# Bank-Web-App

## API Gateway-Service

# TABLE OF CONTENTS

# 1.INTRODUCTION

API Gateway

The API Gateway is the single-entry point for all client requests in a microservices architecture.

**Key Features in Local Setup:**

- **Routing:** Forwards requests to local microservices based on predefined paths or routes (e.g., /auth/**, /notification/**, /download/**).
- **Centralized Access Point:** Allows frontend or API clients to interact with multiple services via a single local host and port.
- **Local Debugging:** Helps test end-to-end flows across services while debugging or logging at the gateway level.

# 2. TECH STACK

| Technology | Version |
|---|---|
| JDK | 17 |
| Maven | 3.6.3 |
| Spring Boot | 3.5.3 |
| Kafka | 7.4.3 |
| Zookeeper | 7.4.3 |
| Zipkin | 2.23 |

# 3.FILTER

## 3.1.1. Security-Config

**API Gateway – Security Configuration**

The SecurityConfig class is responsible for configuring security for the **API Gateway** using **Spring WebFlux Security** and **OAuth2 JWT** authentication. This configuration ensures secure access control across microservices by defining public routes, enforcing role-based authorization, and integrating with an external identity provider (e.g., **Keycloak**).

**Key Responsibilities**

- **Token-based Authentication**
  The API Gateway acts as a resource server, validating incoming **JWT tokens** issued by the authorization server. A custom KeycloakRoleConverter extracts and maps user roles from the token.
- **Public Route Configuration**
  Specific endpoints (e.g., authentication, heartbeat checks, and calculators) are publicly accessible without authentication.
- **Role-Based Access Control (RBAC)**
  Access to protected endpoints is controlled using predefined user roles:
    - ROLE_service: Access to service-level operations (e.g., balance retrieval, EMI deductions).
    - ROLE_employee: Access to employee-level operations (e.g., freeze account, branch reporting).
    - ROLE_admin: Access to administrative features (e.g., adding branches, generating reports).
- **CSRF Protection Disabled**
  Since this is a stateless API Gateway, **CSRF protection is disabled** for simplicity and compatibility with RESTful patterns.

### 3.1.2 Keycloak-Role-Converter

The KeycloakRoleConverter is a custom component used in the **API Gateway** to extract user roles from a **JWT token** issued by **Keycloak**.

**What It Does:**

- Reads the realm_access.roles field from the JWT.
- Converts each role into a Spring Security authority (e.g., admin → ROLE_admin).
- Returns a JwtAuthenticationToken containing the user's roles.

**Why It's Needed:**

Spring Security requires roles in a specific format (ROLE_...). This converter makes sure roles from Keycloak are understood and enforced correctly by the API Gateway.

**Example JWT Claim:**

```
"realm_access": {
  "roles": ["admin", "employee"]
}
```

This will be converted to:

- ROLE_admin
- ROLE_employee

### 3.1.3 JwtDecoderConfig

The JwtDecoderConfig class configures how the **API Gateway** decodes and validates JWT tokens issued by **Keycloak**.

🔧 **What It Does:**

- Defines a ReactiveJwtDecoder bean using the **JWK Set URI** from Keycloak:

```
http://localhost:9098/realms/bank-web-app/protocol/openid-connect/certs
```

- Uses **NimbusReactiveJwtDecoder** to fetch public keys from Keycloak for verifying token signatures.
- **Disables the default expiration check**, allowing expired tokens to pass validation (not recommended for production).

### 3.1.4 TokenRefreshFilter

The TokenRefreshFilter is a **custom GlobalFilter** used in the **Spring Cloud Gateway** to handle **JWT expiration and automatic token refresh** using **Keycloak**.

**Purpose**

This filter intercepts incoming HTTP requests, checks if the **access token is expired**, and if so:

- Validates the refresh token stored in **Redis**.
- Requests a new access token from **Keycloak**.
- Injects the refreshed access token into the request header.
- Proceeds with the updated request transparently.

| Feature | Description |
|---|---|
| **JWT Expiration Check** | Parses and checks if the access token has expired using SignedJWT. |
| **Redis Integration** | Retrieves and matches stored access_token and refresh_token by user ID. |
| **Keycloak Integration** | Uses the refresh token to obtain a new access token from Keycloak. |
| **Request Mutation** | Adds the new access token to the Authorization header before forwarding. |
| **Graceful Failure Handling** | Returns 401 Unauthorized if tokens are invalid or expired. |