COMP3011 Web Services and Web Data:

A RESTful Airline API for

Flight Search and Booking System

|  |  |
| --- | --- |
| Student Name | Chemseddine-Wassim Benimoussa |
| Student Email | sc20cwb@leeds.ac.uk |
| Student ID: | 201429246 |
| Git repository: | https://github.com/sc20cwb/CW2WebData |

# **Summary**

This projects consists of developing one of the Airlines web-based API’s. The airline API is part of the flight search and booking system, for which the architecture was previously set out in addition to the framework’s technical specifications. This documents intends to discuss the implementation of the airline API while reviewing some design choices made during the previous report and explaining the endpoints integrations.

Using a RESTful API, the aim of this project is to simulate an airline company that can provide flights to the flight search and booking system. A database which stores flights, airports, airline, and booking details is created for the airline API. Two endpoints are designed for the communication between the airline and the federal aviation authority (FAA). Each of these endpoints regroups at least 2 functionalities for transferring data between the airline and federal aviation authority. The airline provides the FAA with the available flight’s details. The FAA supplies the airline with booking details when a booking is made or cancelled.

Some changes have been made to the initial design that was documented on the previous report. The database is modified to allow more flexibility while ensuring data accuracy and high performance of the API. The structure of the communication has also been altered, new endpoints were added while old ones were moved to between two other APIs the aggregator and FAA.

In conclusion an airline web-based API is implemented while fulfilling all the requirements of the project ensuring the fast and efficient communication between the airline and the federal aviation authorities.

Contents

[**Summary** 2](#_Toc134517332)

[**Chapter 1: Introduction** 1](#_Toc134517333)

[**1.1 Objectives and Taks:** 1](#_Toc134517334)

[**1.2 Requirements:** 1](#_Toc134517335)

[**Chapter 2: Database** 2](#_Toc134517336)

[**2.1 Database changes and justification:** 2](#_Toc134517337)

[**Chapter 3: API Architecture and Endpoints** 3](#_Toc134517338)

[**3.1 API Endpoints:** 3](#_Toc134517339)

[**Flights Endpoint:** 3](#_Toc134517340)

[**Bookings Endpoint:** 3](#_Toc134517341)

[**3.2 API changes and justification:** 4](#_Toc134517342)

[**Chapter 4: Testing** 5](#_Toc134517343)

[**Chapter 5: Conclusion** 6](#_Toc134517344)

[**Bibliography:** 7](#_Toc134517345)

[**Appendix A** 8](#_Toc134517346)

[**A.1 Old Database:** 8](#_Toc134517347)

[**A.2 Old API architecture and Endpoints:** 8](#_Toc134517348)

# **Chapter 1: Introduction**

This project consists of designing a RESTful web-based API for flight searching and booking system. A team of 13 students is made to develop this system. Each student develops and deploy his API while ensuring efficient communication with other API’s.

This section of the document discusses the requirements of the airline web-based API, the achieved tasks, and objectives of the project to complete and deploy the API. The Airline company implemented in this report is called API Airlines. The API Airlines is deployed to the web through PythonAnywhere which is an integrated development environment mainly used for hosting python project online (pythonanywhere, 2023).

This API is deployed to the following URL: <https://sc20cwb1.pythonanywhere.com/> . The admin interface can be accessed by the following usernames and passwords:

* Username: ammar
* Password: admin2023

## **1.1 Objectives and Taks:**

This following subsection details the achieved objectives and tasks to finalise the development of the API:

* Decide on API endpoints error codes and URLs.
* Implement the API endpoints between the airline and the FAA.
* Test the endpoints locally using test units.
* Deploy the API to PythonAnywhere.
* Test the endpoints communication with FAA.

## **1.2 Requirements:**

A list of requirements for the Airline web-based API is detailed on this section. Each of these requirements is achieved through the development of the API:

* Database: The airline API had an initial local database which is an SQLite database. Upon the deployment of the service, the database has been changed to MySQL database which provides more performance while ensuring secure access.
* One-way Flights: The API is made abstract by only allowing one-way flights which means no connections or return flights.
* Managing Bookings: The airline API save bookings with passport details and flight code. These bookings can be cancelled at any time.
* Managing Flights: Every time a flight is created or modified through the admin interface a communication with the FAA starts by posting the flight details. Deleting flights is also possible, which allows a constant synchronization between the API and FAA.
* Performance: Low latency and fast database queries are ensured using MySQL database which optimize API performance.
* Modularity and maintainability: The web service is made as modular as possible for future improvements and new endpoints.
* Management: The management of the data of the airline API is made through the admin interface.

# **Chapter 2: Database**

Figure 2.1.1 describes the structure of the airline database. The general structure did not change from the initial design. The database is constituted of four different models. Firstly, The airline class airline Code as the primary key, it stores the API Airlines information as name of the airline, country, and phone number. Secondly, The airport table which stores different airport instances by saving the name, city, and country of the airport. Thirdly, the booking model, which stores the booking reference, passport number of the customer and finally the flight code. Finally, the flight class, this table is the largest table of the database. It is used to save the flight details as the flight code which is the primary key of the model, the departure and arrival airports, the departure and arrival dateTime, duration time, the price of the flight, available number of seats and finally the airline code.

## **2.1 Database changes and justification:**

Figure A.1.1 shows the structure of the old database, this subsection will enumerate the changes made to the old database with a justification for each change:

* Change of Primary Key: In each of the four tables the primary key has been changed from the default id Integer field that increments for each instance to a more significant naming with a CharField. This change is made to preserve uniqueness of the data, so when the flight instances or bookings are communicated from the airline API to the FAA and vice versa, having a unique integer field that increments for each instance creates synchronization issues from different airlines. As an example: if we have 2 airlines and both creates two flights both flights have the same ID = 1. Having two different flights with the same ID shared to the FAA synchronization problems. It goes the same of the booking class. So the best way to fix this is to create unique character fields identifiers for each class.
* Change of the relationship between the Flight and Airport Model. In the initial design, a one-to-one relationship was implemented, in fact this was a mistake during the old design of the system as the relationship should be a one-to-many relationship because each airport can have many flights instead of just one.

Graphical user interface, text

Description automatically generated

Figure 2.1.1: Database Architecture of the API Airlines.

# **Chapter 3: API Architecture and Endpoints**

Figure 3.2.1 represents the new design of the airline API and the connection that the API has with the federal aviation authorities. The old architecture that is described on Figure A.2.1, was judged to be inefficient during development phase of the project so changes were made to it to facilitates and optimize the API workload.

## **3.1 API Endpoints:**

This new design englobes 2 main endpoint for the airline API. Firstly, the flights endpoint which regroups 3 methods “POST”, “PATCH”, and “DELETE”, these methods are used for handling flight requests. Secondly, the bookings endpoint which regroups 2 methods “POST” and “DELETE” which are used for handling . Each method is described in this section:

### **Flights Endpoint:**

* **Post Flight:** The “POST” and “PATCH” methods are used to send flights details from the airline to the aviation authority. In the models for the database the save() method has been overridden for the flight class. As soon as a new instance (flight) of this class in created (saved) or an old flight is being modified it calls the save() method of the class. The new save() method firstly checks if the instance exists on the database or no by querying the object using the flight code. If the instance does not exist it sends a “POST” request to the FAA with all the flight details, if the communication in achieved effectively it saves this instance in the airline server database. If the instance already exists in the database, it sends a “PATCH” request instead to the FAA.
* **Delete Flight:** The “DELETE” method is used when a flight instance is not needed anymore. The flight class of the database models has a default delete() method; in the implementation of the API this method has been overridden. The delete() method is called once a database instance is requested to be deleted. Upon the deletion call, the delete() method sends a “DELETE” request to the FAA by providing the flight code through the URL as follow:

url = f'http://sc20osc.pythonanywhere.com/flights/?flight\_code={self.flight\_code}'

### **Bookings Endpoint:**

* **Create Booking Airline:** In this functionality, the airline API receives a “POST” request with all booking details. These information are taken from the request, they are processed and used for the creation of a booking object. The function checks first if the booking reference does not already exist in the database then verifies if the flight of the booking exists. After that, if everything is valid the object is then saved to the database. Once a booking is saved, the available seat number of that flight is decremented by one which will call the save() method described previously to save the changes. This call will result in the “PATCH” request described in the flight endpoint.
* **Delete Booking Airline:** For this function, a “DELETE” request is received by the API server with the booking reference to delete the booking. This functionality is used if the client wants to cancel his booking. The booking reference is supplied through the URL the same way as in the Delete Flight functionality. The function tries to retrieve the booking from the database, if the booking exists, it will delete it from the database than increase the available number of seats before returning the https response status code with comment.

## **3.2 API changes and justification:**

Figure 3.2.1 shows the new API architecture and endpoints compared to the old one in figure A.2.1. The list of changes and their justification is explained in this section:

* Initially the Airline API was used as a mean of communication between both the Flight aggregator and the FAA. In the new design the airline is swapped with FAA to make the FAA the only communication with aggregator. This change was decided because having the aggregator communicate with multiple airlines at the same time is not efficient and can relate to synchronization issues of the data.
* On the old design the aggregator gets the flights from the airlines directly while sending new bookings and cancellation requests to the airline. The new design is made in a way that each airline updates the FAA on the available flights. Instead of receiving the booking from the aggregator to the airline, the booking is sent from aggregator to FAA then from FAA to the specific airline because if the bookings are sent to the airline directly there is no way to know if the created booking ref is unique with all the airlines as the aggregator does not have a local database.
* Another point is that the aggregator will request the flights from the FAA as it contains all the flights for all the airlines. If the booking is sent from the aggregator to the airline, then from airline to FAA, there will be a time lapse where the FAA won’t have information that a booking was made and the available seats will not be update on time, so if another client is requesting a booking on that time lapse there is chance he gets the previously booked seat. The new design prevents this from happening.

Diagram, timeline

Description automatically generated

Figure 3.2.1: API architecture and Endpoints.

# **Chapter 4: Testing**

This section discusses the testing procedures that were used for this project. The testing was conducted using two types of testing. Firstly, Unit tests were implemented for each endpoint method used on the project. Secondly, manual testing that was performed mainly through the admin interface of the API.

## **4.1 Unit Tests:**

## **4.2 Manual Testing:**

# **Chapter 5: Conclusion**

# **Bibliography:**

* <https://www.pythonanywhere.com/>

# **Appendix A**

# **A.1 Old Database:**

Graphical user interface, text, application

Description automatically generated

Figure A.1.1: Old database design.

# **A.2 Old API architecture and Endpoints:**

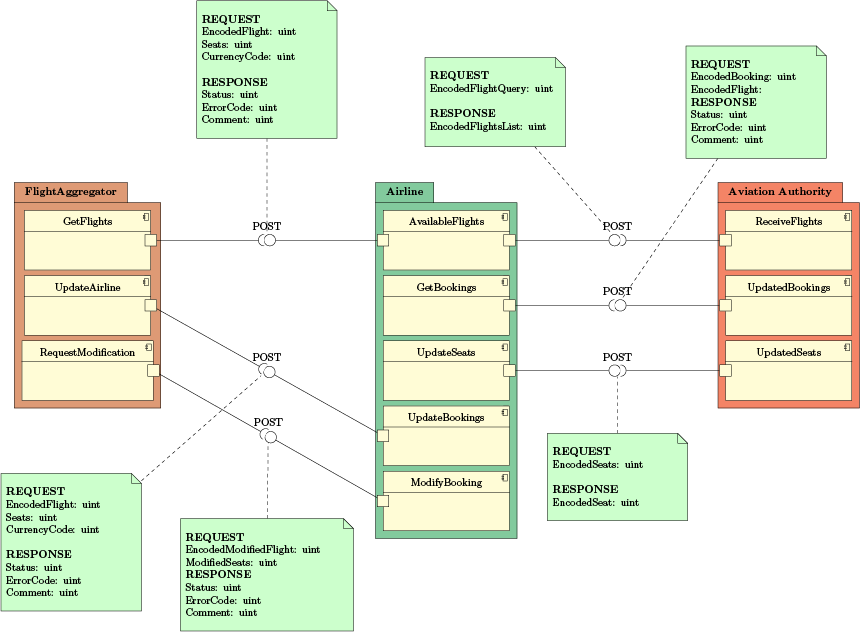


Figure A.2.1: Old Airline API endpoints.