**Step-by-Step Guide to Build a Banking & Payment System Using Spring Boot and Microservices**

We will follow an **incremental approach**, building each microservice step by step.

**Step 1: Setup Development Environment**

✅ **Install Required Tools:**

* **Java 17** (Recommended for Spring Boot 3)
* **IntelliJ IDEA / VS Code**
* **Postman** (For API testing)
* **Docker Desktop** (For containerization)
* **PostgreSQL / ElephantSQL** (Free cloud DB)
* **Apache Kafka / RabbitMQ** (For messaging)
* **Git & GitHub** (For version control)

**Step 2: Create a GitHub Repository**

✅ Create a **GitHub repo** to store your microservices.  
✅ Initialize a project structure:

mkdir banking-system && cd banking-system

git init

echo "# Banking and Payment System" >> README.md

git add . && git commit -m "Initial commit"

**Step 3: Create the Spring Boot Microservices**

We will create the following services **one by one**:

**3.1 Create User Service (Authentication & User Management)**

✅ **Go to** [**Spring Initializr**](https://start.spring.io/) and generate a project with:

* Spring Boot Version: **3.x**
* Dependencies:
  + Spring Web
  + Spring Security
  + Spring Data JPA
  + PostgreSQL
  + Lombok
  + Validation
  + JWT (io.jsonwebtoken)

✅ **Extract & Open in IntelliJ**  
✅ **Configure application.properties:**

server.port=8081

spring.datasource.url=jdbc:postgresql://localhost:5432/bank\_users

spring.datasource.username=postgres

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

jwt.secret=your\_secret\_key

✅ **Create User Entity:**

@Entity

@Table(name = "users")

public class User {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

private String email;

private String role; // ROLE\_USER, ROLE\_ADMIN

}

✅ **Create JWT Authentication:**

1. Implement **JWT Token Provider**
2. Create **Authentication Filter**
3. Configure **Spring Security**
4. Implement **User Registration & Login APIs**

✅ **Run the Service on Port 8081**

mvn spring-boot:run

**3.2 Create Account Service (Bank Accounts Management)**

✅ **Go to Spring Initializr & Create Another Project**

* Dependencies:
  + Spring Web
  + Spring Data JPA
  + PostgreSQL
  + Lombok
  + Spring Cloud OpenFeign

✅ **Configure application.properties for Account Service:**

server.port=8082

spring.datasource.url=jdbc:postgresql://localhost:5432/bank\_accounts

spring.datasource.username=postgres

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

✅ **Create Account Entity:**

@Entity

@Table(name = "accounts")

public class Account {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Long userId; // Link to User Service

private Double balance;

private String accountNumber;

}

✅ **Implement REST API:**

* **GET /accounts/{id}** - Fetch Account Details
* **POST /accounts** - Create a New Account
* **PUT /accounts/{id}/deposit** - Deposit Money
* **PUT /accounts/{id}/withdraw** - Withdraw Money

✅ **Run the Account Service on Port 8082**

**3.3 Create Transaction Service (Fund Transfers & History)**

✅ **Go to Spring Initializr & Create Another Project**

* Dependencies:
  + Spring Web
  + Spring Data JPA
  + PostgreSQL
  + Kafka (Spring for Apache Kafka)

✅ **Configure application.properties for Transaction Service:**

server.port=8083

spring.datasource.url=jdbc:postgresql://localhost:5432/bank\_transactions

spring.datasource.username=postgres

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

✅ **Create Transaction Entity:**

@Entity

@Table(name = "transactions")

public class Transaction {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Long senderId;

private Long receiverId;

private Double amount;

private String status; // PENDING, SUCCESS, FAILED

}

✅ **Implement Transaction API:**

* **POST /transactions** - Initiate a Transaction
* **GET /transactions/{id}** - Get Transaction Details

✅ **Enable Event-Driven Transactions Using Kafka**

**3.4 Create API Gateway (For Routing & Security)**

✅ **Go to Spring Initializr & Create API Gateway Project**

* Dependencies:
  + Spring Cloud Gateway
  + Spring Boot Security
  + Spring Cloud Config Client
  + JWT

✅ **Configure application.properties:**

server.port=8080

spring.cloud.gateway.routes[0].id=user-service

spring.cloud.gateway.routes[0].uri=http://localhost:8081

spring.cloud.gateway.routes[0].predicates=Path=/users/\*\*

spring.cloud.gateway.routes[1].id=account-service

spring.cloud.gateway.routes[1].uri=http://localhost:8082

spring.cloud.gateway.routes[1].predicates=Path=/accounts/\*\*

✅ **Run API Gateway on Port 8080**

**Step 4: Service Discovery (Eureka)**

✅ **Create Eureka Server Project (Spring Initializr)**

* Dependencies:
  + Eureka Server
  + Spring Boot Web

✅ **Configure Eureka Server (application.properties)**

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

✅ **Run Eureka Server on Port 8761**

mvn spring-boot:run

✅ **Register All Services with Eureka**

**Step 5: Deployment on Free Resources**

✅ **Use ElephantSQL for PostgreSQL**  
✅ **Deploy on Render or Railway**  
✅ **Use Docker for Containerization**  
✅ **Use GitHub Actions for CI/CD**

**Next Steps**

* Would you like a **detailed guide on Kubernetes deployment**?
* Need help with **Kafka setup for event-driven transactions**?
* Want **frontend integration** (React/Angular)?

Let me know how you want to proceed! 🚀