The main functionality of the reservation system comes down to 4 basic fundamental use cases: Customer Registration, Customer Login, Staff Registration, and Staff Login. As soon as the program starts to run, the index html page is called and outputs 6 links to different use cases. 4 are the fundamentals, and the other 2 are for searching up flight information and for searching a flight's status. These 2 use cases are independent of login.

**Searching Flights without Being Logged In**

Structure:

```
    The user inputs the departing and arrival airport, and the departure date.
    They also may input the return date for searching for flights that are round trips.
    If the user does not input a return date:
        A search in the flight table is made using equivalent checks to the airports and departure date inputted
        If results are found:
            Each equivalent flight's information is outputted to the user
    If the user inputs a return date:
        The same check is made to the flight table with the addition of return date as an equivalent check
        If results are found:
            Each flight's information is outputted
```

**Searching Flight Status without Being Logged In**

Structure:

```
    The user inputs an airline, a flight number, and a departure date.
    It searches the flight table to find an equivalent row to all of the above
    If there is a result:
        The flight status is outputted
```

**Registering Customer**

Structure:

```
    The user inputs the username and password they wish to set up.
    The login ID is created using a string of randomly chosen digits
    The login_cust table is checked to see if the user already is registered
    If the user is not registered:
        A new row is created in the customer table with the user's information
        A new row is created in the login_cust table with the user's information
        The password before being inserted into the login_cust table is hashed using bcrypt
```

**Registering Staff Member**

Structure:

```
    The user inputs their username, password, airline, and email.
    The login ID is created using a string of randomly chosen digits
    The login_staff table is checked to see if the user already is registered
    If the user is not registered:
        The airline table is checked to see if the airline already exists
        If the airline does not exist:
            It is added to the airline table
        A new row is created in the airline_staff table with the user’s information
        A new row is created in the email table with the user’s email
        A new row is created in the staff_email table with the user’s information
            The staff_email table is a relationship set between a user and their potentially multiple emails
        A new row is created in the login_staff table with the user’s information
        The password before being inserted into the login_staff table is hashed using bcrypt
```

**Login Staff**

Structure:

```
    The user inputs their username, password, and airline
    The login_staff table is checked to see if the user has registered
    If the user has registered:
        The airline_staff table is searched to find the staff member’s entry
    If the entry exists:
        The password is checked against the hashed password stored in login_staff
    If the passwords are the same:
        A session is created storing the username and airline name
```

**Login Customer**

Structure:

```
    The user inputs their username and password
    The login_cust table is checked to see if the user has registered
    If the user has registered:
        The password is checked against the hashed password stored in login_staff
    If the passwords are the same:
        A session is created storing the username
```

After logging in as a customer, 6 more use cases become available to the user.

**Customer Use Cases:**

**Viewing Their Future Flights**

Structure:

```
    The user can choose to enter their username, password, departure airport, arrival airport, starting range date, and ending range date
    If none are inputted:
        A flight natural join table is searched for the rows that are in the future and that match the session email variable
    If the rows exist:
        Each row’s information from the table is outputted to the user
    If the fields are inputted:
        A flight  natural join ticket table is searched for equivalent checks to the inputted information
    If rows exist:
        It iterates over every row, takes the departure date out of the row, and compares it to the start and end inputted range date
        If the departure date is between the dates, the row is added to a separate list
        The new list’s rows are then outputted to the user (i.e. flights that have inputted qualities that happened during this time range)
```

**Searching Flights**

Structure:

```
    The user inputs the departing and arrival airport, and the departure date.
    They also may input the return date for searching for flights that are round trips.
    If the user does not input a return date:
        A search in the flight table is made using equivalent checks to the airports and departure date inputted and that the flight has not already happened
        If results are found:
            Each equivalent flight’s information is outputted to the user
    If the user inputs a return date:
        The same check is made to the flight table with the addition of return date as an equivalent check
        If results are found:
            Each flight’s information is outputted
```

**Purchase Flight**

Structure:

```
    The user inputs the airline name, the flight number, their card number, the name on their card, the expiration date on their card, and their card type (credit or debit)
    If the card type is credit or debit:
        A ticket ID is made by forming a string of random characters
        A search in the ticket table is made using equivalent checks for the customer’s email, the airline, and flight number to see if the user already has a ticket on the flight
    If they do not find a match:
        The ticket table is searched again but this time it outputs the total number of tickets where the flight number is equivalent to the inputted one
            This is the total number of seats sold so far
        A flight natural join airplane table is searched with flight number equivalent and the number of seats physically on the plane is outputted
            This is the total number of seats physically on the flight
        If the flight is 60% full or over, then the base price is increased by 20%
        If there are still seats on the plane:
```

It checks to see if there are any available tickets that were previously cancelled

If there are previously cancelled tickets matching the airline and flight number, then that ticket is updated with this user's inputted information and this user is assigned that ticket

Else, a row is inserted into the ticket table with the user's information and he is assigned a brand new ticket

Regardless of the outcome, both paths insert a row into the attempt_purchase table documenting purchase history

**Cancel Ticket**

Structure:

The user inputs the airline name and the flight number of the flight they want to cancel their ticket for

A search is conducted in a flight natural join table for rows that depart at least a day from now and are equivalent to inputs

If this row exists, then you delete it from attempt_purchase

A search in ticket for equivalent to inputs is conducted

If search is found:

Update ticket and set every field other than ticket_id to NULL

You look in the cancel ticket table and if the ticket_id, customer combo doesn't exist you insert a row

**Rate and Comment on Flight**

Structure:

The user inputs the airline name, flight number, departure and arrival airports, their rating and comment

Searching the flight natural join ticket table, if the flight already happened and it's equivalent to inputted information, then it validates that the customer had a ticket and that the flight was in the past

If the ticket was found, you check login_cust to see if the customer has an entry

If they do, then you check to see if the user already has an entry in the create_review table

If they do, then you update that entry with the new rating and comment

If not, then you insert a row into the create_review table with the inputted information, rating, and comment

**Check Total Spent**

Structure:

The user inputs a beginning and ending range

The range is to learn how much money they spent during that time

If the ranges are empty, you search the ticket table and find the sum price of all the rows that were purchased at most a year ago and have the customer email

If the rows exist, you iterate through a for loop defined by the year that was exactly 1 year ago and the noninclusive ending range as the year after the current one
In each iteration, you find the sum price for every ticket sold during each month of the last 6 months
You now have every month over the past 6 months and the total amount that was spent during that month ex [‘july’] = $60
The total price spent during the past year is outputted along with a month sum price for that month chart for the past 6 months

If the user inputs start and end dates, the same structure occurs but instead of the total being by default of the past year it’s not the total price spent between those ranges
Additionally, the months between those ranges and the total price spent during each month is outputted to a table
The total is also output

**Staff Member Use Cases:**

**Staff Views Airline Flights**
Structure:
The user can input the departure and arrival airport, and beginning and ending ranges
If the fields are empty, you check the flight table for any rows that depart within the next 30 days and have the airline name
If flights were found, you do the search again with flight natural join ticket natural join customer and then you output that information (outputs every customer and their information that rode that flight)
If the fields are filled out, you search flights where the departure date is between the ranges and the rest are equivalent
If flights were found, you search flight natural join ticket natural join customer with departure date between the ranges and equivalent to inputted information
You then output this information (outputs every customer and their information that rode that flight)

**Create New Flight**
Structure:
The user inputs flight number, departure and arrival date and time, departure and arrival airport, airplane ID, the baseprice, and the flight status
If the fields are empty, you search the flight table and output every row’s information that departs in the next 30 days
If the fields are filled in, you search the airplane table to see if the idnum exists
If the ID matches an existing airplane, you search the airport table to check that both arrival and departure airports are actual airports
If all the checks return true, then you check the staff login to recover the staff’s password
If this is true, you search flight with the new flight number and airline to see if the flight already exists

If the flight does not exist, then you insert a new row into the flight table with the inputted values and insert a new row into create_set_price with the inputted information and base price

**Change Flight Status**

Structure:

The user inputs flight number, departure and arrival airports, departure date, and the new flight status
It checks that the flight status is either ‘on-time’,’delayed’, or ‘cancelled’
If so, it checks login staff table to get the staff password
If that works, then you search flight table with all equal to inputted information
If that returns a row, then you check set_flight_status to determine if that staff member flight number combo already has an entry
If the entry exists then you update the row with the new status
If the entry doesn’t exist, then you add a new row to set_flight_status with the inputted information
You then update the flight table and set the flight status as the new status

**Add Airplane To Airline**

Structure:

The user inputs the airplane ID, the number of seats on the airline, the manufacturing company, and the age of the airplane
You check login staff to get the staff password
If that works, then you search the airplane table to see if any rows equal the inputted information
If it does not, then you insert a new row into the airplane table with its ID number and the airline that it now belongs to
You also insert a new row into the add airplane table with the input values
You then search the airplane table again and get the rows of every airplane that the airline owns
These rows and information are outputted and shown

**Add Airport**

Structure:

The user inputs the airport name, the city, the country, and the airport type (domestic, international, or both)
It checks to make sure the airport type is one of the three options
If it is, you search login staff to get the staff password
If that works, then you search the airport table to see if any rows equal the inputted information
If it does not, then you insert a new row into the airport table which registers the airport
You also insert a new row into the add airport table with the input values

**View Ratings**

Structure:

From searching the ticket natural join create review table, it calculates the total amount each flight operated by the airline has earned in ratings and divides that by the number of reviews for each flight to get the average rating for each flight and that is outputted if the search works

If the search works, then you do another search with ticket natural join create_review that gets every customers comment and rating for each flight which is outputted

**View Customers**

Structure:

The ticket table is searched and the total number of tickets bought by each customer is returned

After iterating through the rows and finding the customer with the most amount of tickets, that customer and the number are outputted

User inputs the email of the customer that they want to see the flight history of

A search happens for all rows in a ticket natural join flight natural join customer table where the flight had already happened and is equal to the inputted information

If that exists, then the rows are outputted and the customers flight history is shown

**View Reports**

Structure:

User inputs a beginning and ending range

The total number of tickets sold for the airline between the ranges is searched

If the rows exist, you iterate through a for loop defined by the year that begins at the starting range and ends plus one year of the ending range

In each iteration, you find the total number of tickets sold during each month between the two range dates

You now have every month over the range and the total number of tickets that were bought during that month ex [‘july’] = 7

The total number of tickets bought during the range is outputted along with a month total number sold for that month chart for every month in the range period

**Change Base Price**

Structure:

User inputs a flight number and a new base price

The login_staff table is checked to find the staff’s password

If it exists, you search the flight table that matches the airline and flight_number

If the flight exists, then you update the flight by setting the flight’s base price to the new price

You then search the create_set_price table with it being equal to the inputted entries to see if that user had already interact with the baseprice before

If they had interacted with it and a row is returned, then you update the row by setting the base price to the new price

If the row does not exist, then you insert a row into the table with the corresponding username, flight number, and airline values

**View Revenue**

Structure:

The ticket table is searched and the sold prices of the tickets that were purchased within the last month are added together (i.e. total revenue for this airline over the past month)
If this returns a row, the above query is searched again but this time instead of month it's year (total revenue over the last year)
Both are outputted

**Add Email**

Structure:

User inputs a new email address that they want to be associated with their account
The email table is searched to see if the email address is already registered
If the email is not registered, then you insert a new row into the email table and you insert a new row into the staff_email table (staff_email is the relationship set between the entities email and airline_staff)

**Add Number**

Structure:

User inputs a new phone number that they want to be associated with their account
The phone_number table is searched to see if the phone number is already registered
If the number is not registered, then you insert a new row into the phone_number table and you insert a new row into the staff_phone table (staff_phone is the relationship set between the entities phone_number and airline_staff)

**Count the Number of Flights With __ Status**

Structure:

User inputs the flight status that they want to query to see how many flights of the airline have that status
The flight_status table is searched and the total number of flights in the airline that have that flight status is counted
If the total is not 0, then the total is outputted