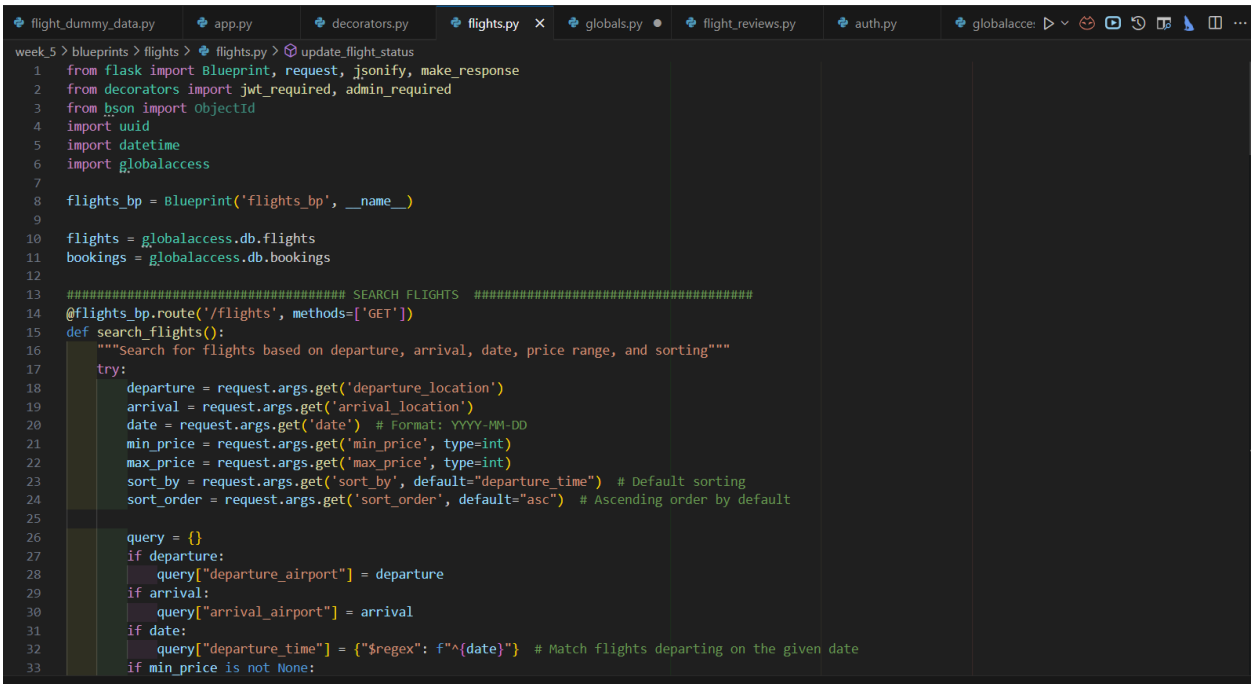
Airline Booking System

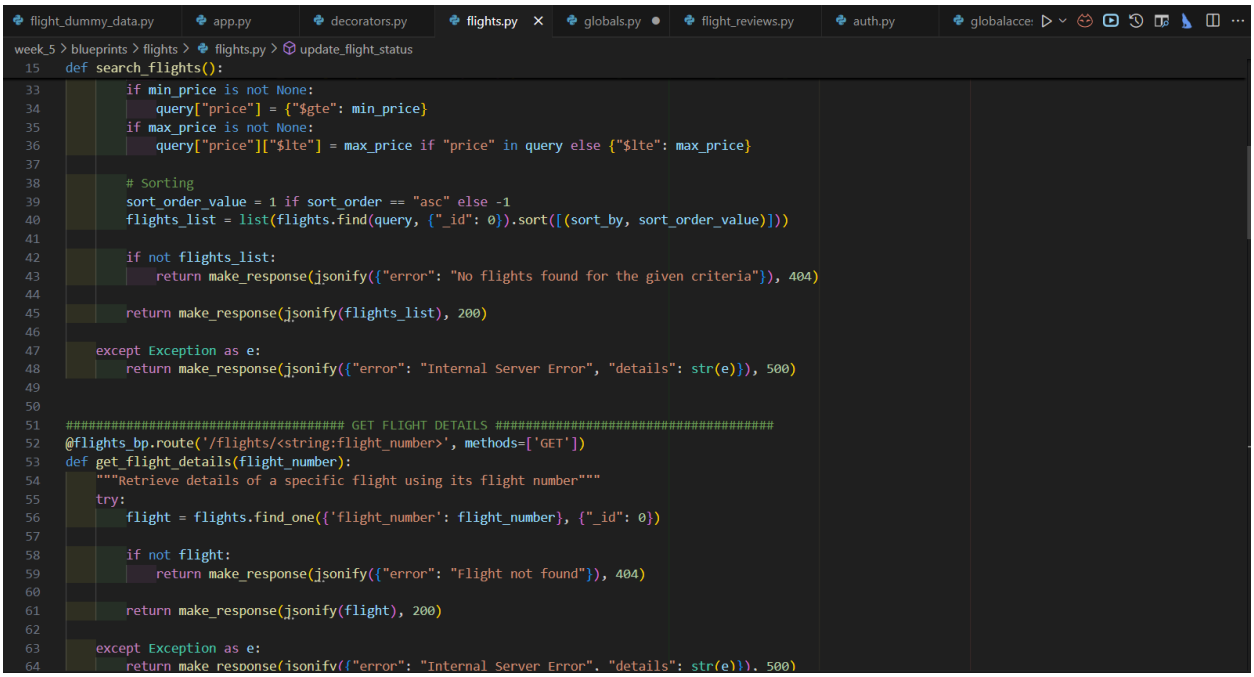
Introduction:

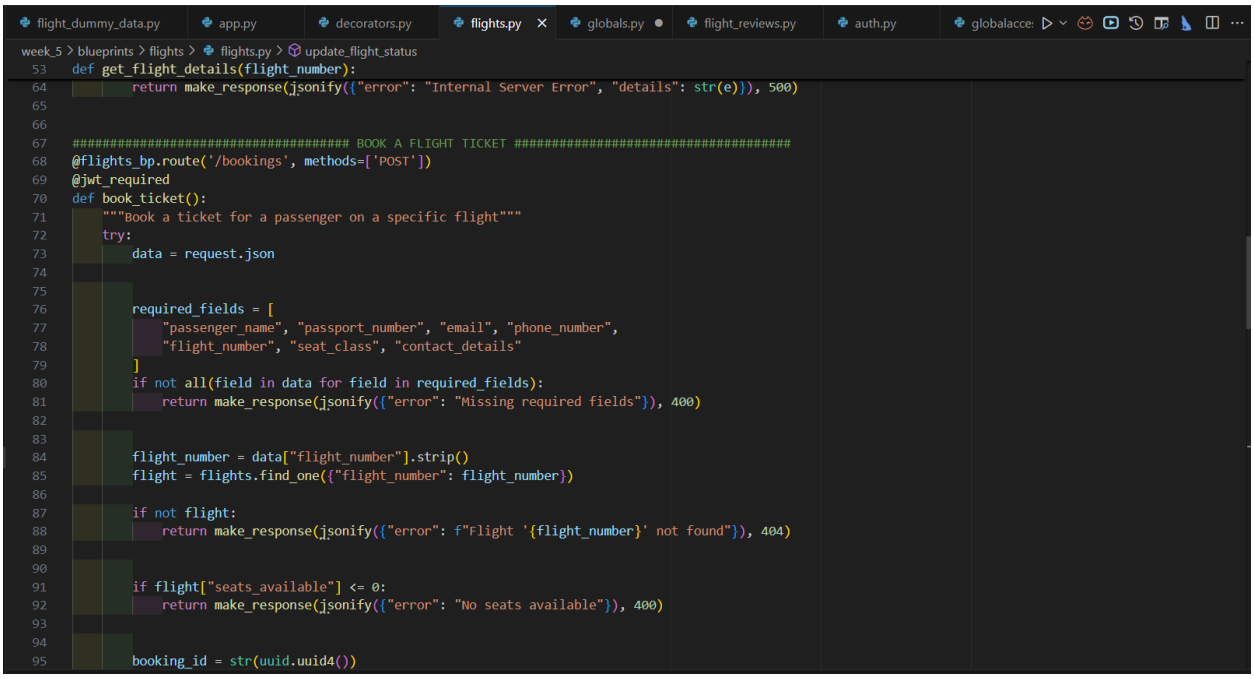
The Airline Booking System API allows users to search for flights, book tickets, manage bookings, and leave reviews. Administrators have the ability to modify reservations, remove reviews, and amend flight status. Postman testing, implementation, and API endpoints are covered in this report.

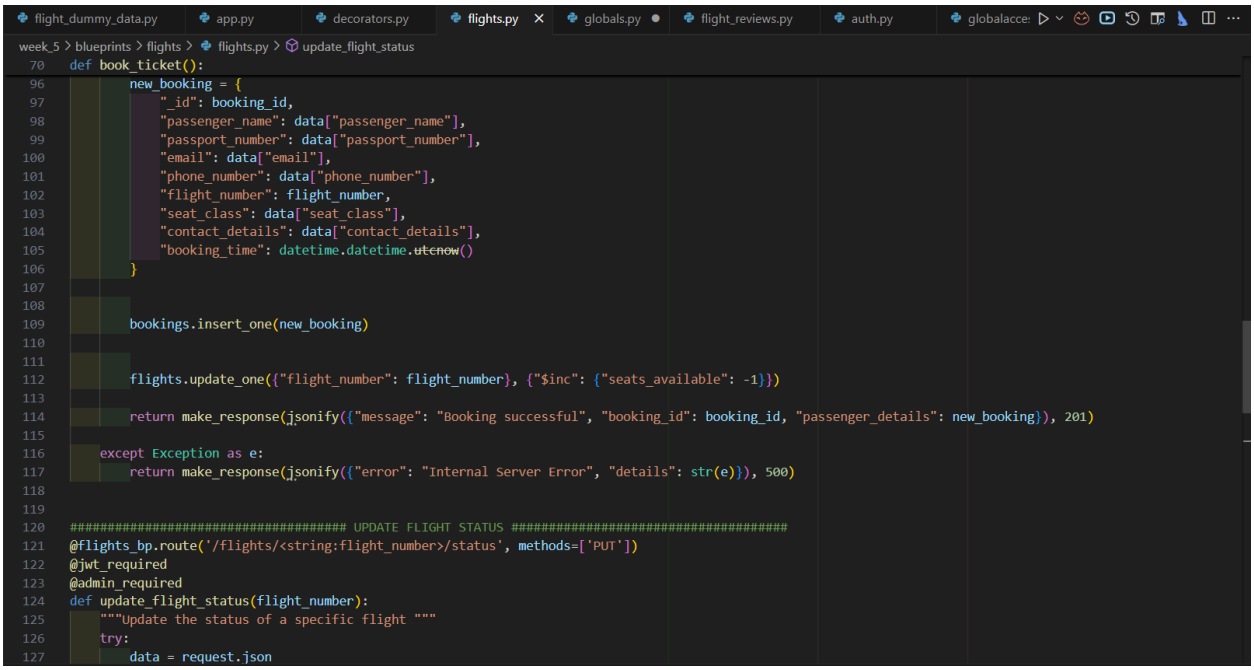
Backend API code Listing:

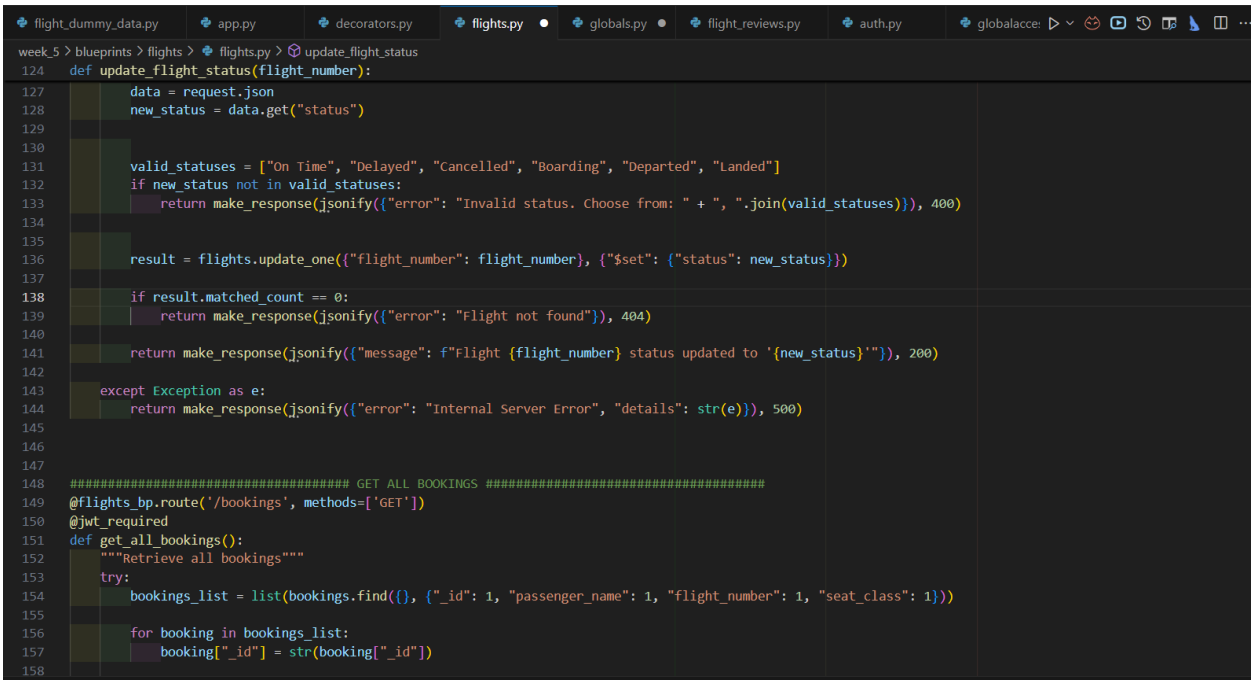
flights.py

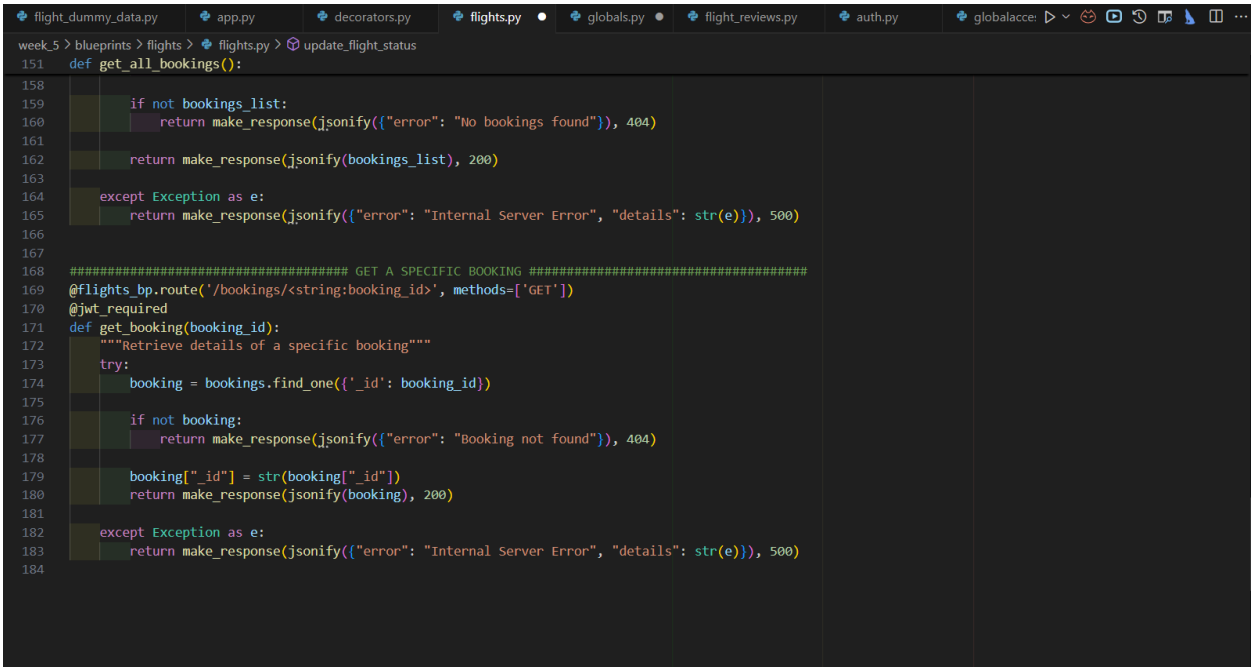



flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > flights > flights.py > update_flight_status
def search_flights():
if min_price is not None:
query["price"] = {"$gte": min_price}
if max_price is not None:
query["price"]["$lte"] = max_price if "price" in query else {"$lte": max_price}
# Sorting
sort_order_value = 1 if sort_order == "asc" else -1
flights_list = list(flights.find(query, {"_id": 0}).sort([(sort_by, sort_order_value)]))
if not flights_list:
return make_response(jsonify({"error": "No flights found for the given criteria"}), 404)
return make_response(jsonify(flights_list), 200)
except Exception as e:
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
################################### GET FLIGHT DETAILS ###################################
@flights_bp.route('/flights/<string:flight_number>', methods=['GET'])
def get_flight_details(flight_number):
"""Retrieve details of a specific flight using its flight number"""
try:
flight = flights.find_one({'flight_number': flight_number}, {"_id": 0})
if not flight:
return make_response(jsonify({"error": "Flight not found"}), 404)
return make_response(jsonify(flight), 200)
except Exception as e:

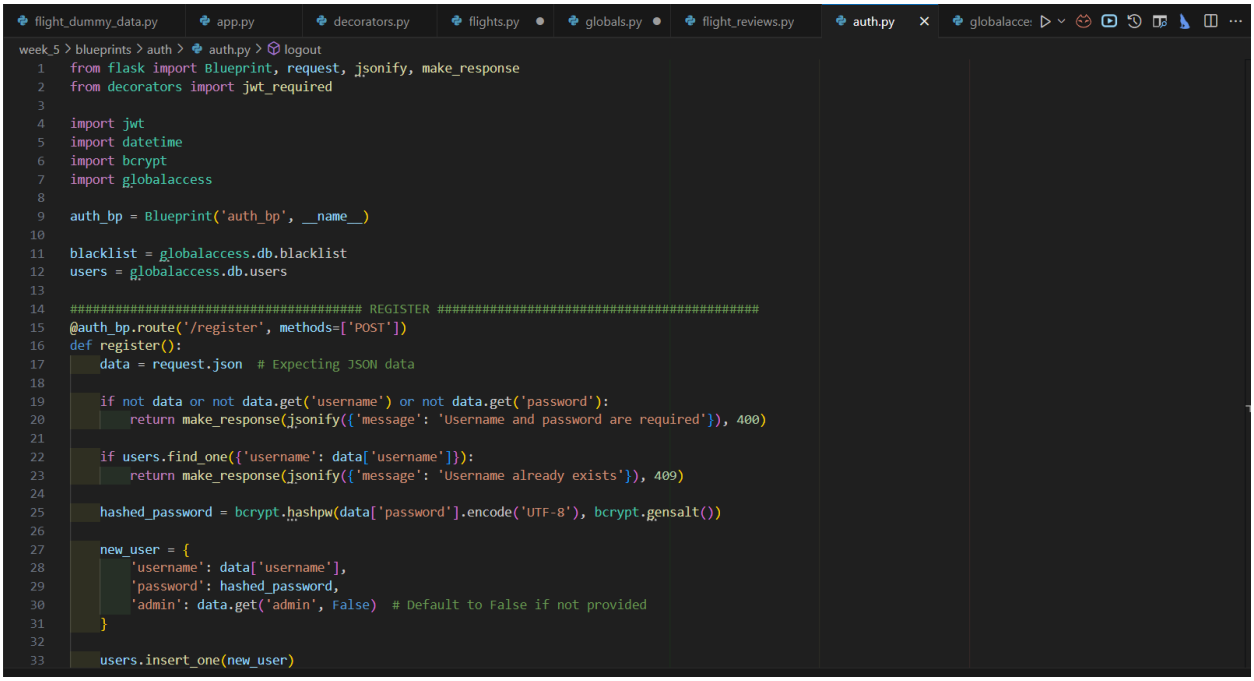

flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > flights > flights.py > update_flight_status
def get_flight_details(flight_number):
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
################################### BOOK A FLIGHT TICKET ###################################
@flights_bp.route('/bookings', methods=['POST'])
@jwt_required
def book_ticket():
"""Book a ticket for a passenger on a specific flight"""
try:
data = request.json
required_fields = [
"passenger_name", "passport_number", "email", "phone_number",
"flight_number", "seat_class", "contact_details"
]
if not all(field in data for field in required_fields):
return make_response(jsonify({"error": "Missing required fields"}), 400)
flight_number = data["flight_number"].strip()
flight = flights.find_one({"flight_number": flight_number})
if not flight:
return make_response(jsonify({"error": f"Flight '{flight_number}' not found"}), 404)
if flight["seats_available"] <= 0:
return make_response(jsonify({"error": "No seats available"}), 400)
booking_id = str(uuid.uuid4())

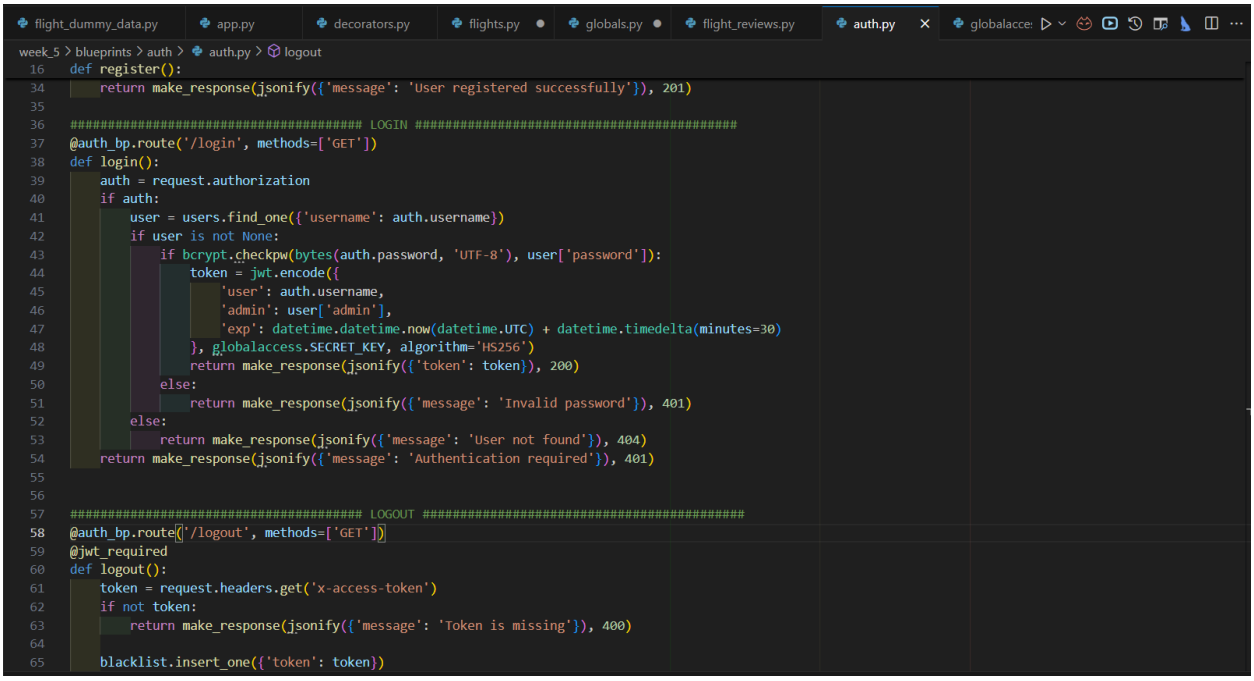

flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > flights > flights.py > update_flight_status
def book_ticket():
new_booking = {
"_id": booking_id,
"passenger_name": data["passenger_name"],
"passport_number": data["passport_number"],
"email": data["email"],
"phone_number": data["phone_number"],
"flight_number": flight_number,
"seat_class": data["seat_class"],
"contact_details": data["contact_details"],
"booking_time": datetime.datetime.utcnow()
}
bookings.insert_one(new_booking)
flights.update_one({"flight_number": flight_number}, {"$inc": {"seats_available": -1}})
return make_response(jsonify({"message": "Booking successful", "booking_id": booking_id, "passenger_details": new_booking}), 201)
except Exception as e:
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
################################## UPDATE FLIGHT STATUS ##################################
@flights_bp.route('/flights/<string:flight_number>/status', methods=['PUT'])
@jwt_required
@admin_required
def update_flight_status(flight_number):
"""Update the status of a specific flight """
try:
data = request.json

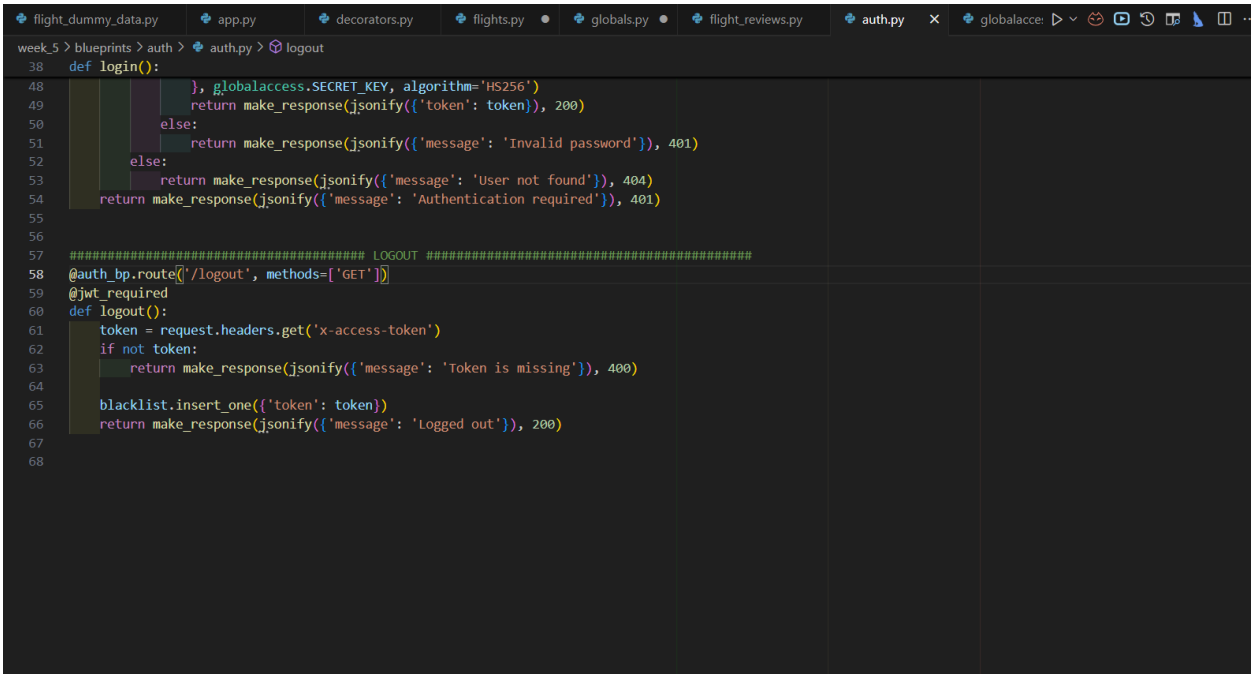

flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > flights > flights.py > update_flight_status
def update_flight_status(flight_number):
data = request.json
new_status = data.get("status")
valid_statuses = ["On Time", "Delayed", "Cancelled", "Boarding", "Departed", "Landed"]
if new_status not in valid_statuses:
return make_response(jsonify({"error": "Invalid status. Choose from: " + ", ".join(valid_statuses)}), 400)
result = flights.update_one({"flight_number": flight_number}, {"$set": {"status": new_status}})
if result.matched_count == 0:
return make_response(jsonify({"error": "Flight not found"}), 404)
return make_response(jsonify({"message": f"Flight {flight_number} status updated to '{new_status}'"}), 200)
except Exception as e:
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
################################## GET ALL BOOKINGS ##################################
@flights_bp.route('/bookings', methods=['GET'])
@jwt_required
def get_all_bookings():
"""Retrieve all bookings"""
try:
bookings_list = list(bookings.find({}, {"_id": 1, "passenger_name": 1, "flight_number": 1, "seat_class": 1}))
for booking in bookings_list:
booking["_id"] = str(booking["_id"])


flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > flights > flights.py > update_flight_status
def get_all_bookings():
if not bookings_list:
return make_response(jsonify({"error": "No bookings found"}), 404)
return make_response(jsonify(bookings_list), 200)
except Exception as e:
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
################################### GET A SPECIFIC BOOKING ###################################
@flights_bp.route('/bookings/<string:booking_id>', methods=['GET'])
@jwt_required
def get_booking(booking_id):
"""Retrieve details of a specific booking"""
try:
booking = bookings.find_one({'_id': booking_id})
if not booking:
return make_response(jsonify({"error": "Booking not found"}), 404)
booking["_id"] = str(booking["_id"])
return make_response(jsonify(booking), 200)
except Exception as e:
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)

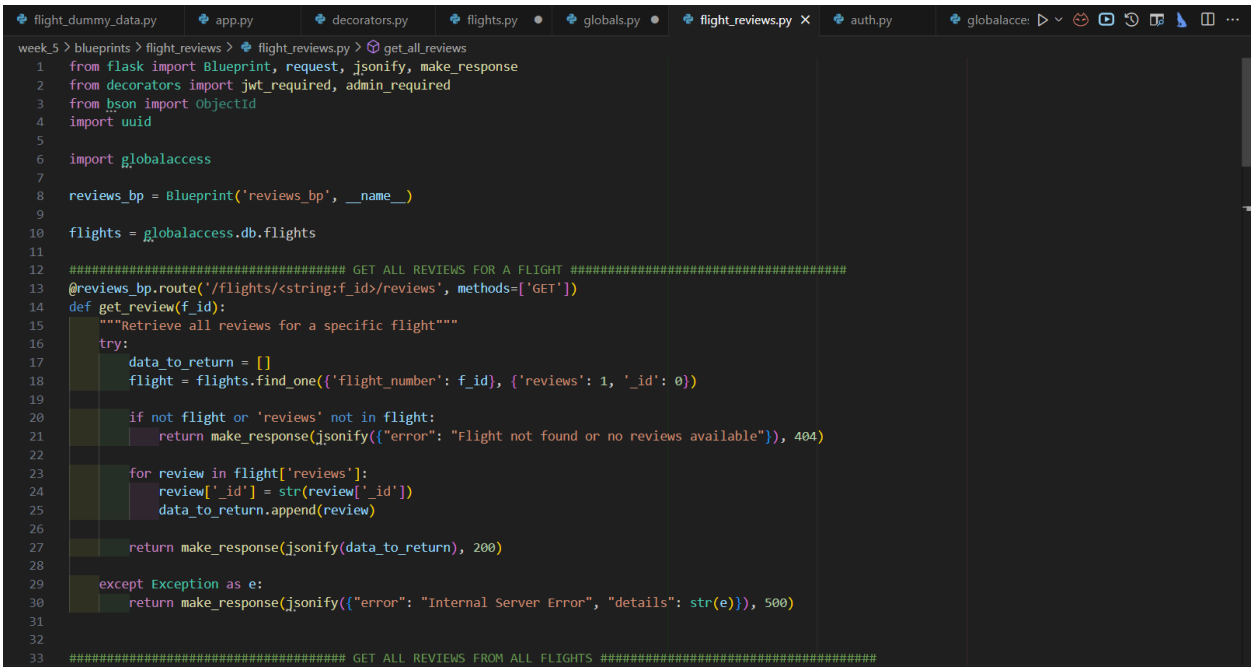
Auth.py


flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > auth > auth.py > logout
from flask import Blueprint, request, jsonify, make_response
from decorators import jwt_required
import jwt
import datetime
import bcrypt
import globalaccess
auth_bp = Blueprint('auth_bp', __name__)
blacklist = globalaccess.db.blacklist
users = globalaccess.db.users
################################### REGISTER ###################################
@auth_bp.route('/register', methods=['POST'])
def register():
data = request.json # Expecting JSON data
if not data or not data.get('username') or not data.get('password'):
return make_response(jsonify({'message': 'Username and password are required'}), 400)
if users.find_one({'username': data['username']}):
return make_response(jsonify({'message': 'Username already exists'}), 409)
hashed_password = bcrypt.hashpw(data['password'].encode('UTF-8'), bcrypt.gensalt())
new_user = {
'username': data['username'],
'password': hashed_password,
'admin': data.get('admin', False) # Default to False if not provided
}
users.insert_one(new_user)

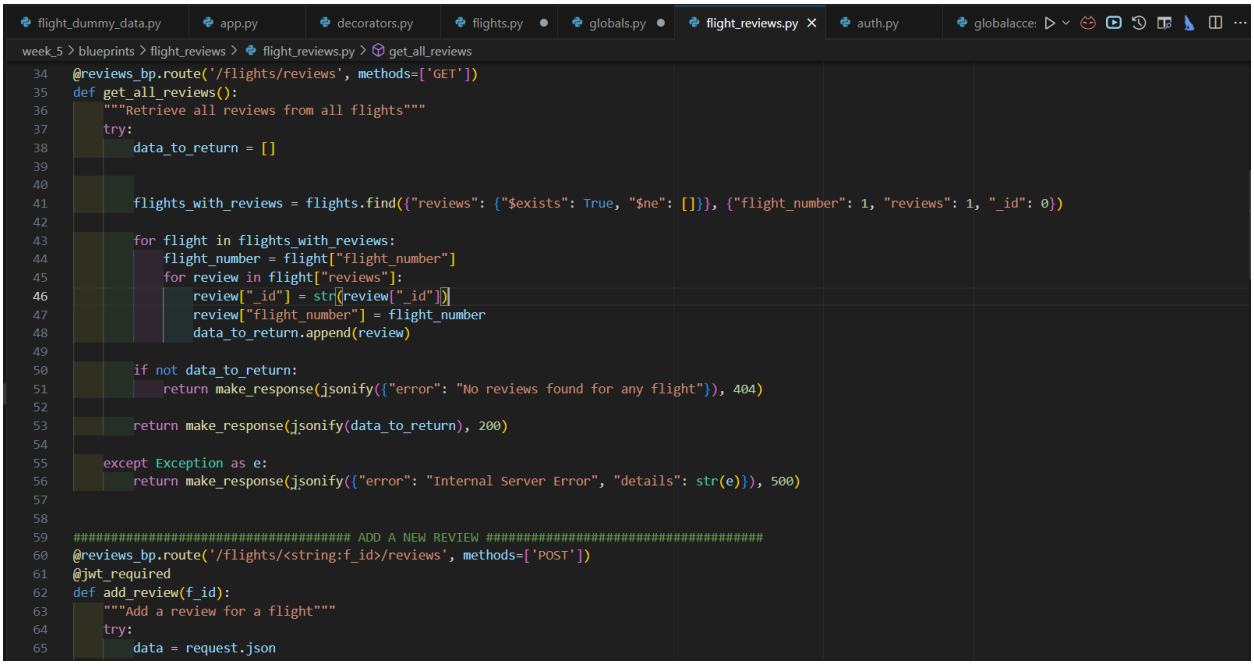

flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > auth > auth.py > logout
def register():
return make_response(jsonify({'message': 'User registered successfully'}), 201)
LOGIN
@auth_bp.route('/login', methods=['GET'])
def login():
auth = request.authorization
if auth:
user = users.find_one({'username': auth.username})
if user is not None:
if bcrypt.checkpw(bytes(auth.password, 'UTF-8'), user['password']):
token = jwt.encode({
'user': auth.username,
'admin': user['admin'],
'exp': datetime.datetime.now(datetime.UTC) + datetime.timedelta(minutes=30)
}, globalaccess.SECRET_KEY, algorithm='HS256')
return make_response(jsonify({'token': token}), 200)
else:
return make_response(jsonify({'message': 'Invalid password'}), 401)
else:
return make_response(jsonify({'message': 'User not found'}), 404)
return make_response(jsonify({'message': 'Authentication required'}), 401)
LOGOUT
@auth_bp.route('/logout', methods=['GET'])
@jwt_required
def logout():
token = request.headers.get('x-access-token')
if not token:
return make_response(jsonify({'message': 'Token is missing'}), 400)
blacklist.insert_one({'token': token})


flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > auth > auth.py > logout
def login():
}, globalaccess.SECRET_KEY, algorithm='HS256')
return make_response(jsonify({'token': token}), 200)
else:
return make_response(jsonify({'message': 'Invalid password'}), 401)
else:
return make_response(jsonify({'message': 'User not found'}), 404)
return make_response(jsonify({'message': 'Authentication required'}), 401)
LOGOUT
@auth_bp.route('/logout', methods=['GET'])
@jwt_required
def logout():
token = request.headers.get('x-access-token')
if not token:
return make_response(jsonify({'message': 'Token is missing'}), 400)
blacklist.insert_one({'token': token})
return make_response(jsonify({'message': 'Logged out'}), 200)

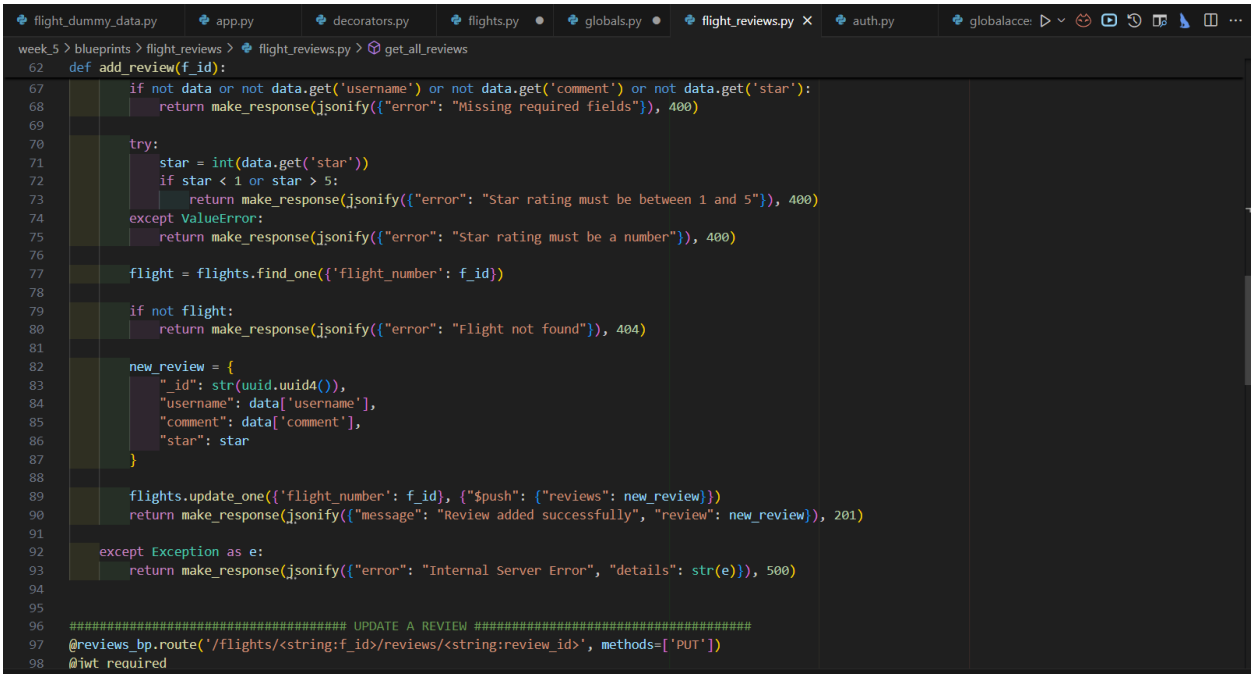
flight_review.py



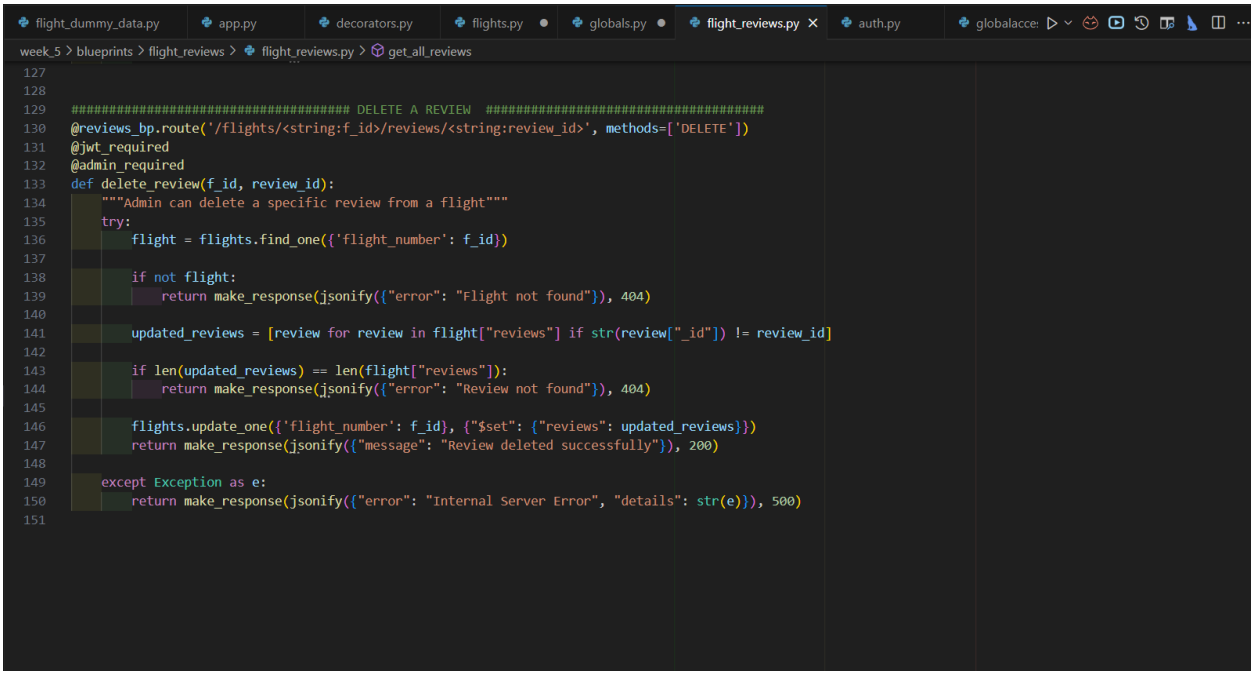
```
flight_dummy_data.py  app.py  decorators.py  flights.py  globals.py  flight_reviews.py  auth.py  globalacce
week_5 > blueprints > flight_reviews > flight_reviews.py > get_all_reviews
1  from flask import Blueprint, request, jsonify, make_response
2  from decorators import jwt_required, admin_required
3  from bson import ObjectId
4  import uuid
5
6  import globalaccess
7
8  reviews_bp = Blueprint('reviews_bp', __name__)
9
10 flights = globalaccess.db.flights
11
12 ################################### GET ALL REVIEWS FOR A FLIGHT ###################################
13 @reviews_bp.route('/flights/<string:f_id>/reviews', methods=['GET'])
14 def get_review(f_id):
15     """Retrieve all reviews for a specific flight"""
16     try:
17         data_to_return = []
18         flight = flights.find_one({'flight_number': f_id}, {'reviews': 1, '_id': 0})
19
20         if not flight or 'reviews' not in flight:
21             return make_response(jsonify({"error": "Flight not found or no reviews available"}), 404)
22
23         for review in flight['reviews']:
24             review['_id'] = str(review['_id'])
25             data_to_return.append(review)
26
27         return make_response(jsonify(data_to_return), 200)
28
29     except Exception as e:
30         return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
31
32
33 ################################### GET ALL REVIEWS FROM ALL FLIGHTS ###################################
```



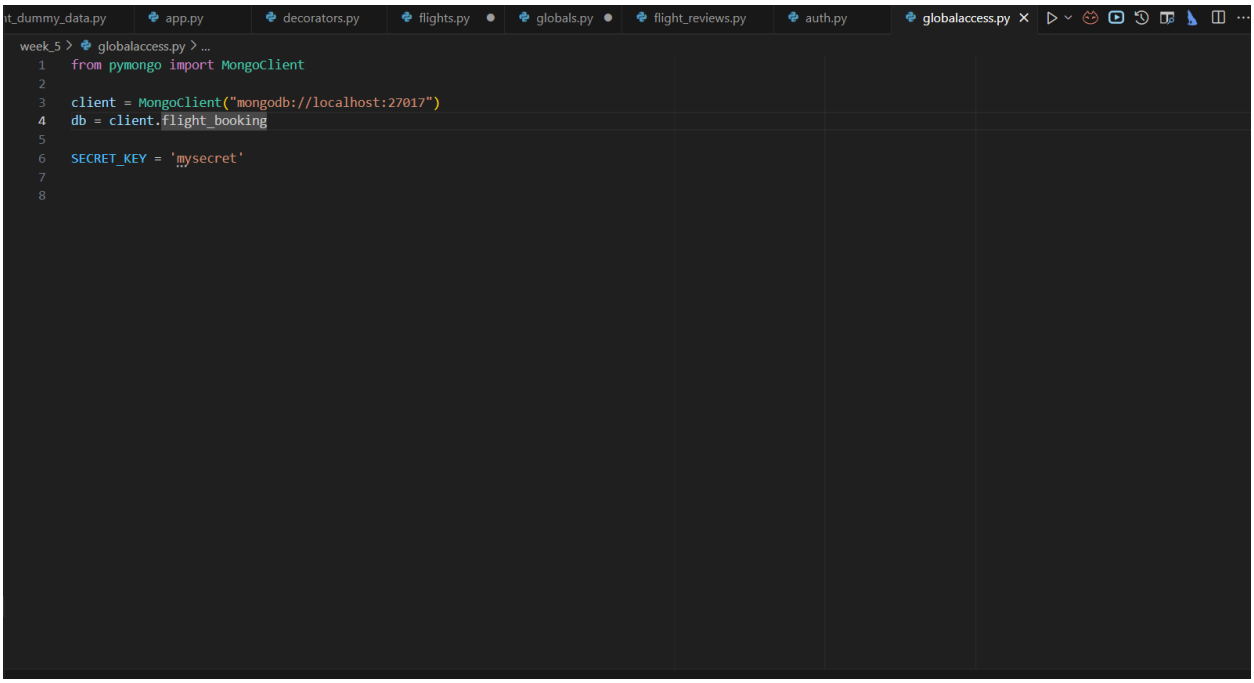
```
flight_dummy_data.py  app.py  decorators.py  flights.py  globals.py  flight_reviews.py  auth.py  globalacce
week_5 > blueprints > flight_reviews > flight_reviews.py > get_all_reviews
34 @reviews_bp.route('/flights/reviews', methods=['GET'])
35 def get_all_reviews():
36     """Retrieve all reviews from all flights"""
37     try:
38         data_to_return = []
39
40
41         flights_with_reviews = flights.find({"reviews": {"$exists": True, "$ne": []}}, {"flight_number": 1, "reviews": 1, "_id": 0})
42
43         for flight in flights_with_reviews:
44             flight_number = flight["flight_number"]
45             for review in flight["reviews"]:
46                 review["_id"] = str(review["_id"])
47                 review["flight_number"] = flight_number
48                 data_to_return.append(review)
49
50         if not data_to_return:
51             return make_response(jsonify({"error": "No reviews found for any flight"}), 404)
52
53         return make_response(jsonify(data_to_return), 200)
54
55     except Exception as e:
56         return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
57
58
59 ################################### ADD A NEW REVIEW ###################################
60 @reviews_bp.route('/flights/<string:f_id>/reviews', methods=['POST'])
61 @jwt_required
62 def add_review(f_id):
63     """Add a review for a flight"""
64     try:
65         data = request.json
```


flight_dummy_data.py
app.py
decorators.py
flights.py
globals.py
flight_reviews.py
auth.py
week_5 > blueprints > flight_reviews > flight_reviews.py > get_all_reviews
def add_review(f_id):
if not data or not data.get('username') or not data.get('comment') or not data.get('star'):
return make_response(jsonify({"error": "Missing required fields"}), 400)
try:
star = int(data.get('star'))
if star < 1 or star > 5:
return make_response(jsonify({"error": "Star rating must be between 1 and 5"}), 400)
except ValueError:
return make_response(jsonify({"error": "Star rating must be a number"}), 400)
flight = flights.find_one({'flight_number': f_id})
if not flight:
return make_response(jsonify({"error": "Flight not found"}), 404)
new_review = {
"_id": str(uuid.uuid4()),
"username": data['username'],
"comment": data['comment'],
"star": star
}
flights.update_one({'flight_number': f_id}, {"$push": {"reviews": new_review}})
return make_response(jsonify({"message": "Review added successfully", "review": new_review}), 201)
except Exception as e:
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
##################################### UPDATE A REVIEW #####################################
@reviews_bp.route('/flights/<string:f_id>/reviews/<string:review_id>', methods=['PUT'])
@jwt required

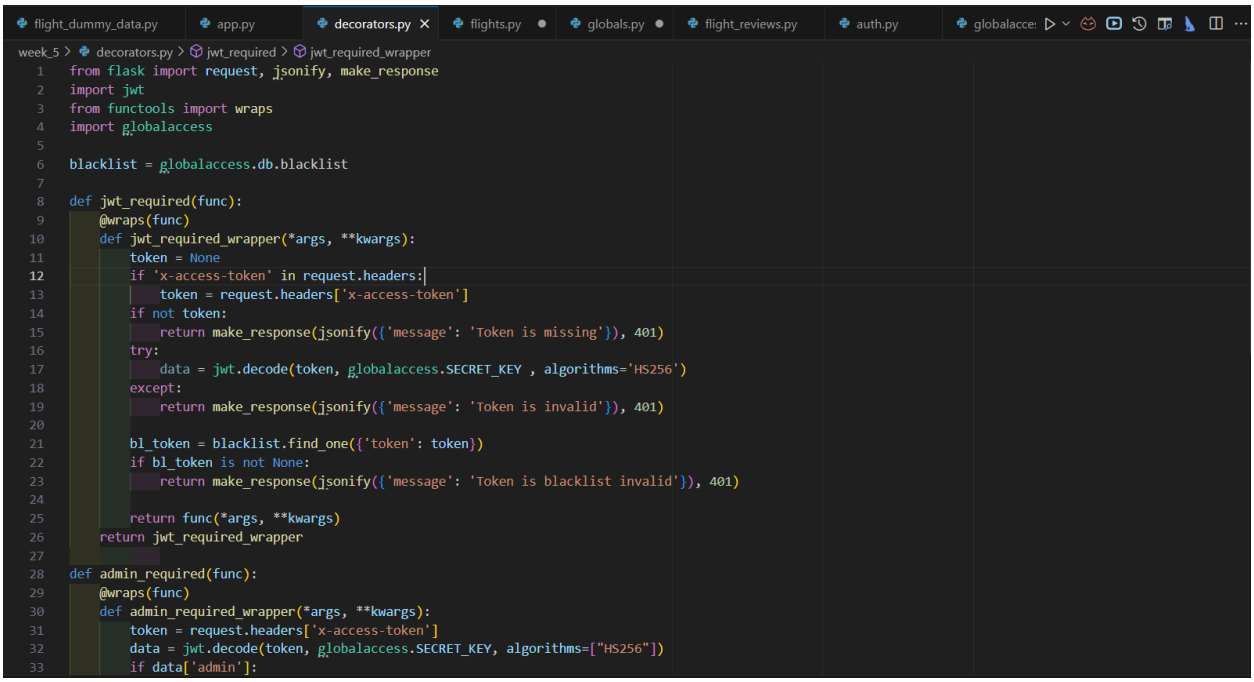

def update_review(f_id, review_id):
"""Update a specific review for a flight"""
try:
data = request.json
flight = flights.find_one({'flight_number': f_id})
if not flight:
return make_response(jsonify({"error": "Flight not found"}), 404)
updated_reviews = []
review_found = False
for review in flight["reviews"]:
if str(review["_id"]) == review_id:
review["username"] = data.get("username", review["username"])
review["comment"] = data.get("comment", review["comment"])
review["star"] = data.get("star", review["star"])
review_found = True
updated_reviews.append(review)
if not review_found:
return make_response(jsonify({"error": "Review not found"}), 404)
flights.update_one({'flight_number': f_id}, {"$set": {"reviews": updated_reviews}})
return make_response(jsonify({"message": "Review updated successfully"}), 200)
except Exception as e:
return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
##################################### DELETE A REVIEW #####################################
@reviews_bp.route('/flights/<string:f_id>/reviews/<string:review_id>', methods=['DELETE'])
@jwt required

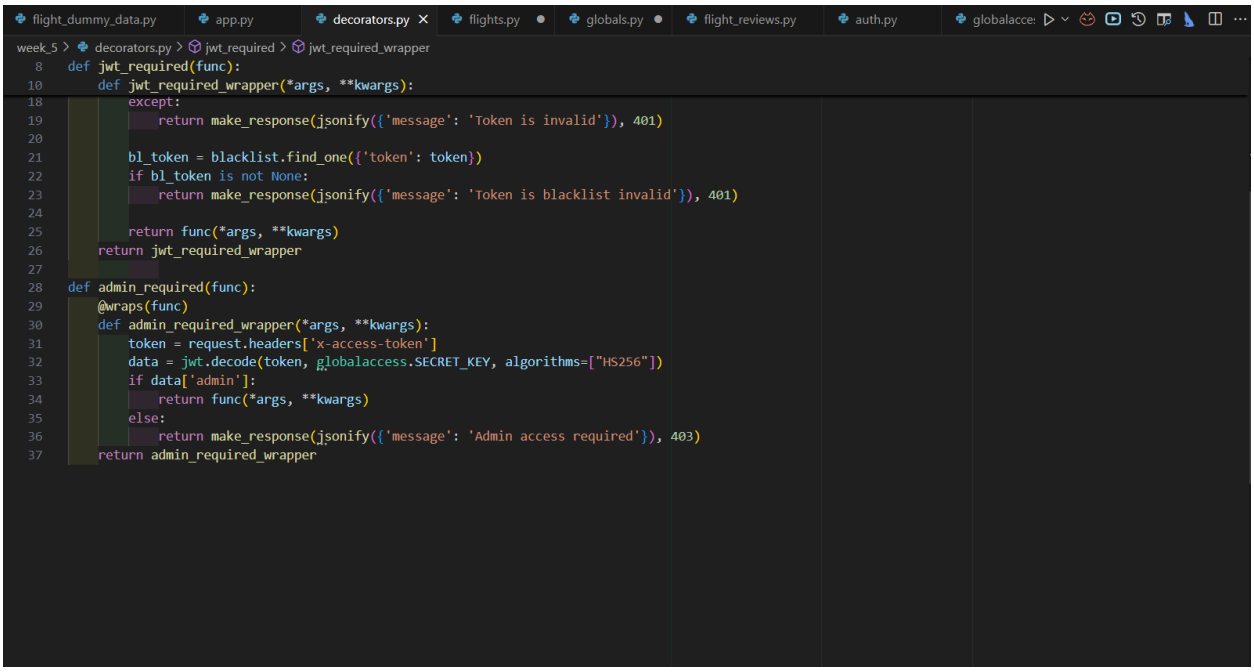


globalaccess.py

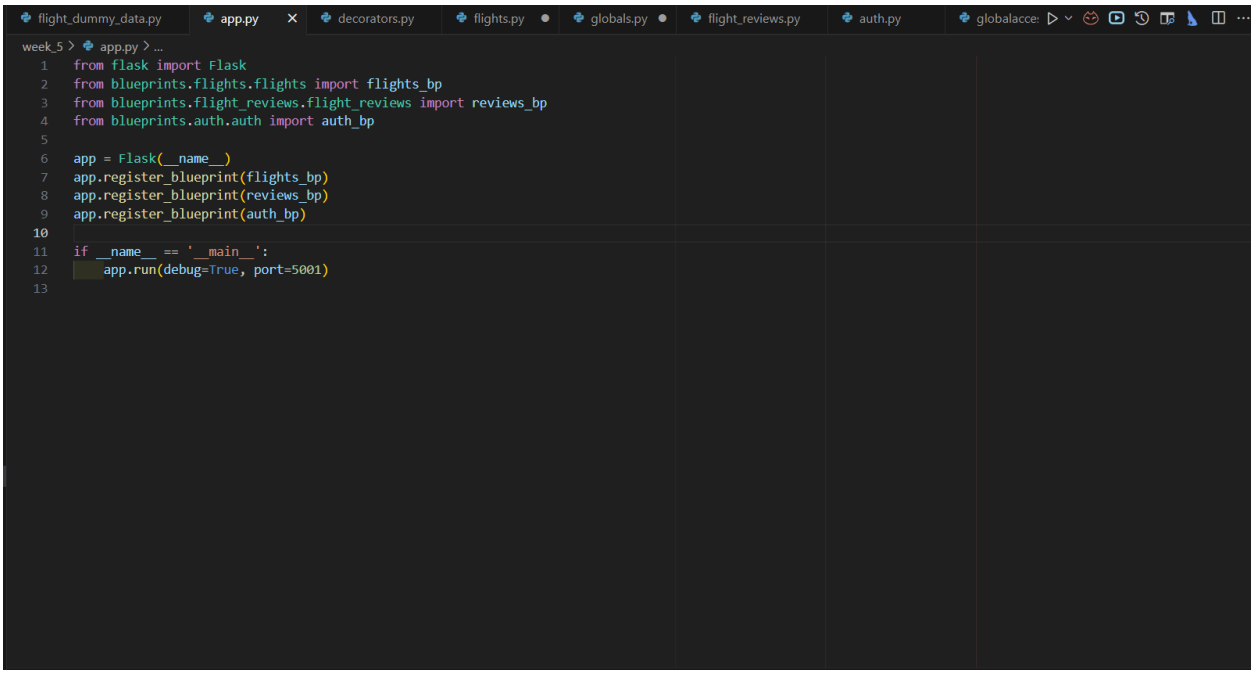


decorators.py





app.py



Summary of API endpoints:

3.1 /flights (GET)

Description: Retrieve a list of available flights based on optional filters such as departure location, destination, and date.

Parameters:

- departure_location *(optional)*: String - Departure airport.
- arrival_location *(optional)*: String - Arrival airport.
- date *(optional)*: String - Flight date.

Response:

- **Success (200):** Returns a list of available flights.
- **Internal Server Error (500):** If an unexpected error occurs.

3.2 /flights/<flight_number> (GET)

Description: Retrieve details of a specific flight using its flight number.

Parameters:

- flight_number: String - The unique identifier of the flight.

Response:

- **Success (200):** Returns flight details such as airline, departure and arrival times, duration, and price.
- **Not Found (404):** If the specified flight does not exist.
- **Internal Server Error (500):** If an unexpected error occurs.

3.3 /bookings (POST)

Description: Create a new flight booking by providing passenger details and flight information.

Response:

- **Success (201):** Booking successfully created.
- **Bad Request (400):** If required fields are missing.
- **Not Found (404):** If the flight does not exist.
- **Internal Server Error (500):** If an unexpected error occurs.

3.4 /bookings/<booking_id> (GET)

Description: Retrieve booking details using the booking ID.

Parameters:

- booking_id: String - Unique identifier of the booking.

Response:

- **Success (200):** Returns booking details including passenger information and flight details.
- **Not Found (404):** If the booking does not exist.
- **Internal Server Error (500):** If an unexpected error occurs.

3.5 /bookings/<booking_id> (PUT)

Description: Update an existing booking, such as changing the seat class or contact details.

Response:

- **Success (200):** Booking successfully updated.
- **Not Found (404):** If the booking does not exist.
- **Internal Server Error (500):** If an unexpected error occurs.

3.6 /bookings/<booking_id> (DELETE)

Description: Cancel a booking. Regular users can only cancel their own bookings. Admins can cancel any booking.

Response:

- **Success (200):** Booking successfully canceled.
- **Not Found (404):** If the booking does not exist.
- **Forbidden (403):** If a non-admin user tries to delete someone else's booking.
- **Internal Server Error (500):** If an unexpected error occurs.

3.7 /flights/<flight_number>/reviews (POST)

Description: Allow users to post reviews for a specific flight.

Response:

- **Success (201):** Review successfully added.
- **Bad Request (400):** If required fields are missing.
- **Not Found (404):** If the flight does not exist.
- **Internal Server Error (500):** If an unexpected error occurs.

3.8 /flights/<flight_number>/reviews (GET)

Description: Retrieve all reviews for a specific flight.

Parameters:

- flight_number: String - The unique identifier of the flight.

Response:

- **Success (200):** Returns a list of reviews.
- **Not Found (404):** If no reviews exist for the flight.
- **Internal Server Error (500):** If an unexpected error occurs.

3.9 /flights/<flight_number>/reviews/<review_id> (PUT)

Description: Update an existing flight review.

Response:

- **Success (200):** Review successfully updated.
- **Not Found (404):** If the review does not exist.
- **Internal Server Error (500):** If an unexpected error occurs.

3.10 /flights/<flight_number>/reviews/<review_id> (DELETE)

Description: Delete a flight review. Only **admins** can delete reviews.

Response:

- **Success (200):** Review successfully deleted.
- **Not Found (404):** If the review does not exist.
- **Forbidden (403):** If a non-admin user tries to delete a review.
- **Internal Server Error (500):** If an unexpected error occurs.

3.11 /flights/<flight_number>/status (PUT)

Description: Update the flight status (e.g., "Delayed", "On Time"). **Admins only.**

Request Body (JSON):

Response:

- **Success (200):** Flight status successfully updated.
- **Not Found (404):** If the flight does not exist.

- **Forbidden (403):** If a non-admin user tries to update the status.
- **Internal Server Error (500):** If an unexpected error occurs.

3.12 /auth/register (POST)

Description: Allow new users to register.

Response:

- **Success (201):** User successfully registered.
- **Bad Request (400):** If required fields are missing.
- **Internal Server Error (500):** If an unexpected error occurs.

3.13 /auth/login (GET)

Description: Authenticate a user and provide a JWT token for further API access.

Parameters:

- username: String - The username.
- password: String - The password.

Response:

- **Success (200):** Returns a **JWT token** for authentication.
- **Unauthorized (401):** If credentials are invalid.
- **Internal Server Error (500):** If an unexpected error occurs.

3.14 /auth/logout (GET)

Description: Logs out a user by blacklisting their JWT token.

Headers:

- x-access-token: The authentication token.

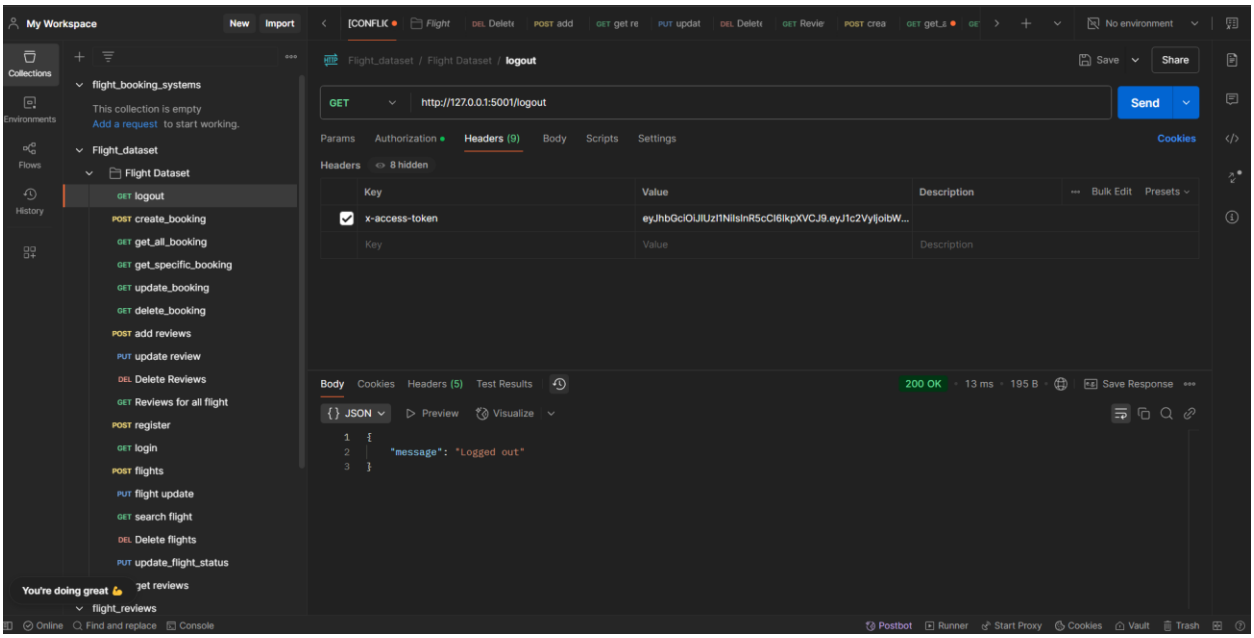
Response:

- **Success (200):** User successfully logged out.
- **Unauthorized (401):** If token is invalid.

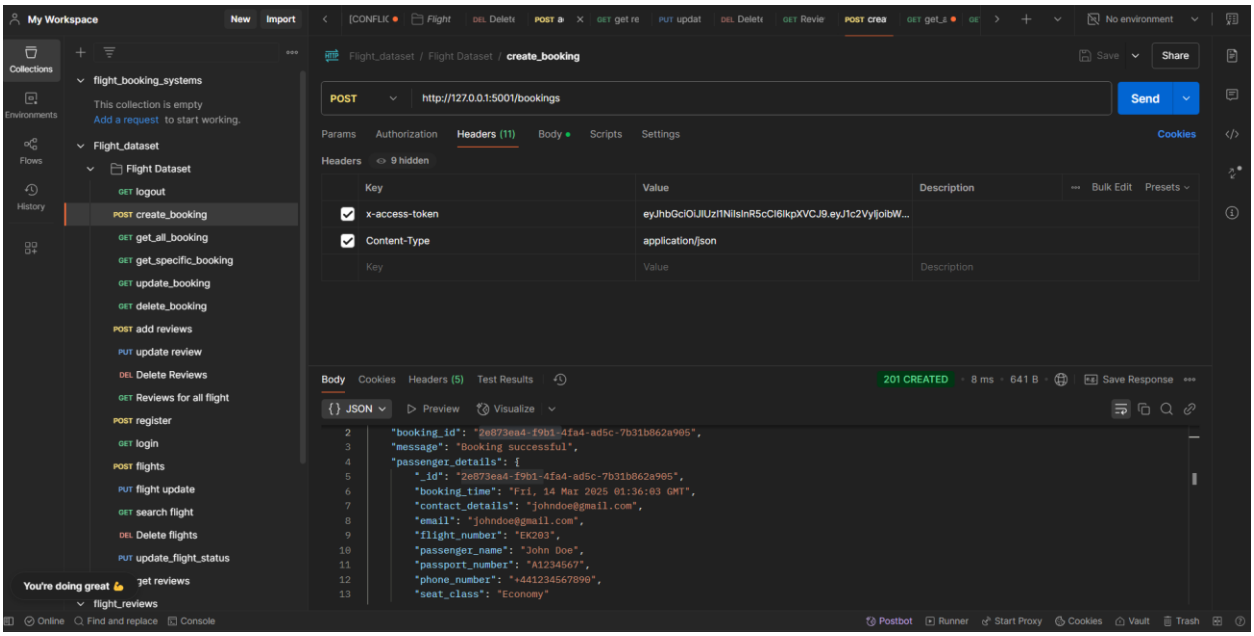
- **Internal Server Error (500):** If an unexpected error occurs

Printout of collection of tests on the API through Postman and generated API documentation

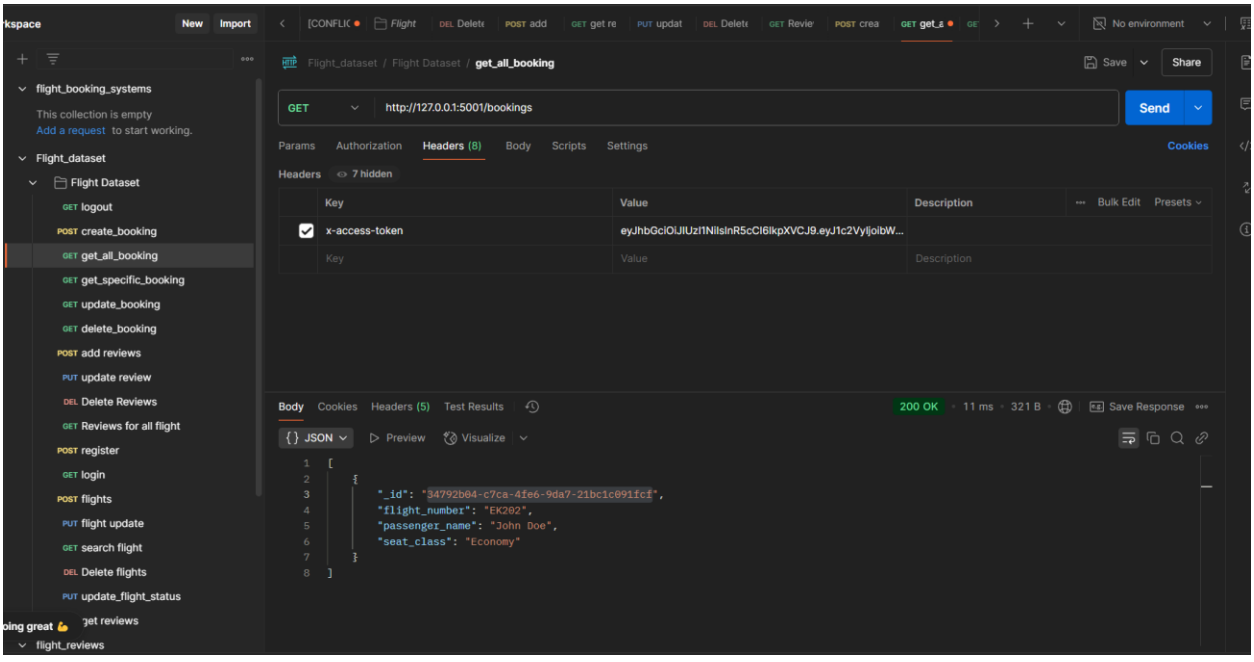
GET logout



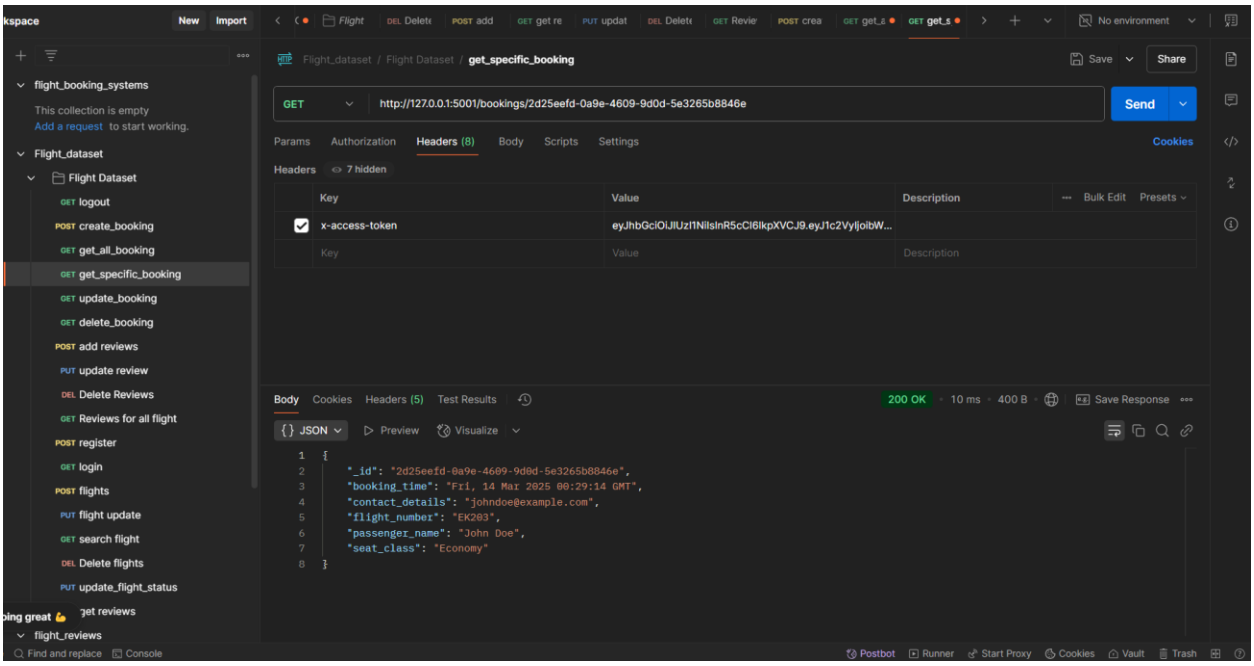
POST create_booking



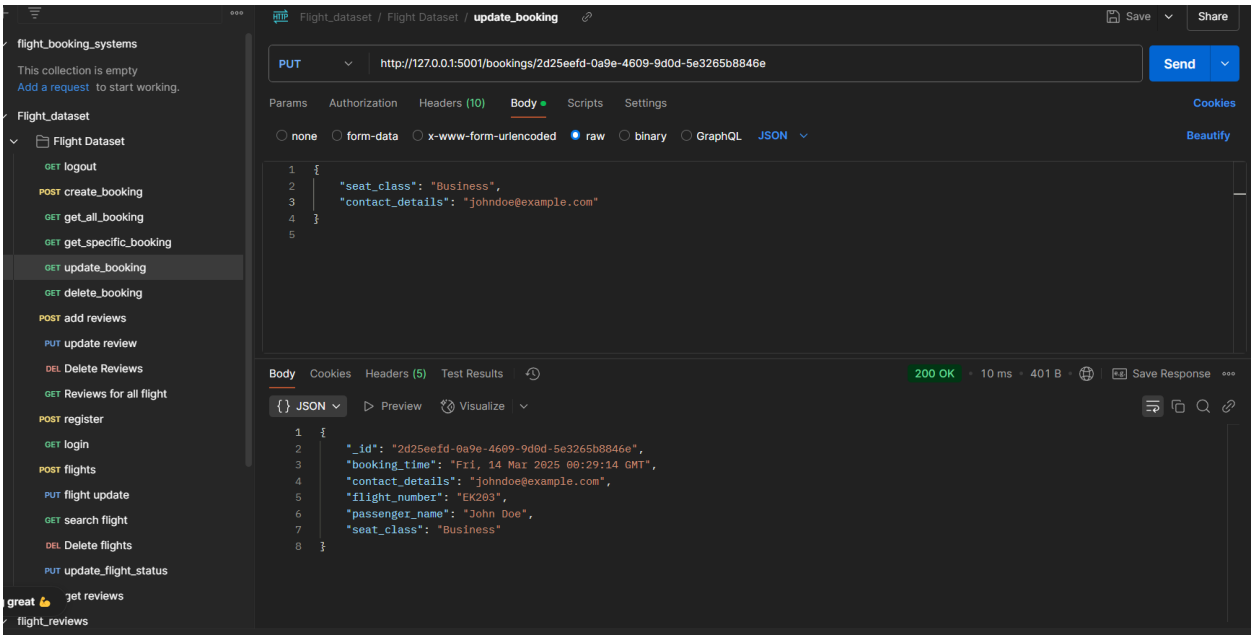
GET get_all_booking



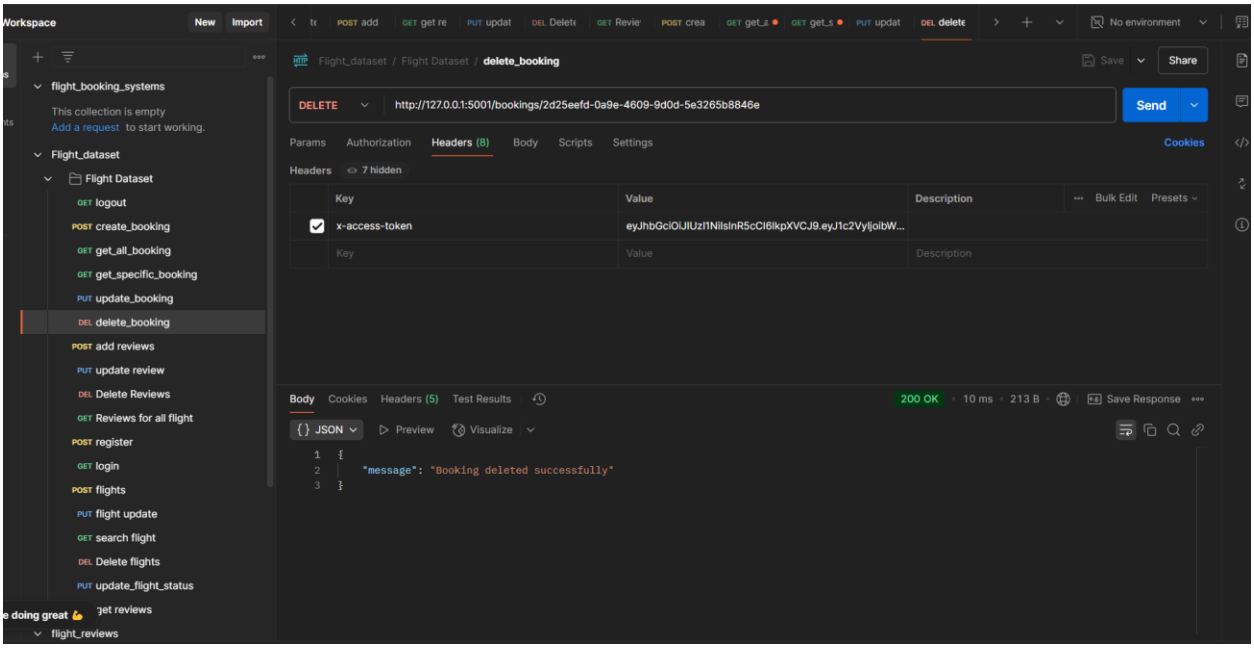
GET get_specific_booking



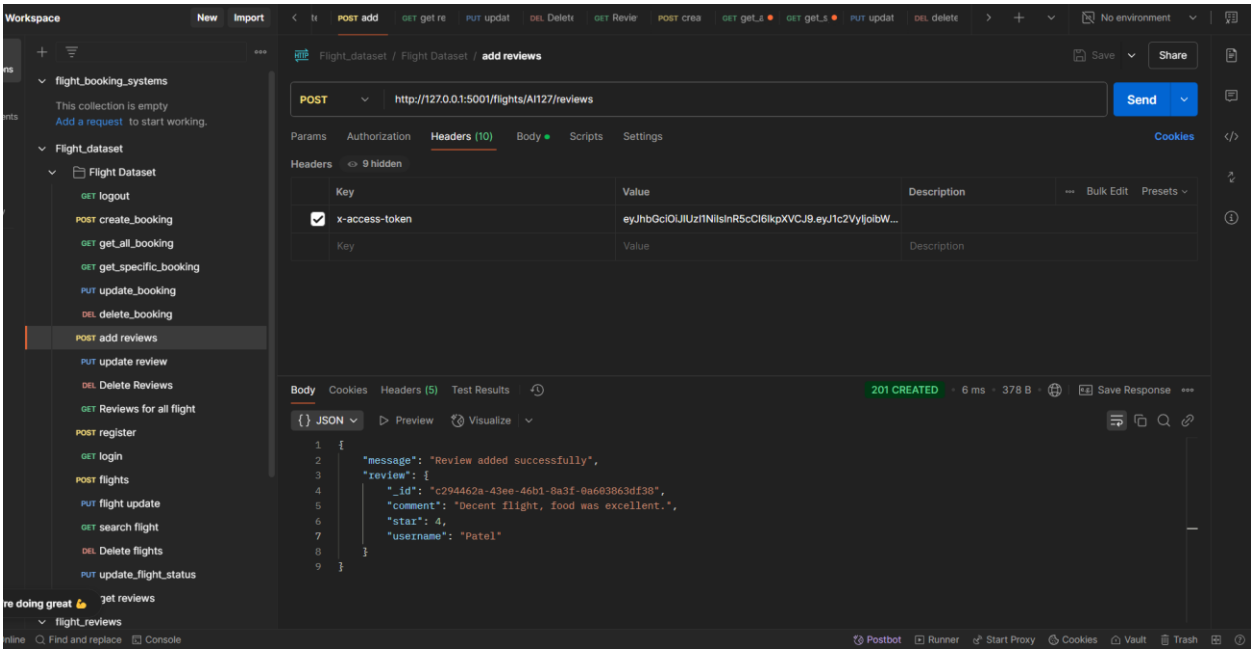
PUT update_booking



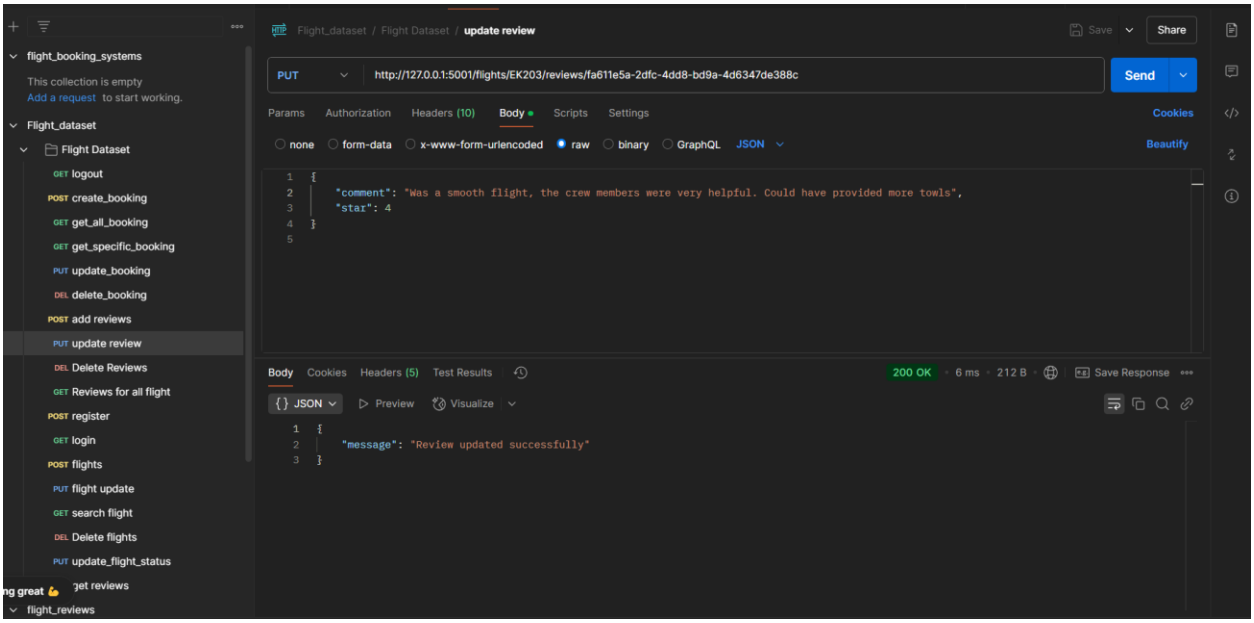
DELETE delete_booking



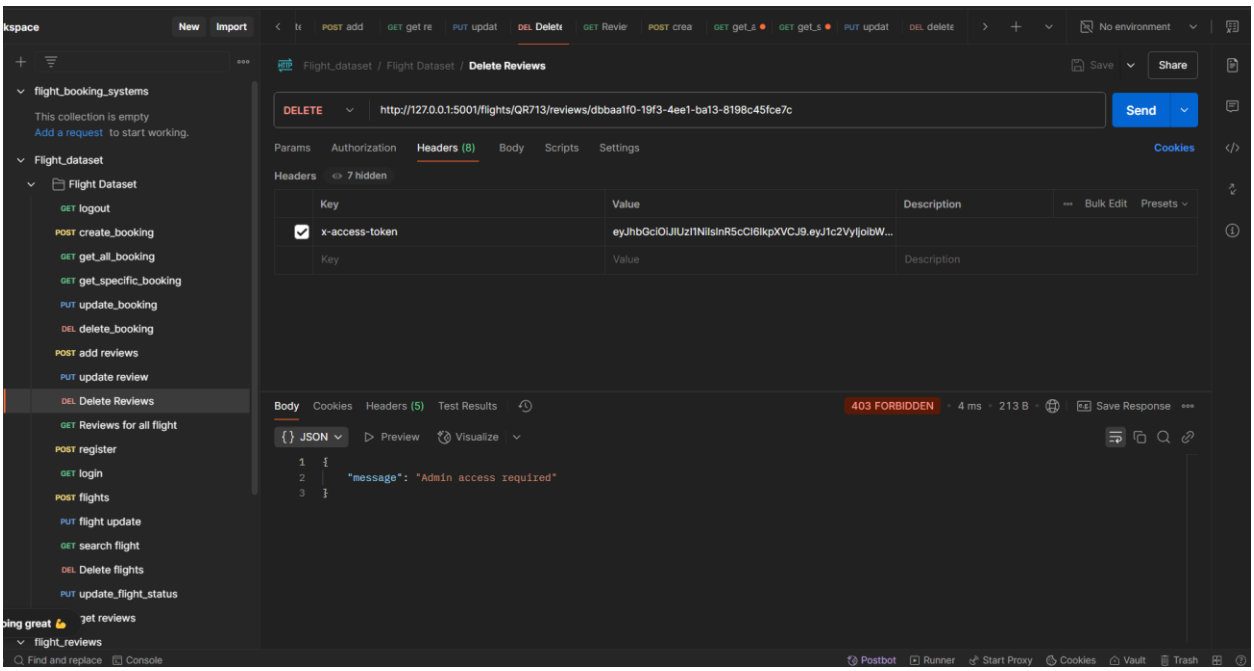
POST add reviews



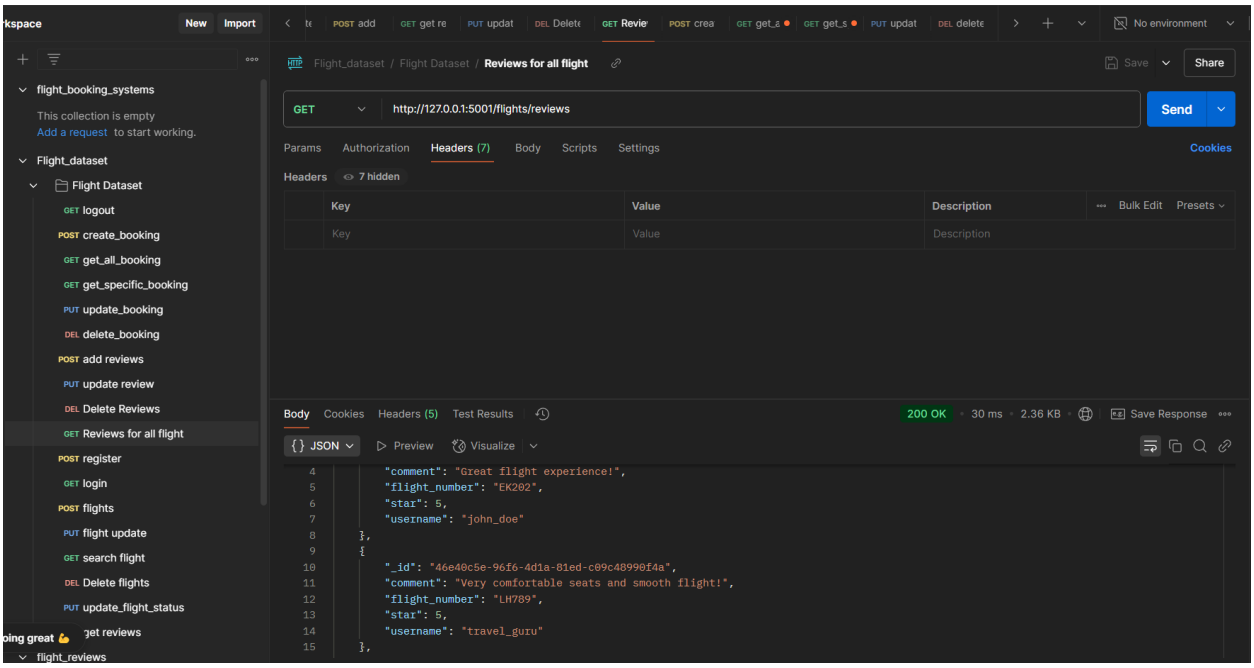
PUT update review



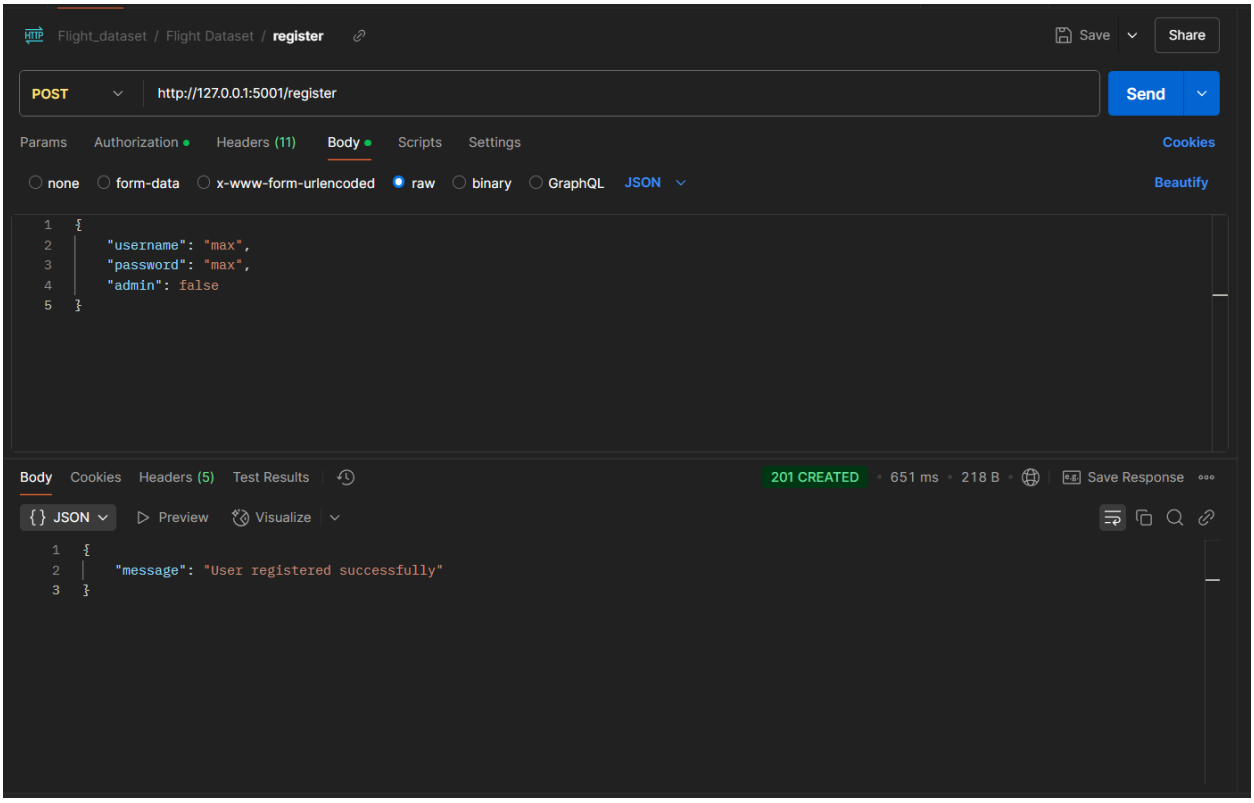
DELETE Delete Reviews



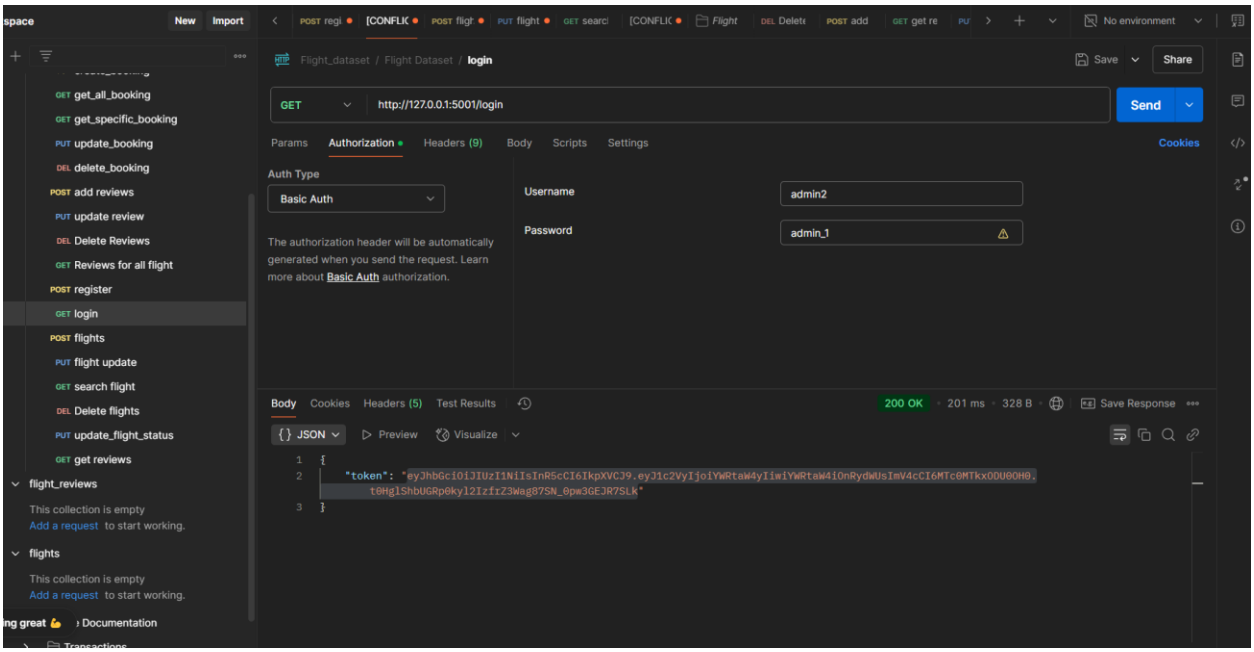
GET reviews for all flights



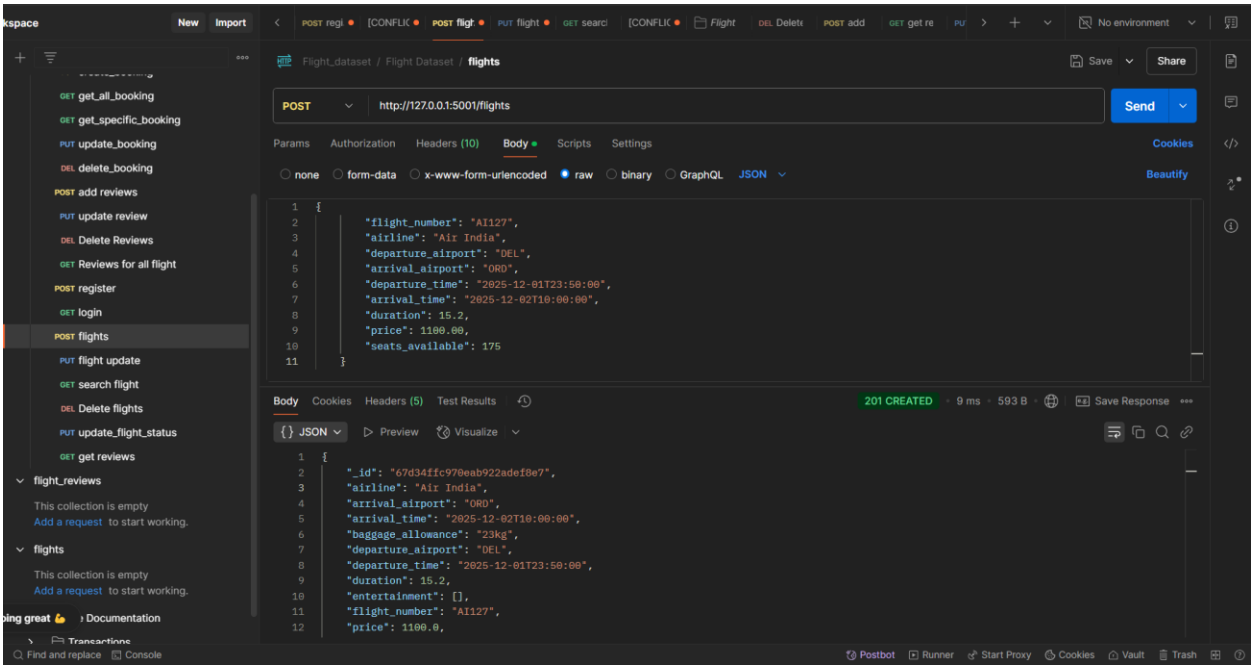
POST register



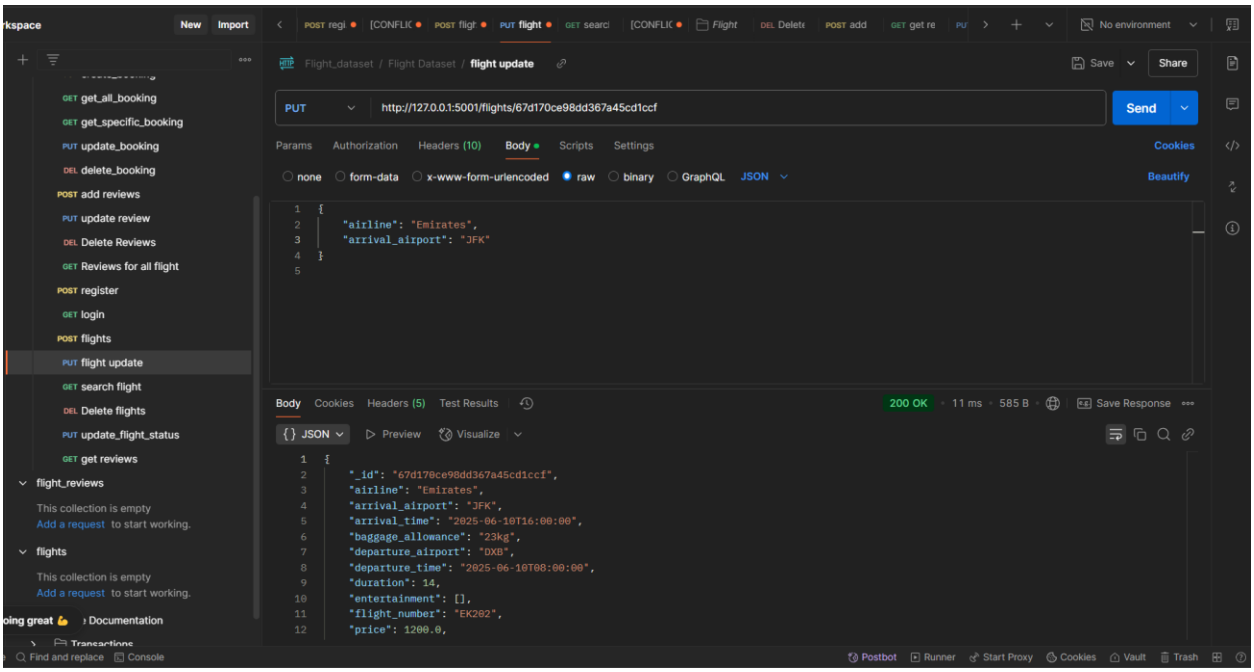
GET login



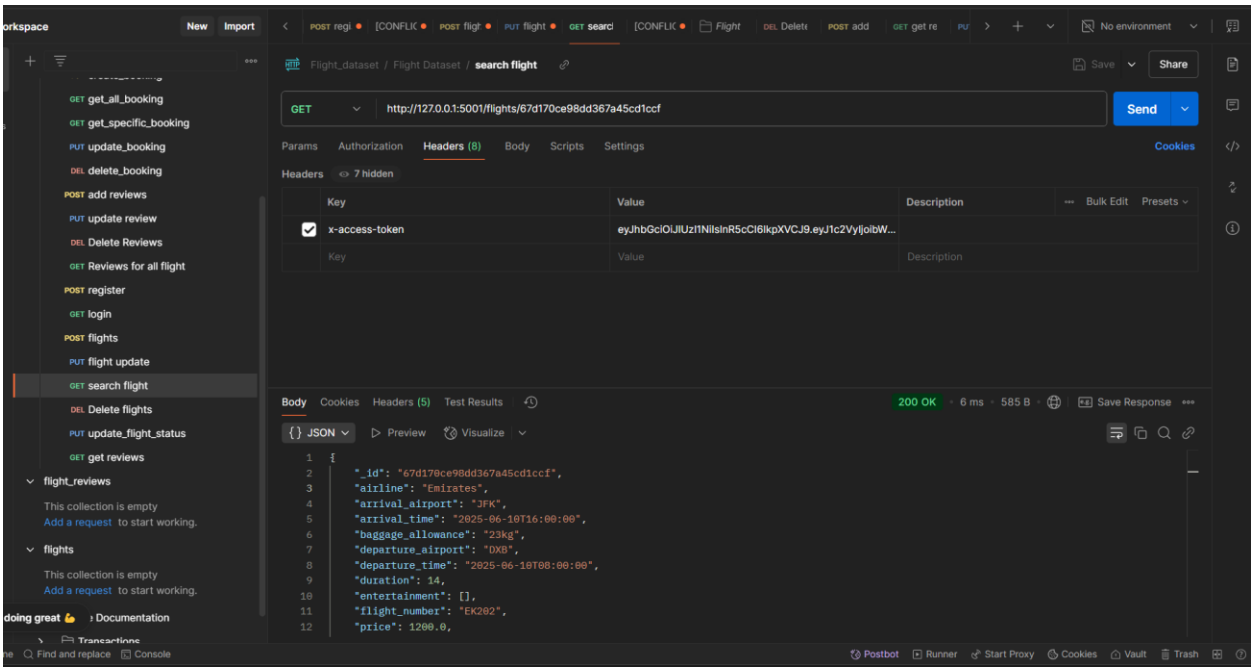
POST flights



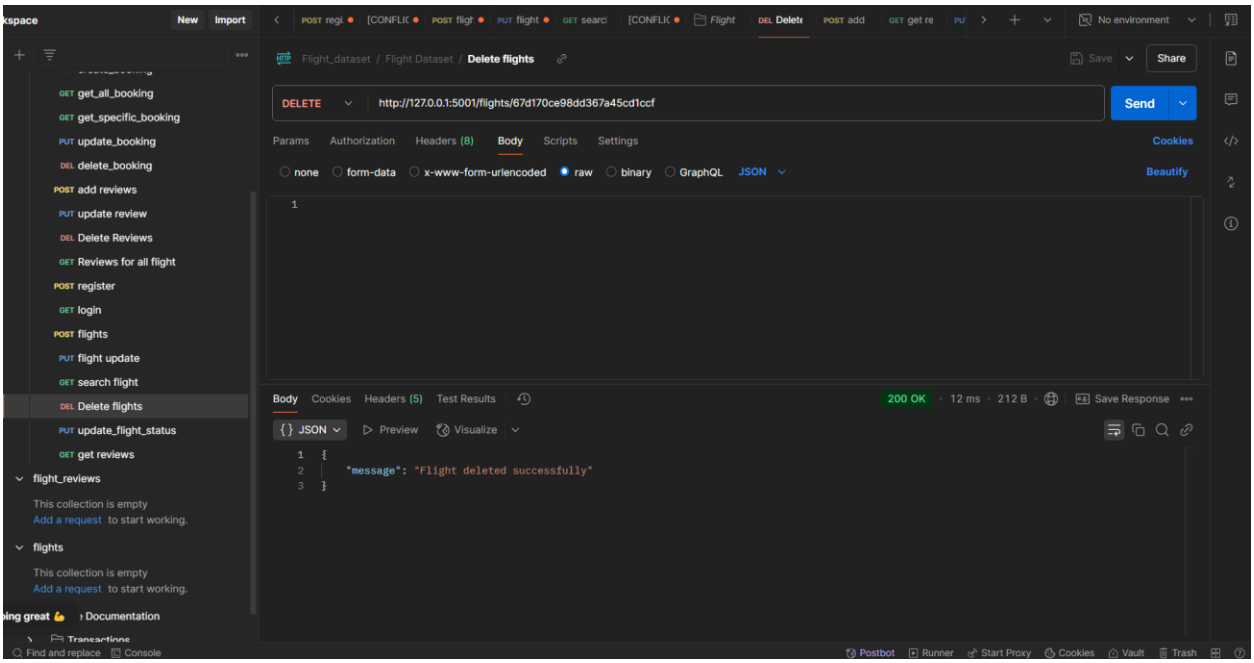
PUT flight update



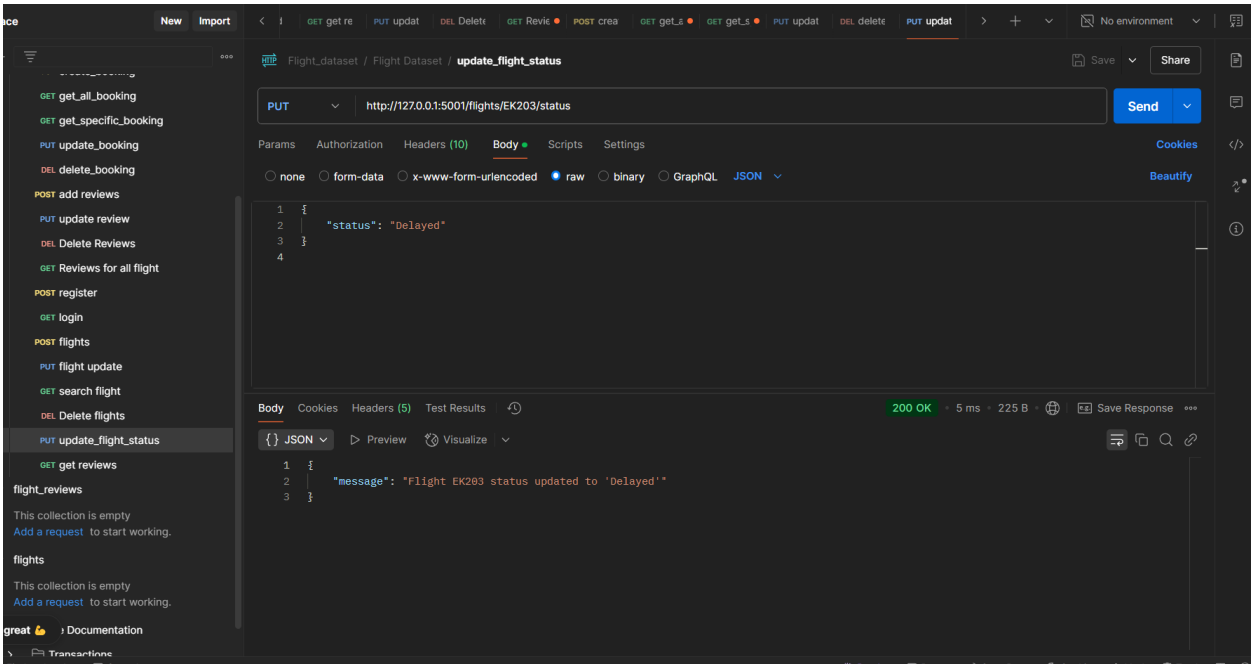
GET search flight



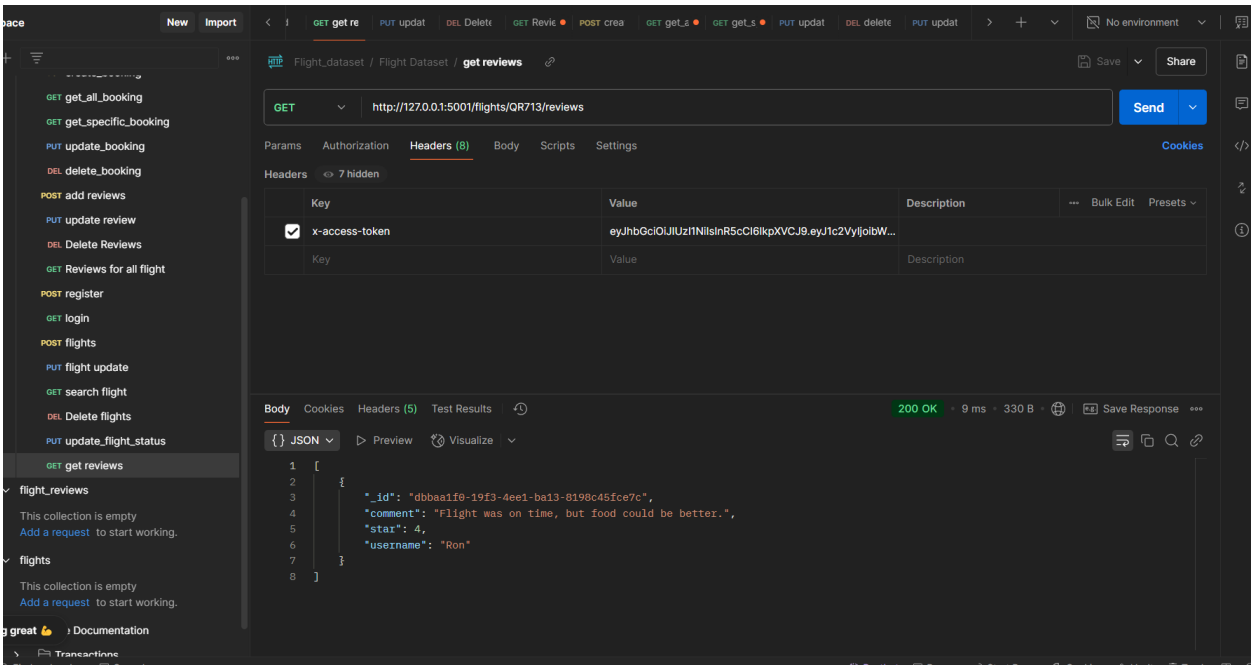
DELETE delete flights



PUT update_fight_status



GET get reviews



Postman Documentation:

https://documenter.getpostman.com/view/41817092/2sAYkAQhnX

Conclusion:

The Flight Booking System API successfully implements a nicely designed backend to organize flights, bookings, and reviews with secure authentication and role-based access. The API successfully enables users to search and book flights, provide reviews, and obtain real-time flight status, while administrators can efficiently manage flights and bookings. Thoroughly tested using Postman, the system is fully functionally compliant. Payment gateway integration and automated notifications can be potential future enhancements to elevate the user experience to the next level.