Software Design Specification

Movie Theatre Ticketing System (MTTS)

Team Members:

Julio Nevarez

Matthew Kloth

Doan Quoc Tien Nguyen

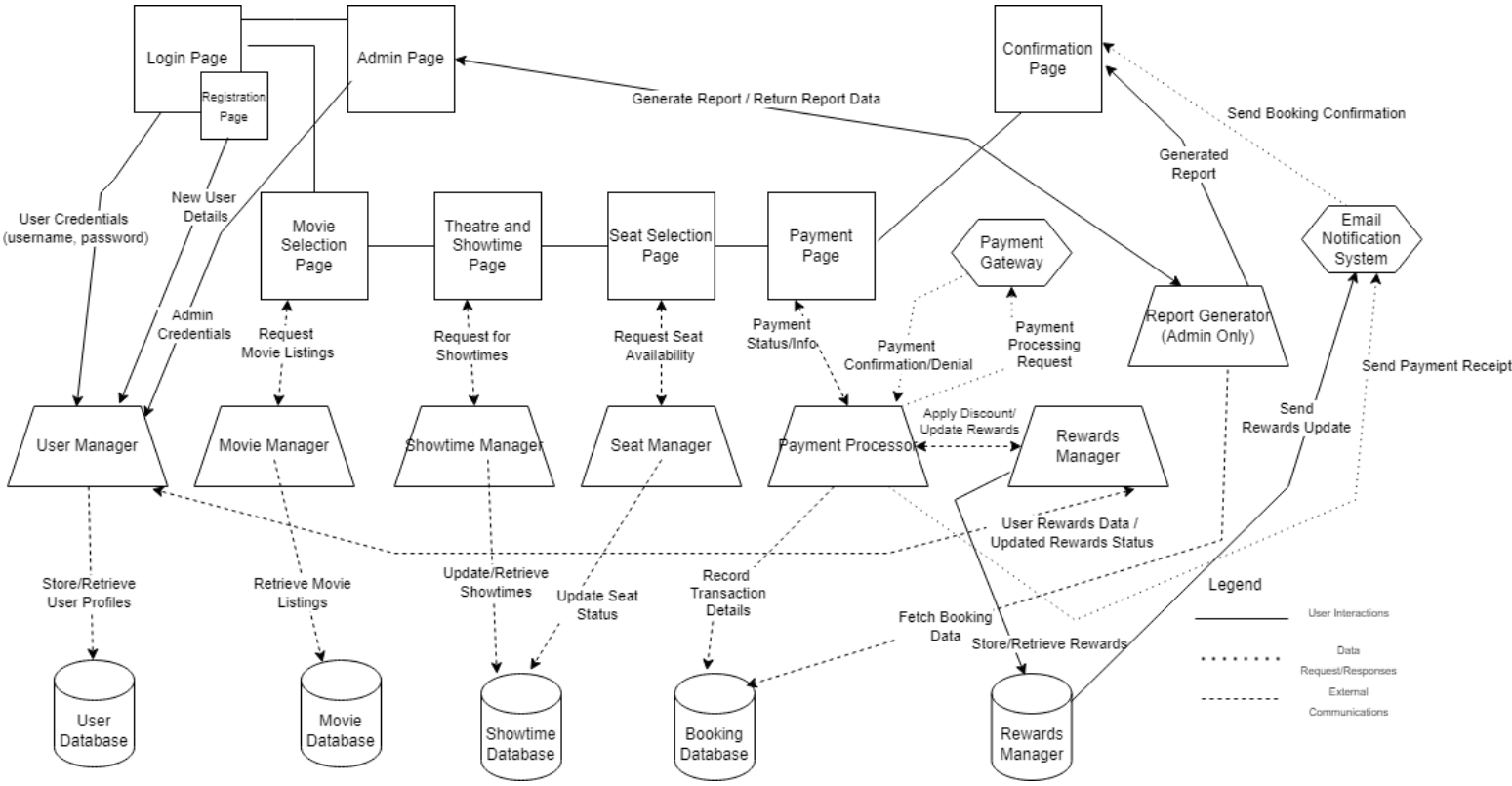
Jingyi Chen

System Description

The Movie Theatre Ticketing System (MTTS) is designed to optimize the management and sale of movie tickets, offer an intuitive interface for customers to acquire tickets, and incorporate administrative features for theatre personnel to oversee showtimes, access transaction history, and produce sales reports. The system seeks to provide a seamless experience for customers and theatre personnel, incorporating features like real-time seat availability, adaptable payment methods, and comprehensive show details.

Software Architecture Overview

SWA Diagram



Software Architecture Diagram Description

The software architecture for the **Movie Theatre Ticketing System (MTTS)** is structured into distinct layers that manage different functionalities, interactions, and data flow. The architecture can be divided into the following main components:

1. User Interface Layer

The User Interface Layer consists of various pages that provide an interactive experience for users and administrators. These pages capture user input, display system information, and initiate requests to the backend for data and operations. Pages include:

- **Login Page**: Handles user authentication, allowing existing users to log in and new users to navigate to the Registration Page.
- **Registration Page**: Manages new user registration by capturing details such as username, password, and contact information.
- **Movie Selection Page**: Displays available movies for users to choose from and requests movie listings from the backend.
- **Theatre and Showtime Page**: Shows detailed information about showtimes and theatre locations for selected movies.
- **Seat Selection Page**: Allows users to select available seats for a chosen showtime and sends seat selection data to the backend.
- **Payment Page**: Collects payment details, applies rewards or discounts, and initiates payment processing.
- **Confirmation Page**: Displays a summary of the completed booking and sends confirmation details to the user.

The Admin Page serves as a specialized interface for administrative users, providing access to generate reports, update showtimes, and manage the theatre's backend data.

2. Backend Application Layer

The Backend Application Layer is responsible for processing the logic, managing data transactions, and coordinating operations across different modules. This layer includes various **managers** that handle specific parts of the system:

- **User Manager**: The User Manager communicates the User Database for various operations:
    - **Login and Registration**: The User Manager sends a request to the User Database to verify login credentials or store new user details. If the credentials are valid, the database returns the user's profile, including rewards status.
    - **Profile Updates**: For any user profile changes (e.g., updating contact info or password), the User Manager updates the corresponding record in the User Database and retrieves the updated information for display.

- **Movie Manager**: The Movie Manager retrieves movie data from the Movie Database:
    - **Movie Listings**: When the Movie Selection Page requests movie details, the Movie Manager queries the Movie Database to fetch the latest movie titles, descriptions, genres, and ratings. The movie data is then sent back to the frontend for display.
    - **Movie Updates**: If new movies are added or existing ones are modified (by an admin, for example), the Movie Manager updates the Movie Database accordingly and ensures that all movie listings are current.
- **Showtime Manager**: The Showtime Manager interacts with the Showtime Database to manage showtime data:
    - **Retrieve Showtimes**: For a given movie, the Showtime Manager requests available showtimes and theatre locations from the Showtime Database and provides the results to the Theatre and Showtime Page.
    - **Update Showtimes**: In case of changes (like added showtimes or changes in availability), the Showtime Manager sends updated showtime data to the Showtime Database.
- **Seat Manager**: The Seat Manager is responsible for maintaining seat availability data in the Showtime Database:
    - **Seat Availability**: The Seat Manager queries the Showtime Database to check seat availability for a given showtime and updates the UI accordingly.
    - **Seat Reservation**: When a user reserves a seat, the Seat Manager updates the seat status in the Showtime Database, marking the seat as taken.
- **Payment Processor**: The Payment Processor interacts with both the Booking Database and the Rewards Manager during a transaction:
    - **Transaction Recording**: After a successful payment, the Payment Processor stores the transaction details (e.g., ticket information, payment method) in the Booking Database.
    - **Rewards Application**: If the user is eligible for rewards or discounts, the Payment Processor checks with the Rewards Manager to retrieve the user's rewards status and apply any applicable points or discounts before storing the final transaction data in the Booking Database.
- **Rewards Manager**: The Rewards Manager maintains and updates rewards data in the Rewards Database:

    - **Rewards Retrieval**: The Rewards Manager queries the Rewards Database to retrieve the user's current rewards balance and eligibility for discounts during login or checkout.
    - **Rewards Update**: After a successful transaction, the Rewards Manager updates the user's rewards points in the Rewards Database, adding any newly earned points based on the transaction value.
- **Report Generator (Admin Only)**: The Report Generator fetches data from the Booking Database:
    - **Report Generation**: When an admin requests a report, the Report Generator queries the Booking Database for relevant transaction and booking data. It then compiles this information into a report that is returned to the admin.

The **Email Notification System** is integrated into this layer as a notification handler, responsible for sending automated emails triggered by various backend managers, such as the Payment Processor and Rewards Manager.

3. Database Layer

The Database Layer stores all persistent data and ensures the integrity of the system's operations. Each database interacts with the backend managers to perform data-related operations. The following databases are integral to the system:

- **User Database**: Stores user profiles, credentials, purchase history, and rewards status. The User Manager performs Create, Read, Update, Delete operations on this database during login, registration, profile updates, and rewards retrieval.
- **Movie Database**: Contains movie listings, genres, and ratings. The Movie Manager retrieves this data to display available movies, and admins can update or add new movies via the Movie Manager.
- **Showtime Database**: Manages showtime schedules and seat availability. The Showtime Manager retrieves and updates showtime data, while the Seat Manager retrieves seat availability and updates the status of reserved or released seats.
- **Booking Database**: Keeps track of completed bookings, seat assignments, and transaction details. The Payment Processor stores each confirmed transaction, and the Report Generator retrieves booking data for report creation.
- **Rewards Database**: Stores user-specific rewards data, including points earned and discounts available. The Rewards Manager handles the retrieval and updating of this data, allowing users to accumulate and apply rewards during checkout.

4. External Systems Layer

The External Systems Layer integrates the MTTS with third-party services to enable additional functionality:

- **Payment Gateway**: An external service that validates and processes user payments. The Payment Processor communicates with the gateway to send payment requests and receive confirmations.
- **Email Notification System**: An external component responsible for sending transactional emails, such as booking confirmations, payment receipts, and rewards updates. The Payment Processor, Rewards Manager, and Confirmation Page send triggers to the Email Notification System to notify users of completed transactions, rewards status changes, and booking confirmations.

Each external system is represented as a separate component and uses secure communication channels to ensure data integrity and privacy.

5. Data Flow and Communication

The data flow in the system is facilitated using a combination of user interactions, data requests/responses, and external communications:

1. **User Interactions (Solid Lines)**: These represent direct interactions between the user and the UI pages as they navigate through the system.
2. **Data Requests/Responses (Dashed Lines)**: Backend managers communicate with databases and the UI components using these lines to handle data operations, including rewards management and seat availability checks.
3. **External Communications (Dotted Lines)**: Represent interactions with external systems, such as the Payment Gateway and Email Notification System, for secure payment processing and email notifications.

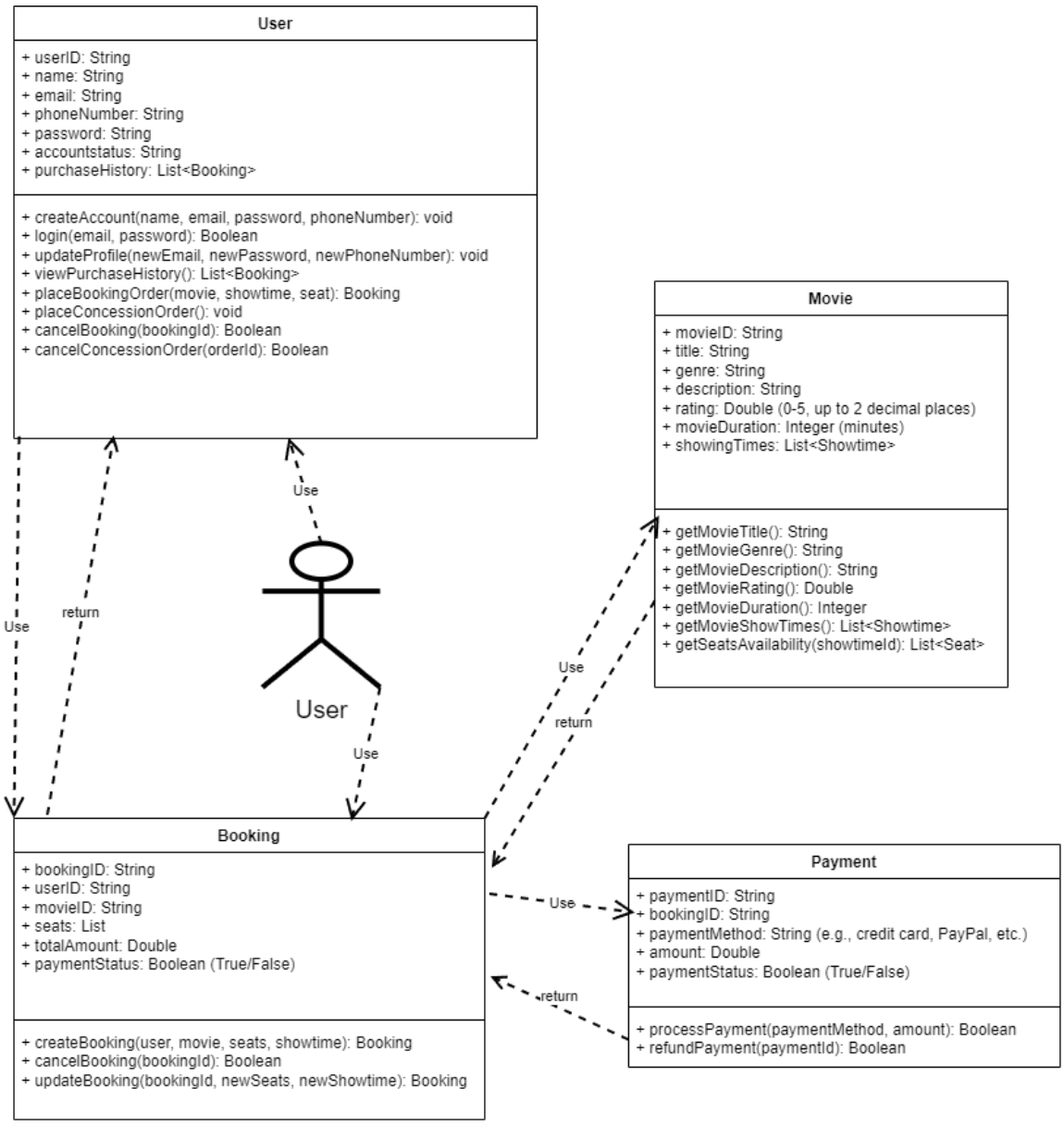
6. Architectural Considerations

- **Modularity**: Each component operates independently, allowing for easy updates or replacements.
- **Scalability**: The architecture can accommodate additional features or modifications without disrupting existing functionalities.
- **Security**: Sensitive data like user credentials and payment information is handled through secure communications with the User Manager and Payment Gateway.

- **Maintainability**: The clear separation of responsibilities between managers and UI components simplifies debugging and future extensions.

This architecture provides a robust foundation for the Movie Theatre Ticketing System, ensuring a seamless experience for both users and administrators while supporting advanced features like rewards and email notifications.

UML Class Diagram



Class Descriptions

- Class User: Represents a customer that uses the system to book movie tickets and manage their account. It lets the user manage their profile, interact with booking and rewards, and place concession orders.

- Attributes:

- userID: String - A unique identifier for each user.
- name: String - The user's full name (first and last name).
- email: String - The user's email address, used for account login and notifications.
- phoneNumber: String (Optional) - User's phone number for notifications and optional two-factor authentication.
- password: String - Password for the user’s account.
- accountStatus: String - Account status, which could be "active," "inactive," or "suspended."
- purchaseHistory: List<Booking> - List of bookings made by the user, including past ticket purchases and concessions.

- Operations:

createAccount(name: String, email: String, password: String, phoneNumber: String?): void

- Creates a new account with an optional phone number.

login(email: String, password: String): Boolean

- Authenticates the user by checking the email and password.

updateProfile(newEmail: String, newPassword: String, newPhoneNumber: String?): void

- Allows the user to update their account details such as email, password, and phone number.

viewPurchaseHistory(): List<Booking>

- Retrieves the list of the user's past bookings and purchases.

placeBookingOrder(movie: Movie, showtime: Showtime, seat: Seat): Booking

- Allows the user to place an order for a movie ticket with a selected showtime and seat.

placeConcessionOrder(orderItems: List<ConcessionItem>): void

- Allows the user to order items from the concession stand.

cancelBooking(bookingID: String): Boolean

- Cancels a booking if the movie has not started and the ticket has not been used.

cancelConcessionOrder(orderID: String): Boolean

- Cancels a concession order if the items have not been picked up.

- Class Movie: Represents the details of each individual movie. It contains information like the movie's title, genre, description, and ratings. It is linked to the showtimes and seat availability for the movie.

- Attributes:

- movieID: String - A unique identifier for each movie.
- title: String - The title of the movie.
- genre: String - The genre of the movie (e.g., Action, Comedy, Drama).
- description: String - A brief summary of the movie.
- rating: Double - Audience rating on a scale from 0 to 5 with two decimal places.
- movieDuration: Integer - The duration of the movie in minutes.
- showingTimes: List<Showtime> - A list of available showtimes for the movie.

- Operations:

getMovieTitle(): String

- Returns the title of the movie.

getMovieGenre(): String

- Returns the genre of the movie.

getMovieDescription(): String

- Returns the description of the movie.

getMovieRating(): Double

- Returns the audience rating for the movie.

getMovieDuration(): Integer

- Returns the duration of the movie in minutes.

getMovieShowTimes(): List<Showtime>

- Returns the list of available showtimes for the movie.

getSeatsAvailability(showtimeID: String): List<Seat>

- Checks and returns the availability of seats for the given showtime.

- Class Booking: Manages the ticket booking process for users. It records the user's selected movie, showtime, seats, and payment status. It also allows users to cancel or update their bookings.

- Attributes:

- bookingID: String - A unique identifier for each booking.
- userID: String - The unique identifier of the user who made the booking.
- movieID: String - The unique identifier of the movie being booked.
- seats: List<Seat> - The list of seats that the user has selected for the booking.
- totalAmount: Double - The total amount to be paid for the booking.
- paymentStatus: Boolean - Indicates whether the payment has been confirmed (True) or declined (False).

- Operations:

createBooking(user: User, movie: Movie, seats: List<Seat>, showtime: Showtime): Booking

- Creates a new booking for the user with the selected movie, seats, and showtime.

cancelBooking(bookingID: String): Boolean

- Cancels an existing booking and refunds the user if applicable.

updateBooking(bookingID: String, newSeats: List<Seat>, newShowtime: Showtime): Booking

- Allows the user to update the seats or showtime for an existing booking.

- Class Payment: Handles the payment process for bookings. It processes the user's payment and stores payment information, including the payment method and status on if the payment has been processed or not.

- Attributes:

- paymentID: String - A unique identifier for the payment.
- bookingID: String - The unique identifier for the booking associated with the payment.
- paymentMethod: String - The method used for payment (e.g., Credit Card, PayPal).
- amount: Double - The total amount paid for the booking.
- paymentStatus: Boolean - Indicates whether the payment has been processed successfully (True) or failed (False).

- Operations:

processPayment(paymentMethod: String, amount: Double): Boolean

Processes the payment and returns True if successful or False if declined.
refundPayment(paymentID: String): Boolean

Issues a refund to the user based on the provided paymentID and returns True if the refund is successful.

Development Plan and Timeline

Project Phases

The development of the Movie Theatre Ticketing System will be divided into several key phases.

Phases Overview

Phase 1: Requirements Gathering and Analysis

Duration: 1 week

Tasks:

Gather detailed requirements from stakeholders (users, administrators).

Define functional and non-functional requirements.

Document use cases for user interactions and administrative tasks.

Phase 2: System Design

Duration: 2 weeks

Tasks:

Design the overall architecture and create class diagrams.

Create wireframes for the user interface (UI) and administrative pages.

Define the database schema and data flow between components.

Review design with stakeholders for feedback and approval.

Phase 3: Development

Duration: 4 weeks

Tasks:

Week 1: Develop the User Interface Layer (Login, Registration, Movie Selection pages).

Week 2: Develop the Seat Selection, Payment, and Confirmation pages.

Week 3: Implement the Backend Application Layer (User Manager, Movie Manager, Showtime Manager).

Week 4: Implement Payment Processor and Rewards Manager; integrate external systems (Payment Gateway, Email Notification System).

Phase 4: Testing

Duration: 2 weeks

Tasks:

Unit testing for individual components and classes.

Integration testing to ensure different layers work together seamlessly.

User Acceptance Testing (UAT) with stakeholders to validate functionality.

Phase 5: Deployment

Duration: 1 week

Tasks:

Deploy the system to a staging environment for final review.

Address any issues found during the staging review.

Launch the system to production.

Phase 6: Documentation and Training

Duration: 1 week

Tasks:

Prepare user documentation and admin manuals.

Conduct training sessions for users and administrators.

Gather feedback for future improvements.

Team Member Responsibilities

1. **Julio Nevarez**
    a. **Backend Application Layer**:
        i. Responsible for developing the User Manager, Movie Manager, and Showtime Manager components.
        ii. Implements the logic for user authentication, movie listings, and showtime management.
        iii. Works on the integration with the Rewards Manager.
2. **Matthew Kloth**
    a. **Database Layer:**
        i. Oversee the design and management of the User Database, Movie Database, Showtime Database, and Booking Database.
        ii. Ensures data consistency, integrity, and proper communication between the backend and the databases.
        iii. Implements the data models and storage for user profiles, rewards, and booking transactions.
3. **Doan Quoc Tien Nguyen**
    a. **User Interface Layer:**
        i. Focuses on developing the Login Page, Registration Page, Movie Selection Page, Theatre and Showtime Page, and Seat Selection Page.
        ii. Ensures the UI is user-friendly, interactive, and properly connected to backend services.
        iii. Collaborate with Julio to ensure proper data flow between the UI and the backend.
4. **Jingyi Chen**
    a. **External Systems Layer:**
        i. Responsible for integrating the Payment Gateway and Email Notification System into the backend.
        ii. Implements the Payment Processor and ensures secure transaction handling and notification triggers.
        iii. Works on automating confirmation emails and
        iv. other user notifications.

Shared Responsibilities

Report Generator: Both Julio Nevarez and Matthew Kloth will collaborate on developing the admin-specific report generation functionality, ensuring it pulls the correct data from the backend and databases.

Testing and Debugging: All team members will contribute to the testing phase, ensuring that their respective layers communicate properly with one another and handle edge cases effectively.

Database Management Strategy

(1) **Database Type Choice:**

(i) **Primary Choice:** SQL Database

a. **Structured Data:** Given the structured nature of MTTS's data (e.g., user profiles, bookings, showtimes), an SQL relational database is optimal.

b. **ACID Compliance:** SQL databases offer ACID properties (Atomicity, Consistency, Isolation, Durability), which are crucial for the MTTS, ensuring each booking or payment transaction is processed accurately without risk of data corruption.

c. **Consistency Across Modules:** SQL enables consistent data management across multiple modules, essential for handling tasks such as seat selection, which requires real-time updates and availability checks.

(ii) **Alternative Consideration:** NoSQL Database

a. **Scalability and Flexibility:** NoSQL databases, like document or key-value stores, could offer advantages in handling unstructured data or scaling with fluctuating loads.

b. **Drawbacks:** However, NoSQL lacks the strong consistency and relational structure SQL offers, which is critical for transactional systems like MTTS. This makes SQL the preferred choice, given MTTS's current functional requirements.

(2) Data Segmentation and Organization:

(i) Logical Separation:

Data is segmented into specific databases based on functionality:

a. **User Database:** Stores user credentials and profile data, accessible only by the User Manager.

b. **Movie Database:** Holds movie listings, genres, and details, managed by the Movie Manager.

c. **Showtime Database:** Manages showtimes for each movie, organized by date and time for easy retrieval by the Showtime Manager.

d. **Booking Database:** Tracks seat selection, booking transactions, and records payment statuses, critical for Seat Manager and Payment Processor.

e. **Rewards Database:** Stores rewards data, tracking user points and discounts.

(ii) Normalization and Indexing:

a. Each database is normalized to minimize redundancy, improving storage and efficiency

b. Indexing is applied to frequently queried fields, like user ID, showtime ID, and booking ID, which enhances query performance for high-traffic operations.

(3) **Security, Backup and Recovery**:

(i) **Data Security**:

a. Sensitive data fields (e.g., payment details) are encrypted to protect user information.

b. Role-based access controls restrict data access to only authorized users, with different access levels for admins, regular users, and system managers.

(ii) **Backup and Recovery**:

a. Regular database backups ensure data is not lost in case of a system failure. Backups are automated and stored securely offsite.

b. The recovery protocol includes both manual and automated processes to restore data up to the last backup, minimizing downtime.

(4) **Concurrency and Transaction Management**:

(i) Transaction Management:

a. SQL's transaction handling ensures that seat selection and booking remain atomic, where each transaction completes fully or rolls back in case of errors.

b. **Two-Phase Locking**: Implemented to prevent issues like double-booking during concurrent seat reservations. The growing phase locks all necessary records, and the shrinking phase releases locks only after the transaction completes.

(ii) **Deadlock Detection**:

a. A deadlock detection mechanism monitors for potential deadlocks, such as two users trying to book the same seat, and releases locks if necessary to resolve the conflict.

(5) **Tradeoff Discussion**:

(i) **SQL vs. NoSQL**:

a. SQL databases are chosen due to their structured format and reliability for transactional data, which is essential for MTTS's booking and payment processes.

b. NoSQL could be advantageous if MTTS expands to handle high volumes of non-transactional data or needs more flexible schema management. For now, SQL meets the system's needs with minimal complexity and maximum consistency.

(ii) Multiple vs. Single Database:

a. Using multiple specialized databases enhances modularity and performance by allowing each module to access only its relevant data.

b. A single database could reduce management overhead but could introduce performance bottlenecks, especially in high-traffic scenarios.