FastAPI Backend with MongoDB Integration and RESTful API using Swagger and Postman

### 1. Introduction

# This project implements a **FastAPI-based RESTful API** that integrates with **MongoDB** for persistent storage and consumes data from a **3rd-party weather API** (OpenWeatherMap). The system emphasizes modular design, validation, error handling, and testing via Swagger UI and Postman.

### 2. Technologies and Libraries Used

* FastAPI: Main framework used to build the backend RESTful API.

Why it was chosen: Lightweight, modern, and asynchronous framework that supports automatic Swagger UI generation.

Role: Defines the API routes, handles HTTP methods (GET, POST, PUT, DELETE), manages requests/responses, and integrates with MongoDB.

* Pydantic*: Used with FastAPI for data validation and serialization.*

Role: Validates incoming request payloads and enforces data types before processing or saving to MongoDB.

* Motor: Async Python driver for MongoDB

Role: Enables non-blocking interaction with MongoDB from FastAPI, allowing scalability and better performance.

* MongoDB Compass: GUI for MongoDB database.

Role: Used to visualize and manually inspect the stored API data for debugging and verification purposes.

* Swagger UI: Automatically generated API documentation and testing interface.

Role: Enables live interaction with API endpoints directly from the browser for testing.

* Postman: API testing and debugging tool.

Role: Used to test API endpoints manually, simulate client requests, verify response formats, and check status codes.

* OpenWeatherMap(3rd Party API): Source for real-time weather data.  
  API key is source of external JSON data  
  Role: Used to demonstrate real-world API consumption, data validation, and persistence in MongoDB.

To use OpenWeatherMap:

* Sign up at [openweathermap.org/api](https://openweathermap.org/api)
* Use: https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

### 3. Backend Architecture Overview

The architecture of the system followed a modular layered structure:

1. API Router Layer: Defines HTTP routes and handles incoming requests.
2. Schema Layer (Pydantic): Defines data models and ensures correct data structure and types.
3. Database Layer (MongoDB via Motor): Handles data insertion, retrieval, update, and deletion.
4. Service Layer : Contains business logic like transformation, filtering, and error management.
5. External API Integration: Fetches and validates weather data before storing it.

### 4. Designing and Implementing RESTful API with CRUD Operations

Build REST API (FastAPI + MongoDB)

**Framework: FastAPI** was used to build the API.  
Install Required Libraries: pip install fastapi uvicorn

|  |  |  |
| --- | --- | --- |
| Endpoint | Method | Description |
| /weather/ | GET | List all entries |
| /weather/{id} | GET | Get by ID/ specific record |
| /weather/ | POST | Add a new entry |
| /weather/{id} | PUT | Update entry |
| /weather/{id} | DELETE | Delete entry |

The FastAPI framework makes routing straightforward and allows automatic generation of OpenAPI documentation.  
Data Flow:  
Fetch weather data using requests.get(), and Validate using Pydantic:

|  |  |
| --- | --- |
| Fetch weather data using requests.get() | Example validated data: |
| class WeatherDataIn(BaseModel):  name: str  main: WeatherMain  wind: WeatherWind | {  "name": "Paris",  "main": {"temp": 65.5, "humidity": 45},  "wind": {"speed": 8.2}  } |

Save to MongoDB using Motor (async/await)

### 5. API Documentation and Testing

#### Swagger UI Install the required dependencies using the following commands: pip install requests pydantic pymongo python-dotenv fastapi uvicorn

Run main.py and Swagger UI is automatically available at http://localhost:8000/docs.

Lists all available endpoints.

Provides a test console where developers can:

* A screenshot of a computer

  AI-generated content may be incorrect.Submit HTTP requests.
* View live API responses.
* See required request formats and example payloads.
* Swagger was auto-generated using FastAPI’s built-in OpenAPI support.

#### Postman

* Used to simulate client behavior and test:
* A screenshot of a computer

  AI-generated content may be incorrect.Header management (adding API keys, city names, imperial units).
* Saving request-response history.
* Each endpoint (GET, POST, PUT, DELETE) was tested for:
* Valid/invalid payloads.
* Error scenarios such as missing fields, invalid IDs.
* Status code accuracy : 200 OK, 201 Created, 404 Not Found, 422 Validation Error.

### 6. Error Handling and Security

#### Error Handling

* Pydantic handles basic validation errors automatically.
* FastAPI catches and formats errors for invalid input and returns clear HTTP responses.
* HTTPException(status\_code=404) for "weather data not found"
* For future consideration, I will add custom exceptions for duplicates and invalid formats

#### Security Practices

* All data access was through validated models to prevent injection.
* API was tested over http://localhost, but HTTPS is recommended for production.
* Sensitive data like API keys from 3rd-party APIs was managed via environment variables (.env) and using python-dotenv.

### 7. MongoDB Compass & Testing

MongoDB Compass: was connected using the same connection URI used in the FastAPI backend.

Collections and documents were manually reviewed for:

* Schema structure.
* Data persistence.
* CRUD operation verification (record created, updated, or deleted from API).
* Compass provided a graphical way to monitor data during development.

Testing Tools: test\_main.py

* fastapi.testclient for endpoint testing.  
  pip install pytest  
  pytest test\_main.py

## Tests Included

* test\_update\_weather()  
  ✔ Creates a weather record  
  ✔ Updates it via PUT request  
  ✔ Asserts that the updated fields (e.g., name and temperature) are correctly modified
* test\_delete\_weather()  
  ✔ Creates a weather record  
  ✔ Deletes it via DELETE request  
  ✔ Confirms deletion by asserting a 404 on a follow-up GET request
* test\_get\_weather\_by\_id()  
  ✔ Creates a weather record  
  ✔ Fetches it via its ID  
  ✔ Asserts that the returned data matches the inserted values  
  A screenshot of a computer

  AI-generated content may be incorrect.

# Conclusion:

This project demonstrated a complete backend API development cycle using FastAPI with MongoDB integration and Swagger/Postman testing. The steps involved:

* Selecting and integrating a 3rd-party API.
* Validating external data.
* Designing a RESTful API architecture.
* Persisting validated data using a NoSQL database.
* Testing endpoints and functionality through Swagger and Postman.

This architecture ensures scalability, maintainability, and real-world applicability in full-stack systems.