Cinemate

Project Report

3/27/24

Bryce Rambach

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2023

Design Documents

Software Requirements Specification

Cinemate

Software Requirements Specification

1.5

4/17/24

Group #15
Bryce Rambach, Nikoli Oudo, Atah Habibi

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Fall 2023

Revision History

Date	Description	Author	Comments
2/14/24	Version 1.0	Bryce Rambach Nikoli Oudo Atah Habibi	First draft - 3.6
2/28/24	Version 1.1	Bryce Rambach Nikoli Oudo Atah Habibi	3.6 - 4.3
3/13/24	Version 1.2	Bryce Rambach Nikoli Oudo Atah Habibi	UML changes
3/27/24	Version 1.3	Bryce Rambach Nikoli Oudo Atah Habibi	Added 4.4 to data management
4/17/24	Version 1.4	Bryce Rambach Nikoli Oudo Atah Habibi	Added user manual
5/1/24	Version 1.5	Bryce Rambach Nikoli Oudo Atah Habibi	Added summary, conclusion, life cycle model, made flow of document much better

Document Approval

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

Signature	Printed Name	Title	Date
	<Your Name>	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

Revision History **2**
Document Approval **2**
1. Introduction **1**
1.1 Purpose 1
1.2 Scope 1
1.3 Definitions, Acronyms, and Abbreviations 1
1.4 References 1
1.5 Overview 1
2.1 Product Perspective 2
2.2 Product Functions 2
2.3 User Characteristics 2
2.4 General Constraints 2
2.5 Assumptions and Dependencies 2
3. Specific Requirements **3**
3.1 External Interface Requirements 3
3.1.1 User Interfaces 3
3.1.2 Hardware Interfaces 3
3.1.3 Software Interfaces 3
3.1.4 Communications Interfaces 3
3.2 Functional Requirements 3
3.2.1 Purchasing a Ticket 3
3.2.2 Creating an Account 4
3.3 Use Cases 4
3.3.1 Use Case #1 - Purchasing a Ticket 4
3.3.2 Use Case #2 - Creating an Account 5
3.3.3 Use Case #3 - Checking Available Seats 5
3.4 Classes / Objects 5
3.4.1 User Account 5
3.4.2 Theater 6
3.5 Non-Functional Requirements 6
3.5.1 Performance 6
3.5.2 Reliability 6
3.5.3 Availability 6
3.5.4 Security 6
3.5.5 Maintainability 7
3.5.6 Portability 7
3.6 Inverse Requirements 7
3.7 Design Constraints 7
3.8 Logical Database Requirements 7

3.9 Other Requirements..... 7
4. Analysis Models..... 8
4.1 Sequence Diagram..... 8
4.2 Software Architecture Diagram (SWA)..... 9
4.3 UML Class Diagram..... 12
4.4 Data Management..... 14
A. Appendices..... 16
A.1 User Manual..... 16

1. Introduction

Theater ticketing systems play a vital role in the movie industry by streamlining the process of selling and distributing tickets for live performances. They enhance the experience for both customers and theater owners by providing a centralized platform for ticket purchases and management. Movie Theater Ticketing System is a web-based application designed to meet this need, offering a user-friendly interface and a variety of features.

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to provide a comprehensive overview of the Movie Theater Ticketing System, outlining its features, functionalities, and constraints. The intended audience includes stakeholders such as developers, designers, project managers, and quality assurance teams involved in the development and testing of the software.

1.2 Scope

Software Product: Movie Theater Ticketing System
Description: The software product will facilitate the online purchase of movie tickets, providing users with a user-friendly interface to search for movies, select seats, and make secure payments. It will not handle physical ticket distribution or on-site box office operations.
Application: The Movie Theater Ticketing System aims to streamline the ticketing process for customers and theater owners, enhancing the overall movie-going experience. It will provide benefits such as centralized ticket management, multilingual support, and secure payment processing.

1.3 Definitions, Acronyms, and Abbreviations

Refer to Appendix A for definitions of terms, acronyms, and abbreviations used throughout the SRS.

1.4 References

Software Requirements Specification (SRS)
Title: Cinemate Movie Theater Ticketing System - Software Requirements Specification
Authors: Bryce Rambach, Nikoli Oudo, Atah Habibi
Version: 1.5
Date: 5/1/24
Course: CS 250 - Introduction to Software Systems
Institution: San Diego State University
Source: Appendix A of the Cinemate SRS document.

Unified Modeling Language (UML) Specification
Title: Unified Modeling Language (UML) User Guide
Author: Grady Booch, James Rumbaugh, Ivar Jacobson
Edition: 2nd Edition
Publisher: Addison-Wesley
Reference for: Use Case Diagrams, Class Diagrams, and Sequence Diagrams used in Cinemate's design phase.

Web Accessibility Guidelines
Title: Web Content Accessibility Guidelines (WCAG) 2.1
Publisher: World Wide Web Consortium (W3C)
Date: June 2018
Reference for: Accessibility standards applied to ensure the interface is accessible to users with disabilities.

PCI DSS Standards for Payment Security
Title: Payment Card Industry Data Security Standard (PCI DSS) Requirements and Security Assessment Procedures
Publisher: PCI Security Standards Council
Version: 3.2.1
Date: May 2018
Reference for: Secure handling of payment information in the ticketing system.

1.5 Overview

This SRS contains detailed information about the Movie Theater Ticketing System, organized into sections such as General Description, Specific Requirements, and Verification Test Plan. The document is structured to provide clarity and accessibility for all stakeholders involved in the software development process.

2. General Description

2.1 Product Perspective

The Movie Theater Ticketing System exists within the context of the broader movie industry and its related software applications. While it operates independently as a web-based ticketing platform, it may interface with other systems such as movie databases, payment gateways, and customer relationship management tools.

2.2 Product Functions

The primary functions of the Movie Theater Ticketing System include:

- Facilitating online ticket purchases for movie showings.
- Providing users with a searchable database of movies and showtimes.
- Allowing users to select seats and make reservations.
- Processing payments securely through various payment methods.
- Managing user accounts, including purchase history and loyalty points.
- Generating and issuing electronic tickets to users upon successful purchase.

2.3 User Characteristics

The typical users of the Movie Theater Ticketing System include:

- Moviegoers: Individuals who wish to purchase tickets for movie showings online.
- Theater Owners/Managers: Administrators responsible for managing theater operations and ticket sales.

- Customer Support Staff: Personnel tasked with assisting users and resolving issues related to the ticketing system.

2.4 General Constraints

Constraints affecting the development of the Movie Theater Ticketing System may include:

- Compliance with industry standards and regulations governing online ticketing and data security.
- Integration with existing theater management systems and databases.
- Consideration of user accessibility requirements, such as support for screen readers and keyboard navigation.
- Adherence to budget and resource limitations for software development and maintenance.

2.5 Assumptions and Dependencies

The development and functionality of the Movie Theater Ticketing System are dependent on various factors, including:

- Availability of stable internet connectivity for users accessing the system.
- Compatibility with commonly used web browsers and operating systems.
- Availability of movie listings and showtime data from external sources.
- Dependence on third-party payment processors for transaction processing and security.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface shall be intuitive and user-friendly, featuring:

- A search bar for users to search for movies by title.
- Clear navigation menus for accessing different sections of the application.
- Visual representations of available seats during seat selection.
- Input fields for users to enter payment information securely.
- Confirmation pages with details of the transaction and e-ticket delivery.

3.1.2 Hardware Interfaces

The Movie Theater Ticketing System shall be accessible via standard hardware devices, including desktop computers, laptops, tablets, and smartphones. It shall be compatible with common peripherals such as keyboards and touchscreens.

3.1.3 Software Interfaces

The system shall interface with external software components, including:

- Payment gateways for processing transactions securely.
- Database systems for storing and retrieving movie listings, showtimes, user accounts, and transaction data.

3.1.4 Communications Interfaces

The system will communicate with users through standard web communication protocols (e.g., HTTP, HTTPS). It will also support email notifications for transaction confirmations and user account updates.

3.2 Functional Requirements

3.2.1 Purchasing a Ticket

3.2.1.1 Introduction
- This functionality enables users to purchase movie tickets through the Movie Theater Ticketing System.

3.2.1.2 Inputs
- Selected movie and showtime.
- Number of tickets to purchase.
- Payment information (credit card details, PayPal account, Bitcoin wallet, etc.).

3.2.1.3 Processing
- Validate user inputs and check ticket availability.
- Calculate the total transaction amount based on ticket prices and any applicable discounts.
- Process payment securely through integrated payment gateways.

3.2.1.4 Outputs
- Confirmation page displaying details of the transaction.
- E-tickets delivered to the user via email or available for download/printing.

3.2.1.5 Error Handling
- Display error messages for invalid inputs or transaction failures.
- Provide assistance options for users encountering issues during the purchasing process.

3.2.2 Creating an Account

3.2.2.1 Introduction
- This functionality allows users to create accounts on the Movie Theater Ticketing System.

3.2.2.2 Inputs
- User registration details (name, email address, password).
- Personal information (optional) for account customization and loyalty programs.

3.2.2.3 Processing
- Validate user inputs and check for duplicate accounts.
- Store user information securely in the database.
- Track purchase history and loyalty points for registered users.

3.2.2.4 Outputs
- User profile page displaying account details and purchase history.
- Loyalty points balance and rewards information (if applicable).

3.2.2.5 Error Handling
- Notify users of registration errors or account-related issues.
- Provide options for account recovery (e.g., password reset).

3.2.3 Holding a Ticket for Purchase

3.2.3.1 Introduction
- This feature allows users to hold tickets for a limited time while they complete the purchase process.

3.2.3.2 Inputs
- Selected movie and showtime.
- Desired number of tickets.

- Reservation duration (if applicable).

3.2.3.3 Processing
- Reserve selected seats for the user.
- Apply a hold on the selected seats for the specified duration.
- Prevent other users from purchasing the held seats during the reservation period.

3.2.3.4 Outputs
- Confirmation message indicating successful reservation.
- Timer displaying the remaining reservation duration.

3.2.3.5 Error Handling
- Notify users if the selected seats are unavailable for reservation.
- Provide options for users to extend or release the hold on their reserved tickets.

3.3 Use Cases

3.3.1 Use Case #1 - Purchasing a Ticket

Actor: User

Description: This use case describes the process of a user purchasing a movie ticket through the ticketing system.

Scenario:

1. User logs in to the ticketing system.
2. User searches for a desired movie and showtime.
3. User selects the desired number of tickets and seating preferences.
4. User proceeds to checkout and selects a payment method.
5. User enters payment information and confirms the transaction.
6. System processes the payment and issues e-tickets to the user.
7. User receives confirmation of the transaction via email.

3.3.2 Use Case #2 - Creating an Account

Actor: User

Description: This use case outlines the process of a user creating an account on the ticketing system.

Scenario:

1. User navigates to the registration page of the ticketing system.
2. User enters required personal information such as name, email, and password.
3. User agrees to terms and conditions and submits the registration form.
4. System validates the information and creates a new user account.
5. User receives a confirmation email with account details.
6. User can now log in to the system using the newly created account.

3.3.3 Use Case #3 - Checking Available Seats

Actor: User

Description: This use case describes how a user can check the availability of seats for a specific movie and showtime.

Scenario:

1. User selects a movie and showtime from the list of available options.
2. User navigates to the seating selection page.

3. System displays a seating chart for the selected movie and showtime, indicating available and occupied seats.
4. User can view seat availability and select desired seats based on preference.
5. System updates the seating chart in real-time to reflect seat selections by other users.
6. User completes the seat selection process and proceeds with ticket purchase or reservation.

3.4 Classes / Objects

3.4.1 User Account

3.4.1.1 Attributes

- Username: String
- Password: String
- Email: String
- Name: String
- Purchase History: List
- Loyalty Points: Integer

3.4.1.2 Functions

- Create Account: Allows a user to create a new account with the system.
- Update Account Information: Enables a user to update their account details such as email or password.
- View Purchase History: Retrieves and displays the user's purchase history.
- Earn Loyalty Points: Adds loyalty points to the user's account based on ticket purchases.
- Redeem Loyalty Points: Allows the user to redeem loyalty points for discounts or rewards.

(Reference to functional requirements and/or use cases: Purchasing a Ticket, Creating an Account)

3.4.2 Theater

3.4.1.1 Attributes

- Theater Name: String
- Location: String
- Showtimes: List
- Available Seats: Integer
- Ticket Prices: Map
- Employee List: List

3.4.1.2 Functions

- Add Showtime: Enables theater management to add new showtimes for movies.
- Update Showtime: Allows theater management to modify existing showtimes.
- Check Available Seats: Retrieves and displays the number of available seats for a specific showtime.
- Set Ticket Prices: Sets prices for tickets based on factors such as movie popularity or seating location.
- Manage Employees: Enables theater management to add, remove, or update employee information.

(Reference to functional requirements and/or use cases: Holding a Ticket for Purchase)

3.5 Non-Functional Requirements

3.5.1 Performance

- The system will process ticket purchases within 5 seconds of user submission.
- Response time for loading pages shall be less than 2 seconds under normal load conditions.
- The system will support concurrent usage by at least 1000 users without performance degradation.

3.5.2 Reliability

- The system will have a mean time between failures (MTBF) of at least 30 days.
- The probability of data loss or corruption shall be less than 0.01% per transaction.

3.5.3 Availability

- The system will be available 24/7, with scheduled downtime not exceeding 1 hour per month for maintenance.
- Availability of core functionalities, such as ticket purchasing and account management, shall be maintained at 99.9% uptime.

3.5.4 Security

- User authentication shall be enforced for all system access.
- Payment information shall be encrypted during transmission and storage.
- The system shall adhere to industry standards for data protection and compliance (e.g., PCI DSS for payment card data).

3.5.5 Maintainability

- The system shall be modularly designed to facilitate future updates and enhancements.
- Codebase documentation shall be maintained and updated regularly.
- Bug fixes and security patches shall be deployed within 24 hours of identification.

3.5.6 Portability

- The system will be compatible with major web browsers (Chrome, Firefox, Safari, Edge) and operating systems (Windows, macOS, Linux).
- Mobile responsiveness shall be ensured for seamless usage on smartphones and tablets.

3.6 Inverse Requirements

- The system will not require more than 10 seconds to process any transaction under normal operating conditions.
- System downtime shall not exceed 1 minute per month for maintenance.

3.7 Design Constraints

- The system design must comply with industry standards for web application development (e.g., RESTful APIs, MVC architecture).
- Database schemas and queries must adhere to normalization principles to ensure data integrity and performance.

3.8 Logical Database Requirements

- A relational database management system (RDBMS) shall be used to store system data.

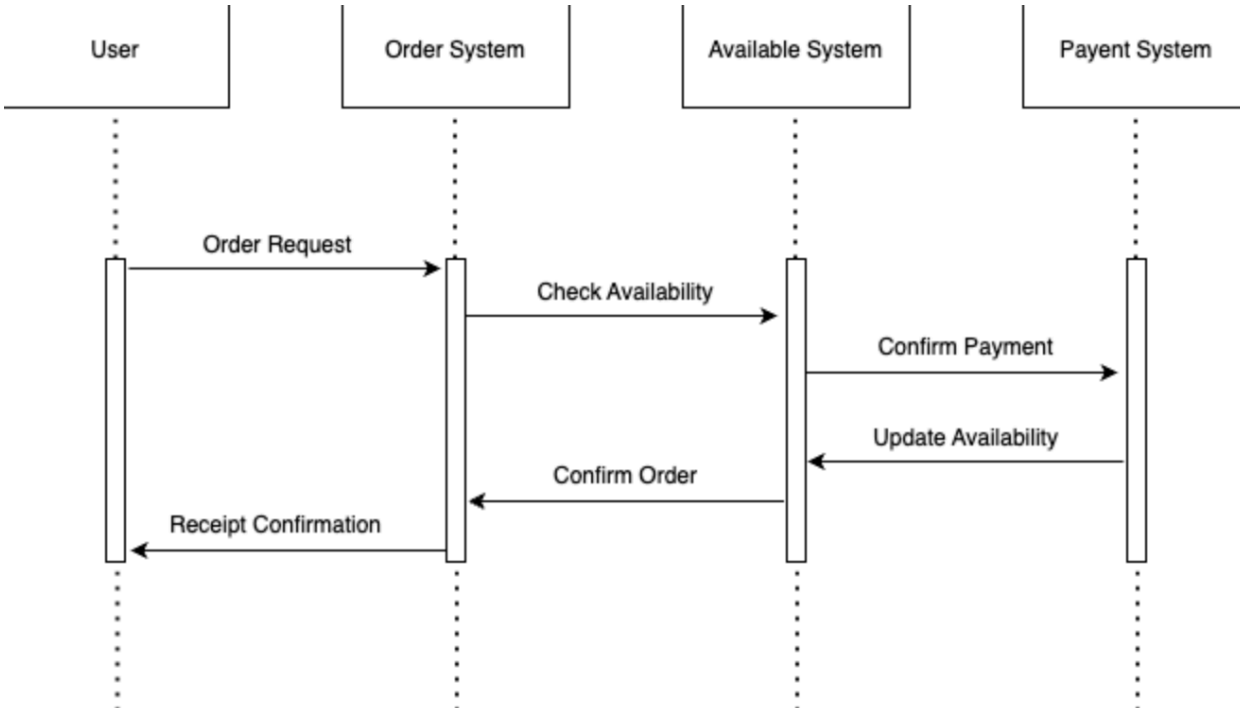
- Data formats shall follow industry standards for consistency and compatibility.
- The database shall have sufficient storage capacity to handle anticipated data volumes for at least 5 years.

3.9 Other Requirements

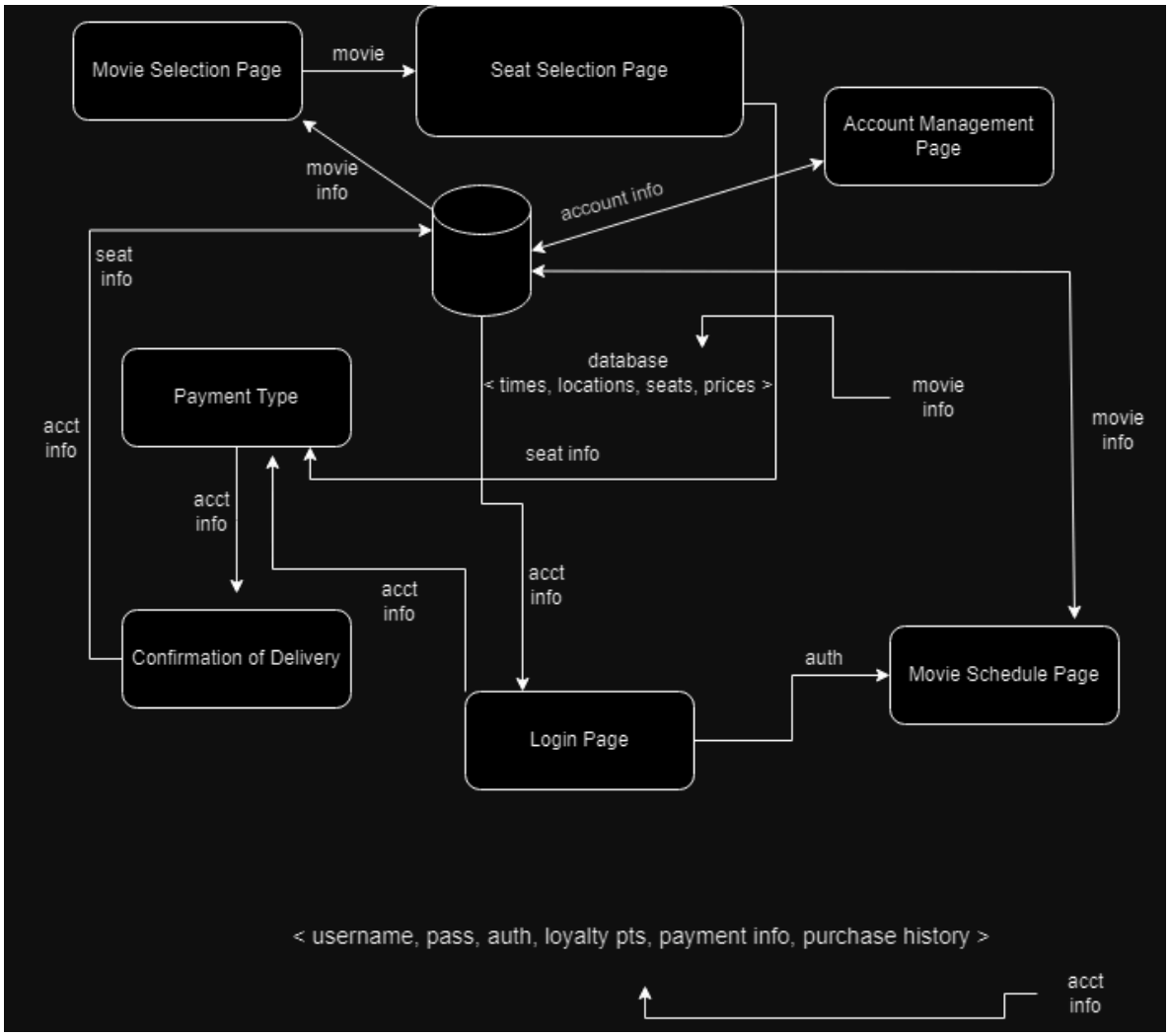
- The system shall provide multilingual support for English, Spanish, and Swedish languages.
- User interface elements shall comply with accessibility standards (e.g., WCAG) to ensure usability for users with disabilities.

4. Analysis Models

4.1 Sequence Diagram



4.2 Software Architecture Diagram (SWA)



Components Overview

- **Movie Selection Page**
  - **Function:** This is where users select the movie they want to watch.
  - **Connectors:**
    - Sends selected movie information to the Seat Selection Page.
    - Receives movie info from a database to display available movies.
- **Seat Selection Page**
  - **Function:** Users choose their preferred seats on this page.
  - **Connectors:**
    - Sends seat info to the Payment Type for transaction processing.
    - Receives account info from the Account Management Page to verify user details.
    - Receives seat info from the database to display available seats.

- **Account Management Page**
  - **Function:** Handles user account details, preferences, and history.
  - **Connectors**:
    - Sends account info to the Seat Selection Page for booking tickets.
    - Sends movie info to the Movie Schedule Page to display the movie timings and details.
- **Payment Type**
  - **Function:** Allows users to choose a payment method and processes the payment.
  - **Connectors:**
    - Sends acct info to the Confirmation of Delivery to finalize the transaction.
    - Receives seat info and acct info from the Seat Selection Page to process the payment.
- **Confirmation of Delivery**
  - **Function:** Confirms the successful transaction and ticket delivery.
  - **Connectors:**
    - Receives acct info from the Payment Type to confirm the transaction and update the user's account.
- **Login Page**
  - **Function:** Authenticates users before they access their accounts or make a booking.
  - **Connectors:**
    - Sends auth (authentication) information to the Movie Schedule Page to ensure the user is logged in before accessing movie schedules.
    - Receives acct info from users attempting to log in.
- **Movie Schedule Page**
  - **Function:** Displays the schedule of movies.
  - **Connectors:**
    - Receives auth from the Login Page to display schedules to authenticated users.
    - Sends movie info to the Account Management Page to allow users to manage their bookings.
- **Database**
  - **Function:** Stores all the data required for the system to function, like movie times, locations, seat availability, and pricing.
  - **Connectors:**
    - Interacts with multiple components, providing times, locations, seats, and prices to the Seat Selection Page, and seat info to the Payment Type.

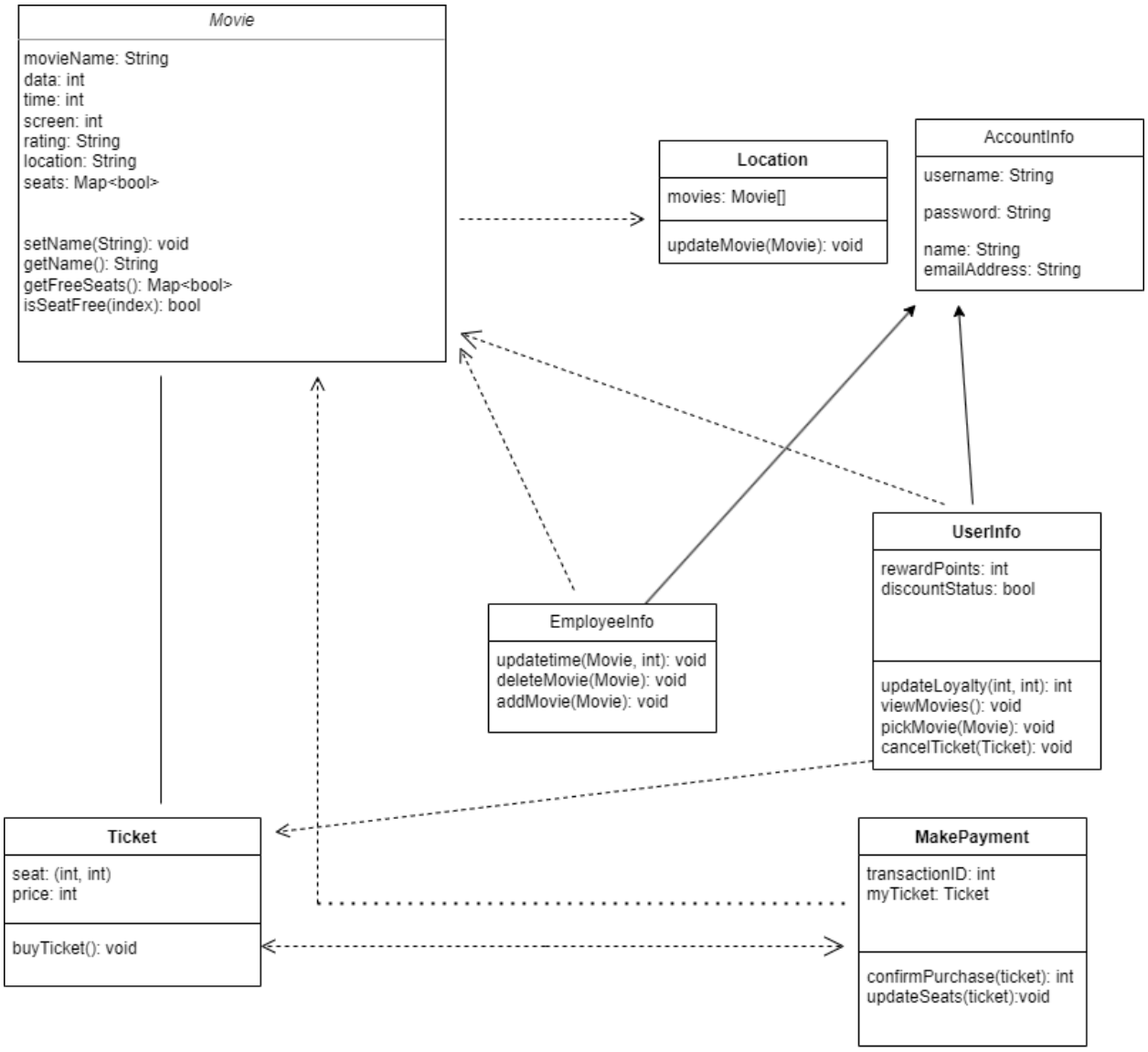
Bottom Connector (username, pass, auth, loyalty pts, payment info, purchase history):

- This represents the flow of information that is likely used throughout the system for different purposes. It connects the entire system architecture and suggests that these pieces of information are central to the system's operations and are passed between various components.

Explanation of Connectors and Flow:

- **Movie Selection to Seat Selection:** After selecting a movie, the relevant information is passed to the Seat Selection Page, so the user can choose their seats.
- Seat Selection to Payment: Once seats are selected, this information, along with the user's account details, is passed to the Payment Type to proceed with the transaction.
- **Payment to Confirmation:** After payment details are provided, the information is sent to the Confirmation of Delivery component to confirm the transaction and issue the ticket.
- **Account Management to Seat Selection and Movie Schedule:** User account details are shared with the Seat Selection for booking and the Movie Schedule Page to display personalized movie schedules.
- **Login to Movie Schedule:** Ensures that only authenticated users can access the movie schedule.
- **Database Interactions:** The database acts as a central repository, providing necessary data to the Seat Selection and Payment Type components and receiving updates from them as transactions occur.

4.3 UML Class Diagram



Classes Description:

- **Movie**
  - **Attributes:**
    - **movieName: String** - A text attribute to store the name of the movie.
    - **date: int** - An integer to store the date, which is likely a timestamp or similar numerical representation.
    - **time: int** - An integer to store the time, which could also be a timestamp.
    - **screen: int** - An integer indicating which screen the movie is showing on.
    - **rating: String** - A text attribute to store the movie's rating.
    - **location: String** - A text attribute to store the location where the movie is being shown.
    - **seats: Map<bool>** - A map structure with boolean values to represent the occupancy status of each seat.

    - **Operations:**
        - **setName(String): void** - A method to set the movie's name, takes a string as a parameter.
        - **getName(): String** - A method to get the movie's name, returns a string.
        - **getFreeSeats(): Map<bool>** - A method to get a map of the free seats, returns a map with boolean values.
        - **isSeatFree(index): bool** - A method to check if a seat is free, takes an index as a parameter and returns a boolean.
- **Ticket**
    - **Attributes**:
        - **seat: (int, int)** - A tuple of two integers representing the seat location (row, column).
        - **price: int** - An integer to store the price of the ticket.
    - **Operations:**
        - **buyTicket(): void** - A method to initiate the purchase of a ticket.
- **Location**
    - **Attributes**:
        - **movies: Movie[]** - An array of Movie objects.
    - **Operations**:
        - **updateMovie(Movie): void** - A method to update a movie, takes a Movie object as a parameter.
- **AccountInfo**
    - **Attributes**:
        - **username: String** - A text attribute to store the username.
        - **password: String** - A text attribute to store the password.
        - **name: String** - A text attribute to store the name of the account holder.
        - **emailAddress: String** - A text attribute to store the email address.
    - **Operations**: None specified.
- **UserInfo**
    - **Attributes**:
        - **rewardPoints: int** - An integer to store the reward points.
        - **discountStatus: bool** - A boolean to store the status of a discount.
    - **Operations**:
        - **updateLoyalty(int, int): int** - A method to update loyalty points, takes two integers as parameters and returns an integer.
        - **viewMovies(): void** - A method to view movies.
        - **pickMovie(Movie): void** - A method to select a movie, takes a Movie object as a parameter.
        - **cancelTicket(Ticket): void** - A method to cancel a ticket, takes a Ticket object as a parameter.
- **EmployeeInfo**
    - **Operations**:
        - **updateMovie(Movie, int): void** - A method to update movie information, takes a Movie object and an integer as parameters.
        - **deleteMovie(Movie): void** - A method to delete a movie, takes a Movie object as a parameter.

        - **addMovie(Movie): void** - A method to add a new movie, takes a Movie object as a parameter.
- **MakePayment**
    - **Attributes**:
        - **transactionID: int** - An integer to store the transaction ID.
        - **myTicket: Ticket** - A Ticket object.
    - **Operations**:
        - **confirmPurchase(ticket): int** - A method to confirm the purchase of a ticket, takes a Ticket object as a parameter and returns an integer.
        - **updateSeats(ticket): void** - A method to update the seats after purchase, takes a Ticket object as a parameter.

Relationships Description:

- The **Location** class has a one-to-many relationship with the **Movie** class, indicating that one location can have multiple movies.
- The **UserInfo** class is related to **AccountInfo**, **EmployeeInfo**, and **MakePayment**. The exact nature of these relationships isn't specified but could indicate inheritance (UserInfo is a type of AccountInfo) or association (UserInfo interacts with EmployeeInfo and MakePayment).
- The **MakePayment** class is associated with the **Ticket** class, indicating that the payment process is related to purchasing tickets.

4.4 Data Management

SQL Table #1 - Theater DB

UID	Theater ID	# of seats	Layouts	ADA	Location
LM01	01	20	Reg	Y	La Mesa
LM02	02	10	Deluxe	Y	La Mesa
	01	50	Reg	Y	La Mesa
	02	100	Reg	Y	La Mesa

SQL Table #2 - Movie DB - Location Specific La Mesa

Name	Showtime	Rating	Genre	Duration	Theater ID	Available Seats
Minions	7 PM	★★★★★	Comedy	91	01	1
Batman	7 PM		Deluxe		02	10
Minions	9 PM		Reg		01	0
			Reg			

SQL Table #3 - User Accounts

Email	Password	Payment Info	Rewards	Name

Centralization vs. Decentralization: We have chosen a centralized approach where all related data resides within a single database system but in separate tables. This makes it easier to manage and ensure data integrity through referential constraints and transactions.

Security: For the User Accounts table, ensure the protection of sensitive data like passwords (which should be hashed and salted) and payment information (which should be encrypted and possibly tokenized).

Design Decisions:

- **Three Tables for Different Domains:** The decision to have separate tables for movies, theaters, and users helps segregate the data logically. This aids in data management and can help with performance if properly indexed.
- **Normalization:** It seems the tables are designed with normalization in mind, reducing redundancy by, for example, referencing theater IDs instead of repeating theater information.

Logical Data Split:

- **Movies and Theaters Relationship:** The MovieDB table includes a foreign key to TheaterDB, suggesting a one-to-many relationship (one theater can have many movies shown).
- **Normalization Trade-offs:** While normalization minimizes redundancy and ensures data integrity, it can sometimes lead to more complex queries and potentially slower performance on joins.

Technology Alternatives:

- **SQL Alternatives:** Using a different SQL database system might provide performance or feature benefits. For example, PostgreSQL offers advanced data types and Oracle Database might provide more robust enterprise features.
- **NoSQL Alternatives:** If scalability becomes an issue or if the data becomes more unstructured, we might consider NoSQL databases. For instance, a document-based NoSQL database could store all movie and theater information in a single document, simplifying queries at the cost of transactional integrity.'

Tradeoffs:

- **SQL Constraints vs. Flexibility:** By using SQL databases, we're opting for a system with strong consistency and ACID (Atomicity, Consistency, Isolation, Durability) properties, which is excellent for transactions. However, this can come at the cost of scalability and flexibility compared to NoSQL databases.
- **Performance Optimization:** Depending on the size and query patterns, a single, well-indexed SQL database could be optimized for performance. However, as your dataset grows, you might encounter scaling issues that NoSQL databases can handle more gracefully.

A. Appendices

A.1 User Manual

Getting Started

1. Open your web browser and navigate to the Movie Theater Ticketing System website.
2. Create an Account: Click on the "Sign Up" button and enter your required information.
3. Login: If you already have an account, enter your username and password on the login page.

Searching for Movies

1. On the homepage, you will see a search bar where you can enter the title of the movie you want to see.
2. The system will display a list of matching movies currently playing.
3. Click on the movie poster or title to view more details, including showtimes and available seats.

Selecting Seats and Booking Tickets

1. Choose the showtime that best suits you.
2. A seating chart will be displayed, highlighting available seats.

3. Click on the seats you want to purchase.
4. The system will display the total price for your selected seats.
5. Review your selections and proceed to checkout.

Payment Options

1. The system supports various payment methods, including credit cards, PayPal, and Bitcoin.
2. Choose your preferred payment method and enter your payment information securely.
3. Once your payment is confirmed, you will receive a confirmation email with your e-tickets attached.

Additional Features

- Account Management: Update your profile information, view your purchase history, and manage loyalty points (if applicable).
- Ticket Reservations: Reserve tickets for upcoming shows (subject to availability).
- Movie Reviews: Read reviews and critic quotes from various online platforms (if available).

System Requirements

- Internet connection
- Web browser (latest version recommended)

For any questions or assistance, please visit our Help Center or contact customer support.

Design Documents

SDS: Test Plan

Test Case

Component	Component	Priority	Description/Test Summary	Pre-requisites	Test Steps	Expected Result	Actual Result	Status	Test Executed By
MovieNameTest_1	Movie Class	High	Test the setName() and getName() methods for correct operation.	Movie class instance is created.	1. Call setName('Inception'). 2. Call getName().	getName() should return 'Inception'.	getName() should return 'Inception'.	Pass	Nikolo
SeatAvailabilityTest_1	Movie Class	High	Test the isSeatFree(index) method for correct seat availability.	Movie instance with initialized seats is created.	1. Call isSeatFree(5) for a seat that is known to be free.	isSeatFree(5) should return true.	isSeatFree(5) should return true.	Pass	Atah
SelectMovieTest_1	User Interface	Medium	Verify that selecting a movie shows correct available seats.	User is on the Seat Selection Page with a specific movie selected.	1. User selects 'Inception'. 2. System displays available seats.	Available seats for 'Inception' are displayed as per the current database state.	Available seats for 'Inception' are displayed as per the current database state.	Pass	Atah
AccountCreationLoginTest_1	Account Management Page	High	Test account creation and subsequent login functionality.	Account Management Page is available.	1. Create an account with username 'user1'. 2. Logout. 3. Attempt to login with 'user1'.	Account is created successfully and login with 'user1' is successful.	Account is created successfully and login with 'user1' is successful.	Pass	Nikolo
CompleteTicketPurchaseTest_1	Entire System	Critical	Test the end-to-end process of purchasing a ticket.	User is logged in and at the Movie Selection Page.	1. Select a movie. 2. Choose a seat. 3. Make a payment. 4. Receive a confirmation.	User is able to select, pay for the seat, and receive a ticket confirmation.	User is able to select, pay for the seat, and receive a ticket confirmation.	Pass	Atah
MovieScheduleUpdateTest_1	Movie Schedule Page	Medium	Ensure updates to the movie schedule are shown to the user.	A change in the movie schedule has been made in the database.	1. Go to the Movie Schedule Page. 2. Check the schedule for 'Inception'.	The updated schedule for 'Inception' is displayed correctly.	The updated schedule for 'Inception' is displayed correctly.	Pass	Bryce
PaymentTicketConfirmationTest_1	Payment Type	High	Validate that the payment process and ticket issuing works correctly.	User has selected a movie and a seat.	1. Enter payment details for the selected seat. 2. Confirm the purchase.	Payment is processed and a ticket is issued.	Payment is processed and a ticket is issued.	Pass	Nikolo
UserAccountUpdateTest_1	Account Management Page	Medium	Test updating user account information.	User is logged in and on the Account Management Page.	1. Change account email to 'user1@test.com'. 2. Submit the changes.	Account information is updated in the database.	Account information is updated in the database.	Pass	Bryce
EmployeeMovieManagementTest_1	Employee Interface	High	Test the ability of an employee to manage movies in the system.	Employee is logged in and has access to movie management tools.	1. Add a new movie 'Matrix'. 2. Update the 'Matrix' details. 3. Delete 'Matrix'.	New movie 'Matrix' is added, updated, and then deleted successfully.	New movie 'Matrix' is added, updated, and then deleted successfully.	Pass	Atah
SeatSelectionReservationTest_1	Seat Selection Page	High	Ensure that seat selection and reservation operate correctly.	User is on the Seat Selection Page after selecting a movie.	1. Choose a seat. 2. The system reserves the seat.	The selected seat is reserved and confirmed for the user.	The selected seat is reserved and confirmed for the user.	Pass	Bryce

Life Cycle Model

For our Cinemate project, our software development lifecycle mirrored the sequence of assignments in our class. We commenced by delving into the Software Requirements Specification (SRS) phase, focusing on gathering essential information like hardware and software requirements, functional and non-functional specifications, as well as outlining various use cases. Once we amassed the fundamental requirements, we transitioned to the analysis phase, where we scrutinized these requirements and crafted our software design models accordingly. This involved creating artifacts such as the Software Architecture Diagram and the UML Class Diagram.

Following this, we momentarily diverged from the SRS to draft a comprehensive test plan for the verification and validation phase. This allowed us to ensure the robustness and reliability of our software. After finalizing the test plan, we returned to the SRS to address data management aspects, integrating these considerations into our design and requirements.

Reflecting on our software development lifecycle, it proved effective for our Cinemate project. While there may have been opportunities for streamlining certain stages, particularly in a solo context, the chosen approach demonstrated scalability, especially in team settings. Although there were minor adjustments prompted by feedback from teaching assistants, the overall structure of our lifecycle remained intact.

One notable departure from conventional practices was the absence of an implementation phase in our project timeline. Typically situated between the design and verification phases, this phase was omitted, leading us directly to the verification phase. Consequently, we had to simulate test runs instead of executing them in practice.

In summary, while there could have been optimizations for efficiency, particularly in solo endeavors, the chosen lifecycle proved adept for collaborative projects like Cinemate. However, the exclusion of the implementation phase necessitated some adjustments, underscoring the need for adaptability in software development methodologies.

Summary

The Cinemate project is a comprehensive endeavor aimed at revolutionizing the movie ticketing experience through the development of a web-based Movie Theater Ticketing System. This system is designed to streamline the process of purchasing tickets online, enhancing convenience for users while providing theater owners with effective tools for managing ticket sales. The Software Requirements Specification (SRS) serves as the cornerstone of this project, offering a detailed roadmap that outlines the system's features, functionalities, and constraints.

Within the SRS, various aspects of the Movie Theater Ticketing System are meticulously explored. From the general description to specific requirements, each section provides valuable insights for stakeholders involved in the development and testing process. The SRS covers a wide array of topics, including user interfaces, hardware and software interfaces, functional requirements such as purchasing tickets and creating accounts, and non-functional requirements like performance, reliability, and security.

Analysis models such as sequence diagrams, software architecture diagrams, and UML class diagrams offer a deeper understanding of the system's design and structure. These models facilitate effective communication among team members and provide a visual representation of key system components and their interactions.

Furthermore, considerations regarding data management are carefully addressed within the SRS. Discussions on centralization versus decentralization, security measures, and design decisions underscore the importance of robust data management practices in ensuring the integrity and security of the system.

In conclusion, the Cinemate project is poised to deliver a cutting-edge solution that redefines the movie ticketing experience. With a focus on user convenience, system reliability, and data security, the Movie Theater Ticketing System promises to revolutionize the way users engage with movie theaters. As the project progresses through the implementation and testing phases, stakeholders can expect a seamless and enjoyable movie ticketing experience that caters to the needs of both users and theater owners alike.

Conclusion

The Cinemate project represents a significant step forward in the realm of movie ticketing systems, offering a comprehensive solution that addresses the needs of users and theater owners alike. With the development of a web-based Movie Theater Ticketing System, the project aims to streamline the ticket purchasing process, enhance user convenience, and provide theater owners with effective tools for managing ticket sales.

The Software Requirements Specification (SRS) serves as the foundation for this endeavor, providing a detailed roadmap that outlines the system's features, functionalities, and constraints. From user interfaces to data management considerations, the SRS covers a wide array of topics essential for the successful development and implementation of the Movie Theater Ticketing System.

Throughout the project, adherence to industry standards and best practices has been a top priority. From ensuring compliance with regulations to implementing robust security measures, every aspect of the system is designed with user security and satisfaction in mind.

Looking ahead, the project will proceed to the implementation phase, where the system's design will be translated into code. Rigorous testing will follow to validate the system's functionality and reliability, ensuring a seamless and enjoyable experience for users and theater owners alike.

In conclusion, the Cinemate project represents a groundbreaking effort to redefine the movie ticketing experience. With a focus on innovation, convenience, and user satisfaction, the Movie Theater Ticketing System promises to revolutionize the way users interact with movie theaters, ushering in a new era of convenience and efficiency in the process.