

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  globalacce: ▶ ◂ ◃ ◅ ◆ ◇ ◈ ◉ ◊ ○ ◌ ◍ ◎ ● ◐ ◑ ◒ ◓ ◔ ◕ ◖ ◗ ◘ ◙ ◚ ◛ ◜ ◝ ◞ ◟ ◠ ◡ ◢ ◣ ◤ ◥ ◦ ◧ ◨ ◩ ◪ ◫ ◬ ◭ ◮ ◯ ◰ ◱ ◲ ◳ ◴ ◵ ◶ ◷ ◸ ◹ ◺ ◻ ◼ ◽ ◾ ◿ ◾ ◿ ◾ ◿ ...
week_5 > blueprints > flights > flights.py > update_flight_status
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  import datetime
6  import globalaccess
7
8  flights_bp = Blueprint('flights_bp', __name__)
9
10 flights = globalaccess.db.flights
11 bookings = globalaccess.db.bookings
12
13 ##### SEARCH FLIGHTS #####
14 @flights_bp.route('/flights', methods=['GET'])
15 def search_flights():
16     """Search for flights based on departure, arrival, date, price range, and sorting"""
17     try:
18         departure = request.args.get('departure_location')
19         arrival = request.args.get('arrival_location')
20         date = request.args.get('date') # Format: YYYY-MM-DD
21         min_price = request.args.get('min_price', type=int)
22         max_price = request.args.get('max_price', type=int)
23         sort_by = request.args.get('sort_by', default="departure_time") # Default sorting
24         sort_order = request.args.get('sort_order', default="asc") # Ascending order by default
25
26         query = {}
27         if departure:
28             query["departure_airport"] = departure
29         if arrival:
30             query["arrival_airport"] = arrival
31         if date:
32             query["departure_time"] = {"$regex": f"^{date}"} # Match flights departing on the given date
33         if min_price is not None:
```



```
flight_dummy_data.py  app.py  decorators.py  flights.py  globals.py  flight_reviews.py  auth.py  globalacce: ▶ ☺ 📺 🔍 📄 ...
week_5 > blueprints > flights > flights.py > update_flight_status
70 def book_ticket():
96     new_booking = {
97         "_id": booking_id,
98         "passenger_name": data["passenger_name"],
99         "passport_number": data["passport_number"],
100        "email": data["email"],
101        "phone_number": data["phone_number"],
102        "flight_number": flight_number,
103        "seat_class": data["seat_class"],
104        "contact_details": data["contact_details"],
105        "booking_time": datetime.datetime.utcnow()
106    }
107
108
109    bookings.insert_one(new_booking)
110
111
112    flights.update_one({"flight_number": flight_number}, {"$inc": {"seats_available": -1}})
113
114    return make_response(jsonify({"message": "Booking successful", "booking_id": booking_id, "passenger_details": new_booking}), 201)
115
116    except Exception as e:
117        return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
118
119
120    ##### UPDATE FLIGHT STATUS #####
121    @flights_bp.route('/flights/<string:flight_number>/status', methods=['PUT'])
122    @jwt_required
123    @admin_required
124    def update_flight_status(flight_number):
125        """Update the status of a specific flight """
126        try:
127            data = request.json
```

```
flight_dummy_data.py  app.py  decorators.py  flights.py  globals.py  flight_reviews.py  auth.py  globalacce: ▶ ☺ 📺 🔍 📄 ...
week_5 > blueprints > flights > flights.py > update_flight_status
124 def update_flight_status(flight_number):
127     data = request.json
128     new_status = data.get("status")
129
130
131     valid_statuses = ["On Time", "Delayed", "Cancelled", "Boarding", "Departed", "Landed"]
132     if new_status not in valid_statuses:
133         return make_response(jsonify({"error": "Invalid status. Choose from: " + ", ".join(valid_statuses)}), 400)
134
135
136     result = flights.update_one({"flight_number": flight_number}, {"$set": {"status": new_status}})
137
138     if result.matched_count == 0:
139         return make_response(jsonify({"error": "Flight not found"}), 404)
140
141     return make_response(jsonify({"message": f"Flight {flight_number} status updated to '{new_status}'"}), 200)
142
143     except Exception as e:
144         return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
145
146
147
148    ##### GET ALL BOOKINGS #####
149    @flights_bp.route('/bookings', methods=['GET'])
150    @jwt_required
151    def get_all_bookings():
152        """Retrieve all bookings"""
153        try:
154            bookings_list = list(bookings.find({}, {"_id": 1, "passenger_name": 1, "flight_number": 1, "seat_class": 1}))
155
156            for booking in bookings_list:
157                booking["_id"] = str(booking["_id"])
158
```

```
flight_dummy_data.py app.py decorators.py flights.py • globals.py • flight_reviews.py auth.py globalacce: ▶ ◀ 🔍 📄 📁 📌 ...
week_5 > blueprints > flights > flights.py > update_flight_status
151 def get_all_bookings():
158     if not bookings_list:
159         return make_response(jsonify({"error": "No bookings found"}), 404)
160
161     return make_response(jsonify(bookings_list), 200)
162
163     except Exception as e:
164         return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
165
166
167 ##### GET A SPECIFIC BOOKING #####
168 @flights_bp.route('/bookings/<string:booking_id>', methods=['GET'])
169 @jwt_required
170 def get_booking(booking_id):
171     """Retrieve details of a specific booking"""
172     try:
173         booking = bookings.find_one({'_id': booking_id})
174
175         if not booking:
176             return make_response(jsonify({"error": "Booking not found"}), 404)
177
178         booking["_id"] = str(booking["_id"])
179         return make_response(jsonify(booking), 200)
180
181     except Exception as e:
182         return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
183
184
```

Auth.py

```
flight_dummy_data.py app.py decorators.py flights.py • globals.py • flight_reviews.py auth.py x globalacce: ▶ ◀ 🔍 📄 📁 📌 ...
week_5 > blueprints > auth > auth.py > logout
1 from flask import Blueprint, request, jsonify, make_response
2 from decorators import jwt_required
3
4 import jwt
5 import datetime
6 import bcrypt
7 import globalaccess
8
9 auth_bp = Blueprint('auth_bp', __name__)
10
11 blacklist = globalaccess.db.blacklist
12 users = globalaccess.db.users
13
14 ##### REGISTER #####
15 @auth_bp.route('/register', methods=['POST'])
16 def register():
17     data = request.json # Expecting JSON data
18
19     if not data or not data.get('username') or not data.get('password'):
20         return make_response(jsonify({'message': 'Username and password are required'}), 400)
21
22     if users.find_one({'username': data['username']}):
23         return make_response(jsonify({'message': 'Username already exists'}), 409)
24
25     hashed_password = bcrypt.hashpw(data['password'].encode('UTF-8'), bcrypt.gensalt())
26
27     new_user = {
28         'username': data['username'],
29         'password': hashed_password,
30         'admin': data.get('admin', False) # Default to False if not provided
31     }
32
33     users.insert_one(new_user)
```


flight_review.py

```
flight_dummy_data.py  app.py  decorators.py  flights.py  globals.py  flight_reviews.py x  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 x  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 X auth.py globalacce: D v ☺ 📺 🔄 📄 📁 ...
week_5 > blueprints > flight_reviews > flight_reviews.py > get_all_reviews
62 def add_review(f_id):
63     if not data or not data.get('username') or not data.get('comment') or not data.get('star'):
64         return make_response(jsonify({"error": "Missing required fields"}), 400)
65
66     try:
67         star = int(data.get('star'))
68         if star < 1 or star > 5:
69             return make_response(jsonify({"error": "Star rating must be between 1 and 5"}), 400)
70         except ValueError:
71             return make_response(jsonify({"error": "Star rating must be a number"}), 400)
72
73     flight = flights.find_one({'flight_number': f_id})
74
75     if not flight:
76         return make_response(jsonify({"error": "Flight not found"}), 404)
77
78     new_review = {
79         "id": str(uuid.uuid4()),
80         "username": data['username'],
81         "comment": data['comment'],
82         "star": star
83     }
84
85     flights.update_one({'flight_number': f_id}, {"$push": {"reviews": new_review}})
86     return make_response(jsonify({"message": "Review added successfully", "review": new_review}), 201)
87
88 except Exception as e:
89     return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
90
91 ##### UPDATE A REVIEW #####
92 @reviews_bp.route('/flights/<string:f_id>/reviews/<string:review_id>', methods=['PUT'])
93 @jwt required
94
95
96
97
98
```

```
99 def update_review(f_id, review_id):
100     """Update a specific review for a flight"""
101     try:
102         data = request.json
103         flight = flights.find_one({'flight_number': f_id})
104
105         if not flight:
106             return make_response(jsonify({"error": "Flight not found"}), 404)
107
108         updated_reviews = []
109         review_found = False
110
111         for review in flight["reviews"]:
112             if str(review["id"]) == review_id:
113                 review["username"] = data.get("username", review["username"])
114                 review["comment"] = data.get("comment", review["comment"])
115                 review["star"] = data.get("star", review["star"])
116                 review_found = True
117                 updated_reviews.append(review)
118
119         if not review_found:
120             return make_response(jsonify({"error": "Review not found"}), 404)
121
122         flights.update_one({'flight_number': f_id}, {"$set": {"reviews": updated_reviews}})
123         return make_response(jsonify({"message": "Review updated successfully"}), 200)
124
125     except Exception as e:
126         return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
127
128 ##### DELETE A REVIEW #####
129 @reviews_bp.route('/flights/<string:f_id>/reviews/<string:review_id>', methods=['DELETE'])
130 @jwt required
131
```

```
flight_dummy_data.py  app.py  decorators.py  flights.py  globals.py  flight_reviews.py x  auth.py  globalacce:
week_5 > blueprints > flight_reviews > flight_reviews.py > get_all_reviews
127
128
129 ##### DELETE A REVIEW #####
130 @reviews_bp.route('/flights/<string:f_id>/reviews/<string:review_id>', methods=['DELETE'])
131 @jwt_required
132 @admin_required
133 def delete_review(f_id, review_id):
134     """Admin can delete a specific review from a flight"""
135     try:
136         flight = flights.find_one({'flight_number': f_id})
137
138         if not flight:
139             return make_response(jsonify({"error": "Flight not found"}), 404)
140
141         updated_reviews = [review for review in flight["reviews"] if str(review["_id"]) != review_id]
142
143         if len(updated_reviews) == len(flight["reviews"]):
144             return make_response(jsonify({"error": "Review not found"}), 404)
145
146         flights.update_one({'flight_number': f_id}, {'$set': {"reviews": updated_reviews}})
147         return make_response(jsonify({"message": "Review deleted successfully"}), 200)
148
149     except Exception as e:
150         return make_response(jsonify({"error": "Internal Server Error", "details": str(e)}), 500)
151
```

globalaccess.py

```
it_dummy_data.py  app.py  decorators.py  flights.py  globals.py  flight_reviews.py  auth.py  globalaccess.py x
week_5 > globalaccess.py > ...
1  from pymongo import MongoClient
2
3  client = MongoClient("mongodb://localhost:27017")
4  db = client.flight_booking
5
6  SECRET_KEY = 'mysecret'
7
8
```


decorators.py

```
flight_dummy_data.py  app.py  decorators.py X  flights.py  globals.py  flight_reviews.py  auth.py  globalacce: ▶ 🔍 📄 🔄 🏠 ...
week_5 > decorators.py > jwt_required > jwt_required_wrapper
1  from flask import request, jsonify, make_response
2  import jwt
3  from functools import wraps
4  import globalaccess
5
6  blacklist = globalaccess.db.blacklist
7
8  def jwt_required(func):
9      @wraps(func)
10     def jwt_required_wrapper(*args, **kwargs):
11         token = None
12         if 'x-access-token' in request.headers:
13             token = request.headers['x-access-token']
14         if not token:
15             return make_response(jsonify({'message': 'Token is missing'}), 401)
16         try:
17             data = jwt.decode(token, globalaccess.SECRET_KEY, algorithms='HS256')
18         except:
19             return make_response(jsonify({'message': 'Token is invalid'}), 401)
20
21         bl_token = blacklist.find_one({'token': token})
22         if bl_token is not None:
23             return make_response(jsonify({'message': 'Token is blacklist invalid'}), 401)
24
25         return func(*args, **kwargs)
26     return jwt_required_wrapper
27
28 def admin_required(func):
29     @wraps(func)
30     def admin_required_wrapper(*args, **kwargs):
31         token = request.headers['x-access-token']
32         data = jwt.decode(token, globalaccess.SECRET_KEY, algorithms=["HS256"])
33         if data['admin']:
```

```
flight_dummy_data.py  app.py  decorators.py X  flights.py  globals.py  flight_reviews.py  auth.py  globalacce: ▶ 🔍 📄 🔄 🏠 ...
week_5 > decorators.py > jwt_required > jwt_required_wrapper
8  def jwt_required(func):
10     def jwt_required_wrapper(*args, **kwargs):
18         except:
19             return make_response(jsonify({'message': 'Token is invalid'}), 401)
20
21         bl_token = blacklist.find_one({'token': token})
22         if bl_token is not None:
23             return make_response(jsonify({'message': 'Token is blacklist invalid'}), 401)
24
25         return func(*args, **kwargs)
26     return jwt_required_wrapper
27
28 def admin_required(func):
29     @wraps(func)
30     def admin_required_wrapper(*args, **kwargs):
31         token = request.headers['x-access-token']
32         data = jwt.decode(token, globalaccess.SECRET_KEY, algorithms=["HS256"])
33         if data['admin']:
34             return func(*args, **kwargs)
35         else:
36             return make_response(jsonify({'message': 'Admin access required'}), 403)
37     return admin_required_wrapper
```

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

The screenshot shows the Postman interface with a collection named 'flight_booking_systems' and a sub-collection 'Flight_dataset'. The 'logout' endpoint is selected. The request is a GET method to 'http://127.0.0.1:5001/logout'. The 'Headers' tab is active, showing an 'x-access-token' header with a long alphanumeric value. The 'Body' tab shows a JSON response: { 'message': 'Logged out' }. The status is '200 OK' with a response time of 13 ms and a body size of 195 B.

Key	Value	Description
x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoibW...	

```
1 {
2   "message": "Logged out"
3 }
```

POST create_booking

The screenshot shows the Postman interface with a workspace named "My Workspace". The left sidebar displays a collection named "flight_booking_systems" containing a sub-collection "Flight_dataset". The "Flight_dataset" sub-collection is expanded, showing a list of requests. The "POST create_booking" request is selected and highlighted in orange. The main panel shows the details of this request. The URL is "http://127.0.0.1:5001/bookings". The method is "POST". The "Headers" tab is active, showing two headers: "x-access-token" with value "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9ibW..." and "Content-Type" with value "application/json". The "Body" tab is also active, showing a JSON body with the following structure:

```
{  "booking_id": "2e873ea4-f9b1-4fa4-ad5c-7b31b862a985",  "message": "Booking successful",  "passenger_details": {    "_id": "2e873ea4-f9b1-4fa4-ad5c-7b31b862a985",    "flight_number": "EK203",    "booking_time": "Fri, 14 Mar 2025 01:36:03 GMT",    "contact_details": "johndoe@gmail.com",    "email": "johndoe@gmail.com",    "flight_number": "EK203",    "passenger_name": "John Doe",    "passport_number": "A1234567",    "phone_number": "+441234567890",    "seat_class": "Economy"  }  }
```

The response status is "201 CREATED" with a response time of 8 ms and a body size of 641 B. The response body is shown in the "Body" tab as a JSON object.

GET get_all_booking

The screenshot shows the Postman interface with a workspace named "My Workspace". The left sidebar displays a collection named "flight_booking_systems" containing a sub-collection "Flight_dataset". The "Flight_dataset" sub-collection is expanded, showing a list of requests. The "GET get_all_booking" request is selected and highlighted in orange. The main panel shows the details of this request. The URL is "http://127.0.0.1:5001/bookings". The method is "GET". The "Headers" tab is active, showing one header: "x-access-token" with value "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9ibW...". The "Body" tab is also active, showing a JSON body with the following structure:

```
[  {    "_id": "34792b04-c7ca-4fe6-9da7-21bc1c091fcf",    "flight_number": "EK202",    "passenger_name": "John Doe",    "seat_class": "Economy"  }  ]
```

The response status is "200 OK" with a response time of 11 ms and a body size of 321 B. The response body is shown in the "Body" tab as a JSON array.

GET get_specific_booking

The screenshot shows the kspace API client interface. On the left, a sidebar lists the API endpoints under the 'Flight_dataset' collection. The 'get_specific_booking' endpoint is selected. The main panel displays the request details for a GET request to the URL 'http://127.0.0.1:5001/bookings/2d25eefd-0a9e-4609-9d0d-5e3265b8846e'. The 'Headers' tab is active, showing a single header: 'x-access-token' with a long alphanumeric value. The 'Body' tab is also visible, showing a JSON response with the following structure:

```
1 {
2   "_id": "2d25eefd-0a9e-4609-9d0d-5e3265b8846e",
3   "booking_time": "Fri, 14 Mar 2025 00:29:14 GMT",
4   "contact_details": "johndoe@example.com",
5   "flight_number": "EK203",
6   "passenger_name": "John Doe",
7   "seat_class": "Economy"
8 }
```

The response status is '200 OK' with a response time of 10 ms and a body size of 400 B.

PUT update_booking

The screenshot shows the kspace API client interface. On the left, the 'update_booking' endpoint is selected. The main panel displays the request details for a PUT request to the URL 'http://127.0.0.1:5001/bookings/2d25eefd-0a9e-4609-9d0d-5e3265b8846e'. The 'Body' tab is active, showing a JSON request body with the following structure:

```
1 {
2   "seat_class": "Business",
3   "contact_details": "johndoe@example.com"
4 }
5
```

The 'Body' tab also shows the JSON response, which is identical to the one in the previous screenshot, indicating a successful update. The response status is '200 OK' with a response time of 10 ms and a body size of 401 B.

DELETE delete_booking

The screenshot shows a REST client interface with a workspace on the left containing a collection named 'flight_booking_systems' and a sub-collection 'Flight_dataset'. The 'Flight_dataset' sub-collection is expanded, showing a list of endpoints. The 'delete_booking' endpoint is selected, and the 'DELETE' method is chosen. The URL is 'http://127.0.0.1:5001/bookings/Zd25eefd-0a9e-4609-9d0d-5e3265b8846e'. The 'Headers' tab is active, showing a single header 'x-access-token' with a value 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9ibW...'. The 'Body' tab is active, showing a JSON response:

```
{ 1: { 2: { 3: { "message": "Booking deleted successfully" } } }
```

. The status is '200 OK' with a response time of 10 ms and a body size of 213 B.

POST add reviews

The screenshot shows a REST client interface with a workspace on the left containing a collection named 'flight_booking_systems' and a sub-collection 'Flight_dataset'. The 'Flight_dataset' sub-collection is expanded, showing a list of endpoints. The 'add reviews' endpoint is selected, and the 'POST' method is chosen. The URL is 'http://127.0.0.1:5001/flights/A1127/reviews'. The 'Headers' tab is active, showing a single header 'x-access-token' with a value 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9ibW...'. The 'Body' tab is active, showing a JSON response:

```
{ 1: { 2: { 3: { "message": "Review added successfully", 4: { "review": { 5: { "id": "c2944462a-43ee-46b1-8a3f-8a683863df38", 6: { "comment": "Decent flight, food was excellent.", 7: { "star": 4, 8: { "username": "Patel" } } } } } } } }
```

. The status is '201 CREATED' with a response time of 6 ms and a body size of 378 B.

PUT update review

The screenshot shows a REST client interface with a sidebar on the left containing a list of endpoints under 'Flight_dataset'. The main panel displays a PUT request to `http://127.0.0.1:5001/flights/EK203/reviews/fa611e5a-2dfc-4dd8-bd9a-4d6347de388c`. The request body is a JSON object: `{ "comment": "Was a smooth flight, the crew members were very helpful. Could have provided more towels", "star": 4 }`. The response is a 200 OK status with a JSON body: `{ "message": "Review updated successfully" }`.

```
PUT http://127.0.0.1:5001/flights/EK203/reviews/fa611e5a-2dfc-4dd8-bd9a-4d6347de388c

{
  "comment": "Was a smooth flight, the crew members were very helpful. Could have provided more towels",
  "star": 4
}
```

```
{
  "message": "Review updated successfully"
}
```

DELETE Delete Reviews

The screenshot shows a REST client interface with a sidebar on the left containing a list of endpoints under 'Flight_dataset'. The main panel displays a DELETE request to `http://127.0.0.1:5001/flights/QR713/reviews/dbbaaf0-19f3-4ee1-ba13-8198c45fce7c`. The request headers include `x-access-token` with a value. The response is a 403 FORBIDDEN status with a JSON body: `{ "message": "Admin access required" }`.

```
DELETE http://127.0.0.1:5001/flights/QR713/reviews/dbbaaf0-19f3-4ee1-ba13-8198c45fce7c

Headers (8)
x-access-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm9ibW...
```

```
{
  "message": "Admin access required"
}
```

GET reviews for all flights

The screenshot shows the kspace API client interface. On the left, a sidebar lists various endpoints under the 'Flight_dataset' collection. The main panel displays a GET request to 'http://127.0.0.1:5001/flights/reviews'. The 'Headers' tab is active, showing 7 hidden headers. The 'Body' tab shows a JSON response with a 200 OK status, 30 ms response time, and 2.36 KB body size. The response body contains two flight review objects:

```
4 {
5   "comment": "Great flight experience!",
6   "flight_number": "EK262",
7   "star": 5,
8   "username": "john_doe"
9 },
10 {
11   "_id": "46e48c5e-96f6-4d1a-81ed-c99c4899f4a",
12   "comment": "Very comfortable seats and smooth flight!",
13   "flight_number": "LH789",
14   "star": 5,
15   "username": "travel_guru"
16 }
```

POST register

The screenshot shows the kspace API client interface. On the left, a sidebar lists various endpoints under the 'Flight_dataset' collection. The main panel displays a POST request to 'http://127.0.0.1:5001/register'. The 'Body' tab is active, showing a JSON request body with the following fields:

```
1 {
2   "username": "max",
3   "password": "max",
4   "admin": false
5 }
```

The response is a 201 CREATED status with a 651 ms response time and 218 B body size. The response body contains a JSON object with a success message:

```
1 {
2   "message": "User registered successfully"
3 }
```

GET login

The screenshot shows the Postman interface with a GET request to `http://127.0.0.1:5001/login`. The request is configured with Basic Auth, using the username `admin2` and password `admin_1`. The response is a 200 OK status, indicating a successful login. The response body is a JSON object containing a token.

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmRtaW4iOiIiYmRtaW4iOnRydWUsImV4cCI6MTYwOTU0ODh9.t8Hg1ShbuGRpky121zfrz3Wag87SN_8pa3GEJR7SLk"
}
```

POST flights

The screenshot shows the Postman interface with a POST request to `http://127.0.0.1:5001/flights`. The request is configured with a raw JSON body. The response is a 201 CREATED status, indicating a successful flight creation. The response body is a JSON object containing the details of the created flight.

```
{
  "_id": "67d34ffc970eab922adeb8e7",
  "airline": "Air India",
  "arrival_airport": "DEL",
  "arrival_time": "2025-12-02T10:00:00",
  "baggage_allowance": "23kg",
  "departure_airport": "DEL",
  "departure_time": "2025-12-01T23:50:00",
  "duration": 15.2,
  "entertainment": [],
  "flight_number": "AI127",
  "price": 1100.0
}
```

PUT flight update

The screenshot shows the Postman interface with a PUT request to `http://127.0.0.1:5001/flights/67d170ce98dd367a45cd1ccf`. The request body is a JSON object:

```
1 {
2   "airline": "Emirates",
3   "arrival_airport": "JFK"
4 }
5
```

The response is a 200 OK status with a 11 ms response time and 585 B of data. The response body is a JSON object:

```
1 {
2   "_id": "67d170ce98dd367a45cd1ccf",
3   "airline": "Emirates",
4   "arrival_airport": "JFK",
5   "arrival_time": "2025-06-10T16:00:00",
6   "baggage_allowance": "23kg",
7   "departure_airport": "DXB",
8   "departure_time": "2025-06-10T08:00:00",
9   "duration": 14,
10  "entertainment": [],
11  "flight_number": "EK202",
12  "price": 1200.0,

```

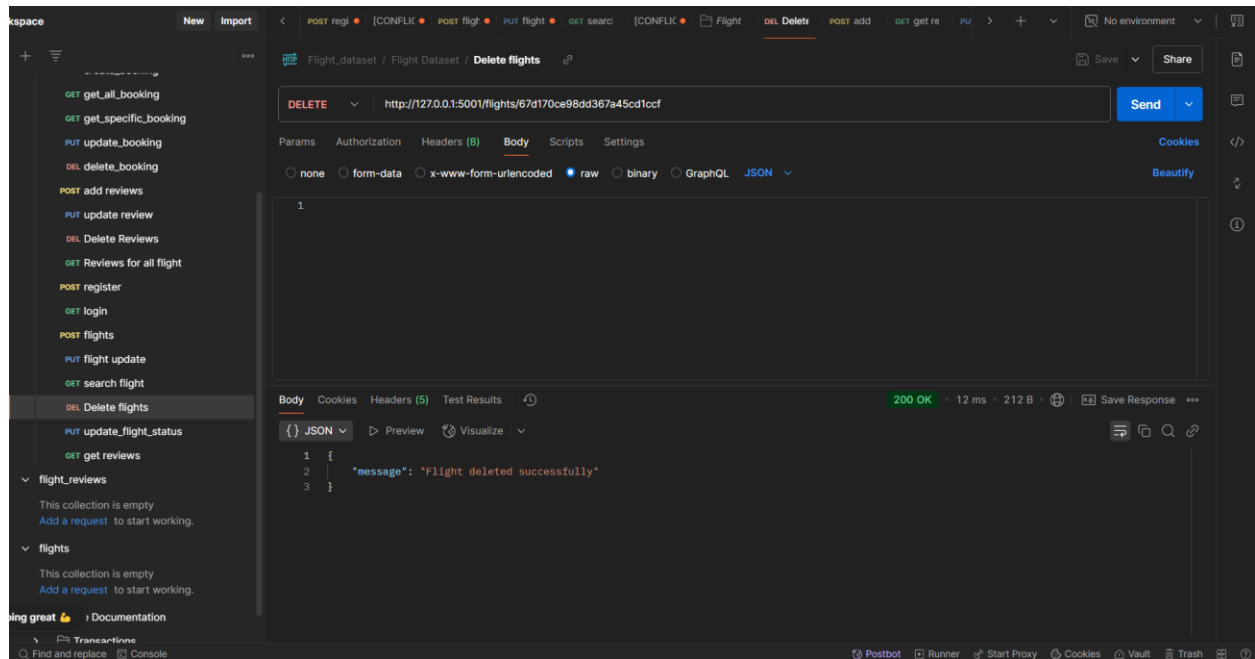
GET search flight

The screenshot shows the Postman interface with a GET request to `http://127.0.0.1:5001/flights/67d170ce98dd367a45cd1ccf`. The request headers include an `x-access-token` with the value `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VYljoibW...`. The response is a 200 OK status with a 6 ms response time and 585 B of data. The response body is a JSON object:

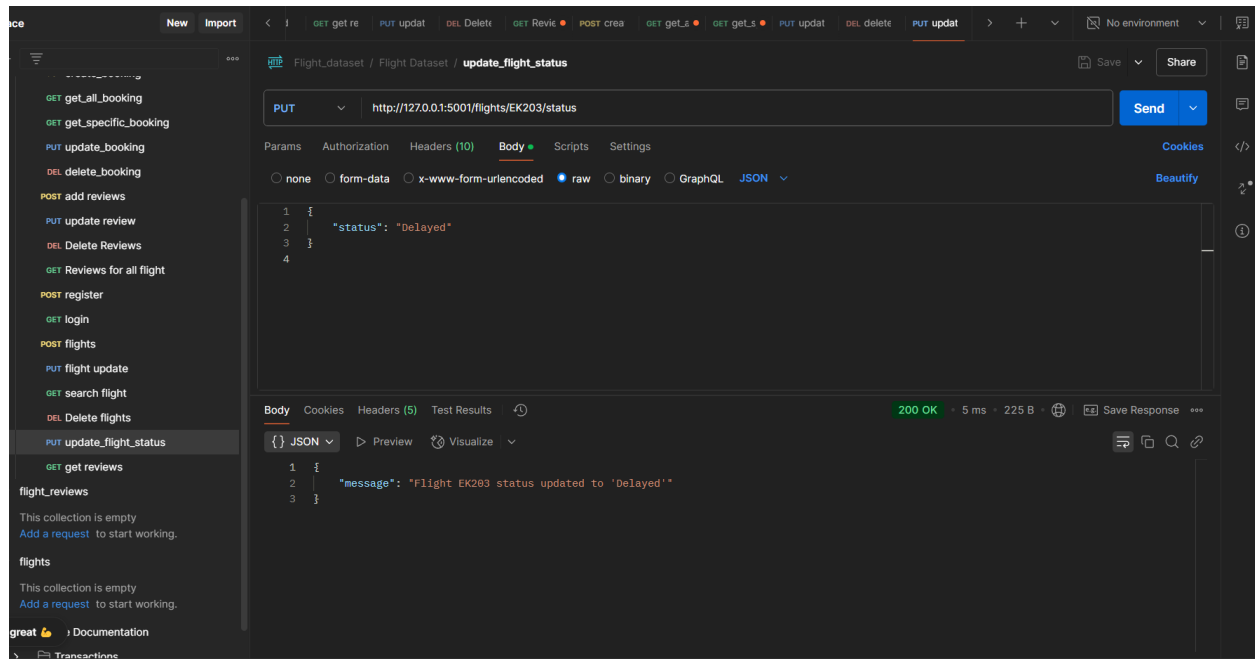
```
1 {
2   "_id": "67d170ce98dd367a45cd1ccf",
3   "airline": "Emirates",
4   "arrival_airport": "JFK",
5   "arrival_time": "2025-06-10T16:00:00",
6   "baggage_allowance": "23kg",
7   "departure_airport": "DXB",
8   "departure_time": "2025-06-10T08:00:00",
9   "duration": 14,
10  "entertainment": [],
11  "flight_number": "EK202",
12  "price": 1200.0,

```

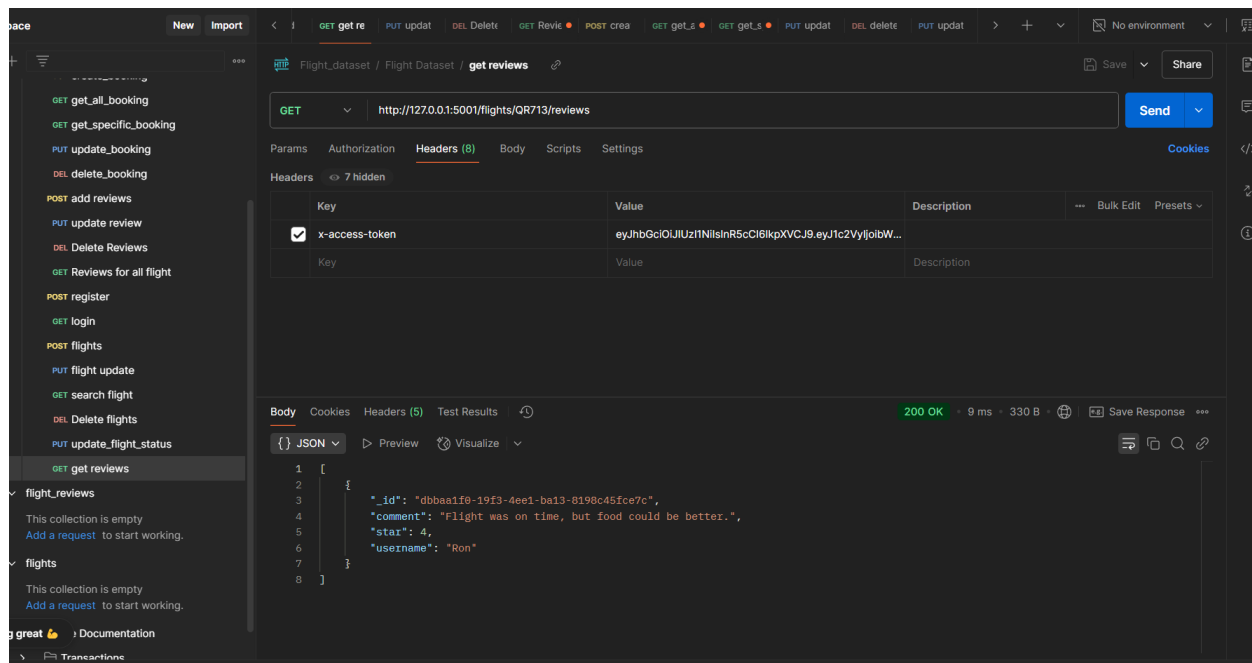
DELETE delete flights



PUT update_flight_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.