Customer

1. 1. **View My flights:** Provide various ways for the user to see flights information which he/she purchased. The default should be showing for the future flights. **Optionally** you may include a way for the user to specify a range of dates, specify destination and/or source airport name or city name etc.

```
with connection.cursor() as cursor:
    cursor.execute("""SELECT DISTINCT
                f.Name,
                f.flightNum,
                f.depDate,
                f.depTime,
                f.depAirport,
                f.arrDate,
                f.arrTime,
                f.arrAirport,
                f.ID,
                f.basePrice,
                f.status
            FROM flight AS f
            JOIN ticket AS t
                ON f.flightNum = t.flightNum
                AND f.depDate = t.flightDepDate
                AND f.depTime = t.flightDepTime
            WHERE t.nameOfHolder = %s""", (username))
    flight_records = cursor.fetchall()
```

The above SELECT query lets customer view flights that they have previously purchased in one table

2. 2. **Search for flights:** Search for future flights (one way or round trip) based on source city/airport name, destination city/airport name, dates (departure or return).

```
start_date = request.form.get(escape_html('start_date'))
end_date = request.form.get(escape_html('end_date'))
source = request.form.get(escape_html('source'))
destination = request.form.get(escape_html('destination'))
flight_type = request.form.get(escape_html('flight_type'))

if not start_date:
    start_date = '2020-01-01'
if not end_date:
    end_date = '2100-12-31'

connection = get_db_connection()

if flight_type == 'one-way':
    try:
        with connection.cursor() as cursor:
            cursor.execute("SELECT * FROM flight WHERE depAirport = %s AND arrAirport = %s AND depDate = %s", (source, destinatic
            flight_records = cursor.fetchall()
    finally:
        connection.close()
else:
    try:
        with connection.cursor() as cursor:
            cursor.execute("SELECT * FROM flight WHERE depAirport = %s OR depAirport = %s AND (depDate = %s OR depDate = %s)", (s
            flight_records = cursor.fetchall()
```

Takes input from the customer to search for the flight based on source city, destination etc

3. **Purchase tickets:** Customer chooses a flight and purchase ticket for this flight, providing all the needed data, via forms. You may find it easier to implement this along with a use case to search for flights.

```
flight_name = request.form.get(escape_html('flight_name'))
flight_flightNum = request.form.get(escape_html('flight_flightNum'))
flight_depDate = request.form.get(escape_html('flight_depDate'))
flight_depTime = request.form.get(escape_html('flight_depTime'))
flight_ID = request.form.get(escape_html('flight_id'))
flight_ticketPrice = request.form.get(escape_html('flight_basePrice'))
cardNum = request.form['cardNum']
cardType = request.form.get(escape_html('cardType')) #input by customer
nameOfHolder = request.form.get(escape_html('nameOfHolder')) #input by customer
expirationDate = request.form.get(escape_html('expirationDate')) #input by customer
email = session['customer_username']
now = datetime.now()
purchaseTime = now.strftime('%H:%M:%S')
purchaseDate = now.strftime('%Y-%m-%d')
connection = get_db_connection()
try:
    with connection.cursor() as cursor:
        sql = """
        INSERT INTO ticket
            (flightName, flightNum, flightDepDate, flightDepTime, flightID, ticketPrice, cardNum, cardType, nameOfHolder, exp
        VALUES
            (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
```

We take inputs from the database on the details of the flights needed and then we use INSERT statements to record the purchase into the tickets table.

```
insert_purchase = """
INSERT INTO purchases
    (CustomerEmail, flightName, flightNum, flightDepDate, flightDepTime, flightID, PurchaseDate, PurchaseTime)
VALUES
    (%s, %s, %s, %s, %s, %s, %s, %s)
"""
# Data tuple contains all the values to be inserted
data = (email, flight_name, flight_flightNum, flight_depDate, flight_depTime, flight_ID, purchaseDate, purchaseTime)

# Execute the query with the data tuple
cursor.execute(insert_purchase, data)
```

We take payment inputs from the customer and then we input them into the purchases table using an INSERT statement as shown above

4. **Cancel Trip:** Customer chooses a purchased ticket for a flight that will take place more than 24 hours in the future and cancel the purchase. After cancellation, the ticket will no longer belong to the customer. The ticket will be available again in the system and purchasable by other customers.

```
with connection.cursor() as cursor:
    sql = "SELECT ticketID FROM ticket WHERE flightNum = %s AND flightName = %s"
    cursor.execute(sql, (purchase_number, purchase_name))
    result = cursor.fetchone()
    tixID = result['ticketID'] if result and result['ticketID'] is not None else 0

    print(email, purchase_name, purchase_number, purchase_depDate, purchase_depTime)
    query1 = "DELETE FROM purchases WHERE CustomerEmail = %s AND flightName = %s AND flightNum = %s AND flightDepDate = %
    cursor.execute(query1, (email, purchase_name, purchase_number, purchase_depDate, purchase_depTime))
    # cursor.execute("DELETE FROM purchases WHERE CustomerEmail = %s AND flightName = %s AND flightNum = %s AND flightDep
    connection.commit()
    query2 = "DELETE FROM ticket WHERE ticketID = %s"
    cursor.execute(query2, (tixID))
    # cursor.execute("DELETE FROM ticket WHERE ticketID = %s", tixID)
    connection.commit()
```

When a customer cancels their flight, it will get all the data of that flight and the delete statements will delete the entries from both the purchases and ticket tables

5. 5. **Give Ratings and Comment on previous flights:** Customer will be able to rate and comment on their previous flights (for which he/she purchased tickets and already took that flight) for the airline they logged in.

```
email = request.form.get('Email')
ticketID = request.form.get('ticketID')
rating = request.form.get('Rating')
comment = request.form.get('Comment')
flightDate = request.form.get('flightDate')
flightDateStr = datetime.strptime(flightDate, '%Y-%m-%d').date()
tdyDate = datetime.now().date()
connection = get_db_connection()

if(flightDateStr>tdyDate): #if flight is in future
    connection.close()
    return redirect('/')

else:
    try:
        with connection.cursor() as cursor:
            cursor.execute("INSERT INTO review (emailAddress, ticketID, Rating, Comment) VALUES (%s, %s, %s, %s)", (email, 
            connection.commit()
    finally:
        connection.close()
    return redirect('/')
```

Takes in the inputs from customer and the insert statement puts it into the review table. Customers only can do it

6. 6.**Track My Spending**: Default view will be total amount of money spent in the past year and a bar chart/table showing month wise money spent for last 6 months. He/she will also have option to specify a range of dates to view total amount of money spent within that range and a bar chart/table showing month wise money spent within that range.

```
with connection.cursor() as cursor:
    cursor.execute("SELECT SUM(ticketPrice) AS total FROM ticket where flightDepDate >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR) AND 
    result = cursor.fetchone()
    total_past_year = result['total'] if result and result['total'] is not None else 0

    monthly_spending_query = """
    SELECT YEAR(flightDepDate), MONTH(flightDepDate), SUM(ticketPrice)
    FROM ticket
    WHERE flightDepDate >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
    AND nameOfHolder = %s
    GROUP BY YEAR(flightDepDate), MONTH(flightDepDate)
    ORDER BY YEAR(flightDepDate), MONTH(flightDepDate)
    """
```

Top select statement is to return the total spending of the customer from the past year. The second select statement is to return the total spending of a customer from the past 6 months

```
start_date = request.form['start_date']
end_date = request.form['end_date']
cursor.execute("SELECT SUM(ticketPrice) AS total FROM ticket WHERE flightDepDate BETWEEN %s AND %s", (start_date, end_da
result = cursor.fetchone()
range_total = result['total'] if result and result['total'] is not None else 0
#range_total = cursor.fetchone()[0] or 0
monthly_range_query = """
SELECT YEAR(flightDepDate), MONTH(flightDepDate), SUM(ticketPrice)
FROM ticket
WHERE flightDepDate BETWEEN %s AND %s
AND nameOfHolder = %s
GROUP BY YEAR(flightDepDate), MONTH(flightDepDate)
ORDER BY YEAR(flightDepDate), MONTH(flightDepDate)
"""
cursor.execute(monthly_range_query, (start_date, end_date, name))
```

This chunk of code is to display the amount of money a customer spent in each of the months of a selected range

7. 7.**Logout:** The session is destroyed and a “goodbye” page or the login page is displayed.

```
@app.route('/logout', methods=['GET', 'POST'])
def logout():
    # Check if the user is logged in as staff or customer and then log them out.
    session.pop('customer_username')
    print("logged out")
    session.clear()  # Clear all session data
    return redirect('/customer-login')
```

Pops the session and brings customer back to their log in page

Staff

1. 1. **View flights:** Defaults will be showing all the future flights operated by the airline he/she works for the next 30 days. He/she will be able to see all the current/future/past flights operated by the airline he/she works for based range of dates, source/destination airports/city etc. He/she will be able to see all the customers of a particular flight.

```
username = session.get('staff_username')
connection = get_db_connection()
try:
    with connection.cursor() as cursor:
        cursor.execute("SELECT Airline_Name FROM airlinestaff WHERE Username = %s", (username,))
        result = cursor.fetchone()
        airline = result['Airline_Name'] if result and result['Airline_Name'] is not None else 0
        cursor.execute("SELECT * FROM flight WHERE Name = %s", (airline))
        flight_records = cursor.fetchall()
```

First select statement is to extract the airline the staff works for. After which that is used in the second select statement which will only show the staff flight of their airline

2. 2**. Create new flights:** He or she creates a new flight, providing all the needed data, via forms. The application should prevent unauthorized users from doing this action. While creating a flight, a check should be performed that if an airport is domestic, it should only be able to handle domestic flights, and same applies for international airports. Defaults will be showing all the future flights operated by the airline he/she works for the next 30 days.

```
airline = result['Airline_Name']
flight_id = request.form.get('flight_number')
airplane_id = request.form.get('aircraft_id')
dep_airport = request.form.get('departure_airport')
arr_airport = request.form.get('arrival_airport')
dep_date = request.form.get('departure_date')
dep_time = request.form.get('departure_time')
arr_date = request.form.get('arrival_date')
arr_time = request.form.get('arrival_time')
price = request.form.get('price')
status = "on-time"
cursor.execute("INSERT INTO flight (Name, flightNum, depAirport, arrAirport, depDate, depTime, arrDate, arrTime, ID,
connection.commit()
```

```
 arrTime, ID, basePrice, status) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
```

This insert statement takes all the information given by the staff and insertes it into the flight table of the database

3. 3. **Change Status of flights:** He or she changes a flight status (from on-time to delayed or vice versa) via forms.

```
flight_id = request.form.get('flight_id')
status = request.form.get('new_status')


connection = get_db_connection()
try:
    with connection.cursor() as cursor:
        cursor.execute("UPDATE flight SET status = %s WHERE flightNum = %s", (status, flight_id))
        connection.commit()
finally:
    connection.close()
return redirect('/flights')
```

Get the flight ID and the new status from the staff. Update query then updates the database with the new status

4. 4. **Add airplane in the system:** He or she adds a new airplane, providing all the needed data, via forms. The application should prevent unauthorized users from doing this action. In the confirmation page, she/he will be able to see all the airplanes owned by the airline he/she works for.

```
airplane_id = request.form.get(escape_html('airplane_id'))
manufacturing_company = request.form.get(escape_html('manufacturing_company'))
manufacturing_date = request.form.get(escape_html('manufacturing_date'))
NumberOfSeats = request.form.get(escape_html('number_of_seats'))
model_number = request.form.get(escape_html('model_number'))
AirlineName = session['staff_airline']
connection = get_db_connection()
try:
    with connection.cursor() as cursor:
        cursor.execute("INSERT INTO airplanes (ID, ManufacturingCompany, ManufacturingDate, NumberOfSeats, ModelNumber, Airli
        connection.commit()
finally:
    connection.close()
return redirect('/airplanes')
:
return render_template('add_airplane.html')#view airports route
```

The form requests input from the staff and using an INSERT query, inserts the details of a new plane into the airplanes table in the database

5. 5. **Add new airport in the system:** He or she adds a new airport, providing all the needed data, via forms. The application should prevent unauthorized users from doing this action.

```
if request.method == 'POST':
    airport_id = request.form.get('airport_id')
    airport_name = request.form.get('airport_name')
    airport_type = request.form.get('airport_type')
    airport_city = request.form.get('airport_city')
    airport_country = request.form.get('airport_country')
    terminal = request.form.get('terminal')
    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            cursor.execute("INSERT INTO airport (Code, Name, AirportType, City, Country, Terminals) VALUES (%s, %s, %s, %s, %s, %
            connection.commit()
```

Takes in an input from staff such as airport code, name etc and the insert query puts it into the airport tables in the database

6. 6. **View flight ratings:** Airline Staff will be able to see each flight's average ratings and all the comments and ratings of that flight given by the customers.

```
ticket_id = request.form.get('ticket_id')
connection = get_db_connection()
reviews = []
try:
    with connection.cursor() as cursor:
        sql_query = "SELECT emailAddress, ticketID, Rating, Comment FROM review WHERE ticketID = %s"
        cursor.execute(sql_query, (ticket_id,))
        reviews = cursor.fetchall()
```

Query allows staff of an airline to view the reviews given by customers based on the ticketID

7. 7. **Schedule Maintenance:** Airline staff should be able to schedule maintenance (by providing start date and time and end date and time) for a particular airplane (identified by airline name and airplane id). Airplanes under maintenance cannot be assigned to a flight during the maintenance period.

```
if request.method == 'POST':
    start_date = request.form.get('start_date')
    start_time = request.form.get('start_time')
    end_date = request.form.get('end_date')
    end_time = request.form.get('end_time')
    airplane_id = request.form.get('airplane_id')
    connection = get_db_connection()
    try:
        with connection.cursor() as cursor:
            cursor.execute("INSERT INTO maintenance (Start_Date, Start_Time, End_Date, End_Time, AirplaneID) VALUES (%s, %s, %s,
            connection.commit()
    finally:
        connection.close()
    return redirect('/view-maintenance')
else:
    return render_template('schedule_maintenance.html')
```

Takes the input given by the staff like airplane_id etc and insert query to insert the information into the maintenance table in the database.

8. 8. **View frequent customers:** Airline Staff will also be able to see the most frequent customer within the last year. In addition, Airline Staff will be able to see a list of all flights a particular Customer has taken only on that particular airline.

```
with connection.cursor() as cursor:
    cursor.execute("SELECT Airline_Name FROM airlinestaff WHERE Username = %s", (username))
    result = cursor.fetchone()
    airline = result['Airline_Name'] if result and result['Airline_Name'] is not None else 0
    print(airline)
    sql_query = """
        SELECT DISTINCT t.nameOfHolder, t.flightNum
        FROM ticket AS t
        INNER JOIN (
            SELECT nameOfHolder
            FROM ticket
            WHERE flightName = %s
            GROUP BY nameOfHolder
            HAVING COUNT(*) >= 3
        ) AS frequent_flyers ON t.nameOfHolder = frequent_flyers.nameOfHolder
        WHERE t.flightName = %s
        """
    cursor.execute(sql_query,(airline, airline))
    frequent = cursor.fetchall()
```

We define a frequent customer as someone who has flown more than 3 times. These queries allow staff to see customers who have flown with the airline they work for, for more than 3 times.

9. **View Earned Revenue:** Show total amount of revenue earned from ticket sales in the last month and last year.

9.

```
# Fetch total revenue from the last month filtered by flightName
last_month_query = """
SELECT SUM(ticketPrice) AS total
FROM ticket
WHERE flightDepDate BETWEEN DATE_SUB(CURDATE(), INTERVAL 1 MONTH) AND CURDATE()
AND flightName = %s
"""
cursor.execute(last_month_query, (flight_name,))
result = cursor.fetchone()
last_month_revenue = result['total'] if result and result['total'] is not None else 0

# Fetch total revenue from the last year filtered by flightName
last_year_query = """
SELECT SUM(ticketPrice) AS total
FROM ticket
WHERE flightDepDate BETWEEN DATE_SUB(CURDATE(), INTERVAL 1 YEAR) AND CURDATE()
AND flightName = %s
"""
cursor.execute(last_year_query, (flight_name,))
```

Top portion of the picture is the revenue generated by the airline in the past month. This is denoted by the interval 1 month in the WHERE clause.
Bottom portion of the picture is the revenue generated by the airline in the past year. This is denoted by the interval 1 year in the WHERE clause

10. **Logout:** The session is destroyed and a “goodbye” page or the login page is displayed.

10.

```
@app.route('/logout-staff', methods=['GET'])
def logoutStaff():
    session.pop('staff_username')
    session.pop('staff_logged')
    print("logged out")
    session.clear()  # Clear all session data
    return redirect('/staff-login')
```

Pops the session and brings the staff back to their log in page