**1. Overview**

The Investment Portfolio Management System is a web-based application that enables financial

institutions and individual investors to manage and monitor their investment portfolios. It provides

functionality for portfolio creation, asset allocation, risk assessment, performance tracking, and reporting.

This system is designed using the MVC architecture, making it compatible with both Java (Spring MVC) and .NET (ASP.NET Core MVC) frameworks.

**2. Core Modules Description**

The system is divided into five primary modules, each addressing a critical aspect of investment portfolio

management.

**2.1 User Management**

This module handles user account creation, role assignment (Investor, Admin), and profile

management.

**2.2 Portfolio Management**

This module allows users to create and manage investment portfolios, including adding and modifying assets like stocks, bonds, and mutual funds.

**2.3 Risk Assessment**

This module calculates the risk level of portfolios based on asset allocation, market trends, and user-defined parameters.

**2.4 Performance Tracking**

This module monitors the performance of assets and portfolios, providing historical and real-time data.

**2.5 Reporting and Analytics**

This module generates reports and analytics for users and administrators to make informed decisions.

**3. Module-Level Design**

**3.1 User Management Module**

Purpose: Handles user registration, authentication, and profile management.

• **Controller:**

o UserController

▪ registerUser()

▪ loginUser()

▪ updateUserProfile()

• Service:

o UserService

▪ Validates user details and manages authentication.

**• Model:**

o User Entity

▪ Attributes:

▪ userId (PK)

▪ username

▪ password

▪ email

▪ role (Investor, Admin)

**3.2 Portfolio Management Module**

Purpose: Enables users to manage investment portfolios.

• Controller:

o PortfolioController

▪ createPortfolio()

▪ addAssetToPortfolio()

▪ viewPortfolioDetails()

• Service:

o PortfolioService

▪ Handles portfolio creation and asset allocation logic.

• Model:

o Portfolio Entity

▪ Attributes:

▪ portfolioId (PK)

▪ userId (FK)

▪ portfolioName

▪ creationDate

o Asset Entity

▪ Attributes:

▪ assetId (PK)

▪ portfolioId (FK)

▪ assetType (Stock, Bond, Mutual Fund)

▪ assetName

▪ quantity

▪ purchasePrice

**3.3 Risk Assessment Module**

Purpose: Assesses the risk level of portfolios based on asset allocation.

• Controller:

o RiskController

▪ calculateRisk()

▪ viewRiskAnalysis()

• Service:

o RiskService

▪ Implements algorithms for risk calculation.

• Model:

o Risk Entity

▪ Attributes:

▪ riskId (PK)

▪ portfolioId (FK)

▪ riskLevel (Low, Medium, High)

▪ analysisDate

**3.4 Performance Tracking Module**

Purpose: Tracks the performance of assets and portfolios.

• Controller:

o PerformanceController

▪ getAssetPerformance()

▪ getPortfolioPerformance()

• Service:

o PerformanceService

▪ Aggregates data from market APIs for performance tracking.

• Model:

o Performance Entity

▪ Attributes:

▪ performanceId (PK)

▪ assetId (FK)

▪ currentValue

▪ profitLoss

▪ lastUpdated

**3.5 Reporting and Analytics Module**

Purpose: Provides insights into portfolio performance and risk.

• Controller:

o ReportController

▪ generatePortfolioReport()

▪ generateRiskReport()

▪ generateAssetAnalysisReport()

• Service:

o ReportService

▪ Generates detailed reports and analytics.

• Model:

o Report Entity

▪ Attributes:

▪ reportId (PK)

▪ reportType (Portfolio, Risk, Asset)

▪ generatedDate

**4. Database Schema**

**4.1 Table Definitions**

**1. User Table**

CREATE TABLE User\_portfolio (

userId INT IDENTITY(1,1) PRIMARY KEY,

username VARCHAR(100),

password VARCHAR(100),

email VARCHAR(100),

role VARCHAR(100) CHECK (role IN ('Investor', 'Admin'))

);

**2. Portfolio Table**

CREATE TABLE Portfolio (

portfolioId INT IDENTITY(1,1) PRIMARY KEY,

userId INT,

portfolioName VARCHAR(100),

creationDate DATE,

FOREIGN KEY (userId) REFERENCES User\_portfolio(userId)

);

**3. Asset Table**

CREATE TABLE Asset (

assetId INT IDENTITY(1,1) PRIMARY KEY,

portfolioId INT,

assetType VARCHAR(50) CHECK (assetType IN ('Stock', 'Bond', 'Mutual Fund')),

assetName VARCHAR(100),

quantity INT,

purchasePrice DECIMAL(10, 2),

FOREIGN KEY (portfolioId) REFERENCES Portfolio(portfolioId)

);

**4. Risk Table**

CREATE TABLE Risk (

riskId INT IDENTITY(1,1) PRIMARY KEY,

portfolioId INT,

riskLevel VARCHAR(50) CHECK (riskLevel IN ('Low', 'Medium', 'High')),

analysisDate DATE,

FOREIGN KEY (portfolioId) REFERENCES Portfolio(portfolioId)

);

**5. Performance Table**

CREATE TABLE Performance (

performanceId INT IDENTITY(1,1) PRIMARY KEY,

assetId INT,

currentValue DECIMAL(10, 2),

profitLoss DECIMAL(10, 2),

lastUpdated DATE,

FOREIGN KEY (assetId) REFERENCES Asset(assetId)

);

**6. Report Table**

CREATE TABLE Report (

reportId INT AUTO\_INCREMENT PRIMARY KEY,

reportType ENUM('Portfolio', 'Risk', 'Asset'),

generatedDate DATE

);

**5. Local Deployment Details**

1. Environment Setup:

o Install MySQL or SQL Server for database management.

o Install Java (JDK) or .NET SDK (ASP.NET Core).

o Configure database connections in the application’s configuration file.

2. Deployment Steps:

o Clone the repository.

o Build the project using Maven (Java) or Visual Studio (ASP.NET Core).

o Run the application server.

o Access the application at http://localhost:8080 (Java) or http://localhost:5000 (.NET).

**Investment Portfolio Management System**

**DAO Model**

InvestmentPortfolioManagementDA\_LIB

**Entity Classes** - User / Portfolio / Asset / Risk / Performance / Report

**Data Access Classes** - UserDARepository, PortfolioDARepository, AssetDARepository, RiskDARepository, PerformanceDARepository, ReportDARepository

These repositories use ADO.NET syntax (or equivalent ORM if chosen, like Entity Framework Core for .NET or JPA/Hibernate for Java) to interact with a SQL Server (or MySQL as specified) database. They would contain methods for CRUD operations (Create, Read, Update, Delete) and other database-specific logic.

InvestmentPortfolioManagementBO\_Lib

**Entity Classes** - User / Portfolio / Asset / Risk / Performance / Report (These can be the same entity classes as in the DAO layer or separate DTO - Data Transfer Objects)

**BO Classes** - UserBO, PortfolioBO, AssetBO, RiskBO, PerformanceBO, ReportBO. These classes contain the business logic, validation, and orchestration of data access through the DAOs. For example, the PortfolioBO might use the PortfolioDARepository and AssetDARepository to create a portfolio and add assets to it.

ConsoleApp/WebApp - InvestmentPortfolioManagementApp