**Flight Tracker API**

This project is a **FastAPI-based microservice** that scrapes flight information from <https://www.flightstats.com/> and stores it in a PostgreSQL database. I have used **Scrapy** for web scraping and **Celery + Redis** for background processing.

**Highlights**

* FastAPI endpoint to trigger scraping based on airline code, flight number, and date
* Scrapy spider to extract real-time flight details from FlightStats
* PostgreSQL database for storing flight data
* Celery task queue for asynchronous scraping jobs
* Dockerized for easy deployment and scaling
* Redis (Message Broker / Queue) used as a broker for Celery.

**Architecture**

Client --> FastAPI --> Celery --> Scrapy --> FlightStats.com --> PostgreSQL

| |

Redis (Queue) Store Extracted Data

**Client**: Sends a GET request with flight\_number and flight\_date.

**FastAPI**: Receives the request and triggers an async task via **Celery**.

**Celery**: Pushes the task into **Redis**, acting as a queue.

**Scrapy**: Picked up by Celery, Scrapy scrapes FlightStats.com for live flight details.

**FlightStats.com**: The target website we scrape using HTML parsing.

**Scrapy**: Extracts flight data (origin, destination, time, status).

**PostgreSQL**: The cleaned data is saved to the database.

**Project Structure**

flight-tracker-api/

app/

main.py # FastAPI entrypoint

crud.py # DB functions

database.py # SQLAlchemy engine/session setup

models.py # SQLAlchemy models

celery\_config.py # Celery settings

tasks.py # Celery task to run spider

scraper/

flight\_spider.py # Scrapy logic to extract DOM data

pipelines.py # Scrapy pipeline to insert data to DB

run\_spider.py # Manual spider runner (for debugging)

tests/

test\_api.py # Unit tests

debug\_flight\_detail.html # Sample saved HTML for offline selector debugging

Dockerfile # Build API container

docker-compose.yml # Run API + Redis + Celery + PostgreSQL

requirements.txt # Dependencies

**Setup & Execution**

**1. Clone the Repository**

git clone https://github.com/Vinoth-RN/flight-tracker-api.git

cd flight-tracker-api

**2. Start Services**

docker-compose down -v # optional clean-up, not required every time.

docker-compose up –build

docker compose logs -f api # to check the logs if needed  
  
example of how it looks:  


**3. Access the API**

<http://localhost:8000/docs>

**API Endpoint**

**GET /scrape-flight/**

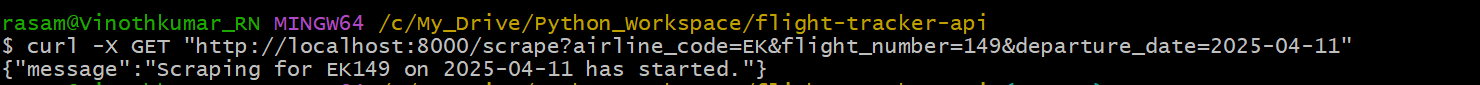
Trigger flight scraping in the background.

**Query Parameters:**

* flight\_number: e.g. EK547
* flight\_date: e.g. 2025-04-11

**Example:**

curl <http://localhost:8000/scrape-flight/?flight_number=EK547&flight_date=2025-04-11>

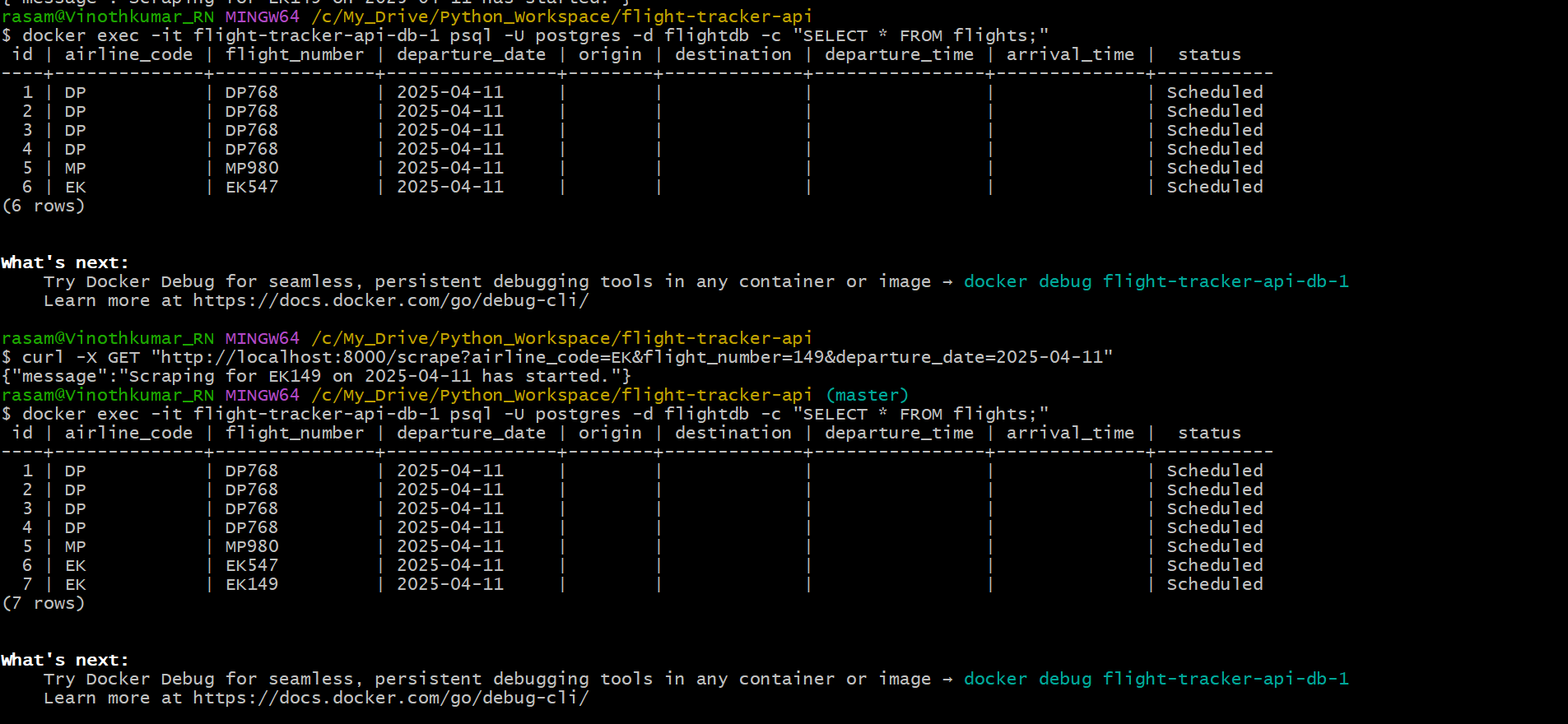


**Response:** {"message":"Scraping for EK549 on 2025-04-11 has started."}

**How DB insertion looks like :**

**Check with below**

docker exec -it flight-tracker-api-db-1 psql -U postgres -d flightdb -c "SELECT \* FROM flights;"



**Web Scraping Logic (Scrapy)**

* Target URL is constructed dynamically using flight number and date.
* Scrapy selectors pull DOM elements like:

airports = response.css("div.text-helper\_\_TextHelper-sc-8bko4a-0.route-airport-code::text").getall()

departure\_time = response.css("div.ticket\_\_TicketBody-sc-1rrbl5o-3 span::text").get()

* Extracted info includes:
  + Airline code & flight number
  + Departure & arrival airports
  + Scheduled & actual times
  + Flight status

**Tests**

Test the API using Pytest:

docker-compose exec api pytest

Example: tests/test\_api.py

response = client.get("/scrape-flight/", params={"flight\_number": "EK547", "flight\_date": "2025-04-09"})

assert response.status\_code == 200

Note : Make sure to always get the correct flight details, rather than a random a string to see the complete data in postgres.