How to run:

The website opens by typing http://127.0.0.1:5000 in the URL bar in a browser while the python script "app.py" is running in the terminal.

When we start (Registration/Login):

Flight Information Lookup:

All users can search for future flights using the source airport, destination airport, and departure date. The corresponding SQL query is:

```
SELECT * FROM Flight WHERE departure_airport = %s AND
arrival_airport = %s AND DATE(departure_date_time) =  %s AND
DATE(departure_date_time) >= %s
```

Explanation: This query iterates through the Flight table and fetches flights matching the specified criteria that will occur in the future.

Staff and Customer Login/Signup:

On the same page we have Log In/Signup for staff and customers. Staff and customers can login to their existing account or signup and create a new account by clicking on either button.

Staff Login:

Staff can log in by providing a username and password. The password is securely hashed using MD5:

```
password = request.form['password'].encode()

hashed_pass = hashlib.md5(password).hexdigest()
```

Authentication query:

```
SELECT * FROM Airline_Staff WHERE username = %s AND password =
%s
```

Successful authentication initiates a user session and redirects to the staff home page.

Staff Signup:

Staff can Signup by providing a username, password, their first name, last name, date of birth, the airline name of where they are employed, their phone number and email address. The password is securely hashed using MD5:

```
password = request.form['password'].encode()

hashed_pass = hashlib.md5(password).hexdigest()
```

Airline existence query:

```
SELECT name FROM Airline WHERE name = %s
```

Explanation: This query is executed with the airline name provided by the staff member to determine if there exists an airline with the provided name. If there isn't one we display an error message

Username uniqueness query:

```
SELECT username FROM Airline_Staff WHERE username = %s
```

Explanation: If the username already exists, the user is redirected to the login page.

Data insertion into database query:

```
INSERT INTO Airline_Staff (username, password, first_name,
last_name, date_of_birth, airline_name) VALUES (%s, %s, %s, %s,
%s, %s)
```

Explanation: If all checks pass, the staff member's details are inserted into the database. This involves multiple insertions to handle various data elements:

Inserting phone number query:

```
INSERT INTO Airline_Staff_Phone (username, phone_number) VALUES
(%s, %s)
```

Inserting email address query:

```
INSERT INTO Airline_Staff_Email (username, email_address) VALUES
(%s, %s)
```

After the queries are executed we redirect the user to the login page to initiate a session.

Customer Login:

Customers can log in by providing an email address and password. The password is securely hashed using MD5 as seen previously.

Account authentication query:

```
SELECT * FROM Customer WHERE email_address = %s AND password = %s
```

Explanation: Queries the Customer table to determine if that account exists. Successful authentication initiates a user session and redirects to the Customer home page.

Customer Signup:

Customers can Sign up by providing an email address, password, first name, last name, date of birth, building number, street name, apartment number, city, state, zip code, date of birth, phone number, passport number, passport expiration, passport country. The password is hashed using md5 as seen previously.

Email uniqueness query:

```
SELECT * FROM Customer WHERE email_address = %s
```

Explanation: The system checks if the provided email already exists in the database. If the email already exists, the user is redirected to the login page.

If the email is unique, the customer's details are inserted into the database. This involves multiple insertions:

Data insertion into database query:

Inserting main customer details:

```
INSERT INTO Customer (email_address, password, first_name,
last_name, building_number, street_name, apartment_number, city,
state, zip_code, date_of_birth) VALUES (%s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s)
```

Inserting phone number:

```
INSERT INTO Customer_Phone (email_address, phone_number) VALUES
(%s, %s)
```

Inserting passport details:

```
INSERT INTO Customer_Passport (email_address, passport_number,
passport_expiration, passport_country) VALUES (%s, %s, %s, %s)
```

After the queries are executed we redirect the user to the login page to initiate a session.

Staff Home Page:

When staff logs in, they have the option to click on any of these options:

view flights, create flights, change flight statuses, add airplanes, add airports, view flight ratings, schedule maintenance, view frequent customers, view the earned revenue, update profile, and log out.

Before any function is executed, the system verifies if the user is authenticated and authorized:

```
    username = session.get('username')

    if not username:

        return redirect(url_for('login_staff'))
```

Functionalities

1) Display/Search for Flights:

Staff can search for flights by entering the source airport, destination airport, departure date, and/or arrival date.

Default display query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s AND
departure_date_time BETWEEN %s AND %s
```

Explanation: This query displays flights for that airline in the next 30 days

The user can also enter values into the fields. In this case, as not all the fields are required, if statements are used to add additional conditions on to the query depending on which attributes are provided. As seen below:

This is the initial query:

```
    SELECT airline_name FROM Airline_Staff WHERE username =
    %s", (username)
```

Then, additional conditions are appended as needed:

```
AND departure_airport = %s

AND arrival_airport = %s

AND DATE(departure_date_time) = %s

AND DATE(arrival_date_time) = %s
```

We then execute the query and display the results.

2) Create New Flights:

Staff can create new flights by providing flight number, departure airport, departure date and time, arrival airport, arrival date and time, base price of ticket, airplane id, and the status of the flight.

Airline confirmation query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member attempting to add a new flight is associated with an airline in the system. It ensures that only authorized airline staff can add new flights.

Default flights display query:

```
SELECT * FROM Flight WHERE airline_name = %s AND
departure_date_time BETWEEN %s AND %s
```

Explanation: This query is used to display flights scheduled by the airline within the next 30 days, where the Parameters are today (datetime.now().date()), and today + timedelta(days=30)

Plane existence query:

```
SELECT * FROM Airplane WHERE identification_number = %s
```

Explanation: Checks if the airplane with the given identification number already exists in the database. It ensures that the flight is being assigned to an existing airplane.

Maintenance check query:

```
SELECT * FROM Maintenance_procedure WHERE airplane_id = %s AND
(%s BETWEEN start_date_time AND end_date_time) OR (%s BETWEEN
start_date_time AND end_date_time)
```

Explanation: Checks whether there are any maintenance procedures scheduled for the airplane during the imputed flight times. It prevents scheduling flights during times when the airplane is unavailable due to maintenance.

Flight insertion query:

```
INSERT INTO Flight (flight_number, airline_name,
departure_airport, departure_date_time, arrival_airport,
arrival_date_time,  base_price_of_ticket, airplane_id, status)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
```

Explanation: After all checks are passed, this query is used to insert the new flight details into the Flight table.

3) Change Status of flights:

If a staff member clicks on the change a flight status option, they are redirected to a page where they are asked to input the flight number and select a new status either on time, canceled or delayed.

Airline confirmation query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member attempting to change a flight status is associated with an airline in the system. It ensures that only authorized airline staff can change flight status.

Flight existence check query:

```
SELECT * FROM Flight WHERE flight_number = %s AND airline_name =
%s
```

Explanation: This query checks if the flight, identified by flight_number and linked to the airline_name, exists in the Flight table. it prevents the system from attempting to update non-existent flights.

Flight status update query:

```
UPDATE Flight SET status = %s WHERE flight_number = %s AND
airline_name = %s
```

Explanation: After confirming the existence of the flight and the staff's authorization, this query updates the status of the specified flight. The status field of the flight, identified by flight_number and associated with the airline_name, is updated to the new status provided by the user.

4) Add an Airplane:

Staff can add a new airplane by providing the plane's identification number, number of seats, manufacturing company, model number, and manufacturing date.

Airline confirmation query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member attempting to add a new airplane is associated with an airline in the system. It ensures that only authorized airline staff can add new airplanes.

Error check for plane identification:

```
SELECT * FROM Airplane WHERE identification_number = %s
```

Explanation: This query checks whether an airplane with the provided identification number already exists in the database.

Airplane insertion query:

```
INSERT INTO Airplane (identification_number, airline_name,
number_of_seats, manufacturing_company, model_number,
manufacturing_date, age) VALUES (%s, %s, %s, %s, %s, %s, %s)
```

Explanation: Once confirmed that the information provided is unique, an entry is inserted into the database using the query:

5) Add new airport in the system:

When staff members select the option to add a new airport, they are directed to a page where they can input a unique code, a name, a city, a country, number of terminals, and an airport type (domestic/international/both).

Existing Airports Query:

```
SELECT * FROM Airport
```

Explanation: Before adding a new airport, the system displays existing airports through a query.

Check for Existing Airport Code query:

```
SELECT * FROM Airport WHERE code = %s
```

Explanation: Before adding a new airport, we confirm that the passed in Airport ID doesn't already exist in the database with a query.

Insert new Airport query:

```
INSERT INTO Airport (code, name, city, country, number_of_terminals,
airport_type) VALUES (%s, %s, %s, %s, %s, %s)
```

Explanation: If the airport code is unique, this query inserts a new airport into the Airport table.

6) View flight ratings:

When staff members select the option to view flight ratings, they are directed to a page where they can input a flight number to view the flight's average rating along with individual ratings and comments.

Airline confirmation query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member is associated with an airline. It ensures that only authorized staff can access flight ratings for their airlines.

Flight ratings query (when a flight number is provided):

```
SELECT fr.flight_number, fr.customer_email, fr.rating,
fr.comment, avg_ratings.average_rating
     FROM Flight_Rating fr
     JOIN (
         SELECT flight_number, AVG(rating) as average_rating
         FROM Flight_Rating
         WHERE airline_name = %s AND flight_number = %s
         GROUP BY flight_number
     ) avg_ratings ON fr.flight_number =
     avg_ratings.flight_number
     WHERE fr.airline_name = %s AND fr.flight_number = %s
```

Explanation: This query is used to retrieve the flight number, customer email, rating, comment, and the average rating of the specified flight. The subquery calculates the average rating for the given flight number of that user's airline.

Default flight ratings query:

```
SELECT fr.flight_number, fr.customer_email, fr.rating,
fr.comment, avg_ratings.average_rating
FROM Flight_Rating fr
JOIN (
        SELECT flight_number, AVG(rating) as average_rating
        FROM Flight_Rating
        WHERE airline_name = %s
        GROUP BY flight_number
) avg_ratings ON fr.flight_number = avg_ratings.flight_number
WHERE fr.airline_name = %s
```

Explanation: This query is executed if no specific flight number is provided. It retrieves the ratings for all flights associated with the airline, along with the average ratings for each flight.

7) Schedule maintenance:

When staff members select the option to schedule maintenance, they are directed to a page where they must enter the airplane ID and specify the start and end dates and times for the maintenance procedure.

Airline confirmation query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member is associated with an airline. It ensures that only authorized staff can schedule maintenance only for airplanes belonging to their airline.

Flight schedule conflict check query:

```
SELECT * FROM Flight WHERE airplane_id = %s AND airline_name = %s
AND (
        (departure_date_time BETWEEN %s AND %s) OR(arrival_date_time
     BETWEEN %s AND %s)
)
```

Explanation: Before scheduling maintenance, this query checks whether the specified airplane is already assigned to any flights during the proposed maintenance period.

Maintenance scheduling query:

```
INSERT INTO Maintenance_procedure (airline_name, airplane_id,
start_date_time, end_date_time)
VALUES (%s, %s, %s, %s)
```

Explanation: Once it's confirmed that there are no scheduling conflicts, this query is used to insert the maintenance schedule into the Maintenance_procedure table.

8) View frequent customers:

Airline staff have the capability to view the most frequent customers within the last year and access a list of all flights a particular customer has taken with their airline.

When staff members click on the view frequent customer option, they are redirected to a page where they are asked to input a certain customer's email to view a table with the customer's flight count. By default we show the amount of flights each customer has taken in the last year

Airline confirmation query:

```
    SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member is associated with an airline. It ensures that only authorized staff can access customer data for their airlines.

Frequent customers query (default):

```
SELECT customer_email, COUNT(*) as flight_count
FROM Ticket
Where airline_name = %s and
purchase_date_time BETWEEN curdate() - INTERVAL 1 YEAR AND CURDATE()
GROUP BY customer_email
ORDER BY flight_count DESC
```

Explanation: This query identifies frequent customers of the airline by counting the number of flights each customer has taken in the last year. It provides a list of customers ranked by their flight count in descending order.

Customer flight details query:

```
SELECT * FROM Flight
WHERE flight_number IN
(SELECT flight_number
FROM Ticket
WHERE customer_email = %s AND airline_name = %s)
```

Explanation: This query is executed when a staff member searches for a specific customer's email. It retrieves all flights taken by that customer on the staff member's airline. This allows the staff to view a customer's flight history with the airline.

9) View earned revenue:

When a staff member selects the earned revenue option, they gain access to the airline's revenue from the previous month and the previous year.

Airline confirmation query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member is associated with an airline. It ensures that only authorized staff can revenue data belonging to their airline.

Last month revenue query:

```
SELECT SUM(calculated_ticket_price) AS month
FROM Ticket
WHERE airline_name = %s AND
purchase_date_time BETWEEN DATE_SUB(NOW(), INTERVAL 1 MONTH) AND
NOW()
```

Explanation: This query calculates the total revenue generated from ticket sales for the previous month. It selects the sum of calculated_ticket_price from the Ticket table for tickets sold within the last month (DATE_SUB(NOW(), INTERVAL 1 MONTH) to NOW())

Last year revenue query:

```
SELECT SUM(calculated_ticket_price) AS year
FROM Ticket
WHERE airline_name = %s AND
purchase_date_time BETWEEN DATE_SUB(NOW(), INTERVAL 1 YEAR) AND NOW()
```

Explanation: This query calculates the total revenue generated from ticket sales for the previous year. It selects the sum of calculated_ticket_price from the Ticket table for tickets sold within the last year (DATE_SUB(NOW(), INTERVAL 1 YEAR) to NOW())

10) Update profile:

A staff member can update their email and/or phone number associated with their profile using this feature. Because the entries for the page are not requested we first check if the provided email/phone exists before we add it to its respective table.

Airline confirmation query:

```
SELECT airline_name FROM Airline_Staff WHERE username = %s
```

Explanation: This query is used to confirm that the staff member is associated with an airline. It ensures that only authorized staff can have accounts and thus can modify their profiles.

Check email query:

```
SELECT * FROM Airline_Staff_Email WHERE email_address = %s
```

Explanation: Before adding a new email, the system checks if the provided email address already exists in the Airline_Staff_Email table.

Insert email query:

```
INSERT INTO Airline_Staff_Email (username, email_address) VALUES
(%s, %s)
```

Explanation: If the email does not already exist, this query adds the new email address to the staff member's profile.

Check phone number query:

```
SELECT * FROM Airline_Staff_Phone WHERE phone_number = %s
```

Explanation: Before adding a new phone number, the system checks if the provided phone number already exists in the Airline_Staff_Phone table.

If the phone number does not already exist, this query adds the new number to the staff member's profile.

Insert phone number query:

```
INSERT INTO Airline_Staff_Phone (username, phone_number) VALUES
(%s, %s)
```

Explanation: If the phone number does not already exist, this query adds the new number to the staff member's profile.

11) Log Out:

A staff member can log out using the link on the home page. Log out is simply performed by popping the username from the session and returning to home.

Customer Home Page:

->When customers log in or register, they have the option to click on any of these options:

view personal flights, search for flights, purchase tickets, cancel trips, give ratings, comment on previous flights, track spending, and log out and update their profile information.

Functionalities

1) To view personal flights

If a customer chooses to click on the option to view their personal flights, they are redirected to a page where they are asked to first choose if they are looking for their a past flights, future flights or both and then a table with their booked flights will appear with the flight number, departure airport and arrival airport.

Query for future flights:

```
SELECT Flight.* FROM Flight

JOIN Ticket ON Flight.flight_number = Ticket.flight_number

AND Flight.departure_date_time = Ticket.departure_date_time

WHERE Ticket.customer_email = %s AND Flight.departure_date_time
> %s
```

Query for past flights:

```
SELECT Flight.* FROM Flight

JOIN Ticket ON Flight.flight_number = Ticket.flight_number

AND Flight.departure_date_time = Ticket.departure_date_time

WHERE Ticket.customer_email = %s AND Flight.departure_date_time
< %s
```

Query for any flight:

```
SELECT Flight.* FROM Flight
```

```
JOIN Ticket ON Flight.flight_number = Ticket.flight_number

AND Flight.departure_date_time = Ticket.departure_date_time

WHERE Ticket.customer_email = %s
```

Explanation: This query retrieves all available information from the Flight table for flights booked by a specific customer, linking the Flight and Ticket tables to ensure accurate matching of flight numbers and departure dates/times. It specifically targets flights associated with the customer's email and filters for flights departing after/before a specified date and time. The purpose is to enable customers to view their booked flights, whether past, future, or both. The process involves the customer selecting the flight type they wish to view, the system executing the query based on the customer's email and chosen time frame, and then displaying the relevant flight details.

2) To search for flights

If a customer chooses to click on the option to view their personal flights, they are redirected to a page where they are asked to input the departure airport code, arrival airport code, departure and return date and if it's a round trip or one way trip. Flights that match the customer's requirements will then appear on a table that includes the flight number and departure/arrival airports.

```
SELECT flight_number, airline_name, departure_airport,
arrival_airport, departure_date_time, arrival_date_time,
base_price_of_ticket

FROM Flight

WHERE departure_airport = %s AND arrival_airport = %s AND
departure_date_time >= %s
```

Explanation:
This query is designed to help customers search for flights by selecting specific columns from the Flight table, including flight number, airline name, departure and arrival airports, departure and arrival times, and the base ticket price. The data is sourced from the Flight table, which contains comprehensive flight details. The query filters flights based on the customer's input for departure airport, arrival airport, and a date and time threshold, ensuring only flights that match these criteria are included. The purpose is to enable customers to find available flights that align with their specific travel needs. The process involves the customer entering their search criteria, the system executing the query to locate matching flights, and then displaying these flights to the customer.

3) To purchase tickets

If a customer chooses to click on the option to purchase flight tickets, they are redirected to a page where they are asked to input the flight number, airline name, their first and last name, date of birth, card type, card number, name on card and card expiration date.

```
SELECT departure_date_time, base_price_of_ticket, airplane_id
FROM Flight WHERE flight_number = %s AND airline_name = %s
```

Explanation: The first query selects the departure date and time, base ticket price, and airplane ID from the Flight table, filtering results based on a specific flight number and airline name. This ensures that only details of the chosen flight are fetched.

```
INSERT INTO Ticket (flight_number, departure_date_time,
airline_name, purchase_date_time, base_ticket_price,
calculated_ticket_price, customer_email, first_name, last_name,
date_of_birth, card_type, card_number, name_on_card,
expiration_date)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
```

Explanation: The second query, INSERT INTO Ticket, adds the customer's ticket purchase information to the Ticket table. The purpose of these queries is to facilitate the ticket booking process: the customer provides necessary details like flight number and payment information, the system retrieves relevant flight details to confirm availability and pricing, and then the customer's ticket information is recorded in the database, completing the purchase.

4) To cancel trips

If a customer chooses to click on the option to cancel flight tickets, they are redirected to a page where they are asked to input the ticket ID

```
SELECT flight_number, departure_date_time FROM Ticket

JOIN Flight ON Ticket.flight_number = Flight.flight_number

WHERE ticket_ID = %s AND customer_email = %s
```

Explanation: This process involves a query sequence to manage the cancellation of flight tickets. The initial query uses SELECT to retrieve the flight number and departure date and time from the joined Ticket and Flight tables, ensuring the flight details match the specific ticket. The JOIN condition requires the flight_number to be the same in both tables. The query filters results by ticket_ID and customer_email to identify the specific ticket of a particular customer.

```
DELETE FROM Ticket WHERE ticket_ID = %s
```

Explanation: DELETE FROM Ticket query is executed to remove the ticket from the database, but only if it matches the specified ticket ID. The purpose is to allow customers to cancel their bookings, subject to certain conditions. The steps involve the customer providing the ticket ID for cancellation, the system verifying the ticket's existence and ownership, and then, if the cancellation criteria are met, removing the ticket from the database to complete the cancellation process.

5) To give ratings and comments

If a customer chooses to click on the option to ratings and comments, they are redirected to a page where they are asked to input the flight number and departure date/time and give a rating out of 5. A customer also has the option to leave a comment on their flight experience.

Check if the customer has taken this flight:

```
SELECT * FROM Ticket

WHERE customer_email=% AND flight_number=% AND
departure_date_time = %
```

Insert Rating and Comment Query:

```
INSERT INTO Flight_Rating (flight_number, airline_name,
departure_date_time, customer_email, rating, comment)

VALUES (%s, %s, %s, %s, %s, %s)
```

Explanation: This process first involves making sure the customer has taken the flight they want to rate, using the INSERT INTO Flight_Rating query to add new customer feedback to the Flight_Rating table, which is designed to store ratings and comments about flights. taken, inputting a rating and, if desired, a comment.

6) To track my spending

If a customer chooses to click on the option to track my spending, they are redirected to a page where they are asked to input the period in which they want to track their spendings in and then a table with the month and it's spendings will be displayed.

Total Spending Query:

```
SELECT SUM(calculated_ticket_price) AS total_spending

FROM Ticket
```

```
WHERE customer_email = %s AND purchase_date_time BETWEEN %s AND
%s
```

Explanation: This query calculates the total spending of a customer on flight tickets by summing up the calculated_ticket_price from the Ticket table. The SELECT SUM(calculated_ticket_price) AS total_spending part of the query computes the total amount spent, and AS total_spending serves as an alias for this sum, making it easily identifiable in the results. The query pulls data from the Ticket table, specifically targeting records that match the customer's email (WHERE customer_email = %s) and fall within a specified purchase date range (AND purchase_date_time BETWEEN %s AND %s). This approach ensures that only the tickets bought by a particular customer within the defined time period are considered for the total spending calculation.

Month-wise Spending Query:

```
SELECT MONTH(purchase_date_time) AS month,
SUM(calculated_ticket_price) AS monthly_spending

FROM Ticket

WHERE customer_email = %s AND purchase_date_time BETWEEN %s AND
%s

GROUP BY MONTH(purchase_date_time)
```

Explanation: This query is designed to calculate a customer's monthly spending on flight tickets. It extracts the month from the purchase_date_time column (SELECT MONTH(purchase_date_time) AS month), representing the month of ticket purchase. The query then calculates the total spending per month (SUM(calculated_ticket_price) AS monthly_spending) from the Ticket table. It filters the data to include only tickets purchased by a specific customer (WHERE customer_email = %s) and within a defined time range (AND purchase_date_time BETWEEN %s AND %s). The GROUP BY MONTH(purchase_date_time) clause groups the results by month, enabling the calculation of monthly spending totals. This approach provides a clear view of a customer's expenditure on flight tickets, broken down by month, based on their email and the specified time period.

7) To update my profile

If a customer chooses to click on the option to update their profile, they are redirected to a page where they are asked to input their new phone number/ passport number/expiration date/country

Check and Insert Phone Number:

Check: `"SELECT * FROM Customer_Phone WHERE email_address = %s AND phone_number = %s"`

Explanation: This allows us to verify if a specific phone number is already associated with a given customer's email address in the Customer_Phone table.

Insert: `"INSERT INTO Customer_Phone (email_address, phone_number) VALUES (%s, %s)"`

Explanation: If the phone number is not found, the insert query adds the new phone number for the customer to the database. This two-step process ensures that the customer's phone number is updated without creating duplicate entries.

Check and Insert Passport Details:

`Check: "SELECT * FROM Customer_Passport WHERE email_address = %s AND passport_number = %s"`

Explanation: Similarly, for passport details, the initial check query (SELECT * FROM Customer_Passport WHERE email_address = %s AND passport_number = %s) searches the Customer_Passport table to confirm whether the passport number is already linked to the customer's email address.

`Insert: "INSERT INTO Customer_Passport (email_address, passport_number, passport_expiration, passport_country) VALUES (%s, %s, %s, %s)"`

Explanation: If the passport details are new, the insert query (INSERT INTO Customer_Passport (email_address, passport_number, passport_expiration, passport_country) VALUES (%s, %s, %s, %s)) is executed to record the customer's passport information, including the number, expiration date, and issuing country, thereby keeping the customer's travel documents up-to-date in the system.

8) Log Out:

A customer can log out using the link on the home page. Log out is simply performed by popping the username from the session and returning to home.