Potential test cases

- Keep track of capacity so a theater is not overcrowded
- Make sure age restrictions are met.

- Booking:
- When someone clicks on a specific ticket type, make sure the ticket type is what they clicked on
- When they cancel an order, make sure all the tickets are voided
- Make sure the ticket counter integers keep track of the number of tickets of their type
- Make sure the booking number is not repeated for two separate bookings

- Customer account:
- Make sure that the email of the user is a valid email address
- Make sure that two passwords are never identical
- Make sure that two usernames are never identical
- When log out is called, take user back to home page

Log in:

- Prereq: browser is launched
- The user should be asked to input their username and password
- If the password or user name is wrong they should not be able to have further access and should see the login page again with a note that says “try again”, a button that says “make an account”, and a button that says “forgot password”
- If the user's password is right they should be directed to the home page again but logged in.

Showing:

- Make sure that a theater does not have any showings that overlap the same date and time

Test Sets

Verifying and maintaining account information

- This is testing the validity of our service in dealing with valid emails
- **Verify email**
    - Check the input of an email that is entered by a user while logging in. Then check if the string input has at least an @, and a . with text between and after them. This will make sure that the email is formatted as an email
- **Check username**
    - Make sure the entered user username does not match with any other username from any other user. If the username is unique, then the username will be accepted and applied to the customer account. Otherwise, an error will be displayed and the username will not be set onto the account.
- **Check password**

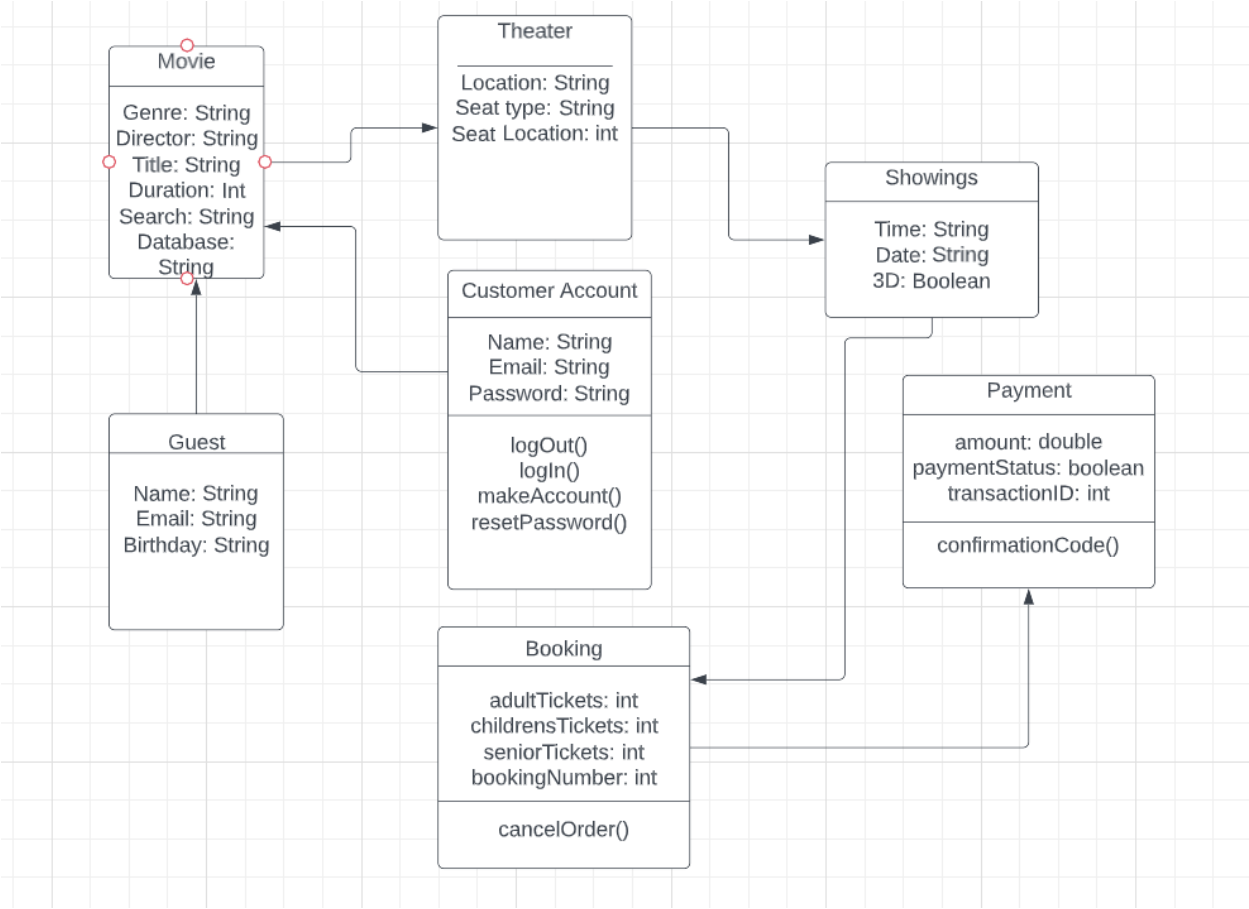
- Make sure the entered user password does not match with any other password from any other user. If the password is unique, then the password will be accepted and applied to the customer account. Otherwise, an error will be displayed and the password will not be set onto the account.

Searching and maintaining the movie database:

- This tests the ability of our database and logistics to store, sort, and retrieve movies from our database to the user
- Upload 500 movies to a database
- **Make sure the movie still has seats, if not then notify the user.**
    - A variable will be created to hold the number of seats available in a theater for a specific showing. Each time a ticket is purchased the system will decrease the number of seats available for the showing. When no more seats are available users will no longer be able to purchase a ticket for that showing.
    - If a user tries to order a ticket for a show with no more seats available a string will be displayed that says, “There are no more available seats for this showing”.
- **When someone looks for a movie, make sure they are shown the movie that they search for.**
    - The way I would go about testing this is by typing in a movie that is in the database and if the option to order a ticket for that movie pops up then the test passes. However, if the movie does not pop up then there is a mistake in the program.
- **A user should not be able to purchase a ticket to a movie that is no longer playing at a theater.**
    - I would look up a movie that used to be shown at the theater but no longer is. Since the movie is no longer showing in the theater a user should not be able to order tickets for it. The way I would make sure a user cannot buy a ticket to an old movie is by completely deleting the movie from the system's database once it is no longer playing at the theater.

- **When a user clicks on a movie that they searched for they should be able to choose how many tickets they want to order, what showing they want to view, and a seat location.**
    - In order to test this I would run through the process of ordering tickets to a movie through the search bar and I would enter a random number of tickets that I would like to purchase then I would pick a theater and seat location. I would do this numerous times with different variations of the number of tickets, theater locations, and seat locations. If any of these tests fail then I would debug until I could find out whether the issue was occurring with the ticket number, if so which number, the theater location, and which location was the issue, or the seat location and which location was causing an error.

UML Class Diagram:



Movie: this class stores the attributes of a movie so that it may be identifiable to other classes and assigned to guests.

- **Genre:** a string attribute that contains the general category of the movie.
- **Director:** a string attribute that contains the person who directed the movie.
- **Title:** a string attribute that contains the title of the movie, this is most commonly used to identify a movie.
- **Duration:** a string attribute that contains the time duration of the movie.
- **Search:** a sting attribute that holds the movie that the user wants to look for.
- **Database:** a string attribute with all of the movies available.

Guest: this class stores the attributes of a single guest.

- **Name:** a string attribute that contains the individual name of the guest.
- **Email:** a string attribute that contains the email address of the guest.
- **Birthday:** a string attribute that displays the yearly birthday of the guest.

Theater: this class stores the attributes of a theater, a place where movies are displayed and guests watch them.

- **Location:** a string attribute that contains the physical location of the theater.
- **Seat type (aka standard or luxury):** a string attribute that contains the type of seats in the theater.

Showings: this class stores the attributes of the showings of movies, or when the movies are played.

- **Time:** a string attribute that contains the time that a showing takes place.
- **Date:** a string attribute that contains the date that a showing takes place.
- **3D:** a boolean attribute that identifies whether the showing is in 3D or not.

Customer Account: this class stores the attributes of an account tied to a customer.

- **Name:** a string attribute that contains the individual name of the owner of the account.
- **Email:** a string attribute that contains the email address of the owner of the account.
- **Password:** a string attribute that contains the "password" or key that identifies the owner of the account.
    - **makeAccount():** an operation that allows the user to create an account if they do not already have one.
    - **resetPassword():** an operation that allows the user to "reset" or change the password attribute by stating their old password to confirm they are the owners of the account, and then passing their new password.
    - **logOut():** an operation that allows the user to "log out" or stop actively using the account. that they are tied to by stating that they want to log out.
    - **logIn():** an operation that allows the user to "log in" or enter their account. Once the user has logged in they can purchase a ticket.

Booking: this class stores the booking of one or multiple tickets to a movie.

- **adultTickets:** an integer value that contains the number of tickets that belong to adults.
- **childrens_Tickets:** an integer value that contains the number of tickets that belong to children.
- **seniorTickets:** an integer value that contains the number of tickets that belong to seniors.
- **bookingNumber:** an integer value that contains the identification number of the booking.
    - **cancelOrder():** an operation that allows the order of the booking to be canceled.

Payment: this class stores the attributes of a payment for a booking to a movie.

- Amount: a double value that stores the dollar amount that will be owed in the payment.
- paymentStatus: a boolean attribute that identifies whether or not the payment has been completed.
- transactionID: an integer value that contains the identification number of the transaction.
    - confirmationCode() an operation that confirms to the paying and receiving parties that the payment has been completed.