MicroBank

API Documentation v1.0

Author: Ahmet ATAR

Table of Contents

1. **Introduction** 1
   - 1.1. Overview of Microservice Architecture 2
   - 1.2. Purpose of Documentation 3
   - 1.3. Target Audience 4
2. **Getting Started** 1
   - 2.1. Prerequisites 2
   - 2.2. Base URL Structure 3
   - 2.3. Postman Collection 4
   - 2.4. Keycloak Realm Export 5
3. **Project Architecture** 1
   - 3.1. Backend Project Diagram 2
   - 3.2. Microservices of the project 3
   - 3.3. Storage Solutions 4
   - 3.4. Containerization 5
   - 3.6. Service Communications 6
4. **Identity and Access Management** 1
   - 4.1. Keycloak Integration Overview 2
   - 4.2. Authentication Protocol 3
   - 4.3. Web Security and Web Flux Security 4
     - 4.3.1 API Gateway (AuthN Layer) 5
     - 4.3.2 Microservices (Authorization Layer) 6
     - 4.3.3 Why WebFlux Security for API Gateway? 7
     - 4.3.4 Why Traditional Web Security for Microservices? 8
5. **Endpoints Overview** 2
6. **Microservices** 1
   - 6.1. API Gateway 2
   - 6.2. Eureka Discovery Service 3
   - 6.3. Authentication Service 4
     - 6.3.1 Register 5
     - 6.3.2 Activate 6
     - 6.3.3 Login 7
     - 6.3.4 Refresh Token 8
     - 6.3.4 Forgot Password 1
     - 6.3.5 Reset Password 2
     - 6.3.5 Get Current User's Profile 3
     - 6.3.5 Get User by ID 4
     - 6.3.5 Get All Users 5
     - 6.3.5 Update User Role 6
     - 6.3.5 Update User Access 7
     - 6.3.5 Delete User 8
   - 6.4. Account Service 9

6.4.1 Create Bank Account 2
6.4.2 Update Account Balance 1
6.4.2 Get Current User's All Accounts 2
6.4.2 Get Current User's Account by ID 3
6.4.2 Get All Accounts 4
6.4.2 Get Accounts by User ID 5
6.4.2 Update Account Status 6
6.4.2 Delete Own Account 7
6.4.2 Delete Account 8
6.6. Transaction Service 9
6.6.1 Create Transaction 1
6.6.4 Get Current User's All Transactions 2
6.6.4 Get Current User's Transaction by ID 3
6.6.2 Get All Transactions 4
6.6.3 Get Transaction by ID 5
6.6.3 Get Transactions by Account ID 6
6.6. Document Service 7
6.6.1 Create Transaction Receipt PDF 8
6.6.4 Download Transaction Receipt PDF 9
6.7. Notification Service 1

1. Introduction

This section introduces the project, providing an overview of its architecture, the purpose of the documentation, and the intended audience. It sets the context for understanding and utilizing the system effectively.

1.1 Overview of Microservice Architecture

This project is built using a microservice architecture to achieve scalability, flexibility, and maintainability. Each microservice in the system is designed around a specific business domain, allowing independent development, deployment, and scaling. Key features of this architecture include:

- **Service Autonomy**: Each microservice manages its own database and operates independently, ensuring data consistency and reducing dependencies.
- **API Gateway**: A centralized gateway handles all incoming requests, routing them to the appropriate services and enforcing security measures such as authentication and authorization.
- **Event-Driven Communication**: The system leverages RabbitMQ for asynchronous communication between services, enabling real-time processing of events such as transactions, account activations, and notifications.
- **Containerized Deployment**: Using Docker and Docker Compose, the entire system is packaged and deployed as containers, ensuring consistency across environments and simplifying scaling.

This architecture is ideal for a modern banking application, supporting high availability, fault tolerance, and the ability to handle complex workflows across multiple domains.

1.2 Purpose of Documentation

The purpose of this documentation is to provide a comprehensive guide for understanding, deploying, and interacting with the backend system. Specifically, this document aims to:

- Describe the system architecture and its components in detail.
- Provide setup instructions to get the system running in local or production environments.
- Offer detailed API references for developers to integrate and test the backend services.
- Encourage contributions by offering clear guidance on how developers can extend or improve the system.

This documentation is designed to facilitate both understanding and collaboration, making it easier to utilize and contribute to the project.

1.3 Target Audience

This documentation is intended for two main groups of people:

- **Developers who want to utilize the project**: Frontend and mobile developers looking to integrate with the backend system. This documentation provides clear guidance on how to interact with the APIs and leverage the system's functionality.
- **Developers who want to contribute to the project**: Backend developers or contributors interested in enhancing or extending the system. The documentation explains the architecture and internal workings to make contributing as seamless as possible.

Whether you're here to use the system or to contribute to its growth, this documentation is written to support your goals and make working with the project as straightforward as possible.

2. Getting Started

This section provides all the necessary information to set up and begin using the backend system. By following these steps, developers and testers can ensure they have the required tools and configuration to interact with the API seamlessly.

2.1 Prerequisites

Before you start using or deploying this project, ensure that the following prerequisites are met:

1. **Environment Requirements**:
    - Java 17 or higher for running the backend services.
    - Maven (latest version) for building the microservices.
    - Docker and Docker Compose for containerization and orchestration.
2. **Tools for API Interaction**:
    - Postman or any similar REST client for testing API endpoints.
    - A web browser for accessing services like Keycloak and testing frontend interactions.
3. **Database and Messaging Setup**:

    - PostgreSQL for each microservice's data storage.
    - RabbitMQ for message queueing.
4. **Additional Dependencies**:
    - Ensure that Redis, MinIO, MailDev, and Keycloak are set up, preferably using Docker Compose provided in this project.
5. **Access Credentials**:
    - A default Keycloak admin user is required to manage realms, clients, and roles.
    - SMTP credentials for testing email notifications (or rely on MailDev).

2.2 Base URL Structure

The API is structured around a central API Gateway, which routes requests to the appropriate backend microservices. Below is the base URL structure for the API:

- **Base URL**: `http://localhost:8123/api/v1` (default for local environments)
- **Common Endpoint Patterns**:
    - `/auth/**` – Authentication-related endpoints (e.g., register, activate, login, password reset).
    - `/accounts/**` – User account operations (e.g., account creation, balance inquiries).
    - `/transactions/**` – Financial transaction operations (e.g., transfers).
    - `/documents/**` – Document-related operations (e.g., generating and downloading transaction receipts).
    - `/notifications/**` – Email notifications (e.g., account activation, transaction receipt information).

For production environments, replace `localhost` with the public domain or IP address of the deployment server.

2.3 Postman Collection

To simplify testing and interaction with the API, a Postman Collection has been provided as part of this project. The collection includes pre-configured requests for all microservice endpoints, along with example payloads and headers. Follow these steps to import and use the collection:

1. Download the Postman Collection from the link below, or check the root directory of the project repository:

2. Open Postman and import the collection:
    - Go to File > Import in Postman and select the JSON file.
3. Set up environment variables in Postman:
    - Add variables like `baseURL` with the value `http://localhost:8123/api/v1` and authentication tokens for seamless testing.
4. Test endpoints:
    - Use the imported collection to test endpoints across Authentication, Account, Transaction, Document, and Notification services.

This collection allows developers to quickly interact with the API without manually configuring requests, making the testing process streamlined and efficient.

2.4 Keycloak Realm Export

To simplify the setup of Keycloak for this project, a pre-configured `realm-export.json` file has been provided. This file includes all the necessary configurations, roles, and permissions to get started quickly with authentication and authorization. Using this file, you can import the realm into your Keycloak instance without manually creating roles or configuring client settings.

Steps to Import the Keycloak Realm Export:

1. **Access Keycloak Admin Console**:
    - Open your browser and navigate to the Keycloak Admin Console.
    - Default URL for local setup: `http://localhost:9098`.
2. **Login to Admin Console**:
    - Use the administrator credentials set during Keycloak setup.
3. **Import the Realm**:
    - Go to the "Realm Settings" section in the admin console.
    - Click on "Add Realm" and select the `realm-export.json` file provided in the repository.
    - Confirm the import process.
4. **Test the Configuration**:
    - Use the credentials for test users or create new users in the realm and assign appropriate roles.

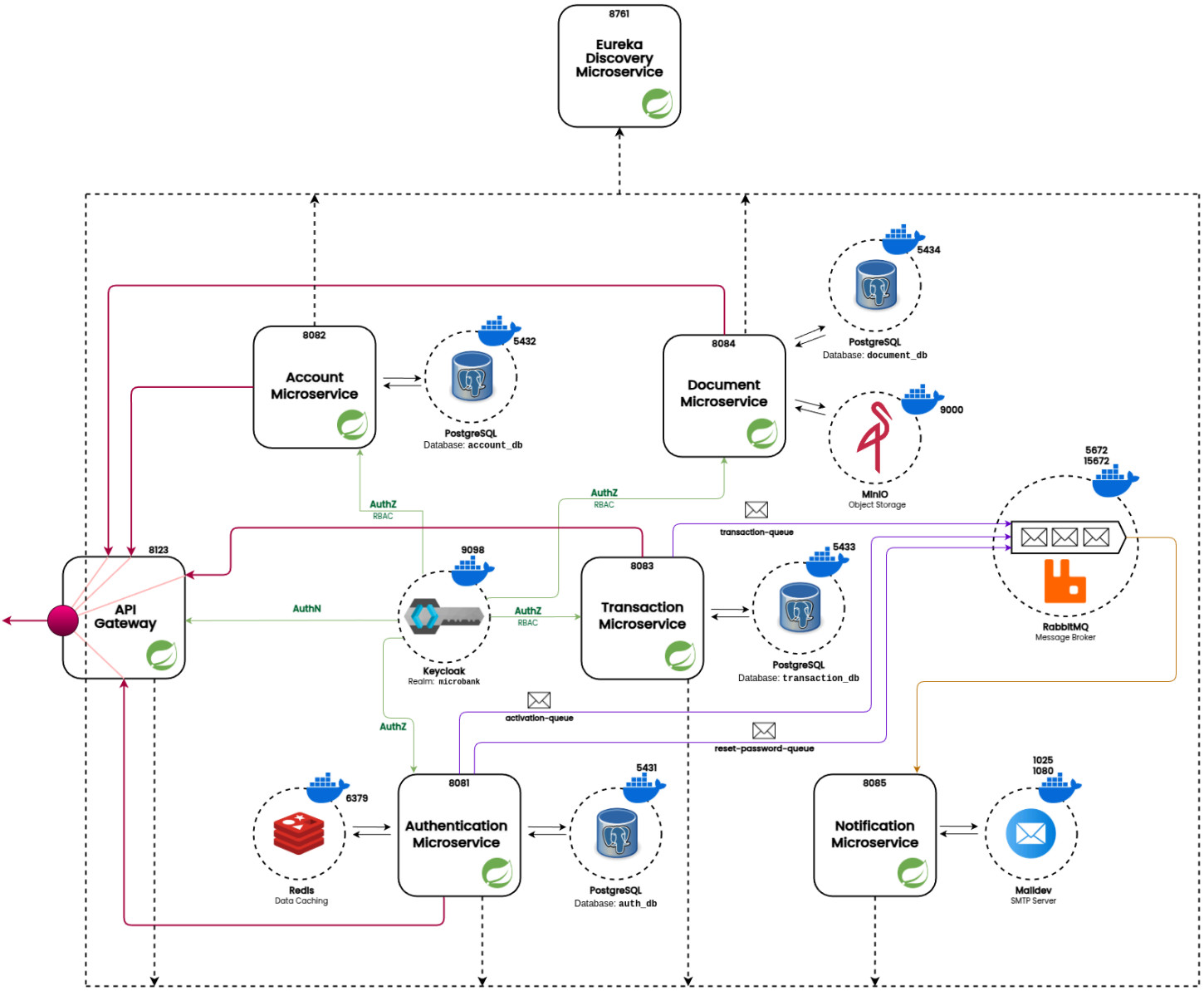
File Location: You can download the `realm-export.json` file below or visit the root directory of the project repository.

3. Project Architecture

The backend architecture of this project is based on the microservices paradigm, which ensures scalability, maintainability, and fault tolerance. Each microservice is responsible for a specific domain or functionality, and the overall architecture leverages cutting-edge technologies such as Docker, RabbitMQ, PostgreSQL, Redis, Keycloak, and MinIO. Below, a detailed breakdown of the project architecture is provided.

3.1 Backend Project Diagram

The system architecture revolves around several independent microservices that are orchestrated and interconnected via an API Gateway. The diagram outlines how these services communicate with each other and external systems. Central components include service discovery, authentication, database integration, message queues, and object storage.



3.2 Microservices of the Project

Each microservice in this project is independently deployable and designed around a specific business capability. These services work together to deliver the functionality of the platform while adhering to core principles of microservice architecture. Key highlights include:

- **Loose Coupling**: Services communicate primarily through HTTP/REST APIs and message queues, minimizing dependencies and enabling flexibility in service updates.
- **Autonomy**: Each service manages its own database, ensuring data isolation and reducing the risk of cross-service data corruption.
- **Scalability**: Microservices can be scaled independently based on workload and performance requirements, ensuring efficient resource utilization.

The core microservices and infrastructure components in this project are:

- **API Gateway**: The API Gateway acts as the single entry point for all client interactions. It efficiently routes requests to the appropriate backend microservices while handling authentication and authorization. By integrating with Keycloak, the gateway ensures secure authentication and token validation before forwarding requests. It also simplifies external access by abstracting the internal microservices' architecture, providing a unified API interface for clients.
- **Eureka Discovery Service**: Eureka serves as the backbone of service discovery and registration in this microservice ecosystem. It allows each microservice to register itself dynamically at runtime, enabling discovery of service endpoints. This eliminates the need for hardcoded service URLs and ensures load balancing. By leveraging Eureka, the architecture supports dynamic scaling, making it possible to deploy additional instances of any microservice.
- **Authentication Service**: The Authentication Service is the cornerstone of identity and access management in the system, working closely with Keycloak for token issuance and validation. It provides core functionalities such as user registration, login, account activation, and password recovery. To enhance performance, it utilizes Redis as a caching layer for temporary data. Its tight integration with the API Gateway and Keycloak ensures robust security and streamlined authentication workflows.
- **Account Service**: The Account Service is responsible for managing user account operations, including account creation, retrieval, and updates. It uses PostgreSQL for persistent data storage, ensuring data integrity and consistency. By enforcing role-based access control through Keycloak, it ensures that only authorized users can access or modify account information. As a core business microservice, it is accessible through the API Gateway, allowing external clients to perform account-related actions securely.
- **Transaction Service**: The Transaction Service handles the platform's financial operations, including processing transfers, payments, and deposits. It uses PostgreSQL for storing transaction records and maintains a robust audit trail. By publishing transaction events to the transaction queue in RabbitMQ, it supports asynchronous communication with the Notification Service.
- **Document Service**: The Document Service focuses on generating, storing, and delivering documents such as receipts, account statements, and transaction reports. Using Apache PDFBox, it dynamically creates PDF documents, which are then stored in MinIO. This microservice also utilizes PostgreSQL for metadata storage to retrieve and manage documents. By providing secure pre-signed URLs, the service ensures that users can access their documents safely.
- **Notification Service**: The Notification Service operates as an internal microservice dedicated to sending email notifications based on events occurring within the system. It consumes messages from RabbitMQ queues such as transaction, account activation, and password recovery messages, ensuring event-driven communication and real-time notification delivery. Emails are sent using MailDev which is an SMTP server. By remaining internal and not directly accessible through the API Gateway, this service enhances system security while efficiently handling event-driven notification workflows.

3.3 Storage Solutions

This project uses several stateful components, each tailored to its purpose, all deployed via Docker containers for consistent and isolated environments:

- **PostgreSQL**:
  - Used for persistent, relational data storage across microservices.
  - Each service (e.g., Account, Transaction, Document, Authentication) maintains its own PostgreSQL instance with service-specific schemas.
- **Redis**:
  - Serves as a high-speed, in-memory cache for temporary data such as authentication tokens and session data.
  - Accelerates the performance of the Authentication Service by reducing frequent database hits.
- **MinIO**:
  - Acts as an object storage solution for the Document Service.
  - Stores user-generated content like receipts and reports in a scalable, S3-compatible storage layer.
  - Files are accessed via pre-signed URLs, ensuring secure delivery to clients.

These solutions are configured to run as containerized services, ensuring seamless integration with the rest of the system.

3.4 Containerization

The entire backend infrastructure is containerized using Docker, which ensures consistency across development, testing, and production environments. Key components include:

- **Docker Images**:
  - Microservices and dependencies (e.g., Keycloak, RabbitMQ, PostgreSQL, Redis, MailDev) are deployed as Docker images.
  - Services are isolated yet interoperable within the same Docker network.

- **Docker Compose**:
  - A `docker-compose.yml` file orchestrates the startup of all services.
  - Ensures that all containers, including infrastructure services like RabbitMQ and MinIO, are launched in the correct order.
  - Simplifies local development and testing by allowing the entire system to be brought up with a single command.

This approach accelerates deployment, facilitates scaling, and ensures reproducibility across environments.

3.5 Service Communications

The project employs a hybrid communication model for inter-service interactions, balancing performance and reliability:

- **Asynchronous Communication**:
  - RabbitMQ is utilized for event-driven messaging between services.
  - Key queues include:
    - `activation-queue:` Handles user activation events.
    - `reset-password-queue:` Processes password reset requests.
    - `transaction-queue:` Publishes transaction events for downstream services.
  - This asynchronous approach decouples services, enabling them to operate independently and improving system resiliency.
- **Synchronous Communication**:
  - Microservices interact synchronously via HTTP calls using Spring Cloud OpenFeign.
  - OpenFeign simplifies REST client creation and supports load balancing and fault tolerance via Eureka Service Discovery.

4. Identity and Access Management

This section covers the answer of the questions how the MicroBank backend is being protected and why those methods are used.

4.1 Keycloak Integration Overview

Keycloak serves as the central authentication and authorization provider for this project. Instead of implementing a custom authentication system, Keycloak offers out-of-the-box support for essential security features such as user authentication, role-based access control (RBAC), single sign-on (SSO), multi-factor authentication (MFA), and identity federation.

A pre-configured Keycloak realm named **`microbank`** is included in the project root directory as **`realm-export.json`**. This file allows developers to quickly set up Keycloak without manually creating users, roles, or clients.

Preconfigured Realm Features:

- **Realm Name:** `microbank`
- **User Roles:**
  - **USER** – Standard users with limited access.
  - **ADMIN** – Administrators with full control over the system.
- **Predefined Clients:**
  - **admin-cli** – Used for managing Keycloak configurations and user registrations.
  - **microbank-client** – Used by the backend services for authentication and authorization.
- **Client Credentials:**
  - The client ID and secret keys can be regenerated and updated in `application.yml` for each microservice.
- **Security Algorithms:**
  - Keycloak JWKs (JSON Web Keys) are encrypted using the RS256 signature algorithm, which is an asymmetric encryption mechanism that ensures high-security standards.

Keycloak supports multiple authentication and authorization protocols, including:

- **OAuth 2.0** – For API authentication and token-based authorization.
- **OpenID Connect (OIDC)** – A layer on top of OAuth 2.0, providing identity verification and user authentication.
- **SAML 2.0** – For single sign-on (SSO) integrations with enterprise identity providers.

These capabilities make Keycloak an ideal choice for securing modern microservice architectures.

4.2 Authentication Protocol

The project follows the OAuth 2.0 Authorization Framework, using Keycloak as the authorization server. The API Gateway validates JWT tokens issued by Keycloak before forwarding requests to internal microservices.

Authentication Flow:

1. **User Registration & Login:**
   - Users register via the `/api/v1/auth/register` endpoint.
   - After successful registration, users activate their accounts via `/api/v1/auth/activate`.
   - Users log in via `/api/v1/auth/login`, and Keycloak issues a JWT (JSON Web Token).
2. **Token Validation & API Access:**
   - The API Gateway verifies the JWT token and extracts user roles and permissions.
   - If the token is valid, the request is forwarded to the respective microservice.
   - If the token is invalid or expired, the request is rejected with a `401 Unauthorized` response.
3. **Role-Based Access Control (RBAC):**
   - **Keycloak roles** (`USER, ADMIN`) are mapped to Spring Security authorities (`ROLE_USER, ROLE_ADMIN`).
   - Microservices enforce authorization using Spring Security annotations or path-based access control.

The authentication mechanism relies on OAuth 2.0 Bearer Tokens, which are included in the `Authorization` header of each request:

```
Authorization: Bearer <JWT>
```

JWT tokens contain user claims, including roles, which are used for authorization at the microservice level.

4.3 Web Security and Web Flux Security

The project separates authentication (AuthN) from authorization (AuthZ) by implementing Spring Security differently in the API Gateway and microservices.

4.3.1 API Gateway (AuthN Layer)

- The API Gateway is reactive and built using Spring WebFlux.
- Reactive security mechanisms are required, so `@EnableWebFluxSecurity` is used instead of `@EnableWebSecurity`.
- Security is configured via `SecurityWebFilterChain:`

The security configuration of the API Gateway Service is constructed as follows:

```
@Configuration
@EnableWebFluxSecurity
@EnableReactiveMethodSecurity
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity serverHttpSecurity) {
        return serverHttpSecurity
                .csrf(ServerHttpSecurity.CsrfSpec::disable)
                .authorizeExchange(auth -> auth
                        .pathMatchers("/api/v1/auth/register").permitAll()
                        .pathMatchers("/api/v1/auth/activate").permitAll()
                        .pathMatchers("/api/v1/auth/login").permitAll()
                        .pathMatchers("/api/v1/auth/refresh-token").permitAll()
                        .pathMatchers("/api/v1/auth/forgot-password").permitAll()
                        .pathMatchers("/api/v1/auth/reset-password").permitAll()
                        .anyExchange().authenticated()
                )
                .oauth2ResourceServer(oauth2 -> oauth2.jwt())
                .build();
    }
}
```

The API Gateway:

- Authenticates all incoming requests using JWT tokens.
- Forwards only authenticated requests to microservices.
- Rejects unauthorized access attempts at the gateway level.

4.3.2 Microservices (Authorization Layer)

- Each microservice is **stateful** and uses **Spring MVC (non-reactive security)**.
- Standard `@EnableWebSecurity` and `HttpSecurity` are used instead of reactive security.
- Security is configured via `SecurityFilterChain`:

The security configuration of the API Gateway Service is constructed as follows:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Value("${spring.security.oauth2.resourceserver.jwt.jwk-set-uri}")
    private String jwkSetUri;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        return httpSecurity
                .csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/api/v1/users/**").hasRole("USER")
                        .requestMatchers("/api/v1/accounts/**").hasRole("USER")
                        .requestMatchers("/api/v1/transactions/**").hasRole("USER")
                        .requestMatchers("/api/v1/documents/**").hasRole("USER")
                        .requestMatchers("/api/v1/admin/users/**").hasRole("ADMIN")
                        .requestMatchers("/api/v1/admin/accounts/**").hasRole("ADMIN")
                        .requestMatchers("/api/v1/admin/transactions/**").hasRole("ADMIN")
                        .requestMatchers("/api/v1/admin/documents/**").hasRole("ADMIN")
                        .anyRequest().authenticated()
                )
                .oauth2ResourceServer(oauth2 -> oauth2
                        .jwt(jwt -> jwt.jwtAuthenticationConverter(jwtAuthenticationConverter())))
                .build();
    }

    @Bean
    public JwtDecoder jwtDecoder() {
        return NimbusJwtDecoder.withJwkSetUri(jwkSetUri).build();
    }

    @Bean
    public JwtAuthenticationConverter jwtAuthenticationConverter() {
        JwtAuthenticationConverter converter = new JwtAuthenticationConverter();
        converter.setJwtGrantedAuthoritiesConverter(new KeycloakRoleConverter());
        return converter;
    }
}


public class KeycloakRoleConverter implements Converter<Jwt, Collection<GrantedAuthority>> {

    @Override
    public Collection<GrantedAuthority> convert(Jwt jwt) {
        Map<String, Object> claims = jwt.getClaims();
        if (claims == null || !claims.containsKey("realm_access")) {
            return Collections.emptyList();
        }

        Map<String, Object> realmAccess = claims.get("realm_access");
        Collection<String> roles = realmAccess.getOrDefault("roles", Collections.emptyList());

        return roles.stream()
                .map(role -> new SimpleGrantedAuthority("ROLE_" + role))
                .collect(Collectors.toList());
    }
}
```

Microservices:

- Authorize requests based on user roles extracted from JWT tokens.
- Apply fine-grained authorization at the service level.
- Convert JWT roles into Spring Security authorities using `JwtAuthenticationConverter`.

4.3.3 Why WebFlux Security for API Gateway?

- The API Gateway is reactive, built using Spring WebFlux.
- Spring WebFlux does not support traditional `HttpSecurity`, so `ServerHttpSecurity` is required.
- API Gateway security must be non-blocking and fully asynchronous, which is why reactive security (WebFlux) is required.

4.3.4 Why Traditional Web Security for Microservices?

- Microservices handle business logic and persistent data, which work best with Spring MVC.
- Since microservices are not fully reactive, blocking security mechanisms (`HttpSecurity`) work better.
- Stateful authorization mechanisms are easier to manage in non-reactive microservices.

5. Endpoints Overview

There are 30 endpoints to be utilized in the MicroBank backend. Overview of all the endpoints are shown in the table below.

Function Name	Method	Endpoint URL	Authorization
Register User	`POST`	`/api/v1/auth/register`	`Public`
Activate User	`PATCH`	`/api/v1/auth/activate`	`Public`
Login User	`POST`	`/api/v1/auth/login`	`Public`
Refresh Token	`POST`	`/api/v1/auth/refresh-token`	`Public`
Forgot Password	`POST`	`/api/v1/auth/forgot-password`	`Public`
Reset Password	`PATCH`	`/api/v1/auth/reset-password`	`Public`
Get Current User's Profile	`GET`	`/api/v1/users/me`	`USER`
Get User by ID	`GET`	`/api/v1/admin/users/{userId}`	`ADMIN`
Get All Users	`GET`	`/api/v1/admin/users`	`ADMIN`
Update User Role	`PATCH`	`/api/v1/admin/users/{userId}/role`	`ADMIN`
Update User Access	`PATCH`	`/api/v1/admin/users/{userId}/access`	`ADMIN`
Delete User	`DELETE`	`/api/v1/admin/users/{userId}`	`ADMIN`
Create Bank Account	`POST`	`/api/v1/accounts`	`USER`
Update Account Balance	`PATCH`	`/api/v1/accounts/{accountId}/balance`	`USER`
Get Current User's Accounts	`GET`	`/api/v1/accounts`	`USER`
Get Current User's Account by ID	`GET`	`/api/v1/accounts/{accountId}`	`USER`
Get All Accounts	`GET`	`/api/v1/admin/accounts`	`ADMIN`
Get Accounts by User ID	`GET`	`/api/v1/admin/users/{userId}/accounts`	`ADMIN`
Update Account Status	`PATCH`	`/api/v1/admin/accounts/{accountId}/status`	`ADMIN`
Delete Own Account	`DELETE`	`/api/v1/accounts/{accountId}`	`USER`
Delete Account	`DELETE`	`/api/v1/admin/accounts/{accountId}`	`ADMIN`
Create Transaction	`POST`	`/api/v1/transactions`	`USER`
Get Current User's All Transactions	`GET`	`/api/v1/transactions`	`USER`
Get Current User's Transaction by ID	`GET`	`/api/v1/transactions/{transactionId}`	`USER`
Get Current User's Transactions by Account ID	`GET`	`/api/v1/accounts/{accountId}/transactions`	`USER`
Get All Transactions	`GET`	`/api/v1/admin/transactions`	`ADMIN`
Get Transactions by ID	`GET`	`/api/v1/admin/transactions/{transactionId}`	`ADMIN`
Get User's Transactions by Account ID	`GET`	`/api/v1/admin/accounts/{accountId}/transactions`	`ADMIN`
Create Transaction Receipt PDF	`POST`	`/api/v1/documents`	`USER`
Download Transaction Receipt PDF	`GET`	`/api/v1/documents/{documentId}`	`USER`

6. Microservices

The backend architecture of this project is based on the microservices paradigm, which ensures scalability, maintainability, and fault tolerance. Each microservice is responsible for a specific domain or functionality, and the overall architecture leverages cutting-edge technologies such as Docker, RabbitMQ, PostgreSQL, Redis, Keycloak, and MinIO. Below is a detailed breakdown of each microservice and its responsibilities.

6.1 API Gateway Service

The API Gateway serves as the entry point for all external clients, including mobile and web applications. It routes requests to the appropriate microservices while handling authentication. By integrating with Keycloak, it ensures secure authentication and token validation. Additionally, it simplifies service discovery and communication by acting as a reverse proxy, allowing clients to interact with microservices without needing direct access to them.

As a matter of fact, the base URL of each microservice is distinct since they are separate backend projects and use different ports. For example:

- Authentication Microservice is running on port **`8081`** thus the base URL is **`http://localhost:8081/api/v1,`**
- Account Microservice is running on port **`8082`** thus the base URL is **`http://localhost:8082/api/v1,`**
- Transaction Microservice is running on port **`8083`** thus the base URL is **`http://localhost:8083/api/v1,`**
- Document Microservice is running on port **`8084`** thus the base URL is **`http://localhost:8084/api/v1`**

The base URL of the API Gateway is:

```
http://localhost:8123/api/v1
```

Every endpoint ready to be utilized by the client is available on this URL.

Eureka Discovery (**`8761`**) and Notification (**`8085`**) microservices are also running on a different port, Yet their endpoints are not being redirected to the API Gateway.

6.2 Eureka Discovery Service

The Eureka Discovery Service acts as a service registry, enabling dynamic registration and discovery of microservices. Instead of relying on static IP addresses or hardcoded service endpoints, microservices register themselves with Eureka, allowing them to discover and communicate with each other dynamically. This enhances fault tolerance and scalability, as services can be added or removed without requiring manual configuration updates.

6.3 Authentication Service

The Authentication Service is responsible for handling user authentication. It integrates with Keycloak to manage user credentials, token issuance, and access control. Additionally, it produces events to RabbitMQ queues for handling user activation and password reset requests asynchronously.

6.3.1 Register User `POST` `{baseURL}/auth/register`

Registers a new user and produces an activation code to be sent to the user's email address, caches the user data in the Redis along with the activation code for 15 minutes, tops, or till the user activates their account.

No path variable or header is required.

Request Body Field	Type	Mandatory
`username`	`String`	`Yes`
`firstName`	`String`	`Yes`
`lastName`	`String`	`Yes`
`email`	`String`	`Yes`
`password`	`String`	`Yes`

Example Request

```
{
    "username": "capellax",
    "firstName": "Ahmet",
    "lastName": "ATAR",
    "email": "ahmet@email.com",
    "password": "123456"
}
```

Example Response

```
{
    "success": true,
    "message": "Registration successful, activation code sent to ahmet@email.com",
    "data": null,
    "timestamp": "2025-01-04T17:27:16.893505158",
    "errors": null
}
```

6.3.2 Activate User `PATCH` `{baseURL}/auth/activate`

Activates the user's account, clears the cached user data and permanently saves it to the actual database, also completes the registration process in the Keycloak realm so Single Sign-On can be effectively utilized across the permitted microservices.

No path variable or header is required.

Request Body Field	Type	Mandatory
email	String	Yes
activationCode	String	Yes

Example Request

```
{
    "email": "ahmet@email.com",
    "activationCode": "348254"
}
```

Example Response

```
{
    "success": true,
    "message": "Account activation successful, you can login to the system.",
    "data": null,
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.3.3 Login `POST` `{baseURL}/auth/login`

Users can log in to the system using this endpoint and get their access and refresh tokens. In MicroBank, users can access all authenticated endpoints, of course if they have an appropriate role, using the access token returned from this endpoint since Keycloak's Single Sign-On (SSO) support across the microservices.

No path variable or header is required.

Request Body Field	Type	Mandatory
username	String	Yes
password	String	Yes

Example Request

```
{
    "username": "capellax",
    "password": "123456"
}
```

Example Response

```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJDbGhsT3JRS...",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI0Y2JjMTxJ...",
    "token_type": "Bearer",
    "not-before-policy": 0,
    "session_state": "c4645297-4886-4607-b88e-ec5f17cf0564",
    "scope": "profile email"
}
```

6.3.4 Forgot Password `POST` `{baseURL}/auth/forgot-password`

Sends 6-digit password recovery code to the provided email address. In the next endpoint, this code is required to reset the user's password.

No path variable or header is required.

Request Body Field	Type	Mandatory
email	String	Yes

Example Request

```
{
    "email": "ahmet@email.com"
}
```

Example Response

```
{
    "success": true,
    "message": "Password recovery code sent to ahmet@email.com",
    "data": null,
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.3.5 Reset Password `PATCH` `{baseURL}/auth/reset-password`

Resets the user's password using their email information as well as the 6-digit reset password code that had been sent to their email address earlier.

No path variable or header is required.

Request Body Field	Type	Mandatory
email	String	Yes
resetPasswordCode	String	Yes

Example Request

```
{
    "email": "ahmet@email.com",
    "newPassword": "753123",
}
```

Example Response

```
{
    "success": true,
    "message": "Password reset successful, you can log back into the system.",
    "data": null,
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.3.6 Refresh Token `POST` `{baseURL}/auth/refresh-token`

Refreshes the user's access token using the refresh token. This feature can be utilized on the client side by setting up an automated token refreshing logic in order to keep user's session times longer.

No path variable or header is required.

Request Body Field	Type	Mandatory
refreshToken	String	Yes

Example Request

```
{
    "refreshToken": "eyJhbGciOiJIUzAbCiwia2xJpr..."
}
```

Example Response

```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwiCJ...",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwiC...",
    "token_type": "Bearer",
    "not-before-policy": 0,
    "session_state": "c4645297-4886-4607-b88e-ec5f17cf0564",
    "scope": "profile email"
}
```

6.3.6 Get Current User's Profile `GET` `{baseURL}/users/me`

Authenticated users with the role USER can fetch their own information using this endpoint.

No request body or path variable is required.

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,
    "message": "Current user retrieved successfully.",
    "data": {
        "id": "501ee3cd-0b68-4d8e-aeef-110bdf3fdd47",
        "username": "capellax",
        "email": "ahmet@email.com",
        "firstName": "Ahmet",
        "lastName": "ATAR"
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.3.8 Get All Users `GET` `{baseURL}/admin/users`

Retrieves all users in the system. Accessible only by ADMIN users.

No path variable or header is required.

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,3facd6bd-c6ef-4eee-b40b-d8be4f2ed105
    "message": "Users retrieved successfully.",
    "data": [
        {
            "id": "501ee3cd-0b68-4d8e-aeef-110bdf3fdd47",
            "username": "capellax",
            "email": "ahmet@email.com",
            "firstName": "Ahmet",
            "lastName": "ATAR",
            "role": "USER",
            "isBanned": false,
```

```
        },
        {
            "id": "501ee3cd-0b68-4d8e-aeef-110bdf3fdd47",
            "username": "doejohn",
            "email": "john@email.com",
            "firstName": "John",
            "lastName": "DOE",
            "role": "USER",
            "isBanned": false,
        },
        ...
    ],
    "timestamp": "2024-12-12T17:21:14.893505158",
    "errors": null
}
```

6.3.9 Get User by ID `GET` `{baseURL}/admin/users/{userId}`

Retrieves a particular user in the system. Accessible only by ADMIN users.

No request body is required.

Path Variable	Type	Mandatory
userId	String	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,
    "message": "User with the ID: 501ee3cd-0b68-4d8e-aeef-110bdf3fdd47 retrieved successfully.",
    "data": {
        "id": "501ee3cd-0b68-4d8e-aeef-110bdf3fdd47",
        "username": "capellax",
        "email": "ahmet@email.com",
        "firstName": "Ahmet",
        "lastName": "ATAR",
        "role": "USER",
        "isBanned": false,
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.3.10 Update User Role `PATCH` `{baseURL}/admin/users/{userId}/role`

Updates the role of a particular user. Accessible only by ADMIN users.

Path Variable	Type	Mandatory
userId	String	Yes
Request Body Field	**Type**	**Mandatory**
newRole	String	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Request:

```
{
    "newRole": "ADMIN"
}
```

Example Response:

```
{
    "success": true,
    "message": "User with the ID: 501ee3cd-0b68-4d8e-aeef-110bdf3fdd47 retrieved successfully.",
    "data": {
        "id": "501ee3cd-0b68-4d8e-aeef-110bdf3fdd47",
        "username": "capellax",
        "email": "ahmet@email.com",
        "firstName": "Ahmet",
        "lastName": "ATAR",
        "role": "ADMIN",
        "isBanned": false,
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.3.11 Update User Access `PATCH` `{baseURL}/admin/users/{userId}/access`

ADMIN users can ban or unban a user using this endpoint.

No request body is required.

Path Variable	Type	Mandatory
userId	String	Yes
Request Body Field	**Type**	**Mandatory**
isBanned	Boolean	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Request:

```
{
    "isBanned": true
}
```

Example Response:

```
{
    "success": true,
    "message": "User with the ID: 501ee3cd-0b68-4d8e-aeef-110bdf3fdd47 retrieved successfully.",
    "data": {
        "id": "501ee3cd-0b68-4d8e-aeef-110bdf3fdd47",
        "username": "capellax",
        "email": "ahmet@email.com",
        "firstName": "Ahmet",
        "lastName": "ATAR",
        "role": "USER",
        "isBanned": true,
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.3.12 Delete User `DELETE` `{baseURL}/admin/users/{userId}`

Deletes a particular user permanently from the system. Accessible only by ADMIN users.

No request body is required.

Path Variable	Type	Mandatory
userId	String	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,
    "message": "User with the ID: 123aa1bc-0c12-3c7f-efde-230bdf3fee49 deleted successfully.",
    "data": null,
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.4 Account Service

The Account Service manages user accounts, including account creation, retrieval, and balance updates. It integrates with PostgreSQL for storing account-related data and enforces RBAC (Role-Based Access Control) via Keycloak. This service is exposed through the API Gateway, allowing clients to manage accounts securely.

6.4.1 Create Account `POST` `{baseURL}/accounts`

Authenticated users can create a new bank account using this endpoint. If `initialBalance` is not provided, the account will be created with zero balance. Users can deposit money later on.

No path variable is required.

Request Body Field	Type	Mandatory
initialBalance	Integer	No
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Request:

```
{
    "initialBalance": 1000
}
```

Example Response:


```
{
    "success": true,
    "message": "New bank account created successfully.",
    "data": {
        "id": "3facd6bd-c6ef-4eee-b40b-d8be4f2ed105",
        "IBAN": "MB249628087271",
        "balance": 1000.00,
        "isBlocked": false,
        "ownerName": "AHMET ATAR",
        "ownerId": "b1d12fee-a32a-4c7a-8633-acc716cb0502"
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```


6.4.2 Update Account Balance `PATCH` `{baseURL}/accounts/{accountId}`

Users can deposit money to their account or withdraw money from their account using this endpoint. If the value of the `isDeposit` field is `true`, this is a deposit action, if `false`, then it is a withdrawal.

Path Variable	Type	Mandatory
accountId	String	Yes
Request Body Field	**Type**	**Mandatory**
amount	Integer	Yes
isDeposit	Boolean	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Request

```
{
    "amount": 300,
    "isDeposit": true
}
```

Example Response


```
{
    "success": true,
    "message": "Account balance updated successfully.",
    "data": {
        "id": "3facd6bd-c6ef-4eee-b40b-d8be4f2ed105",
        "IBAN": "MB249628087271",
        "balance": 1300.00,
        "isBlocked": false,
        "ownerName": "AHMET ATAR",
        "ownerId": "b1d12fee-a32a-4c7a-8633-acc716cb0502"
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```


6.4.3 Get Current User's Accounts `GET` `{baseURL}/accounts`

Users can retrieve their own accounts' information using this endpoint.

No request body or path variable is required.

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "Current user's accounts fetched successfully.",
    "data": [
        {
            "id": "3facd6bd-c6ef-4eee-b40b-d8be4f2ed105",
            "IBAN": "MB249628087271",
            "balance": 1300.00,
            "isBlocked": false,
            "ownerName": "AHMET ATAR",
            "ownerId": "b1d12fee-a32a-4c7a-8633-acc716cb0502"
        },
        {
            "id": "92fa818e-178d-46ae-af1f-4d70e04674fe",
            "IBAN": "MB297812143253",
            "balance": 2500.00,
            "isBlocked": false,
            "ownerName": "AHMET ATAR",
            "ownerId": "b1d12fee-a32a-4c7a-8633-acc716cb0502"
        }
    ],
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.4.4 Get Current User's Account by ID `GET` `{baseURL}/accounts/{accountId}`

Users can retrieve the information of one particular account that they owe using this endpoint.

No request body is required.

Path Variable	Type	Mandatory
accountId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "Current user's account with the ID: 3facd6bd-c6ef-4eee-b40b-d8be4f2ed105 retrieved successfully.",
    "data": {
            "id": "92fa818e-178d-46ae-af1f-4d70e04674fe",
            "IBAN": "MB297812143253",
            "balance": 2500.00,
            "isBlocked": false,
            "ownerName": "AHMET ATAR",
            "ownerId": "b1d12fee-a32a-4c7a-8633-acc716cb0502"
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```


6.4.5 Get All Accounts `GET` `{baseURL}/admin/accounts`

Retrieves all accounts and can only be utilized by ADMIN users.

No request body or path variable is required.

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "All accounts retrieved successfully.",
    "data": [
        {
            "id": "92fa818e-178d-46ae-af1f-4d70e04674fe",
            "IBAN": "MB297812143253",
            "balance": 2500.00,
            "isBlocked": false,
            "ownerName": "AHMET ATAR",
            "ownerId": "b1d12fee-a32a-4c7a-8633-acc716cb0502"
        },
        {
            "id": "b2b7af3f-8de0-4b6c-bf1c-55515c13c3b0",
            "IBAN": "MB660355043532",
            "balance": 1250.00,
            "isBlocked": false,
            "ownerName": "JOHN DOE",
            "ownerId": "f7e21c55-9d9e-4c00-9bc2-57873346486d"
        },
        ...
    ],
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```


6.4.6 Get Account by ID `GET` `{baseURL}/admin/accounts/{accountId}`

Retrieves a particular account. Only ADMIN users can utilize this endpoint.

No request body is required.

Path Variable	Type	Mandatory
accountId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "Account with the ID: b2b7af3f-8de0-4b6c-bf1c-55515c13c3b0 retrieved successfully.",
    "data": {
            "id": "b2b7af3f-8de0-4b6c-bf1c-55515c13c3b0",
```

```
            "IBAN": "MB660355043532",
            "balance": 1250.00,
            "isBlocked": false,
            "ownerName": "JOHN DOE",
            "ownerId": "f7e21c55-9d9e-4c00-9bc2-57873346486d"
    },
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```


6.4.6 Get Accounts by User ID `GET` `{baseURL}/admin/users/{userId}/accounts`

Retrieves a user's accounts. Only ADMIN users can utilize this endpoint.

No request body is required.

Path Variable	Type	Mandatory
userId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "Accounts belong to user with ID: f7e21c55-9d9e-4c00-9bc2-57873346486d retrieved successfully.",
    "data": [
        {
            "id": "b2b7af3f-8de0-4b6c-bf1c-55515c13c3b0",
            "IBAN": "MB660355043532",
            "balance": 1250.00,
            "isBlocked": false,
            "ownerName": "JOHN DOE",
            "ownerId": "f7e21c55-9d9e-4c00-9bc2-57873346486d"
        },
        {
            "id": "6264906c-7740-4393-a384-e559fb780a58",
            "IBAN": "MB228553314137",
            "balance": 2000.00,
            "isBlocked": false,
            "ownerName": "JOHN DOE",
            "ownerId": "f7e21c55-9d9e-4c00-9bc2-57873346486d"
        }
    ],
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```


6.4.7 Update Account Status `PATCH` `{baseURL}/admin/accounts/{accounts}/status`

Blocks or reactivates a bank account. Only ADMIN users can utilize this endpoint.

Path Variable	Type	Mandatory
accountId	String	Yes

Request Body Field	Type	Mandatory
isBlocked	Boolean	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Request:


```
{
    "isBlocked": true
}
```


Example Response:


```
{
    "success": true,
    "message": "Status of the account with the ID: 6264906c-7740-4393-a384-e559fb780a58 updated successfully.",
    "data": {
            "id": "6264906c-7740-4393-a384-e559fb780a58",
            "IBAN": "MB228553314137",
            "balance": 2000.00,
            "isBlocked": true,
            "ownerName": "JOHN DOE",
            "ownerId": "f7e21c55-9d9e-4c00-9bc2-57873346486d"
    }
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.4.6 Delete Own Account `DELETE` `{baseURL}/accounts/{accountId}`

Authenticated users can permanently delete their particular bank account.

No request body is required.

Path Variable	Type	Mandatory
accountId	String	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,
    "message": "Current user's account with the ID: 6264906c-7740-4393-a384-e559fb780a58 deleted successfully.",
    "data": null,
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.4.6 Delete Account by ID `DELETE` `{baseURL}/admin/accounts/{accountId}`

Deletes a bank account using ADMIN authority.

No request body is required.

Path Variable	Type	Mandatory
accountId	String	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,
    "message": "Account with the ID: b2b7af3f-8de0-4b6c-bf1c-55515c13c3b0 deleted successfully.",
    "data": null,
    "timestamp": "2024-11-17T17:27:16.893505158",
    "errors": null
}
```

6.5 Transaction Service

The system architecture revolves around several independent microservices that are orchestrated and interconnected via an API Gateway. The diagram outlines how these services communicate with each other and external systems. Central components include service discovery, authentication, database integration, message queues, and object storage.

6.5.1 Create Transaction `POST` `{baseURL}/transactions`

Users can send money to another account using this endpoint.

No path variable is required.

Request Body Field	Type	Mandatory
targetAccountIBAN	String	Yes
amount	String	Yes
description	String	Yes
Header	**Type**	**Mandatory**
Authorization	Bearer Token	Yes

Example Request

```
{
    "targetAccountIBAN": "MB660355043532",
    "amount": 250,
    "description": "Rent money for January."
}
```

Example Response

```
{
    "success": true,
    "message": "Transaction completed successfully.",
    "data": {
        "id": "eeafee9b-4320-495b-ba2f-8ae1138d5bee",
        "sourceAccountIBAN": "MB249628087271",
        "targetAccountIBAN": "MB660355043532",
        "amount": 250.00,
        "description": "Rent money for January.",
        "timestamp": "2025-01-17T17:27:17.158163752"
    },
    "timestamp": "2025-01-17T17:27:16.893505158",
    "errors": null
}
```

6.5.2 Get Current User's All Transactions `GET` `{baseURL}/transactions`

Users can retrieve all the transactions they made. The response contains both TRANSFERRED and RECEIVED transactions.

No request body or path variable is required.

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "Current user's past transactions retrieved successfully.",
    "data": [
        {
            "id": "eeafee9b-4320-495b-ba2f-8ae1138d5bee",
            "sourceAccountIBAN": "MB249628087271",
            "targetAccountIBAN": "MB660355043532",
            "amount": 250.00,
            "description": "Rent money for January.",
            "timestamp": "2025-01-05 15:23:37.497873"
        },
        {
            "id": "f2d912dd-e2a6-4b94-8acf-8d41739eb594",
            "sourceAccountIBAN": "MB660355043532",
            "targetAccountIBAN": "MB249628087271",
            "amount": 250.00,
            "description": "Rent money for January.",
            "timestamp": "2025-01-05 15:23:37.541563"
        },
        ...
    ],
    "timestamp": "2025-01-17T17:28:16.893505158",
    "errors": null
}
```


6.5.3 Get Current User's Transaction by ID `GET` `{baseURL}/transactions/{transactionId}`

Users can retrieve their particular transaction information using this endpoint.

No request body is required.

Path Variable	Type	Mandatory
transactionId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "Past transactions of the user with the ID: b1d12fee-a32a-4c7a-8633-acc716cb0502 retrieved successfully.",
    "data": [
        {
            "id": "eeafee9b-4320-495b-ba2f-8ae1138d5bee",
            "sourceAccountIBAN": "MB249628087271",
            "targetAccountIBAN": "MB660355043532",
            "amount": 850.00,
            "description": "Rent money for January.",
            "timestamp": "2025-01-05 15:23:37.497873"
        },
        {
            "id": "f2d912dd-e2a6-4b94-8acf-8d41739eb594",
            "sourceAccountIBAN": "MB157859872002",
            "targetAccountIBAN": "MB249628087271",
            "amount": 300.00,
            "description": "Personal payment.",
            "timestamp": "2025-01-08 12:41:08.541563"
        },
        ...
    ],
    "timestamp": "2025-01-17T17:28:16.893505158",
    "errors": null
}
```


6.5.4 Get Current User's Transactions by Account ID `GET` `{baseURL}/accounts/{accountId}/transactions`

Users can retrieve their particular transaction information using this endpoint.

No request body is required.

Path Variable	Type	Mandatory
accountId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "Past transactions belong the current user’s account with the ID: 3facd6bd-c6ef-4eee-b40b-d8be4f2ed105
retrieved successfully.",
    "data": [
        {
            "id": "d4c321ba-4320-495b-ba2f-8ae1138d2bac",
            "sourceAccountIBAN": "MB249628087271",
            "targetAccountIBAN": "MB660355043532",
            "amount": 500.00,
            "description": "Rent money for February",
            "timestamp": "2025-02-01 12:19:39.297873"
        },
        {
            "id": "a1b123cd-e2a6-4b94-8acf-8d41739e35ca",
            "sourceAccountIBAN": "MB157859872002",
            "targetAccountIBAN": "MB249628087271",
            "amount": 300.00,
            "description": "Personal loan",
            "timestamp": "2025-01-08 12:41:08.541563"
        },
        ...
    ],
    "timestamp": "2025-01-17T17:28:16.893505158",
    "errors": null
}
```


6.5.5 Get All Transactions `GET` `{baseURL}/admin/transactions`

ADMIN users can retrieve all the transactions that have been made.

No request body or path variable is required.

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:


```
{
    "success": true,
    "message": "All past transactions in the system retrieved successfully.",
    "data": [
        {
            "id": "d4c321ba-4320-495b-ba2f-8ae1138d2bac",
            "sourceAccountIBAN": "MB249628087271",
            "targetAccountIBAN": "MB660355043532",
            "amount": 3500.00,
            "description": "Some description",
            "timestamp": "2025-03-05 12:41:08.541563"
        },
        {
            "id": "a1b123cd-e2a6-4b94-8acf-8d41739e35ca",
            "sourceAccountIBAN": "MB854435893450",
            "targetAccountIBAN": "MB034958300531",
            "amount": 2250.00,
            "description": "Some description",
            "timestamp": "2025-03-05 12:19:39.297873"
        },
        ...
    ],
    "timestamp": "2025-01-17T17:28:16.893505158",
    "errors": null
}
```


6.5.6 Get Transaction by ID `GET` `{baseURL}/admin/transactions/{transactionId}`

ADMIN users can retrieve any particular transaction information.

No request body is required.

Path Variable	Type	Mandatory
transactionId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,
    "message": "Transaction with the ID: eeafee9b-4320-495b-ba2f-8ae1138d5bee retrieved successfully.",
    "data": [
        {
            "id": "eeafee9b-4320-495b-ba2f-8ae1138d5bee",
            "sourceAccountIBAN": "MB249628087271",
            "targetAccountIBAN": "MB660355043532",
            "amount": 250.00,
            "description": "Rent money for January.",
            "timestamp": "2025-01-05 15:23:37.497873"
        }
    ],
    "timestamp": "2025-01-17T17:28:16.893505158",
    "errors": null
}
```

6.5.7 Get Transactions by Account ID `GET` `{baseURL}/admin/accounts/{accountId}/transactions`

ADMIN users can retrieve transactions belonging to a particular account.

No request body is required.

Path Variable	Type	Mandatory
accountId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Response:

```
{
    "success": true,
    "message": "Past transaction of account with the ID: 3facd6bd-c6ef-4eee-b40b-d8be4f2ed105 retrieved successfully.",
    "data": [
        {
            "id": "d4c321ba-4320-495b-ba2f-8ae1138d2bac",
            "sourceAccountIBAN": "MB249628087271",
            "targetAccountIBAN": "MB660355043532",
            "amount": 500.00,
            "description": "Rent money for February",
            "timestamp": "2025-02-01 12:19:39.297873"
        },
        {
            "id": "a1b123cd-e2a6-4b94-8acf-8d41739e35ca",
            "sourceAccountIBAN": "MB157859872002",
            "targetAccountIBAN": "MB249628087271",
            "amount": 300.00,
            "description": "Personal loan",
            "timestamp": "2025-01-08 12:41:08.541563"
        },
        ...
    ],
    "timestamp": "2025-01-17T17:28:16.893505158",
    "errors": null
}
```

6.6 Document Service

The system architecture revolves around several independent microservices that are orchestrated and interconnected via an API Gateway. The diagram outlines how these services communicate with each other and external systems. Central components include service discovery, authentication, database integration, message queues, and object storage.

6.6.1 Create Transaction Receipt PDF `POST` `{baseURL}/documents/transactions`

Description here.

No path variable is required.

Request Body Field	Type	Mandatory
transactionId	String	Yes

Header	Type	Mandatory
Authorization	Bearer Token	Yes

Example Request:

```
{
    "transactionId": "eeafee9b-4320-495b-ba2f-8ae1138d5bee"
}
```

Example Response:


```
{
    "success": true,
    "message": "Receipt PDF for the transaction with the ID: eeafee9b-4320-495b-ba2f-8ae1138d5bee created successfully.",
    "data": {
        "documentUrl": "http://localhost:9000/documents/TRANSACTION-eeafee9b-4320-495b-ba2f-8ae1138d5bee.pdf?..."
    },
    "timestamp": "2025-01-17T17:28:16.893505158",
    "errors": null
}
```


6.6.2 Download Transaction Receipt PDF `GET` `{baseURL}/documents/transactions/{transactionId}`

Description here.

No request body is required.

Path Variable	Type	Mandatory
`transactionId`	`String`	`Yes`

Header	Type	Mandatory
`Authorization`	`Bearer Token`	`Yes`

The response will be the transaction receipt PDF directly.

6.7 Notification Service

activation-queue
reset-password-queue
transaction-queue