# Requirement:

Bank X has defined a specification for a new Banking Application. In keeping up with the trends around the world, Bank X wants to be able to allow both internal and external systems to connect with the new application. The Application must allow new customers to be onboarded and to obtain new accounts. Each customer will be provided with a Current and Savings account upon onboarding and will have their Savings Account credited with a joining bonus of R500.00. Customers should be able to move money between their accounts. Only the Current Account should be enabled to make payments to other accounts. All payments made into the Savings Account will be credited with a 0.5% interest of the current balance. All payments made from the customer’s account will be charged 0.05% of the transaction amount. The application must keep track of every transaction performed on the accounts and allow other systems to retrieve these. The system must send out notifications to the customer for every transaction event. (Note: this is an offline process).

1. Create a Java Application that addresses the above business case.
2. The application should be able to run on its own.
3. Zip the code base and send over mail.

# Planning & Understanding:

### Step 1: Plan the Application

* **Core Features**:
  + Customer onboarding.
  + Account creation with Current and Savings accounts.
  + Account transactions (move money, payments).
  + Transaction history management.
  + Notifications for every transaction.
* **Key Classes**:
  + Customer
  + Account (with subclasses CurrentAccount and SavingsAccount)
  + Transaction
  + Notification
  + BankingSystem (main application)

### Step 2: Create the Project Structure

### Step 3: Implement the Core Models

1. **Customer**:
   * Attributes: id, name, email, phone, accounts.
2. **Account**:
   * Attributes: accountNumber, balance, transactions.
   * Subclasses: CurrentAccount, SavingsAccount.
3. **Transaction**:
   * Attributes: transactionId, type (e.g., CREDIT/DEBIT), amount, date, description.

### Step 4: Build Core Functionalities

1. **Customer Onboarding**:
   * Create a new customer and initialize their Current and Savings accounts.
   * Credit R500.00 to the Savings account as a joining bonus.
2. **Account Transactions**:
   * Implement methods for:
     + Transferring money between accounts.
     + Payments (debits with charges).
     + Crediting interest on payments into Savings.
3. **Transaction Management**:
   * Maintain a list of all transactions for an account.
   * Log each transaction in the account's history.
4. **Notification System**:
   * Create a NotificationService that sends out transaction notifications (simulated via console output).
5. **Interest and Charges**:
   * Deduct 0.05% on outgoing payments.
   * Add 0.5% interest for payments into the Savings account.

### Step 5: Add Offline Notifications

* Simulate notifications by storing messages or printing them to the console.

### Step 6: Create a Runnable Main Class

* Allow users to onboard, view accounts, make transactions, and view transaction history.

### Step 7: Zip and Share

* Package the project into a .zip file.

# ****Project Setup/Configuration:****

Basic Requirement:

1. Integrated Development Environment (IDE like Visual Code, Eclipse, etc.)
2. Java Development Kit (JDK 21 version)
3. MySQL Workbench 8.0 CE

Extract:

Download the zip and extract the folder in your favourite IDE open the **BankingApplication** folder.

Application.Properties Configuration:

I used **Maven**, as it’s widely used for Java projects and easy to manage dependencies.

Please note that in **application.properties** you have to change the

spring.jpa.hibernate.ddl-auto=update

in **line number 7** to **create** for the first time after that again change it back to **update**.

I used external **mySql** database and created the **bankx\_db schema**.

For connecting the database download the **mySql** and create the **schema** with the name **bankx\_db**.

**Link:** <https://dev.mysql.com/downloads/workbench/>

spring.datasource.url=jdbc:mysql://localhost:3306/bankx\_db

spring.datasource.name=bankx\_db

spring.datasource.username=root

spring.datasource.password=admin

Note change the **url, schema name, username, password** as you configured at the time of installation of mysql workbench.

For notification Service you must configure your email & password.

spring.mail.port=587

spring.mail.username=your\_email@example.com

spring.mail.password=your\_password

Running the application:

In terminal pass command

* mvn clean install
* mvn clean test

# Module Structure:

* Controller (handling incoming HTTP requests and sending responses)
* Services (core business logic of the application)
* Entity/Model (for data representation)
* Repository (for database interaction)
* Exception (for handling custom exceptions)
* Application.properties (configuration database configuration, loggings, profile, port, etc.)
* Pom.xml (Maven/Gradle dependencies configuration file)
* Logback.xml (logging configuration)
* Junit Test (for testing the application)

### ****1. Controller (For handling HTTP requests)****

* **Purpose**:
  + The Controller layer is responsible for handling incoming HTTP requests and sending responses.
  + It maps endpoints (URLs) to specific business logic in the service layer.
* **Example**:
  + GET /customers → Fetch all customers.
  + POST /customers → Add a new customer.
* **Spring Annotation**:
  + @RestController or @Controller

### ****2. Services (For business logic)****

* **Purpose**:
  + The Service layer contains the core business logic of the application.
  + It acts as an intermediary between the Controller and Repository layers.
  + Implements rules like:
    - Moving money between accounts.
    - Applying transaction charges.
    - Crediting interest for savings accounts.
* **Spring Annotation**:
  + @Service

### ****3. Entity/Model (For data representation)****

* **Purpose**:
  + Defines the structure of the application's data (e.g., Customer, Account, Transaction).
  + Used to map objects to database tables (if using JPA) or as simple data objects in the application.
* **Annotations**:
  + @Entity for JPA-based database mapping.
  + @Table for specifying table names.
  + @Id, @Column for defining fields.
* **Examples**:
  + Customer: id, name, email, phone.
  + Account: accountId, accountType, balance.

### ****4. Repository (For database interactions)****

* **Purpose**:
  + Handles the data storage and retrieval logic.
  + Interacts with the database (or in-memory storage).
  + Provides methods like save(), findById(), findAll().
* **Spring Annotation**:
  + @Repository
* **Implementation**:
  + Can use **Spring Data JPA** for easy CRUD operations.
  + For in-memory storage, you can use a HashMap.

### ****5.**** application.properties ****(Configuration settings)****

* **Purpose**:
  + Holds the configuration settings for the Spring application.
  + Examples:
    - Database connection details.
    - Logging levels.
    - Port configuration.
* **Examples**:

spring.datasource.url=jdbc:mysql://localhost:3306/bankx

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.jpa.hibernate.ddl-auto=update

server.port=8080

### ****6.**** pom.xml ****(Maven configuration file)****

* **Purpose**:
  + The pom.xml is the central Maven configuration file for managing:
    - Dependencies (e.g., Spring Boot, JPA, MySQL).
    - Plugins.
    - Project metadata.
* **Example**:

xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

### ****7.**** logback.xml ****(Logging configuration)****

* **Purpose**:
  + Configures the logging behavior of the application.
  + Allows you to specify:
    - Log levels (INFO, DEBUG, ERROR).
    - Log file output.
    - Formatting for logs.
* **Example**:

xml

<configuration>

<appender name="FILE" class="ch.qos.logback.core.FileAppender">

<file>logs/application.log</file>

<encoder>

<pattern>%d [%t] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="info">

<appender-ref ref="FILE"/>

</root>

</configuration>

### ****8.**** Junit Test ****(For testing the application)****

* **Purpose**:
  + Ensures that the application works as expected by testing individual components.
  + Tests:
    - Service methods.
    - Repository CRUD operations.
    - Controller endpoints (with MockMvc).
* **Annotations**:
  + @Test for test methods.
  + @BeforeEach for setup.
  + @SpringBootTest for full application context testing.

**Note:**

For using the email service you have to add the email into **application.properties** and **NotificationService.class**

All other test cases are performed though the **postman**.