CENTRO UNIVERSITARIO INTERNACIONAL UNINTER
CURSO: Analise e Desenvolvimento de Sistemas
DISCIPLINA: Projeto multidisciplinar: Desenvolvimento back-end
ALUNO: Robinson Francisco Silva de Mesquita
RU: 4522562
POLO DE APOIO: PAP SÃO LUIS (CUTIM ANIL - LAPAD) MA
SEMESTRE: 2º Semestre de 2025
PROFESSOR: Winston Sen Lun Fung

PROJETO MULTIDICIPLINAR: DESENVOLVIMENTO BACK-END

SÃO LUIS - MA
2025

ROBINSON FRANCISCO SILVA DE MESQUITA
RU: 4522562

PROJETO MULTICIPLINAR: DESENVOLVIMENTO BACK-END

Trabalho apresentado à disciplina Projeto multidisciplinar: Desenvolvimento back-end, do curso de Análise e Desenvolvimento de Sistemas, afim de ser realizado a entrega do trabalho pratico no 2º semestre de 2025, sob orientação do(a) Prof. Winston Sen Lun Fung

SÃO LUIS - MA
2025

SUMÁRIO

1. INTRODUÇÃO

A crescente demanda por soluções tecnológicas na área da saúde tem impulsionado o desenvolvimento de sistemas capazes de integrar, automatizar e otimizar processos hospitalares e clínicos. Nesse contexto, a instituição VidaPlus, responsável pela administração de hospitais, clínicas de bairro, laboratórios e equipes de atendimento domiciliar (home care), identificou a necessidade de implementar um Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS).

O SGHSS tem como objetivo facilitar o gerenciamento de informações relacionadas ao atendimento de pacientes, à atuação de profissionais da saúde, à administração hospitalar e à realização de consultas por telemedicina. A proposta do sistema contempla funcionalidades como cadastro de pacientes, agendamento de consultas, registro de exames, emissão de receitas digitais, controle de suprimentos, além de mecanismos de segurança e conformidade com a Lei Geral de Proteção de Dados (LGPD).

Os principais usuários do sistema são os pacientes, os profissionais de saúde (médicos, enfermeiros e técnicos) e os administradores (admins) hospitalares.
A relevância do SGHSS está na sua capacidade de promover maior eficiência operacional, melhorar a qualidade do atendimento, reduzir erros administrativos e ampliar o acesso à saúde por meio de recursos como a telemedicina. Além disso, o sistema foi desenvolvido seguindo os princípios da arquitetura RESTful, utilizando tecnologias modernas e escaláveis, o que permite sua adaptação a diferentes unidades hospitalares e contextos clínicos. Tendo como base o modelo MVC (Model–View–Controller), onde organiza o código em três camadas principais.

2. ANÁLISE E REQUISITOS

A análise de requisitos é uma etapa fundamental no desenvolvimento de sistemas, pois permite identificar e documentar as funcionalidades esperadas, bem como os critérios técnicos e operacionais que devem ser atendidos. No contexto do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS), desenvolvido para a instituição VidaPlus, os requisitos foram definidos com base nas necessidades dos usuários e nas boas práticas de engenharia de software.

2.1 Requisitos funcionais

Os requisitos funcionais descrevem as ações que o sistema deve ser capaz de realizar, considerando os diferentes perfis de usuários: pacientes, profissionais de saúde e administradores.

- **RF01 – Cadastro de pacientes:** O sistema deve permitir o registro e a atualização dos dados pessoais dos pacientes, incluindo informações clínicas e de contato.
- **RF02 – Agendamento de consultas e exames:** Deve ser possível agendar, reagendar e cancelar consultas e exames, tanto por pacientes quanto por profissionais de saúde.
- **RF03 – Gestão de profissionais de saúde:** O sistema deve possibilitar o cadastro de médicos, enfermeiros e técnicos, com dados sobre especialidades, horários de atendimento e agendas.
- **RF04 – Módulo de telemedicina:** Deve ser implementada a funcionalidade de consultas online por meio de videochamadas seguras, com registro automático no prontuário do paciente.
- **RF05 – Controle de acesso e auditoria:** O sistema deve permitir a criação de perfis de usuário (paciente, profissional, administrador) e registrar todas as ações realizadas, garantindo rastreabilidade e conformidade com normas de auditoria.
- **RF06 – Gestão de suprimentos e estoques:** O sistema deve controlar a entrada, saída e consumo de materiais hospitalares e medicamentos, permitindo o gerenciamento eficiente dos recursos.

2.2 Requisitos não funcionais

Os requisitos não funcionais referem-se às características técnicas e operacionais que o sistema deve apresentar, como desempenho, segurança, compatibilidade e usabilidade.

- **RFN01 – Disponibilidade** O sistema deve estar disponível pelo menos 99,5% do tempo mensal, com backups automáticos realizados diariamente.

- **RFN02 – Backup e recuperação** Deve existir um plano de backup e restauração de dados, assegurando a recuperação completa em caso de falhas ou perda de informações.
- **RFN03 – Compatibilidade** O sistema deve ser compatível com os principais navegadores modernos (Chrome, Edge, Firefox, Safari) e funcionar adequadamente em dispositivos móveis.
- **RFN04 – Verificabilidade** Todas as ações realizadas pelos usuários devem ser registradas em logs invioláveis, contendo data, hora e identificação do responsável.
- **RFN05 – Manutenibilidade** A arquitetura do sistema deve ser modular, permitindo atualizações e melhorias sem comprometer o funcionamento geral da aplicação.
- **RFN06 – Desempenho** O tempo de resposta para operações críticas, como a busca por prontuários, não deve ultrapassar três segundos.
- **RFN07 – Acessibilidade** A interface do sistema deve ser de fácil compreensão, respeitando os padrões de acessibilidade definidos pelo W3C/WCAG, de modo a garantir o uso por todos os perfis de usuários.

2.3 Diagrama de caso de uso

A seguir (figura 1), apresenta-se o diagrama de casos de uso que representa as principais interações entre os atores do sistema e suas respectivas funcionalidades. Esse artefato auxilia na visualização dos processos e na validação dos requisitos levantados.

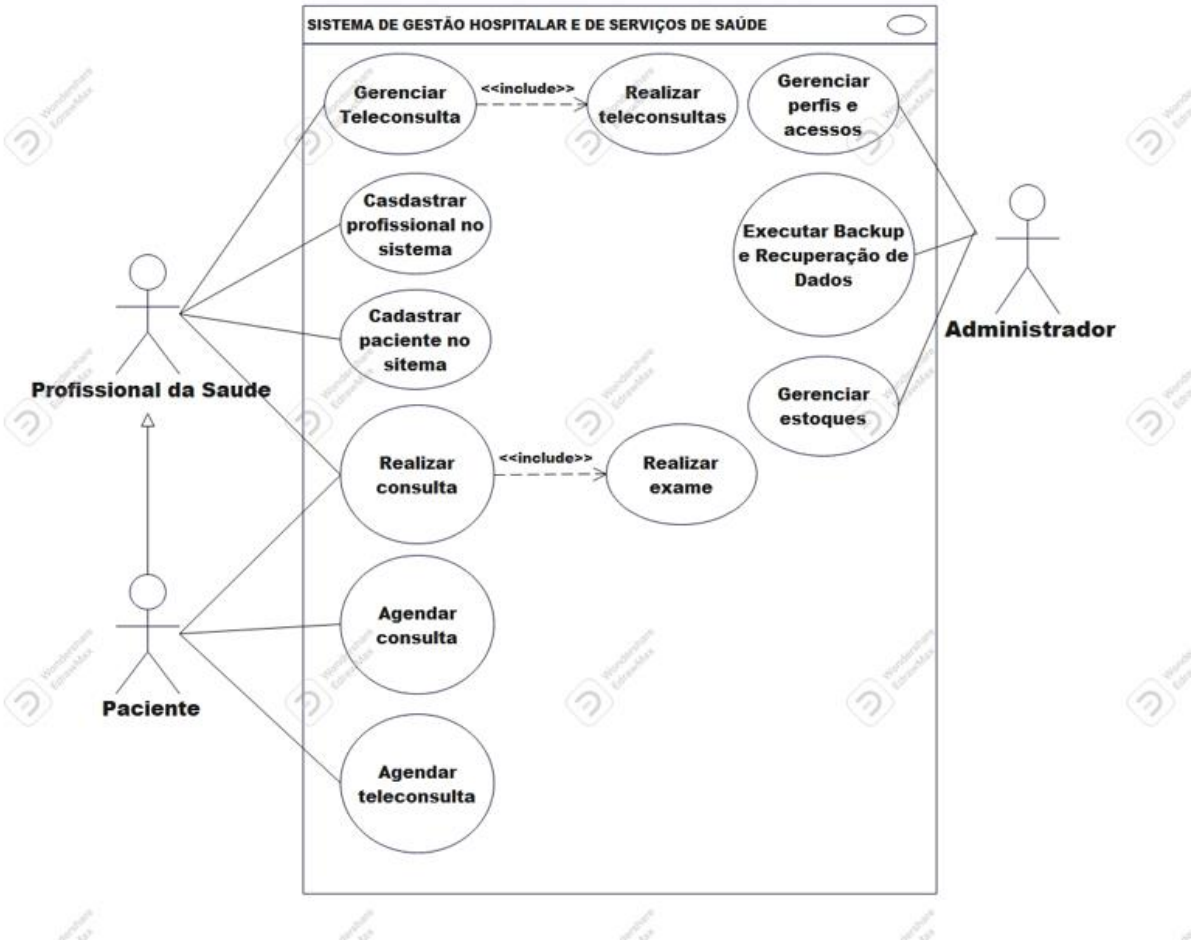

Figura 1 – Diagrama de Casos de Uso do Sistema SGHSS Fonte: Elaboração própria.

3. MODELAGEM E ARQUITETURA

A modelagem e definição da arquitetura do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS) foram elaboradas com foco na organização lógica das entidades, na estrutura de dados e na padronização das funcionalidades da API. A seguir, são apresentados os principais artefatos de modelagem e as decisões arquiteturais adotadas.

3.1 Diagrama de classes

O diagrama de classes (figura 2) define as principais entidades do sistema, seus atributos, métodos e relacionamentos. Foram modeladas classes como `Paciente`, `ProfissionalSaude`, `Consulta`, `Exame`, `Telemedicina`, `Administrador`, `Estoque`, `Suprimentos`, entre outras, com associações do tipo 1:N e N:N.

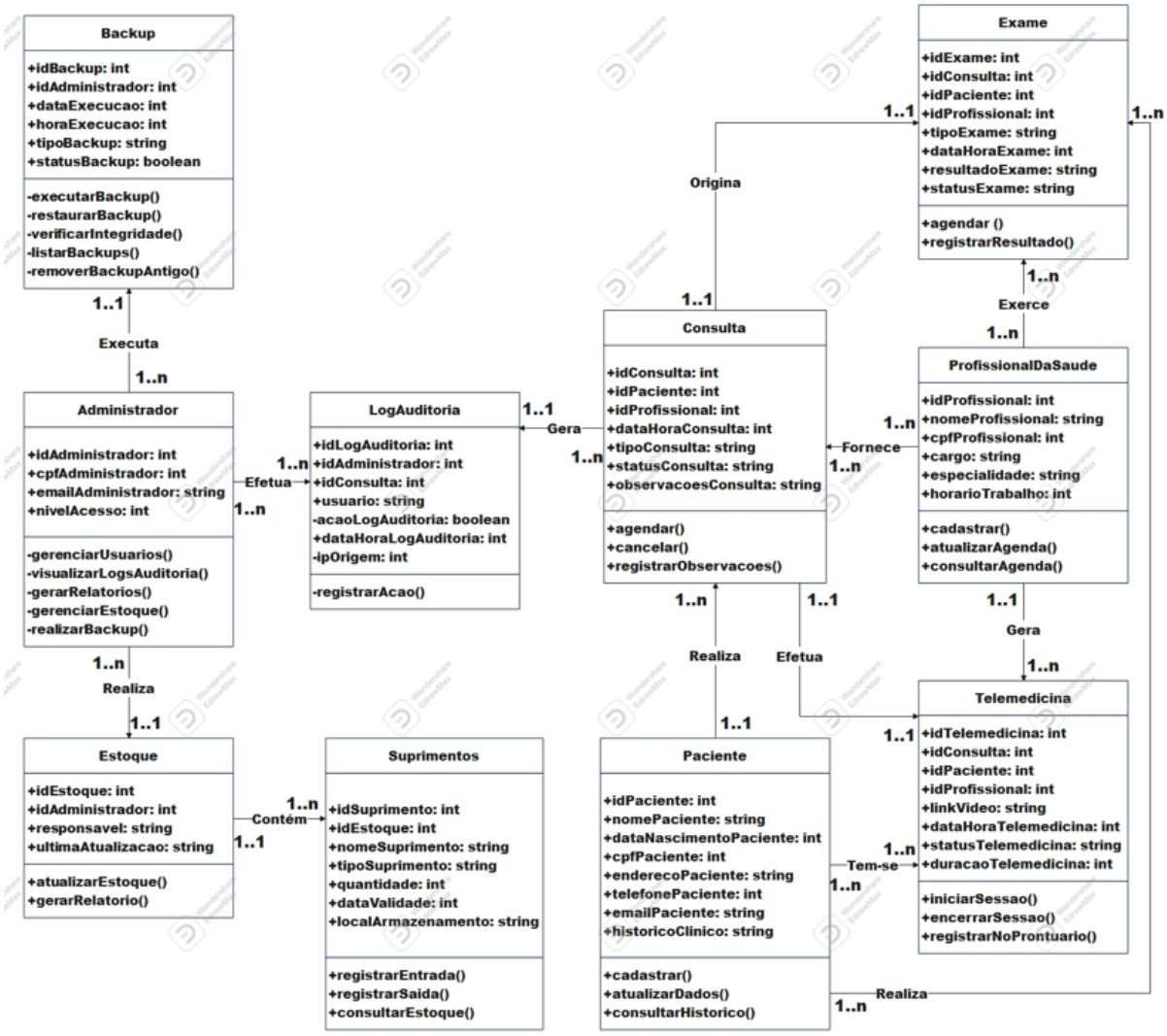

Figura 2 – Diagrama de Classes do Sistema SGHSS Fonte: Elaboração própria.

3.2 Diagrama entidade-relacionamento (DER)

O modelo relacional do banco de dados foi construído com base nas entidades identificadas no diagrama de classes. O DER (figura 3) contempla tabelas como `paciente`, `consulta`, `exame`, `profissional_saude`, `telemedicina`, `estoque`, `suprimentos`, `login`, `administrador`, entre outras, com suas respectivas chaves primárias e estrangeiras.

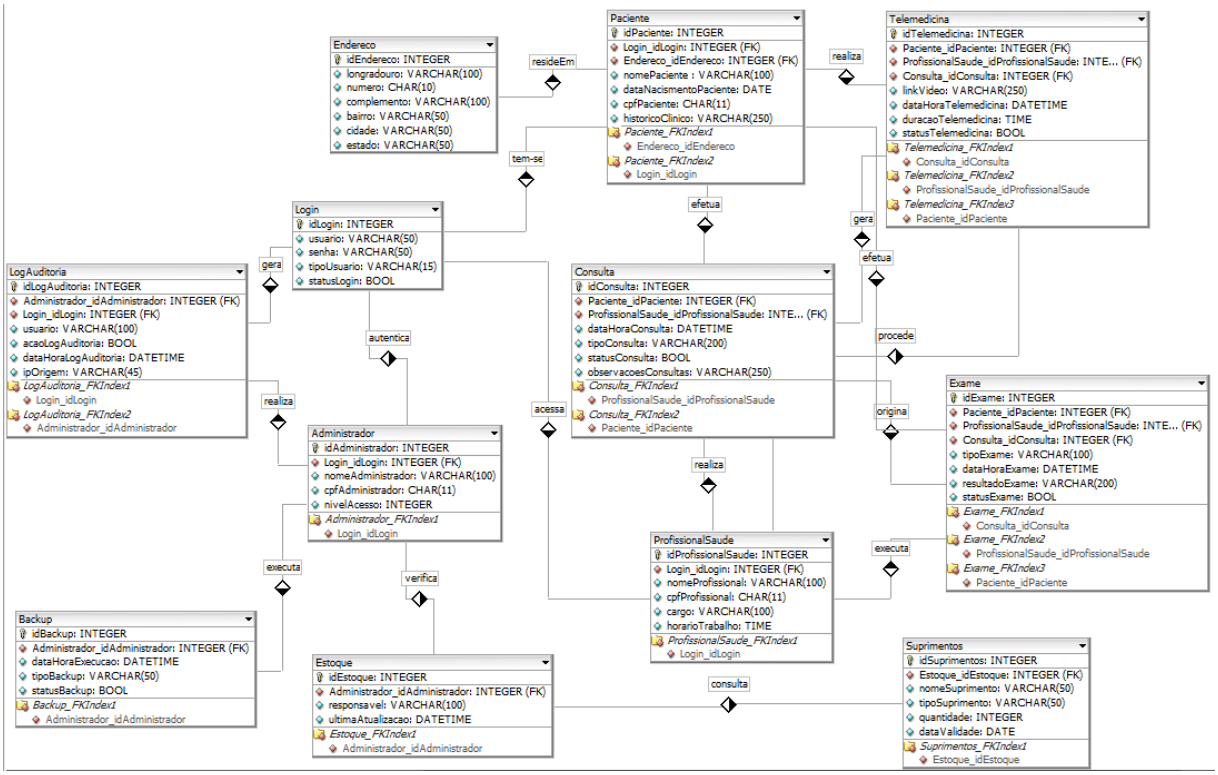

Figura 3 – Diagrama Entidade-Relacionamento do Sistema SGHSS Fonte: Elaboração própria.

3.3 Tecnologias utilizadas

- **JavaScript**: linguagem principal do backend.
- **Node.js**: ambiente de execução para JavaScript no servidor.
- **Express.js**: framework para criação de rotas e gerenciamento de requisições.
- **MySQL**: banco de dados relacional utilizado para persistência dos dados.
- **mysql2**: driver para conexão entre Node.js e MySQL.
- **bcryptjs**: biblioteca para criptografia de senhas.
- **jsonwebtoken (JWT)**: autenticação baseada em tokens.
- **dotenv**: gerenciamento de variáveis de ambiente.
- **CORS**: controle de acesso entre origens distintas.
- **Insomnia**: ferramenta para testes de requisições HTTP.

3.4 Arquitetura do sistema

O sistema foi desenvolvido com base em uma **arquitetura monolítica modular**. A aplicação backend foi estruturada em camadas, conforme descrito abaixo:

- **Camada de controle (`controllers`)**: contém a lógica de negócio e manipulação dos dados recebidos.
- **Camada de middleware (`middlewares`)**: responsável pela autenticação JWT, controle de acesso e tratamento de erros.
- **Camada de modelos (`models`)**: realiza a conexão com o banco de dados MySQL utilizando o `mysql2`.
- **Camada de rotas (`routes`)**: define os endpoints da API e direciona as requisições para os controladores.

A arquitetura adotada permite a separação de responsabilidades, facilita a manutenção do código e garante escalabilidade futura.

3.5 Principais endpoints da API

A seguir, são apresentados os principais endpoints implementados na API RESTful do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS). Quase todas as rotas foram desenvolvidas com autenticação via token JWT e segue os princípios da arquitetura REST, permitindo operações de cadastro, consulta, atualização e exclusão de dados.

- **POST /api/admin/signup** Cadastro de administrador do sistema.
- **POST /api/admin/login** Autenticação de administrador com geração de token JWT.
- **POST /api/pacientes** Registro de novo paciente, incluindo dados pessoais, clínicos e endereço.
- **DELETE /api/pacientes/{id}** Exclusão de paciente por ID.
- **GET /api/pacientes** Listagem de pacientes cadastrados no sistema.
- **POST /api/funcionarios** Cadastro de profissional da saúde com cargo e horário de trabalho.
- **DELETE /api/funcionarios/{id}** Exclusão de profissional da saúde por ID.
- **GET /api/funcionarios** Listagem de profissionais da saúde ativos.

- **POST /api/consultas** Agendamento de consulta presencial entre paciente e profissional.
- **GET /api/consultas/simples** Listagem simplificada de consultas agendadas.
- **POST /api/teleconsultas** Agendamento de sessão de telemedicina com link de videochamada.
- **GET /api/teleconsultas/simples** Listagem de teleconsultas realizadas ou agendadas.
- **GET /api/estoques** Consulta aos estoques hospitalares e suprimentos disponíveis.

Tais endponts estão melhores documentados no link do github disponibilizado no arquivo README.md no próximo tema.

4. IMPLEMENTAÇÃO (PROTOTIPAGEM)

A implementação do SGHSS foi realizada com foco na construção de uma API funcional, segura e escalável, utilizando tecnologias modernas e boas práticas de desenvolvimento backend. O projeto está disponível publicamente no GitHub, conforme indicado abaixo.

Repositório do projeto: https://github.com/robinn04/sghss-backend

4.1 Estrutura do projeto

O projeto foi organizado em diretórios específicos para cada camada da aplicação:

- `controllers/`: lógica de negócio das rotas.
- `middlewares/`: autenticação, validação e tratamento de erros.
- `models/`: conexão e manipulação do banco de dados.
- `routes/`: definição dos endpoints da API.
- `.env/`: variáveis de configuração de conexão com o banco de dados
- `index.js`: ponto de entrada da aplicação.

4.2 Boas práticas adotadas

- Utilização de **Git** para controle de versão e histórico de alterações.
- Padronização de nomenclatura de arquivos, funções e variáveis.
- Tratamento de erros.
- Separação de responsabilidades por camadas.
- Uso de variáveis de ambiente para dados sensíveis.

5. PLANO DE TESTES

Esta etapa de testes tem como objetivo validar o funcionamento da API RESTful desenvolvida para o Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS). Foram realizados testes funcionais utilizando a ferramenta **Insomnia**, além de testes de carga e desempenho com o **Apache JMeter**, simulando varios usuários e cenários de uso intensivo.

5.1 Casos de testes funcionais

CT001 – Cadastro de administrador com dados válidos Objetivo: Verificar se o sistema permite o cadastro de um novo administrador. Requisição http:

```
POST /api/admin/signup
Content-Type: application/json
body:
{
  "usuario": "admin5",
  "senha": "123456",
  "nomeAdministrador": "Robinson",
  "cpfAdministrador": "12345678901",
  "nivelAcesso": 1
}
```

Resultado esperado:

```
{
  "message": "Administrador cadastrado com sucesso!"
}
```

CT002 – Login de administrator com credenciais válidas Objetivo: Validar o processo de autenticação e geração de token JWT. Requisição http:

```
POST /api/admin/login
Content-Type: application/json
body:
{
  "usuario": "admin5",
  "senha": "123456"
}
```

Resultado esperado:

```
{
  "message": "Login realizado com sucesso",
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZExvZ2luIjoxMywidGlwb1VzdWFyaW8iOiJhZG1p
bmlzdHJhZG9yIiwiaWF0IjoxNzYwOTE2MDA0LCJleHAiOjE3NjEwMDI0MDR9.vWdfbT2elshfJwawHdWPZ
dG-ArK-cm9A3S6qDseFOl0"
}
```

CT003 – Cadastro de paciente com dados completos Objetivo: Verificar se o sistema registra corretamente os dados do paciente. Requisição http:

```
POST /api/pacientes
Authorization: Bearer <token>
Content-Type: application/json
body:
{
  "usuario": "joaopaciente",
  "senha": "123456",
  "nomePaciente": "João da Silva",
  "cpfPaciente": "12345678901",
  "dataNascimentoPaciente": "1990-05-10",
  "historicoClinico": "Hipertensão leve",
  "endereco": {
    "longradouro": "Rua das Flores",
    "numero": "123",
    "complemento": "Apto 2",
    "bairro": "Centro",
    "cidade": "Vargem Grande",
    "estado": "MA"
```

```
  }
}
```

Resultado esperado:

```
{
  "message": "Paciente cadastrado com sucesso!"
}
```

CT004 – Remoção de paciente por ID Objetivo: Validar a exclusão de um paciente específico. Requisição http:

```
DELETE /api/pacientes/1
Authorization: Bearer <token>
Content-Type: application/json
```

Resultado esperado:

```
{
  "message": "Paciente removido com sucesso!"
}
```

CT005 – Listagem de pacientes Objetivo: Verificar se o sistema retorna corretamente os dados dos pacientes cadastrados. Requisição http:

```
GET /api/pacientes
Authorization: Bearer <token>
Content-Type: application/json
```

Resultado esperado:

```
{
  "pacientes": [
    {
      "idPaciente": 1,
      "nomePaciente": "Carlos Mendes",
      "cpfPaciente": "55555555555",
      "dataNacismentoPaciente": "1990-05-10T03:00:00.000Z",
      "historicoClinico": "Hipertensão",
      "longradouro": "Rua A",
      "numero": "101",
      "complemento": "Apto 1",
```

```
      "bairro": "Centro",
      "cidade": "São Luís",
      "estado": "MA"
    }
  ]
}
```

CT006 – Agendamento de consulta Objetivo: Validar o agendamento de uma consulta entre paciente e profissional. Requisição http:

```
POST /api/consultas
Authorization: Bearer <token>
Content-Type: application/json
Body:
{
  "Paciente_idPaciente": 5,
  "ProfissionalSaude_idProfissionalSaude": 5,
  "dataHoraConsulta": "2025-10-20 14:30:00",
  "tipoConsulta": "Consulta clínica geral",
  "statusConsulta": true,
  "observacoesConsultas": "Paciente relatou dores de cabeça recorrentes"
}
```

Resultado esperado:

```
{
  "message": "Consulta agendada com sucesso!"
}
```

CT007 – Agendamento de teleconsulta Objetivo: Verificar se o sistema registra corretamente uma sessão de telemedicina. Requisição http:

```
POST /api/teleconsultas
Authorization: Bearer <token>
Content-Type: application/json
body:
{
  "Paciente_idPaciente": 5,
  "ProfissionalSaude_idProfissionalSaude": 2,
  "linkVideo": "https://meet.example.com/5",
  "dataHoraTelemedicina": "2025-10-25 10:00:00",
```

```
  "duracaoTelemedicina": "00:45:00",
  "statusTelemedicina": true
}
```

Resultado esperado:

```
{
  "message": "Teleconsulta agendada com sucesso!"
}
```

CT008 – Listar funcionário sem token de autenticação: Validar o comportamento da API ao tentar acessar o endpoint de listagem de funcionários sem fornecer o token de verificação JWT. Requisição http:

```
GET /api/funcionarios
Authorization: (não informado)
Content-Type: application/json
```

Resultado esperado:

```
ERRO 403
{
  "message": "Token não fornecido"
}
```

CT009 – Listar paciente com token de autenticação errado: Validar o comportamento da API ao tentar acessar o endpoint de listagem de pacientes ao fornecer o token de verificação JWT de forma errada. Requisição http:

```
GET /api/pacientes
Authorization: (token digitado errado)
Content-Type: application/json
```

Resultado esperado:

```
ERRO 401
{
  "error": "Token inválido"
}
```

CT010 – Cadastrar profissional da saúde: Verificar se o sistema registra corretamente o cadastro de um funcionário. Requisição http:

```
POST /api/funcionarios
Authorization: Bearer <token>
Content-Type: application/json
body:
{
       "usuario": "enfermeira",
       "senha": "123456",
       "nomeProfissional": "Maria Oliveira",
       "cpfProfissional": "98765432100",
       "cargo": "Enfermeira",
       "horarioTrabalho": "08:00:00"
}
```

Resultado esperado:

```
{
  "message": "Funcionário cadastrado com sucesso!"
}
```

CT011 – Verificar estoques e suprimentos hospitalares: Verificar se o sistema esta registrando os estoques e suprimentos do hospital. Requisição http:

```
GET /api/estoques
Authorization: Bearer <token>
Content-Type: application/json
```

Resultado esperado:

```
{
"estoques": [
  {
    "idEstoque": 1,
    "ultimaAtualizacao": "2025-10-10T11:00:00.000Z",
    "suprimentos": [
      {
          "idSuprimentos": 1,
          "nomeSuprimento": "Luvas Cirúrgicas",
          "tipoSuprimento": "EPI",
          "quantidade": 500,
          "dataValidade": "2026-01-01T03:00:00.000Z"
          }
        ]
```

```
      }
    ]
  }
```

5.2 Casos de testes não funcionais

Com o objetivo de validar os requisitos não funcionais de desempenho, escalabilidade e disponibilidade do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS), foram realizados testes de carga e desempenho utilizando a ferramenta **Apache JMeter**. Os testes simularam múltiplos usuários em cenários críticos, com foco em dois pontos essenciais da aplicação: autenticação e cadastro de pacientes.

5.2.1 Teste de desempenho – Login simultâneo de administradores

Este teste teve como finalidade avaliar o tempo de resposta da API durante o processo de autenticação de administradores, em um cenário com alto volume de requisições simultâneas.

- **Endpoint testado**: POST /api/admin/login
- **Número de usuários simulados**: 500
- **Tempo de ramp-up**: 60 segundos
- **Número de repetições**: 1
- **Objetivo**: medir o tempo médio de resposta e a taxa de sucesso em autenticações simultâneas.

Resultados obtidos (conforme relatório de sumário do JMeter):

- **Tempo médio de resposta**: 19 ms
- **Tempo mínimo**: 18 ms
- **Tempo máximo**: 35 ms
- **Desvio padrão**: 1,37 ms
- **Taxa de erro**: 0,00%
- **Vazão**: 8,3 requisições por segundo

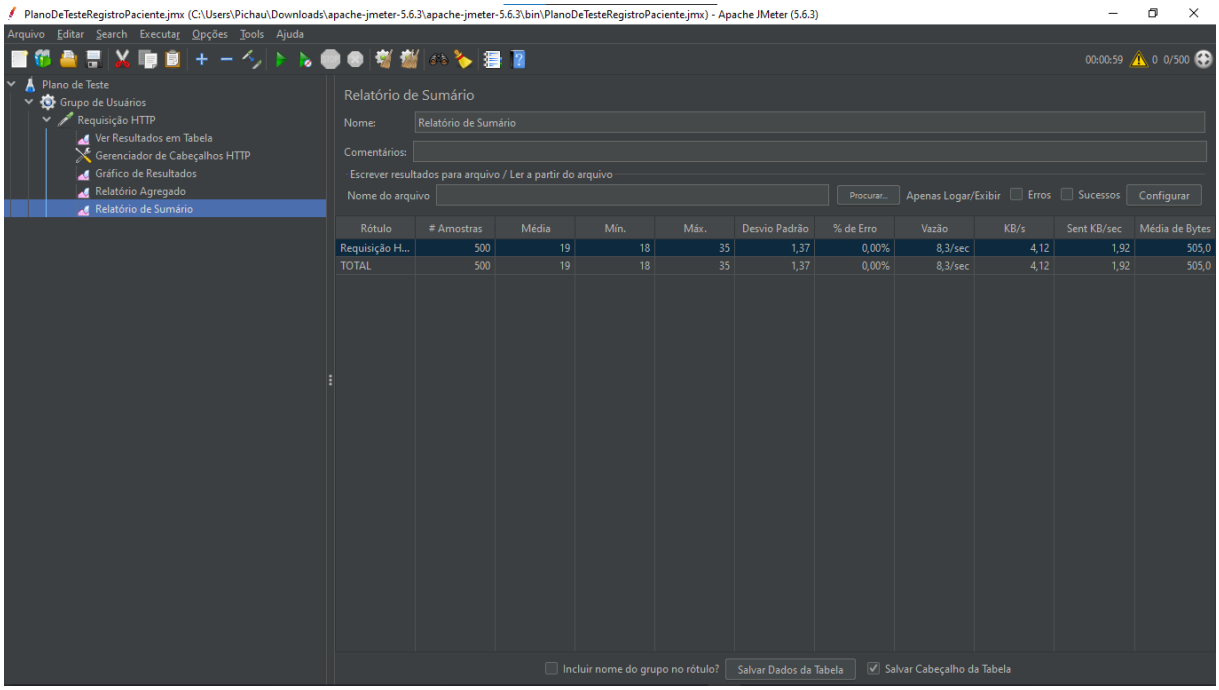

Figura 4 – Resultado do teste de desempenho do Sistema SGHSS (Jmater) Fonte: Elaboração própria.

A análise dos dados demonstra que o sistema manteve estabilidade e desempenho satisfatório mesmo sob carga elevada, sem apresentar falhas ou lentidão crítica. O tempo médio de resposta permaneceu dentro dos limites aceitáveis para aplicações hospitalares.

5.2.2 Teste de carga – Cadastro em massa de pacientes

Este teste teve como objetivo verificar a capacidade da API de lidar com múltiplos cadastros consecutivos de pacientes, simulando um ambiente de alta demanda operacional.

- **Endpoint testado**: POST /api/pacientes
- **Número de usuários simulados**: 100
- **Tempo de ramp-up**: 20 segundos
- **Número de repetições**: 10
- **Objetivo**: avaliar a estabilidade da API durante operações repetidas de inserção de dados.

Resultados obtidos (conforme relatório de sumário do JMeter):

- **Tempo médio de resposta**: 758 ms
- **Tempo mínimo**: 55 ms

- **Tempo máximo**: 1303 ms
- **Desvio padrão**: 357,31 ms
- **Taxa de erro**: 0,00%
- **Vazão**: 37,4 requisições por segundo

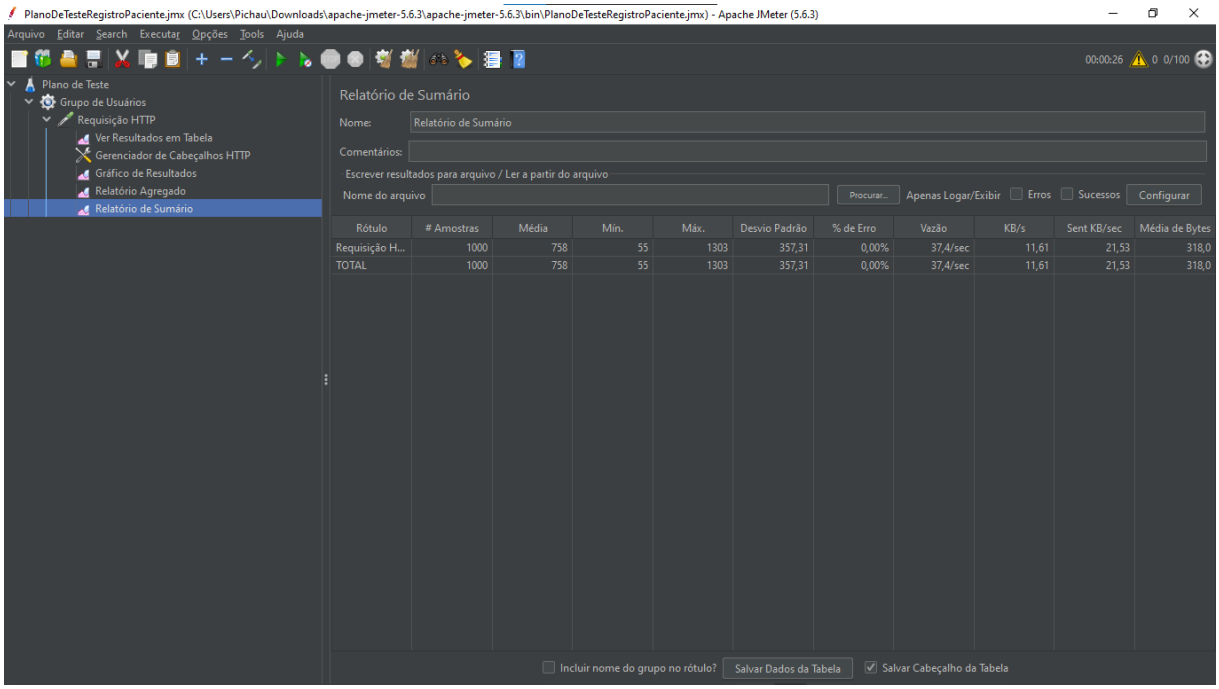

Figura 5 – Resultado do teste de carga do Sistema SGHSS (Jmater) Fonte: Elaboração própria.

Os resultados indicam que a API apresentou alta eficiência e consistência, com tempo de resposta baixo e ausência de variações significativas. A taxa de erro nula reforça a confiabilidade do sistema em operações de cadastro em lote.

5.2.3 Conclusões dos testes não funcionais

Os testes realizados confirmam que o SGHSS atende aos requisitos não funcionais definidos, especialmente no que se refere a:

- **Desempenho**: tempos de resposta inferiores a três segundos, conforme especificado nos requisitos.
- **Disponibilidade**: ausência de falhas durante os testes, mesmo sob carga elevada.
- **Escalabilidade**: capacidade de suportar múltiplas requisições simultâneas e operações em lote sem comprometimento da estabilidade.

Esses resultados reforçam a robustez da aplicação backend.

6. CONCLUSÃO

O desenvolvimento do Sistema de Gestão Hospitalar e de Serviços de Saúde (SGHSS) representou uma oportunidade significativa de aplicar conceitos fundamentais da engenharia de software em um contexto real e socialmente relevante.

Durante o projeto, foi possível consolidar conhecimentos sobre arquitetura monolítica, modelagem de dados, segurança da informação e o desenvolvimento de uma APIs RESTful. A utilização de tecnologias como Node.js, Express.js e MySQL, junto à aplicação de boas práticas de codificação e controle de versão, contribuiu para a construção de uma base sólida e escalável.

Entre os principais desafios enfrentados, destacam-se a definição precisa dos requisitos, a modelagem das entidades, a validação dos fluxos de autenticação, implementação de um token de autenticação no sistema e o uso dependências dentro do sistema. Esses pontos exigiram uma atenção especial e ajustes ao longo do desenvolvimento.

Adições futuras ao sistema:

- A implementação de uma interface front-end responsiva para ampliar a usabilidade do sistema.
- A adoção de testes automatizados.
- Criação de diferentes níveis de autorizações de usuários.
- Melhoramento na pesquisa de estoque e suprimentos
- Adição de relatórios em relação aos estoques e suprimentos do hospital
- A evolução para uma arquitetura baseada em microsserviços, visando maior flexibilidade e escalabilidade.
- A integração com sistemas externos, como plataformas de prescrição eletrônica e prontuários digitais.

Conclui-se que o SGHSS é uma solução viável às demandas contemporâneas da área da saúde, que pode ser expandida e adaptada a diferentes realidades hospitalares pelo país.

7. REFERÊNCIAS

DEVMEDIA. Teste de performance com JMeter. Disponível em: <https://www.devmedia.com.br/teste-de-performance-com-jmeter/34621>. Acesso em: 19 out. 2025.

DEVMEDIA. MER e DER: modelagem de bancos de dados. Disponível em: <https://www.devmedia.com.br/mer-e-der-modelagem-de-bancos-de-dados/14332>. Acesso em: 19 out. 2025.

GOOGLE CLOUD. Autenticando usuários com JWT. Disponível em: <https://cloud.google.com/api-gateway/docs/authenticating-users-jwt?hl=pt-br>. Acesso em: 19 out. 2025.

TESRAIL. How to create a test plan. Disponível em: <https://www.testrail.com/blog/create-a-test-plan/>. Acesso em: 19 out. 2025.

VENNGAGE. Diagrama de classe: o que é e como fazer. Disponível em: <https://pt.venngage.com/blog/diagrama-de-classe/>. Acesso em: 19 out. 2025.