Theater Ticketing System

Software Requirements Specification

Version 3

June 3, 2024

Group 2
Aaron Garcia, Mohamed Ali, Yousif Sabri

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

Revision History

Date	Description	Author	Comments
5/27/2024	Version 1	Aaron Garcia, Mohamed Ali, Yousif Sabri	Added introduction of the SRS, general description, and specific requirements
6/3/2024	Version 2	Aaron Garcia, Mohamed Ali, Yousif Sabri	Added the UML and the software architectural diagrams. We also added the descriptions of classes, attributes, and operations.
6/10/2024	Version 3	Aaron Garcia, Mohamed Ali, Yousif Sabri	Added the test cases and over all test plan

Document Approval

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

Signature	Printed Name	Title	Date
	Dr. Gus Hanna	Instructor, CS 250	5/27/2024
Yousif Sabri	Yousif Sabri	Software Eng.	5/27/2024
Mohamed Ali	Mohamed Ali	Software Eng.	5/27/2024
Aaron Garcia	Aaron Garcia	Software Eng.	5/27/2024

Table of Contents

1. Introduction

This Software Requirement Specification (SRS) document will give a clear understanding of the entire project intended for creation, a theater ticketing system that handles the entire process of buying movie tickets, seat selection, payments, and more.

1.1 Purpose

This document starts by declaring its scope, definitions, acronyms, abbreviations, references, and overview of the software. It will then describe the environment of this project, which refers to the interactions of the product itself, its relation with other related projects, its intended users, and the developers of the software. The SRS will also list the requirements of the project.

1.2 Scope

The product that this document outlines is the Theater Ticketing System, a software that allows for ease of access for its users to purchase a movie ticket which reserves a specific seat chosen by the client. This system will disable the ability to choose a seat if a room is full, prompting the user to choose a different time to visit. The software will also provide an option to register for an account, which allows the user to store their personal information to speed up the process for future visits, as well as reward them with loyalty points for visiting our locations. The loyalty points can be redeemed to reward the user with discounted or free tickets.

For handlers, we aspire to obtain user reviews that can provide feedback about the site itself as well as the location they may have visited. If any complaints or questions may arise from a customer, they will have access to customer support, which the software will act as a medium to allow communication via chat box or inquiries.

1.3 Definitions, Acronyms, and Abbreviations

SRS	Software Requirement Specification
Etc	Et Cetera
GUI	Graphical User Interface

1.4 References

References included:

1. IEEE Computer Society. “IEEE Recommended Practice for Software Requirements Specifications.” Approved 25 June 1998, reaffirmed 9 December 2009, Software Engineering Standards Committee, IEEE-SA Standards Board.
2. “Theater Ticketing System Qs.rtf” Canvas, San Diego State University.
3. “Theater Ticketing Requirements.rtf” Canvas, San Diego State University.

4. “Theater Ticketing Questions.rtf” Canvas, San Diego State University.

1.5 Overview

The following sections of this document will allow for a clear description of the product and any needed requirements it may have. A general description section will cover the overall characteristics of the project as a whole, a specific requirements section will go over the functional and non-functional requirements, which includes security, maintainability, and portability. Furthermore, an analysis model section will showcase the system visually, whereas the change management process will outline the process for updating the SRS in the event that changes are made within the scope of the project or requirements.

2. General Description

The theater ticketing system is a web-based consumer-friendly software that will allow consumers to purchase tickets online or in person at the theater using a digital kiosk. The web browser will be designed to incorporate a user-friendly interface, which will allow for easy access for both the consumer, and administrator mode.

2.1 Product Perspective

This software is driven to create an accessible, innovative, and seamless ticketing system. This will allow quick and easy access for both the consumer and the administrators, improving the consumer experience and management efficiency. By prioritizing user convenience and operational productivity, we focus on transforming and maximizing customer experience and management processes.

Some of the features we will implement include:

- Allowing for tickets to be purchased online using different types of currency.
- Automatically disabling buyers from purchasing more than 20 tickets at one time.
- Allowing seating arrangements so that the consumer will have the ability to choose between the open seats.
- Track orders for the management team.
- Create a queuing system during high volumes of requests.
- Block out bots and generate unique tickets that can not be duplicated to guarantee authenticity.

This software will improve productivity and be a valuable asset to any movie theater that implements it.

2.2 Product Functions

The functions that this software will integrate within the web browser and digital kiosk will allow consumers to seamlessly purchase movie tickets online and or at the theater.

Key functions include:

- Standardized prices across all platforms with currency conversion based on the consumer's location.
- The program will accept payment methods from multiple different sources, including credit cards, PayPal, and Bitcoin.
- Option to return or exchange a purchase given that the purchase was made from the same account.
- Ability to choose a preferred seat after purchasing a ticket.
- Issuance of unique, non-replicable tickets to prevent fraud.

2.3 User Characteristics

The system's eventual users include consumers, theater personnel, administrators, actors, production crew members, and system developers. Customers vary in age and technical skill, and they prefer to purchase tickets both online and in person. Theater employees assist with sales and customer service, while administrators manage system operation and security. Ticketing data indirectly benefits performers and crew workers, while developers ensure that the system functions properly. Understanding the various user demands is critical for successful system design.

2.4 General Constraints

The system must comply with a variety of limitations, which influence design decisions. These restrictions include compatibility with current hardware and software infrastructure, adherence to regulatory requirements governing data privacy and security, and interaction with third-party payment processing and review scraping services. Furthermore, the system's performance must fulfill specific standards, such as reaction speed and scalability, in order to support a large number of concurrent users.

2.5 Assumptions and Dependencies

The SRS's criteria can be affected by several things. The following assumptions and dependencies are considered:

- Internet Connectivity: It is assumed that users will have consistent internet connectivity
- Operating Systems: The software is assumed to be compatible with major operating systems, including Windows, macOS, IOS, and Android.
- Web Browsers: The system is assumed to support popular web browsers, such as Google Chrome, Firefox, Safari, and Microsoft Edge.
- Third-Party Services: The software relies on the stability and reliability of third-party services for payment processing (e.g./ PayPal, Bitcoin) and currency conversion.
- Data Sources: The reliability of external review sites for data scraping is assumed.
- Theater Scheduling and Seating: The software's access to up-to-date theater scheduling and seating arrangements is essential for accurate ticket sales.

Any modifications to these variables can call for a revision to the SRS's list of system requirements.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- Account login/guest checkout: Allows the user the option to create an account or to directly purchase a ticket without doing so.
- The home page featuring movies: Provides an overview of available movies that are available to choose from.
- Seat selection/booking system: Allows the user to reserve seats for themselves based on availability.
- Payment screen: Provides the ability to purchase the tickets with a payment method of their choice.

3.1.2 Hardware Interfaces

- The website must be accessible for any device with a compatible web browser.

3.1.3 Software Interfaces

- Payment API: Allows for the integration of payment methods such as credit card, PayPal, Venmo, etc.
- Notification Service: Allows our locations to send out confirmation for bookings, reminders for seating reservations, and promotional offers.
- Movie and booking database: Provides information about current seating openings and upcoming movies to keep the site up to date.

3.1.4 Communications Interfaces

- Email Service: Sends emails with regards to transactions created by the user or verification upon account creation.
- SMS Gateway: Allows users to receive text message notifications for certain events such as but not limited to: movie start time reminders or promotional sales.

3.2 Functional Requirements

3.2.1 Pricing:

Introduction: The system shall display accurate and consistent pricing information about its corresponding movie.

- Validity checks on the inputs: The system should ensure user inputs are valid numeric values.

- The exact sequence of operations: When the user enters the homepage, the system should display a list of prices and their corresponding movies.
- Responses to abnormal situations:
  1) Error handling and recovery: If the user provides a non-numeric value, the system should display an error message.
- Effect of parameters: The system should properly display the value provided with proper formatting regardless of the type of number provided.
- Relationship of outputs to inputs:
  1) Input/output sequences: The output displayed to the user should be the number provided by the input.

3.2.2 System language support:

Introduction: The system shall provide accessibility to convert the system language to a compatible and supported language.

- Validity checks on the inputs: The system shall ensure user inputs are made within the corresponding GUI.
- Exact sequence of operations: The user clicks on the “Change Language” icon. When given a list of available languages, the user will choose a single option. The system should display the website with converted text.
- Responses to abnormal situations:
  1) Error handling and recovery: When the preferred language is not compatible with the system, the user will have an option to request for the support of the language within the site.
- Effect of parameters: The mouse-clicking action allows for the user to interact with the icon.
- Relationship of outputs to inputs:
  1) Input/output sequences: The icon being clicked shows a list of languages. Clicking on a language converts the text within the website to the chosen language.

3.2.3 Purchase limits:

Introduction: The system shall limit the user to purchase an amount of tickets within 1-20 at a time.

- Validity checks on the inputs: The system shall ensure user inputs are valid numeric values ranging from 1-20.
- Exact sequence of operations: The user selects a movie from the homepage to purchase, they are then prompted with a payment screen asking for the amount of tickets to be purchased. The user shall input a value within 1-20 and are shortly prompted to complete the transaction.
- Responses to abnormal situations:
  1) Overflow: The user will be prevented from inputting a value less than 1 or greater than 20.

- Effect of parameters: The number of tickets imputed will correspond to the amount of tickets to be purchased within the system.
- Relationship of outputs to inputs:
  1) Input/output sequences: The number of tickets will vary in price.

3.2.4 User Accounts:

Introduction: The system shall allow users to create an account that grants access to restricted features such as storing information for quicker use, purchase history, loyalty points, and to request for a refund on eligible purchases.

- Validity checks on the inputs: The system shall ensure user inputs are combinations of strings and integers that adhere to security requirements.
- Exact sequence of operations: The user is prompted to purchase a ticket as a guest, or to create and log into an account. When selecting the option to create an account, the user shall follow a set of rules when creating a username and password to follow certain guidelines that promote more secure combinations. After successfully logging into an account, they are greeted with pages previously inaccessible without an account, which allows them to view a purchase history page, an account information page, and loyalty point page, and a refund page.
- Responses to abnormal situations:
  1) Overflow: The user shall be prevented from creating an account username or password exceeding 32 characters.
- Effect of parameters: The inputs of combinations of strings and integers provided by the user will be added to the database, and allow the user access to restricted parts of the webpage.
- Relationship of outputs to inputs:
  1) Input/output sequences: Username and password allows for access to previously restricted pages.

3.2.5 Seating:

Introduction: Users will have the option to reserve tickets for specific seats, depending on availability.

- Validity checks on the inputs: Ensure the selected seat is available and not already reserved.
- The exact sequence of operations: Display available seats -> User selects seat -> System reserves seat -> Confirmation is sent to the user.
- Responses to abnormal situations:
  1) Overflow: Handle cases where multiple users attempt to reserve the same seat simultaneously.
  2) Communication facilities: Notify users of seat availability and reservation status.
  3) Error handling and recovery: Provide alternative seat options if the selected seat is unavailable.
- Effect of parameters: Seat availability changes based on user selection and time

- Relationship of outputs to inputs:
  1) Input/output sequences: User selects seat -> System updates availability -> Reservation confirmed.
  2) Formulas for input-to-output conversion:

3.2.6 Queueing System:

Introduction: Perform a queuing system to manage high volumes of requests for a single showing.

- Validity checks on the inputs: Verify user identity and request validity
- Exact sequence of operations: User request -> Enter queue -> Process request in order -> Allocate resources.
- Responses to abnormal situations:
  1) Overflow: Manage and prioritize high volumes of simultaneous requests.
  2) Communication facilities: Notify users of their queue status and estimated wait time.
  3) Error handling and recovery: Provide feedback and retry options if the system fails.
- Effect of parameters: Queue length and wait times vary based on the number of requests.
- Relationship of outputs to inputs:
  1) Input/output sequences: User request -> Queue -> Process request ->Notification of completion
  2) Formulas for input-to-output conversion: Not applicable

3.2.7 Security:

Introduction: All purchased tickets will be unique and non-replicable.

- Validity checks on the inputs: Verify user identity and ticket purchase details.
- Exact sequence of operations: Generate unique ticket -> Encrypt ticket details -> Send to user -> Validate at entry.
- Responses to abnormal situations:
  1) Overflow: Manage large volumes of ticket generation requests securely.
  2) Communication facilities: Inform users of ticket status and validation.
  3) Error handling and recovery: Detect and prevent ticket duplication or fraud.
- Effect of parameters: Security protocols adjust based on threat levels and user activity.
- Relationship of outputs to inputs:
  1) Input/output sequences: User purchase -> Generate unique ticket -> Encrypt and send -> Validate at entry.
  2) Formulas for input-to-output conversion: Not applicable.

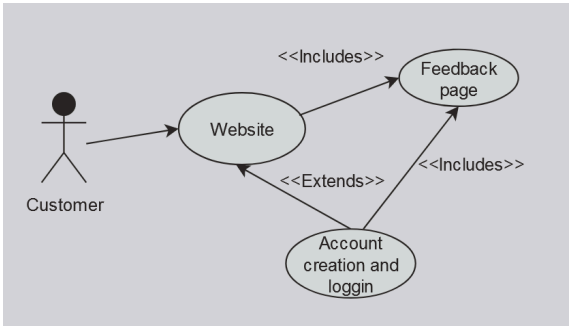
3.2.8 Refund

Introduction: Customers will have the ability to refund a ticket purchased 24 hours before the start of the movie

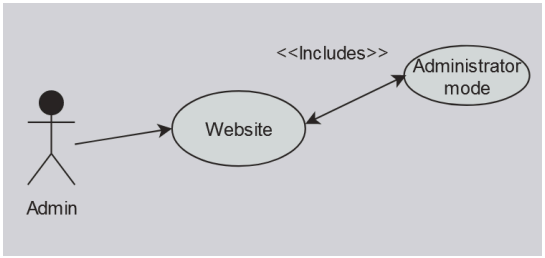
- Validity checks on the inputs: Verify purchase date and refund eligibility within the 24-hour window.
- Exact sequence of operations: User requests refund -> Validate request -> Process refund -> Notify user.
- Responses to abnormal situations:
  1) Overflow: User requests refund -> Validate request -> Process refund -> Notify user.
  2) Communication facilities: Inform users of refund status and processing time.
  3) Error handling and recovery: Provide support for failed or delayed refund processes.
- Effect of parameters: Refund policy and eligibility criteria impact refund processing.
- Relationship of outputs to inputs:
  1) Input/output sequences: User request -> Validate -> Process refund -> Notification.
  2) Formulas for input-to-output conversion: Not applicable

3.3 Use Cases

3.3.1 Use Case #1

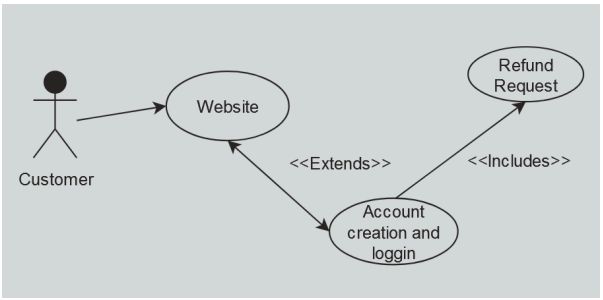
- Name of Use Case: Giving Feedback
- Actor(s): Customer
- The flow of events:
  - 1. After the customer has purchased a movie ticket using the software and watched the movie, they will be capable of providing feedback on the movie using the integrated feedback system in the software.
  - 2. Each movie showcased at the movie theater will have its own feedback section, in which the customer can provide their own opinions on the movie.
  - 3. The feedback feature will allow the user to rate the movie from 1-5 stars and offer comments explaining their rating choice.
  - The customer creates a personal account using the movie theater webpage and purchases a ticket for an upcoming movie.
  - 4. After watching the movie, they go back to the webpage and using their account click on the movie they watched and then click on ratings, they then rate the movie 5 stars, with the comment “Wow that movie was amazing”.
- Entry conditions:
  - 1. Computing requirements: supported browser.
  - 2. Customers must create and be logged into their account.
- Diagram:
  - 

3.3.2 Use Case #2

- Name of Use Case: Using Administrator mode
- Actor(s): Admin, Customer, Manager
- The flow of events:
    - 1. Administrator mode will grant admins control over certain areas within the theater.
    - 2. Admins will be capable of rearranging seating areas, providing refunds, managing movie listings, configuring ticket pricing, feedback management, promotions, and discounts.
    - 3. A customer approached the current manager and requested a seat change for an upcoming movie.
    - 4. The manager logged into the webpage using the administrator mode, clicked on the specific movie the customer requested, and clicked the seat rearrangement catalog, which redirected him to another tab where they were capable of rearranging the seat for the customer.
- Entry conditions:
    - 1. Computing requirements: supported browser.
    - 2. Manager must be provided with an administrator account with admin privileges.
- Diagram:
    - 

3.3.2 Use Case #3

- Name of Use Case: Requesting a Refund
- Actor(s): Customer
- The flow of events:
    - 1. A customer purchases using the webpage a couple of days before the movie starts. However, other plans came up, and are not capable of going to the movie.
    - 2. Therefore, they used the movie theater webpage to get a refund for the ticket they purchased 72 hours before the movie started.
- Entry conditions:
    - 1. Computing requirements: supported browser.
    - 2. Customers must create and be logged into their account.
- Diagram:



3.4 Classes / Objects

3.4.1 Movie Genres and Movies/ Object #1

3.4.1.1 Attributes

- Genre ID
- Movie ID
- Genre name
- Movie name
- Movie description
- Genre description

3.4.1.2 Functions

- Add genre ID
- Add movie ID
- Add genre
- Add movie
- Add genre description
- Add movie description
- Delete Genre
- Delete movie

Reference to use cases: Movie Genres and movies will allow the admins to acquire control over the movies that are posted on the web page and assign them to the correct genre.

3.4.2 Feedback / Object #2

3.4.2.1 Attributes

- Feedback ID
- User ID
- Movie Name

- Rating
- Comment

3.4.2.2 Functions

- Get Feedback ID
- Get Movie Title
- Get Theater Location
- Add Comment
- Get Comment
- Add Rating
- Get Rating

Reference to use cases: The feedback object will be used to collect data from the users based on their opinion of a movie they have watched. This data will allow other users to view the level of engagement the movie has had on others, which can factor in whether or not they may want to watch it for themselves.

3.5 Non-Functional Requirements

3.5.1 Performance

The home page should load within 3 seconds upon opening in order to maintain a seamless browsing user experience. In addition to that, navigating to other pages within the site should be under 2 seconds for consistency.

3.5.2 Reliability

The website should be capable of handling a minimum of 1,000 users interacting at a single moment without experiencing any system failure or slowdowns in order to keep user satisfaction at all times. Events that may allow for a peak in the amount of users visiting the website could include the release of a new, sought after movie.

3.5.3 Availability

The system is required to maintain an uptime of at least 99.5% to provide for continuous availability for the users. If any downtime is to be expected or unexpected, a backup server will take the role of receiving engagement whilst the main server undergoes maintenance.

3.5.4 Security

In order to protect user data from unauthorized access and to maintain the overall system security in general, regular security checks on both the website and database will be made at least three

times a week. Regular checks are made in order to identify early if any vulnerabilities are present within the system before possible exploits are developed.

3.5.5 Maintainability

Proper documentation of the coding conventions will allow for easy maintenance and modification of the system and database, allowing for reduced time required for future updates. This documentation will also include comments, coding standards, and any practices that will allow for optimization within the system.

3.5.6 Portability

The website requires proper display and functionality when being utilized across different devices, catering to as many users as possible. With this in mind, it is important to ensure proper display and functionality are present within different sizes of screens, taking into account, but not limited to, devices such as tablets, desktops, and smartphones.

3.6 Inverse Requirements

State any *useful* inverse requirements.

3.7 Design Constraints

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

3.8 Logical Database Requirements

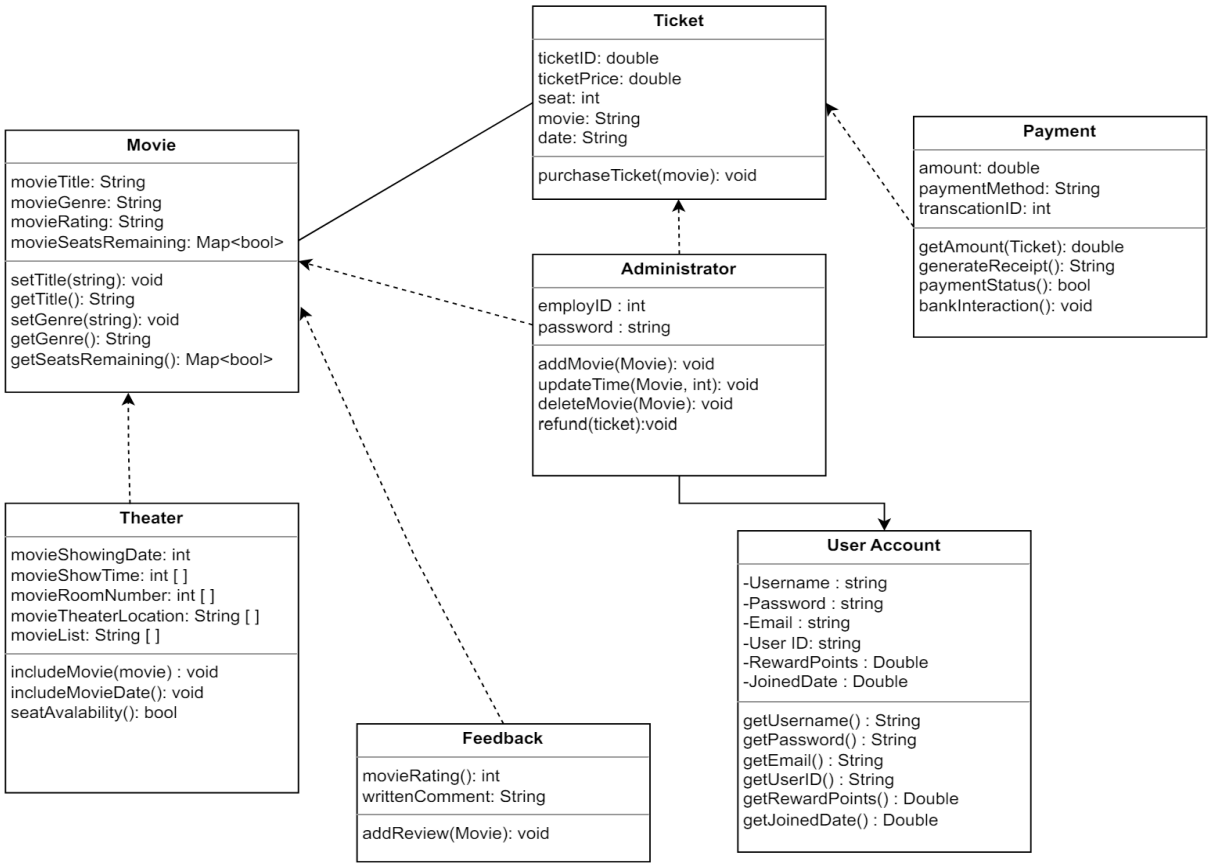
Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.

3.9 Other Requirements

Catchall section for any additional requirements.

4. Analysis Models

4.1 UML Diagram



4.1.1 UML Class Diagram Diagram

Class name: Movie

Purpose:

This class is representative of a movie available to be viewed at the theater of the user's choice.

Attributes:

‘movieTitle’: String - The title of the movie.

‘movieGenre’: String - The genre of the movie.

‘movieRating’: String - The rating of the movie obtained from prior feedback.

‘movieSeatsRemaining’: Map<bool> - A Map that contains the amount of available/unoccupied seats for this particular movie. The parameter, bool, represents whether the seat is available (true) or not (false).

Methods/Operations:

‘setTitle(string)’: void - Sets the title of the movie. Accepts a String parameter representing the new title.

‘getTitle()’: String - Retrieves the title of the movie.

‘setGenre(string)’: void - Sets the genre of the movie. Accepts a String parameter representing the new genre.

‘getGenre()’: String - Retrieves the genre of the movie.

‘getSeatsRemaining()’: Map<bool> - Retrieves the amount of available/unoccupied seats for this particular movie. Returns a map that shows the amount of available seats a particular movie has.

Description:

The “Movie” class holds information about a movie that can be viewed at the theater at a particular time. It provides information about the movie’s title, genre, and the amount of available seats remaining at the theater room during the chosen time. This class provides methods that allow for changes of these attributes, which would be used to access movie information that is present in the database.

Class name: Ticket

Purpose:

This class is representative of the ticket purchased by the user for the movie of their choice.

Attributes:

‘ticketID’: double - An identifier for the ticket.

‘ticketPrice’: double - The price of the ticket.

‘seat’: int - The seat number associated with the ticket.

‘movie’: String - The title of the movie associated with the ticket.

‘date’: String - The date the movie will be viewed.

Methods/Operations:

‘purchaseTicket(movie)’: void - Allows for the purchase of a ticket for a movie as specified with the parameter. Accepts a string parameter containing the title of the movie.

Description:

The “Ticket” class holds information about a ticket object created after being purchased by a user for a particular movie. It provides information about the ticket ID, price, seat number, associated movie title, and the date of the movie. This class provides a method that allows for the user to purchase a ticket for a movie and stores information about the transaction within the system.

Class name: Administrator

Purpose:

This class is representative of an administrator role with access to the system’s administrative features.

Attributes:

‘employeeID’: int - An identifier for the employee.

‘password’: string - The password associated with the administrator account.

Methods/Operations:

‘addMovie(Movie)’: void - Adds a new movie into the system/database. Accepts a “Movie” object in the parameters representing the new movie to be added.

‘updateTime(Movie, int)’: void - Updates the movie date for which it will be viewed. Accepts a “Movie” object representing the movie being updated, as we as an integer representing the new date/time.

‘deleteMovie(Movie)’: void - Deletes an existing movie from the system/database. Accepts a “Movie” object representing the movie being removed.

‘refund(ticket)’: void - Creates a refund for a specific ticket. Accepts a “Ticket” object representing the ticket being refunded.

Description:

The “Administrator” class represents a role that allows for administrative access within the system. It stores information about the administrator's employee ID and password. This class provides methods that allow for the use of administrative features such as adding movies, updating movie information, deleting existing movies, and refunding transactions.

Class name: Payment

Purpose:

Class used to allow the customer to use a digital payment system.

Attributes:

‘amount’: double - The amount paid by the user.

‘paymentMethod’: String - The method of payment used for the transaction.

‘transaction ID’: int - The identifier for the payment/transaction.

Methods/Operations:

‘getAmount(Ticket)’: double - Returns the amount paid for by the user for the specified ticket. Accepts a “Ticket” object which represents the ticket being referred to within the transaction.

‘generateReceipt()’: String - Creates a receipt for the payment.

‘paymentStatus()’: bool - Returns the status of the payment. Returns true if successful, returns false if unsuccessful.

‘bankInteraction()’: void - Starts the interaction process within the bank and the system to process the payment.

Description:

The “Payment” class represents the payment being made after a ticket is purchased. It stores information about the payment with attributes such as the amount paid, the payment method, and the ID of the transaction. This class provides methods that retrieve information about the payment being made such as getting the amount, creating a receipt, and the status of the payment as well as starting the bank interaction itself.

Class name: User Account

Purpose:

The "UserAccount" class manages user information and account details within the system.

Attributes:

-Username: string- retains the user's username.
-Password: string- keeps the password connected to the user's account in storage.
-Email: string- stores the email address of the user.
-User ID: string- retains the user's distinct identification.
-RewardPoints: Double- tracks the reward points accumulated by the user.
-JoinedDate: Double- records the date when the user account was created.

Methods/Operations:

‘getUsername()’: String- retrieves the username associated with the user account.
‘getPassword()’: String- gets the password linked to the user account.
‘getEmail()’: String- obtains the email address linked to the user profile.
‘getUserID()’: String- retrieves the unique identifier associated with the user account.
‘getRewardPoints()’: Double- retrieves the total reward points accumulated by the user.
‘getJoinedDate()’: Double- retrieves the date when the user account was created

Description:

The "UserAccount" class serves as a container for storing and accessing user-related information within the system. It contains attributes such as username, password, email, user ID, reward points, and joined date. Additionally, it provides methods to retrieve each of these attributes. This class facilitates the management of user accounts and their associated details, allowing for seamless interaction and authentication within the system.

Class name: Theater

Purpose:

The "Theater" class represents a physical theater location where movies are shown.

Attributes:

‘movieShowingDate’: int- keeps the movie's premiere date in storage
‘movieShowTime’: int [ ]- retains a variety of movie screening times.
‘movieRoomNumber’: int [ ]- stores an array of room numbers corresponding to each movie show.
‘movieTheaterLocation’: String [ ]- maintains a collection of movie theater locations.
‘movieList’: String [ ]- stores a list of movies currently being shown.

Methods/Operations:

‘includeMovie(movie)’: void- adds a new movie to the list of movies being shown.
‘includeMovieDate()’: void- adds a new movie showing date.
‘seatAvailability()’: bool- checks the availability of seats for a movie showing.

Description:

The "Theater" class manages the scheduling and availability of movies at a theater location. It tracks attributes such as movie showing dates, show times, room numbers, theater locations, and the list of movies being shown. The class provides methods to include new movies, add movie showing dates, and check the availability of seats for a particular movie showing. This class facilitates the organization and management of movie screenings within the theater.

Class name: Feedback

Purpose:
The "Feedback" class manages user feedback and ratings for movies.

Attributes:

movieRating(): int- records the user's rating for a film.
writtenComment: String- saves any textual feedback or evaluations that users submit.

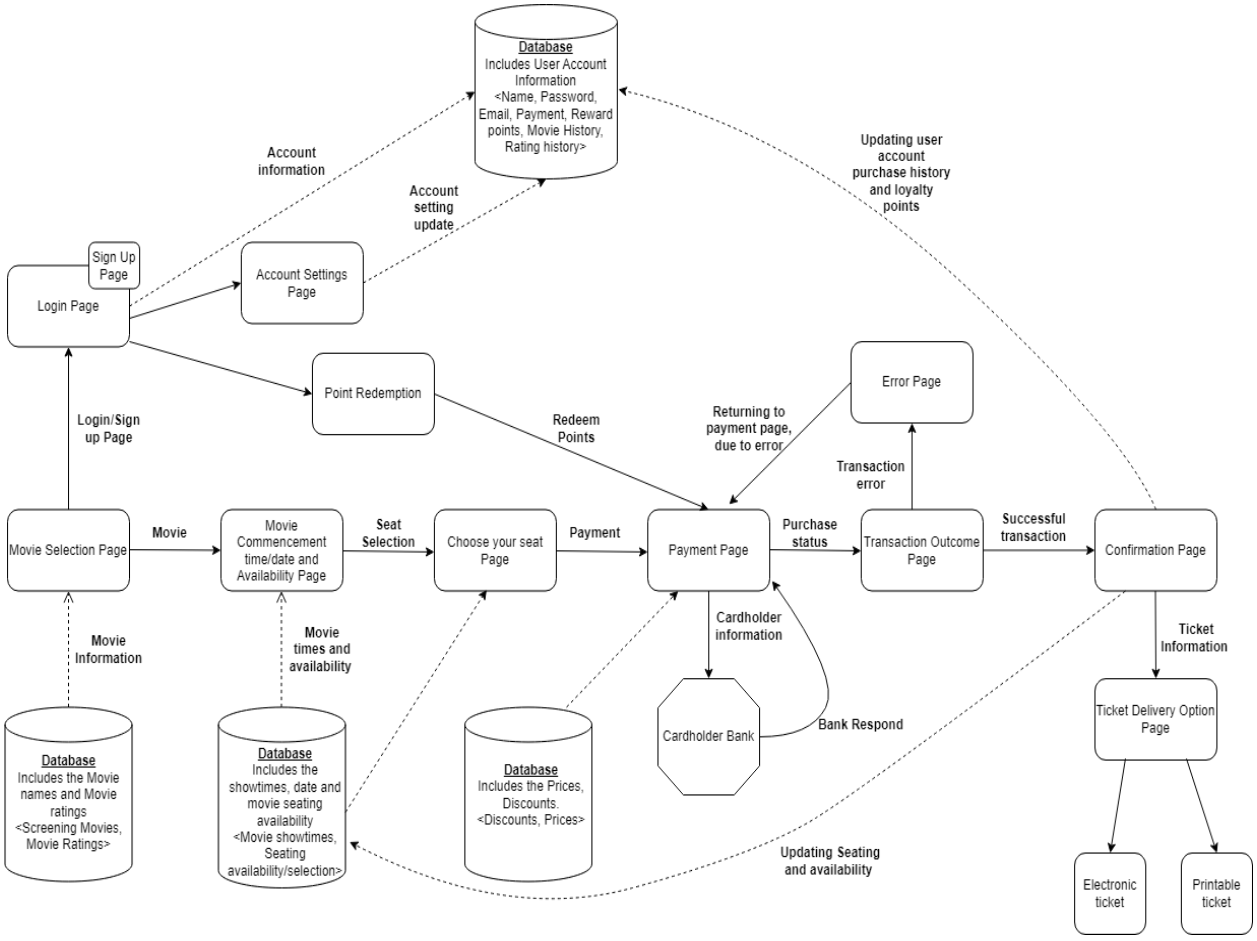
Methods/Operations:

addReview(Movie): void- adds a review for a specific movie.

Description:

The "Feedback" class handles user feedback and ratings for movies shown in the theater. It includes attributes such as movie ratings and written comments. The class provides a method to add a review for a particular movie, allowing users to share their opinions and experiences. This class facilitates the collection and management of feedback, contributing to the improvement of movie offerings and overall customer satisfaction.

4.2 Software Architecture Diagram



4.2.1 Architecture Diagram Description'

- The Architecture Diagram is a detailed diagram outlining the Theater Ticketing System's main components and their interconnections. Each step is provided and illustrated in this diagram, which helps provide a visual representation of the flow of operations this website will incorporate. Furthermore, the diagram will also include the databases that will be utilized and what they are responsible for managing.
- The system begins with a "movie selection" page, where the users can choose a movie, place a rating on a movie, and or be redirected to the login page. On the "Login" page, users have the option to sign up, update account settings, or redeem points. If a user has chosen a movie, the system redirects them to the "movie commencement time/date and availability" page where they can check the movie's availability and showtime. After selecting a date and time, users are directed to the "Choose your seat" page where they can select an available seat. The next page after seat selection is the "payment" page. Here, users can use their bank account to purchase the ticket. The system contacts the cardholder's bank, and upon receiving a response, it will direct the user to the

"transaction outcome" page. Here, users can see if the transaction was successful, or unsuccessful. If an error occurs, the user is redirected to the "Error" page and then back to the payment page if they wish to retry. If the transaction is successful, the system proceeds to the "confirmation" page and then the "ticket delivery options" page where users can choose between an electronic ticket or a printable ticket.

- Certain pages will be connected to databases to hold information. The "Movie Selection" page will be linked to a database containing current movie showings and movie ratings. Additionally, the "Login" page and the account settings page will both be connected to the same database holding personal information and previous movie-related data. Moreover, the "Movie commencement time/date and availability" page and the "Choose your seat" page will both be linked to the database containing movie and seating availability along with showtimes. Furthermore, the confirmation page will also be connected to this database to update it every time a successful transaction is made. Lastly, the payment page will be connected to the discounts and prices database.

Development Plan and Timeline

Month 0-1:

Mohamed Ali (Creating Outline for Architecture):

- Mohamed Ali will be tasked with defining the overall architecture of the system. This includes determining the structural design and organization of the software components to meet the project's requirements and objectives. Mohamed Ali will play a crucial role in laying the foundation for the system's development by creating a well-defined and comprehensive architectural outline that guides the development team throughout the project

Month 1-3:

Yousif Sabri (Web/Software Development):

- Yousif Sabri will be leading the web development team to ensure the User Interface (UI) aligns with the client's requirements. Additionally, Yousif will contribute to the research and development of the system's software by identifying the most suitable programming language and data structures that maximize cost efficiency and longevity.

Month 4-5:

Aaron Garcia (Tester):

- Aaron Garcia will be responsible for developing test cases and test planning with regard to the system's demands. Furthermore, Aaron will create specific test cases to identify any possible defects present within the software in order to ensure the proper functionality of the system. In the event that an issue is found within the system, Aaron will report and document the defect found during testing, as well as the procedures made to reveal the issue that has been found.

GitHub Repository:

Yousif Sabri GitHub:

- https://github.com/YousifSabri9631/Software-Requirements-Specification/blob/main/CS%20250%20Official%20Theater%20Ticketing%20System%20.pdf

Aaron Garcia GitHub:

- https://github.com/Aarong4cs/G2-SRS

Mohamed Ali GitHub:

- https://github.com/MohamedAli4537/CS-250-Official-Theater-Ticketing-System-.git

5. SDS: Test Plan

5.1 Introduction

- The purpose of this section is to provide a foundation and plan as to how the movie ticketing system will be validated as a working system. In order to achieve this, a series of test cases will be conducted which will ensure that the system will meet the functional and non-functional requirements stated in the document.

5.2 Scope

- The scope of the test plan will cover function requirements, security checks, and the performance of the system. Some examples of the tests that will be conducted would be checking the validity of the payment processing, seating selection, and user interface.

5.3 Objectives

- Meet all functional requirements

- Ensure the software architecture is followed.
- Ensure software security for both the user and the host.
- Meet certain performance requirements.
- Allow for scalability for larger traffic.

5.4 Test Strategy

The test strategy encompasses both verification and validation activities:

Verification:

- Conduct reviews and inspections of requirements, design documents, and source code.
- Perform static analysis and walkthroughs to ensure adherence to standards and identify potential issues early.

Validation:

- Execute dynamic testing, including functional testing, unit testing, integration testing, system testing, and acceptance testing.
- Perform performance testing and load testing to ensure the system can handle expected usage scenarios and beyond.

5.5 Test Environment

Hardware:

- Utilize standard user devices such as PCs, smartphones, and tablets, as well as servers for backend processing to simulate real-world conditions.

Software:

- Test on various operating systems, including Windows, macOS, iOS, and Android.
- Use multiple web browsers, such as Chrome, Firefox, and Safari, to ensure compatibility.
- Employ testing tools like JUnit for unit testing, Selenium for automated functional testing, and LoadRunner for performance and load testing.

5.6 Functional Requirements

The functional requirements that will be tested include:

- User Registration and Login: Verify the process for new user registration and existing user login.
- Movie Listing: Ensure accurate and comprehensive listing of available movies.
- Seat Selection: Validate the seat selection process for accuracy and user-friendliness.
- Ticket Booking: Test the ticket booking functionality for a seamless user experience.
- Payment Processing: Ensure the payment processing system is secure and handles transactions correctly.

- Confirmation and Notification: Verify that users receive accurate and timely confirmations and notifications.

5.7 Test Cases

5.7.1 Test Case Diagram

- Test Cases Document Link

5.7.2 Functional Test Cases

User Account

- TC01: Verify that any user with an existing account can log in to their personal account.
- TC02: Verify that a user can log out of the system.
- TC03: Verify that a user can successfully reset their password using the password reset functionality.

Seat Selection

- TC04: Verify that a user can successfully select seats for a movie.

5.7.3 Unit Test Cases

Rewards

- TC05: Verify that the reward page can be viewed by the user.
- TC06: Validate that a user is prevented from redeeming prizes with an insufficient amount of points.

Search

- TC07: Verify that when a user writes a valid Movie name on the Movie Theater website and presses enter, search results should be displayed in the movie

Registration

- TC08: Verify that a new user can successfully register and an existing user can log into their personal account.

5.7.4 System Test Cases

Ticket Booking

- TC09: Validate that a user can purchase a ticket Successfully

Rewards

- TC10: Validate that upon redeeming a reward, the user is sent a "Free Ticket Voucher" via email.

5.8 Verification and Validation Processes

- **Verification**: Routinely perform throughout inspections of the code, and conduct code audits, and collaborative walkthroughs to maintain code quality.
- **Validation**: Conduct thorough testing covering all aspects of the system including functional testing, security requirements, and integration of the system.

5.9 Entry and Exit Criteria

- **Entry Criteria**:
    - The system has finished development and integration.
    - A test environment has been created.
    - Test data and tools are available for use.
    - Test plans and cases have been approved.

- **Exit Criteria**:
    - All documented test cases have been executed.
    - Test results have been documented.
    - Defects have been marked and resolved.
    - System documentation has been updated.

5.10 Deliverables

- Test plan: The test plan will be organized by test cases that are going to be a part of different sections of the system, for example functional, unit, and system test cases. This layout will help ensure that all the parts of the system are working correctly.
- Test cases: The test cases will test the overall functionality of the website, for example, testing the login, and logout features. Other tests include but are not limited to password reset, seat selection, rewards, movie search, registration, and movie ticket purchase.
- Test execution reports: The testers will report back after all the tests have been completed and provide a clear outline of which tests need to be fixed and which ones have passed.
- Defect reports: After the tests had been completed two of the test cases were identified as failed. The first test that returned failed was the logout module, where the expected result was that the user would be redirected to the login page, however, the system redirected the user to the feedback page instead. Furthermore, the second test that returned as failed was the reward redemption module, where the expected result was that the user would receive a voucher to redeem in their email. However, the user received a blank email.
- Test summary report: The test report indicated that the test process was successful. The system testers clearly documented all the passed and failed tests, providing the developer team with important information to address the issues.

5.11 Schedule

- Planning phase- 2 weeks.
- Test case production- 2 weeks.
- Test environment configuration- 1 week.
- Testing phase- 5 weeks.
- Issue resolution- 3 weeks.
- Closing procedures: 1 week.

5.12 Responsibilities

- Test manager- Overall lead and organize the test plan of the system by making a clear plan of execution.
- System testers- These are the engineers who will test the system and document all the test cases that have been run on the system and which ones have passed and failed.
- Application developers- These are the engineers who will go back into the system code and modify it if need be. They will fix the bugs and figure out ways to avoid them in the future.
- Business Analysts- These analysts will verify that all the conditions have been met in the test plan.

5.13 Risk Management

- Data Breach
    - Mitigation: Utilize powerful encryption for all any sensitive data including any user data. Update the system regularly to patch any vulnerabilities that might emerge.
- Software Bugs
    - Mitigation: Create a clear testing environment for all the test analysts and developers to establish and maintain a robust bug-tracking system.
- Server Downtime
    - Mitigation: Utilizing large servers and creating recovery plans for the servers.

5.14 Approval

- With thorough test planning and coordination, it is assured that the system will perform to its best potential. Consequently, it has received approval from all project stakeholders, the test manager, the project manager, application developers, and the business analyst.