



UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO MULTIDISCIPLINAR DE PAU DOS FERROS
CURSO DE BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

João Gabriel Dantas

Desenvolvimento De Uma Api Para Sistemas De Anamnese: Um Estudo De Caso

PAU DOS FERROS

2024

João Gabriel Dantas

Desenvolvimento De Uma Api Para Sistemas De Anamnese: Um Estudo De Caso

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito parcial para obtenção do título de Bacharel em Tecnologia da Informação.

Orientador: Prof. Msc. José Ferdinandy Silva Chagas

PAU DOS FERROS

2024

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei nº 9.279/1996 e Direitos Autorais: Lei nº 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

Dd Dantas, João Gabriel .
 Desenvolvimento de uma API para sistemas de
 anamnese: um estudo de caso / João Gabriel
 Dantas. - 2024.
 59 f. : il.

 Orientador: José Ferdinandy Silva Chagas.
 Monografia (graduação) - Universidade Federal
 Rural do Semi-árido, Curso de Tecnologia da
 Informação, 2024.

 1. Anamnese. 2. Gestão hospitalar. 3.
 Tecnologia da informação. 4. API. 5. Digitalização.
 I. Chagas, José Ferdinandy Silva, orient. II.
 Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade
com AACR2 e os dados fornecidos pelo(a) autor(a).

Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC's) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

João Gabriel Dantas

Desenvolvimento De Uma Api Para Sistemas De Anamnese: Um Estudo De Caso

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito parcial para obtenção do título de Bacharel em Tecnologia da Informação.

Defendida em: **28 / 03 / 2025.**

BANCA EXAMINADORA

Prof. Me. José Ferdinandy Silva Chagas - UFERSA
Presidente

Prof^a. Dr^a. Huliane Medeiros da Silva - UFERSA
Membro Examinador

Prof. Dr. Reudismam Rolim de Sousa - UFERSA
Membro Examinador

Você não consegue ligar os pontos olhando pra frente, você só consegue ligá-los olhando pra trás. Então você tem que confiar que os pontos se ligarão algum dia no futuro. Você tem que confiar em algo – Deus, destino, vida, carma, o que for, pois acreditar que os pontos se ligarão estrada afora, te dará confiança para seguir seu coração, mesmo que ele te leve longe do caminho esperado, isso fará toda a diferença.

Steve Jobs

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por ter me dado saúde, força e sabedoria para chegar até aqui.

Aos meus pais Canindé e Livaldina, pelo amor incondicional, apoio e ensinamentos ao longo da vida. Mesmo nas mudanças de trajetória, sempre fizeram tudo o que foi necessário para me apoiar. Vocês são minha base, meu alicerce.

Ao meu irmão Diego e à minha cunhada Kamilla, por serem companheiros e estarem sempre ao meu lado. Aos meus sobrinhos, Luís e José que com sua alegria e energia me inspiram a ser uma referência para eles e a ser forte em todos os momentos. Vocês trazem alegria e motivação aos meus dias.

Aos meus tios Dario e Higia pela convivência e o amor de pais desde criança, sempre oferecendo amor e suporte. Aos meus tios e primos, que fazem parte da minha história e me deram apoio em diversas formas durante essa caminhada. Aos meus avós Dionísio, Manoel, Maria, Gumércia e Celia, que, mesmo com a distância do tempo, continuam sendo uma inspiração de força e carinho.

Aos meus amigos de toda a vida, José Felix, Emilly, Clezinaldo, Hemily e Luan por estarem presentes nos diversos momentos da minha jornada, seja para apoiar nos desafios e nos momentos difíceis ou celebrar as vitórias de cada um.

Aos meus professores, que compartilharam conhecimentos fundamentais para minha formação. Especialmente, ao meu orientador José Ferdinandy, por fazer tudo que estive ao alcance dele para esse TCC, nunca seria possível sem ele.

Aos meus colegas de curso, Otávio, Izaías, Diego e Kerllanya, com quem convivi durante os últimos anos, pela companhia nesses momentos em que estive longe de casa.

A todos vocês, minha eterna gratidão.

RESUMO

O presente trabalho propõe uma solução tecnológica voltada para a melhoria dos processos de anamnese e atendimento prévio em hospitais, utilizando como estudo de caso o Hospital Regional de Pau dos Ferros, no estado do Rio Grande do Norte. Diante dos desafios enfrentados por instituições de saúde, especialmente durante a pandemia, a pesquisa busca desenvolver um sistema de informação que digitalize e otimize a coleta de dados dos pacientes. O objetivo geral é criar uma API (*Application Programming Interface*) para suportar o processo de anamnese, fundamentada em princípios de desenvolvimento centrados no usuário e em arquitetura de software moderna. A metodologia adotada inclui levantamento de requisitos por observação direta e análise documental, seguido pelo desenvolvimento da API e realização de testes unitários utilizando JUnit e Mockito. Os resultados mostraram uma arquitetura bem estruturada, capaz de facilitar a gestão hospitalar e promover eficiência nos fluxos de trabalho. Além disso, o trabalho destaca as limitações da solução proposta e as experiências adquiridas ao longo do processo de desenvolvimento, enfatizando a necessidade de um ciclo contínuo de aprimoramento no ambiente hospitalar. Acredita-se que a implementação deste sistema possa contribuir significativamente para a agilidade e precisão da anamnese de pacientes, favorecendo uma melhor organização e qualidade no atendimento à saúde.

Palavras-chave: anamnese; gestão hospitalar; tecnologia da informação; api; digitalização;

ABSTRACT

The present work proposes a technological solution aimed at improving anamnesis and preliminary care processes in hospitals, using the Hospital Regional de Pau dos Ferros, located in the state of Rio Grande do Norte, as a case study. In light of the challenges faced by healthcare institutions, especially during the pandemic, this research seeks to develop an information system that digitizes and optimizes patient data collection. The main objective is to create an API (Application Programming Interface) to support the anamnesis process, based on user-centered development principles and modern software architecture. The adopted methodology includes requirements gathering through direct observation and document analysis, followed by API development and the execution of unit tests using JUnit and Mockito. The results demonstrate a well-structured architecture capable of facilitating hospital management and enhancing workflow efficiency. Furthermore, the work highlights the limitations of the proposed solution and the insights gained throughout the development process, emphasizing the need for a continuous improvement cycle within the hospital environment. It is believed that the implementation of this system can significantly contribute to the agility and accuracy of patient triage, promoting better organization and quality in healthcare delivery.

Keywords: *anamnesis. hospital management. information technology. api. digitalization.*

LISTA DE FIGURAS

Figura 1 – Arquitetura do Spring	22
Figura 2 – Arquitetura do DDD	24
Figura 3 – Módulos da Arquitetura	42
Figura 4 – Diagrama de Classes	44
Figura 5 – Casos de Uso Atendimento	46
Figura 6 – Caso de Uso de Verificar dados	46
Figura 7 – Registro de Paciente	47
Figura 8 – Registro de atendimento	48
Figura 9 – Atualizar dados de atendimento	49
Figura 10 – Testes classe paciente	50
Figura 11 – Swagger UI	51
Figura 12 – Swagger UI Complemento	52
Figura 13 – Corpo de uma rota no Swagger	52
Figura 14 – Classe Jwt Service	54
Figura 15 – Classe Jwt Service Complemento	55

LISTA DE QUADROS

Quadro 1 – Requisitos Funcionais	35
Quadro 2 – Requisitos Não Funcionais	38
Quadro 3 – Rotas do Controller da API	45

LISTA DE ABREVIATURAS E SIGLAS

HTTP	<i>Hypertext Transfer Protocol</i>
WWW	<i>World Wide Web</i>
RG	Registro Geral
CPF	Cadastro de Pessoas Físicas
CNS	Cartão Nacional de Saúde
CEP	Código de Endereço Postal
ID	<i>Identity</i>
JPA	<i>Java Persistence API</i>
API	<i>Application Programming Interface</i>
SQL	<i>Structured Query Language</i>
DTO	<i>Data Transfer Object</i>
JWT	<i>JSON Web Tokens</i>
JSON	<i>JavaScript Object Notation</i>
REST	<i>Representational State Transfer</i>
AAA	<i>Arrange, Act, Assert</i>
URL	<i>Uniform Resource Locator</i>

SUMÁRIO

1. INTRODUÇÃO	13
1.1 Justificativa	14
1.2 Objetivos	14
1.2.1 Objetivo Geral	15
1.2.2 Objetivos Específicos.....	15
2. FUNDAMENTAÇÃO TEÓRICA.....	16
2.1 Gestão Hospitalar e Tecnologia da Informação	16
2.2 Anamnese e Processos de Atendimento	17
2.3 Desafios da Implementação de Tecnologias em Saúde.....	18
2.4 API (Application Programming Interface).....	19
2.5 Ferramentas e Frameworks	20
2.5.1 Spring	20
2.5.2 PostgreSQL e Hibernate.....	22
2.5.3 DDD	23
2.5.4 Insomnia	24
2.5.5 Swagger	25
2.5.6 Testes Unitários.....	26
2.5.7 Metodologia Triple A (Arrange, Act, Assert)	26
2.5.8 Modelo de Linguagem Unificada (UML)	27
3. TRABALHOS RELACIONADOS	28
3.1 Sistema para Acompanhantes e Visitantes em Hospital.....	28
3.2 Sistema para Controle de Vacinação no Hospital	28
3.3 Aplicação Móvel para registro de histórico clínico.....	29
4. METODOLOGIA DE PESQUISA.....	30
4.1 Levantamento de Requisitos	30
4.1.1 Observação Direta	31
4.1.2 Análise Documental	31
4.2 Desenvolvimento da API	31
4.2.1 Arquitetura da Solução.....	31
4.2.2 Modelagem de Dados.....	32
4.2.3 Modelagem dos Endpoints	32
4.2.4 Segurança e Controle de Acesso	33
4.3 Validação.....	33
4.4 Documentação	34
5. RESULTADOS E DISCUSSÕES	35
5.1 Modelo da API.....	35
5.1.1 Requisitos Funcionais	35
5.1.2 Requisitos não funcionais.....	38
5.1.3 Arquitetura do Sistema.....	40
5.1.3.1 Model	40
5.1.3.2 Service Layer.....	44
5.1.3.3 Controller	45

5.1.3.4 Rotas.....	45
5.2 Simulação	46
5.3 Testes Unitários.....	49
5.4 Swagger e Insomnia	51
5.5 Acesso ao sistema.....	53
5.6 Limitações do Estudo	56
5.7 Experiência no Processo.....	57
6. CONCLUSÕES E TRABALHOS FUTUROS.....	59
REFERÊNCIAS BIBLIOGRÁFICAS	60

1. INTRODUÇÃO

A Constituição Federal Brasileira de 1988¹ estabelece a saúde como direito de todos e dever do estado. Entre esses deveres estão a criação de políticas sociais e econômicas que busquem reduzir o risco de doença e o acesso universal e igualitário aos serviços de saúde. Segundo Paiva e Teixeira (2014), esse texto dava origem ao processo de criação de um sistema de saúde que atendesse a essas políticas. O que mudaria a forma como o Brasil organiza as suas políticas de saúde (Paiva e Teixeira, 2014).

Porém, a expectativa de profundas mudanças na saúde pública, logo foi confrontada com uma dura realidade. A de um sistema que, mesmo após suas reformas, continuava a enfrentar dificuldades (Paiva e Teixeira, 2014).

Em outras palavras, as mudanças, embora tenham estabelecido marcos importantes, como a afirmação da saúde como um direito social, não esgotaram os problemas. A necessidade de vigilância contínua se tornou evidente, uma vez que a implementação do SUS constitucional e a construção de uma sociedade mais equitativa dependiam de esforços constantes. Essa reflexão crítica sobre o legado da reforma sanitária evidencia que, apesar das iniciativas e progressos alcançados, a luta por uma saúde pública de qualidade e acessível a todos os cidadãos brasileiros ainda se faz urgente e necessária (Paiva e Teixeira, 2014). Dificuldades essas que ainda foram agravadas no final do ano de 2019.

A partir do fim de 2019, a maneira como o mundo organiza suas políticas de saúde passaria a mudar novamente e ter novas preocupações. Segundo Santos (2021), a pandemia de *SARS-CoV-2 (Covid-19)*, que começou em dezembro de 2019, trouxe desafios e causou um impacto em especial aos sistemas de saúde. Esse fenômeno trouxe altos números de infectados com complicações e óbitos (Santos, 2021). Segundo dados do Ministério da Saúde do Brasil², estima-se que o país no início do ano de 2025 acumula mais de 39 milhões de casos confirmados e pouco mais de 700 mil óbitos.

A tecnologia impactou de forma benéfica a população durante a pandemia em vários aspectos, como na produção da vacina, bem como na adaptação das formas de trabalho (Vargo, 2021). Segundo Vargas, Alves e Mrejen (2021), a pandemia de *Covid-19* foi um fator que impulsionou diversos países a aumentar o apoio para o avanço da ciência,

¹ BRASIL. **CONSTITUIÇÃO DA REPÚBLICA FEDERATIVA DO BRASIL DE 1988**. Disponível em: http://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm.

² MINISTÉRIO DA SAÚDE. **Coronavírus Brasil**. Disponível em: <https://covid.saude.gov.br/>.

tecnologia e inovação. Isso envolveu um maior financiamento de pesquisas, com a cooperação entre países para esse fim. No momento de crise mundial, as pesquisas e o desenvolvimento em saúde estava com prioridade principalmente na busca por vacinas (Vargas, Alves e Mrejen, 2021).

Levando em conta esse contexto histórico como um todo e incluindo também o contexto de pandemia, é importante preparar os ambientes hospitalares para novos desafios. É necessário aplicar tecnologias e buscar melhoria na qualidade e no tempo dos atendimentos.

Este trabalho busca oferecer uma proposta de suporte tecnológico para melhoria dos processos hospitalares de anamnese. E foi desenvolvido considerando como estudo o caso os recursos e processos de anamnese do Hospital Regional de Pau dos Ferros (HRPF), no estado do Rio Grande do Norte.

1.1 Justificativa

Santos (2021) diz que desde sempre e, durante a pandemia isso foi agravado, os hospitais precisam de uma reestruturação nos seus serviços de saúde para a alta demanda. Entre as principais necessidades, pode-se destacar a organização dos recursos humanos, insumos e tecnologias (Santos, 2021). Assim, esta necessidade impactou instituições de saúde na busca por avançar tecnologicamente. Camilo (2019), destaca para a importância de sistemas de informação em espaços profissionais gerais, inclusive hospitalares. Segundo o autor, em ambientes empresariais, têm se buscado usar a tecnologia para melhorar o trabalho dos colaboradores. No caso de ambientes hospitalares, o serviço e os dados coletados necessitam ser extremamente precisos e seguros. Além disso, é importante garantir a integridade dos mesmos e que o fluxo de informações seja feito em tempo real (Camilo, 2019).

Segundo Fonseca (2019), as filas nos hospitais estão cada vez maiores, o que aumenta o tempo de atendimento e gera uma maior dificuldade não só para ser atendido, mas para marcar novas consultas. Nesse contexto, é importante buscar recursos que beneficiem o processo de anamnese. Este processo trata-se do atendimento prévio e coleta dos dados e queixas principais do paciente. No HRPF, o processo é realizado no setor de entrada para emergência. Um sistema que apoie e automatize a anamnese pode reduzir esse tempo de espera entre os atendimentos (Fonseca, 2019).

No HRPF, o processo de anamnese ainda é realizado em fichas de papel, deixando dados sensíveis registrados de maneira inadequada, não automatizados e inseguros. Esse processo manual colabora também para uma lentidão do atendimento e não garante a conformidade dos dados. Assim, surge a proposta de buscar uma solução tecnológica para o Hospital Regional de Pau dos Ferros. Buscando auxiliar os colaboradores e agilizar o atendimento.

1.2 Objetivos

A seção a seguir descreve os objetivos gerais e específicos do trabalho proposto como forma de delinear o escopo do que será realizado.

1.2.1 Objetivo Geral

O objetivo geral do trabalho é desenvolver uma API para sistemas de triagem hospitalar, apresentando uma solução tecnológica que apoie o processo de anamnese. A proposta consiste em um modelo de aplicação distribuída baseada em API (*Application Programming Interface*), que pode servir como base para o desenvolvimento de sistemas de triagem e atendimento prévio em diferentes contextos ou dispositivos.

1.2.2 Objetivos Específicos

- Modelar um sistema de informação para anamnese;
- Implementar uma *API* baseada no modelo;
- Implementar testes automatizados;
- Implementar criptografia de dados, controle de acesso baseado em papeis;
- Testar a *API* no contexto de uma interface web.

Os objetivos foram definidos delimitando o escopo da pesquisa e do desenvolvimento da solução proposta. O foco principal é a criação de uma *API* distribuída que sirva como base para sistemas de anamnese e atendimento prévio em ambientes hospitalares. Dessa forma, a pesquisa busca contribuir para o desenvolvimento de soluções tecnológicas que aprimoram a confiabilidade, segurança e interoperabilidade no atendimento a pacientes, fortalecendo a gestão hospitalar e a qualidade dos serviços prestados.

2. FUNDAMENTAÇÃO TEÓRICA

A seguir serão apresentados os principais conceitos relacionados à problemática do trabalho e as tecnologias envolvidas no desenvolvimento da solução. O objetivo deste capítulo é fornecer os fundamentos necessários para complementar o entendimento do trabalho que foi realizado.

2.1 Gestão Hospitalar e Tecnologia da Informação

Segundo Muhammad *et al* (2024), os sistemas de informação são cruciais para a gestão hospitalar, pois auxiliam a ter uma melhora da eficiência e fluxos de trabalho. O aumento do uso desses sistemas também é diretamente proporcional à melhoria da qualidade do atendimento e a satisfação do paciente. Obviamente, a experiência do paciente é o objetivo final da gestão hospitalar (Muhammad *et al*, 2024).

Os sistemas possibilitam acesso rápido e seguro das informações dos pacientes e, com o passar dos anos, esse volume e complexidade desses dados só aumentam. Esse acesso rápido, ajuda a tomada de decisões clínicas também mais rápidas. (Muhammad *et al*, 2024).

Esses sistemas ajudam auxiliando em um gerenciamento eficaz dos recursos hospitalares, seja de materiais ou alocação de profissionais. O que ajuda a manter a saúde financeira da instituição. Os sistemas hospitalares não apenas facilitam a coleta e análise de dados, mas também asseguram que esses dados sejam precisos e atualizados. Esses fatores garantem que as decisões hospitalares sejam tomadas baseadas em dados reais, promovendo uma tomada de decisão também mais livre de grandes riscos. Essa abordagem é essencial para melhorias contínuas nos serviços oferecidos, permitindo decisões estratégicas mais informadas e fundamentadas (Muhammad *et al*, 2024).

Segundo Faeni e Octaviany (2024), a gestão hospitalar é fundamental para assegurar que as instituições de saúde funcionem como se espera. A implementação de Sistemas de Informação atua como uma ponte que relaciona o uso de dados e a tomada de decisão. Também chamam a atenção para a questão da coleta e análise de dados e alocação de recursos. Segundo os autores, esses sistemas facilitam a coleta e a análise de dados. Além disso, também melhoram a comunicação entre os diversos setores da saúde, como recursos humanos, finanças e serviços médicos (Faeni e Octaviany, 2024).

Sistemas de informação utilizam tecnologia baseada na internet para oferecer serviços de saúde de forma mais acessível e eficiente. A digitalização permite que pacientes acessem informações sobre saúde rapidamente, agendem consultas, e até realizem teleconsultas, reduzindo as barreiras físicas e melhorando a experiência do usuário (Faeni e Octaviany, 2024).

Os SI inovam a forma como os serviços de saúde são entregues, mas também diversas unidades a manterem uma comunicação, como entidades, como hospitais, centros de saúde, universidades, e empresas de telecomunicações. Essa colaboração é importante para melhorar a infraestrutura de saúde e reduzir custos, permitindo um acesso universal às informações e serviços de saúde (Faeni e Octaviany, 2024).

Nesse contexto, a anamnese desempenha um papel importante no processo de atendimento, pois representa o primeiro contato estruturado entre o paciente e o profissional de saúde. A informatização desse procedimento pode aumentar a precisão das informações coletadas, padronizar a anamnese e contribuir para diagnósticos mais assertivos.

2.2 Anamnese e Processos de Atendimento

Na gestão hospitalar dentre os principais processos está a anamnese. Segundo De Resende (2022), a anamnese é um processo fundamental no contexto clínico, pois é usado na coleta de informações que auxiliam no diagnóstico médico. Consiste em uma série de etapas que buscam explorar diversos aspectos da vida do paciente em relação à sua condição de saúde. A anamnese não faz referência apenas à identificação de sintomas, mas também busca entender as repercussões da doença na vida cotidiana do paciente, incluindo impactos em sua família e ambiente de trabalho (De Resende, 2022).

Além disso, esse processo busca experiências semelhantes passadas pelo próprio paciente ou por familiares, permitindo que o médico tenha uma visão mais ampla do histórico de saúde e predisposição genética daquele paciente (De Resende, 2022).

Outros elementos importantes incluem a identificação de preocupações atuais do paciente e suas percepções sobre a atribuição de causas à sua condição. O exame do estado emocional do paciente em relação à doença, bem como suas percepções sobre o tratamento recebido, incluindo tanto as dificuldades quanto os benefícios percebidos, também são componentes que ajudam a construir um quadro mais completo (De Resende, 2022).

Por fim, a anamnese deve abordar como o paciente vê o médico e a equipe de saúde, assim como a qualidade dos serviços prestados. As questões que o paciente considera relevantes também devem ser ouvidas (De Resende, 2022).

A informatização da anamnese não apenas padroniza a coleta de informações essenciais, mas também representa um avanço na qualidade do atendimento, permitindo que os profissionais de saúde tomem decisões mais assertivas com base em dados estruturados. No entanto, a implementação de tecnologias no setor hospitalar enfrenta desafios significativos, desde a resistência dos profissionais à adoção de novos sistemas até a necessidade de garantir a segurança e privacidade das informações dos pacientes.

2.3 Desafios da Implementação de Tecnologias em Saúde

Segundo Muhammad *et al* (2024), apesar de suas vantagens, a implementação desses sistemas em ambientes hospitalares não está livre de desafios. Problemas relacionados à usabilidade e à integração com sistema vão sempre estar presentes. Por isso, chama atenção para a importância de um plano de implementação que inclua treinamentos adequados para os usuários. Entre os problemas mais comuns, temos: Insatisfação dos usuários com as interfaces do sistema, dificuldades de operação por parte de funcionários mais velhos, mecanismos de backup de dados inadequados e integrações limitadas com plataformas externas (Muhammad *et al*, 2024).

Segundo Faeni e Octaviany (2024), um grande obstáculo na implementação de sistemas de informação é a resistência por parte dos profissionais de saúde. Muitos colaboradores, por costume a métodos tradicionais de trabalho, podem relutar em adotar novas tecnologias (Faeni e Octaviany, 2024). Ainda, segundo Maara et al. (2024), é difícil garantir que todos os grupos de pacientes sejam representados e que eles continuem engajados com a tecnologia com o passar do tempo. Podendo gerar uma má adesão ao uso de dispositivos, especialmente entre populações mais velhas ou com comorbidades, podendo levar a dados incorretos (Maara et al, 2024).

Segundo Faeni e Octaviany (2024), os sistemas também requerem que os usuários sejam treinados adequadamente. A falta de treinamento pode resultar em subutilização dos sistemas ou em erros no uso, o que compromete a eficácia da assistência ao paciente. Esses fatores acarretam em um alto custo de implementação. As instituições de saúde precisam justificar os custos e garantir que vai haver o retorno sobre o investimento a longo prazo. Esse fator pode

limitar hospitais menores ou com orçamento restrito a implementarem as tecnologias (Faeni e Octaviany, 2024). Ainda, segundo Maara et al. (2024), a necessidade de treinamento nos dispositivos e suporte técnico é crucial para assegurar que os pacientes possam usar as tecnologias sem problemas, evitando aumentar as desigualdades no acesso às tecnologias. (Maara et al, 2024).

Diante desses desafios, a adoção de APIs na área da saúde surge como uma solução para facilitar a integração de novos sistemas, garantindo maior interoperabilidade entre diferentes plataformas. Uma API bem estruturada permite que hospitais otimizem seus processos sem a necessidade de substituir sistemas legados, reduzindo custos e o impacto da curva de aprendizado dos profissionais.

2.4 API (*Application Programming Interface*)

Dentro da complexidade dos processos hospitalares é importante buscar a interoperabilidade entre sistemas e equipamentos. Assim, foi buscada uma solução tecnológica para a problemática baseada em API. Segundo Ofoeda, Boateng e Effah (2019), as API (*Application Programming Interfaces*) são um um facilitador da comunicação entre dois programas. Essa tecnologia se tornou essencial para o desenvolvimento de software e seu ecossistema, sendo uma das maiores inovações da área. Se tornaram um caminho ideal a seguir e uma tecnologia para se basear ao buscar desenvolver grandes softwares que buscam escalabilidade. Softwares com base nessa tecnologia buscam facilitar a interconexão de pessoas e sistemas (Ofoeda, Boateng e Effah 2019).

Segundo Garimilla (2024), as API permitem que diferentes sistemas e aplicações se comuniquem da melhor maneira possível, independentemente das tecnologias utilizadas. Essa capacidade é essencial em um mundo onde a eficiência na troca de dados impacta diretamente na qualidade dos serviços, como é o caso na área da saúde. Por exemplo, sistemas de saúde que utilizam APIs podem integrar dados de várias fontes, criando registros mais completos e melhorando a coordenação dos cuidados ao paciente. Essa possibilidade ajuda também na troca de dados entre as diversas unidades presentes no procedimento de atendimento de saúde (Garimilla, 2024).

Quando se lida com softwares complexos, é importante que os componentes possam ser escalados independentemente para gerenciar variações na carga de trabalho. Ou seja, os componentes precisam dar a possibilidade de crescer em questão de recurso computacional com

uma maior solicitação, e diminuir com uma menor. *APIs* consideradas eficazes são aquelas que facilitam a adoção de arquiteturas modulares, como a arquitetura de microsserviços, o que também se relaciona a escalabilidade. Essa flexibilidade é um diferencial, pois permite a adaptação rápida a novas demandas do mercado e manutenção de um alto nível de serviço. Isso é verdade pois permite com que os componentes da aplicação sejam alterados individualmente (Garimilla, 2024).

Para o desenvolvimento de *APIs*, existem algumas práticas que são recomendadas. O uso de convenções de nomenclatura é importante. Adoção de nomes descritivos, que ditem o que o parâmetro está fazendo, para facilitar a compreensão. Usar *URLs* para controlar números de versão, por exemplo, para uma *API* de usuários: `/v1/users`. Essa convenção permite gerenciar as mudanças ao longo do tempo (Garimilla, 2024).

Outro ponto importante, é a validação das entradas para garantir a integridade dos dados e prevenir problemas de segurança. Um fator crucial em sistemas hospitalares que envolvem dados críticos para o atendimento adequado. O uso de formatos padrões, como *JSON* ou *XML*, para troca de dados é recomendado devido à sua compatibilidade e facilidade de uso. Além disso, é crucial fornecer mensagens de erro significativas e utilizar códigos de status *HTTP* apropriados. Essa prática melhora a experiência do desenvolvedor e facilita a depuração. Sobre arquitetura de *API*, foi usada o *REST*, que é ideal para operações *CRUD* simples (Garimilla, 2024).

É importante conhecer sobre o ciclo de vida da *API*, que envolve toda a fase de planejamento, design, desenvolvimento, testes, lançamento e eventual descontinuação da *API*. O conhecimento e adaptação desse ciclo é fundamental para garantir que a *API* permaneça funcional, adaptando-se às mudanças nas necessidades dos usuários e nas exigências de segurança (Garimilla, 2024).

2.5 Ferramentas e Frameworks

Nesta seção serão destacadas as principais ferramentas e frameworks utilizadas para a construção da solução. O que envolve o utilizado desde a ideia inicial, passando pelo desenvolvimento, até a fase de validação.

2.5.1 Spring

Como auxílio para o melhor desenvolvimento na linguagem Java, foi utilizado o *framework Spring*. Segundo a documentação oficial do Spring³, o framework facilita e torna mais fácil a programação em Java e é o *framework* mais popular para Java hoje. O Spring foi criado por volta de 2003 como uma forma de responder à complexidade do Java até então. Entre as vantagens buscadas estão que o framework provê um ambiente *serveless*. O spring permite ao desenvolvedor focar na escrita do código enquanto gerencia o servidor e suas especificidades.

Além disso, tem a vantagem de facilitar o desenvolvimento de aplicações *web*, justamente por eliminar essa parte da configuração para rodar o programa. Em resumo, a adição do *Spring* ao desenvolvimento Java facilita o processo de desenvolvimento da aplicação. Usando o *Spring*, temos uma cobertura alta de uso em diversas situações de desenvolvimento de aplicações

O chamado “Spring” faz referência a todos os projetos do spring, que envolvem *Spring Cloud*, *Spring Boot*, *Spring AI*, *Spring Security*, entre outros. Cada um com sua função e especificidades. Entre os princípios que o spring segue, pensados desde sua criação estão:

- Prover escolha em todos os níveis: O *Spring framework* busca permitir que os elementos possam ser alterados unitariamente sem alterações de outros. Dessa forma, decisões de estrutura podem ser tomadas em qualquer momento do desenvolvimento. Por exemplo, mudar a persistência, sem alterar o código.
- Permitir diversas perspectivas de necessidades da aplicação, permitindo que o desenvolvedor decida como as coisas devem ser feitas.
- Forte compatibilidade com versões antigas. A manutenção e evolução no *Spring* é feita com bastante cuidado, buscando não realizar grandes mudanças, para não prejudicar versões do Java e bibliotecas de terceiros que dependam do Spring.
- Design de *API* intuitivo que permanece dessa forma ao longo dos anos

Esses princípios são seguidos com o principal objetivo de manter a facilidade e a usabilidade da ferramenta (SPRING, 2019).

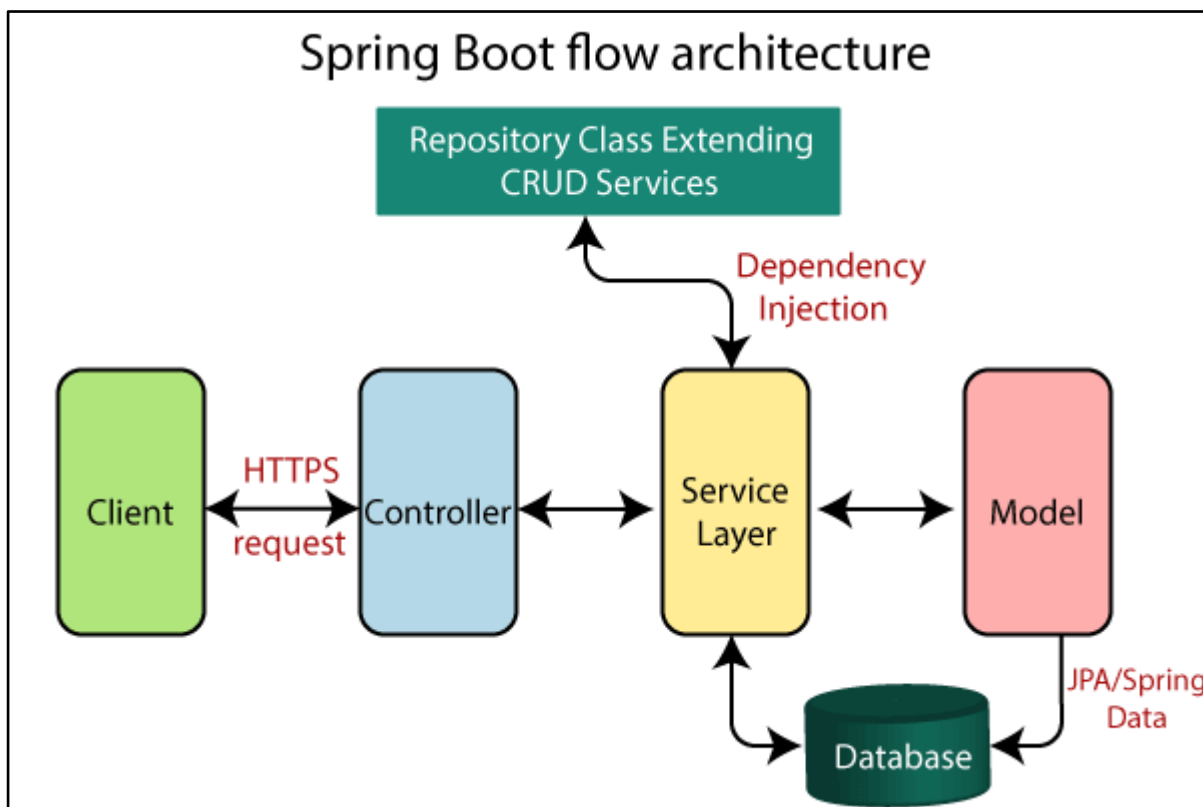
Como apresentado na Figura 1, a arquitetura do Spring é dividida em *layers* que se comunicam hierarquicamente, cada uma com sua função específica:

- Camada de Apresentação: Recebe e valida as entradas (requisições *HTTP*) e as envia para a camada;

³ SPRING. spring.io. Disponível em: <<https://spring.io/>>. Acesso em jan. 2025

- Camada de negócio: Responsável pela lógica de negócio, nessa camada estão contidas as classes, cálculos, transformações de dados e validações;
- Camada de persistência: Responsável por traduzir a classe e as opções da lógica de negócio para o banco de dados;
- Camada de Banco de Dados: Responsável por realizar as operações *CRUD* (*Create, Retrieve, Update, Delete*).

Figura 1 - Arquitetura do Spring



Fonte: <https://www.tpointtech.com/spring-boot-architecture>

2.5.2 PostgreSQL e Hibernate

Segundo sua documentação oficial⁴, o PostgreSQL é um sistema de banco de dados objeto-relacional de código aberto, amplamente usado. O sistema é mantido através de contribuintes como a AWS, o NTT Group, Fujitsu, Google, Red Hat e diversas outras.

O PostgreSQL possui diversas funcionalidades que ajudam o desenvolvedor a desenvolver suas aplicações, ajudando a administrar e proteger os dados e seu fluxo. É

⁴ PostgreSQL: Documentation. Disponível em: <<https://www.postgresql.org/docs/>>.

altamente personalizável, compatível com diversas linguagens de programação, como o Java. Além de ser gratuito e de código e documentação abertos.

O projeto POSTGRES, foi liderado pelo professor Michael Stonebraker e começou em 1986. O primeiro sistema surgiu em 1987. O projeto foi amplamente utilizado em aplicações de pesquisa e produção, incluindo bancos de dados médicos e de análise financeira. Em 1992, tornou-se o principal gerenciador de dados do projeto Sequoia 2000. Em 1994, Andrew Yu e Jolly Chen adicionaram um interpretador SQL ao sistema, lançando o Postgres95 como um projeto de código aberto. Esta versão trouxe melhorias de desempenho, suporte a SQL e um novo programa interativo (psql). Em 1996, o nome foi alterado para PostgreSQL para refletir sua evolução, e a numeração retomou a sequência iniciada pelo projeto original, começando na versão 6.0. O desenvolvimento passou a focar na adição de novos recursos e aprimoramentos, embora a tradição de chamar o sistema de "Postgres" tenha permanecido.

O SQL consiste em uma sequência de comandos, cada um formado por uma série de tokens e finalizado por um ponto e vírgula. O fim do fluxo de entrada também encerra um comando.

Para mapeamento foi utilizado o *Object Relational Mapper* (ORM) que, segundo Bonteanu (2024), é uma abordagem que permite mapear classes Java para tabelas de banco de dados, criando uma camada que oculta essa complexidade do desenvolvedor. Isso permite que os programadores se concentrem na lógica de programação, em vez de escrever manualmente consultas SQL e mapear atributos de classes para colunas de tabelas (Bonteanu, 2024).

O ORM utilizado foi o Hibernate que, ainda segundo Bonteanu (2024), é um dos frameworks ORM mais utilizados em Java. Facilita a interação com bancos de dados relacionais. Ele oferece mecanismos para gerenciar a persistência de dados, transações e consultas, permitindo que os desenvolvedores trabalhem com objetos em vez de lidar diretamente com SQL (Bonteanu, 2024).

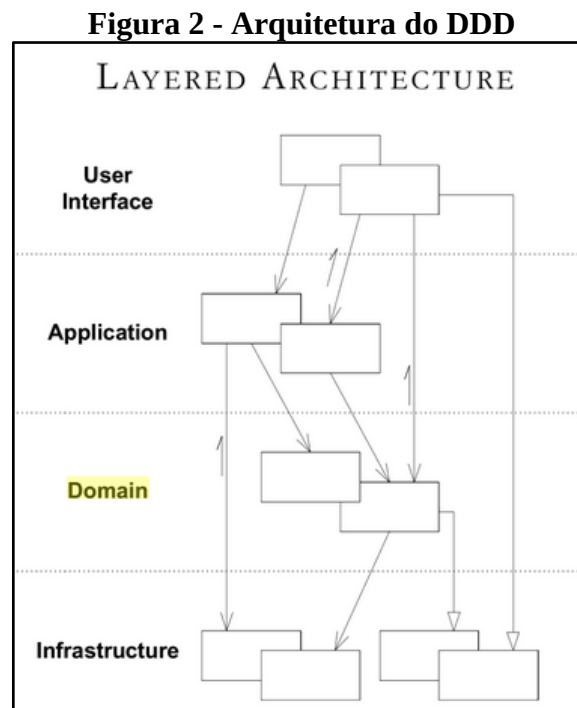
2.5.3 DDD

Segundo Millett e Tune (2015), o *DDD* é uma filosofia de desenvolvimento que busca orientar desenvolvedores a construir e manter softwares. O *DDD* possui um conjunto de práticas, padrões e princípios que buscam esse fim (Millett e Tune, 2015).

Segundo Evans (2004), entre os passos importantes do desenvolvimento de software baseado em *DDD* está isolar o domínio da aplicação. O domínio da aplicação é parte do software

que resolve o problema proposto, geralmente representa apenas uma pequena parte do código. É importante isolar esse domínio do restante da aplicação (Evans, 2004).

Geralmente essa filosofia busca dividir a aplicação em uma arquitetura de layers. Entre essas camadas estão: A interface de usuário, camada de aplicação, camada de domínio e camada de infraestrutura do sistema (Evans, 2004). Como apresentado na Figura 2:



Fonte: Evans, 2004.

Assim, a abordagem do DDD proporciona uma estrutura clara e modular para o desenvolvimento de sistemas complexos, permitindo que a lógica de negócio seja melhor organizada e mantida ao longo do tempo. Ao dividir a aplicação em camadas bem definidas, essa arquitetura facilita a reutilização de componentes, a evolução do sistema e a integração com outras tecnologias, fatores essenciais para garantir a escalabilidade e a manutenção de sistemas hospitalares, que demandam alta confiabilidade e segurança. Essa estrutura facilita a criação de documentações mais claras e alinhadas à realidade do hospital, tornando a comunicação entre desenvolvedores e profissionais da saúde mais eficiente.

2.5.4 Insomnia

Segundo sua documentação⁵, é uma ferramenta desktop *open source* que facilita o teste e experimentação de APIs, permitindo testar rotas e requisições *HTTP*. Através da ferramenta, é possível depurar APIs em diversos protocolos, projetar usando *OpenAPI*, realizar testes nativos, simular APIs com servidores *mock* e integrar pipelines *CI/CD* via linha de comando. Além disso, oferece armazenamento local, sincronização em nuvem com criptografia e integração com *Git* para versionamento.

Como cliente de API, o Insomnia permite enviar requisições sem necessidade de código ou uso de *URL*, tornando mais simples a recuperação e envio de dados. As requisições são estruturadas com elementos como *URL*, tipo de solicitação (*GET*, *POST*, *PATCH*, *PUT*, *DELETE*) e autenticação, quando necessária. Neste contexto, os endpoints são *URLs* específicas acessadas para comunicação com a API. O Insomnia suporta diferentes tipos de requisições, incluindo *HTTP*, *gRPC* e *GraphQL*, proporcionando flexibilidade para desenvolvedores trabalharem com diferentes padrões. O Insomnia foi utilizado em paralelo com o Swagger para buscar uma documentação clara e que também exibisse algum tipo de interface.

2.5.5 Swagger

Segundo Casas (2021), o Swagger é uma estrutura amplamente utilizada para descrever serviços web e suas funcionalidades, proporcionando um contrato que permite a interoperabilidade entre diferentes clientes e serviços. Ele suporta a definição de *URLs*, métodos *HTTP*, parâmetros de entrada e formatos de resposta, além de permitir a documentação interativa por meio do Swagger *UI*, que gera uma interface para visualizar e testar as especificações da API (Casas, 2021).

O Swagger, sendo um dos formatos mais populares para a documentação de APIs, se destaca por sua comunidade ativa e suporte contínuo. O Swagger se tornou uma escolha preferida, especialmente em ambientes empresariais, por sua habilidade de atender a requisitos desejáveis e sua versatilidade na geração de documentação e código a partir das especificações. Isso porque, o Swagger se destaca pela flexibilidade nos formatos suportados. Ele permite a criação de documentos em *JSON* e *YAML*, oferecendo aos desenvolvedores a liberdade de escolher o formato que melhor se adapta às suas necessidades e preferências. Essa versatilidade torna o Swagger uma opção atraente para diversos perfis de projeto (Casas, 2021).

⁵ Insomnia Docs. Disponível em: <<https://docs.insomnia.rest/>>.

Entretanto, existem lacunas nas funcionalidades de geração automática de especificações a partir de modelos, sugerindo que futuras versões poderiam integrar essa capacidade para facilitar ainda mais o desenvolvimento de *APIs*. Ou seja, embora as ferramentas oferecidas pelo projeto Swagger sejam valiosas, ainda há espaço para melhoria, especialmente na validação automática de dados de entrada e na compatibilidade com a versão mais recente do *OpenAPI*. (Casas, 2021). Além da documentação, para o bom funcionamento e confiabilidade da API, também se torna importante a testagem da aplicação.

2.5.6 Testes Unitários

Segundo Hunt e Thomas (2003), os testes unitários são essenciais para detectar erros e melhorar o sistema como um todo. Melhoram a eficiência e a fatoração do código. (Hunt e Thomas, 2003).

Segundo Campopiano (2022), teste unitário é uma técnica crítica que oferece acompanhamento imediato durante o processo de codificação. Permitindo que os desenvolvedores verifiquem a precisão e a eficiência de seu código, essa abordagem ajuda a identificar erros rapidamente. Dessa forma, os testes unitários se tornam uma ferramenta importante para a garantia da qualidade, ajudando a prevenir falhas lógicas que poderiam passar despercebidas no código. (Campopiano, 2022).

Além de seu papel preventivo, os testes unitários trazem diversos benefícios ao processo de desenvolvimento. Eles não apenas facilitam a criação de soluções robustas, mas também oferecem uma base para outros tipos de testes, como os testes de integração e os testes ponta a ponta. A prática de realizar testes unitários contribui para aumentar a confiabilidade e a manutenção do software, além de diminuir o tempo exigido para a depuração de falhas. Dessa maneira, os testes são vistos como peças fundamentais para garantir a qualidade do produto final. A testagem de software como um todo vem se tornando uma das principais medidas para facilitar as partes mais manuais e repetitivas do desenvolvimento de software. Os testes unitários possibilitam verificar praticamente todos os trechos de código. (Campopiano, 2022).

2.5.7 Metodologia Triple A (*Arrange, Act, Assert*)

Segundo Campopiano (2022), o padrão de teste "AAA" (*Arrange, Act, Assert*) é uma estrutura que auxilia na organização da escrita de testes unitários. Ele divide o teste em três

etapas distintas. Cada uma dessas etapas desempenha um papel no processo de teste, permitindo uma compreensão mais fácil tanto para quem escreve quanto para quem lê o código. (Campopiano, 2022).

Na etapa "*Arrange*", o objetivo é preparar o cenário do teste. Isso envolve a inicialização de variáveis, a configuração de *mocks* e a definição de todos os elementos necessários para simular o ambiente em que o teste ocorrerá. A qualidade da configuração realizada nesta fase é crucial, pois cenários muito específicos podem levar a erros e resultados enganosos, como falsos positivos. Portanto, é recomendado que o desenvolvedor use casos gerais para garantir que os testes são robustos e representativos. (Campopiano, 2022).

A etapa "*Act*" refere-se à ação em si, onde o desenvolvedor executa o código que está sendo testado. Neste ponto, se trata de chamar uma função ou realizar uma ação específica, como clicar em um botão na interface. Essa fase é essencial para observar como o sistema se comporta em resposta à ação definida. (Campopiano, 2022).

Por fim, na etapa "*Assert*", o desenvolvedor faz verificações sobre o resultado esperado do código após a execução. Essa fase é onde as expectativas são validadas, verificando se as funções foram chamadas corretamente, se as variáveis apresentaram os valores esperados e se todas as ações desejadas foram concluídas. Essa estrutura não apenas fornece clareza ao teste, mas também garante que o código passou por uma análise crítica e sistemática, promovendo a qualidade e a confiabilidade do software (Campopiano, 2022).

2.5.8 Modelo de Linguagem Unificada (UML)

Para a modelagem de dados e das classes, foi utilizada a linguagem de modelagem unificada (*UML*). Segundo KOC, Hatice *et al* (2021), os diagramas UML são reconhecidos como padrão para representar modelos de design orientados a objetos. O que facilita a identificação de requisitos e escopos de sistemas e aplicações através de representações visuais. Entre os diversos tipos de diagramas *UML*, temos diagramas de classe, diagramas de atividades, diagramas de sequência, entre outros, cada um com suas funções específicas no processo de design e modelagem. Os diagramas de classe são os mais utilizados nas publicações revisadas, destacando seu papel essencial no processo de design e modelagem na engenharia de software (KOC, Hatice, 2021).

3. TRABALHOS RELACIONADOS

Neste capítulo são apresentados trabalhos relacionados que discutem tecnologias em saúde apresentando estudos semelhantes a problemática abordada, bem como propostas de soluções no mesmo contexto.

3.1 Sistema para Acompanhantes e Visitantes em Hospital

Neste trabalho, Silva (2022) busca uma solução para melhorar o controle de acesso em instituições de saúde. Solução essa que foi feita através de uma aplicação *web* em conjunto com uma aplicação *mobile*.

Essa aplicação permite o agendamento de visitas pelos usuários acompanhantes de pacientes através de um perfil na aplicação. Para o lado administrativo do hospital, permite, principalmente, o controle dos agendamentos e das informações dos visitantes, podendo-se aceitar ou rejeitar uma visita. Essas funções permitem à instituição de saúde manter um maior controle de acesso, como forma de manter a organização e a segurança da entrada e saída de pessoas do hospital.

Esse trabalho ajudou a entender um pouco mais sobre a necessidade desse tipo de solução tecnológica em instituições de saúde. E a buscar inspirações para as funções.

Esse trabalho teve uma relação direta na inspiração da modelagem das entidades dos pacientes no presente trabalho. Ambos envolvem o cadastro de pessoas, com dados semelhantes, então entender como foi desenvolvida essa ideia em outra pesquisa foi fundamental para a modelagem do sistema, apesar de serem premissas diferentes.

3.2 Sistema para Controle de Vacinação no Hospital

Neste trabalho, Camilo (2019), buscou uma solução para armazenamento dos dados de vacinação em hospitais. Esse sistema permite ao agente realizar o cadastro, atualização e verificação dos dados de vacinas dos pacientes. Essas informações precisam ser mantidas seguras e a integridade dos dados precisam ser mantidas. Essas informações podem substituir os cartões de vacina em caso de perda por qualquer natureza.

Essa solução tecnológica foi aplicada como uma aplicação *web*, os agentes e profissionais de saúde do hospital podem verificar e cadastrar essas informações. Pelo lado do paciente, podem ser enviadas as informações do cartão de vacina físico, e verificadas as vacinas

no hospital. Dos dois lados, essas funcionalidades estão disponíveis para os níveis das contas dos usuários.

A relação do presente trabalho com este se dá pela similaridade da modelagem dos dados e de como as informações são armazenadas no sistema. Camilo também segue um modelo de cadastro de dados através de um simples *CRUD*, o que torna o sistema simples porém funcional. Além disso, em seu trabalho o autor também busca uma solução de digitalização de registro de dados para substituir fichas de papel.

3.3 Aplicação Móvel para registro de histórico clínico

No trabalho, De Resende (2022) propõe o desenvolvimento de uma aplicação móvel que permite aos usuários registrar periodicamente seu histórico clínico, incluindo a anotação de sintomas e outros dados relevantes sobre sua saúde.

Uma das características diferenciais da solução foi a utilização da tecnologia de *Cloud Computing*, que garante a armazenagem segura e acessível de informações. Ao fazer uso da nuvem, os dados coletados podem ser analisados posteriormente. O que proporciona ao médico uma visão mais completa do histórico de saúde do paciente, assim como padrões e tendências que podem interferir no diagnóstico. Com a análise de dados históricos, o médico pode diagnosticar de forma mais eficaz, mas o paciente também pode aprender sobre suas condições de saúde e alterar seus hábitos.

A solução busca se adequar às exigências do Regulamento Geral sobre a Proteção de Dados (GDPR), lei Europeia, semelhante a nossa Lei Geral de Proteção de Dados (LGPD) brasileira. Lei essa que busca proteger os direitos fundamentais à liberdade e à privacidade do usuário. Os dados coletados são de acesso exclusivo do usuário, e somente este pode autorizar o compartilhamento com terceiros, assegurando um controle maior sobre suas informações pessoais. Isso foi feito de forma a buscar reforçar a confiança do paciente na saúde digital, como valorizar a ética no manejo de dados sensíveis.

O presente trabalho não desenvolveu, pelo tempo e necessidade do estudo técnico, as regulamentações da LGPD, tampouco utiliza tecnologia de *cloud*. Apesar disso, se relaciona com o trabalho do De Resende, por este se tratar de uma solução para registro de histórico clínico. A inspiração neste trabalho foi usada para ideias e modelagem do registro de atendimentos, que segue a mesma premissa do histórico clínico do trabalho citado.

4. METODOLOGIA DE PESQUISA

Este trabalho caracteriza-se como uma pesquisa aplicada, pois visa o desenvolvimento de um artefato tecnológico (uma *API* para interoperabilidade hospitalar) com o objetivo de modernizar e integrar sistemas hospitalares. O estudo de caso foi usado como metodologia.

Segundo Ventura (2007), o estudo de caso representa uma modalidade de pesquisa reconhecida por sua importância e versatilidade em diversas áreas do conhecimento. É uma ferramenta essencial que possibilita uma análise profunda e rica em detalhes sobre fenômenos individuais ou coletivos. Este tipo de pesquisa é caracterizado por sua flexibilidade, podendo adotar abordagens tanto qualitativas quanto quantitativas, dependendo dos objetivos do estudo (Ventura, 2007).

As principais vantagens do estudo de caso incluem sua capacidade de fornecer uma compreensão profunda e detalhada de um fenômeno específico, levando em consideração sua complexidade e contexto. Um estudo de caso significativo deve ser elaborado de maneira atrativa e completa, integrando evidências e diferentes perspectivas que enriquecem a análise (Ventura, 2007).

No entanto, o estudo de caso também apresenta limitações. Como a necessidade de cuidado ao generalizar os achados de um único ou poucos casos, uma vez que eles podem não representar a totalidade de um fenômeno mais amplo. Além disso, o estudo de caso não segue um roteiro rígido, o que pode levar a desafios na sistematização e replicação dos resultados. É imprescindível que o pesquisador mantenha rigor científico em sua análise e considere as implicações mais amplas dos casos estudados (Ventura, 2007).

4.1 Levantamento de Requisitos

O levantamento de requisitos constituiu-se como uma etapa fundamental no desenvolvimento deste projeto e foi realizado através de uma visita técnica ao Hospital Regional de Pau dos Ferros, conduzida por pesquisadores do grupo LISA - Laboratório de Inovação em Software e Automação da UFERSA, localizado no Centro Multidisciplinar de Pau dos Ferros. Essa visita permitiu a coleta de informações valiosas sobre as rotinas hospitalares e levantou a necessidade de digitalização dos processos existentes, motivando o desenvolvimento da solução proposta. O levantamento de requisitos seguiu um protocolo estruturado, dividido em etapas.

4.1.1 Observação Direta

A visita técnica ao hospital propiciou uma análise detalhada das rotinas e fluxos operacionais associados à anamnese e ao registro de informações dos pacientes, permitindo a identificação de ineficiências nos processos atuais;

Durante essa observação, foram detectadas práticas informais que evidenciam as dificuldades enfrentadas tanto pelos profissionais de saúde quanto pelos pacientes, decorrentes do uso de registros manuais. Isso ressalta a urgência de implementar um sistema digital que otimize essas interações.

4.1.2 Análise Documental

A análise documental foi realizada com base nos formulários utilizados no processo de anamnese que o hospital disponibilizou. Esses documentos foram examinados para mapear os dados essenciais que a *API* deveria abrigar. Essa etapa foi crucial para garantir que a implementação da solução digital atendesse às necessidades práticas da equipe médica e não comprometesse a completude das informações necessárias para um atendimento de qualidade.

A combinação destas abordagens possibilitou a formulação de requisitos mais alinhados às necessidades do hospital, seguindo os princípios da *DSR* ao articular um problema real e propor um artefato tecnológico que o soluciona.

4.2 Desenvolvimento da API

Após o levantamento e análise dos requisitos, começou a fase de desenvolvimento e implementação. Nesta fase, foi desenvolvida a *API*. A prioridade foi desenvolver as funcionalidades das classes propostas. Desenvolver todos os *endpoints* e garantir que eles estivessem acessíveis e operacionais seguindo boas práticas de desenvolvimento.

4.2.1 Arquitetura da Solução

A estrutura da *API* foi projetada seguindo os princípios de *RESTful*, garantindo que suas funcionalidades fossem facilmente compreendidas e integradas por outros sistemas. A separação clara entre as funcionalidades de autenticação, gerenciamento de usuários e operações clínicas facilitou tanto a manutenção quanto a escalabilidade do sistema. Cada

endpoint foi documentado, assegurando que desenvolvedores possam compreender e interagir com a API sem dificuldades.

A arquitetura do sistema foi estruturada com base nos princípios do *Domain-Driven Design (DDD)*, visando criar um modelo orientado ao domínio do problema. Utilizando o *Spring Framework* como base, a arquitetura foi modularizada em diferentes pacotes, o que permite a organização das principais funcionalidades do sistema. Essa abordagem não apenas assegura a separação clara de responsabilidades entre os módulos, mas também facilita a manutenção e a evolução do software ao longo do tempo, permitindo que novos desenvolvimentos sejam integrados sem comprometer a integridade do sistema.

4.2.2 Modelagem de Dados

A modelagem de dados do sistema foi elaborada utilizando UML (*Unified Modeling Language*), o que possibilitou a criação de uma representação visual padronizada das entidades e seus respectivos relacionamentos. Esta abordagem não apenas facilita o entendimento das interações entre as entidades, como também serve como um guia contínuo durante o desenvolvimento. Os diagramas UML associados abrangem diversas entidades, incluindo Pacientes, Atendimentos e Usuários, e detalham as associações e restrições necessárias para garantir a integridade e a eficiência do sistema.

Através dos diagramas UML, foi possível definir a relação entre as principais classes, como Pacientes (*Patients*), Atendimentos (*Treatments*) e Usuários (*Users*), além de mapear as associações e restrições entre elas.

As classes foram desenvolvidas conforme o diagrama, pensado de forma que as mesmas pudessem englobar todas as entidades do negócio.

4.2.3 Modelagem dos Endpoints

Os *endpoints* da API foram cuidadosamente projetados para implementar um padrão *CRUD (Create, Read, Update, Delete)* básico para cada entidade do sistema, incluindo Pacientes, Atendimentos e Usuários. A modelagem das rotas foi feita com atenção às melhores práticas *RESTful*, onde cada *endpoint* é claramente definido para representar uma operação específica. Isso não somente melhora a semântica das URLs, mas também facilita o entendimento por parte dos desenvolvedores que utilizam a API. Adicionalmente, o uso de

verbos claros nas rotas, como "register", "update/{id}" e "delete/{id}", contribui para uma utilização mais intuitiva.

Além disso, os *endpoints* foram projetados para manter consistência e semântica *RESTful*, tornando a *API* intuitiva e fácil de integrar. A separação entre funcionalidades de autenticação, gerenciamento de usuários e operações clínicas também contribui para a organização e escalabilidade do sistema.

4.2.4 Segurança e Controle de Acesso

A segurança do sistema foi implementada com o uso do Spring Security. A estratégia adotada fundamenta-se em um controle de acesso baseado em funções (*roles*), onde cada usuário é atribuído a uma função específica que determina as permissões de acesso aos diferentes endpoints da *API*. Esta implementação não apenas garante a proteção dos dados sensíveis, mas também assegura que apenas usuários devidamente autorizados possam acessar funcionalidades críticas do sistema, proporcionando assim uma camada adicional de segurança.

A autenticação foi estruturada utilizando *JWT (JSON Web Token)*. O uso de *tokens* permite que as credenciais do usuário sejam verificadas sem a necessidade de armazenar sessões no servidor, reduzindo vulnerabilidades. Além disso, os dados contidos no *token* são criptografados, aumentando a proteção contra acessos não autorizados.

Com essa estratégia, o sistema assegura que apenas usuários devidamente autenticados e com permissões adequadas possam acessar determinados recursos.

4.3 Validação

A validação do sistema foi realizada através de testes unitários elaborados, seguindo a metodologia AAA (*Arrange, Act, Assert*). Essa abordagem assegura que cada teste é estruturado de forma e padronizada, aumentando a confiabilidade do sistema. Os testes foram projetados para cobrir todos os *endpoints* da *API*, servindo não apenas para verificar o correto funcionamento das operações, mas também como uma base para o desenvolvimento de futuros testes à medida que a *API* evolui. A inclusão de testes automatizados garante uma detecção precoce de erros e contribui significativamente para a manutenção eficiente do software.

Os testes foram elaborados considerando todos os *endpoints* atuais do sistema, verificando se os serviços e operações funcionam conforme esperado. Além disso, eles foram

planejados para servirem como base para o desenvolvimento de novos testes à medida que a *API* for evoluindo e sendo customizada para novas instituições.

Além disso, os testes fazem parte de um processo contínuo de melhoria, podendo ser revisados e expandidos conforme novas funcionalidades são adicionadas ou modificadas na *API*. A existência desses testes permite a detecção precoce de erros, reduzindo custos de manutenção e tornando o desenvolvimento mais seguro e eficiente. Esses fatores contribuem para uma base sólida para futuras evoluções do sistema.

Ainda, foram escritos de forma organizada, permitindo facilidade de manutenção. Com isso, outros desenvolvedores podem entender, reutilizar e expandir os casos de teste conforme necessário, sem comprometer a estrutura existente. Essa abordagem também facilita a identificação de problemas e a adaptação dos testes à medida que o sistema evolui.

4.4 Documentação

A documentação do sistema foi realizada utilizando duas principais ferramentas: Swagger e Insomnia. Essas ferramentas foram escolhidas por sua eficiência na visualização, organização e, principalmente, teste dos *endpoints* da *API*.

O Swagger foi utilizado para gerar uma documentação interativa da *API*. Através dele, os endpoints foram descritos, incluindo suas respectivas rotas, métodos *HTTP* suportados, parâmetros esperados e os possíveis retornos. Uma documentação dinâmica permite que qualquer usuário explore a *API* de forma intuitiva, realizando testes diretamente pela interface gráfica do Swagger UI, sem a necessidade de ferramentas externas.

Além disso, o Insomnia foi empregado para validar e testar as requisições da *API* de maneira prática. A ferramenta possibilitou a organização dos endpoints em coleções, facilitando a reutilização e a execução de testes durante o desenvolvimento. Com o Insomnia, foi possível verificar o comportamento das requisições *HTTP* e analisar as respostas retornadas pelo sistema, observando que a *API* operava conforme esperado.

A metodologia adotada permitiu a estruturação do sistema por meio de uma modelagem detalhada, gerando uma base de conhecimento para o desenvolvimento da *API*. Além disso, a documentação e as diretrizes estabelecidas facilitaram a organização e a padronização dos componentes do sistema, assegurando sua escalabilidade e interoperabilidade com outros sistemas hospitalares. Com essa fundamentação técnica concluída, a próxima seção abordará os resultados obtidos, apresentando a implementação da solução proposta e a análise dos desafios e benefícios identificados durante o desenvolvimento.

5. RESULTADOS E DISCUSSÕES

Esta seção apresenta os resultados obtidos durante o desenvolvimento do sistema de anamnese hospitalar, destacando a modelagem da API, simulações realizadas, testes unitários implementados, limitações enfrentadas e experiências adquiridas durante o processo. Ao final, busca-se refletir sobre o impacto dessas atividades na qualidade e eficácia do sistema desenvolvido.

5.1 Modelo da API

A API foi desenvolvida com base no modelo de arquitetura do Spring, uma escolha que permite flexibilidade e escalabilidade no desenvolvimento de aplicações, favorecendo a construção de sistemas robustos e de fácil manutenção. A modelagem dos dados foi conduzida através de UML, garantindo uma estrutura organizada que facilita a compreensão e o uso posterior. No contexto hospitalar, onde os processos geralmente são complexos e constantemente atualizados, ter modelos bem definidos é essencial para representar com precisão os cenários reais da instituição. Essa abordagem não apenas facilita futuras adaptações e evoluções do sistema, como também promove a interoperabilidade da API com outros sistemas hospitalares. Além disso, a API foi projetada desde o início com base nos requisitos identificados no início do estudo, garantindo que sua estrutura atenda diretamente às necessidades do hospital.

5.1.1 Requisitos Funcionais

A Tabela 1 apresenta os requisitos funcionais identificados através das observações e análise documental dos artefatos disponibilizados pelo hospital.

Tabela 1 - Requisitos Funcionais

ID	Requisito	Prioridade	Descrição	Justificativa	Impacto no Hospital
RF001	Cadastro de pacientes	Alta	O sistema deve permitir o cadastro de pacientes.	Permite armazenar os dados dos pacientes para acompanhame	Facilita o acesso rápido às informações dos pacientes

				nto e atendimento.	no momento da anamnese.
RF002	Consulta de pacientes	Alta	O sistema deve permitir a consulta dos pacientes cadastrados no sistema.	Garante que profissionais possam acessar rapidamente os dados necessários.	Agiliza os atendimentos e evita cadastros duplicados.
RF003	Atualização de pacientes	Média	O sistema deve permitir que o usuário atualize os dados do paciente no sistema.	Mantém as informações dos pacientes sempre atualizadas.	Reduz erros administrativos e melhora a gestão hospitalar.
RF004	Remoção de pacientes	Baixa	O sistema deve permitir remover os dados de um paciente no sistema.	Atende às diretrizes de privacidade e controle de dados.	Evita armazenamento desnecessário e melhora a organização dos registros.
RF005	Registro de atendimentos	Alta	O sistema deve permitir o registro de atendimentos.	Mantém o histórico clínico dos pacientes atualizado.	Facilita o acompanhamento da evolução do paciente.
RF006	Consulta de atendimentos por paciente	Alta	O sistema deve permitir a consulta dos atendimentos cadastrados	Ajuda médicos e enfermeiros a entenderem o histórico do paciente.	Reduz retrabalho e melhora a precisão dos diagnósticos.

			no sistema por paciente.		
RF007	Remoção de atendimentos	Média	O sistema deve permitir remover os dados de um atendimento no sistema.	Permite correção de registros errôneos.	Garante integridade e conformidade com a política de dados.
RF008	Consulta de atendimentos por médico	Alta	O sistema deve permitir a consulta dos atendimentos cadastrados no sistema por médico.	Facilita a gestão de pacientes por profissional.	Aumenta a eficiência dos atendimentos.
RF009	Atualização de atendimentos	Média	O sistema deve permitir que o usuário atualize os dados de um atendimento no sistema.	Garante que informações estejam sempre corretas.	Evita erros médicos e melhora o acompanhamento clínico.
RF010	Registro de usuários	Alta	O sistema deve permitir o registro de usuários no sistema.	Garante controle de acesso e permissões.	Evita acessos indevidos a informações sensíveis.
RF011	Login de usuários	Alta	O sistema deve permitir o login de usuários no sistema.	Restringe o acesso conforme as funções de cada usuário.	Protege informações confidenciais dos pacientes.

RF012	Listagem de usuários	Média	O sistema deve exibir uma lista dos usuários registrados no sistema.	Facilita a gestão dos profissionais e administradores.	Permite o controle e administração de contas.
RF013	Exibir usuário logado	Média	O sistema deve exibir o atual usuário logado no sistema.	Auxilia no monitoramento e auditoria de ações no sistema.	Evita erros operacionais por conta de múltiplos acessos.
RF014	Exibir usuário por ID	Baixa	O sistema deve exibir um usuário registrado no sistema por ID.	Permite busca rápida de um usuário específico.	Facilita a administração de contas no hospital.

Fonte: Elaborado pelo autor (2025)

Os requisitos foram pensados de forma a atenderem as necessidades para um processo de anamnese e registro dos pacientes e consultas do hospital, buscar realizar o necessário para um desenvolvimento da *API* que servisse para essa funcionalidade e que fosse intuitivo para os agentes de saúde.

5.1.2 Requisitos não funcionais

Além das funcionalidades necessárias para atender ao fluxo de processo da anamnese existem implementações importantes que impactam transversalmente a qualidade do sistema desenvolvido. A Tabela 2 lista os requisitos não-funcionais relacionados ao sistema.

Tabela 2 - Requisitos Não Funcionais

ID	Requisito	Prioridade	Descrição	Impacto no Hospital
----	-----------	------------	-----------	---------------------

RNF001	Apenas usuários autenticados podem acessar funcionalidades gerenciais do sistema	Alta	Restringe funcionalidades sensíveis apenas a usuários autorizados, garantindo maior segurança e controle sobre os dados e operações hospitalares.	Protege informações críticas e evita acessos não autorizados, garantindo integridade e confiabilidade nos processos hospitalares.
RNF002	Todos os dados sensíveis (senhas, documentos, informações médicas) devem ser armazenados de forma segura, utilizando criptografia sempre que necessário.	Alta	Implementa criptografia para proteger dados sigilosos, evitando acessos indevidos e vazamentos de informações sensíveis.	Garante confidencialidade e segurança dos dados dos pacientes.
RNF003	O sistema deve fornecer mensagens de erro esclarecedoras para ajudar usuários leigos a utilizarem o sistema.	Média	Oferece <i>feedback</i> claro sobre erros e problemas, facilitando a experiência do usuário e reduzindo falhas operacionais.	Melhora a usabilidade para profissionais da saúde, reduzindo tempo de treinamento e evitando erros em procedimentos críticos.

RNF004	Campos como CPF, RG, CNS e título de eleitor devem ser únicos no banco de dados para evitar duplicidades.	Alta	Garante a integridade dos dados ao impedir registros duplicados, evitando inconsistências e possíveis fraudes.	Assegura a precisão das informações cadastrais dos pacientes, evitando erros administrativos e otimizando atendimentos.
RNF005	O sistema deve implementar controle de acesso baseado em roles para definir permissões conforme a função do usuário.	Alta	Cada usuário possui permissões específicas conforme seu papel (atendente, enfermeiro, médico, gerência e paciente), garantindo que apenas funções autorizadas acessem determinados endpoints.	Reforça a segurança e evita que usuários acessem informações além do necessário, garantindo a integridade e confiabilidade no uso do sistema hospitalar.

Fonte: Elaborado pelo autor (2025)

5.1.3 Arquitetura do Sistema

Nessa sessão será apresentada a arquitetura do sistema, mostrando como foi pensada se baseando no contexto hospitalar e falando de forma técnica sobre o sistema.

5.1.3.1 Model

A primeira etapa foi a definição das classes principais para cada entidade envolvida no sistema. Foi feita uma análise das classes que seriam necessárias considerando o contexto hospitalar e os requisitos evidenciados anteriormente. A partir disso, foram criadas as classes

Patient, *User* e *Treatment*, que representavam os objetos do domínio. Cada classe de entidade foi anotada com as anotações do *JPA*, facilitando a persistência automática no banco de dados com o uso de *@Entity*, *@Id*, e *@GeneratedValue* (onde foi usada a estratégia de *UUID* para os *ID* únicos), *@Getter*, *@Setter*.

Os repositórios são responsáveis por interagir diretamente com o banco de dados. Com o uso do *Spring Data JPA*, não foi necessário escrever código *SQL* diretamente, bastando apenas utilizar interfaces que estendiam *JpaRepository* do *Spring*.

Por exemplo, o repositório de pacientes (*PatientRepository*) permite realizar buscas customizadas utilizando métodos padrão do próprio *JpaRepository*, como *findById*, que busca uma entidade pelo *id* associado, esse método também permite a criação de consultas personalizadas quando necessário.

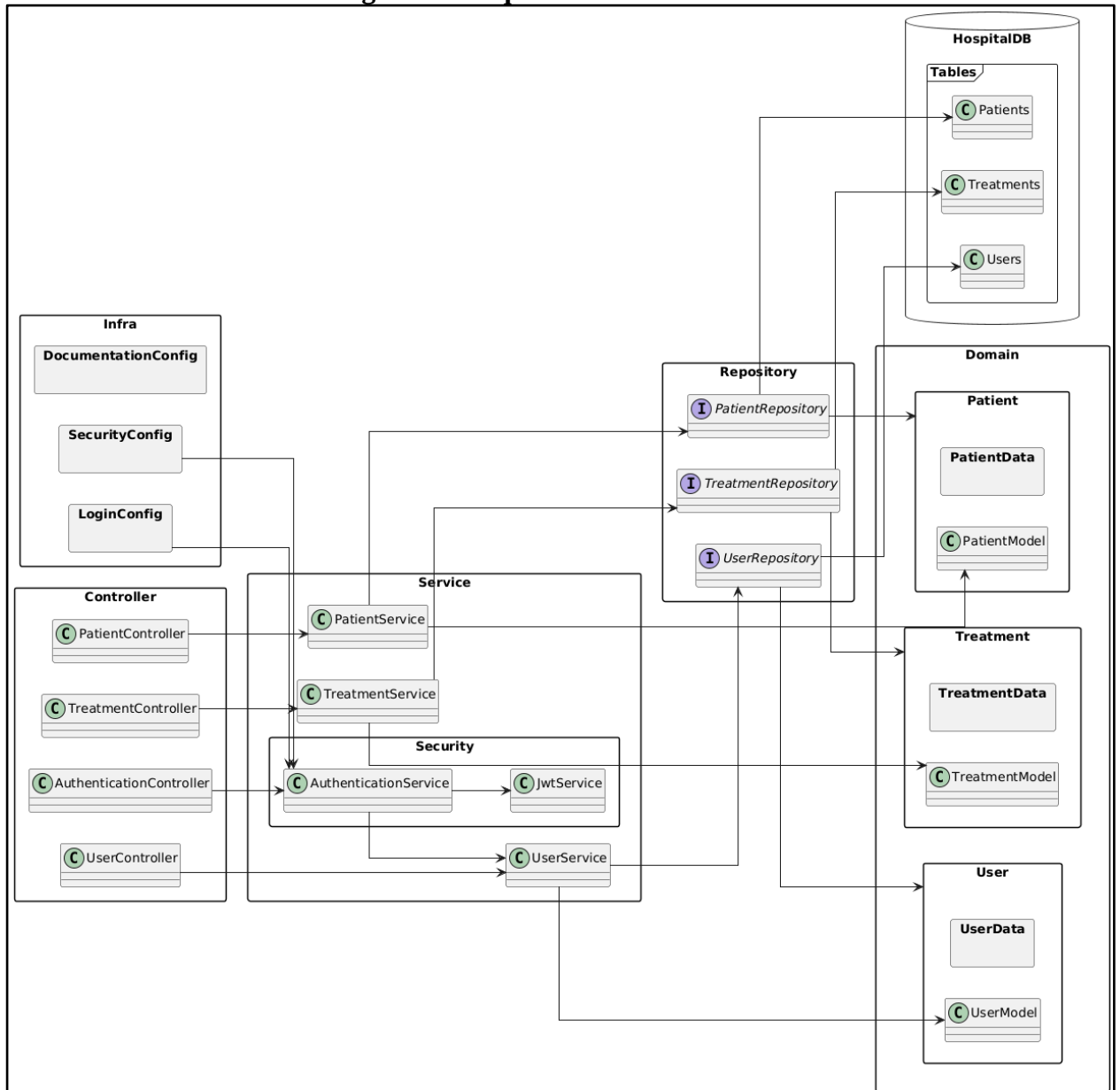
O PostgreSQL foi escolhido como o banco de dados relacional para armazenar as informações críticas do sistema. A integração foi feita através do *Spring Data JPA*, que mapeou os objetos do sistema para o banco de dados, através das anotações passadas nas classes. Para o controle das versões de banco de dados, foi usado o *Flyway*, o que permite garantir a consistência das migrações de banco de dados enquanto o sistema aumenta sua complexidade, não precisando refazer essa modelagem, foi possível fazer o mapeamento e a criação das tabelas no banco de dados através da criação de arquivos *.sql* com as informações necessárias para a criação. Por isso, nesse projeto, não foi desenvolvido um modelo tradicional de banco de dados, pois foi utilizado um ORM (*Object-Relational Mapping*), mais especificamente o *Hibernate*. O ORM permite o mapeamento dos objetos do sistema diretamente para o banco de dados, eliminando a necessidade de uma modelagem relacional explícita. Com isso, a estrutura das tabelas é gerada automaticamente a partir das classes de entidade definidas no código, garantindo flexibilidade e consistência nas migrações.

E, seguindo a arquitetura do *spring* e os princípios do *DDD*, a arquitetura do sistema foi dividida por módulos. Os módulos foram divididos observando a lógica de negócio e a separação das atividades descritas na instituição de saúde. Dessa forma, manutenção e criação de novas funcionalidades podem ser feitas se atentando a um serviço específico fornecido no hospital. O modelo utilizado para representar a arquitetura do sistema segue a UML, especificamente através do diagrama de componentes que, Segundo KOC, Hatice *et al* (2021), é uma representação visual utilizada na modelagem de sistemas para descrever a organização e as interações entre os componentes de um sistema. Dentro do contexto de sistemas de software, ele fornece uma visão de como os componentes individuais, como módulos, classes e interfaces,

se relacionam entre si e formam a estrutura do sistema mais amplo. Este tipo de diagrama é especialmente útil quando se deseja mostrar a arquitetura do sistema em um nível de abstração que revela como as partes interagem sem entrar em detalhes de implementação.

Além disso, o diagrama de componentes é utilizado para ilustrar as dependências e as interfaces entre componentes, permitindo identificar como as mudanças em um componente podem afetar outros (KOC, Hatice, 2021). Como apresentado na Figura 3.

Figura 3 - Arquitetura do Sistema



Fonte: Elaborado pelo autor (2025)

Os módulos da arquitetura são detalhados a seguir:

- *Controller*: Camada responsável por receber e processar as requisições *HTTP*. Contém os seguintes controladores:
 - *AuthenticationController*
 - *PatientController*
 - *TreatmentController*
 - *UserController*

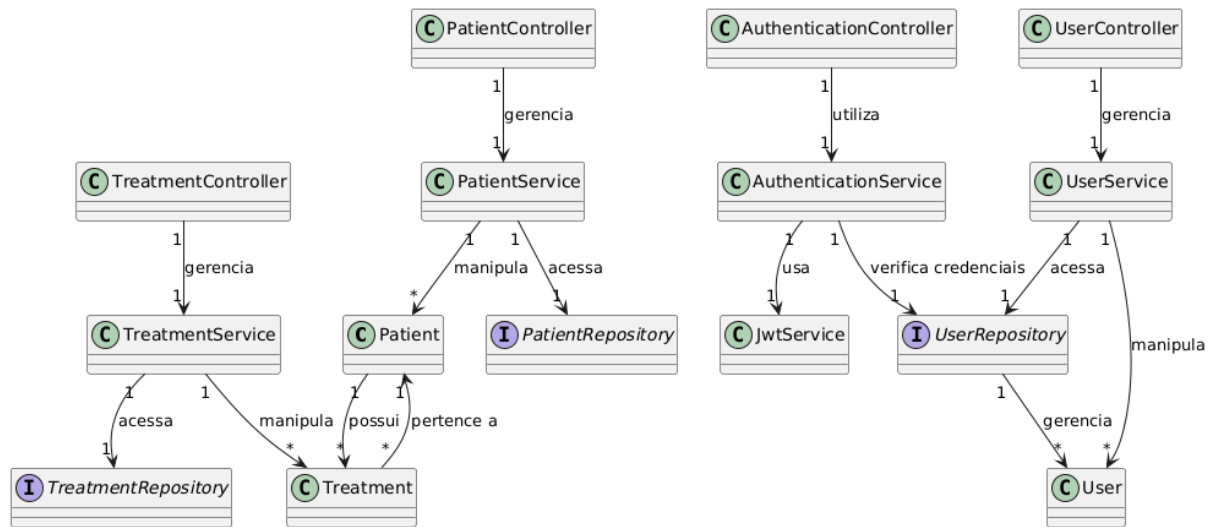
- *Domain*: Representa o núcleo do sistema, contendo as entidades e regras de negócio.
 - *Exceptions*: Define classes de exceção personalizadas.
 - *Patient*: Contém a entidade *Patient* e o pacote *PatientData* para manipulação de dados.
 - *Treatment*: Contém a entidade *Treatment* e o pacote *TreatmentData*.
 - *User*: Contém a entidade *User* e o pacote *UserModel* para estruturação dos dados do usuário.
 - *Repository*: Inclui interfaces para comunicação com o banco de dados:
 - *PatientRepository*
 - *TreatmentRepository*
 - *UserRepository*
 - *Service*: Implementa a lógica de negócio e operações do sistema:
 - *PatientService*
 - *TreatmentService*
 - *UserService*

- *Security* (módulo de segurança):
 - *AuthenticationService*
 - *JwtService*

- *Infra*: Contém a infraestrutura da aplicação, incluindo segurança e configurações.
 - *SecurityConfig*: Configuração de autenticação e autorização.
 - *LoginConfig*: Gerenciamento de login e autenticação.
 - *DocumentationConfig*: Configuração da documentação da *API*.

Esse contexto hospitalar para a arquitetura de sistema reflete também no diagrama de classes, como apresentado na Figura 4. É importante destacar que os atributos e operações foram omitidos para facilitar a visualização dos relacionamentos.

Figura 4 - Diagrama de Classes



Fonte: Elaborado pelo autor (2025)

5.1.3.2 Service Layer

Dentro das classes de serviço, como *PatientService*, *TreatmentService* e *UserService*, foram implementados métodos que executavam as operações mais complexas, como a criação das entidades, associação entre elas e busca de informações no banco de dados. Esses métodos chamam os repositórios, validam as regras de negócio e retornavam *DTOs* para serem enviados como resposta às requisições.

Para garantir uma boa separação de responsabilidades entre as camadas, foram utilizados *DTOs* (*Data Transfer Objects*) para a comunicação entre consumidores da *API* e o *backend*. Os *DTOs* encapsulam os dados que seriam expostos ou recebidos, mantendo as entidades protegidas e evitando problemas de segurança ou inconsistência, essa abordagem permite o controle do que é compartilhado, evitando compartilhamento de informação considerada sensível.

Os *DTOs* foram utilizados tanto na entrada quanto na saída de dados, sendo convertidos para entidades dentro dos serviços e vice-versa, permitindo a criação de uma camada de abstração entre a lógica de negócio e a representação dos dados.

Além disso, os *DTOs* foram pensados através dos dados sensíveis dos pacientes e tratamentos dos mesmos, de forma a melhorar a segurança desses dados.

5.1.3.3 Controller

Os controladores são responsáveis por expor os *endpoints REST* para que outros sistemas ou interfaces pudessem interagir com o *backend*. Esses controladores usam o Spring Web, e cada método corresponde a uma rota específica da *API*.

5.1.3.4 Rotas

Com base nesses requisitos, estudos e o contexto hospitalar, as rotas (*endpoints*) foram criadas de forma que atendessem o que se espera do sistema e do que ele precisa fazer, seja no método necessário ou sua função, com mostra a tabela 3.

Tabela 3 - Rotas da API

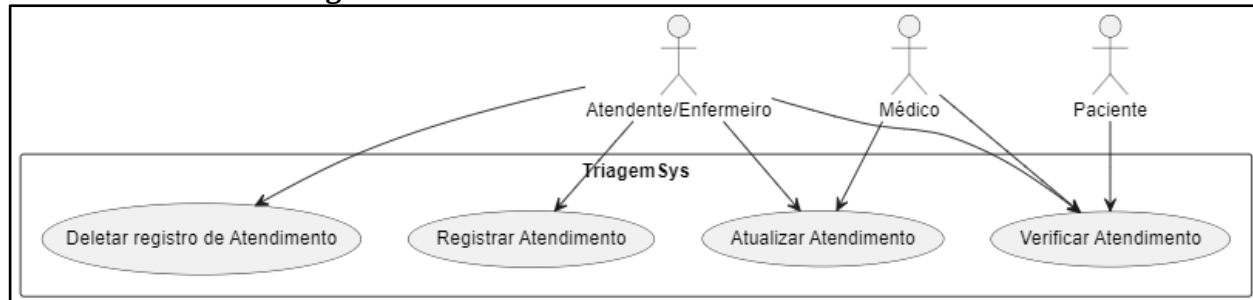
Função	Endpoint	Método HTTP
Registrar atendimento	/treatments/register	POST
Retornar todos os atendimentos	/treatments/all	GET
Retornar atendimento por protocolo	/treatments/{protocolId}	GET
Atualizar Atendimento	/treatments/update/{id}	PUT
Deletar registro de atendimento	/treatments/delete/{id}	DELETE
Retornar atual usuário logado	/users/me	GET
Retornar todos os usuários	/users/all	GET
Retornar usuário por id	/users/{id}	GET
Cadastro	/auth/signUp	POST
Login	/auth/login	POST
Registrar Novo paciente	/patients/register	POST
Retornar todos os pacientes	/patients/all	GET
Retornar paciente por id	/patients/{id}	GET
Atualizar Dados do Paciente	/patients/update/{id}	PUT
Deletar dados do paciente	/patients/delete/{id}	DELETE

Fonte: Elaborado pelo autor (2025)

Foram desenvolvidos diagramas de caso de uso de forma a entender e representar melhor o fluxo do processo de anamnese para realizar o mapeamento de funcionalidades do

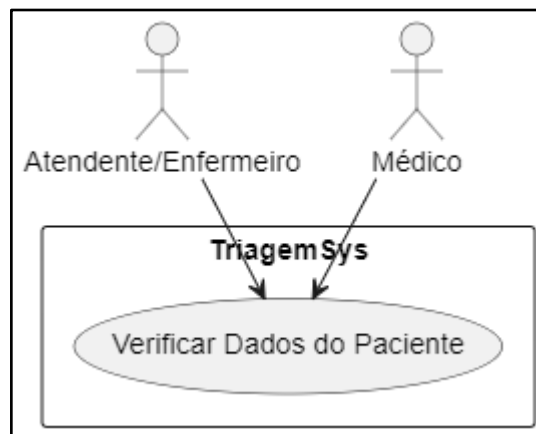
sistema. A Figura 5 apresenta o cenário de casos de uso para atendimento destacando os principais atores do processo que são Paciente, Atendente e Médico. Enquanto a Figura 6 apresenta o caso de uso específico de verificar dados.

Figura 5 - Casos de Uso Atendimento



Fonte: Elaborado pelo autor (2025)

Figura 6 - Caso de Uso de Verificar dados



Fonte: Elaborado pelo autor (2025)

É importante lembrar que os casos de uso foram pensados se baseando nas categorias de acesso do hospital e dos pacientes, imaginando um uso desses papéis no sistema em soluções futuras que serão desenvolvidas consumindo a *API* desenvolvida neste trabalho, para as rotas poderem ser acessadas, principalmente no caso dos pacientes que não terão acesso direto a *API*.

5.2 Simulação

Nesta seção são apresentadas simulações que foram realizadas com dados fictícios para avaliar adequação das funcionalidades esperadas na perspectiva do consumo de dados da *API*, considerando o uso da mesma em um sistema criado a partir do seu consumo.

Imaginando um cenário: O paciente Francisco Canindé chegou ao Hospital se queixando de fortes dores de cabeça e sensibilidade à luz. Ao chegar na recepção, ele foi prontamente atendido pelo setor de anamnese, onde uma enfermeira coletou suas informações básicas. O cadastro de paciente foi feito pela enfermeira cadastrada no sistema através da requisição “/patients/register” passando os dados necessários. Com os dados passados corretamente, a requisição retorna um 201, created. Isso significa uma resposta de um sistema de que a requisição foi solicitada da forma correta. Mostra que o sistema executou uma criação no sistema. Além disso, é exibida a mensagem “Paciente registrado com sucesso”. Como mostrado na Figura 7.

Figura 7 – Registro de Paciente

POST	_.base_url	patients/register	Send	201 Created	251 ms	32 B
Params	Body	Auth	Headers	Scripts	Docs	Preview
JSON				Preview		
<pre>1 { 2 "name": "Francisco Canindé", 3 "birth": "1972-02-20", 4 "rg": "12345678", 5 "cpf": "12345678900", 6 "electoralCard": "000000", 7 "cns": "123456789012345", 8 "motherName": "Maria da Silva", 9 "fatherName": "José da Silva", 10 "addressLineOne": "Rodovia BR-405", 11 "addressLineTwo": "Km 3", 12 "addressNumber": "000", 13 "city": "Pau dos Ferros", 14 "state": "RN", 15 "cep": "50000000", 16 "bloodType": "O+", 17 "isBloodDonor": true, 18 "isOrganDonor": false, 19 "isBoneMarrowDonor": false, 20 "notes": "Paciente com histórico de hipertensão" 21 }</pre>				1 Paciente registrado com sucesso!		

Fonte: Elaborado pelo autor (2025)

Após isso, a enfermeira registrada no sistema cria um registro de atendimento, através da requisição “/treatments/register”. Nesse momento, é necessário passar os dados relevantes da anamnese. O que inclui o id do protocolo de atendimento e o id do registro do paciente. Além disso, há um campo para informar a análise do que foi investigado na anamnese inicial, que a enfermeira coleta após o paciente se registrar. Além disso, é exibida a mensagem “Atendimento registrado com sucesso”, como mostra a Figura 8.

Figura 8 – Registro de atendimento

The screenshot displays a REST client interface for a POST request to the endpoint `..base_url treatments/register`. The status bar indicates a successful response with **201 Created**, a response time of **55 ms**, and a body size of **35 B**. The **Body** tab is selected, showing the JSON payload. The **Preview** tab on the right shows the response message.

Request Body (JSON)	Response Preview
<pre> 1 { 2 "protocol": "ABC123", 3 "priority": "Baixa", 4 "patientId": "618e4beb-6ed0-4272-80b5-dfc8d7ea85c3", 5 "type": "Atendimento na clínica de Urgência", 6 "arrival": "Conta própria", 7 "wardNumber": "-", 8 "bed": "-", 9 "admission": "2025-02-08T14:00:00", 10 "discharge": "2025-02-08T14:00:00", 11 "bloodPressure": "120/80", 12 "heartRate": "72", 13 "respiratoryRate": "16", 14 "temperature": 36, 15 "oxygenSaturation": 98, 16 "bloodSugar": 90, 17 "chiefComplaint": "Anamnese: Paciente chegou se queixando de dor de cabeça e sensibilidade a luz. Foi feita a anamnese inicial, nada de anormal foi detectado" 18 } </pre>	<pre> 1 Atendimento registrado com sucesso! </pre>

Fonte: Elaborado pelo autor (2025)

Após isso, o paciente é levado para o atendimento médico, onde será examinado devidamente pelo profissional de saúde. Após atendimento, o médico registrado no sistema que atendeu ao paciente atualiza os dados do atendimento através da requisição `"/treatments/update/{id}"`. Nessa atualização, irá fazer a mudança do horário de saída do paciente e do registro do atendimento, informando dados do seu atendimento, a análise do caso clínico e possíveis medicamentos passados. Após isso, é exibida a mensagem "Registro de Atendimento atualizado com sucesso! Como mostra a Figura 9.

Figura 9 – Atualizar dados de atendimento



Fonte: Elaborado pelo autor (2025)

Essa simulação ajuda a entender o comportamento da API, e o seu uso no dia a dia pelos profissionais de saúde a partir das interfaces disponibilizadas que serão integradas a API.

5.3 Testes Unitários

A implementação de testes unitários foi uma parte fundamental do projeto, pois permite garantir a funcionalidade correta de cada componente do sistema de forma isolada. Em um ambiente hospitalar, onde a confiabilidade dos dados e a integridade das informações são essenciais para a tomada de decisões médicas, garantir que cada funcionalidade do sistema opere corretamente reduz riscos e aumenta a segurança dos processos.

Utilizando as anotações do JUnit, como `@Test`, e integração com o Mockito para simular dependências, os testes foram configurados para garantir que a API funcione conforme esperado, fornecendo segurança e confiabilidade durante o desenvolvimento e manutenção contínua. Além disso, foram desenvolvidos testes unitários para cada *endpoint* da aplicação, garantindo que as operações críticas, como cadastro de pacientes, consultas e atualizações, sejam realizadas corretamente.

A abordagem adotada foi o método AAA (*Arrange, Act, Assert*), estruturando os testes de maneira clara e organizada. Além do JUnit, foram utilizados recursos como Mockito para a criação de *mocks* e simulação de dependências. A execução dos testes foi automatizada por meio do Maven/Gradle, possibilitando ciclos contínuos de desenvolvimento e validação.

Os testes foram pensados dessa forma para servirem como base para o desenvolvimento de novos testes conforme novas funcionalidades forem adicionadas, garantindo a escalabilidade e confiabilidade do sistema hospitalar.

Na imagem abaixo, destacam-se dois testes unitários do controller da entidade Patients, responsáveis por validar os *endpoints* da aplicação. O primeiro verifica se o registro de paciente retorna a mensagem e o status necessários. O segundo testa se o status correto é retornado e se os dados do paciente são recuperados corretamente. Os métodos de teste criados evidenciam as etapas de *Arrange*, *Act* e *Assert*, ajudando no entendimento da metodologia para a criação de novos testes à medida que novas funcionalidades são implementadas, conforme mostrado na Figura 10.

Figura 10 – Testes classe paciente

```
@Test
void editPatient_ShouldReturnUpdatedMessage() {
    // Arrange
    when(patientService.editPatient(patientId, registerPatientDTO)).thenReturn(new Patient());

    // Act
    ResponseEntity<String> response = patientController.editPatient(patientId, registerPatientDTO);

    // Assert
    assertEquals(200, response.getStatusCodeValue());
    assertEquals("Paciente atualizado com sucesso!", response.getBody());
}

@Test
void deletePatientById_ShouldReturnDeletedMessage() {
    // Arrange
    doNothing().when(patientService).deletePatientById(patientId);

    // Act
    ResponseEntity<String> response = patientController.deletePatientById(patientId);

    // Assert
    assertEquals(200, response.getStatusCodeValue());
    assertEquals("Paciente deletado com sucesso!", response.getBody());
}
```

Fonte: Elaborado pelo autor (2025)

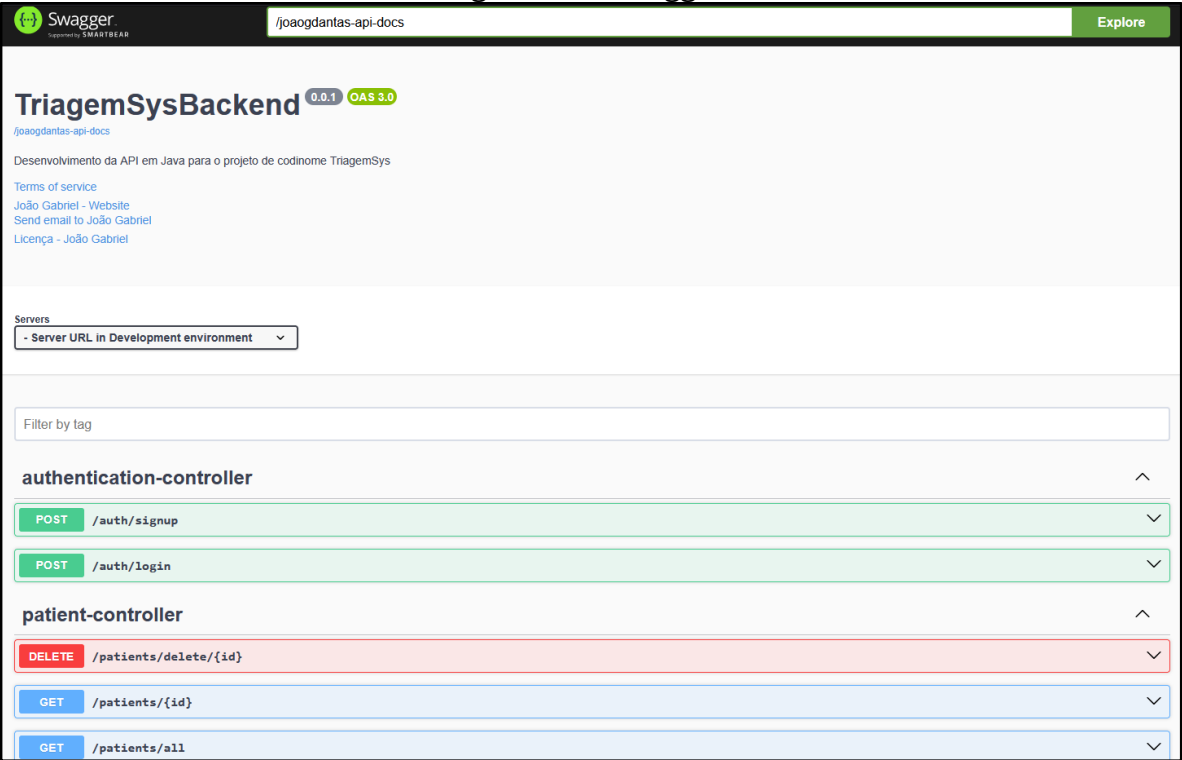
Ao serem executados, os testes acusam sucesso ou falha. Em caso de falha, provavelmente há um erro no trecho de código que o método está testando, nesse caso, alguma funcionalidade pode não estar funcionando como se espera.

5.4 Swagger e Insomnia

Nesta seção, é apresentada a estrutura da documentação e organização dos *endpoints* da API utilizando as ferramentas Swagger UI e Insomnia. As imagens a seguir ilustram como essas ferramentas foram utilizadas para facilitar a interação com a API.

O Swagger fornece uma visão geral dos *endpoints* disponíveis, permitindo que os desenvolvedores compreendam rapidamente as operações suportadas pela API, como mostram as Figuras 11 e 12.

Figura 11 – Swagger UI



Fonte: Elaborado pelo autor (2025)

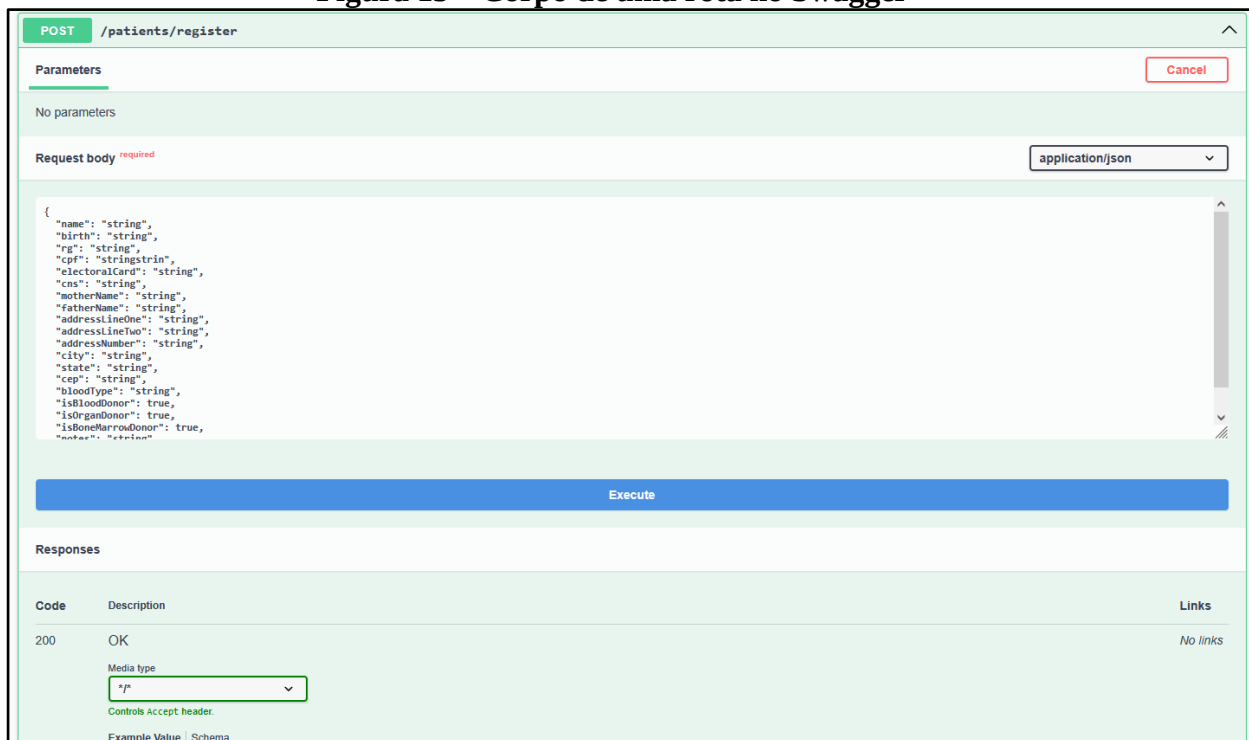
Figura 12 – Swagger UI Complemento



Fonte: Elaborado pelo autor (2025)

Há uma visão detalhada de uma rota no Swagger UI. Nessa visualização, é possível identificar o método HTTP associado, os parâmetros necessários, a estrutura do corpo da requisição e as possíveis respostas retornadas pelo servidor. Essa documentação interativa facilita que a API esteja sendo utilizada corretamente. Como mostra a Figura 15.

Figura 13 – Corpo de uma rota no Swagger



Fonte: Elaborado pelo autor (2025)

Por fim, a imagem a seguir exibe as rotas organizadas no Insomnia em coleções e pastas. Essa abordagem permite que os testes das requisições HTTP sejam feitos de forma estruturada, organizando os endpoints de acordo com suas funcionalidades. Dessa forma, é possível realizar testes rápidos, simular diferentes cenários de requisição e verificar as respostas obtidas da API.

Durante o desenvolvimento da API, o Insomnia foi utilizado para testar e validar as requisições de forma eficiente. A ferramenta permitiu a criação de coleções organizadas por funcionalidades, facilitando a verificação dos endpoints em diferentes cenários. Além disso, foi utilizada para simular autenticação via JWT, garantindo que apenas usuários autorizados acessassem determinadas rotas. O Insomnia também foi essencial para testar parâmetros dinâmicos, analisar respostas do servidor e identificar possíveis falhas, tornando o processo de desenvolvimento mais ágil e confiável.

5.5 Acesso ao sistema

A segurança do sistema foi implementada utilizando autenticação baseada em JWT (*JSON Web Token*), garantindo que apenas usuários autenticados tenham acesso às funcionalidades sensíveis do hospital. Em um ambiente hospitalar, onde informações médicas e dados de pacientes são altamente sigilosos, a implementação de uma autenticação robusta é essencial para manter a integridade.

A autenticação com JWT permite que os usuários realizem login e obtenham um *token* de acesso, que deve ser incluído nas requisições para acessar recursos protegidos. Esse método impede que usuários não autorizados acessem dados confidenciais, garantindo que apenas os profissionais de saúde e gestores com os devidos privilégios possam interagir com informações críticas.

Após a autenticação do usuário, o sistema gera um *token* que contém dados essenciais, como o ID do usuário e sua função dentro do hospital (atendente, enfermeira, médico, gerência ou paciente). Esse token é assinado digitalmente para evitar adulterações e garantir sua autenticidade. Assim, quando um usuário realiza uma requisição a um endpoint protegido, o sistema verifica a validade do *token* e as permissões associadas, assegurando que ele tenha acesso apenas aos recursos adequados ao seu nível de acesso.

Além disso, cada *token* possui um tempo de expiração, reduzindo o risco de acessos indevidos caso o *token* seja comprometido. Para manter a experiência do usuário sem comprometer a segurança, podem ser aplicados mecanismos de *refresh token*, permitindo que sessões sejam renovadas de maneira segura.

Essa abordagem fortalece a proteção de informações críticas, garantindo que apenas profissionais autorizados tenham acesso a dados sensíveis. Dessa forma, o sistema assegura a confidencialidade das informações dos pacientes, minimizando riscos de vazamento e proporcionando um ambiente digital seguro para operações hospitalares. As classes foram desenvolvidas pensando nesse contexto, como mostram as Figuras 16 e 17.

Figura 14 – Classe Jwt Service

```
@Service
public class JwtService {
    @Value("${security.jwt.secret-key}")
    private String secretKey;

    @Value("${security.jwt.expiration-time}")
    private long jwtExpiration;

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }

    public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {
        return buildToken(extraClaims, userDetails, jwtExpiration);
    }

    public long getExpirationTime() {
        return jwtExpiration;
    }

    private String buildToken(
        Map<String, Object> extraClaims,
        UserDetails userDetails,
        long expiration
    ) {
```

Fonte: Elaborado pelo autor (2025)

Figura 15 – Classe Jwt Service Complemento

```
public class JwtService {
    private String buildToken(
        .setSubject(userDetails.getUsername())
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() + expiration))
        .signWith(getSignInKey(), SignatureAlgorithm.HS256)
        .compact();
    }

    public boolean isValidToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
            .parserBuilder()
            .setSigningKey(getSignInKey())
            .build()
            .parseClaimsJws(token)
            .getBody();
    }

    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

Fonte: Elaborado pelo autor (2025)

Em um ambiente hospitalar, onde a segurança das informações dos pacientes e a organização dos fluxos de trabalho são essenciais, o sistema foi projetado para implementar um controle de acesso baseado em *roles*. Dessa forma, cada profissional tem permissões específicas de acordo com sua função no hospital, garantindo que apenas usuários autorizados possam acessar determinadas informações e funcionalidades.

As contas são classificadas em diferentes níveis hierárquicos, incluindo atendente, enfermeiro, médico, gerência e paciente, cada uma com acesso restrito a *endpoints* específicos. Por exemplo, atendentes podem cadastrar e consultar pacientes, garantindo que o processo de admissão seja eficiente e seguro. Já os médicos possuem acesso aos atendimentos e informações clínicas, podendo atualizar prontuários e registrar diagnósticos. A gerência, por sua vez, possui

permissões administrativas, permitindo a gestão de usuários e garantindo que os processos internos estejam alinhados com as diretrizes da instituição.

Esse modelo de controle de acesso não apenas reforça a segurança e a integridade das informações, mas também otimiza a operação hospitalar. Ele garante que cada profissional possa interagir apenas com os dados necessários para sua atuação, reduzindo riscos de acessos indevidos e assegurando conformidade com normas de segurança e regulamentações do setor da saúde.

5.6 Limitações do Estudo

O ambiente hospitalar é dinâmico, exigindo soluções tecnológicas que acompanhem constantes mudanças nos protocolos de atendimento, regulamentações e necessidades dos profissionais de saúde. Além disso, a integração entre diferentes setores, como recepção, enfermagem, corpo clínico e administração, demanda sistemas que garantam um fluxo eficiente de informações, mantendo a segurança e a confiabilidade dos dados dos pacientes.

O desenvolvimento focou na criação de uma API genérica e flexível, para buscar o equilíbrio entre flexibilidade e especialização. Isso resultou em algumas limitações, como a dificuldade em incluir funcionalidades específicas, como a priorização automática no atendimento de emergência, que poderiam otimizar processos hospitalares, que será necessário em futuras atualizações. Além disso, a integração de certos recursos pode ter sido restringida pelo tempo e pelos recursos disponíveis. Em projetos futuros, a inclusão de um escopo mais detalhado e uma alocação adequada de tempo e equipe poderiam permitir a implementação de funcionalidades mais robustas e específicas.

Uma das principais dificuldades encontradas foi a necessidade de manter a API simples e abrangente ao mesmo tempo. Ao priorizar uma estrutura genérica, algumas funcionalidades mais específicas, que poderiam otimizar determinados processos, não puderam ser usadas para garantir que a API continuasse adaptável a diferentes cenários. Esse equilíbrio entre flexibilidade e especialização exigiu um planejamento cuidadoso na definição dos endpoints, modelos de dados e regras de negócio.

Como o sistema foi pensado para servir de base para diferentes contextos, algumas decisões arquiteturais foram tomadas sem considerar demandas específicas de cada possível uso. Isso significa que, em certos casos, será necessário realizar ajustes adicionais conforme a API for implementada em diferentes ambientes.

Além disso, a necessidade de manter a API modular e extensível limitou a complexidade de algumas implementações. Algumas funcionalidades avançadas, como regras de negócio mais detalhadas ou integrações específicas, não foram incluídas nesta versão inicial, pois poderiam restringir a capacidade de adaptação do sistema a outros cenários. Esse aspecto reforça a importância de futuras iterações e melhorias à medida que o sistema for sendo utilizado e novos requisitos forem identificados.

Um fator de impacto do projeto foi a falta de uma aplicação aprofundada dos princípios da Lei Geral de Proteção de Dados (LGPD). Necessário em um trabalho futuro o estudo da sua complexidade, regulamentações e busca de um conhecimento jurídico e técnico mais especializado, para essas diretrizes serem implementadas. Questões como o consentimento explícito dos pacientes para o tratamento de seus dados, a anonimização de informações sensíveis e a definição de políticas claras para retenção e descarte de dados ainda precisam ser aprimoradas. Essa limitação ressalta a necessidade de futuras revisões e aprimoramentos no sistema, garantindo que ele esteja totalmente alinhado com as exigências legais e assegure um alto nível de proteção e privacidade para os pacientes.

A ausência de uma interface gráfica integrada também limitou a validação, uma vez que o foco deste projeto foi exclusivamente o desenvolvimento da API. A falta de uma interface dificultou a validação direta da usabilidade por parte dos usuários finais. Esse trabalho faz parte de um projeto maior, que, devido à sua complexidade, foi dividido em duas partes. Enquanto este estudo se concentrou no desenvolvimento da API e na garantia de sua robustez, um outro trabalho paralelo está sendo desenvolvido para a construção de uma interface web que consumirá essa API, oferecendo uma experiência mais acessível e intuitiva aos profissionais de saúde.

5.7 Experiência no Processo

O processo de desenvolvimento não apenas proporcionou aprendizado significativo sobre a arquitetura Spring Boot e os princípios do DDD, mas também enfatizou a importância de boas práticas de gestão de projetos. A estruturação adequada dos testes e a colaboração periódica entre membros da equipe foram cruciais para garantir a entrega de um produto final de qualidade. Recomenda-se que futuras equipes adotem uma abordagem ágil e colaborativa para maximizar a eficiência e a qualidade do desenvolvimento.

Durante o processo, um dos principais desafios enfrentados foi estruturar corretamente os testes unitários de forma que cobrisse todos os *endpoints* sem gerar dependências

desnecessárias. A aplicação da metodologia AAA ajudou a permitir uma validação eficiente das regras de negócio implementadas. Também foi necessário compreender como lidar com mocks e dependências do Spring, garantindo que os testes fossem independentes e reproduzíveis.

Outra experiência relevante foi perceber a importância dos testes para a evolução contínua do sistema. Durante o desenvolvimento, os testes permitiram identificar e corrigir erros antecipadamente, reduzindo o risco de falhas no ambiente de produção. Além disso, a existência de uma base sólida de testes unitários possibilita a expansão da aplicação sem comprometer funcionalidades já implementadas, servindo como uma base para futuras customizações e otimizações.

Por fim, o processo evidenciou a necessidade de estudo contínuo. Embora a aplicação atenda aos requisitos iniciais, novas melhorias podem ser implementadas com base no uso real do sistema. A experiência adquirida reforçou a importância de boas práticas de desenvolvimento, testes automatizados e uma arquitetura bem planejada para garantir a qualidade e a manutenção eficiente do software.

6. CONCLUSÕES E TRABALHOS FUTUROS

Este projeto teve como objetivo a criação de uma *API* para uso de base para sistemas de anamnese hospitalar, voltado para digitalizar e otimizar o processo de anamnese e atendimento prévio em hospitais. O método tradicional, baseado em fichas de papel, se mostrou um fator limitante, tanto no armazenamento de informações quanto na eficiência do fluxo de trabalho. Com isso, surgiu a ideia de desenvolver uma solução que pudesse servir como base para o desenvolvimento de outras soluções. Soluções essas que buscam agilizar o atendimento e facilitar a consulta do histórico dos pacientes, contribuindo para uma gestão mais eficiente.

Em síntese, o desenvolvimento desta *API* para sistemas de anamnese busca significar a digitalização dos processos hospitalares, com potencial de melhorar a experiência tanto para profissionais de saúde quanto para pacientes. Espera-se que, com o uso para desenvolvimento de soluções e implantação em um ambiente real, a *API* possa contribuir para uma gestão mais ágil e precisa, oferecendo uma base para futuras inovações no setor hospitalar e inspirando melhorias contínuas na organização e qualidade do atendimento.

Para trabalhos futuros, sugere-se o consumo da *API* para desenvolver interfaces gráficas intuitivas. Isso irá facilitar a usabilidade para profissionais de saúde e pacientes, aplicando ao desenvolvimento de sistemas completos com as necessidades próprias da instituição de saúde que irá utilizar.

Outro aspecto importante para o aprimoramento da *API* é sua adequação ao protocolo HL7 (*Health Level Seven*), um padrão internacional para a interoperabilidade de sistemas de informação em saúde. A adoção desse protocolo permitirá a integração da *API* com outros sistemas hospitalares, como prontuários eletrônicos, dispositivos médicos e plataformas de telemedicina, garantindo a troca de informações entre diferentes instituições de saúde. Além disso, é importante o aprimoramento da *API* para que esteja alinhada às exigências da Lei Geral de Proteção de Dados (LGPD), assegurando que a coleta, o armazenamento e o compartilhamento de informações sensíveis dos pacientes sejam realizados de maneira ética e segura.

REFERÊNCIAS BIBLIOGRÁFICAS

- BONTEANU, Alexandru Marius; TUDOSE, Cătălin. Performance Analysis and Improvement for CRUD Operations in Relational Databases from Java Programs Using JPA, Hibernate, Spring Data JPA. **Applied Sciences**, v. 14, n. 7, p. 2743, 2024.
- CAMILO, Gabriel Magalhães Menezes. Sistema web para controle de vacinação de um hospital. 2019.
- CAMPOPIANO, Giovanni Teixeira. Importância de testes unitários no desenvolvimento web. 2022.
- CASAS, Sandra et al. Uses and applications of the OpenAPI/Swagger specification: a systematic mapping of the literature. In: **2021 40th International Conference of the Chilean Computer Science Society (SCCC)**. IEEE, 2021. p. 1-8.
- COSTEIRA, Elza Maria Alves. A arquitetura hospitalar pós pandemia: adequando hospitais aos novos tempos. **Revista Sustinere**, v. 9, n. 1, p. 398-404, 2021.
- DE RESENDE, Mateus Augusto Mendonça et al. What Did I Do?: Innovation in the Anamnesis Process. 2022.
- EVANS, Eric. **Domain-driven design: tackling complexity in the heart of software**. Addison-Wesley Professional, 2004.
- FAENI, Dewi Puspaningtyas; OCTAVIANY, Nabila Zahra. The Role of Management Information Systems in the Healthcare Industry in E-Health and E-Hospital Programs. **JURNAL RISET MANAJEMEN DAN EKONOMI**, v. 2, n. 3, p. 130-137, 2024.
- GARIMILLA, Madhu. The Art Of Api Design: Best Practices For Modern Software Development. **International Journal of Engineering and Technology Research (IJETR)**, v. 9, n. 2, p. 229-239, 2024.
- GEORGE, Bobby; WILLIAMS, Laurie. A structured experiment of test-driven development. **Information and software Technology**, v. 46, n. 5, p. 337-342, 2004.
- KOC, Hatice et al. UML diagrams in software engineering research: a systematic literature review. In: **Proceedings**. MDPI, 2021. p. 13.
- LANGR, Jeff; HUNT, Andy; THOMAS, Dave. **Pragmatic unit testing in java 8 with JUnit**. The Pragmatic Programmers, 2015.
- LUCKOW, Décio Heinzelmann; DE MELO, Alexandre Altair. **Programação Java para a WEB**. Novatec Editora, 2010.
- MARRA, Caroline et al. Addressing the challenges of integrating digital health technologies to measure patient-centred outcomes in clinical registries. **The Lancet Digital Health**, 2024.

MILLETT, Scott; TUNE, Nick. **Patterns, principles, and practices of domain-driven design**. John Wiley & Sons, 2015.

OFOEDA, Joshua; BOATENG, Richard; EFFAH, John. Application programming interface (API) research: A review of the past to inform the future. **International Journal of Enterprise Information Systems (IJEIS)**, v. 15, n. 3, p. 76-95, 2019.

PAIVA, Carlos Henrique Assunção; TEIXEIRA, Luiz Antonio. Reforma sanitária e a criação do Sistema Único de Saúde: notas sobre contextos e autores. **História, Ciências, Saúde-Manguinhos**, v. 21, n. 1, p. 15-36, 2014.

PONELAT, Josh; ROSENSTOCK, Lukas. **Designing APIs with Swagger and OpenAPI**. Simon and Schuster, 2022.

RIFIAL, Muhammad et al. Impact of Health System Usage, Patient Satisfaction, Information Quality, and Service Quality on Hospital Management Information System Utilization at Madani Regional General Hospital.

SANTOS, José Luís Guedes dos et al. Mudanças implementadas no ambiente de trabalho de enfermeiros na pandemia de COVID-19. **Revista Brasileira de Enfermagem**, v. 75, p. e20201381, 2021.

SILVA, Fernando Aparecido da. VisitSys: um sistema web para controle de visitantes e acompanhantes de um hospital. 2022.

VAISHNAVI, V.; KUECHLER, W. **Design Research in Information Systems**. 2009. Disponível em: <<http://desrist.org/design-research-in-information-systems>>. Acesso em: 18 out. 2011.

VARGAS, Marco Antonio; ALVES, Nathalia Guimarães; MREJEN, Matias. Ciência, tecnologia e inovação em tempos de pandemia: implicações da Covid-19. **Cadernos do desenvolvimento**, v. 16, n. 28, p. 145-172, 2021.

VARGO, Deedra et al. Digital technology use during COVID-19 pandemic: A rapid review. **Human Behavior and Emerging Technologies**, v. 3, n. 1, p. 13-24, 2021.

VENTURA, Magda Maria. O estudo de caso como modalidade de pesquisa. **Revista SoCERJ**, v. 20, n. 5, p. 383-386, 2007.

WANG, Na; WANG, Jinguo. Application of Information Engineering in Hospital Management. In: **ITM Web of Conferences**. EDP Sciences, 2019. p. 02005.