# Microservices - Assignments

## Environment Setup Details:

Software to be installed and available on your local PC/Laptop. You can raise software request for the below software in TARMAC.

OpenJDK 17 or above

STS IDE Latest version 4.

Any database (MySQL/Oracle) OR any in-memory database (h2/hsql)

## General Instructions:

a. Use H2 in-memory database for the assignments wherever applicable.

b. Use Maven build tool/ Maven build based applications.

c. Create separate project for each of the assignments. You can use Spring Initializr to create projects ( Spring Initializr)

d. Capture screen shots of the output of the programs wherever applicable and submit it along with the solutions.

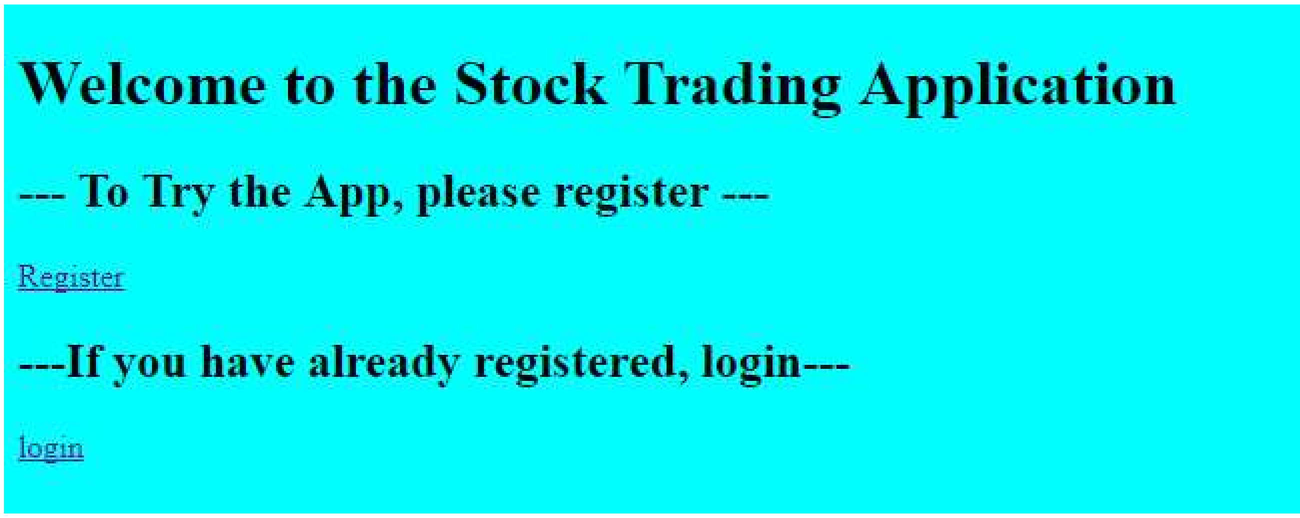
e. Submit only the src and pom.xml and supporting files(if any) (target, .settings, .project, .classpath NOT required) for each project.

## Assignments:

### 1. Assignment – 1:

Create a Spring Boot based Monolithic application for the 'Stock trading' app based on the requirements mentioned below:

a. The main page (the home page) looks like the below:



Welcome to the Stock Trading Application

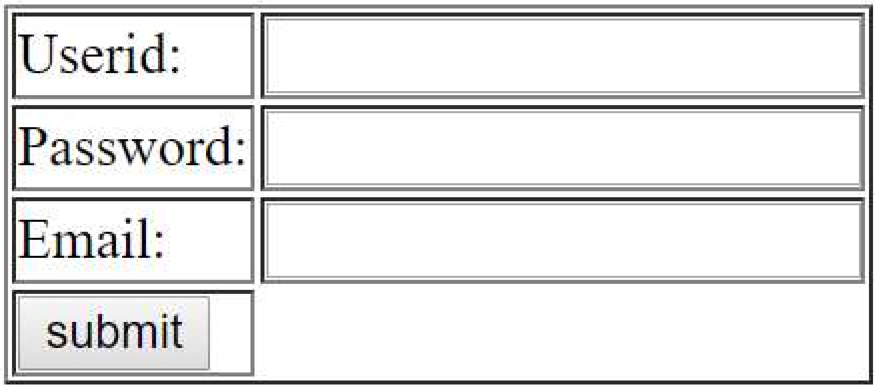
--- To Try the App, please register ---

Register

---If you have already registered, login---

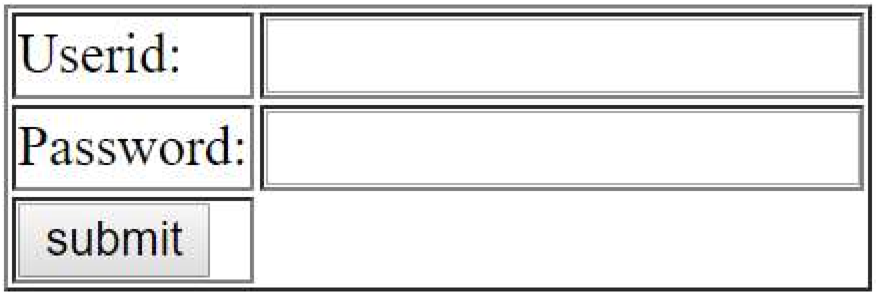
login

b. Click on “Register” link takes to the Register page as below. The user can register with the details mentioned. Upon successful registration, the details are persisted in database (eg. H2). Every registered user will also be given an initial balance of Rs. 1,00,000 for trading.



| | |
|---|---|
| Userid: | |
| Password: | |
| Email: | |
| submit | |

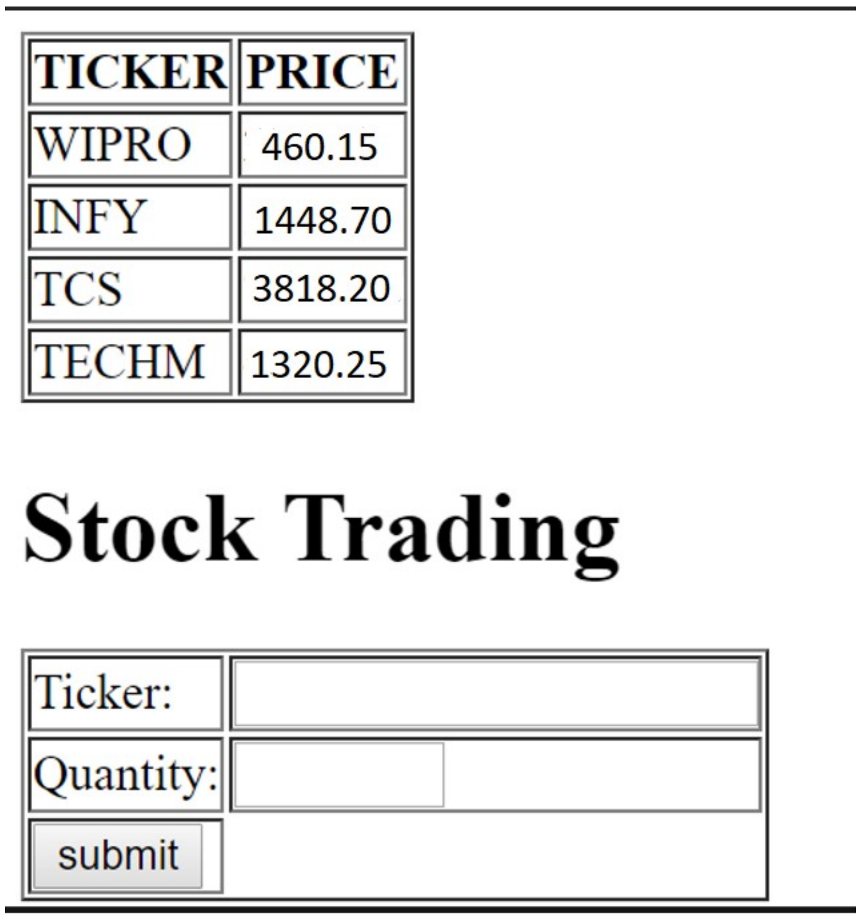
c. Once successfully registered, the user can now use the “login” link to login by providing the userid and password as below:



| | |
|---|---|
| Userid: | |
| Password: | |
| submit | |

d. Show appropriate error messages if the user tries to login with wrong userid or password.

e. On successful login, the user will be provided trading screen where the user can buy shares of one of the companies listed by specifying the 'ticker' symbol and the 'quantity' of the shares/stock that the user wants to buy. (like below). The quantity must be 5 to 25 in steps of 5.



| TICKER | PRICE |
|---|---|
| WIPRO | 460.15 |
| INFY | 1448.70 |
| TCS | 3818.20 |
| TECHM | 1320.25 |

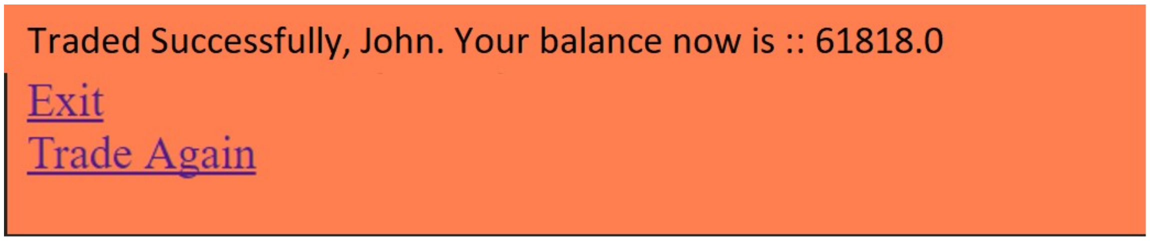
# Stock Trading

| | |
|---|---|
| Ticker: | |
| Quantity: | |
| submit | |

f. On submission of the above, the confirmation should be shown as below. And the balance amount also should be shown.

   Balance calculation is as below: If John has bought 10 shares of TCS, Total = 10 X 3818.20= 38182. Initial balance= 100000. So, remaining balance = 100000-38182 = 61818. If John trades again, balance will further reduce.

   The user is also free to Trade again or Exit from the application.



Traded Successfully, John. Your balance now is :: 61818.0

g. All the above services/functionalities need to implemented as RESTful Services

## 2. Assignment – 2:

Logically divide the monolithic application created in Assignment-1 into multiple Microservices (Hint: You can logically divide components/services/model classes into two Micro Services 'User Service' and 'Trade Service'.

User Service - takes care of registration and login
Trade Service - takes care of trading tasks.

Enable these microservices to communicate with each other using Eureka.

a. Configure the Spring Cloud Netflix Eureka Server which would act as the 'SR/SD'.
b. Configure the two Microservices to act as Eureka Clients.
c. Configure each of these Microservices to retrieve references from the Eureka Server and call the services/functionalities found within each other.

3. **Assignment-3**:

The previous assignment (Assignment-2) used “DiscoveryClient” to find the other microservice and communicate with each other. With this approach, manually you chose one of the microservices which were returned by the Eureka Server. Now instead, configure Client side load balancer called “Spring Cloud Load Balancer” and rewrite the Microservices to use ‘LoadBalancerClient’

4. **Assignment-4:**

After successful login, the user proceeds to trading. This is the place in your app, the communication happens between the microservices. If Trade microservice is not up, then this communication will fail. Enable Resilience4j Circuit Breaker and write fallback method to show some useful message to the end user.