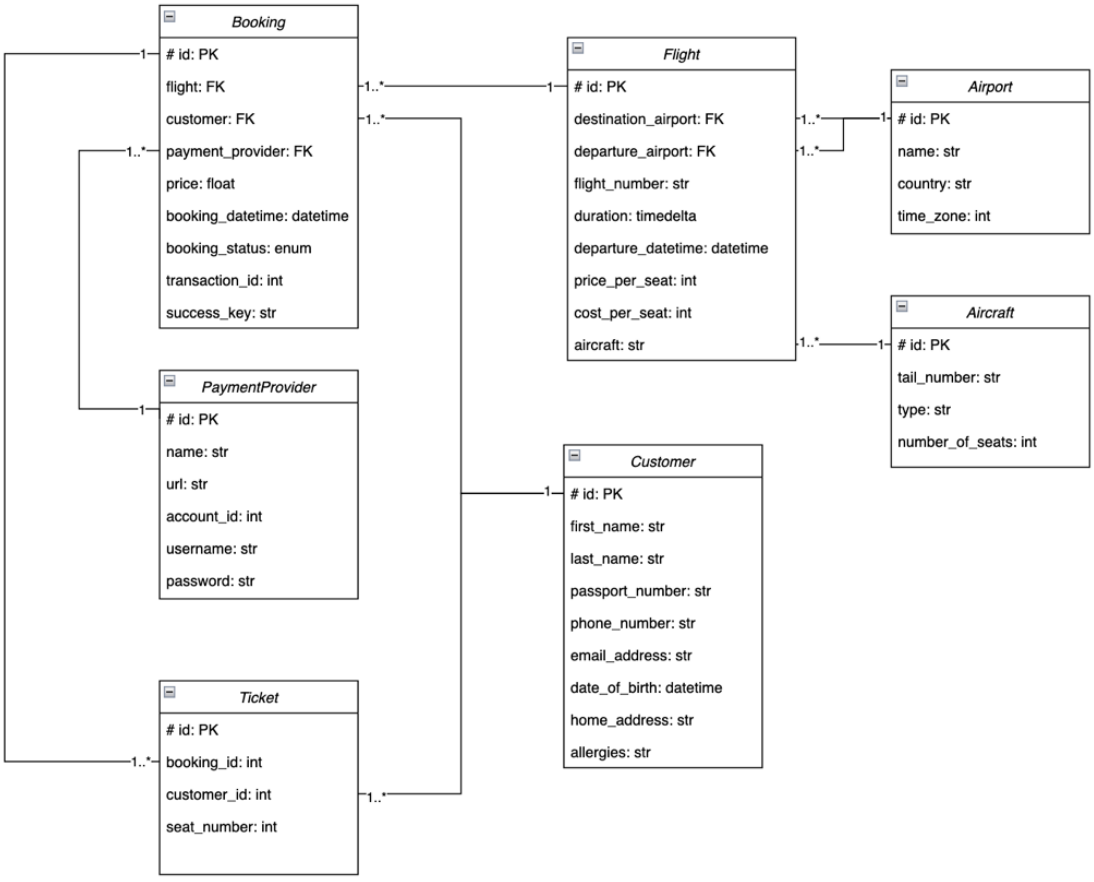
Airline API Implementation Explanation

Alyaqdhan Almaskari (fy19ayyn)

Implementing the Airline company API required a good understanding of the Django framework. Working with a team enabled us to achieve a well-structured API and ensure the connection between these APIs is smooth and reliable. Based on the documentation specifications and requirements, we worked as a team to create databases and endpoints that allow us to have a fully functional booking system using Airlines, Aggregators, and payment service providers. As we discussed while writing the documentation, the implementation of one of the Airlines API companies has been assigned to me.

Database models:

The implementation of these models reflected the requirements that can be found in the Design Specification document:



This figure illustrates the database complements suitable for a distributed web-based booking system.

API endpoints:

The most important part of any API implementation is the endpoints and how they work based on required specifications.

1. **Flights list:**

   This endpoint returns all available flights with specific departure and destination airports and departure dates/times. It receives a ‘get’ request with JSON data continuing the airports and departure date/time. It returns a list with all flights according to the given data with a ‘200 OK’ response if everything is as expected. If there is missing data, it returns ‘404 NOT FOUND’. If there are any other errors, it returns ‘503 SERVICE UNAVAILABLE’.

2. **Flight details:**

   This endpoint is used to get specific flight information. It receives a ‘get’ request with JSON data containing flight id. For a successful request, it returns ‘200 OK’ with flight details. Return ‘404 NOT FOUND’ if data is not found. If an unexpected error occurs, it returns ‘503 SERVICE UNAVAILABLE’.

3. **Book flight:**

   This endpoint allows users to book a flight. Users can use ‘post’ requests with JSON data containing flight id and customer details, including first name, last name, passport number, phone number, email addresses, date of birth, home address [optional], and allergies [optional].

   This request checks if the customer exists using their passport number. A new customer is created if they do not exist in the database. If already in the system, their details are used. Then, a flight booking is created for that customer and saved in the database.

   Successful creation returns a ‘201 CREATED’ response to the booking id created, booking status, price, and customer details. If any of the data provided is invalid, return ‘400 BAD REQUEST’. Other unexpected errors return ‘503 SERVICE UNAVAILABLE’.

4. **Update booking:**

   This endpoint allows users to change their or their booking details. It expects a ‘put’ request with JSON data, including the details they want to update (departure date/time and customer details). These are optional fields, and users can provide the details of what they are changing.

It returns a ‘200 OK’ response if the details update has been successful. Returns ‘404 NOT FOUND’ if any required fields are missing. If any other unexpected error occurs, return ‘503 SERVICE UNAVAILABLE’

5. **Confirm booking:**

   This endpoint updates the booking status to ‘Confirmed’. It receives a ‘put’ request with JSON data containing the success key and booking id. It returns the booking id, undated booking status and tickets in the booking with a ‘200 OK’ response for a successful request. It returns ‘404 NOT FOUND’ if their missing data. It returns ‘503 SERVICE UNAVAILABLE’ if there any other error occurs.

6. **Cancel booking:**

   This endpoint expects a ‘put’ request with JSON data containing the booking id to cancel. This endpoint changes the booking status to ‘Cancelled’ if the provided id is valid and returns ‘200 OK’. If the booking id is not found, it returns ‘404 NOT FOUND’. For any other unexpected errors, it returns ‘503 SERVICE UNAVAILABLE’.

7. **Pay booking:**

   Users can pay for their booking using this endpoint. It expects a ‘post’ request with JSON data containing the booking id and payment service provider id. It checks if the booking and PSP exist and then generates a random transaction id for this booking. If the process succeeds, it returns ‘201 CREATED’. If any of the required fields are not found, return ‘404 NOT FOUND’. If any other errors occur, it returns ‘503 SERVICE UNAVAILABLE’.

8. **Payment service providers:**

   This endpoint allows users to check all available PSPs in this airline. It receives a ‘get’ request to return a list of all PSPs and a ‘200 OK’ response for a successful request. Otherwise, it returns ‘503 SERVICE UNAVAILABLE’ if any other error occurs.