



# Bootcamp: Arquiteto de Software

Aluno: Renan Ferreira Lima

## Relatório de Entrega do Desafio Final

### Introdução

Este relatório desenvolve o solicitado na atividade denominada desafio final do curso de Bootcamp de Arquitetura de Software realizado na Faculdade XP no primeiro semestre de 2025. A proposta é a aplicação dos conceitos de arquitetura de software abordados durante o curso, tendo para isso que projetar, documentar e implementar uma API REST.

A API terá funcionalidades de CRUD (Create, Read, Update, Delete) e permitirá consultas adicionais sobre o domínio de Clientes, o qual foi escolhido entre as opções propostas, para isso será utilizado o padrão arquitetural MVC (Model-View-Controller). A solução visa disponibilizar dados de forma pública para parceiros da empresa conforme o enunciado disponibilizado.

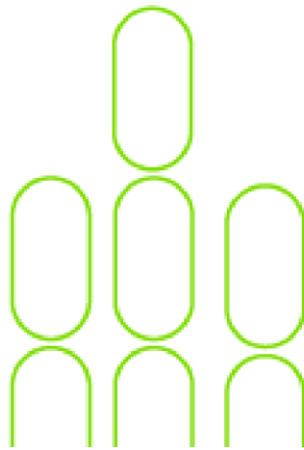
### Desenvolvimento da Solução do Desafio Final

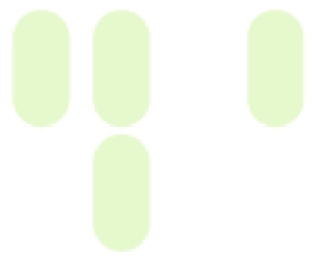
#### 1. Escolha da Plataforma e Linguagem

API RESTful com Java (Spring Boot).

#### 2. Funcionalidades da API

CRUD completo, i.e., Criação (Create), Leitura (Read), Atualização (Update) e Exclusão (Delete). Além dos métodos a seguir para o domínio de Clientes:

- **Contagem:** Endpoint para retornar o número total de registros.
  - **Find All:** Endpoint para retornar todos os registros.
- 



- **Find By ID:** Endpoint para retornar um registro específico com base no ID.
- **Find By Name:** Endpoint para retornar registros que correspondam a um nome específico.

### 3. Arquitetura

Padrão Arquitetural: MVC (Model-View-Controller) para estruturar a aplicação.

Camada de dados: Banco NoSQL de Documentos MongoDB

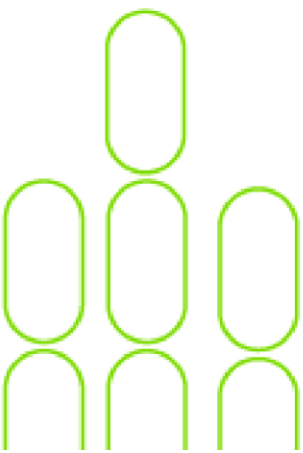
Infraestrutura: Contêineres Docker

Orquestração: Docker Compose

Diagrama de Arquitetura: Padrão C4 do Nível 1 ao 3 utilizando a plataforma Draw.io

Repositório Github: <https://github.com/lima-renan/api-rest-desafio>

### 4. Estrutura de pastas



```
└─ src/main/java

    └─ com/desafio/api # Pacote base da aplicação

        └─ controller # Camada de Apresentação (Endpoints REST)

            └─ CustomerController.java # Gerencia endpoints HTTP para
entidades de Customer

        └─ model # Camada de Domínio (Documentos MongoDB)

            └─ Customer.java # Entidade/Documento representando um Cliente

        └─ repository # Camada de Acesso a Dados (MongoDB)

            └─ CustomerRepository.java Interface para interagir com a
collection Customer

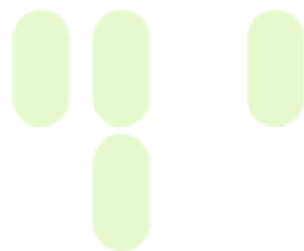
        └─ service # Camada de Negócios

            └─ CustomerService.java Contém a lógica de negócios para
Customer

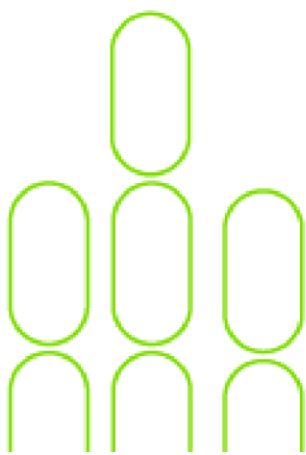
        └─ ApiApplication # Classe principal que inicia a aplicação Spring
Boot
```

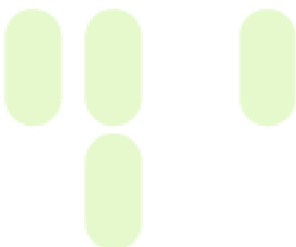
### Explicação dos Componentes e Pastas:

- **com.desafio.api:** Este é o pacote raiz da aplicação. Todas as outras classes e subpacotes residirão dentro dele.
- **ApiApplication.java:** Localizada diretamente no pacote base, esta é a classe principal que inicializa a aplicação Spring Boot. Ela contém o método main e geralmente a anotação @SpringBootApplication.
- **controller (Camada de Apresentação):**
  - **Componente:** CustomerController.java



- **Papel:** Responsável por lidar com as requisições HTTP que chegam à API. Atua como a interface entre o cliente (quem consome a API) e a aplicação.
  - **Componentes:** Contém as classes anotadas com `@RestController` (ex: `ClienteController`).
  - **Responsabilidades:** Mapear os endpoints (URLs como `/clientes`, `/clientes/{id}`), receber dados das requisições (ex: `@RequestBody`, `@PathVariable`), chamar os métodos apropriados na camada de Service para processar a lógica de negócio, e formatar e retornar as respostas HTTP (geralmente em JSON), incluindo códigos de status apropriados (200 OK, 201 Created, 404 Not Found, entre outros.). Não contém lógica de negócio complexa.
- **model (Camada de Domínio):**
    - **Componente:** `Customer.java`
    - **Papel:** Representa os dados fundamentais da aplicação e as regras de negócio intrínsecas a esses dados. No contexto do MongoDB, representa os documentos que serão armazenados nas coleções.
    - **Componentes:** Contém as classes de entidade (POJOs - Plain Old Java Objects) que representam os objetos do domínio (ex: `Cliente.java`). Com MongoDB, essas classes vão ser anotadas com `@Document` para mapeá-las às coleções do banco.
    - **Responsabilidades:** Definir a estrutura dos dados (atributos como id, nome, email), seus tipos e quaisquer validações ou regras inerentes à própria entidade.
  - **repository (Camada de Acesso a Dados):**
    - **Componente:** `CustomerRepository.java`
    - **Papel:** Abstrai a interação com a fonte de dados e fornece uma interface clara para realizar operações de persistência.
    - **Componentes:** Contém as interfaces que estendem interfaces do Spring Data MongoDB, como `MongoRepository` (ex: `ClienteRepository`).

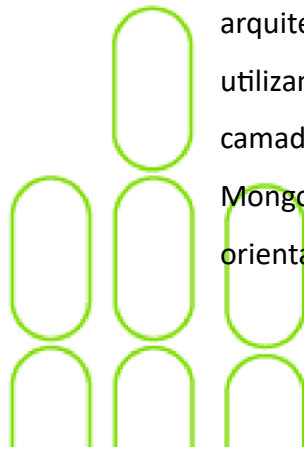


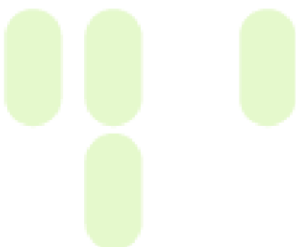
- 
- **Responsabilidades:** Definir métodos para operações CRUD (Create, Read, Update, Delete) e consultas personalizadas sobre as entidades (model). O Spring Data MongoDB implementa automaticamente muitos desses métodos com base na assinatura da interface. Isola o resto da aplicação dos detalhes específicos de como os dados são armazenados e recuperados.
  - **service (Camada de Negócio):**
    - **Componente:** CustomerService.java
    - **Papel:** Orquestra a lógica de negócio da aplicação. Atua como intermediário entre a camada de Controller e a camada de Repository.
    - **Componentes:** Contém as classes anotadas com @Service (ex: ClienteService).
    - **Responsabilidades:** Implementar os casos de uso da aplicação. Contém a lógica de negócio principal, validações que envolvem múltiplas entidades ou regras complexas, coordena transações (se aplicável), e utiliza os Repository para buscar e persistir dados. Mantém a camada de Controller enxuta, focada apenas em HTTP.

## Conclusão

A realização deste Desafio Final foi essencial para consolidar e aplicar os diversos conceitos e técnicas de arquitetura de software abordados ao longo do bootcamp. O processo envolveu desde a análise dos requisitos e modelagem arquitetural até a implementação e documentação de uma API REST funcional.

**Aplicação dos Conhecimentos:** Para resolver o desafio, foram empregados conhecimentos fundamentais do bootcamp, como o padrão arquitetural MVC para estruturação do código, e a modelagem utilizando o C4 Model para visualização da arquitetura em diferentes níveis de abstração. A implementação foi realizada em Java utilizando o framework Spring Boot, aproveitando recursos como Spring Web para a camada de Controller, Spring Data MongoDB para a camada de Repository. A escolha do MongoDB como banco de dados NoSQL permitiu explorar a flexibilidade de um banco orientado a documentos.






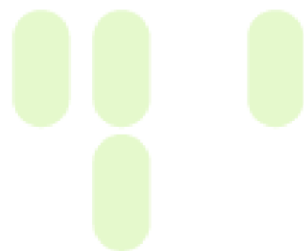
**Principais Dificuldades e Superações:** Uma das principais dificuldades foi em sair abordagem teórica para a implementação prática, especialmente na configuração correta das dependências do Spring Boot e no mapeamento inicial das responsabilidades de cada camada MVC. A refatoração para integrar o MongoDB exigiu um entendimento adicional das particularidades do Spring Data MongoDB, como o uso de `MongoRepository` e as anotações `@Document` e `@Id` (com tipo `String`). Esses obstáculos foram superados através da consulta à documentação oficial do Spring, exemplos da comunidade e apoio de ferramentas de IA Generativa como Copilot, ChatGPT e Gemini.

**Resultados Obtidos:** A solução final consiste em uma API REST plenamente funcional, capaz de realizar todas as operações solicitadas para a entidade "Cliente": CRUD (Create, Read, Update, Delete), contagem total de registros, busca de todos os registros, busca por ID específico e busca por nome. A API segue a estrutura MVC, com código organizado em pacotes distintos por responsabilidade (controller, service, repository, model) e utiliza o MongoDB para persistência de dados. A arquitetura foi documentada utilizando o C4 Model (Contexto, Contêiner, Componente) e este relatório.

**Lições Aprendidas:** Este desafio reforçou a importância da modelagem arquitetural prévia para guiar o desenvolvimento e facilitar a comunicação sobre o sistema. A aplicação prática do padrão MVC demonstrou seu valor na organização do código, promovendo a separação de responsabilidades e facilitando a manutenção. Além disso, a experiência com Spring Boot e Spring Data MongoDB evidenciou a produtividade e eficiência dos frameworks modernos na aceleração do desenvolvimento de APIs robustas.

**Melhorias Futuras:** Como próximos passos, a solução poderia ser aprimorada com a implementação de um tratamento de exceções mais granular e centralizado, a adição de validações mais robustas nos dados de entrada (DTOs - Data Transfer Objects), a implementação de paginação nas listagens para lidar com grandes volumes de dados, e a incorporação de mecanismos de segurança, como autenticação e autorização (ex: JWT ou OAuth2), para proteger os endpoints da API. A adição de testes unitários e de integração também aumentaria a confiabilidade da aplicação. Outro ponto também





seria a possibilidade de criar esteiras de CI/CD, utilizando por exemplo o Azure DevOps, tornando o desenvolvimento mais padronizado, integrável e eficiente.

Esta experiência prática foi fundamental para solidificar o aprendizado e desenvolver uma visão mais crítica e aplicada sobre as decisões arquiteturais em projetos de software.

