MicroService Portfolio Management Project

Overview

In this part of the project you will continue to use and develop the REST API and (probably) Angular front-ends as the core of an application that will manage a portfolio of assets for a single user.

The secondary aim is to create a system that will allow users to **see** and **interact** with the contents of their portfolio.

If/When you have implemented the above basic system, you might take on one or more of the suggested improvements in this document. These improvements are open to your own enhancements whereby you can make use of your particular skills and experience.

At a minimum you should aim for the following features:

- A web UI should facilitate users to:
    - Place an order for a particular stock
    - Browse their trading history
    - View the status of each of their historical Trades
- As before, there will be no authentication and a single user is assumed, i.e. there is no requirement to manage multiple users.
- You should use MySQL for any persistent storage.
- You should aim to complete the core functionality described above before attempting any of the extensions described later in the document. The core technical elements should be designed by you. This document is provided as an outline, however all data storage, REST APIs etc. should be designed by your team.
- **Important**: You should develop an application Roadmap. This is YOUR TEAM'S application, part of the project is to design the application - your presentation should contain a description of a roadmap of features that you would add **if you were to continue work on this**.

  What features would be added to this application? What would a fully featured UI look like?

  You don't need to implement these features. This is a roadmap for what you would implement IF you were given more resources for this project.

  You can bring in your expertise from other fields - some examples might be: Any ML features or components that you would consider. Any web services or cloud services that

this system could interface with. Any further or interesting UI features. Integrations with other applications. Security concerns.

Suggested Schedule

Your team needs to allocate people to work on work on each area of this project. E.g. some of your team may continue work on the previously started Spring application, or some of your team should be designing the user interface.

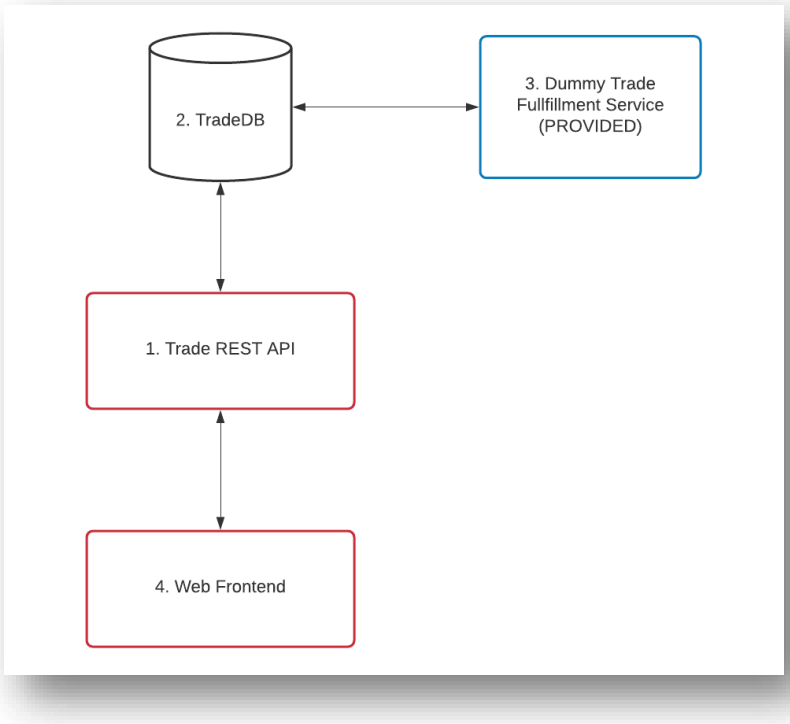
The entire system is not expected to be complete until the final day.

The first step should be to come together as a team and decide on the core vision for what your application should do. When you have decided on this, or if you have questions on this, you should then schedule a meeting with your instructor to give guidance before you proceed to far down a particular path.

System Architecture

It is up to you and your team to decide on an overall architecture for the system. However, you should consider enterprise deployment environments and aim for a microservice style architecture.

The following architecture is a **possible** architecture for the core components. In this architecture, the "Dummy Trade Fulfillment Service will be provided to you IF you want to include it in your architecture. However, again bear in mind that this is **your design**, this example is just to help you with ideas.



1. Trade REST API: Written in Java as a spring boot REST application. It will accept HTTP REST requests to Create Orders & Retrieve Trade records from the TradeDB. Update and Delete of Trade records should be subject to sensible business considerations. You should consider unit tests, sensible JavaDocs and use parameters in your application.properties file(s).

2. TradeDB: Data stored in a MySQL database.

3. Dummy Order/Trade Fullfillment Service: Sample source code for this component is available. This is **test component** that simulates a system that would send your trade orders to a stock exchange. You could write this or something similar that suits your application yourself. However, if you want to use the default one, the example order filling simulator is here: https://bitbucket.org/fcallaly/dummy-order-filler

4. Web Frontend: This is a web UI written in Javascript, Angular, or any other front-end framework you are experienced with (check with your instructor if using something else). This UI would allow users to browse their Trading history, and request for Trades to be made by sending HTTP requests to the Trade REST API.

Note on terminology and limitations: For the purposes of this exercise there are limitations to the functionality. The terms Order and Trade are largely interchangeable though in the real world they are different things. An order is an "intention to trade", e.g. I'd like to buy 1000 Tesla shares, or I'd like to sell 2500 Apple shares, while a trade is the confirmation "you bought 200 Tesla shares @ 500USD and 372 Tesla shares @501 USD, etc. As such, in a real scenario, a single order may result in many smaller separate trades at a range of prices. For this exercise all orders are considered as "immediate" or "Market" orders and will fill completely or not at all, and they will always fill at the current market price, there is no concept of "limit" orders and no orders will "rest" in the market.

Optional Extras

These are JUST IDEAS, they are not requirements, there is no pressure to complete any or all of these

Once you have finished the core features, you may add some of the optional extras described below, or come up with your own extensions that you think would suit this system. Check with your instructors before beginning implementation of any extension features.

Some extra optional features you may add include:

- A service that manages the users current portfolio, e.g. what Stocks, Cash, Bonds etc. they are currently holding.

OR

- A service that will give advice on suggested trades e.g. for a given stock should the user BUY, SELL or HOLD

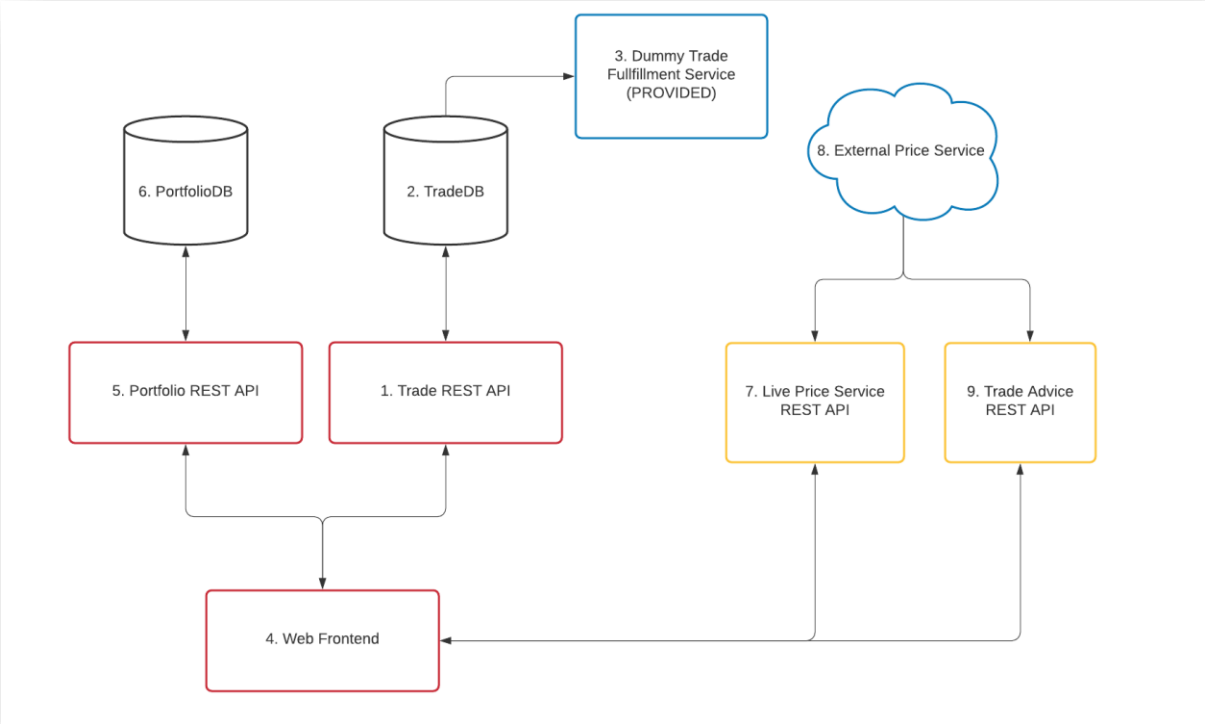
OR

- A service that will automatically make trades based on user defined criteria e.g. the stock to trade, the maximum value of trades, when to stop making trades

OR

- A service that will integrate with Slack to give users updates on the status of trades

An example of how some of these components might fit within the system is shown below. However, these are SUGGESTIONS, the design of these elements should be driven by your team. You are definitely NOT expected to complete all of the elements shown below.



5. Portfolio REST API: This could be written in Java and may even be part of the same spring boot application as the "Trade REST API". Alternatively, it may be a standalone spring boot application. This service should keep track of multiple asset classes (Cash, Stocks etc.) currently held.

6. PortfolioDB: Similar to TradeDB this should be data stored in a MySQL database.

7. Live Price Service REST API: If included, this may be written in Python or Java. This component would make price data available to your other components through a REST API. E.g. so that live price data could be shown in the UI. This component may read it's price data from a live service such as Yahoo Finance or elsewhere.

8. External Price Service: This is an External service e.g. Yahoo Finance.

9. Trade Advice REST API: If included, this may be written in Python OR Java. This service would receive RESTful requests for advice regarding stock tickers. It should return an answer indicating whether the advice is to BUY, SELL or HOLD that stock. E.g. The UI may use this service to indicate to the user what may be a sensible Trading option.

Teamwork

It is expected that you work closely as a team during this project.

Your team should be self-organising, but should raise issues with instructors if they are potential blockers to progress.

Your team should use a task management system such as Trello to keep track of tasks and progress. You should include details of this in your final presentation.

Your team should keep track of all source code with git and bitbucket. You should include some details of this in your final presentation.

You should create a separate bitbucket repository for each component that you tackle e.g. front-end code can be in its own repository. If you create more than one spring-boot application, then each can have its own bitbucket repository. To keep track of your repositories, you can use a single bitbucket 'Project' that each of your repositories is part of.

Your instructor and team members need to access all repositories, so they should be either A) made public OR B) shared with your instructor and all team members.

Throughout your work, you should ensure good communication and organise regular check-ins with each other and your instructor.

Appendix A: Optional Extra - Portfolio REST API

This API would be used to keep track of a personal portfolio of cash and investments (e.g. stocks, bonds).

The API would facilitate CRUD operations for this data, however you may choose to only implement some of the standard CRUD operations given the available time.

Some example operations that would be facilitated by this API:

- Add some stocks for a particular ticker to your portfolio
- Calculate the users' stock holdings by querying the Trade history.
- Add some cash to your portfolio.
- Remove some cash from your portfolio
- Add other assets to your portfolio.
- Remove other assets from your portfolio

Appendix B: UI Ideas

The screen below might give you some ideas about the type of UI that could be useful. You are NOT expected to implement the screen below exactly as it is shown. This is JUST FOR DEMONSTRATION of the type of thing that COULD be shown. Consider including charts and graphs using an Angular extension (search for them!)

Appendix C: Some Useful Services

Some of these may be useful depending on the features you choose to put in your system.

SimpleMarketData Server:

We have provided a simple simulated market data service that can be used without the need to sign up for any web account. This is hosted at the following address: https://marketdata.multicode.uk – note that there is no swagger page for this server.

It supports two modes of operation:

1. Short form, predictable data.

This is intended for simple testing of models and has a very predictable cycle of prices. Any Symbol can be used, each symbol will get a unique set of prices. The end point takes the form /API/PseudoFeed/[Ticker] where Ticker is the name of the symbol to be retrieved.

For example https://marketdata.multicode.uk/API/PseudoFeed/TESTTICKER would return JSON data as follows:

```
{TESTTICKER:{last:98465}}
```

2. Simulated Market Data.

This endpoint provides a more realistic set of services that can be used to create reasonable graphs and/or trading predictions for strategies. The API endpoints are as follows:

https://marketdata.multicode.uk/API/StockFeed/GetSymbolList

Retrieves the list of valid symbols. Returns JSON data as follows:

```
[{"companyName":"ConocoPhillips","symbol":"cop","symbolID":1},{"c
ompanyName":"cpgx","symbol":"cpgx","symbolID":2},...]
```

https://marketdata.multicode.uk/API/StockFeed/GetSymbolDetails/[Ticker]

Retrieves the details of a single symbol. Returns JSON data as follows:

```
[{"companyName":"cpgx","symbol":"cpgx","symbolID":2}]
```

Example request: https://marketdata.multicode.uk/API/StockFeed/GetSymbolDetails/cpgx

https://marketdata.multicode.uk/API/StockFeed/GetStockPricesForSymbol/[Ticker]

Retrieves the prices of a given symbol. This returns a JSON data packet similar to the following:

```
[{"companyName":"cpgx","periodNumber":1494,"price":16.97,"symbol":"c
pgx","timeStamp":"08:17:44"}]
```

This is the default behaviour, returning a single row of "realtime" prices. Repeated requests will get the new "latest" price.

https://marketdata.multicode.uk/API/StockFeed/GetStockPricesForSymbol/[Ticker]?HowManyValues=[N]

Retrieves the N most recent prices of a given symbol. This request returns a JSON array similar to the following:

```
[{"companyName":"cpgx","periodNumber":1569,"price":17.41,"symbol"
:"cpgx","timeStamp":"08:42:47"},{"companyName":"cpgx","periodNumb
er":1569,"price":17.41,"symbol":"cpgx","timeStamp":"08:42:48"},{"
companyName":"cpgx","periodNumber":1569,"price":17.38,"symbol":"c
pgx","timeStamp":"08:42:49"},{"companyName":"cpgx","periodNumber"
:1569,"price":17.39,"symbol":"cpgx","timeStamp":"08:42:50"},{"com
panyName":"cpgx","periodNumber":1569,"price":17.4,"symbol":"cpgx"
,"timeStamp":"08:42:51"}]
```

https://marketdata.multicode.uk/API/StockFeed/GetStockPricesForSymbol/[Ticker]?HowManyValues=[N]&WhatTime=[HH:MM:SS]

Retrieves the N most recent prices of a given symbol at the specified time. This request returns a JSON array similar to the following:


```
[{"companyName":"cpgx","periodNumber":1094,"price":25.02,"symbol":"cpgx","timeStamp":"06:04:29"},{"companyName":"cpgx","periodNumber":1094,"price":25.02,"symbol":"cpgx","timeStamp":"06:04:30"},{"companyName":"cpgx","periodNumber":1094,"price":25.025,"symbol":"cpgx","timeStamp":"06:04:31"},{"companyName":"cpgx","periodNumber":1094,"price":25.03,"symbol":"cpgx","timeStamp":"06:04:32"},{"companyName":"cpgx","periodNumber":1094,"price":25.025,"symbol":"cpgx","timeStamp":"06:04:33"}]
```


https://marketdata.multicode.uk/API/StockFeed/GetOpenCloseMinMaxForSymbolAndPeriodNumber/[Ticker]?PeriodNumber=[N]

Retrieves the stats for the artificial "day" number N, known as a period. The feed also provides the notion of a "period". The trading period can be used as if it were a trading day when computing statistical trends. The statistical data endpoint `GetOpenCloseMinMaxForSymbolAndPeriodNumber` provides access to the computed High, Low, Open, and Close.

This request returns a JSON similar to the following:


```
[{"closingPrice":121.647,"maxPrice":17.5662,"minPrice":17.2285,"openingPrice":121.654,"periodEndTime":"01:39:59","periodNumber":300,"periodStartTime":"01:39:40","symbol":"cpgx","symbolID":2}]
```


Note: that the period for the current time is not provided, thus avoiding issues with wrap around data and algorithms accidentally predicting the future :-)

Design and limitations of the feed

The price data was converted from real NYSE Market data at 15 minute intervals which have been reduced to one seconds intervals. there are 21600 rows for each unique instrument starting from 00:00:00 to 05:59:59.

At 06:00:00 the data is reversed, this avoids a sawtooth-like reset of just replaying the data in loops.

A full 24 hour cycle is thus created from 4 segments, two forward and two reverse.

TickerList

Gives a list of tickers from Yahoo Finance giving details such as the name, country and industry sector and may be useful for categorising or validating ticker entry.

https://v588nmxc10.execute-api.us-east-1.amazonaws.com/default/tickerList