Java developer microservices

Para accede a la versión web de este documento: https://pw-0223.notion.site/Java-developer-microservices-1875e48ff3ff808c86b0f9b7f8e9033a?pvs=4

Para la presente prueba se opto por crear un enfoque Data Base First, donde se modelo una base de datos en SQL server para almacenar los registros de las entidades principales: `users`, `accounts` y `account_transactions`

Diagrama relacional de base de datos completa

Si toda la base de datos de la solución estuviera centralizada el acercamiento propuesto seria el siguiente.

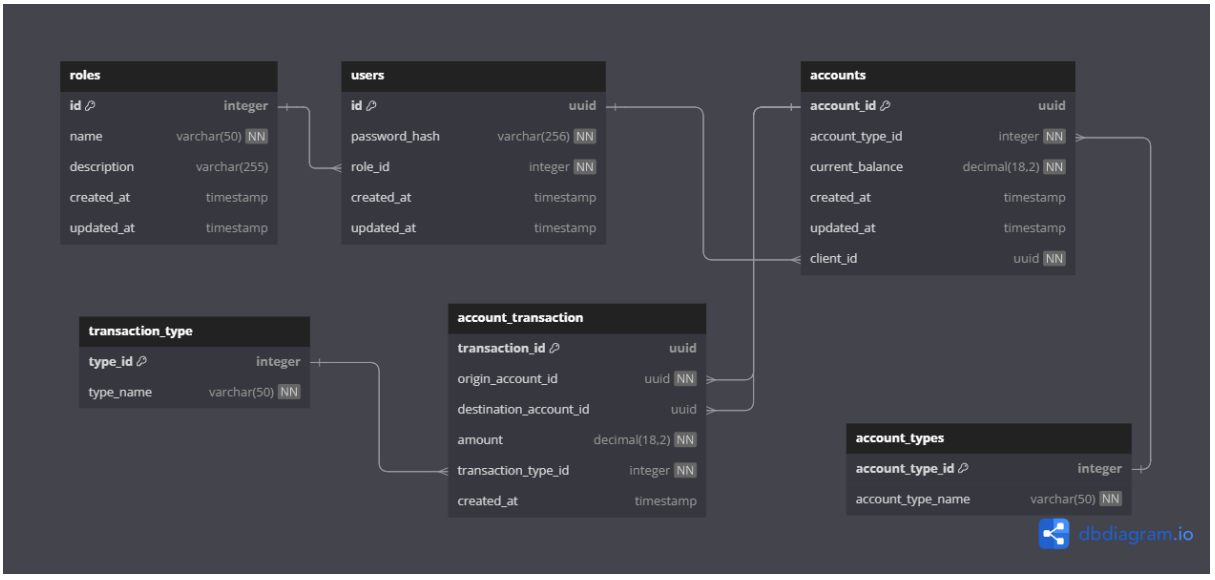
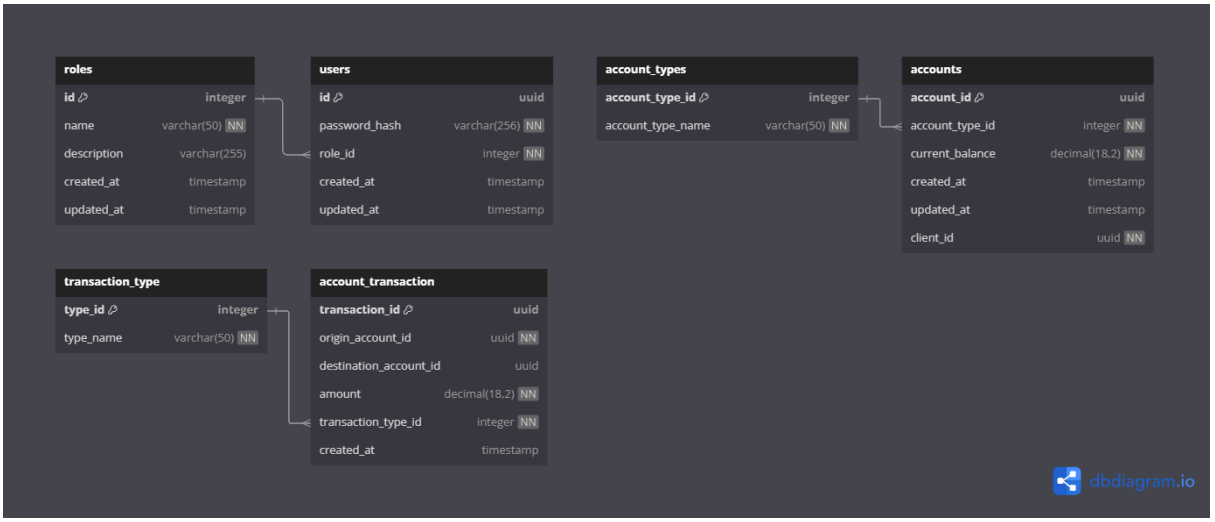


Diagrama relacional de base de datos distribuida para microservicios

Ya que se esta abordando una arquitectura por microservicios se opto en segmentar la base de datos en 3, leyendo los diagramas de izquierda a derecha y arriba a abajo se segmento de la siguiente manera:



Nota: el diagrama contiene 3 islas de tablas, cada una corresponde a un microservicio.

DLL de bases de datos

users_database para ***auth_service***

```
-- DATABASE users

CREATE TABLE roles (
    id INT IDENTITY(1,1) PRIMARY KEY,
  name NVARCHAR(50) NOT NULL UNIQUE,
  description NVARCHAR(255) NULL,
  created_at DATETIMEOFFSET DEFAULT SYSDATETIMEOFFSET() NOT NULL,
  updated_at DATETIMEOFFSET DEFAULT SYSDATETIMEOFFSET() NOT NULL
);

CREATE TABLE users (
    id UNIQUEIDENTIFIER DEFAULT NEWID() PRIMARY KEY,
  password_hash NVARCHAR(256) NOT NULL,
  role_id INT NOT NULL FOREIGN KEY REFERENCES roles(id),
  created_at DATETIMEOFFSET DEFAULT SYSDATETIMEOFFSET() NOT NULL,
  updated_at DATETIMEOFFSET DEFAULT SYSDATETIMEOFFSET() NOT NULL
);

INSERT INTO roles (name)
VALUES
    ('ADMIN'),
    ('USER');
```

bankaccounts para ***bank_accounts_service***

```
-- DATABASE bankaccounts

CREATE TABLE account_types (
    account_type_id INT PRIMARY KEY IDENTITY(1,1),
  account_type_name NVARCHAR(50) NOT NULL
);

CREATE TABLE accounts (
    account_id UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
  account_type_id INT NOT NULL,
  current_balance DECIMAL(18, 2) NOT NULL,
  created_at DATETIMEOFFSET NOT NULL DEFAULT SYSDATETIMEOFFSET(),
  updated_at DATETIMEOFFSET NOT NULL DEFAULT SYSDATETIMEOFFSET(),
  client_id UNIQUEIDENTIFIER NOT NULL,
  CONSTRAINT fk_account_type FOREIGN KEY (account_type_id) REFERENCE
);
```

account-transactions para ***transactions_service***

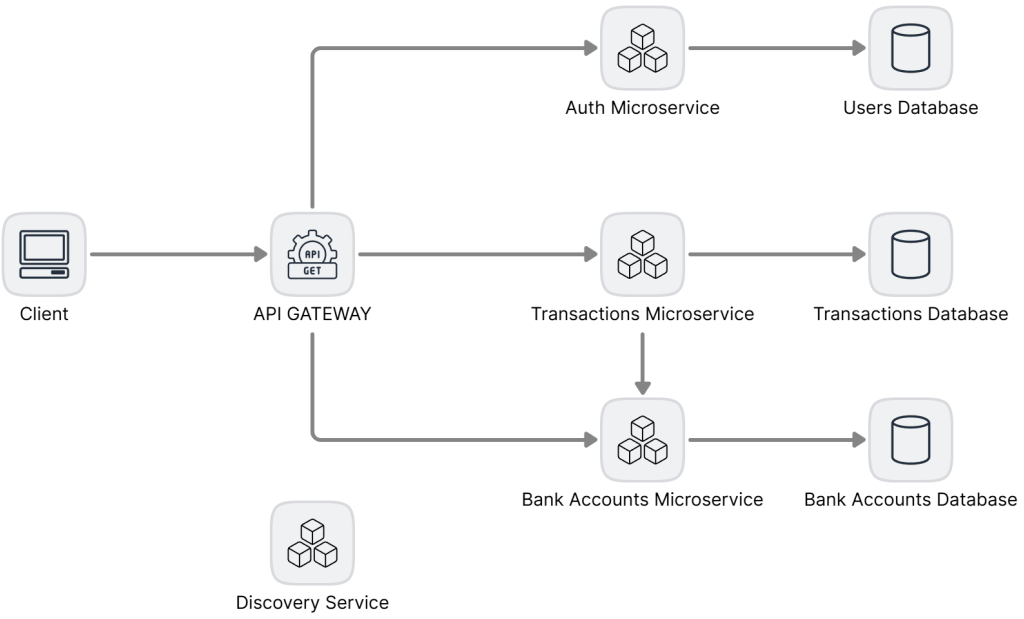
```
-- DATABASE account-transactions

CREATE TABLE transaction_type (
    type_id INT PRIMARY KEY IDENTITY(1,1),
    type_name NVARCHAR(50) NOT NULL
);

CREATE TABLE account_transaction (
    transaction_id UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    origin_account_id UNIQUEIDENTIFIER NOT NULL,
    destination_account_id UNIQUEIDENTIFIER NULL,
    amount DECIMAL(18,2) NOT NULL,
    transaction_type_id INT NOT NULL,
    created_at DATETIMEOFFSET NOT NULL DEFAULT SYSDATETIMEOFFSET(),
    CONSTRAINT fk_account_transaction_type FOREIGN KEY (transaction_
);

INSERT INTO transaction_type (type_name) VALUES
    ('Abono'),
    ('Retiro'),
    ('Transferencia');
```

Diagrama de arquitectura de microservicios



La arquitectura propuesta cuenta con los diferentes componentes:

- API Gateway
- Discovery Server
- Auth Microsevice
- Users Database
- Transactions Microsevice
- Transactions Database
- Bank Accounts Microsevice
- Bank Accounts Database

Sobre secretos y variables de entorno para microservicios

- Auth Microsevice

```
// .env
SPRING_DATASOURCE_URL=
SPRING_DATASOURCE_USERNAME=
SPRING_DATASOURCE_PASSWORD=
```

```
jwt.secret-key=
jwt.expiration.auth=
```

- Bank Accounts Microsevice y Transactions Microsevice

```
// .env
SPRING_DATASOURCE_URL=
SPRING_DATASOURCE_USERNAME=
SPRING_DATASOURCE_PASSWORD=
jwt.secret-key=
```

Ejecutar microservicios

Los microservicios fueron creados utilizando:

- SpringBoot 2.7.17
- Java 11
- Gradle
- IDE IntelliJ

Para correr los microservicios, se puede abrir cada directorio utilizando IntelliJ y ejecutarlo, por el alcance de esta prueba no se necesita agregar ningún profije o configuración adicional, salvo por el archivo `.env` que debe tener las configuraciones pertinentes.

También se pueden ejecutar desde la consola accediendo a los directorios de cada microservicio y utilizando el comando:

```
./gradlew bootRun
```

Ejemplo:

```
PS C:\Users\carlo\Documents\spring-micro-services\bank-accounts-serv
```

Notas adicionales

La documentación de swagger accede a los microservicios desde el puerto destinado para cada microservicio, sin embargo si se quiere acceder a ellos usando el **API GATEWAY**, de deberá apuntar al puerto `8080`

También se incluye una colección de request de Insomnia (`insomnia.json`) por cualquier eventualidad o por si se desea probar las requests directamente al **API_GATEWAY**.

En caso de obtener un error **503 Service Unavailable** es debido a que los micro servicios necesitan tiempo para registrarse en el discovery server y ser visibles para

todos los demás servicios en el ecosistema.