Test Task QA / SDET TRADING

Task: Build an automated test suite for a RESTful API that simulates a trading platform with WebSocket support. Dockerize the test suite and server.

Description: You need to create a sample RESTful API server that exposes a set of endpoints to simulate a trading platform. The API should support managing and executing orders. In advanced the requirements, the platform should use WebSocket connections also, for receiving real-time order status messages. There's also a requirement to build an automated test suite for the API using Python and pytest (or any other framework you like), and Dockerize both the test suite and the server.

Platform: Windows, Python 3.6+

Knowledge base: Python, Pytest, RobotFramework, FastAPI, Docker

Success criteria: Please feel free to implement all the requirements that you feel is adequate for your knowledge level. Do not hesitate to send us incomplete tasks. We are looking for your ability to solve real world tasks and interact with new technologies. If you have any questions during the execution of test task, do not hesitate to ask your People and Culture person of contact for some clarifications.

To open API documentation, you can use Swagger online tool

Server requirements:

All provided schemas and requests requirements are optional. Use them if they suit you. It's important that you implement all required endpoints with described functionality.

1. Implement the following API endpoints:
   - GET /orders
   - POST /orders
   - GET /orders/{orderId}
   - DELETE /orders/{orderId}
2. For more info regarding responses from each endpoint, you can follow the provided OpenAPI documentation or use your own solution.
3. After client sends POST/orders request, server sends confirmation and orderId. Then a client can request an order info by using GET /orders/{orderId} and receive status.
4. Each endpoint should have a random short delay between 0.1 and 1 second.
5. The Database can be kept in memory.

Advanced requirements:

1. Make server asynchronous.
2. Implement WebSocket functionality into your server.
3. After orders are received from the client, the server sends back orderId and orderStatus as a response. Assume that Order has three statuses: PENDING, EXECUTED, CANCELLED. Order is created with PENDING status then after short delay status is changed to EXECUTED.
4. Order can be cancelled only when the order status is PENDING.
5. Notify all subscribed clients about the order status change (PENDING, CANCELLED or EXECUTED) through WebSocket connection.

Test cases requirements:

1. The test suite should cover all endpoints and methods of the API.
2. The test cases should be organized into test suites and test functions using pytest or any other testing framework of your choice.
3. The test cases should include both positive and negative scenarios, including input validation errors.
4. The test cases should assert the correctness of the API responses and the expected behavior of the API.
5. The test suite and the API server should be dockerized and should be able to run in separate containers
6. Generate report as a standalone file (e.g. html).

Advanced requirements:

1. The test suite should include checks for the WebSocket connections, including:
   a. Ensuring that real-time order status events are properly received by connected WebSocket clients.

    b. Ensuring that the order of receiving messages are correct (orderStatus=PENDING, received before orderStatus=EXECUTED or no messages are received after receiving orderStatus=CANCELLED).

2. Implement performance testing according to the following scenario:
    a. place 100 orders at the same time, validate the responses from REST API,
    b. receive WS messages from the server,
    c. calculate average order execution delay and standard deviation based on the *a.* and *b.* timestamps,
    d. print all the metrics to the console.

Submission: Please provide your solution as a GitHub repository with a clear README.md file that explains how to run the test suite and server in Docker containers, any dependencies or setup required, a link to the Swagger documentation.

openapi3...(2).yaml