Super Simple Stock

# UseCases

|  |  |
| --- | --- |
| **UseCase No** | **Description** |
| UC\_DIVIDEND | Given a market price as input, calculate the dividend yield |
| UC\_PE\_RATIO | Given a market price as input, calculate the P/E Ratio |
| UC\_TRADE | Record a trade, with timestamp, quantity of shares, buy or sell indicator and trade price |
| UC\_VOLUME\_WEIGHTED | Calculate Volume Weighted Stock Price based on trades in past 15 minutes |
| UC\_GBCE\_INDEX | Calculate the GBCE All Share Index using the geometric mean of prices for all stocks |
| UC\_RECENT\_TRANSACTIONS | Get the list of transactions that happened for the cache configured time span |

# Technologies

|  |  |
| --- | --- |
| Name | Description |
| Maven | Used for build. Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework |
| SpringBoot with web, test and tomcat starter | Spring boot makes development faster and the starters option gives the spread to choose the right one that suits the requirement. |
| Guava | Used for caching, all the in-memory requirement is achieved by this. Guava has its own internal eviction event, which would remove the eelments after specified internal. Guava is quick when it comes to development and matched the requirement. There are other Caching like Redis,EHCache, Infinispan etc |
| Swagger | Swagger gives nice api documentation and can be smoothly integrated. Also, gives an opportunity to test the code seamlessly |
| Testng | For testing with datasets |
| Mockito | For mocking the services and testing individual components |
| BDDMOckito | For rest testing |
| Junit | For testing |

# API’s to fulfill usecase requirements

All the apis are exposed over rest and swagger integrated. This (**localhost:8080/swagger-ui.html**) can be used as the entry point as the application would be on port 8080

|  |  |  |  |
| --- | --- | --- | --- |
| API Name | Method | Satisfied UseCases | Description |
| /stocks/dividend | GET | UC\_DIVIDEND | Calculate dividend based on Symbol and MarketPrice |
| /stocks/pe-ratio | GET | UC\_PE\_RATIO | Calculate price earning ratio based on Symbol and MarketPrice |
| /trade/index | GET | UC\_GBCE\_INDEX | Calculate GBCE Index |
| /trade/transact | POST | UC\_TRADE | Record a trade, with timestamp, quantity of shares, buy or sell indicator and trade price |
| /trade/transactions | GET | UC\_RECENT\_TRANSACTIONS | Get all the recent transactions |
| /trade/volumeWeightedPrice | GET | UC\_VOLUME\_WEIGHTED | Calculate Volume Weighted Weight Price for the Symbol |

## Assumption and Improvement

* All the transaction details are getting fetch directly from cache. In real time scenario, considering volume, we should have got it done paginated
* Calculation logic - currently everything is happening within java code, we should be using some external tools like MAPLE, MATLAB, numpy a python library is also helpful
* Storage service - currently service layer is calling the repositories and repositories DB(Cache). We could have had a StorageService, with different implementations for Cache and DataBase
* Support Json request instead of query params, and pass the same Json to Service
* In real world, user would be having a login page, we can have a request tracer (unique id) associated with his session. All the information that is getting logged would have this id printed. This makes analyzing an issue on prod easier when integrated with SPLUNK or Kibana
* Guava - intention was to move the data that is getting removed from in-memory cache to database(another cache). Guava was not calling the onRemoval event immediately when the data is getting evicted

# Exceptions

* Have created a custom exception specific to this application
* ExceptionHandler created to cater to all exceptions

# Running the application

* spring-boot:run – to bring the application up
* mvn clean install – to build with testcases

# Testing the application

* Using Tests - Tests have been written for RestServices, Service and Repositories using Mockito, BDDMockito, Hamcrest, TestNg, MockMvc
* Using Swagger – Once the application is up, use **localhost:8080/swagger-ui.html** to test