**Public Information Use Cases:**

The user can see the upcoming flights based on (departure, arrival) airports, (departure, arrival) time and (departure, arrival) city together or either of them. Also, the user will be able to see the flight status based on flight number and (departure, arrival) date.

- Based on City:

```
searchtext = request.form['citysearchbox']
    query = 'select * from flight,airport where 
(airport.airport_name=flight.departure_airport or 
airport.airport_name=flight.arrival_airport) and airport.airport_city=%s 
and (departure_time >= curtime() or arrival_time >= curtime())'
```

- Based on dates:

```
cursor = conn.cursor()
    query = 'select * from flight where ((departure_time between %s and 
%s) or (arrival_time between %s and %s)) and (departure_time >= curtime() 
or arrival_time >= curtime())'
    # query = 'select * from flight where ((departure_time between %s and 
%s) or (arrival_time between %s and %s))'
    cursor.execute(query, (begintime, endtime, begintime, endtime))
    data = cursor.fetchall()
    cursor.close()
```

- Based on airports:

```
query = 'select * from flight where (departure_airport = %s or 
arrival_airport = %s) and (departure_time >= curtime() or arrival_time >= 
curtime())'
    # query = 'select * from flight where (departure_airport = %s or 
arrival_airport = %s)'
    cursor.execute(query, (searchtext, searchtext))
    data = cursor.fetchall()
    cursor.close()
```

- Based on them together:

```
    begintime = request.form['departure_time']
    endtime = request.form['arrival_time']
    dep_airport = request.form['Departure']
    arr_airport = request.form['Arrival']
    dep_city = request.form['dep_city']
    arr_city = request.form['arr_city']
    if not validateDates(begintime, endtime):
        error = 'Invalid date range'
        return redirect(url_for('searchpage', error=error))
    if len(dep_airport)>3 or len(arr_airport)>3:
        error = "Please enter the abbreviation of airport."
        return redirect(url_for('searchpage', error=error))
    cursor = conn.cursor()
    query = 'SELECT * FROM flight, airport, ticket \
          WHERE airport.airport_name=flight.departure_airport \
          AND flight.flight_num = ticket.flight_num \AND
flight.airline_name = ticket.airline_name\
          AND airport.airport_city = %s \
          AND airport.airport_name = %s \
          # AND flight.status = "Upcoming"\
          AND %s BETWEEN DATE_SUB(flight.departure_time, INTERVAL 2 DAY)
AND DATE_ADD(flight.departure_time, INTERVAL 2 DAY) \
          AND %s BETWEEN DATE_SUB(flight.arrival_time, INTERVAL 2 DAY) AND
DATE_ADD(flight.arrival_time, INTERVAL 2 DAY) \
          AND (flight.airline_name, flight.flight_num) in \
            (SELECT flight.airline_name, flight.flight_num FROM flight,
airport \
            WHERE airport.airport_name=flight.arrival_airport \
            AND airport.airport_city = %s \
            AND airport.airport_name = %s)'

cursor.execute(query,(dep_city,dep_airport,begintime,endtime,arr_city,arr_
airport))
    data = cursor.fetchall()
    cursor.close()
```

- Search for the status:

```
flight_num = request.form['flight number']
    departure_time = request.form['departure_time']
    arrival_time = request.form['arrival_time']

    cursor = conn.cursor()
    query = 'select status from flight where flight_num = %s and
date(departure_time) = %s and date(arrival_time) = %s'
    cursor.execute(query,(flight_num,departure_time,arrival_time))
    data = cursor.fetchall()
    cursor.close()
```

**<u>Login and Register:</u>**

- Login:

```
 username = request.form['username']
  password = request.form['password']
  usrtype = request.form['usrtype']

  #cursor used to send queries
  cursor = conn.cursor()
  #executes query
  if usrtype == 'staff':
    query = 'SELECT * FROM airline_staff WHERE username = %s and password
= md5(%s)'
  elif usrtype == 'customer':
    query = 'SELECT * FROM customer WHERE email = %s and password =
md5(%s)'
  else:
    query = 'SELECT * FROM booking_agent WHERE email = %s and password =
md5(%s)'

  cursor.execute(query, (username, password))
  #stores the results in a variable
  data = cursor.fetchone()
  #use fetchall() if you are expecting more than 1 data row
  cursor.close()
  error = None
  if(data):
    #creates a session for the the user
    #session is a built in
    session['username'] = username
    if usrtype == 'staff':
      return redirect(url_for('staffHome'))
    elif usrtype == 'customer':
      return redirect(url_for('customerHome'))
    else:
      return redirect(url_for('agentHome'))
```

```
else:
  #returns an error message to the html page
  error = 'Invalid login or username'
  return render_template('login.html', error=error)
```

- Register:
  - ❏ Customer:

```
email = request.form['email']
 name = request.form['name']
 password = request.form['password']
 building_number = request.form['building_number']
 street = request.form['street']
 city = request.form['city']
 state = request.form['state']
 phone_number = request.form['phone_number']
 passport_number = request.form['passport_number']
 passport_expiration = request.form['passport_expiration']
 passport_country = request.form['passport_country']
 date_of_birth = request.form['date_of_birth']

 #cursor used to send queries
 cursor = conn.cursor()
 #executes query
 query = 'SELECT * FROM customer WHERE email = %s'
 cursor.execute(query, (email))
 #stores the results in a variable
 data = cursor.fetchone()
 #use fetchall() if you are expecting more than 1 data row
 error = None
 if(data):
   #If the previous query returns data, then user exists
   error = "This user already exists"
   return render_template('registerCustomer.html', error = error)
 else:
   ins = 'INSERT INTO customer VALUES(%s, %s, md5(%s), %s, %s, %s, %s,
%s, %s, %s, %s, %s)'
```

```
    cursor.execute(ins, (email, name, password, building_number, street,
city, state, phone_number, passport_number, passport_expiration,
passport_country, date_of_birth))
    conn.commit()
    cursor.close()
    return render_template('index.html')
```

- Agent:

```
  email = request.form['email']
  password = request.form['password']
  booking_agent_id = request.form['booking_agent_id']

  cursor = conn.cursor()
  query = 'SELECT * FROM booking_agent WHERE email = %s'
  cursor.execute(query, (email))
  data = cursor.fetchone()
  error = None
  if(data):
    error = "This user already exists"
    return render_template('registerAgent.html', error = error)
  else:
    ins = 'INSERT INTO booking_agent VALUES(%s, md5(%s), %s)'
    cursor.execute(ins, (email, password, booking_agent_id))
    conn.commit()
    cursor.close()
    return render_template('index.html')
```

- Staff:

```
username = request.form['username']
  password = request.form['password']
  first_name = request.form['first_name']
  last_name = request.form['last_name']
  date_of_birth = request.form['date_of_birth']
  airline_name = request.form['airline_name']

  cursor = conn.cursor()
  query = 'SELECT * FROM airline_staff WHERE username = %s'
  cursor.execute(query, (username))
  data = cursor.fetchone()
  error = None
  if(data):
    error = "This user already exists"
    return render_template('registerStaff.html', error = error)
  else:
    ins = 'INSERT INTO airline_staff VALUES(%s, md5(%s), %s, %s, %s, %s)'
    cursor.execute(ins, (username, password, first_name, last_name,
date_of_birth, airline_name))
    conn.commit()
    cursor.close()
    return render_template('index.html')
```

**Customer Use Cases:**

1. View My flights:
   View all the flights bought by the user and the departure time is later than the current time.

```
   username = session['username']
   cursor = conn.cursor()
   query = 'SELECT purchases.ticket_id, ticket.airline_name, ticket.flight_num,
departure_airport, departure_time, arrival_airport, arrival_time \
   FROM purchases, ticket, flight \
   WHERE purchases.ticket_id = ticket.ticket_id \
   AND ticket.airline_name = flight.airline_name \
   AND ticket.flight_num = flight.flight_num \
   AND customer_email = %s AND departure_time > curdate()'
   cursor.execute(query, (username))
   data = cursor.fetchall()
   cursor.close()
```

Also, the user can choose the specified range of time, departure or arrival airport and departure or arrival city to view the flights which are purchased by themselves.

By city:

```
city = request.form.get('citysearchbox', False)
     if city:
       query='SELECT purchases.ticket_id, ticket.airline_name, ticket.flight_num,
departure_airport, departure_time, arrival_airport, arrival_time \
               FROM purchases, ticket, flight, airport \
               WHERE purchases.ticket_id = ticket.ticket_id \
               AND ticket.airline_name = flight.airline_name \
               AND ticket.flight_num = flight.flight_num \
               AND (flight.arrival_airport = airport_name or flight.departure_airport
= airport_name)\
               AND airport.airport_city = %s\
               AND customer_email = %s AND departure_time > curdate();'
       cursor.execute(query, (city, username))
       data = cursor.fetchall()
       cursor.close()
```

By airport:

```
if airport:
       query='SELECT purchases.ticket_id, ticket.airline_name, ticket.flight_num,
departure_airport, departure_time, arrival_airport, arrival_time \
               FROM purchases, ticket, flight\
               WHERE purchases.ticket_id = ticket.ticket_id \
               AND ticket.airline_name = flight.airline_name \
               AND ticket.flight_num = flight.flight_num \
               AND (flight.arrival_airport = %s or flight.departure_airport = %s)\
               AND customer_email = %s AND departure_time > curdate();'
       cursor.execute(query, (airport, airport, username))
       data = cursor.fetchall()
```

By date:

```
begintime = request.form["begintime"]
     endtime = request.form["endtime"]

     if datetime.datetime.strptime(begintime, "%Y-%m-%d") >
datetime.datetime.strptime(endtime, "%Y-%m-%d"):
       return render_template('viewUpcomingFlights.html', username=username,
error="The dates you entered are invalid.")
     else:
       query="SELECT distinct purchases.ticket_id, ticket.airline_name,
ticket.flight_num, departure_airport, departure_time, arrival_airport, arrival_time \
             FROM purchases, ticket, flight\
             WHERE purchases.ticket_id = ticket.ticket_id \
             AND ticket.airline_name = flight.airline_name \
             AND ticket.flight_num = flight.flight_num \
             AND purchases.customer_email = %s \
             AND flight.departure_time > curdate()\
             and ((date(flight.departure_time) between %s and %s) or
(date(flight.arrival_time) between %s and %s));"
       cursor.execute(query, (username, begintime, endtime, begintime, endtime))
       data = cursor.fetchall()
       cursor.close()
```

2. Purchase Tickets:
    - Search for the tickets available:

    Users specify the (departure,arrival) airport, (departure,arrival) city and (departure,arrival) time, the query will show all the airline name, flight number, airplane id, ticket price, (departure,arrival) airport and (departure,arrival) time which serve the requirement.
    Note that, we don’t expect the exact match of (departure,arrival) time. For all the flights, we have a plus/minus 2 days interval on their departure and arrival time. If the given begin and end time are correspondingly in this interval, the results will be shown.
    Also, if the seats on the plane are full, the flight information won’t be shown.

```
 cursor = conn.cursor()
 fromcity = request.form['fromcity']
 fromairport = request.form['fromairport']
 fromdate = request.form['fromdate']
 tocity = request.form['tocity']
 toairport = request.form['toairport']
 todate = request.form['todate']
 query = "SELECT distinct f.airline_name, f.flight_num, departure_airport,
departure_time, arrival_airport, arrival_time, price, airplane_id \
         FROM flight as f, airport \
         WHERE airport.airport_name=f.departure_airport \
         AND airport.airport_city = %s \
         AND airport.airport_name = %s \
         AND %s BETWEEN DATE_SUB(f.departure_time, INTERVAL 2 DAY) AND
DATE_ADD(f.departure_time, INTERVAL 2 DAY)\
         AND %s BETWEEN DATE_SUB(f.arrival_time, INTERVAL 2 DAY) AND
DATE_ADD(f.arrival_time, INTERVAL 2 DAY)\
         AND (f.airline_name, f.flight_num) in \
           (SELECT flight.airline_name, flight.flight_num FROM flight, airport \
           WHERE airport.airport_name=flight.arrival_airport \
           AND airport.airport_city = %s \
           AND airport.airport_name = %s) \
         AND (SELECT DISTINCT seats \
             FROM flight, airplane \
             WHERE flight.airplane_id = airplane.airplane_id AND flight.airline_name
= airplane.airline_name \
             AND flight.airline_name = f.airline_name AND flight.flight_num =
f.flight_num) \
             >= (SELECT COUNT(*) \
             FROM ticket \
            WHERE ticket.airline_name = f.airline_name AND ticket.flight_num =
f.flight_num)"
 cursor.execute(query, (fromcity, fromairport, fromdate, todate, tocity, toairport))
```

- Purchase the tickets:
  If the flight number doesn't exist in the database or it doesn't match with the given airline name or the seats are full, an error message will be given.

```
 username = session['username']
 cursor = conn.cursor()
 airline_name = request.form['airline_name']
 flight_num = request.form['flight_num']

 ##################################################################
 message = request.args.get('message')
 query1 = "select * from flight where airline_name = %s and flight_num = %s\
           AND (SELECT DISTINCT seats \
           FROM flight, airplane \
           WHERE flight.airplane_id = airplane.airplane_id AND flight.airline_name =
airplane.airline_name \
           AND flight.airline_name = f.airline_name AND flight.flight_num =
f.flight_num) \
           >= (SELECT COUNT(*) \
           FROM ticket \
           WHERE ticket.airline_name = f.airline_name AND ticket.flight_num =
f.flight_num)"
 cursor.execute(query1,(airline_name,flight_num))
 data1 = cursor.fetchall()
 if not data1:
   error = " The airline name and flight number don't exist or the flight is full,
please check the enter."
   return render_template('purchaseCustomer.html',purchaseError = error)

 ##################################################################
  # Find the number of tickets to generate the next ticket_id
 queryCount = 'SELECT COUNT(*) as count FROM ticket \
               WHERE ticket.airline_name = %s AND ticket.flight_num = %s'
 cursor.execute(queryCount, (airline_name, flight_num))
 ticketCount = cursor.fetchone()
 ticketCountVal = 0
 if ticketCount != None:
   ticketCountVal = ticketCount['count']
 # ticket_id = _genTix(ticketCountVal, airline_name.strip().replace(' ', ''),
flight_num)
 ticket_id = _genTix()
 # print("WHAT FUCKING NUMBER: ", ticket_id)
 # Create the new ticket
 queryNewTicket = 'INSERT INTO ticket VALUES(%s, %s, %s)'
 cursor.execute(queryNewTicket, (ticket_id, airline_name, flight_num))
 # Finalize the purchase
 queryPurchase = 'INSERT INTO purchases VALUES(%s, %s, %s, CURDATE())'
 cursor.execute(queryPurchase, (ticket_id, username, None))
 data = cursor.fetchone()
 conn.commit()
 cursor.close()
 #######################################################################
 message = " Successfully purchased!"
 return render_template('purchaseCustomer.html',message = message)
```

3. Search for Flights:
   This user case is realized along with the purchase search in purchase tickets case.

4. Track my spending:
   By default, it will show the total amount of money spent in the past year and a bar chart showing month wise money spent for the last 6 months.
   x axis: month
   y axis: monthly spending

   Customers will also have the option to specify a range of dates to view the total amount of money spent within that range and a bar chart showing month wise money spent within that range. In the code, we specify them by assigning different from_date and to_date values.

   Here is how we convey the format of date and setting the from_date and to_date:

```
   username = session['username']
   if request.method == 'POST':
         to_date = request.form['to_date']
         from_date = request.form['from_date']
         if datetime.datetime.strptime(from_date, "%Y-%m-%d") >
datetime.datetime.strptime(to_date, "%Y-%m-%d"):
             return render_template('viewCustomerSpending.html', error="The dates you
entered are invalid.")
         to_date_format = datetime.datetime.strptime(to_date, '%Y-%m-%d')
         from_date_format = datetime.datetime.strptime(from_date, '%Y-%m-%d')
```

```
      year = to_date_format.year
      month = to_date_format.month
      date = to_date_format.day
      from_year = from_date_format.year
      from_month = from_date_format.month
      from_date_date = from_date_format.day
      from_date_string = '{}-{}-{}'.format(from_year, from_month, from_date_date)
      to_date_string = '{}-{}-{}'.format(year, month, date)
      monthnum = (year - from_year)*12 + month - from_month
else:
      to_date = datetime.datetime.now()
      year = to_date.year
      month = to_date.month
      if month-6 < 0:
        from_month = 12 + (month-6)
        from_year = to_date.year - 1
      else:
        from_month = month-6
        from_year = year
      date = to_date.day
      monthnum = 5
      string = '{} 1 {} 00:00'.format(month, from_year)
      from_date = datetime.datetime.strptime(string, '%m %d %Y %H:%M')
      from_date_string = '{}-{}-{}'.format(from_year, from_month, date)
      to_date_string = '{}-{}-{}'.format(year, month, date)
```

Sum spending query:

```
cursor = conn.cursor()
sumPrice = 'SELECT COALESCE(sum(flight.price),0) as sum\
          FROM purchases,ticket,flight\
          WHERE purchases.customer_email = %s\
          AND ticket.ticket_id = purchases.ticket_id\
          and ticket.flight_num = flight.flight_num\
          and ticket.airline_name = flight.airline_name\
          and purchases.purchase_date >= %s\
          and purchases.purchase_date <= %s;'
cursor.execute(sumPrice, (username, from_date_string, to_date_string ))
sumSpending = float(cursor.fetchone()['sum'])
```

Monthly spending query:

```
if month < 12:
    string = '{} 1 {} 00:00'.format(month+1, year + 1)
else:
    string = '{} 1 {} 00:00'.format(1, year + 2)
temp_date = datetime.datetime.strptime(string, '%m %d %Y %H:%M')

labels = []
values = []
temp_year = year
temp_month = month
for i in range(0,monthnum + 1):
    this_date = temp_date
    this_month = temp_month
    temp_month = (month-i)%12
    if temp_month == 0:
        temp_month = 12
    if temp_month > this_month:
        temp_year = temp_year - 1
    string = '{} 1 {} 00:00'.format(temp_month, temp_year)
    temp_date = datetime.datetime.strptime(string, '%m %d %Y %H:%M')
    query = 'SELECT COALESCE(sum(flight.price),0) as sum_month\
          FROM purchases,ticket,flight\
          WHERE purchases.customer_email = %s\
          AND ticket.ticket_id = purchases.ticket_id\
          and ticket.flight_num = flight.flight_num\
          and ticket.airline_name = flight.airline_name\
          and purchases.purchase_date > %s\
          and purchases.purchase_date < %s;'
    cursor.execute(query, (username, temp_date, this_date, ))
    data = cursor.fetchone()
```

5. Logout: The login page will be shown.

```
@app.route('/logout')
def logout():
  session.pop('username')
  return redirect('/login')
```

**Booking Agent Use Cases:**

1. View My flights:
   View all the flights bought by the specific booking agent and the departure time is later than the current time.

```
username = session['username']
```

```
    cursor = conn.cursor()
    query = 'SELECT * \
    FROM purchases, ticket, flight, booking_agent \
    WHERE purchases.ticket_id = ticket.ticket_id \
    AND ticket.airline_name = flight.airline_name \
    AND ticket.flight_num = flight.flight_num \
    AND booking_agent.email = %s AND booking_agent.booking_agent_id = 
purchases.booking_agent_id \
    AND departure_time > curdate() \
    ORDER BY customer_email'
    cursor.execute(query, (username))
    data = cursor.fetchall()
```

Also, the user can choose to specify the range of time, departure or arrival airport and departure or arrival city to view the flights which are purchased by himself.

- Specify by city:

```
username = session['username']
    cursor = conn.cursor()
    if request.method == 'POST':
      city = request.form.get('citysearchbox', False)
      if city:
        query = 'SELECT * \
            FROM purchases, ticket, flight, booking_agent, airport \
            WHERE purchases.ticket_id = ticket.ticket_id \
            AND ticket.airline_name = flight.airline_name \
            AND ticket.flight_num = flight.flight_num \
            AND (flight.arrival_airport = airport_name or 
flight.departure_airport = airport_name)\
            AND booking_agent.email = %s AND 
booking_agent.booking_agent_id = purchases.booking_agent_id \
            AND airport.airport_city = %s\
            AND departure_time > curdate() \
            ORDER BY customer_email'
        cursor.execute(query, (username, city))
        data = cursor.fetchall()
        cursor.close()
```

- Specify by airport:

```
 airport = request.form.get('airportsearchbox', False)
```

```
      if airport:
        query = 'SELECT * \
            FROM purchases, ticket, flight, booking_agent \
            WHERE purchases.ticket_id = ticket.ticket_id \
            AND ticket.airline_name = flight.airline_name \
            AND ticket.flight_num = flight.flight_num \
            AND (flight.arrival_airport = %s or flight.departure_airport
= %s)\
            AND booking_agent.email = %s AND
booking_agent.booking_agent_id = purchases.booking_agent_id \
            AND departure_time > curdate() \
            ORDER BY customer_email'
        cursor.execute(query, (airport, airport, username))
        data = cursor.fetchall()
        cursor.close()
```

- Specify by time range:

```
      begintime = request.form["begintime"]
      endtime = request.form["endtime"]

      if datetime.datetime.strptime(begintime, "%Y-%m-%d") >
datetime.datetime.strptime(endtime, "%Y-%m-%d"):
        return render_template('viewUpcomingFlights.html',
username=username, error="The dates you entered are invalid.")
      else:
        query = 'SELECT * \
            FROM purchases, ticket, flight, booking_agent \
            WHERE purchases.ticket_id = ticket.ticket_id \
            AND ticket.airline_name = flight.airline_name \
            AND ticket.flight_num = flight.flight_num \
            AND booking_agent.email = %s AND
booking_agent.booking_agent_id = purchases.booking_agent_id \
            AND departure_time > curdate()\
            and ((date(flight.departure_time) between %s and %s) or
(date(flight.arrival_time) between %s and %s))\
            ORDER BY customer_email;'
        cursor.execute(query, (username, begintime, endtime, begintime,
endtime))
        data = cursor.fetchall()
        cursor.close()
```

2. Purchase Tickets:
    - Search for the tickets available:

Users specify the (departure,arrival) airport, (departure,arrival) city and (departure,arrival) time, the query will show all the airline name, flight number, airplane id, ticket price, (departure,arrival) airport and (departure,arrival) time which serve the requirement.
Note that, we don't expect the exact match of (departure,arrival) time. For all the flights, we have a plus/minus 2 days interval on their departure and arrival time. If the given begin and end time are correspondingly in this interval, the results will be shown.
Also, if the seats on the plane are full, the flight information won't be shown.

```
  username = session['username']
  cursor = conn.cursor()
  fromcity = request.form['fromcity']
  fromairport = request.form['fromairport']
  fromdate = request.form['fromdate']
  tocity = request.form['tocity']
  toairport = request.form['toairport']
  todate = request.form['todate']
  if datetime.datetime.strptime(fromdate, "%Y-%m-%d") >
datetime.datetime.strptime(todate, "%Y-%m-%d"):
        return render_template('purchaseAgent.html', searchError="The
dates you entered are invalid.")

  query = 'SELECT distinct f.airline_name, f.flight_num,
departure_airport, departure_time, arrival_airport, arrival_time, price,
airplane_id \
          FROM flight as f, airport \
          WHERE airport.airport_name=f.departure_airport \
          AND airport.airport_city = %s \
          AND airport.airport_name = %s \
          AND %s BETWEEN DATE_SUB(f.departure_time, INTERVAL 2 DAY) AND
DATE_ADD(f.departure_time, INTERVAL 2 DAY)\
          AND %s BETWEEN DATE_SUB(f.arrival_time, INTERVAL 2 DAY) AND
DATE_ADD(f.arrival_time, INTERVAL 2 DAY)\
          AND (f.airline_name, f.flight_num) in \
            (SELECT flight.airline_name, flight.flight_num FROM flight,
airport \
            WHERE airport.airport_name=flight.arrival_airport \
            AND airport.airport_city = %s \
            AND airport.airport_name = %s) \
          AND (SELECT DISTINCT seats \
              FROM flight, airplane \
              WHERE flight.airplane_id = airplane.airplane_id AND
flight.airline_name = airplane.airline_name \
              AND flight.airline_name = f.airline_name AND
flight.flight_num = f.flight_num) \
              >= (SELECT COUNT(*) \
              FROM ticket \
              WHERE ticket.airline_name = f.airline_name AND
ticket.flight_num = f.flight_num)'

  cursor.execute(query, (fromcity, fromairport, fromdate, todate, tocity,
toairport))
  # print cursor._executed
  data = cursor.fetchall()
  cursor.close()
```

- Purchase the tickets:

Agents are able to purchase the tickets on behalf of customers. According to the searched information given above,the agent is able to purchase tickets by entering the customer email, flight number and airline name.

```
username = session['username']
customer_email = request.form['customer_email']
airline_name = request.form['airline_name']
flight_num = request.form['flight_num']
queryCount = 'SELECT COUNT(*) as count FROM ticket \
                WHERE ticket.airline_name = %s AND ticket.flight_num =
%s'
  cursor.execute(queryCount, (airline_name, flight_num))
  ticketCount = cursor.fetchone()
  ticketCountVal = 0
  if ticketCount != None:
    ticketCountVal = ticketCount['count']
  # ticket_id = _genTix(ticketCountVal, airline_name.strip().replace(' ',
''), flight_num)
  ticket_id = _genTix()
  # Create the new ticket
  queryNewTicket = 'INSERT INTO ticket VALUES(%s, %s, %s)'
  cursor.execute(queryNewTicket, (ticket_id, airline_name, flight_num))
  # Get booking_agent_id
  queryGetID = 'SELECT booking_agent_id FROM booking_agent WHERE
email=%s'
  cursor.execute(queryGetID, username)
  agentID = cursor.fetchone() # returns a dict
  # Finalize the purchase
  queryPurchase = 'INSERT INTO purchases VALUES(%s, %s, %s, CURDATE())'
  cursor.execute(queryPurchase, (ticket_id, customer_email,
agentID['booking_agent_id']))
  data = cursor.fetchone()
  conn.commit()
  cursor.close()
  error = None
  if(data):
    return render_template('agent.html', username = username,
results=data)
  if not (data):

    message = " Successfully purchased!"
    return render_template('purchaseAgent.html',message = message)
```

Here, if you don't enter a customer email, the system will give an error message.

```
username = session['username']
  customer_email = request.form['customer_email']
  cursor = conn.cursor()
  query1 = 'select * from customer where email = %s'
  cursor.execute(query1,customer_email)
  data1 = cursor.fetchall()
  cursor.close()
  if not data1:
    error = "Customer doesn't exist. Check the email."
    return render_template('CustomerError.html',error = error)
```

Also, if the flight number doesn't exist in the database or it doesn't match with the given airline name or the seats are full, an error message will be given.

```
 message = request.args.get('message')
 query1 = "select * from flight where airline_name = %s and flight_num = %s\
          AND flight.airline_name = f.airline_name AND flight.flight_num =
f.flight_num) \
          >= (SELECT COUNT(*) \
          FROM ticket \
          WHERE ticket.airline_name = f.airline_name AND ticket.flight_num =
f.flight_num)"
 cursor.execute(query1,(airline_name,flight_num))
 data1 = cursor.fetchall()
 if not data1:
   error = " The airline name and flight number don't exist or the flight is full,
please check the enter."
   return render_template('purchaseAgent.html', error= error)
```

3. Search for Flights:
   This user case is realized along with the purchase search in purchase tickets case.

4. View my Commissions:
   By default, view will show the total amount of commission received in the past 30 days and the average commission the agent received per ticket booked in the past 30 days and total number of tickets sold by the agent in the past 30 days.

   - View total commissions in the past 30 days:

```
# Get booking_agent_id
    queryGetID = 'SELECT booking_agent_id FROM booking_agent WHERE
email=%s'
    cursor.execute(queryGetID, username)
    agentID = cursor.fetchone()
    # Get total commsion in the past 30 days
    queryGetCommission = 'SELECT sum(price)*.10 as totalComm FROM
purchases, ticket, flight \
                          WHERE purchases.ticket_id = ticket.ticket_id \
                          AND ticket.airline_name = flight.airline_name
AND ticket.flight_num = flight.flight_num \
                          AND purchases.purchase_date BETWEEN
DATE_SUB(CURDATE(), INTERVAL 30 DAY) AND CURDATE()  \
                          AND purchases.booking_agent_id = %s'
    cursor.execute(queryGetCommission, agentID['booking_agent_id'])
    totalComm = cursor.fetchone()
    totalCommVal = 0
    if totalComm['totalComm'] != None:
      totalCommVal = totalComm['totalComm']
```

   - View total tickets in the past 30 days:

```
queryGetTicketCount = 'SELECT count(*) as ticketCount FROM purchases,
ticket, flight \
                          WHERE purchases.ticket_id = ticket.ticket_id \
                          AND ticket.airline_name = flight.airline_name
AND ticket.flight_num = flight.flight_num \
                          AND purchases.purchase_date BETWEEN
DATE_SUB(CURDATE(), INTERVAL 30 DAY) AND CURDATE() \
                          AND purchases.booking_agent_id = %s'
    cursor.execute(queryGetTicketCount, agentID['booking_agent_id'])
    ticketCount = cursor.fetchone()
    ticketCountVal = ticketCount['ticketCount']
    avgComm = 0
```

- View the average commissions in the past 30 days:

```
if ticketCountVal != 0:
     avgComm = totalCommVal/ticketCountVal
```

Also, we can to see the commission and the number of tickets in the specified range:

- View total commissions in the specified time range:

```
    username = session['username']
    cursor = conn.cursor()
    fromdate = request.form['fromdate']
    todate = request.form['todate']
    # print(fromdate, todate)
    if not validateDates(fromdate,todate):
      error = "Invalid Time Range"
      return redirect(url_for('agentHome',error = error))

    # Get booking_agent_id
    queryGetID = 'SELECT booking_agent_id FROM booking_agent WHERE
email=%s'
    cursor.execute(queryGetID, username)
    agentID = cursor.fetchone()
    # print('~~~DEBUG: ', agentID)
    # Get total commsion in the past 30 days
    queryGetCommission = 'SELECT sum(price)*.10 as totalComm FROM
purchases, ticket, flight \
                          WHERE purchases.ticket_id = ticket.ticket_id \
                          AND ticket.airline_name = flight.airline_name
AND ticket.flight_num = flight.flight_num \
                          AND purchases.purchase_date BETWEEN CAST(%s AS
DATE) AND CAST(%s AS DATE) \
                          AND purchases.booking_agent_id = %s'
```

```
    cursor.execute(queryGetCommission, (fromdate, todate,
agentID['booking_agent_id']))
    totalComm = cursor.fetchone()
    totalCommVal = 0
    if totalComm['totalComm'] != None:
      totalCommVal = totalComm['totalComm']
```

- View total tickets in the specified time range:

```
queryGetTicketCount = 'SELECT count(*) as ticketCount FROM purchases,
ticket, flight \
                          WHERE purchases.ticket_id = ticket.ticket_id \
                          AND ticket.airline_name = flight.airline_name
AND ticket.flight_num = flight.flight_num \
                          AND purchases.purchase_date BETWEEN CAST(%s AS
DATE) AND CAST(%s AS DATE) \
                          AND purchases.booking_agent_id = %s'
    cursor.execute(queryGetTicketCount, (fromdate, todate,
agentID['booking_agent_id']))
    ticketCount = cursor.fetchone()
    ticketCountVal = ticketCount['ticketCount']
```

5. View Top Customers:
   Use two barchats to show top 5 customers based on number of tickets bought from the booking agent in the past 6 months and top 5 customers based on amount of commission received in the last year.
   x axis: customer emails
   y axis: commissions amount
   - In the past 6 months

```
if not authenticateAgent():
    error = 'Invalid Credentials'
    return redirect(url_for('errorpage', error=error))
  else:
    username = session['username']
    cursor1 = conn.cursor()
    queryGetID = 'SELECT booking_agent_id FROM booking_agent WHERE
email=%s'
    cursor1.execute(queryGetID, username)
    agentID = cursor1.fetchone()['booking_agent_id']
    query1 = "select count(ticket_id) as cnt,customer_email \
            from purchases \
            where booking_agent_id = %s\
            and purchase_date between DATE_SUB(curdate(), INTERVAL 6
MONTH) and curdate()\
            group by customer_email\
            order by cnt DESC limit 5"
    cursor1.execute(query1,agentID)
    data1 = cursor1.fetchall()
    cursor1.close()
    error = None
```

- In the past 1 year:

```
cursor2 = conn.cursor()
    query2 = "SELECT sum(price)*.10 as totalComm,
purchases.customer_email FROM purchases, ticket, flight\
              WHERE purchases.ticket_id = ticket.ticket_id\
              AND ticket.airline_name = flight.airline_name AND
ticket.flight_num = flight.flight_num\
              AND purchases.purchase_date BETWEEN DATE_SUB(curdate(),
INTERVAL 1 YEAR) and curdate() \
              AND purchases.booking_agent_id = %s\
              group by purchases.customer_email\
              order by totalComm DESC limit 5"
    cursor2.execute(query2,agentID)
    data2 = cursor2.fetchall()
    cursor2.close()
```

6. Log out: The login page will be shown.

```
@app.route('/logout')
def logout():
  session.pop('username')
  return redirect('/login')
```

**<u>Airline Staff Use Cases:</u>**

1. View My flights:
   Defaults will be showing all the upcoming flights operated by the airline he/she works for the next 30 days. The staff can choose to specify the range of time, departure or arrival airport and departure or arrival city to view the upcoming flights operated by the airline he/she works for.
   By city:

```
cursor = conn.cursor()
      city = request.form['citysearchbox']
```

```
        airline = getStaffAirline()
        query = "select * from flight,airport \
        where (airport.airport_name=flight.departure_airport or
airport.airport_name=flight.arrival_airport) \
        and airport.airport_city=%s and airline_name=%s"
        cursor.execute(query, (city, airline))
        data = cursor.fetchall()
        cursor.close()
```

By airport:

```
        cursor = conn.cursor()
        airport = request.form['airportsearchbox']
        airline = getStaffAirline()
        query = 'select * from flight where (departure_airport = %s or arrival_airport
= %s) and airline_name=%s'
        cursor.execute(query, (airport, airport, airline))
        data = cursor.fetchall()
        cursor.close()
```

By date:

```
        begintime = request.form['begintime']
        endtime = request.form['endtime']

        if not validateDates(begintime, endtime):
            error = 'Invalid date range'
            return redirect(url_for('searchFlightsPage', error=error))

        airline = getStaffAirline()

        cursor = conn.cursor()
        query = "select * from flight \
        where ((departure_time between %s and %s) \
        or (arrival_time between %s and %s)) and airline_name=%s"
        cursor.execute(query, (begintime, endtime, begintime, endtime, airline))
        data = cursor.fetchall()
        cursor.close()
```

2. Create new flights:
   The staff is able to create a new flight by entering a new flight number, an airplane ID, the ticket price, (departure and arrival) airport and (departure and arrival) time.

```
    username = session['username']
    flightnum = request.form['flightnum']
    departport = request.form['departport']
    departtime = request.form['departtime']
    arriveport = request.form['arriveport']
    arrivetime = request.form['arrivetime']
    price = request.form['price']
    status = "Upcoming"
    airplaneid = request.form['airplanenum']
query = 'insert into flight values (%s, %s, %s, %s, %s, %s, %s, %s, %s)'
    cursor.execute(query, (airline, flightnum, departport, departtime,
arriveport, arrivetime, price, status, airplaneid))
    conn.commit()
    cursor.close()
```

There are also several error messages will be popped to the staff when creating the new flight:

- Repeated flight number:

```
airline = getStaffAirline()

    cursor = conn.cursor()
    query1 = 'select * from flight where airline_name = %s and flight_num
= %s'
    cursor.execute(query1,(airline,flightnum))
    data1 = cursor.fetchall()

    if data1:
        error = "The flight number already exists, please enter another
one."
        return redirect(url_for('createFlightPage', error=error))
    cursor.close()
```

- Doesn’t exist airport:

```
cursor = conn.cursor()
    query2 = 'select * from airport where airport_name = %s '
    cursor.execute(query2,(departport))
    data2 = cursor.fetchall()
    query3 = 'select * from airport where airport_name = %s '
    cursor.execute(query3,(arriveport))
    data3 = cursor.fetchall()
    if (not data2):
        error = "The Departure Airport does not exist, please add the
airport first."
        return redirect(url_for('createFlightPage', error=error))
    if (not data3):
        error = "The Arrival Airport does not exist, please add the
airport first."
        return redirect(url_for('createFlightPage', error=error))
    cursor.close()
```

- Invalid date range:

```
 if not validateDates(departtime, arrivetime):
            error = 'Invalid date range'
            return redirect(url_for('createFlightPage', error=error))
```

- Invalid airplane ID:

```
 cursor = conn.cursor()
    query = 'select * from airplane where airplane_id = %s'
    cursor.execute(query, (airplaneid))
    data = cursor.fetchall()
    if not data:
        error = 'Invalid Airplane ID'
        return redirect(url_for('createFlightPage', error=error))
```

To better serve the creation function, the information needed to create a new flight will be given on the page including the upcoming flight in the following 30 days and the current airplane ID and airports.

```
airline = getStaffAirline()
        cursor = conn.cursor()
        airline = getStaffAirline()

        query = "select * from flight where airline_name = %s \
                and ((departure_time between curdate() and
date_add(curdate(), interval 30 day)) \
                or (arrival_time between curdate() and
date_add(curdate(), interval 30 day)))"
        cursor.execute(query, (airline))
        data = cursor.fetchall()

        cursor = conn.cursor()
        query = 'select distinct airport_name from airport'
        cursor.execute(query)
        airportdata = cursor.fetchall()

        query = 'select distinct airplane_id from airplane where 
airline_name=%s'
        cursor.execute(query, (airline))
        airplanedata = cursor.fetchall()

        cursor.close()
```

3. Change Status of flights:
   The staff can change the flight status by entering the flight number and choosing the status from 4 radios (upcoming, delayed, on-time, past).

```
    username = session['username']
    cursor = conn.cursor()
    flightnum = request.form['flightnum']
    status = request.form['status']
    query = 'update flight set status=%s where flight_num=%s and 
airline_name = %s'
    cursor.execute(query, (status, flightnum, airline))
    conn.commit()
    cursor.close()
```

Several error messages will be popped :

- Chosen status is the same one as the flight has now:

```
  username = session['username']
    cursor = conn.cursor()
    flightnum = request.form['flightnum']
    status = request.form['status']
    if not status:
        error = 'Did not select new status'
        return redirect(url_for('changeFlightStatusPage', error=error))
```

- Entered flight number is not on the airline the staff working for:

```
#Check that the flight is from the same airline as the staff
    query = 'select * from flight where flight_num = %s and airline_name 
= %s'
    cursor.execute(query, (flightnum, airline))
    data = cursor.fetchall()
    if not data:
        error = 'Incorrect enter - flight number is not in your airline '
        return redirect(url_for('changeFlightStatusPage', error=error))
```

4. Add airplane in the system:
   The staff can add an airplane in the system by entering a new airplane ID and number of seats. Error message will be popped if the airplane id is already taken.

```
username = session['username']

    planeid = request.form['id']
    seats = request.form['seats']
    airline = getStaffAirline()

    #Check if planeid is not taken
    cursor = conn.cursor()
    query = 'select * from airplane where airplane_id = %s'
    cursor.execute(query, (planeid))
    data = cursor.fetchall()

    if data:
        error = "Airplane ID already taken"
        return redirect(url_for('addAirplanePage', error=error))

    #Insert the airplane
    query = 'insert into airplane values (%s, %s, %s)'
    cursor.execute(query, (airline, planeid, seats))
    conn.commit()

    #Get a full list of airplanes
    query = 'select * from airplane where airline_name = %s'
    cursor.execute(query, (airline))
    data = cursor.fetchall()
    cursor.close()
```

5. Add new airport in the system:
   The staff can add a new airport by entering the airport name and the corresponding city name. The airport name is limited into the 3 character long Abbreviations and error message will pop up if it’s not the format.

```
username = session['username']

    name = request.form['name']
    city = request.form['city']
    if len(name)>3:
        error = "Please enter the abbreviation of airport."
        return redirect(url_for('addAirportPage', error=error))
    cursor = conn.cursor()
    query = "select * from airport where airport_name = %s and 
airport_city = %s"
    cursor.execute(query,(name,city))
    data1 = cursor.fetchall()
    cursor.close()
    if data1:
        error = "Airport Already exits."
        return redirect(url_for('addAirportPage', error=error))

    cursor = conn.cursor()
    query = 'insert into airport values (%s, %s)'
    cursor.execute(query, (name, city))
    conn.commit()
    cursor.close()
```

6. View all the booking agents:
   The staff will be able to view : Top 5 booking agents based on number of tickets sales for the past month and past year. Top 5 booking agents based on the amount of commission received for the last year.
    - Top 5 agents based on tickets sales:

```
        daterange = request.form['range']
        airline = getStaffAirline()

        #datrange specify the past 1 month or year
        cursor = conn.cursor()
        query = 'select email,count(ticket_id) as sales \
        from booking_agent natural join purchases natural join ticket \
        where purchase_date >= date_sub(curdate(), interval 1 ' + 
daterange + ') \
        and airline_name=%s group by email order by sales DESC limit 5'
        cursor.execute(query, (airline))
        data = cursor.fetchall()
        cursor.close()
```

- Top 5 agents based on commissions:

```
        airline = getStaffAirline()

        cursor = conn.cursor()
        query = "select email,sum(flight.price)*0.1 as commission \
        from booking_agent natural join purchases natural join ticket
natural join flight \
        where purchase_date >= date_sub(curdate(), interval 1 year) and
airline_name=%s\
         group by email order by commission DESC limit 5"
        cursor.execute(query, (airline))
        data = cursor.fetchall()
        cursor.close()
```

7. View frequent customers:
   The staff will also be able to see the most frequent customer within the last year. Also, the staff can see the number of tickets a specified customer bought in the past on their airline.
   - Specified customer ticket on the staff's airline:
     If it's not a customer email, an error message will be popped.

```
 airline = getStaffAirline()
        customer = request.form['email']

        cursor = conn.cursor()
        query1 = "select * from customer where email = %s"
        cursor.execute(query1,customer)
        data1 = cursor.fetchone()
        error = request.args.get('error')
        cursor.close()

        if not data1:
            error = "Not a customer email, please enter a customer
email."
            return redirect(url_for('viewCustomersPage',error = error))
        else:
            cursor = conn.cursor()
            query = 'select distinct flight_num from purchases natural
join ticket where airline_name = %s and customer_email=%s'
            cursor.execute(query, (airline, customer))
            data = cursor.fetchall()
            cursor.close()
```

   - The most frequent customer:

```
        airline = getStaffAirline()
        cursor = conn.cursor()
        query = 'select customer_email, count(ticket_id) as
customerpurchases \
                from purchases natural join ticket \
                where airline_name= %s \
                and purchase_date >= date_sub(curdate(), interval 1 year)
group by customer_email \
                having customerpurchases \
                  >= all (select count(ticket_id) \
                  from purchases natural join ticket \
                  where airline_name = %s \
                  and purchase_date >= date_sub(curdate(), interval 1
year) GROUP by customer_email)'
        cursor.execute(query, (airline, airline))
        data = cursor.fetchall()
cursor.close()
```

8. View reports:
   Total amounts of tickets sold could be searched based on the range of dates/last year/last month. Month wise tickets sold are shown in a bar chart.
   - Month wise tickets in the bar chart:

```
        airline = getStaffAirline()
        currentmonth = datetime.datetime.now().month
        monthtickets = []

        cursor = conn.cursor()
        for i in range(0, 12):
            query = 'select count(ticket_id) as sales \
            from purchases natural join ticket \
            where year(purchase_date) = year(curdate() - interval ' +
str(i) + ' month) \
            and month(purchase_date) = month(curdate() - interval ' +
str(i) + ' month) \
            and airline_name=%s'
            cursor.execute(query, (airline))
            data = cursor.fetchall()
            salemonth = ((currentmonth - (i+1)) % 12) + 1
            # print (data[0]['sales'])
            monthtickets.append([data[0]['sales'], salemonth])
        cursor.close()
```

- Search based on time range:

```
airline = getStaffAirline()
        begintime = request.form['begintime']
        endtime = request.form['endtime']

        if not validateDates(begintime, endtime):
            error = 'Invalid date range'
            return redirect(url_for('viewReportsPage', error=error))

        cursor = conn.cursor()
        query = 'select count(ticket_id) as sales \
                from purchases natural join ticket where airline_name=%s\
                and purchase_date between %s and %s'
        cursor.execute(query, (airline, begintime, endtime))
        data = cursor.fetchall()
        cursor.close()
```

- Search based on month/year:

```
        airline = getStaffAirline()
        daterange = request.form['range']

        cursor = conn.cursor()
        query = 'select count(ticket_id) as sales \
        from purchases natural join ticket where airline_name=%s \
        and purchase_date >= date_sub(curdate(), interval 1 ' + daterange
+ ')'
        cursor.execute(query, (airline))
        data = cursor.fetchall()
        cursor.close()
```

9. Comparison of Revenue earned:
   Comparing the direct sales and indirect sales in the past 1 month using pie charts.

```
 # query for direct purchase revenue (last month)
        query1 = "select sum(flight.price) as rev\
                from purchases, ticket, flight\
                where purchases.ticket_id = ticket.ticket_id \
                and ticket.flight_num = flight.flight_num\
                and ticket.airline_name = flight.airline_name\
                and flight.airline_name = %s\
                and purchases.purchase_date between DATE_SUB(curdate(),
INTERVAL 1 MONTH) and curdate()\
                and purchases.booking_agent_id is null"
        cursor.execute(query1,str(airline_name))
        direct_revenue = cursor.fetchone()['rev']
        # query for indirect purchase revenue (last month)
        query2 = "select sum(flight.price) as rev\
                from purchases, ticket, flight\
                where purchases.ticket_id = ticket.ticket_id \
                and ticket.flight_num = flight.flight_num\
                and ticket.airline_name = flight.airline_name\
                and flight.airline_name = %s\
                and purchases.purchase_date between DATE_SUB(curdate(),
INTERVAL 1 MONTH) and curdate()\
                and purchases.booking_agent_id is not null"
        cursor.execute(query2,str(airline_name))
        indirect_revenue = cursor.fetchone()['rev']
```

10. ViewTop destinations:
    Show the top 3 most popular destinations for the past 3 months and the past 1 year.
    - Past 3 months

```
username = session['username']
        message = request.args.get('message')
        cursor = conn.cursor()

        queryGetairline = "SELECT airline_name FROM airline_staff WHERE
username= %s"
        cursor.execute(queryGetairline, username)
        airline_name = cursor.fetchone()['airline_name']
        # query top destination for the past 3 months
        query1 = "select count(ticket.ticket_id) as cnt,
airport.airport_city as city\
                from airport,flight,ticket,purchases\
                where airport.airport_name = flight.arrival_airport\
                and flight.flight_num = ticket.flight_num\
                and flight.airline_name = %s\
                and purchases.ticket_id = ticket.ticket_id\
                and purchases.purchase_date between DATE_SUB(curdate(),
INTERVAL 3 MONTH) and curdate()\
                group by city \
                order by cnt DESC limit 3"
        cursor.execute(query1,airline_name)
        data1 = cursor.fetchall()
```

- Past 1 year:

```
 query2 = "select count(ticket.ticket_id) as cnt, airport.airport_city as
city\
                from airport,flight,ticket,purchases\
                where airport.airport_name = flight.arrival_airport\
                and flight.flight_num = ticket.flight_num\
                and flight.airline_name = %s\
                and purchases.ticket_id = ticket.ticket_id\
                and purchases.purchase_date between DATE_SUB(curdate(),
INTERVAL 1 YEAR) and curdate()\
                group by city \
                order by cnt DESC limit 3"
        cursor.execute(query2,airline_name)
        data2 = cursor.fetchall()
```

11. Logout: The login page will be shown.

```
@app.route('/logout')
def logout():
  session.pop('username')
  return redirect('/login')
```