Project: Treasury Management System (TMS)

1. Overview

The **Treasury Management System (TMS)** is a web application designed for financial institutions and corporate organizations to streamline their treasury operations. It helps manage cash flow, investments, financial risk, and bank account operations. The system offers robust reporting and compliance features, ensuring adherence to financial regulations.

The project is built using **MVC architecture** and is compatible with both **Java (Spring MVC)** and **.NET (ASP.NET Core MVC)** frameworks for cross-technology adaptability.

2. Core Modules Description

2.1 Cash Flow Management

Facilitates real-time tracking of cash inflows and outflows, providing a clear picture of liquidity.

2.2 Investment Portfolio Management

Manages the organization's investment activities, including bonds, equities, and mutual funds.

2.3 Risk Management

Identifies and mitigates financial risks, such as currency fluctuations or interest rate risks.

2.4 Bank Account Management

Provides centralized control over multiple bank accounts, including reconciliation and transaction tracking.

2.5 Reporting and Compliance

Generates financial reports and ensures regulatory compliance with government and industry standards.

3. Module-Level Design

3.1 Cash Flow Management Module

Purpose: Tracks real-time cash inflows and outflows.

- **Controller:**
  - CashFlowController
    - recordTransaction()
    - getCashFlowReport()
- **Service:**
  - CashFlowService
    - Validates and records transactions in the database.
    - Generates cash flow summaries.

- **Model:**
  - **CashFlow Entity**
    - Attributes:
      - transactionId (PK)
      - amount
      - transactionType (e.g., inflow/outflow)
      - transactionDate
      - description

3.2 Investment Portfolio Management Module

Purpose: Tracks and manages organizational investments.

- **Controller:**
  - InvestmentController
    - addInvestment()
    - getPortfolioSummary()
- **Service:**
  - InvestmentService
    - Manages investment records and calculates portfolio performance.
- **Model:**
  - **Investment Entity**
    - Attributes:
      - investmentId (PK)
      - investmentType (e.g., bond, equity)
      - amountInvested
      - currentValue
      - purchaseDate
      - maturityDate

3.3 Risk Management Module

Purpose: Assesses and mitigates financial risks.

- **Controller:**
  - RiskManagementController
    - analyzeRisk()
    - getRiskScore()
- **Service:**
  - RiskManagementService
    - Implements algorithms for assessing risks and suggesting mitigation strategies.
- **Model:**
  - **RiskAssessment Entity**
    - Attributes:
      - riskId (PK)
      - riskType
      - transactionReference
      - riskScore
      - assessmentDate

3.4 Bank Account Management Module

Purpose: Manages operations across multiple bank accounts.

- **Controller:**
  - BankAccountController
    - addBankAccount()
    - reconcileAccount()
- **Service:**
  - BankAccountService
    - Tracks transactions and reconciles discrepancies in accounts.
- **Model:**
  - **BankAccount Entity**
    - Attributes:
      - accountId (PK)
      - bankName
      - accountNumber
      - accountType
      - balance

3.5 Reporting and Compliance Module

Purpose: Provides financial reports and ensures regulatory adherence.

- **Controller:**
  - ReportingController
    - generateReport()
    - submitComplianceReport()
- **Service:**
  - ReportingService
    - Generates and validates financial reports.
- **Model:**
  - **Report Entity**
    - Attributes:
      - reportId (PK)
      - reportType
      - generationDate
      - submittedBy
      - status

4. Database Schema

4.1 Table Definitions

1. **CashFlow Table**

```
CREATE TABLE CashFlow (
  transactionId INT AUTO_INCREMENT PRIMARY KEY,
  amount DECIMAL(15, 2),
  transactionType ENUM('Inflow', 'Outflow'),
  transactionDate DATE,
  description VARCHAR(255)
);
```

2. **Investment Table**

```
CREATE TABLE Investment (
  investmentId INT AUTO_INCREMENT PRIMARY KEY,
  investmentType VARCHAR(50),
  amountInvested DECIMAL(15, 2),
  currentValue DECIMAL(15, 2),
  purchaseDate DATE,
  maturityDate DATE
);
```

3. **RiskAssessment Table**

```
CREATE TABLE RiskAssessment (
  riskId INT AUTO_INCREMENT PRIMARY KEY,
  riskType VARCHAR(50),
  transactionReference VARCHAR(50),
  riskScore DECIMAL(5, 2),
  assessmentDate DATE
);
```

4. **BankAccount Table**

```
CREATE TABLE BankAccount (
  accountId INT AUTO_INCREMENT PRIMARY KEY,
  bankName VARCHAR(100),
  accountNumber VARCHAR(50),
  accountType ENUM('Checking', 'Savings'),
  balance DECIMAL(15, 2)
);
```

5. **Report Table**

```
CREATE TABLE Report (
  reportId INT AUTO_INCREMENT PRIMARY KEY,
  reportType VARCHAR(50),
  generationDate DATE,
  submittedBy VARCHAR(100),
  status ENUM('Pending', 'Submitted')
);
```

5. Local Deployment Details

1. **Environment Setup:**
    - Install **MySQL** or **SQL Server** for the database.
    - Configure application.properties (Java) or appsettings.json (.NET) with the database connection details.
    - Set up the environment with the required JDK or .NET SDK.
2. **Deployment Steps:**
    - Clone the repository from the version control system.
    - Build the application using Maven or Visual Studio.
    - Run the application locally and access it through http://localhost:<port>.

6. Conclusion

The **Treasury Management System** provides a robust platform for managing treasury operations, ensuring compliance, and streamlining financial processes. Its modular design makes it scalable and adaptable for large organizations.