Summer Internship Project

(June 11, 2025 – August 11, 2025)

Flight Booking Management System

Submitted By: Archit Jha (Backend Intern)

Mentor: Palak Chawla (Software Developer)

INDEX

Objective

The **Flight Management System** is a backend-only project that allows users to securely sign up, log in, and search for details using API endpoints tested via **Postman**. Instead of using public APIs, the system retrieves flight data from a **MongoDB database** containing manually inserted dummy data. Access to flight search is protected using **JWT authentication**, ensuring only logged-in users can retrieve flight details. Built with **Node.js**, **Express.js**, and **Mongoose**, the project demonstrates key backend concepts like RESTful API design, password hashing with **bcrypt**, and secure route protection.

Problem Statement

In real-world airline and travel booking systems, the management of user authentication and real-time flight search requires secure, efficient, and scalable backend systems. However, building such systems using live flight APIs can be expensive, complex, and dependent on third-party services.

Moreover, many beginner and intermediate developers find it challenging to understand how the backend of a flight booking system works — including user login/signup flows, database integration, and secure access to protected data. There is also a need for testing backend APIs in a structured environment without relying on user interfaces or production-level data.

To address these challenges, this project proposes a backend-only **Flight Management System** that simulates a flight booking experience using **predefined (dummy) flight data** stored in **MongoDB**. The system includes **user authentication with secure login/signup**, **JWT-based route protection**, and a **flight search API** — all of which are tested through the **Postman API client**.

This solution provides a controlled, offline-ready environment to practice and demonstrate core backend development skills while simulating how a real flight booking platform might operate behind the scenes.

Technologies, stack and tools used

Backend Stack

Component	Technology Used	Purpose
Runtime	**Node.js**	Server-side JavaScript execution
Framework	**Express.js**	To create RESTful API endpoints
Database	**MongoDB**	To store dummy flight data and user credentials
ODM Library	**Mongoose**	To define schemas and interact with MongoDB
Security	**bcryptjs**	To hash passwords securely before saving
Authentication	**jsonwebtoken (JWT)**	To generate and verify authentication tokens
HTTP Client	**axios / node-fetch**	(Optional) For external API simulation if required

Testing & Simulation Tools

Tool	Purpose
Postman	Used to test and verify all backend API endpoints (e.g., login, signup, flight search) without a frontend UI

Development Tools

Tool	Purpose
Visual Studio Code	Used as the primary code editor for writing and managing the Node.js backend code
MongoDB Compass	GUI tool to view and manage MongoDB collections (users, flights)

Module 1

Login & Signup:

The **Signup and Login Module** of the Flight Management System is responsible for managing user registration and authentication. It ensures that only authorized users can access protected parts of the application, such as the flight search functionality.

Signup Functionality

- Method: POST
- Endpoint: 8000/api/register
- During signup, the user is required to provide:
    - **Full name**
    - **Email**
    - **Phone number**
    - **Password**
- **Validation logic** is implemented to ensure data integrity:
    - The **email** must include the '@' symbol to follow a basic email format.
    - The **phone number** must be exactly **10 digits long**.
- If the input fails validation, an appropriate error message is returned.
- Upon passing validation:
    - The password is **hashed using bcrypt** to ensure security.
    - The user data is then stored in the **MongoDB** database using a **Mongoose schema**.

This process ensures that only valid and secure data is accepted into the system.

Login Functionality

- Method: POST
- Endpoint: 8000/api/login
- The login endpoint verifies a user's **email and password**.
- If the email exists and the password matches (checked using bcrypt.compare()):
    - A **JWT (JSON Web Token)** is generated and sent back in the response.
    - This token is used to authenticate subsequent requests to protected routes.

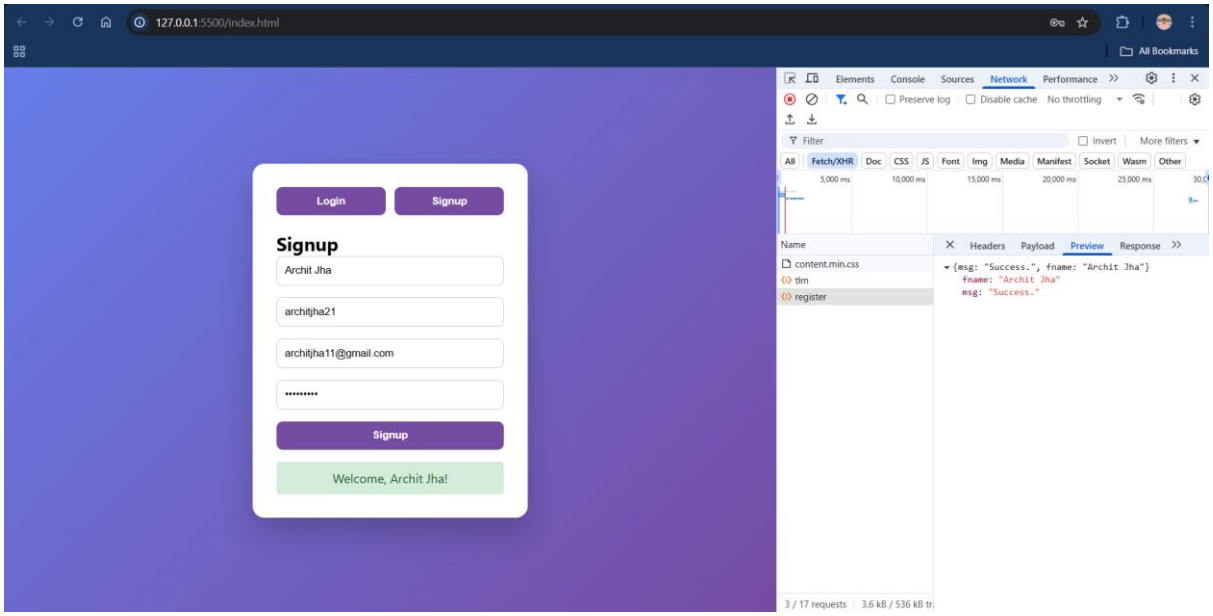
Access Control Using JWT

- The system uses **middleware** to verify JWT tokens.
- Only users with valid tokens can access protected routes like /search-flights.
- This ensures that unauthenticated users cannot access sensitive data.

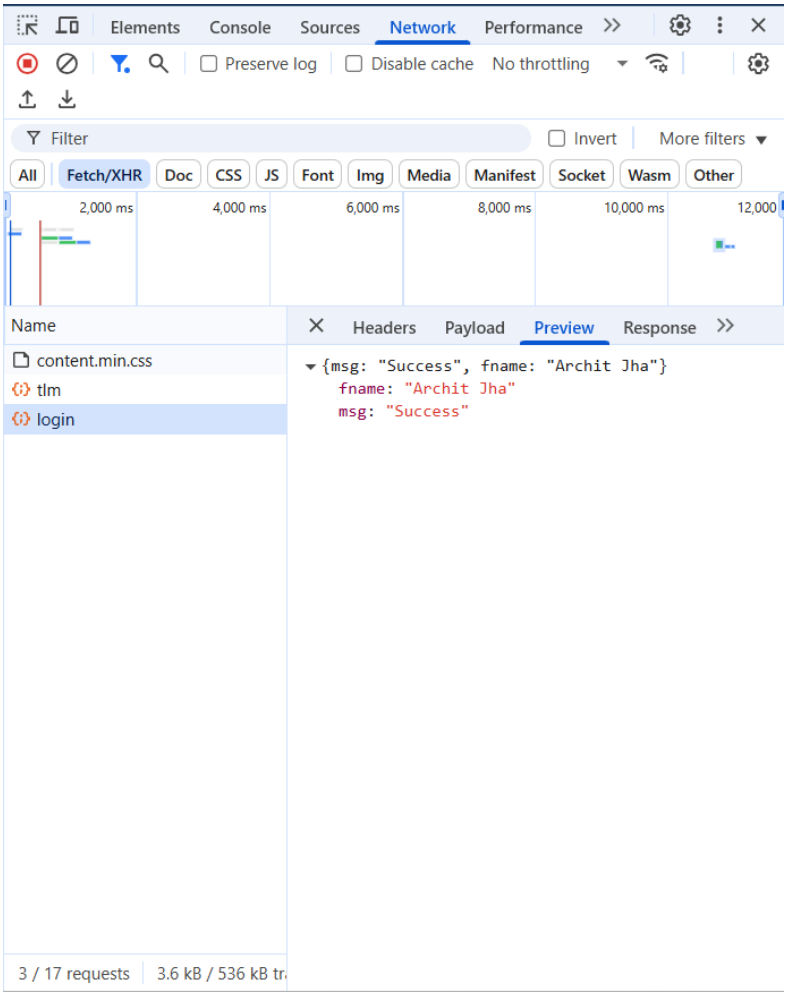
Testing in Postman

- The module is tested entirely using **Postman**.
- Signup and login requests are sent with JSON payloads.
- After a successful login, the returned JWT token is added in the Authorization header (as Bearer <token>) to test protected APIs like flight search.

  - Signup page:



- Login page


Elements
Console
Sources
Network
Performance
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
Socket
Wasm
Other
2,000 ms
4,000 ms
6,000 ms
8,000 ms
10,000 ms
12,000
Name
Headers
Payload
Preview
Response
content.min.css
tlm
login
{msg: "Success", fname: "Archit Jha"}
fname: "Archit Jha"
msg: "Success"
3 / 17 requests
3.6 kB / 536 kB tr

- Entry in Database

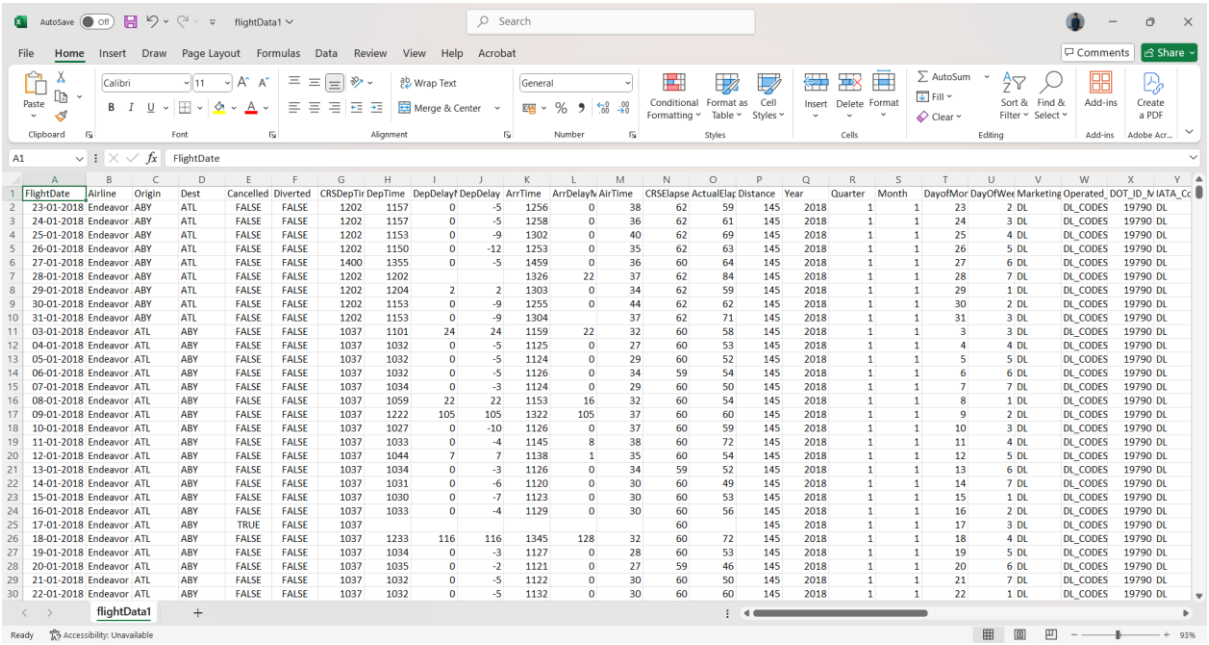
```
test> use userData
switched to db userData
userData> db.users.find({})
[
  {
    _id: ObjectId('686caf3121e39c0195f7acc0'),
    username: 'architjha21',
    fname: 'Archit Jha',
    emailid: 'architjha11@gmail.com',
    password: '$2b$10$rLM.Zdoj5EbSIgbli42LiedECnyVj.VcZN7xdTnbFU56FCRu5./VK',
    role: 'user',
    __v: 0
  },
  {
    _id: ObjectId('686cb3ae894f5c77bb4951b8'),
    username: 'anurag123',
    fname: 'Anurag',
    emailid: 'anuragkumeri123@gmail.com',
    password: '$2b$10$1oostd.nLLyhy3lDClYdtOauRMjETTvz8Fz9V9tqBurZgOiJs2u1.',
    role: 'user',
    __v: 0
  }
]
userData>
```

Module 2

Data Collection

The flight-related datasets used in this project were sourced from **Kaggle**, a popular online platform for datasets and data science competitions. The data was curated and cleaned to match the application's schema requirements, including flight details, pricing, layovers, seat availability, and city information. Some additional fields like **layoverDuration**, **seatCounts**, and **onDate** were generated or randomized using Excel formulas for simulation purposes.

- Raw data:



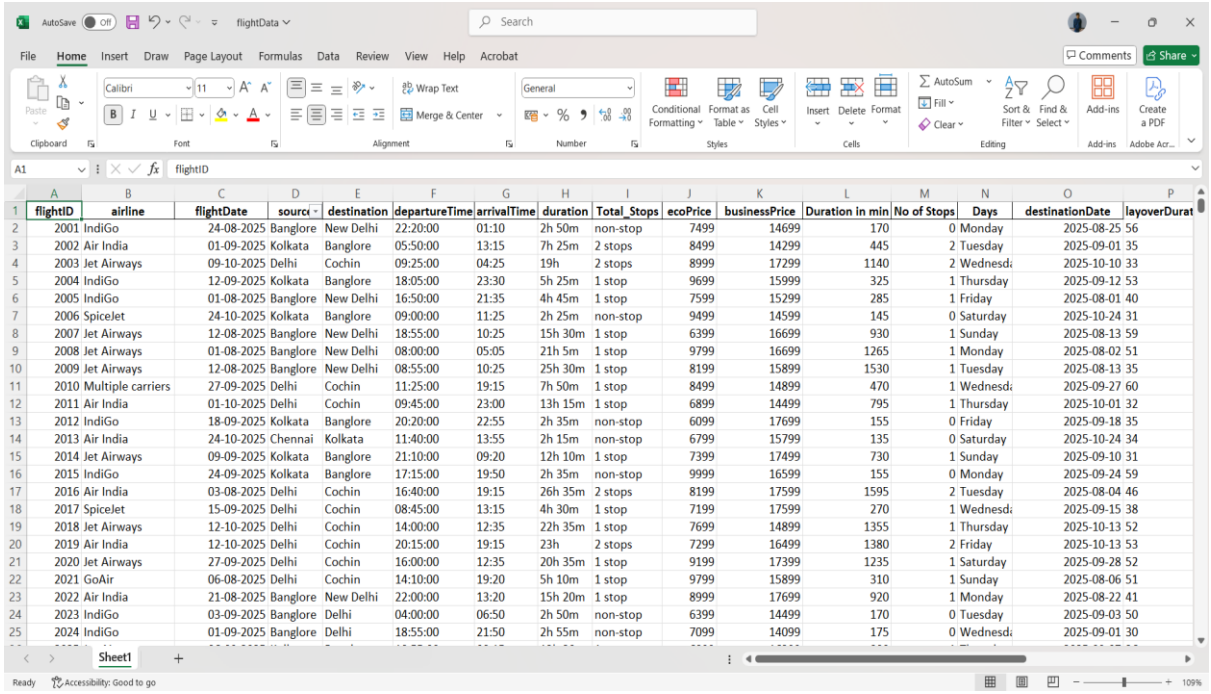
FlightDate	Airline	Origin	Dest	Cancelled	Diverted	CRSDepTir	DepTime	DepDelayI	DepDelay	ArrTime	ArrDelayN	AirTime	CRSElapse	ActualElap	Distance	Year	Quarter	Month	DayofMor	DayOfWe	Marketing	Operated_	DOT_ID_M	IATA_Cc
23-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1157	0	-5	1256	0	38	62	59	145	2018	1	1	23	2	DL	DL_CODES	19790	DL
24-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1157	0	-5	1258	0	36	62	61	145	2018	1	1	24	3	DL	DL_CODES	19790	DL
25-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1153	0	-9	1302	0	40	62	69	145	2018	1	1	25	4	DL	DL_CODES	19790	DL
26-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1150	0	-12	1253	0	35	62	63	145	2018	1	1	26	5	DL	DL_CODES	19790	DL
27-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1400	1355	0	-5	1459	0	36	60	64	145	2018	1	1	27	6	DL	DL_CODES	19790	DL
28-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1202			1326	22	37	62	84	145	2018	1	1	28	7	DL	DL_CODES	19790	DL
29-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1204	2	2	1303	0	34	62	59	145	2018	1	1	29	1	DL	DL_CODES	19790	DL
30-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1153	0	-9	1255	0	44	62	62	145	2018	1	1	30	2	DL	DL_CODES	19790	DL
31-01-2018	Endeavor	ABY	ATL	FALSE	FALSE	1202	1153	0	-9	1304		37	62	71	145	2018	1	1	31	3	DL	DL_CODES	19790	DL
03-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1101	24	24	1159	22	32	60	58	145	2018	1	1	3	3	DL	DL_CODES	19790	DL
04-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1032	0	-5	1125	0	27	60	53	145	2018	1	1	4	4	DL	DL_CODES	19790	DL
05-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1032	0	-5	1124	0	29	60	52	145	2018	1	1	5	5	DL	DL_CODES	19790	DL
06-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1032	0	-5	1126	0	34	59	54	145	2018	1	1	6	6	DL	DL_CODES	19790	DL
07-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1034	0	-3	1124	0	29	60	50	145	2018	1	1	7	7	DL	DL_CODES	19790	DL
08-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1059	22	22	1153	16	32	60	54	145	2018	1	1	8	1	DL	DL_CODES	19790	DL
09-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1222	105	105	1322	105	37	60	60	145	2018	1	1	9	2	DL	DL_CODES	19790	DL
10-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1027	0	-10	1126	0	37	60	59	145	2018	1	1	10	3	DL	DL_CODES	19790	DL
11-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1033	0	-4	1145	8	38	60	72	145	2018	1	1	11	4	DL	DL_CODES	19790	DL
12-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1044	7	7	1138	1	35	60	54	145	2018	1	1	12	5	DL	DL_CODES	19790	DL
13-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1034	0	-3	1126	0	34	59	52	145	2018	1	1	13	6	DL	DL_CODES	19790	DL
14-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1031	0	-6	1120	0	30	60	49	145	2018	1	1	14	7	DL	DL_CODES	19790	DL
15-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1030	0	-7	1123	0	30	60	53	145	2018	1	1	15	1	DL	DL_CODES	19790	DL
16-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1033	0	-4	1129	0	30	60	56	145	2018	1	1	16	2	DL	DL_CODES	19790	DL
17-01-2018	Endeavor	ATL	ABY	TRUE	FALSE	1037							60		145	2018	1	1	17	3	DL	DL_CODES	19790	DL
18-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1233	116	116	1345	128	32	60	72	145	2018	1	1	18	4	DL	DL_CODES	19790	DL
19-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1034	0	-3	1127	0	28	60	53	145	2018	1	1	19	5	DL	DL_CODES	19790	DL
20-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1035	0	-2	1121	0	27	59	46	145	2018	1	1	20	6	DL	DL_CODES	19790	DL
21-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1032	0	-5	1122	0	30	60	50	145	2018	1	1	21	7	DL	DL_CODES	19790	DL
22-01-2018	Endeavor	ATL	ABY	FALSE	FALSE	1037	1032	0	-5	1132	0	30	60	60	145	2018	1	1	22	1	DL	DL_CODES	19790	DL

Data pre-processing

To align the dataset with the application's backend schema, the data was thoroughly pre-processed. Fields were cleaned, normalized, and transformed as needed to ensure consistency across all modules. The data was organized into five main MongoDB collections:

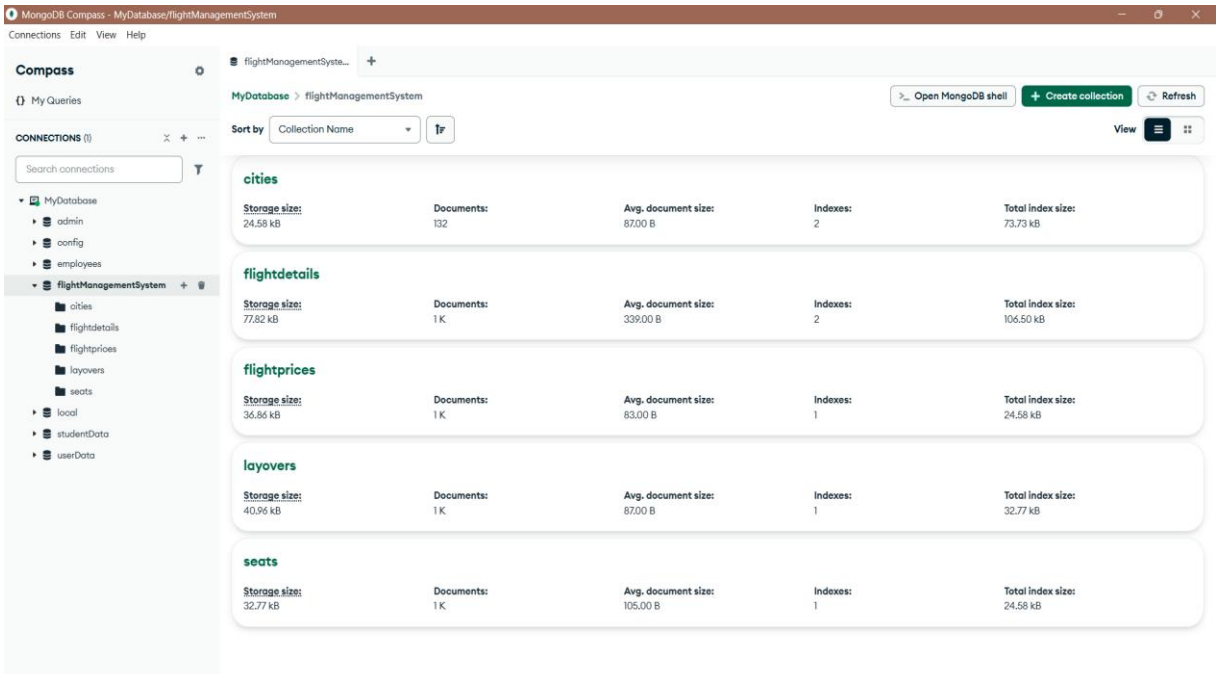
- **cities** (containing city information)
- **flightdetails** (storing flight schedules and routes)
- **flightprice** (containing pricing information for economy and business classes)
- **seats** (tracking available seats per flight)
- **layovers** (detailing intermediate stop durations). Additional fields such as layoverDuration and onDate were either generated or adjusted to match the application's requirements.

- Pre-processed data



flightID	airline	flightDate	source	destination	departureTime	arrivalTime	duration	Total_Stops	ecoPrice	businessPrice	Duration in min	No of Stops	Days	destinationDate	layoverDurat
2001	IndiGo	24-08-2025	Banglore	New Delhi	22:20:00	01:10	2h 50m	non-stop	7499	14699	170	0	Monday	2025-08-25	56
2002	Air India	01-09-2025	Kolkata	Banglore	05:50:00	13:15	7h 25m	2 stops	8499	14299	445	2	Tuesday	2025-09-01	35
2003	Jet Airways	09-10-2025	Delhi	Cochin	09:25:00	04:25	19h	2 stops	8999	17299	1140	2	Wednesd	2025-10-10	33
2004	IndiGo	12-09-2025	Kolkata	Banglore	18:05:00	23:30	5h 25m	1 stop	9699	15999	325	1	Thursday	2025-09-12	53
2005	IndiGo	01-08-2025	Banglore	New Delhi	16:50:00	21:35	4h 45m	1 stop	7599	15299	285	1	Friday	2025-08-01	40
2006	SpiceJet	24-10-2025	Kolkata	Banglore	09:00:00	11:25	2h 25m	non-stop	9499	14599	145	0	Saturday	2025-10-24	31
2007	Jet Airways	12-08-2025	Banglore	New Delhi	18:55:00	10:25	15h 30m	1 stop	6399	16699	930	1	Sunday	2025-08-13	59
2008	Jet Airways	01-08-2025	Banglore	New Delhi	08:00:00	05:05	21h 5m	1 stop	9799	16699	1265	1	Monday	2025-08-02	51
2009	Jet Airways	12-08-2025	Banglore	New Delhi	08:55:00	10:25	25h 30m	1 stop	8199	15899	1530	1	Tuesday	2025-08-13	35
2010	Multiple carriers	27-09-2025	Delhi	Cochin	11:25:00	19:15	7h 50m	1 stop	8499	14899	470	1	Wednesd	2025-09-27	60
2011	Air India	01-10-2025	Delhi	Cochin	09:45:00	23:00	13h 15m	1 stop	6899	14499	795	1	Thursday	2025-10-01	32
2012	IndiGo	18-09-2025	Kolkata	Banglore	20:20:00	22:55	2h 35m	non-stop	6099	17699	155	0	Friday	2025-09-18	35
2013	Air India	24-10-2025	Chennai	Kolkata	11:40:00	13:55	2h 15m	non-stop	6799	15799	135	0	Saturday	2025-10-24	34
2014	Jet Airways	09-09-2025	Kolkata	Banglore	21:10:00	09:20	12h 10m	1 stop	7399	17499	730	1	Sunday	2025-09-10	31
2015	IndiGo	24-09-2025	Kolkata	Banglore	17:15:00	19:50	2h 35m	non-stop	9999	16599	155	0	Monday	2025-09-24	59
2016	Air India	03-08-2025	Delhi	Cochin	16:40:00	19:15	26h 35m	2 stops	8199	17599	1595	2	Tuesday	2025-08-04	46
2017	SpiceJet	15-09-2025	Delhi	Cochin	08:45:00	13:15	4h 30m	1 stop	7199	17599	270	1	Wednesd	2025-09-15	38
2018	Jet Airways	12-10-2025	Delhi	Cochin	14:00:00	12:35	22h 35m	1 stop	7699	14899	1355	1	Thursday	2025-10-13	52
2019	Air India	12-10-2025	Delhi	Cochin	20:15:00	19:15	23h	2 stops	7299	16499	1380	2	Friday	2025-10-13	53
2020	Jet Airways	27-09-2025	Delhi	Cochin	16:00:00	12:35	20h 35m	1 stop	9199	17399	1235	1	Saturday	2025-09-28	52
2021	GoAir	06-08-2025	Delhi	Cochin	14:10:00	19:20	5h 10m	1 stop	9799	15899	310	1	Sunday	2025-08-06	51
2022	Air India	21-08-2025	Banglore	New Delhi	22:00:00	13:20	15h 20m	1 stop	8999	17699	920	1	Monday	2025-08-22	41
2023	IndiGo	03-09-2025	Banglore	Delhi	04:00:00	06:50	2h 50m	non-stop	6399	14499	170	0	Tuesday	2025-09-03	50
2024	IndiGo	01-09-2025	Banglore	Delhi	18:55:00	21:50	2h 55m	non-stop	7099	14099	175	0	Wednesd	2025-09-01	30

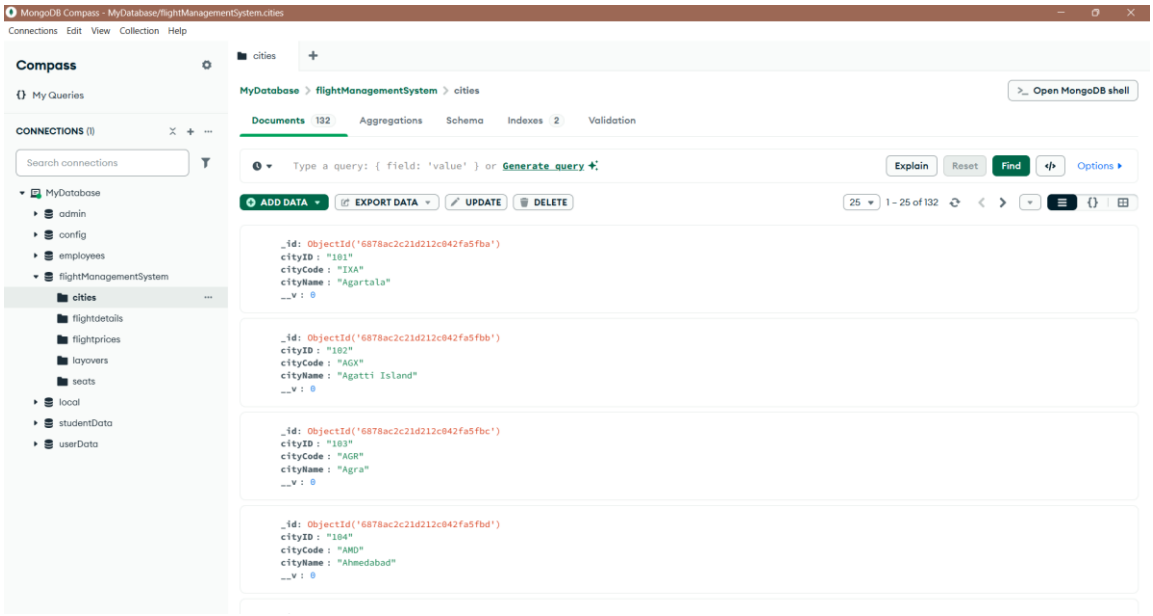
- Collections imported in MongoDB



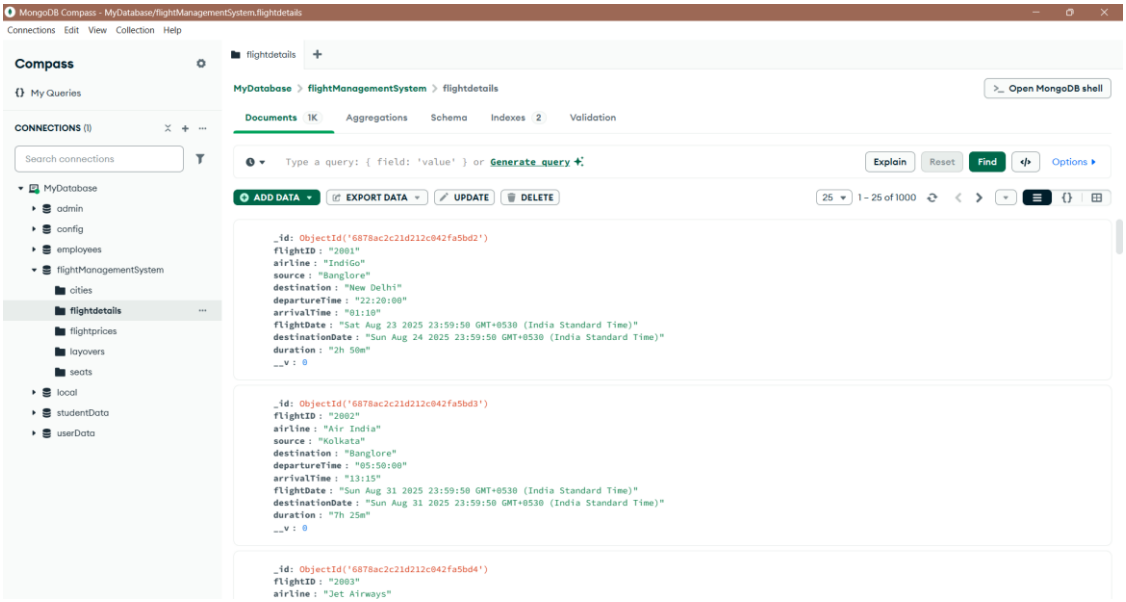
Data Insertion

The data used in this project was imported from **pre-processed Excel files** tailored to meet the structural requirements of the flight management system. Using **Node.js** and the **xlsx** library, Excel sheets were read and converted into JSON format. Dedicated **Mongoose schemas** were created for each MongoDB collection to maintain data integrity and validation. The data was then inserted into MongoDB using the insertMany() method for each collection.

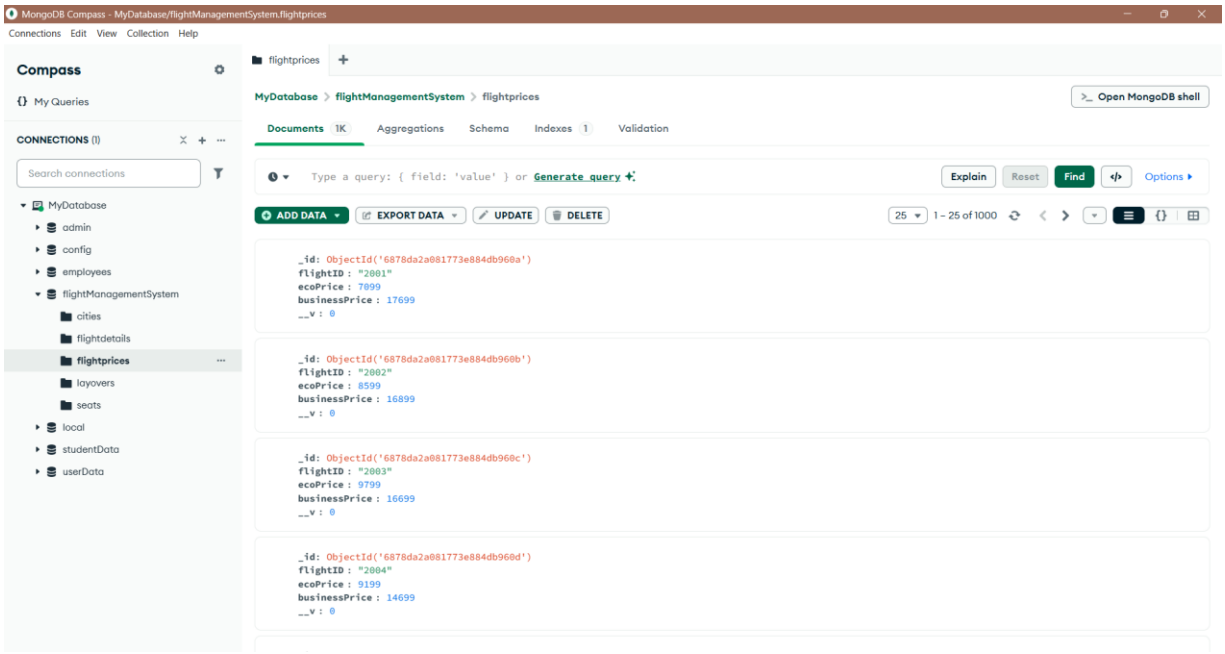
- Cities collection



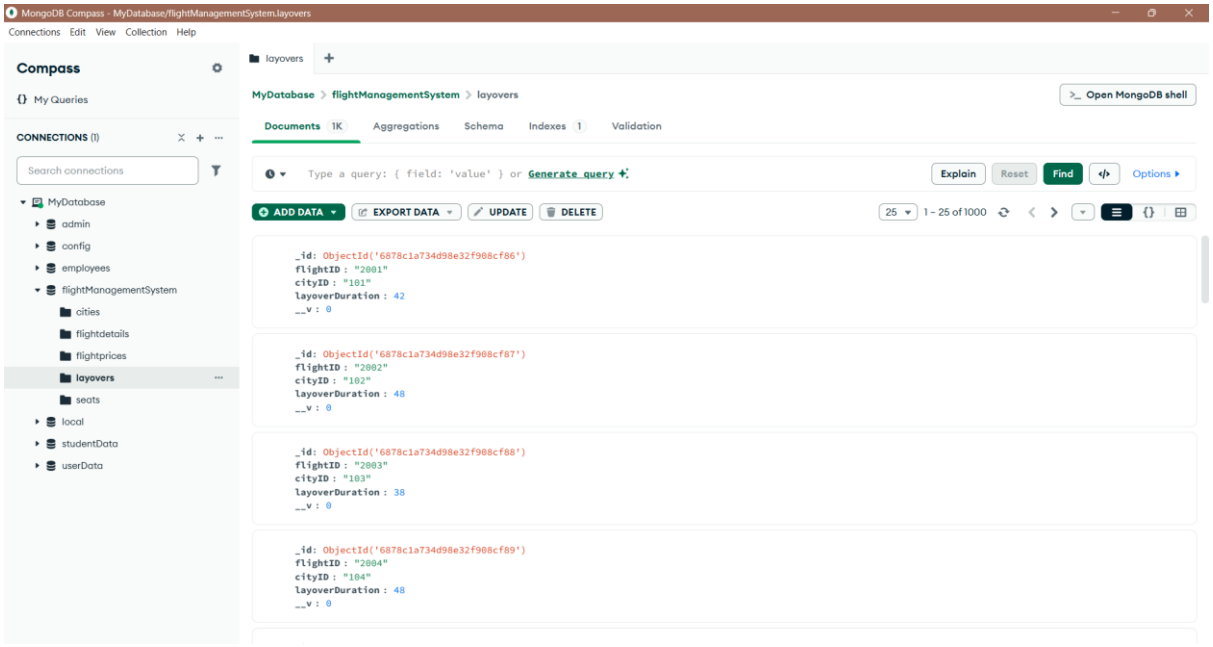
- flightDetails collection



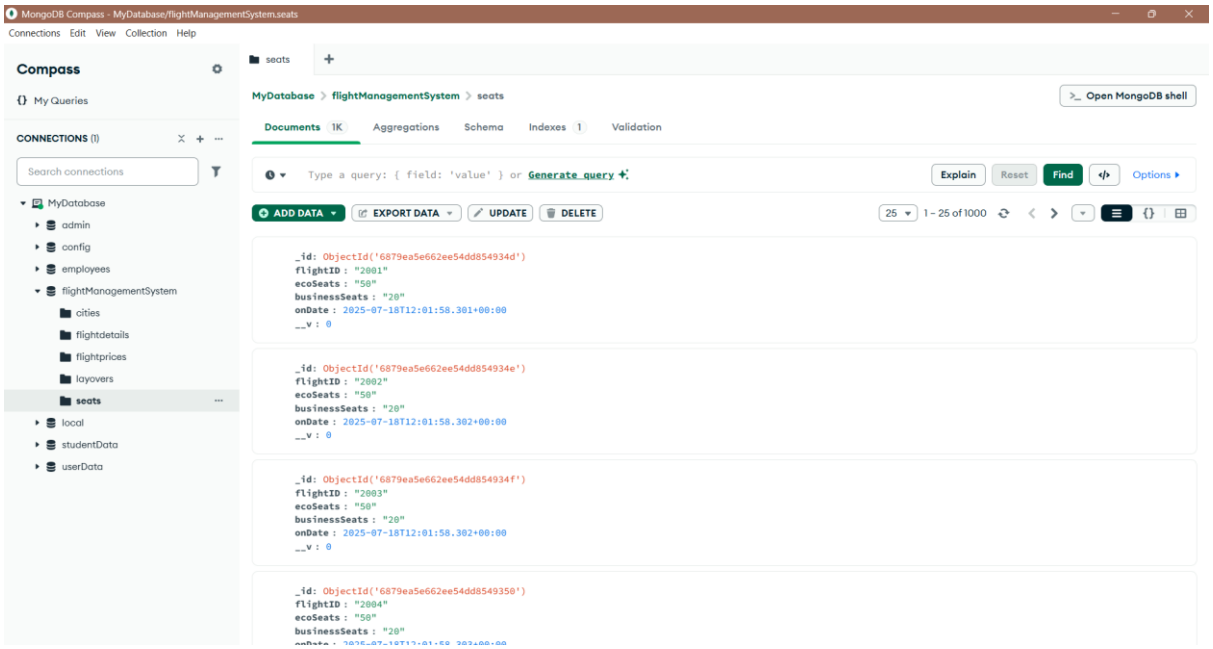
- flightprices collection



- layovers collection



- seats collection



Data Updation

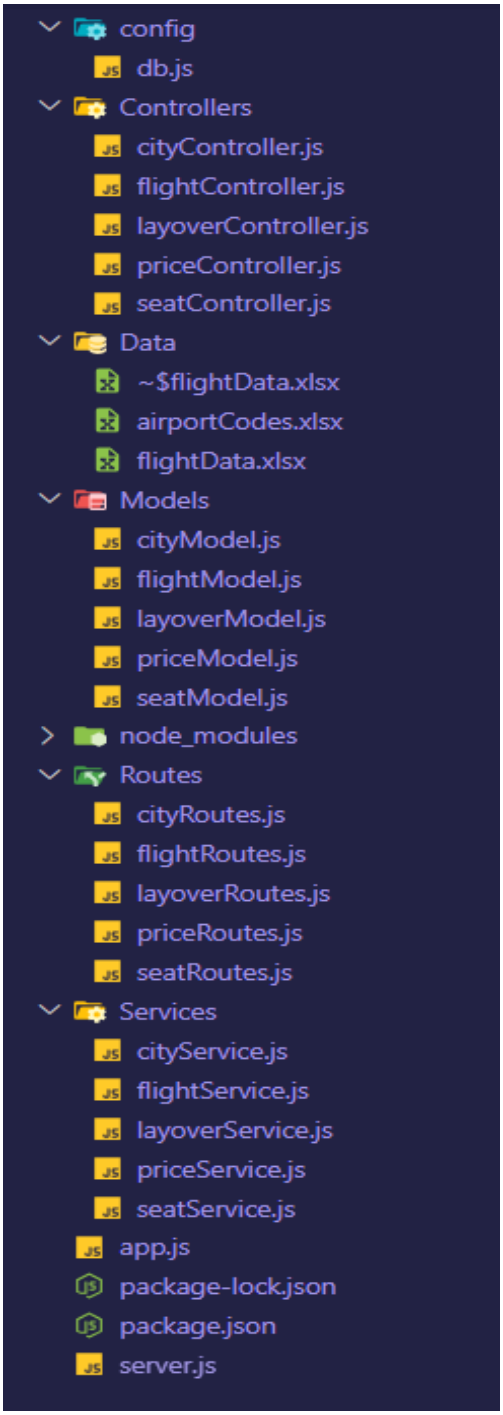
Data updation in the flight management system ensures that key fields remain current and accurate. The onDate field in the seats collection is designed to automatically capture the current date (IST) at the time of insertion, reflecting real-time availability. Other fields such as ecoSeats and **businessSeat** are dynamically updated. whenever a booking is made—deduction of required number of seats per transaction. Similarly, updates can be performed on flightprice, layovers, or flightdetails collections when schedules, pricing, or routing information change. This ensures the system maintains up-to-date records aligned with actual flight operations.

Module 3

Folder Structure:

I have organized my Node.js project by creating a clean folder structure to improve readability and maintainability. The main entry point is app.js, and I have separated the code into folders like:

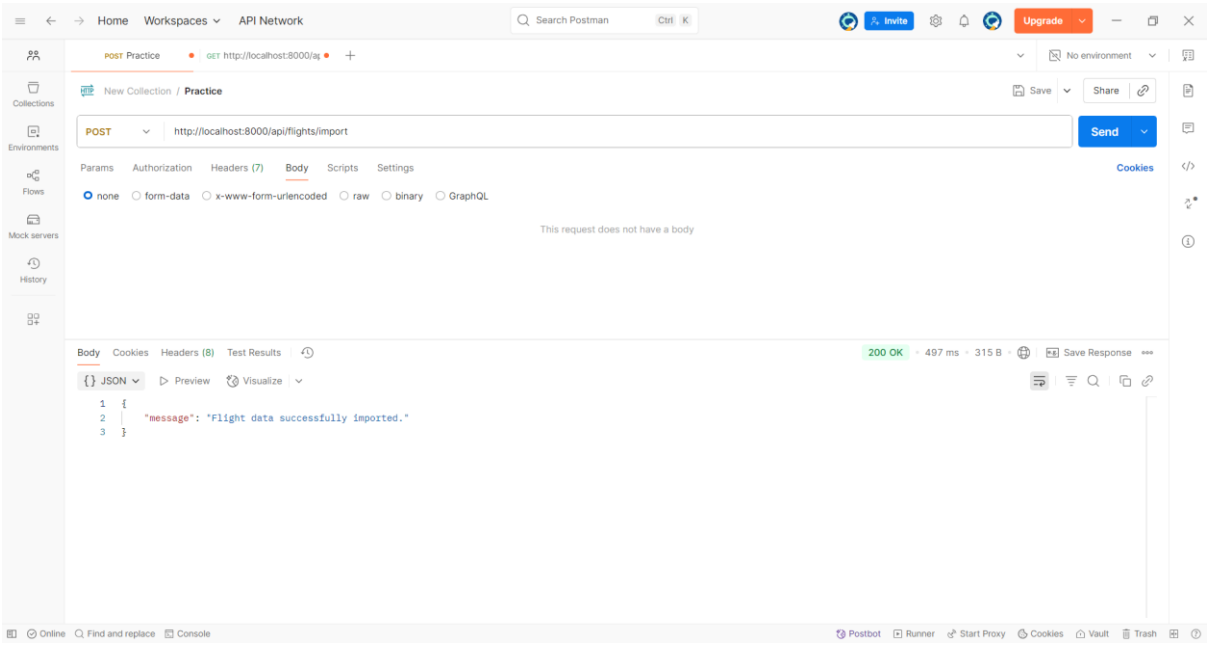
- **routes** for defining API endpoints
- **controllers** for handling request-response logic
- **services** for business logic
- **config** folder to manage the database connection setup



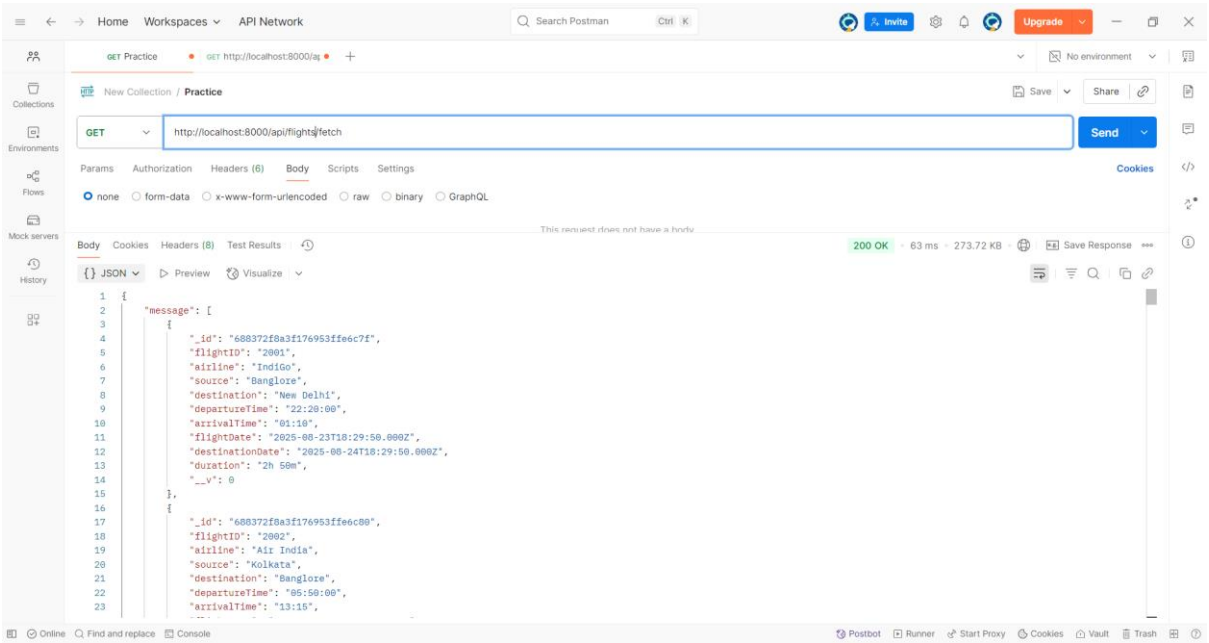
GET/POST API Method

I have implemented both **GET** and **POST** APIs in my project. The GET APIs are used to fetch data from the database. The POST APIs allow inserting new data into the respective collections. Each API is connected through routes, handled by **controllers**, and processed with the help of **services** for clean and modular code management.

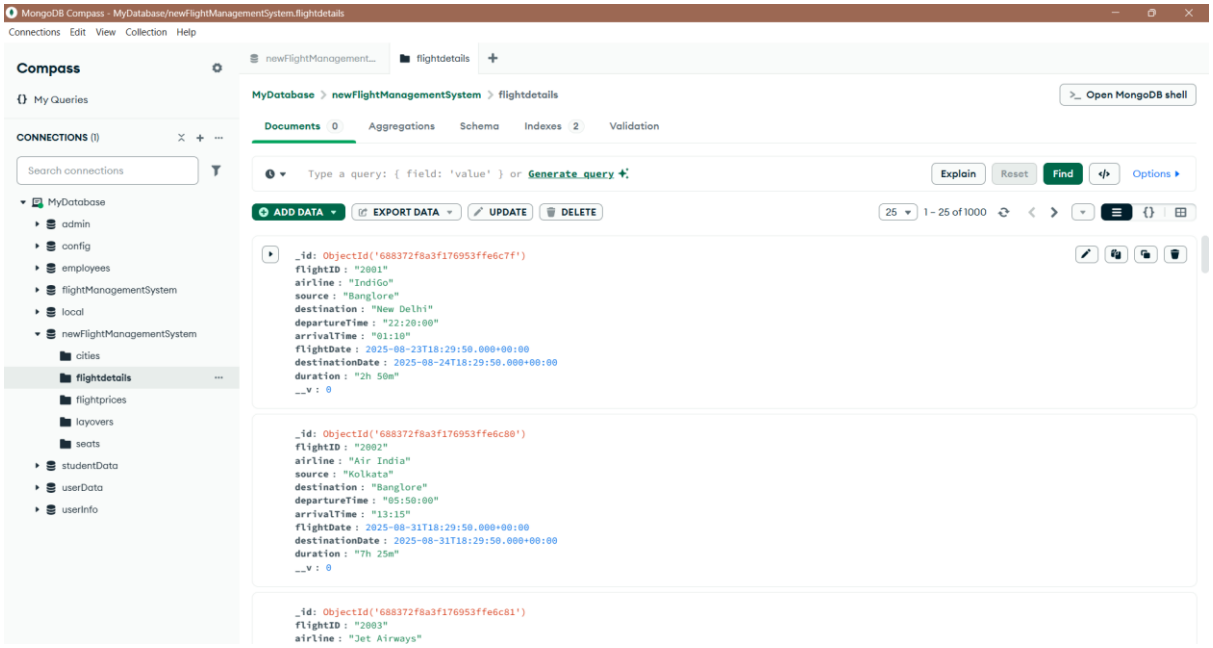
- flightDetails:
  - POST API-



  - GET API-

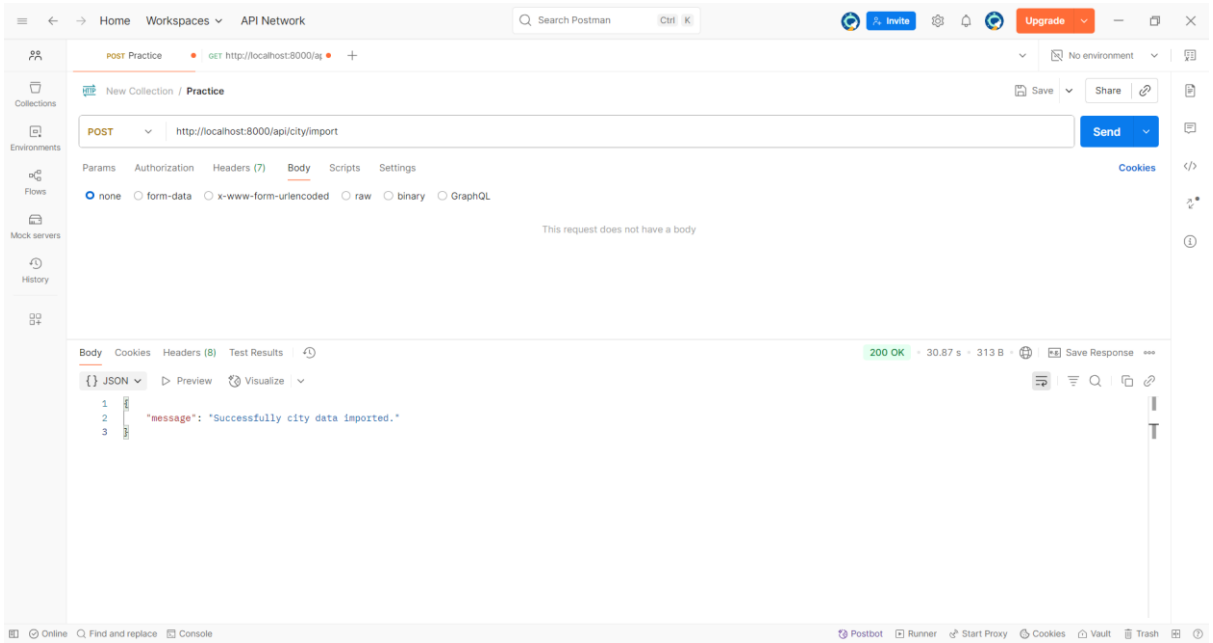


- DB Collection-

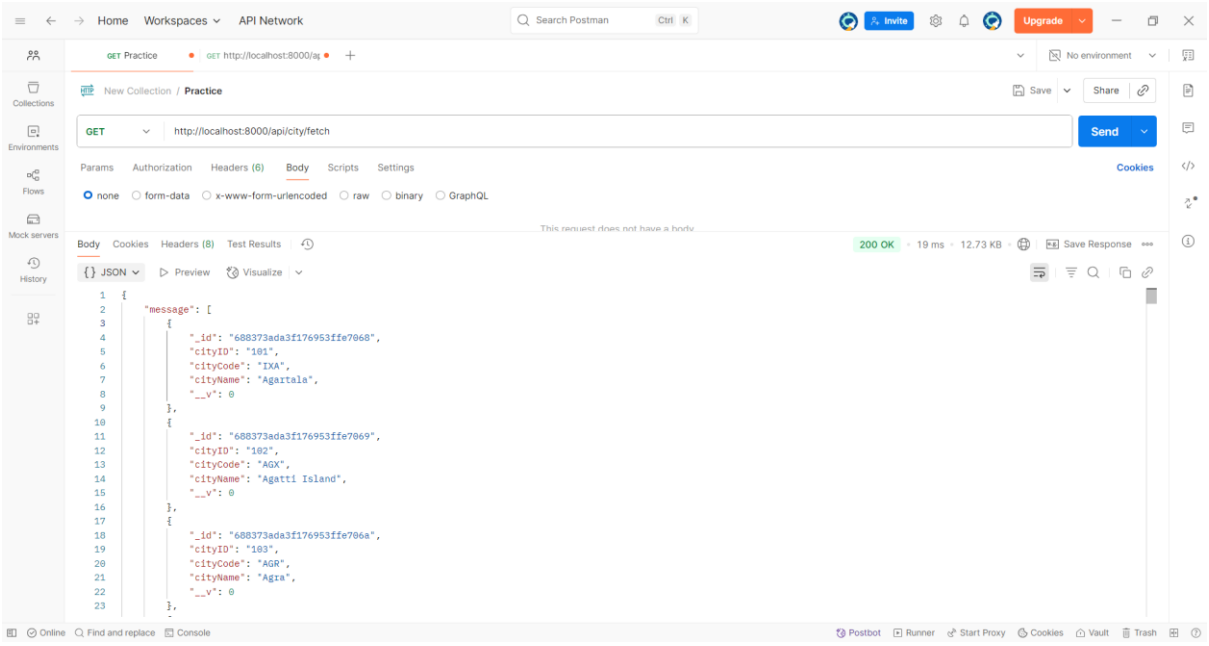


- cities:

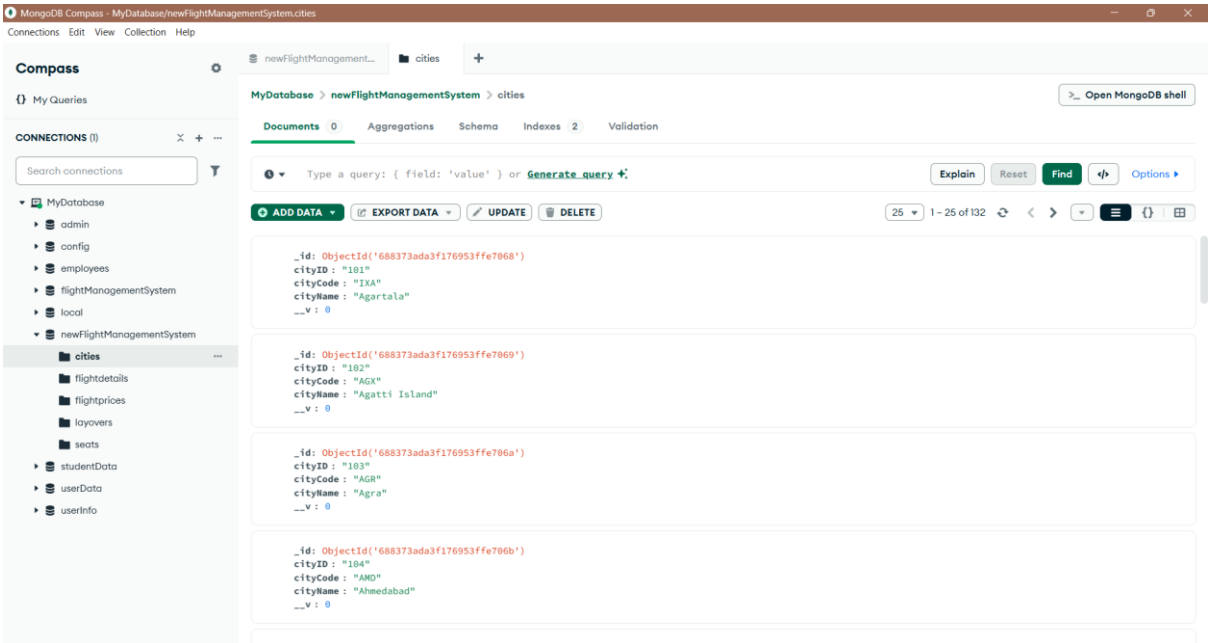
  - POST API-



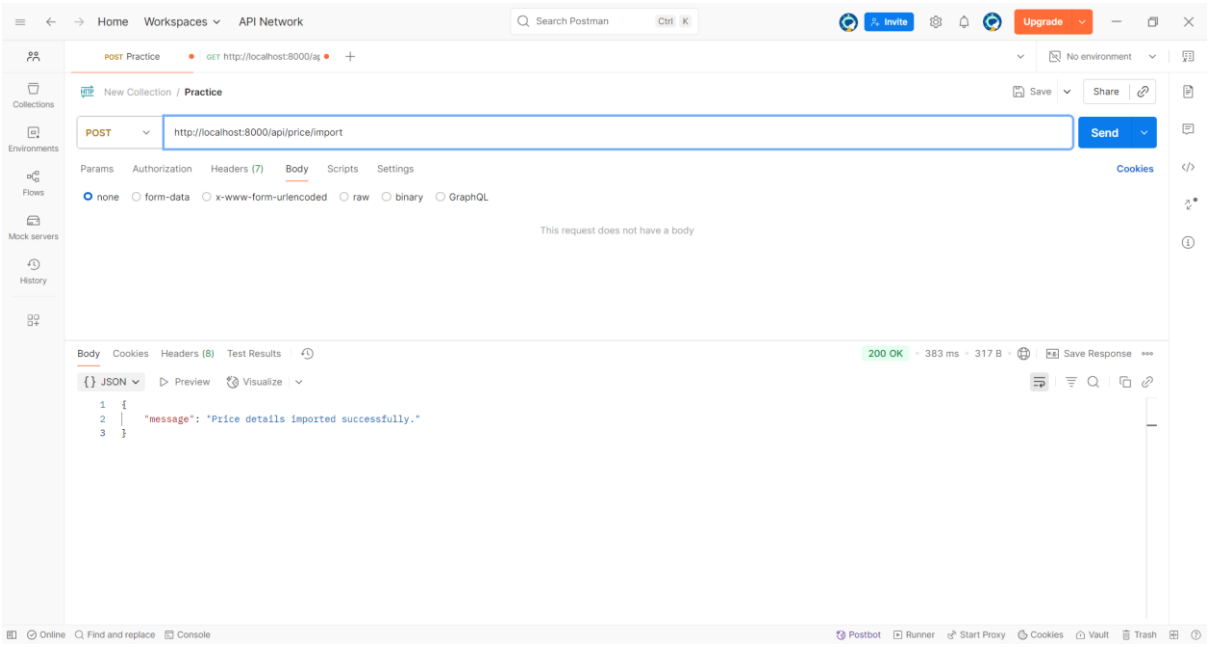
- GET API-



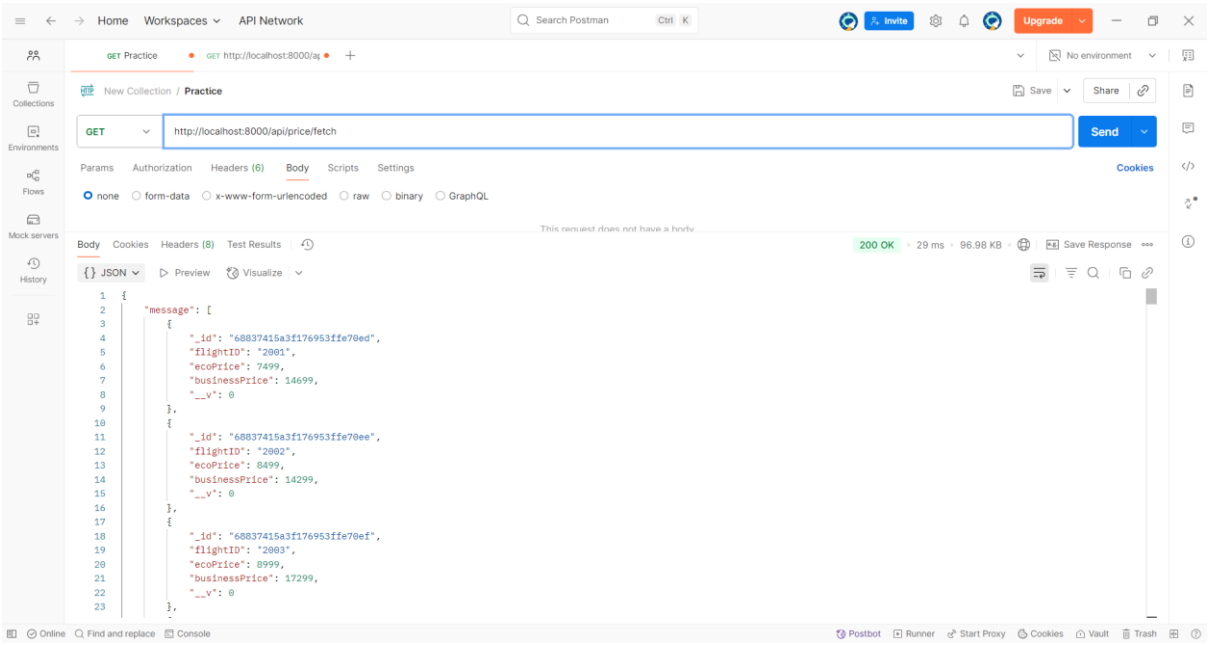
- DB Collection-



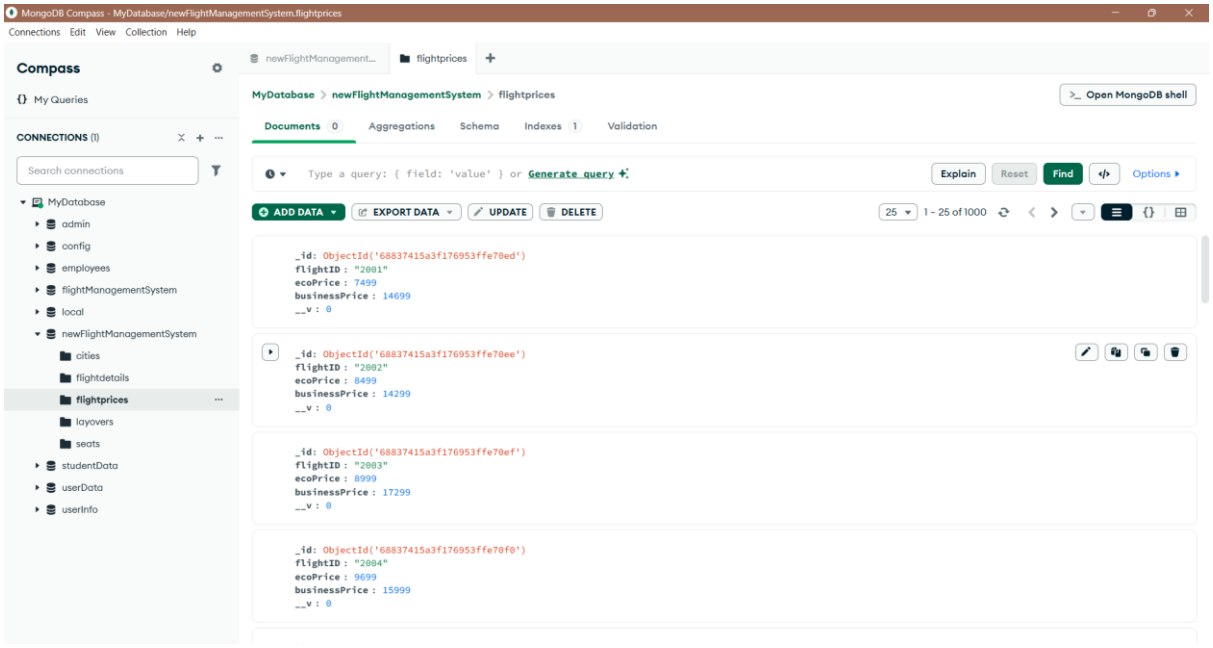
- flightPrices:
    - POST API-



    - GET API-

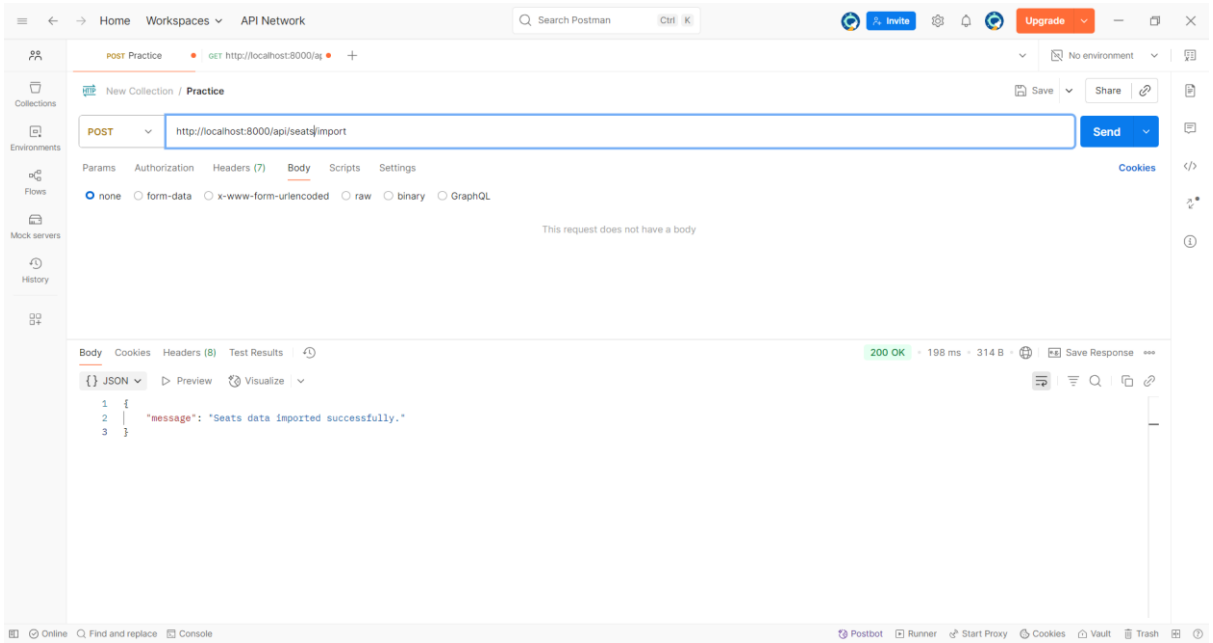


- DB Collection-

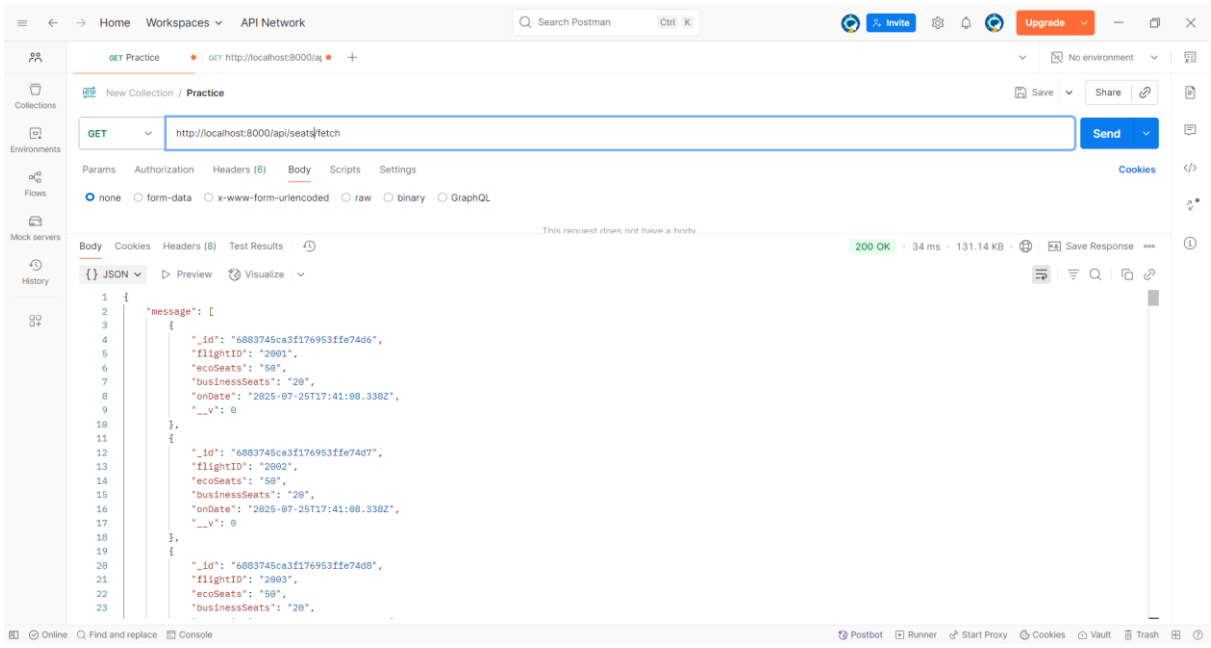


- seats:

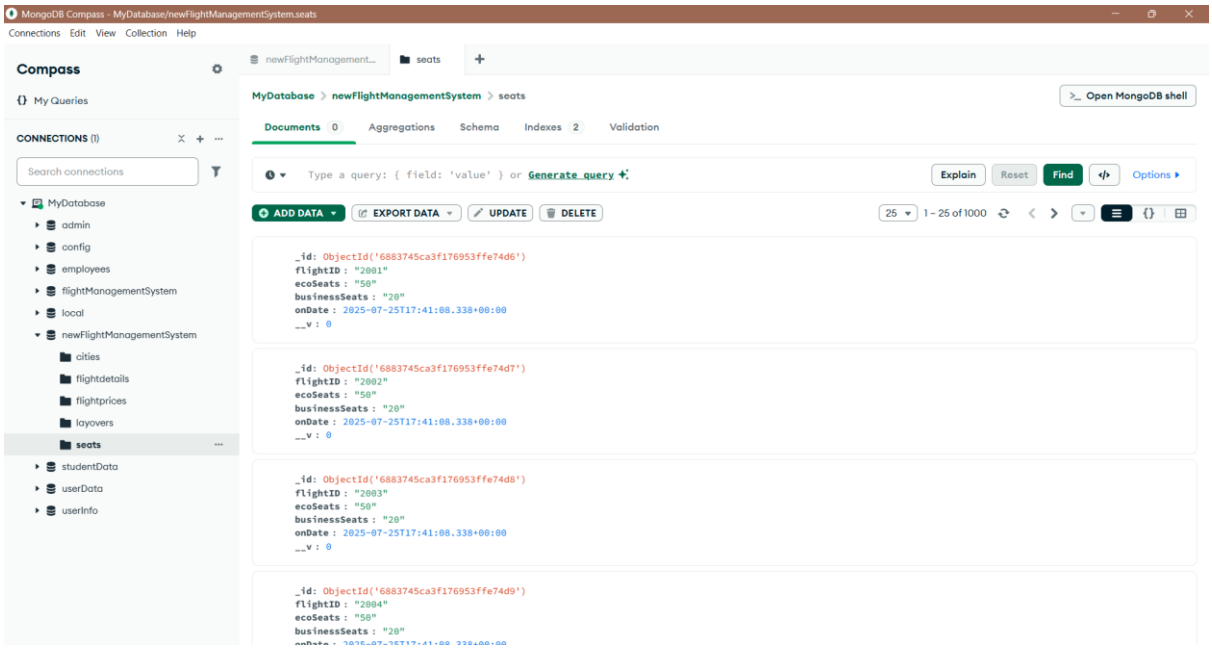
  - POST API-



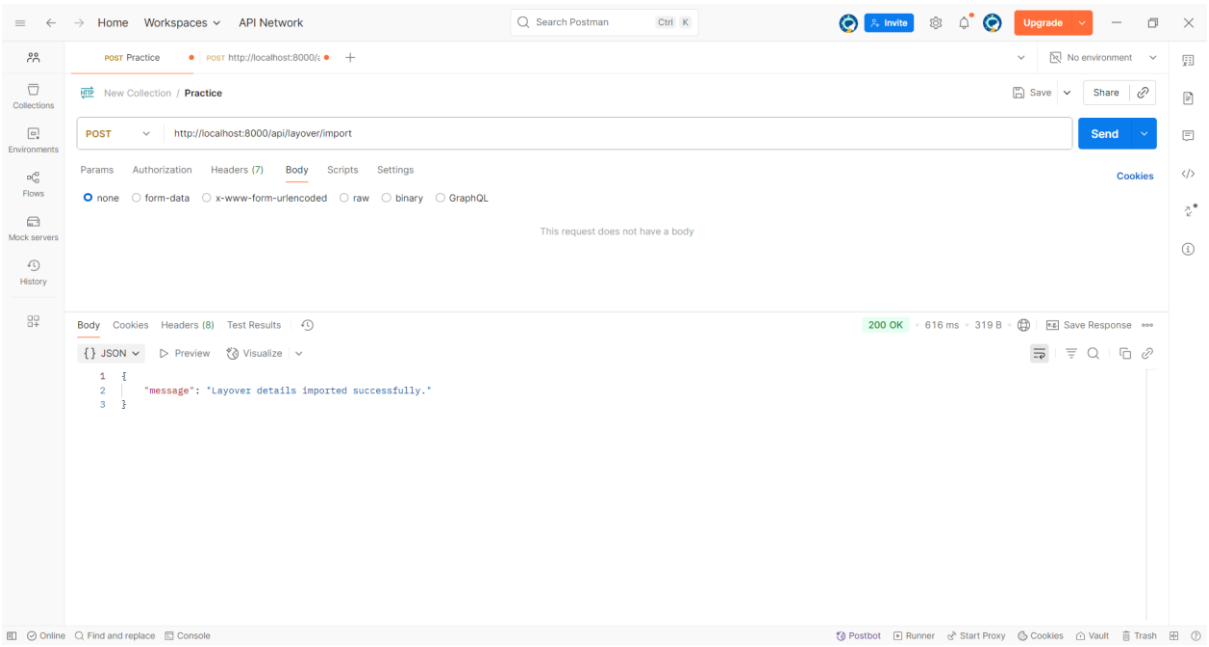
- GET API-



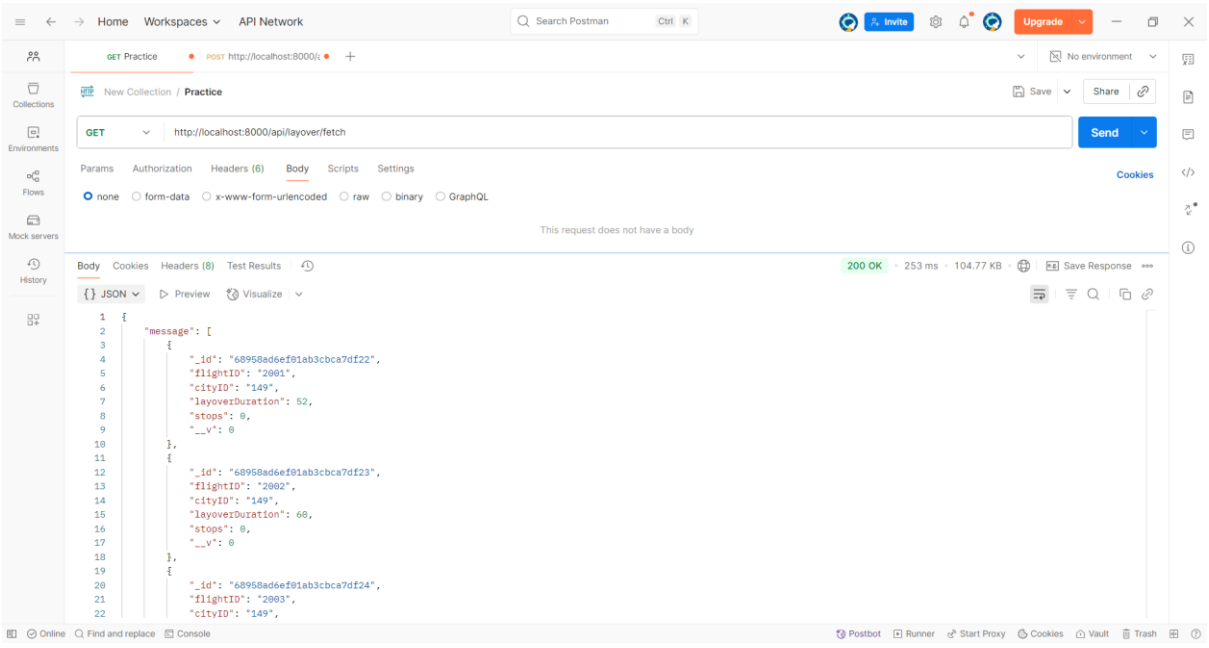
- DB Collection-



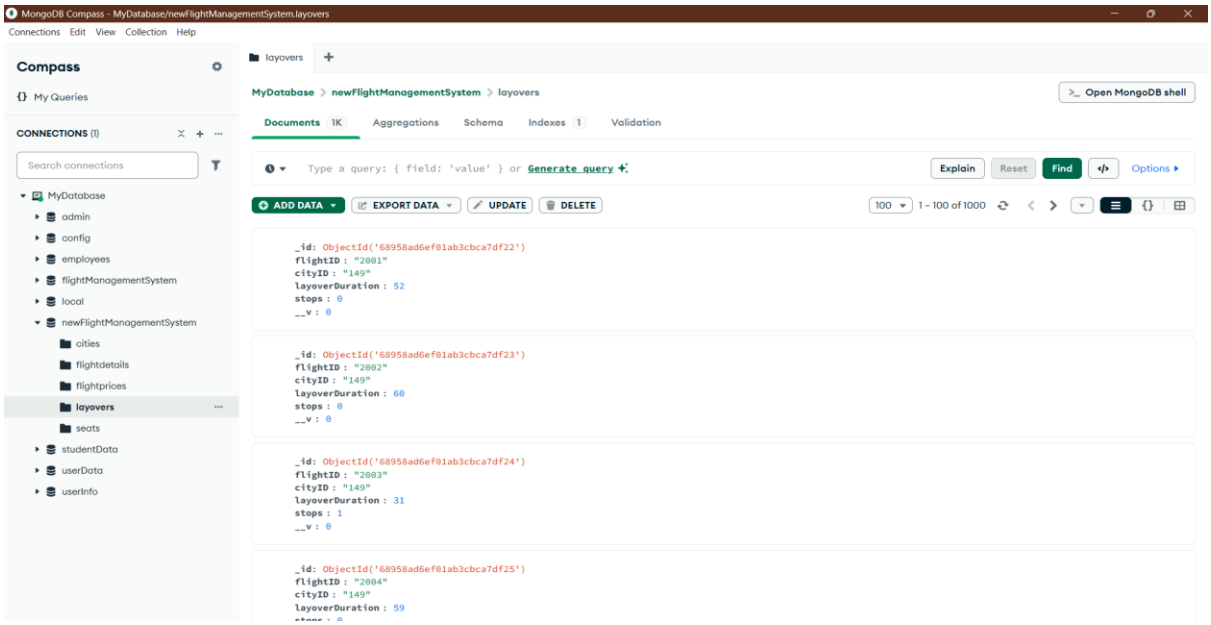
- Layover:
    - POST API-



    - GET API-



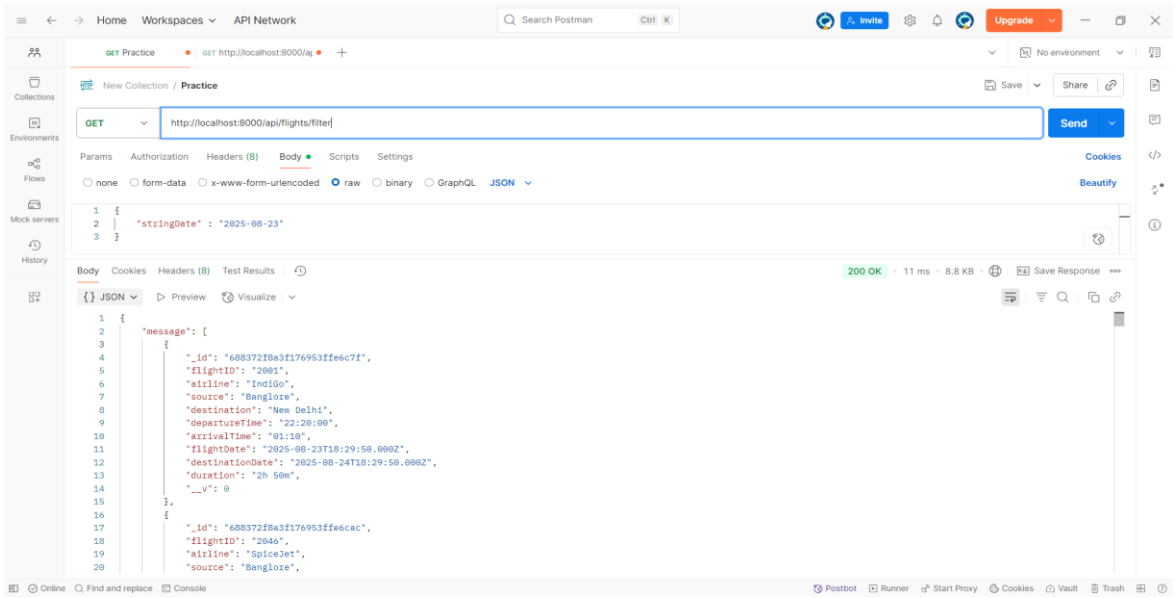
- DB Collection-



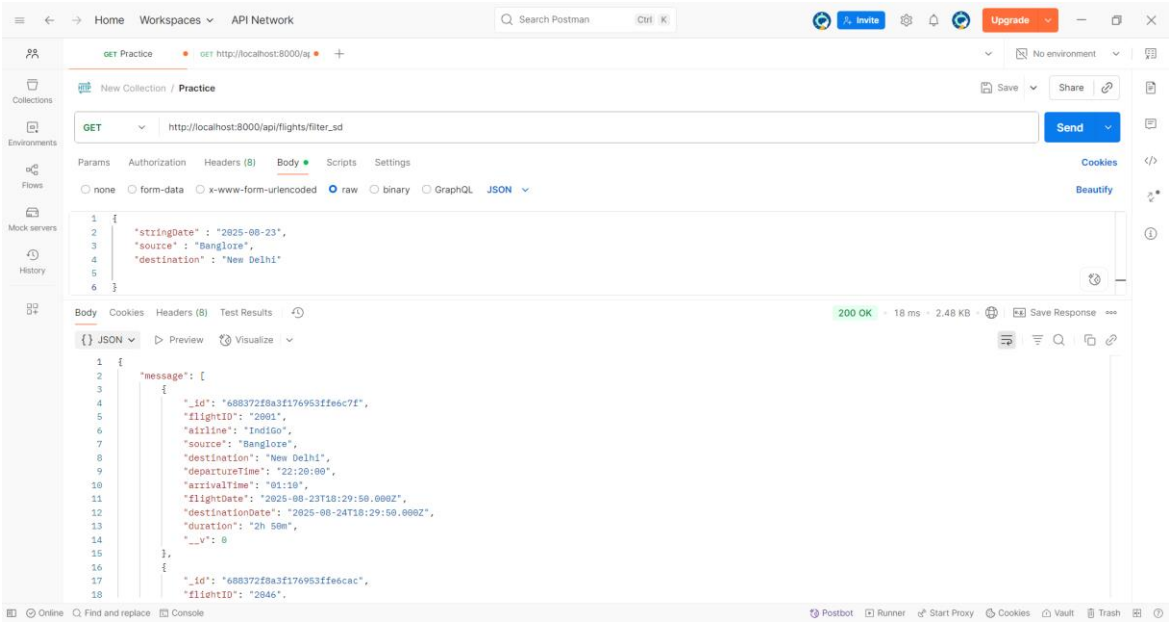
Filters

In the program, a date filter is implemented to retrieve flight data based on a specific date provided by the user. The user sends a date as a string (e.g., "2025-08-23"), which is then converted into a JavaScript Date object. The code sets the time range from 00:00:00.000 to 23:59:59.999 for that day to cover the entire date in IST. The converted date range is then used in a MongoDB query using **$gte** (greater than or equal to) and **$lte** (less than or equal to) operators to return all flights scheduled on that particular day. This ensures accurate filtering of records that match the complete date window.

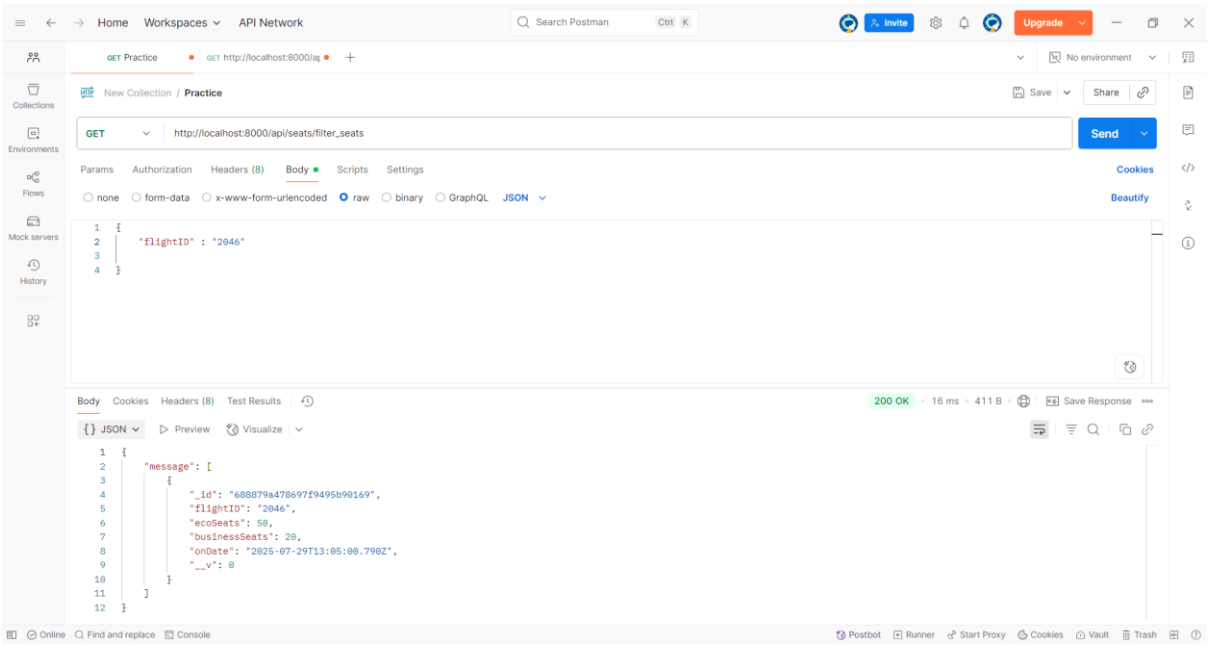
- flightDetails:
  - Date filter-



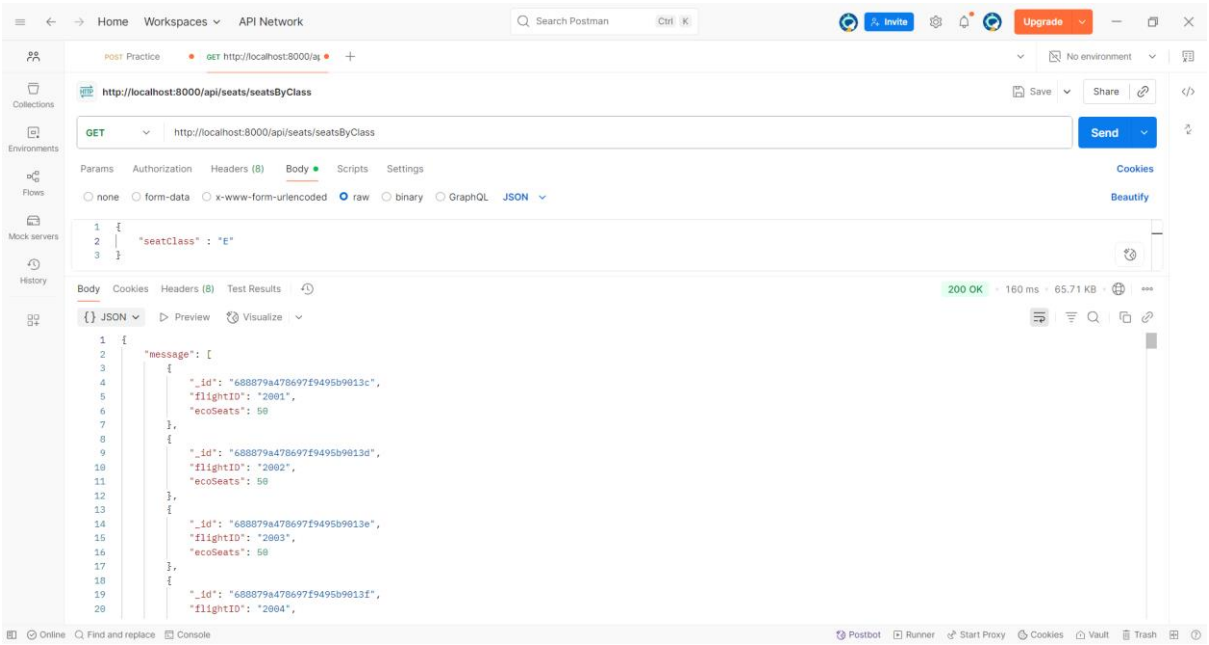
- Source and Destination filter-



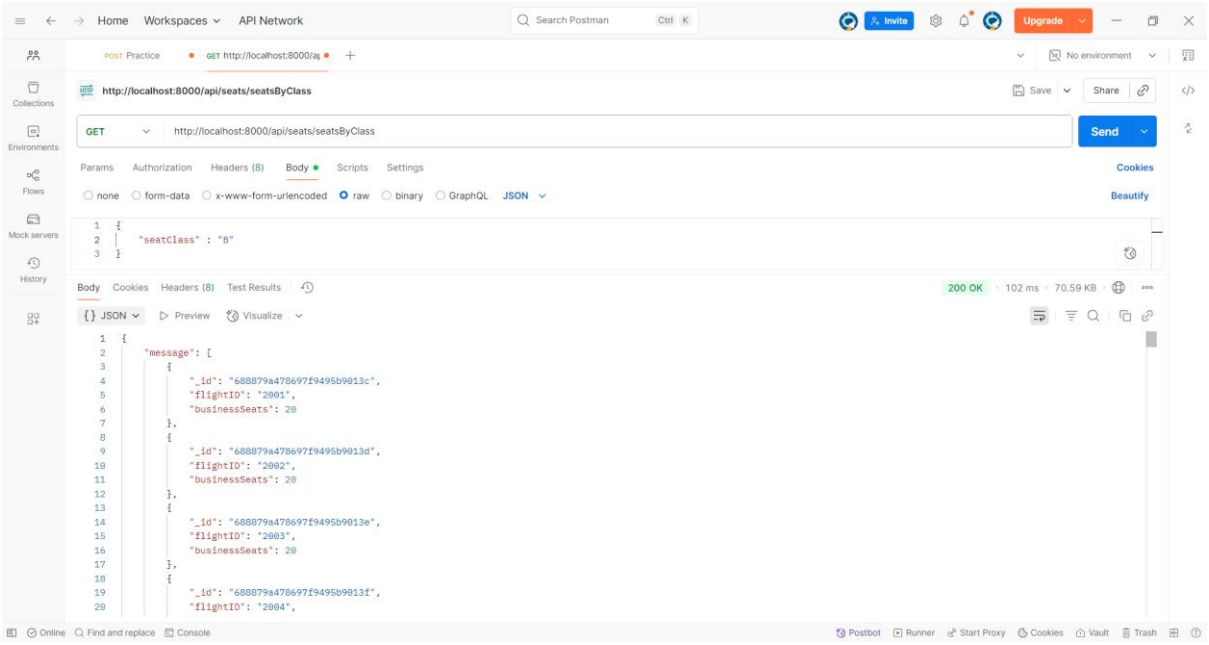
- seats:
  - Seats using flightID-



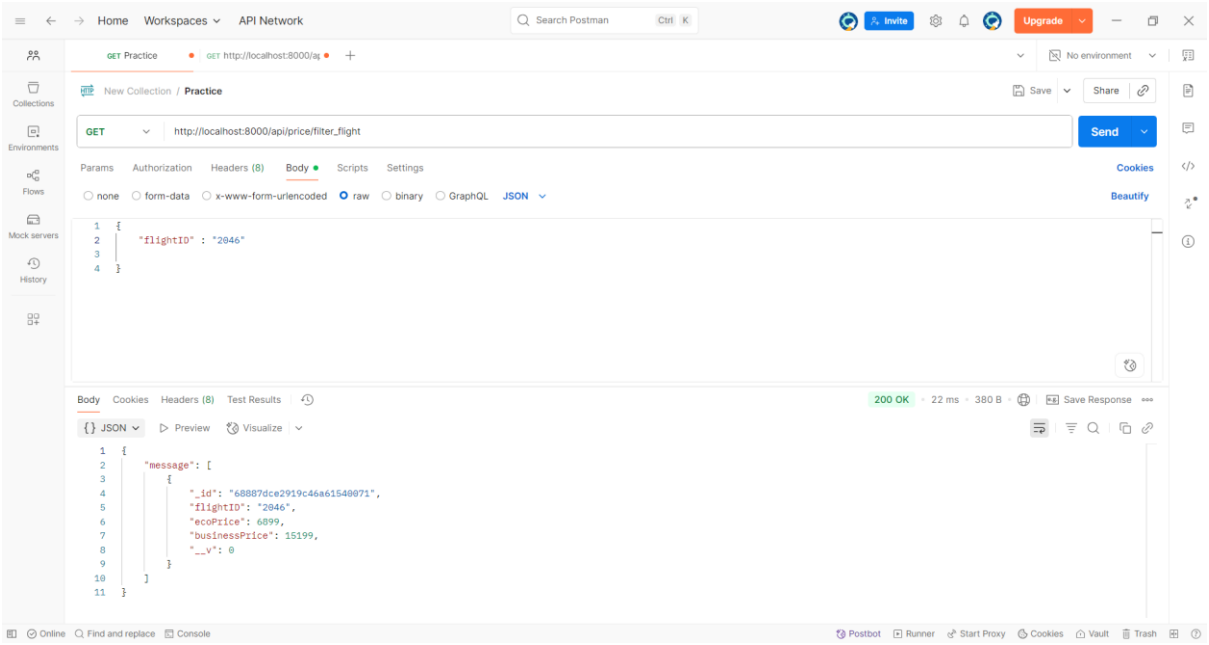
- Seats by class-
  - Economy seats



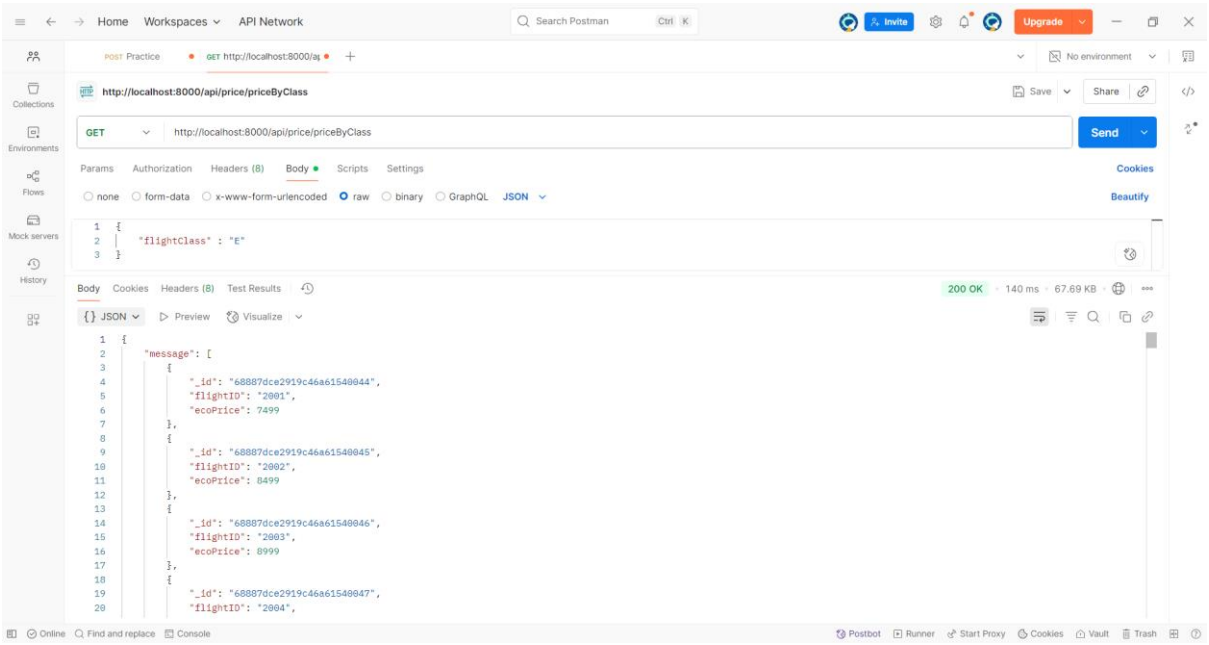
  - Business seats



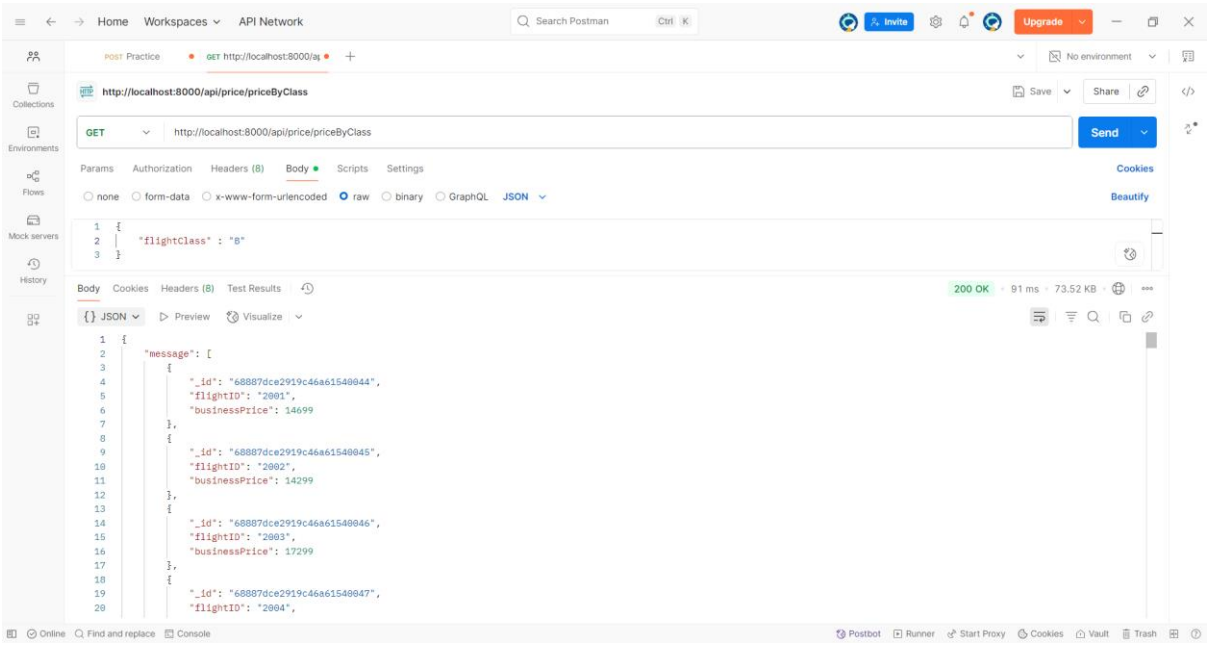
- flightPrices:
    - Price using flightID-



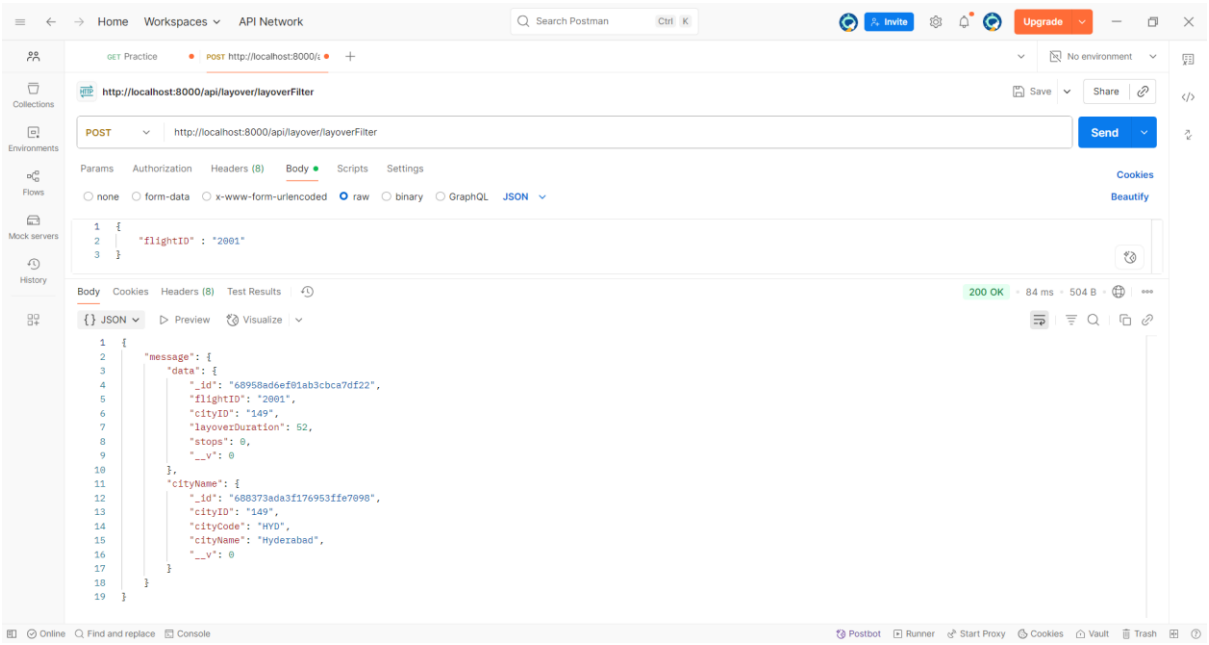
- Price by Class-
    - Economy-



- Business-



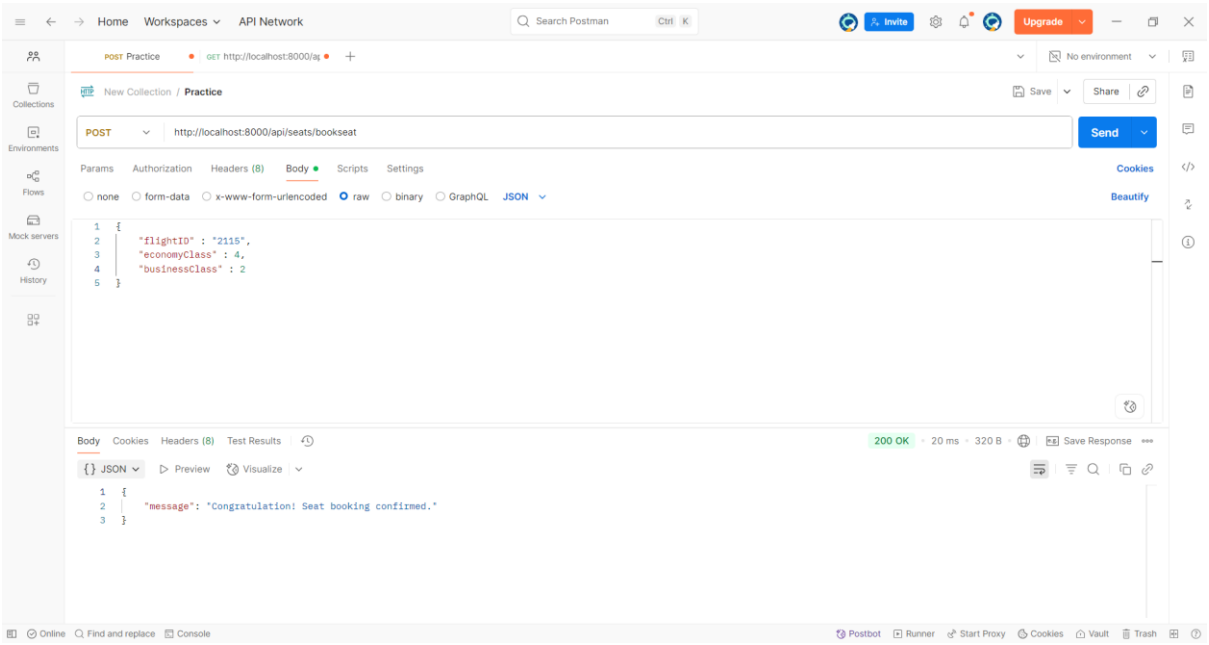
- Layover:
  - Layover by flightID



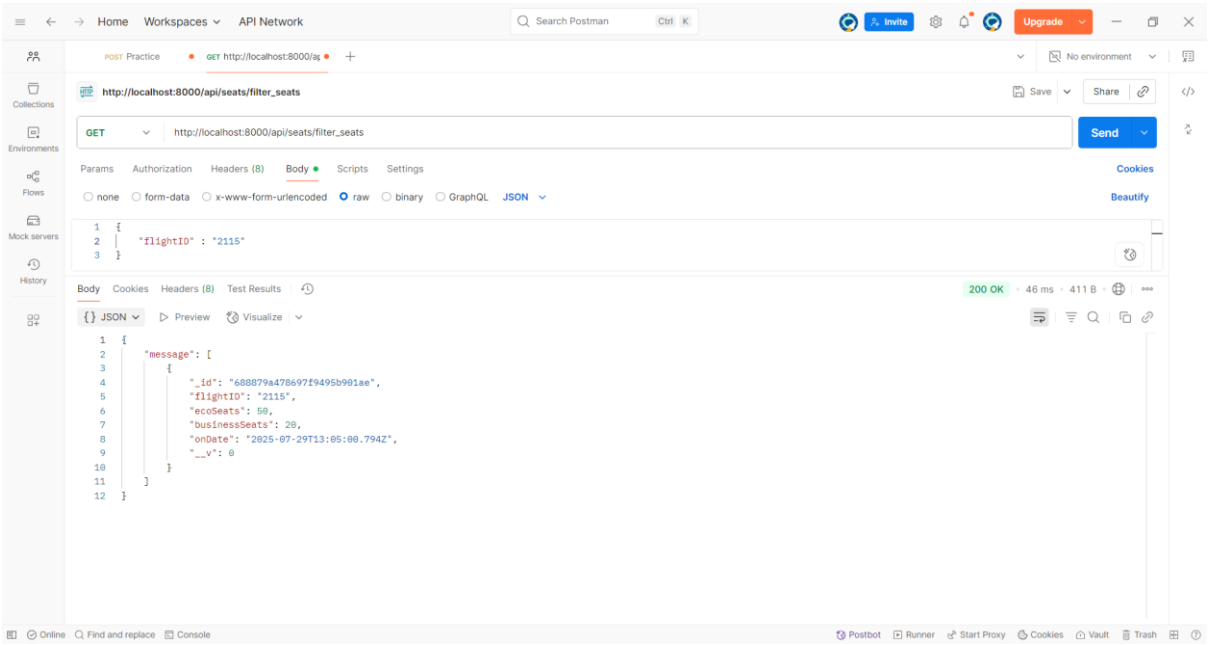
Module 4

Dummy Booking API

In the dummy booking API of the Flight Management System, the user books a flight by entering the **flightID**, selecting the class (**economy or business**), and specifying the number of **travellers**. The system checks for seat availability based on these inputs, and if the required seats are available, the booking gets **confirmed**.

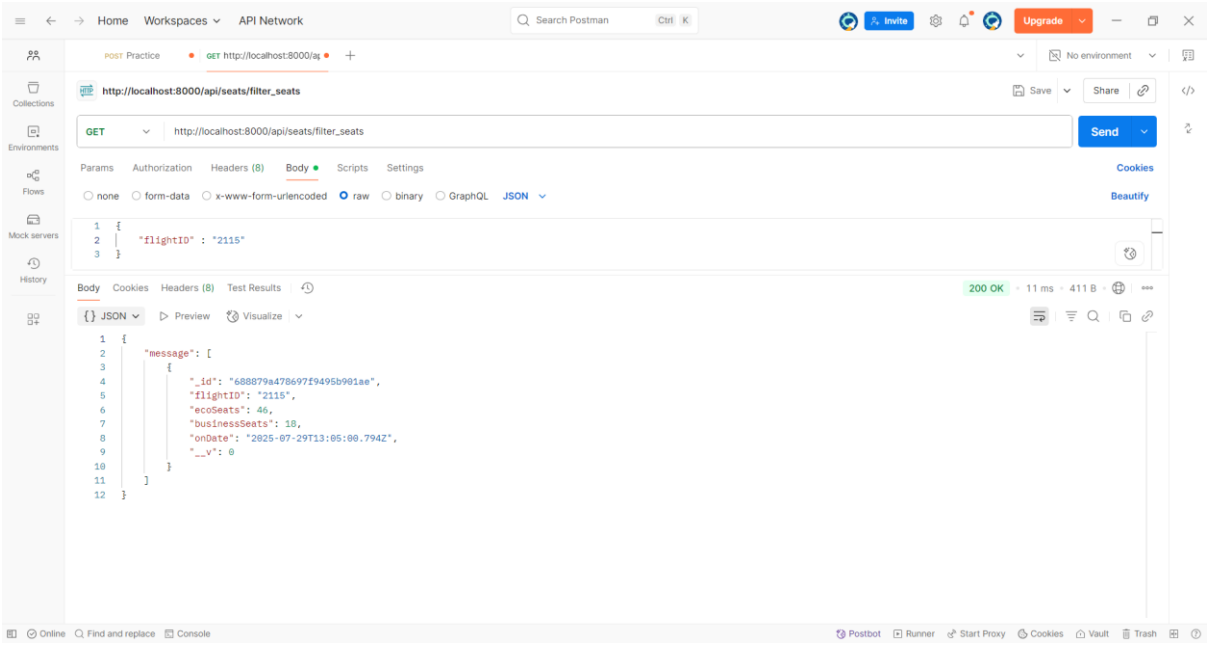


Before update



After update

After the successful transaction, the available seat count in the selected class is automatically updated in the MongoDB database, reflecting the new seat availability in real time.



Summary

The **Flight Management System** is a backend-centric project developed to manage core airline booking operations such as flight search, filtering, and reservation management. Built using **Node.js** and **Express.js**, the system emphasizes performance, scalability, and security while functioning entirely on the server side without a dedicated frontend. **MongoDB** serves as the database, containing structured collections such as flightDetails for storing flight information, aeroLayover for managing stopover details and layover durations, and users for storing registered user credentials. Instead of connecting to live APIs, the database is preloaded with dummy flight data to enable controlled testing of backend logic and query optimizations.

Security and authentication are key aspects of the system. User passwords are securely hashed using **bcrypt** before being stored in the users collection, ensuring that sensitive data remains protected. Authentication and authorization are implemented to restrict flight search and booking features to verified users only. The system also incorporates **CORS** middleware to handle cross-origin requests, ensuring compatibility for future integration with different frontend clients. Flight booking operations are tightly linked to the database, with the seat count in the flightDetails collection automatically decrementing upon successful booking. For flights involving stopovers, the aeroLayover collection stores the number of stops and layover durations, either set to default values or assigned randomly to simulate realistic travel scenarios.

The API endpoints are structured for flexibility and efficiency, supporting advanced filtering such as searching flights based on price, number of stops, or other attributes. All API testing and validation were conducted using the **Postman** application, ensuring that each route functioned as expected under different request scenarios. Postman collections were also created for organized testing of authentication routes, flight search, booking, and layover management endpoints.

By focusing purely on backend development, this project showcases skills in API design, database management, and secure authentication workflows. It demonstrates the ability to build a robust server-side architecture ready for integration with any frontend interface, while maintaining clean, maintainable, and extensible code for future enhancements such as real-time API integration, payment gateways, or notification services.

Thank You