Cosmos Company

Movie Ticketing Software System

Software Requirements Specification

Version 2.0

January 31, 2024

Group 6
Dominic Parker
Alex Vo
Kevin Le

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

Revision History

Date	Description	Author	Comments
02/13/2024	Version 1	Group 6	First Revision
02/14/2024	Version 2	Group 6	Second Revision
02/28/2024	Version 3	Group 6	Third Revision
03/13/2024	Version 4	Group 6	Fourth Revision
03/27/2024	Version 5	Group 6	Fifth Revision

Document Approval

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

Signature	Printed Name	Title	Date
Dominic Parker	Dominic Parker	Student, CS 250	02/13/2024
Alex Vo	Alex Vo	Student, CS 250	02/13/2024
Kevin Le	Kevin Le	Student, CS 250	02/13/2024
	Dr. Gus Hanna	Instructor, CS 250	

Development plan and Timeline

- **Week 1: Project Kickoff and Planning**
  - Responsibilities: Entire Team
  - Conduct a project kickoff meeting to discuss goals, expectations, and timelines.
  - Collaboratively define the purpose and scope of the Movie Ticketing Software System.
  - Identify potential challenges and risks.
  - Establish a communication plan.

- **Week 2: Requirements Gathering**
  - Responsibilities: Entire Team
  - Begin the Software Requirements Specification (SRS) document.
  - Gather references and research existing movie ticketing systems.

Kevin Le: Purpose and Scope
Dominic Parker: Product Perspective and Functions
Alex Vo: Interface and Functional Requirements

- **Week 3: Detailed Requirements and Use Cases**
  - Responsibilities: Entire Team
  - Elaborate on specific requirements, user scenarios, and system behaviors.

Dominic Parker: User Characteristics, Non-Functional Requirements
Alex Vo: Use cases, Classes/Objects, Diagrams
Kevin Le: Use cases, Diagrams.

- **Week 4-5: System Design**
  - Responsibilities: Entire Team
  - Define the overall system architecture, constraints, and high-level design.

Dominic Parker: Design Constraints, UML Diagram
Kevin Le: Software Architect Diagrams
Alex Vo: Development plan

- **Week 6-7: Prototyping and Validation**
  - Responsibilities: Entire Team
  - Develop a prototype for key features.
  - Validate the prototype with potential users and gather feedback.
  - Refine the design based on user input.

- **Week 8-10: Implementation**
  - Responsibilities: Entire Team
  - Begin coding based on the refined design.
  - Collaborate on integrating components.
  - Conduct regular code reviews.

- **Week 11-12: Testing and Quality Assurance**
  - Responsibilities: Entire Team
  - Implement comprehensive testing procedures.
  - Identify and fix bugs.
  - Ensure the system meets specified requirements.

- **Week 13-14: Documentation and Finalization**
  - Responsibilities: Entire Team
  - Complete the SRS document.
  - Document codebase and procedures.
  - Prepare for final presentation and delivery.

- **Week 15: Final Review and Submission**
  - Responsibilities: Entire Team
  - Conduct a final review of the software.
  - Ensure all documentation is complete.
  - Submit the final version of the Movie Ticketing Software System.

Table of Contents

1. Introduction

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire SRS with purpose, scope, definitions, acronyms, abbreviations, references, and overview of the SRS. The aim of this document is to gather and analyze an in-depth insight into the **Cosmos Company software system** in accordance with the requirements provided by the **Cosmos Company.**

1.1 Purpose

The purpose of this SRS is to carefully document and address the requirements to create an efficient online system to manage the transactions of movie tickets between consumers and the client. Also, we shall detail the test cases and goals with respect to the requirements and provide our own feedback as well.

In short, the purpose of this document is to provide a detailed overview of the software product, address the requirements, test cases, and provide feedback.

1.2 Scope

(1) The scope pertains to making the Movie Ticketing System for Cosmos Company live, allowing for online sales, distribution, and marketing of movie tickets.

(2) The Movie Ticketing System will provide a User Interface (UI) for the consumer, that will allow consumers to (securely):

1. Select seats, purchase tickets, payment processing
2. Resell tickets
3. Make refunds
4. Browse movies, search movies, view movies details
5. Select theaters, view showtimes, filter showtimes
6. View booking history, cancel or modify booking, download, or email tickets
7. Create and manage accounts, sign in and out

(3) One goal is providing the consumer with a user-friendly UI that will prove effective in high consumer return rates. Another goal is to provide the consumer with a full range of capabilities that will satisfy the requirements of the client.

The scope of this product will provide the consumer with the requirements of the client that will allow them to have access to the listed requirements (1-7) above.

1.3 Definitions, Acronyms, and Abbreviations

UI	*User Interface, the website consumers will interact with*
User friendly	*An effective website that consumers can easily interact with and is not confusing*

1.4 References

The references are:
https://www.fandango.com/
https://www.atomtickets.com/
https://www.movietickets.com/

1.5 Overview

The remaining sections of this document will provide a general description of all the characteristics, functionalities, test cases, constraints, assumptions made during the creation of the **Movie Ticketing System.** Section 2 will provide the general factors that affect the products and their requirements. It should be noted that Section 2 does not state specific requirements and is only to make those requirements easier to understand. Section 3 will address the specific requirements such as functions, data, and constraints of the product. Section 3 will go over some of the use cases as well. Section 4 will address supporting information such as visual diagrams, use case analysis, etc.

2. General Description

The movie ticketing software system contains a solution to streamline the ticket purchasing process for users and manage the operational aspects of theater ticket sales. It aims to cater to both the customer's ease of access to buying tickets and the theater's need for efficient management of showtimes, seating, and customer engagement. This overview sets the stage for a deeper dive into the system's capabilities, emphasizing its role in enhancing the movie-going experience while addressing the business requirements of the theater.

2.1 Product Perspective

The theatre ticketing system is a standalone web-based application that is intended to integrate with existing theater databases. It will show a database of showtimes, available tickets, and user accounts, allowing for real-time updates and information that is accurate. It is its own application but can interact with online review sites, payment gateway, and any other needed software interfaces.

2.2 Product Functions

The theater ticket system is designed to be a web-based platform that can support a large user load with the capacity. It should be able to handle this large load concurrently as users engage in ticket purchases and inquiries. Security is an important factor, as advanced measures must be taken against unauthorized bot activity, ensuring a fair purchasing process. A core part of its functionality should be its real-time interaction with a database that provides up-to-date information on showtimes and ticket availability. Users should have a purchasing limit as well.

The system should also have an administration mode, allowing authorized personnel the ability to manage the system's operations effectively. A customer feedback mechanism should also be

incorporated allowing the gathering of valuable user insights. A discounting structure is also in place to cater for customers and should accommodate both digital and physical ticket formats. Furthermore, the system should present movie reviews and critiques from online sources.

2.3 User Characteristics

The users of the system would include online customers and theater staff that use digital kiosks. Users can range from tech-savvy individuals to those with minimal technical experience. This means it is necessary to have an intuitive and easy-to-navigate graphical UI. The system will offer multilingual support, allowing for English, Spanish, and Swedish speakers. There will also be an option for enhanced customer service and transaction management.

2.4 General Constraints

- The system must operate efficiently as a web-based platform without the need for an app.
- It should prevent scalping by ensuring tickets are unique and secure.
- Tickets should be able to be delivered both digitally and physically.
- It must be able to handle a high volume of users and transactions simultaneously.

2.5 Assumptions and Dependencies

- It is assumed that users will have access to a compatible web browser.
- The performance of the system is dependent on the ability of the database to handle multiple concurrent transactions.
- The availability of third-party services for payment, processing, and reviews is assumed.
- The effectiveness of bot prevention measures may depend on evolving tactics.

3. Specific Requirements

The specific requirements are –

3.1 External Interface Requirements

3.1.1 User Interfaces:

- Should be intuitive and user-friendly but also provide a professional-looking ticket booth.
- Should be graphical through

3.1.2 Hardware Interfaces:

- Should be minimized/smallest-possible app/platform to run even on a smartwatch
- Should be compatible with standardized web browsers (Chrome, Safari, etc.)
- Should be accessible mainly through laptop and desktop devices.

3.1.3 Software Interfaces:

- Should be synchronized and providing real-data back and forth between devices in real time
- Should be integrated with payment gateways for secure, fast, seamless transactions

- Should be implemented with movies and prices information updates constantly
- Database, review software that displays onto the website, Messaging/Notification software
- Should have Payment Gateway Interface to process payments securely and integrate with services like PayPal, Apple Pay, etc.
- Should have a Movie Database API to fetch information about movies including titles, release dates, ratings
- Should use frontend languages/technologies such as React.js framework, CSS, HTML, and JavaScript
- Should include a relational database with API's built using Python Flask

3.1.4 Communications Interfaces:

- The system shall use HTTPS for secure communication.

- API endpoints for integration with external systems shall be documented.

3.2 Functional Requirements

3.2.1 User Registration and Authentication

3.2.1.1 Introduction: users could email to register/log in for accounts and use phone number to verify identity.
3.2.1.2 Inputs: emails, passwords, phone numbers, DOBs
3.2.1.3 Processing: The system shall verify user credentials and send confirmation emails for registration.
3.2.1.4 Outputs: successful verification directs to home page of the website.
3.2.1.5 Error Handling: an alert pops up to notify the user input that causes error (wrong password, seats unavailability, etc.)

3.2.2 Movie Selection and Booking

3.2.1.1 Introduction: users can browse, select movies, choose seats, and pay.
3.2.1.2 Inputs: Selection of movie, date, time, and preferred seats.
3.2.1.3 Processing: The system shall check seat availability and process the booking.
3.2.1.4 Outputs: Booking confirmation with QR code for entry.
3.2.1.5 Error Handling: //Same as above//

3.3 Use Cases

3.3.1 Use Case #1: Canceling a Booking – User decides to cancel a booked movie ticket.

a. User logs into their account.

b. User navigates to the transaction or booking history section.

c. User selects the booked movie ticket they want to cancel.

d. User initiates the cancellation process.

e. System verifies the cancellation request and prompts the user for confirmation.

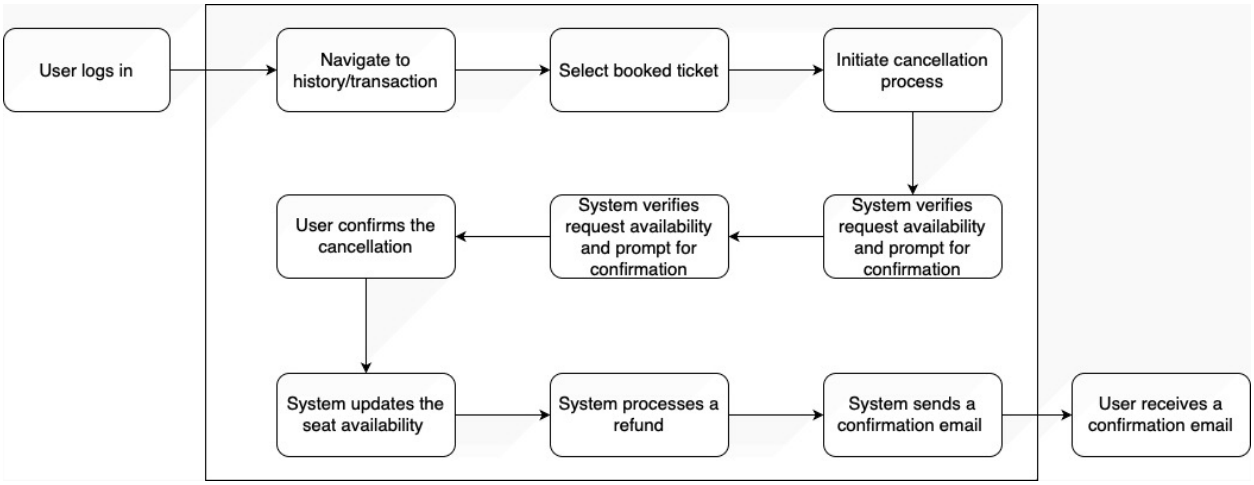
f. User confirms the cancellation.

g.System updates the seat availability for the canceled ticket.

h.System processes a refund if applicable.

i. System sends a confirmation email to the user.

- Preconditions:
  - Users must be logged into their account.
  - Users must have a valid booked ticket.
- Postconditions:
  - The booked ticket is canceled.
  - If applicable, a refund is processed.
  - User receives a confirmation email.



3.3.2 Use Case #2: Leaving a Review for a Movie – User wants to share their feedback and leave a review for a movie they have watched.

a. User logs into their account.

b. User navigates to the movie history or review section.

c. Users select the movie they want to review.

d. User initiates the review process.

e. System displays a review form with fields such as rating, title, and comments.

f. User fills in the review form, providing a rating, title, and optional comments.

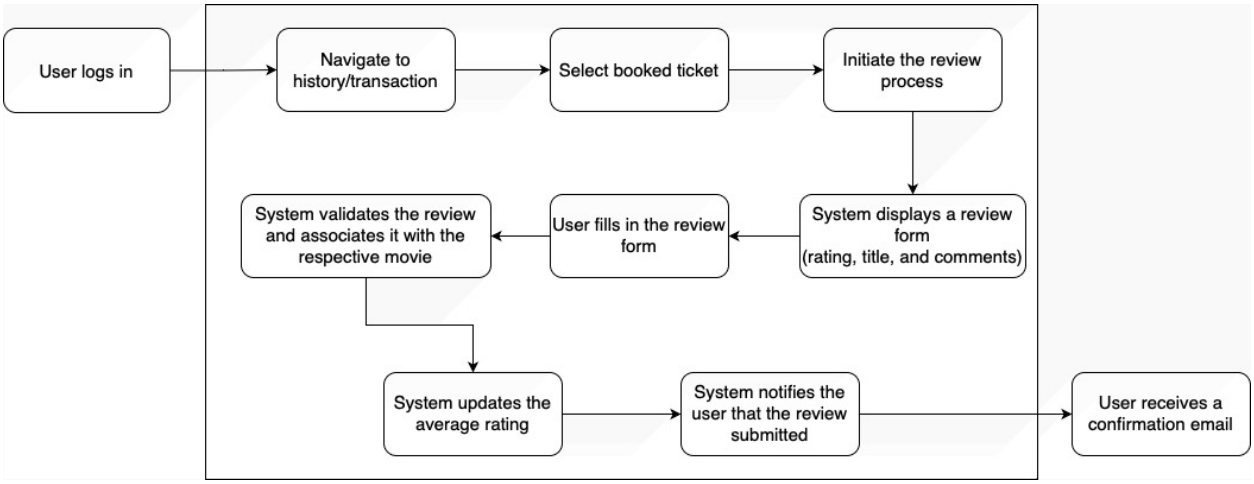
g. User submits the review.

h. System validates the review and associates it with the respective movie.

i. System updates the average rating for the movie based on the new review.

j. System notifies the user that the review has been successfully submitted.

- Preconditions:
  - Users must be logged into their account.
  - Users must have watched the movie they want to review.
- Postconditions:
  - The review is successfully submitted and associated with the movie.
  - The average rating for the movie is updated.



3.3.3 Use Case #3: Purchasing a Ticket for a Movie – User wants to purchase a movie ticket for a movie.

a. User logs into their account.

b. User navigates to the search bar and searches up the movie they want to see.

c. User selects the move they want to watch.

d. System displays a screen with movie information regarding the description, runtime, casts, box office, etc. The system also displays the current date with 11 days from the current date and underneath, it also displays movie theaters near the user's zip code. It will also display the different movie viewing options such as standard, Dolby Cinema, IMAX, etc. It will also include the show times for each movie theater. If the movie time is still available, it will appear up as a blue, if no more seats or if the showing is not available it will show up as black.

e. Users will then select the move time they want to see it at, and the system will route them to a new page that displays a visual of the seats of that specific theater, and seats available will be lit up as blue, if not, it will be clear.

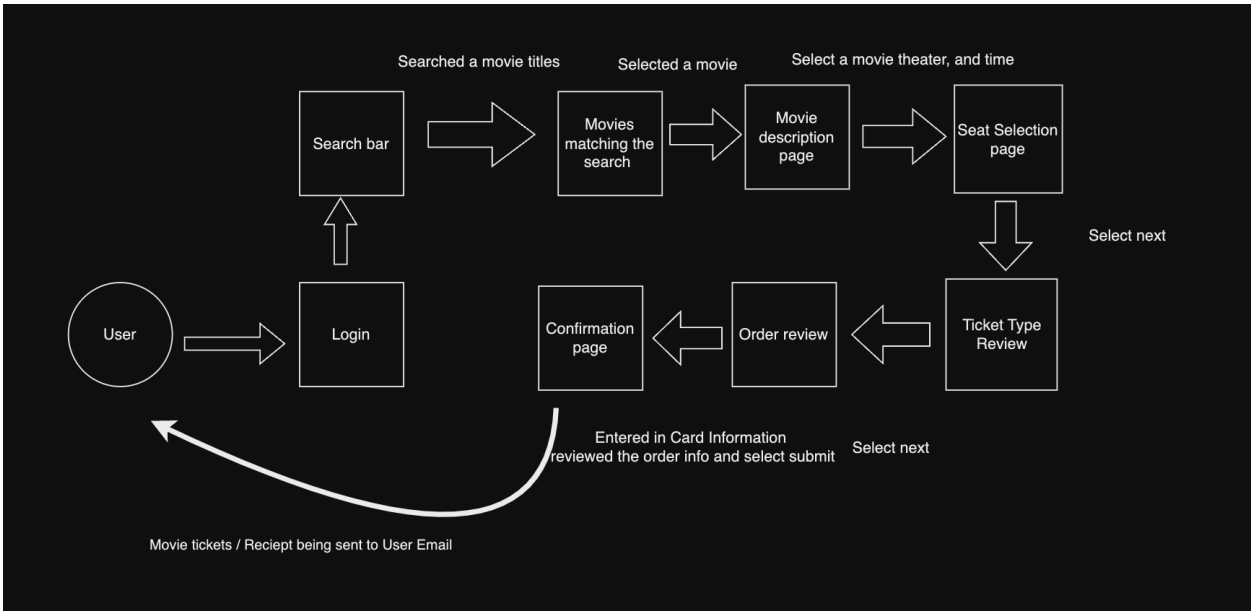
f. The user will then choose which seats they want and for each seat they click, the system will automatically update how many tickets they are selecting. When ready, they will click the ‘NEXT” button which will take them to review your movie ticket screen where they will select the ticket type according to children, adults, or adults. The system will automatically adjust the ticket totals, so they are not selecting more than how many seats they wanted.

g. User will hit next when ready and will be directed to a checkout screen where they must enter in their card information (card number, expiration month and year, billing zip code and security code).

h. When ready, the screen will display the total showing how many tickets, ticket type, the price including tax and User will hit submit.

i. User will then be sent an email with the receipt and another email that includes the digital tickets they ordered with a unique confirmation ID. In the Users account page/orders page will show the receipt and information displaying the ticket information.

- Preconditions:
    - Users must be logged into their account.
    - User has selected a movie to watch.
    - User has chosen a specific showtime and movie theater.
- Postconditions:
    - The transaction went through successfully.
    - User received the ticket unique confirmation ID in their email/home page/account order.



3.4 Classes / Objects

3.4.1 User Account

3.4.1.1 Attributes: Username, email, password, transaction history.

3.4.1.2 Functions: Register, log in, view transaction history.

3.4.2 Movie

3.4.2.1 Attributes: Title, genre, release date, available seats.

3.4.2.2 Functions: Retrieve movie details, update seat availability.

3.4.3 Ticket

3.4.2.1 Attributes: Ticket ID, Movie ID, User ID, Seat number, booking date and time, Showtime, Status (booked, canceled, etc.)

3.4.2.2 Functions: Generate ticket ID, Reserve a seat for a specific showtime, Cancel a booked ticket, Retrieve ticket details.

3.4.3 Booking

3.4.2.1 Attributes: Booking ID, User ID, Movie ID, Date, Seats, Status (e.g., confirmed, pending, canceled).

3.4.2.2 Functions: Create Booking, Cancel Booking, Retrieve Booking Details, Update Booking Status (e.g., from pending to confirmed), Calculate Booking Price, Check Seat Availability.

3.5 Non-Functional Requirements

3.5.1 Performance

- 95% of transactions shall be processed in less than 3 seconds.
- The system shall support concurrent transactions for at least 500 users.

3.5.2 Reliability

- The system shall have a Mean Time Between Failures (MTBF) of at least 30 days.

3.5.3 Availability

- The system shall have an uptime of 99.9%.

3.5.4 Security

- User passwords shall be encrypted and stored securely.
- Payment transactions shall use secure encryption protocols.

3.5.5 Maintainability

- The system shall provide an admin interface for easy maintenance tasks.

3.5.6 Portability

- The system shall be compatible with major operating systems.

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

3.7 Design Constraints (do 3.7)

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

The design and development of the web-based movie ticketing theater system are subject to several constraints that are critical to ensuring compliance with industry standards, company policies, and technical limitations. These constraints will influence various aspects of the software project, from choice of technology to deployment strategies.

3.7.1 Compliance with Web Standards

- The system shall adhere to the latest web standards as defined by the World Wide Web Consortium (W3C) to ensure compatibility and accessibility across different browsers and devices.

3.7.2 Scalability and Performance

- The architecture must support scalability to handle peak loads during major movie releases or promotional events, ensuring a responsive user experience without degradation in performance.

3.7.3 Security Standards

- The application shall comply with current cybersecurity standards, including data encryption for sensitive information such as payment details and personal user data, to protect against breaches and unauthorized access.

3.7.4 Payment Processing Compliance

- Integration with payment gateways must adhere to the Payment Card Industry Data Security Standard (PCI DSS) to ensure secure transaction processing.

3.7.5 Accessibility Requirements

- The design shall meet the accessibility guidelines outlined in the Web Content Accessibility Guidelines (WCAG) 2.1 Level AA to ensure the system is usable by people with a wide range of disabilities.

3.7.6 Localization and Internationalization

- The system must support multiple languages and cultural settings to cater to a global audience, including date formats, currency conversion, and content localization.

3.7.7 Environmental Constraints

- Hosting and operational practices should align with green computing principles to minimize the environmental impact of running the system, including energy-efficient coding practices and selection of eco-friendly hosting services.

3.7.8 Legal and Regulatory Compliance

- The system shall comply with local and international laws and regulations related to digital commerce, digital rights management (DRM), and privacy laws, including the General Data Protection Regulation (GDPR).

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

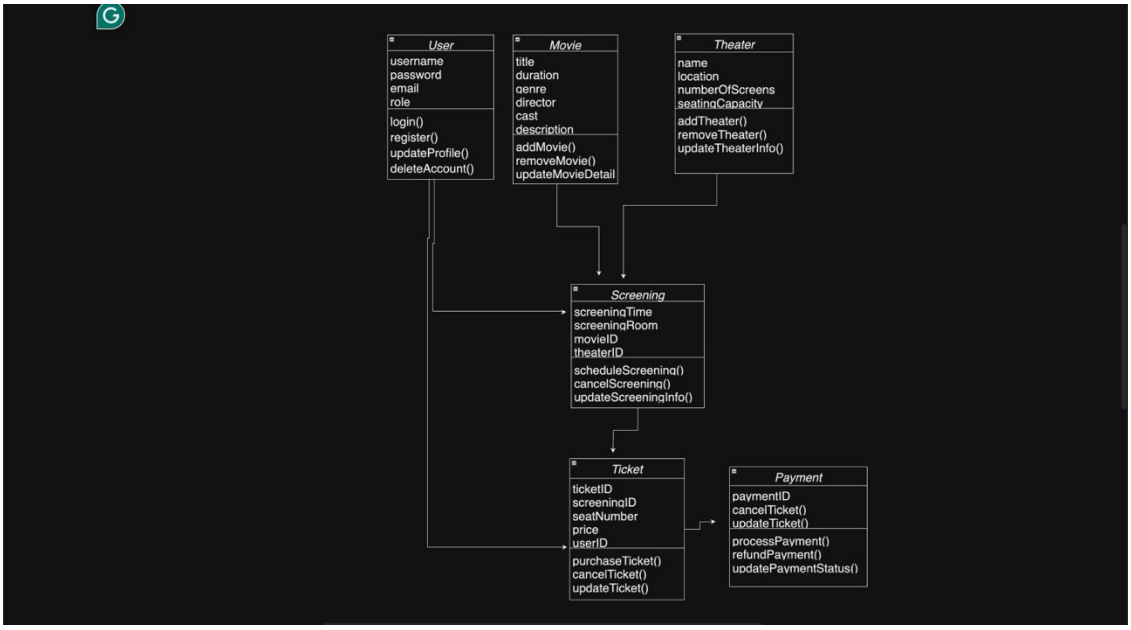
List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS's requirements.

4.1 UML Diagram

The UML Class Diagram for our web-based movie ticketing system is a structured representation that outlines the system's core functionalities and their connections. At its foundation, the diagram segments the system into six primary classes: User, Movie, Theater, Screening, Ticket, and Payment, each equipped with specific attributes and operations that define their role within the system. The User class, characterized by attributes such as username, password, email, and role, facilitates operations including login, registration, profile updates, and account deletion, allowing for both administrative and customer needs. Movie and Theater classes serve as the system's backbone, storing essential details like movie titles, durations, genres, and theater locations, and offering functionalities for adding, removing, or updating their respective entities. The Screening class bridges Movies and Theaters, organizing screenings with attributes that specify times, rooms, and linking IDs, while enabling scheduling and cancellation operations. Tickets are closely tied to Screenings, showing seat allocations and prices, and are purchasable or cancellable by users, showcasing the direct interaction between the system and its end-users. Lastly, the Payment class completes the transactional aspect of the system, managing

financial exchanges with attributes for payment ID, amount, method, and transaction status, alongside operations for processing and refunding payments.

This diagram not only highlights the individual components and their functionalities but also illustrates the relationships between them—such as Users purchasing Tickets for Screenings of Movies in specific Theaters and making Payments for those Tickets. These relationships are important for understanding how the system facilitates the journey from movie selection to the actual purchase of tickets, ensuring a seamless user experience. Through this detailed class structure and the defined operations, the UML Class Diagram provides a detailed blueprint for developers, laying the groundwork for the development and maintenance of the movie ticketing system, ensuring all components interact harmoniously to meet the users' needs.

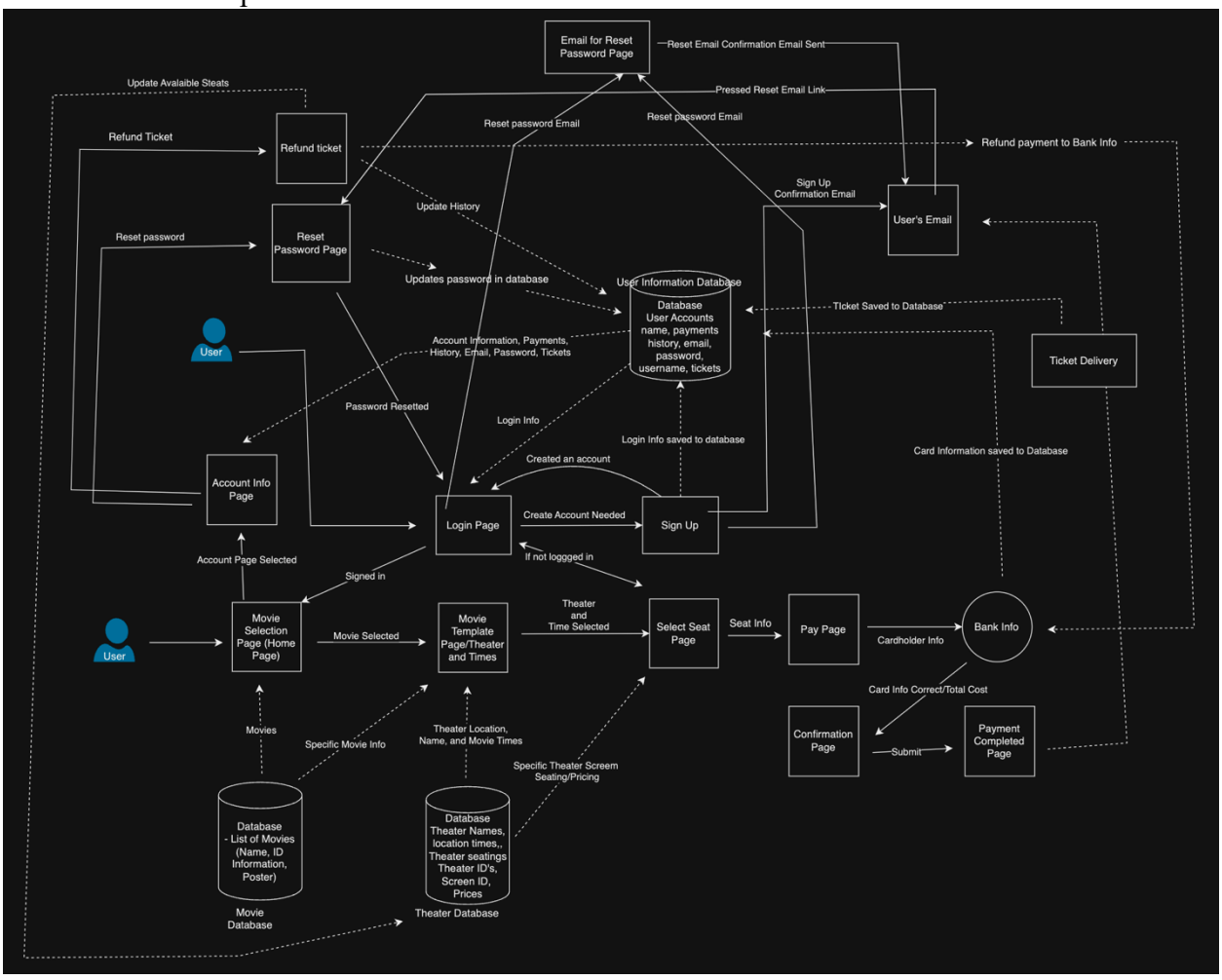


4.2 Software Architect Diagrams (SWA)

The SWA Diagram for our web-based movie ticketing system is a structured representation that outlines the architecture of the application. It outlines the different components and connections that make up the system and how they are configured to work together.

There are 3 databases: User Information Database, Movie Database, and Theater Database. The User Info Database the User’s account information such as name, username, password, email address, ticket information, payment information, and history. The Movie Database contains the list of movies, id, and information such as descriptions, and poster images. The Theater Database has theater names, location, movie times, Screen ID’s, Theater ID’s, and Movie Ticket Prices. The 2 main points of access to our system are the Login Screen and the Movie Selection Page that doubles as the Home Page. The Login Screen is where the User is prompted to login into their account to have access to their Account Information Page; it is not necessary to login and the User could create an account if they wanted in which they will be directed to the Sign-Up Screen. The Sign-up Screen would require a full name, email, password, and username to create

an account. The Sign-Up Screen, Login Screen and Account page has options to reset the password, which will redirect to the Reset Password Screen. After resetting password, User will be directed to Login Screen again. If Users access the system from the Home page, they would be able to look and select the movie they want to go see. They will be directed then to the specific movie page which uses the Movie Template page. The Database uses the Movie ID to fill in the Movie Template Page and Theater Database page passes in available theaters, screens. and screening times. User then selects the available theater and time which redirects them to Select Seat Page where they choose the available seats. When they click next, if not signed in, they will be directed to Login Page. When they login they will be directed back to the Select Seat Page and then they can continue to Pay Page. The Pay Page is where they enter information for payment and billing which then takes them to the Confirmation page, and they can review the total and their tickets. When ready, they click submit in which they get charged, and Tickets gets sent to their email and into the User Info Database in which their tickets/card information gets stored. To access their Account Page, they must go back to the Home Screen and in the Account Page, they are able to review all their user information because the User Info Database passes information into there. They would be able to refund their ticket and as well update their password. If they were to refund their ticket, they would receive a refund to their bank account, and it would also update the Theater Database.



4.3 Software System Overview

The Movie Ticketing System is a web application that has 6 overarching classes outlined by the UML Diagram of the system: Users, Movie, Theater, Screening, Ticket, and Payment. The user class has 4 attributes: username, password, email, and role. Functions include login, register, update profile, and delete account features. The Movie class attributes include title, duration, genre director, cast, and description. The functions are adding movies, removing movies, and updating movies. The Theater class includes name, location, number of screens, and seating capacity. The functions include adding theater, removing theater, and updating the theater. The Screening class attributes include screening time, screening time, movie ID, and theater ID. The functions include schedule screening and cancel screening. The Ticket class includes ticket ID, screening ID, seat number, price, user ID. The functions include purchasing tickets, canceling the ticket, and updating the ticket. The payment class includes payment ID. The functions include canceling tickets, updating tickets, processing payments, refund payments, and updating payment status. The Payment class attributes include payment ID. The functions include ticket, update ticket, process payment, refund payment, and update payment status.

The SWA Diagram goes more into depth about the different smaller components that connect to each other and make up the entire application. Think of the SWA as the more detailed in-depth diagram about how each component interacts with another and how the user may interact with the application. There are 3 databases, Movie, Theater, and User Information. All 3 databases are what the system relies on to be functional because they contain the information necessary for the system to pull from the users. The main pages and routes are the Home Page, Login/Sign up pages, Ticketing/Payment route. The main page is where the user has access to all the movies available is the main hub connecting the Login page and Sign-up Page and to the Account page and the Ticket/payment route that the users take to secure a ticket. Alex Vo will oversee the main page; Dominic Parker with the Login/Sign Up features and Kevin will oversee the Payment route. Each part should take a month to complete Overall, the UML AND SWA Diagram outline a bigger picture and a more in-depth view of the Movie Ticketing System.

4.4 Test Plan

	TestCaseId	Component	Priority	Description/Test Summary	Test Steps	Expected Result	Actual Result	Status	Test Executed By
2	SignUp_1	SignUp_Module	P0	Verify when a user is able to press sign up to acess the sign up page and enter in name, email, password, and date of birth and whenSign up button is pressed, it will sucessfuly sign them up and send confirmation email to User.	1. Press the Sign up button 2. Enter in name, email, password, and date of birth 3. Press Sign up button and sucessfully creates an account.	Account should be created within the User_Information database	Account created within the User_Information database.	Pass	Kevin Le
3	Login_2	Login_Module	P0	Verify when a user is able to press login and takes them to login page and then enter in email and password and presses the actual login buttom, it will take them to the Main page if it worked and if it is wrong, will display it is wrong. It should also render a 'Logout button if user is logged in.	1. Press the Login on Main Page, 2. Takes them to Login Page 3. Enter in correct email and password 4. Press login button Test 2: 1. Press the Login on Main Page, 2. Takes them to Login Page 3. Enter in wrong email and password 4. Press login button	If email and password wrong, then it should prompt user 'Email or password wrong.' If Login Info correct, then it should take user to Main Page and render a logout button.	Test 1 (Correct User Info): User logged in sucessfully, taken to Main Page, and log out button sucessfully rendered. Test 2: (Wrong User Info): User not taken to Main Page, and 'Email or password wrong' displayed	Pass	Kevin Le
4	Payment_3	Payment_Module	P0	Verify when at the Payment Page, the user is able to enter in Card information, Expiration Date, CVV, and Name. When press submit, it will process the payment and charge the user the correct amount.	1. Enter in Name, Card Information CVV, Expiration Date 2. press Submit and will charge customer.	User will be charged the total amount.	User was charged the total amount.	Pass	Kevin Le
5	ResetPassword_4	Reset_Password_Module	P0	Verify when at the Login or Sign Up page, User is able to press reset email, enter in email and reset password confirmation email will be sent to user email and when pressed on link, User is directed to Password reset page where user will be able to enter in new password	1. Press Reset Password, 2. User taken to Reset password page and needs to enter in email 3. Email sent with password confirmation to User's Email 4. User presses on link 5. Takes user to Reset Password Page 6. User enter in new password and presses submit 7. User is redirected to Login Page	User's old password would be updated with new password and be directed to Login Page.	User's old password was updated with new password and be directed to Login Page.	Pass	Kevin Le
6	Email_Validation_5	SignUp_Module	P1	Verify that the email validation method accepts only properly formatted emails.	1. Input "test@example.com" into the validateEmail method. 2. Input "test@.com" into the validateEmail method. 3. Input "test@example" into the validateEmail method.	Method returns true for "test@example.com" and false for the others.	Method returned true for "test@example.com" and false for "test@.com" and "test@example".	Pass	Dominic Parker
7	Password_Encryption_6	SignUp_Module	P0	Verify the password encryption method correctly encrypts plaintext passwords.	1. Input "Password123" into the encryptPassword method. 2. Repeat with "Password123" to confirm consistent encryption.	Method returns an encrypted string not equal to the plaintext; same input results in the same encrypted output.	Encrypted strings matched and were not in plaintext.	Pass	Dominic Parker
8	Seat_Availability	MovieSelection_Module	P1	Test if the seat availability method correctly identifies if a seat is occupied or not.	1. Mark seat ID "A1" as occupied. 2. Call checkSeatAvailability for "A1". 3. Mark "A2" as available. 4. Call checkSeatAvailability for "A2".	Method returns "false" for "A1" and "true" for "A2".	Method correctly returned false for "A1" and true for "A2".	Pass	Dominic Parker
9	Add_Movie	Movie_Module	P0	Verify that the addMovie() function correctly adds a movie to the system's movie database.	1. Call addMovie() with valid movie details. 2. Query the movie database to check if the movie has been added. 3. Retrieve the movie from the database.	The movie is added successfully, and the retrieved movie details match the provided details.	1. The movie is added successfully, and the retrieved movie details match the provided details. 2. The movie isn't added successfully, no retrieved movie details.	Pass	Alex Vo
10	Purchase_Ticket	Ticket_Module	P0	Verify that the purchaseTicket() function correctly processes a ticket purchase.	1. Simulate a ticket purchase using purchaseTicket() with valid input parameters. 2. Retrieve the user's transaction history. 3. Query the database to confirm the seat is marked as booked.	A ticket is successfully purchased, and the transaction history reflects the purchase.	1. A ticket is successfully purchased, and the transaction history reflects the purchase. 2. None ticket purchased successfully, no transaction history	Pass	Alex Vo
11	Refund_Ticket	Refund_Module	P1	Verify that the refundTicket() function correctly processes a ticket refund when requested.	1. Stimulate a ticket refund using refundTicket() with valid input parameters. 2. Retrieve the user's transaction history and check seat availability. 3. Query the database to check if the refunded seat is now available.	The ticket is successfully refunded, and the transaction history and seat availability are updated accordingly.	1. The ticket is successfully refunded, and the transaction history and seat availability are updated accordingly. 2. Ticket not refunded, no transaction history and seat availability updated.	Pass	Alex Vo
12	BuyMovieTicketSystem	Buy_Movie_System	P0	Verify the process of going through to buy a movie ticket.	1.Login into your account, create account first if not yet. 2. Go to Home Page and select a movie to watch. 3. Pick Theater and Show Time and date and press next. 4. Pick what seats and press next. 5. Proceed to checkout, enter in card information, CVV, name and expiration date, and hit submit 6. Confirmation email gets sent to User and the tickets and purchase logged in Database and in User Account	The tickets were sucesfully purchased and the ticket logged in Account and in email. Tickets are for correct time and date and Theater and time	The tickets were sucesfully purchased and the ticket logged in Account and in email. Tickets are for correct time and date and Theater and time	Pass	Kevin Le
13	Refund_Ticket_System	RefundTicket_System	P0	Verify the process of refunding a movie ticket	1. Log into Account 2. Go to Account Page 3. Press Refund Ticket 4. Select what Ticket to Refund 5. Press Submit 6. Money refunded to account and Refund Ticket Confirmation sent to Email	Tickets are refunded sucessfully and database and purchase history updated to show that	Tickets were refunded sucessfully and database and purchase history updated to show that	Pass	Kevin Le

The Verification Test Plan is designed to assess the Movie Ticketing Software System against the outlined specifications in the SDS document. The plan delineates a structured approach to testing, using a blend of functional, unit, and system tests to ensure all aspects of the software function harmoniously.

Unit tests are the foundation of our testing strategy. For example, in the User Authentication module, we craft unit test cases to validate individual functions, such as verifying the encryption of passwords. Here, a test case would input a known plaintext password into the encryption function and assert that the output is a unique, non-reversible hash. These tests are executed in isolation from the system to ensure that each method performs exactly as intended.

Functional tests are designed to evaluate the system's functional requirements. Taking the Seat Selection module as a case in point, we design functional test cases that interact with the UI and back-end services to create a user's journey from selecting a movie to choosing a seat. These tests cover scenarios such as the proper display of seat availability and the correct updating of the seat map after a transaction, ensuring that the system behaves correctly in response to user inputs under varied conditions.

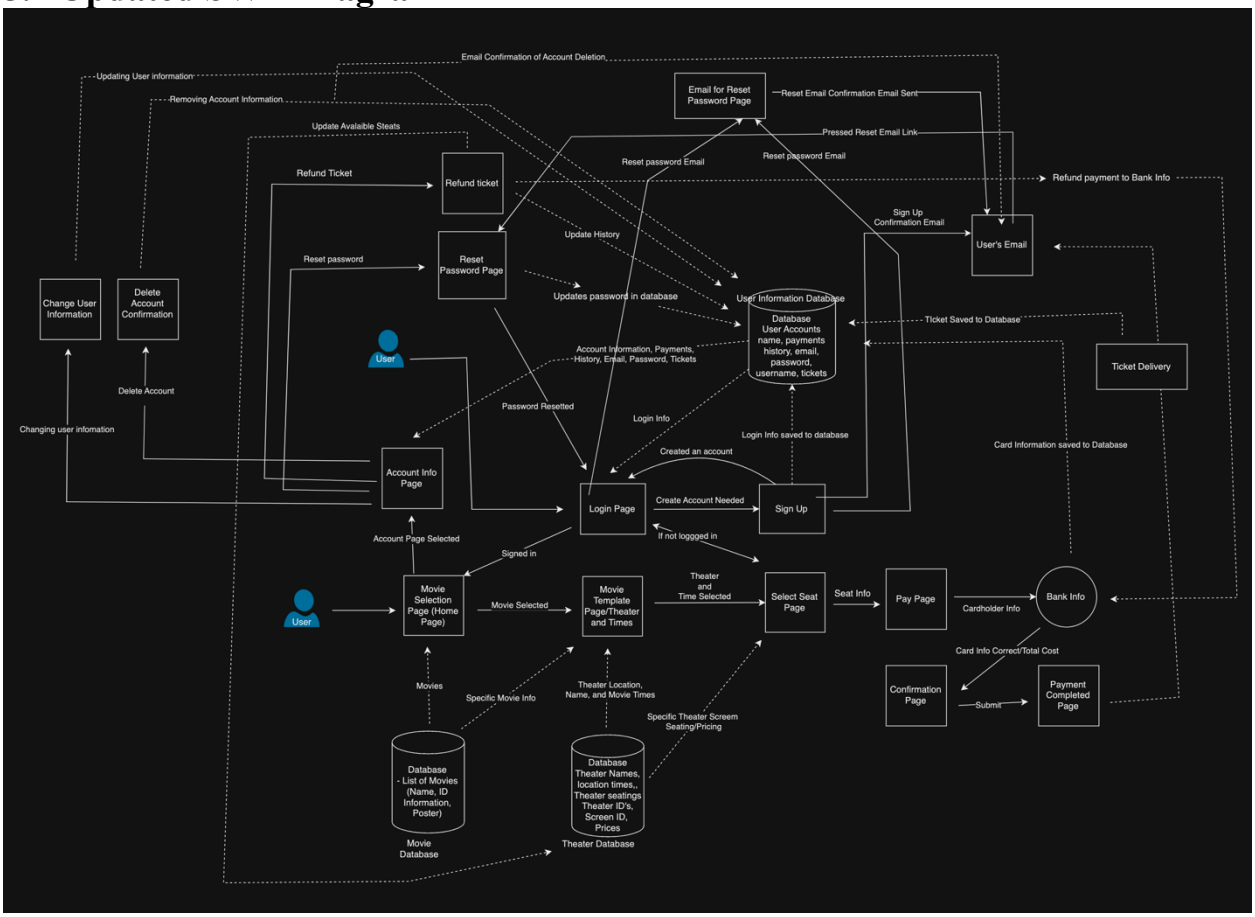
Lastly, system testing is employed to validate the fully integrated application's behavior and its compliance with the requirements. This includes end-to-end scenarios covering the entire process of booking a movie ticket, from user registration to final payment. For instance, a system test case would walk through the complete workflow to ensure that a user can navigate through the interface, select a movie, choose a seat, and complete a transaction without any system errors.

Across all three testing methodologies, we incorporate positive, negative, and boundary cases. Positive cases validate the expected behavior under normal conditions, negative cases ensure graceful handling of invalid inputs or unexpected user actions, and boundary cases test the limits of the system's capabilities. By implementing this tri-fold testing strategy, we provide thorough coverage of the software's functionality, guarantee a robust unit-level performance, and assure the system's readiness for deployment and real-world use. The results from these tests are crucial, offering detailed insights into the system's behavior and providing a clear pathway for any necessary refinements to align with the SDS.

The selected features we chose to test are the primary features needed for the application to function. For the functional test: SignUp_1, Login_2, Payment_3, and ResetPassword_4. These are the pimrary functions that are needed to bring in revenue. For the Unit testing, we Email_Validation_5, Password_Encryption_6, Seat_Availability, Add_Movie, Purchase_Ticket, and Refund_Ticket. These are the important methods that allow the components to run. For System testing: BuyMovieTicketSystem and Refund_Ticket_System. These are the overall routes that Users will use to buy their movie ticket, and allowing refunds.

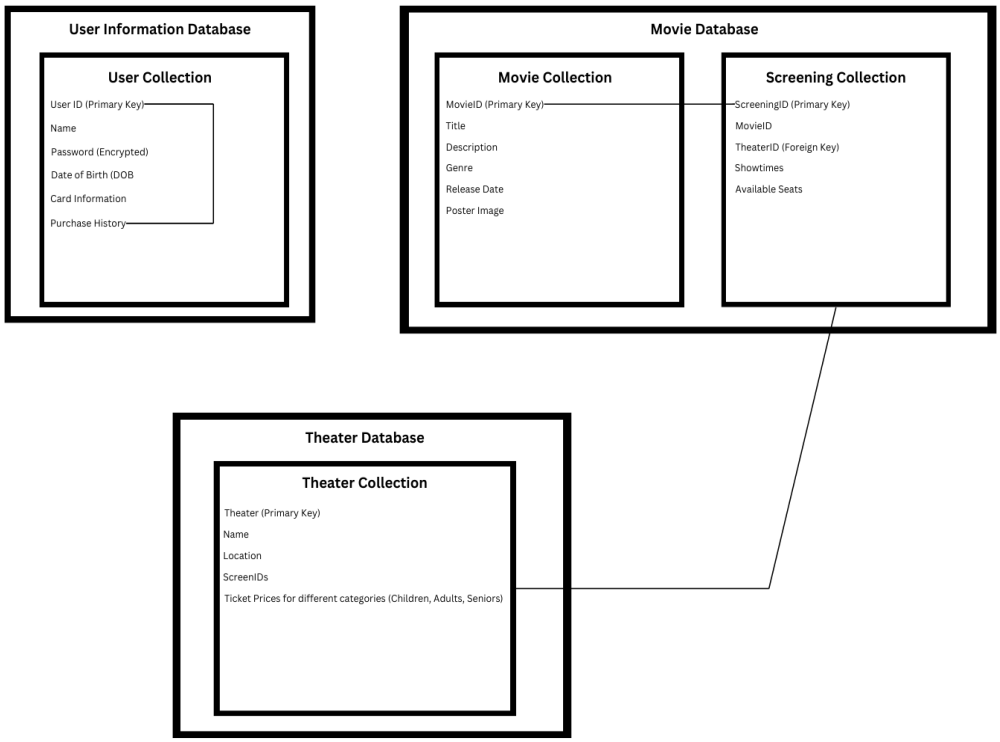
5. Change Management Process

5.1 Updated SWA Diagram



The updated SWA Diagram includes 2 new features for Users to have access to basic features for their accounts: Account Deletion and Edit User Information. Account Deletion allows the user to delete their accounts by clicking on a button to delete their account and confirm it was not an accidental. Edit User Information allows the user to edit their name, date of birth, and card information. To change their password, they must follow the route set up already to reset the password and is not included in Edit User Information for security purposes. The SWA Diagram lists the major routes and actions a User will potentially access. The initial SWA Diagram already had most of the major routes needed for it to be functional but the Updated SWA Diagram included more features for User experience.

5.2 Data Management Strategy



For managing data, we are using 3 Databases: User Information, Movie Database, and a Theater Database. They are separated in this way in order to reduce confusion when trying to take care of maintenance and prevent hackers from gaining access to all User data from one leak. Our system utilizes Relational Database Management Systems (RDBMS) to store and manage data efficiently. RDBMS excel at handling structured data like movie details, theater information, user accounts, and booking records. They enforce data integrity through constraints, ensuring accuracy and consistency in booking information. This is a needed feature as one User can be treated as an object and each User object can include information such as their name, password, email, DOB, Card Information, Purchase History, etc. It makes it convenient when needing to fix a user issue as we can access one person case by case. We can use MongoDb to host our database as it allows for easy creation a Cluster which contains multiple databases in one making it easy to use.

5.3 Trade-Offs

Managing multiple databases adds complexity compared to a single one. However, the benefits in data integrity, performance, and security outweigh this complexity for a system of this scale. When using relational SQL databases compared to NoSQL databases, SQL Databases allow for easy data deletion and clean up for chaining data. Such as an account being deleted and which other tables storing related data that share the same user ID will also be deleted as well. But in

the case of this Movie Ticketing System, Account Information being stored as documents within a database are separated by User and therefore, all information relating to a User will also be deleted in one single document. When using a NoSQL database such as MongoDB, it allows for complex data and a mix of data types to be stored, such strings, integers, arrays of strings, arrays of Objects, arrays of arrays of Objects, etc. It allows for customization to what is needed.

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

GitHub Link: https://github.com/Domm6/CosmosCompany-MovieTicketingSoftwareSystem