VIDAPLUS SGHSS
Sistema de Gerenciamento Hospitalar

Autor: Deleon da Paixão Santos
RU: 4556494
Curso: CST – Análise e Desenvolvimento de Sistemas
Polo de Apoio: Itapevi
Disciplina: Desenvolvimento Back-end
Professor(a): Luciana Yanase Hiabara Kanashiro
Título: Documentação Técnica do Sistema VIDAPLUS SGHSS
Ano: 2025

Sumário

1. Introdução

O **VIDAPLUS SGHSS (Sistema de Gerenciamento Hospitalar)** é uma aplicação **Back-end** desenvolvida em **Python** com o framework **Flask**, projetada para gerenciar **consultas médicas**.

O objetivo principal é **automatizar o fluxo de agendamento e registro de atendimentos** através de autenticação de usuários e rastreabilidade de logs e tokens JWT.

A aplicação segue boas práticas de desenvolvimento seguro e está em conformidade com a **Lei Geral de Proteção de Dados (LGPD)**, garantindo o sigilo e integridade das informações médicas e pessoais através de criptografia.

2. Requisitos Funcionais e Não Funcionais

Requisitos Funcionais

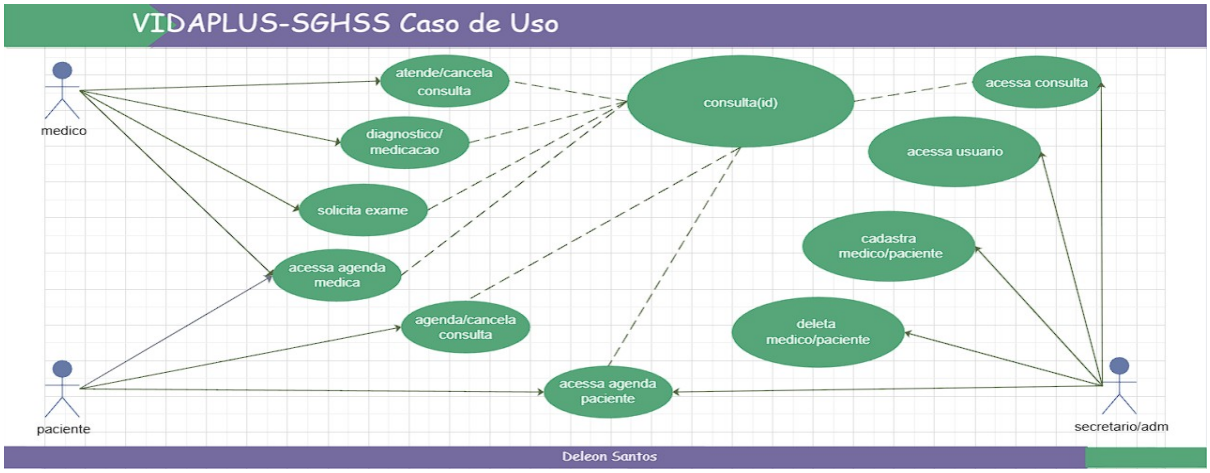
ID	Descrição	Prioridade
RF01	Cadastrar pacientes	Alta
RF02	Cadastrar médicos	Alta
RF03	Cadastrar secretarias	Alta
RF04	Realizar autenticação (login)	Alta
RF05	Agendar consultas	Alta
RF06	Listar e filtrar consultas (por status e data)	Média
RF07	Realizar consulta e registrar prescrição	Média
RF08	Gerar logs de atividade	Média
RF09	Documentar rotas via Swagger	Alta
RF10	Deletar paciente	Alta
RF11	Deletar médico	Alta
RF12	Editar senha de autenticação	Alta
RF13	Cancelar consulta	Alta
RF14	Ver agenda	Alta

Requisitos Não Funcionais

ID	Descrição
RNF01	O sistema deve ser implementado em **Python 3.12** com **Flask**
RNF02	O banco de dados utilizado deve ser **SQLite3**
RNF03	A API deve seguir o padrão **RESTful**
RNF04	O sistema deve utilizar **JWT** para autenticação
RNF05	As senhas devem ser protegidas por criptografias
RNF06	O sistema deve registrar logs de operações sensíveis
RNF07	Deve ser possível testar via **Swagger**, **Postman**

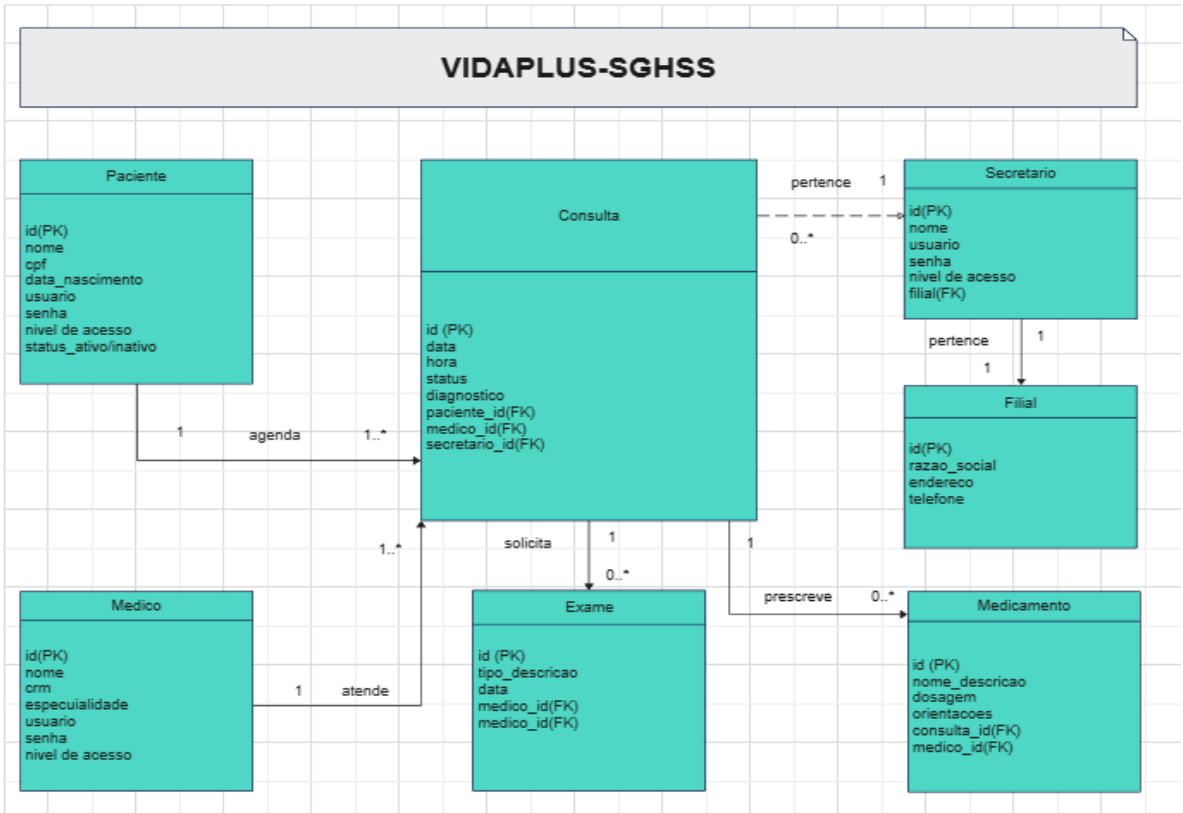
3. Modelagem

3.1 Diagrama de Caso de Uso



(Representação simplificada do caso de uso .)

3.2 Diagrama de Classes



(Diagrama de classe e relacionamentos)

3.3 Estrutura do Banco de Dados

- Tabela Consultas

```
CREATE TABLE consultas (
  id INTEGER NOT NULL,
  data DATE NOT NULL,
  hora TIME NOT NULL,
  paciente_id INTEGER NOT NULL,
  medico_id INTEGER NOT NULL,
  secretario_id INTEGER,
  status VARCHAR(30),
  diagnostico TEXT,
  PRIMARY KEY (id),
  FOREIGN KEY (paciente_id) REFERENCES pacientes (id),
  FOREIGN KEY (medico_id) REFERENCES medicos (id),
  FOREIGN KEY (secretario_id) REFERENCES secretarios (id)
)
```

- Tabela Paciente

```
CREATE TABLE pacientes (
  id INTEGER NOT NULL,
  nome VARCHAR(120) NOT NULL,
  data_nascimento DATE,
  cpf VARCHAR(20),
  usuario VARCHAR(80) NOT NULL,
  senha VARCHAR(20) NOT NULL,
  nivel_acesso VARCHAR(20) NOT NULL,
  ativo BOOLEAN,
  PRIMARY KEY (id),
  UNIQUE (cpf),
  UNIQUE (usuario)
)
```

- Tabela Médicos

```
CREATE TABLE medicos (
  id INTEGER NOT NULL,
  nome VARCHAR(120) NOT NULL,
  crm VARCHAR(50),
  especialidade VARCHAR(80),
  usuario VARCHAR(80) NOT NULL,
  senha VARCHAR(20) NOT NULL,
  nivel_acesso VARCHAR(20) NOT NULL,
  PRIMARY KEY (id),
  UNIQUE (crm),
  UNIQUE (usuario)
)
```

- Tabela Medicamentos

```
CREATE TABLE medicamentos (
  id INTEGER NOT NULL,
  nome VARCHAR(200) NOT NULL,
  dosagem VARCHAR(120),
  orientacoes TEXT,
  consulta_id INTEGER NOT NULL,
  medico_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (consulta_id) REFERENCES consultas (id),
  FOREIGN KEY (medico_id) REFERENCES medicos (id)
)
```

- Tabela Exame

```
CREATE TABLE exames (
  id INTEGER NOT NULL,
  tipo VARCHAR(200) NOT NULL,
  data_solicitacao DATE,
  consulta_id INTEGER NOT NULL,
  medico_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (consulta_id) REFERENCES consultas (id),
  FOREIGN KEY (medico_id) REFERENCES medicos (id)
)
```

- Tabela Secretários

```
CREATE TABLE secretarios (
  id INTEGER NOT NULL,
  nome VARCHAR(120) NOT NULL,
  matricula VARCHAR(50),
  usuario VARCHAR(80) NOT NULL,
  senha VARCHAR(20) NOT NULL,
  nivel_acesso VARCHAR(20) NOT NULL,
  filial_id INTEGER NOT NULL,
  PRIMARY KEY (id),
  UNIQUE (matricula),
  UNIQUE (usuario),
  FOREIGN KEY (filial_id) REFERENCES filiais (id)
)
```

- Tabela Filiais

```
CREATE TABLE filiais (
  id INTEGER NOT NULL,
  nome VARCHAR(120) NOT NULL,
  endereco VARCHAR(200) NOT NULL,
  telefone VARCHAR(20) NOT NULL,
  PRIMARY KEY (id)
)
```

4. Implementação

O repositório remoto desta aplicação é público está disponível no link: https://github.com/Deleon-Santos/SGHSS, acessando o Readme encontraram mais informações e o guia de como clonar o repositório e instalar as dependências.

4.1 Arquitetura do Sistema

O projeto SGHSS foi estruturado em camadas (MVC) e módulos para melhor organização do código.

A pasta models representa a camada Models comporta todas as entidades e relacionamentos que compõem o banco de dados.

A pasta static no nosso projeto abriga a interface e como as rotas serão visualizadas no navegador (camada View).

A pasta routs abriga todas as rotas que controlam a navegação pelo sistema (camada Control).

```
SGHSS/
├── app/
│   ├── database/
│   ├── models/
│   ├── routes/
│   └── static/
├── docs/
├── requirements.txt
├── run.py
├── .gitignore
└── README.md
```

A api está pronta para integração com um front-end simples ou sistemas móveis através da biblioteca Flask-CORS.

```python
from app import create_app
from flask_cors import CORS

app = create_app()
#A API está liberada para todas as origens
CORS(app)

if __name__ == '__main__':
    app.run(debug=True)
```

(O ‘CORS’ prepara a api para a integração com front-end em ambiente de testes)

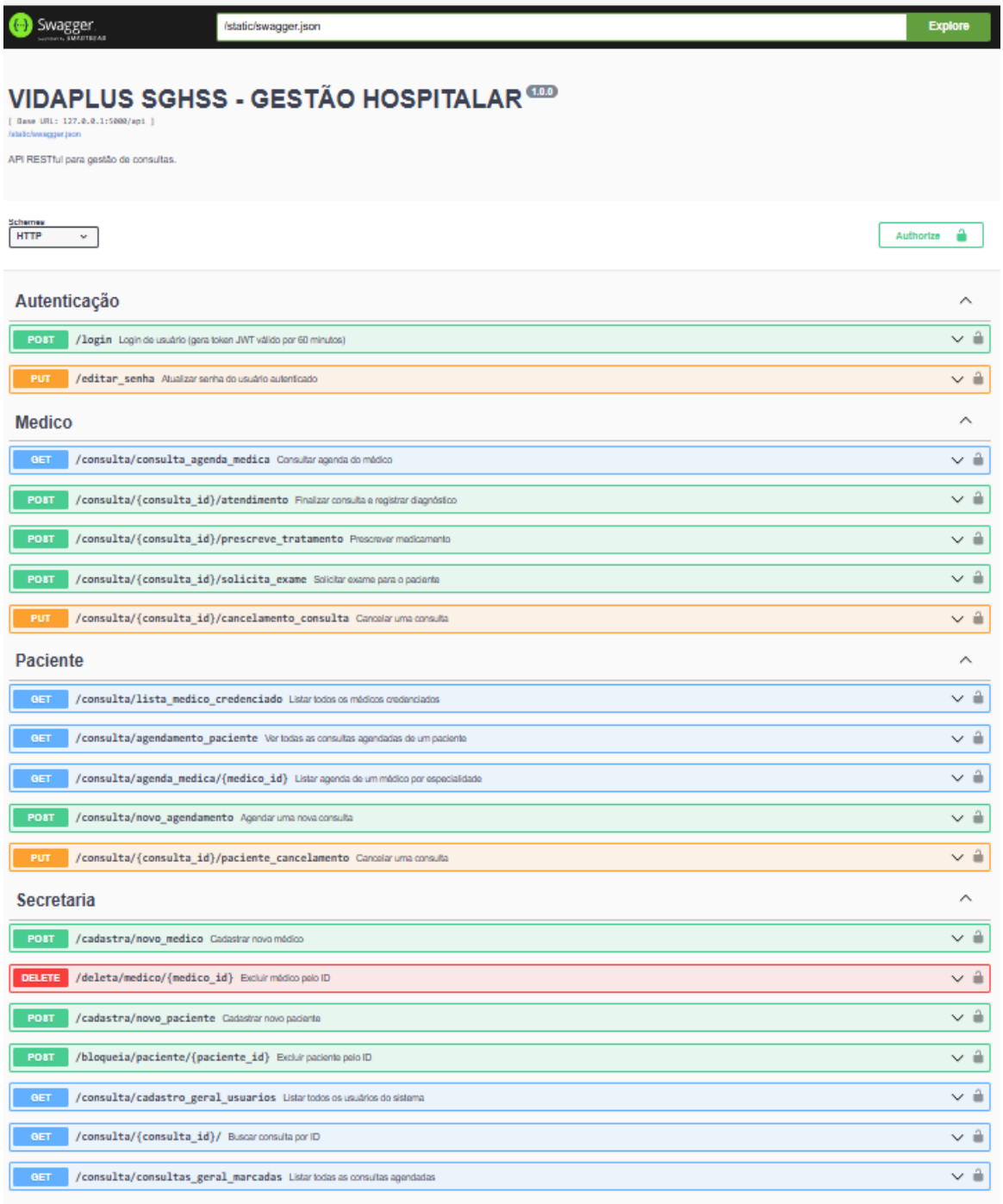
As rotas estão divididas em blueprints para cada tipo de usuário (médico, paciente, secretario e Login).

```python
from app.routes.consultas_medico import consultas_medico_bp
from app.routes.consultas_paciente import consultas_paciente_bp
from app.routes.consultas_secretaria import consultas_secretaria_bp
from app.routes.login import login_bp

app.register_blueprint(consultas_medico_bp, url_prefix="/api")
app.register_blueprint(consultas_paciente_bp, url_prefix="/api")
app.register_blueprint(consultas_secretaria_bp, url_prefix="/api")
app.register_blueprint(login_bp, url_prefix="/api")
```

Para persistência de dados foi configurado o SQLite e é populado com dados iniciais ao carregar a aplicação. A informações adicionadas já possibilitam os testes mais primários de integração da nossa APIrest.

```python
# Criação da filial
filial1 = Filial(nome='Filial Central', endereco='Rua Principal, 123', telefone='1234-5678')
db.session.add(filial1)

# Criação do secretário
sec1 = Secretario(nome='Deleon Santos', matricula='4556949', usuario= 'deleon@santos', senha=generate_password_hash('4556949'), nivel_acesso='secretario',filial_id=1)
db.session.add(sec1)

# Criação dos médicos
plant = Medico(nome='Plantonista', crm='CRM000', especialidade='Plantonista', usuario='plantonista@plantonista', senha=generate_password_hash('crm000'), nivel_acesso='medico')
med1 = Medico(nome='Dr. João Alcantara', crm='CRM123', especialidade='Clinico_geral', usuario='joao@alcantara', senha=generate_password_hash('crm123'), nivel_acesso='medico')
med2 = Medico(nome='Dr. Maria Magalhaes', crm='CRM456', especialidade='Pediatria', usuario='maria@magalhaes', senha=generate_password_hash('crm456'), nivel_acesso='medico')
med3 = Medico(nome='Dr. Pedro Cardoso', crm='CRM789', especialidade='Obstetricia', usuario='pedro@cardoso', senha=generate_password_hash('crm789'), nivel_acesso='medico')
db.session.add_all([med1, med2, med3, plant])

# Criação dos pacientes
pac1 = Paciente(nome='Alice Souza', data_nascimento=date(2000, 12, 12), cpf='12345678901', usuario='alice@souza', senha=generate_password_hash('12345678901'), nivel_acesso='paciente')
pac2 = Paciente(nome='Bruno Santos', data_nascimento=date(2000, 8, 12), cpf='23456789012', usuario='bruno@santos', senha=generate_password_hash('23456789012'), nivel_acesso='paciente')
pac3 = Paciente(nome='Carla Castro' , data_nascimento=date(2000, 10, 12), cpf='34567890123', usuario='carla@caastro', senha=generate_password_hash('34567890123'), nivel_acesso='paciente')
db.session.add_all([pac1, pac2, pac3])

db.session.commit()  # Precisa para pegar IDs

# Criação das consultas (1 paciente por médico)
cons1 = Consulta(medico_id=med1.id, paciente_id=pac1.id, data=datetime.now().date(), hora=time(9, 0), secretario_id=sec1.id,status='Agendada')
cons2 = Consulta(medico_id=med2.id, paciente_id=pac2.id, data=datetime.now().date(), hora=time(10, 0),secretario_id=sec1.id,status='Agendada')
cons3 = Consulta(medico_id=med3.id, paciente_id=pac3.id, data=datetime.now().date(), hora=time(11, 0),secretario_id=sec1.id, status='Agendada')

db.session.add_all([cons1, cons2, cons3])

db.session.commit()
```

O swagger ui foi integrado para facilitar a documentação e testes da API. Ao executar a aplicação e abrir o link: http://127.0.0.1:5000 a interface Swagger será carregada com todas as rotas possibilitando os testes via Browser.



4.2 Endpoints (Exemplos Principais)

Os testes das principais rotas foram feitos no Postman. O Usuário deve esta logado para acessar as rotas, o sistema valida o nível de cada usuário autenticado e logs de erros serão retornados para acesso não permitido

TABELA ÚNICA — TODOS OS USUÁRIOS

ID	Nome	Usuário	Senha	Nível de Acesso
1	Deleon Santos	deleon@santos	'4556949'	secretario
1	Plantonista	plantonista@plantonista	'crm000'	medico
2	Dr. João Alcantara	joao@alcantara	'crm123'	medico
3	Dr. Maria Magalhaes	maria@magalhaes	'crm456'	medico
4	Dr. Pedro Cardoso	pedro@cardoso	'crm789'	medico
1	Alice Souza	alice@souza	'12345678901'	paciente
2	Bruno Santos	bruno@santos	'23456789012'	paciente
3	Carla Castro	carla@caastro	'34567890123'	paciente

A tabela de usuários oferece os cadastros iniciais (e-mail, senha) e podemos usar os modelos abaixo para testar alguns dos principais requisito no Postman ou no Swagger.

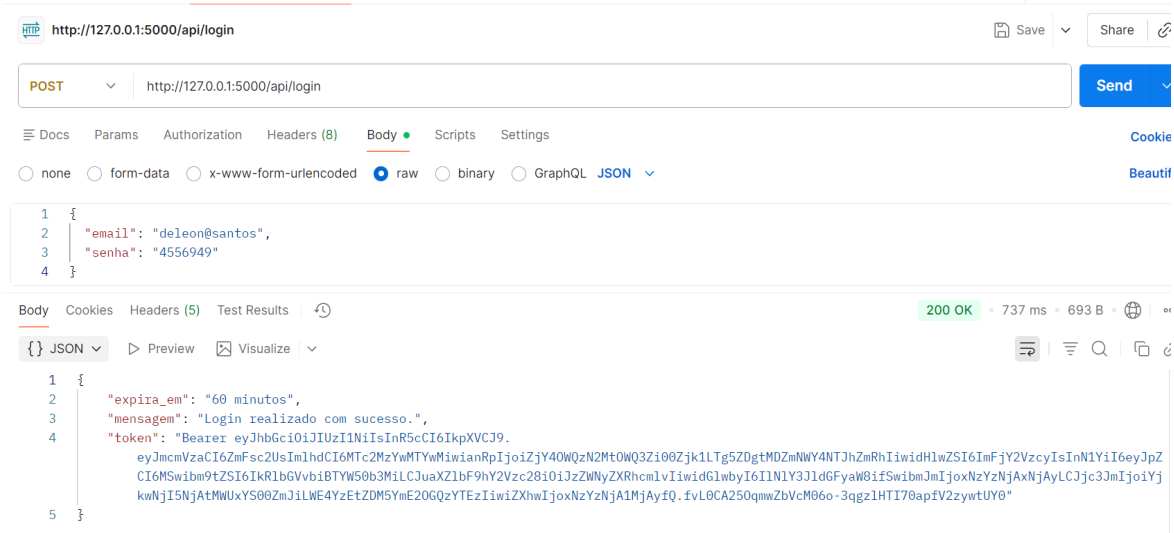
```
//login para obter o token JWT

POST api/login
{
  "usuario": "deleon@santos",
  "senha": "4556949"
}

//altera a senha do usuario autenticado
PUT api/editar_senha
{
  "senha_atual": "4556949",
  "nova_senha": "nova_senha123"
}

//secretario cadastra um novo paciente
POST /api/consulta/cadastro_paciente
{
  "nome": "Maria Silva",
    "cpf": "12345678901",
    "data_nascimento": "1990-05-15",
    "email": "maria@silva",
    "senha": "maria"
}

//paciente agenda uma consulta com o medico de id 2
POST /api/consulta/novo_agendamento
```

```
{
   "medico_id": 2,
   "data": "2024-07-20",
   "hora": "14:30"
}

//medico finaliza o atendimento da consulta de id 1
POST api/consulta/1/atendimento
{
   "diagnostico": "Gripe comum"
}

//medico prescreve tratamento para a consulta de id 1
POST api/consulta/1/prescreve_tratamento
{
   "medicacao": "Paracetamol 500mg,
   "dosagem":"1 comprimido a cada 8 horas por 5 dias"
   "orientacoes": "Descansar e manter-se hidratado"
}
```

- ***Autenticação***

POST: (/api/login)

A autenticação “api/login” requer uma entrada de usuário (e-mail) e senha, o resultado esperado é um “status 200 ok”, indicando que a validação está funcionando (a criação de um token)



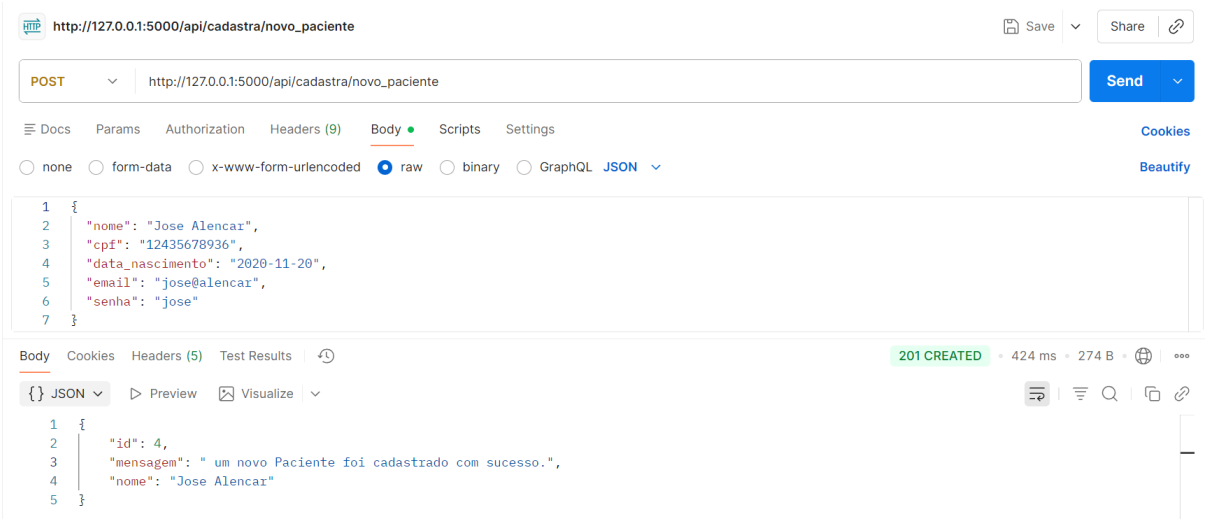
PUT: (/api/editar_senha

Esta rota permite ao usuário autenticado alterar a senha atual. Serão exigidas a senha atual e a nova senha.

- ***Cadastro de Paciente***

POST: (/api/cadastro/novo_paciente

Cadastrar um paciente é permissível ao secretário, requer nome, cpf, data_nascimento, e-mail e senha. Espera-se um log “201 CREATED” para criação bem-sucedida.

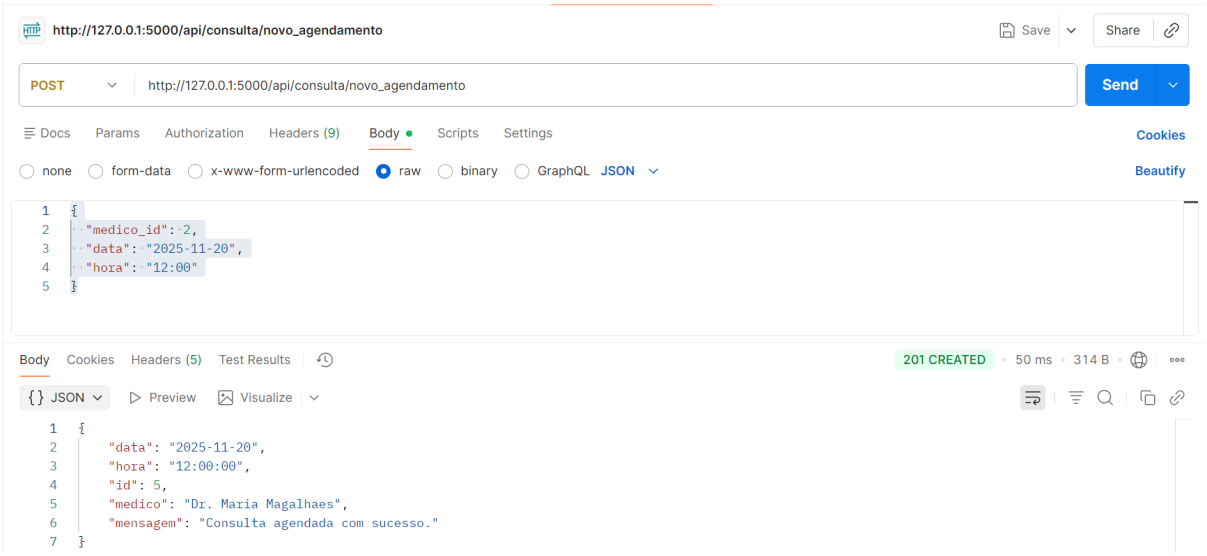


- ***Marcar uma Consulta***

POST: (http://127.0.0.1:5000 /api/consulta/novo_agendamento)

A rota “/api/consulta/novo_agendamento” permite a um paciente autenticado via JWT marcar uma consulta informando o ID do médico, data e hora. O resultado esperado deve ser “201 CREATED” e o retorno no Body.

Os pacientes ainda contam com a possibilidade de ver quais médicos estão credenciados na filial e a agenda parcial de cada médico por ID e ainda cancelar uma consulta por ID.

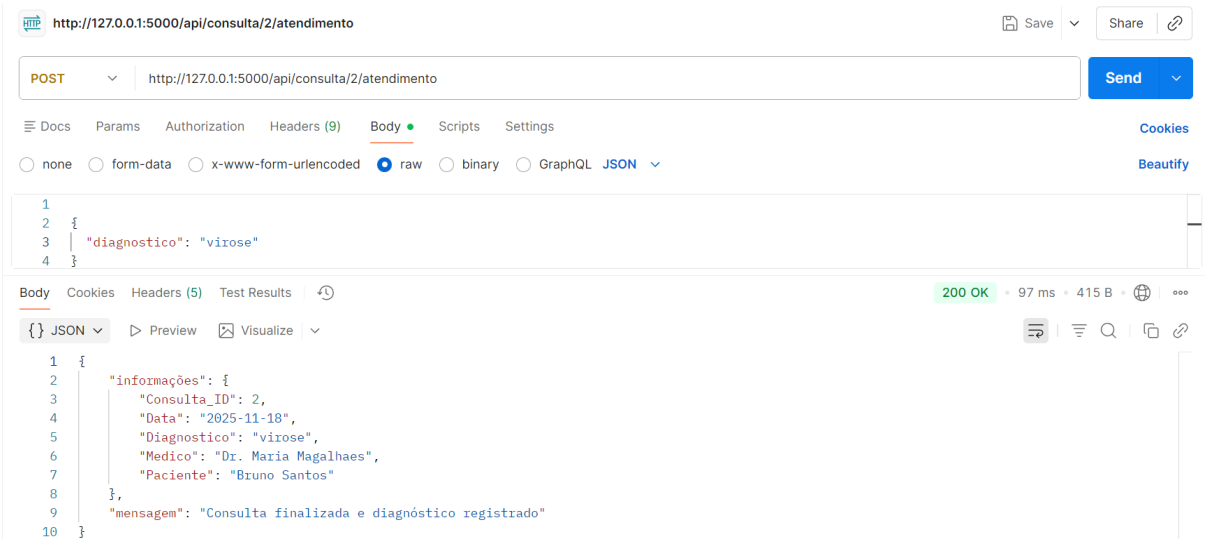


- ***Atender uma Consulta***

POST : (/api/consultas/{consulta_id}/atendimento)

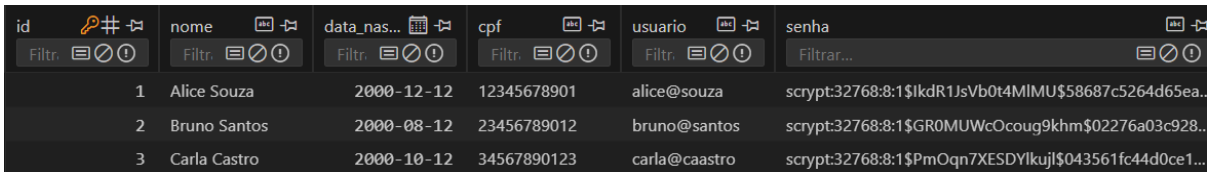
O atendimento de uma consulta será possível apenas ao médico titular e, portanto, o sistema deve retornar log de erro para acesso não permitido.

Ao médico, estará disponível uma rota que retorna todas as consultas de sua agenda, prescrever um tratamento, solicitar exames ou cancelar uma consulta.



4.3 Autenticação e Segurança

A autenticação é feita via flask_jwt_extended para garantir a segurança das informações e autenticação de usuário. As senhas são criptografadas no banco de dados usando a biblioteca werkzeug evitando que informações sensíveis fiquem expostas.



id	nome	data_nas...	cpf	usuario	senha
1	Alice Souza	2000-12-12	12345678901	alice@souza	scrypt:32768:8:1$IkdR1JsVb0t4MlMU$58687c5264d65ea..
2	Bruno Santos	2000-08-12	23456789012	bruno@santos	scrypt:32768:8:1$GR0MUWcOcoug9khm$02276a03c928..
3	Carla Castro	2000-10-12	34567890123	carla@caastro	scrypt:32768:8:1$PmOqn7XESDYlkujl$043561fc44d0ce1...

5. Plano de Testes (Swagger/Postman)

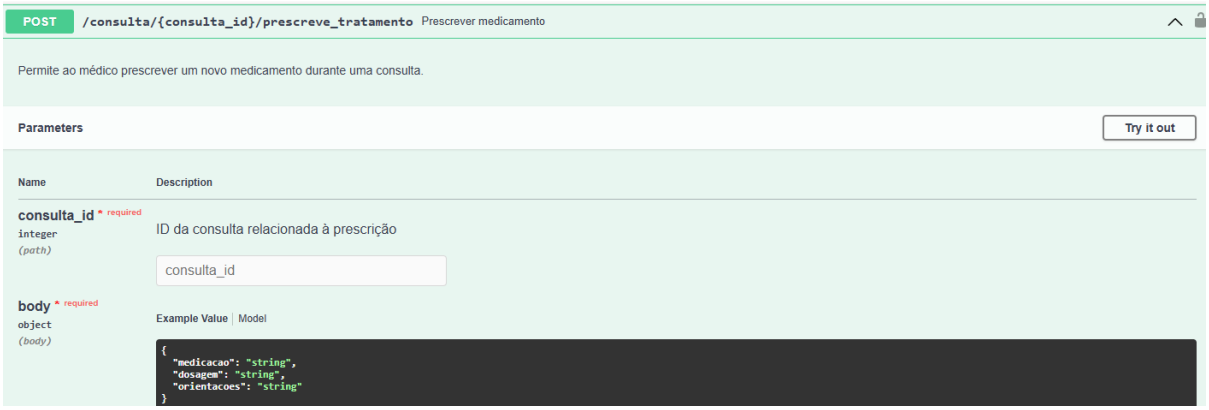
Tabela de Endpoints – SGHSS – Sistema de Gerenciamento Hospitalar

Nesta tabela foram listados todos os endpoints disponível na API e o que cada uma das rotas retornam considerando o nível de acesso de cada usuário autenticado.

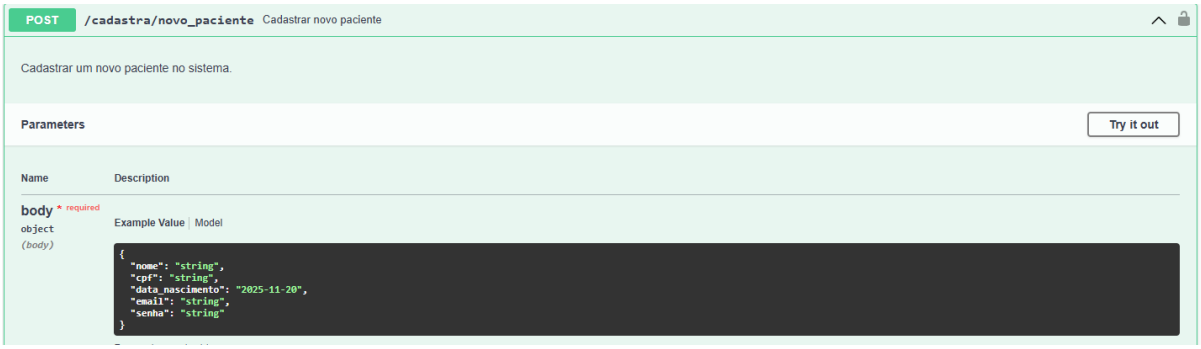
Endpoint	Método	Descrição da Funcionalidade
api/login	POST	Realiza login de médico, secretário ou paciente e retorna um token JWT válido por 60 minutos.
api/editar_senha	PUT	Usuário autenticado atualiza sua senha informando a senha atual e a nova.
api/consulta/ consulta_agenda_medica	GET	Retorna todas as consultas de um médico, com opção de filtrar por data.
api/consulta/{consulta_id}/ atendimento	POST	Médico finaliza uma consulta e registra o diagnóstico do paciente.
api/consulta/{consulta_id}/ prescreve_tratamento	POST	Médico prescreve medicamento, incluindo dosagem e orientações.
api/consulta/{consulta_id}/ solicita_exame	POST	Médico solicita exame para um paciente durante a consulta.
api/consulta/{consulta_id}/ cancelamento_consulta	PUT	Médico cancela uma consulta agendada.
api/consulta/ lista_medico_credenciado	GET	Lista todos os médicos credenciados (id, nome e especialidade) para pacientes autenticados.
api/consulta/agendamento_paciente	GET	Lista todas as consultas do paciente autenticado.
api/consulta/agenda_medica/ {medico_id}	GET	Lista horários disponíveis de um médico específico, com filtro opcional por data.
api/consulta/novo_agendamento	POST	Paciente agenda uma nova consulta informando médico, data e hora.
api/consulta/{consulta_id}/ paciente_cancelamento	PUT	Paciente cancela uma consulta previamente agendada.
api/cadastra/novo_medico	POST	Secretário cadastra um novo médico no sistema.
api/deleta/medico/{medico_id}	DELETE	Secretário exclui um médico pelo ID.
api/cadastra/novo_paciente	POST	Secretário cadastra um novo paciente.
api/bloqueia/paciente/{paciente_id}	POST	Secretário marca um paciente como inativo (bloqueado).
api/consulta/ cadastro_geral_usuarios	GET	Lista todos os usuários (pacientes, médicos, secretários), com filtro por tipo.
api/consulta/{consulta_id}	GET	Secretário busca os detalhes completos de uma consulta pelo ID.
api/consulta/ consultas_geral_marcadas	GET	O secretario lista todas as consultas

Para testes, recomenda-se o uso da interface Swagger UI que deve proporcionar uma experiência mais intuitiva e trazer uma demonstração visual com exemplos de entradas, saídas esperadas e logs de exceções.

(modelo de requisição para autenticação ‘api/login’)



(modelo de requisição para prescrição de tratamento)



(modelo para requisição para cadastro de novos pacientes)

(visualização de usuários com filtro)

6. Conclusão

O sistema **SGHSS** demonstra o funcionamento completo de um **back-end seguro, escalável e documentado**, com autenticação, CRUD de entidades médico, paciente, secretario, filial e rotas dedicadas RESTful.

O projeto foi desenvolvido com foco em **boas práticas, modularização e segurança.**

7. Referências

- Documentação oficial do Flask (https://flask.palletsprojects.com)
- JWT Authentication (https://jwt.io)
- LGPD – Lei nº 13.709/2018
- PEP8 – Python Enhancement Proposals
- ChatGPT
- Biblioteca da instituição Uninter