Homework 4 – Design patterns

Our application uses stock data that is scraped with a python script (using scrapy) and this data is further processed to make predictions and analysis of the prices. The solution consists of 3 parts connected with each other. Those are Java Sprong Boot application (with 4-tier architecture), Flask and a python script that scrapes the necessary data daily. Each of these parts has its own role and when we connect them together our Stock Prediction Application is formed.

In order to provide a clean, maintainable, loosely coupled application we chose to implement the strategy design pattern in the service layer with which each functionality will be clear to the future developers and easy to maintain. No major methods are directly implemented in any of the services, they all have their own implementations, and each implementation uses strategies where the logic is more complex. We chose this pattern because it was the most suitable one for our project and it provides us easier maintainability of the application. For example, while I was implementing the design pattern, I noticed that we had made a mistake in the 3rd homework and was able to fix it without changing the whole service. Also, while I was cleaning the code, I found a duplicate in the calculation of averages of the minimum and maximum price which was very easy to clean up with the fact that it was in a strategy, I just had to make a very small modification in the service implementation. That is when we were sure that the strategy design pattern was right for our services. Out Flask analysis is also implemented using the factory pattern, with which no specific issuer, indicator or interval should be set in the code, the inputs are taken from the user and passed on dynamically to this analysis in which they are processed respectively. Our database is an endpoint to which all the parts of the application have access. The web scraper writes to the database and the other parts read from it and uses the data to provide useful information. This is why it was best to create our database as a singleton and use the singleton design pattern.

The implementation of the Strategy, Factory, and Singleton design patterns has made our Stock Prediction Application more flexible, maintainable, and scalable. These patterns simplify future upgrades, improve testing and collaboration, and ensure that the application can easily adapt to new features and technologies.

* Given the structure and complexity of our java spring application, we decided that there was no need for a microservice