SDS: Test Plan

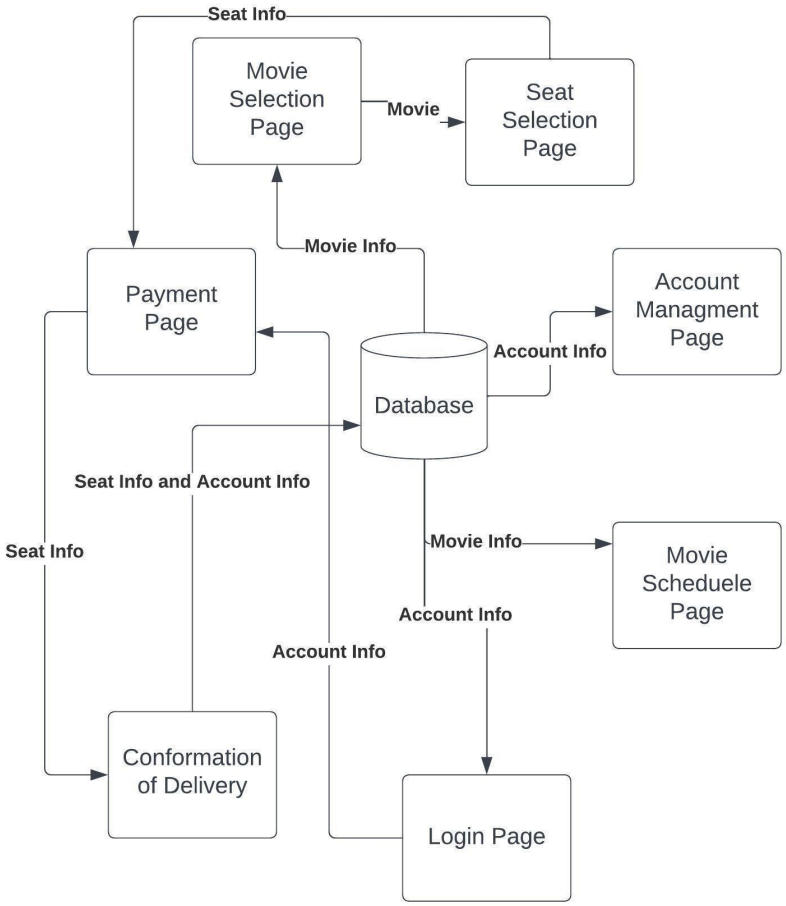
By: Luis Suarez, Hailun Xu

System Description

The system to be constructed will be a ticketing service for an AMC theater as well as providing a subscription to a premium viewing service. It will allow customers to buy tickets online and earn reward points as well as allowing administrators to manage customer profiles. It will have verification systems to prevent scalping and botting.

Software Architecture Overview

The architecture diagram of all major components is the following:



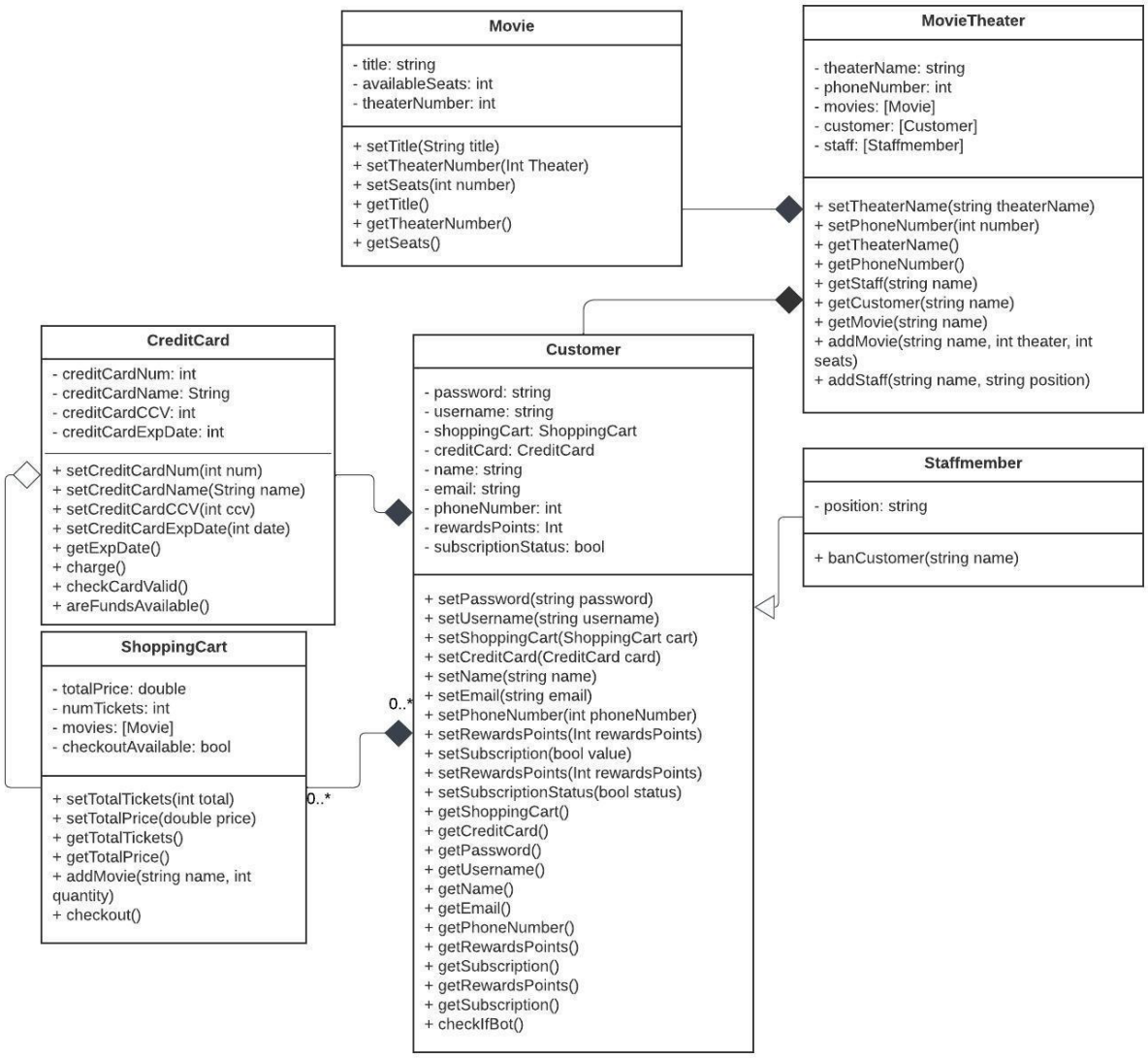
<showtimes, locations, seats, prices> = movie info

<username, password, authorization loyalty points, payment information, purchase history> = account info.

To purchase a ticket for viewing, the user must first have an account. They begin on the Movie Selection Page, whose info is updated from the database. Once they

choose a movie, they are brought to a Seat Selection Page where they can choose a seat if any are available. Then they are brought to a Payment Page. If they are not logged in already, they are taken to a Login Page to do so. Once they have decided on the number of tickets to purchase and decide to check out, the system will go to the user's credit card info in the database and query their bank to make sure it is a valid transaction. If so, the user gets a confirmed delivery form the Conformation of Delivery Page.
The user also has the option of managing their account on their Account Management Page and seeing the schedule of films available from the Movie Schedule Page.

UML Class Diagram



Classes, Operations and Attributes

MovieTheater [class]: This is the movie theater. It has a string name, an integer phone number, a collection of user accounts including staff, customer, and a a collection of movies containing Movie objects to show. All collections are Dictionaries and not arrays, for easy lookup.

- setTheaterName(string theaterName) sets the name of the theater to the inputted string.

- setPhoneNumber(int number) sets the theater's number using the inputted integer with no spaces, dashes or symbols, IE 9191119999.
- setMovies(Movie[] movies) updates what movies are available to be shown and takes in an array of movies.
- for each of these methods, there is a corresponding get() method.

Movie [class]: This is a movie that will be shown. It cannot exist without a theater. It has a title, which is a string, an integer array of showtimes, and an int that tells you which theater it is playing in.

- setTitle(string title) updates the title with the inputted string.
- setShowTimes(int[] showTimes) takes in an integer array with the showtimes. Times are formatted in 24-hour format. An example would be [0800,1200,1700] for 8am, 12pm, and 5pm respectively.
- setTheaterNumber(int theaterNumber) takes in an integer denoting the theater number. - for each of these methods, there is a corresponding get() method.

TicketInfo [class]: This class updates ticket information for a particular Movie. It shows the price as a double and the seats available as a collection of integers, with each seat labeled by a number.

- setPrice(double Price) updates the price with the inputted double.
- setSeats(Int[] Seats) updates the array of available seats. An unavailable seat will be set to "0" in the array.
- getPrice() returns the current price.
- getSeats() loops through the array and checks if the index contains a 0. Any seat that is not a 0 gets added to a new array of integers and the new array is returned.

Account [class]: This is a user account. It cannot exist without a theater. It has the person's real name, username, password, email address which are all strings, and subscription status which is a boolean. It has their phone number and reward point count which are integers. It also has a shopping cart that they can use to buy movie tickets or an AMC subscription.

- setName(string name) takes in a string as the person's name.
- setPassword(string password) takes in a string as a password. Any password less than 8 characters is not accepted.
- setUsername(string username) takes in a string as a username. it must be a minimum of 4 characters in length.
- setEmail(string email) holds the user's email address.
- setPhoneNumber(int number) sets the user's number using the inputted integer with no spaces, dashes or symbols, IE 9191119999.
- setRewardsPoints(int rewardsPoints) updates the number of rewards points this user has.
- setSubscriptionStatus(bool status) determines whether the user is subscribed to the AMC service.

- setShoppingCart(ShoppingCart cart) updates the user's shopping cart to the inputted cart.
- for each of these methods, there is a corresponding get() method.

Admin [class]: This is an administrator. it extends the Account class. Administrators can attend screenings and earn rewards like regular customers, but they have the ability to affect other Accounts.

- changeAccountStatus(Account account) allows an administrator to change some aspect of a customer's Account, like subscription status or password, or to ban certain accounts from buying tickets if necessary. returns void.

ShoppingCart [class]: This is a shopping cart. It can't exist without an account. It keeps track of the total price of tickets or a subscription using totalPrice, which is a double, and the number of tickets being bought via the integer numTickets. It also deters bots and scalpers. The boolean checkoutAvailable can disable checkout if needed.

- setTotalTickets(int total) updates the cart based on the number of tickets being bought, which can't exceed 20. Calls TicketInfo.getSeats() when the ShoppingCart is created to update how many tickets are available.
- setTotalPrice(double price) updates the amount due based on the number of tickets, and whether a subscription is being bought.
- for each of these methods, there is a corresponding get() method.
- checkIfBot() runs an algorithm at checkout time that determines if the user is a bot. If true, checkoutAvailable is set to false and checkout is unavailable. Then returns boolean true.
- checkout() calls checkCardValid() on the user's CreditCard and checkIfBot(). If the Account's card is valid and the user is not a bot, call charge() on the CreditCard. Otherwise checkout is disabled. returns void.

CreditCard [class]: The card on file for a user. It can't exist without an Account. It has methods to set card info but no public methods to get any of it, to protect the user. The credit card number, expiration date, and CVV number are all stored as integers. The name is stored as a string.

- setCreditCardNum(int num) inputs the 16-digit credit card number as a single integer with no spaces. It will only accept an integer of this length.
- setCreditCardName(String name) stores the full name of the user in a string.
- setCreditCardCVV(int ccv) stores the 3-digit CVV as an integer.
- setCreditCardExpDate(int date) The date is stored as a four-digit number, MMYY.
- checkCardValid() contacts the user's bank to see if sufficient balance is available and if card details are correct. Returns boolean false if there isn't.

Verification Test Plan

Test Plan #1

For the following Unit tests, we will test by passing in a string and an int as arguments to see how the program will respond to a valid vs an invalid argument

Unit Test #1
setCreditCardNum(8845949372);
assert(CreditCardNum == 8845949372);

This unit test should test to see if a valid int is correctly passed and assigned.

Unit Test #2
setCreditCardNum(ABCDEFT123);
assert(CreditCardNum != ABCDEFT123);

This unit test should test to see if an invalid argument is not passed and assigned.

Integration Test #1
CreditCard creditCard = new CreditCard;
creditCard.setCreditCardNum(8845949372);
creditCard.setCreditCardName(Marisol);
creditCard.setCreditCardCCV(557);
creditCard.setCreditCardExpDate(032023);
assert(creditCard.getCreditCardNum == 8845949372);

This integration test should test to see if the function setCreditCardNum() will be able to integrate into the rest of the CreditCard class by passing in all valid arguments into the functions.

Integration Test #2

CreditCard creditCard = new CreditCard;
creditCard.setCreditCardNum(8845949372);
creditCard.setCreditCardName(“Marisol”);
creditCard.setCreditCardCCV(557);
creditCard.setCreditCardExpDate(032023);
assert(creditCard.checkCardValid());
assert(creditCard.getCreditCardName() == “Marisol”);
assert(creditCard.getCreditCardCVV() == 557);

This integration test should test to see if the function setCreditCardNum() will be able to integrate into the rest of the CreditCard class by passing in all valid arguments into the functions.

System Test #1

```
Account account = new Account();
account.setPassword(password);
account.setUsername(username);
account.setName(“Marisol”);
account.setEmail(Marisol@gmail.com);
account.setPhoneNumber(6196667777);
account.setShoppingCart(ShoppingCart shoppingCart = new ShoppingCart());

//Assume a MovieTheater object called “cinema” has been declared and has had all it’s
//variables instantiated
cinema.getMovies();

shoppingCart.setTotalTickets(1)
shoppingCart.setTotalTicketPrice(5.00)
shoppingCart.checkout();

CreditCard creditCard = new CreditCard;
creditCard.setCreditCardNum(8845949372);
creditCard.setCreditCardName(Marisol);
creditCard.setCreditCardCCV(557);
creditCard.setCreditCardExpDate(032023);


shoppingCart.checkout();
shoppingCart.charge()
```

assert(shoppingCart.empty());
assert(creditCard.getCreditCardNum == 8845949372);

This system test will check to see if a user will be able to setup an account, shop for a film, and purchase said film.

—----------

Test Set Two: Invalid Tests: In the second test set we will attempt to buy tickets but with no money in our bank account. In this case, we should not be able to check out.

Unit Test: This will be a check to see if the movie theater and user accounts are created properly.

- MovieTheater theater;
- theater.setTheaterName(“AMC Fashion Valley”);
- theater.addMovie(“Indiana Jones”, theater 1, 100 seats);
- theater.addMovie(“Terminator 2”, theater 2, 100 seats);
- theater.addMovie(“The Goonies”, theater 3, 0 seats);
- assert(theater.getMovie(“The Goonies”).getSeats() == 100);

In this unit test, we create a MovieTheater object and give it a name. Then we add 3 movies. Suppose the title “The Goonies” is added with the number of available seats accidentally set to 0. We assert that all newly added movies must have the right number of seats available.

Functional/Integration Test:

- theater.addStaff(“Tony L”, manager);
- theater.addStaff(“Mary J”);
- assert(theater.getCustomer(“Mary J”).checkIfBot() == false)
- theater.getStaff(“Tony L”).banCustomer(theater.getCustomer(“Mary J”));
- assert(theater.getCustomer(“Mary J”) != null);

Suppose a manager, Tony L, can only ban a customer if they turn out to be a bot. A customer named Mary J opens an account and they are not a bot. Tony tries to ban Mary J’s account but we assert that the account is not null because it can’t be banned.

System Test: This will be an end to end test that uses some elements from the previous tests as well as some new elements. The customer will be using a valid but *expired* card so the transaction should not complete.

add movies, add customer account, and add card:

- theater.addMovie(“Indiana Jones”, theater 1, 100 seats);
- theater.addMovie(“Terminator 2”, theater 2, 100 seats);
- theater.addMovie(“The Goonies”, theater 3, 100 seats);
- theater.addCustomer(“Jessie C”);
- theater.getCustomer(“Jessie C”).addCreditCard(new CreditCard);

add movies to the shopping cart. assert that enough seats are available and less than 20 tickets are being bought:

- int requestedSeats = theater.getCustomer(“Jessie C”).getShoppingCart()
  .getTotalTickets()
- int seatsAvailable = theater.getMovies(“Indiana Jones”).getSeats();
- assert(seatsAvailable < requestedSeats)
- assert(requestedSeats >20)

add order to shopping cart and adjust available seats:

- theater.getCustomer(“Jessie C”).getShoppingCart()
  .addMovie(“Indiana Jones”, requestedSeats);
- theater.getMovies(“Indiana Jones”).setNumSeats(seatsAvailable-requestedSeats);

validate credit card. It should return false that funds are available:

- CreditCard card = theater.getCustomer(“Jessie C”).getCreditCard();
- assert(card.checkCardValid());
- assert(card.getExpDate() >= current date);
- assert(!areFundsAvailable());