Movie Theater Reservation System

Software Design Specification

1.0.4

June 3, 2024

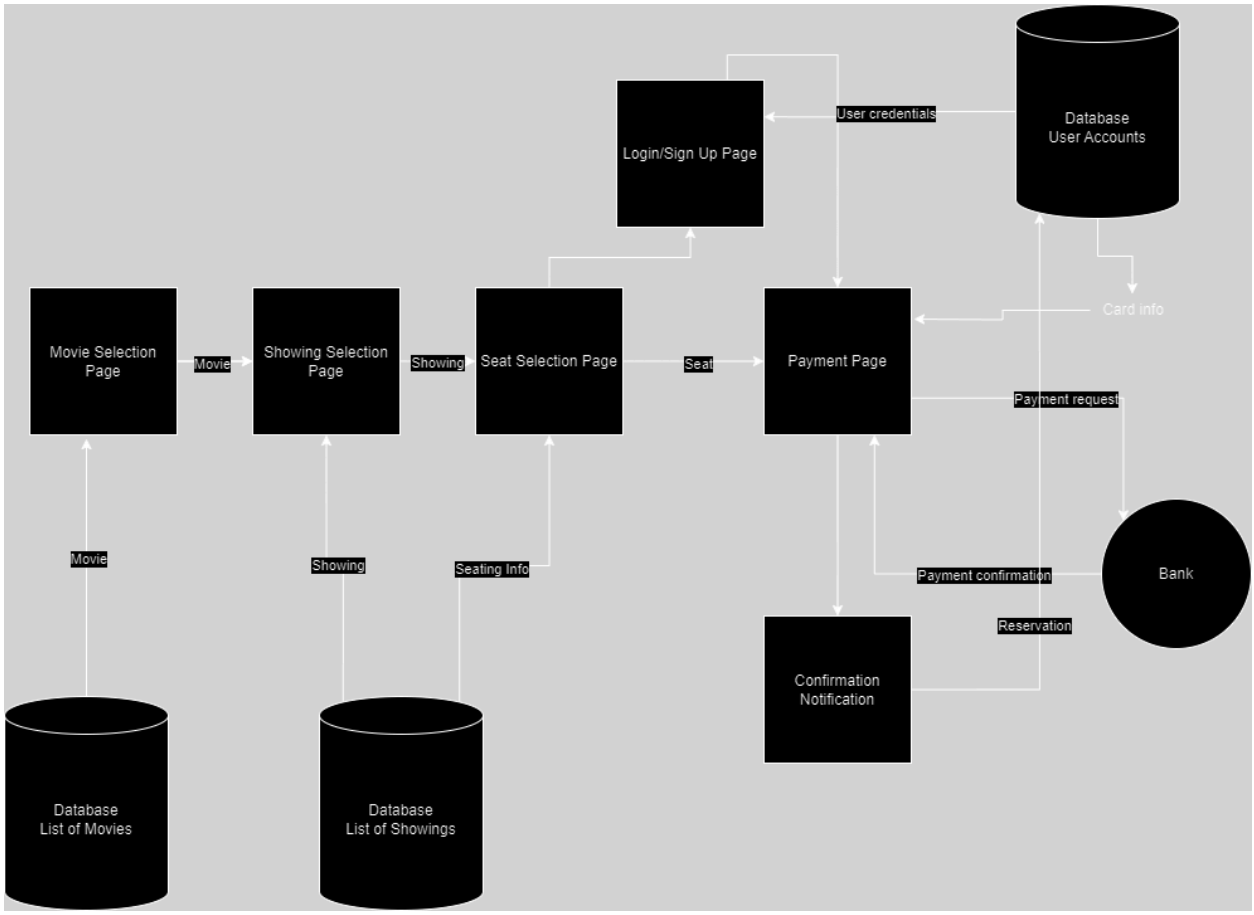
Group 4
Nolan Lee, Akil Parrish, Grant Mauger

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

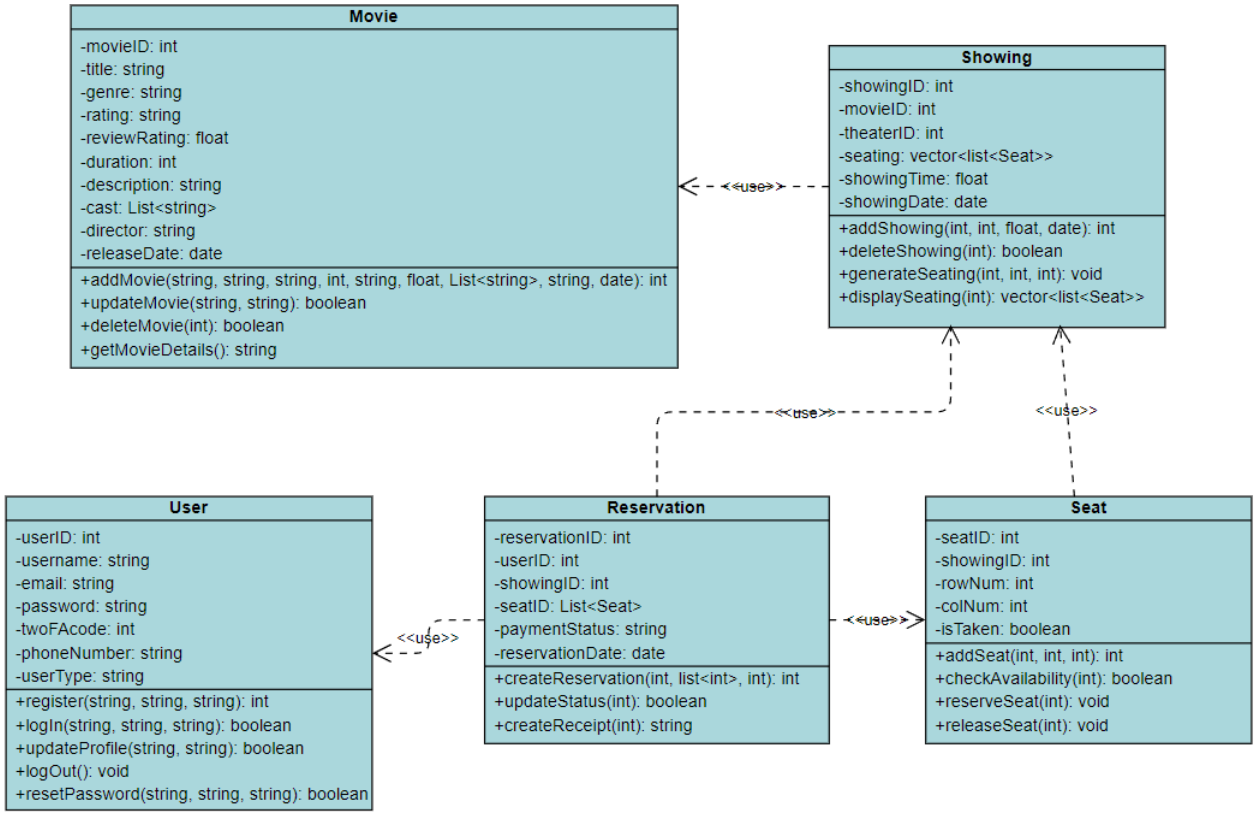
System Description

1. User Interface (UI): This is the front-end of the system where users interact with the system. It could be a web application, mobile application, or even a kiosk at the theater. The UI allows users to browse movies, select a movie, choose a showtime, select seats, and make a payment.
2. Reservation System: This is the core of the application. It handles all the business logic such as checking seat availability, reserving seats, calculating prices, and confirming reservations. It communicates with the User Interface and the Database.
3. Database: This is where all the data is stored. It includes information about the movies, showtimes, theaters, seats, and reservations. The database ensures that the data is stored and retrieved in a secure and efficient manner.
4. Payment Gateway: This is an external service that handles payment transactions. Once the user confirms the reservation, the Reservation System sends a request to the Payment Gateway to process the payment. The Payment Gateway ensures that the payment is processed securely and notifies the Reservation System once the payment is successful.
5. Notification System: Once a reservation is confirmed and payment is successful, the system sends a confirmation notification to the user. This could be an email or a text message.
6. Admin Interface: This is a separate interface for administrators to manage the system. It allows administrators to add or remove movies, change showtimes, update seat layouts, etc.

Software Architecture Overview

- 

    - The movie database provides movies for the movie selection page.
    - The showings database provides times and locations for specific movie showings as well as seating info for that showing in the seat selection page.
    - Once a seat is selected, the user is directed to either log in to their account or create their account. Login credentials are verified with the database of user accounts.
    - On the payment page, any safely stored card information in the user accounts database is safely entered or entered manually by the user and then confirmed.
    - After bank confirmation, the user is notified that their reservation has been confirmed, either through email or text messages.
    - The reservation is stored in the user accounts database as important information about the user.



-
- Description of classes
    - The User class defines the characteristics of each user in the system. It contains user-inputted personal information and passwords, and operations to modify and access that data.
    - The Movie class is what organizes the list of showings that users can view. It houses information about the movie itself, theater it's showing at, and the available Showings. If a user has staff access, there exists functions for adding, deleting, and modifying Movie objects.
    - The Showing class holds the 2D array of Seat objects and the movie ID it points to. Each theater will have a list of Showing objects that each point to the general instance of the Movie it is a showing of.
    - The Seat class is the helper class for functions involving reserving and displaying available seats. It stores its own movie and theater ID, row and seat number, and a boolean flag to indicate whether the seat is taken.
    - A Reservation class object is created when a seat is selected and a payment is requested to the Payment Gateway. It contains a set of IDs to identify itself in the system, and the reservation is updated when payment is complete. Reservations can later be used to verify purchase or for customer service operations.
- Description of attributes
    - User
        - userID (int) : Uniquely generated value for each user.

        - username (string) : User-defined name to be displayed. Does not need to be unique.
        - email (string) : User-entered email, checked for authenticity and uniqueness.
        - password (string) : Encrypted passkey for authentication.
        - twoFAcode (int) : 2FA code sent to email for verification.
        - phoneNumber (string) : Optional method of verification stored.
        - userType (string) : Defines user (customer, staff, admin).
    - Movie
        - movieID (int) : Uniquely generated value for each movie.
        - title (string) : Displayed title, used for search.
        - genre (string) : Displayed genre, used for sorting with search.
        - rating (string): MPAA rating (G, PG, PG-13, R, NC-17).
        - reviewRating (float) : Decimal value 0 through 5, determined from online reviews and websites such as Rotten Tomatoes.
        - duration (int) : Length of movie in minutes, rounded up.
        - description (string) : Description of movie given by producers.
        - cast (list<string>) : Names of primary actors featured in the movie, with links to their IMDb pages.
        - director (string) : Director of the film, with an IMDb link as well.
        - releaseDate (date) : Date of the film's release to theaters.
    - Showing
        - showingID (int) : Uniquely generated value for identifying each showing.
        - movieID (int) : Assigned movie ID that is used for accessing information to display.
        - theaterID (int) : Generated value for identifying the location of the showing.
        - seating (vector<list<Seat>>) : 2D array of seats, ordered by row and seat number, and grayed out in color if taken.
        - showingTime (float) : Time of showing in hours since midnight (Military time) with fractional minute value.
        - showingDate (date) : Date of showing, given in *DD/MM/YYYY*.
    - Seat
        - seatID (int) : Uniquely generated value for each seat, generated using row and seat number as seeds
        - showingID (int) : Stored value for information accessing.
        - rowNum (int) : Value of row given by Showing.
        - colNum (int) : Seat number given by Showing.
        - isTaken (boolean) : A check for the availability of seat.
    - Reservation
        - reservationID (int) : Uniquely generated value for each reservation.
        - userID (int) : Stored value of user for authentication.
        - showingID (int) : Stored value of showing for receipt generation.

        - seatID (int) : Stored values used for receipt generation.
        - paymentStatus (string) : Current status of Payment Gateway's processing.
        - reservationDate (date) : Date of purchase, shown on receipt and stored.
- Description of operations
    - User
        - register : Once all fields have been filled and verified, given data is packaged into a User object and stored in the database.
            - Parameters - (string username, string email, string password, string phoneNumber, string userType)
            - Return Type - Generated userID, [-1] if unsuccessful (int)
        - logIn : Verifies and pushes filled-in fields (either username OR email) to the database, and if authenticated, advances to the home page.
            - Parameters - (string username, string email, string password)
            - Return Type - Indication of successful login (boolean)
        - updateProfile : Takes in a modified piece of user data, attempts to signal the change to the database, and returns whether it was successful.
            - Parameters - (string fieldOfModification, string modifiedData)
            - Return Type - Indication of successful modification (boolean)
        - logOut : Returns to the login screen and clears any user data from cache.
            - Parameters - N/A
            - Return Type - void
        - resetPassword : Authenticates that the user indeed owns that email by verification code, then sends the new encrypted password to the database.
            - Parameters - (string userEmail, string oldPassword, string newPassword)
            - Return Type - Indication of successful change (boolean)
    - Movie
        - addMovie - Admin-only method to add new movies to the list available to staff. ID generated given a seed of the title and current date.
            - Parameters - (string title, string genre, string rating, int duration, string description, float reviewRating, list<string> cast, string director, date releaseDate)
            - Return Type - Generated movieID, [-1] if unsuccessful (int)
        - updateMovie - Admin-only method to modify existing data in a movie object. Separate methods exist for non-string types.
            - Parameters - (string fieldOfModification, string modifiedData)
            - Return Type - Indication of successful modification (boolean)
        - deleteMovie - Admin-only method for manually deleting movies that no theater currently supports or other related reasons.
            - Parameters - (int movieID)
            - Return Type - Indication of successful deletion (boolean)

        - getMovieDetails - Data retrieval methods for staff purposes or for functions in Showing class. Separate functions exist for each return variable.
            - Parameters - (string;int;float;list<string> requestedData)
            - Return Type - Requested data (string;int;float;list<string>)
    - Showing
        - addShowing - Staff-only method for adding new showings for their theater using the generalized associated Movie object. Creates an ID and ties showing information to that ID.
            - Parameters - (int movieID, int theaterID, float showingTime, date showingDate)
            - Return Type - Generated showingID, [-1] if unsuccessful (int)
        - deleteShowing - Staff-only method for manual deletion before the showing has started.
            - Parameters - (int showingID)
            - Return Type - Indication of successful deletion (boolean)
        - generateSeating - Staff-only function to define the size of the individual theater room and create Seat objects to fill the vector.
            - Parameters - (int showingID, int rowLength, int colLength)
            - Return Type - void
        - displaySeating - Method called when a customer clicks on a showing timestamp button, returns available seats.
            - Parameters - (int showingID)
            - Return Type - 2D array of seating (vector<list<Seat>>)
    - Seat
        - addSeat - Helper method for when a showing is generated, instantiates a Seat that is unfilled and contains its given row and column number.
            - Parameters - (int showingID, int seatRow, int seatCol)
            - Return Type - Generated seatID, [-1] if unsuccessful (int)
        - checkAvailability - Additional verification of a seat remaining unfilled.
            - Parameters - (int seatID)
            - Return Type - Filled/unfilled status of seat (boolean)
        - reserveSeat - Reserves the seat for the user, updating the Seat object to be filled.
            - Parameters - (int seatID)
            - Return Type - void
        - releaseSeat - Reverts a seat to unfilled, whether by user choice or staff intervention.
            - Parameters - (int seatID)
            - Return Type - void
    - Reservation

- createReservation - New reservation is created when a seat is selected and payment details are entered and submitted. Contains details about the showing and seats chosen.
    - Parameters - (int showingID, list<int> seatIDs, int userID)
    - Return Type - Generated reservationID, [-1] if unsuccessful (int)
- updateStatus - Status starts at pending when the request is initialized, then updated to completed when indicated by the Payment Gateway.
    - Parameters - (int reservationID)
    - Return Type - Indication of successful change (boolean)
- create Receipt - Assembles user, showing, seating, and payment information using their associated IDs and organizes in a manner for digital or paper receipt creation.
    - Parameters - (int reservationID)
    - Return Type - Completed string for future formatting (string)

Development plan and timeline

1. Project Planning (Duration: 1 week)

- **Objective**: Establish project goals, scope, and constraints.
- **Tasks**:
    - Define the project's objectives and deliverables.
    - Identify the technical and business requirements.
- **Responsibilities**:
    - **Team Lead**: Oversee the planning process and ensure objectives are clear.

2. System Design (Duration: 2 weeks)

- **Objective**: Design the overall system architecture and user interface.
- **Tasks**:
    - Create wireframes for the user interface.
    - Design the database schema and system architecture.
- **Responsibilities**:
    - **UI/UX Designer**: Develop the interface design.
    - **System Architect**: Outline the system's technical design.

3. Development (Duration: 6 weeks)

- **Objective**: Code the application according to the design specifications.

- **Tasks**:
    - Implement the backend logic and database interactions.
    - Develop the frontend interface and connect it to the backend.
- **Responsibilities**:
    - **Backend Developer**: Build the server-side functionality.
    - **Frontend Developer**: Implement the client-side application.

4. Testing (Duration: 3 weeks)

- **Objective**: Ensure the system is reliable, secure, and user-friendly.
- **Tasks**:
    - Conduct unit tests, integration tests, and system tests.
    - Perform user acceptance testing (UAT) with a focus group.
- **Responsibilities**:
    - **Quality Assurance Engineer**: Lead the testing efforts and document results.
    - **All Team Members**: Participate in UAT and address feedback.

5. Deployment (Duration: 1 week)

- **Objective**: Launch the system for public use.
- **Tasks**:
    - Set up the production environment.
    - Deploy the application and monitor initial performance.
- **Responsibilities**:
    - **DevOps Engineer**: Manage the deployment process and monitor system health.

6. Maintenance & Updates (Ongoing)

- **Objective**: Provide ongoing support and updates based on user feedback.
- **Tasks**:
    - Address any issues or bugs that arise post-launch.
    - Plan and implement feature updates and enhancements.
- **Responsibilities**:
    - **Support Team**: Handle user inquiries and troubleshoot issues.

    - **Development Team**: Work on iterative improvements and new features.

Timeline Overview

- **Week 1**: Project Planning
- **Weeks 2-3**: System Design
- **Weeks 4-9**: Development
- **Weeks 10-12**: Testing
- **Week 13**: Deployment
- **Week 14+**: Maintenance & Updates

Partitioning of Tasks & Team Member Responsibilities

- Lead Backend Developer
    - Responsible for server-side logic, database management, and API development.
- Lead Frontend Developer
    - In charge of developing the user interface, ensuring usability, and implementing client-side logic.
- Quality Assurance & DevOps
    - Focuses on testing, identifying, and fixing defects, as well as managing the deployment and operational aspects.