Secure Banking Microservices – PoC Project

By Abinaya Ramesh – Java Backend Developer

🧩 1. Project Overview

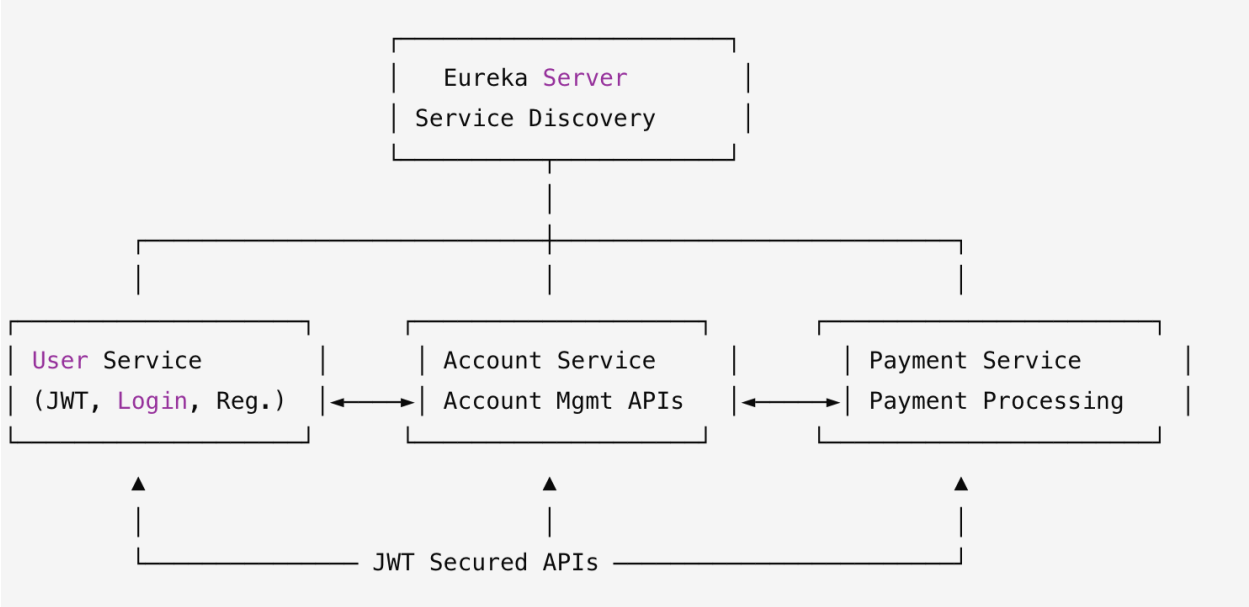
This Proof-of-Concept (PoC) demonstrates a secure, scalable microservices architecture using:

- **Spring Boot**
- **Spring Cloud Eureka (Service Discovery)**
- **Microservices (User, Account, Payment)**
- **Spring Security (JWT Authentication + Authorization)**
- **BCrypt Password Encryption**
- **MySQL Database**
- **REST APIs**
- **Docker (Optional enhancement)**
- **Azure/AWS knowledge (Optional add-on)**

This PoC showcases my backend engineering skills after returning from a career break, demonstrating proficiency in:

✔ Java
✔ Spring Boot
✔ JWT Security
✔ Microservices
✔ SQL
✔ System Design
✔ API Integration
✔ Git & CI/CD
✔ Cloud Awareness (AWS/Azure)
✔ Docker & Docker Compose (Basic)

🏗️ 2. High-Level System Architecture



Eureka Server
Manages service discovery so microservices can talk to each other dynamically.

User Service
Handles registration, login, password encryption, JWT token generation, and authentication.

Account Service
Manages accounts, balances, account creation, and validation.

Payment Service
Handles fund transfers, payment validation, transaction history.

🔐 3. Security Features

JWT Authentication

- User logs in → receives JWT token
- All other microservices require this token
- JwtFilter validates token in every request

Password Encryption

- Passwords stored using **BCrypt**
- Never saved as plain text

Secure Communication

- Inter-microservice calls include JWT tokens
- No session state stored anywhere (stateless architecture)

🛠️ 4. Technologies Used

Backend

- Java 17
- Spring Boot 3+
- Spring Security 6+
- Spring Data JPA
- REST API
- Spring Cloud Eureka

Database

- MySQL (User DB, Account DB, Payment DB)

Authentication

- JWT
- BCryptPasswordEncoder

Cloud / DevOps

- AWS (EC2, S3, IAM — basic knowledge)
- Azure (AZ-900 level knowledge)

- Docker (Containerization)
- GitHub for versioning

📁 5. Project Modules

Module 1: Eureka Server

- Registers all microservices
- Each service is discoverable dynamically
- Enables load balancing if scaled

Module 2: User Microservice

Responsibilities

- User Registration
- Login Authentication
- Password Encryption (BCrypt)
- JWT Token Creation
- Token Validation

Tables

- `users`
- `roles` (optional)

Key APIs

```
POST /auth/register
POST /auth/login
GET  /auth/profile   (secured)
```

Module 3: Account Microservice

Responsibilities

- Create new bank account
- Fetch account details
- Update balances
- Validate account ownership using JWT

Tables

- `accounts`

Key APIs

```
POST /account/create
GET  /account/{id}
PUT  /account/updateBalance/{id}
```

Module 4: Payment Microservice

Responsibilities

- Fund transfer between accounts
- Validate account balances
- Record transaction history

Tables

- `transactions`

Key APIs

```
POST /payment/transfer
GET  /payment/history/{accountId}
```

🧪 6. Database Design (MySQL)

users table

Column	Type	Description
id	BIGINT	PK
username	VARCHAR	unique
password	VARCHAR	encrypted
email	VARCHAR	optional
created_at	TIMESTAMP	default

accounts table

Column	Type
id	BIGINT
user_id	BIGINT
account_number	VARCHAR
balance	DOUBLE

transactions table

Column	Type
id	BIGINT
from_account	BIGINT
to_account	BIGINT
amount	DOUBLE
timestamp	TIMESTAMP

📦 7. Installation Instructions

Step 1: Clone the Repository

```
git clone
https://github.com/abinaya-ramesh/banking-microservices-poc.git
cd banking-microservices-poc
```

Step 2: Start Eureka Server

```
cd eureka-server
```

```
mvn spring-boot:run
```

Step 3: Start Microservices

(Separate terminals)

```
cd user-service
mvn spring-boot:run

cd account-service
mvn spring-boot:run

cd payment-service
mvn spring-boot:run
```

Step 4: Test Login API

```
POST /auth/login
{
  "username": "admin",
  "password": "admin123"
}
```

Copy JWT token → use in Postman Authorization Header.

```
Authorization: Bearer <token>
```

📝 8. Testing the Microservices Flow

Test the Order

1. Register user
2. Login → get JWT
3. Create account
4. Transfer money
5. View transaction history

🎯 9. Goals Showcased in This Project

✔ Microservices architecture
✔ Hands-on Spring Boot development
✔ JWT + Spring Security
✔ SQL queries and joins
✔ Clean code, layered architecture (Controller → Service → Repository)
✔ Modern backend engineering practices
✔ Ability to build a real-world project end-to-end
✔ Ability to restart career strongly with current technologies

---

📝 10. Future Enhancements (Optional Add-ons)

- Add Docker / Docker Compose
- Add API Gateway (Spring Cloud Gateway)
- Add Kafka for async events
- Add Azure deployment
- Add Grafana/Prometheus monitoring
- Add unit tests using JUnit + Mockito

⭐ 11. Author

Abinaya Ramesh
Java Backend Developer | Microservices | Spring Boot | SQL
Open to roles in Singapore