

Flight Booking Management System

(Draft File)

Archit Jha

(Backend Intern at
Orbit Techsol)

(11/06/2025 to 11/08/2025)

INDEX

S.No.	Module	Subsections
1	Introduction	Objective Problem Statement Tech Stack & Tools
2	Login & Signup	Signup Page for New Users Login Page for Registered Users
3	Database Structure	Data collection Data pre-processing Data insertion Data updation (Creation & Retrieval using Kaggle)
4	Data Filtration	Folder structure GET/POST Method Filter on flight data
5	Submission of Data	Dummy booking API Updation of Seat Availability Email Confirmation on booking
6	Backend & frontend Integration (Extra)	Backend-Frontend Connectivity Validation & Verification

Objective

The **Flight Management System** is a backend-only project that allows users to securely sign up, log in, and search for flights 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 — all without a frontend interface.

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

a. 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

b. 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

c. 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

- Endpoint: POST /signup
- 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

- Endpoint: POST /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.
-

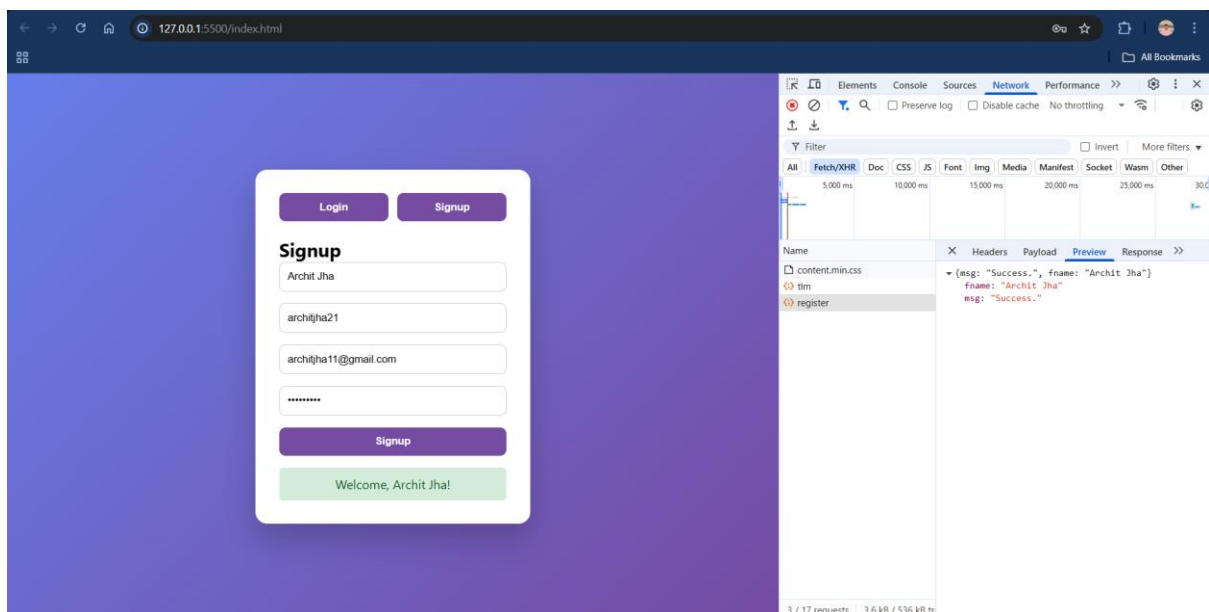
Access Control Using JWT

- 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.
-

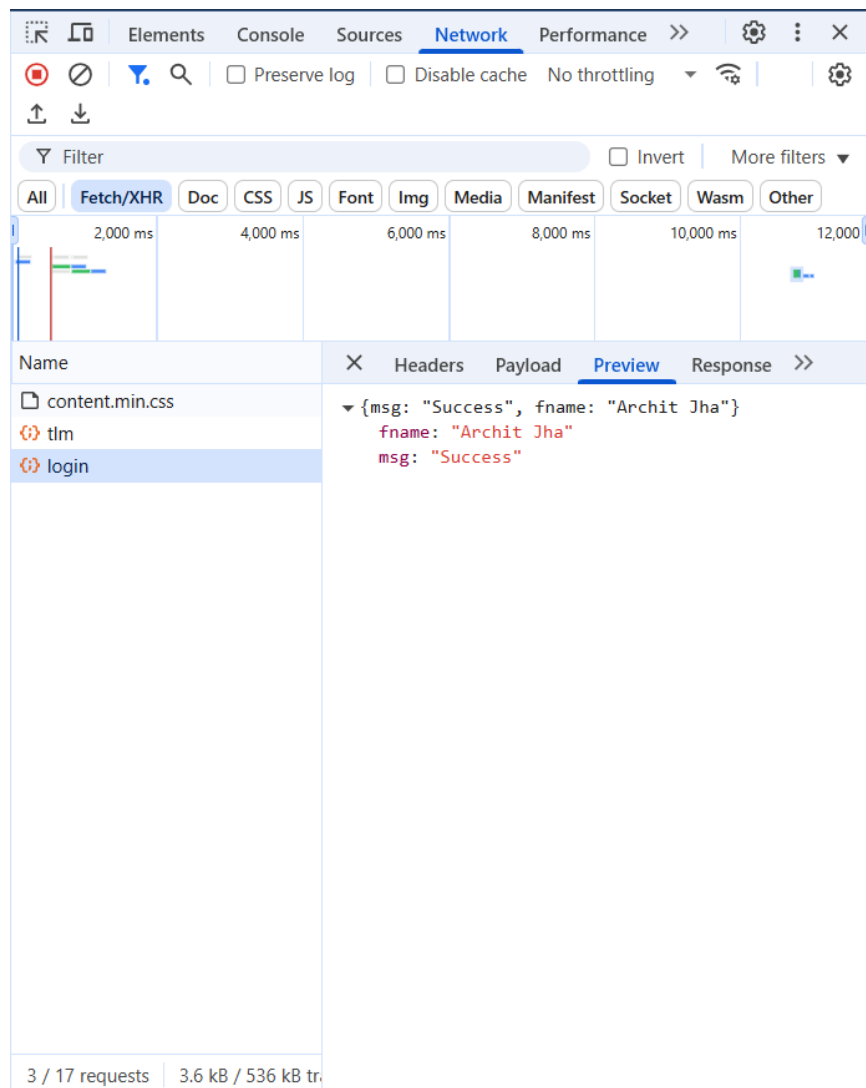
Testing in Postman

- 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



- 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.nLLyhy3lDClYdt0auRMjETT vz8Fz9V9tqBurZg0iJs2u1.',
    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:

FileHomeInsertDrawPage LayoutFormulasDataReviewViewHelpAcrobat

Calibri

11

A⁺

A⁻

Wrap Text

General

Conditional Formatting

Format as Table

Cell Styles

Insert

Delete

Format

Σ AutoSum

Sort & Filter

Find & Select

Add-ins

Create a PDF

Clipboard

Font

Alignment

Number

Styles

Cells

Editing

Add-ins

Adobe Acrobat

FlightData

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

FlightDate

Airline

Origin

Dest

Cancelled

Diverted

CRSDepT

DepTime

DepDelay

DepDelay

ArrTime

ArrDelay

AirTime

CRSElapse

ActualElag

Distance

Year

Quarter

Month

DayofMor

DayOfWee

Marketing

Operated

DOT_ID

IATA

Cc

1

24-01-2018

Endeavor

ABY

ATL

FALSE

FALSE

1202

1157

0

-5

1258

0

38

62

61

145

2018

1

1

24

3

DL

DL_CODES

19790

DL

4

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

5

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

6

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

7

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

8

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

9

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

10

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

11

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

12

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

13

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

14

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

15

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

16

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

17

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

18

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

19

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

20

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

21

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

22

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

23

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

24

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

25

17-01-2018

Endeavor

ATL

ABY

TRUE

FALSE

1037

26

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

27

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

28

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

29

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

30

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

Ready

Accessibility: Unavailable

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), and **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	Wednesday	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	Wednesday	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	Wednesday	2025-09-13	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	Wednesday	2025-09-01	30

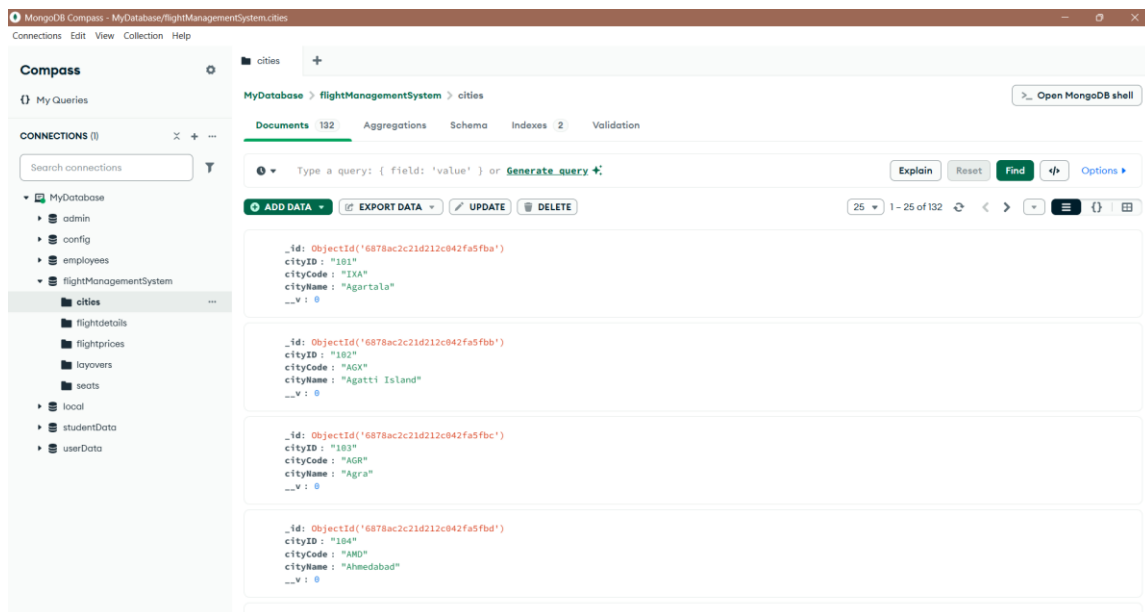
- Collections in MongoDB

Collection Name	Storage size	Documents	Avg. document size	Indexes	Total index size
cities	24.58 kB	132	87.00 B	2	73.73 kB
flightdetails	77.82 kB	1 K	339.00 B	2	106.50 kB
flightprices	36.86 kB	1 K	83.00 B	1	24.58 kB
layovers	40.96 kB	1 K	87.00 B	1	32.77 kB
seats	32.77 kB	1 K	105.00 B	1	24.58 kB

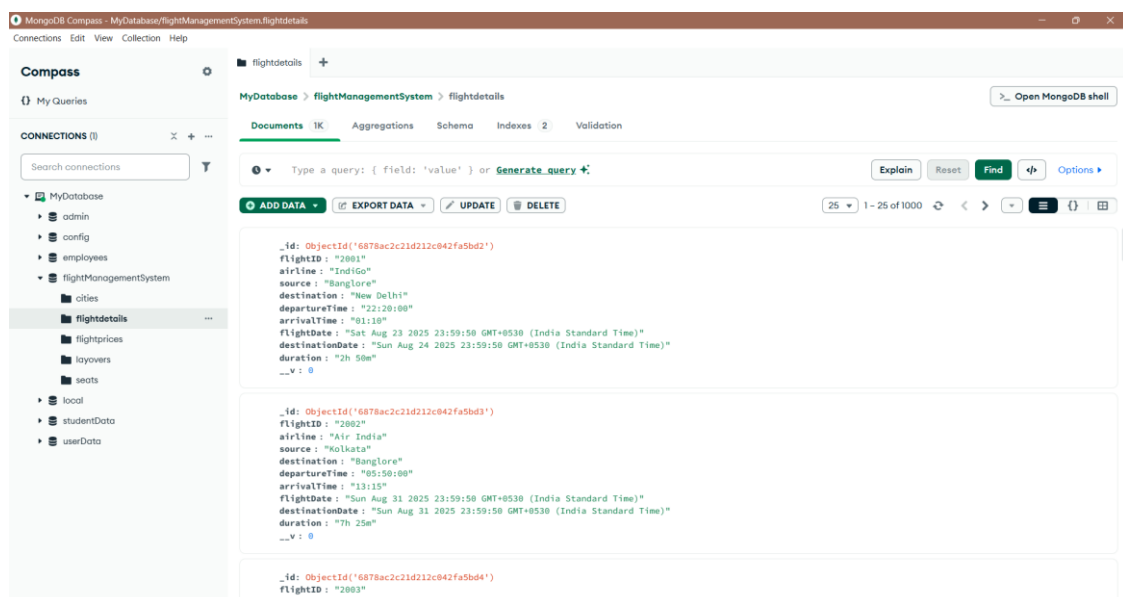
Data Insertion

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—**cities**, **flightdetails**, **flightprice**, **seats**, and **layovers**—to maintain data integrity and validation. The data was then inserted into MongoDB using the **insertMany()** method for each collection. The **code used for importing and inserting the data into the database is available on GitHub** in the [Flight Management System](#) repository for reference and reuse.

- Cities collection



- flightDetails collection



- flightprices collection

MongoDB Compass - MyDatabase/flightManagementSystem.flightprices

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (1)

Search connections

MyDatabase

- admin
- config
- employees
- flightManagementSystem
 - cities
 - flightdetails
 - flightprices**
 - layovers
 - seats
- local
- studentData
- userData

flightprices

MyDatabase > flightManagementSystem > flightprices

Documents 1K Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 25 of 1000

```

_id: ObjectId('6878da2a081773e884db960a')
flightID: "2001"
ecoPrice: 7999
businessPrice: 17699
__v: 0

_id: ObjectId('6878da2a081773e884db960b')
flightID: "2002"
ecoPrice: 8599
businessPrice: 16899
__v: 0

_id: ObjectId('6878da2a081773e884db960c')
flightID: "2003"
ecoPrice: 9799
businessPrice: 16699
__v: 0

_id: ObjectId('6878da2a081773e884db960d')
flightID: "2004"
ecoPrice: 9199
businessPrice: 14699
__v: 0

```

- layovers collection

MongoDB Compass - MyDatabase/flightManagementSystem.layovers

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (1)

Search connections

MyDatabase

- admin
- config
- employees
- flightManagementSystem
 - cities
 - flightdetails
 - flightprices
 - layovers**
 - seats
- local
- studentData
- userData

layovers

MyDatabase > flightManagementSystem > layovers

Documents 1K Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 25 of 1000

```

_id: ObjectId('6878c1a734d98e32f98cf86')
flightID: "2001"
cityID: "101"
layoverDuration: 42
__v: 0

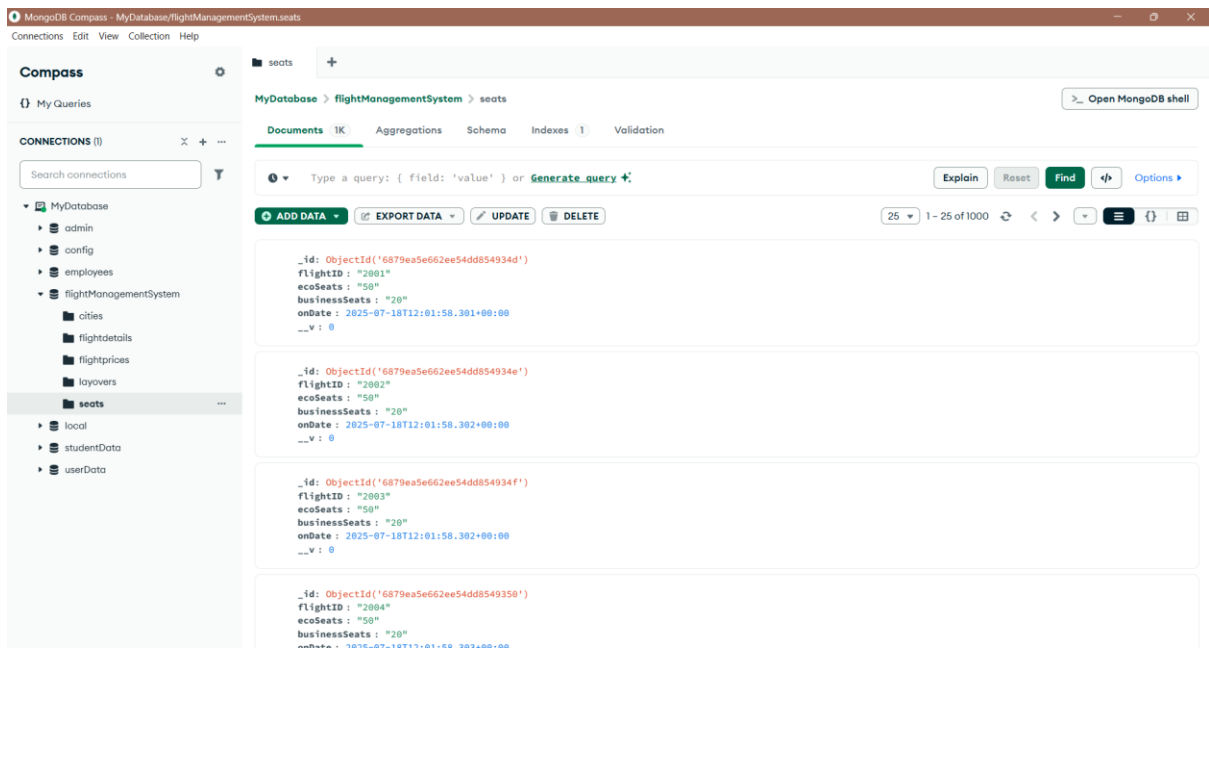
_id: ObjectId('6878c1a734d98e32f98cf87')
flightID: "2002"
cityID: "102"
layoverDuration: 48
__v: 0

_id: ObjectId('6878c1a734d98e32f98cf88')
flightID: "2003"
cityID: "103"
layoverDuration: 38
__v: 0

_id: ObjectId('6878c1a734d98e32f98cf89')
flightID: "2004"
cityID: "104"
layoverDuration: 48
__v: 0

```

- seats collection



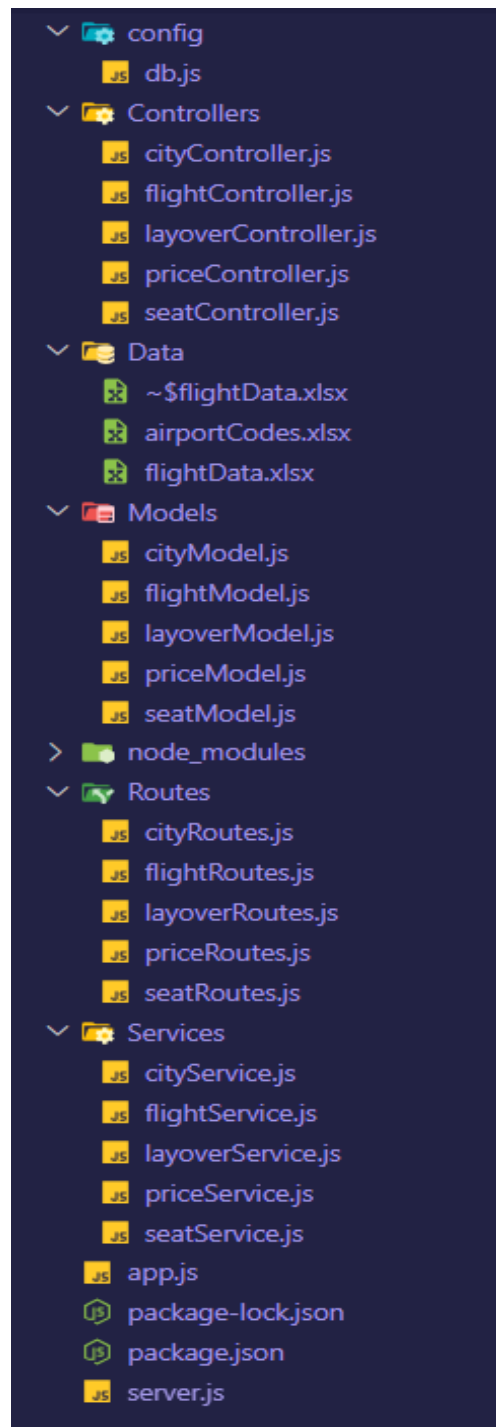
Data Updation

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:

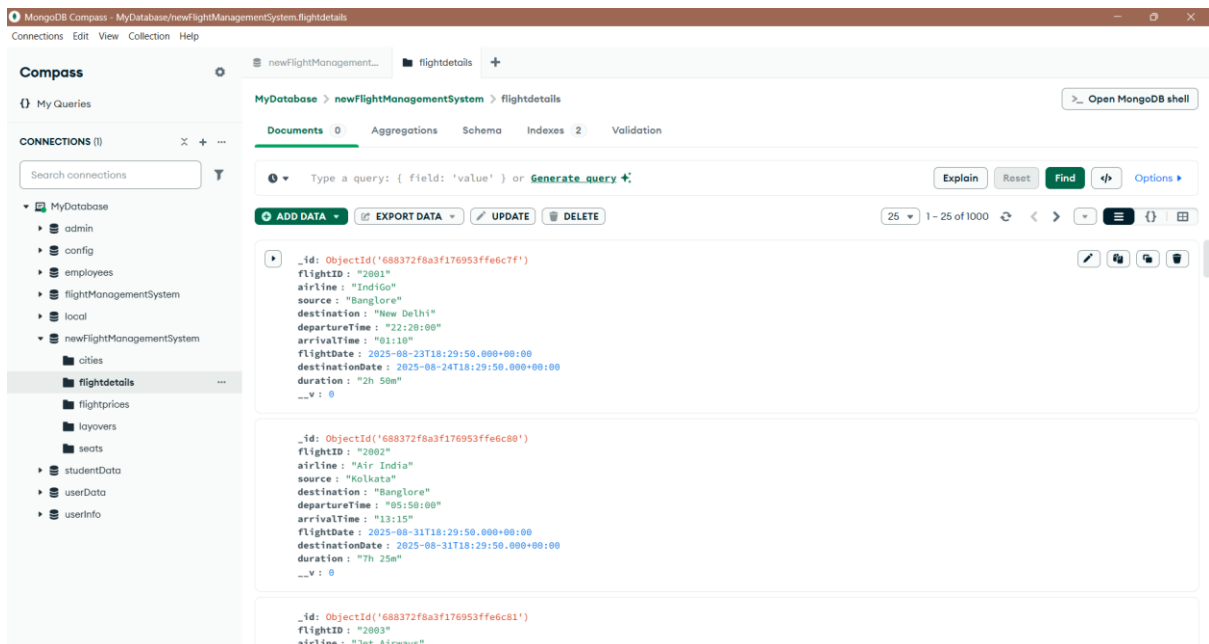
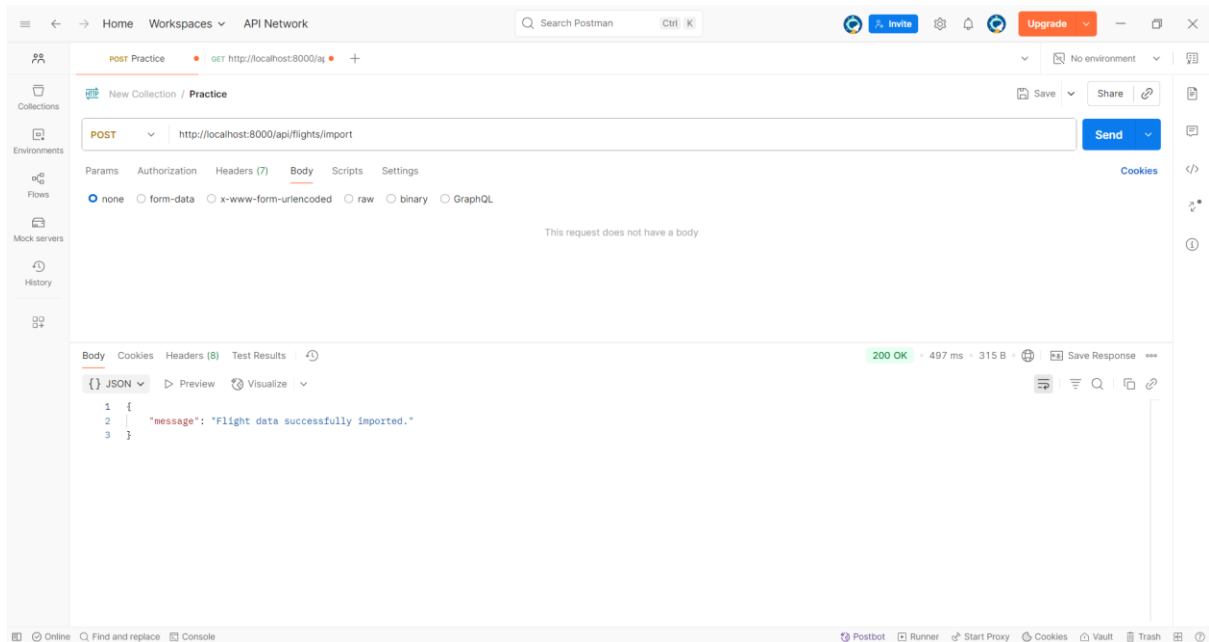
I have organized my Node.js project by creating a clean folder structure to improve readability and maintainability. The main entry point is `app.js`, and I have separated the code into folders like `routes` for defining API endpoints, `controllers` for handling request-response logic, and `services` for business logic. I used the `models` folder for database schemas and created a `config` folder to manage the database connection setup. This structured approach has made my code more modular and easier to scale.



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, such as flight details, cities, layovers, prices, and seat information. 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.

- POST Method:
 - Flight details



○ City details

The screenshot shows the Postman application interface. At the top, the URL bar indicates a POST request to `http://localhost:8000/api/city/import`. The request body is empty, with a note stating "This request does not have a body". The response is displayed in the bottom pane, showing a 200 OK status with a response time of 30.87 s and a body size of 313 B. The response body is a JSON object: `{ "message": "Successfully city data imported." }`. The left sidebar shows the "Practice" collection and the "Environments" tab.

The screenshot shows the MongoDB Compass application interface. The left sidebar displays the "Connections" list, including "MyDatabase" and "newFlightManagementSystem". The main pane shows the "cities" collection in the "newFlightManagementSystem" database. The "Documents" tab is active, displaying a list of documents. The first document is:

```
{
  "_id": ObjectId('688373ada3f176953ffe7068'),
  "cityID": "101",
  "cityCode": "IXA",
  "cityName": "Agartala",
  "__v": 0
}
```

The second document is:

```
{
  "_id": ObjectId('688373ada3f176953ffe7069'),
  "cityID": "102",
  "cityCode": "AGX",
  "cityName": "Agatti Island",
  "__v": 0
}
```

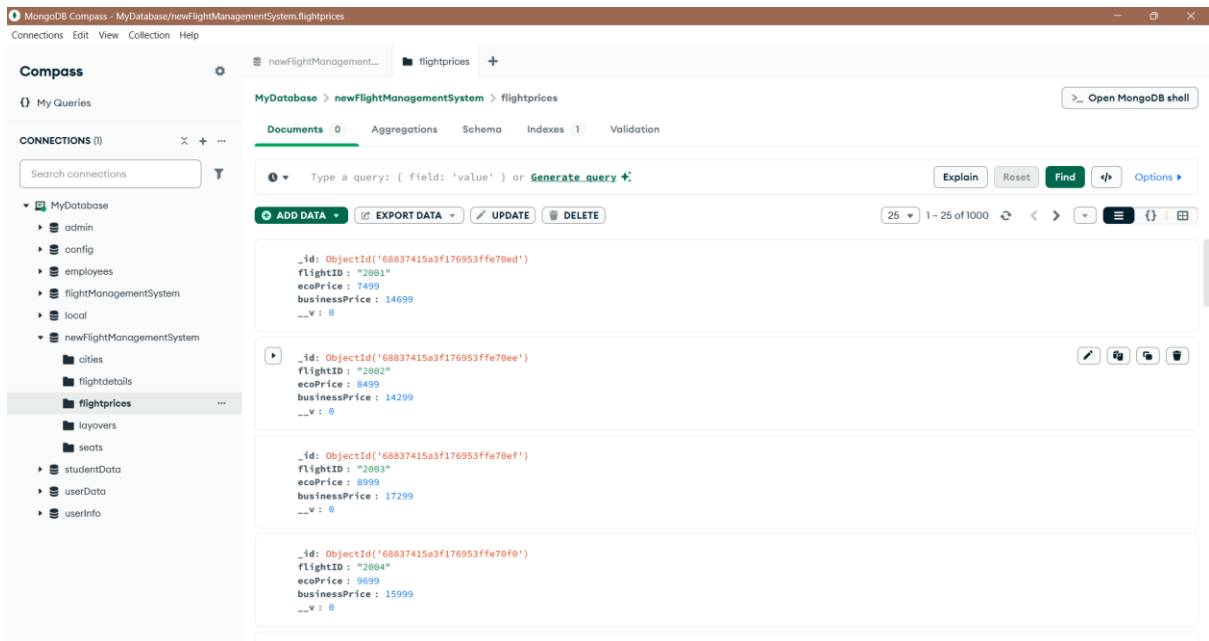
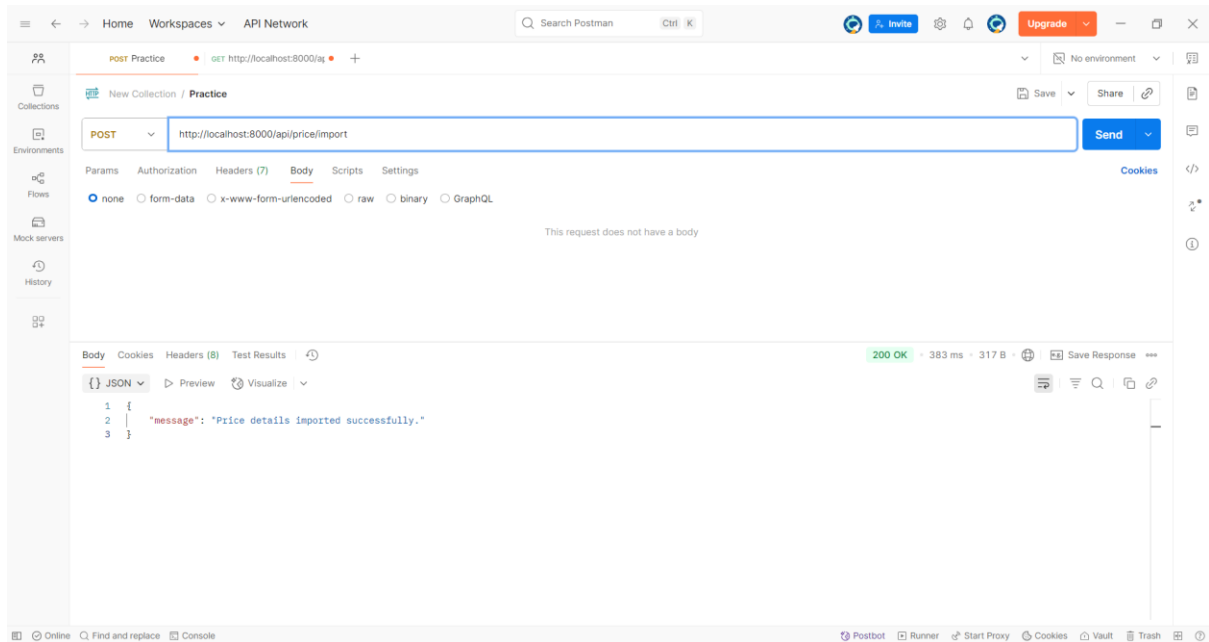
The third document is:

```
{
  "_id": ObjectId('688373ada3f176953ffe706a'),
  "cityID": "103",
  "cityCode": "AGR",
  "cityName": "Agra",
  "__v": 0
}
```

The fourth document is:

```
{
  "_id": ObjectId('688373ada3f176953ffe706b'),
  "cityID": "104",
  "cityCode": "AMD",
  "cityName": "Ahmedabad",
  "__v": 0
}
```


○ Price details



○ Seat details

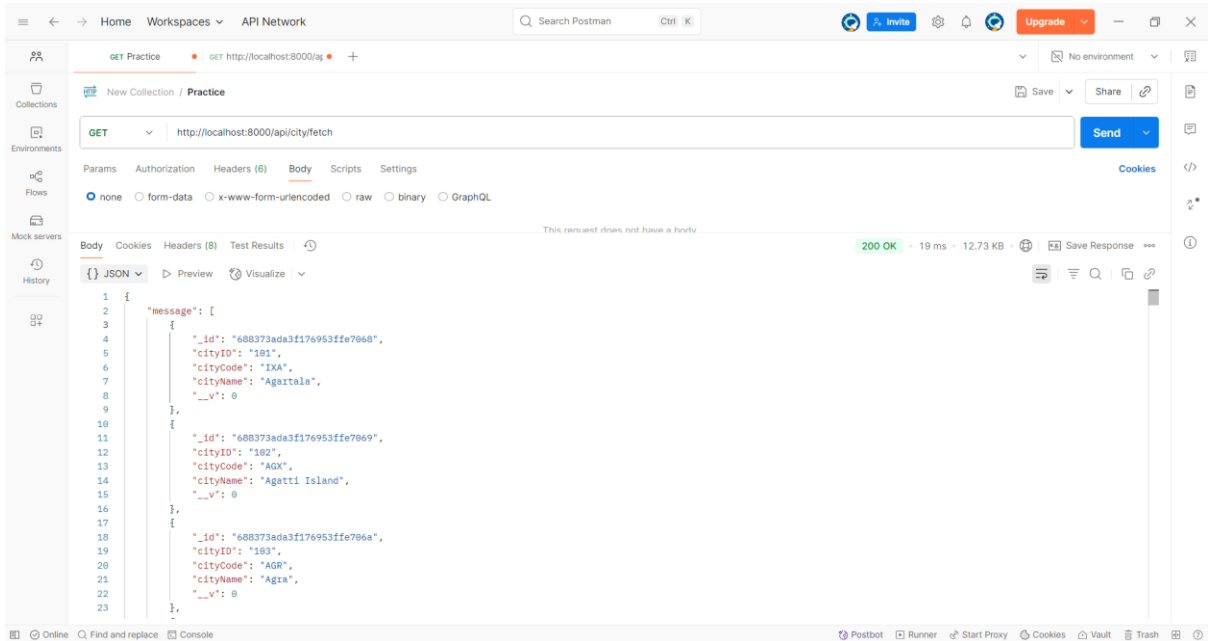
The screenshot shows the Postman application interface. At the top, the URL bar displays `http://localhost:8000/api/seats/import` with a `POST` method selected. The left sidebar shows the 'Collections' tab with a new collection named 'Practice'. The main area shows the request details, including headers, body, and scripts. The response is displayed in the bottom panel, showing a `200 OK` status with a response time of 198 ms and a body of `{ "message": "Seats data imported successfully." }`.

```
{
  "message": "Seats data imported successfully."
}
```

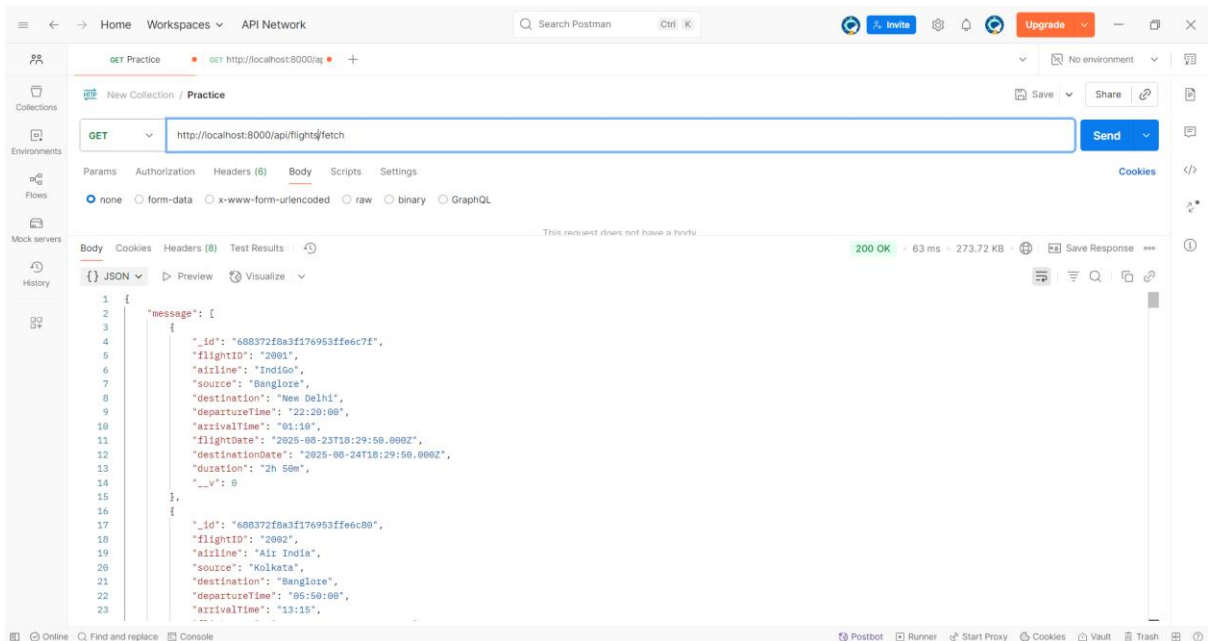
The screenshot shows the MongoDB Compass application interface. The left sidebar displays the 'Connections' tab with a list of databases, including 'MyDatabase'. The main area shows the 'Documents' tab for the 'seats' collection in the 'newFlightManagementSystem' database. The interface displays a list of documents, each containing fields like `_id`, `flightID`, `ecoSeats`, `businessSeats`, and `onDate`.

```
{ "_id": ObjectId('6883745ca3f176953ffe74d6'), "flightID": "2001", "ecoSeats": "50", "businessSeats": "20", "onDate": "2025-07-25T17:41:00.338+00:00", ... }
```

- GET Method
 - City details



- Flight details



○ Price details

GET Practice GET http://localhost:8000/api/price/fetch

Params Authorization Headers (6) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Body Cookies Headers (8) Test Results

200 OK · 29 ms · 96.98 KB

```
1 {
2   "message": [
3     {
4       "_id": "68837415a3f176953ffe78ed",
5       "flightID": "2001",
6       "ecoPrice": 7499,
7       "businessPrice": 14699,
8       "_v": 0
9     },
10    {
11      "_id": "68837415a3f176953ffe78ee",
12      "flightID": "2002",
13      "ecoPrice": 8499,
14      "businessPrice": 14299,
15      "_v": 0
16    },
17    {
18      "_id": "68837415a3f176953ffe78ef",
19      "flightID": "2003",
20      "ecoPrice": 8999,
21      "businessPrice": 17299,
22      "_v": 0
23    }
24  ]
25 }
```

○ Seat details

GET Practice GET http://localhost:8000/api/seats/fetch

Params Authorization Headers (6) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Body Cookies Headers (8) Test Results

200 OK · 34 ms · 131.14 KB

```
1 {
2   "message": [
3     {
4       "_id": "6883745ca3f176953ffe74d6",
5       "flightID": "2001",
6       "ecoSeats": "50",
7       "businessSeats": "20",
8       "onDate": "2025-07-25T17:41:00.338Z",
9       "_v": 0
10    },
11    {
12      "_id": "6883745ca3f176953ffe74d7",
13      "flightID": "2002",
14      "ecoSeats": "50",
15      "businessSeats": "20",
16      "onDate": "2025-07-25T17:41:00.338Z",
17      "_v": 0
18    },
19    {
20      "_id": "6883745ca3f176953ffe74d8",
21      "flightID": "2003",
22      "ecoSeats": "50",
23      "businessSeats": "20",
24      "_v": 0
25    }
26  ]
27 }
```

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.

