AdAlpha Solutions Technical Task – Ryan Jeynes

Assumption

* Only one investor will be using this system.

When I was first given the technical challenge, it seemed rather vague. I went back to the customer to ask for some initial clarification to the requirement.

Questions

* “As stated in the task an investor wants to withdraw £X and from that request an order sheet is created with options to BUY, INVEST, SELL and RAISE. If the investor wanted to withdraw £X, why would an order sheet contain the BUY and INVEST? Those two options appear to be a deposit rather than a withdrawal.”
* “My second question is that the BUY and SELL options on the order sheet mention units rather than currency. The information you have given me mentions the £220,000 portfolio on current assets, which assets they have and the % of each asset towards their portfolio. How would I acquire the necessary information for how much a unit is worth for each asset?”
* “My final question for now is what is the difference between BUY and INVEST as well as between SELL and RAISE? All I notice from the information is unit and currency.”

When I had the answers to my questions, I felt as though I had enough information to start the task. I analysed the requirements set by the customer and knowing the amount of time I had to do the task, planned a route of action. My initial thoughts were that I was going to using the following technologies – Go, VueJS, Postgres for DB storage and Elasticsearch for logging and possibly metrication. Rather than setting all this up on my development environment, I planned to use docker and docker-compose to make the project easy to set up on different environments. Therefore I wrote boilerplate code to connect to a Postgres database and then started to build my docker image for the backend and write the compose file to connect my backend to Postgres. Once these two components were connected, I figured that before I started writing logic for manipulating data in the database, I would need to add tables and entries for an investor, the assets and the initial portfolio therefore I investigated and implemented the solution into a docker-compose mount.

Thinking about how to store the data, I realised that I would need to convert the initial 220,000£ into units and store that into the database as their investments could fluctuate based on price of the assets (this was a static price from when I started development). I wanted to update the asset price dynamically every time a change was made, therefore I investigate an API for the financial times. I found the API could not be accessed therefore I investigate html scraping for the price. Found a package called colly which seemed simple to use for scraping the price.

As I started writing logic for the invest endpoint, I began to think about testing the endpoint and services I was writing. I began to wonder how I could make these tests reusable when a lot of them would require a database to be there with database in, the obvious answer would be to have another docker-compose file for the test database. Advantages of this was it could stand up a test database, test all functions it needed and then get rid of the database without needing any prerequisites on the environment other than Go and Docker. I figured as I didn’t quite understand all the features of go testing and I was pressed with time, I would leave testing until a later time and concentrate on getting the endpoint working and start with a basic user interface. In an ideal scenario, I would’ve taken the TDD approach but I was aware of multiple features I wanted to add to the system to go beyond the task set to me. Throughout the development of the tool, I was conscious about where I should test and made sure I tested my code as I developed it.

I added Elasticsearch and Kibana for logging purposes so that instead of trawling through application logs in a file, a developer could go to a dashboard and easily filter on time or specific packages/methods etc. This way a developer could get metrics over time on how much a package/method is failing and possibly suggest a rethink/redesign.

After all the endpoints were written for the API, I started developing the User Interface. This took longer than I would’ve hoped but I think I have represented to applications functionalities well with my limited knowledge of VueJS.

Also I added functionality for the user to enter different currencies and the backend would use an API to find the current exchange rate and convert the value to GBP before carrying on with the instruction.

If I had more time on the project I would’ve added metrics to Elasticsearch so that we could get an idea on how much each people are using the application and which parts of the application too. Also, I would add history of what users do and make that accessible through the User Interface so that they could look back over time at what investments they made/sold. I would also add the asset name into the user interface which wouldn’t be difficult as the API already exists but the User Interface element would take longer than I’d like. Lastly, I would add the ability to store more than one investor and create the User Interface in a way that multiple investors could use it.

As you can see from this write-up, I feel as though I have achieved what I set out to do according to my opinion on the technical task and have asked the customer questions where needed. Also, I acknowledge that given more time I would add more functionality which is above the ask for this challenge.