ATIVIDADE PRÁTICA DA DISCIPLINA
METODOLOGIAS/ MÉTODOS ÁGEIS
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Projeto: Desenvolvimento Back-end
Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS)

PAULO HENRIQUE DA LUZ LIMA – RU: 4376405

BARRA MANSA - RJ
2025

Sumário

1 Introdução

O avanço da tecnologia da informação nas últimas décadas tem impactado significativamente a área da saúde, promovendo a modernização dos processos clínicos, administrativos e operacionais das instituições hospitalares. Diante da crescente complexidade na gestão de hospitais, clínicas e serviços de atendimento domiciliar, torna-se fundamental o desenvolvimento de sistemas informatizados que garantam não apenas eficiência e agilidade, mas também segurança, escalabilidade e conformidade legal.

Neste contexto, o presente projeto propõe o desenvolvimento de uma solução teórica com ênfase em back-end, voltada para o estudo de caso do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS). A solução será direcionada à instituição fictícia VidaPlus, responsável pela administração de hospitais, clínicas de bairro, laboratórios e equipes de home care. A instituição necessita de um sistema robusto que centralize a gestão de pacientes, profissionais de saúde, recursos hospitalares, atendimento por telemedicina e processos administrativos.

A aplicação será desenvolvida com a linguagem Python, utilizando o framework Flask, pela sua leveza, simplicidade e adequação à criação de APIs RESTful. O foco será a implementação das funcionalidades essenciais do sistema, como cadastro e autenticação de usuários, gerenciamento de pacientes e consultas, registro de prontuários, e segurança de dados conforme os princípios da Lei Geral de Proteção de Dados (LGPD).

Os perfis de usuários contemplados pelo sistema incluem três grupos principais: (1) pacientes, que poderão realizar agendamentos, acessar histórico clínico, receber notificações e participar de teleconsultas; (2) profissionais de saúde, com acesso à gestão de agendas, prontuários, prescrições digitais e histórico dos pacientes; e (3) administradores, responsáveis pelo controle dos cadastros, gerenciamento de leitos e geração de relatórios.

A importância do SGHSS se justifica pela sua capacidade de agregar valor aos processos de atendimento, promovendo maior agilidade, redução de erros, segurança da informação e suporte à tomada de decisões em ambientes hospitalares. Além disso, o projeto visa consolidar os conhecimentos adquiridos ao longo do curso, aplicando práticas de engenharia de software, modelagem de dados, desenvolvimento orientado a serviços e testes de validação, resultando em um artefato técnico relevante para o portfólio acadêmico e profissional do autor.

1.1 Objetivo Geral

Desenvolver um sistema de gestão hospitalar e de serviços de saúde (SGHSS), com ênfase no back-end, utilizando Python e Flask, que permita o gerenciamento de pacientes, profissionais, consultas, prontuários e prescrições, garantindo segurança, eficiência e conformidade com a Lei Geral de Proteção de Dados (LGPD).

1.2 Objetivos Específicos

Implementar cadastro e autenticação de usuários com controle de perfis (paciente, profissional, administrador).

- Disponibilizar funcionalidades para agendamento e gerenciamento de consultas.
- Registrar e consultar informações de prontuários e receitas médicas.
- Garantir a segurança e a integridade das informações armazenadas.
- Criar relatórios para apoio à tomada de decisão.

1.3 Escopo

O sistema abrangerá as funcionalidades essenciais para o gerenciamento hospitalar, incluindo módulos de autenticação, cadastro de usuários, gestão de pacientes e consultas, registro de prontuários e emissão de receitas. Não serão implementadas funcionalidades avançadas como integração com sistemas externos, faturamento hospitalar ou módulos de estoque de medicamentos.

2 Análise e Requisitos

Os requisitos apresentados foram definidos com base no estudo de caso do SGHSS fornecido pela disciplina de Projeto Multidisciplinar (2025), adaptados para a implementação com foco em back-end.

Nesta seção são descritos os requisitos do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS), organizados em requisitos funcionais (RF), que definem as funcionalidades que o sistema deve executar, e requisitos não funcionais (RNF), que determinam características de desempenho, segurança e qualidade.

A tabela a seguir apresenta os principais requisitos identificados para a versão teórica deste sistema:

2.1 Requisitos Funcionais

A Tabela 1 consolida os Requisitos Funcionais (RF001–RF008) do sistema SGHSS, ou seja, as funções observáveis pela(o) usuária(o) e indispensáveis para a operação do back-end. Os itens de prioridade Alta compõem o MVP e foram implementados na API Flask; os de prioridade Média/Baixa são evoluções planejadas.

- **RF001 – Cadastro de pacientes (Alta):** permite registrar dados pessoais e clínicos básicos, assegurando unicidade por CPF e integridade mínima dos campos obrigatórios.
- **RF002 – Autenticação e controle de acesso (Alta):** provê login/logout com geração e validação de token (JWT), restringindo o uso das rotas protegidas.
- **RF003 – Agendamento de consultas (Alta):** cria e lista consultas presenciais ou por telemedicina, com verificação de conflitos de horário por profissional/paciente.
- **RF004 – Cancelamento de consultas (Média):** possibilita cancelar agendamentos, preservando o histórico para auditoria.
- **RF005 – Prontuários médicos (Alta):** registra e consulta evoluções clínicas associadas ao paciente e à consulta.
- **RF006 – Emissão de receitas (Média):** gera receitas vinculadas ao atendimento, com itens e posologia.
- **RF007 – Agenda de profissionais (Média):** gerencia janelas de atendimento (disponibilidades), base para o agendamento.
- **RF008 – Relatórios administrativos (Baixa):** produz listagens/sínteses operacionais (ex.: consultas por período), priorizadas para fase posterior.

Esses requisitos norteiam as **rotas da API** (auth, pacientes, profissionais e consultas), garantindo cobertura do ciclo essencial: **autenticar → cadastrar paciente → agendar/registrar atendimento → gerar registros clínicos e administrativos**.

Tabela 1 – Requisitos Funcionais

ID	Descrição	Prioridade
RF001	Permitir o cadastro de pacientes com dados pessoais e clínicos	Alta
RF002	Realizar autenticação de usuários (login/logout) com controle de acesso	Alta
RF003	Permitir o agendamento de consultas presenciais e por telemedicina	Alta
RF004	Permitir o cancelamento de consultas pelo paciente	Média
RF005	Permitir o registro e consulta de prontuários médicos	Alta
RF006	Permitir a emissão de receitas digitais pelos profissionais de saúde	Média
RF007	Gerenciar a agenda dos profissionais de saúde	Média
RF008	Permitir a geração de relatórios administrativos	Baixa

2.2 Requisitos Não Funcionais

A Tabela 2 apresenta os Requisitos Não Funcionais (RNF001–RNF006), que especificam qualidades e restrições do sistema, independentes da funcionalidade em si, mas determinantes para usabilidade, desempenho, disponibilidade e segurança.

- **RNF001 – Acessibilidade e responsividade:** a interface cliente (quando existente) deve seguir boas práticas de acessibilidade (W3C/WCAG) e se adaptar a diferentes dispositivos.
- **RNF002 – Segurança e LGPD:** autenticação forte com proteção de dados sensíveis; uso de criptografia para segredos e políticas de acesso mínimo necessário.
- **RNF003 – Auditoria (logs):** registro de ações relevantes (quem fez, o quê, quando e onde), permitindo rastreabilidade e investigações.
- **RNF004 – Desempenho:** tempo de resposta médio da API **≤ 2 s** para operações usuais em ambiente de referência.
- **RNF005 – Disponibilidade:** serviço disponível **≥ 99,5%** com procedimentos de backup e restauração documentados.
- **RNF006 – Escalabilidade:** possibilidade de ampliar recursos para atender múltiplas unidades/instalações da organização, mantendo estabilidade e performance.

Em conjunto, os RNF asseguram que a solução, além de correta funcionalmente, seja confiável, segura e sustentável em produção, alinhada às exigências regulatórias e às metas operacionais da instituição.

Tabela 2 – Requisitos Não Funcionais

ID	Descrição	Prioridade
RNF001	Interface deve ser acessível e responsiva (padrões W3C/WCAG)	Média
RNF002	O sistema deve possuir autenticação segura e proteger dados sensíveis (conforme LGPD)	Alta
RNF003	O sistema deve registrar logs de acesso e ações para fins de auditoria	Alta
RNF004	O tempo de resposta das consultas à API deve ser inferior a 2 segundos	Alta
RNF005	O sistema deve estar disponível no mínimo 99,5% do tempo, com backups regulares	Alta
RNF006	O sistema deve ser escalável para suportar múltiplas unidades da VidaPlus	Média

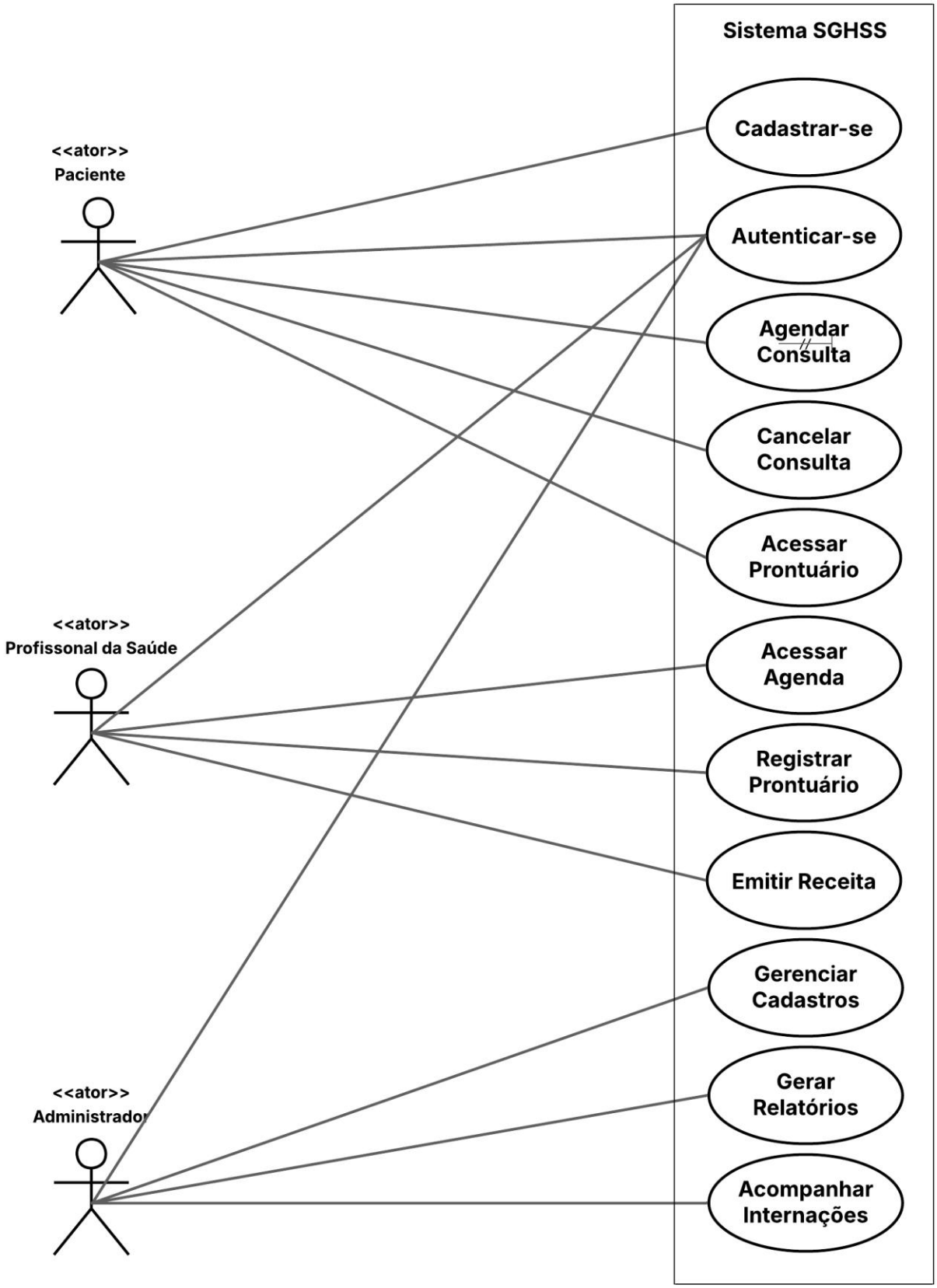

Figura 1 – Diagrama de Casos de Uso (UML)

O **Diagrama de Casos de Uso** da Figura 1 ilustra as interações entre os atores externos e o Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS). Esse diagrama segue a notação UML (Unified Modeling Language) e tem como objetivo representar de forma visual os requisitos funcionais levantados na fase de análise do projeto.

No modelo apresentado, os três atores principais são:

- **Paciente**: interage com o sistema para realizar o cadastro, autenticar-se, agendar e cancelar consultas, acessar prontuário e participar de funcionalidades relacionadas ao seu atendimento.
- **Profissional de Saúde**: utiliza o sistema para autenticar-se, acessar sua agenda de consultas, registrar prontuários e emitir receitas digitais.
- **Administrador**: é responsável por autenticar-se, gerenciar cadastros, gerar relatórios e acompanhar internações.

Os casos de uso, representados pelas elipses dentro do retângulo "Sistema SGHSS", correspondem às funcionalidades que serão implementadas no sistema. As linhas que conectam atores e casos de uso representam a participação ou interação desses usuários externos com cada funcionalidade. Essa modelagem facilita a compreensão das responsabilidades do sistema e orienta o desenvolvimento das próximas etapas, como o diagrama de classes e a implementação dos módulos.

3 Modelagem e Arquitetura

A modelagem e arquitetura do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS) foram definidas com base nos requisitos funcionais e não funcionais previamente levantados, visando garantir clareza estrutural, integridade dos dados e aderência às boas práticas de engenharia de software. Nesta etapa, são apresentados o diagrama de classes, a visão geral da arquitetura escolhida, as tecnologias de persistência e os principais endpoints da API.

3.1 Diagrama de Classes

O Diagrama Entidade-Relacionamento (DER) apresentado na Figura 2 foi elaborado com base no levantamento de requisitos e no Diagrama de Classes do sistema SGHSS. Ele representa a estrutura lógica do banco de dados, evidenciando as entidades, seus atributos, chaves primárias (PK), chaves estrangeiras (FK) e os relacionamentos com suas respectivas cardinalidades. O

objetivo é organizar e padronizar as informações de forma a garantir a integridade e consistência dos dados, servindo como base para a implementação física do banco.

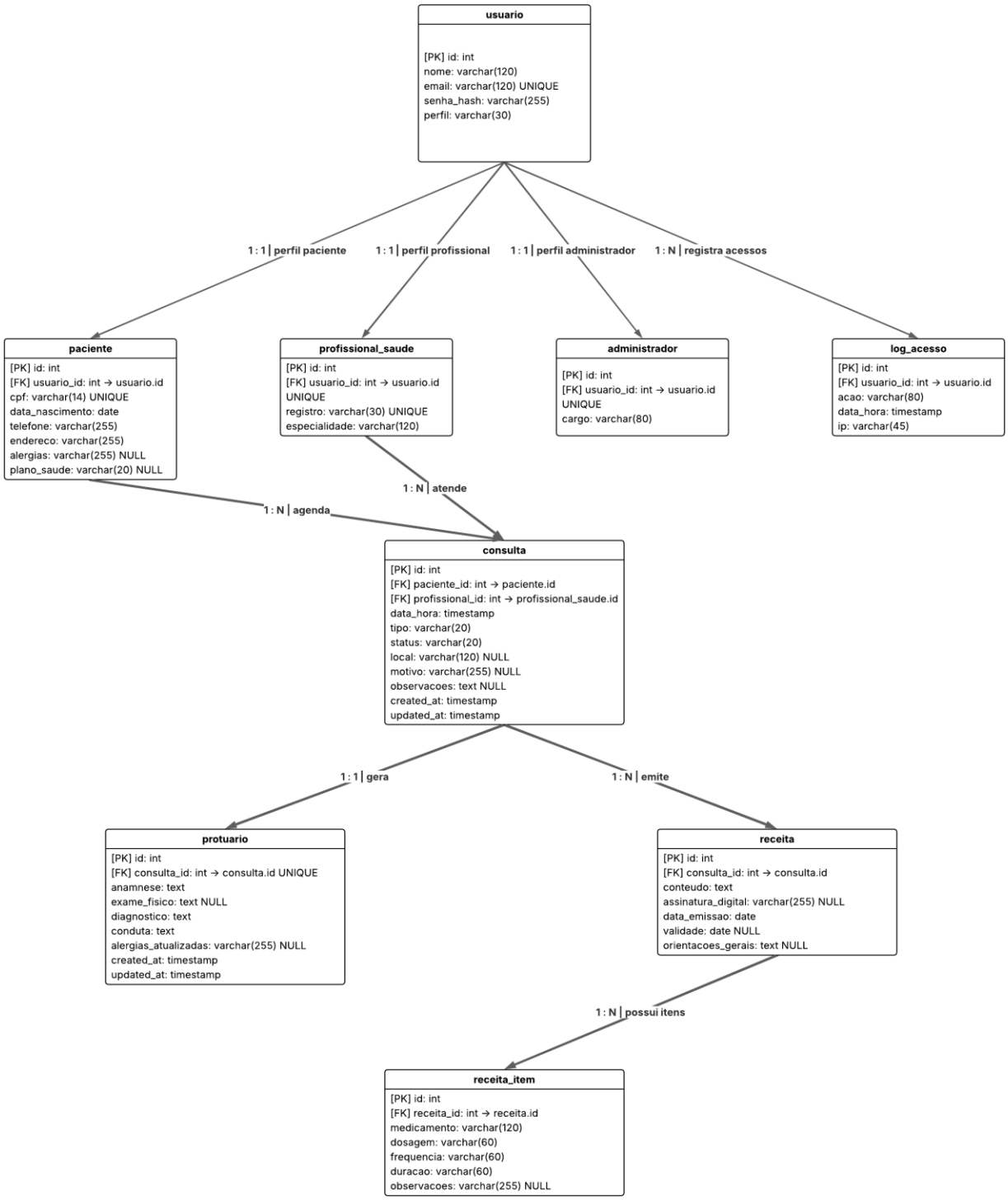

Figura 2 – Diagrama Entidade-Relacionamento (DER)

No DER, cada entidade corresponde a um conjunto de informações relevantes para o funcionamento do sistema, enquanto os relacionamentos indicam como essas entidades interagem entre si. A seguir, descrevem-se cada entidade e seus atributos:

Usuario

Armazena as informações de identificação e autenticação de todos os usuários do sistema, independentemente do perfil (paciente, profissional de saúde ou administrador).

- Atributos:
  - `id` (PK, int): Identificador único do usuário.
  - `nome` (varchar(120)): Nome completo.
  - `email` (varchar(120), UNIQUE): Endereço de e-mail único para login.
  - `senha_hash` (varchar(255)): Senha criptografada.
  - `perfil` (varchar(30)): Tipo de perfil (paciente, profissional, administrador).
- Relacionamentos:
  - 1:1 com paciente, profissional_saude e administrador.
  - 1:N com log_acesso.

Paciente

Representa os dados pessoais e clínicos básicos dos pacientes cadastrados.

- Atributos:
  - `id` (PK, int)
  - `usuario_id` (FK): Referência para usuario.id.
  - `cpf` (varchar(14), UNIQUE): Documento único.
  - `data_nascimento` (date): Data de nascimento.
  - `telefone` (varchar(20)): Telefone de contato.
  - `endereco` (varchar(255)): Endereço residencial.
  - `alergias` (varchar(255), NULL): Informações sobre alergias.
  - `plano_saude` (varchar(20), NULL): Nome do plano de saúde.
- Relacionamentos:
  - 1:N com consulta.

ProfissionalSaude

Contém os dados dos profissionais de saúde.

- Atributos:
  - `id` (PK, int)
  - `usuario_id` (FK, UNIQUE): Referência para usuario.id.
  - `registro` (varchar(30), UNIQUE): Registro profissional (CRM, CRO, etc.).
  - `especialidade` (varchar(120)): Área de atuação médica ou de saúde.
- Relacionamentos:
  - 1:N com consulta.

Administrador

Guarda informações dos usuários administradores do sistema.

- Atributos:
  - `id` (PK, int)
  - `usuario_id` (FK, UNIQUE)
  - `cargo` (varchar(80)): Função ocupada.
- Relacionamentos:
  - Sem relacionamento direto com tabelas de operações, mas com função de gerenciamento.

LogAcesso

Registra todas as ações realizadas no sistema para auditoria.

- Atributos:
  - `id` (PK, int)
  - `usuario_id` (FK): Usuário que realizou a ação.
  - `acao` (varchar(80)): Descrição da ação.
  - `data_hora` (timestamp): Momento do registro.
  - `ip` (varchar(45)): Endereço IP de origem.
- Relacionamentos:
  - N:1 com usuario.

Consulta

Registra as consultas realizadas.

- Atributos:
  - `id` (PK, int)
  - `paciente_id` (FK): Paciente atendido.
  - `profissional_id` (FK): Profissional responsável.
  - `data_hora` (timestamp): Data e hora da consulta.
  - `tipo` (varchar(20)): Tipo da consulta (presencial, online).
  - `status` (varchar(20)): Situação da consulta (agendada, concluída).
  - `local` (varchar(120), NULL)
  - `motivo` (varchar(255), NULL): Motivo do atendimento.
  - `observacoes` (text, NULL): Observações adicionais.
  - `created_at` (timestamp)
  - `updated_at` (timestamp)
- Relacionamentos:
  - 1:1 com prontuario.
  - 1:N com receita.

Prontuario

Armazena o histórico médico de cada consulta.

- Atributos:
  - `id` (PK, int)
  - `consulta_id` (FK, UNIQUE): Consulta associada.
  - `anamnese` (text)
  - `exame_fisico` (text, NULL)
  - `diagnostico` (text)
  - `conduta` (text)
  - `alergias_atualizadas` (varchar(255), NULL)
  - `created_at` (timestamp)
  - `updated_at` (timestamp)
- Relacionamentos:
  - N:1 com consulta.

Receita

Registra as prescrições geradas nas consultas.

- Atributos:
  - `id` (PK, int)
  - `consulta_id` (FK)
  - `conteudo` (text): Texto da prescrição.
  - `assinatura_digital` (varchar(255), NULL): Assinatura eletrônica do profissional.
  - `data_emissao` (date)
  - `validade` (date, NULL)
  - `orientacoes_gerais` (text, NULL)
- Relacionamentos:
  - 1:N com receita_item.

ReceitaItem

Detalha os itens de cada receita.

- Atributos:
  - `id` (PK, int)
  - `receita_id` (FK): Receita à qual pertence.
  - `medicamento` (varchar(120))
  - `dosagem` (varchar(60))
  - `frequencia` (varchar(60))
  - `duracao` (varchar(60))
  - `observacoes` (varchar(255), NULL)
  - Relacionamentos:
  - N:1 com receita.

3.2 Arquitetura do Sistema

O sistema seguirá uma arquitetura em camadas (Model-View-Controller adaptado para API), onde:

- Camada de Modelo (Model): responsável pela definição das entidades e comunicação com o banco de dados.
- Camada de Controle (Controller): responsável por processar as requisições e aplicar as regras de negócio.
- Camada de Rota (Routes): responsável por expor os endpoints da API REST para comunicação com o front-end ou sistemas externos.

3.3 Tecnologias e Ferramentas

- **Nenhuma entrada de índice de ilustrações foi encontrada.Nenhuma entrada de índice de ilustrações foi encontrada.**Linguagem: Python 3.13
- Framework: Flask (microframework para criação de APIs RESTful)
- Banco de Dados: MySQL
- Ferramentas de Modelagem: Lucidchart (diagramas UML), MySQL Workbench (modelagem física)

Também será incluído o diagrama de classes com explicação, para mostrar a relação entre entidades no nível de software.

4 Implementação

4.1 Segurança e LGPD (medidas práticas)

As senhas são armazenadas com hash bcrypt; o acesso às rotas privadas utiliza JWT com expiração configurável, e os segredos ficam em variáveis de ambiente (.env). Dados sensíveis não são retornados em respostas. Para produção, recomenda-se HTTPS (terminação TLS), rotação de chaves JWT e política de backup/restauração do banco. Essas medidas atendem aos RNFs de segurança/privacidade declarados.

4.2 Auditoria e rastreabilidade (LogAcesso)

O sistema registra eventos em log_acesso para ações críticas (login, criação/alteração de pacientes, criação/cancelamento de consultas, criação de prontuário, emissão de receita/itens). Campos: usuario_id, acao, recurso, recurso_id, ip, data_hora, status_http, duracao_ms. Isso viabiliza rastreabilidade e conformidade.

Tabela 3 – Esquema lógico de LogAcesso (síntese)

Campo	**Tipo**	**Exemplo**	**Observação**
id	BIGINT PK	125	autoincremento
usuario_id	BIGINT FK	1	usuário autenticado
acao	VARCHAR	CRIAR_CONSULTA	taxonomia curta
recurso	VARCHAR	consulta	nome lógico
recurso_id	BIGINT	17	id afetado
ip	VARCHAR	127.0.0.1	origem
data_hora	DATETIME	2025-08-22 14:33	timestamp
status_http	INT	201	retorno
duracao_ms	INT	42	performance

4.3 Arquitetura e Tecnologias

O sistema será desenvolvido como uma aplicação monolítica modularizada, separando camadas de controle, serviço e persistência, conforme o padrão MVC (Model-View-Controller). A camada de controle será responsável por gerenciar as requisições HTTP e direcionar as operações para a camada de serviço, onde residem as regras de negócio. A camada de persistência ficará encarregada do acesso e manipulação dos dados no banco de dados relacional.

Implementação do back-end. A API será desenvolvida em Python com Flask, dada a sua simplicidade e flexibilidade para construção de serviços RESTful. Para persistência dos dados utilizaremos um banco relacional acessado via SQLAlchemy (ORM), o que abstrai a camada de acesso e facilita manutenção e portabilidade do código entre diferentes SGBDs. Na implementação deste trabalho o ambiente utilizou MySQL 8, mantendo aderência a transações (ACID), índices e chaves estrangeiras requeridas pelo escopo do sistema.

4.4 Principais Endpoints da API

A API RESTful será estruturada para atender às funcionalidades essenciais do sistema, incluindo autenticação de usuários, gestão de pacientes, consultas e prontuários. A Tabela 4 apresenta os principais endpoints planejados.

Tabela 4 - Principais endpoints

Método	Endpoint	Descrição	Resposta Esperada
POST	**/auth/signup**	Cadastra usuário (massa/admin)	{"id": 1, "nome": "João", "email": "joao@...", "perfil":"ATEND"}
POST	**/auth/login**	Autentica e retorna **JWT**	{"token": "eyJ...", "perfil":"ADMIN"}
GET	**/ping**	Verificação de disponibilidade	{"status":"ok"}
GET / POST	**/pacientes**	Lista/cria pacientes	GET → [{...}] • POST → {"id": 1, "nome":"Maria"}
GET / POST	**/profissionais**	Lista/cria profissionais	GET → [{...}] • POST → {"id": 1, "nome":"Dra. Maria"}
GET / POST	**/consultas**	Lista/agenda consultas	POST → {"id": 10, "status":"AGENDADA"}
PATCH	**/consultas/{id}/cancelar**	Cancela consulta	{"id": 10, "status":"CANCELADA"}
GET	**/prontuarios/{id}**	Consulta prontuário (RF005)	{"id": 5, "diagnostico": "..."}

POST	**/prontuarios**	Cria prontuário (RF005)	{"id": 5, "consulta_id": 1, ...}
GET / POST	**/receitas** / **/receitas/{id}/itens**	Emite receita e itens (RF006)	{"id": 3, "consulta_id": 1, ...} / {"id": 7, "receita_id": 3, ...}
GET	**/profissionais/{id}/agenda?inicio=YYYY-MM-DD&fim=YYYY-MM-DD**	Agenda do profissional (RF007)	[{"consulta_id":17,"data_hora":"...."}]
GET	**/relatorios/consultas?inicio=YYYY-MM-DD&fim=YYYY-MM-DD**	Resumo por período (RF008)	{"total":42,"por_status":{"AGENDADA":30,...},"por_tipo":{"PRESENCIAL":28,"ONLINE":14}}

4.4 Visão geral da implementação

A aplicação foi implementada em Python com o framework Flask, expondo uma API RESTful para o gerenciamento de usuários, pacientes, profissionais e consultas. O MySQL 8 foi utilizado como banco de dados relacional, com acesso via SQLAlchemy e o driver mysql-connector. A autenticação é feita com JWT (JSON Web Tokens); senhas são armazenadas com hash bcrypt (biblioteca passlib[bcrypt]).

Todas as variáveis sensíveis (URL do banco, chave JWT e porta) são carregadas de um arquivo .env, mantendo as boas práticas de segurança e a aderência à LGPD.

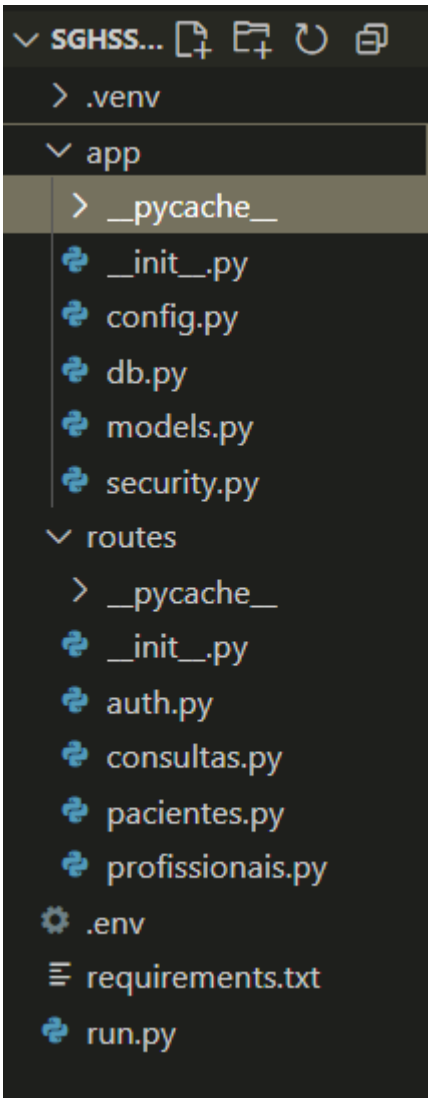

Figura 3 – Estrutura do projeto (VS Code)

4.5 Ambiente e ferramentas

- Sistema operacional: Windows 11
- IDE: Visual Studio Code (terminal integrado)
- Terminal de testes: Windows PowerShell / VS Code
- Linguagem: Python 3.13
- Framework web: Flask 2.3
- Banco de dados: MySQL 8
- ORM: SQLAlchemy 2.0
- Autenticação: PyJWT (JWT) + passlib[bcrypt]
- Carregamento de variáveis: python-dotenv

4.6 Organização do código (arquitetura)

O projeto segue o padrão Application Factory e separa regras por *blueprints*:

```
SGHSS_API/
├─ app/
│  ├─ __init__.py           # create_app(), registro de blueprints e /ping
│  ├─ config.py             # leitura de .env (DB_URL, JWT_SECRET, PORT)
│  ├─ db.py                 # engine/session SQLAlchemy
│  ├─ models.py             # modelos e relacionamentos
│  ├─ security.py           # utilitários de JWT e hashing
│  └─ routes/
│     ├─ __init__.py        # registro dos blueprints
│     ├─ auth.py            # /auth/signup, /auth/login
│     ├─ pacientes.py       # /pacientes (CRUD)
│     ├─ profissionais.py   # /profissionais (CRUD)
│     └─ consultas.py       # /consultas (CRUD, conflito de horário)
├─ run.py                   # sobe o servidor local
├─ .env                     # variáveis de ambiente (não versionar)
└─ requirements.txt         # dependências do projeto
```

Figura 4 – Arquitetura em camadas

4.7 Modelo de dados (resumo)

- **usuarios**: id, nome, email (único), senha_hash, perfil, criado_em
- **pacientes**: id, nome, cpf (único), telefone, endereco, criado_em
- **profissionais**: id, nome, registro (único), especialidade, criado_em
- **consultas**: id, paciente_id (FK), profissional_id (FK), data_hora, tipo, status, local, motivo, criado_em

Regras principais:

- cpf e registro possuem **unicidade**.
- Em **consultas**, há validação de **conflito de agenda** por profissional_id + data_hora.

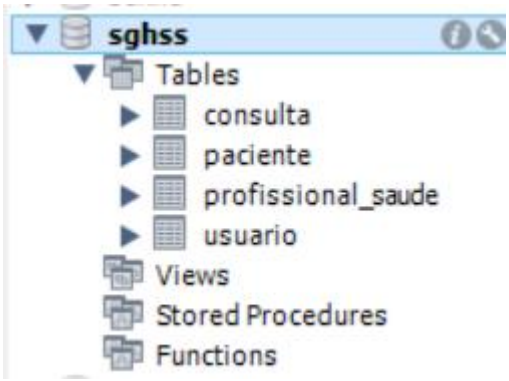

Figura 5 – Tabelas no MySQL (Workbench).

4.8 Configuração do ambiente

As credenciais não são codificadas no código-fonte. O arquivo **.env** contém:

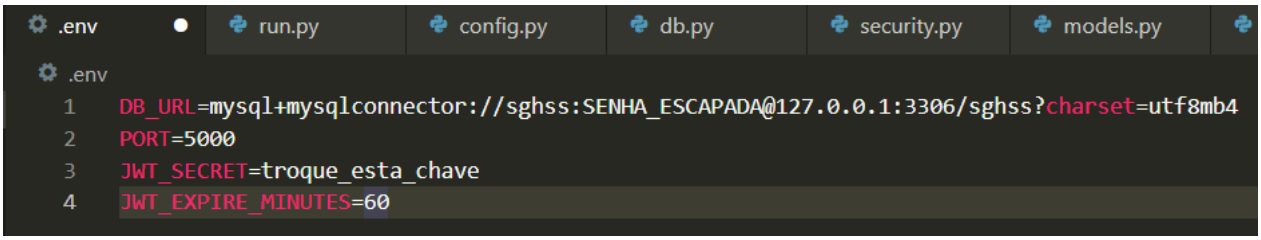

Figura 6 – .env no VS Code

5 Plano de Testes e Resultados

5.1 Inicialização do servidor

No terminal (VS Code/PowerShell), com a venv ativa:

```
(.venv) PS C:\SGHSS_API> cd C:\SGHSS_API
>> . .\.venv\Scripts\Activate.ps1
>> python run.py
>>
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
```

Figura 7 – Servidor Flask em execução

Validação do *health-check*:

```
mysql+mysqlconnector://sghss:Sghss2025%40@127.0.0.1:3306/sghss?charset=utf8mb4
(.venv) PS C:\SGHSS_API> python -c "import mysql.connector as mc; mc.connect(user='sghss', password='Sgh
ss2025@', host='127.0.0.1', database='sghss'); print('Conectou OK')"
>>
Conectou OK
(.venv) PS C:\SGHSS_API> Invoke-RestMethod "http://127.0.0.1:5000/ping" -Method Get
>>

status
------
ok
```

Figura 8 – Resposta do /ping (status ok)

5.2 Testes funcionais da API (PowerShell)

Observação prática: para caracteres acentuados, os bodies foram enviados como **UTF-8**:
-Body ([System.Text.Encoding]::UTF8.GetBytes($json)) -ContentType "application/json; charset=utf-8"

5.3 Autenticação (signup/login)

Criação do administrador e obtenção do token:

```
PS C:\WINDOWS\system32> $signup = @{ nome="Admin"; email="admin@vida.plus"; senha="Admin#2025"; perfil="ADMIN" } | ConvertTo-Json
>> Invoke-RestMethod "http://127.0.0.1:5000/auth/signup" -Method Post -Body $signup -ContentType "application/json"

email          id nome  perfil
-----          -- ----  ------
admin@vida.plus  1 Admin ADMIN
```

Figura 9 – Signup (201 Created)

```
PS C:\WINDOWS\system32> $login = @{ email="admin@vida.plus"; senha="Admin#2025" } | ConvertTo-Json
>> $resp  = Invoke-RestMethod "http://127.0.0.1:5000/auth/login" -Method Post -Body $login -ContentType "application/json; charset=utf-8"
>> $token = $resp.token
>> $H     = @{ Authorization = "Bearer $token" }
>>
```

Figura 10 – Login (200 OK) e token JWT

5.4 Pacientes (CRUD)

```
PS C:\WINDOWS\system32> $pac = @{
>>   nome     = "João"
>>   cpf      = "12345678900"
>>   telefone = "(24)99999-0000"
>>   endereco = "Rua A, 123"
>> } | ConvertTo-Json
>>
PS C:\WINDOWS\system32> Invoke-RestMethod "http://127.0.0.1:5000/pacientes" `
>>   -Method Post `
>>   -Headers $H `
>>   -Body ([System.Text.Encoding]::UTF8.GetBytes($pac)) `
>>   -ContentType "application/json; charset=utf-8"
>>

cpf         id nome
---         -- ----
12345678900  1 João
```

Figura 11 – POST /pacientes (201)

```
PS C:\WINDOWS\system32> Invoke-RestMethod "http://127.0.0.1:5000/pacientes" -Method Get -Headers $H
>>


cpf      : 12345678900
endereco : Rua A, 123
id       : 1
nome     : João
telefone : (24)99999-0000
```

Figura 12 – GET /pacientes (200)

5.5 Profissionais (CRUD)

```
PS C:\WINDOWS\system32> chcp 65001 | Out-Null
>>
PS C:\WINDOWS\system32> # criar
>> $prof = @{
>>   nome="Dra. Maria"
>>   registro="CRM0001"
>>   especialidade="Clínica Geral"
>> } | ConvertTo-Json
>>
>> Invoke-RestMethod "http://127.0.0.1:5000/profissionais" `
>>   -Method Post `
>>   -Headers $H `
>>   -Body ([System.Text.Encoding]::UTF8.GetBytes($prof)) `
>>   -ContentType "application/json; charset=utf-8"
>>
```

Figura 13 – POST /profissionais (201)

```
>> # listar
>> Invoke-RestMethod "http://127.0.0.1:5000/profissionais" -Method Get -Headers $H
>>

id nome       registro
-- ----       --------
 1 Dra. Maria CRM0001
 1 Dra. Maria CRM0001
```

Figura 14 – GET /profissionais (200)

5.6 Consultas (validação de agenda)

```
PS C:\WINDOWS\system32> $consulta = @{
>>   paciente_id=1
>>   profissional_id=1
>>   data_hora="2025-09-01 14:30:00"
>>   tipo="PRESENCIAL"
>>   status="AGENDADA"
>>   local="Sala 3"
>>   motivo="Rotina"
>> } | ConvertTo-Json
>>
>> Invoke-RestMethod "http://127.0.0.1:5000/consultas" `
>>   -Method Post `
>>   -Headers $H `
>>   -Body ([System.Text.Encoding]::UTF8.GetBytes($consulta))
>>   -ContentType "application/json; charset=utf-8"
>>
```

Figura 15 – POST /consultas (201)

```
>> # listar
>> Invoke-RestMethod "http://127.0.0.1:5000/consultas" -Method Get -Headers $H
>>

data_hora           id status
---------           -- ------
2025-09-01 14:30:00  1 AGENDADA
```

Figura 16 – GET /consultas (200)

6 Conclusão

Este trabalho apresentou a concepção e a implementação do núcleo de uma API REST para o Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS), destinada à instituição VidaPlus. A solução foi construída em Python/Flask, com organização em *Application Factory* e blueprints por domínio (autenticação, pacientes, profissionais e consultas), persistência em MySQL 8 via SQLAlchemy, e configuração externa por meio de variáveis no arquivo .env. Do ponto de vista de segurança, foram adotadas senhas com hash (bcrypt), autenticação JWT com expiração e logs de auditoria (LogAcesso) para ações críticas, alinhando a aplicação aos requisitos não funcionais de segurança e rastreabilidade previstos. A API expõe endpoints coerentes com os requisitos funcionais priorizados e foi validada com testes manuais no PowerShell, incluindo *health-check* (/ping), cadastro/autenticação e operações de CRUD para pacientes, profissionais e agendamento de consultas.

Os resultados obtidos demonstram que o objetivo geral foi atendido: entregar um back-end funcional que centraliza dados e regras essenciais do processo assistencial (cadastro, autenticação, agenda), com arquitetura em camadas, padronização de endpoints, e documentação mínima para operação e testes. Em particular, destacam-se: (i) a detecção de conflito de agenda por profissional/horário, (ii) o registro de auditoria com campos de usuário, ação, recurso e duração, e (iii) a separação de responsabilidades entre modelos, regras e rotas, o que favorece manutenção e evolução. Além disso, a estratégia de testes garantiu evidências objetivas (status HTTP e payloads JSON) para cada caso verificado, o que dá transparência à validação dos requisitos contemplados.

Do ponto de vista acadêmico, o projeto consolidou conhecimentos de engenharia de software, modelagem de dados e desenvolvimento orientado a serviços, resultando em um artefato tecnicamente consistente e pronto para evolução. Em termos práticos, a base entregue permite integrar futuramente um frontend (web ou mobile), ampliar o escopo clínico (prontuários e receitas com fluxos completos), e endereçar demandas operacionais como observabilidade, disponibilidade e escalabilidade para ambientes de produção.

Limitações e próximos passos (síntese). Como todo MVP, a versão atual priorizou o núcleo transacional; não abrange faturamento/estoque, integração com prontuário eletrônico nacional, assinatura digital avançada, monitoramento contínuo, nem *deploy* automatizado. Recomenda-se, como sequência natural: (1) cobertura de testes automatizados (unitários e de API), (2) migrações de banco com Alembic, (3) empacotamento Docker e *pipeline* CI/CD, (4) métricas e *logs* centralizados (p.ex., Prometheus/Grafana/ELK), (5) endurecimento de segurança (rate-limit, CORS, rotação de chaves JWT, TLS), e (6) implementação completa dos módulos de prontuário e receitas com geração de documentos e trilhas de auditoria ampliadas.

Em síntese, o trabalho cumpriu o que se propôs: entregar um back-end coerente, seguro e testado para o SGHSS, servindo como base sólida para continuidade acadêmica e profissional — seja no aprofundamento clínico do domínio, seja na preparação da solução para um cenário de produção em larga escala.

7 Referências

7.1 Normas e legislação

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). *NBR 6023:2018 — Informação e documentação — Referências — Elaboração*. Rio de Janeiro: ABNT, 2018.

BRASIL. *Lei nº 13.709, de 14 de agosto de 2018*. Lei Geral de Proteção de Dados Pessoais (LGPD). Diário Oficial da União, Brasília, DF, 15 ago. 2018. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: 27 ago. 2025.

(opcional) ABNT. *NBR 14724:2011 — Informação e documentação — Trabalhos acadêmicos — Apresentação*. Rio de Janeiro: ABNT, 2011.

7.2 Livros e obras de apoio

PRESSMAN, Roger S.; MAXIM, Bruce R. *Engenharia de Software: uma abordagem profissional*. 8. ed. Porto Alegre: AMGH, 2016.
SOMMERVILLE, Ian. *Engenharia de Software*. 10. ed. São Paulo: Pearson, 2011.

FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Tese (Doutorado) — University of California, Irvine. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Acesso em: 27 ago. 2025.

7.3 Documentação oficial e especificações técnicas

PYTHON SOFTWARE FOUNDATION. *Python 3.13 Documentation*. Disponível em: https://docs.python.org/3/. Acesso em: 27 ago. 2025.

PALLETS. *Flask Documentation*. Disponível em: https://flask.palletsprojects.com/. Acesso em: 27 ago. 2025.

SQLALCHEMY. *SQLAlchemy 2.0 Documentation*. Disponível em: https://docs.sqlalchemy.org/en/20/. Acesso em: 27 ago. 2025.

ORACLE. *MySQL 8.0 Reference Manual*. Disponível em: https://dev.mysql.com/doc/refman/8.0/en/. Acesso em: 27 ago. 2025.

INTERNET ENGINEERING TASK FORCE (IETF). *RFC 7519 — JSON Web Token (JWT)*. 2015. Disponível em: https://datatracker.ietf.org/doc/html/rfc7519. Acesso em: 27 ago. 2025.

PYJWT. *PyJWT Documentation*. Disponível em: https://pyjwt.readthedocs.io/en/stable/. Acesso em: 27 ago. 2025.

PYPI. *bcrypt — Python library*. Disponível em: https://pypi.org/project/bcrypt/. Acesso em: 27 ago. 2025.

MICROSOFT. *Invoke-RestMethod (PowerShell)*. Documentação oficial. Disponível em: https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/invoke-restmethod. Acesso em: 27 ago. 2025.

MOZILLA DEVELOPER NETWORK (MDN). *HTTP response status codes*. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Acesso em: 27 ago. 2025.

7.4 Observações (rápidas)

- Se o seu TCC exigir **ordem alfabética**, mantenha a lista ordenada por sobrenome/entidade (ABNT permite ordenar por ordem de citação, mas muitas bancas preferem alfabética).
- Atualize “Acesso em” se fizer a revisão final outro dia.
- Caso tenha usado outras bibliotecas (por exemplo, **python-dotenv**, **Flask-JWT-Extended**, **Alembic**), inclua as páginas oficiais na mesma formatação.