Movie Theater Ticketing System

Software Design Specification

Robert Ashe

Jaden Perleoni

Matthew Press

Mohammed Almousawi

System Description:

The Movie Theater Ticketing System is a web-based application that is designed using a three-tier architecture made up of the Presentation Layer, Business Logic Layer, and Data Access Layer. Users will have various capabilities depending on their status as a customer, theater staff, or administrator.

The Presentation Layer will provide the user interface (UI) of the system. It is intended to be responsive to ensure compatibility with various devices like laptops, desktops, tablets, and smartphones. The UI will provide functionality for customers to browse movies, view showtimes, and purchase tickets, with provisions for theater staff and administrators to perform duties related to business operations.

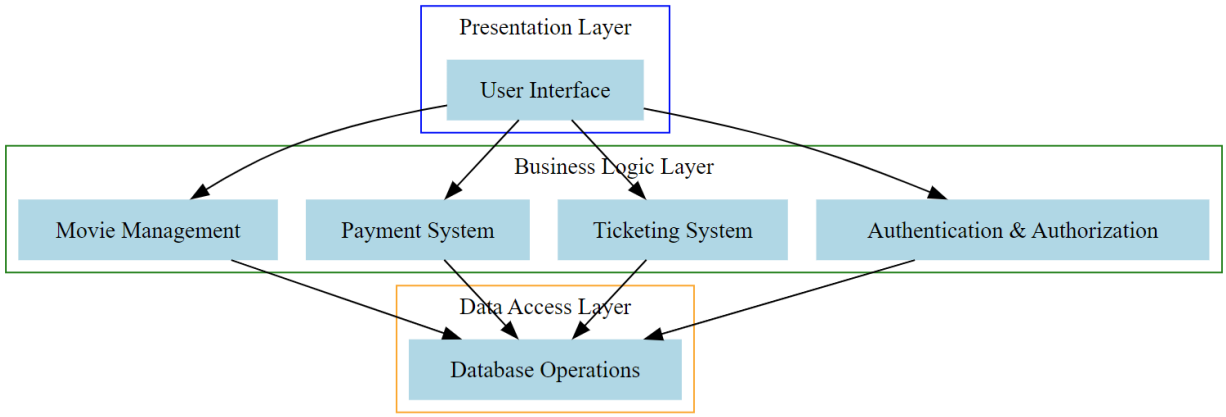
The Business Logic Layer will contain the core logic of the system. It manages operations such as user authentication, ticket purchasing, movie listing updates, and payment processing. The logic will ensure that customer data is handled securely and will comply with all current data integrity and security standards.

The Data Access Layer interacts with the database, handling all database operations. It provides consistent and up-to-date information to the Presentation Layer, as well as transaction history management.

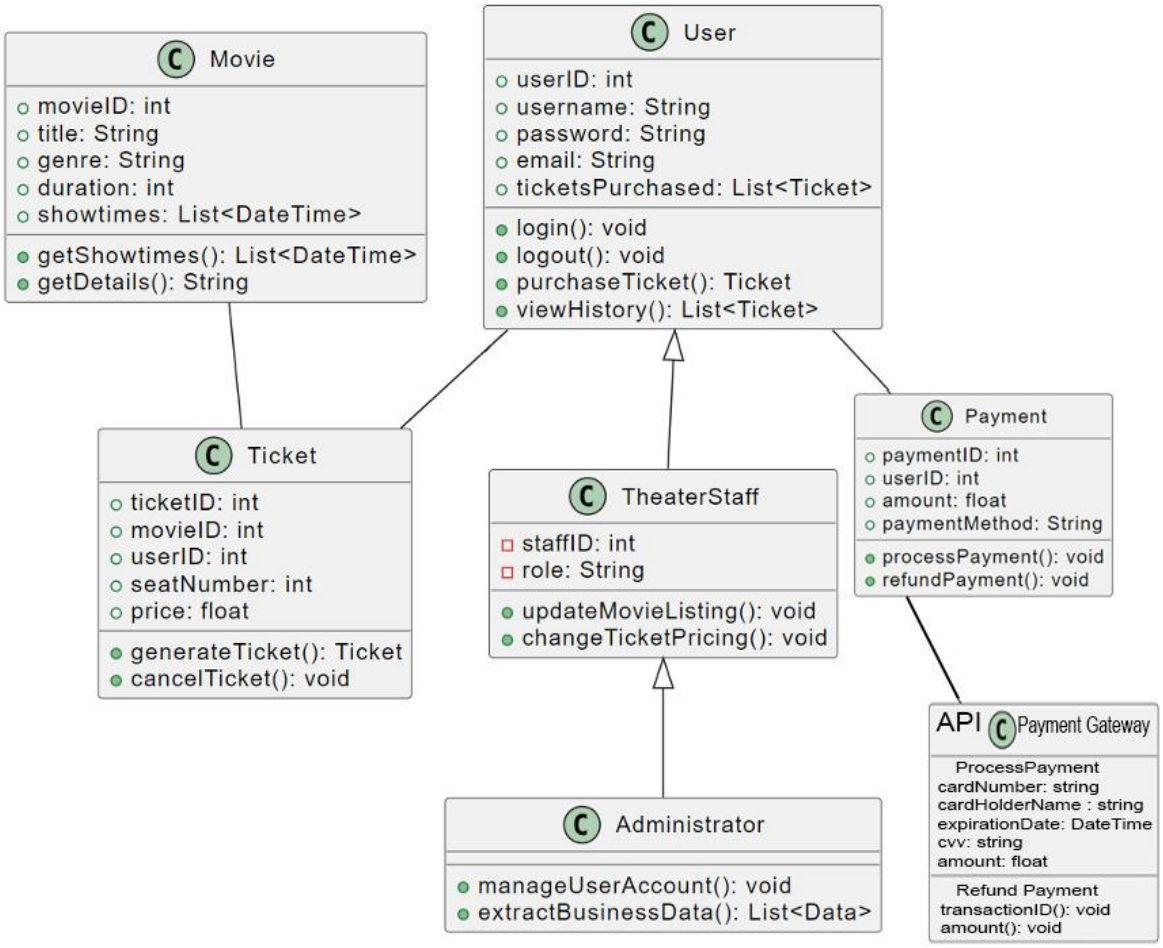
Software Architecture Overview:

The architecture of the system is comprised of three layers, the Presentation Layer, Business Logic Layer, and Data Access Layer.

Architectural Diagram:



UML Diagram:



Description of Classes:

User: Represents the end-users of the system. Users can login and logout, purchase tickets, and view their purchase history.

TheaterStaff: Inherits from the User class and represents an employee of the theater. In addition to the capabilities of the User class, they can update the movie listings and change ticket pricing.

Administrator: Inherits from the TheaterStaff class and represents the system administrator. They have all the capabilities of the User and TheaterStaff classes, with the additional ability to extract business data. Administrators are also responsible for managing user accounts.

Movie: Represents the movies listed by the system. Each movie has details like title, genre, duration, and showtimes.

Ticket: Represents the tickets purchased by users. Each ticket is associated with a Movie and a User.

Payment: Represents the payment transactions in the system. Each payment is associated with a User and has a specific transaction amount.

Description of Attributes:

User:

- **userID**: Unique identifier for the user. Automatically generated by the Business Logic Layer. Integer type. Public.
- **username**: User's chosen name for login. Chosen by user in the Presentation Layer, verified to be unique by the Business Logic Layer. String type. Public.
- **password**: User's password for authentication. Chosen by the user in the Presentation Layer, verified by the Business Logic Layer to include at least one capital letter, one number, and one special character. String type. Public.
- **email**: User's email address. Entered by the user in the Presentation Layer, confirmed to be authentic by the Business Logic Layer. String type. Public.
- **ticketsPurchased**: List of tickets that the user has purchased. Maintained by the business logic layer and presented to the user by the Presentation Layer. List<Ticket> type. Public.

TheaterStaff:

- **staffID**: Unique identifier for the employee of the movie theater. Automatically generated by the Business Logic Layer. Integer type. Private.
- **role**: Specific role for the staff member (Cashier, Manager, etc.). Created by the Administrator in the Presentation Layer. String type. Private.

Administrator:

- Inherits attributes from TheaterStaff.

Movie:

- **movieID**: Unique identifier for the movie. Automatically generated by the Business Logic Layer. Integer type. Public.
- **title**: Name of the movie. Updated by the Business Logic Layer and sourced from the Data Access Layer. String type. Public.
- **genre**: Category of the movie (Action, Drama, etc.). Updated by the Business Logic Layer and sourced from the Data Access Layer. String type. Public.
- **duration**: Length of the movie in minutes. Updated by the Business Logic Layer and sourced from the Data Access Layer. Integer type. Public.
- **showtimes**: List of showtimes for the movie. Updated by the Business Logic Layer and sourced from the Data Access Layer. List<DateTime> type. Public.

Ticket:

- **ticketID**: Unique identifier for the ticket. Automatically generated by the Business Logic Layer. Integer type. Public.
- **movieID**: Unique identifier for the movie. Automatically generated by the Business Logic Layer. Integer type. Public.
- **userID**: Unique identifier for the user. Automatically generated by the Business Logic Layer. Integer type. Public.
- **seatNumber**: Seat number for the ticket. Automatically generated by the Business Logic Layer. Integer type. Public.
- price: Cost of the ticket. Updated by the Business Logic Layer. Float type. Public.

Payment:

- **paymentID**: Unique identifier for the payment. Automatically generated by the Business Logic Layer. Integer type. Public.
- **userID**: Unique identifier for the user. Automatically generated by the Business Logic Layer. Integer type. Public.
- **amount**: Amount of the transaction. Calculated by the Business Logic Layer. Float type. Public.
- **paymentMethod**: Method used to pay the transaction amount (Credit Card, Debit Card, etc.). Chosen by the user in the Presentation Layer and verified by the Business Logic Layer. String type. Public.

Description of Operations:

User:

- **login(username)**: String, password: String), Boolean: Allows the User to log into the system. Takes String parameters for username and password. Returns 'true' if the login was successful, 'false' otherwise.
- **logout(), void**: Allows the User to log out of the system. Does not return any value.
- **purchaseTicket(movieID: Integer, showtime: DateTime, seatNumber: Integer), Ticket**: Allows the user to purchase a ticket for a specific movie, showtime, and seat. Returns the purchased 'Ticket' object.
- **viewHistory(), List<Ticket>**: Allows the User to view their purchase history. Returns a list of 'Ticket' objects.

TheaterStaff:

- **updateMovieListing(), Boolean**: Allows the theater staff to update the movie listings using the Movie Listing Database. Returns 'true' if the listing was updated successfully, 'false' otherwise.
- **changeTicketPricing(ticketID): Integer, newPrice: float), Boolean**: Allows the staff to modify ticket pricing using the ticket's ID and a new price. Returns 'true' if the change was successful, 'false' otherwise.

Administrator:

- **manageUserAccount(userID)**: Integer, action: String), Boolean: Allows the administrator to manage user accounts using the user's ID and a specific action (Delete User, Add User, Add TheaterStaff, etc.). Returns 'true' if the action was successful, 'false' otherwise.
- **extractBusinessData(dateRange: DateTime), List<Data>**: Allows the administrator to extract business-related data for a specific date range. Returns a list of 'Data' objects.

Movie:

- **getShowtimes(movieID: Integer), List<DateTime>**: Retrieves the list of showtimes for a specific movie using the movie's ID. Returns a list of 'DateTime' objects.
- **getDetails(movieID: Integer), String**: Retrieves the detailed information about a specific movie using the movie's ID. Returns a String that contains the details related to the movie.

Ticket:

- **generateTicket(userID): Integer, movieID: Integer, seatNumber: Integer), Ticket**: Allows the User to purchase a ticket for a specific movie, showtime, and seat. Returns a 'Ticket' object that represents the purchased ticket.
- **cancelTicket(ticketID: Integer), Boolean**: Allows the User to cancel a purchased ticket using the ticket ID. Returns 'true' if the cancellation was successful, 'false' otherwise.

Payment:

- **processPayment(userID: Integer, amount): Float, paymentMethod: String), Boolean**: Processes the payment transaction for a specific User, amount, and payment method. Returns 'true' if the payment was successful, 'false' otherwise.
- **refundPayment(paymentID: Integer, Boolean)**: Refunds a payment using the payment ID. Returns 'true' if the refund was successful, 'false' otherwise.

Payment Gateway API:

Description of Operations: this module is to integrate the system with any merchant such as Visa, master card, American Express, etc. This method is used to initiate a payment transaction with the payment gateway.

Create two Booleans, ProcessPayment and RefundPayment.

Parameters:

- **cardNumber (string)**: The card number associated with the user's payment method.
- **cardHolderName (string)**: The name of the cardholder as it appears on the payment card.
- **expirationDate (DateTime)**: The expiration date of the payment card.
- **cvv (string)**: The Card Verification Value, typically a security code on the back of the card.
- **amount (float)**: The amount of money to be charged or authorized for the payment transaction.

Return Value:

- **bool**: The method returns a Boolean value (true or false) to indicate whether the payment processing was successful (true) or not (false).

RefundPayment Method:

- **Purpose**: This method is used to initiate a refund transaction with the payment gateway. It is responsible for processing a refund for a previous payment transaction, using the transaction ID and the refund amount.

Parameters:

- **transactionID (string)**: A unique identifier or reference associated with the original payment transaction that needs to be refunded.
- **amount (float)**: The amount of money to be refunded.

Return Value:

- **bool**: The method returns a Boolean value (true or false) to indicate whether the refund processing was successful (true) or not (false).

Development Plan and Timeline:

Task	Description	Estimated Date	Team Member
Create Movie class	Create the movie class and design its functions. Write logic for the getShowtimes() and cancelTicket() functions. The class must also define variables movieID, title, genre, duration, and showtimes.	10/18/2023	Robert Ashe
Create TheaterStaff class	Create a class for the theater staff and its functions. Define variables staffID and role. Write logic for functions: updateMovieListing() and changeTicketPricing().	11/4/2023	Matthew Press
Create User class	Create the user class and its functions. Variables userID, username, password, email, and ticketsPurchased must be defined in this class. Write the logic for the login(), logout(), purchaseTicket(), and viewHistory() functions.	10/31/2023	Jaden Perleoni
Create Administrator class	Create the administrator class and its two functions: manageUserAccount() and extractBusinessData().	10/16/2023	Mohammed Almousawi
Create Ticket class	Create the movie class, its variables, and its functions. Define variables movieID, title, genre, duration, and showtimes. Design the functions getShowtimes() and getDetails().	11/09/2023	Matthew Press
Create Payment class	Create the payment class, its variables, and its functions. Define variables paymentID, userID, amount, and paymentMethod. Write the logic for functions processPayment() and refundPayment().	11/01/2023	Jaden Perleoni

Verification Test Plan:

The Verification Test Plan aims to ensure that the Movie Theater Ticketing System meets the design specifications and functions correctly. Each of the following test cases are designed to target specific features of the system and verify desired functionality.

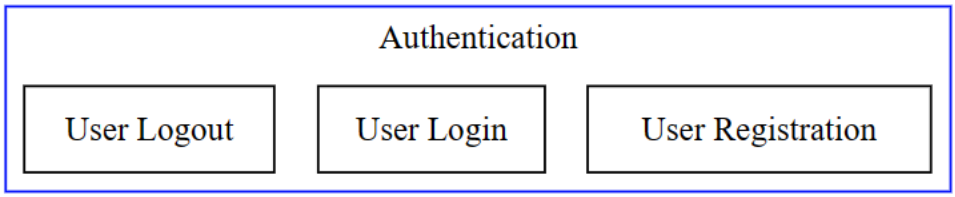
- **Test Case: Authentication_0 (User Registration)**
    - **Features Tested:** User registration functionality.
    - **Test Vectors:** Valid username, password, and email address.
    - **Coverage:** This test ensures that a new user can register a new account successfully and that the system can handle and store the new user data correctly.

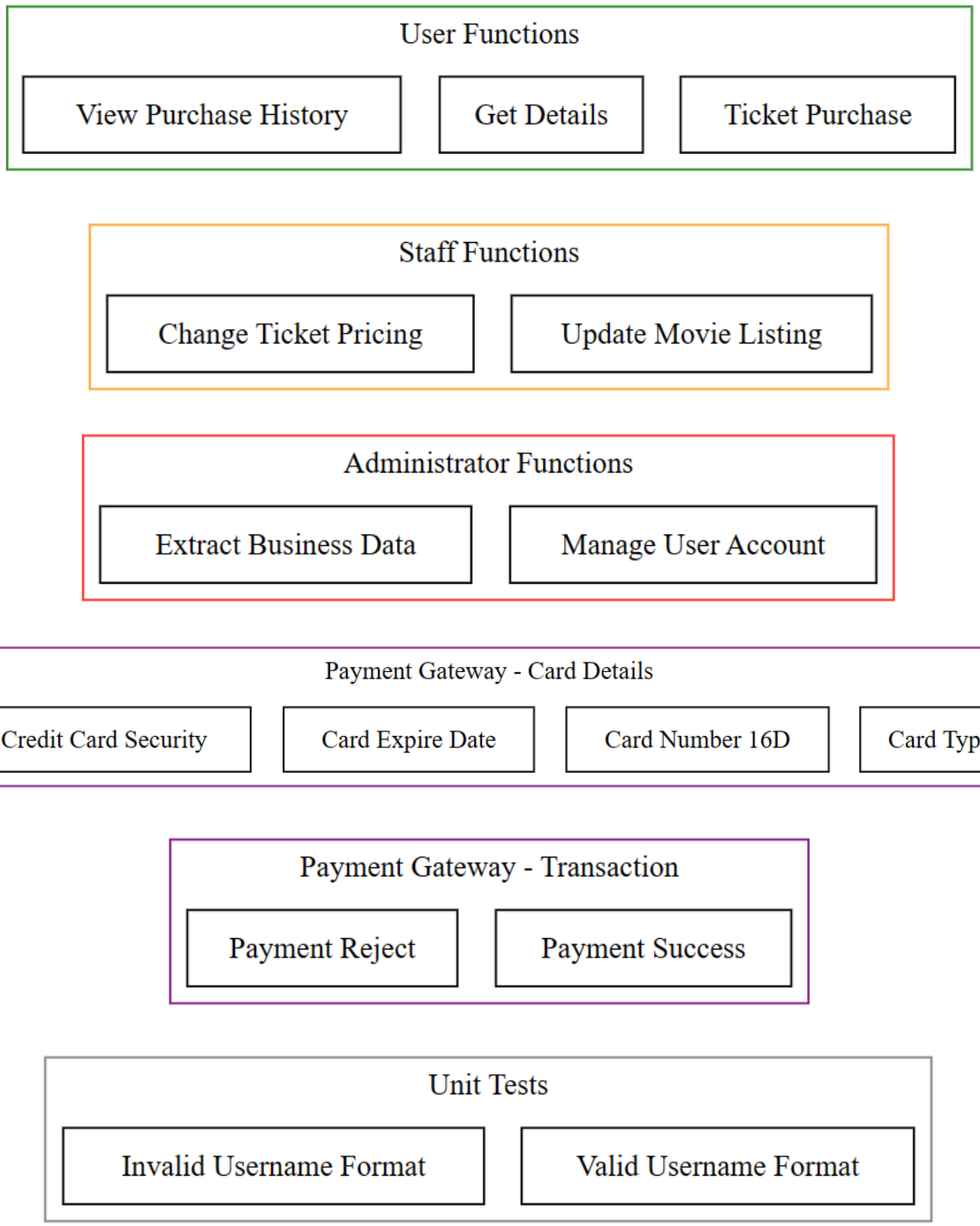
1. **Test Case: Authentication_1 (User Authentication)**
    - **Features Tested:** User login functionality.
    - **Test Vectors:** Valid username and password for an existing account.
    - **Coverage:** This test verifies that a registered user can log into the system using their credentials.
2. **Test Case: Authentication_2 (User Logout)**
    - **Features Tested:** User logout functionality.
    - **Test Vectors:** Active user session.
    - **Coverage:** This test ensures that a user that is logged in to the system can log out successfully and end their current session.
3. **Test Case: User_Functions_0 (Ticket Purchase)**
    - **Features Tested:** Ticket purchasing process.
    - **Test Vectors:** Desired movie, showtime, and seat selection.
    - **Coverage:** This test checks the system's capability to handle ticket purchases and update the user's purchase history.
4. **Test Case: User_Functions_1 (Get Details)**
    - **Features Tested:** Retrieval of detailed movie information.
    - **Test Vectors:** Selected movie title.
    - **Coverage:** This test ensures that the system can display the detailed information related to a selected movie title.
5. **Test Case: User_Functions_2 (View Purchase History)**
    - **Features Tested:** Viewing of user's purchase history.
    - **Test Vectors:** User profile that has previous ticket purchases.
    - **Coverage:** This test ensures that the system can display detailed information, including cast, director, and runtime.
6. **Test Case: Staff_Functions_0 (Update Movie Listing)**
    - **Features Tested:** Update movie listings.
    - **Test Vectors:** Selected movie and updated details.
    - **Coverage:** This test ensures that theater staff has the ability to manually update movie details.
7. **Test Case: Staff_Functions_1 (Change Ticket Pricing)**

- **Features Tested:** Changing ticket pricing.
- **Test Vectors:** Selected ticket and new price.
- **Coverage:** This test verifies the ability of theater staff to change ticket pricing.

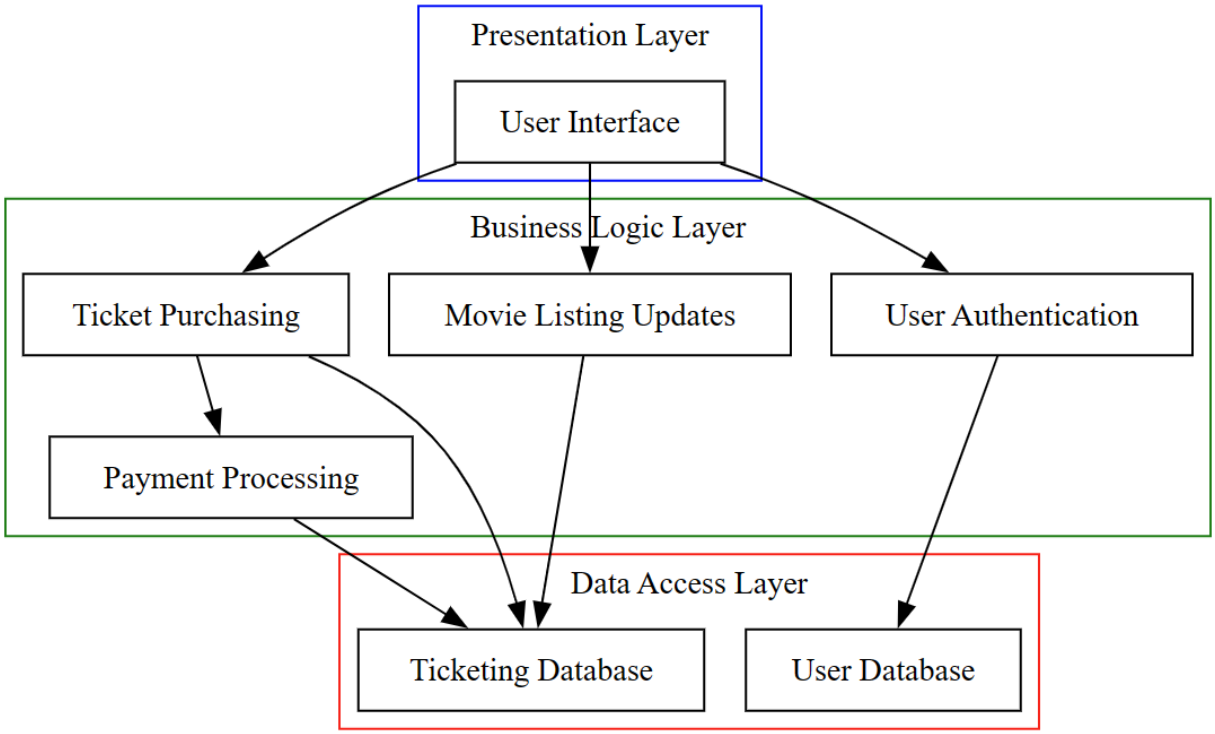
8. **Test Case: Administrator_Functions_0 (Manage User Account)**
    - **Features Tested:** Managing user accounts.
    - **Test Vectors:** Selected user account.
    - **Coverage:** This test ensures that administrators can manage user accounts.
9. **Test Case: Administrator_Functions_1 (Extract Business Data)**
    - **Features Tested:** Extraction of business data.
    - **Test Vectors:** Specified date range.
    - **Coverage:** This test verifies that administrators have the ability to extract business data for a specified date range.
10. **Test Case: Movie_Functions_0 (Get Show Times)**
    - **Features Tested:** Retrieval of movie showtimes.
    - **Test Vectors:** Selected movie title.
    - **Coverage:** This test ensures that the system can display the showtimes for a selected movie title.
11. **Test Case: Ticket_Functions_0 (Generate Ticket)**
    - **Features Tested:** Generation of a ticket after a purchase has been made.
    - **Test Vectors:** Valid user ID, selected movie, showtime, and seat number.
    - **Coverage:** This test checks the system's capability to generate a ticket after a user has made a purchase, ensuring the ticket reflects the correct details.
12. **Test Case: Ticket_Functions_1 (Cancel Ticket)**
    - **Features Tested:** Cancellation of a purchased ticket.
    - **Test Vectors:** User ID and ticket ID.
    - **Coverage:** This test verifies that a user can cancel a previously purchased ticket and that the system updates the available seats accordingly.
13. **Test Case: Payment_GW_Functions (Validate Card and Information)**
    - **Features Tested:** Verify all card information are valid with merchant though PaymentGW API.
    - **Test Vectors:** Validate card name, type, Expire date & CVV.
    - **Coverage:** this test will make sure the credit card information is valid, proof of payment or rejection with transaction ID.

Test Plan Diagram:




User Functions
View Purchase History
Get Details
Ticket Purchase
Staff Functions
Change Ticket Pricing
Update Movie Listing
Administrator Functions
Extract Business Data
Manage User Account
Payment Gateway - Card Details
Credit Card Security
Card Expire Date
Card Number 16D
Card Type
Payment Gateway - Transaction
Payment Reject
Payment Success
Unit Tests
Invalid Username Format
Valid Username Format

Software Architecture Diagram:



Data Management Strategy:

Due to the structured nature of the Movie Theater Ticketing System (MTTS) and the relationships between the various entities that make up the system, a relational database system is the most suitable choice. The data that will be used by the MTTS will be maintained in two separate SQL databases. One of the databases will be known as the User Database, and its purpose is to contain the information related to the users of the system. The second database will be used to hold information related to the business operations of the MTTS, known as the Ticketing Database.

The number of databases has been chosen to segregate user data from ticketing data. This ensures the MTTS system will be able to meet security requirements and also allows for independent scalability of each database. The ability for the system to be able to handle growth in both its users and its movie selection will be crucial to its maintenance in the future.

Here is a breakdown of how the data will be split between the two databases:

User Database:

The user database will contain the personal details of the various users of the system and is comprised of a single entity named User. Here is a description of the User entity and its attributes.

User: Represents a user of the Movie Theater Ticketing System.

- **userID:** Primary key of the User table. A unique number that identifies each user. Stored as a string.
- **Username:** Chosen when the user creates their account and verified to be unique by the system before initialization. Stored as a string.
- **Password:** String chosen by the user that will be used for authentication to allow the user access to the MTTS system. Stored in the database as a salted hash of the original string.
- **Email:** Email address of the user. Stored as a string.
- **TicketsPurchased:** Number of tickets purchased by the user. Stored as an integer.
- **Role:** Type of user of the system. Can be one of three sting values: “customer”, “theaterStaff”, or “administrator”.

Ticketing Database:

The Ticketing Database will contain the information related to the movies listed by the system, the theaters that these movies are shown in, the tickets that permit a user to attend a showing, and the payments used to purchase a ticket. The Ticketing Database is comprised of the entities named Move, Movie Details, Theater, Ticket, and Payment. Following is a description of the entities and the attributes they contain.

Movie: Represents a movie listed in the Movie Theater Ticketing System.

- **movieID:** Primary key of the Movie table. A unique number that identifies each movie. Stored as a string.
- **movieDetailsID:** Foreign key to the Movie Details table. A unique number that identifies the details of each movie. Stored as a sting.
- **Showtimes:** DateTime objects that represent the different showtimes for a movie. Stored as DateTime objects.
- **theaterID:** Foreign key to the Theater table. A unique number that identifies a theater. In relation to a Movie, this identifies which theater the movie is being shown in. Stored as a string.

Movie Details: Contains detailed information about a movie.

- **movieDetailsID:** Primary key of the Movie Details table. A unique identifier that identifies the details for a movie. Stored as a string.
- **movieID:** Foreign key to the Movie table. A unique number that identifies each movie. In relation to Movie Details, this identifies the movie that the details belong to. Stored as a string.
- **Title:** Title of the movie. Stored as a string.

- **Genre:** Genre of the movie. Stored as a string.
- **Actors:** A list of actors that play in the movie. Stored as a string.
- **Director:** The director of the movie. Stored as a string.
- **Year:** The year of the movie. Stored as a string.
- **Rating:** The motion picture content rating of the movie. Can take on values "G – General Audience", "PG – Parental Guidance Suggested", "PG-13 – Parents Strongly Cautioned", "R – Restricted", or "NC-17 – No One 17 and Under Admitted". Stored as a string.
- **Description:** A brief synopsis of the movie. Stored as a string.
- **Duration:** The duration of the movie in minutes. Stored as an integer.
- **ratingIMDB:** The IMDB rating of the movie. Stored as a float.

Theater: Information about the theater where movies are screened.

- **theaterID:** Primary key to the Theater table. A unique number that identifies a theater. Stored as a string.
- **numSeats:** The number of seats available in a theater. Stored as an integer.
- **seatLayout:** The layout of the seating in the theater. Can take on values "Standard", "Auditorium", or "Cinema". Stored as a string.
- **handicapAccess:** Indicates if a theater is equipped with handicapped access and seating. Stored as a Boolean.

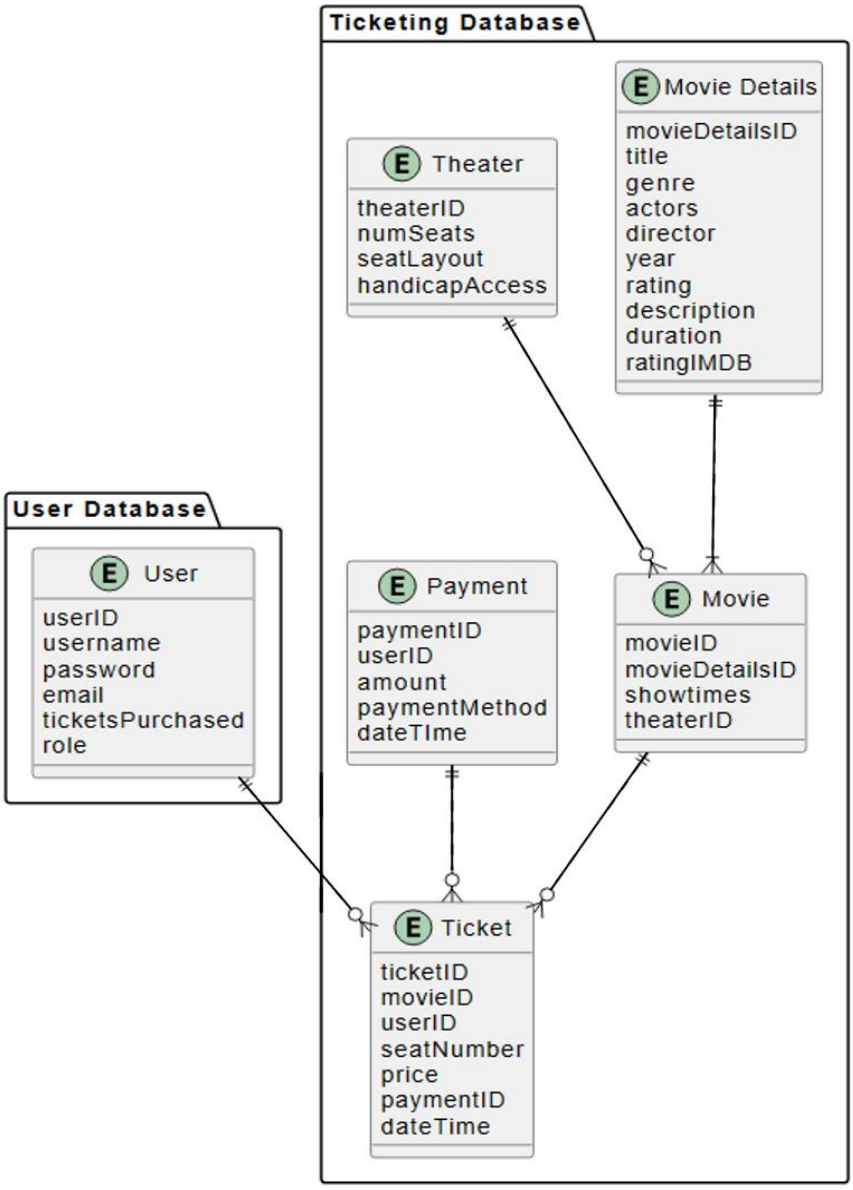
Ticket: Details of tickets purchased by users.

- **ticketID:** Primary key to the Ticket table. A unique number that identifies a ticket. Stored as a string.
- **movieID:** Foreign key to the Movie table. A unique number that identifies a movie. In relation to the Ticket table, this identifies the movie that is associated with a ticket. Stored as a string.
- **userID:** Foreign key to the User table. A unique number that identifies each user. In relation to the Ticket table, this identifies which user purchased a ticket. Stored as a string.
- **seatNumber:** The seat number associated with a ticket. Stored as an integer.
- **Price:** The price of the ticket in dollars and cents. Stored as a float.
- **paymentID:** Foreign key to the Payment table. A unique number that identifies a payment. In relation to the Ticket table, this identifies the payment details that were used to purchase the ticket. Stored as a string.
- **dateTime:** Time and date of the movie associated with the ticket. Stored as a string.

Payment: Stores payment details related to ticket purchases.

- **paymentID:** Primary key to the Payment table. A unique number that identifies a payment. Stored as a string.
- **userID:** Foreign key to the User table. A unique number that identifies each user. In relation to the Payment table, this identifies which user is associated with a payment. Stored as a string.
- **Amount:** The amount of the payment in dollars and cents. Stored as a float.
- **paymentMethod:** The method of payment that was used in the purchase. Can take on values "Visa", "Mastercard", "American Express", "Administrator Account", or "Cash". Stored as a string.
- **dateTime:** The time and date that the payment was made. Stored as a string.`

Database Schema



Possible Alternatives to Current Design

If the data being stored for the Movie Theater Ticketing system were less structured or if there was a need to provide more flexibility for future expansion of the system's capabilities, for instance if the system were to be expanded to handle user reviews or other social features, a NoSQL database could be considered. Since there is not currently a plan to expand the system in this way, the current system's need to maintain relational integrity demonstrates the relational database system (RDBMS) remains the most suitable choice. Another alternative to the current design could be to use a single database for both user and ticketing data. However, this approach could pose security risks and would not be as scalable as the current design. By keeping the User Database and the Ticketing Database separate from each other, the current design allows for the User Database to grow at a rate independent of the Ticketing Database, providing scalability and easier management of the system into the future.