UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO - UFES
CENTRO UNIVERSITÁRIO NORTE DO ESPÍRITO SANTO - CEUNES

Documentação do Micro Serviço de Posts

Grupo: Addison Alves Viana
Breno Souza do Nascimento
João Paulo Souza Ferrete
Lázaro José Pedrosa dos Reis
Ramon Pezzin Ton

1. Introdução

A documentação a seguir descreve a infraestrutura necessária, as ferramentas de software e outros componentes essenciais para colocar a aplicação em modo de produção. Esta documentação servirá como guia para configurar e implantar a aplicação de forma eficiente e confiável.

1.1 Descrição

O micro serviço é responsável por gerenciar os posts. Pode ser utilizado para criar, editar e apagar posts, de acordo com o descrito mais à frente no texto.

2. Infraestrutura

2.1 Docker

A infraestrutura da aplicação é baseada no uso de containers Docker para garantir a portabilidade e isolamento dos componentes. Para executar a aplicação em modo de produção, são necessários os seguintes contêineres:

- Contêiner da aplicação: Este contêiner contém todos os arquivos e dependências necessários para executar a aplicação. Ele é construído a partir de uma imagem Docker específica e pode ser escalado horizontalmente conforme necessário.
- Contêiner do banco de dados: Este contêiner hospeda o banco de dados MySQL utilizado pela aplicação. Ele é configurado com as tabelas e esquemas necessários e pode ser ajustado de acordo com as necessidades de armazenamento e desempenho.
- Docker Compose: O Docker Compose é usado para orquestrar e gerenciar os contêineres da aplicação e do banco de dados. Ele permite definir a configuração e as dependências dos contêineres em um arquivo YAML para facilitar a implantação e o gerenciamento.

2.2 Endpoints

Criar Post:
Método POST
/posts
Body:

```
{
  "name": "example",
  "title": "example",
  "content": "text",
  "author": 0,
  "medias": [1, 2, 3]
}
```

Acessar todos os Posts:
Método GET
/posts/{id}

Acessar Post:
Método GET
/posts/{id}

Editar Post:
Método PUT
/posts/{id}/update
Body: Apenas dados que deseja alterar

```
{
  "name": "example",
  "title": "example",
  "content": "text",
  "author": 0,
  "medias": [1, 2, 3]
}
```

Apagar Post:
Método DELETE

/posts/{id}

Acessar Post por Autor:
Método GET
/posts/author/{id}

Acessar Id de Mídias por Post:
Método GET
/posts/{id}/media

3. Ferramentas de Software

3.1 DBeaver

DBeaver é a ferramenta de software escolhida para gerenciar o banco de dados MySQL. Ele fornece uma interface gráfica intuitiva e abrangente para executar consultas, visualizar e modificar dados, gerenciar esquemas e tabelas, além de oferecer recursos de monitoramento e otimização do banco de dados.

3.2 Padrão de Codificação KtLint

KtLint é utilizado como padrão de formatação de código para garantir uma base de código consistente e legível. Ele é integrado ao processo de desenvolvimento, verificando automaticamente o estilo de codificação e fornecendo feedback instantâneo para manter o código alinhado com as diretrizes estabelecidas.

3.3 Kotlin com Framework Spring Boot

A aplicação é desenvolvida em Kotlin, uma linguagem moderna e concisa que se integra perfeitamente ao ecossistema Java. O Framework Spring Boot é utilizado para simplificar o desenvolvimento e fornecer um conjunto abrangente de recursos para a criação de aplicativos web robustos.

3.4 IDE

IntelliJ IDEA é a IDE (Integrated Development Environment) escolhida para desenvolver a aplicação. Ela oferece suporte completo ao Kotlin e ao ecossistema Spring,

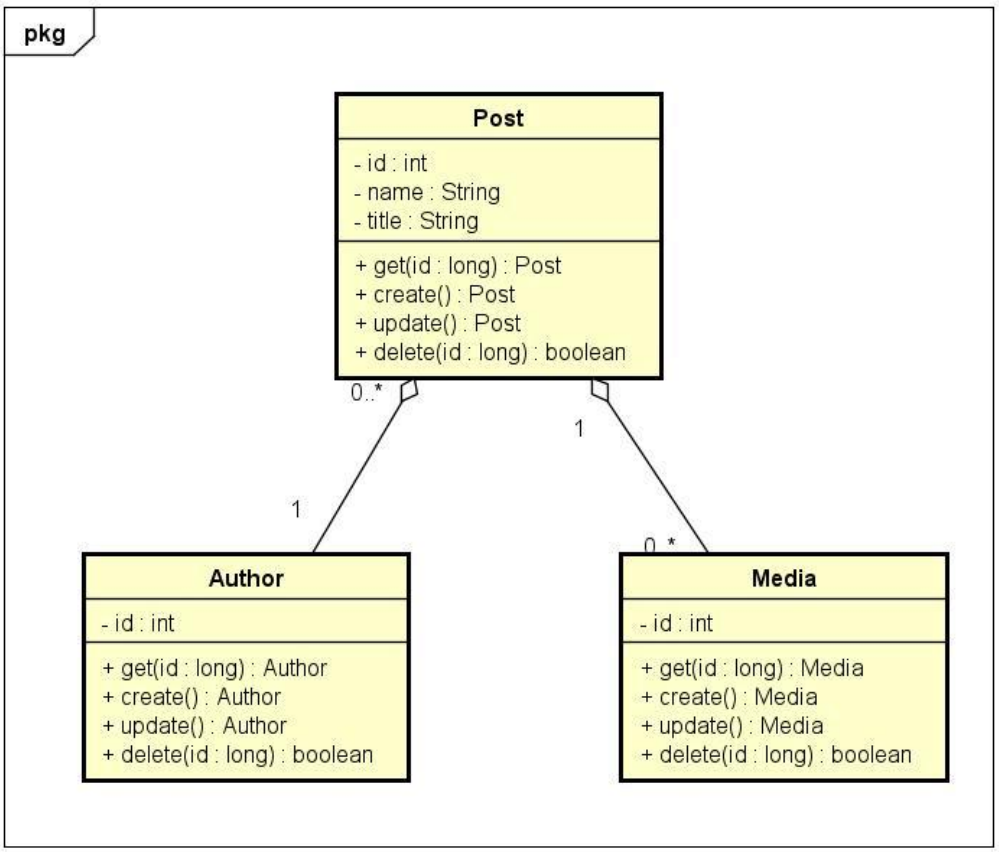
além de fornecer ferramentas avançadas de depuração, refatoração e gerenciamento de dependências.

4. Diagramas

Esta seção apresenta os diagramas que foram criados com base no código da aplicação. Os diagramas fornecem uma representação visual das estruturas e interações do sistema, facilitando a compreensão do fluxo de dados e das relações entre os componentes.

4.1 Diagrama de Classes

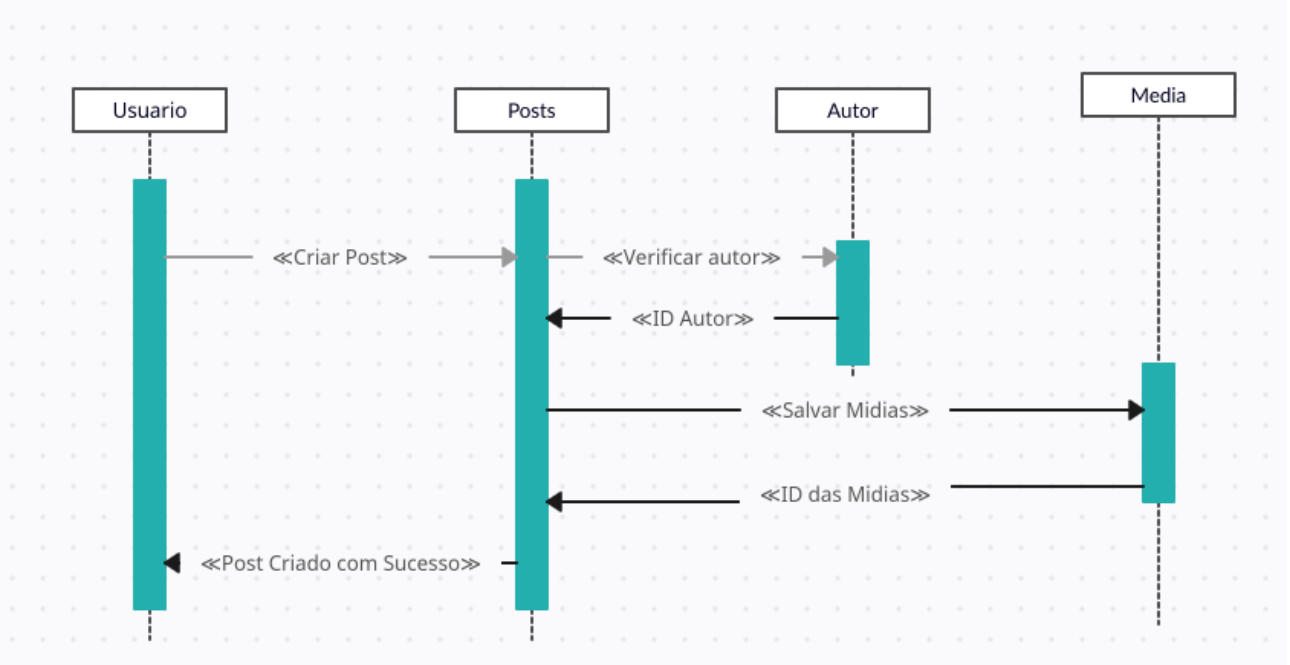
O diagrama de classes ilustra a estrutura das classes presentes na aplicação, incluindo seus atributos, métodos e relacionamentos. Ele fornece uma visão geral das entidades e suas interações, ajudando a compreender a organização do código e as responsabilidades de cada classe



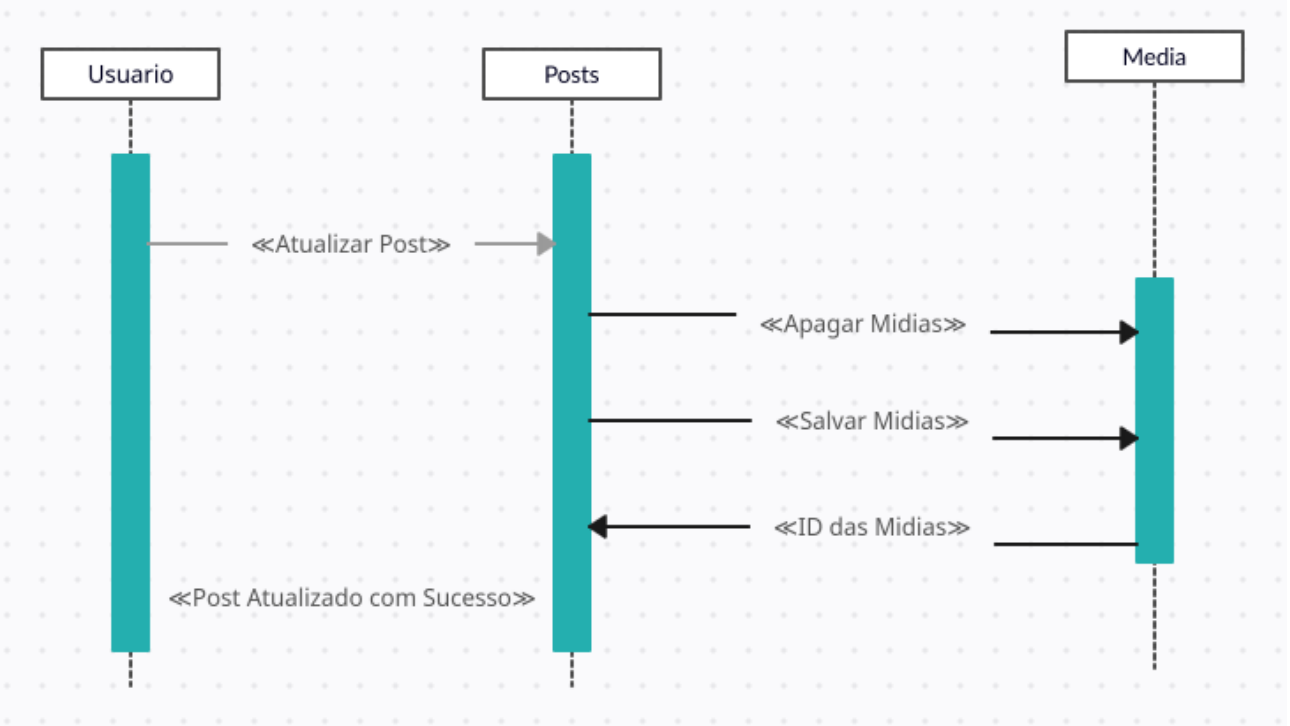
4.2 Diagrama de Sequência

O diagrama de sequência mostra a interação entre os objetos da aplicação ao longo do tempo, destacando a ordem das chamadas de métodos e as trocas de mensagens entre os componentes. Ele ajuda a visualizar o fluxo de execução e o comportamento dos objetos em diferentes cenários, auxiliando na identificação de possíveis problemas e na análise do fluxo de dados.

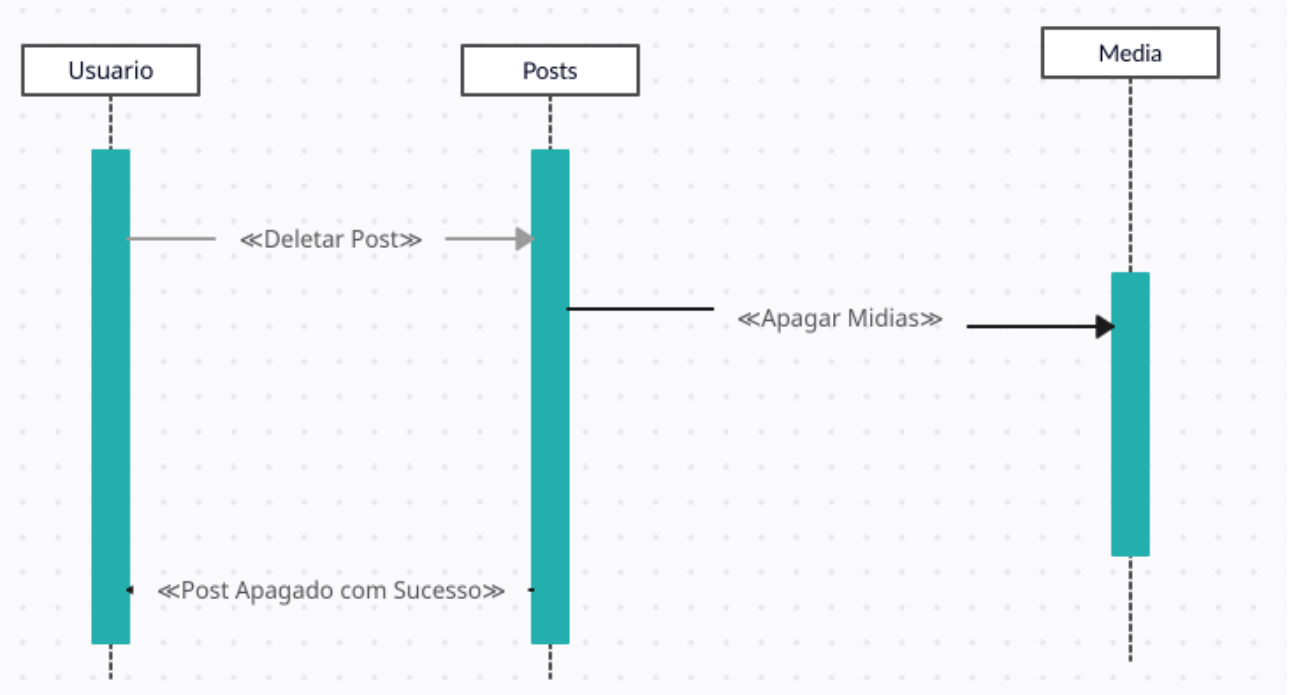
4.2.1 Diagrama de sequência de criação de post



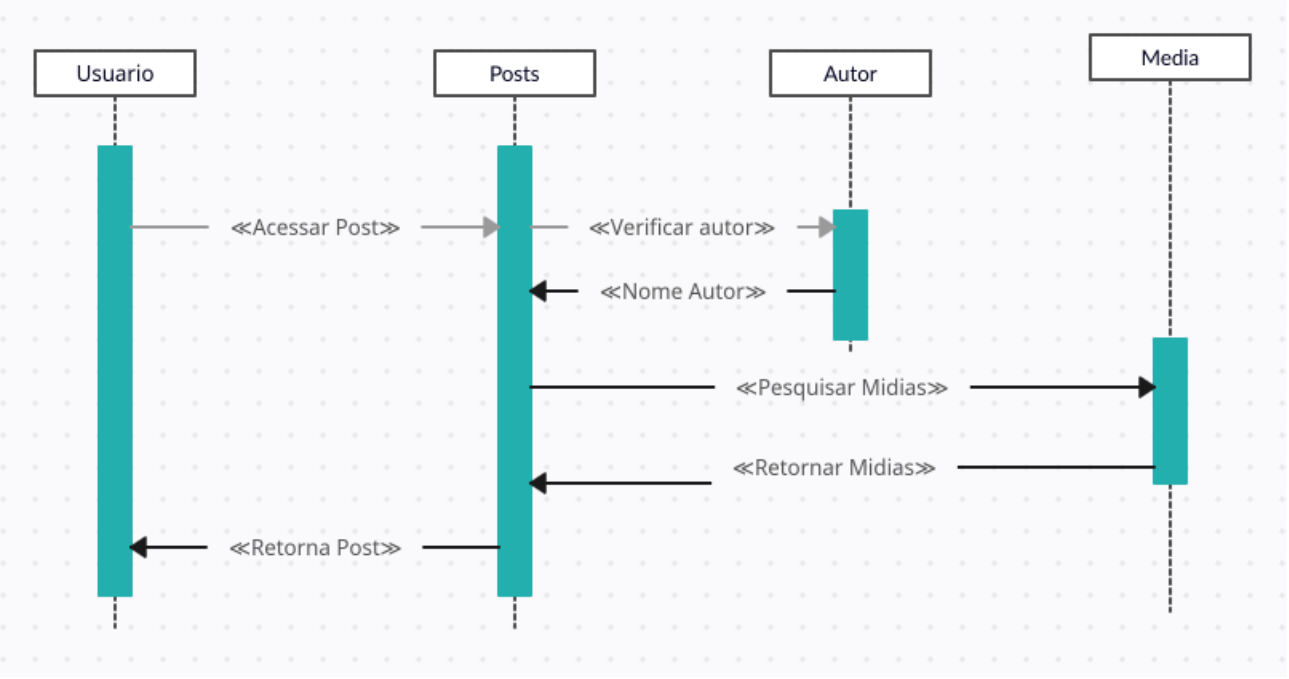
4.2.2 Diagrama de sequência de atualizar post



4.2.3 Diagrama de sequência de deletar post



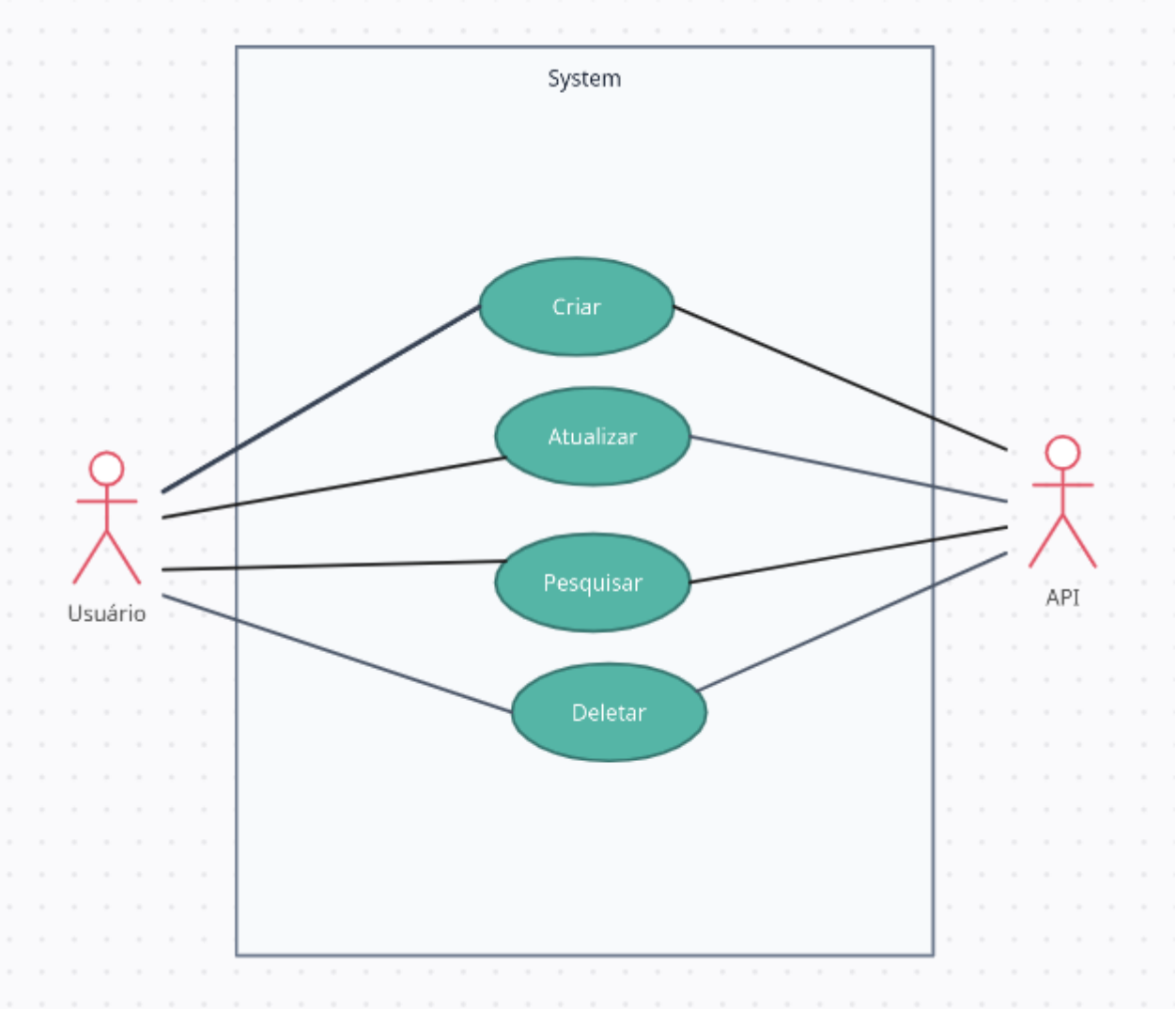
4.2.4 Diagrama de sequência de Acessar post



4.3 Diagrama de Caso de Uso

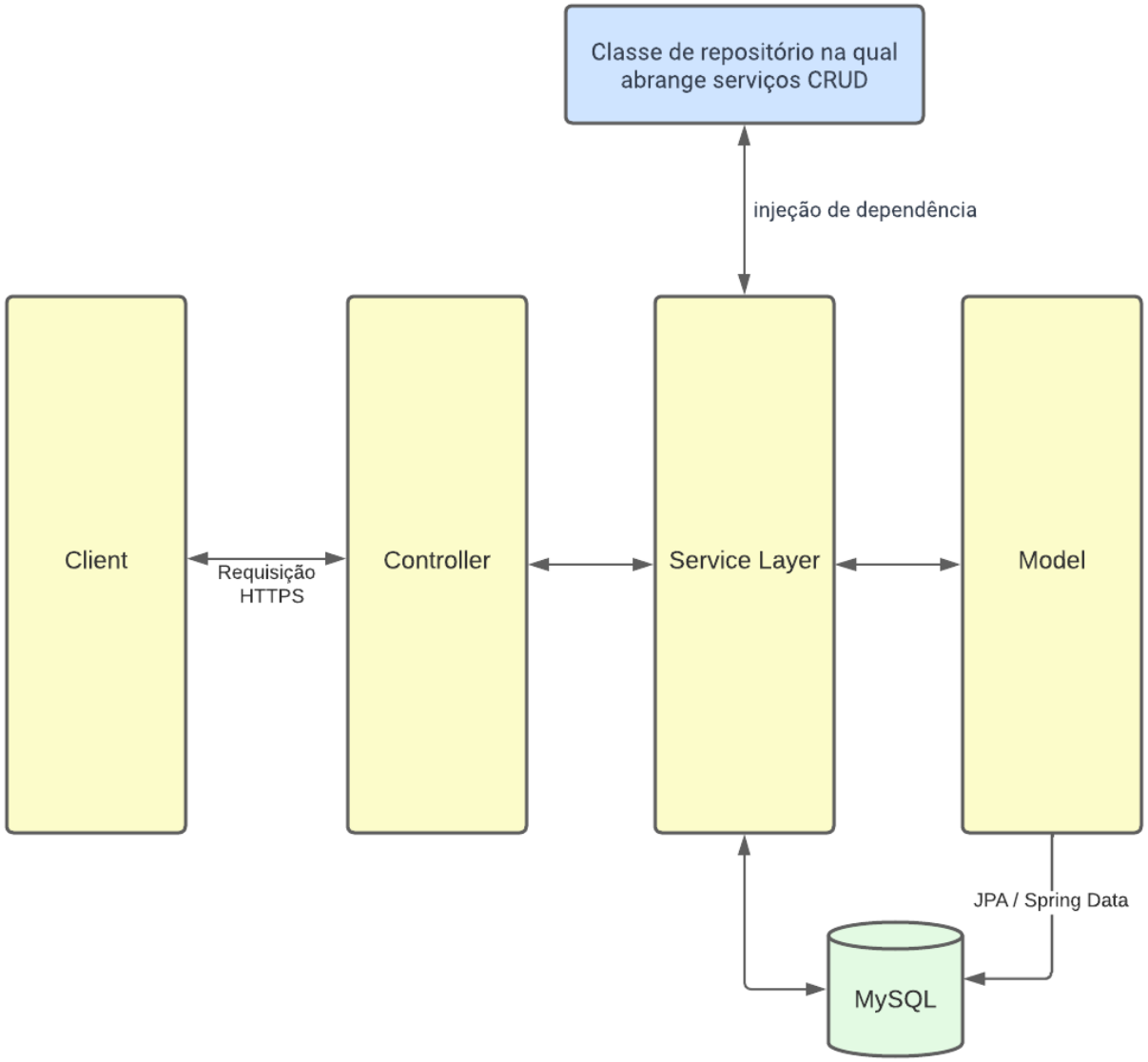
O diagrama de caso de uso é uma representação visual que descreve as interações entre atores externos e o sistema, mostrando os principais cenários de uso da

aplicação. Ele ajuda a identificar os diferentes papéis envolvidos na interação com o sistema e os principais casos de uso que a aplicação oferece.



4.4 Diagrama da Arquitetura

O diagrama da arquitetura é uma representação da estrutura e organização geral da aplicação. Ele mostra os principais componentes do sistema, suas interações e como eles se relacionam entre si. Esse diagrama fornece uma visão de alto nível da arquitetura da aplicação, permitindo entender sua divisão em camadas, módulos ou serviços.



O Spring Boot é um framework popular para o desenvolvimento de aplicativos Java e Kotlin. Ele segue uma arquitetura em camadas para organizar e separar as responsabilidades do código. Podemos separar a aplicação nas seguintes camadas:

1. Camada de Apresentação (Presentation Layer): Essa camada é responsável pela interação com o usuário ou outros sistemas externos. Ela inclui controladores (Controllers) que recebem solicitações HTTP, gerenciam a lógica de negócios e retornam respostas adequadas. Os controladores podem receber e validar parâmetros de entrada, chamar serviços e fornecer respostas formatadas.

2. Camada de Serviço (Service Layer): Essa camada contém a lógica de negócios do aplicativo. Ela coordena a execução das operações, manipula as regras de negócio e realiza chamadas a outras camadas, como a camada de acesso a dados. Os serviços encapsulam a funcionalidade do domínio e são responsáveis por orquestrar as operações de negócios.

3. Camada de Acesso a Dados (Data Access Layer): Essa camada é responsável pela persistência e recuperação dos dados. Ela interage com bancos de dados ou outros meios de armazenamento, como APIs de terceiros. Os componentes dessa camada, como repositórios (Repositories), fornecem métodos para criar, ler, atualizar e excluir dados.

4. Camada de Domínio (Domain Layer): Essa camada representa o núcleo do domínio da aplicação. Ela contém as entidades de negócio, que são classes que representam conceitos e objetos relevantes para o domínio da aplicação. As entidades encapsulam o estado e o comportamento relacionados ao domínio, como validações e operações específicas.

5. Dependências

A aplicação possui as seguintes dependências para execução:

- Java SDK >= 17: É necessário ter o Java Development Kit (JDK) instalado em sua máquina, com a versão igual ou superior a 17. O JDK fornece as bibliotecas e ferramentas necessárias para compilar e executar a aplicação Java.
- Java RE >= 18: É recomendado ter o Java Runtime Environment (JRE) instalado em sua máquina, com a versão igual ou superior a 18. O JRE permite executar a aplicação Java sem a necessidade de compilar o código-fonte.
- Java Server VM >= 18: É recomendado ter o Java Server Virtual Machine (JSVM) instalado em sua máquina, com a versão igual ou superior a 18. O JSVM é uma implementação da máquina virtual Java otimizada para executar aplicativos em servidores.

- MySQL DB: É necessário ter um banco de dados MySQL configurado e acessível para a aplicação. Certifique-se de ter as credenciais corretas e as tabelas/esquemas necessários criados no banco de dados.

6. Executando localmente

Para executar a aplicação localmente, siga as instruções abaixo:

- No diretório "post-microservice", execute o seguinte comando: "make run". Isso iniciará a aplicação utilizando o Gradle Wrapper.

7. Como compilar

Para compilar a aplicação, siga as etapas abaixo:

- Clone o repositório para o seu ambiente de desenvolvimento.
- No diretório "post-microservice", execute o seguinte comando: "make build". Isso iniciará o Gradle Wrapper e ele será responsável por baixar todas as dependências necessárias e construir o projeto.
    - Observação: Na primeira execução, pode levar algum tempo para baixar todas as dependências.

8. Executando os testes

Para executar os testes da aplicação, siga as etapas abaixo:

- No diretório "post-microservice", execute o seguinte comando: "./gradlew test". Isso iniciará o Gradle Wrapper e ele será responsável por executar todos os testes unitários da aplicação.

9. Testes

Para testes, utilizamos a própria biblioteca de *Coverage* do Spring Boot. Neste caso, para cada função implementada foi implementado um teste unitário para garantir o funcionamento da mesma.