ESTADO DE MINAS GERAIS
UNIVERSIDADE ESTADUAL DE MONTES CLAROS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Unimontes

T3 - MICROSSERVIÇOS COM SPRING BOOT

SAMUEL FREITAS DE OLIVEIRA

MONTES CLAROS/MG
2025/01

SAMUEL FREITAS DE OLIVEIRA

T3 - MICROSSERVIÇOS COM SPRING BOOT

Trabalho acadêmico apresentado à disciplina de Engenharia de Software II do 4º período do Curso de Bacharelado em Sistemas de Informação, como parte das exigências para a aprovação.

Disciplina: Engenharia de Software II

Prof. Allysson Costa e Silva

MONTES CLAROS/MG

2025/01

LISTA DE FIGURAS

FIGURAS

LISTA DE LISTAGENS

LISTAGENS

SUMÁRIO

1. INTRODUÇÃO

A arquitetura baseada em microsserviços tem se destacado no desenvolvimento de sistemas distribuídos por permitir maior escalabilidade, modularização e independência entre componentes. Segundo Richardson (2019), essa abordagem organiza uma aplicação como um conjunto de serviços pequenos, autônomos e implementados em torno de capacidades de negócios específicas. Cada microsserviço é responsável por sua própria lógica e por seu próprio banco de dados, o que reduz o acoplamento e melhora a manutenção do sistema.

No presente trabalho, foi implementado um exemplo funcional de comunicação entre microsserviços utilizando o framework Spring Boot, com base no tutorial disponibilizado por JavaGuides (2022). O cenário proposto simula uma aplicação empresarial em que o microsserviço responsável pelo gerenciamento de usuários precisa acessar dados provenientes de outro microsserviço voltado para departamentos. Cada serviço foi projetado de forma independente, com seu próprio banco de dados, seguindo os princípios da arquitetura de microsserviços.

A comunicação entre os serviços foi viabilizada por meio do serviço de descoberta Eureka, que atua como um registro central no qual os microsserviços se registram e a partir do qual podem descobrir uns aos outros. Dessa forma, o microsserviço de usuário é capaz de consultar o endereço do microsserviço de departamento de forma dinâmica, sem depender de valores fixos ou configurações estáticas.

Para garantir a comunicação entre os microsserviços, foi utilizado o componente RestTemplate, uma ferramenta da biblioteca Spring para efetuar chamadas HTTP entre aplicações. O desenvolvimento foi realizado no ambiente Spring Tools, integrado ao IntelliJ IDEA Ultimate.

O presente relatório técnico documenta o processo completo de construção dos microsserviços, desde a criação dos projetos Spring Boot até a comunicação entre os serviços por meio do Eureka e do RestTemplate. Também serão abordadas as dificuldades encontradas ao seguir o tutorial original, como a ausência de anotações essenciais e falhas no uso da biblioteca Lombok, além de comparações com práticas recomendadas da literatura e opiniões de desenvolvedores especializados.

2. DESENVOLVIMENTO

2.1. Preparação do Ambiente de Desenvolvimento

Iniciei o desenvolvimento do trabalho me contextualizando sobre a proposta, lendo o texto/projeto base intitulado "Spring Boot Microservices Communication using RestTemplate" (JavaGuides, 2022), disponibilizado pelo professor Alysson Costa e Silva na plataforma Classroom, referente à disciplina de Engenharia de Software II, após a leitura, dei início à instalação dos softwares necessários para a execução do projeto.

Primeiramente, instalei o IntelliJ IDEA, aproveitando a assinatura premium gratuita disponibilizada pela JetBrains por meio do e-mail institucional da Unimontes. Dentre os recursos oferecidos, destaca-se o suporte nativo ao framework Spring Boot, uma vez que ele abstrai grande parte da configuração necessária e acelera o desenvolvimento de aplicações corporativas em Java (WALLS, 2016).

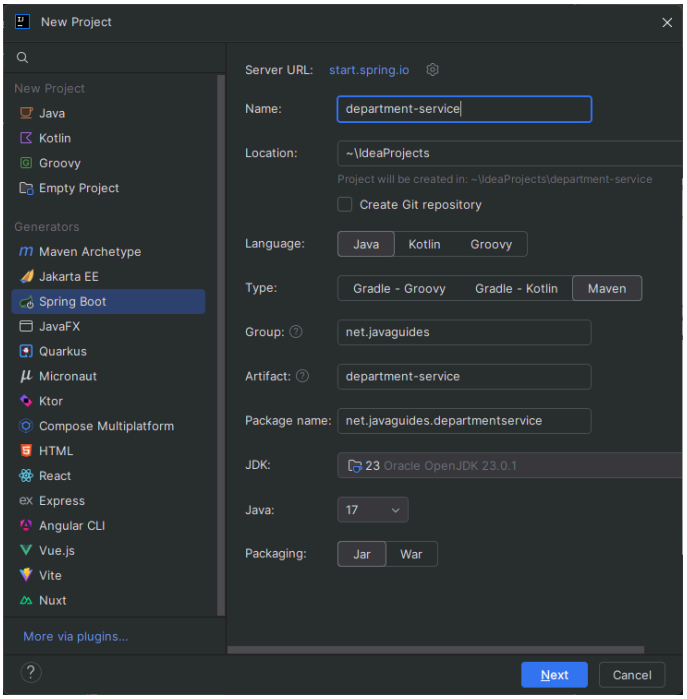
Em seguida, instalei o HeidiSQL, ferramenta responsável pela administração e gerenciamento dos bancos de dados MySQL utilizados no projeto. Por fim, instalei o Postman, que será utilizado para testar os endpoints dos microsserviços desenvolvidos, permitindo requisições HTTP independentes de qualquer interface gráfica frontend.

2.2. Estrutura e Implementação do Microsserviço department-service

2.2.1. Configuração do Projeto

Com o ambiente de desenvolvimento devidamente preparado, iniciei a criação do primeiro microsserviço: department-service. A criação do projeto foi realizada no IntelliJ IDEA, utilizando o assistente do Spring Boot com as dependências necessárias, como apresentado na Figura 1(a). As únicas modificações aplicadas foram no campo 'Name', que impacta diretamente os campos 'Artifact' e 'Package name'.

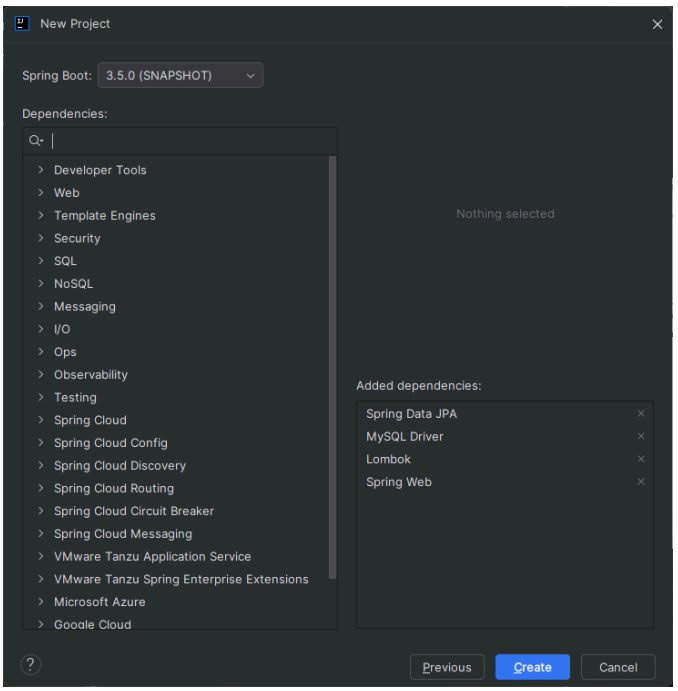
Figura 1(a) – Tela 1 de criação de projeto com o gerador Spring Boot



Fonte: elaboração própria.

Após finalizar esta primeira tela de configurações do projeto, ocorreu o redirecionamento para tela seguinte, em que é representado na Figura 1(b), nesta segunda tela, já se mantém o padrão de configurações para os dois microsserviços.

Figura 1(b) – Tela 2 de criação de projeto com o gerador Spring Boot



Fonte: elaboração própria.

Em seguida, configurei o arquivo application.properties, localizado em src/main/resources, adicionando os parâmetros de conexão com o banco de dados. A Listagem 1 apresenta essa configuração:

Listagem 1 – Configuração de conexão do department-service:

```
spring.datasource.url=jdbc:mysql://localhost:3306/department_db
spring.datasource.username=root
spring.datasource.password=1234
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update
```

Fonte: Compilação do Autor

2.2.2. Arquitetura de Pacotes e Camadas

Criei os pacotes seguindo a arquitetura MVC, padrão amplamente adotado em projetos com microsserviços por promover a separação de responsabilidades e facilitar a manutenção do sistema (RICHARDSON, 2021). O primeiro pacote estruturado foi o entity, com a classe Department, responsável por mapear os atributos do departamento e anotada com @Entity e @Table. Sua função é permitir a persistência e o mapeamento de dados no banco relacional.

Na sequência, criei o pacote repository, contendo a interface DepartmentRepository, que estende JpaRepository, provendo acesso direto aos métodos de CRUD. Essa interface é anotada com @Repository, o que permite a injeção de dependência no serviço.

O pacote service centraliza a lógica de negócios. A interface DepartmentService define os métodos de salvamento e busca de departamentos, enquanto a classe DepartmentServiceImpl os implementa, utilizando a anotação @Service e injetando o repositório por meio de @Autowired.

Por fim, o pacote controller expõe os endpoints REST através da classe DepartmentController, que recebe requisições HTTP, como POST para criação de novos departamentos e GET para consulta por ID, retornando respostas apropriadas com os dados manipulados via ResponseEntity.

Listagem 2 – Classe Department:

```
package net.javaguides.departmentservice.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "departments")
@NoArgsConstructor
@AllArgsConstructor

public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String departmentName;
    private String departmentAddress;
    private String departmentCode;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getDepartmentName() {
        return departmentName;
    }

    public void setDepartmentName(String departmentName) {
        this.departmentName = departmentName;
    }

    public String getDepartmentAddress() {
        return departmentAddress;
    }

    public void setDepartmentAddress(String departmentAddress) {
        this.departmentAddress = departmentAddress;
    }

    public String getDepartmentCode() {
        return departmentCode;
    }

    public void setDepartmentCode(String departmentCode) {
        this.departmentCode = departmentCode;
    }
}
```

Fonte: Compilação do Autor

Listagem 3 – Interface DepartmentRepository:

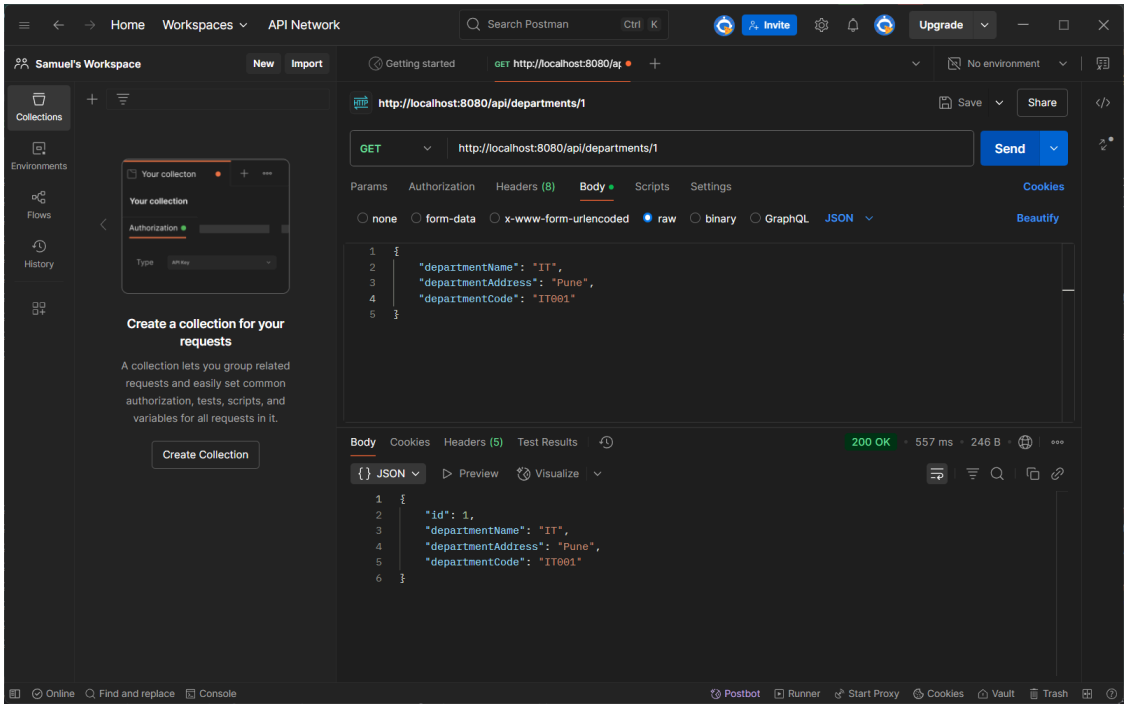
```
package net.javaguides.departmentservice.repository;

import net.javaguides.departmentservice.entity.Department;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface DepartmentRepository extends JpaRepository<Department, Long> {
}
```

Fonte: Compilação do Autor

2.2.3. Testes com Postman

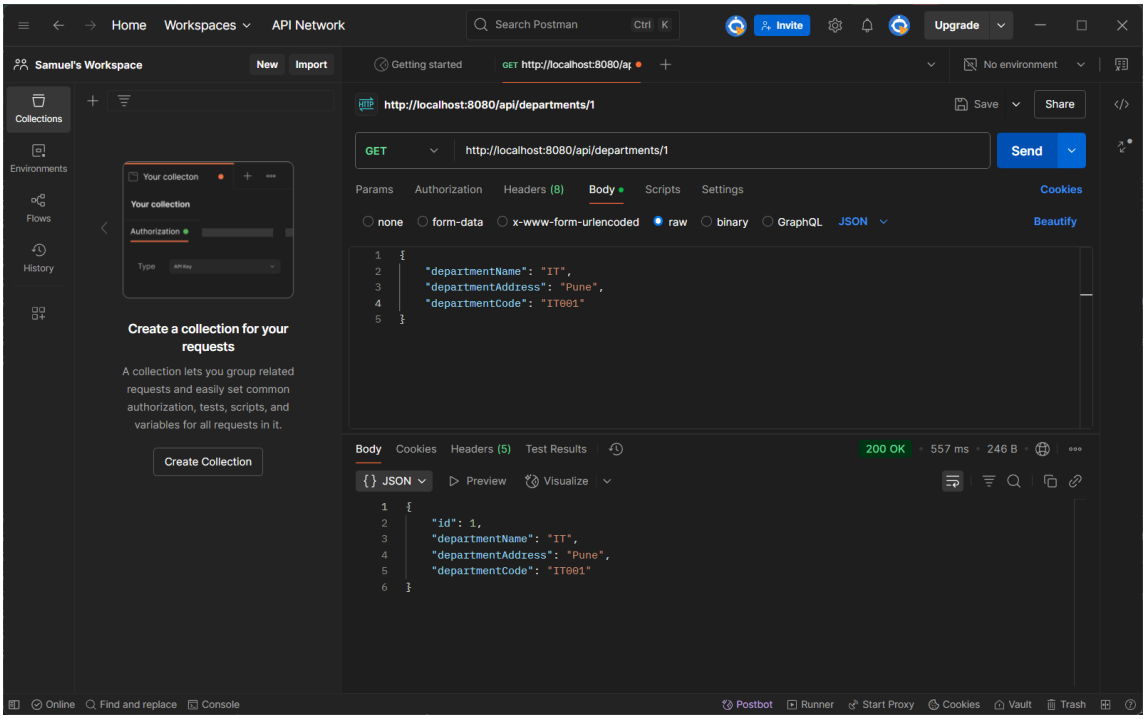
Realizei testes utilizando o Postman para verificar os endpoints criados. A requisição POST criou um novo departamento no banco de dados, conforme demonstrado na Figura 2(a), e a requisição GET buscou corretamente os dados persistidos, como mostrado na Figura 2(b).

Figura 2(a) – Tela de POST do banco de dados department



Fonte: elaboração própria.

Figura 2(b) – Tela de GET do banco de dados department



Fonte: elaboração própria.

2.3. Estrutura e Implementação do Microsserviço user-service

2.3.1. Configuração do Projeto

A configuração do projeto user-service seguiu os mesmos padrões do department-service, com a única diferença na porta (8081). As propriedades de conexão estão definidas no arquivo application.properties, garantindo a persistência em uma base de dados separada chamada employee_db.

2.3.2. Arquitetura de Pacotes e Camadas

A estrutura do projeto user-service também seguiu o modelo MVC (Model-View-Controller), que ajuda a organizar o código dividindo as responsabilidades em partes separadas. Isso facilita a manutenção e o entendimento do sistema.

Começando pelo pacote entity, nele foi criada a classe User, que é responsável por representar os dados de um usuário da empresa, como o nome, sobrenome, e-mail e o ID do departamento ao qual esse usuário pertence. Ela funciona como um "molde" para armazenar e manipular essas informações no banco de dados. Essa classe é anotada com @Entity, o que indica ao Spring que ela representa uma tabela no banco de dados.

Em seguida, temos o pacote repository, que contém a interface UserRepository, que é uma

interface em que herda os métodos prontos da biblioteca JpaRepository, como salvar, buscar por ID, excluir, etc. Com isso, não precisamos criar manualmente comandos SQL para essas ações básicas, pois o Spring cuida disso automaticamente. Esse repositório se conecta diretamente com o banco e permite que os dados dos usuários sejam persistidos (armazenados).

A parte da lógica de negócio, ou seja, as regras que controlam o funcionamento do sistema, está no pacote service. Nele, criamos duas classes: a interface UserService (que define o que o sistema deve fazer) e a classe UserServiceImpl (que de fato implementa essas funcionalidades). É nessa parte que, por exemplo, decidimos como um novo usuário deve ser salvo e como buscar seus dados corretamente.

Além disso, foi necessário criar um pacote chamado dto, sigla para Data Transfer Object (Objeto de Transferência de Dados). Esse pacote contém três classes que ajudam a organizar e controlar as informações que são trocadas entre os dois microsserviços (user-service e department-service) e também com os clientes que consomem a API:

- UserDto: carrega apenas os dados relevantes de um usuário, como nome e e-mail, sem incluir detalhes técnicos ou sensíveis.
- DepartmentDto: guarda os dados que vêm do serviço de departamentos, como o nome e a descrição do departamento.
- ResponseDto: é uma classe especial que junta os dois anteriores, combinando as informações do usuário com as do departamento em um único objeto. Esse objeto é o que é devolvido ao cliente (como o Postman ou um frontend), facilitando o consumo dos dados.

A principal vantagem de usar essas classes DTO é que protegemos a estrutura interna da aplicação, evitando expor diretamente as entidades (como a classe User) para o mundo externo, deixando a aplicação mais segura, flexível e fácil de modificar no futuro.

2.3.3. Implementação das Classes

No pacote service, a classe UserServiceImpl implementa os métodos definidos na interface UserService e introduz a comunicação entre microsserviços por meio do RestTemplate. Com essa ferramenta, foi possível buscar os dados do departamento com base no ID informado pelo usuário. Para viabilizar essa integração, foi necessária a criação de uma classe de configuração anotada com @Configuration, responsável por fornecer uma instância

gerenciada de RestTemplate ao Spring. Essa configuração, ausente no tutorial original, é indispensável para que o mecanismo de injeção de dependência funcione corretamente e permita a comunicação entre o user-service e o department-service.

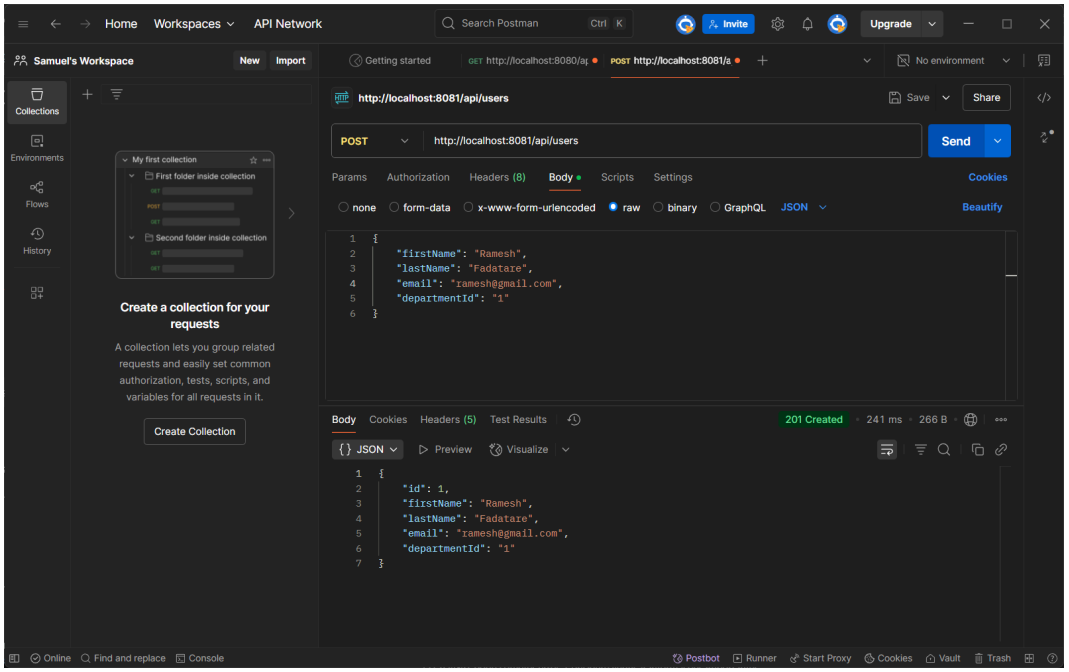
2.3.4. Comunicação entre Microsserviços

A comunicação entre os microsserviços user-service e department-service foi realizada por meio de requisições HTTP, utilizando o RestTemplate para consumo de dados externos. Essa abordagem permitiu que o user-service, ao buscar um usuário, enviasse uma requisição para o department-service com base no departmentId e integrasse os dados recebidos à resposta final. O objeto ResponseDto foi utilizado para encapsular de forma unificada as informações do usuário e do departamento, promovendo uma resposta completa ao cliente.

2.3.5. Testes com Postman

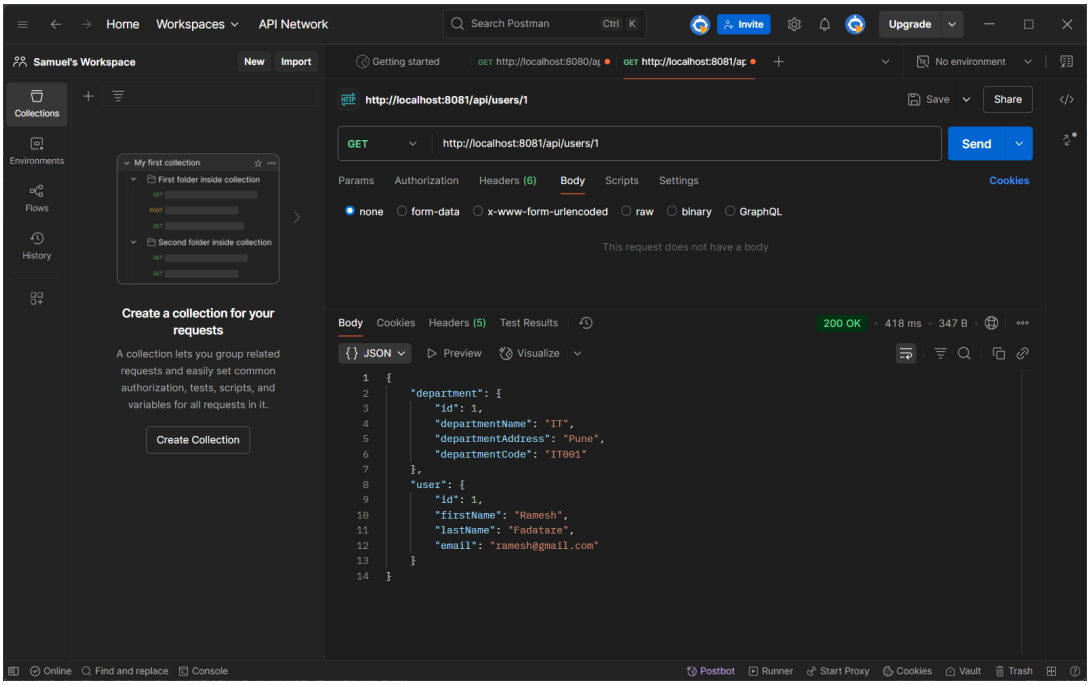
As requisições feitas via Postman demonstraram o funcionamento da integração entre os serviços. Os testes retornaram corretamente os dados agrupados conforme especificado no DTO ResponseDto, conforme demonstrado na Figura 3(a), onde observamos o Post e na Figura 3(b), em que é observado a função Get.

Figura 3(a) – Tela de POST do banco de dados employee



Fonte: elaboração própria.

Figura 3(b) – Tela de GET do banco de dados employee



Fonte: elaboração própria.

2.4. Considerações sobre a Arquitetura de Microsserviços

A arquitetura adotada segue boas práticas de design, com separação de responsabilidades, injeção de dependências e uso de padrões estabelecidos como MVC e DTOs. A comunicação via RestTemplate é compatível com soluções de mercado, ainda que bibliotecas modernas como WebClient e ferramentas como FeignClient ofereçam mais funcionalidades.

As principais correções aplicadas ao projeto original incluem:

- Adição de @Autowired e @Repository onde faltavam.
- Criação da configuração para RestTemplate.
- Implementação de DTOs para respostas agregadas.

Tais mudanças contribuíram para a legibilidade, manutenibilidade e robustez do sistema. Ainda que básico, o projeto serve como ponto de partida sólido para implementações mais avançadas, como o uso de gateways de API, balanceadores de carga e serviços de descoberta (EUREKA).

3. CONCLUSÃO

O desenvolvimento do projeto baseado na comunicação entre microsserviços com Spring Boot avançado permitiu compreender de forma prática os conceitos de arquitetura distribuída, integração via RestTemplate e organização em camadas utilizando o padrão MVC. A construção dos microsserviços department-service e user-service, aliada à utilização de ferramentas como IntelliJ IDEA, HeidiSQL e Postman, proporcionou uma visão clara sobre como diferentes sistemas podem se comunicar de forma eficiente, mesmo estando isolados em contextos distintos de execução.

Além de reforçar os conhecimentos adquiridos na disciplina, o trabalho evidenciou a importância de boas práticas de codificação, como o uso de DTOs para transferência de dados entre serviços, a separação clara de responsabilidades entre camadas e o uso de frameworks robustos que facilitam a criação de APIs RESTful.

4. REFERÊNCIAS

JAVAGUIDES. **Spring Boot Microservices Communication using RestTemplate.** 2022. Disponível em: https://www.javaguides.net/2022/09/spring-boot-microservices-communication-using-resttemplate.html. Acesso em: 01 abr. 2025.

RICHARDSON, Chris. **Microsserviços: princípios, práticas e padrões.** 1. ed. São Paulo: Novatec, 2021.

WALLS, Craig. **Spring Boot em ação.** 1. ed. São Paulo: Novatec, 2016.