# Cognizant Academy

# Portfolio Management System

# FSE – Business Aligned Project Case Study Specification

## Version 1.0

| | Prepared By / Last Updated By | Reviewed By | Approved By |
|---|---|---|---|
| **Name** | Ramasubramanian B | | |
| **Role** | Full Stack Trainer | | |
| **Signature** | | | |
| **Date** | | | |

# Table of Contents

## 1.0 Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technolgy 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 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)
8. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable
9. You are supposed to use an In-memory database or sessions as specified, for the Microservices that will be deployed in cloud. No Physical database is suggested.

# 2.0 Introduction

## 2.1 Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Portfolio Management System” that has to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

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 specificed wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

## 2.2 Project Overview

A leading Financial Services Organization wants to strengthen its Middleware by exposing the core logic related to Portfolio Management as Microservices. This middle ware Microservices will be hosted on Cloud so that all the up/downstream applications can get an access to this for performing business transactions.

There will also be a customer Portal to be developed part of this scope that consumes these Microservices to view their portfolio information and sell their assets.

## 2.3 Scope

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

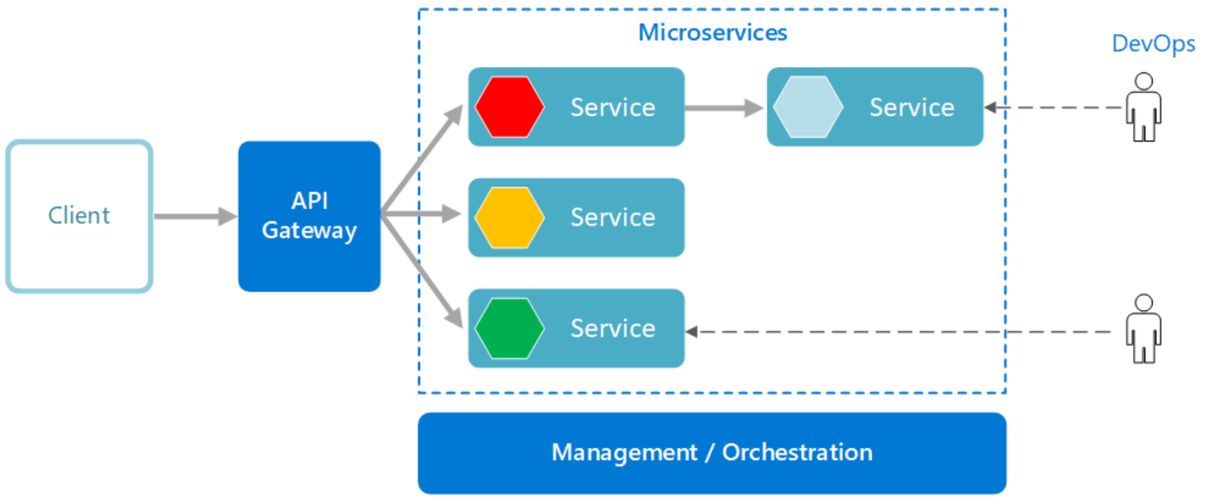
| Req. No. | Req. Name | Req. Description |
|---|---|---|
| REQ_01 | Daily Share Price Module | Daily Share Price Module is a Middleware Microservice that performs following operation:<br><br>• Get daily share price of a stock |
| REQ_02 | Daily Mutual Fund NAV | Daily Mutual Fund NAV Module is a Middleware Microservice that performs the following operation<br><br>• Get NAV value of a Mutual Fund |
| REQ_03 | Calculate Net worth Module | Calculate Net worth Module is a Middleware Microservice that performs the following operations:<br><br>• Calculate the current value of share holdings and mutual fund holdings and find out the total current value or net-worth<br><br>• Sell Assets and determine the final net-worth |

| | | |
|---|---|---|
| REQ_04 | Customer Portal | An Web Portal that allows a customer to Login and allows to do following operations:<br>• Login<br>• View the portfolio holdings & networth<br>• Sell assets |

## 2.4 Hardware and Software Requirement

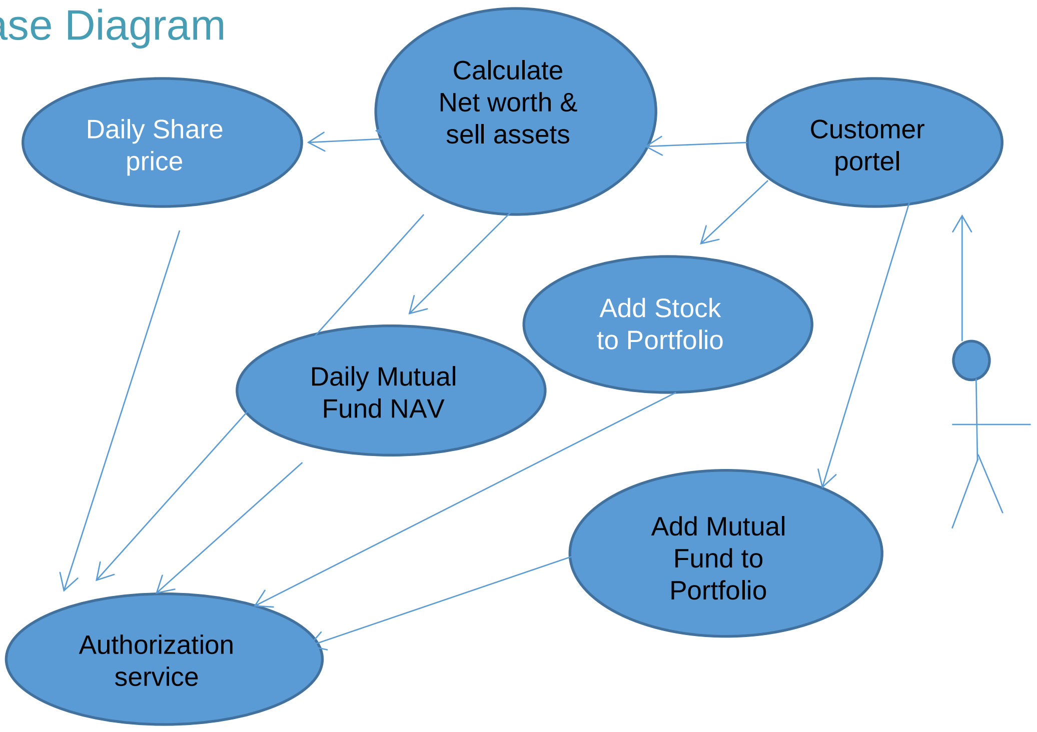
1. Hardware Requirement:
    a. Developer Desktop PC with 8GB RAM
2. Software Requirement (Java)
    a. Spring Tool Suite (STS) Or any Latest Eclipse
        i. Have PMD Plugin, EclEmma Code Coverage Plugin and AWS Code Commit Enabled
        ii. Configure Maven in Eclipse
    b. Maven
    c. Docker (Optional)
    d. Postman Client in Chrome
3. Software Requirement (Dotnet)
    a. Visual studio 2017 enterprise edition
    b. SQL Server 2014
    c. Postman Client in Chrome
    d. Azure cloud access

### 2.5 System Architecture Diagram



# 3.0 Functional Requirements and High Level Design

### 3.1 Use Case Diagram



## 3.2 Individual Components of the System

### 3.2.1 Calculate Net worth Microservice

| **Porfolio Management System** | **Calculate Networth Microservice** |
|---|---|

**Functional Requirements**

Can assume that Customer Portal App is the only client to this Microservice. An authorized customer view the net worth of his or her investment portfolio.

Post Authorization, it communicates to the below Microservices for retrieving necessary information.

Calculate Net worth Microservice will retrieve the stocks and mutual funds corresponds to the customer portfolio. For each stock it will interact with the Daily share price Microservice to get current market price for that stock and calculate the stock value by multiplying no of shares and current market price of the share. It will sum all the stock value.

The Calculate Networth microservice, for each mutual fund it will interact with Daily Mutual fund NAV Module, to get NAV value of of each mutual fund. It then calculates mutual fund value by multiplying number of units and NAV.

The networth is calculated as below

Networth = [number of shares of Stock1 * current price of stock1+ number of shares of Stock2 * current price of stock2+… + number of shares of StockN * current price of stockN]+ [number of units of Mutual Fund1 * NAV of Mutual Fund1+ number of units of Mutual Fund2 * NAV of Mutual Fund2+… +number of units of Mutual FundN * NAV of Mutual FundN]

This Microservice has another feature to sell the assets. The requested assets to sell should be sent thru the Portfolio Details business entity along with the current portfolio detail. Upon selling(out of the current requirement scope), the final networth should be calculated and returned to the web portal.

**Entities**

1. **Portfolio Details**

   a. **PortfolioId** - it is a unique id representing the portfolio
   b. **StockList** - These are the stocks hold by the customer. Each stock has details which is described below
   c. **MutualFundList** - These are the Mutual Funds hold by the customer. Each Mutual Fund has details described below

2. **Stock Details**

   a. **StockName** - Name of the stock
   b. **StockCount** - The number of shares of that stock held by the customer

3. **Mutual Fund Details**

   a. **MutualFundName** - Name of the Mutual Fund
   b. **MutualFundUnits** - The number of Mutual Fund units hold by the customer

4. **AssetSaleResponse**
    a. **SaleStatus** – True/False
    b. **Networth – balance**

**REST End Points**

**Calculate Net worth Microservice**

- GET: /calculateNetworth (Input: PortfolioDetails | Output: Networth – the computed total amount worth in INR)
- POST: /sellAssets (Input: currentDetail : PortfolioDetails, saleDetail : PortfolioDetails | AssetSaleReponse – Sale response status and the balance amount)

**Trigger** – Can be invoked from customer Portal (local MVC app)

**Steps and Actions**

1. Customer Portal will request for the portfolio networth of a customer to display his or her networth
2. The web portal should fetch the Portfolio details of the customer and send that as input to this Microservice.
3. The calculate networth microservice will interact with
    - daily share price microservice to get the current market price of each stock and multiply number of share with current market price.
    - daily mutual fund NAV microservice to find the NAV value of each mutual fund and multiply the number mutual fund units with NAV
4. The calculate networth microservice then sum up all the values to come up with portfolio networth
5. Upon viewing the Networth, there can be a request from the web portal to sell some of the assets. This microservice should be invoked to sell the requested assets. The sale response containing the sale status with the final networth should be returned to the web portal

**Non-Functional Requirement:**

- Only Authorized Customer can access these REST End Points

### 3.2.2 Daily Share Price Microservice

| Porfolio Management System | Daily Share Price Microservice |
|---|---|

**Functional Requirements**

Calculate Networth Microservice interacts with Daily Share Price Microservice. Post authorization of request, Daily Share Price Microservice allows the following operations:

To fetch the current market price of a stock:

- Retrieve the current market price from database and return

**Entities**

1. **Stock Details**

   Stock Id - unique id representing the stock
   Stock Name - Name of the stock
   Stock Value - Market price in INR

**REST End Points**

**Daily Share price Microservice**

a. GET: /dailySharePrice (Input: stockName | Output: StockDetail)

**Trigger** – Can be invoked from Calculate Net worth Microservice

**Steps and Actions**

1. Daily share price Microservice will have 1 End Point exposed to provide the stock market price
2. When /dailySharePrice end point is invoked by Calculate Networth Microservice, the daily share price Microservice will fetch the current market price from database and will return it to Calculate Networth Microservice.

### 3.2.3 Daily Mutual Fund NAV Microservice

| Portfolio Management System | Daily Mutual Fund NAV Microservice |
|---|---|

**Functional Requirements**

Calculate Net worth Microservice interacts with Daily Mutual Fund NAV Microservice. Post authorization of request, Daily Mutual Fund NAV Microservice does the following operation:

To fetch the Net Asset Calue (NAV) of of a mutual fund unit:

- Retrieve the NAV from database and return

**Entities**

1. **Mutual Fund Details**

   Mutual Fund Id - It is a unique id representing the Mutual Fund
   Mutual Fund Name - Name of the Mutual Fund
   Mutual Fund Value - NAV of Mutual Fund

**REST End Points**

**Daily Mutual Fund NAV Microservice**

- GET: /mutualFundNav (Input: Mutual Fund Name| Output MutualFundDetail)

|  |
|---|
| **Trigger** – Can be invoked from Calculate Networth Microservice |
| **Steps and Actions**<br><br>1. Daily Mutual Fund NAV Microservice will have one End Point exposed to provide<br>o The market value of the mutual fund NAV |

### 3.2.4 Authorization Microservice

| Portfolio Management System | Authorization Microservice |
|---|---|
| **Security Requirements**<br>o Service to Service communication has to happen using JWT<br>o Pass End User Context across Microservices<br>o Have the token expired after specific amount of time say 15 minutes.<br>o Have this service configured in the cloud along with other services | |

### 3.2.5 Swagger

| Portfolio Management System | Swagger |
|---|---|
| **Documentation Requirements**<br>o All the Microservices must be configured with Swagger for documentation<br><br>**Java implementation**<br>o Register the swagger resources in the Swagger Microservice and enable them as REST end points<br>o Configure this service along with other services in the cloud | |

### 3.2.6 Customer Portal (MVC)

| Portfolio Management System | Customer Portal |
|---|---|
| **Customer Portal Requirements**<br>o Customer Portal must allow a customer to Login. Once successfully logged in, the customer do the following operation:<br>o View the networth of his or her investment portfolio. | |

    - An option to sell existing assets – Stocks or Mutual funds. Upon selling assets, the final networth value should be calculated and displayed on the UI
    - The functionality of buying stocks/mutual funds is not in the scope of this requirement. The database for the web portal can be filled with portfolio details of few users.
- The above operations will reach out to the middleware Microservices that are hosted in cloud.

## 4.0 Cloud Deployment requirements

- All the Microservices must be deployed in Cloud
- All the Microservices must be independently deployable. They have to use In-memory database or user sessions wherever applicable
- The Microservices has to be dockerized and these containers must be hosted in Cloud using CI/CD pipelines
- The containers have to be orchestrated using AWS/Azure Kubernetes Services.
- These services must be consumed from an MVC app running in a local environment.

## 5.0 Design Considerations

These design specifications, technology features have to be strictly adhered to.

CDE-Project-Design Considerations.pptx

## 6.0 Reference learning

Please go through all of these k-point videos for Microservices deployment into AWS.

| |
|---|
| https://cognizant.kpoint.com/app/video/gcc-6e36500f-c1af-42c1-a6c7-ed8aac53ab22 |
| https://cognizant.kpoint.com/app/video/gcc-92f246c9-024a-40b7-8bfc-96b3ce7c1a39 |
| https://cognizant.kpoint.com/app/video/gcc-cfedd9c1-e29e-4e3e-b3e2-1960277f72a3 |
| https://cognizant.kpoint.com/app/video/gcc-900a7172-43b7-42f3-a6cc-e301bd9cc9b3 |

**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 |
| Swagger (Optional) | 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<br>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 .Netcore application | https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/<br><br>https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1 |

## 7.0 Change Log

| | Changes Made | | | |
|---|---|---|---|---|
| V1.0.0 | Initial baseline created on <28-Jul-2020> by <Ramasubramanian B> | | | |
| | | | | |
| | **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| | | | | |