My Solution:

Although I didn't use FastAPI before, I thought it would be a great idea to use it as the API server as a recommendation given from you and also as a challenge for me and it was really interesting.

- My final solution is creating 3 more containers:
  a- MQTT Client that listens to VerneMQ for the given topic "thndr-trading", then process the data and reformat it and save it to a postgresql database.
  b- PostgreSQL database that is used to save the data (stocks data and users data as well) and be the shared database for both MQTT client and the API server.
  c- API Server, which is based on FastAPI framework.

Install and start testing:

use the following command to start the containers: "sudo docker-compose up -d"
then using "http://localhost:8000/docs" in the browser, you can provide inputs for a give API endpoint using the built-in Swagger UI that comes with FastAPI framework.

API Endpoints:

--- POST /user { "user_id": <user_id> }
if user was not found, status_code with 404 and detailed message " user not found" is sent
if user was found,
expected response:

```
{
    user_id : <integer>,
    name: <string>,
    balance: <integer>,
     stocks:  [
                {
                    stock_id: <string>,
                    name: <string>,
                    number: <integer>

                }
                .
                .
                .
            ]
 }
```

--- POST /stock { "stock_id": <stock_id> }
if stock was not found, status_code with 404 and detailed message "stock not found" is sent
if stock was found,
expected response:

```
{
      stock_id : <string>,
      name: <string>,
```

availability: <integer>,
current_price: <integer>,
day_highest_price: <integer>,
day_lowest_price: <integer>,
hour_highest_price: <integer>,
hour_lowest_price: <integer>
}

--- POST /buy { “user_id”: <user_id>, “stock_id: <stock_id>, “total”: <total>, “upper_bound”: <upper_bound>, “lower_bound”:<lower_bound” }

response in 4 cases:
1- the “total” is more than the available stocks of a given stock, then status code with 409 and detailed message is sent.
2- the current price of a given stock was beyond the limits “more than the upper_bound or less than the lower_bound”, then status code of 409 and detailed message is sent.
3- the user balance was not sufficient for the given transaction (total number of stocks × stock price > user balance), then a status code of 409 and detailed message is sent.
4- everything was find, then a status code of 200 and { “message”: “success” } is sent.

--- POST /sell { “user_id”: <user_id>, “stock_id: <stock_id>, “total”: <total>, “upper_bound”: <upper_bound>, “lower_bound”:<lower_bound” }

response in 4 cases:
1- the “total” is more than the available stocks of a given stock of a user, then status code of 409 and detailed message is sent.
2- the current price of a given stock was beyond the limits “more than the upper_bound or less than the lower_bound”, then status code of 409 and detailed message is sent.
3- everything was fine, then a status code of 200 and { “message”: “success” } is sent.

--- POST /deposit { “user_id”: <user_id>, “amount”: <amount> }
if the user was not found, status_code with 404 and detailed message “ user not found” is sent
if the user was found,
expected response:
a status code of 200 and { “message”: “success” } is sent.

--- POST /withdraw { “user_id”: <user_id>, “amount”: <amount> }
If the user was not found, status_code with 404 and detailed message “ user not found” is sent and if the user balance was not sufficient, a status code with 409 and detailed message is sent.
If everything was fine,
then expected response:
a status code of 200 and { “message”: “success” } is sent.

Future Enhancements:

Due to the limited time I had, in order for this application to be better, I should do the following:
1- add unit tests for different API endpoints.
2- remove the time.sleep(10) from both MQTT client and API server as this should not be in the

code and make the delay from docker compose (the delay was necessary to make sure DB is initialized.

3- refactor or clean the code, by removing duplication and make it more readable and maintainable.
4- use pydantic models in API endpoints request objects and response model.
5- I assumed that the highest and the lowest price is reassigned for every new hour or day and not the last 60 minutes or the last 24 hours, and this is not accurate.

At the end, I really want to thank you for this experience as I learned a lot from it.