UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO

MARCELO AUGUSTO SOARES GOMES

ESTUDO DE CASO DE DESENVOLVIMENTO DE UMA APLICAÇÃO DE AGENDAMENTOS POR MEIO DE TECNOLOGIAS WEB E PRÁTICAS DE ENGENHARIA DE SOFTWARE

ALEGRE – ES
2024

MARCELO AUGUSTO SOARES GOMES

ESTUDO DE CASO DE DESENVOLVIMENTO DE UMA APLICAÇÃO DE AGENDAMENTOS POR MEIO DE TECNOLOGIAS WEB E PRÁTICAS DE ENGENHARIA DE SOFTWARE

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Bruno Vilela Oliveira

ALEGRE - ES
2024

Marcelo Augusto Soares Gomes

Estudo de caso de desenvolvimento de uma aplicação de agendamentos por meio de tecnologias web e práticas de Engenharia de Software

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 30 de setembro de 2024.

COMISSÃO EXAMINADORA

Prof. Dr. Bruno Vilela Oliveira
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Dayan de Castro Bissoli
Universidade Federal do Espírito Santo

Prof. MSc. Giuliano Prado de Morais Giglio
Universidade Federal do Espírito Santo

AGRADECIMENTOS

Primeiramente, gostaria de expressar minha profunda gratidão a Deus, cuja orientação e bênçãos foram fundamentais em cada etapa deste caminho acadêmico. Sua sabedoria e graça foram a luz que iluminou meu percurso, e sua presença constante foi a força que me impulsionou nos momentos desafiadores.

À minha família, em especial aos meus pais, quero dedicar palavras de apreço e reconhecimento. Seu amor incondicional, encorajamento e apoio constante foram pilares essenciais durante toda essa jornada. Cada conquista alcançada é também fruto do investimento de vocês em meu crescimento, e por isso, sou eternamente grato.

Aos meus irmãos em cristo, aos amigos que compartilharam comigo não apenas momentos de fé, mas também de amizade sincera, agradeço por serem uma fonte de inspiração e conforto espiritual. Suas orações e palavras de ânimo foram como um alicerce sólido em meio às turbulências, e por isso, cada vitória alcançada é também compartilhada com vocês.

Aos meus amigos de república e de trabalho, cuja jornada se entrelaçou com a minha, agradeço pela colaboração, pela troca de conhecimento e pela camaradagem. Juntos, enfrentamos desafios, celebramos conquistas e construímos memórias que levarei comigo como tesouros preciosos. Em resumo, esta conquista não é apenas minha, mas de todos que, de alguma forma, fizeram parte dessa trajetória. Que este trabalho represente não apenas o fechamento de um ciclo acadêmico, mas o início de muitos outros ciclos, nos quais continuarei a crescer, aprender e compartilhar.

Foco, força e fé.

RESUMO

A tecnologia tem se tornado cada vez mais importante no meio acadêmico, especialmente quando aplicada a sistemas que visam melhorar a eficiência e o bem-estar de alunos e funcionários. Este trabalho apresenta um estudo de caso sobre o desenvolvimento de uma aplicação web funcional, utilizando diversas tecnologias de front-end e back-end, com o objetivo de demonstrar o processo completo de desenvolvimento de software. A aplicação, desenvolvida para o Setor de Saúde e Assistência Social da Universidade Federal do Espírito Santo, oferece funcionalidades básicas de um sistema de agendamento de consultas, com funcionalidades de cadastro, visualização de horários e organização de atendimentos. Este estudo de caso destaca as etapas do processo de desenvolvimento, as ferramentas utilizadas, os desafios enfrentados e as soluções adotadas para criar um sistema que funcione de maneira ágil e segura.

Palavras-chave: Agendamento de Consultas. Sistema de Agendamento. Desenvolvimento de Sistema *Web*

ABSTRACT

Technology has become increasingly important in academia, especially when applied to systems aimed at improving the efficiency and well-being of students and staff. This paper presents a case study on the development of a functional web application, utilizing various front-end and back-end technologies, with the goal of demonstrating the complete software development process. The application, developed for the Health and Social Assistance Department of the Federal University of Espírito Santo, aims to simulate an appointment scheduling system, with functionalities for registration, viewing available times, and organizing appointments. This case study highlights the stages of the development process, the tools used, the challenges faced, and the solutions adopted to create a system that operates efficiently and securely.

Keywords: Appointment Scheduling. Scheduling System. Web System Development.

LISTA DE FIGURAS

Figura 1 - Tela inicial do site que oferece o Tua Agenda..... 24
Figura 2 - Tela inicial do sistema Amplimed..... 25
Figura 3 - Tela inicial do site que oferece o SimDoctor..... 26
Figura 4 - Tela inicial do site que oferece o eAgenda..... 27
Figura 5 - Diagrama de classes.....34
Figura 6 - Diagrama de caso de uso..... 35
Figura 7 - Modelo Lógico do banco de dados..... 37
Figura 8 - Arquitetura Cliente/Servidor..... 39
Figura 9 - Criação de tabelas com o Prisma..... 41
Figura 10 - Estrutura de pastas do back-end..... 41
Figura 11 - Prisma studio.....43
Figura 12 - Endpoints de consulta.....44
Figura 13 - Arquivo Consulta.model.js..... 45
Figura 14 - Arquivo consulta.controller.js..... 46
Figura 15 - Demonstração da funcionalidade Autenticar usuário..... 47
Figura 16 - Arquivo Auth.js, checkAdm..... 48
Figura 17 - Estrutura de pastas front-end admin.....49
Figura 18 - Arquivo axiosConfig.js.....50
Figura 19 - Arquivo index.js Routes..... 51
Figura 20 - Tela de login paciente..... 53
Figura 21 - Tela de criação de conta Paciente..... 54
Figura 22 - Tela de solicitação de consulta..... 55
Figura 23 - Tela de visualização de consultas.....55
Figura 24 - Tela de login administrador..... 56
Figura 25 - Tela de listagem de consultas..... 57
Figura 26 - Tela de detalhes/editar consulta.....57
Figura 27 - Teste de endpoint no insomnia..... 60
Figura 28 - Teste POST no endpoint /login..... 61
Figura 29 - Agendamento de consultas(SASAS)..... 79

LISTA DE TABELAS

Tabela 1 - Tabela usuario..... 38
Tabela 2 - Tabela com os erros e soluções..... 58
Tabela 3 - Plano de testes..... 61
Tabela 4 - Testes de usabilidade..... 70
Tabela 5 - Formulário 1(2024): Identificação de estudante..... 80
Tabela 6 - Formulário 2 (2024) : Questões preparatórias para atendimento psicológico..... 81
Tabela 7 - Formulário 3 (2024) : Questões preparatórias para atendimento psicológico..... 82
Tabela 8 - Mensagens dos casos de uso..... 88
Tabela 9 - Tabela paciente..... 89
Tabela 10 - Tabela consulta..... 89
Tabela 11 - Tabela profissional..... 90
Tabela 12 - Tabela notificacao..... 91
Tabela 13 - Tabela compromisso..... 91

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface*

ECDSA - *Elliptic Curve Digital Signature Algorithm*

HMAC - *Hash-based Message Authentication Code*

HTTP – *Hypertext Transfer Protocol*

JWT – *Json Web Token*

ORM – *Object Relational Mapper*

RSA - *Rivest-Shamir-Adleman*

SGBD – Sistema de Gerenciamento de Banco de Dados

SQL – *Structured Query Language*

TCC – Trabalho de Conclusão de Curso

UI – *User Interface*

UFES – Universidade Federal do Espírito Santo

UML – *Unified Modeling Language*

JSON - JavaScript *Object Notation*

UI - User Interface

UX - User Experience

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Problema e sua Importância 13
1.2 Objetivos 14
1.2.1 Objetivo Geral 14
1.2.2 Objetivos Específicos 14
2 REVISÃO DE LITERATURA 16
2.1 Desenvolvimento Web 16
2.2 Tecnologias envolvidas 16
2.2.1 VueJS 16
2.2.2 Vuetify 17
2.2.3 Vuex 18
2.2.4 Axios 18
2.2.5 Node.JS 19
2.2.6 JavaScript 19
2.2.7 Express 20
2.2.8 JWT 20
2.2.9 PostgreSQL 21
2.2.10 GitHub 21
2.2.11 Paradigma de Orientação a Objetos 22
2.3 Sistemas relacionados com a aplicação a ser desenvolvida com o estudo de casos 23
2.3.1 Tua Agenda 23
2.3.2 Amplimed 24
2.3.3 SimDoctor 25
2.3.4 EAgenda 26
2.4 Trabalhos Relacionados 27
2.4.1 Sistema de Agendamento 27
2.4.1.1 Sistema para agendamento de serviços 28
2.4.1.2 Sistema de Agendamento Universal 28
2.4.1.3 Agendamento online de doações de sangue durante a pandemia 28
2.4.1.4 Desenvolvimento de um sistema de marcação online de consultas 29
3 METODOLOGIA 30
3.1 Usuários do Sistema 30
3.2 Análise de requisitos 31
3.2.1 Requisitos Funcionais 31
3.2.2 Requisitos Não Funcionais 32

3.3 Modelagem do Sistema....32
3.3.1 Diagrama de classes....33
3.3.2 Casos de uso....34
3.3.3 Narrativas de caso de uso....35
3.3.3.1 UC 1 - Autenticar Usuário....36
3.4 Modelagem do banco de dados....36
3.5 Descrição das tabelas do banco de dados....37
4 DESENVOLVIMENTO....39
4.1 Arquitetura....39
4.2 Back-End....40
4.2.1 Estrutura do Projeto....41
4.2.2 Conexão com banco de dados....42
4.2.3 Endpoints....43
4.2.4 Models....44
4.2.5 Controllers....45
4.2.6 Autenticação....46
4.3 Front-end....48
4.3.1 Estrutura do projeto....48
4.3.2 Comunicação com o servidor....50
4.3.3 Navegação....50
4.4 Views....52
4.4.1 Tela de login paciente....52
4.4.2 Tela de criação de conta Paciente....53
4.4.3 Tela de Solicitar Consulta....54
4.4.4 Tela de visualização de consultas....55
4.4.5 Tela de login administrador....56
4.4.6 Tela de listagem de consultas....56
4.4.7 Tela de editar consulta....57
5 TESTES....58
5.1 Testes de sistema....58
5.2 Teste de Desenvolvimento....59
5.3 Teste de Usuário....67
5.4 Teste de Usabilidade....69
6 CONSIDERAÇÕES FINAIS E CONCLUSÕES....72
6.1 Trabalhos Futuros....73
REFERÊNCIAS....75
ANEXO A - FORMULÁRIOS PARA SOLICITAÇÃO DE AGENDAMENTO....79
APÊNDICE B - NARRATIVAS DE CASO DE USO....86
APÊNDICE C - DESCRIÇÃO DAS TABELAS DO BANCO DE DADOS....89

1 INTRODUÇÃO

O desenvolvimento web está em constante evolução e desempenha um papel crucial na criação de soluções tecnológicas para atender às crescentes demandas da sociedade moderna. Utilizando uma combinação de linguagens de programação, frameworks e práticas de design, desenvolvedores podem construir aplicações web que são eficientes, seguras e escaláveis. Ferramentas como Node.js e Vue.js, por exemplo, proporcionam uma base robusta para o desenvolvimento de sistemas dinâmicos e responsivos, garantindo uma experiência fluida para os usuários (GARCIA, 2019).

Além disso, o desenvolvimento de aplicações web envolve uma gama diversificada de tecnologias, como HTML, CSS, JavaScript, PHP, além de frameworks como React, Angular, Vue e Laravel. Essas ferramentas possibilitam a criação de diversos tipos de aplicações, como sistemas de gerenciamento de conteúdo (CMS), plataformas de e-commerce e sistemas de agendamento. O domínio dessas tecnologias é fundamental para atender às necessidades específicas dos usuários e clientes, resultando em soluções personalizadas e eficientes(CARREIRAEMTI, 2023).

No ambiente universitário, muitos estudantes têm hábitos de vida não saudáveis, muitas vezes relacionados ao estresse acadêmico. Segundo Da Costa (2017), foram observados índices significativos de excesso de peso, sedentarismo, tabagismo e outras condições entre universitários. Profissionais da saúde no ambiente acadêmico, como médicos e psicólogos, desempenham um papel fundamental no tratamento e prevenção desses problemas.

Entretanto, o agendamento de consultas pode ser um processo complicado e burocrático, especialmente quando feito por telefone ou presencialmente. Um sistema de agendamento de consultas online surge como uma solução prática para facilitar esse processo e melhorar o acesso aos serviços de saúde.

O presente trabalho se propõe a desenvolver uma aplicação web funcional para o agendamento de consultas na Seção de Atenção à Saúde e Assistência Social da Universidade Federal do Espírito Santo, em Alegre. Essa aplicação será

construída utilizando tecnologias web modernas, aplicando conhecimentos adquiridos ao longo do curso para criar um sistema eficiente e intuitivo.

1.1 Problema e sua Importância

A crescente demanda por sistemas web reflete a transformação digital acelerada em diversos setores, nos quais empresas buscam maior visibilidade, eficiência e acessibilidade. Disponibilizando cada vez mais suas operações nos ambientes online, empresas têm adotado sistemas web para otimizar processos, melhorar a comunicação e facilitar o acesso a dados em tempo real. Essa transição é impulsionada pela necessidade de atender o crescente número de usuários de dispositivos móveis e que se beneficiam da conveniência do acesso remoto (BENLIAN, 2018). Além disso, o uso de sistemas web reduz custos operacionais, pois elimina a necessidade de infraestrutura robusta local, dependendo apenas de servidores online e das práticas de segurança (VIAL, 2019).

Outro aspecto que aumenta a procura por sistemas web é o crescimento do comércio eletrônico e de serviços online. A pandemia da COVID-19 acelerou essa tendência, pois muitas empresas tiveram que adaptar suas operações rapidamente para atender a uma clientela que não podia visitar estabelecimentos físicos (GARCIA, 2021). Além disso, a escalabilidade dos sistemas web permite que as empresas ampliem seus serviços de forma mais rápida e com menos investimento inicial, o que se torna uma vantagem competitiva (VIAL, 2019). Assim, a demanda por sistemas web deve continuar em ascensão, especialmente em setores como educação, saúde e finanças, onde a digitalização dos serviços é essencial para atender à crescente demanda do público conectado.

Dada a crescente tendência e importância do desenvolvimento web, o presente trabalho se baseia em um estudo de caso voltado ao desenvolvimento de um sistema web por meio de tecnologias voltadas ao desenvolvimento desse tipo de sistema e no uso de conhecimentos de engenharia de software no que diz respeito à algumas atividades básicas como análise, design, documentação e testes. O sistema oferecerá funcionalidades básicas de agendamento de consultas, com alguns requisitos obtidos a partir das necessidades da Seção de Atenção à Saúde e Assistência Social (SASAS) da Universidade Federal do Espírito Santo (UFES)

(UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, 2024). Atualmente, o processo de agendamento de consultas no SASAS é realizado por meio de formulários do Google Forms, o que tem gerado uma série de problemas, especialmente devido ao preenchimento incorreto por parte dos usuários. Essas falhas comprometem a eficiência do agendamento, ocasionando atrasos e dificultando a gestão das consultas pela equipe administrativa.

Diante desse cenário, este estudo de caso torna-se relevante ao propor uma solução tecnológica que visa não apenas resolver esses problemas operacionais, mas também demonstrar o uso de modernas tecnologias web no desenvolvimento de um sistema de agendamento funcional. O formulário do Google Forms mencionado pode ser consultado no Anexo A.

1.2 Objetivos

A seguir serão apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é realizar um estudo de caso sobre o desenvolvimento de uma aplicação web funcional, explorando as etapas do processo de criação e implementação de tecnologias de front-end e back-end. O estudo foca em demonstrar como essas tecnologias podem ser aplicadas para criar um sistema de agendamento online, enfatizando o processo de desenvolvimento, a tomada de decisões técnicas e os desafios enfrentados.

1.2.2 Objetivos Específicos

De modo a alcançar o objetivo geral, destacam-se os seguintes objetivos específicos:

1. Realizar a revisão bibliográfica sobre tecnologias de desenvolvimento de aplicações web e estudos correlacionados.
2. Analisar e definir as principais tecnologias de front-end e back-end adequadas para o desenvolvimento da aplicação.

3. Realizar o levantamento, análise e documentação de requisitos para a aplicação do estudo de casos.
4. Implementar funcionalidades básicas da aplicação, como cadastro de usuários, visualização de agendamentos e outras funcionalidades essenciais, destacando os desafios técnicos enfrentados.
5. Avaliar o desempenho e a usabilidade da aplicação, com base nos conceitos de boas práticas de desenvolvimento de software.

2 REVISÃO DE LITERATURA

Nesta seção, é apresentada uma revisão da literatura relacionada com o tema do presente trabalho.

2.1 Desenvolvimento Web

O desenvolvimento de aplicações web tem se tornado cada vez mais sofisticado com o uso de frameworks modernos. Flanagan (2020) observa que o JavaScript, juntamente com tecnologias como *Node.js* e *Vue.js*, oferece uma base robusta para o desenvolvimento de aplicações web interativas, melhorando a experiência do usuário. *Node.js* se destaca por permitir a execução de código JavaScript no lado do servidor, enquanto *Vue.js* facilita o desenvolvimento de interfaces dinâmicas e modulares (FLANAGAN, 2020).

Segundo Larsen e Grace (2019), o uso dessas tecnologias tem sido amplamente adotado devido à sua flexibilidade e à capacidade de criar aplicações de uma única página (SPA, do inglês *Single Page Application*), melhorando o desempenho e a interatividade das plataformas. Tais frameworks são amplamente utilizados na construção de interfaces de usuário modernas, focadas em experiência responsiva e escalável.

2.2 Tecnologias envolvidas

Nesta subseção serão apresentadas as tecnologias e ferramentas que foram utilizadas.

2.2.1 VueJS

O desenvolvimento *web front-end* refere-se à prática de construir e manter os aspectos visuais e interativos de um site, como *layout*, *design* e experiência do usuário. Envolve criar e trabalhar com código HTML, CSS e JavaScript para criar páginas da *web* e aplicativos que são exibidos em um navegador da *web.* Desenvolvedores *front-end* são os profissionais responsáveis por garantir que um sistema seja visualmente atraente, amigável e responsivo, bem como por construir e testar a interface do usuário de aplicativos da *web* (EBAC, 2023).

Segundo Souto (2022), *front-end* é um termo utilizado para se referir a parte visual de um site, plataforma ou um sistema de computador. É tudo aquilo que podemos interagir e que é responsável por toda a interface gráfica de um sistema.

Vue.js é um *framework JavaScript* de código aberto para a criação de interfaces de usuário (UI). Ele foi criado por Evan You e é mantido por uma equipe de desenvolvedores ativos e uma grande comunidade de usuários. Vue.js é projetado para ser um *framework* de construção de interface de usuário fácil de usar, mas ainda poderoso, com uma curva de aprendizado suave. Ele utiliza uma sintaxe HTML especial, chamada de "*template*", que permite aos desenvolvedores escrever código HTML de maneira mais concisa e declarativa. Vue.js também tem suporte a componentes, o que permite que os desenvolvedores dividam sua interface de usuário em partes menores e reutilizáveis(VUEJS, 2023).

Segundo Filipova (2016), em seu livro de título *Learning* Vue.js 2, Evan You, criador do Vue.js se deparou com uma necessidade que sua equipe tinha, que era a criação rápida de protótipos de uma interface de usuário. Se escrevia muito HTML repetido e isso claramente consumia muito o tempo de sua equipe, assim sendo, Evan procurou por alternativas para resolver este problema, e não encontrou nenhuma que atendesse exatamente a sua demanda, para sua surpresa. Por fim, Evan criou o Vue.js baseado nessa premissa, um *framework* que ajuda na prototipagem rápida, oferecendo uma maneira fácil e flexível de ligação de dados reativa e de componentes reutilizáveis.

2.2.2 Vuetify

O Vuetify é um framework de componentes baseado no Material Design, utilizado para desenvolver interfaces de usuário responsivas e consistentes em aplicações Vue.js (VUETIFY, 2023).

Ele se destaca por ser baseado no Material Design, o que garante que as interfaces sigam diretrizes modernas de design visual, proporcionando uma experiência de usuário (UX) padronizada e consistente. A escolha do Vuetify para este projeto foi motivada pela sua vasta gama de componentes prontos para uso, como botões, modais, grids, tabelas e menus, que simplificam a criação de interfaces dinâmicas e complexas. Além disso, a biblioteca oferece suporte nativo

para responsividade, o que é essencial para garantir que a aplicação funcione corretamente em diferentes dispositivos, desde desktops até smartphones.

2.2.3 Vuex

Vuex é um sistema de gerenciamento de estado associado à biblioteca Vue.js. Ele atua como um repositório central para todos os componentes do aplicativo, com diretrizes que asseguram que as alterações no estado sejam feitas de maneira controlada e previsível (VUEX, 2023).

No projeto, o Vuex desempenhou um papel fundamental ao permitir o compartilhamento de dados entre diferentes componentes sem a necessidade de repassar as informações diretamente de um componente pai para os filhos, o que poderia gerar código confuso e difícil de manter. Em vez disso, o *Vuex* armazena o estado da aplicação em uma store centralizada, que pode ser acessada e manipulada por qualquer componente dentro da aplicação, mantendo a consistência dos dados de forma organizada.

O *Vuex* se baseia em quatro conceitos principais: *state*, *getters*, *mutations* e *actions*. O *state* é onde os dados são armazenados centralmente. Os *getters* são usados para acessar o estado e realizar cálculos ou manipulações leves. As *mutations* são funções responsáveis por modificar o estado, garantindo que essas alterações sejam feitas de forma previsível e rastreável. Já as actions lidam com operações assíncronas, como requisições a APIs, e chamam as mutations para atualizar o estado conforme necessário.

Assim, a escolha do *Vuex* foi essencial para garantir a organização e a manutenção do estado da aplicação, proporcionando uma base sólida e escalável para o gerenciamento de dados ao longo do projeto.

2.2.4 Axios

Axios é uma biblioteca para fazer requisições HTTP, amplamente utilizada em projetos *JavaScript* tanto no navegador quanto em ambientes *Node.js.* Ela permite que os desenvolvedores realizem chamadas de API e manipulem respostas de

maneira simples e eficiente, utilizando *promises* para um controle mais fácil de operações assíncronas (AXIOS, 2022).

Com uma API intuitiva e suporte para funcionalidades como interceptação de requisições e respostas, cancelamento de requisições e configuração global de parâmetros, o *axios* facilita a comunicação com servidores e o gerenciamento de dados. Além disso, sua compatibilidade com a maioria dos navegadores e seu suporte a *JSON* tornam a integração com serviços *web* e APIs uma tarefa mais direta e menos propensa a erros.

2.2.5 Node.JS

O Node.js é uma plataforma baseada em JavaScript que permite a execução do código do lado do servidor, utilizando o motor V8 do Google Chrome. Com uma arquitetura não bloqueante e orientada a eventos, o Node.js é ideal para construir aplicações escaláveis e de alto desempenho, sendo amplamente utilizado em sistemas que requerem uma comunicação em tempo real. Essa eficiência e a capacidade de gerenciar múltiplas conexões simultâneas fazem do Node.js uma escolha popular entre desenvolvedores para o desenvolvimento de APIs e servidores web (NODEJS, 2023).

Para o desenvolvimento back-end do sistema de agendamento proposto, foi utilizado o Node.js, aproveitando seu suporte a JavaScript assíncrono e Promises. Esses recursos permitem a execução de operações assíncronas, como chamadas a bancos de dados ou APIs, sem bloquear o fluxo do programa, garantindo maior eficiência no tratamento de múltiplas requisições simultâneas.

2.2.6 JavaScript

O JavaScript é uma linguagem de programação amplamente utilizada para desenvolvimento web, tanto no lado do cliente quanto no servidor. Ela é conhecida por sua capacidade de criar interatividade nas páginas *web* e sua natureza assíncrona, que permite realizar operações como requisições de rede de forma não bloqueante, melhorando a experiência do usuário e o desempenho das aplicações (JAVASCRIPT, 2023).

Para o desenvolvimento do sistema de agendamento proposto, foi utilizado JavaScript, uma linguagem versátil que pode ser empregada tanto no front-end quanto no back-end. Essa flexibilidade permite utilizar a mesma linguagem em diferentes partes do sistema, facilitando a comunicação entre elas e a reutilização de código. A capacidade do JavaScript de se integrar facilmente a diversas bibliotecas e frameworks, como o Vue e Node.js, contribui para a criação da aplicação, aprimorando a eficiência do desenvolvimento.

2.2.7 *Express*

O *Express.js* é um framework web minimalista para Node.js que facilita o desenvolvimento de aplicações e APIs. Ele fornece uma série de funcionalidades essenciais, como roteamento, manipulação de requisições e middleware, permitindo que os desenvolvedores criem servidores robustos de forma rápida e eficiente. Com sua arquitetura flexível, o Express é amplamente utilizado na construção de aplicações web escaláveis, sendo uma escolha popular entre desenvolvedores devido à sua simplicidade e ao vasto ecossistema de middleware que o acompanha.(EXPRESS, 2023).

No *back-end* deste projeto, o *Express* foi escolhido por sua simplicidade e eficiência ao lidar com rotas e requisições HTTP. Ele permite organizar o código de forma modular, através do uso de rotas para diferentes endpoints da aplicação. Essa abordagem melhora a legibilidade e a manutenção do código, cada rota é responsável por lidar com uma funcionalidade específica, como por exemplo, a criação, leitura, atualização e exclusão de dados.

2.2.8 JWT

O *JSON Web Token* (JWT) é um padrão aberto que define um modo compacto e autônomo para a transmissão segura de informações entre partes como um objeto JSON. O JWT é amplamente utilizado para autenticação e autorização em aplicações web, permitindo que usuários façam login e acessem recursos de maneira segura. Sua estrutura é composta por três partes: cabeçalho, *payload* e assinatura, o que garante que as informações não sejam alteradas durante a

transmissão. Além disso, o uso de *tokens* reduz a necessidade de armazenar sessões no servidor, contribuindo para a escalabilidade das aplicações. Devido à sua simplicidade e segurança, o JWT se tornou uma escolha popular entre desenvolvedores para gerenciar sessões de usuários (JWT, 2023).

No projeto, o JWT foi utilizado para implementar uma solução de autenticação baseada em *tokens*. Após o usuário realizar o login com sucesso, um token JWT é gerado e enviado ao cliente. Este *token* é então incluído nas requisições subsequentes ao servidor, geralmente no cabeçalho de autorização, permitindo que o *back-end* verifique a autenticidade do usuário e autorize o acesso a recursos protegidos. O JWT contém três partes principais: o cabeçalho (*header*), o corpo (*payload*) e a assinatura (*signature*). O cabeçalho especifica o algoritmo de assinatura usado, o corpo contém as reivindicações (claims) sobre o usuário e a assinatura garante que o token não foi alterado.

2.2.9 PostgreSQL

O PostgreSQL é amplamente reconhecido por sua conformidade com os padrões SQL e sua capacidade de estender suas funcionalidades através de um sistema modular. Suportando uma variedade de tipos de dados, incluindo JSON, ele permite a gestão eficiente de dados estruturados e não estruturados, tornando-se uma solução versátil para diferentes tipos de aplicações POSTGRESQLGLOBALDEVELOPMENTGROUP, 2023).

O postgres foi a escolha para ser o SGBD desta aplicação, e teve como principal função, a administração dos dados para o funcionamento do sistema.

2.2.10 GitHub

O GitHub é uma plataforma de hospedagem de código-fonte que utiliza o sistema de controle de versão Git, permitindo que desenvolvedores colaborem em projetos de software de maneira eficiente. Através do GitHub, é possível gerenciar versões de arquivos, registrar alterações e realizar o acompanhamento de problemas, facilitando a colaboração entre equipes distribuídas. Além de ser uma ferramenta essencial para desenvolvedores, o GitHub oferece recursos como pull

requests, forks e GitHub Actions, que permitem automatizar fluxos de trabalho e integrar testes contínuos. Com uma vasta comunidade e um ecossistema rico de projetos open source, o GitHub se tornou um padrão na indústria de desenvolvimento de software, promovendo a inovação e a transparência (GITHUB, 2022). O GitHub foi essencial para o desenvolvimento deste projeto. Um repositório foi criado com o objetivo de armazenar as versões do código.

As tecnologias selecionadas para o projeto foram escolhidas com base na experiência prévia do autor. A familiaridade com cada uma delas não apenas facilita o processo de desenvolvimento, mas também garante uma implementação mais eficiente. Essa escolha reflete a intenção de utilizar ferramentas que possibilitem um melhor aproveitamento dos recursos disponíveis.

2.2.11 Paradigma de Orientação a Objetos

A orientação a objetos é um paradigma de desenvolvimento de software que trouxe avanços significativos, especialmente na organização e reutilização de código. Uma linguagem de programação orientada a objetos deve fornecer suporte a conceitos-chave como abstração, encapsulamento, herança e polimorfismo, que são fundamentais para o design de sistemas complexos e robustos (SILVA et al., 2019).

Entre os principais conceitos da programação orientada a objetos estão:

- **Classe:** Representa um conjunto de objetos com características e comportamentos comuns. Funciona como um modelo ou gabarito, descrevendo os atributos e operações dos objetos.
- **Objeto:** Uma instância de uma classe, sendo uma entidade específica que reflete o domínio do problema no contexto do software em execução.
- **Métodos:** São as implementações das ações ou operações que um objeto pode realizar. Eles definem o comportamento dos objetos.
- **Atributos:** Referem-se às propriedades ou dados que uma classe ou suas instâncias devem manter. Para os objetos, os atributos representam o estado ou as informações que eles armazenam.

- **Encapsulamento:** Trata-se da prática de restringir o acesso a certos atributos ou métodos de um objeto, garantindo que apenas as operações definidas pela própria classe possam manipulá-los, protegendo a integridade dos dados.
- **Polimorfismo:** Permite que diferentes objetos respondam de forma distinta à mesma mensagem ou método, proporcionando flexibilidade no comportamento dos objetos.
- **Herança:** Permite que uma classe herde as características (atributos e métodos) de outra, formando uma hierarquia de classes. Isso facilita o reaproveitamento de código e a extensão das funcionalidades.

2.3 Sistemas relacionados com a aplicação a ser desenvolvida com o estudo de casos

Diversos sistemas web disponíveis no mercado oferecem soluções inovadoras para o desenvolvimento de aplicações interativas e dinâmicas. Nos próximos tópicos, serão destacados alguns desses sistemas, que servirão como inspiração e fundamento para o desenvolvimento da aplicação proposta, permitindo uma compreensão mais profunda das funcionalidades disponíveis e das práticas adotadas no campo do desenvolvimento web.

2.3.1 Tua Agenda

O Tua Agenda é um aplicativo mobile, feito para Android e IOS, que oferece funcionalidades como agendamentos online, envio de lembretes e gerenciamento de múltiplas agendas, permitindo uma gestão eficiente de compromissos e faturamento para profissionais de diversas áreas (TUAAGENDA, 2024). A Figura 1 a seguir apresenta a tela inicial do site que oferece o sistema Tua Agenda.

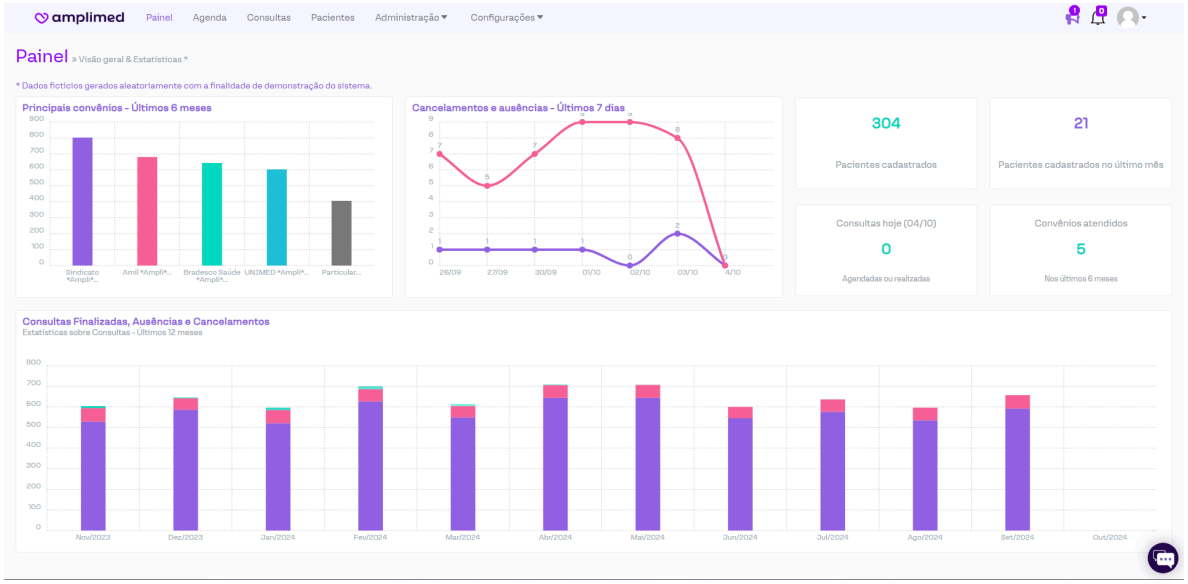
Figura 1 - Tela inicial do site que oferece o Tua Agenda

Fonte: (TUAAGENDA, 2024)

2.3.2 Amplimed

O Amplimed é um sistema de gestão desenvolvido para clínicas e consultórios, oferecendo soluções que incluem prontuário eletrônico, agendamento online e faturamento TISS. Com o objetivo de melhorar a eficiência e a qualidade do atendimento na saúde, o Amplimed também permite a gestão financeira das clínicas, possibilitando o controle de contas a pagar e a receber. Além disso, a plataforma tem se destacado pela facilidade de uso e pela integração de dados, visando aprimorar a experiência dos profissionais de saúde e seus pacientes (AMPLIMED, 2024). A Figura 2 a seguir, mostra a tela inicial do sistema, usando uma conta teste que foi criada para este trabalho.

Figura 2 - Tela inicial do sistema Amplimed



Fonte: (AMPLIMED, 2024)

2.3.3 SimDoctor

O SimDoctor é uma plataforma desenvolvida para otimizar o gerenciamento de consultas médicas, permitindo agendamentos, controle de atendimentos e uma melhor organização da agenda dos profissionais de saúde. Segundo informações disponíveis no site oficial, “o SimDoctor oferece um sistema eficiente e prático para a gestão de consultórios, integrando funcionalidades que facilitam o dia a dia dos médicos e de suas equipes" (SIMDOCTOR, 2024). Essa ferramenta se destaca por sua interface intuitiva e pelas soluções que proporciona para a área da saúde, visando melhorar a experiência tanto para os profissionais quanto para os pacientes. A seguir, na Figura 3, é apresentada a tela inicial do site que oferece o SimDoctor.

Figura 3 - Tela inicial do site que oferece o SimDoctor

Fonte:(SIMDOCTOR, 2024)

2.3.4 EAgenda

O eAgenda é uma plataforma online que permite o agendamento de compromissos e eventos, oferecendo recursos de integração com ferramentas como Google Agenda e Zoom, além de personalização de formulários e notificações automáticas por e-mail e SMS. Essa solução é voltada para diversos setores e facilita a gestão de equipes e de clientes, proporcionando um controle eficiente de agendas e filas de espera. A ferramenta pode ser personalizada conforme a necessidade de cada negócio, otimizando a organização e o atendimento ao cliente (EAGENDA, 2024). A seguir, na Figura 4, é apresentado o site oficial que oferece o eAgenda.

Figura 4 - Tela inicial do site que oferece o eAgenda

Fonte: (EAGENDA, 2024)

2.4 Trabalhos Relacionados

Na literatura, foram identificados alguns trabalhos relacionados ao tema deste estudo. A seguir, são apresentados alguns trabalhos que contribuíram para o desenvolvimento da pesquisa, fornecendo uma base teórica importante para a abordagem escolhida. Essas referências serviram como apoio na compreensão das questões abordadas e na definição das soluções propostas.

2.4.1 Sistema de Agendamento

Um sistema de agendamento é uma ferramenta projetada para gerenciar a programação de atividades e eventos dentro de uma organização. Ele é utilizado para marcar compromissos, reuniões e reservas de recursos, oferecendo uma maneira eficiente de organizar a agenda de profissionais e serviços. Entre as funcionalidades comuns estão a verificação de disponibilidade, agendamento de compromissos e envio de lembretes automáticos. Esses sistemas não apenas facilitam o agendamento, mas também ajudam a otimizar o uso de recursos, reduzir conflitos de horários e melhorar a experiência do cliente, permitindo que ele agende

a qualquer momento. Além disso, muitos sistemas oferecem recursos adicionais, como pagamentos online e integração com outras ferramentas de gerenciamento de tempo, o que pode aumentar a produtividade e proporcionar dados valiosos para a tomada de decisões estratégicas(COWORKINGMEDICO, 2022).

2.4.1.1 Sistema para agendamento de serviços.

De acordo com Kieras (2019), um sistema de informação de agendamento é essencial para profissionais que dependem de agendamentos prévios, como médicos, cabeleireiros e massagistas, pois melhora a gestão e o controle dos serviços oferecidos. Essa solução não apenas facilita a divulgação dos serviços, mas também aprimora a experiência do cliente, permitindo a busca por serviços específicos em uma determinada localidade. Com recursos que possibilitam a comparação de preços, detalhes dos serviços e disponibilidade de horários, o sistema torna-se uma ferramenta valiosa para profissionais e clientes. A utilização de tecnologias como Laravel, Angular e Ionic proporciona uma plataforma robusta e acessível. Assim, o sistema contribui significativamente para otimizar o atendimento e a satisfação do cliente.

2.4.1.2 Sistema de Agendamento Universal.

Siqueira e Evangelisti (2004) propõem uma solução informatizada de Agendamento Universal, que permite a visualização por meio de uma única interface de todas as vagas disponíveis na área auxiliar da Agenda dos Profissionais de Saúde. Essa abordagem visa otimizar o processo de agendamento, tornando-o mais acessível e eficiente tanto para os profissionais quanto para os pacientes. Com essa solução, espera-se reduzir a ociosidade dos profissionais e melhorar a experiência do usuário, garantindo um atendimento mais ágil e organizado.

2.4.1.3 Agendamento online de doações de sangue durante a pandemia

ANEGAWA et al. (2020) relata uma experiência de satisfação de usuários de um sistema de agendamento para doação de sangue no Hemocentro Regional de Londrina-PR, durante a pandemia COVID -19. O agendamento *online* foi instituído para evitar aglomerações e fornecer segurança aos doadores e se mostrou eficaz para um atendimento de fluxo constante. Ajudou consideravelmente na pré triagem

dos candidatos, resultando em uma grande quantidade de candidatos com menor frequência de inaptidão clínica do que o habitual. Com isso, pôde-se perceber que um sistema de agendamento contribui positivamente para o sucesso de agendamentos feitos no Hemocentro Regional de Londrina.

2.4.1.4 Desenvolvimento de um sistema de marcação online de consultas

Olaposi et al. (2022) desenvolveu um sistema de informação que permite aos estudantes agendar consultas com médicos universitários remotamente usando um dispositivo móvel, o que acarreta na redução do tempo de espera no centro de saúde universitário. Os resultados obtidos ao utilizar a ferramenta, mostrou que foi possível resolver problemas do método manual, aumentando a produtividade dos facilitadores de saúde e aumentando a satisfação dos universitários.

Ao realizar pesquisas sobre os temas em questão e sobre sistemas já existentes no mercado, foram identificadas as funcionalidades que um sistema de agendamento pode oferecer, bem como diferentes abordagens que serviram como base para o desenvolvimento deste trabalho. Esses estudos proporcionaram direções importantes na escolha de metodologias e ferramentas, além de auxiliar na definição de práticas adequadas ao desenvolvimento do sistema. A análise dos trabalhos relacionados possibilitou a adaptação das soluções às necessidades específicas do projeto.

3 METODOLOGIA

A metodologia deste trabalho concentrou-se na utilização de tecnologias web e práticas de engenharia de software para demonstrar a criação de uma aplicação web. O processo iniciou-se com uma análise das atividades necessárias para o desenvolvimento do sistema, seguida por uma revisão das tecnologias e ferramentas relevantes, como Node.js, Vue.js e Express. Essa pesquisa possibilitou a organização das etapas do desenvolvimento e a identificação das melhores práticas para a implementação de um sistema funcional.

Subsequentemente, foram avaliadas as necessidades específicas da Seção de Atenção à Saúde e Assistência Social (SASAS) por meio de entrevistas e coleta de dados locais, com o objetivo de reconhecer os desafios e requisitos para o desenvolvimento de um sistema que respondesse às demandas reais da instituição.

Nesse contexto, a escolha do paradigma de orientação a objetos destacou-se como a mais adequada, em razão de sua capacidade de organizar o software de maneira modular e flexível.

Com base nas necessidades identificadas, foi possível definir a abordagem técnica mais adequada para o desenvolvimento do sistema, que incluiu a realização de testes para garantia de qualidade. Esses testes abrangeram o teste de desenvolvimento, testes de usabilidade e testes com usuários.

A modelagem do domínio foi representada por meio de diagramas de classes e diagramas de casos de uso, facilitando a visualização das interações e comportamentos dos componentes do sistema.

3.1 Usuários do Sistema

No desenvolvimento de *software*, é de grande importância identificar quem são as partes interessadas. De acordo com Sommerville (2011), *stakeholders* são todas as pessoas ou organizações que estão ativamente envolvidas no projeto ou cujos interesses podem ser positiva ou negativamente afetados pela execução ou conclusão do projeto. Os *stakeholders* podem ser classificados em internos, como funcionários, gerentes e acionistas, e externos, como clientes, fornecedores, governo e comunidade.

O gerenciamento adequado desses grupos é essencial para garantir que suas necessidades e expectativas sejam atendidas, o que pode resultar em melhor comunicação.

3.2 Análise de requisitos

No desenvolvimento de sistemas, uma atividade importante é a análise de requisitos, pois trata-se do processo de derivação dos requisitos do sistema por meio da observação de sistemas existentes, discussões com usuários potenciais e compradores, análise de tarefas, entre outros métodos. Esse processo pode incluir o desenvolvimento de um ou mais modelos e protótipos do sistema, que auxiliam o analista a compreender melhor o sistema que será especificado (SOMMERVILLE, 2011).

É nessa fase que é possível identificar, juntamente com as partes interessadas do sistema (*stakeholders*), a especificação, refinamento e negociação dos requisitos, permitindo assim determinar como será a interface para o usuário, criação de testes para qualificar o *software*.

Com a definição do sistema e o levantamento dos requisitos, foram selecionadas as metodologias e ferramentas adequadas para o desenvolvimento.

3.2.1 Requisitos Funcionais

Os requisitos de um sistema podem ser descrições sobre o que o sistema deve fazer, os serviços que ele fornece, e as restrições em sua operação. Esses requisitos refletem as necessidades dos *stakeholders* para um sistema que atenda a uma finalidade específica, como controlar um dispositivo, fazer um pedido ou recuperar informações. Encontrar, analisar, documentar e controlar esses serviços e restrições é chamado de elicitação de requisitos. Os requisitos funcionais de um sistema definem o que o sistema deve realizar. Esses requisitos variam conforme o tipo de *software* em desenvolvimento, os usuários-alvo e a abordagem adotada pela organização ao redigi-los (SOMMERVILLE, 2011). Os requisitos funcionais esperados para este sistema são:

RF001 - **Manter paciente:** O sistema deve permitir ao usuário paciente, se cadastrar, editar suas informações e visualizar suas consultas.
RF002 - **Manter usuário:** O sistema deve permitir a inclusão, edição e visualização de um usuário administrador do sistema.
RF003 - **Manter Agendamento:** O sistema deve permitir a criação, edição e cancelamento de um agendamento de uma consulta pelo usuário paciente e usuário administrador.
RF004 - **Visualizar Agendamentos:** O sistema deve permitir ao usuário administrador visualizar todos os agendamentos.
RF005 - **Manter profissional:** O sistema deve permitir ao usuário administrador, a criação de um novo profissional podendo ser um médico, dentista ou psicólogo.
RF006 - **Gerar lembrete de agendamento:** O sistema deve gerar lembrete de agendamento de um agendamento contendo o nome do paciente, horário e local da consulta e deve ser emitido por meio de notificações para os pacientes.

3.2.2 Requisitos Não Funcionais

RNF001 - Segurança: O sistema deve ser protegido contra acesso não autorizado. Todas as transmissões de dados entre o *front-end* e o *back-end* devem ser seguras, utilizando HTTPS.
RNF002 - Compatibilidade: A aplicação deve ser compatível com uma variedade de navegadores, como o Google Chrome, Microsoft Edge e Mozilla Firefox, garantindo uma boa experiência do usuário.
RNF003 - Usabilidade: A interface do usuário deve ser intuitiva e fácil de usar, promovendo uma boa experiência do usuário.

3.3 Modelagem do Sistema

A fase de modelagem de um sistema desempenha um papel essencial no processo de desenvolvimento, pois identifica as classes e entidades que comporão o sistema e fornece uma representação clara das funcionalidades a serem implementadas. A modelagem de domínio é um processo crucial nesse contexto, pois permite representar as entidades principais e suas interações dentro de um determinado domínio de negócio. De acordo com Evans (2004), a modelagem de

domínio visa identificar e descrever os objetos do mundo real e suas relações, facilitando a compreensão do problema a ser resolvido. No contexto deste trabalho, a modelagem de domínio foi fundamental para capturar os principais elementos relacionados ao agendamento de consultas e aos usuários do sistema. A partir dessa modelagem, foi possível criar uma visão clara de como os dados e os processos se inter-relacionam.

A UML (*Unified Modeling Language*), por sua vez, é uma linguagem visual amplamente adotada para modelar sistemas orientados a objetos, sendo uma linguagem de propósito geral que pode ser aplicada em diversos domínios de aplicação. A modelagem do sistema tem a responsabilidade de criar modelos que capturam uma visão do sistema físico, abstraindo-o com um propósito específico, como a descrição de aspectos estruturais e comportamentais do software (GUEDES, 2011).

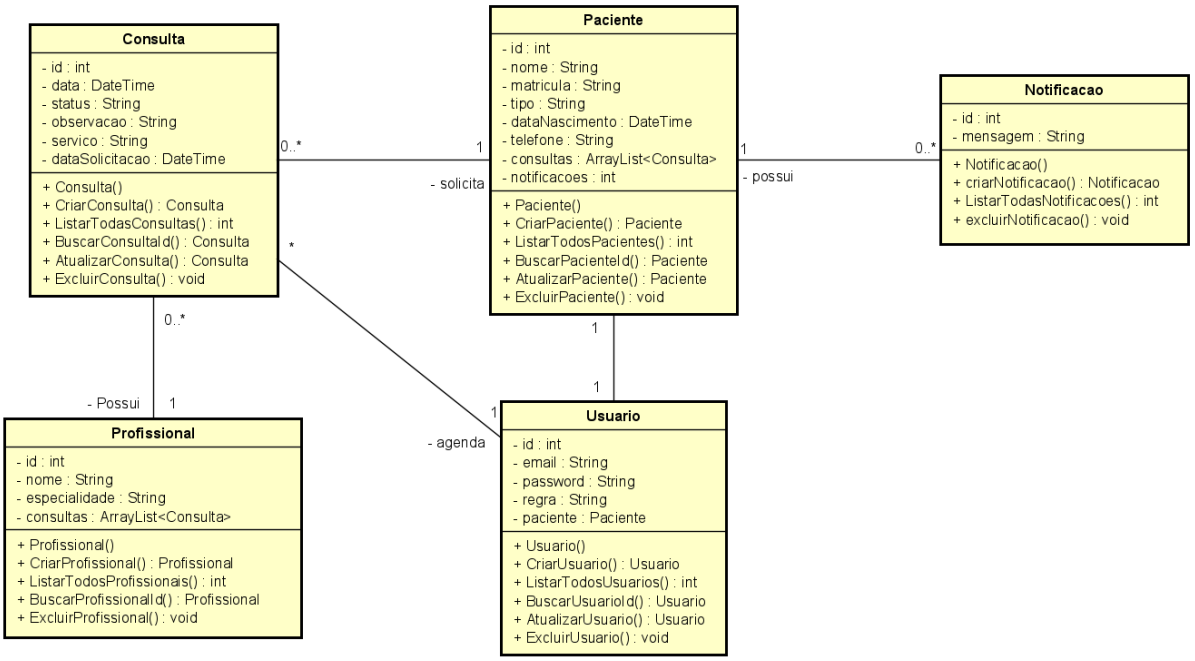
Neste trabalho, serão apresentados os diagramas de classes e os diagramas de caso de uso que fazem parte da modelagem do sistema.

3.3.1 Diagrama de classes

O diagrama de classes é amplamente utilizado na UML e é fundamental para o desenvolvimento de sistemas. Ele serve como base para outros diagramas, definindo a estrutura das classes no sistema. Este diagrama especifica os atributos e métodos de cada classe, além de estabelecer as relações e trocas de informações entre elas (GUEDES, 2018).

Na Figura 5, apresenta-se o diagrama de classes do sistema desenvolvido, as classes de domínio são representadas por retângulos, cada um contendo seus atributos e métodos principais. As linhas conectando as classes representam os diferentes tipos de associações entre elas, delineando como as classes estão interligadas no contexto da aplicação.

Figura 5 - Diagrama de classes



Fonte: Do autor.

3.3.2 Casos de uso

Os atores envolvidos no sistema desempenham papéis fundamentais na definição de suas funcionalidades, como evidenciado no diagrama de casos de uso, que delineia a funcionalidade proposta durante a fase de *design*. Esta ferramenta é crucial para a identificação dos requisitos funcionais, sendo utilizada tanto pela equipe interna de desenvolvimento quanto na apresentação das funcionalidades planejadas ao cliente (MORAIS, 2020).

No contexto do sistema feito neste presente trabalho, os principais atores são:

- Administrador: Responsável por cadastrar novos profissionais no sistema, como psicólogos ou médicos. Além disso, o administrador tem acesso privilegiado para cancelar agendamentos, excluir profissionais e visualizar todos os profissionais e consultas associadas a cada um deles.
- Paciente: Este ator pode se cadastrar no sistema para realizar agendamentos conforme sua necessidade. Ele também pode visualizar seus

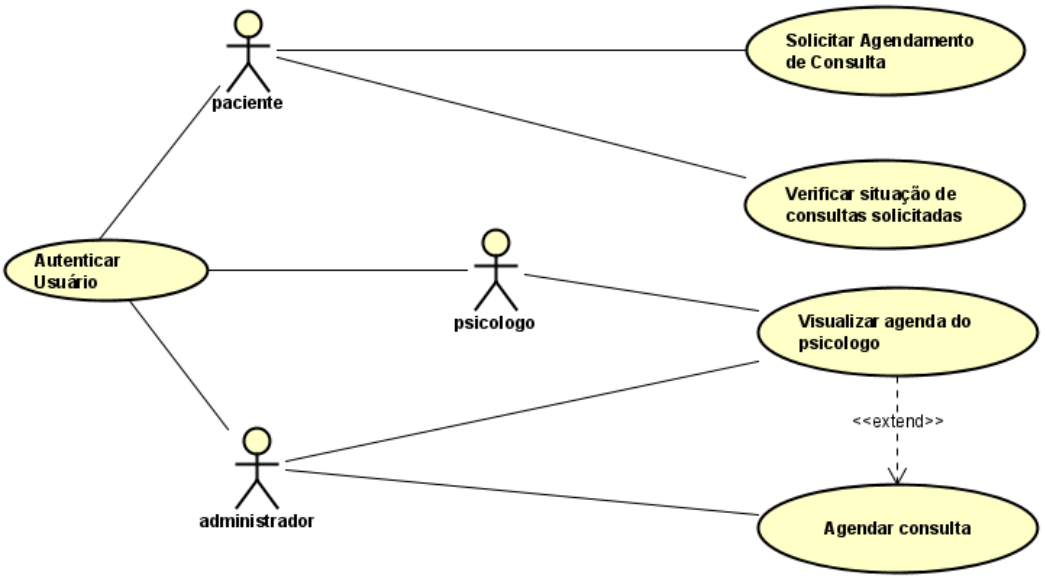
próprios agendamentos por meio de uma interface online. Os pacientes podem ser docentes, discentes ou servidores.

- Psicólogo: O psicólogo é responsável por acessar respostas confidenciais dos pacientes e criar planilhas com os dados dessas respostas para análise.

Cada um desses atores interage de maneira distinta com o sistema, refletindo diferentes necessidades e permissões, conforme estabelecido nos casos de uso.

A Figura 6 apresenta o diagrama de caso de uso para esse sistema. Os elementos da imagem são chamados de: atores, e os casos de uso, onde é apresentado o que cada ator do sistema opera por meio das funcionalidades ligadas a eles.

Figura 6 - Diagrama de caso de uso



Fonte: Do autor.

3.3.3 Narrativas de caso de uso

Uma abordagem alternativa para lidar com a especificação de casos de uso é realizar uma análise minuciosa de cada um deles. Essa análise deve ser conduzida de maneira descritiva, utilizando texto para documentar as nuances de cada caso de

uso (MORAIS, 2020). A seguir, apresenta-se a narrativa de caso de uso Autenticar usuário.

3.3.3.1 UC 1 - Autenticar Usuário

Resumo: Este caso de uso descreve o processo pelo qual um usuário autentica sua identidade no sistema.

Ator: Administrador, Paciente, Psicólogo

Pré-condições:

- O usuário deve estar registrado no sistema.

Fluxo Principal:

1. [RS] O sistema exibe os campos de entrada para login e senha.
2. [EV] O usuário insere seu email e senha.
3. [EV] O usuário clica no botão "Entrar".
4. [RS] O sistema valida as credenciais fornecidas.
5. [RS] Se as credenciais forem válidas, o sistema direciona o usuário para a interface correspondente (administrador, paciente ou psicólogo).

Fluxos Alternativos:

[FA1] Se o *email* ou a senha estiverem incorretos, o sistema exibe uma mensagem de erro **MSG001.**

Pós-condições:

O usuário está autenticado e tem acesso às funcionalidades apropriadas do sistema.

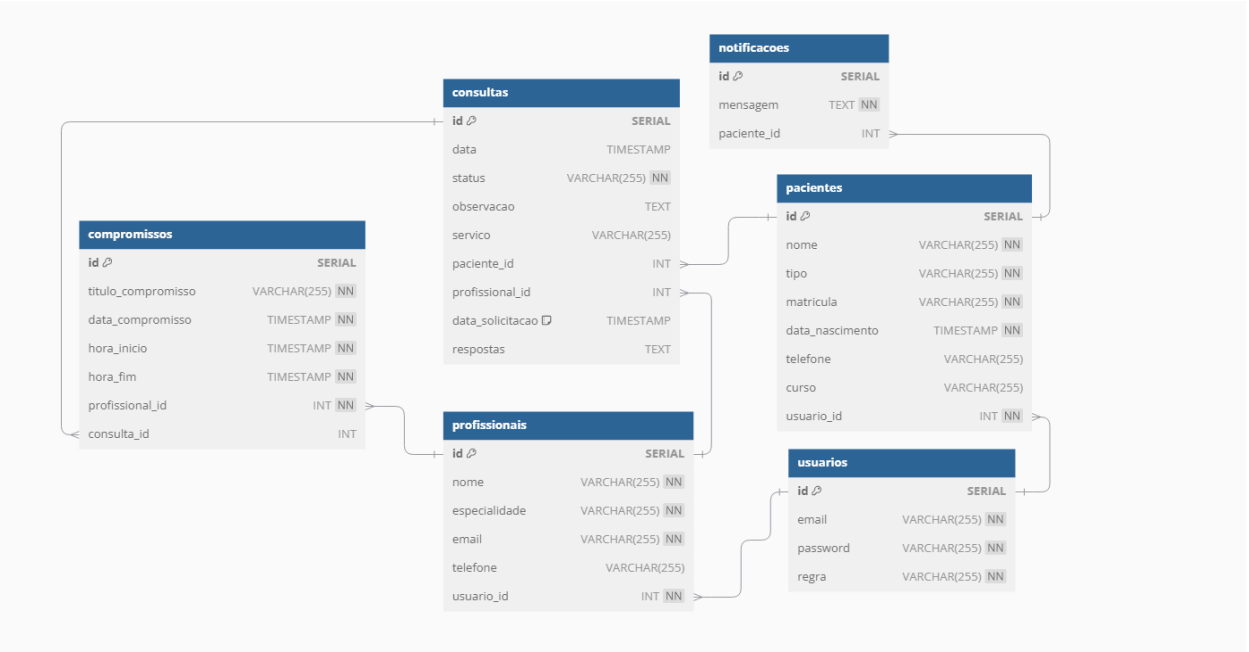
As demais narrativas de caso de uso se encontram no Apêndice A.

3.4 Modelagem do banco de dados

Um modelo de dados é um conjunto de conceitos que permite descrever a estrutura de um banco de dados, fornecendo o significado essencial para a abstração dos dados (HEUSER, 2004). A modelagem, portanto, é fundamental para organizar de maneira lógica as ideias e os conjuntos de dados que serão armazenados e manipulados. Conforme destacado por Alves (2014), o modelo físico

representa a realidade de forma detalhada e específica, refletindo a estrutura exata do banco de dados, incluindo tabelas, colunas, tipos de dados e relacionamentos. Seu objetivo principal é definir como os dados serão fisicamente armazenados no banco, incluindo detalhes de implementação, como chaves primárias, chaves estrangeiras e índices. A Figura 7 mostra um modelo Lógico do banco de dados, onde cada retângulo representa uma tabela, e as linhas que conectam os retângulos indicam os relacionamentos entre as tabelas, definidos pelas chaves estrangeiras.

Figura 7 - Modelo Lógico do banco de dados



Fonte: Do autor.

3.5 Descrição das tabelas do banco de dados

Através do modelo lógico apresentado anteriormente, onde se tem apenas uma representação visual, não é demonstrado de fato a descrição de cada coluna das tabelas. Desta forma, se faz necessário o dicionário de dados para descrever cada entidade, detalhando os atributos por meio de tabelas. A Tabela 1 apresenta a descrição da entidade usuário do banco de dados.

Tabela 1 - Tabela usuario

Coluna	Tipo de Dados	Descrição
id	Int	Chave primária, identificador único do usuário
email	Varchar (255)	Descreve o endereço de e-mail único do usuário
senha	Varchar (255)	Descreve a senha do usuário
regra	Varchar (255)	Descreve o papel ou regra do usuário no sistema
paciente	Relacionamento	Descreve o relacionamento opcional com a tabela "paciente"

Fonte: Do autor.

São apresentadas no Apêndice B, as descrições das demais tabelas do banco de dados.

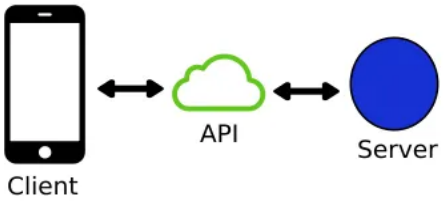
4 DESENVOLVIMENTO

Com a realização da fase de levantamento de requisitos e o estudo sobre as tecnologias escolhidas para a implementação do sistema, tendo como principais o *framework* Vue Js, Node Js usando o *framework Express*, e o Prisma ORM para a configuração e manipulação do banco de dados *Postgress*, então foi realizada a etapa de desenvolvimento do sistema.

4.1 Arquitetura

A estrutura cliente-servidor é um modelo de arquitetura de rede amplamente utilizado em ambientes de TI, onde os computadores são divididos em dois grupos: servidores, que fornecem serviços ou recursos, e clientes, que solicitam esses serviços e recursos, recebendo-os como respostas. A seguir, a Figura 8 demonstra a arquitetura de rede Cliente-Servidor, onde o front-end e o *back-end* possuem códigos distintos e se comunicam por meio de uma API. Essa API fornece os dados que são buscados no banco de dados por meio de requisições feitas no lado do front-end, utilizando o *Axios*, que acessa os endpoints.

Figura 8 - Arquitetura Cliente/Servidor



Fonte: MEDIUM (2021)

No que tange à arquitetura de *software*, é importante destacar que o desenvolvimento de *software* moderno envolve o uso de padrões arquiteturais que organizam a estrutura do sistema e guiam sua implementação. Segundo Bass, Clements e Kazman (2012), uma arquitetura de *software* "é a estrutura ou estruturas de um sistema, que compreendem elementos de *software*, suas propriedades

externamente visíveis e os relacionamentos entre eles". Esses padrões fornecem uma base sólida para a manutenção e evolução do sistema.

Neste projeto, utilizou-se uma variação do padrão *Model-View-Controller* (MVC). No lugar das tradicionais *views* (responsáveis pela interface com o usuário), o *back-end*, desenvolvido em Node.js com *Express*, utiliza rotas (*routes*) para se comunicar com o *front-end*, que foi desenvolvido em *Vue*.js. Essas rotas atuam como endpoints da API, processando as requisições e enviando as respostas para o front-end. A lógica e o gerenciamento de dados residem no *back-end*, enquanto a interface do usuário e a interação ficam no front-end, criando uma separação clara entre a lógica de apresentação e de negócios. A comunicação entre as duas partes é feita por meio de chamadas HTTP assíncronas, o que promove uma arquitetura modular e flexível, facilitando a manutenção e escalabilidade do sistema.

4.2 Back-End

Em aplicações *web*, o *back-end* é responsável por aquilo que vem por trás de uma aplicação, é onde são tratados erros, feitas as requisições para o banco de dados e chamadas para a API.

Para o *back-end* da aplicação, foi desenvolvido uma API utilizando o Node.js para a criação de um servidor local, juntamente com o *framework Express* para a criação e configuração da API. Também foi utilizado o Prisma ORM para a criação de um banco de dados *Postgress*. A Figura 9 apresenta o arquivo schema.prisma, que define toda estrutura do banco de dados. Este arquivo contém a declaração da estrutura de cada tabela, e a definição de qual banco de dados será, que neste caso é o postgress.

Figura 9 - Criação de tabelas com o Prisma

```
prisma > schema.prisma > Paciente
1   generator client {
2     provider = "prisma-client-js"
3   }
4
5   datasource db {
6     provider = "postgresql"
7     url      = env("DATABASE_URL")
8   }
9
10  model Usuario {
11    id       Int       @id @default(autoincrement())
12    email    String    @unique
13    password String
14    regra    String
15    paciente Paciente?
16
17    @@map("usuarios")
18  }
19
20  model Paciente {
21    id              Int           @id @default(autoincrement())
22    nome            String
23    tipo            String
24    matricula       String
25    dataNascimento  DateTime
26    telefone        String?
27    usuarioId       Int           @unique
28    Usuario         Usuario       @relation(fields: [usuarioId], references: [id])
29    consultas       Consulta[]
30    notificacoes    Notificacao[]
31
32    @@map("pacientes")
```

Fonte: Do autor.

4.2.1 Estrutura do Projeto

Na Figura 10, é apresentado a estrutura de pastas do *back-end*, e logo abaixo, apresentado suas definições.

Figura 10 - Estrutura de pastas do back-end

```
BACKEND
  .vercel
  node_modules
  prisma
    migrations
    schema.prisma
  src
    controllers
    models
      compromisso.model.js
      consulta.model.js
      notificacao.model.js
      paciente.model.js
      profissional.model.js
      usuario.model.js
    routes
      compromisso.routes.js
      consulta.routes.js
      index.js
      notificacao.routes.js
      paciente.routes.js
      profissional.routes.js
      usuarios.routes.js
    services
      auth
      Date
      prisma.js
    validations
      consulta.validation.js
      paciente.validation.js
      profissionalSaude.validation.js
    server.js
  .env
  .gitignore
  nodemon.json
  package.json
  vercel.json
  yarn-error.log
  yarn.lock
```

Fonte: Do autor.

Principais pastas:

- *migrations*: A pasta migrations contém arquivos que registram as alterações no esquema do banco de dados.
- *controller*: A pasta controller contém arquivos responsáveis por receber e processar as requisições HTTP.
- *models*: A pasta models contém os arquivos que definem a estrutura dos dados da aplicação, incluindo as entidades e suas relações.
- *routes*: A pasta routes contém os arquivos que definem as rotas da aplicação, mapeando as URLs às suas respectivas funções nos *controllers*.
- *services*: Arquivos necessários para aplicar as regras de negócio.
- validations: Contém arquivos para aplicar as validações.

4.2.2 Conexão com banco de dados

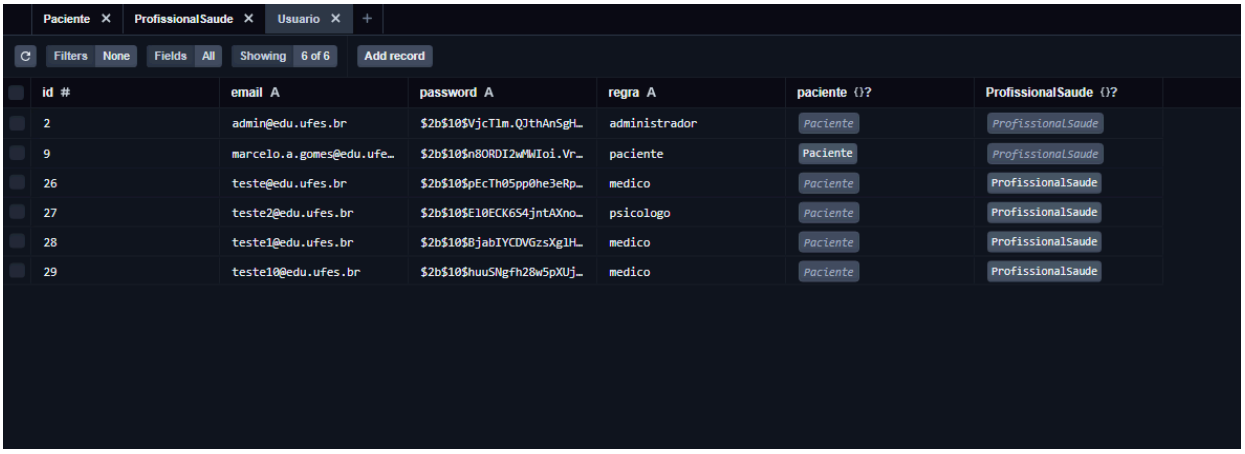
Para a conexão com o banco de dados, foi utilizado o ORM Prisma. Segundo o site oficial do Prisma, o Prisma é um ORM Node.js que consiste em três ferramentas principais: Prisma *Client*, Prisma *Migrate* e Prisma *Studio*, facilitando o gerenciamento de banco de dados e a integração com aplicações.

- Prisma *Client*: Um construtor de consultas gerado automaticamente e fortemente tipado para o *Node* e o *TypeScript*.
- Prisma Migrate: É uma ferramenta declarativa de modelagem e migração de dados que facilita o gerenciamento e versionamento de alterações no esquema do banco de dados de maneira simples e segura. Sempre que o modelo de dados é alterado, como na criação ou modificação de tabelas e colunas, o Prisma *Migrate* gera automaticamente arquivos de migração. Esses arquivos contêm as instruções SQL necessárias para aplicar as mudanças ao banco de dados. O processo de migração segue três etapas principais:
    1. **Gerar a migração** (prisma migrate dev), que cria o arquivo de migração correspondente às alterações no esquema;
    2. **Aplicar as migrações** ao banco de dados, garantindo que as mudanças sejam refletidas no sistema;

3. **Sincronizar o esquema com o código**, assegurando que o modelo de dados e o banco estejam em conformidade. Isso facilita a gestão contínua do banco de dados, mantendo seu esquema atualizado e alinhado com o desenvolvimento.

- Prisma *Studio*: uma GUI para visualizar e editar dados em seu banco de dados. A Figura 11 a seguir demonstra como é o prisma studio.

Figura 11 - Prisma studio



Fonte: Do autor.

4.2.3 Endpoints

Para que se consiga fazer requisições na API, é necessário o acesso aos *endpoints* que ela oferece.

O ponto de extremidade (*endpoint*) de uma API é o local onde as requisições, também referidas como chamadas de API, são processadas. Os localizadores uniformes de recursos (URLs), comumente utilizados na *web* para diversas finalidades, como a localização de páginas *web*, são também empregados para identificar os *endpoints* de uma API (CLOUDFLARE, 2022). A seguir, na Figura 12, é apresentado o arquivo de consultas.routes.js, mostrando alguns endpoints de consultas presentes no projeto.

Figura 12 - Endpoints de consulta

```
consulta.routes.js M ×
src > routes > consulta.routes.js > ...
1   import {
2     create,
3     get,
4     getId,
5     update,
6     excluir,
7     excluirTodasAsConsultas,
8     consultasPaciente,
9     excluirTodasAsConsultasProfissional,
10    consultasPorProfissional,
11    agendarConsulta
12  } from "../controllers/consulta.controller.js";
13  import { checkPac, checkAdm } from "../services/auth/auth.js";
14
15  const consultaRoutes = (app) => {
16    // Criar uma nova consulta
17    app.post("/consulta", create);
18    // Agendar consulta
19    app.post("/consulta/agendar", agendarConsulta);
20    // Listar todas as consultas
21    app.get("/consultas", checkAdm, get);
22    // Listar consultas de um paciente específico (com autenticação)
23    app.get("/paciente/:id/consultas", checkPac, consultasPaciente);
24    // Obter detalhes de uma consulta específica
25    app.get("/consulta/:id", checkAdm, getId);
```

Fonte: Do autor.

4.2.4 Models

A camada de *models* desempenha o papel crucial de administrar e controlar o comportamento dos dados, é a camada que define as suas entidades, cada uma com suas propriedades e métodos (operações). Nos arquivos de modelo do projeto, é onde ocorre a definição das entidades, seus atributos e as validações de *back-end*, juntamente com as especificações de associação entre os modelos(CARVALHO, 2020). A Figura 13 demonstra um exemplo de *model* Consulta do sistema.

Figura 13 - Arquivo Consulta.model.js

```
src > models > consulta.model.js > Consulta > CriarConsulta > consulta > select
1   import { prisma } from "../services/prisma.js";
2
3   class Consulta {
4     async CriarConsulta(data) {
5       const consulta = await prisma.consulta.create({
6         data,
7         select: {
8           id: true,
9           data: true,
10          status: true,
11          observacao: true,
12          servico: true,
13          pacienteId: true,
14          profissionalId: true,
15          data_solicitacao: true,
16          respostas: true
17        },
18      });
19      return consulta;
20    };
```

Fonte: Do autor.

4.2.5 Controllers

A camada de *controllers* e a camada responsável por ligar o model e a *view*, fazendo com que os models possam ser repassados para as *views* e vice-versa (GUEDES, 2020).

Os arquivos de *controllers* no projeto seguem uma estrutura que envolve a criação de uma classe controladora. Essa classe possui métodos assíncronos que são executados quando chamados pelo *endpoint* correspondente configurado no arquivo de rota. A Figura 14, por exemplo, ilustra o método de *create* que se encontra no arquivo consulta.controller.js na classe Consulta.

Figura 14 - Arquivo consulta.controller.js

```
src > controllers > consulta.controller.js > create
1   import Consulta from '../models/consulta.model'
2   import { formatDate } from '../services/Date/Date';
3
4   export const create = async (req, res) => {
5     try {
6       const dataConsulta = req.body
7       const consulta = await Consulta.CriarConsulta(dataConsulta);
8       const dataFormatada = formatDate(consulta.data_solicitacao)
9       consulta.data_solicitacao = formatDate(dataFormatada)
10      res.status(200).json(consulta);
11    } catch (e) {
12      console.log(`${e}`);
13      res.status(400).json({ message: "Erro ao criar uma consulta", error: e });
14    }
15
16  };
```

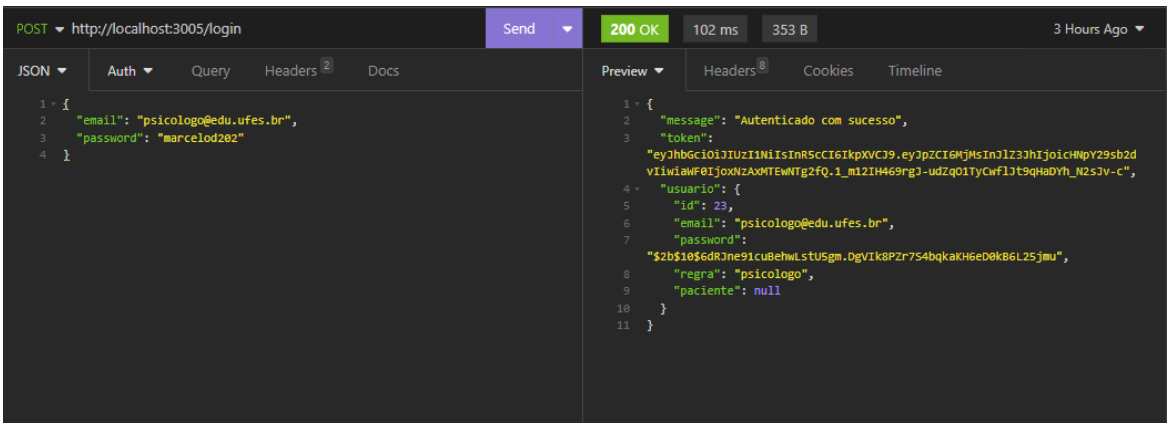
Fonte: Do autor.

4.2.6 Autenticação

Como medida para fortalecer a segurança e restringir o acesso às rotas da API no servidor a usuários não autorizados, foi implementado um processo de autenticação. Isso garante que apenas os usuários cadastrados e autenticados no sistema possam acessar suas funcionalidades correspondentes, levando em consideração seu nível de acesso.

O processo de autenticação ocorre durante o login, onde o usuário, ao fornecer seu email e senha, recebe, na resposta da requisição de login, um *token* de acesso. Este *token* concede aos usuários permissão para realizar outras requisições no sistema, devendo ser incluído no cabeçalho (*header*) de autorização como um *Bearer token*. Caso o *token* não seja informado, a resposta para as rotas protegidas indica a inexistência do *token* ou a invalidação do mesmo, caso seja fornecido um *token* não validado. A Figura 15 ilustra um exemplo de requisição de *login* de um usuário previamente cadastrado no banco de dados.

Figura 15 - Demonstração da funcionalidade Autenticar usuário



Fonte: Do autor.

Para a implementação da autenticação, foi adotado o *JSON Web Token*, conhecido como JWT, citados nas tecnologias utilizadas neste sistema. Este método utiliza um objeto JSON, cuja segurança e confiabilidade são garantidas por assinaturas digitais realizadas por meio de uma chave secreta com o algoritmo HMAC ou um par de chaves pública/privada usando RSA ou ECDSA (JWT, 2022).

A validação do *token* foi incorporada por meio da criação de alguns middlewares de autenticação, que checam se o usuário é um paciente, psicólogo ou um administrador.

Foram implementados 3 *middlewares* para realizar as devidas autenticações do usuário. Os *middlewares* são: checkAdm, checkPac e checkPsi, que respectivamente autenticam o administrador, paciente e psicólogo.

A Figura 16 demonstra um exemplo de middleware checando um administrador. A forma de autorização de um usuário é feita através do campo “regra” (administrador, paciente, psicólogo), contido na tabela usuário no banco de dados. Dessa maneira, toda requisição que for realizada passará pelo *middleware*, permitindo a validação do status do *token*, determinando se este é válido ou inválido. No caso de um *token* válido, a requisição prossegue normalmente em seu fluxo. No entanto, se o *token* for invalidado, uma resposta é imediatamente retornada, comunicando a ocorrência de um erro e interrompendo o fluxo da requisição.

Figura 16 - Arquivo Auth.js, checkAdm

```
export function checkAdm(req, res, next) {
    const authHeader = req.headers['authorization']

    const token = authHeader && authHeader.split(" ")[1]
    if (!token)
        return res.status(401).json({
            message: "Acesso negado"
        })
    try {
        const secret = process.env.SECRET
        const decoded = jwt.verify(token, secret)
        console.log(decoded)

        const regraUsuario = decoded.regra

        if (regraUsuario === 'administrador') {
            next()
        } else {
            return res.status(403).json({ message: "acesso não autorizado!" })
        }

    } catch (error) {
        console.log(error)
        res.status(400).json({ message: "Token inválido" })

    }
}
```

Fonte: Do autor.

4.3 Front-end

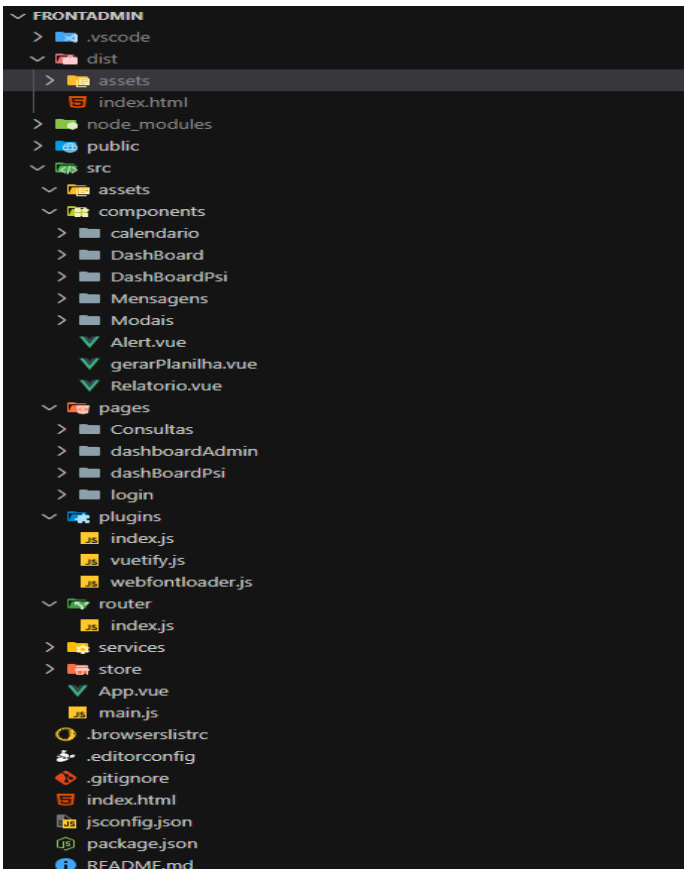
O *front-end* de uma aplicação é a parte visual que tem como objetivo fazer a interação e visualização do usuário por meio de uma interface gráfica.

Pode-se definir como a componente visual de um site tudo aquilo com o qual possa interagir, dentro da interface de um site ou sistema(ALURA, 2021).

4.3.1 Estrutura do projeto

Na Figura 17, é apresentado a estrutura de pastas do *front-end* admin e, logo abaixo, suas definições.

Figura 17 - Estrutura de pastas front-end admin



Fonte: Do autor.

Principais pastas:

- *components*: A pasta components armazena os componentes reutilizáveis da interface do usuário.
- *pages*: A pasta *pages* contém os arquivos que representam as diferentes páginas da aplicação.
- router: A pasta router contém a configuração das rotas da aplicação, permitindo definir como as diferentes páginas e componentes são acessados.
- *plugins*: A pasta plugins é onde contém a configuração e registros de plugins adicionais que estendem as funcionalidades do Vuetify.
- *store*: A pasta store contém a configuração do estado centralizado da aplicação, onde você define o estado, mutações, ações e getters, funcionalidades do Vuex.

4.3.2 Comunicação com o servidor

Na comunicação entre o front-end e o servidor para obtenção de dados da API, foi utilizado o protocolo HTTP. Foi usada também a biblioteca *Axios,* empregada para realizar requisições HTTP no *front-end*. Essa ferramenta simplifica a interação com o servidor, permitindo a busca e o envio de dados de maneira eficiente através das *Promises*. A Figura 18 apresenta o arquivo de configuração axios.*config*.js. Neste arquivo, é definido a baseURL, que é url base para realizar as requisições através dos *endpoints*. Se o token existir, ele é injetado no cabeçalho das requisições, caso contrário, é retornado uma Promisse de rejeição, indicando um erro.

Figura 18 - Arquivo axiosConfig.js

```
axiosConfig.js M ×
src > services > axiosConfig.js > http.interceptors.request.use() callback
1   import axios from "axios";
2
3
4   export const http = axios.create({
5     // baseURL: "http:// 192.168.8.176:3005/",
6     // baseURL: "https://apitcc.vercel.app/",
7     baseURL: "http://10.3.152.107:3005/",
8     // baseURL: "http://localhost:3005/"
9   });
10
11  // Adicione um interceptor para atualizar dinamicamente o cabeçalho de autorização
12  http.interceptors.request.use(
13    (config) => {
14      const token = localStorage.getItem("token");
15
16      // Se o token existir, adicione-o ao cabeçalho de autorização
17      if (token) {
18        config.headers["Authorization"] = `Bearer ${token}`;
19      }
20
21      return config;
22    },
23    (error) => {
24      return Promise.reject(error);
25    }
```

Fonte: Do autor.

4.3.3 Navegação

A implementação da navegação entre as diversas páginas da aplicação foi realizada por meio da integração com a biblioteca de roteamento oficial do Vue.js,

conhecida como Vue *Router*. O Vue *Router* é a biblioteca oficial de roteamento para aplicações Vue.js. A Figura 19 demonstra o arquivo *index*.js que possui todas as rotas nomeadas e direcionadas para os respectivos componentes. Nesse arquivo, é onde também o token é utilizado para verificar se o usuário é realmente autêntico.

Figura 19 - Arquivo index.js Routes

```
index.js M ×
src > router > index.js > routes > beforeEnter
1   import { createRouter, createWebHistory } from "vue-router";
2
3   const routes = [
4     {
5       path: "/",
6       redirect: "/login",
7     },
8     {
9       path: "/login",
10      name: "login",
11      component: () => import("@/pages/login/Login"),
12      meta: { requiresAuth: false },
13      beforeEnter: (next) => {
14        const isAuthenticated = localStorage.getItem("token");
15        if (isAuthenticated) {
16          next("/dashboard");
17        } else {
18          next();
19        }
20      },
21    },
22    {
23      path: "/criarConta",
24      name: "criarConta",
25      component: () => import("@/pages/criarConta.vue"),
26      meta: { requiresAuth: false },
27    },
28    {
29      path: "/logout",
30      name: "logout",
31      component: () => import("@/pages/login/Logout"),
32      meta: { requiresAuth: true },
```

Fonte: Do autor.

Essa biblioteca oferece recursos especializados, como o componente personalizado *router-link*, que facilita a criação de *links*. Ao utilizar o atributo *"to"*, é possível especificar o nome da rota, no caso de uma rota nomeada, ou o caminho correspondente.

4.4 Views

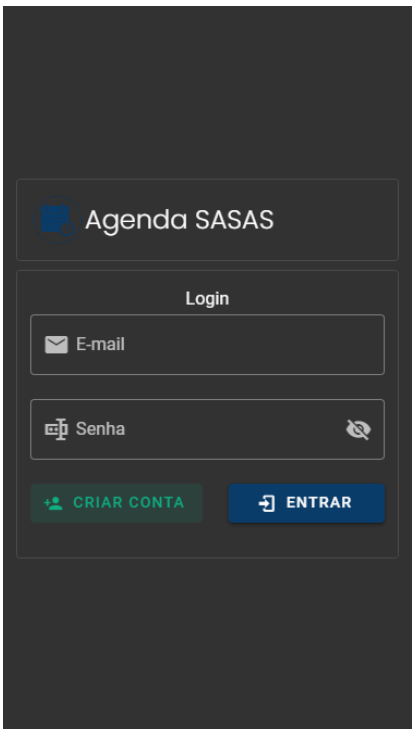
O desenvolvimento das interfaces do sistema seguiu a abordagem de implementar cada uma como um componente Vue único. Esses componentes proporcionam destaque de sintaxe, contendo um bloco dedicado à execução de scripts em *JavaScript* para manipulação dinâmica do componente, bem como outro bloco destinado à definição de estilos CSS específicos para cada componente(VUE, 2023).

Para uma organização estruturada, foram separados os componentes relacionados a cada conjunto de cadastro em pastas específicas dentro do diretório "*views*". Nas pastas, é possível encontrar os componentes encarregados da listagem do objeto associado, o formulário de cadastro em si, além dos componentes voltados para relatórios e visualização. Esse arranjo modular facilita a manutenção e compreensão do código, promovendo uma estrutura mais clara e eficiente para o desenvolvimento e aprimoramento contínuo do sistema.

4.4.1 Tela de login paciente

A tela de Login do Paciente proporciona a autenticação para os perfis de Paciente. Esta tela é apresentada na Figura 20. Esta tela é composta por dois campos e botões para ações específicas. Os campos são destinados à coleta das informações de e-mail e senha do usuário. Ao clicar no botão de Entrar, uma requisição *POST* é enviada, desencadeando a verificação do *token* do usuário durante o processo de autenticação. Uma vez validado o *token*, o sistema progride para a próxima tela.

Figura 20 - Tela de login paciente

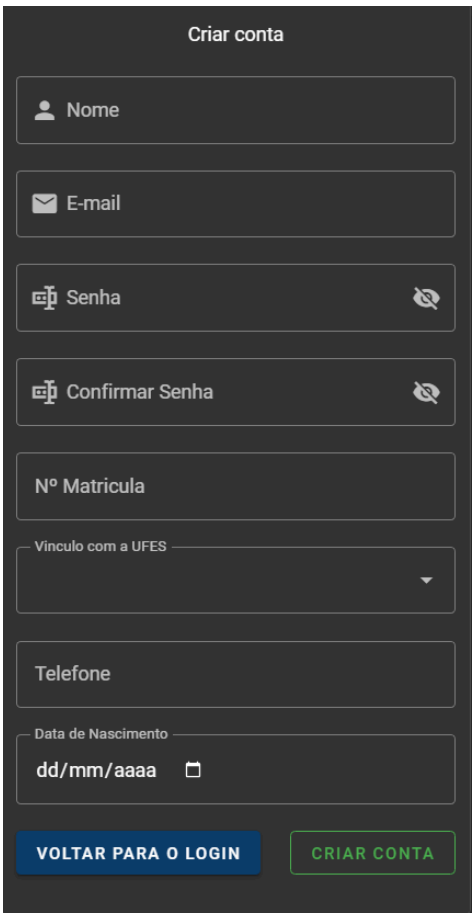


Fonte: Do autor.

4.4.2 Tela de criação de conta Paciente

A tela de Criação de conta do paciente é apresentada na Figura 21, ela proporciona ao usuário paciente preencher um formulário com suas informações, como por exemplo, nome, email, senha, matrícula, o vínculo com a UFES, data de nascimento, nível e curso. Após o preenchimento dessas informações, ao clicar em criar conta, é feita uma requisição *POST* de criação de conta para o *back-end*.

Figura 21 - Tela de criação de conta Paciente



Fonte: Do autor.

4.4.3 Tela de Solicitar Consulta

Após estar logado, o usuário tem acesso a opção de solicitar uma consulta. A Figura 22 apresenta esta tela. Nesta tela, o usuário pode escolher entre solicitar uma consulta com o médico ou o psicólogo. Caso escolha atendimento médico, é pedido que ele preencha o campo de observações, caso tenha, e após isso, poderá clicar em solicitar consulta. Uma mensagem de confirmação é exibida. Caso escolha atendimento psicológico, o usuário é convidado a responder um questionário contendo várias questões sobre o porquê de solicitar uma consulta com o psicólogo. Após preencher esse formulário, o usuário poderá clicar em solicitar consulta e uma mensagem de confirmação é exibida.

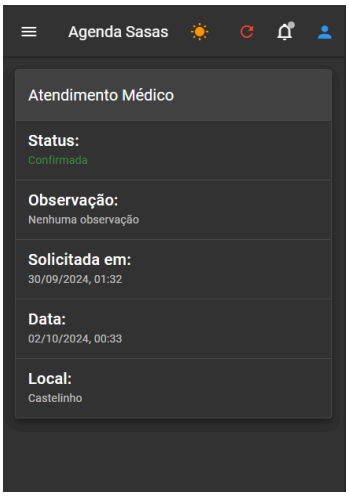
Figura 22 - Tela de solicitação de consulta

Fonte: Do autor.

4.4.4 Tela de visualização de consultas

Após estar logado, o usuário poderá ver suas consultas que foram solicitadas. Caso não tenha nenhuma consulta solicitada, é apresentada uma mensagem dizendo que não há consultas solicitadas, caso contrário, apresenta as consultas que foram solicitadas com suas informações mais relevantes. A Figura 23 apresenta a tela de visualização de consultas do paciente.

Figura 23 - Tela de visualização de consultas.

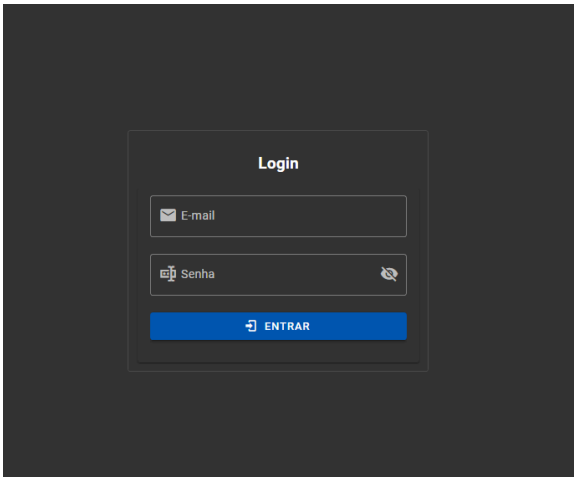


Fonte: Do autor.

4.4.5 Tela de login administrador

A tela de login do administrador proporciona a autenticação para os perfis de administrador e psicólogo. Esta tela é composta por dois campos e botões para ações específicas. Os campos são destinados à coleta das informações de e-mail e senha do usuário. Ao clicar no botão de Entrar, uma requisição *POST* é enviada, desencadeando a verificação do *token* do usuário durante o processo de autenticação. Uma vez validado o *token*, o front-end checa se o usuário que está tentando acessar o sistema é um administrador ou psicólogo de acordo com o atributo regra encontrado no usuário que é retornado no ato da autenticação, o sistema progride para a próxima tela. A Figura 24 demonstra a tela de login do front-end do administrador.

Figura 24 - Tela de login administrador

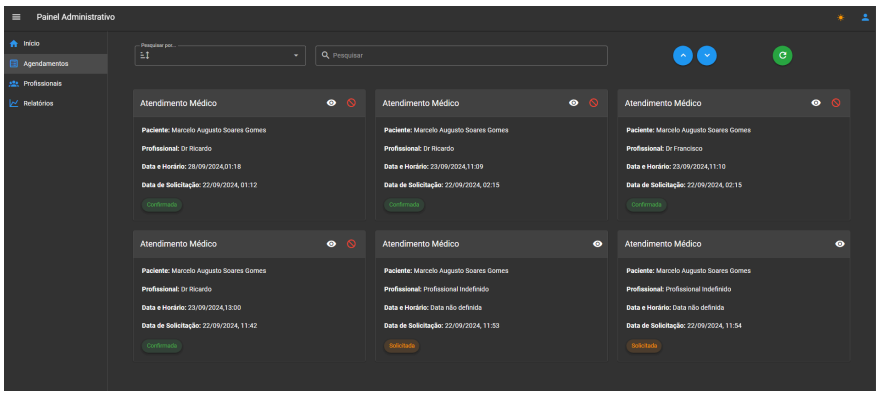


Fonte: Do autor.

4.4.6 Tela de listagem de consultas

Na tela de listagem de consultas dá acesso às informações das consultas com uma visão mais ampla, pode-se saber o status das consultas, realizar filtragem a partir das opções do elemento v-*select*. A Figura 25 demonstra a tela de listagem de consultas.

Figura 25 - Tela de listagem de consultas

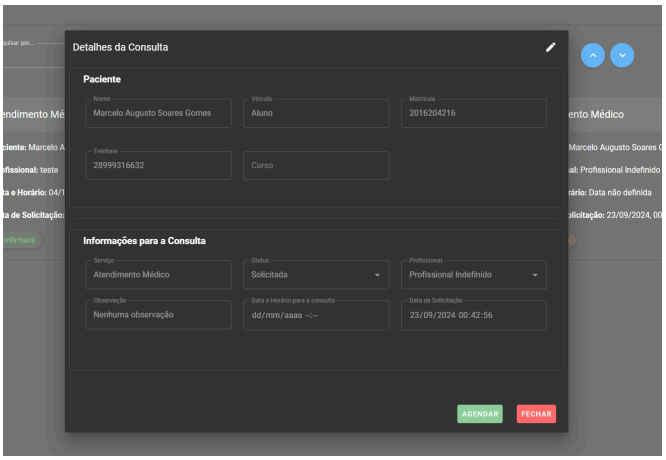


Fonte: Do autor.

4.4.7 Tela de editar consulta

A tela de editar a consulta permite ao usuário administrador mudar o status da consulta, definir a data e horário e escolher o profissional. Quando todos esses campos estão devidamente preenchidos, ao clicar em salvar, uma mensagem de confirmação aparece e uma notificação é enviada ao respectivo paciente que solicitou aquela consulta. Caso a consulta esteja com o status “Confirmada”, o administrador tem a opção de excluí-la. O administrador só pode ver e ter acesso a modificações de todas as consultas que tem o status “Solicitada”. A Figura 26 demonstra a tela de edição e visualização da consulta.

Figura 26 - Tela de detalhes/editar consulta



Fonte: Do autor.

5 TESTES

O objetivo dos testes é verificar se um programa cumpre sua função corretamente e identificar possíveis falhas antes de sua utilização real. Durante a fase de testes, o software é executado com dados simulados para avaliar seu desempenho. Os resultados são analisados com cuidado para identificar erros, anomalias e informações sobre atributos não funcionais do programa. Esses testes são cruciais para aprimorar a qualidade e a confiabilidade contínua do software (SOMMERVILLE, 2011).

5.1 Testes de sistema

A seguir serão apresentados os principais problemas identificados com a realização de testes de sistema e as soluções de correção dos defeitos que promoveram os erros detectados.

Tabela 2 - Tabela com os erros e soluções

Erros apresentados	Solução
Ao agendar uma consulta, a mesma era agendada mesmo sem escolher o profissional e preenchendo o campo de data da consulta.	Foi implementada uma função que checa se o campo de data da consulta e profissional estão nulos, caso estejam, é emitido um modal de aviso para preencher esses campos. Caso contrário, a consulta é agendada
No cadastro de paciente, foi observado que era possível se cadastrar mesmo sem preencher todos campos requeridos.	Foi implementada uma função de validação no front-end, onde os campos não preenchidos ficam com a borda vermelha, indicando que o campo não foi preenchido. Também foi realizada a validação dos dados no back-end.

O usuário administrador estava vendo as respostas dos pacientes de atendimento psicológico.	Foi implementada uma lógica que checa se o usuário logado é um administrador ou um psicólogo, pela coluna ‘regra’ da tabela de usuário do banco de dados.
As datas de solicitação e data da consulta apresentavam um formato não amigável.	Foram implementadas mudanças para impedir essa ação. As datas, por padrão são de um dateTime diferente do Brasil, e por isso, houve uma alteração para que elas seguissem o formato de dd/mm/aaaa, além da hora também ser ajustada para o fuso horário brasileiro.
Inicialmente, um dos status da consulta era “agendada” e não estava muito intuitivo esse status.	Modificou-se o nome do status para “Solicitada”.
Ao gerar o relatório de consultas, caso não tenha nenhuma consulta, o botão de “Gerar PDF” é ativo.	Foi implementada a função que verifica se possui consultas, se sim, o botão é ativado, se não, o botão é desativado, impossibilitando que ele seja clicado.

Fonte: Do autor.

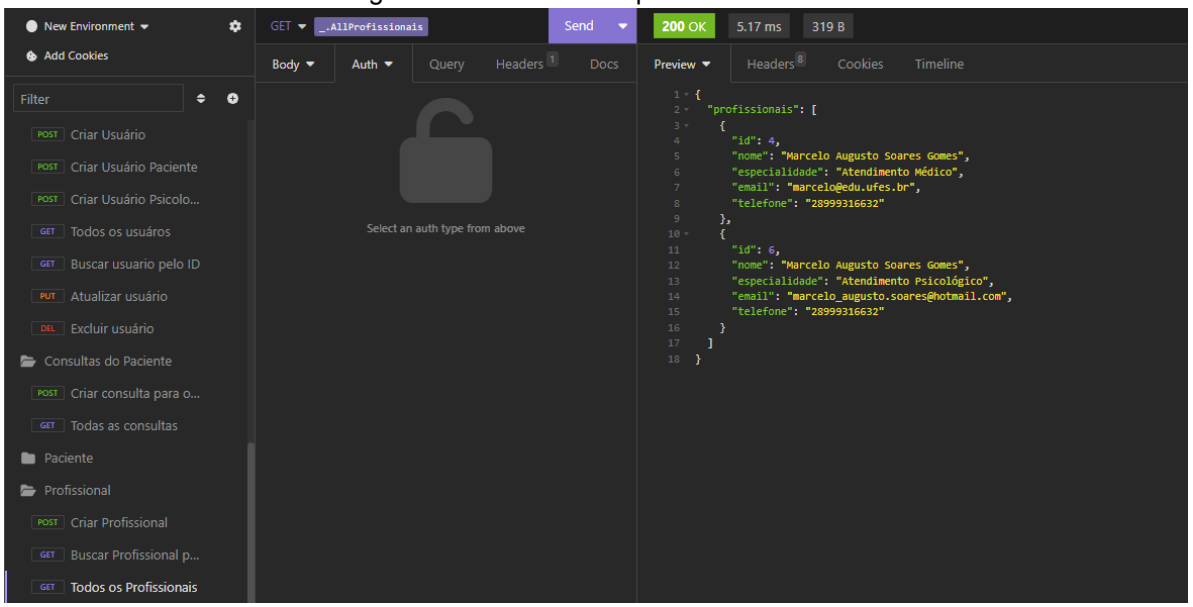
5.2 Teste de Desenvolvimento

Os testes realizados durante o desenvolvimento do sistema foram essenciais para garantir a funcionalidade e a qualidade da aplicação. Através de diferentes abordagens, buscou-se verificar o desempenho e usabilidade corrigindo possíveis erros e otimizando o código conforme necessário. Um dos principais testes que o autor realizou, foram os testes direcionados aos endpoints, utilizando a ferramenta Insomnia, que é uma ferramenta robusta e amplamente utilizada para testar APIs. Essa ferramenta permite realizar requisições HTTP de maneira simples e eficiente, simulando as interações que ocorreriam entre o *front-end* e o *back-end* do sistema. (INSOMNIA, 2021).

Foi utilizado o Insomnia para testar os *endpoints* do sistema, verificando se as requisições estavam sendo processadas corretamente e se as respostas retornadas atendiam aos requisitos esperados.

A seguir, a Figura 27 mostra como um teste é feito no insomnia, para ser verificado a resposta do *endpoint*, usando um parâmetro na url de uma requisição *GET*, no *endpoint* de /profissionais, que retorna todos os profissionais cadastrados no sistema.

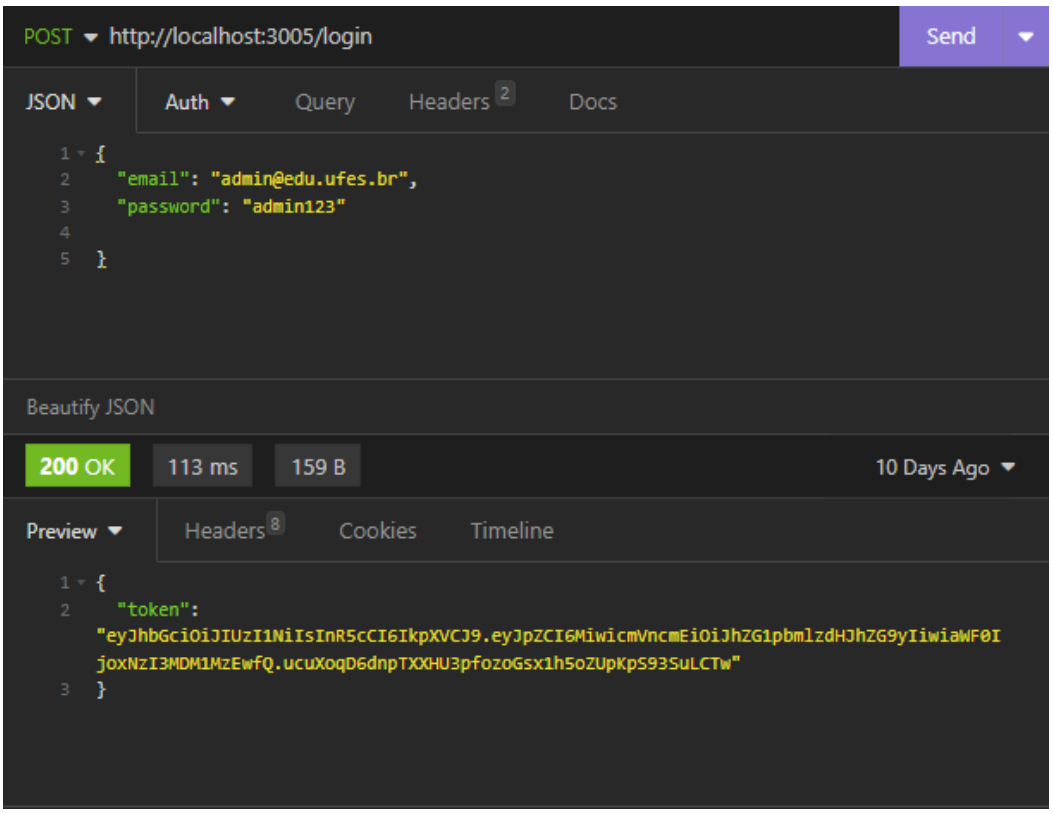
Figura 27 - Teste de endpoint no insomnia



Fonte: Do autor.

Também foram feitos testes no *endpoint* /login. Neste *endpoint*, é preenchido o campo *body* da requisição, requisitando um *POST* e é retornado o *token*, que posteriormente, no *front-end*, é armazenado no local storage para realizar as requisições, e autenticação. A Figura 28 mostra uma requisição *POST* sendo feita no *endpoint login*.Na parte de superior da imagem, é a entrada de dados que é um objeto *JSON* com email e senha, e na parte inferior, é a resposta da requisição, indicando o *statusCode* de 200, que representa sucesso, juntamente com a resposta que retorna o *token* que será usado posteriormente no cabeçalho das requisições que feitas pelo *front-end*.

Figura 28 - Teste POST no endpoint /login



Fonte: Do autor.

Foi desenvolvido um plano de testes, conforme descrito na Tabela 3, detalhando as funcionalidades avaliadas no sistema. O processo de teste abrangeu uma verificação integral do sistema, assegurando o funcionamento de todas as suas funcionalidades de forma completa.

Tabela 3 - Plano de testes.

Funcionalidades	**Comportamento Esperado**	**Verificações**	**Critérios de Aceite**
Criação de Paciente	O paciente pode criar uma conta inserindo nome, e-mail, senha e	- Verificar se todos os campos obrigatórios estão presentes e	- O paciente pode se registrar com sucesso

	dados pessoais.	corretamente preenchidos. - Validação de e-mail e senha.	- Dados são salvos corretamente no banco de dados e senha atende às políticas de segurança.
Login do Paciente	O paciente deve ser capaz de inserir e validar suas credenciais (e-mail/senha) e ter o *token* armazenado no *local storage*.	- Verificar se o paciente é redirecionado para o painel ao fornecer credenciais corretas. - Verificar se o token é salvo no local storage do navegador.	- O login é bem-sucedido com credenciais corretas. - O *token* é salvo no local storage com segurança. - Exibir mensagem de erro para login com falha.
Visualizar Consultas	O paciente pode visualizar suas consultas com informações como serviço, data e status.	- Verificar se todas as consultas do paciente são listadas corretamente com serviço, data e status. - Verificar se o *layout* exibe todas as informações corretamente.	- O paciente visualiza a lista completa das consultas agendadas e seus status (pendente, confirmada, cancelada).
Solicitar Consulta	O paciente pode	- Verificar se o	- O paciente pode

	solicitar uma consulta, escolhendo entre Atendimento Médico ou Atendimento Psicológico.	paciente consegue ver as opções de serviços. - Verificar se a solicitação de consulta é registrada no banco de dados.	selecionar o serviço e fazer a solicitação de consulta com sucesso. - Solicitações são registradas e exibem status "Solicitada" até a confirmação do administrador.
Editar Perfil do Paciente	O paciente pode editar informações pessoais como nome, e-mail, telefone.	- Verificar se as alterações são salvas no banco de dados. - Verificar a validação dos campos (e-mail, telefone e senha).	- O paciente pode editar o perfil e ver as mudanças aplicadas corretamente. - Dados válidos são salvos sem erros de validação.
Logout do Paciente	O paciente pode sair do sistema e o *token* deve ser removido do *local storage.*	- Verificar se o *token* é removido do *local storage*. - Verificar se o paciente é redirecionado para a página de *login* após o *logout*.	- O *logout* remove o *token* da sessão e redireciona o paciente para a página de *login*. - *Logout* é realizado de forma segura.
Login do Administrador	O administrador pode realizar *login*	- Verificar se o *login* é feito	- O administrador faz *login* com

	com e-mail e senha e acessar o painel administrativo.	corretamente. - Verificar se o *token* é salvo no *local storage* do navegador.	sucesso e é redirecionado ao painel administrativo. - O *token* é salvo no local storage com segurança.
Visualizar Consultas	O administrador pode ver todas as consultas, com serviço, data e status.	- Verificar se a lista de consultas é carregada corretamente. - Verificar se todos os dados (data, status, serviço) são exibidos corretamente.	- O administrador visualiza todas as consultas com as informações corretas.
Agendar Consulta	O administrador pode agendar uma consulta, escolhendo o profissional, a data e atualizando o *status* da consulta.	- Verificar se a interface permite selecionar o profissional, a data e o *status*. - Verificar se a consulta é registrada no banco de dados corretamente.	- A consulta é agendada com sucesso e o paciente é notificado. - O sistema atualiza o *status* e os detalhes da consulta no banco de dados.
Cancelar Consulta	O administrador pode cancelar uma consulta alterando	- Verificar se a alteração do status para "Cancelada"	- O *status* é alterado para "Cancelada" e o

	o *status* para "Cancelada".	é refletida no banco de dados. - Verificar se o paciente é notificado sobre o cancelamento.	paciente é notificado. - O cancelamento é registrado corretamente no sistema.
Gerar Relatório de Consultas	O administrador pode gerar relatórios filtrando por data, serviço e status.	- Verificar se os filtros (data, serviço e status) funcionam corretamente. - Verificar se os dados do relatório são precisos e completos.	- O administrador gera relatórios com sucesso, filtrando por data, serviço e status. - Os dados gerados correspondem aos critérios de filtragem aplicados.
Adicionar Novo Profissional	O administrador pode adicionar um profissional, preenchendo nome, e-mail e serviço oferecido, telefone e senha.	- Verificar se o formulário de adição de profissional funciona corretamente. - Verificar se os dados estão salvos no banco de dados.	- O administrador pode adicionar novos profissionais e vê-los listados no sistema. - Os dados são salvos no banco de dados e exibidos corretamente.
Editar Profissional	O administrador pode editar os	- Verificar se as alterações são	- O administrador pode editar as

	detalhes de um profissional (nome, e-mail, serviço, telefone).	salvas corretamente no banco de dados. - Verificar se o profissional atualizado é exibido corretamente na lista.	informações do profissional e as mudanças são refletidas no sistema e no banco de dados.
Deletar Profissional	O administrador pode excluir um profissional do sistema.	- Verificar se o profissional é removido do banco de dados e da lista de profissionais.	- O profissional é deletado com sucesso e não aparece mais na lista de profissionais.
Ver Agenda do Profissional	O administrador pode visualizar a agenda de consultas de cada profissional.	- Verificar se a agenda do profissional é carregada corretamente, mostrando as consultas agendadas. mostra em um componente de calendário	- O administrador pode ver a agenda completa do profissional com todas as consultas agendadas exibidas corretamente.
Criar compromisso	O profissional pode criar um compromisso	Verifica se os dados são salvos no banco de dados	O profissional recebe uma mensagem de

	preenchendo um formulário com os campos de título, hora de início, hora de fim e data	corretamente, verificando se há conflito de horários com outros compromissos e consultas que já foram agendadas para aquele profissional	sucesso que o compromisso foi criado.
Visualizar compromissos	O profissional pode visualizar seus compromisso	Verifica se os compromissos estão carregando corretamente, se as datas aparecem em padrão dd/mm/aaaa e se a hora aparece em hh/mm	O profissional visualiza seus compromissos.

Fonte: Do autor.

5.3 Teste de Usuário

O teste de usuário é uma etapa crucial no processo de desenvolvimento, onde os usuários interagem com um sistema ou produto de *software* para fornecer *feedback* sobre sua usabilidade, funcionalidade e adequação às suas necessidades. Esse retorno direto é essencial para o aprimoramento contínuo do *software*, ajudando a garantir que o produto final ofereça uma experiência satisfatória e atenda às expectativas dos usuários (SOMMERVILLE, 2011).

Existem três tipos principais de testes de usuário no desenvolvimento de *software*:

- Teste Alfa: Nesta fase, os usuários colaboram diretamente com a equipe de desenvolvimento nas instalações dos desenvolvedores para identificar e corrigir problemas antes do lançamento oficial do *software*.
- Teste Beta: Uma versão preliminar do *software* é disponibilizada para um grupo seleto de usuários externos, que fornecem *feedback* sobre a usabilidade e relatam problemas para que sejam ajustados antes do lançamento final.
- Teste de Aceitação: Nesta etapa, os clientes testam o sistema para verificar se ele atende aos requisitos especificados e está pronto para implantação em seu ambiente. O objetivo é assegurar que todas as funcionalidades essenciais estão presentes e operacionais.

Esses testes são de grande importância para garantir a qualidade de um software, visando assegurar que o produto final atenda às expectativas e necessidades dos usuários. Através de uma abordagem criteriosa de testes, é possível identificar e corrigir potenciais falhas, melhorar a experiência do usuário, e promover a confiança no sistema. Na aplicação desenvolvida, foram realizados diversos testes, que incluíram a verificação de funcionalidades essenciais e a compatibilidade com múltiplos dispositivos e navegadores. Estes testes garantem que o *software* seja confiável e preparado para operar em cenários reais. Os testes foram feitos pelo autor, juntamente com um colega, para ter uma visão diferente de quem desenvolveu o sistema, a fim de se ter *feedbacks* para a melhoria do sistema. Foram realizados testes alfa, e se teve os seguintes *feedbacks*:

- Ao acessar o painel de administrador, este não deve ter acesso às respostas confidenciais contidas nas consultas realizadas para os psicólogos.
- Foi identificada a necessidade de adicionar um campo de "curso" no formulário de criação de paciente, destinado a estudantes que também são atendidos.
- Ao solicitar uma consulta, o botão de agendamento permanecia ativo mesmo após a solicitação, permitindo que o usuário fizesse outra solicitação sem selecionar o serviço desejado.

- No painel do profissional, quando uma consulta é agendada para o psicólogo, ao clicar em "ver mais", opção que deveria exibir as respostas confidenciais, as informações não estavam sendo exibidas corretamente.

Durante o desenvolvimento do sistema, todos os testes mencionados foram realizados para garantir que cada funcionalidade fosse avaliada em diferentes cenários. Um dos principais desafios enfrentados foi a dificuldade de conduzir testes com usuários externos, essenciais para obter *feedback* real e validar a experiência do usuário.

A superação desses desafios foi fundamental para a obtenção de resultados confiáveis e para alcançar uma implementação bem-sucedida, alinhada aos requisitos inicialmente definidos.

5.4 Teste de Usabilidade

O teste de usabilidade é uma ferramenta fundamental para garantir a eficácia de um produto, permitindo a coleta de *feedback* diretamente dos usuários finais. Essa prática é essencial para a identificação de melhorias contínuas e direcionadas, contribuindo para o desenvolvimento de produtos mais eficientes. Para maximizar os resultados obtidos, é crucial saber como organizar esses testes (DELAMARO, 2023).

Neste contexto, o autor assumiu a responsabilidade pela validação do sistema, aplicando um modelo de teste focado na descoberta de problemas. O principal objetivo foi identificar falhas e inconsistências que poderiam impactar o desempenho da aplicação. Durante o processo, as funcionalidades do sistema foram analisadas, e todas as questões encontradas foram registradas para posterior correção. Contudo, um dos desafios enfrentados foi a dificuldade de realizar testes com usuários reais, o que limitaria a obtenção de *feedback* direto e a validação prática da experiência do usuário. Embora não tenha sido possível testar com usuários finais, foram conduzidos testes internos, que ajudaram a avaliar a funcionalidade e a usabilidade do sistema. Os resultados desses testes de usabilidade estão detalhados na Tabela 4.

Tabela 4 - Testes de usabilidade

Defeitos e Observações	**Solução**
Validação de campos obrigatórios no cadastro do paciente	Foram feitas as devidas validações tanto no *front-end*, para não submeter sem algum campo não preenchido, quanto no *back-end* para validar e retornar mensagem de erro, caso não seja preenchido ou seja preenchido de forma incorreta.
Após a criação do paciente, limpar os campos e redirecionar	Foi verificado, e implementado uma função que redireciona o usuário para o login após o sucesso da requisição de criação de paciente.
Cores do botões sem ligação com suas respectivas funções, como por exemplo, o botão de fechar, que geralmente é vermelho, o de agendar ou salvar, que geralmente é verde ou azul, botões de ações nas tabelas do sistema	Foi verificado e foram feitas as devidas modificações, principalmente no front-end do admin, onde o uso de botões e ações são mais utilizados
Cores de mensagens de aviso	Foi verificado que algumas mensagens de aviso não estavam com cores correspondentes as cores que geralmente são vistas em mensagens de aviso nos sistemas, como por exemplo o uso da cor laranja, que simboliza aviso(*warning*)
Datas não apresentavam formato amigável para o usuário	Foi verificado que algumas datas não se apresentavam de forma amigável

	para o usuário, ou até mesmo por vezes, não estavam no fuso horário do Brasil
Inicialmente, os status das consultas não tinham cores	Foi verificado e compreendido que seria melhor o uso de cores para sinalizar o status das consultas, a fim de melhorar como as informações são transmitidas para os usuários através de cores que geralmente sinalizam estados e ações. Como por exemplo, para indicar o status das consultas, quando é “Solicitada”, a cor referente é laranja, quando é “Cancelada”, a cor é vermelha, e quando é “Confirmada”, a cor é verde.

Fonte: Do autor.

6 CONSIDERAÇÕES FINAIS E CONCLUSÕES

O presente trabalho teve como objetivo apresentar uma abordagem de desenvolvimento de uma aplicação por meio de diversas tecnologias *web*, como *NodeJs*, *VueJs*, e *JavaScript*. Foram realizadas diferentes atividades de desenvolvimento de *software*, incluindo análise, design, implementação e testes, demonstrando a viabilidade e utilidade de se empregar conhecimentos em programação, engenharia de *software* e desenvolvimento *web*. Para isso, o trabalho foi baseado em um estudo de caso, no qual foi desenvolvida uma aplicação *web* com funcionalidades de agendamento de consultas, cujos requisitos e detalhes foram inspirados na Seção de Atenção à Saúde e Assistência Social (SASAS) da UFES. Nesta parte do texto, serão apresentados os desafios e considerações sobre o aprendizado envolvidos na realização deste trabalho, além de aspectos técnicos do uso das tecnologias de *front-end* e *back-end* adotadas.

A realização da revisão bibliográfica permitiu não apenas a identificação das tecnologias mais adequadas para o desenvolvimento da aplicação *web*, mas também proporcionou a oportunidade de aprofundar o conhecimento sobre novas ferramentas. Esse estudo foi fundamental para guiar as decisões técnicas e metodológicas do projeto, possibilitando a escolha de tecnologias modernas como *Node*.js, *Vue*.js e *Express*. A partir dessa análise, foram definidos os requisitos técnicos e estruturadas as etapas do processo de desenvolvimento, abrangendo tanto a escolha de ferramentas quanto às decisões relacionadas à arquitetura do sistema.

A aplicação desenvolvida, que possui funcionalidades básicas de sistemas de agendamento, serviu como uma plataforma para explorar o uso de tecnologias modernas, proporcionando uma base para futuras implementações de sistemas de apoio ao trabalho do setor de consultas da UFES. Durante o desenvolvimento, os desafios incluíram o aprendizado de novas tecnologias e a aplicação de conceitos de engenharia de *software* que foram aprendidos ao longo do curso, exigindo soluções criativas e adaptações constantes. Esse processo contribuiu significativamente para a compreensão do autor sobre as práticas e metodologias de desenvolvimento de software. Conclui-se que o estudo de caso permitiu uma visão

aprofundada do processo de criação de uma aplicação *web* funcional, alcançando os objetivos de demonstrar como o uso integrado de diferentes tecnologias pode resultar em um sistema prático e eficiente. Espera-se que este trabalho sirva como referência para futuros estudos na área de desenvolvimento de *software*, especialmente em ambientes acadêmicos, e que incentive novas pesquisas e inovações tecnológicas no contexto de agendamentos e outros sistemas web.

6.1 Trabalhos Futuros

Em função do desenvolvimento do sistema de agendamento apresentado neste Trabalho de Conclusão de Curso (TCC), várias direções para trabalhos futuros podem ser exploradas a fim de aprimorar a eficiência e a funcionalidade do sistema. Neste trabalho foi conduzido um estudo de caso do uso de tecnologias *web* para o desenvolvimento de uma aplicação da Seção de Atenção à Saúde e Assistência Social da Universidade Federal do Espírito Santo. Por se tratar de um serviço de grande relevância para a instituição e com a crescente informatização dos serviços institucionais que vem ocorrendo há vários anos, a aplicação tratada no estudo de caso deste trabalho pode evoluir para uma sistema que pode ser incorporado aos sistemas de informação web da UFES.

Assim, com relação ao tipo de aplicação aqui tratada, entre as possíveis propostas de trabalhos futuros é possível destacar:

- Projeto e implementação de um sistema que contemple as principais necessidades da Implementação de novos requisitos e integração da ferramenta aos sistemas de informação da UFES.
- Requisitos adicionais para a aplicação:
    - Lembretes Automáticos por email: A inclusão de um sistema de lembretes automáticos para envio de e-mails com informações das consultas, tanto para pacientes quanto para profissionais, pode ajudar a reduzir faltas e otimizar o fluxo de atendimentos.
    - Histórico de Atendimentos: Criar um histórico acessível de atendimentos permitirá um acompanhamento mais eficaz da saúde dos

pacientes, fornecendo informações valiosas para os profissionais na tomada de decisões clínicas.

- Avaliação do Serviço: A implementação de um sistema que possibilite aos pacientes avaliar o atendimento e a experiência com o sistema de agendamento será crucial para promover melhorias contínuas e garantir que o sistema se adapte às expectativas dos usuários.

Este estudo de caso foi realizado a partir de uma análise simplificada das necessidades da Seção de Atenção à Saúde e Assistência Social (SASAS), levando em consideração as práticas atuais e demandas da instituição. Espera-se que os resultados deste trabalho possam servir de base para futuras pesquisas e aprimoramentos na área de sistemas de informação web. Além disso, o estudo tem potencial para inspirar a ampliação de suas aplicações, transformando-o em um sistema mais abrangente que atenda às necessidades específicas do setor SASAS.

REFERÊNCIAS

ALVES, William P. *Banco de Dados*. São Paulo: Editora Saraiva, 2014. ISBN 9788536518961. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788536518961/. Acesso em: 09 jan. 2023.

ALURA. O que é front-end e back-end? Disponível em: https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end?srsltid=AfmBOopF2evl1yfqIn8p5aPE-Ajaag7nwS7vs7i1oAiT_A5A7_TQBZg5. Acesso em: 05 out. 2024.

AMPLIMED. *Amplimed - Gestão para clínicas e consultórios*. Disponível em: https://www.amplimed.com.br. Acesso em: 02 out. 2024.

ANEGAWA, T. H. et al. Agendamento online de doações de sangue durante a pandemia: relato de experiência no Hemocentro Regional de Londrina-PR. *Hematology, Transfusion and Cell Therapy*, Londrina, 2020.

AXIOS. Introdução ao Axios. Disponível em: https://axios-http.com/ptbr/docs/intro. Acesso em: 16 set. 2024.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 3. ed. Boston: Addison-Wesley, 2012.

BENLIAN, Alexander; KETTINGER, William J.; SUNYAEV, Ali; WINKLER, Thomas J. The transformative value of cloud computing: A decoupling, platformization, and recombination theoretical framework. *Journal of Management Information Systems*, v. 35, n. 3, p. 719-739, 2018. DOI: https://doi.org/10.1080/07421222.2018.1481634.

CARVALHO, Vinicius. Responsabilidade do Model do padrão MVC. *Alura*, 12 maio 2020. Disponível em: https://cursos.alura.com.br/forum/topico-responsabilidade-do-model-do-padrao-mvc-112439. Acesso em: 13 nov. 2023.

CLOUDFLARE. O que é um endpoint de API? 2022. Disponível em: https://www.cloudflare.com/pt-br/learning/security/api/what-is-api-endpoint/. Acesso em: 27 out. 2023.

DELAMARO, Márcio. *Introdução ao Teste de Software*. Grupo GEN, 2016. E-book. ISBN 9788595155732. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595155732/. Acesso em: 04 dez. 2023.

EAGENDA. *eAgenda - Solução completa de agendamentos online*. 2024. Disponível em: https://eagenda.com.br/. Acesso em: 02 out. 2024.

EBAC. Desenvolvedor front-end: o que faz. Disponível em: https://ebaconline.com.br/blog/desenvolvedor-front-end-o-que-faz. Acesso em: 05 out. 2024.

EXPRESS. *Express - Node.js web application framework*. 2023. Disponível em: https://expressjs.com/. Acesso em: 02 out. 2024.

EVANS, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. 1. ed. Boston: Addison-Wesley, 2004.

FLANAGAN, D. *JavaScript: The Definitive Guide*. 7. ed. O'Reilly Media, 2020.

FILIPOVA, Olga. *Learning Vue.js 2*. India: Packt Publishing, 2016.

GARCIA, Rafael. *Desenvolvimento Web: Fundamentos e Práticas*. São Paulo: Editora Novatec, 2019.

GARCÍA, Óscar; SILVA, Luís. E-commerce during the COVID-19 pandemic: Challenges and opportunities. *International Journal of Digital Commerce*, v. 7, n. 2, p. 15-29, 2021.

GITHUB. About GitHub. 2022. Disponível em: https://github.com/about. Acesso em: 02 out. 2024.

GUEDES, Gilleanes T. A. *UML 2: uma abordagem prática*. 3. ed. São Paulo: Novatec Editora, 2018.

GUEDES, Marylene. O que é MVC? *Treinaweb*, 22 ago. 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-mvc. Acesso em: 12 nov. 2023.

INSOMNIA. Introduction to Insomnia. Disponível em: https://docs.insomnia.rest/insomnia/get-started. Acesso em: 16 set. 2024.

JAVASCRIPT. *JavaScript Guide*. 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide. Acesso em: 02 out. 2024.

JWT. Introduction to JSON Web Tokens. 2022. Disponível em: https://jwt.io/introduction. Acesso em: 12 set. 2023.

JWT. *JSON Web Tokens*. 2023. Disponível em: https://jwt.io/. Acesso em: 02 out. 2024.

KIERAS, Ramon W. Sistema para agendamento de serviços. 2019. 50 f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2019.

LARSEN, J.; GRACE, B. *Building Applications with Node.js and Vue.js*. Manning Publications, 2019.

MORAIS, Izabelly S.; ZANIN, Aline. *Engenharia de software*. Grupo A, 2020. E-book. ISBN 9788595022539. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595022539/. Acesso em: 07 dez. 2023.

NODEJS. About Node.js. 2023. Disponível em: https://nodejs.org/en/about/. Acesso em: 02 out. 2024.

OLAPOSI, Dolabomi Daniel. Desenvolvimento de um sistema de marcação online de consultas médicas para o Centro de Saúde Universitário Mountain Top. 2022. Acesso em: 13 jan. 2023.

POSTGRESQLGLOBALDEVELOPMENTGROUP. *PostgreSQL 15 Documentation*. 2023. Disponível em: https://www.postgresql.org/docs/15/index.html. Acesso em: 02 out. 2024.

PRISMA. *Prisma: Next-generation ORM for Node.js and TypeScript*. Disponível em: https://www.prisma.io. Acesso em: 12 set. 2024.

SILVA, Fabricio Machado da; LEITE, Márcia Cristina D.; OLIVEIRA, Diego Bittencourt de. *Paradigmas de programação*. Grupo A, 2019. ISBN 9788533500426. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788533500426/. Acesso em: 09 jan. 2023.

SIQUEIRA, Sandra Regina Cardoso; EVANGELISTI, Luiz Renato Gonçalves. *Sistema de Agendamento Universal*.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. p. 69, cap. 4.5.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. p. 59, cap. 4.1.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. p. 60, cap. 4.1.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. p. 159, cap. 8.4.

SOMMERVILLE, Ian. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. p. 240, cap. 13.2.

SOUTO, Mario. Front-end, Back-end e Full Stack. *Alura*, 2022. Disponível em: https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end#o-que-e-o-full-stack. Acesso em: 05 out. 2024.

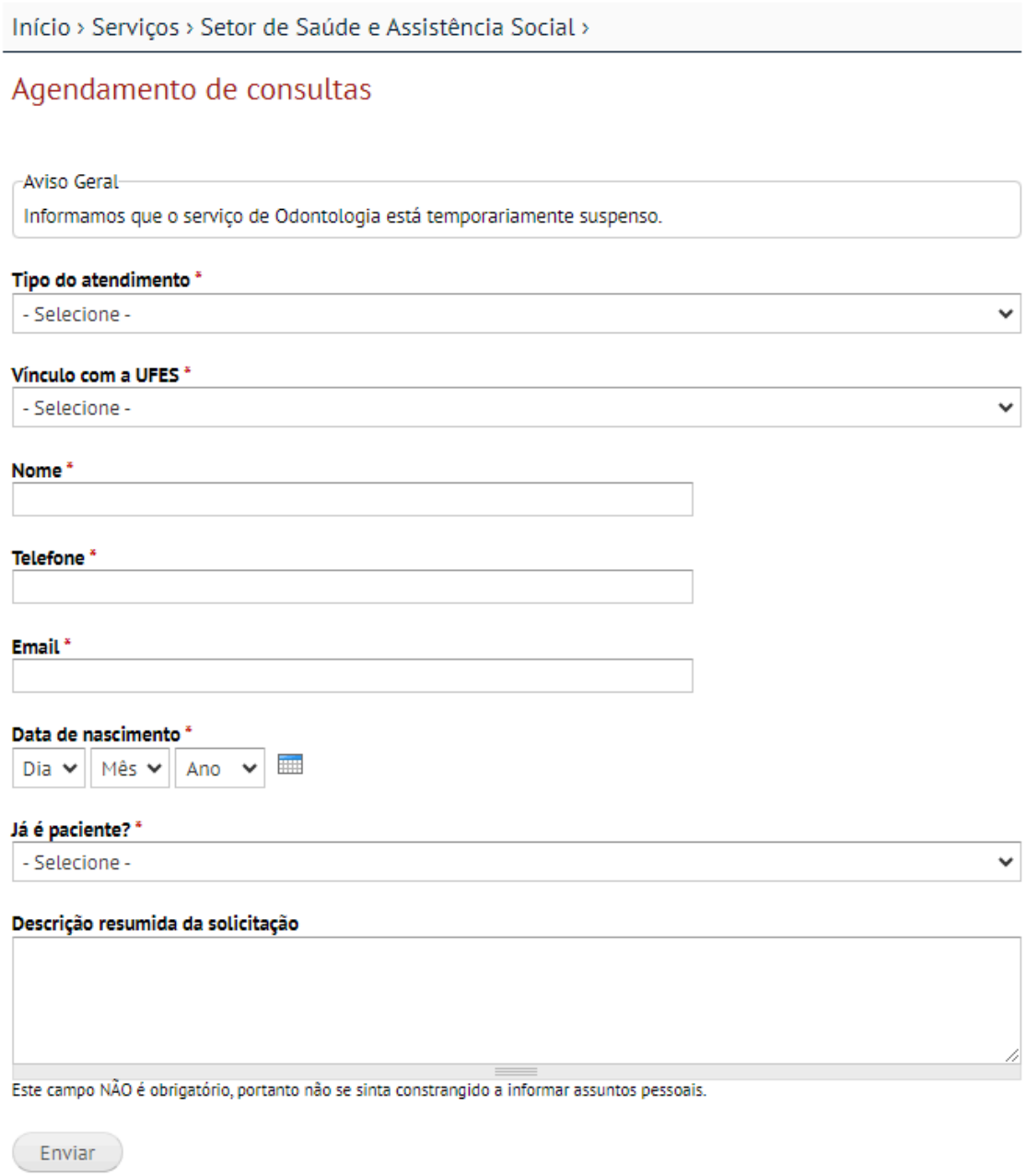
VUEJS. *Vue.js - The Progressive JavaScript Framework*. 2023. Disponível em: https://vuejs.org/. Acesso em: 02 out. 2024.

VUETIFY**.** Vue.js Material Component Framework. 2023. Disponível em: https://vuetifyjs.com/en/. Acesso em: 19 nov. 2023.

ANEXO A - FORMULÁRIOS PARA SOLICITAÇÃO DE AGENDAMENTO

Na Figura 29, apresentada abaixo, é mostrada a tela de solicitação de consultas do Setor de Saúde e Assistência Social.

Figura 29 - Agendamento de consultas(SASAS)



Fonte(UFES, 2024)

Na Tabela 6 abaixo, é apresentada as perguntas do formulário de identificação do estudante para solicitar uma consulta.

Tabela 5 - Formulário 1(2024): Identificação de estudante

Formulário 1(2024): Identificação de estudante	
Campo	Descrição
Nome Completo	Campo para o usuário inserir o nome completo. *(obrigatório)*
Número de matrícula	Campo para o usuário inserir o número de matrícula. *(obrigatório)*
Este espaço é livre para observações, perguntas e/ou explicações	Campo de texto para observações, perguntas e explicações.
Você já foi atendido pelo serviço de psicologia da UFES?	Seção com as opções: Sim ou Não *(obrigatório)*
Endereço eletrônico (e-mail)	Campo para o usuário inserir o e-mail de contato. *(obrigatório)*
Telefone para contato	Campo para o usuário inserir o número de telefone, incluindo o código de área.
Você é estudante de:	Seção com as opções: Graduação ou Pós-graduação. *(obrigatório)*
Você é cadastrado na Assistência Estudantil?	Seção com as opções: Sim ou Não. *(obrigatório)*
Este espaço é livre para observações, perguntas e/ou explicações	Campo de texto para informações adicionais.

Fonte: (UFES, 2024)

Na Tabela 6 abaixo, é mostrado as perguntas do primeiro formulário de questões preparatórias para agendar com o serviço de Atendimento Psicológico.

Tabela 6 - Formulário 2 (2024) : Questões preparatórias para atendimento psicológico

Formulário 2 (2024) 1 : Questões preparatórias para atendimento psicológico	
Campo	Descrição
Nome Completo	(obrigatório)
Data de Nascimento	(obrigatório, formato Data)
Matrícula	(obrigatório)
Você é estudante de:	Graduação / Pós-graduação (obrigatório)
Seu Centro é o:	Centro de Ciências Agrárias e Engenharias (CCAE) / Centro de Ciências Exatas, Naturais e da Saúde (CCENS) (obrigatório)
Seu Curso é:	(obrigatório, selecione na lista)
Você teve que sair do local em que residia e se mudar para Alegre ao entrar na UFES?	Não / Sim (obrigatório)

Atualmente qual é a sua composição familiar?	(Liste as pessoas que fazem parte de seu núcleo familiar, por exemplo: pai, mãe, irmãos...)
Com quem você reside em Alegre?	(obrigatório, selecione na lista)
Explique com suas palavras e de forma breve os motivos que te levaram a procurar atendimento psicológico	(obrigatório)

Fonte: (UFES, 2024)

Na Tabela 7 abaixo, é mostrado as perguntas do segundo formulário de questões preparatórias para agendar com o serviço de Atendimento Psicológico

Tabela 7 - Formulário 3 (2024) : Questões preparatórias para atendimento psicológico

Formulário 2 (2024) 2: Questões preparatórias para atendimento psicológico	
Campo	Descrição
Declara que entendeu os objetivos, riscos e benefícios da sua participação na pesquisa e concorda em participar?	Seção com as opções: Sim ou Não (obrigatório)

Afirmação	Resposta
1. Este problema acontece há bastante tempo.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
2. O problema me causa grande incômodo ou sofrimento.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
3. O problema tem mudado a minha maneira habitual de ser.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
4. Costumo pensar e refletir, constantemente, sobre os meus problemas.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
5. O problema atinge e/ou está presente em mais de uma área da minha vida (ex: família, relações interpessoais, vida social, vida acadêmica...)	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
6. Quando tenho problemas, costumo 'botar para fora' o que sinto.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
7. Me sinto motivado para realizar o que for necessário para resolver o problema.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE

8. O problema está atingindo várias das minhas atividades diárias (sono, alimentação, estudo, lazer, trabalho e convívio com amigos, por exemplo)	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
9. Na psicoterapia eu focaria na resolução de meus problemas diários (estudo, trabalho, dinheiro e relacionamentos) mais do que no autoconhecimento.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
10. Tenho o hábito de conversar com outras pessoas sobre os meus problemas.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
11. O problema atinge várias de minhas tarefas cotidianas (comer, dormir, trabalhar, estudar, divertir-me).	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
12. A intervenção indicada para o problema será uma das prioridades em minha vida.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
13. O problema acontece há tanto tempo que já se tornou parte de mim.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE

14. Geralmente tenho o hábito de refletir sobre os conselhos e orientações que recebo.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
15. Penso que a principal causa de meus problemas sou eu mesmo.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
16. O problema acontece com muita frequência em minha vida.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
17. Quando tenho problemas, costumo 'me fechar' e tentar resolvê-los sozinho(a).	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
18. Eu deixaria de ir a outros compromissos para comparecer aos atendimentos.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
19. O problema já mudou a maneira como eu me vejo e/ou como me comporto.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE
20. Procurei ajuda por vontade própria.	NUNCA - 1 - 2 - 3 - 4 - SEMPRE

Fonte: (UFES, 2024)

APÊNDICE B - NARRATIVAS DE CASO DE USO

UC 2 - Solicitar Agendamento de Consulta

Resumo: Este caso de uso descreve o processo pelo qual um paciente solicita um agendamento de consulta.

Ator: Paciente

Pré-condições:

- O paciente deve estar logado no sistema.

Fluxo Principal:

1. [RS] O sistema exibe a tela de agendamento com opções de tipo de atendimento e descrição
2. [EV] O paciente seleciona o tipo de atendimento, preenche a descrição.
3. [EV] O paciente confirma a solicitação de consulta clicando em "Solicitar Consulta".
4. [RS] O sistema valida as informações e envia a solicitação de agendamento de consulta.
5. [RS] O sistema exibe uma mensagem de confirmação **MSG002** indicando que a solicitação foi realizada com sucesso.

Fluxos Alternativos:

- [FA1] Se o paciente não preencher todos os campos obrigatórios, o sistema exibe uma mensagem de erro **MSG003** e solicita o preenchimento completo dos campos.

Pós-condições:

- A consulta é registrada no sistema e está aguardando confirmação, com status de solicitada.

UC 3 - Verificar Situação de Consultas Solicitadas

Resumo: Este caso de uso descreve o processo pelo qual um paciente verifica a situação de suas consultas solicitadas.

Ator: Paciente

Pré-condições:

- O paciente deve estar logado no sistema.
- O paciente deve ter pelo menos uma consulta agendada.

Fluxo Principal:

1. [RS] O sistema exibe a lista de consultas solicitadas com o status de cada uma (solicitada, confirmada ou cancelada).
2. [EV] O paciente revisa os detalhes das consultas, incluindo data e horário.

Fluxos Alternativos:

- [FA1] Se o paciente não tiver consultas agendadas, o sistema exibe uma mensagem **MSG005** informando que não há consultas no momento.

Pós-condições:

- O paciente visualiza o status de suas consultas.

UC 4: Visualizar Agenda do Profissional Saúde

Resumo: Este caso de uso descreve o processo pelo qual um Administrador pode visualizar as consultas que um profissional pode ter, através da funcionalidade Agenda

Ator: Administrador

Pré-condições:

- O usuário deve estar logado no sistema.

Fluxo Principal:

1. [RS] O sistema exibe um componente de calendário com marcações em laranja, indicando que o profissional tem uma consulta.
2. O administrador revisa as informações e vê os dias que o profissional tem consultas marcadas.

Pós-condições:

- O Administrador visualizar a agenda do profissional selecionado

UC 5: Agendar Consulta

Resumo: Este caso de uso descreve o processo pelo qual o Administrador confirma o agendamento de uma consulta com um profissional.

Ator: Administrador

Pré-condições:

- O Administrador deve estar logado no sistema.
- O Administrador deve selecionar um horário disponível.

Fluxo Principal:

1. [EV] O Administrador acessa as consultas, escolhe a opção de ver detalhes de uma determinada consulta
2. [EV] O Administrador acessa a funcionalidade de editar consulta
3. [EV] O administrador modifica os campos de data da consulta, profissional, e status, para “Confirmado”
4. [RS] O sistema registra o agendamento e atualiza a agenda do profissional.
5. [RS] O sistema exibe uma mensagem de confirmação **MSG004** informando que a consulta foi agendada com sucesso.

Fluxos Alternativos:

- [FA1] Caso os campos não forem todos devidamente preenchidos, o sistema exibe a seguinte mensagem **MSG003**

Pós-condições:

- O agendamento é confirmado e registrado no sistema.

Tabela 8 - Mensagens dos casos de uso

Código	Mensagem
MSG001	Usuário não encontrado
MSG002	Consulta Solicitada com sucesso
MSG003	Preencha todos os campos corretamente
MSG004	Consulta agendada com Sucesso!
MSG005	Nenhuma consulta solicitada

Fonte: Do autor.

APÊNDICE C - DESCRIÇÃO DAS TABELAS DO BANCO DE DADOS

Tabela 9 - Tabela paciente

Coluna	Tipo de Dados	Descrição
id	Int	Chave primária, identificador único do paciente
nome	Varchar (255)	Nome do paciente
tipo	Varchar (255)	Tipo de paciente (pode ser algum status ou categoria)
matricula	Varchar (255)	Número de matrícula do paciente
dataNascimento	Date	Data de nascimento do paciente
telefone	Varchar (255)	Número de telefone do paciente (opcional)
consultas	Relacionamento	Relacionamento com a tabela "consultas" (múltiplas consultas)
usuarioId	Int	Chave estrangeira para relacionamento com a tabela "usuarios"
notificacoes	Relacionamento	Relacionamento com a tabela "notificacoes" (múltiplas notificações)

Fonte: Do autor.

Tabela 10 - Tabela consulta

Coluna	Tipo de Dados	Descrição
id	Int	Chave primária, identificador único da consulta
data	Date	Data da consulta (opcional)

status	Varchar (255)	Status da consulta (pode ser agendada, realizada, cancelada, etc.)
observacao	Varchar (255)	Observações sobre a consulta
servico	Varchar (255)	Tipo de serviço da consulta (opcional)
pacienteId	Int	Chave estrangeira para relacionamento com a tabela "pacientes"
profissionalId	Int	Chave estrangeira para relacionamento com a tabela "profissionais"
data_solicitacao	Date	Data e hora da solicitação da consulta

Fonte: Do autor.

Tabela 11 - Tabela profissional

Coluna	**Tipo de Dados**	**Descrição**
id	Int	Chave primária, identificador único do profissional
nome	Varchar (255)	Nome do profissional
especialidade	Varchar (255)	Especialidade do profissional
consultas	Relacionamento	Relacionamento com a tabela "consultas" (múltiplas consultas)

Fonte: Do autor.

Tabela 12 - Tabela notificacao

Coluna	Tipo de Dados	Descrição
id	Inteiro	Chave primária, identificador único da notificação
mensagem	Varchar (255)	Mensagem da notificação
pacienteId	Int	Chave estrangeira para relacionamento com a tabela "pacientes"

Fonte: Do autor.

Tabela 13 - Tabela compromisso

Coluna	Tipo de Dados	Descrição
id	Int	Chave primária, identificador único do compromisso
titulo_compromisso	Varchar (255)	título do compromisso
data_compromisso	Date	Data do compromisso
hora_inicio	Date	horário de início do compromisso
hora_fim	Date	horário de fim do compromisso
profissional_id		Chave estrangeira para relacionamento com a tabela "profissionais"
consulta_id		Chave estrangeira para relacionamento com a tabela "consultas"

Fonte: Do autor.