CHECKPOINT 3

ARQUITETURA ORIENTADA A SERVIÇOS – 3ESPW

Integrantes:

- Julia Azevedo Lins – RM 98690
- Luís Gustavo Barreto Garrido – RM 99210
- Victor Hugo Aranda Forte – RM 99750

Sumário

Introdução

Este projeto consiste em uma API RESTful desenvolvida em Java com Spring Boot, utilizando arquitetura em camadas para promover organização, manutenção e escalabilidade do código. O objetivo principal é permitir o cadastro, consulta, atualização e exclusão de pacientes, seguindo boas práticas de desenvolvimento orientado a objetos, persistência de dados e documentação de API via Swagger UI.

A aplicação está integrada ao banco de dados MySQL, containerizado com Docker, e segue os padrões estabelecidos para o checkpoint da FIAP.

Estrutura do Projeto

Controller

- **Função:** Responsável por receber e responder às requisições HTTP.
- **Classe principal:** PacienteController
- **Explicação:** Aqui estão mapeados os endpoints (como GET, POST, PUT e DELETE), que chamam os métodos da camada de serviço. Essa camada é a "porta de entrada" para o sistema.

Service

- **Função:** Implementa a lógica de negócio da aplicação.
- **Classe principal:** PacienteService
- **Explicação:** Contém a inteligência da aplicação, como validações de dados, tratamento de erros e chamadas ao repositório. Garante que regras como “nome e email não podem ser nulos” sejam respeitadas.

DTO (Data Transfer Object)

- **Função:** Definem os formatos dos dados recebidos e enviados.
- **Classes utilizadas:**
  - PacienteRequestCreate: dados para criar paciente.
  - PacienteRequestUpdate: dados para atualização.
  - PacienteResponse: dados retornados ao usuário.
- **Explicação:** Usamos os DTOs para evitar expor diretamente a estrutura do banco de dados e para facilitar a validação e a segurança.

Model

- **Função:** Representa a entidade que será persistida no banco de dados.
- **Classe principal:** Paciente
- **Explicação:** Contém os atributos mapeados com anotações JPA como @Entity, @Id, @GeneratedValue, etc. Esta classe reflete a tabela no banco.

Repository

- **Função:** Acesso ao banco de dados via JPA.
- **Interface:** PacienteRepository
- **Explicação:** Extende JpaRepository, fornecendo métodos prontos como save, findById, deleteById e findAll.

Testes via Swagger

- Criar um paciente (POST /pacientes)
- Listar todos (GET /pacientes)
- Buscar por ID (GET /pacientes/{id})
- Atualizar (PUT /pacientes/{id})
- Excluir (DELETE /pacientes/{id})

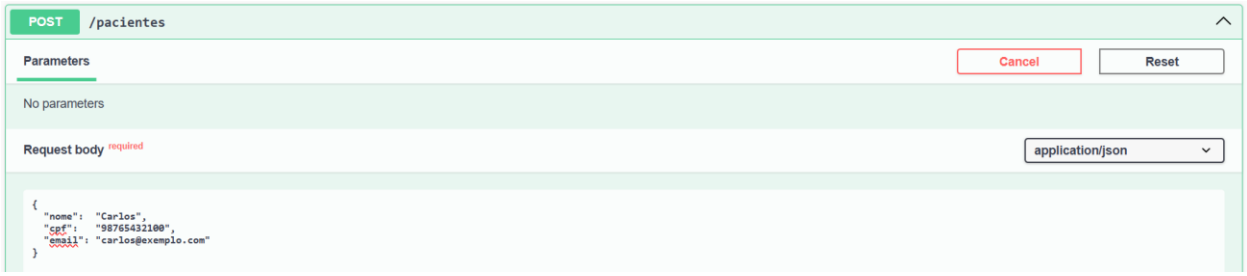
Endpoints Explicados

Criar um Paciente (POST /pacientes)

Este endpoint permite o cadastro de um novo paciente. É necessário informar nome, CPF e email. Todos os campos são obrigatórios.

Exemplo de uso no Swagger:

- Preencha os dados no corpo da requisição e clique em Execute.
- Você receberá de volta os dados do paciente criado com seu respectivo ID.

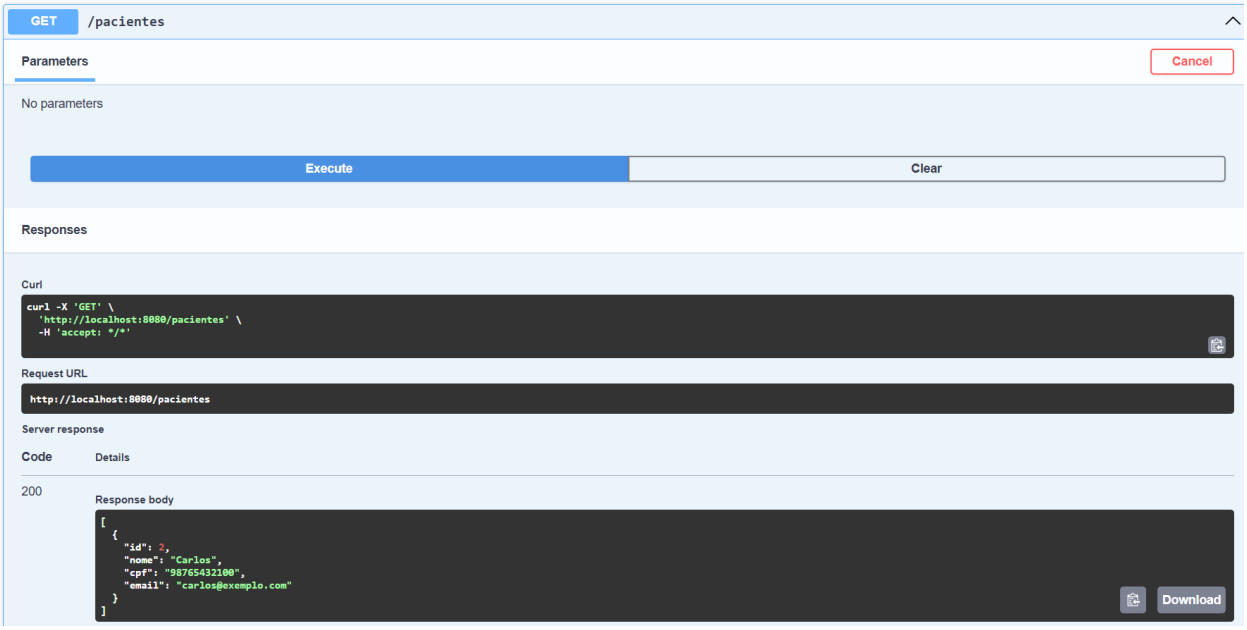


Listar todos os Pacientes (GET /pacientes)

Retorna uma lista com todos os pacientes cadastrados no sistema. Pode ser usado para verificar o conteúdo atual da base de dados.

Exemplo de uso:

- Clique em Execute e confira os pacientes listados.

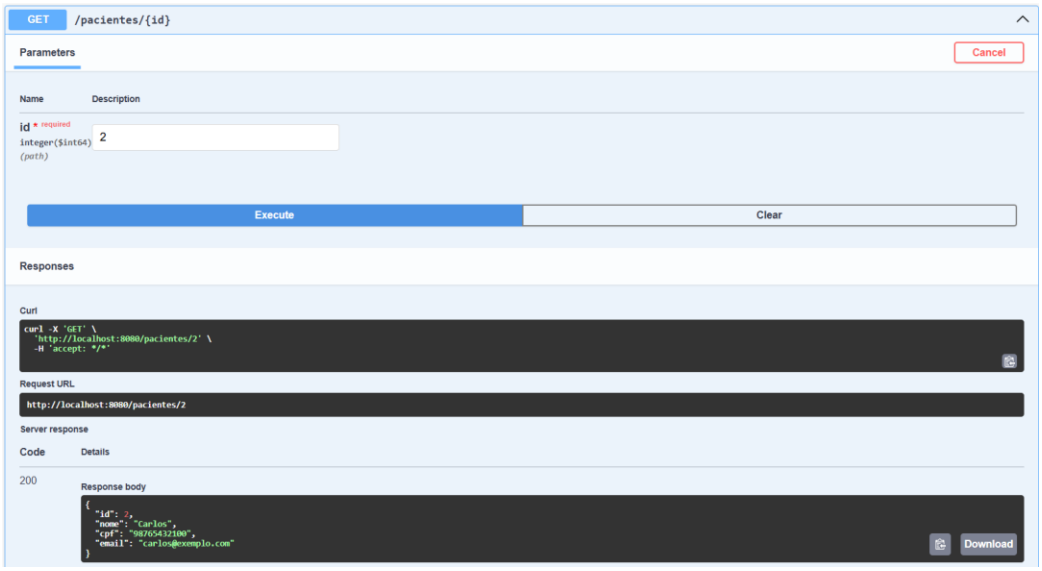


Buscar Paciente por ID (GET /pacientes/{id})

Busca um paciente específico pelo seu identificador único (ID). Se o ID não existir, uma mensagem de erro será exibida.

Exemplo de validação:

- Realizamos um teste buscando o ID 2 com sucesso.

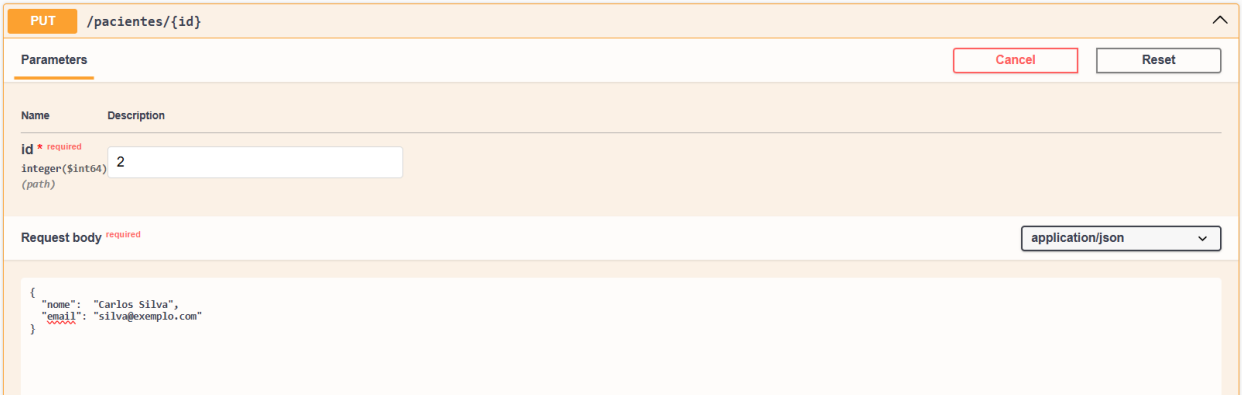


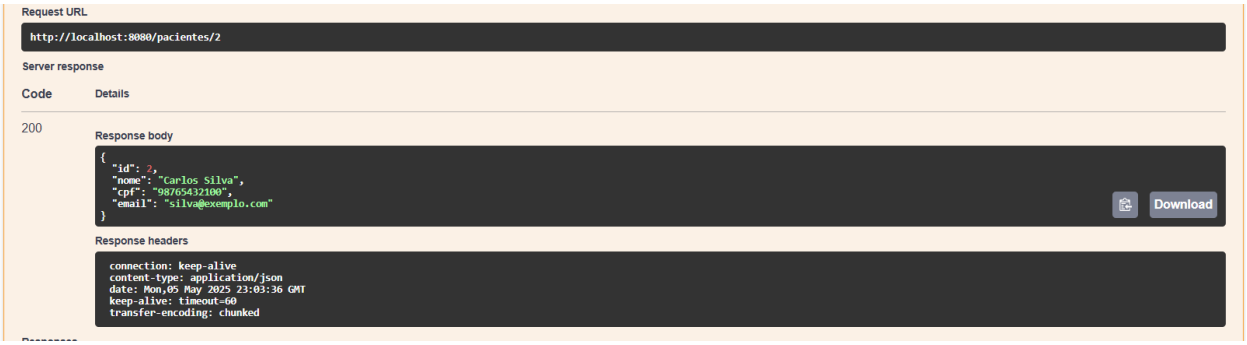
Atualizar Paciente (PUT /pacientes/{id})

Atualiza os dados de um paciente existente. É necessário informar o ID na URL e os novos valores de nome e email no corpo da requisição.

Atenção:

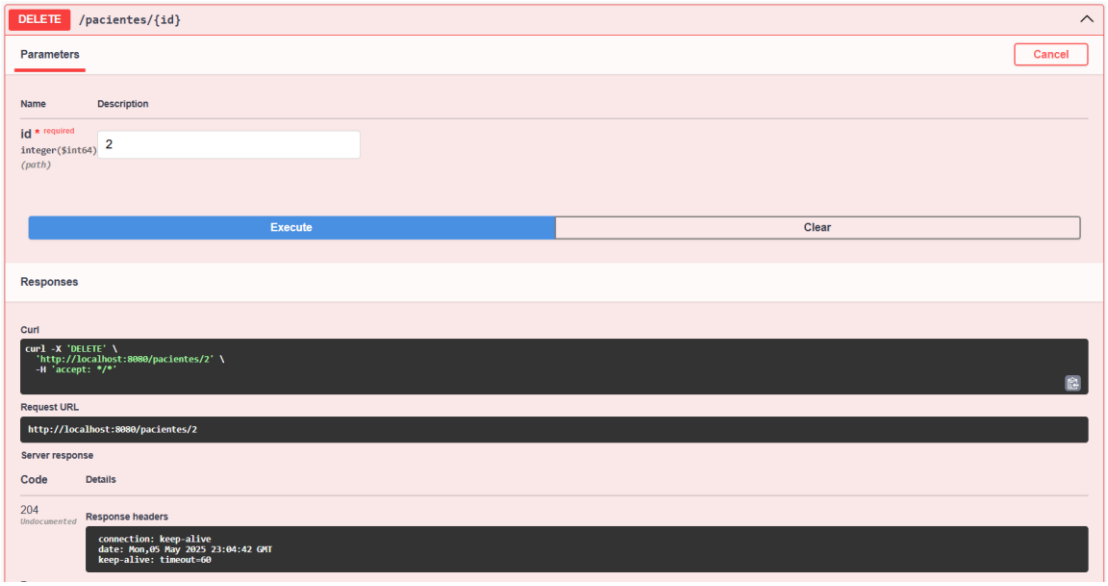
- O CPF não pode ser alterado após a criação.





Deletar Paciente (DELETE /pacientes/{id})

Remove um paciente do sistema com base no ID informado. Se o ID for válido, o paciente será removido permanentemente.



Para validar o delete, executamos um get no ID 2: