CENTRO UNIVERSITARIO INTERNACIONAL – UNINTER

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JESIEL VIANA PFEIFER

SGHSS

SISTEMA DE GESTÃO DE HOSPITALAR E SERVIÇOS DE SAÚDE

Campo Bom

2025

Jesiel Viana Pfeifer

SGHSS – Sistema de Gestão Hospitalar e Serviços de Saúde

Trabalho de Conclusão de Curso apresentado ao curso de Análise e Desenvolvimento de Sistemas do Centro Universitário Internacional UNINTER.

Polo de Apoio Presencial: UNINTER | São Leopoldo - RS

Orientadora: Professora Luciane Yanase Hirabara Kanashiro

Campo Bom

2025

Sumário

1. Introdução

Com foco no backend, este projeto demonstra o funcionamento da estrutura central do Sistema de Gestão Hospitalar e Serviços de Saúde (SGHSS), responsável por processar requisições, gerenciar dados sensíveis e implementar regras de negócio essenciais para o ambiente hospitalar.

O backend garante a integração segura entre diferentes módulos do sistema, como agendamento de consultas, controle de usuários e registos médicos, além de assegurar escalabilidade, desempenho e proteção das informações por meio de autenticação robusta e segmentação de permissões

Além de abordar os aspectos técnicos do desenvolvimento da API, este trabalho busca evidenciar a importância da integração de sistemas hospitalares para otimizar processos, garantir a segurança das informações e aprimorar o atendimento aos pacientes.

A aplicação conta com segmentação de usuários, garantindo que cada usuário tenha suas devidas permissões para realizar as ações dentro do sistema, são eles:

- **Paciente**: permite agendar uma consulta, criar um cadastro, visualizar suas consultas anteriores
- **Profissional da Saúde**: pode gerenciar sua agenda de atendimentos, prescrever receitas, realizar atendimentos e ter acesso ao histórico do paciente que consultou
- **Administradores**: permite o controle dos usuários na aplicação bem como registro, alterações e exclusão de dados, relatório de atendimentos e leitos

O projeto foi criado usando as seguintes tecnologias:

- Prisma: Sistema ORM para comunicação e querys no banco de dados
- Zod: Biblioteca para validação de dados nas requisições e métodos.
- Docker: Abstrai o sistema operacional do usuário para que não haja erros de execução em diferentes máquinas
- Express: controle e gerenciamento das rotas da API.
- TypeScript: garante segurança na tipagem dos dados durante o desenvolvimento.

- NodeJS: Framework amplamente utilizada para aplicações de backend usando JavaScript.

Abaixo estão os links do projeto no GITHUB e a documentação das rotas no POSTMAN, assim como provas de execução dentro de cada rota.

Postman: https://documenter.getpostman.com/view/30201252/2sB3BGGpjD

GITHUB: https://github.com/JesielPfeifer/VidaPlus

2. Análise e Requisitos

O uso de autenticação é crucial para impedir que usuários maliciosos tenham acesso a dados sensíveis da unidade hospitalar e aos dados das pessoas envolvidas (pacientes e profissionais).

Para resolver este problema, cada rota da API possui uma lista de cargos autorizados para realizar cada ação, sem o token de autenticação válido, qualquer requisição é negada, o token também possui tempo de expiração de uma hora garantindo mais segurança em caso de esquecer uma sessão logada.

3. Requisitos Funcionais

A aplicação é dividida por usuários e rota, com isso, temos a separação de responsabilidades e permissões para cada, ou seja, os requisitos são diferentes para cada tipo de uso da aplicação, veja abaixo a listagem de requisitos funcionais para cada usuário

4. Profissional:

ID	Descrição	Tipo	Prioridade
RF001	Permitir o registro de novos profissionais	Funcional	Alta
RF002	Realizar a internação de um paciente	Funcional	Alta
RF003	Obter os dados do paciente	Funcional	Alta
RF004	Necessita realizar login para utilizar o sistema	Funcional	Alta

5. Usuário

ID	Descrição	Tipo	Prioridade
RF001	Permitir o registro de novos pacientes	Funcional	Alta
RF002	Permitir a criação de consultas para os pacientes	Funcional	Alta

RF003	Permitir ver a agenda de consultas do profissional	Funcional	Média
RF004	Necessita realizar login para utilizar o sistema	Funcional	Alta

6. Administrador

ID	Descrição	Tipo	Prioridade
RF001	Permitir criar administradores	Funcional	Alta
RF002	Permitir criar unidades	Funcional	Alta
RF003	Permitir deletar unidades	Funcional	Média
RF004	Necessita realizar login para utilizar o sistema	Funcional	Alta

7. Requisitos Não Funcionais

Durante a definição desta aplicação, alguns requisitos foram levantados, mas não havia a obrigatoriedade de sua implementação, são eles:

ID	Descrição	Tipo	Prioridade
RNF001	Permitir a execução da aplicação em Docker	Não Funcional	Média
RNF002	Autenticação de 2FA para logins de administradores	Não Funcional	Alta
RNF003	Disponibilizar tabelas segmentadas para cada unidade hospitalar	Não Funcional	Baixa

8. Modelagem e Arquitetura

A API foi construída seguindo o modelo MVC (Model – View – Controller) e foi organizada de forma que cada funcionalidade possui seu próprio arquivo e definições, facilitando a manutenção e implementação de novas funcionalidades.

A “services” possui todas as requisições que são feitas ao banco, sendo assim, a camada de controller só possui responsabilidade das regras de negócio e validações de dados, toda query enviada ao banco é feita de forma separada do resto, tornando as regras agnósticas ao controller.

A estrutura de pastas ficou desta forma:

```
VidaPlus/
 |— docs/
 |— mock/
 |— log/
 |— prisma/
 | — src/
 |  |— controllers/
 |  |— middlewares/
 |  |— routes/
 |  |— schemas/
 |  |— services/
 |  |— utils/
 |  |— lib/
 |  |— index.ts
```

9. Diagramas

Abaixo temos dois diagramas desta aplicação, sendo eles Diagrama de Caso e Diagrama de Entidade-Relacionamento.

As tabelas PROFISSIONAL e PACIENTE possuem como FK (Foreign Key) o ID da tabela UNIDADE.

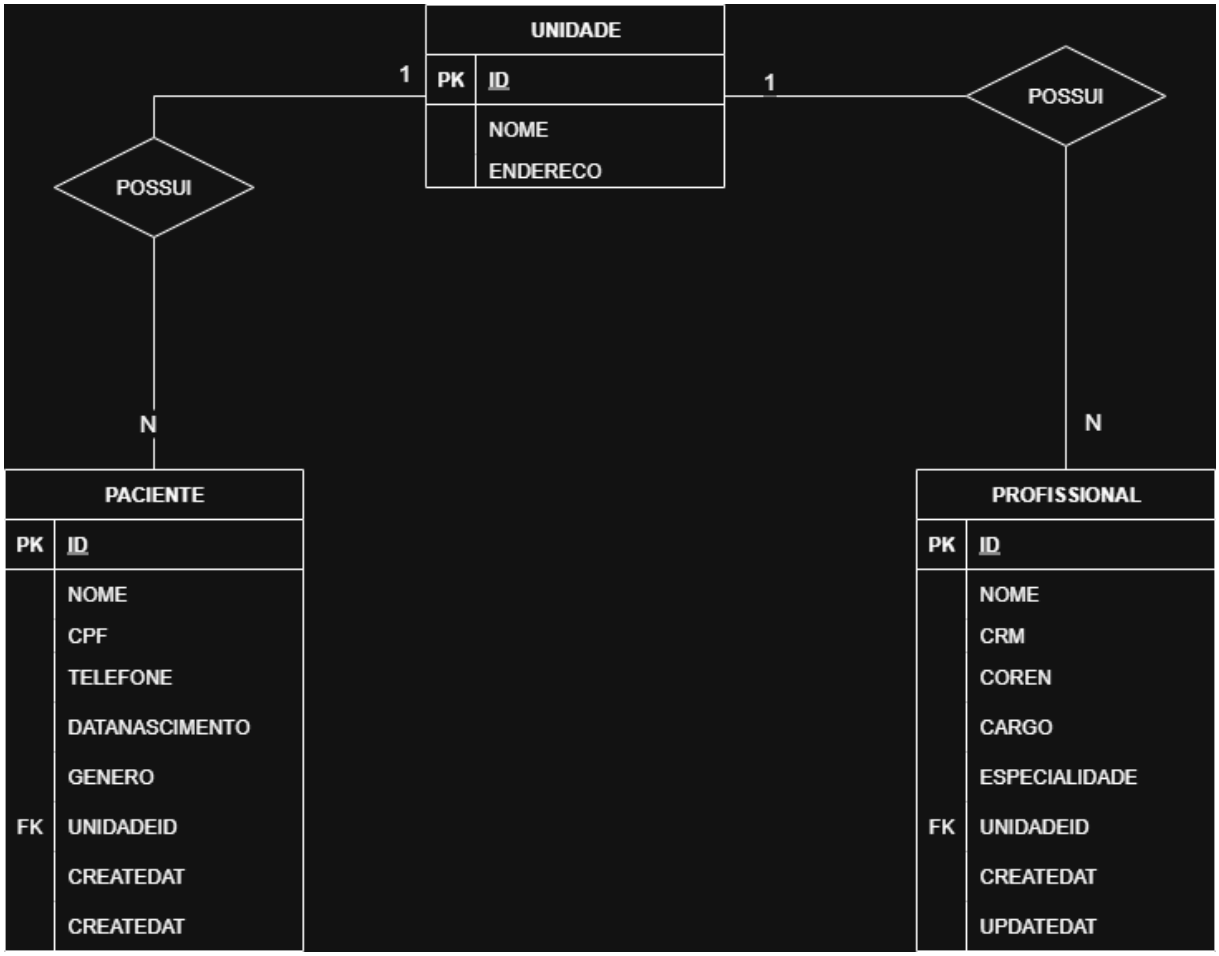

Figura 1. Diagrama de Entidade e Relacionamento

10. Casos de Uso

O ator “usuário” podem realizar todo o CRUD relacionado ao paciente dentro da unidade.

O Profissional pode registrar internações e demais perfis de profissionais, além de poder ver dados dos pacientes.

O Administrador possui acesso completo ao sistema e a rotas únicas envolvendo os dados no banco, sendo o ator mais importante da API.

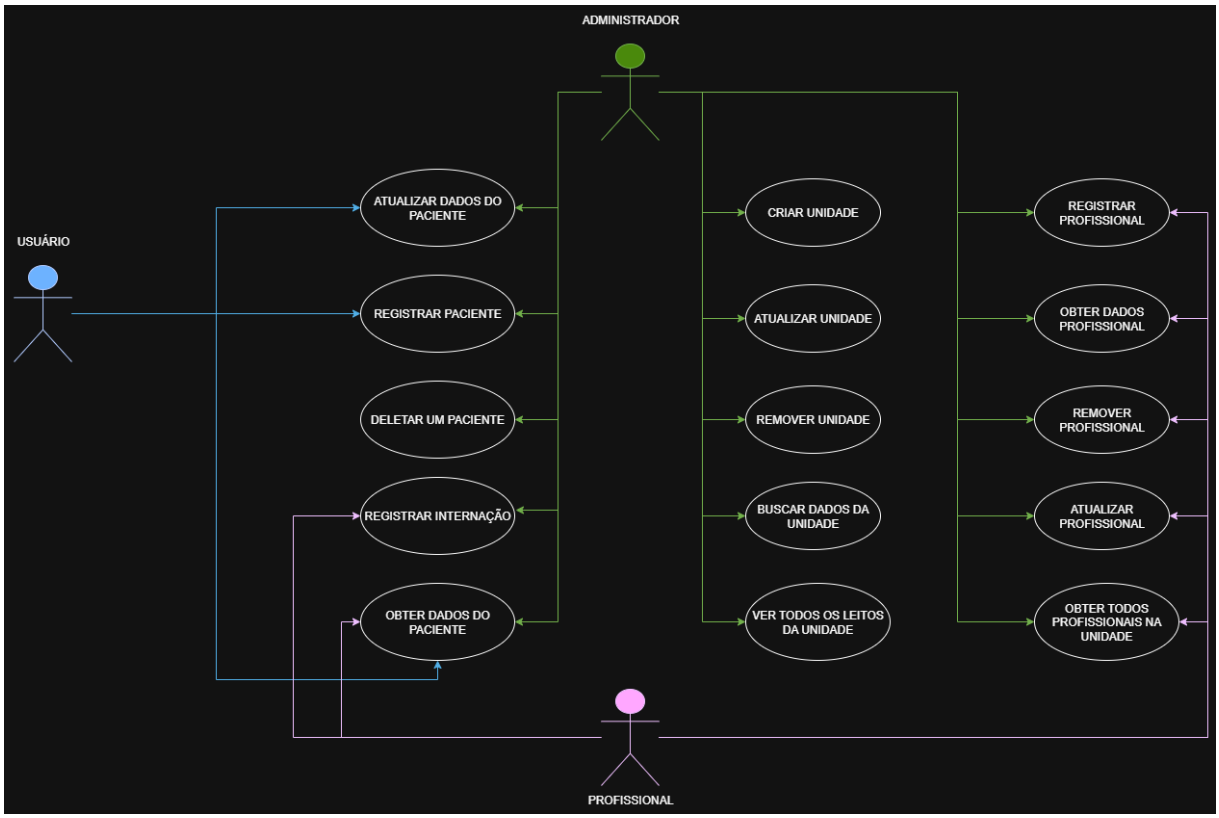

Figura 2. Caso de Uso

11. Implementação

Cada rota da aplicação possui algumas rotas exclusivas, além do CRUD para cada funcionalidade, como mencionado anteriormente, a aplicação possui uma camada de registro e login de usuários, veremos abaixo algumas das rotas desta API.

Administradores

- **POST /admin/register**
  - Rota responsável pela criação dos administradores no Sistema
  - Necessário envio de um objeto JSON com os dados para criação do usuário.
    - Nome
    - Email
    - Senha
    - ConfirmaSenha
    - Perfil
  - Deve retornar como resposta um token válido para login

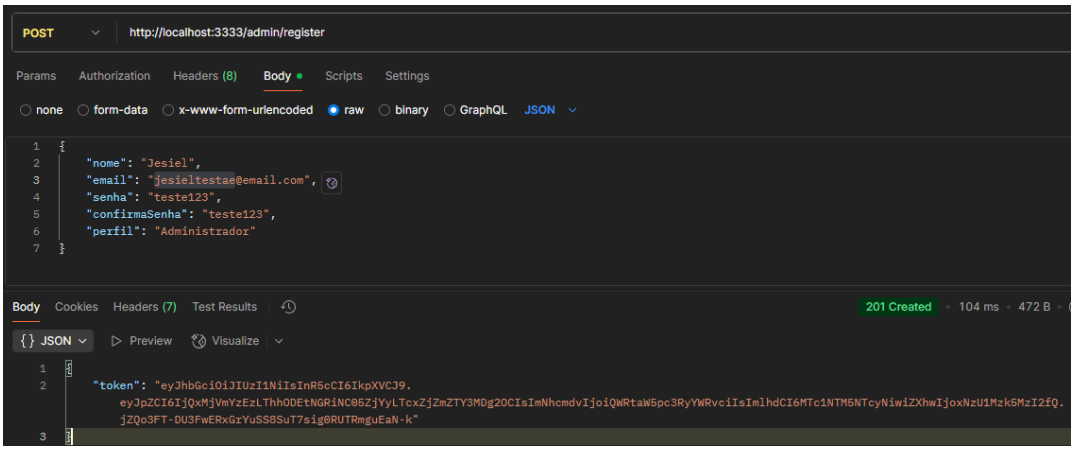

Figura 4. Cadastro de Administrador com sucesso

- **POST /admin/login**
    - Rota responsável pelo Login dos usuários na rede.
    - Necessário o envio de um objeto JSON com os dados para login do usuário.
        - Nome
        - Email
        - Senha
    - Deve retornar um token válido para login que é OBRIGATÓRIO para autenticação das rotas protegidas.

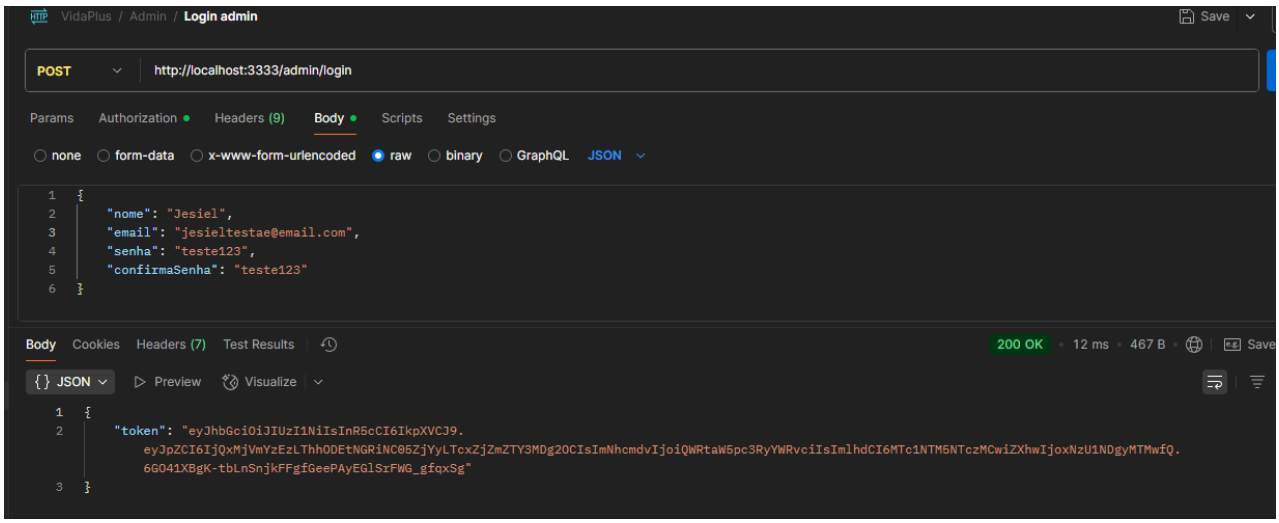

Figura 4. Login com sucesso

- **GET /admin/**
    - Rota responsável pela busca dos dados do administrador
    - Necessário o envio de um objeto JSON com o e-mail do administrador e deve conter um bearer <token> valido para a operação, ou seja, usuário ADM precisa estar logado.
        - Email
    - Deve objeto JSON com os dados do administrador

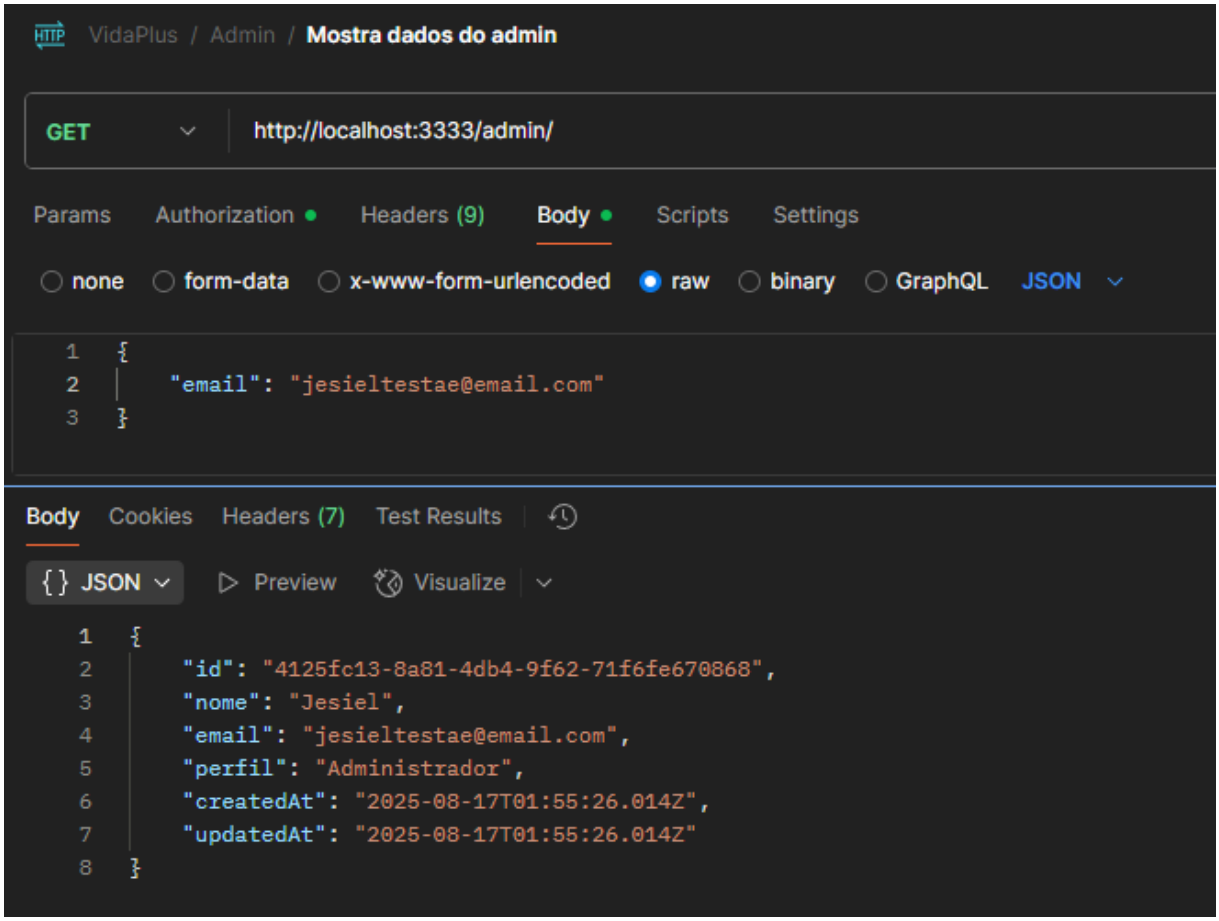

Figura 5. Buscando dados do administrador

- **DELETE /admin/<ID_DO_ADMINISTRADOR>**
    - Rota responsável pela exclusão de um administrador no banco de dados.
    - Necessário o envio de um ID na URL e deve conter um bearer <token> valido para a operação, ou seja, usuário ADM precisa estar logado.
    - Deve uma mensagem de acordo com o resultado da operação.

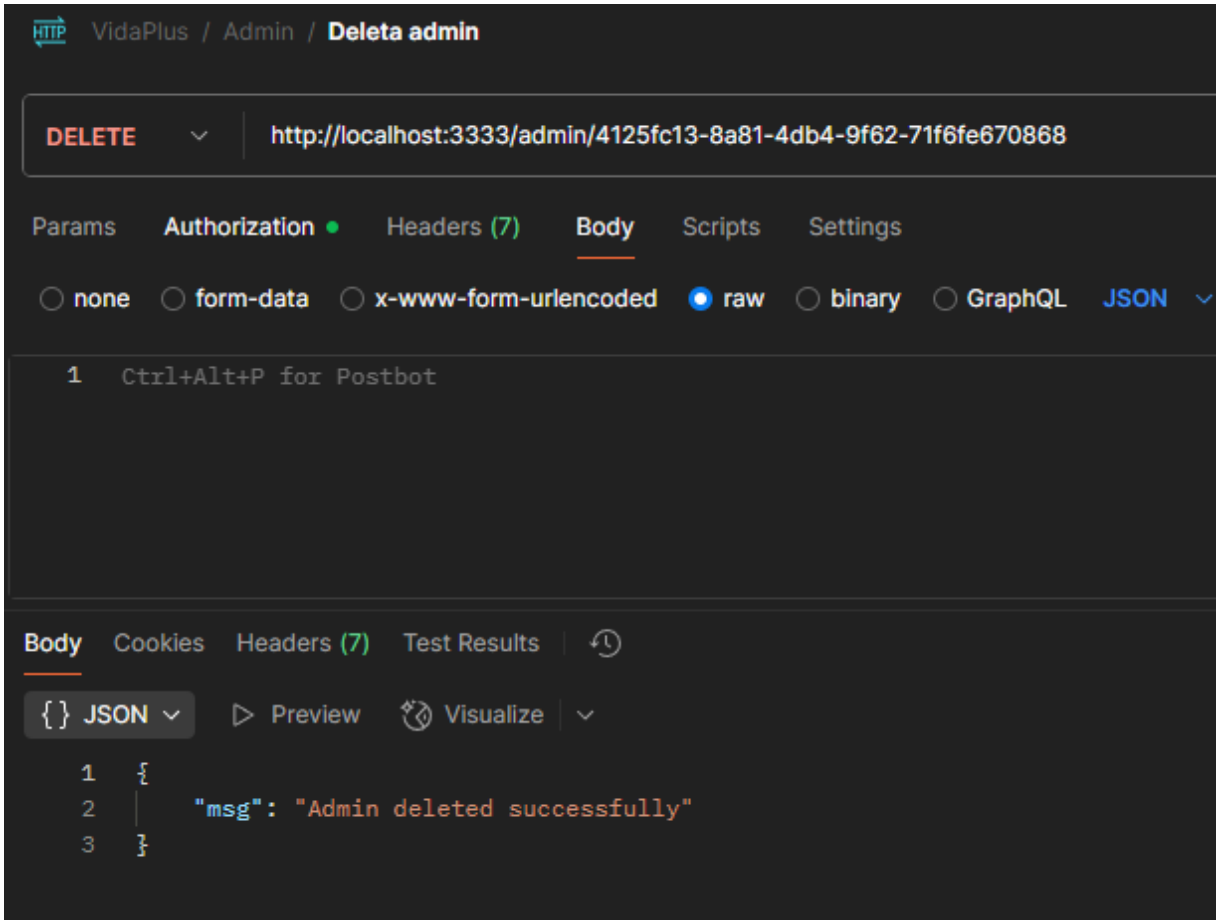

Figura 6. Deletando um administrador

Profissionais

- **POST /profissional/**
  - Rota responsável pela criação dos profissionais no Sistema
  - Necessário envio de um objeto JSON com os dados para criação do profissional.
    - Nome
    - CRM
    - COREN
    - Cargo
    - Especialidade
    - ID_DA_UNIDADE_HOSPITALAR
  - Deve uma mensagem de acordo com o resultado da operação.

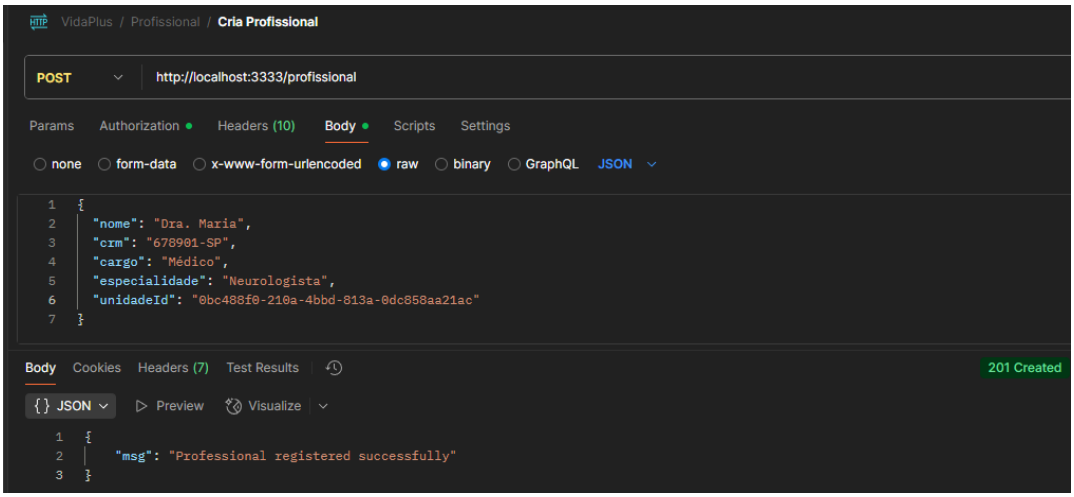

Figura 6. Criando um profissional

- **UPDATE /profissional/<ID_DO_PROFISSIONAL>**
    - Rota responsável pela atualização dos dados de um profissional no banco de dados.
    - Necessário o envio de um ID na URL e deve conter um bearer <token> valido para a operação, ou seja, usuário ADM precisa estar logado.
    - Deve uma mensagem de acordo com o resultado da operação e um objeto JSON com os dados atualizados.

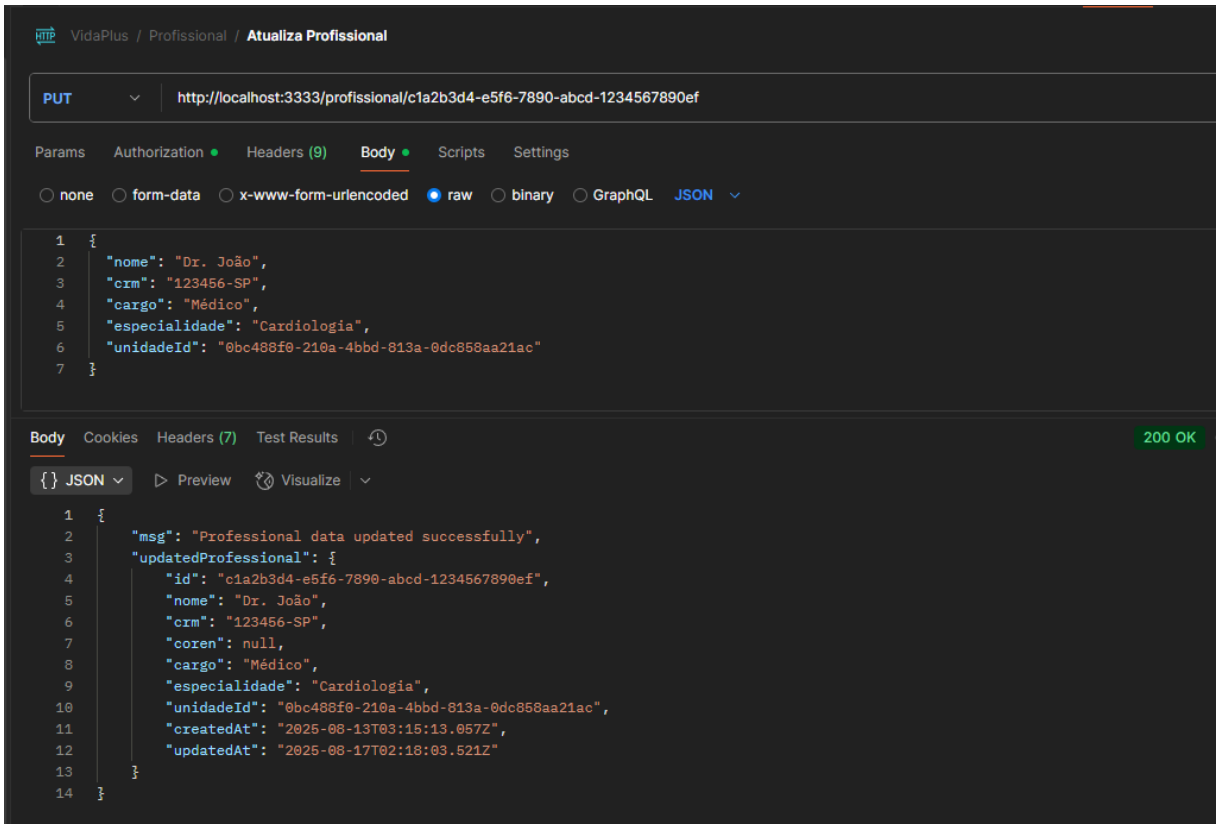

Figura 7. Atualizando os dados de um profissional

- **GET /profissional/<ID_DO_PROFISSIONAL>**
    - Rota responsável pela busca dos dados do administrador
    - Necessário o envio de um ID na URL e deve conter um bearer <token> valido para a operação, ou seja, usuário ADM precisa estar logado.
    - Deve objeto JSON com os dados do administrador

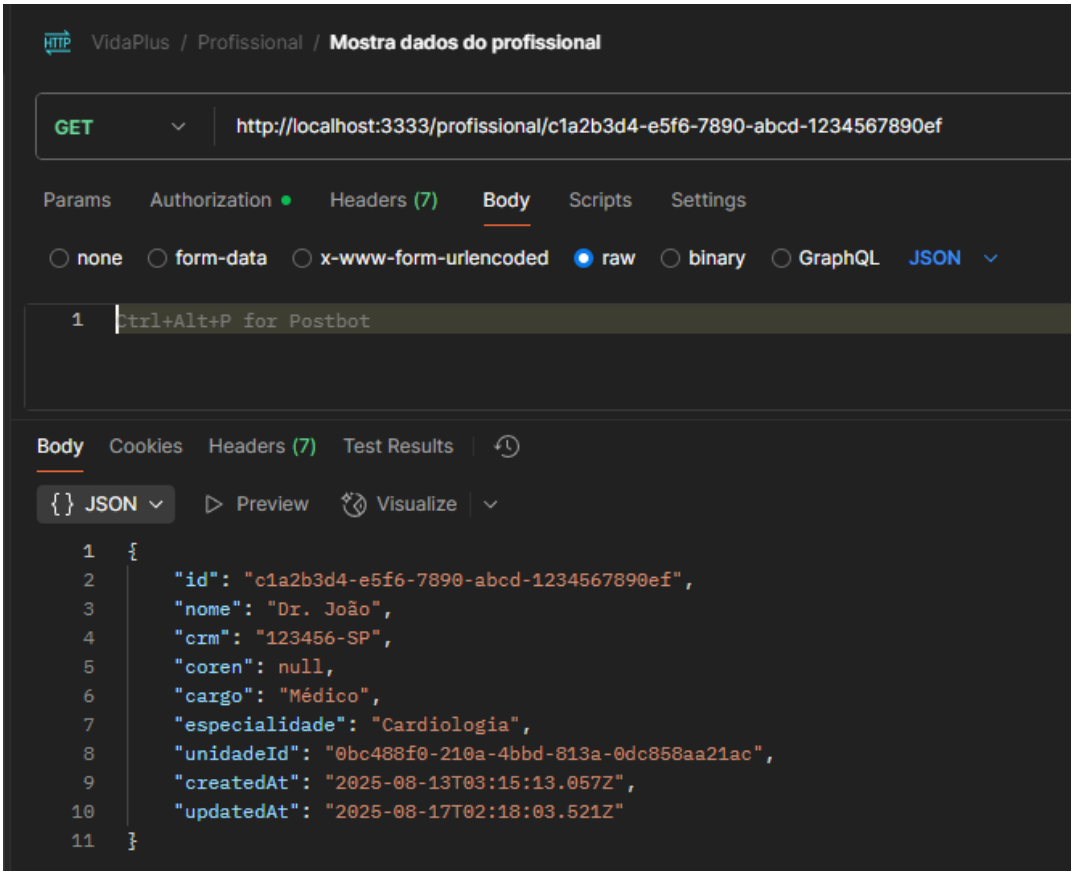

Figura 8. Obtendo os dados de um profissional

- **DELETE /profissional/<ID_DO_PROFISSIONAL>**
    - Rota responsável exclusão de um profissional
    - Necessário o envio de um ID na URl e deve conter um bearer <token> valido para a operação, ou seja, usuário ADM precisa estar logado.
    - Deve uma mensagem com base no status da operação

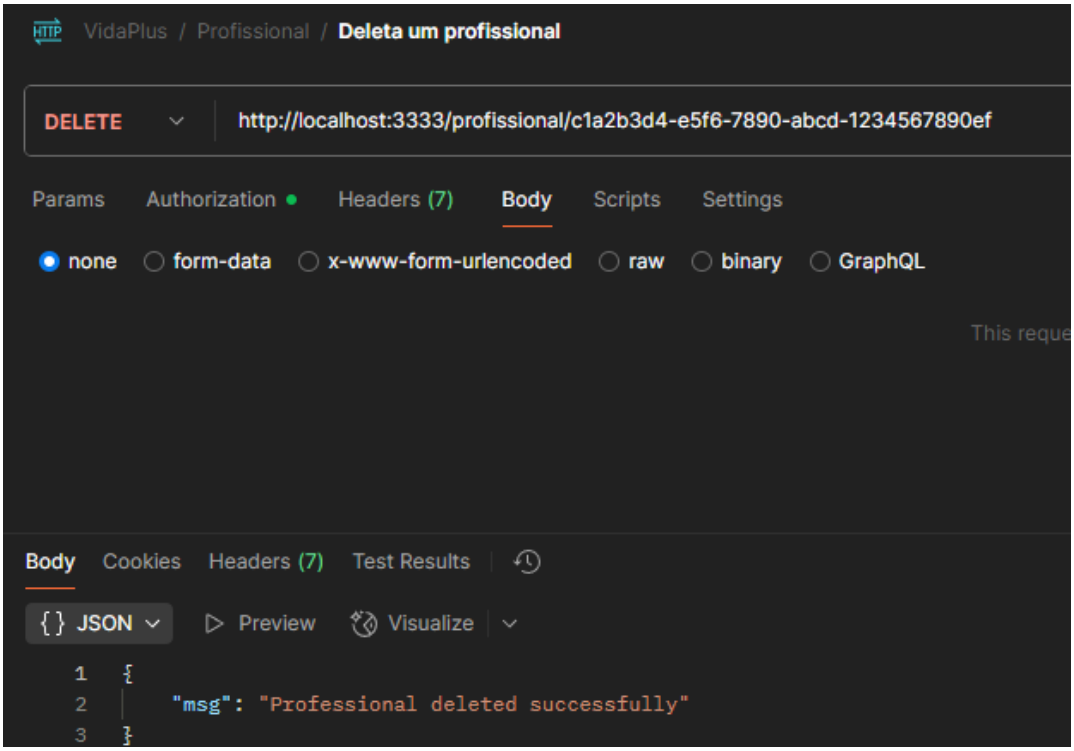

Figura 9. Deletando um profissional da base de dados

12. Plano de Testes

Abaixo veremos os planos de testes para criação de usuários no sistema, tal cenário é necessário para definir em quais funcionalidades os usuários cadastrados terão acesso.

Todas as rotas, exceto /admin/login e /admin/register são protegidas com um tipo de perfil do usuário, sendo assim cada requisição do plano de testes deve ser feita com de acordo com as regras da rota.

O token é do tipo “Bearer <token>” e deve ser válido para que tenha sucesso na requisição.

Os perfis dentro da aplicação são: “Administrador”, “Profissional” e “Usuário”.

Testes para rota /admin:

- ADM-001 - Criação de usuário administrador:
    - Pré-requisitos: Definir perfil do usuário, neste caso, administrador
    - Passos:
        - Deve conter dados de nome, e-mail, senha, confirmação da senha inserida e o perfil, para este último campo, obrigatoriamente deve conter a palavra “Administrador”
    - Resultados com erro:
        - Caso o e-mail já estiver sendo usado deve retornar erro informando que não é possível criar um usuário.
        - Caso a senha e a confirmação da senha estiverem diferentes, deve retornar mensagem informando que as senhas não são iguais.
    - Resultado com sucesso
        - Deve retornar um objeto com a chave “token” e o valor do token criado para que seja possível realizar login no frontend

- ADM-002 - Login de um usuário já cadastrado:
    - Pré-requisitos: Ter executado o teste ADM-001 para criação do usuário
    - Passos:
        - Enviar uma requisição com objeto JSON tendo os dados de nome, senha e e-mail cadastrado
    - Resultados com erro:
        - Os obrigatórios são e-mail e senha, que devem existir para que possa realizar o login, caso não exista, retorna erro informando quem um dos campos está errado, sem especificar.
    - Resultado com sucesso
        - Deve retornar um objeto com o token válido para sessão.

- ADM-003 – Obter os dados de um administrador.
    - Pré-requisitos: Ter executado os testes ADM-001 e ADM-002 com sucesso.
    - Passos:
        - Deve conter o token válido e autenticado na requisição, formato “Bearer <token>”
    - Resultados com erro:
        - Caso o token esteja expirado, retorna um objeto JSON informando para realizar login novamente
    - Resultado com sucesso
        - Deve retornar um objeto JSON contendo todos os dados do usuário, exceto o campo senha.

Testes para rota /profissional:

- PRO-001 - **POST /profissional/** – Registrar um profissional no sistema:
    - Pré-requisitos:
        - Deve ter executado o passo ADM-002 com sucesso usando perfil “Administrador” ou “Profissional”
        - Deve registrar uma unidade hospitalar caso não exista no sistema
    - Passos:
        - Deve ter na requisição um token válido e autenticado.
        - Deve enviar um objeto JSON contendo os campos obrigatórios de nome, CRM/COREN e ID da Unidade.
    - Resultados com erro:
        - Caso a unidade hospitalar não exista, retorna erro.
        - Caso o token esteja expirado ou invalido, retorna mensagem com erro.
    - Resultado com sucesso
        - Retorna os novos dados do usuário e uma mensagem informando sucesso.

- PRO-002 - **PUT/profissional/{ID} -** Atualizar os dados do profissional:
    - Pré-requisitos:
        - Deve ter executado o passo ADM-002 com sucesso usando perfil “Administrador” ou “Profissional”
        - Deve registrar uma unidade hospitalar caso não exista no sistema
    - Passos:
        - Deve ter na requisição um token válido e autenticado.
        - Deve enviar na URL o ID do usuário a ser alterado.

        - Deve enviar um objeto JSON contendo os dados do novo profissional a ser registrado no sistema, contendo os campos obrigatórios de nome, CRM/COREN e ID da Unidade.
    - Resultados com erro:
        - Caso o profissional não exista, retorna erro.
        - Caso o token esteja expirado ou invalido, retorna mensagem com erro.
    - Resultado com sucesso
        - Retorna os novos dados do usuário e uma mensagem informando sucesso.

- PRO-003 – **GET /profissional/{ID}** - Obter os dados do profissional.
    - Pré-requisitos: Deve ter executado os passos ADM-002, PRO-001 com sucesso. Obrigatoriamente deve ser um usuário “Administrador” ou “Profissional”
    - Passos:
        - Deve enviar uma requisição com token válido.
        - Deve ser enviado uma requisição com ID do usuário
    - Resultados com erro:
        - Caso o profissional não exista, retorna erro.
    - Resultado com sucesso
        - Retorna os dados do profissional registrado.

- PRO-004 **DELETE /profissional/{ID}** – Remover um profissional do sistema
    - Pré-requisitos: Deve ter executado os passos ADM-002 e PRO-001. O usuário deve ter perfil de “Administrador”
    - Passos:
        - Deve enviar na requisição um token válido e autenticado.
        - Deve enviar o ID do profissional a ser deletado
    - Resultados com erro:
        - Caso o profissional não exista, retorna erro.
    - Resultado com sucesso
        - Retorna os dados do profissional registrado.

13. Conclusão

O desenvolvimento deste projeto permitiu estruturar de forma clara e eficiente as operações CRUD, base para qualquer desenvolvedor backend.

A implementação dos endpoints seguiu boas práticas de controle de acesso, exigindo tokens válidos e perfis apropriados para cada operação, o que fortalece a confiabilidade do sistema e reduz riscos de acesso indevido. Também ficou evidente a importância de tratar cenários de erro, como ausência de unidades ou profissionais, e de retornar mensagens claras ao usuário, facilitando o uso da aplicação.

Em suma, este projeto proporcionou um aprendizado significativo sobre arquitetura de APIs, segurança, tratamento de erros e clareza na documentação, mostrando que grandes projetos só terão sucessos se houver uma boa pesquisa e levantamento dos requisitos.

14. Referências

[REF0] Documentação e praticas com PRISMA. Disponível em: https://www.prisma.io/docs/orm/overview/databases/postgresql.

[REF1] Documentação para instalar DOCKER. Disponível em: https://docs.docker.com/desktop/setup/install/windows-install/.

[REF2] Aulas 2, 4 e 5 da disciplina “Análise de Sistemas”.

[REF3] Aula 7, atividade prática da disciplina “Banco de Dados Relacional”.

[REF4] DevMedia. O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML. Disponível em: https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408