Objective

Build a Python Flask/Django web application that interacts with the **Amadeus Travel API** to fetch flight prices for a specific route, implement local caching using **Redis** or **MongoDB**, and manage the Amadeus API access token automatically. The application should be containerized using **Docker** and include a README with detailed setup instructions.

Requirements

1. **Public API Integration**
    - Use the **Amadeus Travel API** to fetch flight price data.
    - API Documentation: Amadeus Flight Offers API Docs
    - Fetch and return the cheapest available flights between two locations (airport IATA codes) for a given travel date.
    - You will need to sign up for an API key at Amadeus for Developers.
    - Example API endpoint:
        - GET https://test.api.amadeus.com/v2/shopping/flight-offers?originLocationCode=JFK&destinationLocationCode=LAX&departureDate=2024-12-01&adults=1&max=1
    - Parameters:
        - originLocationCode: Departure airport IATA code (e.g., JFK for New York).
        - destinationLocationCode: Arrival airport IATA code (e.g., LAX for Los Angeles).
        - departureDate: Travel date in YYYY-MM-DD format.
        - adults: Number of passengers.
        - max: Maximum number of results.

2. **Token Management**
    - The Amadeus API requires an **OAuth2 access token** for authentication. Tokens are valid for a limited period, and your application must automatically handle the process of fetching and refreshing the token.
    - You can obtain the **Client ID** and **Client Secret** by signing up at Amadeus for Developers.
    - Tokens are valid for approximately 30 minutes. Your application must manage the token lifecycle automatically, obtaining a new token when necessary without manual input
    - **Your application should:**
        - Obtain the token during startup.
        - Automatically refresh the token when it expires (token expires in around 30 minutes).
        - Store the token in memory (or cache) so it can be used for subsequent API calls.
        - Ensure this process is seamless, and the token management is invisible when running the Docker image.

3. **Endpoints**
   - **Ping Endpoint**: Create an endpoint to check if your service is running.
     - GET /flights/ping → Returns {"data": "pong"}
   - **Flight Price Endpoint**: Create an endpoint that fetches the cheapest flight prices based on the origin, destination, and travel date.
     - GET /flights/price?origin={origin_code}&destination={destination_code}&date={yyyy-mm-dd} → Returns the cheapest flight offer.
     - Example Response:

```
{
  "data": {
    "origin": "JFK",
    "destination": "LAX",
    "departure_date": "2024-12-01",
    "price": "250 USD"
  }
}
```

4. **Caching**
   - Implement caching for the flight price data using **Redis** or **MongoDB**.
   - Cache the flight price data for **10 minutes** to avoid excessive API calls.
   - Add a **nocache** parameter to force fetching fresh data from the Amadeus API and refreshing the cache:
     - GET /flights/price?origin={origin_code}&destination={destination_code}&date={yyyy-mm-dd}&nocache=1

5. **Docker Setup**
   - The application should be packaged as a **Docker image**.
   - Ensure the Dockerfile builds and runs the app correctly.
   - When we run your Docker image, it should automatically manage the access token and caching without any manual setup or intervention.

6. **Deliverables**
   - **Code**:
     - Provide your source code in a .tar.gz archive.
     - Include a README.md with setup and run instructions (including Docker setup).
   - **README** should include:
     - Instructions for running the app locally with Docker.
     - Any assumptions made during development.
     - **How the application manages the access token**, to ensure it's clear that no manual token refresh is needed during testing.
   - We will run your solution on a **Linux environment**, so please ensure compatibility.

- Test your solution using **curl** or a similar tool.

7. **Key Features to Assess**
   - **Code Quality**: Readability, adherence to best practices, and proper error handling.
   - **API Integration**: Correctly fetching data from the Amadeus Travel API and managing the token lifecycle.
   - **Caching Implementation**: Effective caching with Redis or MongoDB, and the ability to force fresh data.
   - **Token Management**: The ability to handle OAuth2 token fetching and refreshing automatically.
   - **Dockerization**: Ability to containerize the app and run it via Docker.

Evaluation Criteria

- **Adherence to Requirements**: The app should match the instructions and output the correct data.
- **Code Structure**: Clean, modular, and well-documented code is expected.
- **Error Handling**: The app should handle invalid input, network issues, and other exceptions gracefully.
- **Caching Efficiency**: The caching should reduce unnecessary API calls, while keeping the data up-to-date.
- **Token Management**: Ensure that the application handles token fetching and refreshing seamlessly.
- **Dockerization**: The application should run without any additional manual steps once the Docker image is launched.

Example Commands

- To test the ping endpoint:

curl http://localhost:8080/flights/ping

- To fetch flight price (with caching):

curl http://localhost:8080/flights/price?origin=JFK&destination=LAX&date=2024-12-01

- To fetch live flight price (without caching):

curl http://localhost:8080/flights/price?origin=JFK&destination=LAX&date=2024-12-01&nocache=1