Banking Microservices – Developer Guide

Banking Microservices – Developer Guide

Overview

A practice banking system built with Spring Boot **3.2.7** and Spring Cloud **2023.0.3** on **Java 21**, split into:

- **Infrastructure**
  - **Eureka Server** (`:8761`) – service discovery
  - **API Gateway** (`:8080`) – Spring Cloud Gateway (MVC) routing
- **Business Services**
  - **auth-service** (`:8081`) – JWT auth (RS256), refresh tokens
  - **customer-service** (`:8082`) – customer profiles & KYC
  - **account-service** (`:8083`) – accounts & balances
  - **credit-service** (`:8084`) – loans/cards (skeleton)
  - **payment-service** (`:8085`) – transfers (skeleton)
  - **notification-service** (`:8086`) – email/sms (skeleton; later via RabbitMQ)
  - **admin-service** (`:8087`) – approvals & orchestration (Feign)

> Notes
>
> - **No** `@EnableEurekaClient` needed—having the Eureka client starter + properties auto-registers.
> - Use `@EnableFeignClients` **only** in services that actually call others (e.g., `admin-service`, later `payment-service`, etc.).
> - Gateway **does not aggregate Swagger**; each service exposes its own OpenAPI UI.

Repository Structure

```
banking-microservices/
├── pom.xml                      # parent (aggregator) – BOMs only
├── infra/
│   ├── eureka-server/
│   └── api-gateway/
├── services/
```

```
│   ├── auth-service/
│   ├── admin-service/
│   ├── customer-service/
│   ├── account-service/
│   ├── credit-service/
│   ├── payment-service/
│   └── notification-service/
├── scripts/
│   ├── Start-Dev.ps1           # Windows start
│   ├── Stop-Dev.ps1            # Windows stop
│   ├── start-dev.sh            # Linux/macOS start
│   └── stop-dev.sh             # Linux/macOS stop
└── dev-keys/
    └── jwt_public_key.pem      # placeholder (real key copied after first auth
run)
```

Prerequisites

- **JDK 21** (confirm with `java -version`)
- **Maven 3.9+** (the repo also supports `mvnw`)
- **MySQL 8** running locally with a user who can create DBs
- IntelliJ IDEA (or Cursor/VS Code) with **Lombok plugin** enabled

IntelliJ / IDE settings (avoid compiler/runtime mismatches)

- **File → Project Structure → Project**
  - Project SDK: **Java 21**
  - Language level: **SDK default (21)**
- **Settings → Build, Execution, Deployment → Build Tools → Maven**
  - **JDK for importer: 21**
  - (If Gradle appears anywhere for you): set it to **JDK 21** as well
- **Settings → Build → Compiler → Annotation Processors**
  - **Enable annotation processing** (for Lombok)

Parent POM essentials (already set)

The root `pom.xml` manages versions and the Java level:

- `maven.compiler.release = 21`
- Spring Boot **3.2.7**, Spring Cloud **2023.0.3**
- Centralized versions for `springdoc`, `jjwt`, `totp`, `lombok`

- `maven-compiler-plugin` with Lombok annotation processor configured
- **Children must inherit** this parent and **must not** use `spring-boot-starter-parent`

Local Configuration (before first run)

1) Databases

Each service uses an isolated DB. Example properties per service (adjust ports/names):

```
spring.datasource.url=jdbc:mysql://localhost:3306/<service_db>?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Recommended DB names:

```
banking_auth_db
banking_customer_db
banking_account_db
banking_credit_db
banking_payment_db
banking_notification_db
banking_admin_db
```

2) Eureka and service naming

```
spring.application.name=<service-name>
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
```

3) JWT keys (critical)

- **Auth-service** generates RSA **private/public keys** on its first successful start.
- **All other services validate JWTs** using the **public key** file path.

Do this once:

1. Ensure this folder exists:
   `D:/Wipro project/banking-microservices/dev-keys/`
2. Place a **placeholder** file at:
   `dev-keys/jwt_public_key.pem` (empty file is fine to start)
3. Start **auth-service** once (after Eureka) so it generates real keys.
4. Copy the **real** `jwt_public_key.pem` over the placeholder in `dev-keys/`.

Non-auth services must point to the public key:

```
# set an absolute path; you can also use an ENV var
jwt.public-key-path=D:/Wipro project/banking-microservices/dev-
keys/jwt_public_key.pem
```

> **Never commit** real keys. Add to `.gitignore`:
>
> ```
> dev-keys/
> *.pem
> src/main/resources/keys/
> ```

Startup Order

> If you're not using the scripts, you can start in this **manual order**:

```
# 1) Eureka
mvn -pl infra/eureka-server spring-boot:run

# 2) Auth (generates keys)
mvn -pl services/auth-service spring-boot:run

# 3) Gateway
mvn -pl infra/api-gateway spring-boot:run

# 4) Core services (after jwt.public-key-path is correct)
mvn -pl services/customer-service spring-boot:run
mvn -pl services/account-service spring-boot:run
mvn -pl services/credit-service spring-boot:run
mvn -pl services/payment-service spring-boot:run
mvn -pl services/admin-service spring-boot:run
```

Scripts

- **Windows**: `.\scripts\Start-Dev.ps1` / `.\scripts\Stop-Dev.ps1`
- **Linux/macOS**: `./scripts/start-dev.sh` / `./scripts/stop-dev.sh`

> If a PowerShell stop script can't kill a process on a port (e.g., `httpd` on 8080), run a terminal as **Admin** or stop that process manually.

Swagger / OpenAPI

- **Gateway** UI: `http://localhost:8080/swagger-ui/index.html`
  (shows Gateway endpoints only; not downstream services)

- **Per-service UIs**:
  - `http://localhost:8081/swagger-ui/index.html` → **auth-service**
  - `http://localhost:8082/swagger-ui/index.html` → **customer-service**
  - `http://localhost:8083/swagger-ui/index.html` → **account-service**
  - `http://localhost:8084/swagger-ui/index.html` → **credit-service**
  - `http://localhost:8085/swagger-ui/index.html` → **payment-service**
  - `http://localhost:8086/swagger-ui/index.html` → **notification-service**
  - `http://localhost:8087/swagger-ui/index.html` → **admin-service**

Swagger "Authorize" button – paste `Bearer <accessToken>` (including the `Bearer` prefix).

Smoke Test (end-to-end flow)

> Validates Eureka, Gateway routing, DBs, and JWT checks in a few calls.

1. **Sign up** (auth-service)

```
POST /auth/signup
{
  "email": "john.doe@example.com",
  "password": "secret123",
  "firstName": "John",
  "lastName": "Doe",
  "roles": ["CUSTOMER"]
}
```

2. **Sign in** → get JWT

```
POST /auth/signin
{
  "email": "john.doe@example.com",
  "password": "secret123"
}
```

Response:

```
{
  "accessToken": "<JWT>",
  "refreshToken": "<JWT>",
  "tokenType": "Bearer"
}
```

Copy `accessToken`.

3. **Create customer** (customer-service)

```
POST /customers
Authorization: Bearer <accessToken>
{
  "firstName":"John",
  "lastName":"Doe",
  "email":"john.doe@example.com",
  "phone":"+1234567890",
  "address":"123 Main St"
}
```

> **Important**: in your controller, use the **userId from the token claim**, not `Long.parseLong(authentication.getName())`. Example:
>
> ```
> String sub = authentication.getName();      // "johndoe" / email / username
> Long userId =
> ((JwtAuthenticationToken)authentication).getToken().getClaim("userId");
> ```

4. **Open account** (account-service)

```
POST /accounts
Authorization: Bearer <accessToken>
{
  "customerId": 1,
  "accountType": "SAVINGS",
  "initialDeposit": 5000
}
```

5. **Transfer money** (payment-service; once implemented)

```
POST /payments/transfer
Authorization: Bearer <accessToken>
{
  "fromAccountId":101,
  "toAccountId":102,
  "amount":500
}
```

6. **Check balance** (account-service)

```
GET /accounts/101/balance
Authorization: Bearer <accessToken>
```

If you get `401` with `invalid_token` / `Jwt expired`, call `POST /auth/refresh` to get a new access token and retry.

API Catalog (current)

auth-service (`/auth`)

- `POST /auth/signup` – register user
- `POST /auth/signin` – login → `{accessToken, refreshToken}`
- `POST /auth/refresh` – exchange refresh token for new access token
- `POST /auth/logout` – revoke refresh tokens for current user
- `GET /auth/public-key` – returns RSA public key (PEM/plain)
- `GET /auth/me` – current user info
- `POST /auth/validate` – (optional) validate a token

customer-service (`/customers`)

- `POST /customers` – create customer (uses `userId` claim)
- `GET /customers/{id}`
- `PUT /customers/{id}`
- `POST /customers/{id}/kyc`
- `GET /customers/approvals`
- `POST /customers/approvals/bulk`
  - **Body shape** is a **map** (Swagger will show `additionalProp` placeholders). Example:

    ```
    {
      "customer-service": { "ids": [1,2], "status": "APPROVED" }
    }
    ```

account-service (`/accounts`)

- `POST /accounts` – create account
- `GET /accounts/{id}`
- `GET /accounts/{id}/balance`
- `GET /accounts/approvals`
- `POST /accounts/approvals/bulk`

credit-service (`/credits`) *(skeleton)*

- `POST /credits/loans`
- `POST /credits/cards`
- `GET /credits/approvals`
- `POST /credits/approvals/bulk`

payment-service (`/payments`) *(skeleton)*

- `POST /payments/transfer`
- `GET /payments/{id}`
- `GET /payments/approvals`
- `POST /payments/approvals/bulk`

notification-service (`/notifications`) *(skeleton)*

- `POST /notifications/send`
- `GET /notifications/{id}`

admin-service (`/admin`)

- `GET /admin/approvals/pending` – aggregates pending from services
- `POST /admin/approvals/execute` – bulk execute approvals across services
  Example body:

  ```
  {
    "customer-service": { "ids": [101,102], "status": "APPROVED" },
    "account-service":  { "ids": [201],     "status": "REJECTED"  }
  }
  ```

Core Data Models (DTOs – current)

> (Lombok used project-wide – don't hand-code getters/setters)

- **Auth**
  - `SignupRequest { email, password, firstName, lastName, roles[] }`
  - `SigninRequest { email, password }`
  - `AuthTokens { accessToken, refreshToken, tokenType }`
  - JWT claims include: `sub`, `roles`, `userId`, `iat`, `exp`
- **Customer**
  - `CustomerRequest { firstName, lastName, email, phone, address }`
  - `CustomerResponse { id, status }`
- **Account**
  - `AccountRequest { customerId, accountType, initialDeposit }`
  - `AccountResponse { accountId, balance, status }`
- **Payment (later)**
  - `TransferRequest { fromAccountId, toAccountId, amount }`
  - `TransferResponse { transactionId, status }`

- **Admin**
  - `ApprovalItem { ids[], status }`
  - `Map<String, ApprovalItem>` as request body

JWT & Security

- Every business service is an **OAuth2 Resource Server** that validates JWTs using `jwt.public-key-path`.
- If a request fails with:

  ```
  WWW-Authenticate: Bearer error="invalid_token", ... "Jwt expired ..."
  ```

  → **Call** `POST /auth/refresh` with the `refreshToken`, get a new `accessToken`, and **retry** the original request.

Permit (in each service):

- `/v3/api-docs/**`, `/swagger-ui/**`
- `/actuator/health`, `/actuator/info`
- Auth endpoints in **auth-service** only

Secure everything else; use `@PreAuthorize` where needed.

Inter-Service Communication – What to implement next

Phase 1 — Money moves (most useful)

- **payment-service** → **account-service** (sync, Feign)
  - Add in **account-service**:
    - `GET /accounts/{id}/balance`
    - `POST /accounts/{id}/authorizations` (place hold)
    - `POST /accounts/{id}/debits` (idempotent)
    - `POST /accounts/{id}/credits` (idempotent)
  - In **payment-service**:
    - `@EnableFeignClients`
    - `@FeignClient(name="account-service", …)`
    - A single **Feign** `RequestInterceptor` bean that forwards the inbound `Authorization` header to downstream.

Phase 2 — Onboarding

- **customer-service** → **account-service** (sync Feign first, later event)
  - Auto-create a default savings account on KYC approval.

Phase 3 — Credit wiring

- **credit-service** → **account-service** (sync Feign)
  - Create liability account, post charges & repayments.

Phase 4 — Notifications (async)

- Everyone emits events (`Payment.Completed`, `Customer.KycApproved`, …)
- **notification-service** consumes via RabbitMQ and sends messages.

Phase 5 — Admin orchestrator (you have it)

- **admin-service** → **{customer, account, credit, payment}** (sync Feign)
  Aggregate pending & execute bulk approvals.

> Only services that declare `@FeignClient` need `@EnableFeignClients`.

Resilience / Safety

- Forward `Authorization` in Feign (downstream re-validates JWT).
- Add **Resilience4j** circuit breakers on payment→account calls.
- Enforce **Idempotency** (header `Idempotency-Key` or `requestId` in body) for debits/credits/transfers.
- Model transfers as a **saga**: authorize → credit → finalize debit (or cancel).

Build & Run

- Build all (skip tests while wiring):

  ```
  mvn -T1C clean package -DskipTests
  ```

- Start manually (order above) or use scripts.
- Verify Eureka at `http://localhost:8761`.

Troubleshooting

1) `com.sun.tools.javac.code.TypeTag :: UNKNOWN` / `ExceptionInInitializerError`

- Caused by **Java 25** toolchain or mixed levels.
  Fix: Set **everything** to **Java 21** (JDK, Maven importer JDK, language level). Clean + rebuild.

2) Swagger shows no “Authorize” or missing Bearer

- Make sure `springdoc-openapi-starter-webmvc-ui` is on the classpath.

- Check your OpenAPI security config (Bearer scheme) is present.
- Use each service's swagger URL (gateway doesn't aggregate).

3) `WWW-Authenticate: invalid_token … Jwt expired`

- Call `POST /auth/refresh` with `refreshToken` and retry with the new `accessToken`.

4) `NumberFormatException: "johndoe"` in `createCustomer`

- Don't `Long.parseLong(authentication.getName())`.
  Extract `userId` from the JWT claim instead.

5) Port in use (8080/8761/…)

- Change `server.port` in the service, or stop the process using that port.

6) Eureka not showing service

- Wait ~30s, check `eureka.client.service-url.defaultZone`, and that the service can reach `8761`.

7) `parserBuilder()` missing on `Jwts`

- Ensure `jjwt` **0.12.5** is used (managed in parent). `mvn dependency:tree` to confirm.

Nice-to-Have Next

- **Observability**: Micrometer + Prometheus + Grafana
- **Centralized config**: Spring Cloud Config or environment-based
- **Secrets**: use env vars or a vault, never commit real keys
- **Rate limiting**: at gateway (per token/user)
- **Postman Collection** / **REST Assured** smoke tests
- **Docker Compose** for MySQL + RabbitMQ + all services (later)

TL;DR – First Run Checklist

1. Java **21** everywhere (IDE + Maven importer + terminal).
2. MySQL up; service DB URLs configured with `createDatabaseIfNotExist=true`.
3. `dev-keys/jwt_public_key.pem` placeholder exists.
4. Start order:
   - Eureka → Auth (generates keys) → Gateway → Others
5. Copy the **real** `jwt_public_key.pem` from auth to `dev-keys/` and set `jwt.public-key-path` in all other services.
6. Sign up → Sign in → use **accessToken** in Swagger "Authorize" → run the smoke flow.
