**Complete Execution Guide**

Here’s a complete execution guide for project, detailing every step from setup to execution, including installing dependencies, running files, and testing via Postman.

**1. Setup Environment**

1.1 Install Poetry

If Poetry is not installed, install it using the following command:

pip install poetry

Verify the installation:

poetry --version

1.2 Clone the Repository

Clone the project repository to your local machine:

git clone <your-repo-url>

cd <your-project-folder>

1.3 Configure Dependencies

Install all project dependencies using Poetry. This will also create and activate a virtual environment for the project**.**

poetry install

Activate the Poetry shell:

poetry shell

**1.4 Required Dependencies**

Here’s a list of dependencies your project uses:

* fastapi: Framework for building APIs.
* uvicorn: ASGI server to run FastAPI applications.
* duckdb: Database engine for handling data.
* redis.asyncio: Asynchronous Redis client for caching and streaming.
* pytest, pytest-asyncio, pytest-cov: For testing.
* pydantic: For request validation.
* ruff: Linting and code style.

Install missing dependencies (if not already included in pyproject.toml):

poetry add fastapi uvicorn duckdb redis.asyncio pydantic

poetry add --dev pytest pytest-asyncio pytest-cov ruff

**2. Configure and Setup Files**

Ensure you have the following files in your project directory with the specified names:

1. data.py: Manages schema definitions and data generation using Faker.
2. logger.py: Handles logging setup.
3. DuckDBManager.py: Manages DuckDB database connection and queries.
4. mainapi.py: Contains the FastAPI application with endpoints.
5. tests/: Contains unit tests for each file.
6. pyproject.toml: Contains Poetry configuration.

**3. Running Each File**

3.1 logger.py

This file sets up logging for the project. Run it to ensure the logger is correctly configured:

Poetry run python logger.py

3.2 DuckDBManager.py

This file sets up and connects to the DuckDB database. Run it to verify the database connection:

Poetry run python DuckDBManager.py

3.3 data.py

This file generates sample data using Faker. Run it to populate the database:

Poetry run python data.py

3.4 mainapi.py

This is the main application file that starts the FastAPI server.

uvicorn mainapi:app --reload

* The server should start successfully, and you will see:
* INFO: Application startup complete.
* INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

3.5 Start redis server in cmd. Ensure it is running on 6379 port

Redis-server

Redis-cli -to check updates manually

XRANGE response\_duck - + ---to view the updates in response stream

**4. Testing the API using Postman**

**4.1 Start the Server**

Ensure mainapi.py is running on http://127.0.0.1:8000.

**4.2 Add Endpoints in Postman**

1. **Root Endpoint**
   * URL: http://127.0.0.1:8000/
   * Method: GET
   * Expected Response:
   * {"message": "Welcome!"}
2. **Update Cell**
   * URL: http://127.0.0.1:8000/update\_cell
   * Method: POST
   * Headers: Content-Type: application/json
   * Body Example:

{

"table": "sales",

"column": "quantity",

"value": "50",

"condition": "customer\_id = 1 AND product\_id = 1",

"level": 0

}

Expected Response:

{"status": "success", "message": "Updated sales"}

1. **Get Updates**
   * URL: http://127.0.0.1:8000/get\_updates
   * **Method: GET**

Expected Response:

{

"status": "success",

"updates": [

{"table": "sales", "column": "quantity", "value": "50", "condition": "customer\_id = 1 AND product\_id = 1"}

]

}

**Check duckdb for the updated tables on another terminal after stopping the server with ctrl+c.**

duckdb

.salesmetrics.duckdb

.open tables

Execute the query for the table:

Select \*from sales\_summary\_by\_product\_family where condition; (to view the changes)

**5. Running Unit Tests**

Run the unit tests to verify the functionality:

**pytest --cov=Challenge tests/**

* Expected Output:
  + All test cases should pass.
  + Coverage report should display a summary of test coverage.

**6. Yet to be done: CI/CD Setup**

Ensure you’ve configured the .github/workflows/python-app.yml for GitHub Actions to automate tests, linting, and deployment. Push the changes to GitHub, and the workflow should trigger automatically.

**Conclusion**

With these steps, you can successfully set up, execute, and test your project from scratch. Following this guide ensures anyone new to the project can easily run and validate its functionality.