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.

Theater Ticketing System

Software Requirements Specification

1.0

24/05/2024

Group 7
Elias Mapendo, Jacob Silva, Lucas Weinstein

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

Revision History

Date	Description	Author	Comments
05/27	1.0	Group 7	First Revision
06/08	2.0	Group 7	SDS: Test Plan, UML Design Inserted
06/09	2.1	Group 7	Test plan continuation
6/16	3.0	Group 7	Data Management Strategy
6/17	3.1	Group 7	Architecture Design

Document Approval

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

Signature	Printed Name	Title	Date
	Group 7	Software Eng.	06/09/2024
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

This SRS document outlines the requirements for the Theater Ticketing System, intended for use by theater customers and administrators. This document will guide the development, implementation, and testing of the system.

1.1 Purpose

The purpose of this SRS document is to outline and formulate the requirements for the Theater Ticketing System, aiming to best serve the customer through a seamless ticketing system. It will highlight all major aspects of the system and account for all functions needed.

1.2 Scope

This software product will enable customers to purchase tickets for movie showings online. It will support various payment methods, user account management, and seat reservations. The product aims to streamline the ticket purchasing process and provide a robust system to handle high volumes of transactions.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Specification
- UI: User Interface
- NFT: Non-Fungible Token
- DBMS: Database Management System
- Downtime: The time the website will be unavailable due to maintenance or updating occurring.

1.4 References

- Lecture notes on use cases: lecture-use-cases.pdf
- Requirements gathered from client interviews: theater-ticketing-reqs.txt, theater-ticketing-questions.txt

1.5 Overview

The SRS document is organized into sections covering the introduction, general description, specific requirements, analysis models, and change management process. Each section details various aspects of the system requirements and design constraints.

2. General Description

The Theater Ticketing System will be integrated along with already existing theater operations, replacing any outdated systems. It will interface with a central database that manages showtimes, ticket & seat availability, and user data for marketing and analytics for later system updates.

2.1 Product Perspective

Theater Ticketing System offers:

- Movie Location Description:
    - Provides a description and address for the movie locations the user can choose from.

- Ticket Details:
    - Includes price, time, and date of the ticket being offered, along with the capacity.
- Movie Selection:
    - Provides for the selection of movies occurring around the user's area or at a specified location..

2.2 Product Functions

- Search movies OR movie theaters
    - if they search for a movie:
        - shows theaters playing that movie
            - shows times the movie is playing at a theater
            - view reviews from Rotten Tomatoes
            - flags movie showings that have passed (but have not exceeded 10 minutes of showing
    - if they search for a theater:
        - shows the movies playing at the theater
            - shows times that these movies are playing
            - view reviews from Rotten Tomatoes
            - flags movie showing that have passed (but have not exceeded 10 minutes of showing
- Purchase movie tickets online
    - contact information of the theater
        - phone number, hours of operation, mailing address, email address
    - see ticket prices (adult, child, senior, as well as any special discounts if applicable [etc. veteran])
    - select the number of tickets
    - support multiple payment methods (ApplePay, Visa, Mastercard)
- Manage Tickets
    - transfer ticket
    - prevent ticket scalping with unique NFT tickets
- Create Account:
    - create an account to purchase tickets
    - email, password, phone number
    - optional sign up for loyalty program (free)
        - For every 10 tickets purchased, one is given for free (non-stackable)

2.3 User Characteristics

- Customers: purchase tickets online, manage their accounts, and provide feedback.
- Administrators: oversee ticket sales, handle refunds, and update showtimes and manual user updates.

2.4 General Constraints

- Blocks any bots utilizing a system.
- Each user has a maximum capacity of 20 tickets.
- ID verification to prevent fraud and ensure user authentication
    - Users under 18 will be denied from purchasing any R-rated tickets
- Refund only extends until two hours before the showtime.

- Tickets are only available for purchase two weeks prior to showtime and 10 minutes after it starts
- Must use a web-based browser to access the product.
- Only San Diego theater chains' tickets can be purchased and offered
    - Our system would only serve as an intermediary between the multiple physical theater chains and the customer, where the chains can choose to place restrictions on locations that the tickets would be able to work on. Otherwise, they are able to work at any theater chain nationwide unless a restriction is put on.
- Must have a loyalty-activated account to earn rewards.
- The 11th free ticket does not give a loyalty point toward the ten pts you must buy prior to being able to redeem the 11th ticket
    - The 11th ticket is automatically applied

2.5 Assumptions and Dependencies

- Assumes the user must have a physical device (phone or computer)
- The system assumes reliable internet connectivity for online transactions
- Dependencies include the central database for managing ticket and user data

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- Front-End Software: JavaScript ES7
- Back-End Software: Java 8

3.1.2 Hardware Interfaces

- Servers hosting the web (AWS, AZURE, or GCP) - for fluctuating traffic & growing business as well as scalability, reliability, and pay-as-you-go pricing model

3.1.3 Software Interfaces

- Integration with the theater's existing databases
- APIs for payment processing from VISA, APPLE PAY, & MASTERCARD

3.1.4 Communications Interfaces

- Secure HTTPS communication for all online transactions

3.2 Functional Requirements

3.2.1 Handle Ticket Purchases

3.2.1.1 Introduction

- Intro - Enable customers to buy tickets for movie shows

3.2.1.2 Inputs

- The user selects the movie and showtime
- Number of tickets to buy <= 20

3.2.1.3 Processing

- Validate ticket availability
- Processing the card information put into the system

- generate unique NFT tickets.
- Process information for creating an account (login info, age, phone number name, email, etc.)

3.2.1.4 Outputs
- Confirmation email with digital tickets.
- Payment confirmation of purchase

3.2.1.5 Error Handling
- Notify users of any payment issues or ticket unavailability.

3.2.2 Manage User Accounts

3.2.1.1 Introduction
- Intro - Allow users to create and manage their accounts.

3.2.1.2 Inputs
- User registration information
- login credentials.

3.2.1.3 Processing
- Stores the user's data in the database
- provide account management features

3.2.1.4 Outputs
- Confirmation that the account has been created
- User account dashboard

3.2.1.5 Error Handling
- If the user fails to log in 5 times, then require them to create a new password through email or text
    - A new password can't be the same as the previous one or prohibits simple passwords such as "123456789", "iloveyou", "password"
    - Password must include a number, special character, and capital letter, and must be at least 8 characters in length
- Verifies with the user if suspicious login attempts
- Allows the user to reset the account password through text or email

3.3 Use Cases

3.3.1 Use Case #1

Name**:** Purchasing Ticket Online

Actor: Ticket Buyer (Customer)

Flow of Events:

a. The Customer navigates to the website (assuming they met the assumptions)
b. Optionally they create or log into their account
c. Select the area or time of showtime for desired movies/theater
d. Customer select their desired movie, showtime, and number of tickets <= 20
e. Select the specific seats (if available for the specified showtime)
f. The system shows the total price and payment options (VISA, APPLE, MASTERCARD)
g. The customer enters payment details and confirms the purchase
h. The system processes the payment for accuracy and then generates the tickets(barcode along NFT for verification or scanning)
i. The system sends a confirmation email and/or text of purchase, with the digital ticket barcode that can be optionally added to the Apple Wallet

3.3.2 Use Case #2

Name: Refunding Purchase of Ticket
Actor: Ticket Buyer (Customer), Admin
Flow of Events:

a. If less than 2 hours before showtime will be unavailable
b. Else if the buyer has an account navigate to the refund section to simply refund the selected ticket linked to the buyer's account
c. Otherwise, if the buyer does not have an account, the user uses a unique NFT generated for a ticket to refund in the same section of the website and verifies the NFT for confirmation of purchase.
d. Otherwise, the help system is available to directly connect to, or if the buyer requires more assistance for their refund, to connect the buyer to the help desk/customer service.
    - If the website is down allow for compensation depending upon the time frame of the request
    - Admin can manually change database with limited function and all changes are logged within the database for security purposes
e. The ticket refund process will take 2 days to refund the purchase amount
f. The System updates the database for showtime, and ticket availability, and logs the transaction

3.3.3 Use Case #3

Name: Loyalty Program
Actor: Ticket Buyer (Customer)
Flow of Events:

a. With each purchase, points are accumulated
b. For every 10 tickets, the 11th is free.
c. Free tickets do not build up
    i. Only 1 free ticket at a time, per account.
    ii. Once a free ticket is redeemed, the 10-ticket purchasing requirement is reinstated to get the next free ticket.
d. Tickets can be redeemed through the website for any movie, no additional purchase is required.
e. Free tickets cannot be refunded.

3.4 Classes / Objects

3.4.1 User

3.4.1.1 Attributes

- userID: Unique identifier for each user.
- firstName: user's first name
- lastName: user's last name
- username: The user's chosen username.
- password: The user's password (stored securely).
- email: The user's email address.
- phoneNumber: The user's phone number.
- inLoyaltyProgram: once the user is created, they choose to sign up - which is automatically false
    - LoyaltyProgram - Object is created, linking up loyalty obj to a user

- store it in a database

3.4.1.2 Functions

- register(firstName, lastName, username, password, email): Registers a new user if they don't exist;
- login(username, password): Authenticates a user.
- updateProfile(email, phoneNumber): Updates the user's profile information.
- viewPurchaseHistory(): Returns the user's purchase history.
- redeemLoyaltyPoints(User user_pts): redeem 10 loyalty points for 1 free ticket
- userExist(User user_id): check if the user ID exists
- signUserToLoyaltyProgram(User user): sets inLoyaltyProgram to true

3.4.2 Ticket

3.4.2.1 Attributes

- ticketID: Unique identifier for each ticket.
- showID: Identifier of the show for which the ticket is valid.
- seatNumber: The assigned seat number for the ticket.
- price: Price of the ticket.
- purchaseDate: Date when the ticket was purchased.
- nonFungibleToken: create a new code as a second ID for fraud detection

3.4.2.2 Functions

Reference to functional requirements and/or use cases

- generateTicket(showID, seatNumber, price): Generates a new ticket for a show.
- getTicketDetails(ticketID): Retrieves details of a specific ticket.
- validateTicket(ticketID): Validates the entry ticket.
- refundTicket(ticketID): Processes the refund for a ticket.
- sendConfirmationEmail(): send confirmation email

3.4.3 Show

3.4.3.1 Attributes

- showID: Unique identifier for each show.
- movieTitle: Title of the movie being shown.
- showTime: Date and time of the show.
- theaterID: Identifier of the theater where the show is playing.
- availableSeats: Number of available seats for the show.

3.4.3.2 Functions

Reference to functional requirements and/or use cases

- addShow(movieTitle, showTime, theaterID, availableSeats): Adds a new show to the system.
- updateShow(showID, showTime): Updates the details of an existing show.
- getShowDetails(showID): Retrieves details of a specific show.
- deleteShow(showID): Deletes a show from the system.

3.4.4 Payment

3.4.4.1 Attributes

- List of payment IDs
- paymentID: Unique identifier for each payment transaction.
- userID: Identifier of the user making the payment.
- ticketID: Identifier of the ticket being purchased.
- amount: Amount of money paid.
- paymentMethod: Method of payment (e.g., credit card, Apple, debit card).
- paymentDate: Date of the payment transaction.

3.4.4.2 Functions

Reference to functional requirements and/or use cases

- processPayment(userID, ticketID, amount, paymentMethod): Processes a payment for a ticket purchase.
- refundPayment(paymentID): Refunds a payment transaction.
- getPaymentDetails(paymentID): Retrieves details of a specific payment transaction.

3.4.5 Theater

3.4.5.1 Attributes

- theaterID: Unique identifier for each theater.
- location: Physical address of the theater.
- numSeats: Total number of seats in the theater.
- theaterType: Type of theater (e.g., regular, deluxe).

3.4.5.2 Functions

Reference to functional requirements and/or use cases

- addTheater(location, numSeats, theaterType): Adds a new theater to the system.
- updateTheater(theaterID, location, numSeats): Updates the details of an existing theater.
- getTheaterDetails(theaterID): Retrieves details of a specific theater.
- deleteTheater(theaterID): Deletes a theater from the system.

3.4.6 LoyaltyProgram

3.4.6.1 Attributes

- numOfTicketPurchased : the 11th ticket is free, which resets to 0 once the user redeems the ticket

3.4.6.2 Functions

Reference to functional requirements and/or use cases

- redeemFreeTicket(): Subtracts 10 once 11th ticket is redeemed.
- incrementNumOf_TicketPurchased(): access the payment database to increase numOfTicketPurchased every time a ticket is purchased
- decrementNumOf_TicketPurchased(): decrements numOfTicketPurchased in case of refund

3.5 Non-Functional Requirements

3.5.1 Performance

- Downtime of the system will occur during low populated hours and will not take over a minute per day.
    - A warning system is in place to warn the user of specified planned downtime.
- 99% of transactions must be processed instantaneously after purchase.
- Deliverance of tickets will take no more than 1 minute
    - Notification through desired communication form (Text notification and/or email) to the user of the confirmation of purchase and ticket/s.
- Messaged a reminder through a selected communication from 1 hour before the showtime of the movie.
- The website can accommodate 10 million concurrent users.
    - The site accommodates and does not slow down during high-volume traffic flow for peak times.
- Supports multiple languages, English, Swedish, Spanish, Japanese, and Chinese.
- Has no more than 150 milliseconds of latency to the user.
- Offers tickets for purchase no more than 1 second after ticket release in theaters.
- Response time for customer service does not exceed 5 minutes.
    - Automated responses are provided to the user for commonly asked questions.

3.5.2 Reliability

- Accessibility of the system will stay at 99% always.
- The data is 100% conserved sitewide.
    - Includes backups of data to prevent corruption and instill integrity in the users' actions and transactions.
- Ensures accurate pricing of all tickets across the system.

3.5.3 Availability

Only upcoming movies or movies within specified frames will be shown to the customer, alongside the location-based display of movies in the general area of the user. Updates continuously to give the user the most accurate ticket availability and information.

3.5.4 Security

- Utilizes the database to store the user information securely and connect the user with secured and verified tickets. Recovery methods are in place in case of widespread crashes or corruption with backed-up archived storage of the current database.
- We can secure user data by using multiple cloud services likes AWS, Azure, and GCP to ensure the reliability and security of users in case one of them goes down or is comprised
    - So no customer data is lost
    - duplicates user data for backup

3.5.5 Maintainability

The website allows for constant ongoing updates to ensure that all ends stay up to date and compatible with the newest technology for browsers.

3.5.6 Portability

The website would be available on any device that has web access (MacOS, Windows, Linux, IOS, iPadOS, Android, etc.).

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

3.7 Design Constraints

- Payment compliance with any of the security or policies imposed by credit card companies.
- Websites must be compatible with all devices that have website accessibility.
- Movie theater alignment, having partnered and mutual agreement to terms of the theaters the tickets are incorporated from.
- Authentication from both ends of purchase and distribution of tickets.
    - Ensures security to both customer and ticket company from inauthentic tickets being sold and available for purchase.
- Age restriction for purchases and rating limitation.
    - Rated R movies only available to 18 years of age and above
- Interface variation to accommodate mobile and computer users.
- Supports multiple languages to cater best to the customer

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

4.2 State-Transition Diagrams (STD)

4.3 Data Flow Diagrams (DFD)

Movie Ticket System

Date: 05/03/2024

Team Member: Lucas Weinstein, Jacob Silva, Elias Mapendo

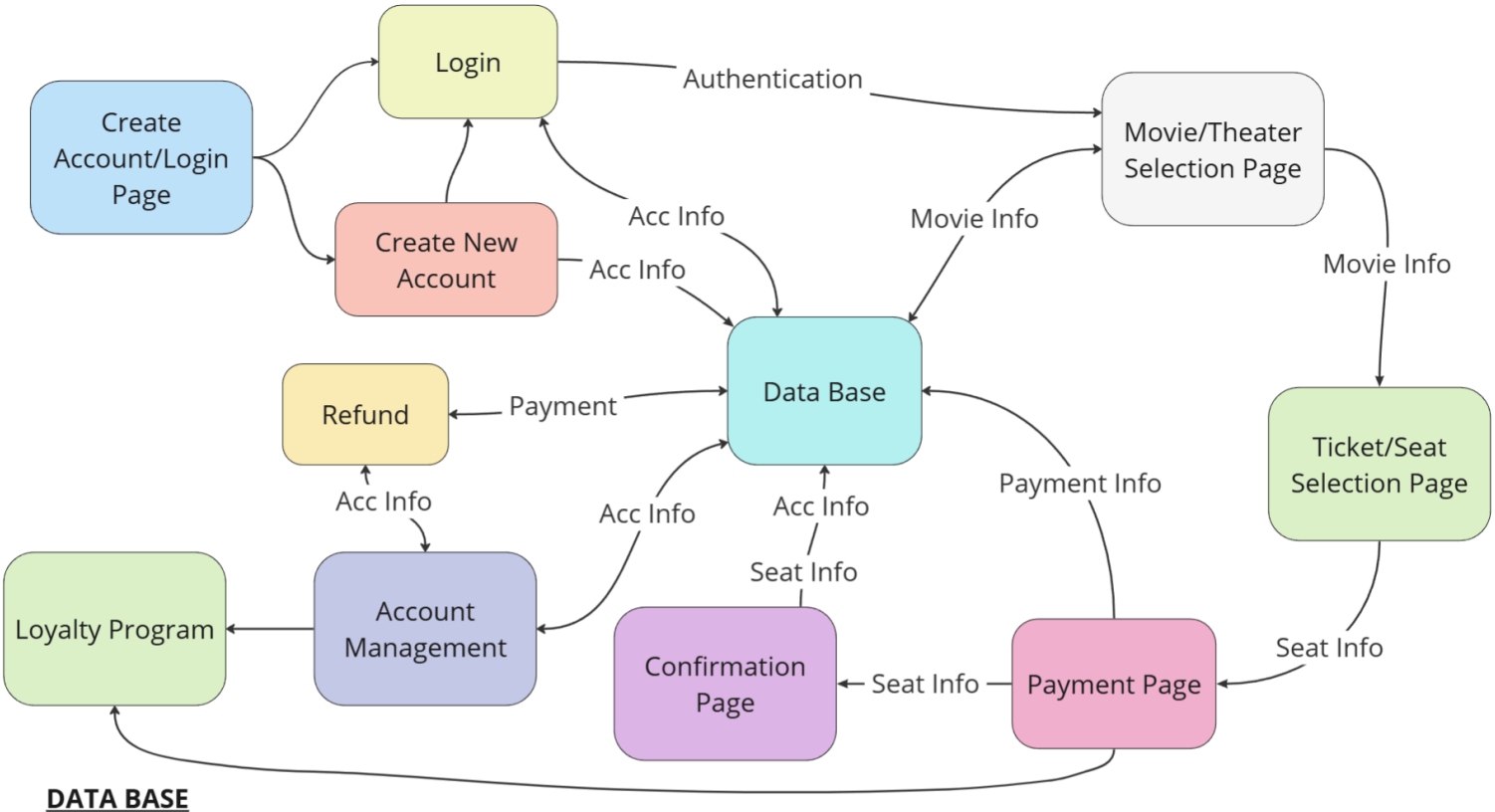
5. System Description:

5.1 Overview: A user-friendly web-based program to purchase tickets at available theaters. Users create an account or log in if they already have an account, to use our program to search for a newly released movie to find theaters in their area showing film. Similarly, the user may search for a theater in their area, which will then show the movies at that theater. Once the user finds a film at a particular theater and at a particular time and with their seat(s), the user may purchase the ticket(s), which will be sent to their email. Optionally, the users' tickets accumulate to receive a free ticket (after their 10th ticket purchase). Once a free ticket is redeemed, the count is reset to zero unless there is another circumstance[1][2]. The user may also change account information such as password, phone number, and username.

[1]Ex: User has 9 tickets already purchased and they purchase 2 more. 1 of the tickets is paid for and then one ticket is redeemed as free, and the counter is reset to zero.
[2]Ex: The user has 9 tickets and purchased 3 more. 1 of the tickets is paid for and then one ticket is redeemed as free, and the counter is set to 1.

5.2 Software Architecture Overview:

5.2.1 Architectural diagram:



DATA BASE

Movie Info:

movies, times, locations, prices, seats, theaters, theaterTypes, theater details

Account Info:

username, password, email, phoneNumber, loyalty points, purchase history

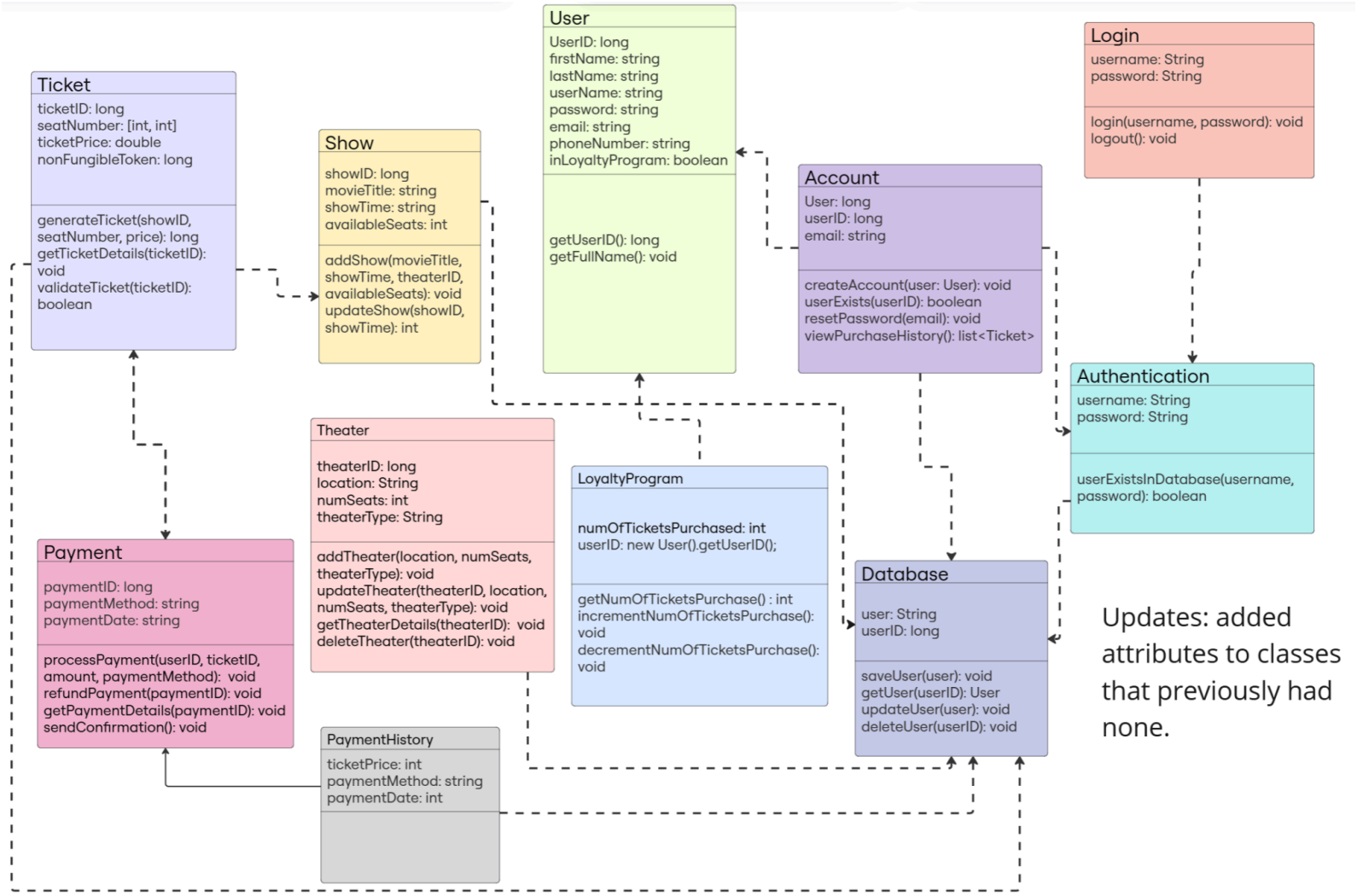
Seat Info:

Seat number, movie, price, theater

Payment Info:

paymentID, date, paymentType, price, user_id

5.2.2 UML Class Diagram:



5.3 Description of Classes:

5.3.1 User:

Description of attributes:

- userID: long - Unique identifier for each user.
- firstName: string - user's first name.
- lastName: string - user's last name.
- userName: string - The user's chosen username.
- password: string - The user's chosen password.
- email: string - The user's email address.

- phoneNumber: string - The user's phone number.
- inLoyaltyProgram: boolean - Holds if the user is in the program. Once a user is created, an option is given to sign up for the loyalty program - is automatically false.

Description of operations:

- getUserID(): int - returns the ID of the user.
- getFullName(): - returns the full name of the user.

5.3.2 Theater: Features and characteristics of a specified theater.

Description of attributes:

- theaterID: long - unique identifier for a movie theater
- location: String - address, city, state, zip code
- numSeats: int - number of seats in a specific theater
- theaterType: String - what type of theater it is (3D, Imax, etc)

Description of operations:

- addTheater(location, numSeats, theaterType): void - add a movie theater into the available search.
- updateTheater(theaterID, location, numSeats, theaterType): void - updates theaterID and/or location and/or numSeats and/or theaterType.
- getTheaterDetails(theaterID): void - retrieves details of theater (location, numSeats, theaterType) for theater (theaterID).
- deleteTheater(theaterID): void - deletes theater (theaterID).

5.3.3 Show: Features showtime specifics such as time and title:

Description of attributes:

- showID: long - Unique identifier for each show.
- movieTitle: string - Title of the movie being shown.
- showTime: string - Date and time of the show.
- availableSeats: int - Number of available seats for the show.

Description of operations:

- addShow(movieTitle, showTime, theaterID, availableSeats): void - Adds a new show with details of title, time, and seat availability to the system.
- updateShow(showID, showTime): int - Updates the details of an existing show, including the time of showing, the seats available, and the title if needed.
- getShowDetails(showID): void - Retrieves details of a specific show, including movie title, show time, and available seats.

- deleteShow(showID): void - Deletes a show and its data from the system.

5.3.4 Ticket:

Description of attributes:

- ticketID: long - Unique identifier for each ticket.
- seatNumber: [int, int] - The assigned seat number for the ticket.
- ticketPrice: double - The specified price for the ticket.
- nonFungibleToken: long - create new code as a second id for fraud detection.

Description of operations:

- generateTicket(showID, seatNumber, price): long - generates a new ticket for a specified show.
- getTicketDetails(ticketID): void - Retrieves the specific details of the ticket, including seat location, ticket price, date, and the non-fungible token.
- validateTicket(ticketID): boolean - validates the authenticity of the ticket.
- refundTicket(ticketID): void - calls to payment class to refund the purchase of the specified ticket.

5.3.5 Payment:

Description of attributes:

- paymentID: long - payment identifier
- paymentMethod: string - type of payment (VISA, MASTERCARD, etc.)
- paymentDate: string - date of payment

Description of operations:

- processPayment(userID, ticketID, amount, paymentMethod): void - Processes a payment for a ticket purchase to make sure methods are accurate
- refundPayment(paymentID): void - Refunds a payment transaction
- getPaymentDetails(paymentID): void - Retrieves details of a specific payment transaction.
- sendConfirmation(): void - sends an email with details of purchase and ticket to use.

5.3.6 Loyalty Program: A loyalty program for customers to redeem 1 free ticket after 10 tickets have been purchased.

Description of attributes:

- numOfTicketsPurchased: int - 0-9 number that displays tickets purchased (if num>9, counter resets, and a free ticket is given to the customer).
- userID: new User().getUserID - retrieve userID from User class.

Description of operations:

- getNumTicketsPurchase(): int - returns number of tickets purchased.
- incrementNumTicketsPurchase(): void - increments tickets purchased by how many tickets were purchased.
- decrementNumOfTicketsPurchase(): void - decrements tickets to 0 if num>9.
- redeemFreeTicket(): void - redeems free redeem once 10 tickets have been purchased.

5.3.7 Account:

Description of attributes:

- N/A

Description of operations:

- createAccount(User user): void - registers a new user
- resetPassword(string email): void - sends a password reset link to the user's email
- vuser'srchaseHistory(): list<Ticket> - returns list of payment history

5.3.8 Login:

Description of attributes:

- N/A

Description of operations:

- login(): void - authenticates user
- logout(): void - logs out the current user

5.3.9 Authentication:

Description of attributes:

- N/A

Description of operations:

- userExistsInDataBase(): boolean - returns true of user already exists in the database

5.3.10 Database:

Description of attributes:

- N/A

Description of operations:

- saveUser(user): void - saves user into database
- getUser(): long - returns userID
- updateUser(user): void - updates user details
- deleteUser(userID): void - deletes the user from the database

5.3.11 PaymentHistory:

Description of attributes:

- ticketPrice: int - the price of the ticket purchased.
- paymentMethod: string - defines if payment method was VISA, MASTERCARD, APPLEPAY, etc.
- paymentDate: int - date the ticket was purchased.

Description of operations:

- N/A

5.4 Development Plan and Timeline:

5.4.1 Partitioning of Tasks:

All team members: worked together to discuss and collaborate
Jacob: How to deal with payments and payment history, as well as how the purchases are changed to our loyalty program.
Elias: How to create/login a user, as well as how their data is sent to our database and authenticated.
Lucas: How to find a movie/theater, and confirm ticket selection.

5.4.2 Team Member Responsibility:

All team members:

- Collaborate to gather requirements and create the initial system design.
- Identify and fix certain errors that would occur when the system is in use to ensure the system is reliable.
- Document the system functionalities, user manuals, and technical specifications.
- Ensure documentation is clear and comprehensive.

Jacob:

- Dealt with formulating the payment history process, and linking it to the loyalty program to complete its functionality entirely.
- Developed the loyalty program system.

Elias:

- How the database schema supports all functionalities.
- And its integration of the other objects

Lucas:

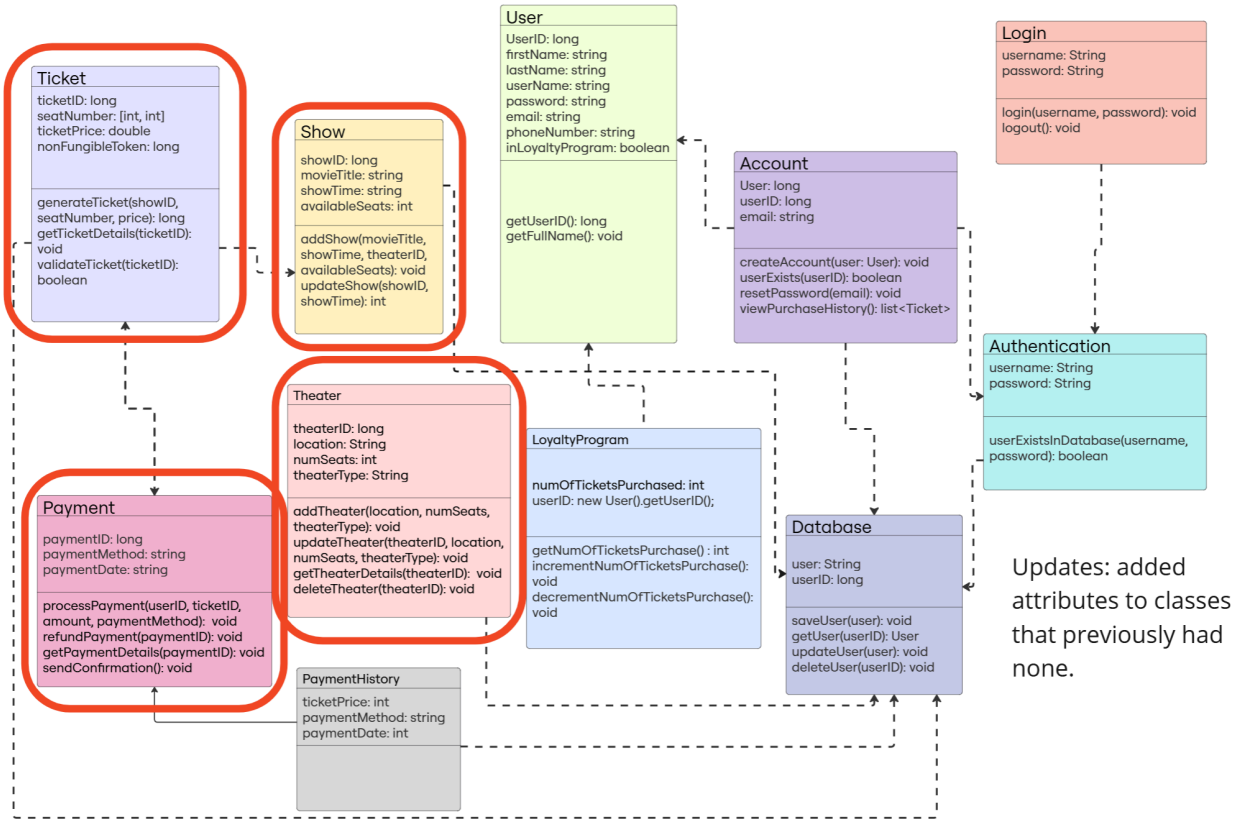
- Dealt with the transferring of information and the functionality between show, ticket, movie, and the database.
- For the Database the timeline of the user using the product.
    - Specifically, log in to create an account to purchase movies then

6. Test Plans:

Test 1:Core Functionalities

6.1 Scope/Overview:

This test plan covers the core functionality of the system, focusing testing on user interactions with the movie and ticketing functions. This would include searching for movies, purchasing tickets, and refunding tickets. This is to ensure that all functional parts are working reliably and as expected.



All Classes/Functions highlighted in red are covered in Test Plan 1.

6.1.1 Testing Specifics:

1. Searching Movie:

Feature verified: Searching both for a non-existent movie and an existent movie

Details:

1. Search for an existent movie
    a. Input: movie title (e.g., “Inception”).
    b. Expected Output: details on Inception showtime, theaters, seats, etc.
    c. The system displays the movie details including showtimes, available seats, and booking options. This test checks that the search functionality relatives and correctly displays information for movies that are in the database. The test covers the integration between the user interface and the database, ensuring that the system can accurately query and return movie data.

2. Search for a non-existent movie
    a. Input: Movie title (e.g., “Abcd1234”)
    b. Expected Output: “No existing movie”
    c. The system displays a message indicating that the movie for that exact move isn’t found, ensuring that that system appropriately handles searches for movies not in the database by providing a user-friendly error message

2. Refunding Ticket:

Feature verified: Refunding a ticket in both the allocated time frame and not in the allocated time frame (must be at least two hours before the showtime to refund)

Details:

1. Refunding in the allocated time frame
    a. Input: Refunding before two hours (e.g., 3 hours before).
    b. Expected Output: Refunds user for ticket purchase.
    c. By testing who is attempting to refund the ticket in the allocated time frame the system can verify the use of refunding the ticket. The system is tested and correctly displays and confirms the refund.
2. Refunding outside of the allocated time frame
    a. Input: Refunding after refund time (e.g., 2 hours after movie showing).
    b. Expected Output: The user is denied the refund with a message showing the reason.
    c. In the non-allocated time frame, the system should not give the user the option to refund the desired ticket as the time threshold for refunding has already passed. Allowing to test outside of the scope of just verified assumed functions and ensuring that the system accounts for other instances.

3. Purchasing Ticket:

Feature verified: Purchasing a specified ticket (including seat section, movie, and time) and attempting to purchase above the ticket threshold of 20 tickets

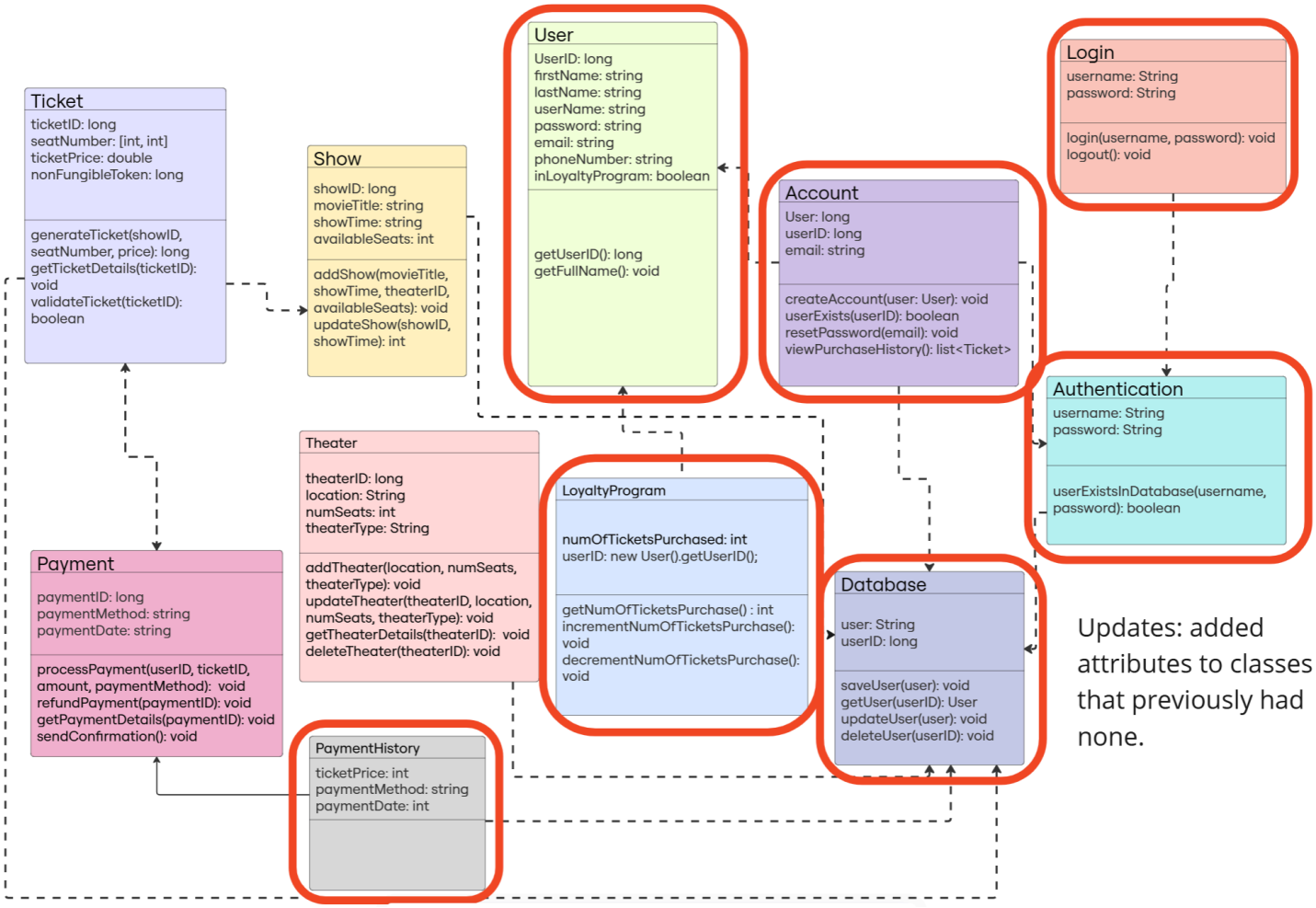
Details:

1. Purchase specified tickets (20 or under tickets)
    a. Input: Purchasing under 20 tickets (e.g., 2 tickets for Oppenheimer for 4:00 pm).
    b. Expected Output: User purchases ticket and receives confirmation and virtual ticket.
    c. Covers the functionality of purchasing tickets under either an account or not. Where the user should correctly be prompted through the use of the system to select and confirm the ticket purchase.
2. Purchase of over 20 tickets
    a. Input: Attempting to purchase above the 20-ticket threshold (e.g., 35 tickets for Puss in Boots: The Last Wish at 2:00 am).
    b. Expected Output: Denied of purchase, along with reason for reasoning.
    c. The verification of a max of 20 tickets comes in the form of the user attempting the same process and then being declined confirmation of the tickets. Ensuring the system can correctly handle all limiting cases.

Test 2: Account Management and System Feature

6.2 Scope/Overview:

This test plan accounts for the more customizable and interactable aspects of the system. With a main focus on user's account information modularity (including login, account creation, and profile customization), account payment history, and engagement in the loyalty program. Confirming the reliability and functionality of the system, ensuring that it best serves the customer's needs.



All Classes/Functions highlighted in red are covered in Test Plan 2.

1. Changing Name:

Feature verified: The ability for users to change their name in both invalid and valid cases.

Details:

1. Valid name Change
    a. Input: From a valid username to a valid username (e.g., Sarah to Samuel).
    b. Expected Output: The account name is changed.
    c. The user logs in, goes to settings to change, enters a new name, and saves the change. The system updates the user's name and confirms the change, ensuring that it allows users to update their personal information accurately and reflects the changes in the user's profile. It verifies the system's ability to handle data updates and user profile management.
2. Invalid Name Change

a. Input: From valid username to invalid username (e.g., Natasha to NLT%9A:$#)(C).
b. Expected Output: “Error: invalid naming format”.
c. The user logs in and enters an invalid name format (empty string or special characters, emoji). The system displays an error message indicating the invalid input. The system checks validation mechanisms, ensuring that invalid inputs are handled correctly and users receive appropriate feedback, verifying the system’s ability to enforce data integrity and provide meaningful error messages.

2. Creating an Account:

Feature verified: Creating an account with both valid and invalid credentials.
Details:

1. Valid Account Creation
    a. Input: name, username, password, email, etc.
    b. Expected Output: Account created successfully.
    c. Go to the website, click Create, enter your name, username, password, email, and other required details, then press enter. The system creates the account and sends a confirmation email to ensure that's a real email. This verifies that the system can successfully create new user accounts with all required information, initiate account setup workflows, and ensure that the registration process is complete and functional.
2. Invalid Credential Account Creation
    a. Input: name, username, missing/invalid password, email, etc.
    b. Expected Output: The user is denied account creation, with a message of what credential/s lack the requirements.
    c. Clicks are entered to create an account, then the system displays an error message indicating the missing or invalid information. This ensures that the system properly handles attempts to create accounts with incomplete or incorrect information and prevents invalid registration.

3. Logging In:

Feature verified: Logging in both with invalid and valid credentials.
Details:

1. Logging in with valid credentials
    a. Input: valid credential log in, in the database.
    b. Expected Output: Logged in successfully.
    c. The user logs in with valid credentials stored within the database, by using the login page. Then the user is successfully logged in and linked up to the credentials inputted. Verifies the security and reliability of logging into the system under an already existing account.
2. Logging in with invalid credentials
    a. Input: invalid login credential not in the database
    b. Expected Output: “invalid login”.
    c. The user uses invalid credentials not stored within the system to attempt to log in. They should be prompted by the system with a message indicating the wrong username or password. Ensuring that each unique account must be registered and stored in the database before attempting to log in.

4. Loyalty Program:

Feature verified: Joining the Loyalty Program with both a preexisting account and no account.
Details:

1. Joining loyalty program with account
    a. Input: The account is logged in and attempting to join the program.
    b. Expected Output: The user is successfully registered to the program and receives all benefits alongside it.
    c. The user is logged in and navigates to the Join Loyalty program page under the account. Confirm to join the program and the system should respond by making the account a part of the loyalty program. Ensuring the functionality of the loyalty system and its interaction with pre-existing accounts.
2. Attempting to join the loyalty program without an account
    a. Input: Not logged into an account trying to join the program.
    b. Expected Output: The user will not find the option to join on the page.
    c. The user does not have an account, and will not find access to join the program as an account is required to reach the page under the account section. Accounts for the error testing if the system will adjust to any unforeseen changes.

5. Viewing Purchase History:

Feature verified: Viewing purchase history in both an account that has a payment history and a newly created account with no history.
Details:

1. An account with a payment history
    a. Input: user login.
    b. Expected Output: Show Purchase History.
    c. The user logs into their account and navigates to the purchase history page, where the system displays a list of all transactions made by the user. This ensures the system is accurately retrieving and displaying a comprehensive history of user's transactions, by being able to handle data retrieval and presentation for user-specific information. Also, the system should display a message indicating no transactions found if the user has never made a purchase.

6.3 Testing Approach:

1. Planning
2. Execution
3. Tracking and Data Collection
4. Recalibration

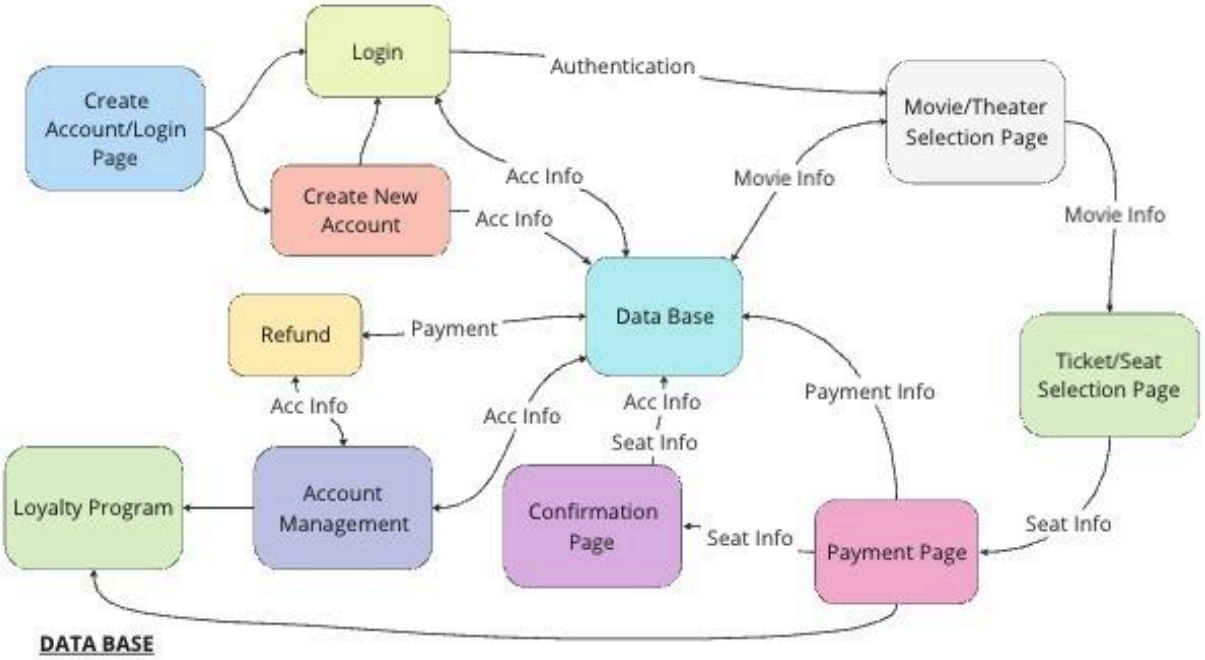
6.4 Test Case Samples:

Test Cases Samples										
	TestCaseID	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
1	search_movie_1	search_module	P1	verify that users can search for movies by title	User is logged in, the movie database is populated	• Go to the search bar Enter the movie title "Inception" Press the enter button	Search result should displays movies related to "Inception"	Search result does display movies related to "Inception"	Pass	Elias Mapendo
2	search_movie_2	search_module	P2	verify the system handles invalid search terms gracefully	User is logged in	Go to the search bar, Enter the movie title "abcd1234", Press enter button	Display a message indicating no result found, but still shows movies that might contain that term	System shows movies that have no connection to "abcd1234" or the movie exist but not showing	Fail	Elias Mapendo
3	refund_ticket_1	payement_module	P1	verify the system correctly refunds purchase of ticket	Ticket is already pruchase under an preexisting account, and time frame is at least 2 hours before showtime	Go to user's account locate the refund section, select desired ticket to refund, Click confirm	Prompts a display indicating refund confirmation, and the ticket should be removed from the user's account, plus the money should be refunded	Refund confimation is displayed, ticket is removed from user's account, and money is refunded	Pass	Lucas Weinstein
4	refund_ticket_2	payement_module	P1	verifies the system correctly denies any invalid refunds	Ticket is alread purchased under an preexistin account, time frame for refund has already passed	Locate refund section under user's account, selects movie ticket that has already passed	Should not give user an option to refund ticket, prompting a message informing the user of the reason	Ticket is not availble for purchase, user is prompted with message	Pass	Lucas Weinstein
5	purchase_ticket_1	payment_module	P1	Verify that a purchased ticket has correct seat, theater, movie, and time	Payment must be valid and have gone through our system	Once ticket is purchased, click "View Ticket"	Seat information should be correctly displayed including the correct time, theater, movie title, and seat	Correct seat information is displayed, including correct time, theater, movie title, and seat	Pass	Jacob Silva
6	purchase_ticket_2	payment_module	P2	verify that user cannot buy more than 20 tickets	Movie is available, tickets are available, and time frame to buy tickets has not passed	Go to search bar, enter movie to search, select movie, select 25 tickets to purchase, click confirm	System should prompt error message displaying too many tickets are being attempted to purchase sending user back to the ticket selection page	Message is correctly prompted and user is sent back to the ticket selection page	Pass	Lucas Weinstein
7	change_name_1	account_module	P2	Ensure that users can change the name associated with their account in standard naming convention(no numbers, special characters, emojis, etc.)	User is logged in	Click on Account Tab, Click on "edit", delete the current name on the account, type in "John Appleseed"	System should correctly changes the name to what the user specifies since it is in standard naming convention	The users name is correctly changed to John Appleseed	Pass	Jacob Silva
8	change_name_2	account_module	P2	Ensure that users cannot change their name to anything that is not standard naming convention (numbers, special characters, emojis, etc.)	User is logged in	Click on Account Tab, Click on "edit", delete the current name on the account, type in "123 !!"	System should tell customer to try again with no special characters, numbers, or emojis.	The system tells the customer " '123 !!' contains a number and/or a special character and/or an emoji. Please enter a new name."	Pass	Jacob Silva

9	account_creation_1	account_module	P1	Confirms that the user can register and create an account	Has preeexisting email, phone number	Click on Account Creation Tab, Fill in specified information including, username, password (containing number, special character, uppercase letter), name, address, email, phone number, Click confirm creation of account	Account should be created	Account is created, allowing user to be loged in	Pass	Lucas Weinstein
10	account_creation_2	account_module	P2	Confirms that the user cannot register and create an account with invalid data		Click on Account Creation Tab, Fill in specified information including, username, password, name, address, email, phone number in which password is invalid not meeting the prerequisits, Click confirm creation of account	Message should be prompted on which specific area (password) failed to meet the account requiremnts and user's focus should move to the (password) area	Message is displayed correctly to user (password lacks requirements), focus of user is on the (password) area that	Pass	Lucas Weinstein
11	view_purchase_history_1	user_account_and_payment_module	P2	verify that can view their booking history, and being recorded correctly	User has previous booking	Login into the user account, Nagivate to "Booking History"	The user can view an accurate list of their booking history	if user has had prevoius purchases, the history appears else its empty	Pass	Elias Mapendo
12	view_purchase_history_2	user_account_and_payment_module	P2	verify that can view their booking history, and being recorded correctly even though on purchases are made	User has no previous booking and payment history	login into the user account, Nagivate to "Booking History"	the list/history is empty since no previous payments/booking	A payment appears even though they've never made a payment	Fail	Elias Mapendo
13	user_login_1	authentication_module	P1	verify that a user who created an account previously can login with valid credential	user account account exist	go to website, click the login button, enter valid username & password, click login	the user should be logged in and directed to the dashboard	the user logins in and reaches the dashboard	Pass	Elias Mapendo
14	user_login_2	authentication_module	P1	ensure that invalid credentials can't gain access to the system	user account account doesn't exist	go to website, click the login button, enter username & password, click login	the system should display message say invalid username or password or both	the incorrect password or username does appear and the user is prompted to change credentials or create new account	Fail	Elias Mapendo
15	loyaltyPrgm_join_1	account_module	P2	verify that the user can join the loyalty program	user has a preexisting account	locate account section, click on loyalty program, click confirm	The system should give the user a confimation message and the user shuold be registered as apart of the loyalty program	The user is given the confimation message, now the account is apart of loyalty program, including all benefits	Pass	Lucas Weinstein
16	loyaltyPrgm_join_2	account_module	P2	verify that the user cannot join the loyalty program if account is not created	user does not a preexisting account	Click on account, click login	Shold not give the user an option to join the loyalty program if no account is logged in/registered	User cannot see/click on the loyalty program join option	Pass	Lucas Weinstein

7. Data Management Strategy

7.1 Software Architecture Diagram (Updated):



DATA BASE

Movie Info:
movies, times, locations, prices, seats, theaters, theaterTypes, theater details

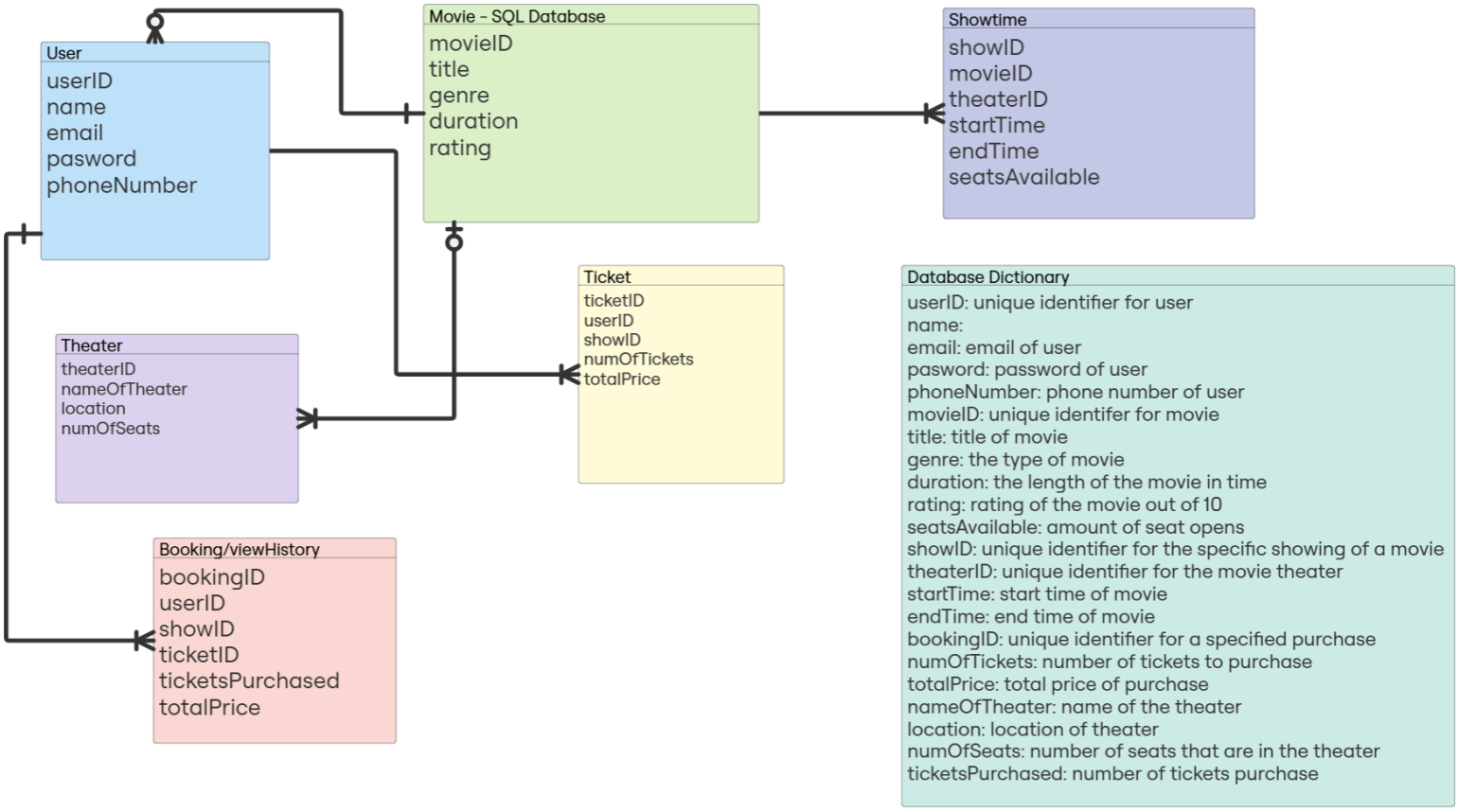
Account Info:
username, password, email, phoneNumber, loyalty points, purchase history

Seat Info:
Seat number, movie, price, theater

Payment Info:
paymentID, date, paymentType, price, user_id

Updates: None

7.2 Data Management Strategy Diagram:



7.3 Data Management Details

7.3.1 Amount of Databases, Type, and Why?

In our strategy, we chose 6 databases all in SQL. This is because all of our data is comfortably stored and organized in a spreadsheet type format as opposed to documents. Additionally, we chose to make 6 databases in order to keep all of our information organized. The user database holds all of our user's unique IDs, as well as their passwords, emails, names and phone numbers. The movie database contains all of the movie's unique IDs, movie titles, genres, duration, and ratings. Our showtime database contains details about specific showings of movies, such as their unique show ID, the movie ID they are connected with, the theater ID which identifies the theater in which they reside, the start and end time of the showing, as well as how many available seats there are at any given time. The theater Database holds information such as the theater ID which identifies a specific theater, the name of the theater name, the location for which city it is in, as well as the number of seats per room within the theater. The ticket database contains the unique ticket ID for a ticket, the user ID of the user it belongs to, the showID for the show it is connected to, the number of tickets contained within the order, and the total price for the order. Finally, the booking history database would contain the booking ID which is the specific ID for the purchase, the userID for the user in which the ticket had belonged to, the showID for the show that the ticket was for, the ticketID for the ticket that had been redeemed, the total number of tickets purchased in the account history, and the total price of the individual purchases.

7.3.2. Tradeoff

1. Choice of SQL Database
    a. ACID Properties: Sql databases provide strong ACID (Atomicity, Consistency, Isolation, Durability) guarantees, which are essential for maintaining data integrity, especially in transactional systems.
    b. SQL is well-suited for applications data is structured and relationships between entities are well-defined, which perfectly aligns with the Movie Theater Ticketing System' need to manage users, movies, theaters, showtimes, and bookings.
    c. SQL provides a powerful and flexible query language that supports complex queries, joins, aggregations and transactions.
2. Database Justification
    a. Number of Databases
        i. Single Database - A single SQL database manager with 6 included databases are sufficient to manage all the data for the Movie Ticketing System, simplifying data management , backup, and ensures referential integrity across all entities.
3. Design Decisions
    a. Normalization: Third Normal Form ensures that the data is normalized to avoid redundancy and maintain data integrity. Each table contains data related to a single entity and relationships are maintained through foreign keys.
4. Possible Alternatives and Trade-offs
    a. NoSQL
        i. Scalability and Flexibility - While NoSQL databases offer these benefits, they might not be necessary for the structured data and transactional nature of a movie ticketing system.
        ii. Consistency - SQL's strong consistency model (ACID) is better suited for ensuring data integrity in booking systems compared to the eventual consistency model of many NoSQL databases.
    b. Separate Databases
        i. Performance Optimization - While having separate databases for different entities (e.g., users, bookings) can optimize performance, it increases complexity in management and data consistency.
        ii. Complex Transactions - Ensuring consistency across multiple databases requires complex distributed transactions, which can be difficult to implement and maintain.

8. 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