

Sprint 3 – Domain Driven Design using Java **IMREA – Sistema de Teleconsultas**

Aluno: Leonardo José Pereira
RM: 563065

Aluno: Ícaro Santos
RM: 563070

Aluno: Fabrício Santos
RM: 563072

Objetivo e Escopo do Projeto

O sistema **IMREA - Sistema de Teleconsultas** foi desenvolvido com o objetivo de modernizar o atendimento remoto do Instituto de Medicina Física e Reabilitação (IMREA).

O objetivo desta 4^a Sprint foi de atualizar um sistema de console para uma **API Restful** robusta e completa, utilizando o framework **Spring Boot**. Esta API serve como o back-end que proverá todos os dados e lógica de negócio necessários para uma aplicação front-end web ou mobile.

O escopo do sistema compreende o gerenciamento completo de cadastros de pacientes, profissionais (médicos), especialidades e, principalmente, o agendamento e cancelamento de teleconsultas, garantindo a integridade dos dados, verificação de conflitos de horário e tratamento de exceções

Descrição das Principais Funcionalidades

A API foi estruturada em uma arquitetura de 5 camadas (Model, Repository, Service, Controller, Exception) e implementa as seguintes funcionalidades de negócio:

- **Agendamento de Consultas (`POST /api/agendamentos`):**
 - Validação de antecedência mínima de 30 minutos.
 - Prevenção de agendamentos duplicados para o mesmo profissional no mesmo horário (checagem de conflito).
- **Cancelamento de Consultas (`PATCH /api/agendamentos/{id}/cancelar`):**
 - Alteração do status do agendamento para "CANCELADO".
- **Reagendamento de Consultas (`PUT /api/agendamentos/{id}/reagendar`):**
 - Alteração da data/hora de um agendamento, com as mesmas validações de antecedência e conflito.
- **Listagem de Recursos:**
 - Exposição de endpoints `GET` para listar todos os Pacientes e todos os Profissionais, permitindo ao front-end popular menus de seleção.
- **Tratamento de Exceções (`@ControllerAdvice`):**
 - Implementação de um handler global que captura erros (ex: "Profissional não encontrado", "Horário ocupado") e retorna um JSON padronizado com o status HTTP correto (ex: 404, 400).

Tabela de Endpoints (API Restful)

A tabela a seguir descreve os principais endpoints implementados para o funcionamento do front-end.

A tabela resume a **API de Agendamentos e Cadastros** com os seguintes endpoints:

Verbo HTTP	URI	Descrição	Sucesso	Erro Cliente
POST	/api/agendamentos	Cria um agendamento.	201	400 (Conf...)
GET	/api/agendamentos/{id}	Busca agendamento por ID.	200	404
PATCH	/api/agendamentos/{id}/cancelar	Cancela agendamento.	200	404
PUT	/api/agendamentos/{id}/reagendar	Reagenda data de agendamento.	200	400, 404
GET	/api/pacientes	Lista pacientes.	200	-
GET	/api/pacientes/{id}	Busca paciente por ID.	200	404
GET	/api/profissionais	Lista profissionais.	200	-
GET	/api/profissionais/{id}	Busca profissional por ID.	200	404

Protótipo – Prints das Telas Implementadas

A seguir, são apresentados os testes funcionais da API, demonstrando o fluxo completo de requisição e resposta para cada endpoint principal.

Teste 1: Cadastro Paciente

JSON com os dados do "Paciente Teste Um"

O servidor recebeu esses dados, cadastrou o paciente com sucesso no banco de dados e respondeu com um status 200 OK.

Confirmação: O mais importante é que a resposta do servidor incluiu o "id": 4. Isso confirma que o paciente foi criado e o banco de dados atribuiu a ele o ID de número 4.

The screenshot shows the Postman interface with a successful POST request to `http://localhost:8080/api/pacientes`. The request body is a JSON object:

```
1 - {  
2   "nome": "Paciente Teste Um",  
3   "dataNascimento": "1998-05-15",  
4   "cpf": "09867890962",  
5   "email": "Papa.Romeu@email.com"  
6 }
```

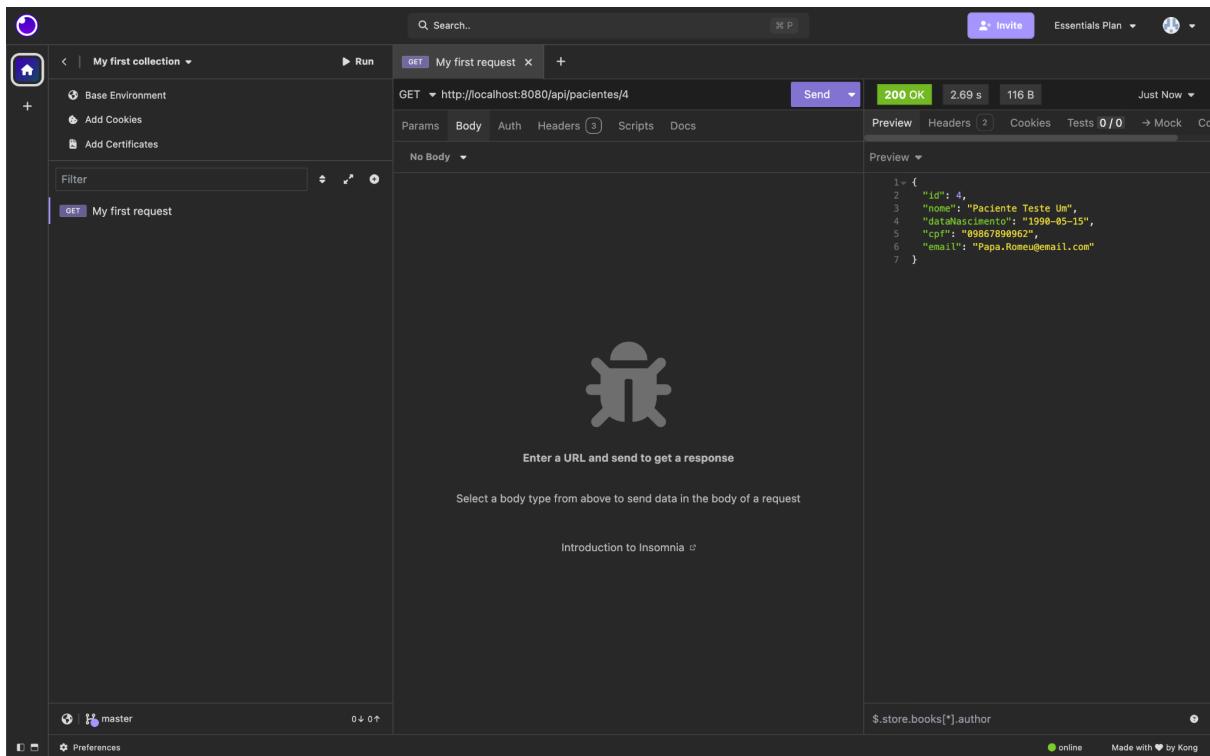
The response status is 200 OK, and the response body shows the patient has been created with id: 4:

```
1 - {  
2   "id": 4,  
3   "nome": "Paciente Teste Um",  
4   "dataNascimento": "1998-05-15",  
5   "cpf": "09867890962",  
6   "email": "Papa.Romeu@email.com"  
7 }
```

Teste 2: Consulta Paciente por ID

Enviamos uma requisição GET para a URL do paciente, usando o ID 4 que foi criado no teste anterior (.../api/pacientes/4). O objetivo é buscar os dados específicos desse paciente.

Confirmação: O servidor respondeu com status 200 OK e retornou o JSON completo do "Paciente Teste Um", provando que ele está corretamente armazenado no banco.



The screenshot shows the Insomnia API client interface. On the left, there's a sidebar with a collection named "My first collection" containing a single request titled "GET My first request". The main panel displays the request details: method "GET", URL "http://localhost:8080/api/pacientes/4", and a response status of "200 OK" with a response time of "2.69 s" and a size of "116 B". Below the status, there are tabs for "Preview", "Headers", "Cookies", and "Tests". The "Preview" tab shows the JSON response:

```
1- {  
2-   "id": 4,  
3-   "name": "Paciente Teste Um",  
4-   "dataNascimento": "1998-05-15",  
5-   "cpf": "09867898962",  
6-   "email": "Papa.Romeu@email.com"  
7- }
```

Teste 3: Listar Todos os Pacientes

Enviamos uma requisição GET para a URL base (.../api/pacientes) para solicitar uma lista de todos os pacientes cadastrados no sistema.

Confirmação: O servidor respondeu com status 200 OK e retornou uma lista (array []) contendo todos os pacientes. É possível ver o "Paciente Teste Um" (ID 4) no meio da lista.

The screenshot shows the Insomnia REST client interface. On the left, there's a sidebar with a collection named "My first collection" containing one request titled "My first request". The main panel shows a request configuration for a GET method to the URL `http://localhost:8080/api/pacientes`. The response status is 200 OK, with a response time of 163 ms and a body size of 239 B. The response body is displayed as a JSON array:

```
1 ↵ [
2 ↵   {
3 ↵     "id": 3,
4 ↵     "nome": "Paciente Teste Um",
5 ↵     "dataNascimento": "1990-05-15",
6 ↵     "cpf": "12345678900",
7 ↵     "email": "paciente.teste@email.com"
8 ↵   },
9 ↵   {
10 ↵     "id": 4,
11 ↵     "nome": "Paciente Teste Um",
12 ↵     "dataNascimento": "1990-05-15",
13 ↵     "cpf": "09876543210",
14 ↵     "email": "Papa.Romeu@email.com"
15 ↵   }
16 ↵ ]
```

Below the response, there's a note: "Enter a URL and send to get a response" and a link "Introduction to Insomnia". At the bottom, it shows the current branch is "master" and there are 0 downvotes and 0 upvotes. The footer indicates the app is "Made with ❤ by Kong".

Teste 4: Atualizar Paciente

Enviamos uma requisição PUT para a URL do paciente (.../api/pacientes/4). No Body da requisição, enviamos um JSON com os campos que queríamos alterar (por exemplo, mudando o email).

Confirmação: O servidor processou a atualização e respondeu com 200 OK, retornando o objeto do paciente já com os dados atualizados (ex: com o novo email), confirmando que a alteração foi salva.

The screenshot shows the Postman interface with a successful API call. The request details are as follows:

- Method: PUT
- URL: <http://localhost:8080/api/pacientes/4>
- Status: 200 OK
- Time: 205 ms
- Size: 125 B
- Timestamp: Just Now

The request body is a JSON object:

```
1~ {  
2~   "id": 4,  
3~   "nome": "Paciente Teste Um",  
4~   "dataNascimento": "1990-05-15",  
5~   "cpf": "88867898962",  
6~   "email": "novo.email.alterado@email.com"  
7~ }
```

The response body is also a JSON object, identical to the request body, indicating the update was successful:

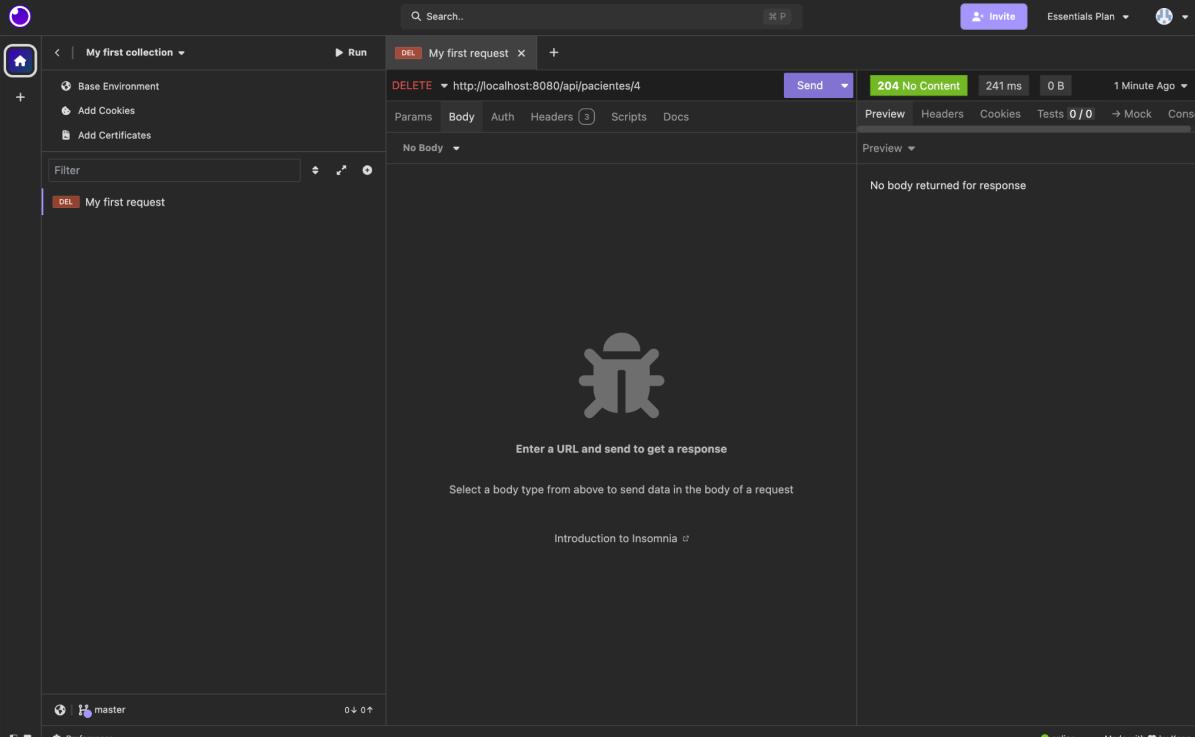
```
1~ {  
2~   "id": 4,  
3~   "nome": "Paciente Teste Um",  
4~   "dataNascimento": "1990-05-15",  
5~   "cpf": "88867898962",  
6~   "email": "novo.email.alterado@email.com"  
7~ }
```

At the bottom, the status bar shows: \$store.books[*].author online Made with ❤ by Kong

Teste 5: Deletar Paciente

Enviamos uma requisição DELETE para a URL do paciente (.../api/pacientes/4) para removê-lo do banco de dados.

Confirmação: O servidor processou a deleção com sucesso e respondeu com o status 204 No Content. Este status (sem conteúdo) é a resposta padrão de sucesso para uma operação de DELETE.



The screenshot shows the Insomnia REST client interface. On the left, there's a sidebar with a collection named "My first collection" containing three items: "Base Environment", "Add Cookies", and "Add Certificates". Below this is a search bar and a "Run" button. The main area displays a request titled "My first request" with the method set to "DELETE" and the URL "http://localhost:8080/api/pacientes/4". The response status is "204 No Content" with a duration of "241 ms" and a size of "0 B". The timestamp is "1 Minute Ago". Below the status, there are tabs for "Preview", "Headers", "Cookies", "Tests", and "Mock". The "Preview" tab shows the message "No body returned for response". At the bottom of the main window, there's a placeholder text "Enter a URL and send to get a response" and a note "Select a body type from above to send data in the body of a request". The footer includes a "master" branch indicator, a "Preferences" link, and a "Made with ❤ by Kong" logo.

Teste 6: Validar Deleção

Para provar que o paciente foi mesmo deletado, enviamos novamente uma requisição GET para a URL do paciente que acabamos de apagar (.../api/pacientes/4).

Confirmação: O servidor respondeu com o status 404 Not Found (Não Encontrado). Isso confirma que o paciente com ID 4 não existe mais, validando o sucesso do teste de deleção.

The screenshot shows the Insomnia REST client interface. On the left, there's a sidebar with 'My first collection' containing a single request named 'My first request'. The main panel displays a request configuration for a GET method to the URL `http://localhost:8080/api/pacientes/4`. The response status is shown as **404 Not Found** with a response time of 135 ms and a body size of 55 B. The response preview shows the following JSON:

```
1 - {
2   "status": 404,
3   "mensagem": "Paciente não encontrado: 4"
4 }
```

The interface includes tabs for Params, Body, Auth, Headers, Scripts, and Docs. The Body tab is currently selected, showing the message 'Enter a URL and send to get a response'. Below the preview, there's a note: 'Select a body type from above to send data in the body of a request'. At the bottom, it says 'Introduction to Insomnia' and shows some footer information like '\$store.books[*].author' and 'Made with ❤ by Kong'.

Modelo de Entidade-Relacionamento (MER)

O modelo de banco de dados Oracle foi implementado com base no script As tabelas principais e seus relacionamentos são:

1. T_IMREA_PACIENTE

Armazena os dados cadastrais dos pacientes.

- **ID_PACIENTE (PK)**
(NOME_PACIENTE, CPF, DATA_NASCIMENTO, TIPO_DEFICIENCIA, CONTATO, ENDERECO, CONVENIO)

2. T_IMREA_ESPECIALIDADE

Tabela de domínio para as especialidades médicas.

- **ID_ESPECIALIDADE (PK)** (NOME_ESPECIALIDADE, DESCRICAO)

3. T_IMREA_PROFISSIONAL

Armazena os dados dos médicos e outros profissionais de saúde.

- **ID_PROFESSONAL (PK)**
(NOME_PROFESSONAL, **ID_ESPECIALIDADE (FK)**: Referencia T_IMREA_ESPECIALIDADE, REGISTRO_PROF, CONTATO)

4. T_IMREA_ATENDIMENTO

Registra um atendimento (consulta ou sessão) realizado.

- **ID_ATENDIMENTO (PK)**
- **ID_PACIENTE (FK)**: Referencia T_IMREA_PACIENTE
- **ID_PROFESSONAL (FK)**: Referencia T_IMREA_PROFESSONAL (DATA_ATENDIMENTO, TIPO_ATENDIMENTO, OBSERVACOES)

5. T_IMREA_TRATAMENTO

Descreve um plano de tratamento contínuo para um paciente.

- **ID_TRATAMENTO (PK)**
- **ID_PACIENTE (FK)**: Referencia [T_IMREA_PACIENTE](#)
- **ID_PROFISSIONAL (FK)**: Referencia [T_IMREA_PROFISSIONAL](#)
- **ID_ESPECIALIDADE (FK)**: Referencia [T_IMREA_ESPECIALIDADE](#) ,
DATA_INICIO, DATA_FIM, DESCRICAO

6. T_IMREA_PROCEDIMENTO

Tabela de domínio para os procedimentos oferecidos.

- **ID_PROCEDIMENTO (PK)**(NOME_PROCEDIMENTO, DESCRICAO, CUSTO)

7. T_IMREA_TRATAMENTO_PROCEDIMENTO

Tabela associativa (N:N) que vincula quais procedimentos fazem parte de um tratamento.

- **ID_TRATAMENTO (PK, FK)**: Referencia [T_IMREA_TRATAMENTO](#)
- **ID_PROCEDIMENTO (PK, FK)**: Referencia [T_IMREA_PROCEDIMENTO](#)

8. T_IMREA_AGENDAMENTO

Tabela principal da API, registra os agendamentos futuros.

- **ID_AGENDAMENTO (PK)**
- **ID_PACIENTE (FK)**: Referencia [T_IMREA_PACIENTE](#)
- **ID_PROFISSIONAL (FK)**: Referencia [T_IMREA_PROFISSIONAL](#)
- DATA_AGENDAMENTO, STATUS

9. T_IMREA_PRONTUARIO

Armazena as anotações e prescrições de um paciente, vinculadas a um atendimento.

- **ID_PRONTUARIO (PK)**
- **ID_PACIENTE (FK)**: Referencia [T_IMREA_PACIENTE](#)
- **ID_ATENDIMENTO (FK)**: Referencia [T_IMREA_ATENDIMENTO](#)
- ANOTACOES, PRESCRICAO)

Diagrama de Classes Atualizado

A arquitetura do projeto foi migrada para Spring Boot, seguindo os padrões de Domain Driven Design e separação de camadas. A arquitetura de classes segue o fluxo de responsabilidade:

1. **Controller** (`AgendamentoController`, `PacienteController`)
 - Recebe as requisições HTTP (`@RestController`).
 - **Depende de:** `Service`.
2. **Service** (`AgendamentoService`, `PacienteService`)
 - Contém a lógica de negócio (regras, validações) (`@Service`).
 - **Depende de:** `Repository`.
3. **Repository** (`AgendamentoRepository`, `PacienteRepository`)
 - Interface que estende `JpaRepository` para acesso aos dados (`@Repository`).
 - **Depende de:** `Model`.
4. **Model** (`Agendamento`, `Paciente`)
 - Classes que mapeiam as tabelas do banco (`@Entity`).
5. **Exception** (`GlobalExceptionHandler`)
 - Classe que captura exceções de todas as camadas (`@ControllerAdvice`).

Modelo de Entidade-Relacionamento (MER)

O modelo de banco de dados Oracle foi implementado com base no script `IMREA_CHALLENGE_ENTREGA4.sql`. As tabelas principais e seus relacionamentos são:

