|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Stock Market**  **FSE – Business Aligned Project**  **Case Study Specification**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Komila Kalra |  |  | | **Role** | Trainer |  |  | | **Signature** | t-komila3 |  |  | | **Date** | 25 October 2022 |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc116039845)

[2.0 Introduction 4](#_Toc116039846)

[2.1 Purpose of this document 4](#_Toc116039847)

[2.2 Project Overview 4](#_Toc116039848)

[2.3 Scope 4](#_Toc116039849)

[3.0 Use Case Diagram 6](#_Toc116039850)

[4.0 System Architecture Diagram 7](#_Toc116039851)

[5.0 Development Phases 7](#_Toc116039852)

[6.0 System Requirements 8](#_Toc116039853)

[**6.1.1** **Module – Stock Exchange** 8](#_Toc116039854)

[**6.1.2** **Module – Trader Management** 12](#_Toc116039855)

[**6.1.3** **Module – Order Management** 17](#_Toc116039856)

[**6.1.4** **Module – Risk Management** 21](#_Toc116039857)

[7.0 Deployment requirements 23](#_Toc116039858)

[8.0 Design Considerations 24](#_Toc116039859)

[8.1.1 Most Important and Common rules 24](#_Toc116039860)

[9.0 Reference learning 25](#_Toc116039861)

[10.0 Project Templates 26](#_Toc116039862)

[11.0 Change Log 27](#_Toc116039863)

# Important Instructions

1. Associate must adhere to the design considerations specific to each technology track.
2. Associate must not submit project with compile-time or build-time errors.
3. Being a full-stack developer project, you must focus on ALL layers of the application development.
4. Unit Testing is mandatory, and we expect a code coverage of 100%. Use unit testing and mocking frameworks wherever applicable.
5. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only.
6. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file.
7. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the microservices in cloud must be specified.

(Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)

1. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
2. You are supposed to use an in-memory database or code level data as specified, for the microservices that should be deployed in cloud. No physical database is suggested for microservice.

# Introduction

## Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Stock Market Management” that has to be developed. Both functional and non-functional requirements are captured in this document.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specified wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

## Project Overview

An online stock market system allows its users to register companies, do trading and risk management of stocks. It allows traders to create watch list and help them to buy or sell stocks

## Scope

Below are the modules that needs to be developed part of the Project:

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Stock Exchange module** | * This module is used by National Stock Exchange employee to register and unregister companies and update the stocks. * This module is also used by traders microservice if they want to fetch a latest stock price from stock exchange * Stock exchange will help users to create a watch list for several stock which they want to monitor * Register traders with stock exchange * Buying and selling of stocks |
| REQ\_02 | **Order** module | * This module is used for placing and processing orders for buying and selling stocks |
| REQ\_03 | **Risk management module** | * All the orders are passed through a Risk management system to ensure that order is safe to be entered into the market |
| REQ\_04 | **Trader module** | * The module will create a watchlist for stock which you want to observe over a period of time * User can set the instructions if he/she wants to buy a stock on market price or limit price. * User can set the instruction if he/she want to sell the stock on stoploss order or market price * Maintain a portfolio of stock sold and purchased by the user |

Table 1 : Application Modules

# Use Case Diagram

The following use case diagram shows various users of the system and their responsibilities.



Figure 1 : Use case diagram

# System Architecture Diagram

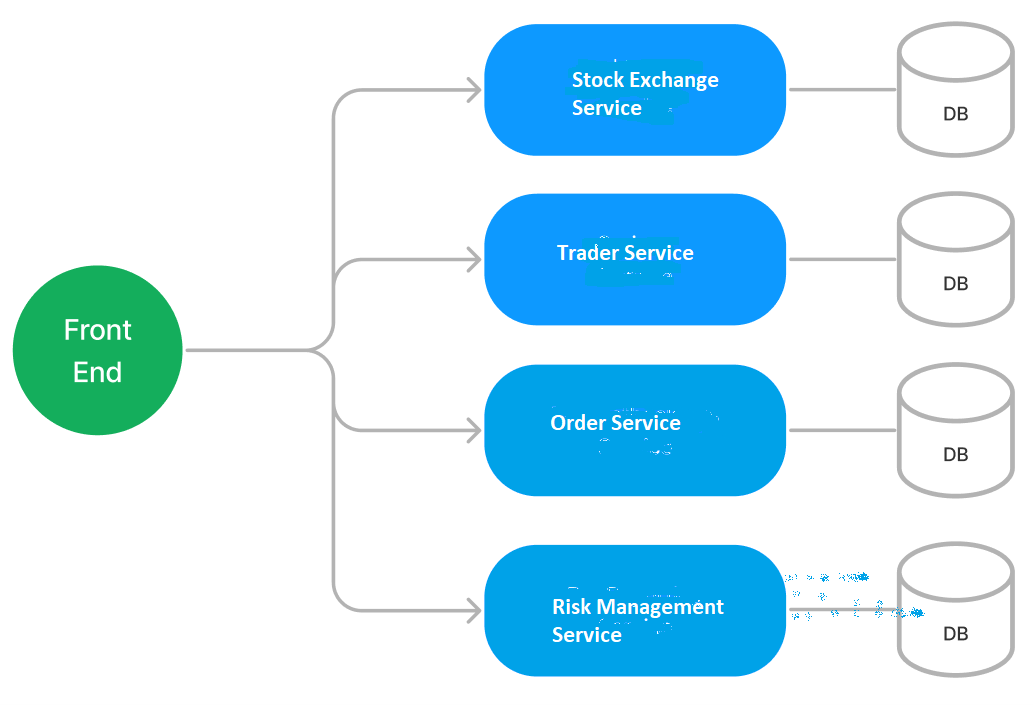


Figure 2 : Application Architecture Diagram

# Development Phases

* The application will be developed in 2 phases.
* Each phase will have 4 stages followed by a review at the end.
* The phase-1 output will be unit tested core business logic of the application.
* In phase-2 the output will be a functional application with micro-service and the font-end.
* Each stage of the development phase must be completed alongside the learning milestone

# System Requirements

### **Module – Stock Exchange**

The Stock exchange module will allow the stock exchange to register and unregister companies. The module will provide the following features.

1. Register and unregister companies
2. Allow companies to update the details
3. Register/unregister traders and allow them to update the details like balance
4. This module will help Trader microservice to pull the latest stock exchange data on bases of company type like Cement, IT, Telecom
5. This module will help Traders microservice to get a watch list. Watch list will give the last five working days stock price of a particular stock
6. Update the trading of stock as per the information received from Order microservice

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

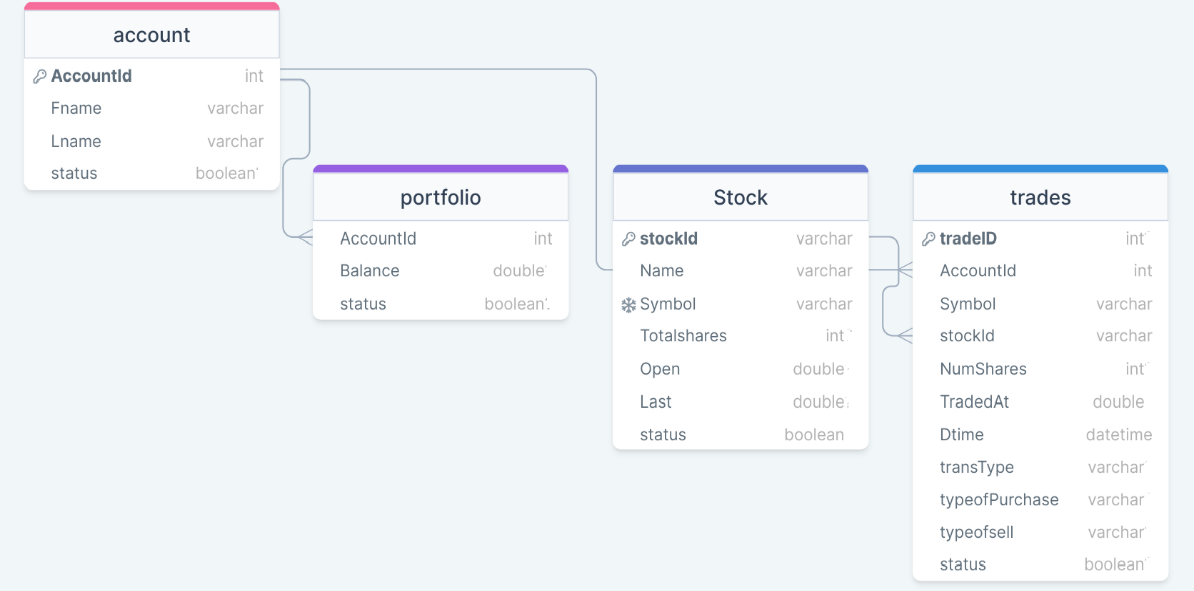


Figure 3 : ER Diagram – Stock Exchange

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. Symbol must be maximum 4 characters long
   2. Dtime cannot be less or greater than current date
   3. TotalShares, volume,open,last cannot be negative

Note: Add few clients on the application startup.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return list of stocks
   2. Register a new company
   3. Update an existing stock information
   4. Return a watch list
   5. Register new traders
   6. Update trader information
   7. Update the balance for the requested accountId.
   8. Register all trading information

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Return list of stocks
   2. Register a new company
   3. Update an existing stock information
   4. Return a watch list
   5. Register new traders
   6. Update trader information
   7. Register all trading information
5. Following business rules must be implemented as part of the business service class
   1. TraderId must be auto-generated. It should be in the format XY0000. First 2 letters of lastname followed by a 4 digit number
   2. HoldingId must be auto-generated. It should be in the format SYXY00. First 2 letters of symbol next two traderID followed by last two 2 digit number of year
   3. BoughtAt, numberofShares, balance cannot be negative
   4. Email address should always have @cognizant.com
   5. Dtime cannot be less or greater than current date
   6. TotalShares, open, last cannot be negative
   7. Trading of stock of de-registered company should not be allowed , in such case raise a user defined exception as “StockNotValid”
   8. Fname and lname should only have alphabets and last name must be minimum 3 characters long.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 2 : Stock Exchange - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/companies/type |
| **Request Type** | Get |
| **User Role** | Stock Exchange employee |
| **Trigger** | Front end |
| **Description** | Using this endpoint we can get a list of stocks of a particular type like construction, IT, FMGC |
| **Inputs** | Type |
| **Outputs** | StockDTOs |

Table 3: Stock Exchange - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/companies/<stockId>/update |
| **Request Type** | PUT |
| **User Role** | Stock exchange employee |
| **Trigger** | Back end |
| **Description** | With the help of this endpoint the user will be able to change the information of company |
| **Inputs** | StockID, StockDTO |
| **Outputs** | Status code |

Table 4 : Stock Exchange - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/companies/new |
| **Request Type** | POST |
| **User Role** | Stock exchange employee |
| **Trigger** | Frontend |
| **Description** | With the help of this endpoint the user will be able to register new stock in the stock exchange |
| **Inputs** | StockDTO |
| **Outputs** | Status code |

Table 5: Stock Exchange - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/trades |
| **Request Type** | GET |
| **User Role** | Trader |
| **Trigger** | Trader microservice |
| **Description** | This endpoint will allow the trader to pull trade information for last 5 days |
| **Inputs** |  |
| **Outputs** | TradeDTOs |

Table 6: Stock Exchange - Endpoint - 6

|  |  |
| --- | --- |
| **URL** | /api/stocktraders/<accountId>/update |
| **Request Type** | PUT |
| **User Role** | Stock exchange employee |
| **Trigger** | Back end |
| **Description** | With the help of this endpoint the user will be able to change the balance amount |
| **Inputs** | AccountID, TraderDTO |
| **Outputs** | Status code |

Table 7: Stock Exchange - Endpoint - 7

|  |  |
| --- | --- |
| **URL** | /api/stocktrader/new |
| **Request Type** | POST |
| **User Role** | Stock exchange employee |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the trader will be able to register him/her self to stock exchange |
| **Inputs** | TraderDTO |
| **Outputs** | Status code |

Table 8: Stock Exchange - Endpoint - 8

|  |  |
| --- | --- |
| **URL** | /api/stock/sell/MarketPlan |
| **Request Type** | PUT |
| **User Role** | Stock Exchange |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the stocks are sold at market price. |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

Table 9: Stock Exchange - Endpoint - 9

|  |  |
| --- | --- |
| **URL** | /api/stock/buy/MarketPlan |
| **Request Type** | PUT |
| **User Role** | Risk management |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the stocks are bought at market price. |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. CreateCompanyComponent
2. Create a component which will be used by employee of Stock Exchange to register the new company
3. The component should provide a form for the user
4. Once all the company details are validated, user should be able to submit the form and get an acknowledgement.

1. UpdateCompanyComponent
2. Design a add resource component and provide a navigation to it via navbar
3. The component must accept the resource details using the HTML5 form elements.
4. The role of user should be selected using the radio buttons
5. Use a dropdown list to choose a project
6. Before submitting the form ensure that all fields are validated
7. Once the form is submitted successfully, display an acknowledgement along with the newly generated user id
8. SearchCompanyComponent
9. Company information can be updated or viewed only by entering the company name or company id .We can approach the update screen only through search

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Trader management**

This module will provide various features to the trader. Below are the features provided by this module

1. Create a new watch list
2. Pull the stock exchange data from stock exchange
3. Buy/sell stocks
4. Register with stock exchange

**Stage: Database Imple mentation**

1. Design a data base as per the following ER diagram provided.

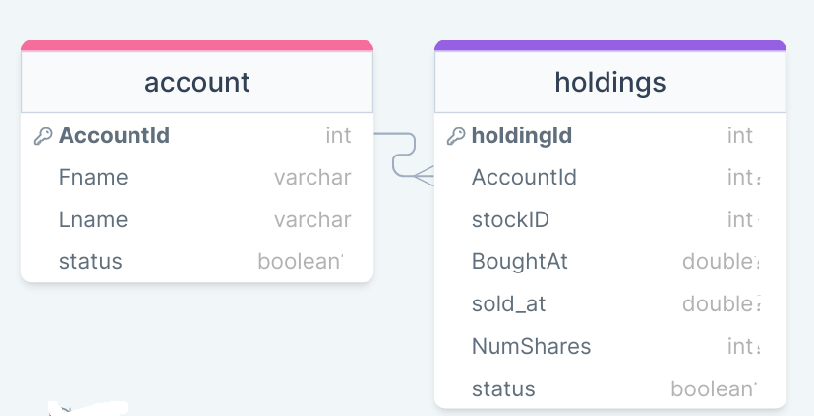


Figure 4 : ER Diagram – Trader management

1. Apply the following constraints apart from primary keys and foreign keys on the database
   1. BoughtAt, NumShares should be positive.
   2. Allowed values for status are – onhold/bought/canceled/sold

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Return list of particular stock for last 5 days for which we want to create a watch list
   2. Register the trader [Trader info is kept both at stock exchange and with trading system also]
   3. Buy new holdings
   4. Update existing stock holding
   5. Retrieve all the holdings by a particular trader i.e AccountId

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes using the single responsibility principle which perform the given operations as follows
   1. Return list of particular stock for last 5 days for which we want to create a watch list
   2. Register the trader [Trader info is kept both at stock exchange and with trading system also]
   3. Buy new holdings
   4. Update existing stock holding
   5. Retrieve all the holdings by a particular trader i.e AccountId
5. Following business rules must be implemented as part of the business service class
   1. We can create two watchlist per AccountID.
   2. We can buy new stock’s only if we have enough balance in the account.
   3. We cannot sell the stock if already sold or onhold or cancelled or if that accountId is not holding the mentioned stock in such case a user-defined exception is thrown as “StockAlreadySoldException” or “AcountIddon’towntheStock”
   4. Latest price of stock i.e entryPrice need to be picked up from Stock Exchange
   5. stopLoss cannot be negative
   6. Set NumOfShares to zero if typeOfPurchase is PositionSizing
   7. Set stopLoss to zero if typeOfsell is marketPrice
   8. RiskPerTrade should be positive value and it has to be less than stock price

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 10 : Trader management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/holdings/watchlist |
| **Request Type** | GET |
| **User Role** | Trader |
| **Trigger** | Backend |
| **Description** | This endpoint will be used to provide a list of last 5 days stock details of a particular stock |
| **Inputs** | stockId, AccountId |
| **Outputs** | StockDTOs |

Table 11: Trader management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/holdings/sell/MarketPlan |
| **Request Type** | PUT |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint order to sell the stock at market price is generated |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

Table 12: Trader management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/holdings/buy/marketPlan |
| **Request Type** | PUT |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint order to buy the stock at market price is generated |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

Table 13: Trader management - Endpoint - 4

|  |  |
| --- | --- |
| **URL** | /api/holdings/accountId |
| **Request Type** | Get |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Get all the stocks bought by a particular id |
| **Inputs** | accountId |
| **Outputs** | HoldingDTOs |

Table 14: Trader management - Endpoint - 5

|  |  |
| --- | --- |
| **URL** | /api/traders/new |
| **Request Type** | POST |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the trader can register him/her in stock exchange |
| **Inputs** | TraderDTO |
| **Outputs** | Status code along with stock name |

Table 15 : Trader management - Endpoint - 6

|  |  |
| --- | --- |
| **URL** | /api/traders/<accountId>/update |
| **Request Type** | PUT |
| **User Role** |  |
| **Trigger** | Back end |
| **Description** | With the help of this endpoint the user will be able to change the balance amount.This will further call the end point of stock exchange microservice |
| **Inputs** | AccountID, TraderDTO |
| **Outputs** | Status code |

Table 16: Trader management - Endpoint - 7

|  |  |
| --- | --- |
| **URL** | /api/holdings/positionSizing |
| **Request Type** | PUT |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the request is sent generate order for position sizing, i.e., he will buy number of stocks suggested by the risk management |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of unit’s purchased |

Table 17: Trader management - Endpoint - 8

|  |  |
| --- | --- |
| **URL** | /api/holdings/stopLoss |
| **Request Type** | PUT |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the request is sent to generate order to sell shares at stopLoss |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

Table 18 : Trader management - Endpoint - 9

|  |  |
| --- | --- |
| **URL** | /api/trades/pull |
| **Request Type** | Get |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint will call an endpoint of stock exchange to pull last 5 days trading information from stock exchange |
| **Inputs** |  |
| **Outputs** | TradeDTOs |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. AddNewTraderComponent
   1. Create a AddNewTrader component which can be navigated to from the menu of the application.
   2. The component will provide a form to accept trader details and save it into the system after validating all the details.

1. SearchStockComponent
   1. Search a stock on basis of stock id
   2. This component will accept the stockID in a textfield and will have a search button
2. UpdateTraderComponent
   1. Develop a component which is accessible from the menu bar for traders.
   2. The component should accept account ID in a textbox and a search button should fetch the information in form for editing purpose
3. WatchListComponent
   1. Develop a component which is accessible from the menu bar for traders
   2. This component should display a watch list which give last 5 days information about stock like opening price, closing price
4. AccountWiseHoldingInformationComponent
   1. Develop a component which is accessible from the menu bar for traders
   2. This component will accept the trader accountId and on bases of accountId it will give all sale and purchase information of stocks for that particular accountId
5. BuyStockComponent
   1. Develop a component which is accessible from the menu bar for traders
   2. This component will provide a form to accept the details of stock which user want to buy
   3. This component will have a dropdown with value MarketPlan/PositionSizing
   4. In case user select’s marketPlan numOfshares textfield will get enabled and for positionSizing numOfShares textfield will be disabled
   5. In case user select’s PositionSizing then RiskPerTrade and stopLoss textfield’s should be enabled. RiskPerTrade and stopLoss textfield’s will be disabled for MarketPlan

1. SellStockComponent
   1. Develop a component which is accessible from the menu bar for traders
   2. This component will provide a form to accept the details of stock which user want to sell
   3. This component will have a dropdown with value MarketPlan/StopLoss

.

1. ViewAllStocks
   1. Develop a component which is accessible from the menu bar for traders
   2. This component will show all the stock information which is pulled from stock exchange on the basis of stock type
2. SearchStockTypeComponent
   1. Search a stock on basis of stock type
   2. This component will accept the stockID in a textfield and will have a search button

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Order Management**

The order processing team will work with order management module. This module will provide various features:

1. An order management team will check the order with risk assessment microservice and accordingly take the decision to buy, sell or hold the stocks

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

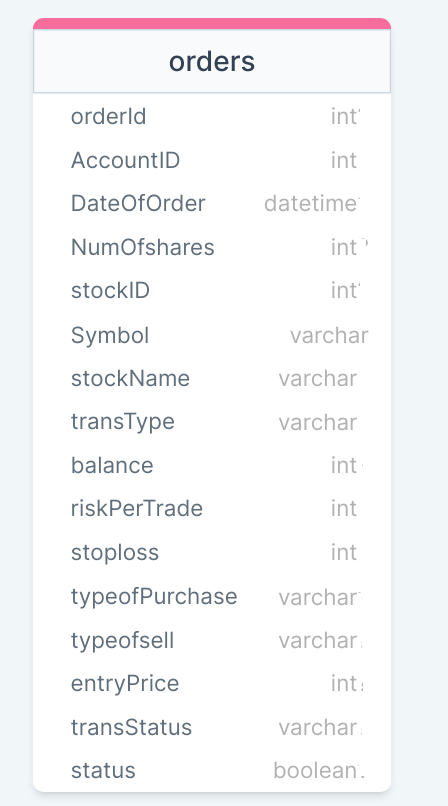


Figure 5 : ER Diagram - Product Backlog Management

1. Apart from primary and foreign keys implement the following additional constraints
   1. TransType should be either buy /sell.
   2. NumOfShares cannot be negative
   3. The values allowed for typeofPurchase are – marketPlan/positionSizing. The default value must be marketPlan.
   4. The values allowed for typeofSell are – marketPlan/stopLoss. The default value must be stopLoss.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
   1. Insert new Order
   2. Update order

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
   1. Insert new Order
   2. Update order
5. Following business rules must be implemented as part of the business service class
   1. stopLoss cannot be negative
   2. Set NumOfShares to zero if typeOfPurchase is PositionSizing
   3. Set stopLoss to zero if typeOfsell is marketPrice
6. Make a call Risk Management microservice to analyze the risk before trading a stock.
7. After the updates from risk management microservice stock exchange should be updated about the transaction

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All the exceptions must be handled and logged using a logging library.
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 19: Order management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/orders/sell/MarketPlan |
| **Request Type** | PUT |
| **User Role** | Risk management |
| **Trigger** | Front end |
| **Description** | Using this endpoint request to sell the stock at market price is send to stock exchange |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

**Table 20: Order management - Endpoint – 2**

|  |  |
| --- | --- |
| **URL** | /api/orders/buy/MarketPlan |
| **Request Type** | PUT |
| **User Role** | Risk management |
| **Trigger** | Front end |
| **Description** | Using this endpoint request to buy the stock at market price is send to stock exchange. |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

Table 21: Order management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/order/buy/positionSizing |
| **Request Type** | PUT |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the request is sent to risk management team |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of unit’s purchased |

**Table 22: Order management - Endpoint - 4**

|  |  |
| --- | --- |
| **URL** | /api/orders/sell/stopLoss |
| **Request Type** | PUT |
| **User Role** | Trader |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the request is sent to risk management team |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of units sold |

**Table 23: Order management - Endpoint - 5**

|  |  |
| --- | --- |
| **URL** | /api/orders/<orderId> |
| **Request Type** | Get |
| **User Role** | Order Management Team |
| **Trigger** | Front end |
| **Description** | Fetch the order on bases of orderID |
| **Inputs** | OrderId |
| **Outputs** | OderDTO |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. UpdateOrderComponent
2. This component will be used to process the order which have opted for StopLoss or PositionSizing
3. Every record will have a check button . On click of check button RiskMicroservice i.e server will check whether the order meet the criteria. In case it meets the criteria order will be processed else would be kept on hold

1. SearchComponent
   1. Develop a component which is accessible from the menu bar for Order management team.
   2. The component should accept order ID in a textbox and a search button should fetch the information in form for editing purpose
2. ViewOrderComponent
   1. This component will display either selected order or all orders

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

### **Module – Risk management**

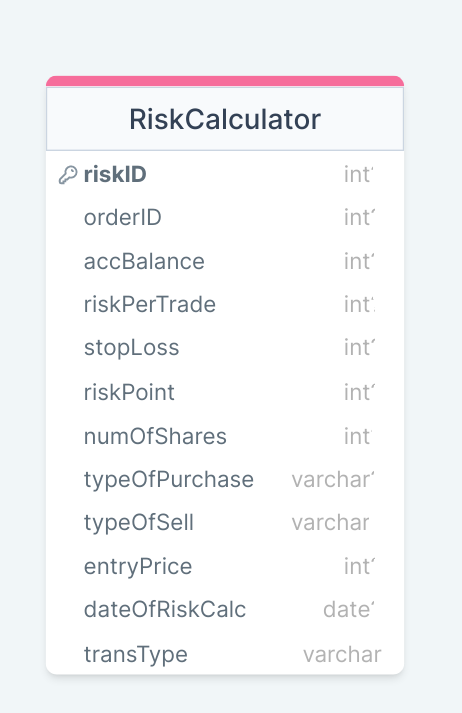
All stocks which have opted for positionSizing or stopLoss have to pass through risk management. This module will help the trader to minimize the risk.

1. In case trader has opted for PositionSizing for a particular stock. Risk Management team will help him to analyze how many stocks he/she should buy to keep the risk minimal
2. In case trader has opted for StopLoss option for selling a stock . Risk management team will help him/her to sell the stock before it goes below the stopLoss.

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

**Stage: Data Access Layer Design**



1. Apart from primary and foreign keys implement the following additional constraints
   1. TransType should be either buy /sell.
   2. RiskId must be auto-generated. It should be in the format XY0000. First two characters “RI” followed by a 4-digit number
   3. NumOfShares cannot be negative
   4. The values allowed for typeofPurchase are – marketPlan/positionSizing. The default value must be marketPlan.
   5. The values allowed for typeofSell are – marketPlan/stopLoss. The default value must be stopLoss.

**Stage: Business Logic Layer Development**

1. Following business rules must be implemented as part of the business service class

a. Position sizing i.e Number of unit which trader should buy will be calculated in following way

* + 1. Let’s assume account balance = 3lakh
    2. RiskPerTrade = 3000
    3. PricePerStock = 100
    4. Stop Loss =95
    5. RiskPoint = PricePerStock – StopLoss = 100-95 = 5 . It means RiskPoint is 5 point Risk
    6. NumOfShares = RiskPerTrade/RiskPoint = 3000/5 = 600
  1. Use the following logic if trader opted for MarketPlan
     1. Buy or sell the units at last market price. Here we will buy number of unit’s specified by the trader at given marketprice
  2. StopLoss mean’s the loss which trader can bear. If stock prices are going down . If trader is willing to sell the stock, he /she will sell the stocks at stopLoss price.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. Create the following end-points and test them using postman and export the requests into a json file.

Table 24: Order management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/risk/positionSizing |
| **Request Type** | POST |
| **User Role** | Risk management |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the risk management team will check how many units’ trader should buy and process the buying process in stock exchange |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of unit’s purchased |

Table 25: Order management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/risk/stopLoss |
| **Request Type** | POST |
| **User Role** | Risk management |
| **Trigger** | Front end |
| **Description** | Using this endpoint, the risk management team will check at what minimum price trader should sell the stocks. In case stock price is equal or greater than stopLoss risk management team will start selling process in stock exchange |
| **Inputs** | OrderDTO |
| **Outputs** | Status code with number of unit’s sold |

**Stage: Font-end design**

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

# Deployment requirements

1. All the Microservices must be deployed on a local web server like IIS or Apache Tomcat
2. All the Microservices must be independently deployable.
3. These services must be consumed from a front-end app running in a local environment.

# Design Considerations

Java and Dotnet specific design considerations are attached here. These design specifications, technology features have to be strictly adhered to.



Refer this link for the coding standards. <https://cognizantonline.sharepoint.com/:w:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/Coding%20standards/Effective%20coding%20standards.docx?d=w6430574d9db5478bbbe37c25b16e68e2&csf=1&web=1&e=84lTVf>

### Most Important and Common rules

|  |  |
| --- | --- |
| **Category** | **Rule** |
| Database | Table names in database must be pascal cased and plural. All primary keys must be named as Pk\_<table>. All foreign keys must be named as FK\_<PrimaryKeyTable>\_<ForeignKeyTable> |
| Database | Column names must be pascal cased. Multi-word column must be split using \_ (underscore) |
| Coding | Follow pascal casing for naming classes, interfaces, methods, properties and other public members |
| Coding | Use camel casing for method parameter name, backing fields for properties and private variables. Consts must be capitalized |
| Coding | All exceptions must be handled and logged using a logging library |
| Coding | For communication between micro-services use the HttpClient class available in .Net and Java |
| Unit testing | Each method in services classes in business logic must be unit tested using nUnit/jUnit |
| Unit testing | Use a mocking library to mock the repositories while performing tests for business logic layer |
| Code Coverage | Should be minimum 90% |
| Front-end(Angular/React ONLY) | Use pascal casing for the component names |
| Front-end(Angular/React ONLY) | Create all components and data services in Angular/React project in dedicated folders |
| GitHub | Create ONLY Private Repositories.  No password should be stored.  DO NOT Mention in the Profile that You work for Cognizant |

Any deviation from the high level design must be approved by trainer, mentor and solutions team

# Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**Other References:**

|  |  |
| --- | --- |
| Java 8 Parallel Programming | <https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8> |
| Feign client | [https://dzone.com/articles/Microservices-communication-feign-as-rest-client](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Swagger (Optional) | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](https://dzone.com/articles/centralized-documentation-in-microservice-spring-b) |
| ECL Emma Code Coverage | <https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php> |
| Lombok Logging | <https://javabydeveloper.com/lombok-slf4j-examples/> |
| Spring Security | <https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world> |
| H2 In-memory Database | <https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot>  <https://www.baeldung.com/spring-boot-h2-database> |
| AppInsights logging | <https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights> |
| Error response in WebApi | <https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api> |
| Read content from CSV | <https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects> |
| Access app settings key from appSettings.json in .Net core application | <https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1> |

# Project Templates









# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on 29-October-2022 by Komila Kalra | | | |
| V1.0.1 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
|  |  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |