# How To Run And Install The Database And Its Instructions

## Table of Contents

---

---

## Project Overview

### Airline Management System Documentation

This document aims to help and guide the clients of the database regarding how to install, run, and interact with the database's RESTful API and to provide a good sense of how the ins and outs of this system work along with an in-depth explanation of each functionality's required parameters.

This project uses:

- **Necessary modules and libraries**: Flask, JWT, Bycrypt, Logging, Psycopg2, Time, Datetime
- **Necessary software**: An IDE for Python, Docker, Postman, and A client to connect to the SQL DBMS.

**Important Note**:

All users should know that to access and interact with some of the functionalities, a previous login will be required, done through the Login endpoint, to verify the user's access authorizations. From this point till the end of the documentation, the functionalities that require admin, passenger, or crew member credentials will be added below their respective functionalities tab and all functionalities listed below the general functionalities tab do not require any login to be used.

---

---

## User Roles Functionalities and API Endpoints

### Administrator Functionalities

1. **Add Admins**
   - **Endpoint**: `POST /cloud_query/admin`
   - **Input Parameters**: "username, email, password" (of the user we want to add as admin).
   - **Result**: The result in case of successful insertion would be the created Admin´s "user_id".
   - **Note**: It is important to note that since only admins can add other admins we need to use the username, email, and password of the person we want to add as admin after logging in.
2. **Add Airport**
   - **Endpoint**: `POST /cloud_query/airport`
   - **Input Parameters**: "city, name, country"
   - **Result**: This functionality returns the "airport_id" of the created airport.
   - **Note**: Using this functionality we can add airports to our database.
3. **Add Flight**
   - **Endpoint**: `POST /cloud_query/flight`
   - **Input Parameters**: "departure_time, arrival_time, existing_seats, ,airport_dep, airport_arr"
   - **Result**: This functionality returns the "flight_code" of the created flight.
   - **Note**: The functionality is used to add flights to the database it is to note that "airport_dep" refers to the departure airports and the "airport_arr" refers to arrival airports.
4. **Add Schedule**
   - **Endpoint**: `POST /cloud_query/schedule`
   - **Input Paramets**: "flight_code, date, crew_id, ticket_price"
   - **Result**: In case of a successful insert it should return the "schedule_id" of the created schedule.
   - **Note**: To create Schedules it will be necessary to have previously created a flight and a crew to be able to associate their IDs with the respective flight schedule.
5. **Create Crew**
   - **Endpoint**: `Post /cloud_query/crew`
   - **Input Parameters**: Just an admin login is required to create the respective crews.
   - **Result**: The result in case of a correct insertion should be the "crew_id".
   - **Note**: In this particular functionality we use the admin credentials to Login and by doing this we automatically generate a "crew-id" which will be later attached to the crew members of the referred crew.
6. **Add Crew Members**
   - **Endpoint**: `Post /cloud_query/crew_member`
   - **Input Parameters**: "username, email, password, role, crew_id"
   - **Result**: Similar to the add passenger and add admin functionalities the result after a successful insertion should be the registered "user_id".
   - **Note**: To add a crew member, a crew has to be previously created with the Add Crew functionality since it is necessary to associate the crew members with their respective crew using the "crew_id" It is also to note that the possible role entries are either "Pilot" or "Flight_attendant".
7. **Get Crew**

- **Endpoint**: `GET /cloud_query/crew`
- **Input Parameters**: Similar to the create crew functionality no input would be required a simple login of an admin would suffice to get all the Crew's information.
- **Result**: The result would be the demonstration of all Crews information which includes the "user_id" of the admin who created the crew, the "user_id" of the crew chief (if non-existent this would be filled with NULL instead), and finally the respective "crew_id".
- **Note**: Demonstrates all the information from the Crew table.

8. **Financial data**
   - **Endpoint**: `GET /cloud_query/financial_data`
   - **Input Parameters**: Like some of the functionalities mentioned before this one also does not require any special input parameters only an admin login is necessary.
   - **Result**: The result of this functionality in case of a successful connection would have this format: "Credit Card, Debit Card, MBway, Flight_code, total" where each of the illustrated keys would be mapped to their respective numbers.
   - **Note**: This functionality aims to get all the information of all payments made in different methods associated with their respective flight_code.
9. **Add Supervisor**
   - **Endpoint**: `Post /cloud_query/crew_supervisor`
   - **Input Parameters**: "crew_id, crew_member"
   - **Result**: In case of a successful insertion the message "Supervisor was added with success. Crew member (user_id) supervisor (crew_id)."
   - **Note**: It is important to note only the admins who have created a respective crew can add a supervisor for it.

## Passenger Functionalities

1. **Booking Flight**
   - **Endpoint**: `POST /cloud_query/Book_flight`
   - **Input Parameters**: "flight_code, date, ticket_quantity, seat_id"
   - **Result**: In case of a successful insertion the result would be a message that confirms the insertion and directs the client for payment with the associated "booking_id".
   - **Note**: It is of great importance to note that in this particular functionality the "seat_id" will correspond to an array of seats that the client is planning to book. The format would be provided in the "Input example for Postman document".
2. **Info Booking**
   - **Endpoint**: `GET /cloud_query/Info_booking`
   - **Input Parameters**: There are none required.
   - **Result**: The result would demonstrate all bookings information which includes: "booking_id, flight_code, flight_date, amount to pay, and ticket_quantity".
   - **Note**: This functionality shows all the existing bookings of the member whose access credentials were used to login.
3. **Make Payment**
   - **Endpoint**: `POST /cloud_query/make_payment`
   - **Input Parameters**: "booking_id, method, payment_amount"
   - **Result**: In this functionality, the result would be a confirmation message that indicates the paid amount and used method associated with the "booking_id", in case of a full payment message "The payment is completed" will show and in case the client wants to use another method of payment the message "The remaining amount to complete booking payment is (amount) will show up.
   - **Note**: Important to note that the accepted payment methods are Credit Card, Debit Card, and MBway.
4. **Add Tickets**:
   - **Endpoint**: `POST /cloud_query/Tickets`
   - **Input Parameteres**: "booking_id, name, TIN(tax-identification-number)"
   - **Result**: In case of correct input parameters and successful insertion the message "The tickets were created successfully" would be shown.
   - **Note**: In order to reduce the client workload in case of multiple bookings made by the same client for different people, multiple names and TIN can be added in the same request. EXAMPLE: "name" = ["João Vieira","Mariana Dias] and "TIN" = ["123456789","987654321"].

## Crew Member Functionalities

1. **Get Work**
   - **Endpoint**: `Get /cloud_query/crew_member/get_work`
   - **Input Parameters**: The only requirement would be a crew member login.
   - **Result**: "arrival_airport, arrival_time, crew_id, date, departure_airport, departure_time, flight_code"
   - **Note**: This is the only Functionality that is accessible only by crew members and aims to provide the crew members with all essential information related to their flights such as departure times, arrival times, crew IDs, flight codes of their flight, etc.

## General functionalities

1. **Login**
   - **Endpoint**: `PUT/cloud_query/user`
   - **Input Parameters**: "username, password"
   - **Result**: The result in case of a successful login would be an authentication token which is necessary to access all role-specified functionalities.
   - **Note**: Important to note that all of the authentication tokens that are generated after each login have an expiry time of 1 hour and after the time runs out another login will be required in order to generate a new Token.
2. **Add Passenger**:
   - **Endpoint**: `POST /cloud_query/passenger`
   - **Input Parameters**: "username, email, password"
   - **Result**: In case of a successful insertion the "user_id" of the created user will be returned.
   - **Note**: A very simple functionality that creates and adds passengers in their respective tables in the database.
3. **Check Available Routes**
   - **Endpoint**: `GET /cloud_query/check_routes`
   - **Input Parameters**: No input OR Origin_airport OR Origin_airport and Destination_airport OR Destination_airport.
   - **Result**: The different types of outputs are explained more in-depth in the note below but the format of a successful return would be "destination_airport, flight_code, origin_airport, schedule"
   - **Notes**: The general idea In case of no input all available flights between all origins and all destinations airports will be displayed. In the case of only the Origin airport inserted all possible destination airports will be displayed and finally in the case of only the Destination airport inserted all possible origin airports that have available flights to that destination will be displayed.
4. **Check Seats**
   - **Endpoint**: `GET /cloud_query/check_seats`
   - **Input Parameteres**: "flight_code, date"
   - **Result**: Simply returns a list containing all seats that are still available(not booked) in this format provided in the "Postman input example document".
   - **Note**: A functionality accessible by all users that shows the available seats on a specific flight and date.

5. **Top N destinations**
    - **Endpoint**: `GET /cloud_query/top_destinations`
    - **Input Parameters**: No input parameters are required.
    - **Result**: The result would demonstrate a list of all last 12 months' destination airports that have received the most amount of passengers and flights during each month.
    - **Note**: This functionality aims to highlight the most visited destinations in the last 12 months showing the busiest airports of each month.
6. **Top Routes**
    - **Endpoint**: `GET /cloud_query/top_routes`
    - **Input Parameters**: No input parameters are required.
    - **Result**: The result of this functionality would show the most traveled routes in the past 12 months which are identified with their flight_id and the number of passengers on board.
    - **Note**: This functionality aims to show the top routes of the past 12 months by showing the flight id which has the most passengers each month.

---

---

## Setup Instructions

1. Clone the repository:
   ```bash git clone https://github.com/franciscamateusPt05/Cloud-Query.git

2. Install the dependencies: A simple installation of the modules mentioned in the Project Overview is needed. ```bash pip install (requirements)

3. Setup the database:

   Note:

   While accessing the database it is important to note that the user used for accessing the database must have the necessary permissions to access and modify the tables a superuser would be advised). Login to the database using your preferred DBMS client(pgadmin4, dbeaver, etc)

4. Install Docker:

   Note: The script creates a container that contains the database and the API.

   ```
   https://www.docker.com/products/docker-desktop/
   ```

   Install docker and activate docker scripts included in the file "docker-compose-python-psql.sh"

5. Run the code:

   ```
   cloud_query.py
   ```

6. Now for a final step you can use Postman to interact with the API and the database using the endpoints provided previously.

7. (Good Luck!!!!)

---

---

## Authors

-Ramyad Raadi: uc2023205631@student.uc.pt

-Francisca Mateus: uc2023212096@student.uc.pt

-Pedro Silva: uc2023235452@student.uc.pt
   ```