Movie Ticket System

Date: 05/03/2024

Team Member: Lucas Weinstein, Jacob Silva, Elias Mapendo

System Description:

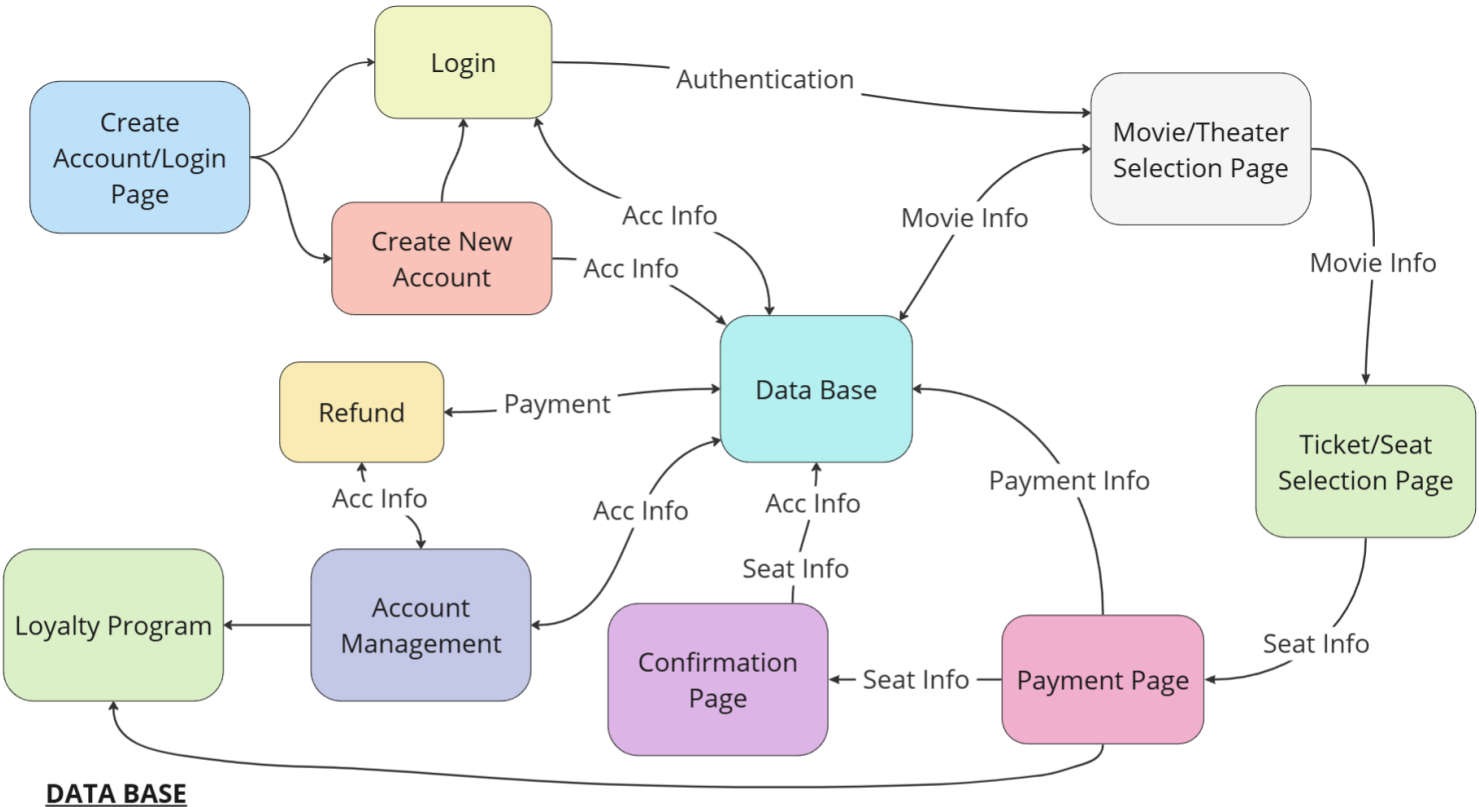
Overview: A user-friendly web-based program to purchase tickets at available theaters. Users create an account or log in if they already have an account, to use our program to search for a newly released movie to find theaters in their area showing film. Similarly, the user may search for a theater in their area, which will then show the movies at that theater. Once the user finds a film at a particular theater and at a particular time and with their seat(s), the user may purchase the ticket(s), which will be sent to their email. Optionally, the users' tickets accumulate to receive a free ticket (after their 10th ticket purchase). Once a free ticket is redeemed, the count is reset to zero unless there is another circumstance[1][2]. The user may also change account information such as password, phone number, and username.

[1]Ex: User has 9 tickets already purchased and they purchase 2 more. 1 of the tickets is paid for and then one ticket is redeemed as free, and the counter is reset to zero.

[2]Ex: The user has 9 tickets and purchased 3 more. 1 of the tickets is paid for and then one ticket is redeemed as free, and the counter is set to 1.

Software Architecture Overview:

Architectural diagram:



DATA BASE

Movie Info:

movies, times, locations, prices, seats, theaters, theaterTypes, theater details

Account Info:

username, password, email, phoneNumber, loyalty points, purchase history

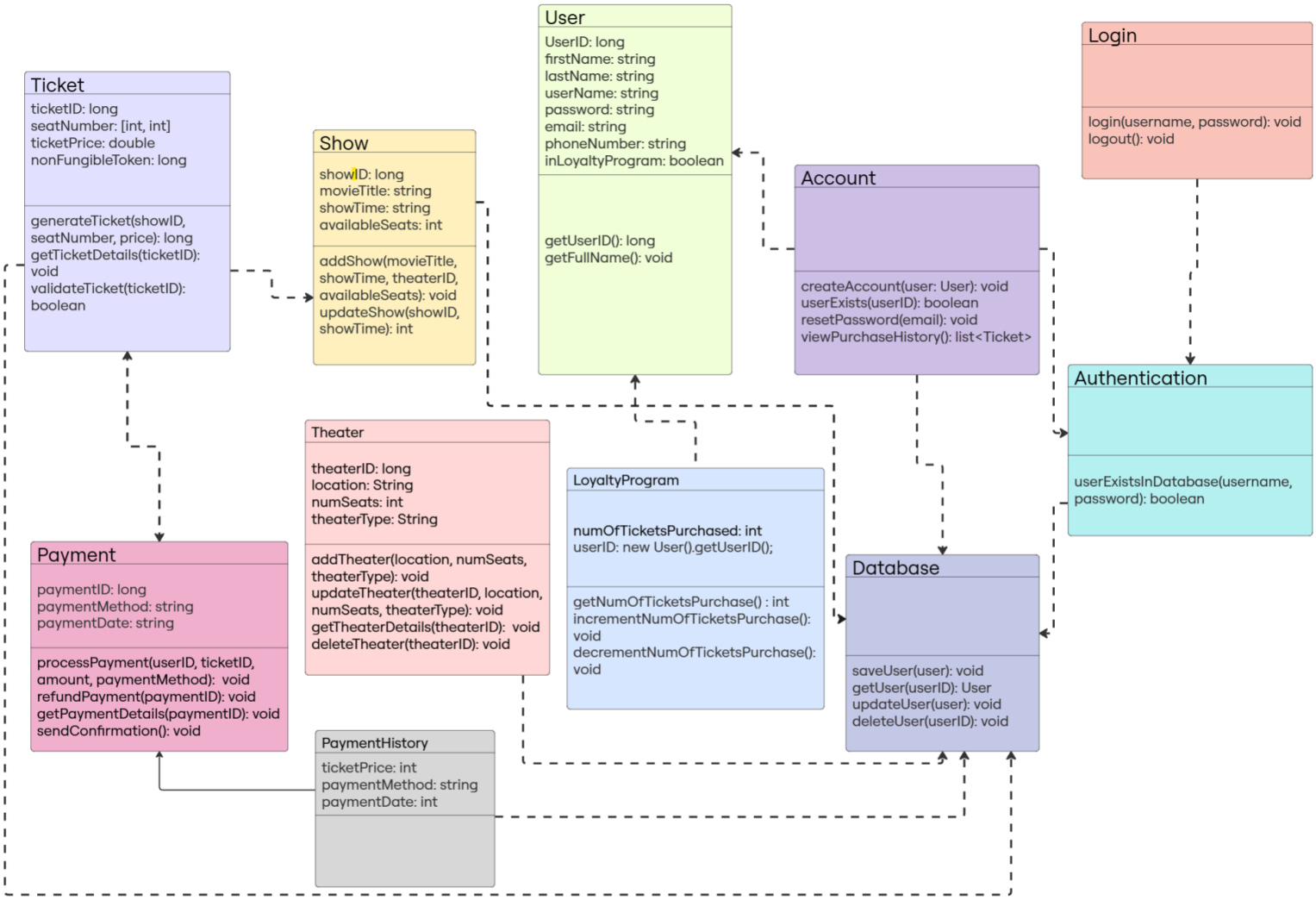
Seat Info:

Seat number, movie, price, theater

Payment Info:

paymentID, date, paymentType, price, user_id

UML Class Diagram:



Description of classes:

User:

Description of attributes:

- UserID: long - Unique identifier for each user.
- firstName: string - user's first name.
- lastName: string - user's last name.
- userName: string - The user's chosen username.
- password: string - The user's chosen password.
- email: string - The user's email address.
- phoneNumber: string - The user's phone number.

- inLoyaltyProgram: boolean - Holds if the user is in the program. Once a user is created, an option is given to sign up for the loyalty program - is automatically false.

Description of operations:

- getUserID(): int - returns the ID of the user.
- getFullName(): - returns the full name of the user.

Theater: Features and characteristics of a specified theater.

Description of attributes:

- theaterID: long - unique identifier for a movie theater
- location: String - address, city, state, zip code
- numSeats: int - number of seats in specific theater
- theaterType: String - what type of theater it is (3D, Imax, etc)

Description of operations:

- addTheater(location, numSeats, theaterType): void - add a movie theater into the available search.
- updateTheater(theaterID, location, numSeats, theaterType): void - updates theaterID and/or location and/or numSeats and/or theaterType.
- getTheaterDetails(theaterID): void - retrieves details of theater (location, numSeats, theaterType) for theater (theaterID).
- deleteTheater(theaterID): void - deletes theater (theaterID).

Show: Features showtime specifics such as time and title:

Description of attributes:

- showID: long - Unique identifier for each show.
- movieTitle: string - Title of the movie being shown.
- showTime: string - Date and time of the show.
- availableSeats: int - Number of available seats for the show.

Description of operations:

- addShow(movieTitle, showTime, theaterID, availableSeats): void - Adds a new show with details of title, time, and seat availability to the system.
- updateShow(showID, showTime): int - Updates the details of an existing show, including the time of showing, the seats available, and the title if needed.

- getShowDetails(showID): void - Retrieves details of a specific show, including movie title, show time, and available seats.
- deleteShow(showID): void - Deletes a show and its data from the system.

Ticket:

Description of attributes:

- ticketID: long - Unique identifier for each ticket.
- seatNumber: [int, int] - The assigned seat number for the ticket.
- ticketPrice: double - The specified price for the ticket.
- nonFungibleToken: long - create new code as a second id for fraud detection.

Description of operations:

- generateTicket(showID, seatNumber, price): long - generates a new ticket for a specified show.
- getTicketDetails(ticketID): void - Retrieves the specific details of ticket, including seat location, ticket price, date, and the non-fungible token.
- validateTicket(ticketID): boolean - validates the authenticity of ticket.
- refundTicket(ticketID): void - calls to payment class to refund the purchase of the specified ticket.

Payment:

Description of attributes:

- paymentID: long - payment identifier
- paymentMethod: string - type of payment (VISA, MASTERCARD, etc.)
- paymentDate: string - date of payment

Description of operations:

- processPayment(userID, ticketID, amount, paymentMethod): void - Processes a payment for a ticket purchase to make sure methods are accurate
- refundPayment(paymentID): void - Refunds a payment transaction
- getPaymentDetails(paymentID): void - Retrieves details of a specific payment transaction.
- sendConfirmation(): void - sends an email with details of purchase and ticket to use.

Loyalty Program: A loyalty program for customers to redeem 1 free ticket after 10 tickets have been purchased.

Description of attributes:

- numOfTicketsPurchased: int - 0-9 number that displays tickets purchased (if num>9, counter resets, and a free ticket is given to the customer).
- userID: new User().getUserID - retrieve userID from User class.

Description of operations:

- getNumTicketsPurchase(): int - returns number of tickets purchased.
- incrementNumTicketsPurchase(): void - increments tickets purchased by how many tickets were purchased.
- decrementNumOfTicketsPurchase(): void - decrements tickets to 0 if num>9.
- redeemFreeTicket(): void - redeems free redeem once 10 tickets have been purchased.

Account:

Description of attributes:

- N/A

Description of operations:

- createAccount(User user): void - registers a new user
- resetPassword(string email): void - sends a password reset link to the user' email
- vuser'srchaseHistory(): list<Ticket> - returns list of payment history

Login:

Description of attributes:

- N/A

Description of operations:

- login(): void - authenticates user
- logout(): void - logs out the current user

Authentication:

Description of attributes:

- N/A

Description of operations:

- userExistsInDataBase(): boolean - returns true of user already exists in the database

Database:

Description of attributes:

- N/A

Description of operations:

- saveUser(user): void - saves user into database
- getUser(): long - returns userID
- updateUser(user): void - updates user details
- deleteUser(userID): void - deletes the user from the database

PaymentHistory:

Description of attributes:

- ticketPrice: int - the price of the ticket purchased.
- paymentMethod: string - defines if payment method was VISA, MASTERCARD, APPLEPAY, etc.
- paymentDate: int - date the ticket was purchased.

Description of operations:

- N/A

Development plan and timeline:

Partitioning of tasks:

All team members: worked together to discuss and collaborate
Jacob: How to deal with payments and payment history, as well as how the purchases are changed to our loyalty program.
Elias: How to create/login a user, as well as how their data is sent to our database and authenticated.
Lucas: How to find a movie/theater, and confirm ticket selection.

Team Member Responsibility:

All team members:

- Collaborate to gather requirements and create the initial system design.
- Identify and fix certain errors that would occur when the system is in use to ensure the system is reliable.
- Document the system functionalities, user manuals, and technical specifications.
- Ensure documentation is clear and comprehensive.

Jacob:

- Dealt with formulating the payment history process, and linking it to the loyalty program to complete its functionality entirely.
- Developed the loyalty program system.

Elias:

- How the database schema supports all functionalities.
- And its integration of the other objects

Lucas:

- Dealt with the transferring of information and the functionality between show, ticket, movie, and the data base.
- Fodatabasethe timeline of the user using the product.
    - Specifically log in to create an account to then purchase movie

Teathe m Member	**Responsibility**
Lucas Weinstein	● Completed Show, Ticket, and Theater in UML. ● Complete User, Theater, and Show in description of classes. ● Completed user interface, login, account creation, and database in the architectural diagram.
Elias	● Completed DataBase, Authentication, Account, and Login in UML. ● Completed account management refund, and database in the architectural diagram.
Jacob	● Completed Payment, Payment History, Loyalty Program, and User in UML ● Completed overview ● Completed ticket payment, loyalty program, and refunds.