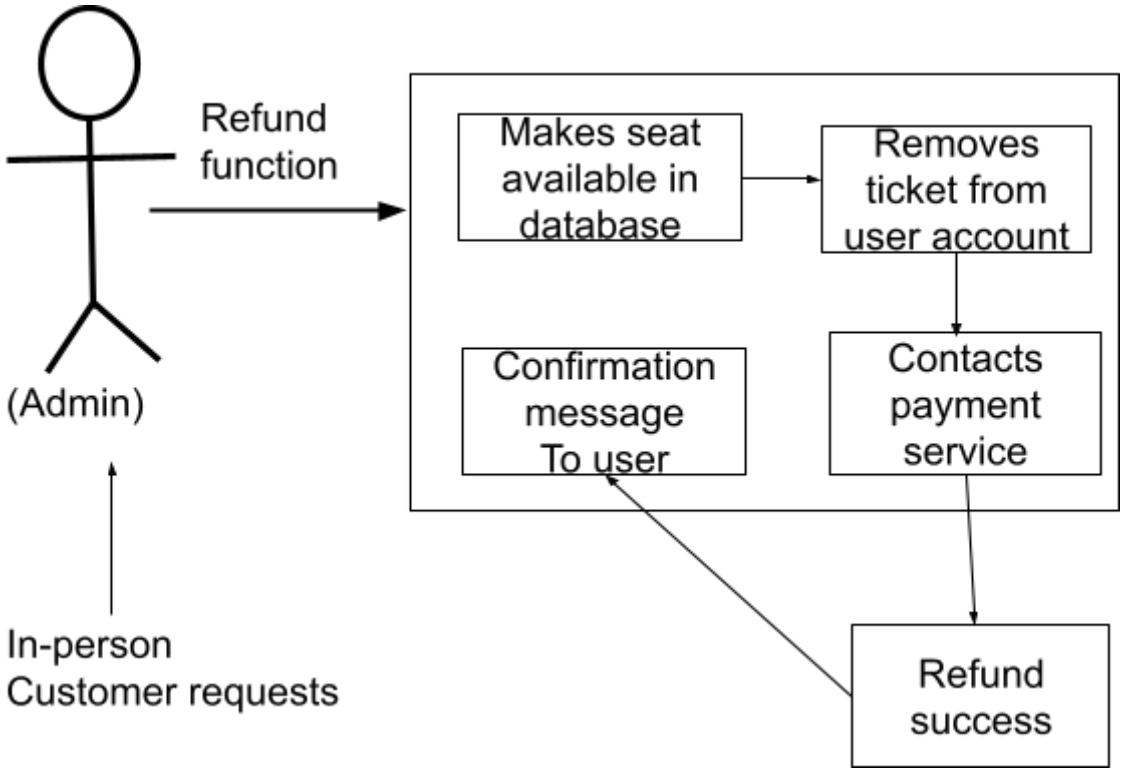
3.3.1 User Purchases Ticket Online: Online purchases are handled in a multi-step process. The user browses, enters specific details about their purchase, and then pays.



3.3.2 Customer Buys From Kiosk: There is no login this time. Rather, the user inserts a card for processing into the kiosk, which is read and confirmed. Then the machine prints physical tickets



3.3.3 Administrator Provides Refund: Administrators can process refund requests, rather than the user themselves. It interacts with the theater database, the user's account, and the payment API to ensure a refund is processed system-wide and the seats are free for purchase again.



…

3.4 Classes / Objects

3.4.1 User

3.4.1.1 Attributes

- Username/password
- Shopping cart
- Payment method
- Purchase history
- Special Status (military, youth, student, etc.)

3.4.1.2 Functions

- Browse
    - Search database
    - Sort by(ascending/descending): rating, popularity, name, date
    - Filter by category, maturity rating
    - View showtimes for each film
- Purchase tickets
- View wallet
- View purchase history

3.4.1.3 References

- Use case 3.3.1
- Functional requirement 3.2.9

3.4.2 Administrator

3.4.2.1 Attributes

- Username/password
- History of changes to database
- List of theaters user is responsible for

3.4.2.2 Functions

- Browse
- Edit Database
    - Change customer seats upon customer request
    - Delay showtime of movie
- Refund customer tickets
- Create showings

3.4.2.3 References

- Use case 3.3.3
- Functional requirement 3.2.7

3.4.3 Kiosk

3.4.3.1 Attributes

- User/password
- log of sales

3.4.3.2 Functions

- Display movies (browse)
- Print tickets
- Accept Payment

3.4.3.3 References

- Use case 3.3.2

3.5 Non-Functional Requirements

3.5.1 Performance

- The system shall process ticket transactions with an average response time of less than 3 seconds.
- Concurrent user capacity shall support a minimum of 1000 users during peak hours.
- The platform shall maintain a minimum throughput of 100 transactions per minute during peak usage periods.
- Response time for loading the ticket purchasing page shall not exceed 1.5 seconds.
- Search functionality for movie listings shall return results within 2 seconds.

3.5.2 Reliability

- The system shall have a mean time between failures (MTBF) of at least 30 days.
- Critical system failures shall be addressed and resolved within 15 minutes of detection.

- Backups of user data and transaction records shall be performed daily, with a recovery time objective (RTO) of less than 1 hour.
- The platform shall have a fault tolerance rate of 99.9% to ensure uninterrupted service availability.

3.5.3 Availability

- The system shall be available for access 24/7, with scheduled maintenance windows communicated to users in advance.
- Planned system downtime for maintenance shall not exceed 1 hour per week.
- In the event of unplanned downtime, the platform shall have an automatic failover mechanism to redirect users to a backup server within 30 seconds.

3.5.4 Security

- User authentication shall be performed using industry-standard encryption protocols to ensure secure access to the platform.
- Payment transactions shall be processed using PCI-DSS compliant methods to safeguard sensitive financial information.
- The platform shall implement role-based access controls (RBAC) to restrict unauthorized access to sensitive data and administrative functions.
- Regular security audits and vulnerability assessments shall be conducted quarterly to identify and address potential security risks.

3.5.5 Maintainability

- Codebase shall adhere to industry best practices and coding standards to facilitate ease of maintenance and future enhancements.
- System documentation, including architecture diagrams and technical specifications, shall be regularly updated to reflect any changes or updates.
- The platform shall have a modular architecture to enable seamless integration of new features and functionalities without disrupting existing system operations.

3.5.6 Portability

- The platform shall be compatible with major web browsers, including Chrome, Firefox, Safari, and Edge, on both desktop and mobile devices.
- Application deployment shall be platform-agnostic, allowing for deployment on cloud-based infrastructure or on-premises servers.
- The platform shall support multi-language and multi-currency capabilities to accommodate international users and markets.

3.6 Inverse Requirements

- The system shall not allow unauthorized access to user accounts or sensitive data.
- User sessions shall not time out during active ticket purchasing processes to prevent loss of data and inconvenience to users.
- System maintenance activities shall not impact user access or functionality during peak usage periods.

- The system shall not allow simultaneous ticket bookings for the same seat to avoid conflicts and discrepancies in seating arrangements.

3.7 Design Constraints

3.7.1 Our team is limited in size, so larger tasks may take longer to complete.
3.7.2 Our team does not have an adequate budget to support vast expansion and growth.
3.7.3 Our team is full of full stack developers, so there is no one with extreme expertise in one area.
3.7.4 We won't be able to test bank transfer functionality since we won't have access to bank software.
3.7.5 Our software will be limited to our selected browsers, Safari, Chrome, Edge and Firefox.
3.7.6 Our software will be limited to devices that can connect to the internet.

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

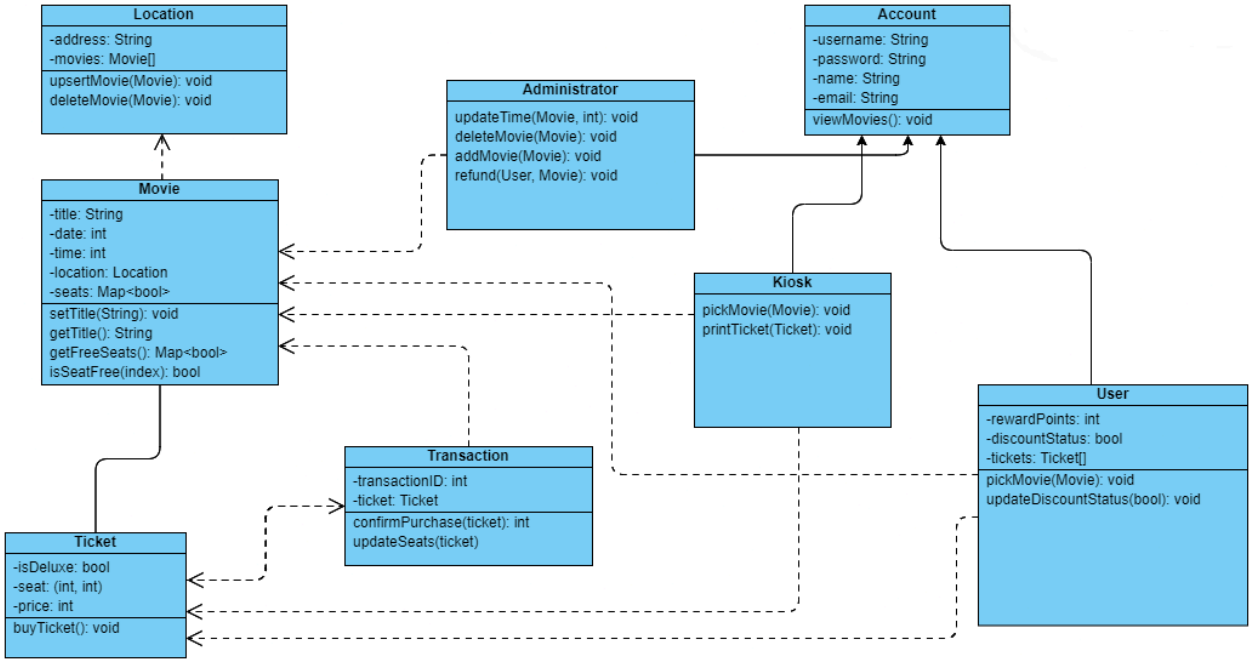
4.1 System Description

Screen Seat Ticketing allows users to conveniently purchase tickets for movies online. Users can browse available movies, view showtimes, select seats, and make secure payments through the website.

Data is kept on a database server. Three types of accounts, Users, Kiosks, and Administrators, all interact with the server. Movies are at a time and place, and each contains a map of tickets. Tickets are purchased, and orders are kept track of through a Transaction object.

4.2 Software Architecture Overview

4.2.1 UML Class Diagram:



In our software, the Account class functions as the parent class to Administrator, Kiosk, and User classes. The Account class contains information about the user, such as username, password, name, and email. This information is critical to our software. Within this class, we also have the viewMovies method, which lets users see all available movies. The User class has 3 inherent variables, rewardPoints, discountStatus, and tickets. These variables can be modified using the methods pickMovie and updateDiscountStatus. The tickets variable will be updated through the Ticket class.

The Ticket class has 3 variables as well, including isDeluxe, seat, and price. It also contains the buyTicket method that will allow the user to purchase tickets. A subclass of the Ticket class is the Movie class, which contains variables for the title, date, time, location, and seats available for each movie. We have getter methods for the title and the available seats, along with a setter for the title. We also have a boolean that can check if an individual seat is available. The last class involving the Movie class is the Location class. This class holds the address of the current theater and an array of movies available at that location. We also give the location class the ability to add or delete movies from the available movie array.

Another class that interacts with the Ticket class is the Transaction class. This class contains secure variables that hold the transaction ID and the ticket(s) purchased. It also has methods that allow for seating to be updated, as well as a method to confirm if the ticket has been confirmed. We will also have a special class for our Ticket Kiosks, which will function in place of the User class when using an in-house kiosk. This class is a subclass of the Account class. This class will contain methods to select a movie and print the ticket once the transaction has been completed. The final child class of the Account class is the Administrator class. This class contains exclusively methods, which will allow us to modify movie showings. We will be

able to update, add, or delete any of our theater's movies. It also has the ability to provide refunds to customers.

4.3 Development Plan and Timeline:

To start, the classes and objects will all be built, to the specifications of the UML diagrams. The UML diagram may need to be updated if unanticipated needs arise (new parameters, methods, entire objects).

Secondly, a database will be built, using the Location, Movie, Ticket, and Transaction classes as various types of entries. Account objects will exist as a separate list in the database. The methods contained within Account classes will be responsible for interacting with the database.

Next, the services will be connected to the database. This includes APIs for payment processing, and retrieving movie data. "Dummy services" may need to be built for testing purposes, until the full software is launched.

Finally, a front-end will be created. This website will be compatible on major computer and mobile browsers. It will display pertinent information based on the type of Account that is logged in. Buttons on the page will send queries to interact with the back-end that was designed before.

As a conservative estimate, this software will take six months to complete the design phase, from start to finish. Our team will consist of full-stack developers, so all members will be involved in all steps of the design process. Testing must be conducted at the end of each phase of design.

5. Test Plan and Verification

5.1 Test Plan

Testing will occur throughout the development cycle. This is in order to ensure confidence in each unit of the software system *before* putting them together.

5.1.1 Website Access: Website should load when the URL is entered into any of the required browsers. The main page should always load when this address is entered.
5.1.2 User Registration: Tests for creating accounts with the service. Should account for if the user fills in an invalid email address, or if they are missing fields. In these cases, the user should be prompted to properly fill in all fields. In the case that all fields are filled, An account should be created, and the user should receive an email confirming their registration
5.1.3 Login: Tests for logging in to an existing account. Login should only occur if both fields are correct. Filling either field blank or incorrectly, results in a message displayed to the user. Entering a true logic statement to the password should not grant access.
5.1.4 Movie Browsing: Websites should allow users to browse movies. This should be tested after login and without a login and should work the same way in both cases. The Movie catalog

should show the title and image poster icon. As an edge case, the page should be tested when there are no movies available in the catalog.
5.1.5 Seat Selection: Users should be able to select seats. If they are not logged in and try to access this page, they should be directed to the login page before accessing it. 1-20 seats should be chosen. If zero seats are chosen, users should not be allowed to proceed. If over 20 seats are chosen, users should be directed to a phone number to speak to a theater employee.
5.1.6 Payment: Payment systems should be tested with PayPal and crypto services, as well as credit card systems. The two former services are working API modules and should be tested using valid credentials for those services. The last one is not an API and requires the most testing. Blank and incorrect information should be revoked. In the case correct card information is presented, the transaction needs to be authorized by the bank the card belongs to. Only after that will the tickets be transferred to users account wallet
5.1.7 Discounts: The special status discounts should be tested. A discount should apply if the user is military, student, or senior. No status should result in the normal price. Discounts should not stack even if someone has multiple categories.

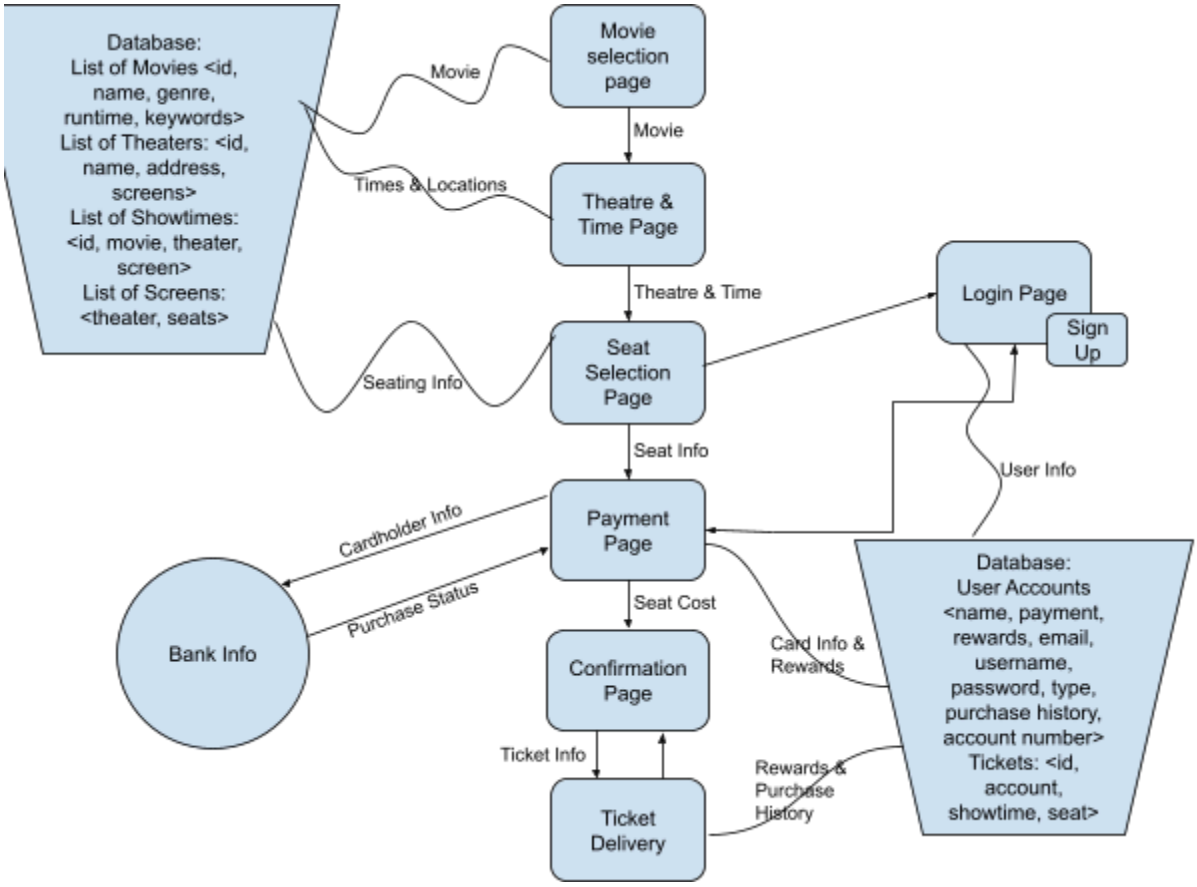
The smaller (unit) tests should be completed first. Services should be chained together based on adjacent modules in diagram 4.2.2. Once all components are tested, verified, and connected, the entire system workflow should be tested. Once confidence in the entire system is adequate, the testing is complete.

5.2 Test Cases

Please see the attached Google sheet for test cases.
https://docs.google.com/spreadsheets/d/186haxiT_V1oSd0w6k5u2pGeB8mUbmahd/edit?usp=sharing&ouid=100124710237653004459&rtpof=true&sd=true

6. Data Management

6.1 Software Architecture Diagram:



The software architecture diagram contains descriptions of the databases that we will be using. We will utilize a database that will contain a list of movies and information relevant to those movies and a database that holds our different theaters. The movie database will connect to the movie selection page. After selecting the movie, they will be brought to the theater & time page, where they will be able to select both a theater at a time. This information will be pulled back into the database that contains our list of movies. Next, we will pull out seating information from our theater database, allow the user to select an open seat, and then update the seating information in the database.

This step will also pull data from our user accounts database, which will allow us to update the user's profile with their ticket information. Next, we will bring the user to the payment page, which will be able to handle transactions securely. We will not store any sensitive information about customers. We will use an API to connect our site to banking services securely. After completing the transaction, we will bring the user to a confirmation page.

After confirming their purchase, the user will receive their ticket information, and we will update their account information within the user account database with the relevant information.

We decided to update our existing software architecture diagram, rather than create a new one. We decided to take this route because our diagram already involved our databases, so we

only needed to update our existing database with the changes we have made to our data management strategy.

6.2 Strategy

The software will utilize two separate databases. The first database will contain tables of Movies, Locations, Showtimes, and Screens. The second one will keep track of Accounts and Purchases

DB 1:

Movies:
- id
- name
- genre
- runtime
- keywords

Theaters:
- id
- name
- address
- # screens

Showtimes:
- id
- movie
- theater
- screen

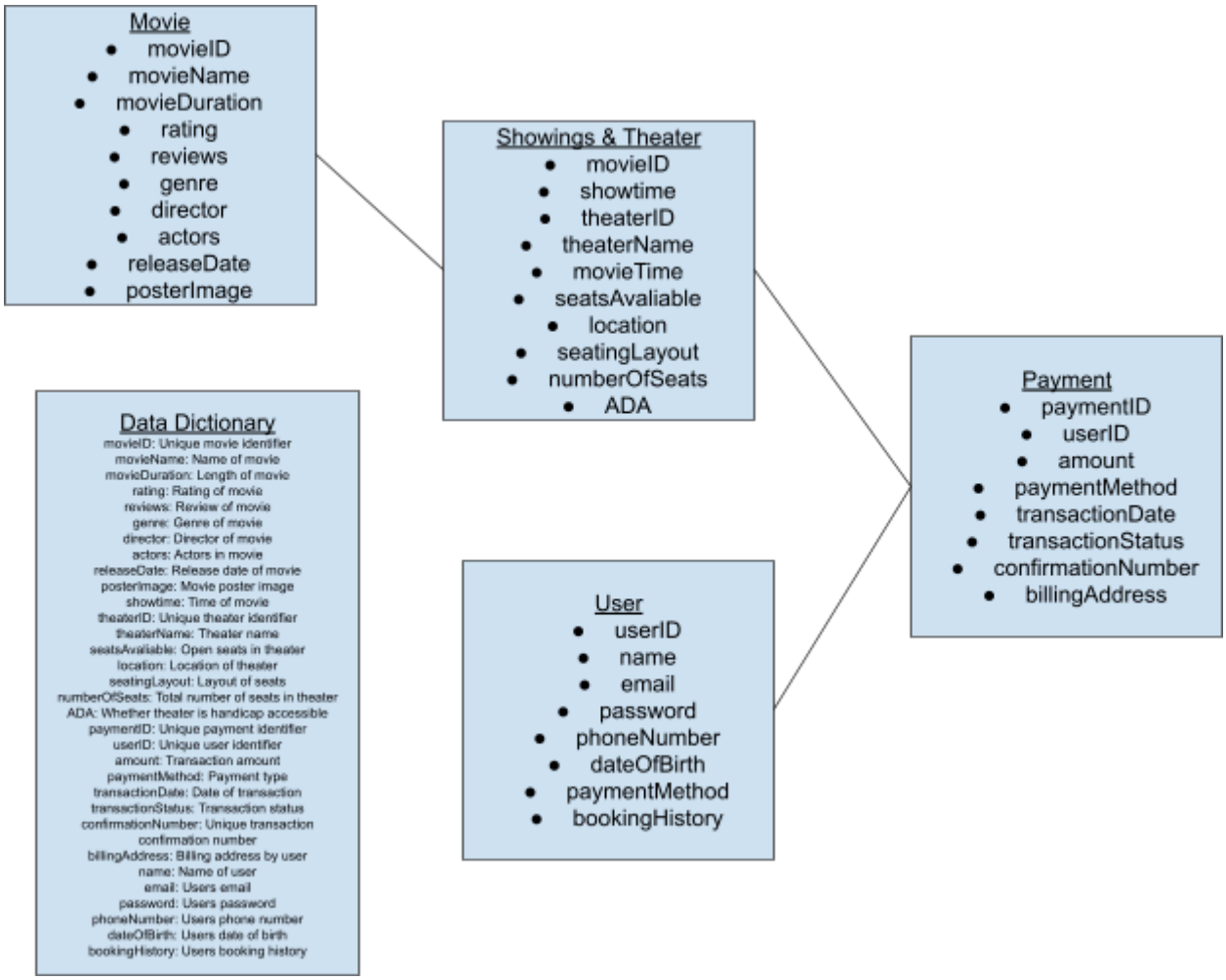
Screens:
- id
- theater
- # seats

DB 2:

Accounts:
- name
- email,
- reward points
- payment information
- username
- password
- account number

Tickets:
- id
- account
- showtime
- seat
- screen

6.3 Relational Diagram



6.4 Trade Off Discussion

These two databases share data with each other, but it is through queries, and there are security measures in place. Everything is encrypted for safety, but items in DB 2 have stronger encryption since personal information is involved. Tables are related according to the relational diagram above.

A structured, relational database like SQL is much better for this system. Data is not going to shift a large amount at any one time, and the structure of objects will very rarely change, if at all

once set. The alternative is a non-relational database like NoSQL, but it doesn't fit well at all with the very structured, and interconnected nature of our system.

Rather than a single database, two databases are implemented, and data between them will be shared with secure queries. Using a single database would likely be faster, but a vulnerability would result in the most sensitive data being leaked. Having a second database, with stronger security measures was the best compromise between speed and security.

Each database has multiple tables, but they are mostly related. For example, multiple screens belong to the same theater, so a screen table will contain the theater id, and a theater table will contain the number of screens. The alternative would be to have less tables, and more columns per table, but this design minimizes the number of redundant columns to almost zero.