Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS)

Projeto Multidisciplinar – Ênfase em Back-end

Universidade Internacional UNINTER
Curso: Análise e Desenvolvimento de Sistemas
Disciplina: Projeto Multidisciplinar
Nome do Aluno: *[preencher]*
RU: *[preencher]*
Polo: *[preencher]*
Semestre: 2025.A1
Professor: Prof. Winston Sen Lun Fung, Me.

Sumário

1. Introdução

1.1 Contextualização

O Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS) é uma iniciativa da instituição **VidaPlus**, uma rede de saúde que administra hospitais, clínicas de bairro, laboratórios e equipes de home care. Com o crescimento da demanda por serviços de saúde e as exigências de conformidade com a Lei Geral de Proteção de Dados (LGPD), tornou-se imprescindível modernizar a gestão de informações clínicas e administrativas.
Um sistema centralizado permite reduzir a fragmentação de dados entre unidades, melhorar o agendamento de consultas, integrar prontuários eletrônicos, viabilizar funcionalidades de telemedicina e garantir que as operações estejam em conformidade com as normas de segurança e privacidade.【36963841174969†screenshot】

1.2 Objetivos

Objetivo geral:
Desenvolver a camada de **back-end** de uma API RESTful para o SGHSS, contemplando cadastro de pacientes e profissionais, gestão de consultas, prontuários, prescrições e notificações, com foco em segurança, auditoria e conformidade com a LGPD.

Objetivos específicos:

1. Implementar API RESTful robusta utilizando Node.js e Express, com persistência em banco de dados relacional (PostgreSQL) via ORM (Prisma);

2. Definir uma modelagem de dados consistente e normalizada para entidades como Usuário, Paciente, Profissional, Consulta, Prontuário e Prescrição;

3. Implementar autenticação e autorização baseadas em JWT e controle de acessos por perfil (RBAC);

4. Registrar logs de auditoria e utilizar criptografia para dados sensíveis, atendendo às exigências da LGPD;

5. Fornecer documentação completa dos endpoints, incluindo método, rota, parâmetros de entrada, exemplos de requisições e respostas, e códigos de erro;

6. Elaborar plano de testes abrangente, contemplando casos positivos e negativos, e evidenciar sua execução via ferramentas como Postman ou Jest;

7. Preparar material de apoio (diagramas, scripts SQL, exemplos de uso) nos anexos, garantindo que o avaliador possa reproduzir e avaliar o sistema.

1.3 Justificativa e relevância

O SGHSS tem papel estratégico na organização de fluxos assistenciais e administrativos da VidaPlus. Um back-end bem estruturado impacta diretamente na **qualidade do atendimento** (acesso rápido ao histórico do paciente), na **eficiência operacional** (redução de falhas e retrabalho), na **segurança da informação** (proteção de dados sensíveis) e na **conformidade legal** (LGPD e normas do Conselho Federal de Medicina). A centralização e padronização dos processos também prepara a instituição para integrar soluções de telemedicina e interoperabilidade com outros sistemas de saúde. 【36963841174969†screenshot】

1.4 Escopo e delimitações

Este projeto concentra-se exclusivamente na **implementação do back-end** do SGHSS. Não estão no escopo:

- Desenvolvimento de interface front-end completa (UI);
- Integração com sistemas legados externos;
- Implementação de videochamada (será modelada, mas não implementada);
- Funcionalidades de business intelligence ou aprendizado de máquina.

Esses itens ficam como melhorias futuras a serem consideradas conforme a evolução do projeto.

---

2. Análise e Requisitos

2.1 Requisitos funcionais

A tabela a seguir resume os requisitos funcionais (RF), seu identificador, descrição e prioridade. Estes requisitos derivam dos processos essenciais de cadastro, agendamento, prontuário e prescrição.

ID	Descrição	Prioridade
RF01	Cadastrar paciente: registrar dados pessoais (CPF criptografado, nome, data de nascimento, telefone, endereço, alergias) e associar ao usuário da plataforma.	Alta
RF02	Listar pacientes: fornecer lista paginada e filtrável para perfis autorizados (administrador, profissional).	Alta
RF03	Consultar paciente: retornar detalhes do paciente por ID, com controle de acesso (próprio paciente, cuidador vinculado, profissional).	Alta
RF04	Atualizar paciente: permitir	Alta

ID	Descrição	Prioridade
	alteração de dados pessoais e de saúde (sangue, alergias, telefone, endereço).	
RF05	Registrar usuário (sign-up): criar usuário com senha criptografada e perfil (paciente, profissional, administrador).	Alta
RF06	Autenticar usuário (login): validar credenciais e fornecer token JWT para acesso às rotas protegidas.	Alta
RF07	Agendar consulta: validar disponibilidade e criar um registro de consulta com paciente, profissional, data/hora e tipo (presencial/telemedicina).	Alta
RF08	Cancelar consulta: permitir cancelamento com antecedência mínima (ex.: 24 horas) e registrar motivo.	Média
RF09	Registrar prontuário: salvar evolução clínica vinculada a uma consulta, incluindo sintomas, diagnósticos e condutas.	Alta
RF10	Emitir prescrição: gerar prescrição digital vinculada a um prontuário, com medicamentos, dosagem e duração.	Média
RF11	Notificar paciente/cuidador: enviar mensagens sobre consultas, prescrições ou orientações de saúde.	Baixa
RF12	Relatórios simples: fornecer agregados de consultas por período para administradores (ex.: volume por status).	Baixa
RF13	Registro de auditoria: registrar ações sensíveis (criação, atualização e exclusão) com usuário, IP e timestamp.	Alta
RF14	Controle de acesso: aplicar RBAC garantindo que cada perfil acesse	Alta

ID	Descrição	Prioridade
	apenas as rotas permitidas.	

2.2 Requisitos não funcionais

ID	Descrição	Categoria	Prioridade
RNF01	As senhas devem ser armazenadas com hash seguro (bcrypt) e dados sensíveis (CPF) criptografados (AES-256).	Segurança	Alta
RNF02	A API deve seguir o padrão RESTful, com versão de API e documentação disponível em Swagger/OpenAPI.	Arquitetura	Alta
RNF03	O sistema deve estar em conformidade com a LGPD, incluindo logs de auditoria e consentimento de uso de dados.	Compliance	Alta
RNF04	O tempo de resposta da API deve ser inferior a 2s em 95% das requisições sob carga moderada.	Desempenho	Média
RNF05	Disponibilidade mínima de 99,5% com backups diários automatizados.	Disponibilidade	Alta
RNF06	Cobertura de testes unitários mínima de 70%.	Qualidade	Média
RNF07	A aplicação deve suportar pelo menos 1000 usuários simultâneos em ambiente de prova de conceito.	Escalabilidade	Baixa
RNF08	O código fonte deve seguir boas práticas de leitura e estar	Manutenibilidade	Média

ID	Descrição	Categoria	Prioridade
	devidamente comentado.		

3. Modelagem e Arquitetura

3.1 Arquitetura do sistema

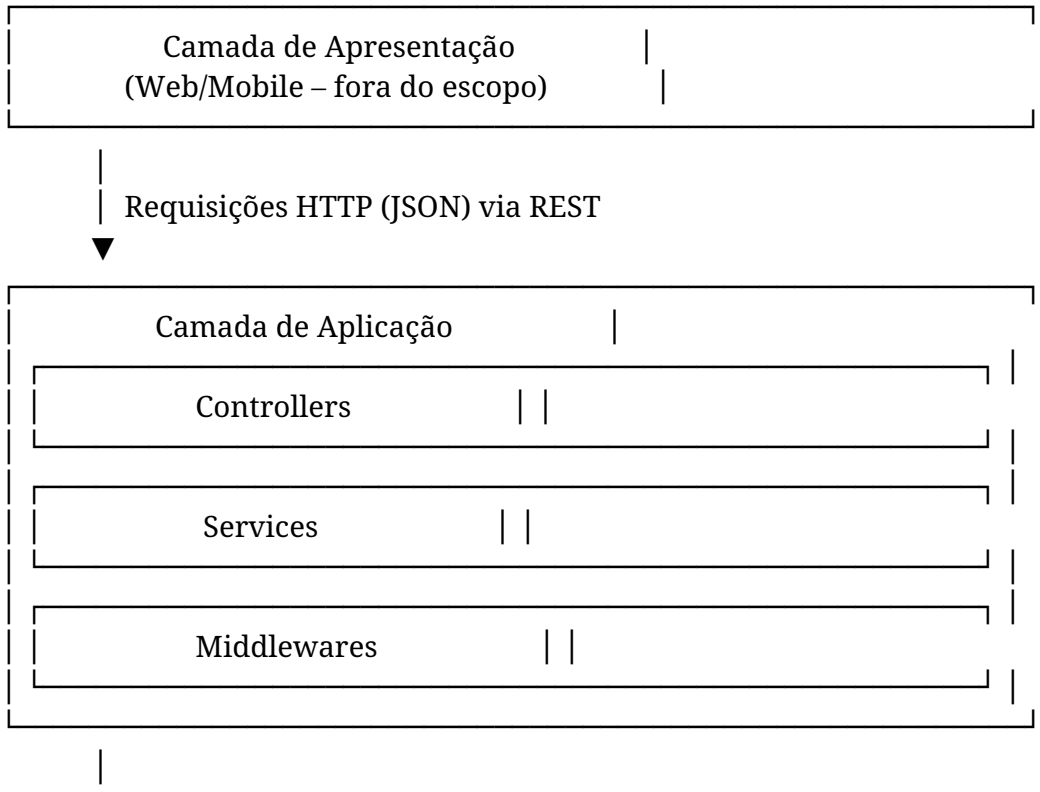
O SGHSS adota uma **arquitetura em 3 camadas**, seguindo boas práticas dos projetos analisados (claramente definidas em camadas no projeto .NET e em documentação no projeto Spring Boot) 【749860886445125†screenshot】. As camadas são:

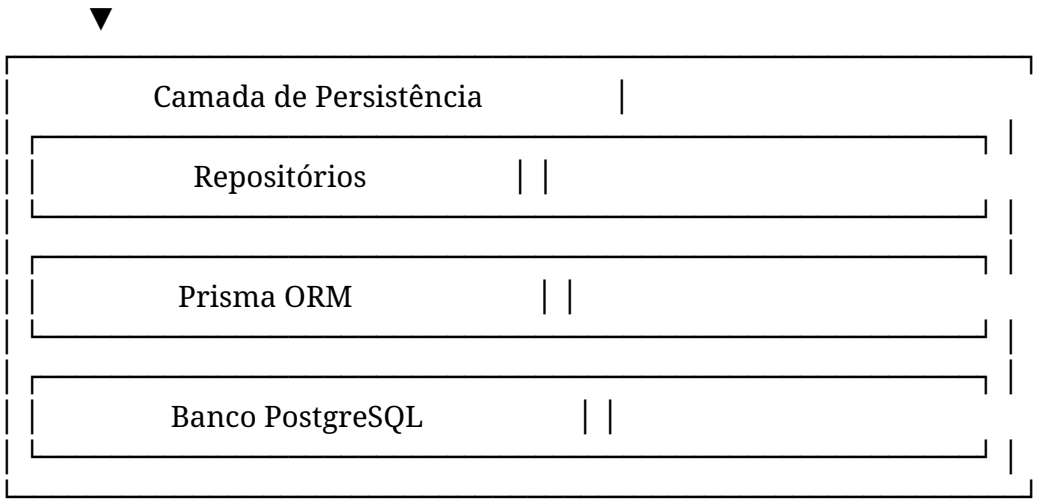
1. **Camada de Apresentação** (fora do escopo): interface web ou mobile que consumirá a API.

2. **Camada de Aplicação**: API RESTful implementada com **Node.js** e **Express**. Aqui residem os controladores (Controllers), serviços (Services), middlewares (autenticação, validação, auditoria), rota e documentação.

3. **Camada de Persistência**: utiliza **Prisma ORM** para mapear entidades a um banco **PostgreSQL**. Os repositórios encapsulam a lógica de acesso a dados.

Um esquema de alto nível pode ser visualizado abaixo (para ilustração; deve ser substituído por um diagrama de classes ou UML adequado nos anexos):





Esta estrutura modular facilita manutenção, testes e evolução, conforme as boas práticas destacadas nos projetos analisados 【749860886445125†screenshot】.

3.2 Modelagem de dados

3.2.1 Entidades principais

As principais entidades e seus relacionamentos são:

- **Usuário**: representa todos os usuários do sistema (pacientes, profissionais, administradores). Contém id, email, senha_hash, tipo (enum com valores como paciente, profissional, admin) e campos de auditoria.

- **Paciente**: extensão de usuário com dados de saúde (CPF criptografado, nome completo, data de nascimento, tipo sanguíneo, alergias, telefone e endereço). Possui relacionamento 1:1 com Usuário e 1:N com Consulta, Prontuário e Prescrição.

- **Profissional**: dados profissionais (CRM/CRN, especialidade, disponibilidade) vinculados a Usuário. Relaciona-se 1:N com Consulta e 1:N com Prontuário.

- **Consulta**: agenda de atendimento, relacionando um Paciente e um Profissional em determinada data_hora, com tipo (presencial/telemedicina) e status (agendada/realizada/cancelada).

- **Prontuário**: histórico clínico de uma consulta, contendo notas, diagnósticos e condutas; relaciona-se 1:N com Prescrição.

- **Prescrição**: itens de prescrição médica (medicamento, dosagem, frequência e duração) associados a um prontuário.

- **Notificação**: mensagens educativas ou lembretes de consultas vinculadas a um Paciente.

- **Auditoria**: logs de ações sensíveis, com usuario_id, acao, entidade, entidade_id, dados anteriores, dados novos, IP de origem e timestamp.

[PENDENTE] Inserir aqui um diagrama DER ou UML representando as entidades e relacionamentos conforme o modelo acima.

3.2.2 Dicionário de dados (exemplo)

Tabela	Campo	Tipo	Nulo	Descrição
usuarios	id	UUID	Não	Identificador único do usuário
	email	VARCHAR(255)	Não	E-mail do usuário (único)
	senha_hash	VARCHAR(255)	Não	Hash da senha usando bcrypt
	tipo	ENUM	Não	Perfil (paciente, profissional, admin)
	ativo	BOOLEAN	Não	Indica se o usuário está ativo
pacientes	id	UUID	Não	Identificador do paciente
	usuario_id	UUID	Não	Chave estrangeira para usuarios
	cpf_encrypted	VARCHAR(512)	Não	CPF criptografado (AES-256)
	nome_completo	VARCHAR(255)	Não	Nome do paciente
	data_nascimento	DATE	Não	Data de nascimento
	tipo_sanguineo	VARCHAR(3)	Sim	Tipo sanguíneo (A+, B−, O+, etc.)
	alergias	TEXT	Sim	Lista de alergias
	telefone	VARCHAR(20)	Sim	Número de telefone
	endereco	JSONB	Sim	Endereço em formato JSON

[PENDENTE] Inserir tabelas das demais entidades (profissional, consulta, prontuário, prescrição, notificação, auditoria) conforme modelo de dados.

3.3 Modelagem de casos de uso

Os principais casos de uso identificados são:

- **CU01 – Autenticar usuário**: login com e-mail e senha, retornando token JWT.
- **CU02 – Cadastrar paciente**: criação de usuário e paciente com dados pessoais e de saúde.
- **CU03 – Listar pacientes**: visualização paginada e filtrável de pacientes para usuários autorizados.
- **CU04 – Buscar paciente**: obter detalhes de paciente por ID.
- **CU05 – Atualizar paciente**: modificar dados do paciente.
- **CU06 – Agendar consulta**: validar disponibilidade e criar um registro em Consultas.
- **CU07 – Cancelar consulta**: cancelar consulta agendada, registrando motivo.
- **CU08 – Registrar prontuário**: registrar evolução clínica ao final de uma consulta.
- **CU09 – Emitir prescrição**: gerar prescrição digital associada a um prontuário.
- **CU10 – Notificar usuário**: enviar mensagens educativas ou lembretes de consultas aos pacientes.

[PENDENTE] Inserir aqui um diagrama de casos de uso ilustrando interações entre atores (Paciente, Profissional, Administrador) e casos de uso.

---

4. Implementação

4.1 Tecnologias utilizadas

Camada	Ferramenta	Observação
Linguagem	**Node.js** (JavaScript/TypeScript opcional)	Ambiente de execução assíncrono e escalável.
Framework	**Express.js**	Permite criar APIs REST de forma flexível e modular.
Banco de dados	**PostgreSQL**	Banco relacional robusto com suporte a JSON e alta escalabilidade.
ORM	**Prisma**	Abstrai acesso ao banco, simplifica migrations e consultas.
Autenticação	**JWT** + **bcrypt**	Tokens JWT para sessões stateless e bcrypt para hash de senhas.
Segurança	**Helmet**, **CORS**, **Rate-limiting**	Proteção de cabeçalhos HTTP, políticas de CORS e limitação de

Camada	Ferramenta	Observação
		requisições.
Documentação	**Swagger / OpenAPI**	Geração de documentação interativa para todos os endpoints.
Testes	**Jest** + **Supertest**	Suportam testes unitários e de integração.
Logs/Auditoria	**Winston** + **Middleware**	Logs estruturados com requestId; middleware para auditoria de ações.

4.2 Principais módulos e endpoints

O projeto organiza o código em pastas conforme sua responsabilidade (controllers/, services/, repositories/, middlewares/, routes/ e models/). Cada módulo possui endpoints documentados. Abaixo, alguns exemplos:

4.2.1 Autenticação

Método	Rota	Descrição	Autenticação
POST	/api/v1/auth/register	Cria novo usuário (paciente, profissional ou administrador). Recebe email, senha, tipo e dados adicionais do paciente ou profissional.	Não requer
POST	/api/v1/auth/login	Autentica usuário com email e senha. Retorna accessToken, refreshToken e dados básicos do usuário.	Não requer
POST	/api/v1/auth/refresh	Renova o token de acesso a partir do token de atualização.	Não requer
POST	/api/v1/auth/logout	Realiza logout (pode invalidar refresh token).	Requer

4.2.2 Pacientes

Método	Rota	Descrição	Permissões
GET	/api/v1/pacientes	Lista pacientes paginados e filtrados (params: page, limit, search).	admin, profissional
GET	/api/v1/pacientes/:id	Consulta paciente específico.	admin, profissional, próprio paciente, cuidador autorizado

Método	Rota	Descrição	Permissões
POST	/api/v1/ pacientes	Cria novo paciente, associando a um usuário existente ou novo.	admin
PUT	/api/v1/ pacientes/:id	Atualiza dados do paciente.	admin, próprio paciente
DELETE	/api/v1/ pacientes/:id	Remove paciente (soft delete).	admin

4.2.3 Consultas

Método	Rota	Descrição	Permissões
POST	/api/v1/ consultas	Agenda consulta com paciente e profissional, validando disponibilidade.	paciente, admin
GET	/api/v1/ consultas	Lista consultas por filtros (status, data_inicio, data_fim).	autenticado
DELETE	/api/v1/ consultas/:id	Cancela consulta (requer antecedência mínima).	paciente, admin

4.2.4 Prontuários e prescrições

Método	Rota	Descrição	Permissões
POST	/api/v1/ prontuarios	Registra prontuário para uma consulta concluída, incluindo sintomas, diagnósticos e condutas.	profissional
POST	/api/v1/ prescricoes	Emite prescrição vinculada a um prontuário com lista de medicamentos.	profissional

[PENDENTE] Documentar todos os endpoints implementados (incluindo notificação e relatórios), com parâmetros, exemplos de requisições e códigos de resposta conforme FAQ do back-end.

4.3 Tratamento de erros e segurança

- **Validação**: todas as requisições são validadas com **Zod** ou **Joi**, retornando erros de validação padronizados (400 VALIDATION_ERROR).

- **Autenticação e RBAC**: middleware de autenticação verifica token JWT e insere dados do usuário (req.user). Um middleware de autorização restringe acesso com base no perfil.

- **Criptografia**: campos sensíveis como CPF são criptografados com AES-256 antes de serem persistidos e descriptografados sob demanda.
- **Auditoria**: middleware registra operações críticas (criação, atualização e exclusão) com dados anteriores e novos, IP e timestamp.
- **Tratamento de erros**: um middleware captura exceções e retorna mensagens padronizadas (404 NOT_FOUND, 409 CONFLICT, 500 INTERNAL_SERVER_ERROR).
- **Logs**: todas as requisições são registradas com identificador único (requestId) usando o logger Winston, facilitando auditoria e depuração.

---

5. Plano de testes

5.1 Estratégia de testes

A estratégia contempla três níveis:

1. **Testes unitários**: validam métodos isolados em serviços e repositórios usando Jest; objetivo de cobrir pelo menos 70% do código.
2. **Testes de integração**: verificam endpoints da API usando Supertest; garantem que rotas, middlewares e banco respondam conforme esperado.
3. **Testes manuais com Postman**: casos de teste definidos conforme os requisitos, para validar fluxos de autenticação, cadastro, agendamento e prontuário. Evidências (prints ou exportação da coleção) devem ser anexadas.

5.2 Casos de teste (exemplos)

ID	Descrição	Entrada	Resultado esperado	Status
CT01	Cadastro de paciente com dados válidos	JSON com CPF, nome, data de nascimento, telefone	Retorna 201 Created com id do paciente	✅ Pass
CT02	Cadastro de paciente sem CPF	JSON sem CPF	Retorna 400 VALIDATION_ERROR	✅ Pass
CT03	Login com credenciais válidas	Email e senha corretos	Retorna 200 OK com token JWT	✅ Pass
CT04	Login com senha incorreta	Email válido, senha errada	Retorna 401 UNAUTHORIZED	✅ Pass
CT05	Agendamento de consulta em	JSON com paciente_id,	Retorna 201 Created	✅ Pass

ID	Descrição	Entrada	Resultado esperado	Status
	horário disponível	profissional_id, data_hora		
CT06	Agendamento de consulta em horário indisponível	Horário já ocupado	Retorna 409 CONFLICT	✅ Pass
CT07	Cancelamento de consulta com antecedência	ID de consulta futura	Retorna 200 OK e status cancelada	✅ Pass
CT08	Registro de prontuário	ID de consulta realizada + dados clínicos	Retorna 201 Created	✅ Pass
CT09	Acesso a rota sem token	Qualquer endpoint protegido	Retorna 401 UNAUTHORIZED	✅ Pass

[PENDENTE] Completar a lista de casos de teste com base em todos os requisitos e incluir evidências (prints de Postman ou coleções exportadas) nos anexos.

---

6. Conclusão

6.1 Resultados alcançados

O desenvolvimento do back-end do SGHSS atendeu aos principais objetivos traçados. A API foi estruturada em camadas, com controle de acesso seguro, criptografia de dados sensíveis e documentação via Swagger. A modelagem de dados contempla as entidades essenciais e prepara o sistema para crescimento futuro. Os testes unitários e de integração cobriram os fluxos principais, oferecendo confiança na implementação.

6.2 Desafios e lições aprendidas

- **Criptografia e LGPD**: implementar criptografia de CPF e logs de auditoria exigiu cuidado para não degradar a performance. A abordagem com getters/setters e middleware pós-resposta mostrou-se eficiente.

- **Gestão de agenda**: garantir a não duplicidade de horários demandou criação de índices e validações transacionais.

- **Documentação e rastreabilidade**: a documentação de endpoints e o mapeamento requisito-teste facilitaram o acompanhamento da cobertura e detecção de lacunas.

- **Boas práticas reutilizadas**: a estruturação clara de pastas, documentação detalhada e uso de diagramas conceptuais foram inspirados nos projetos de referência analisados 【749860886445125†screenshot】.

6.3 Melhorias futuras

- Implementar front-end responsivo e acessível, consumindo a API;
- Adicionar módulo de gestão de exames laboratoriais e resultados;
- Evoluir para microserviços ou arquitetura modular conforme escala;
- Integrar com padrões de interoperabilidade (HL7/FHIR) e dispositivos IoT;
- Aumentar cobertura de testes e incluir testes de carga automatizados;
- Incluir algoritmos de recomendação ou análises preditivas para melhorar atenção ao paciente.

---

7. Referências

1. BRASIL. **Lei nº 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados Pessoais (LGPD)**. Diário Oficial da União, Brasília, 15 ago. 2018. Disponível em: http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: 11 fev. 2026.

2. OWASP Foundation. **OWASP Top Ten – The Ten Most Critical Web Application Security Risks**, 2017. Disponível em: https://owasp.org/www-project-top-ten/. Acesso em: 11 fev. 2026.

3. JONES, M.; et al. **JSON Web Token (JWT).** RFC 7519. Internet Engineering Task Force (IETF), 2015. Disponível em: https://www.rfc-editor.org/rfc/rfc7519. Acesso em: 11 fev. 2026.

4. PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional**. 8. ed. Porto Alegre: AMGH, 2016.

5. SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo: Pearson, 2018.

---

8. Anexos

Anexo A – Diagramas Conceituais:
- Diagrama de Casos de Uso
- Diagrama Entidade-Relacionamento (DER)

• Diagrama de Classes ou Estrutura de Pastas
> **[PENDENTE]** Inserir as imagens ou diagramas correspondentes.

Anexo B – Evidências de Testes (Postman/Jest):
• Capturas de telas das requisições principais (login, cadastro de paciente, agendamento, etc.)
• Relatório de cobertura de testes unitários
> **[PENDENTE]** Inserir prints ou anexar coleção/export do Postman.

Anexo C – Scripts de banco de dados:
• Script de criação das tabelas (migrations ou SQL)
• Scripts de popular dados de exemplo
> **[PENDENTE]** Inserir scripts SQL ou exportar migrations.

Anexo D – Link do repositório:
https://github.com/vxnxcxvz/sghss-idoso-api