<MovieMania>

Software Design Specification

<Version 1>

<7/22/2025>

<Group 1#>
<Suber Ebrahim and Deryn Cabana>

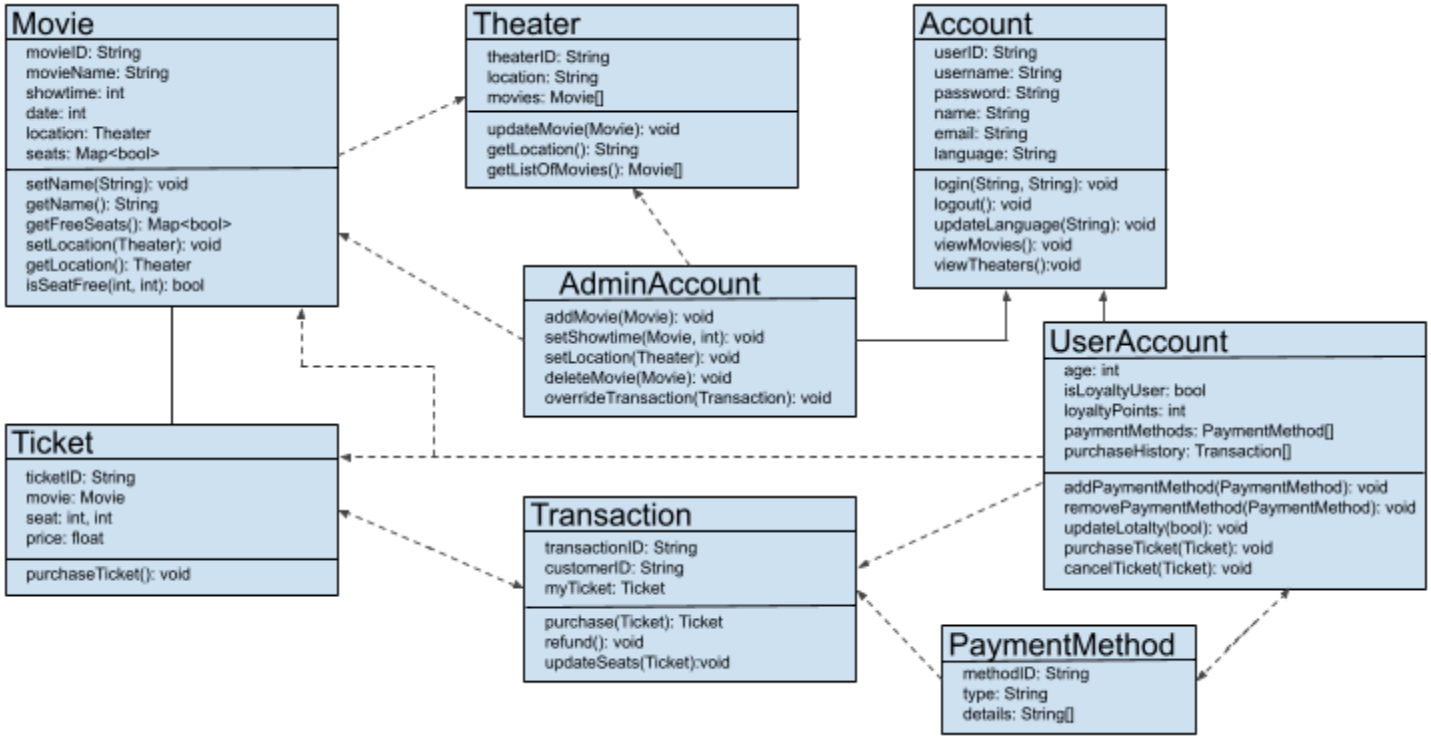
Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.
Summer 2025

1. System Description

The theater ticketing system is a web-based application designed to facilitate the discovery, selection, and purchase of movie tickets across a network of 20 theaters in the San Diego area. The system supports both regular and deluxe theaters, enforces purchase constraints, and accommodates optional user accounts. Key features include secure digital and physical ticket issuance. This system consists of two main sides, the user interface through accounts, and the theater system database, containing theaters, movies, tickets, and more. The system is developed using a modular architecture with distinct classes responsible for user management, ticketing, payments, and administration. It employs centralized data storage partitioned by theater, with APIs to external services for payment and more.

2. Software Architecture Overview

2.1 UML Diagram



2.1.1 Class: Movie

Purpose: Represents an individual film with associated showtime, seating, and other metadata.
Attributes:
movieID: String - Unique identifier for the movie
movieName: String - Title of the movie
showtime: int - Start time of the movie in a simplified integer format
date: int - Date of the movie showing in an integer format

location: Theater - The theater location where the movie is being shown
seats: Map<bool> - Data structure representing seat availability, with seat ID mapped to availability
Operations:
setName(String): void - Updates the movie title
getName(): String - Retrieves the movie title
getFreeSeats(): Map<bool> - Returns a map of available seats
setLocation(Theater): void - Updates the theater the movie is showing at
getLocation(): Theater - Retrieves the movie location
isSeatFree(int,int): bool - Checks is a specific seat is available, by row and column

2.1.2 Class: Theater

Purpose: Represents a physical theater, with multiple movies playing.
Attributes:
theaterID: String - Unique identifier for the movie
location: String - Physical location of the theater (likely a city address)
movies: Movie[] - Array of movies playing at the theater
Operations:
updateMovie(Movie): void - Updates a movie through the operations available in the Movie class
getLocation(): String - Retrieves the location of the theater
getListOfMovies(void): Movie[] - Returns a list of movies playing at this theater

2.1.3 Class: Account

Purpose: Base class for all users who log into the system, either customers or administrators.
Attributes:
userID: String - Unique user ID
username: String - username for login
password: String - Password for authentication
name: String - Full name of the user
email: String - User's email address
language: String - Preferred language for UI localization
Operations:
login(String, String): void - Authenticates user login using credentials. The first parameter is either their username or email; the second parameter is their password
logout(): void - Ends the user session
updateLanguage(String): void - Changes the user's language preference
viewMovies(): void - Allows the user to browse available movies
viewTheaters(): void - Allows the user to browse supported theater locations

2.1.4 Class: UserAccount

Purpose: Represents a registered customer with optional loyalty features. Inherits from Account.
Attributes:

age: int - User's age, used for age-restricted content or discounts
isLoyaltyUser: bool - Indicates if the user is enrolled in a loyalty program
loyaltyPoints: int - Points accrued through purchases with a loyalty program
paymentMethods: PaymentMethod[] - List of stored payment options
purchaseHistory: Transaction[] - List of past ticket purchases and related data
Operations:
addPaymentMethod(PaymentMethod): void - Adds a new payment method to the account
removePaymentMethod(PaymentMethod): void - Removes a stored payment method from the account
updateLoyality(bool): void - Allows the user to enroll in or unsubscribe from the loyalty program
purchaseTicket(Ticket): void - Initiates the ticket purchase process
cancelTicket(Ticket): void - Cancels a previously purchased ticket (if permitted)

2.1.5 Class: AdminAccount

Purpose: Represents system administrators with management permissions. Inherits from Account.
Operations:
addMovie(Movie): void - Adds a new movie to the database
setShowtime(Movie, int): void - Sets or updates the showtime for a specific movie
setLocation(Theater): void - Assigns or changes the theater for a movie
deleteMovie(Movie): void - Removes a movie from the system
overrideTransation(Transaction): void - Manually corrects, cancels, or refunds a transaction

2.1.6 Class: Ticket

Purpose: Represents a ticket for a specific movie and seat.
Attributes:
ticketID: String - Unique identifier for the ticket
movie: Movie - Associated movie object
seat: int, int - Row and column of the seat the ticket is for
price: float - Total price of the ticket after discounts
Operations:
purchaseTicket(): void - Moves onto transaction for the ticket purchase process

2.1.7 Class: Transaction

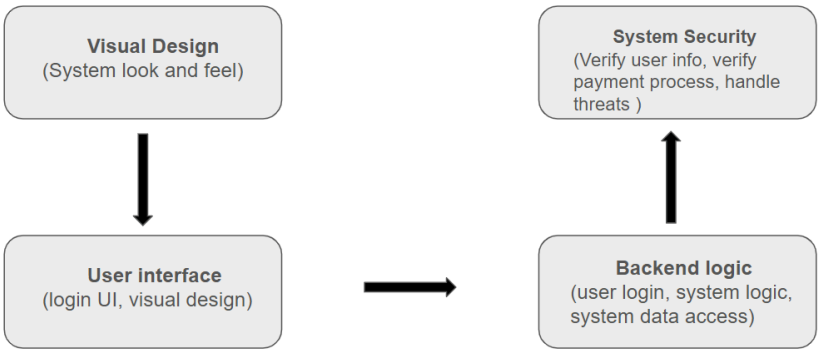
Purpose: Represents a complete payment and booking activity tied to a customer and ticket.
Attributes:
transactionID: String - Unique transaction identifier
customerID: String - ID of the user who made the transaction
myTicket: Ticket - The ticket associated with this transaction
Operations:
purchase(Ticket): Ticket - Performs the purchase logic through the user's account and payment method, and returns the resulting ticket
refund(): void - Refunds this transaction (if permitted)

updateSeats(Ticket): void - Updates the seat availability for the seat associated with the purchased ticket

2.1.8 Class: PaymentMethod

Purpose: Represents a stored payment method used for purchases.
Attributes:
methodID: String - Unique ID for the payment method
type: String - type of payment (e.g. “CreditCard”, “PayPal”, “Bitcoin”)
details: String[] - List of stored details for the payment method (e.g. card number, expiry, etc.)

2.2 SWA Diagram



2.2.1 SWA Description

- The SWA chart illustrates the relationships between different components of the system. Starting with Visual Design (system look and feel) influences the User Interface (login UI, visual design). The Frontend User Interface connects with the Backend Logic to carry out functions such as(user login, system logic, system data access), this system as a whole is inspected at at every level for threats and leaks by the System Security (verify user info, verify payment process, handle threats)

3. Development Plan & Timeline

- Week 1 Collect information and resources for SRS documents.
- Week 2 - 5 Core backend/foundational requirements,
- Week 5 - 7 User authentication and initial frontend design.
- Week 8 Debug and test any bugs with user log in.
- Week 9 Movie showcasing system and general user interface.
- Week 10 Work on payment system.
- Week 11-12 Comprehensive system testing and security test

- Week 13 Add misc features such as holiday sales, non recurring discounts etc.

3.1 Partitioning of Tasks

	Khalid(Backend)	Marcus(Frontend)	Kawhi(System security)	Kent(Visual designer)
Week 1	**Collect info for srs**	**Assists Khalid and Kent with UI**	**Collect info for srs**	**Works on initial mock ups**
Week 2-5	**Work on core functional requirements**	**Begins work on frontend framework**	**Works closely with khalid on foundational security requirements**	**Designs initial frontend design**
Week 5-7	**Implements user authentication**	**Develop login pages and work with khalid to connect to backend**	**Test user authentication for bugs or exploits**	**Inspect UI for early visual bugs**
Week 8	**Fix bugs and test**	**Debug any frontend bugs**	**Fix or flag any bugs found in the system**	**Compare notes with team members and make sure art works with the system functions**
Week 9	**Builds movie times and other UI features and logic**	**Implement movie showcase UI**	**Implement more complex system specific security features such as one user per account.**	**Implement movie showcase art style and animation**
Week 10	**Works on payment processing**	**Build payment UI**	**Add a 3rd party system to verify user payment. And user info**	**Implement look for payment screen**
Week 11-12	**Optimize all features and debug**	**Debug any UI bugs or mismatches**	**Check for any new bugs with focus on user info vulnerability.**	**Final check on UI and system art style for any visual bugs**
Week 13	**Build framework for future implementation of**	**Finalize UI and implement promotional/discou**	**Finalize system security debug**	**Implement promotional/special event art style and UI**

	promotional discounts and events	**nts UI**		

3.2 Team Member Responsibilities

1. Khalid Ayman, Backend developer. Responsible for implementing core functional system requirements, and testing and debugging

2. Marcus Smith, Frontend developer. Responsible for connecting building the frontend UI while closely working with Khalid(backend) and Kent(visual designer)

3. Kawhi Leonard, System Security. Responsible for system security, responsibilities include working with other team members to test and debug

4. Kent Bazemore, Visual Designer. Responsible for the overall system look and works closely with Marcus(frontend)