### ****1.1 API Functional Requirements****

The API should:  
✅ Accept customer transaction details and store them in a database.  
✅ Support money transfers between accounts.  
✅ Allow users to retrieve and format account balances.  
✅ Perform basic data validation.  
✅ Accept and return JSON data.  
✅ Be containerized (Docker).  
✅ Use a message queue (RabbitMQ/Kafka).  
✅ Write logs to a file and an external logging system.  
✅ Implement authentication (JWT-based security).

* Create, read, update, and delete bank accounts
* Create, read, update, and delete bank transactions
* Authenticate and authorize users
* Validate account and transaction data
* Generate API documentation using Swagger

### ****1.2 High-Level Architecture****

The API will be a **Spring Boot** application with the following components:

#### ****1️⃣ API Layer (Controller)****

* Exposes RESTful endpoints for transactions, transfers, and balance retrieval.
* Accepts JSON input and returns JSON responses.

#### ****2️⃣ Service Layer****

* Contains business logic (e.g., validating transactions, calculating balances).

#### ****3️⃣ Repository Layer (Database)****

* Uses Spring Data JPA to interact with PostgreSQL/MySQL.

#### ****4️⃣ Message Queue****

* RabbitMQ/Kafka will handle asynchronous transaction processing.

#### ****5️⃣ Security****

* Uses JWT for authentication.

#### ****6️⃣ Logging****

* Logs important events to a file and an external system (e.g., ELK Stack).

### ****Understanding**** CustomerTransactionRepository

The **CustomerTransactionRepository** is a **Spring Data JPA repository** responsible for interacting with the **database** to perform operations like:

* Saving transactions
* Fetching transactions
* Updating transactions
* Deleting transactions

### ****1.3 API Endpoints Design****

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| **POST** | /transactions | Store a customer transaction |
| **POST** | /transfer | Transfer funds between accounts |
| **GET** | /balance/{accountId} | Retrieve and format an account balance |

## ****Step 2: Setting Up Spring Boot Project****

### ****2.1 Create a Spring Boot Project****

We will use [Spring Initializr](https://start.spring.io/" \t "_new) to generate the project.

✅ Dependencies to include:

* **Spring Web** (For REST API)
* **Spring Boot Starter Data JPA** (For database access)
* **Spring Boot Starter Security** (For authentication)
* **Spring Boot Starter Validation** (For request validation)
* **Spring Boot Starter Logging** (For file logging)
* **Spring Boot Starter AMQP** (For RabbitMQ integration)
* **Spring Boot DevTools** (For development convenience)
* **PostgreSQL Driver** (Or MySQL based on preference)

### ****2.2 Project Folder Structure****

/transaction-api

├── src/main/java/com/example/transactionapi

│ ├── controller/ # API Endpoints

│ ├── service/ # Business Logic

│ ├── repository/ # Database Access

│ ├── model/ # Data Models

│ ├── security/ # JWT Authentication

│ ├── queue/ # Message Queue Handling

│ ├── config/ # Configuration Files

│ ├── TransactionApiApplication.java # Main Application

├── src/main/resources/

│ ├── application.yml # Configuration

The **CustomerTransactionRepository** is a **Spring Data JPA repository** responsible for interacting with the **database** to perform operations like:

* Saving transactions
* Fetching transactions
* Updating transactions
* Deleting transactions

· JpaRepository<CustomerTransaction, Long>:

* · CustomerTransaction → The **entity class** this repository will handle.
* Long → The **type of the primary key** (ID) in the CustomerTransaction entity.

· **Why extend** JpaRepository**?**  
By extending JpaRepository, we **automatically** get common CRUD methods like:

| · **Method** | **Description** |
| --- | --- |
| save(T entity) | Saves or updates an entity |
| findById(ID id) | Finds an entity by its ID |
| findAll() | Retrieves all entities |
| deleteById(ID id) | Deletes an entity by I |

## ****📌 Steps to Run the Query in PostgreSQL****

### ****1️⃣ Open PostgreSQL Command Line (psql)****

If you have already logged into PostgreSQL, skip to Step 3.

Otherwise, open **Command Prompt (cmd)** or **PowerShell** and run:

sh

CopyEdit

psql -U postgres -d transaction\_db -h localhost -p 5432

* -U postgres → Logs in as user postgres
* -d transaction\_db → Connects to the transaction\_db database
* -h localhost → Connects to the local PostgreSQL server
* -p 5432 → Specifies the default PostgreSQL port

It will prompt you for your **PostgreSQL password**. Enter it to proceed.

### ****2️⃣ Verify Connection to Database****

Once connected, check if the database contains the transactions table:

sql

CopyEdit

\dt

If transactions appears in the list, proceed to the next step.

### ****3️⃣ Run the Query****

Now, execute:

sql

CopyEdit

SELECT \* FROM transactions;

This retrieves **all records** from the transactions table.

### ****4️⃣ Exit PostgreSQL****

To **exit the psql console**, type:

sh

CopyEdit

\q

This returns you to the system terminal.

## ****📌 Alternative: Using pgAdmin****

If you prefer a **GUI method**, follow these steps:

1. Open **pgAdmin** (a graphical PostgreSQL management tool).
2. Click on **"Servers" → "PostgreSQL" → "Databases" → "transaction\_db"**.
3. Open **Query Tool** (from the toolbar).
4. Run:

sql

CopyEdit

SELECT \* FROM transactions;

### ****Summary****

| **Layer** | **Code Location** | **Purpose** |
| --- | --- | --- |
| **Repository Layer** | CustomerTransactionRepository.java | Provides save() method (inherited from JpaRepository). |
| **Service Layer** | CustomerTransactionService.java | Calls save() from the repository to save transactions. |
| **Controller Layer** | CustomerTransactionController.java | Exposes an API endpoint to receive transaction data and save it. |

### ****Steps to Test Your API****

We will test the following endpoints:  
✅ Persist customer transactions (POST /transactions)  
✅ Retrieve a transaction (GET /transactions/{id})  
✅ Transfer funds (POST /transfer)  
✅ Retrieve account balance (GET /balance/{accountId})

### ****3️⃣ Use Postman or cURL to Test****

#### ****🔹 Test Saving a Customer Transaction****

📌 **Endpoint:** POST /transactions  
📌 **Request Body (JSON)**:

json

CopyEdit

{

"accountId": 1,

"amount": 1000.50,

"transactionType": "DEPOSIT",

"timestamp": "2025-03-12T12:00:00"}

📌 **cURL Command:**

bash

CopyEdit

curl -X POST http://localhost:8080/transactions -H "Content-Type: application/json" -d '{

"accountId": 1,

"amount": 1000.50,

"transactionType": "DEPOSIT",

"timestamp": "2025-03-12T12:00:00"

}'

📌 **Expected Response:**

json

CopyEdit

{

"id": 1,

"accountId": 1,

"amount": 1000.50,

"transactionType": "DEPOSIT",

"timestamp": "2025-03-12T12:00:00"}

#### ****🔹 Test Retrieving a Transaction****

📌 **Endpoint:** GET /transactions/1  
📌 **cURL Command:**

bash

CopyEdit

curl -X GET http://localhost:8080/transactions/1

📌 **Expected Response (if transaction exists):**

json

CopyEdit

{

"id": 1,

"accountId": 1,

"amount": 1000.50,

"transactionType": "DEPOSIT",

"timestamp": "2025-03-12T12:00:00"}

📌 **If not found, expected response:**

json

CopyEdit

{

"error": "Transaction not found"}

#### ****🔹 Test Fund Transfer****

📌 **Endpoint:** POST /transfer  
📌 **Request Body (JSON)**:

json

CopyEdit

{

"fromAccountId": 1,

"toAccountId": 2,

"amount": 500.00}

📌 **cURL Command:**

bash

CopyEdit

curl -X POST http://localhost:8080/transfer -H "Content-Type: application/json" -d '{

"fromAccountId": 1,

"toAccountId": 2,

"amount": 500.00

}'

📌 **Expected Response:**

json

CopyEdit

{

"message": "Transfer successful",

"fromAccountBalance": 500.50,

"toAccountBalance": 500.00}

#### ****🔹 Test Retrieving Account Balance****

📌 **Endpoint:** GET /balance/1  
📌 **cURL Command:**

bash

CopyEdit

curl -X GET http://localhost:8080/balance/1

📌 **Expected Response:**

json

CopyEdit

{

"accountId": 1,

"balance": 500.50}