AMD Movies

Software Requirements Specification

4.0

June 17, 2024

Group 6

Heidi Lin, Chinmay Patel, Matias Carhuamaca

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

Revision History

Date	Description	Author	Comments
05/24/24	SRS 1.0	Heidi, Chinmay, Matias	Requirements Specification
06/03/24	SRS 2.0	Heidi, Chinmay, Matias	Software Design Specification
06/10/24	SRS 3.0	Heidi, Chinmay, Matias	SDS Test Plan
06/17/24	SRS 4.0	Heidi, Chinmay, Matias	Architectural Design with Data Management

Document Approval

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

Signature	Printed Name	Title	Date
H.L.	Heidi Lin	Software Eng.	06/17/24
C.P.	Chinmay Patel	Software Eng.	06/17/24
M.C.	Matias Carhuamaca	Software Eng.	06/17/24
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

Welcome to the Software Requirement Specification (SRS) document for AMD Movies. This document outlines the requirements for the development of a web app for AMD Movies. AMDmovies.com will be the starting point for the expansion and development of AMD Movies. This document will address the requirements for such development and improvement of the company and its stakeholders.

1.1 Purpose

This SRS serves as the foundational blueprint for AMD Movies' digital transformation, encapsulating the essential elements necessary to create a progressive web application. By delineating the functional and non-functional requirements, this document sets the stage for the seamless integration of advanced technologies and immersive content offerings both functionally and non-functionally, to pave the way for smoothly blending in the latest tech and captivating content, assisting in a new era of entertainment excellence for AMD Movies and its discerning audience.

1.2 Scope

This SRS outlines the specifications for the AMD Movies web application, encompassing user requirements, system features, functional and non-functional requirements, and other relevant considerations.

The software will:

- Provide an online platform for purchasing movie tickets both online and in person
- Support ticket sales to theaters with seat reservations for deluxe theaters
- Handle up to 10 million concurrent users.
- Offer discounts to students, military personnel, and seniors.
- Allow ticket purchases up to 2 weeks in advance and 10 minutes past showtime.
- Have an administrator account to manage showtimes and user errors.
- Display movie reviews and critic quotes from Rotten Tomatoes or IMDB.
- Support payments in credit card, Paypal, and Bitcoin.
- Securely generate unique and non-replicable tickets.
- Provide physical and digital ticket options.
- Using a captcha for the website version prevents digital login from remote locations.

The software will not:

- Allow buying tickets for theaters outside of San Diego.
- Have a mobile app version, only a web browser version.
- Implement nearby theater suggestions.

Benefits, Objectives, and Goals:

- **Scalability** - handles high volumes of concurrent users, ensuring reliability and stability.
- **Efficient** - Allows users to quickly and easily purchase tickets.
- **User-Friendly Interface** - Easy-to-use UI for customers and administrators.
- **Customization Options** - Allows for user interface customization to match brand identity and user preferences, creating a more personalized experience.
- **Security** - All transactions are secure and tickets are unique to prevent scalping and fraud.
- **Secure Transactions** - Implements robust encryption and security measures to ensure all transactions are secure, protecting user data and financial information.
- **Discount support** - Offers discounts on tickets to enhance customer experience.
- **Loyalty Programs** - Supports loyalty programs that reward frequent users with discounts and exclusive offers, encouraging repeat business.
- **Real-time data** - Shows up-to-date ticket availability and sales data with live updates.
- **Analytics Dashboard** - Provides administrators with a real-time analytics dashboard to monitor sales performance, customer behavior, and other key metrics.
- **Feedback** - Gathers customer feedback after purchase to continuously improve the system.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Specifications
- UI: User Interface
- IMDB: Internet Movie Database
- API: Application Programming Interface
- DB: Database
- DBMS: Database Management System
- AMD: Advanced Movie Destination Cinema
- MPA - Motion Picture Association

1.4 References

- IEEE Standard for Software Requirements Specifications.
- Rotten Tomatoes API Documentation.
- IMDB API Documentation.

1.5 Overview

The rest of the SRS contains detailed functional and non-functional requirements, UI specifications and implementations, system structure, design, constraints, and appendices.
The rest of the Software Requirement Specification (SRS) for the AMD Movies ticket booking app includes detailed functional and non-functional requirements, user interface specifications and implementations, system structure and design, constraints, and appendices. The software will provide an online platform for purchasing movie tickets both online and in person, facilitating

seamless ticket booking and purchase. It will support ticket sales to theaters with seat reservations, particularly for deluxe theaters, ensuring users can choose their preferred seats. The system is designed to handle up to 10 million concurrent users, ensuring scalability and performance during peak times. Additionally, the app will offer discounts to students, military personnel, and seniors, enhancing customer satisfaction and accessibility. Users will be able to purchase tickets up to 2 weeks in advance and up to 10 minutes past showtime, providing flexibility and convenience. An administrator account will be available to manage showtimes and user issues, ensuring smooth operation and quick resolution of errors. The app will also display movie reviews and critic quotes from Rotten Tomatoes or IMDB, providing users with valuable insights to make informed decisions. The payment system will support credit card, PayPal, and Bitcoin transactions, offering multiple payment options to cater to diverse user preferences. To prevent fraud, the app will securely generate unique and non-replicable tickets. Both physical and digital ticket options will be available, catering to different user needs. Additionally, the website version will use CAPTCHA to prevent unauthorized digital logins from remote locations, enhancing security. This comprehensive SRS ensures that the AMD Movies ticket booking app will be robust, user-friendly, and scalable, meeting the needs of both users and administrators while maintaining high standards of security and performance.

The SRS is organized into sections that cover all aspects of the system requirements including the introduction(purpose, scope), general description(like perspective and function of a product), specific requirements(declaring design constraints), and appendix(supporting information). These steps will ensure a comprehensive, clear, and structured presentation of the system requirements, facilitating effective communication among stakeholders, and guiding the development process well.

2. General Description

The general factors that might affect the web app for AMD Movies and its requirements include security considerations, user interface design principles, regulatory compliance, and compatibility with various devices and operating systems. While specific requirements will be detailed in subsequent sections, this section provides an overview of the broader considerations that influence the development process. By understanding these factors, stakeholders can gain insight into the overarching goals and constraints guiding the app's design and implementation.

2.1 Product Perspective

AMD Movies is a software designed to improve the current ticketing system in place at the theater chain in San Diego as it integrates with the existing system. The system will provide a combined platform for both online and in-person ticket sales with real-time synchronization of data across different access points. This will be through a web browser implementation. Key benefits include increased scalability, better user experience, and enhanced security features.

2.2 Product Functions

User functionalities:

- Account Creation
- Advance Booking
- Assigned and Open Seating availability
- Bot Protection Software
- Browser-Based Platform
- Capacity Handling
- Discount Support
- Maximum Ticket Purchase
- Nearby Theaters Option
- Online & In-person ticket booking
- Post-Purchase Feedback
- Refund Process
- Regular vs Deluxe Tickets
- Review Scraping from other websites
- Subscription Option
- Ticket Purchase Deadline
- Ticket Purchase Limits
- Timer for ticket purchasing
- Web Compatibility for all browsers (Safari, Google Chrome, Brave, Firefox, Edge, ARC)

Admin functionalities:
- Administrator Mode
- Concurrent User Capacity
- Current System Evaluation
- DB Structure
- Project Budget
- Ticket Sales Data
- Scalability Concerns
- Showtimes per theater
- Movie Review Integration
- Payment Integration

2.3 User Characteristics

The objective is to ensure that users have a convenient and seamless experience throughout the AMD Movies web app, regardless of their technological proficiency or age. Users will have specific needs when purchasing tickets online and will be concerned about their privacy and security. Therefore, AMD Movies must provide a secure and reassuring environment to ensure users' peace of mind.

Additionally, the user base may include individuals with disabilities or special needs, highlighting the necessity of accessibility and inclusivity in the web app's design. This involves offering alternative methods for inputting information, supporting assistive technologies, and adhering to accessibility standards to ensure all users can navigate the platform comfortably and efficiently. Incorporating features such as screen reader compatibility, adjustable text sizes, and voice input options can significantly enhance usability for all users.

Moreover, the web app should provide a responsive design that adapts to various devices and screen sizes, ensuring a consistent and intuitive experience whether users access the platform via smartphones, tablets, or desktops. Offering multiple language options can also cater to a diverse user base, making the app more accessible to non-English speakers.

Overall, the user experience and satisfaction are paramount. By addressing these considerations, AMD Movies can create a user-friendly, secure, and inclusive web app that meets the diverse needs of its audience and enhances their movie-going experience.

2.4 General Constraints

- Age Limit
    - AMD Movies strictly adheres to MPA standards, requiring users to purchase tickets appropriate for their age. No refunds will be issued for violations of this policy.

- Usage of Bots
    - To prevent the use of bots during the checkout process, the website will incorporate a bot detection mechanism such as BotDetect CAPTCHA, KeyCAPTCHA, TextCAPTCHA, etc.

- Ticket Limit
    - Users must log in to purchase tickets online, with a maximum limit of 20 tickets per user every 24 hours. Guest purchases are permitted, but there is a limit of 2 tickets per IP address per movie.

- Time Limit
    - Pre-ordering tickets are allowed only within two weeks of the movie’s release date. Tickets can be purchased up to 10 minutes after the movie has started.

- Refunds
    - Users can request refunds online up to 20 minutes before the movie start time. Staff can provide on-site refunds up to 10 minutes before the movie begins.

- Payment Information
  Ensuring the security of our users is a top priority. AMD Movies will implement advanced security measures to protect user privacy and information, including:
    - **Encryption**: Encrypting all sensitive data to protect against unauthorized access.
    - **Tokenization**: Replacing sensitive data with unique identifiers (tokens) for secure transactions.
    - **Two-Factor Authentication (2FA)**: Implementing 2FA to add an extra layer of security for user accounts.
    - **Fraud Detection Systems**: Utilizing advanced systems to detect and prevent fraudulent activities.
    - **Restricted Access**: Limiting access to sensitive information to authorized personnel only.
    - **Secure Development Practices**: Adopting secure coding practices to minimize vulnerabilities in the app.
    - **PCI DSS Compliance**: Adhering to Payment Card Industry Data Security Standards to ensure secure payment processing.
    - **Regular Security Audits**: Conducting regular audits to identify and address potential security vulnerabilities.

2.5 Assumptions and Dependencies

Assumptions and dependencies of AMD Movies:

- Hardware will be compatible with software requirements.
- External sources and APIs such as Rotten Tomatoes and IMDB will be available and reliable to use.
- Internet connectivity will be good enough and sufficient to support real-time updates.
- The success of AMD Movies depends on user adoption and engagement. It is assumed that users will embrace the platform and actively participate in ticket booking, movie reviews, and other interactive features, contributing to the overall success and growth of the application.
- AMD Movies relies on cooperation from vendors, partners, and third-party service providers for various aspects such as payment processing, cloud hosting, and content licensing. It is assumed that these stakeholders will cooperate effectively to ensure smooth operations and continuous improvement of the platform.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The system will offer an attractive and user-friendly web interface accessible through any web browser. It will adhere to WCAG (Web Content Accessibility Guidelines) standards, ensuring inclusivity by providing accessibility features like keyboard navigation, text alternatives, and input assistance. These measures aim to enhance the accessibility of AMD Movies for all users, making navigation smoother and more intuitive. Additionally, the interface will prioritize aesthetics and user-friendliness, contributing to an enjoyable and seamless experience for customers browsing and booking tickets on the platform.

3.1.2 Hardware Interfaces

The Software will be able to integrate with printers to print tickets or barcodes after purchasing tickets. As well as integrate with receipt printers at Movie Theater locations.

3.1.3 Software Interfaces

The software will be designed to seamlessly interact with a robust database management system (DBMS), facilitating the storage and retrieval of critical information such as ticket reservations, seat assignments, and comprehensive movie data. This integration ensures that all relevant data is efficiently managed and readily accessible whenever needed, contributing to the smooth operation of the ticket booking platform.
Furthermore, the software will incorporate integration with PayPal, a widely recognized and trusted payment gateway. This integration enables secure processing of credit and debit card transactions, whether they occur online during the ticket booking process or in person at the theater. By leveraging PayPal's advanced security features and payment processing capabilities, users can confidently complete their transactions knowing that their sensitive financial information is protected.

3.1.4 Communications Interfaces

This software must integrate with email services, which will send reservation confirmations, receipts, promotional offers, and two-factor authentication (2FA) for users and administrators.

3.2 Functional Requirements

This section describes specific features of the software project.

3.2.1 Purchasing Ticket

3.2.1.1 Introduction

This functionality allows users to purchase movie tickets online or in person at kiosks

3.2.1.2 Inputs

- User account if applicable
- User Selected theater, movie, showtime, and seat (where applicable)
- Number of tickets and type (regular or deluxe)
- Payment information and any applicable discounts
- Feedback

3.2.1.3 Processing

- Validate if seat(s) is/are available
- Apply any discounts and calculate the total cost
- Process payments securely
- Generate unique and non-replicable ticket

3.2.1.4 Outputs
- Confirmation page displaying ticket details
- Email with digital ticket(s) and receipt

3.2.1.5 Error Handling
- Display error messages for invalid payment methods or information
- Alert user if selected showtime is sold out
- Troubleshooting options for failed tasks such as transactions or showtime/seat selection.

3.2.2 Displaying Movie Showtimes and Available Tickets

3.2.2.1: The system is required to showcase movie listings, seat availability, and showtimes.

3.2.2.2: The system is encouraged to accept the chosen movie as input.

3.2.2.3: The system will refresh and update ticket information.

3.2.2.4: The system is expected to provide details on available seats and showtimes for the selected movie.

3.2.2.5: The system will inform the user if no seats are available.

3.2.3 Admin Account

3.2.3.1: The system is obligated to enable the addition of administrators and restrict certain functions to administrator-only access.

3.2.3.2: Employees must be assigned specific administrator roles.

3.2.3.3: The system is recommended to grant administrator privileges to users with designated admin access in their accounts.

3.2.3.4: A system administrator should have the capability to process ticket refunds and view the purchase history of customers.

3.2.3.5: Access to administrator functions will be denied to users attempting to access them, with an error message displayed.

3.2.4 User Account

3.2.4.1: The system is recommended to support the creation of accounts and the signing-in process.

3.2.4.2: Usernames and passwords are expected to be entered as input.

3.2.4.3: The system will verify if the login credentials match the records in the database.

3.2.4.4: Upon successful authentication, the system should guide the user to the home page.

3.2.4.5: In case of incorrect credentials, an error message should be shown to the user.

3.2.5 Display Movie Reviews

3.2.5.1: The system is expected to present audience reviews and critic quotes.

3.2.5.2: Users are required to choose a movie to view its reviews.

3.2.5.3: Reviews should be displayed in a random order, and the system should calculate the average review rating.

3.2.5.4: The system will showcase the reviews.

3.2.5.5: If no reviews or quotes are available, the system will notify the user accordingly.

3.2.6 Movie Search

3.2.6.1: Users should be able to search for movies and showtimes using the system.

3.2.6.2: Movie names will be inputted into the system.

3.2.6.3: The system will retrieve information about seat availability, showtimes, and reviews from the database.

3.2.6.4: Movie showtimes, available seats, and reviews will be displayed to the user by the system.

3.2.6.5: In case the movie does not exist, the system will inform the users that no search results were found.

3.2.7 Updating the latest movie on the website by Admin

3.2.7.1: Administrators will have the capability to generate movie objects encompassing screening times, seat availability, and movie reviews.

3.2.7.2: The system will receive inputs including the movie name, screening times and dates, available seats, and location.

3.2.7.3: It will then proceed to construct a movie object incorporating all the specified attributes from the input.

3.2.7.4: Movies will be presented to users during movie searches by the system.

3.2.7.5: If any of the specified inputs are missing during object creation, the system will notify the administrator of an error in movie creation.

3.2.8 Refunds

3.2.8.1: Users will have the capability to request refunds for their ticket purchases.

3.2.8.2: The system will require inputs such as the user account number and ticket purchase ID.

3.2.8.3: Upon receiving the inputs, the system will remove the reserved seat from the corresponding movie object and initiate the refund process, transferring the money back through the user's chosen payment method.

3.2.8.4: Users will be informed that the refund request is being processed, and they will receive a confirmation email upon completion.

3.2.8.5: In case of any errors encountered during the refund process, the system will notify the user that there was an issue and the refund was not completed.

3.2.9 Customer Feedback

3.2.9.1: The Customer Feedback system will facilitate input from patrons regarding their in-person experiences at the movie theater, covering aspects such as amenities, movie choices, and overall satisfaction.

3.2.9.2: Feedback can be submitted via kiosks located within the theater premises or through automated emails dispatched days after ticket purchases.

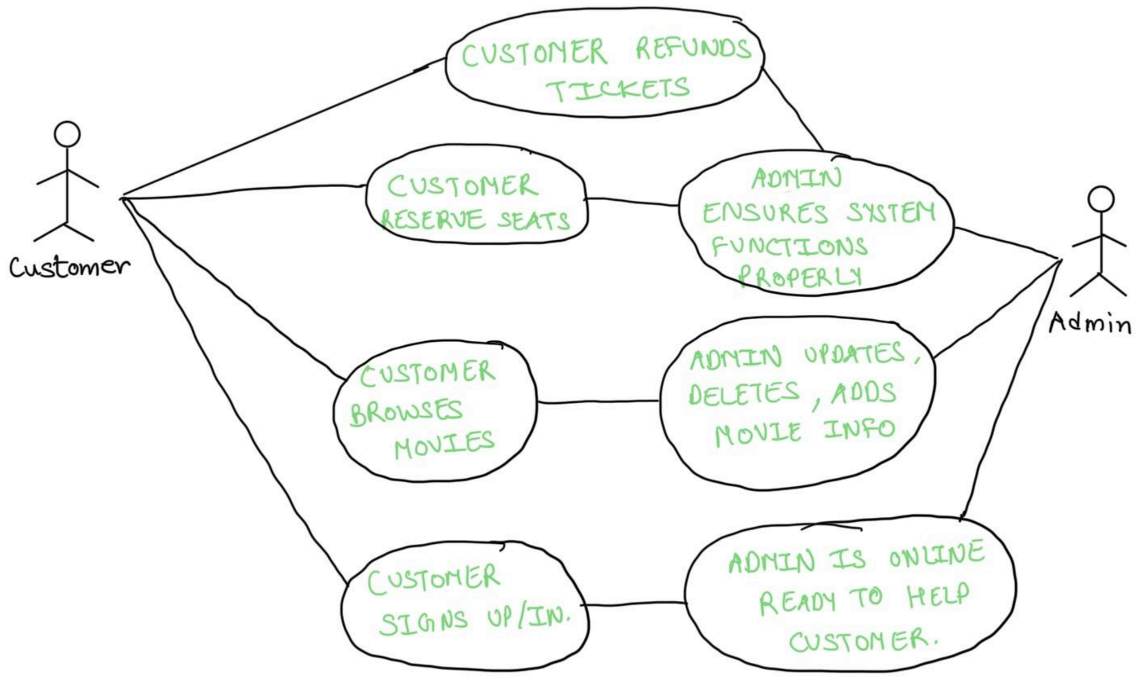
3.2.9.3: Each customer's feedback will be logged by the system, including ratings of their experiences, which will be integrated into a Customer Relationship Management system.

3.2.9.4: The system will generate reports and identify trends through the system, facilitating areas of improvement based on the feedback received.

3.2.9.5: The system will handle any processing or submission errors that occur during the feedback process and provide corresponding error messages to users.

3.3 Use Cases

3.3.1 Use Case Diagram


Customer
Admin
CUSTOMER REFUNDS TICKETS
CUSTOMER RESERVE SEATS
ADMIN ENSURES SYSTEM FUNCTIONS PROPERLY
CUSTOMER BROWSES MOVIES
ADMIN UPDATES, DELETES, ADDS MOVIE INFO
CUSTOMER SIGNS UP/IN.
ADMIN IS ONLINE READY TO HELP CUSTOMER.

3.3.2 Use Case #1 (Reserving a Seat)

Actor: Casual Movie Watching Customer.
Brief Description: A new customer searches online for nearby movie theaters and clicks on AMD Movies.
Stakeholders: AMD Movie Business Owners, Managers, Administrators.
Success Guarantee: The customer successfully purchases tickets and is satisfied.
Minimal Guarantee: The customer reaches the seat selection screen.
Precondition: The AMD Movies website is operational.
Flow of Events:

1. The customer arrives on the home screen.
2. The customer searches for their desired movie under the "Movies" tab or browses through the homepage.
3. After selecting a movie, the customer clicks the "Get Tickets" button on the movie's page.
4. The customer selects their preferred movie theater location.

5. The customer chooses their desired movie time.
6. The customer selects their desired seat.
7. The customer selects an Adult ticket.
8. The customer clicks the "Continue" button and is redirected to the checkout to enter their credit/debit card information and name.
9. Upon completion, the customer receives an email confirmation with ticket details.

Extensions:

1. Location services are disabled.
2. The checkout screen fails to load.
3. The email confirmation is not sent.
4. The website is unresponsive.
5. Any web failure occurs.

3.3.3 Use Case #2 (Refunding a Ticket)

Actor: Customer
Brief Description: A frequent customer buys tickets for a movie at the wrong time and seeks a refund.
Stakeholders: AMD Movie Business Owners, Managers, Administrators
Success Guarantee: The customer receives a full refund.
Minimal Guarantee: The customer can submit a refund request.
Precondition: A monthly member has submitted a refund request online and visits a nearby movie theater location. The administrator is logged into the web app with valid credentials.
Flow of Events:

1. The monthly member locates the "Contact Support" button on the menu inside the movie theater.
2. The monthly member shows their refund request number to the help desk.
3. The admin verifies that the refund request number is valid.
4. The customer states they would like to change the tickets to another time.
5. The admin enforces the policy that requires the customer to purchase tickets again.
6. The admin validates the member's ID and credit/debit card information to process the refund.
7. The customer receives a refund receipt.

Extensions:

1. The refund request number doesn’t appear in the data list.
2. The refund request number was never sent.
3. The receipt printer isn’t functioning.
4. The payment gateway isn’t functioning properly.

3.3.4 Use Case #3 (Updating Movie Showtimes)

Actor: Administrator at Help Desk
Brief Description: An admin updates the information on movie showtimes in the web app for all AMD Movie theaters.
Stakeholders: AMD Movie Business Owners, Managers, Administrators
Success Guarantee: The admin successfully updates the showtimes.
Minimal Guarantee: The admin can view existing showtimes.
Precondition: The admin is logged into the website with valid credentials.
Flow of Events:

1. The admin accesses the management site.
2. The admin is prompted to select a specific AMD Movie theater location.
3. The admin views existing showtimes at the chosen movie theater.
4. The admin selects a showtime.
5. The system displays information about the selected showtime.
6. The admin modifies the necessary information, such as seat availability.
7. The admin confirms the edits once completed.

Extensions:

1. The admin experiences connectivity issues.
2. The admin encounters system errors.
3. The admin adds a new showtime that conflicts with another showtime.

3.4 Classes / Objects

3.4.1 Users

3.4.1.1 Attributes for Account Creation

- Email Address
- Username
- Password

3.4.1.2 Functions for Account Creation

- Create Account
- Sign In
- Reset Password

3.4.2 Movie

3.4.2.1 Attributes for Movie Showtimes

- Movie Showtimes
- Available Seats
- Location of the Theatre

3.4.2.2 Functions for Movie Showtimes

- Modify the showtime information (availability of seats).

- Access the management site.
- Confirm the edits.
- Select a specific movie theater location.
- View existing showtimes.
- Display information about the showtime.

3.4.3 Payment

3.4.3.1 Attributes for Payment

- Movie
- Seats
- Number of tickets
- Payment

3.4.3.2 Functions for Payment

- Process the transaction.
- Inputs (movie, number of tickets, seats, time).
- Email tickets with a unique code.
- Large ticket purchase limit (maximum of 20).
- Display tickets.
- Restrict purchases to within 2 weeks of the movie start date and up to 10 minutes before the movie starts.

3.4.4 Tickets

3.4.4.1 Attributes for Ticket

- Unique ID for Refunds
- Credit/Debit Card Information
- Membership ID (If applicable)

3.4.4.2 Functions for Ticket

- Validate the refund request number.
- Validate member ID and payment information (if applicable).
- Show the refund request number.
- Issue the refund receipt through email or text.

3.4.5 Feedback

3.4.5.1 Attributes for Feedback

- Customer Feedback (input for feedback)
- Customer Ratings (out of 5)

3.4.5.2 Functions for Feedback

- Enable feedback submission for in-person experiences, covering amenities, movie selection, and satisfaction.
- Track individual customer feedback.
- Generate reports and identify trends for areas of improvement.
- Manage errors, and display error messages.

3.5 Non-Functional Requirements

This section addresses the behavioral aspects and associated expectations. It's vital to emphasize non-functional requirements, including performance, reliability, availability, security, and others. These non-functional requirements aim to improve the user experience with AMD Movies.

3.5.1 Performance

The system must be capable of handling 1000 concurrent users without a significant drop in performance. The speed of the application will vary depending on the user's hardware but should remain usable from any device. Additionally, the web application should load and be available to the user within 10 seconds on supported hardware. It should also be capable of supporting peak loads, estimated to be about 10,000 transactions per hour during peak times. Furthermore, the time to display the occupancy of the seat list should not exceed 3 seconds.

3.5.2 Reliability

The system should maintain continuous operation, except for scheduled downtime or maintenance. It should aim for a mean time between failures of at least 99.5%. In case of a system failure, the recovery time should be less than 15 minutes. Additionally, the error rate for booking must be kept below 0.01 percent, while ensuring all personal data is securely stored to prevent loss of information.

3.5.3 Availability

The system should be available all day. The time for out-of-service should not be more than 3 minutes per day. Also, the data backup in each hour.

3.5.4 Security

All sensitive data, including personal and payment information, should be encrypted. The system should also use multi-factor authentication (MFA) for all administrative access.

3.5.5 Maintainability

System architecture should be modular to facilitate easier updates and maintenance. Comprehensive documentation should be provided for users to solve basic affordable problems. Auto-test is needed to make sure the system works successfully all the time.

3.5.6 Portability

The software should be able to run on any modern web browser nowadays and be compatible with major operating systems such as Windows, macOS, and Linux.

3.6 Inverse Requirements

*State any *useful* inverse requirements.*

3.7 Design Constraints

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

3.8 Logical Database Requirements

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

3.9 Other Requirements

Catchall section for any additional requirements.

4. Analysis Models

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

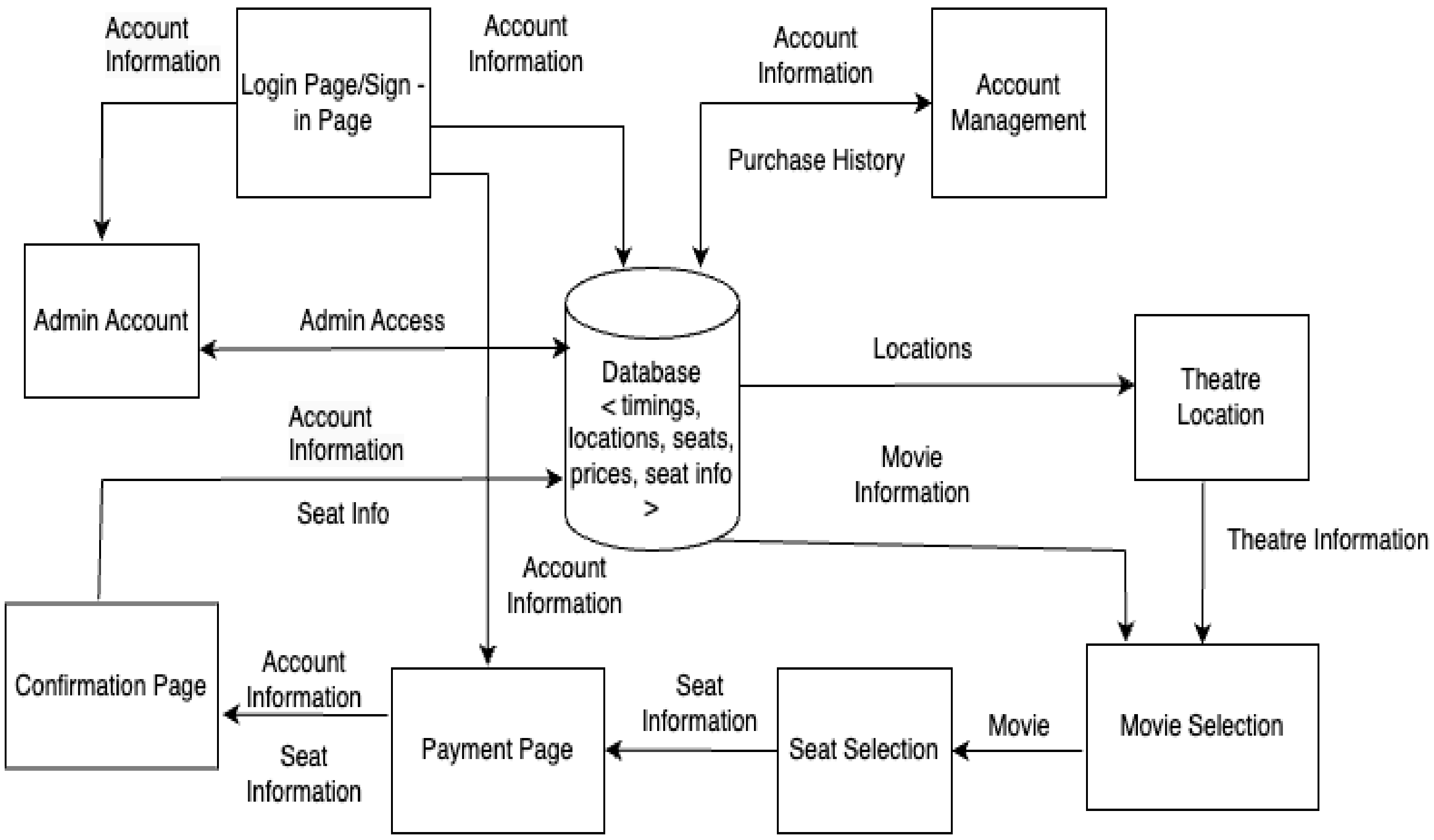
4.1 Sequence Diagrams

4.2 Data Flow Diagrams (DFD)

4.3 State-Transition Diagrams (STD)

4.4 SWA Diagram

Our Software Architecture (SWA) diagram provides a detailed visual depiction of the system's architecture, components, and interactions, aiding in a clearer comprehension of its operation and design. It fundamentally demonstrates the flow of activities among actors and external services. We've carefully designed it to reflect the functionality of this e-ticketing software accurately.



<username, password, payment info, purchase info, loyalty points, movie info, theater info> = account info

4.4.1 SWA Diagram Description

1. **Database:**

- The central system stores crucial information such as movie timings, locations, available seats, prices, and seat information. We will use NoSQL(MongoDB) in this project.

2. **Login Page/Sign-in Page:**

- Allows users to log in or sign up for an account. This interacts with the database to verify user credentials and retrieve account information (authentication).

3. **Account Management:**

- Provides functionalities for users to manage their accounts, such as updating personal information or changing passwords. This component also interacts with the database to update and retrieve user data.

4. **Theater Location:**
- Displays different theater locations available for movie showings. It fetches location data from the database.

5. **Movie Selection:**
- Allows users to browse and select movies. It retrieves movie details such as title, genre, and show timings from the database.

6. **Seat Selection:**
- It lets users choose their seats for a selected movie and timing. It checks seat availability from the database and updates the status once seats are booked.

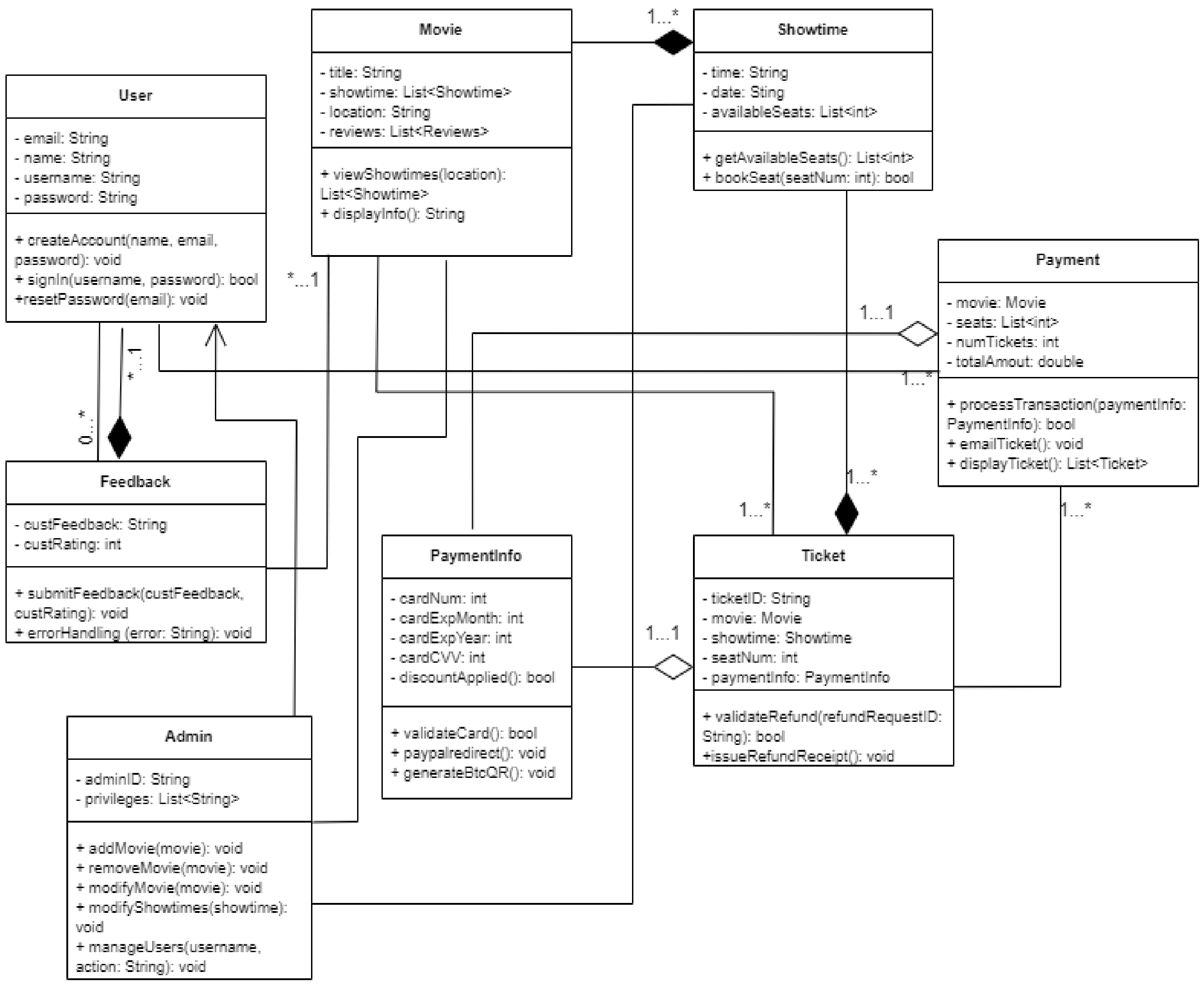
7. **Payment Page:**
- Manages the payment process for booking tickets. It interacts with the database to update booking status and payment confirmation.

8. **Confirmation Page:**
- Shows booking confirmation details to the user after a successful transaction. It retrieves booking details from the database.

9. **Admin Account:**
- Provides an interface for administrators to manage the system. This could include adding or removing movies, updating show timings, and managing theater locations. It has access to update and retrieve all relevant information in the database.

4.5 UML Diagram

Our Unified Modeling Diagram (UML) provides a detailed structural representation of the system and serves as an extensive visual overview. It highlights the classes and objects at a high level and illustrates the sequence of interactions between various components. For example, it shows a customer initiating a ticket purchase, encompassing interactions with the user interface, backend services, and external databases. Essentially, our UML diagram acts as a thorough blueprint for the software system.


Movie
- title: String
- showtime: List<Showtime>
- location: String
- reviews: List<Reviews>
+ viewShowtimes(location): List<Showtime>
+ displayInfo(): String
1...*
Showtime
- time: String
- date: Sting
- availableSeats: List<int>
+ getAvailableSeats(): List<int>
+ bookSeat(seatNum: int): bool
User
- email: String
- name: String
- username: String
- password: String
+ createAccount(name, email, password): void
+ signIn(username, password): bool
+resetPassword(email): void
*...1
Payment
- movie: Movie
- seats: List<int>
- numTickets: int
- totalAmout: double
+ processTransaction(paymentInfo: PaymentInfo): bool
+ emailTicket(): void
+ displayTicket(): List<Ticket>
1...1
1...*
1...1
*
0...*
Feedback
- custFeedback: String
- custRating: int
+ submitFeedback(custFeedback, custRating): void
+ errorHandling (error: String): void
1...*
1...*
1...*
PaymentInfo
- cardNum: int
- cardExpMonth: int
- cardExpYear: int
- cardCVV: int
- discountApplied(): bool
+ validateCard(): bool
+ paypalredirect(): void
+ generateBtcQR(): void
1...1
Ticket
- ticketID: String
- movie: Movie
- showtime: Showtime
- seatNum: int
- paymentInfo: PaymentInfo
+ validateRefund(refundRequestID: String): bool
+issueRefundReceipt(): void
Admin
- adminID: String
- privileges: List<String>
+ addMovie(movie): void
+ removeMovie(movie): void
+ modifyMovie(movie): void
+ modifyShowtimes(showtime): void
+ manageUsers(username, action: String): void

4.5.1 UML Diagram Description

1. User Class

The user class is the primary entity representing the system's users. It handles account-related functionalities, ensuring users can create, sign in, and manage their accounts securely.

- Attributes:

- email: String - Stores the user's email address.

- name: String - Stores the user's full name.
- username: String - Stores the unique username chosen by the user.
- password: String - Stores the user's password for authentication.

- Operations:
  - createAccount(name, email, password): void - Creates a new user account with the provided name, email, and password.
  - signIn(username, password): bool - Authenticates the user using the provided username and password, returning true if successful.
  - resetPassword(email): void - Initiates the process to reset the user's password using their email address.

2. Admin Class

Admin class represents a user with administrative privileges who can manage movies, showtimes, and other users within the movie booking system. This class inherits from the User class, which means it includes all the attributes and methods of a regular user but with additional capabilities specific to administrative tasks.

- Attributes:
  - adminID: String - Stores the unique identifier for the admin user.
  - operations (add, create, delete): List - Stores a list of operations (add, create, delete) that the admin can perform.
- Operations:
  - addMovie(movie): void - Adds a new movie to the system.
  - removeMovie(movie): void - Removes an existing movie from the system.
  - modifyMovie(movie): void - Modifies the details of an existing movie in the system.
  - addShowtimes(showtime): void - Adds new showtimes for movies.
  - removeShowtimes(showtime): void - Removes existing showtimes from the schedule.
  - modifyShowtimes(showtime): void - Modifies the details of existing showtimes.
  - manageUsers(username, action): void - Manages user accounts by performing actions such as adding, updating, or deleting users based on the specified action.

3. Movie Class

Movie class encapsulates the details of a movie, including its showtimes and location. It allows the viewing of showtime details.

- Attributes:
  - title: String - Stores the title of the movie.
  - showtime: List - Stores a list of showtimes for the movie.
  - location: String - Stores the location where the movie is being shown.
  - reviews: List - Stores a list of reviews for the movie.
- Operations:
  - viewShowtimes(location): List - Returns a list of showtimes for the movie at the specified location.
  - displayInfo(): String - Returns a string containing detailed information about the movie.

4. Showtime Class

Showtime class represents the individual showtimes for movies, including time and seat availability.

- Attributes:
  - time: String - Stores the time when the showtime is scheduled.
  - date: String - Stores the date of the showtime.
  - availableSeats: List - Stores a list of seats that are available for the showtime.
- Operations:
  - getAvailableSeats(): List - Retrieves a list of currently available seats for the showtime.
  - bookSeat(seatNum): bool - Attempts to book a specific seat number for the showtime, returning true if successful and false if not.

5. Payment Class

Payment class handles payment transactions, associating payments with users and tickets. It manages transactions, ensuring secure processing of payments and email notifications for ticket purchases.

- Attributes:
    - movie: Movie - Stores the movie associated with the payment.
    - seats: List - Stores a list of seats that have been selected for the transaction.
    - numTickets: int - Stores the number of tickets being purchased.
    - totalAmount: double - Stores the total amount to be paid for the tickets.

- Operations:
    - processTransaction(paymentInfo): bool - Processes the payment using the provided payment information, returning true if the transaction is successful.
    - emailTicket(): void - Sends the purchased tickets to the user's email address.
    - displayTicket(): List - Returns a list containing the details of the purchased tickets for display purposes.

6. Ticket Class

Ticket class represents individual tickets purchased by users, including movie, showtime, seat number, and purchase details. It also includes the ability to validate refund requests and issue receipts.

- Attributes:
    - ticketID: String - Stores the unique identifier for the ticket.
    - movie: Movie - Stores the movie associated with the ticket.
    - showtime: Showtime - Stores the showtime associated with the ticket.
    - seatNum: int - Stores the seat number assigned to the ticket.
    - paymentInfo: PaymentInfo - Stores the payment information related to the ticket purchase.

- Operations:
    - validateRefund(refundRequestID): String - Validates the refund request using the provided refund request ID and returns a status message.
    - issueRefundReceipt(): void - Issues a receipt for the refund if the refund is validated and processed.

7. PaymentInfo Class

PaymentInfo class contains information required to process a payment for movie tickets. This class ensures that the necessary payment details are captured and validated for transactions.

- Attributes:

  - cardNum: int - Stores the credit card number.

  - cardExpMonth: int - Stores the expiration month of the credit card.

  - cardExpYear: int - Stores the expiration year of the credit card.

  - cardCVV: int - Stores the CVV (Card Verification Value) code of the credit card.

  - discountApplied: bool - Indicates whether a discount has been applied to the transaction.

- Operations:

  - validateCard(): bool - Validates the credit card details, returning true if the card is valid and false otherwise.

8. Feedback Class

Feedback class collects and processes customer feedback, allowing users to submit ratings and comments. It manages customer feedback, enabling submission, tracking, reporting, and error handling for feedback processes.

- Attributes:

  - review: String - Stores the text of the customer review.

  - cusRating: int - Stores the customer's rating, typically on a scale (e.g., 1-5).

- Operations:

  - submitFeedback(cusFeedback, cusRating): void - Submits the customer's feedback and rating to the system.

- errorHandling(error: String): void - Handles errors related to feedback submission by taking appropriate actions based on the error message provided.

4.6 Development Plan and timeline

For Section 4, Chinmay Patel is in charge of the SWA Diagram. Matias Carhuamaca is in charge of the UML Diagram. Heidi Lin is in charge of the descriptions of classes, functions, and attributes. Overall, each member is responsible for ensuring work is complete and correct, ensuring work correlates and aligns with software system design. Each is held accountable for the work done. Began on 31 May 2024. Held meeting on 2 June 2024. Finished section 4 on 3 June 2024.

- **Team member responsibilities:**

  Matias Carhuamaca - UML Diagram.

  Chinmay Patel - SWA Diagram.

  Heidi Lin - Description of both diagrams.

4.7 SDS Test Plan

1. **Introduction**
   The purpose of this test plan is to define the strategy and approach to be taken for verifying and validating the movie ticketing system. This includes the methods, types of tests, and the specific test cases to be executed to ensure the system meets its functional and non-functional requirements.

2. **Scope**
   The scope of testing covers functional requirements, unit tests, and system tests of the movie ticketing system. This includes the user interface, ticket booking process, payment processing, and backend functionalities.

3. **Objectives**
   - Ensure all functional requirements are met.
   - Verify the integration of different modules.
   - Validate the user experience.
   - Ensure data integrity and security.
   - Ensure the system's performance and scalability.

4. **Test Strategy**
   The test strategy includes both verification and validation processes:

   Verification involves:
   - Reviews and inspections of requirements, design, and code.
   - Static analysis and walkthroughs.

   Validation involves:
   - Dynamic testing includes functional, unit, integration, system, and acceptance testing.
   - Performance and load testing.

5. **Test Environment**
   - **Hardware:** Standard user devices (PCs, smartphones, tablets), servers for backend processing.
   - **Software:** Operating systems (macOS, Windows, Linux, IOS, Android), web browsers (Chrome, Firefox, Safari), testing units (JUnit, Selenium, LoadRunner).

6. **Functional Requirements**
   Functional requirements to be tested include:
   - User registration and login.
   - Movie Listing.
   - Seat Selection.
   - Ticket Booking.
   - Payment Processing.
   - Confirmation and Notification.

7. **Test Cases**

7.1 Functional Test Cases

User Registration and Login

- **TC01**: Verify user registration with valid data.
  Description: This test case ensures that a new user can register successfully with valid data.
  Pre-requisites: The user is on the registration page.
  Steps:
  1. Enter valid data in all required fields (e.g., username, password, email).
  2. Click on the "Register" button.

Expected Result: The user should be successfully registered and redirected to the login page.

- **TC02**: Verify user login with invalid credentials.
  Description: This test case verifies that the system correctly handles login attempts with invalid credentials.
  Pre-requisites: The user is on the login page.
  Steps:
  1. Enter an invalid username and/or password.
  2. Click on the "Login" button.

  Expected Result: The user should receive an error message indicating invalid credentials.

- **TC03**: Verify password reset functionality.
  Description: This test case ensures that users can reset their passwords using the "Forgot Password" functionality.
  Pre-requisites: The user is on the login page.
  Steps:
  1. Click on the "Forgot Password" link.
  2. Enter the registered email address.
  3. Click on the "Reset Password" button.
  4. Follow the instructions received in the email to reset the password.

  Expected Result: The user should receive an error message indicating invalid credentials.

Movie Listing

- **TC04**: Verify filtering movies by genre, date, and rating.
  Description: This test case checks that users can filter movie listings based on genre, date, and rating.
  Pre-requisites: The user is on the movie listing page.
  Steps:
  1. Select a genre from the genre filter dropdown.
  2. Select a date from the date filter.
  3. Select a rating from the rating filter.

  Expected Result: Movie listings should be filtered according to the selected criteria.

Seat Selection

- **TC05**: Verify availability of seats.
  Description: This test case ensures that users can see the availability of seats for a selected movie and showtime.
  Pre-requisites: The user is on the seat selection page for a chosen movie and showtime.

Steps:
1. View the seat map for the selected showtime.

Expected Result: Available seats should be clearly displayed, and booked seats should be marked as unavailable.

Ticket Booking

- **TC06**: Verify booking of selected seats.
  Description: This test case verifies that users can book the seats they have selected.
  Pre-requisites: The user has selected seats on the seat selection page.
  Steps:
  1. Proceed to the booking page.
  2. Confirm the booking.

  Expected Result: Selected seats should be booked, and the user should be redirected to the payment page.

- **TC07**: Verify booking confirmation message.
  Description: This test case ensures that users receive a confirmation message after completing the booking and payment process.
  Pre-requisites: The user has completed the seat booking and payment process.
  Steps:
  1. Complete the payment process.

  Expected Result: The user should receive a booking confirmation message on the screen and via email.

Payment Processing

- **TC08**: Verify successful payment processing.
  Description: This test case verifies that payments are processed successfully with valid payment details.
  Pre-requisites: The user is on the payment page.
  Steps:
  1. Enter valid payment details.
  2. Confirm the payment.

  Expected Result: Payment should be processed successfully, and the user should receive a payment confirmation.

- **TC09**: Verify the refund process.
  Description: This test case ensures that users can initiate and receive refunds for their bookings.
  Pre-requisites: The user has a confirmed booking.
  Steps:
  1. Request a refund for the booking.

Expected Result: The refund process should be initiated, and the user should receive a confirmation of the refund request.

Confirmation and Notification

- **TC10**: Verify the display of booking details in the user account.
  Description: This test case checks that users can view their booking details in their account.
  Pre-requisites: The user is logged in and has a booking.
  Steps:
    1. Navigate to the "My Bookings" section of the user account.

  Expected Result: Booking details should be displayed correctly.

7.2 Unit Test Cases

- **TC11**: Verify function for calculating the total price.
  Description: This test case verifies the correctness of the function that calculates the total price based on selected seats and price per seat.
  Function: calculateTotalPrice(seats, pricePerSeat)
  Test-data:
  1. seats = [1, 2, 3], pricePerSeat = 10
  Expected Result: Total price should be ‘30’.

- **TC12**: Verify function for generating booking reference numbers.
  Description: This test case ensures that the function for generating booking reference numbers returns unique values.
  Function: generateBookingRef()
  Expected Result: The function should return a unique booking reference number each time it is called.

7.3 System Test Cases

- **TC13**: Verify integration between user registration and login modules.
  Description: This test case verifies the seamless integration between the user registration and login modules.
  Pre-requisites: No pre-requisites.
  Steps:
  1. Register a new user.
  2. Log in with the registered credentials.
  Expected Result: The user should be able to register and then log in successfully with the registered credentials.

- **TC14**: Verify load handling for concurrent users.
  Description: This test case checks the system's ability to handle a large number of concurrent users without performance degradation.
  Pre-requisites: The system is ready for load testing.
  Steps:
  1. Simulate concurrent logins and bookings by multiple users
  Expected Result: The system should handle concurrent users without crashing or significant performance degradation.

8. Verification and Validation Processes

- **Verification:** Conduct code reviews, walkthroughs, and inspections regularly during the development process.
- **Validation:** Perform functional, integration, system, and acceptance testing to validate the system against requirements.

9. Entry and Exit Criteria

Entry Criteria:

- All environments are documented and approved.
- The test environment is set up.
- Test data is prepared.
- Test cases are reviewed and approved.

Exit Criteria:

- All planned test cases are executed.
- All critical and high-priority defects are resolved.
- Test summary report is prepared.
- Acceptance criteria are met.

10. Deliverables

- Test Plan document.
- Test Cases.
- Test Scripts.
- Test Execution Reports.
- Defect reports.
- Test summary report.

11. Schedule

- Test planning: 1 week.

- Test case development: 2 weeks.
- Test environment setup: 1 week.
- Test execution: 4 weeks.
- Defect fixing and retesting: 2 weeks.
- Test closure: 1 week.

12. Responsibilities

- **Test Manager:** Overall test planning and coordination.
- **Test Engineers:** Test case design, execution, and defect reporting.
- **Developers:** Fixing reported defects.
- **Business Analysts:** Reviewing test cases and ensuring requirements are met.

13. Risk Management

- **Risk:** Delays in environment setup.
  - **Mitigation:** Plan and allocate buffer time.
- **Risk:** Incomplete requirements.
  - **Mitigation:** Continuous communication with stakeholders for clarification.
- **Risk:** High defect rates.
  - **Mitigation:** Regular code reviews and early testing.

14. Approval

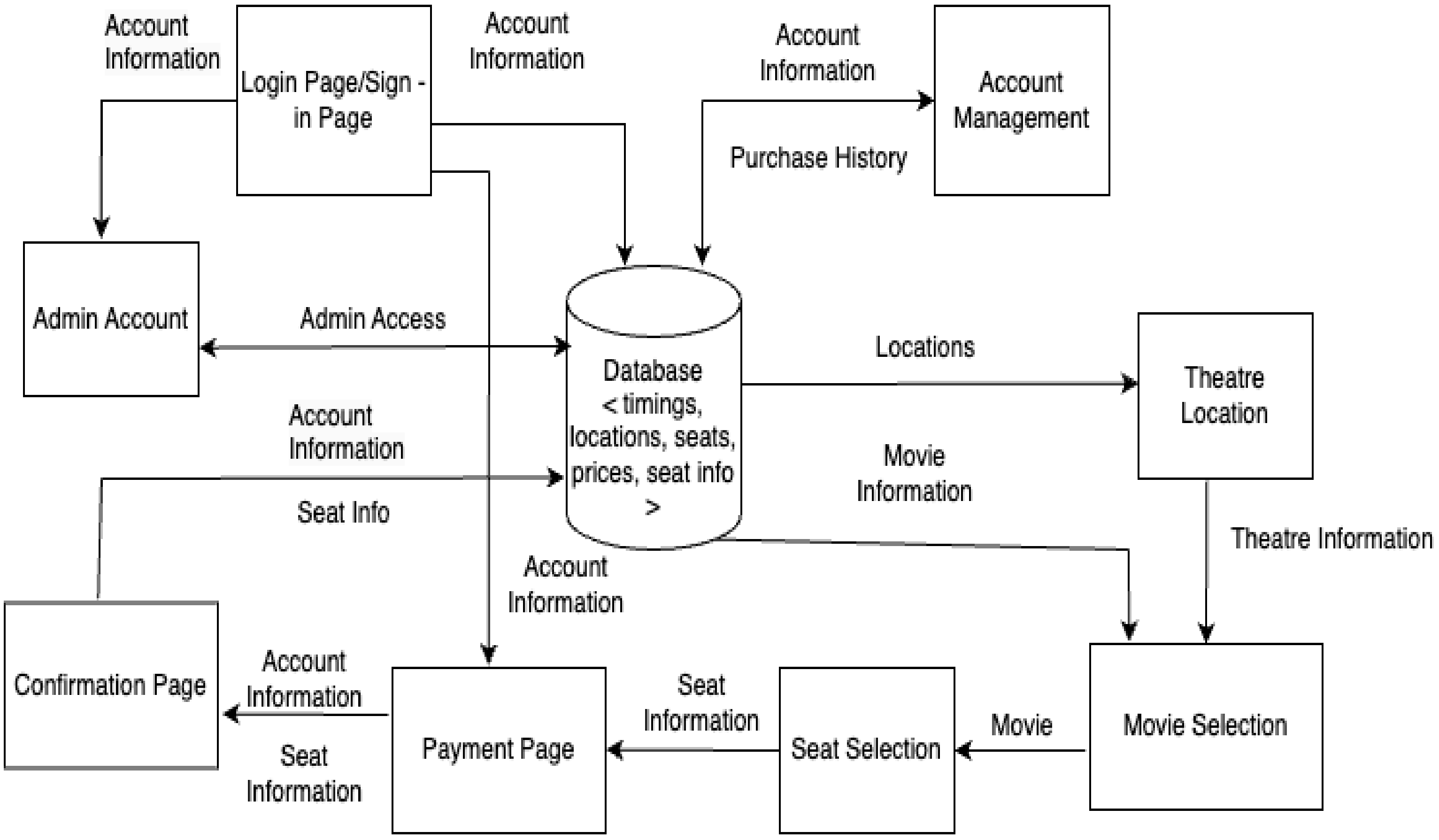
This test plan is approved by the project stakeholders, including the project manager, test manager, and business analyst.
This test plan ensures comprehensive coverage of this movie ticketing system's requirements and functionalities, providing a structured approach to testing and quality assurance.

15. GitHub Link

Heidi - https://github.com/LinHeidi/SRS-project.git
Chinmay - https://github.com/Chinupatel234/CS250
Matias - https://github.com/Matias2003c/CS250.git4

4.8 Architecture Design

4.8.1 SWA Diagram



4.8.1.1 SWA Diagram Description

1. **Database:**
- The central system stores crucial information such as movie timings, locations, available seats, prices, and seat information. We will use NoSQL(MongoDB) in this project.

2. **Login Page/Sign-in Page:**
- Allows users to log in or sign up for an account. This interacts with the database to verify user credentials and retrieve account information (authentication).

3. **Account Management:**
- Provides functionalities for users to manage their accounts, such as updating personal information or changing passwords. This component also interacts with the database to update and retrieve user data.

4. **Theater Location:**
- Displays different theater locations available for movie showings. It fetches location data from the database.

5. **Movie Selection:**
- Allows users to browse and select movies. It retrieves movie details such as title, genre, and show timings from the database.

6. **Seat Selection:**
- It lets users choose their seats for a selected movie and timing. It checks seat availability from the database and updates the status once seats are booked.

7. **Payment Page:**
- Manages the payment process for booking tickets. It interacts with the database to update booking status and payment confirmation.

8. **Confirmation Page:**
- Shows booking confirmation details to the user after a successful transaction. It retrieves booking details from the database.

9. **Admin Account:**
- Provides an interface for administrators to manage the system. This could include adding or removing movies, updating show timings, and managing theater locations. It has access to update and retrieve all relevant information in the database.

4.8.2 Data Management Strategy

4.8.2.1 Entities

1. **User**: Users of the system.

- Attributes: userID, username, password, email, roleID.

2. **Role**: Represents different users (admin, customer, etc).
- Attributes: roleID, roleName.

3. **Movie**: Represents the latest movie and available movies.
- Attributes: movieID, genre, title, duration.

4. **Theatre**: Represents the nearby theatres where movies are shown.
- Attributes: theatreID, numberOfScreens, location.

5. **Showtime**: Schedule for movies.
- Attributes: showtimeID, movieID, theatreID, startTime, endTime.

6. **Seat**: Available seats in the theatres.
- Attributes: seatID, theatreID, isAvailable, seatNumber.

7. **Price**: Represents the pricing for the seats.
- Attributes: priceID, theatreID, seatID, price.

8. **Booking**: Bookings made by users.
- Attributes: bookingID, showtimeID, seatID, status, bookingTime.

9. **Payment**: Represents the payment details for bookings.
- Attributes: paymentID, bookingID, paymentMethod, paymentStatus, amount.

4.8.2.2 Relationships

1. **User to Role**: One-to-Many (Only one role for a user, but each role can be assigned to many users).

2. **Showtime to Movie**: Many-to-One (One movie can have multiple showtimes).

3. **Showtime to Theatre**: Many-to-One (One theatre can have multiple showtimes).

4. **Seat to Theatre**: Many-to-One (One theatre has many seats).

5. **Price to Theatre**: Many-to-One (Many prices for different seats in one theatre).

6. **Price to Seat**: One-to-One (Every seat has a unique price).

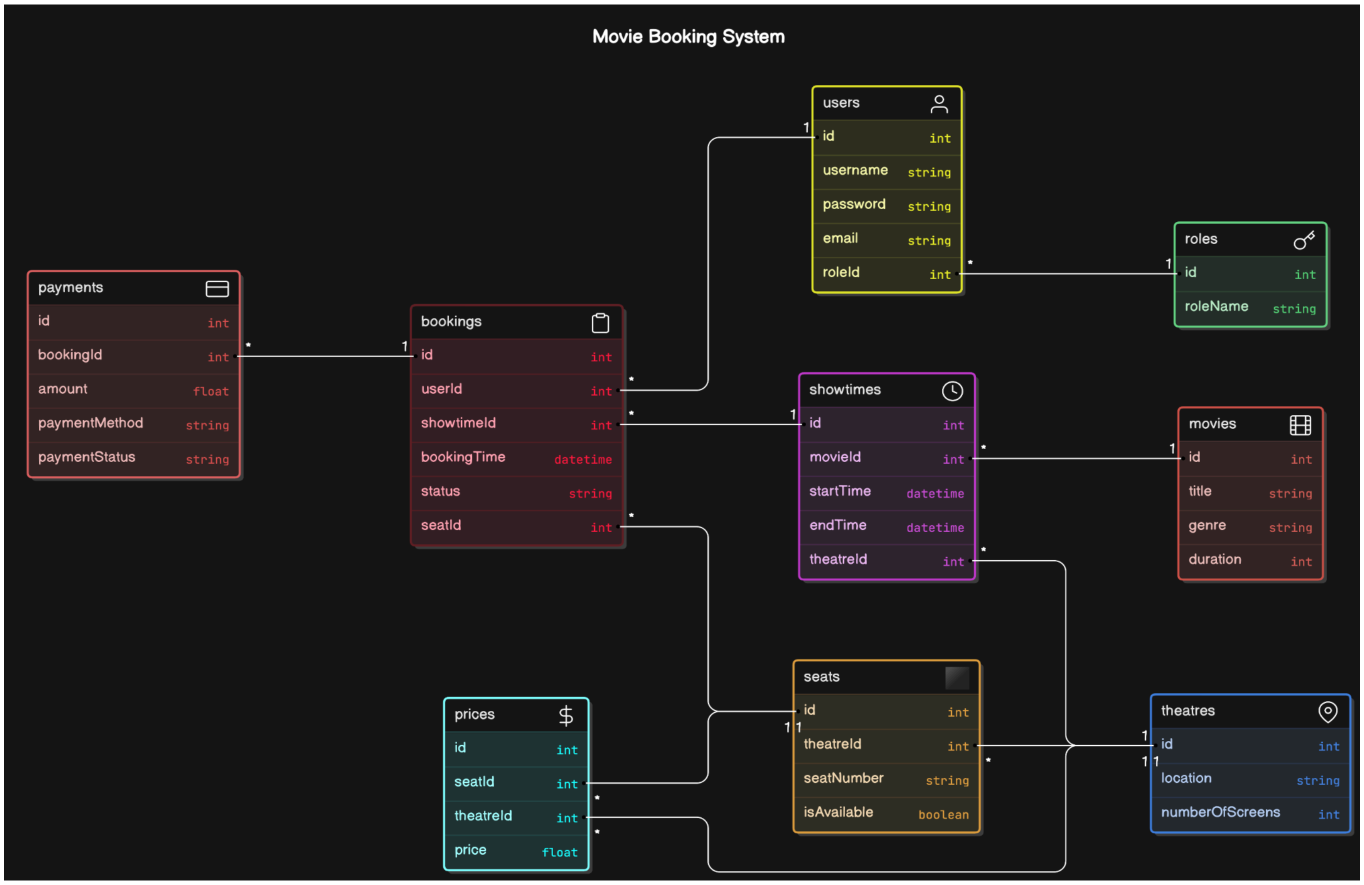
7. **Booking to User**: Many-to-One (User can book multiple seats).

8. **Booking to Showtime**: Many-to-One (Many bookings for one showtime).

9. **Booking to Seat**: Many-to-One (Many bookings for one seat).

10. **Payment to Booking**: One-to-One (Every booking has one payment).

4.8.2.3 Diagram



4.8.3 Alternatives and Tradeoffs

SQL vs NoSQL:

- **SQL**: Perfect for relational data, it guarantees data integrity and facilitates sophisticated queries. Trade-off: Some unstructured data, might be less scalable and flexible.
- **NoSQL**: Excellent flexibility and scalability, ideal for unstructured data. Trade-off: Consistency and sophisticated querying features may be compromised.

Single vs Multiple Databases:

- **Single Database**: Keeps consistency and streamlines management. Trade-off: As the application grows, it might turn into an issue.
- **Multiple Databases**: Increases scalability and performance. Trade-off: More difficult data synchronization and management.

4.8.3 Github Links

Heidi - https://github.com/LinHeidi/SRS-project.git
Chinmay - https://github.com/Chinupatel234/CS250
Matias - https://github.com/Matias2003c/CS250.git

5. Change Management Process

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

A. Appendices

Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.
Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.

A.1 Appendix 1

A.2 Appendix