INSTITUTO FEDERAL GOIANO
CAMPUS URUTAÍ
NÚCLEO DE INFORMÁTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

MATEUS ELIAS VIEIRA

AUTOMAÇÃO DA EQUOTERAPIA POR MEIO DE WEB SERVICES

Urutaí, 18 de setembro de 2024

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

Formulário 705/2024 - DE-UR/CMPURT/IFGOIANO

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

CURSO DE SISTEMAS DE INFORMAÇÃO

AUTOMAÇÃO DA EQUOTERAPIA POR MEIO DE WEB SERVICES

Autores: MATEUS ELIAS VIEIRA

Orientador(a): Cristiane de Fátima dos Santos Cardoso

Monografia defendida por MATEUS ELIAS VIEIRA, e aprovada em 23 de agosto de 2024, pela banca examinadora constituída pelos membros:

APROVADA em 23 de agosto de 2024

gov.br Documento assinado digitalmente
CRISTIANE DE FATIMA DOS SANTOS CARDOSO
Data: 29/08/2024 14:14:58-0300
Verifique em https://validar.iti.gov.br

Profª. Drª. Cristiane de Fátima dos Santos Cardoso

Presidente da Banca

gov.br Documento assinado digitalmente
AMAURY WALBERT DE CARVALHO
Data: 05/09/2024 10:16:54-0300
Verifique em https://validar.iti.gov.br

Profª. Me. Amaury Walbert de Carvalho

Avaliador

Profª. Drª. Nattane Luiza da Costa

gov.br Documento assinado digitalmente
NATTANE LUIZA DA COSTA
Data: 30/08/2024 08:22:15-0300
Verifique em https://validar.iti.gov.br

Avaliadora

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas – SIBI/IF Goiano

V658a Vieira, Mateus Elias.

Automação da equoterapia por meio de web services [manuscrito] / Mateus Elias Vieira. -- Urutaí, GO: IF Goiano, 2024.
134 fls.

Orientadora: Prof. Dra. Cristiane de Fátima dos Santos

Dissertação (Mestrado em Ensino para a Educação Básica) – Instituto Federal Goiano, Campus Urutaí, 2024.

1. Software Web. 2. Web Services. 3. Spring. 4. React. 5. Equoterapia. 6. Gestão de dados. I. Título. II. IF Goiano - Campus Urutaí.

CDU 004.7

Ficha elaborada por Johnathan Pereira Alves Diniz – Bibliotecário/CRB 1 nº 2376

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO

PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano a disponibilizar gratuitamente o documento em formato digital no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

IDENTIFICAÇÃO DA PRODUÇÃO TÉCNICO-CIENTÍFICA

- ☐ Tese (doutorado)
- ☐ Dissertação (mestrado)
- ☐ Monografia (especialização)
- ☑ TCC (graduação)
- ☐ Artigo científico
- ☐ Capítulo de livro
- ☐ Livro
- ☐ Trabalho apresentado em evento
- ☐ Produto técnico e educacional - Tipo:

Nome completo do autor:
Mateus Elias Vieira

Matrícula:
2019101202010116

Título do trabalho:
AUTOMAÇÃO DA EQUOTERAPIA POR MEIO DE WEB SERVICES

RESTRIÇÕES DE ACESSO AO DOCUMENTO

Documento confidencial: ☑ Não ☐ Sim, justifique:

Informe a data que poderá ser disponibilizado no RIIF Goiano: 12 /09 /2024

O documento está sujeito a registro de patente? ☐ Sim ☑ Não

O documento pode vir a ser publicado como livro? ☐ Sim ☑ Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O(a) referido(a) autor(a) declara:

- Que o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
- Que obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autoria, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
- Que cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Urutaí — Local
12 /09 /2024 — Data

Documento assinado digitalmente
gov.br
MATEUS ELIAS VIEIRA
Data: 12/09/2024 23:00:41-0300
Verifique em https://validar.iti.gov.br

Assinatura do autor e/ou detentor dos direitos autorais

Ciente e de acordo:

Documento assinado digitalmente
gov.br
CRISTIANE DE FATIMA DOS SANTOS CARDOSO
Data: 12/09/2024 23:10:30-0300
Verifique em https://validar.iti.gov.br

Assinatura do(a) orientador(a)

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

Formulário 706/2024 - DE-UR/CMPURT/IFGOIANO

INSTITUTO FEDERAL GOIANO – CAMPUS URUTAÍ

DIRETORIA / GERÊNCIA DE GRADUAÇÃO E PÓS-GRADUAÇÃO

COORDENAÇÃO DOS CURSOS DA ÁREA DE INFORMÁTICA

CURSO SUPERIOR DE SISTEMAS DE INFORMAÇÃO

ATA DE APRESENTAÇÃO DE TRABALHO DE CURSO

Aos 23 dias do mês de agosto de dois mil e vinte e quatro, reuniram-se os professores: Cristiane de Fátima dos Santos Cardoso, Amaury Walbert de Carvalho e Nattane Luiza da Costa no núcleo de Informática do Instituto Federal Goiano - Campus Urutaí, para avaliar o Trabalho de Curso do(s) acadêmico(s): **MATEUS ELIAS VIEIRA**, como requisito necessário para a conclusão do Curso Superior de Sistemas de Informação desta Instituição. O presente TC tem como título: AUTOMAÇÃO DA EQUOTERAPIA POR MEIO DE WEB SERVICES, foi orientado por Cristiane de Fátima dos Santos Cardoso.

Após análise, foram dadas as seguintes notas:

Professores	**Aluno / Notas**	
	MATEUS ELIAS VIEIRA	
1. Cristiane de Fátima dos Santos Cardoso	8,8	
2. Amaury Walbert de Carvalho	8,2	
3. Nattane Luiza da Costa	8,4	

MÉDIA FINAL:	8,5	

OBSERVAÇÕES: Realizar correções sugeridas pela banca e submeter o trabalho a nova avaliação por parte do orientador até 23/09/2024. A versão final deve ser depositada no RIIF GOIANO (Repositório Institucional do IF Goiano) até o dia 01/10/2024, conforme tutorial https://www.youtube.com/watch?v=l8h2mNilaMA

Por ser verdade firmamos a presente.

Documento assinado eletronicamente por:

- Nattane Luiza da Costa, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 23/08/2024 15:16:04.
- Amaury Walbert de Carvalho, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 23/08/2024 15:16:01.
- Cristiane de Fatima dos Santos Cardoso, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 23/08/2024 15:13:45.

Este documento foi emitido pelo SUAP em 22/08/2024. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 625472
Código de Autenticação: f4ddccdd10

INSTITUTO FEDERAL GOIANO
Campus Urutaí
Rodovia Geraldo Silva Nascimento, Km 2.5, SN, Zona Rural, URUTAÍ / GO, CEP 75790-000
(64) 3465-1900

AGRADECIMENTOS

Agradeço primeiramente a Deus por estar comigo em todos os momentos difíceis da minha vida, me dando forças para continuar lutando todos os dias. Agradeço aos meus pais, Iron e Adriana, por me apoiarem nessa jornada, me motivando e ajudando em tudo que estava ao seu alcance. Agradeço aos meus irmãos, Lucas, Vitória, Arthur e Fernanda, por me apoiarem em todas as áreas da minha vida e por compartilharem tantos momentos de alegria. Agradeço à minha namorada Ana Clara por me trazer tanta paz, alegria e carinho em minha vida. Agradeço à minha avó Luziana pelo seu amor e carinho especial com todos seus netos. Agradeço ao meu tio Iran pelos inúmeros conselhos que foram fundamentais em minha vida. Agradeço aos meus avôs João Batista e Sebastião e à minha avó Valdivina, que moram nos céus, por todo amor e carinho que me deram.

RESUMO

O presente trabalho aborda a concepção e implementação de um software web empregando Web Services com Spring e React, destinado à equipe de equoterapia do Instituto Federal Goiano Campus Urutaí. O propósito primordial consiste em otimizar a gestão de dados dos praticantes e funcionários, visando a simplificação, celeridade e aprimoramento desse processo. A metodologia adotada compreendeu a pesquisa e aplicação de tecnologias como Spring Boot e React.js, objetivando a criação de uma plataforma de fácil usabilidade e alta performance. Os resultados obtidos englobam um sistema capaz de efetuar cadastros, atualizações e consultas de informações dos envolvidos, contribuindo de forma substancial para a organização e eficiência das atividades de equoterapia. Como resultado é apresentado um software que oferece uma solução efetiva e prática para o gerenciamento de dados nesse contexto específico, impactando positivamente no trabalho da equipe da equoterapia e na vida dos pacientes que são atendidos.

Palavras-chave:

Software web, Web Services, Spring, React, Equoterapia, Gestão de dados.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID Atomicidade, Consistência, Isolamento e Durabilidade. Pag.31

AJAX *Asynchronous JavaScript and XML* ou JavaScript e XML Assíncronos. Pag.29

API *Application Programming Interface* ou Interface de Programação de Aplicações. Pag.22

CMS *Content Management System* ou Sistema de Gerenciamento de Conteúdo. Pag.32

CSRF *Cross-Site Request Forgery* ou Falsificação de solicitação entre sites. Pag.65

CSS *Cascading Style Sheet* ou Folha de Estilo em Cascatas. Pag.28

DOM *Document Object Model* ou Modelo de Objeto de Documento. Pag.29

DTO *Data Transfer Object* ou Objeto de Transferência de Dados. Pag.25

HTML *Hypertext Markup Language* ou Linguagem de Marcação de HiperTexto. Pag.28

HTTP *Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto. Pag.19

IDE *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado. Pag.30

IIS *Internet Information Services* Serviços de Informação da Internet. Pag.33

JSON *JavaScript Object Notation* ou Notação de Objeto JavaScript. Pag.25

JVM *Java Virtual Machine* ou Máquina Virtual Java. Pag.27

JWT *JSON Web Token*. Pag.25

NoSQL *No Structured Query Language* ou Linguagem de Consulta Não Estruturada. Pag.31

REST *Representational State Transfer* ou Transferência de Estado Representacional. Pag.23

SQL *Structured Query Language* ou Linguagem de Consulta Estruturada. Pag.31

UML *Unified Modeling Language* ou Linguagem de Modelagem Unificada. Pag.30

URI *Uniform Resource Identifier* ou Identificador Uniforme de Recurso. Pag.23

WWW *World Wide Web*. Pag.19

XSS *Cross-Site Scripting* ou Script entre sites. Pag.65

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

A equoterapia é uma forma de terapia que utiliza cavalos como parceiros terapêuticos para promover o desenvolvimento físico, emocional, cognitivo e social de pessoas com diferentes tipos de necessidades especiais. Essa prática terapêutica tem sido amplamente reconhecida por seus benefícios multifacetados, que vão desde o fortalecimento muscular e aprimoramento da coordenação motora até o aumento da autoconfiança e autoestima dos praticantes. Atividades como montar, escovar e cuidar dos cavalos proporcionam um ambiente estimulante e motivador, onde os indivíduos podem interagir de maneira terapêutica com os animais (CAMARGO, 2014).

A interação com os cavalos oferece não apenas benefícios físicos, mas também promove a socialização e a integração dos participantes em diferentes contextos. Isso é particularmente importante para pessoas com deficiências, que muitas vezes enfrentam barreiras significativas à inclusão social. A equoterapia, nesse sentido, desempenha um papel social crucial ao proporcionar oportunidades de inclusão e participação ativa na sociedade, contribuindo para a melhoria da qualidade de vida e bem-estar geral dos praticantes (CAMARGO, 2014).

Além dos benefícios diretos aos praticantes, a equoterapia também pode ser uma fonte de emprego e renda para profissionais da área. A demanda por centros e programas de equoterapia tem incentivado o desenvolvimento de instalações dedicadas e a formação de equipes especializadas. Esses centros não só fornecem terapia, mas também servem como espaços de aprendizado e desenvolvimento para profissionais da saúde e do bem-estar animal.

Atualmente, o centro de equoterapia do IF enfrenta uma grande desatualização em relação à tecnologia inserida no ambiente. Como consequência, a forma de organização dos relatórios dos praticantes se dá por meio de uma enorme quantidade de papéis armazenados em um armário de aço. Isso traz vários problemas para a equipe, como a dificuldade e o grande gasto de tempo na busca por documentos, além da possibilidade de perdas devido a eventos inesperados, como fogo, água, traças, entre outros.

Sendo assim, a automação deste centro de equoterapia se torna indispensável. A gestão eficiente desse centro é essencial para garantir a qualidade e a continuidade dos serviços prestados. A complexidade envolvida na administração de atividades, horários, profissionais e informações dos praticantes demanda uma solução organizada e eficaz. Nesse contexto, a utilização de um sistema de gerenciamento se torna fundamental. Um sistema de *web service* pode resolver a parte de gerenciamento dos dados da equipe e dos praticantes, melhorando a experiência e o aproveitamento geral dos recursos.

De acordo com a World Wide Web Consortium (W3C) (2004), um serviço da web é um sistema de software projetado para oferecer suporte à interação máquina a máquina interoperável em uma rede. A implementação de tal sistema no contexto da equoterapia pode informatizar e facilitar a gestão de cadastros, consultas e manipulação de dados. Esse processo de informatização não só aumenta a eficiência administrativa, mas também libera mais tempo e recursos para focar no atendimento terapêutico.

Portanto, o objetivo deste trabalho é desenvolver um sistema web para o gerenciamento das informações dos funcionários e praticantes da Equoterapia do Instituto Federal Goiano - Campus Urutaí. A proposta é informatizar e facilitar toda a parte de cadastros, consultas e manipulação de dados em geral, proporcionando uma administração mais eficaz e, consequentemente, uma melhor qualidade nos serviços prestados.

1.1 Estrutura do Trabalho

O Capítulo 1 apresenta o tema do projeto, abordando a equoterapia, seus benefícios e a gestão de dados dos praticantes. O Capítulo 2 cobre os fundamentos teóricos necessários, incluindo protocolos, servidores, linguagens de programação, UML, bibliotecas, frameworks e bancos de dados. No Capítulo 3, são detalhados os materiais e métodos utilizados, incluindo levantamento de requisitos, características do sistema, diagramas UML e tecnologias empregadas. O Capítulo 4 descreve o desenvolvimento do back-end, abordando estrutura de pacotes, tecnologias e frameworks usados, diagrama do banco de dados e trechos de código. O Capítulo 5 trata do desenvolvimento do front-end, discutindo tecnologias, frameworks, bibliotecas de terceiros e a estrutura de pastas. O Capítulo 6 apresenta imagens do sistema, incluindo telas, mensagens de erro e formulários, além das diferenças entre as interfaces de administrador e usuário. Por fim, no último capítulo é abordado sobre a conclusão do projeto.

CAPÍTULO 2

FUNDAMENTOS TEÓRICOS

Para a construção do sistema de web services, é preciso compreender vários assuntos para dar continuidade ao desenvolvimento. Dessa forma, neste capítulo são abordados todos os assuntos necessários para conhecer antes de prosseguir com a construção do sistema proposto.

2.1 HTTP

O Protocolo HTTP (*Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto) é um protocolo de comunicação utilizado para transferência de dados na WWW (*World Wide Web*) . Ele define a forma como as mensagens são formatadas e transmitidas, além das ações que devem ser tomadas por servidores e clientes web. Tim Berners-Lee, o inventor da WWW, em seu livro "*Weaving the Web*", descreve o HTTP como fundamental para a comunicação entre computadores na web, permitindo a troca de informações de forma padronizada e acessível (BERNERS-LEE, 1999). A Tabela 2.1 mostra as principais características do protocolo HTTP:

2.1.1 Métodos HTTP

Os métodos HTTP são comandos utilizados para indicar a ação que deve ser realizada em um recurso identificado por uma URL. Cada método possui uma semântica específica, determinando como o servidor deve processar a requisição feita pelo cliente. Tim Berners-Lee, aborda a utilidade dos métodos HTTP na estrutura da web, notando que os métodos HTTP fornecem uma abstração uniforme para a interação entre cliente e servidor" (BERNERS-LEE, 1999). A Tabela 2.2 traz uma síntese dos métodos HTTP:

Características	Descrição
Stateless	O HTTP é um protocolo *stateless*, o que significa que cada requisição entre cliente e servidor é independente das anteriores, sem memória de estado entre requisições.
Baseado em Texto	As mensagens HTTP são formatadas em texto legível (como cabeçalhos e corpo), facilitando a depuração e compreensão do tráfego entre clientes e servidores.
Métodos de Requisição	Define métodos como GET, POST, PUT, DELETE, entre outros, que especificam a ação a ser realizada no recurso identificado pela URL.
Códigos de Status	Utiliza códigos de status (como 200 OK, 404 *Not Found*, 500 *Internal Server Error*) para indicar o resultado da tentativa de realizar a requisição.

Tabela 2.1: Tabela de características e descrições do protocolo HTTP.

Método HTTP	Descrição
GET	Solicita a representação de um recurso específico. É um método seguro e idempotente, ou seja, não deve ter efeitos colaterais além da recuperação de dados.
POST	Envia dados para serem processados por um recurso identificado. É utilizado para criar novos recursos no servidor ou enviar dados para serem processados por um recurso específico.
PUT	Atualiza um recurso existente com os dados fornecidos. Substitui completamente o recurso, se existir, ou cria um novo recurso se não existir.
DELETE	Remove um recurso identificado pela URL.
PATCH	Aplica modificações parciais a um recurso.
OPTIONS	Retorna os métodos HTTP suportados pelo servidor para a URL especificada.

Tabela 2.2: Exemplos de métodos HTTP e suas descrições.

2.1.2 Status HTTP

Os status HTTP, são códigos enviados pelo servidor em resposta a uma solicitação HTTP feita por um cliente (como um navegador). Eles indicam se a solicitação foi bem-sucedida, se houve algum erro ou se é necessário realizar alguma ação adicional. Os códigos de status HTTP são divididos em cinco categorias:

Status Http	Tipo	Descrição
100 a 199	Informativo	Indica que a solicitação foi recebida e o processo está em andamento.
200 a 299	Sucesso	Indica que a solicitação foi recebida, entendida e aceita com sucesso.
300 a 399	Redirecionamento	Indica que o cliente precisa realizar ações adicionais para completar a solicitação.
400 a 499	Erro do Cliente	Indica que houve um problema com a solicitação feita pelo cliente.
500 a 599	Erro do Servidor	Indica que o servidor encontrou um erro ao tentar processar a solicitação.

Tabela 2.3: Tabela de status HTTP, tipo e suas descrições.

Por conseguinte, existem alguns status HTTP que são mais famosos por serem mais utilizados. Dessa forma, a partir da Tabela 2.3, destaca-se como principais status HTTP: Sucesso, Erro do Cliente e Erro do Servidor. Sendo assim, a Tabela 2.4 mostra alguns desses status, tipo e descrição:

Status Http	Tipo	Descrição
200 *OK*	Sucesso	Solicitação bem-sucedida e dados retornados.
201 *Created*	Sucesso	Novo recurso criado com sucesso.
204 *No Content*	Sucesso	Solicitação bem-sucedida sem retorno de conteúdo.
301 *Moved Permanently*	Redirecionamento	Recurso movido permanentemente para nova URL.
302 *Found*	Redirecionamento	Recurso encontrado em URL temporária.
400 *Bad Request*	Erro do Cliente	Solicitação inválida devido a sintaxe incorreta.
401 *Unauthorized*	Erro do Cliente	Requer autenticação do usuário.
403 *Forbidden*	Erro do Cliente	Solicitação entendida, mas não autorizada.
404 *Not Found*	Erro do Cliente	Recurso solicitado não encontrado.
500 *Internal Server Error*	Erro do Servidor	Erro inesperado no servidor.
503 *Service Unavailable*	Erro do Servidor	Servidor temporariamente indisponível.

Tabela 2.4: Tabela de status HTTP mais utilizados, tipos e suas descrições.

2.2 Web Services

Um *web service*, é um tipo de software que facilita a comunicação e a troca de dados entre diferentes sistemas e aplicações através da internet. Ele permite que programas escritos em linguagens de programação diferentes e rodando em plataformas distintas possam interagir e compartilhar informações de maneira eficiente. De acordo com Erl (2005), "um *web service* é qualquer pedaço de software que disponibiliza uma interface que pode ser acessada via rede utilizando protocolos de internet padronizados". A Figura 2.1, ilustra como diferentes dispositivos, ambos rodando um mesmo software em diversas plataformas, podem se comunicar de maneira eficiente com a API e o banco de dados. Essa comunicação é essencial para garantir a interoperabilidade e a troca contínua de dados entre os sistemas, independentemente do ambiente em que estejam sendo executados.

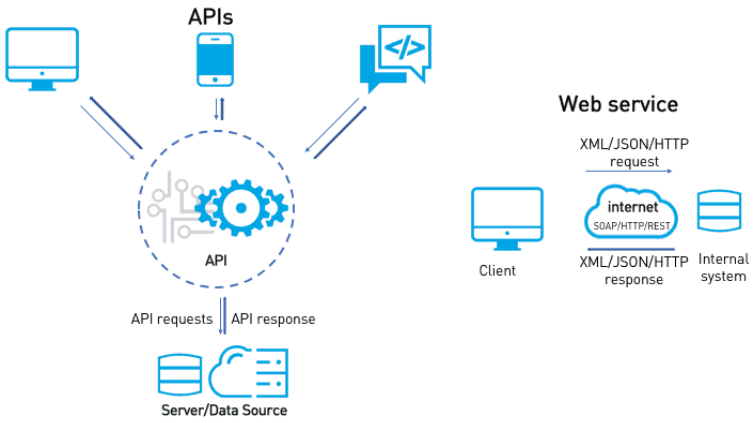


Figura 2.1: Web services, fluxo de funcionamento.
Fonte: https://rapidapi.com/blog/api-vs-web-service/. Acesso em 27/07/2024.

2.3 API

Uma API (*Application Programming Interface* ou Interface de Programação de Aplicações) é um conjunto de definições e protocolos que permite a comunicação entre diferentes softwares. APIs especificam como os componentes de software devem interagir e são usadas para facilitar a integração e a interoperabilidade entre sistemas. De acordo com Fielding (2000), "uma API define uma interface através da qual interações entre diferentes componentes de software são feitas de uma forma consistente e previsível". Alguns aspectos relevantes acerca das APIs são tratados nas seções a seguir.

2.3.1 API REST

Esse tipo de API é amplamente utilizada na construção de sistemas distribuídos, permitindo integrações entre diferentes plataformas e sistemas de maneira eficiente e escalável. Além disso, são essenciais para o desenvolvimento de aplicações web modernas, serviços em nuvem, aplicativos móveis e muitas outras soluções tecnológicas (TILKOV; WOLF, 2015).

Uma API REST (*Representational State Transfer* ou Transferência de Estado Representacional) é um estilo arquitetural usado para projetar serviços web que são simples, escaláveis e fáceis de integrar (RICHARDSON; RUBY, 2013). Baseia-se nos princípios fundamentais da web e utiliza os métodos HTTP padrão (como GET, POST, PUT, DELETE) para acessar e manipular recursos. Essa abordagem é amplamente adotada para criar APIs que são eficientes, flexíveis e bem estruturadas.

Sendo assim, em uma API REST, cada recurso (como um usuário, um produto ou uma transação) é representado por um URI (*Uniform Resource Identifier* ou Identificador Uniforme de Recurso) (FIELDING, 2000). As operações sobre esses recursos são definidas pelos métodos HTTP. Por exemplo, para obter informações de um usuário, uma solicitação GET é feita para o URI correspondente ao usuário específico. Da mesma forma, para criar um novo usuário, uma solicitação POST é feita para o URI da coleção de usuários.

Os princípios de uma API REST são mostrados na Tabela 2.5.

Princípios	**Descrição**
Estado e operações	Os recursos são manipulados através de operações bem definidas (GET para ler, POST para criar, PUT para atualizar, DELETE para excluir), refletindo a representação atual do recurso (estado).
Sem estado (*Stateless*)	Cada solicitação de cliente para o servidor deve conter todas as informações necessárias para entender e processar a solicitação. Isso significa que o servidor não mantém estado da sessão do cliente entre requisições.
Representações de recursos	Os recursos são representados em formatos padronizados, como JSON ou XML, que são facilmente interpretáveis por máquinas e humanos.
Interface uniforme	A interface da API deve ser simples e consistente, facilitando sua compreensão e uso por desenvolvedores.
Arquitetura cliente-servidor	A separação entre o cliente (quem faz as requisições) e o servidor (que processa essas requisições) é clara e definida, promovendo uma estrutura de comunicação eficiente.

Tabela 2.5: Tabela de princípios da arquitetura REST e suas descrições

2.3.2 API RESTful

API RESTful refere-se a uma API que segue os princípios e restrições definidos pelo estilo de arquitetura REST. Ou seja, uma API pode ser chamada de RESTful se ela implementar e aderir completamente aos princípios do REST. Isso inclui, por exemplo, o uso de métodos HTTP (GET, POST, PUT, DELETE) de maneira semântica, a manipulação de recursos via URIs, e a comunicação sem estado entre cliente e servidor. A partir da Figura 2.2, podemos entender melhor como ocorre o fluxo de requisição e resposta em uma API REST:

1º O cliente faz uma requisição HTTP, utilizando algum de seus métodos, para o servidor.

2º O servidor recebe a requisição e verifica qual endpoint poderá atender aquela requisição.

3º A API realiza o processamento interno, acessando serviços, repositórios e banco de dados.

4º O servidor envia a resposta para o cliente com os dados requisitados e o status da resposta.

5º O cliente recebe os dados e, através de sua lógica, gerencia a resposta da melhor forma possível.

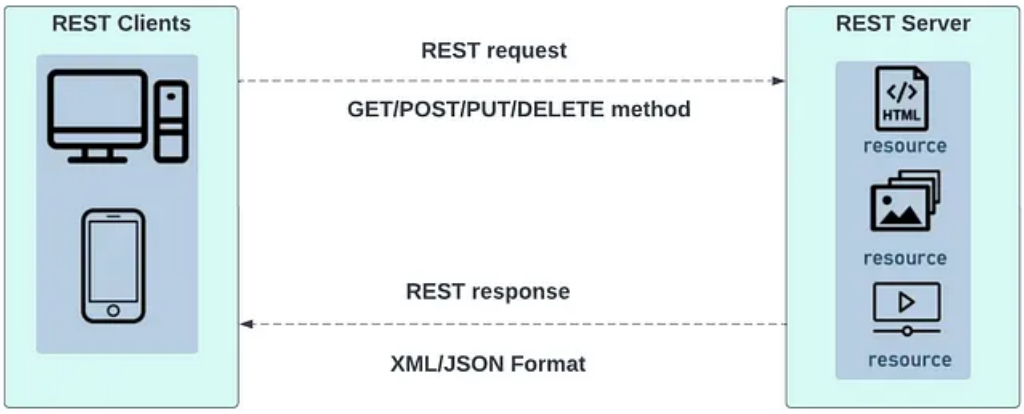


Figura 2.2: API REST, requisição e resposta.
Fonte: https://medium.com/codenx/representational-state-transfer-rest-and-design-principles-98640faa1ab4. Acesso em 28/07/2024.

2.3.3 DTO

O DTO (*Data Transfer Object* ou Objeto de Transferência de Dados) , é um padrão de design usado para transportar dados entre processos, especialmente em arquiteturas de sistemas distribuídos como APIs. O objetivo principal dos DTOs é encapsular os dados que serão transferidos entre a camada de apresentação e a camada de negócios ou de persistência, simplificando a comunicação e reduzindo a complexidade. A camada de apresentação é responsável por interagir com o usuário final, gerenciando a interface e capturando as entradas do usuário. Já a camada de negócios, ou camada de lógica de aplicação, contém as regras de negócio e processos que definem o comportamento do sistema. A camada de persistência, por sua vez, lida com o armazenamento e recuperação dos dados em bancos de dados ou outros sistemas de armazenamento. Segundo Fowler (2002), "O DTO é usado para transferir dados entre diferentes partes de um sistema ou entre sistemas. É um objeto que transporta dados entre processos, facilitando o transporte de múltiplas informações de uma vez sem precisar lidar diretamente com estruturas de dados complexas". A Tabela 2.6 mostra os principais benefícios do uso de DTOs.

Benefícios	**Descrição**
Desacoplamento	DTO's ajudam a desacoplar as diferentes camadas da aplicação, evitando que mudanças em uma camada impactem diretamente outras.
Otimização de Dados	Eles permitem a transferência de apenas dados necessários, reduzindo o tráfego de rede e melhorando a eficiência.
Segurança	DTO's podem ajudar a expor apenas os dados necessários e esconder informações sensíveis ou desnecessárias para o cliente.

Tabela 2.6: Tabela de benefícios do uso de DTO's no sistema.

2.3.4 Token JWT

JWT (*JSON Web Token*) , Figura 2.3, é um padrão aberto (RFC 7519) que define um modo compacto e autônomo de transmitir informações com segurança entre as partes como um objeto JSON (*JavaScript Object Notation* ou Notação de Objeto JavaScript) . Estas informações podem ser verificadas e confiadas porque são assinadas digitalmente. Os tokens JWT são comumente usados para autenticação e autorização em sistemas distribuídos. A Tabela **??** descreve cada uma das partes de um token JWT.

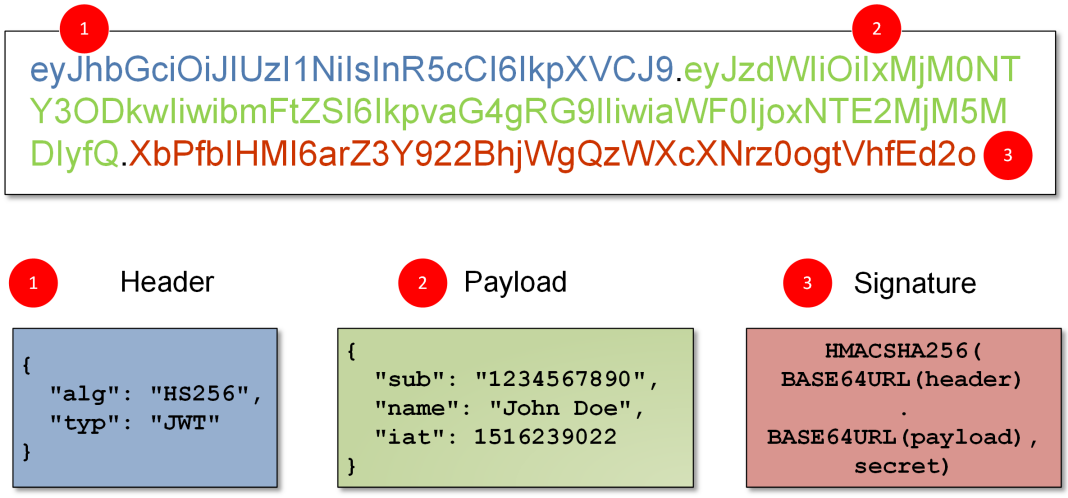


Figura 2.3: Token JWT, estrutura.
Fonte: https://research.securitum.com/jwt-json-web-token-security/. Acesso em 28/07/2024.

Como foi observado na Figura 2.3, o Token JWT é composto por três partes principais:

Header: Contém o tipo de token (JWT) e o algoritmo de assinatura (por exemplo, HMAC SHA256 ou RSA).

Payload: Contém as declarações (claims), que são as informações que queremos transmitir. Essas declarações podem ser de três tipos:

- **Reservadas**: predefinidas no padrão.
- **Públicas**: definidas livremente.
- **Privadas**: definidas por um acordo entre as partes que utilizam o token.

Signature: Usada para verificar se a mensagem não foi alterada. A assinatura é criada combinando o header codificado, o payload codificado e uma chave secreta usando o algoritmo especificado no header. Essa estrutura garante a integridade e a autenticidade das informações transmitidas no token.

2.4 Back-end

O back-end, ou "lado do servidor", refere-se à parte de um sistema de software que lida com o armazenamento e processamento de dados, bem como com a lógica de negócios. De acordo com Evans (2015), "o back-end de um sistema de software é a camada responsável por processar solicitações, armazenar e manipular dados, e garantir que o front-end tenha os recursos necessários para funcionar corretamente". É no back-end onde se encontra a API e por sua vez o web service.

2.5 Java

Java é uma linguagem de programação de propósito geral, conhecida por sua portabilidade, segurança e robustez. Segundo Deitel e Deitel (2017), "Java é uma linguagem de programação de propósito geral, orientada a objetos, concisa, baseada em classes e projetada para ter poucas dependências de implementação".

Java foi desenvolvida pela Sun Microsystems (adquirida pela Oracle Corporation) e é amplamente utilizada para desenvolvimento de aplicativos empresariais, aplicativos móveis (Android), sistemas embarcados, e muitas outras aplicações. Ela se destaca pela sua capacidade de escrever código uma vez e executá-lo em diferentes plataformas sem necessidade de recompilação, graças ao uso da JVM (*Java Virtual Machine* ou Máquina Virtual Java) .

A linguagem Java suporta programação orientada a objetos, oferece uma biblioteca rica de API's e é amplamente adotada na indústria devido à sua combinação de eficiência, segurança e facilidade de uso.

2.6 Framework

Um framework de software pode ser definido como uma estrutura abstrata e reutilizável que fornece suporte genérico para a construção e organização de sistemas de software. Segundo Johnson (2003), "um framework é uma estrutura padronizada que pode ser expandida por um desenvolvedor para criar soluções específicas".

Frameworks frequentemente incluem componentes de software genéricos, classes abstratas e métodos que podem ser personalizados para atender requisitos específicos de um projeto. Eles permitem aos desenvolvedores concentrar-se na lógica específica de suas aplicações, ao invés de reinventar funcionalidades comuns, como manipulação de dados, gerenciamento de

sessão, autenticação, entre outros.

Os frameworks são amplamente utilizados para acelerar o desenvolvimento de software, melhorar a manutenção do código, promover boas práticas de design e facilitar a colaboração entre desenvolvedores. No contexto de desenvolvimento front-end e back-end, o uso de frameworks oferece vantagens e desvantagens que devem ser consideradas.

2.6.1 Spring

O Spring Framework é um framework de aplicação Java de código aberto e baseado em padrões, que oferece suporte abrangente para o desenvolvimento de aplicativos empresariais. Segundo Wall et al. (2021), "o Spring é um framework Java abrangente, modular e baseado em componentes, que simplifica o desenvolvimento de aplicativos corporativos em Java, promovendo boas práticas de programação e aumentando a produtividade do desenvolvedor".

Desenvolvido pela Pivotal Software (anteriormente SpringSource), o Spring Framework facilita a criação de aplicativos robustos e escaláveis, utilizando conceitos como injeção de dependência e inversão de controle para promover uma arquitetura flexível e modular. Ele também inclui diversos módulos que suportam desde o desenvolvimento de aplicações web até a integração com sistemas de persistência de dados, segurança, e serviços em nuvem.

Embora seja amplamente utilizado em aplicações corporativas, o Spring não se limita a este contexto. Devido à sua versatilidade, o Spring pode ser utilizado em uma ampla gama de projetos, incluindo microserviços, aplicações móveis, soluções em nuvem, e até mesmo em projetos de Internet das Coisas (IoT), proporcionando eficiência e facilidade de manutenção ao longo do ciclo de vida do desenvolvimento de software.

2.7 Front-end

O termo "front-end" refere-se à parte de um sistema de software ou aplicação que os usuários interagem diretamente. É a interface gráfica e funcional que permite aos usuários visualizar e interagir com dados e funcionalidades do sistema. Segundo Freeman e Robson (2004), "o front-end é a parte do sistema que os usuários finais veem e interagem diretamente".

No contexto web, o front-end é construído utilizando tecnologias como HTML (*Hypertext Markup Language* ou Linguagem de Marcação de HiperTexto) , CSS (*Cascading Style Sheet* ou Folha de Estilo em Cascatas) e JavaScript, que definem a estrutura, o estilo e a interatividade dos elementos na página web. Frameworks modernos de front-end, como React, Angular e Vue.js,

facilitam o desenvolvimento de interfaces de usuário dinâmicas e responsivas, permitindo uma experiência de usuário mais rica e interativa.

2.7.1 Javascript

JavaScript é uma linguagem de programação de alto nível, dinâmica e interpretada, amplamente utilizada para criar páginas web interativas e dinâmicas. Segundo Flanagan (2020), "JavaScript é uma linguagem de programação de alto nível, dinâmica e orientada a objetos usada principalmente para scripts em páginas web, mas também usada em outros ambientes".

Desenvolvida originalmente pela Netscape, JavaScript tornou-se uma linguagem fundamental para o desenvolvimento web moderno, permitindo aos desenvolvedores criar funcionalidades interativas, como validação de formulários, animações, manipulação do DOM (*Document Object Model* ou Modelo de Objeto de Documento) e comunicação assíncrona com o servidor via AJAX (*Asynchronous JavaScript and XML* ou JavaScript e XML Assíncronos) .

JavaScript é suportado por todos os principais navegadores web e, junto com HTML e CSS, forma o trio fundamental da tecnologia web.

A versatilidade e ubiquidade do JavaScript o tornam uma das linguagens de programação mais populares e essenciais para desenvolvedores web e móveis, permitindo a criação de experiências interativas e dinâmicas para usuários em todo o mundo.

2.7.2 React

Segundo Báez (2020), "React é uma biblioteca JavaScript de código aberto que permite criar interfaces de usuário com componentes reutilizáveis".

Desenvolvida pelo Facebook, React permite aos desenvolvedores criar interfaces de usuário declarativas, onde o estado da aplicação é atualizado automaticamente quando os dados mudam. Utilizando um conceito chamado de "*Component-Based Architecture*" (Arquitetura Baseada em Componentes), React permite que os desenvolvedores dividam a interface do usuário em componentes independentes e reutilizáveis, facilitando a manutenção e escalabilidade do código.

React é amplamente adotado na indústria devido à sua eficiência, desempenho e a capacidade de criar interfaces de usuário complexas de maneira intuitiva. É frequentemente utilizado em conjunto com outras tecnologias JavaScript, como Redux para gerenciamento de estado e React Router para navegação entre páginas em aplicações web.

2.8 IDE

Uma IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) é uma ferramenta de software que oferece recursos integrados para facilitar o desenvolvimento de software. Segundo Johnson (2010), "uma IDE combina ferramentas comuns de desenvolvimento, como um editor de código, depurador, compilador e muitas vezes um gerenciador de versões, em um único ambiente integrado".

As IDE's são projetadas para aumentar a produtividade dos desenvolvedores, oferecendo recursos como realce de sintaxe, preenchimento automático de código, depuração passo a passo, refatoração de código, integração com controle de versão e gerenciamento de projetos. Essas ferramentas ajudam os desenvolvedores a escrever, testar e depurar código de maneira mais eficiente e eficaz.

2.9 UML

A UML (*Unified Modeling Language* ou Linguagem de Modelagem Unificada) é uma linguagem padrão para a modelagem de sistemas de software. Ela fornece uma forma de visualizar, especificar, construir e documentar os artefatos de um sistema. A UML é amplamente utilizada na engenharia de software para representar a arquitetura, o design e a implementação de sistemas complexos, oferecendo uma coleção de diagramas que descrevem diferentes aspectos do sistema. Como afirmam Booch, Rumbaugh e Jacobson, os criadores da UML, "A UML foi criada para responder à necessidade de uma notação de modelagem padrão, uma vez que os métodos anteriores usavam notações diferentes, dificultando a comunicação entre equipes e a interoperabilidade de ferramentas"(MARTIN, 2009).

2.10 Astah

Astah é uma das ferramentas existentes para modelagem visual. É usada principalmente para criar diagramas UML e diagramas de software, facilitando o planejamento e a comunicação de projetos de software (ASTAH, 2020). Como mencionado por Smith (2021), "Astah permite a criação eficiente de diagramas UML, que são essenciais para a comunicação clara e a documentação de projetos de software"(SMITH, 2021). Dessa forma, foi usado para a criação do diagrama de casos de uso, diagrama de sequência e diagrama de classes. Segundo Jones (2019), "O uso de diagramas UML ajuda a obter uma visão clara e objetiva do sistema, facilitando o

início do desenvolvimento"(JONES, 2019).

2.11 Banco de Dados

Um banco de dados é um sistema organizado para armazenar e gerenciar grandes volumes de dados de forma eficiente. Segundo Elmasri e Navathe (2015), "um banco de dados é uma coleção de dados inter-relacionados, armazenados de forma organizada para atender às necessidades de múltiplos usuários em grandes organizações".

Os bancos de dados são fundamentais para a persistência e recuperação de informações em aplicações empresariais e sistemas de software. Eles permitem armazenar dados estruturados de forma que possam ser acessados, atualizados e consultados de maneira eficiente. Além disso, os bancos de dados oferecem recursos para garantir a integridade dos dados, como transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) , e permitem a implementação de esquemas de segurança para proteger informações sensíveis.

Existem diversos tipos de bancos de dados, como bancos de dados relacionais que usam linguagem SQL (*Structured Query Language* ou Linguagem de Consulta Estruturada) e bancos de dados não relacionais que usam linguagem NoSQL (*No Structured Query Language* ou Linguagem de Consulta Não Estruturada) que são otimizados para armazenar dados não estruturados ou semi-estruturados, e também bancos de dados em memória (como Redis) que oferecem alta velocidade de acesso aos dados.

2.11.1 Banco de Dados Relacional

Um banco de dados relacional é um tipo de banco de dados que organiza os dados em tabelas relacionadas umas com as outras através de chaves estrangeiras. Segundo Silberschatz et al. (2010), "Um banco de dados relacional é um conjunto de tabelas, onde cada tabela possui um conjunto de linhas e colunas, e cada coluna representa um atributo específico dos dados que estão sendo armazenados".

Neste modelo, os dados são estruturados em linhas e colunas, e as relações entre as tabelas são estabelecidas através de chaves estrangeiras que garantem a integridade referencial dos dados. Isso permite a realização de consultas complexas utilizando a linguagem SQL, que é padronizada e amplamente utilizada para manipulação e consulta de dados em bancos de dados relacionais.

Os bancos de dados relacionais oferecem várias vantagens, como consistência dos

dados, integridade referencial, suporte a transações ACID, e a capacidade de realizar operações complexas de junção entre tabelas. Exemplos populares de sistemas de gerenciamento de banco de dados relacionais incluem MySQL, PostgreSQL, SQL Server, Oracle Database, entre outros.

Os bancos de dados relacionais são amplamente adotados em aplicações empresariais e sistemas críticos onde a estrutura e a consistência dos dados são essenciais. Eles são ideais para cenários onde é necessário suportar transações complexas e garantir a integridade dos dados ao longo do tempo.

2.11.2 MySQL

MySQL é um sistema de gerenciamento de banco de dados relacional de código aberto amplamente utilizado. Segundo Dubois (2016), "MySQL é um sistema de gerenciamento de banco de dados relacional, que suporta um subconjunto de SQL para a gestão de dados".

Desenvolvido inicialmente pela MySQL AB, adquirida posteriormente pela Oracle Corporation), MySQL é conhecido por sua confiabilidade, desempenho e facilidade de uso. Ele é frequentemente usado em aplicações web e outras aplicações que exigem um banco de dados robusto para armazenamento e recuperação de dados estruturados.

MySQL suporta várias características importantes de bancos de dados relacionais, incluindo integridade referencial, transações ACID, índices para otimização de consultas, e replicação para alta disponibilidade e escalabilidade. Além disso, é compatível com diferentes plataformas, como Windows, Linux, MacOS, e várias linguagens de programação, tornando-o uma escolha popular entre desenvolvedores e administradores de banco de dados.

Como um sistema de código aberto, MySQL possui uma comunidade ativa de desenvolvedores e usuários que contribuem para sua evolução e suporte contínuo. É amplamente utilizado em CMS (*Content Management System* ou Sistema de Gerenciamento de Conteúdo) como WordPress, plataformas de comércio eletrônico, aplicativos de análise de dados e muito mais.

2.12 Servidor

Um servidor, no contexto da tecnologia da informação, é um sistema de computador ou software que fornece serviços, recursos ou funcionalidades para outros dispositivos ou programas, chamados de clientes. Segundo Tanenbaum e Steen (2007), "um servidor é um computador que executa programas que aceitam conexões de clientes através da rede e fornecem algum serviço

em resposta a essas conexões".

Os servidores são projetados para serem confiáveis, escaláveis e capazes de lidar com um grande número de solicitações simultâneas. Eles podem fornecer uma variedade de serviços, como hospedagem de websites, armazenamento de dados, gerenciamento de arquivos, processamento de e-mails, gerenciamento de rede, entre outros.

Existem diferentes tipos de servidores, cada um otimizado para desempenhar funções específicas. Por exemplo, um servidor web (como Apache ou Nginx) é dedicado a hospedar e servir páginas web e outros conteúdos web. Um servidor de banco de dados (como MySQL ou PostgreSQL) é projetado para armazenar e gerenciar grandes volumes de dados estruturados. Servidores de aplicativos (como Tomcat, JBoss ou WildFly) hospedam e executam aplicativos web e serviços. Além disso, há servidores de arquivos, servidores de e-mail, servidores DNS, entre outros tipos.

Os servidores podem ser físicos (hardware dedicado) ou virtuais (executados em máquinas virtuais ou contêineres). Eles geralmente operam em sistemas operacionais robustos e seguros, como Linux ou Windows Server, e podem ser configurados e gerenciados remotamente para garantir alta disponibilidade e desempenho.

2.12.1 Servidor Web

Um servidor web, Figura 2.4, é um computador com alto poder de processamento com um software projetado para processar requisições HTTP de clientes (como navegadores web) e entregar conteúdos requisitados, como, páginas HTML, imagens, arquivos CSS e JavaScript, aos clientes que as solicitam. Segundo Gourley e Totty (2002), "Um servidor web é um programa que aceita requisições HTTP de clientes, como navegadores web, e serve arquivos que compõem páginas web para esses clientes".

Os servidores web são essenciais para a infraestrutura da WWW, permitindo que empresas e indivíduos publiquem e disponibilizem conteúdo na internet. Eles podem ser genéricos e servir diversos tipos de conteúdo estático e dinâmico, ou especializados para atender necessidades específicas de aplicativos web complexos.

Exemplos populares de servidores web incluem o Apache HTTP Server, Nginx, IIS (*Internet Information Services* Serviços de Informação da Internet) , e LiteSpeed Web Server. Cada um desses servidores web possui suas próprias características e configurações que os tornam adequados para diferentes cenários, desde hospedagem simples de páginas estáticas até

suporte para aplicações web dinâmicas e escaláveis.

Além de servir conteúdo estático, servidores web modernos também suportam tecnologias para aplicativos web dinâmicos, como PHP, Python, Ruby, Node.js, e frameworks JavaScript como React e Angular, através de módulos e configurações específicas.

Sendo assim, o sistema de equoterapia desenvolvido, será hospedado e armazenado em servidores específicos, como servidor web e de banco de dados. Dessa forma, o mesmo estará disponível para ser acessado e também armazenar dados.

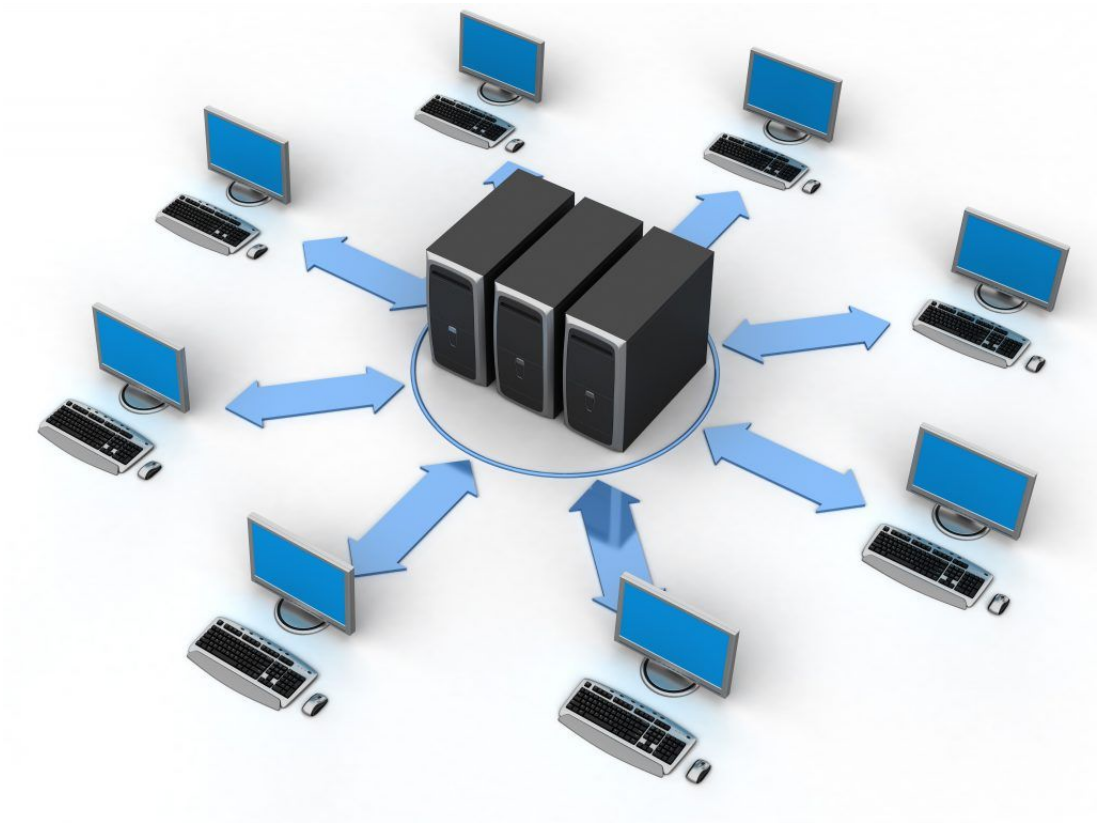


Figura 2.4: Servidores web.
Fonte: https://raul-profesor.github.io/DEAW/ArqWeb/. Acesso em 28/07/2024.

2.12.2 Servidor de Banco de Dados

Um servidor de banco de dados, Figura 2.5, é um computador com alto poder de processamento e armazenamento com um software responsável por gerenciar e fornecer acesso ao HD do computador, permitindo assim, realizar diversos tipos de operações, como, armazenar, alterar, deletar e consultar dados de forma segura e eficiente. Segundo Connolly e Begg (2014), "Um servidor de banco de dados é um sistema de software que gerencia um banco de dados para que o usuário final e outros programas possam acessar os dados".

Além de suportarem operações, como abordado anteriormente, garantem a integridade, segurança e disponibilidade dos dados armazenados. Eles também oferecem mecanismos para otimizar consultas, gerenciar transações e garantir a consistência dos dados por meio de técnicas como transações ACID.

Os servidores de banco de dados podem ser de diferentes tipos, dependendo do modelo de dados que suportam. Além disso, os servidores de banco de dados podem ser instalados em ambientes físicos dedicados (hardware dedicado) ou em ambientes virtualizados (máquinas virtuais ou contêineres), oferecendo flexibilidade na implementação e gerenciamento.

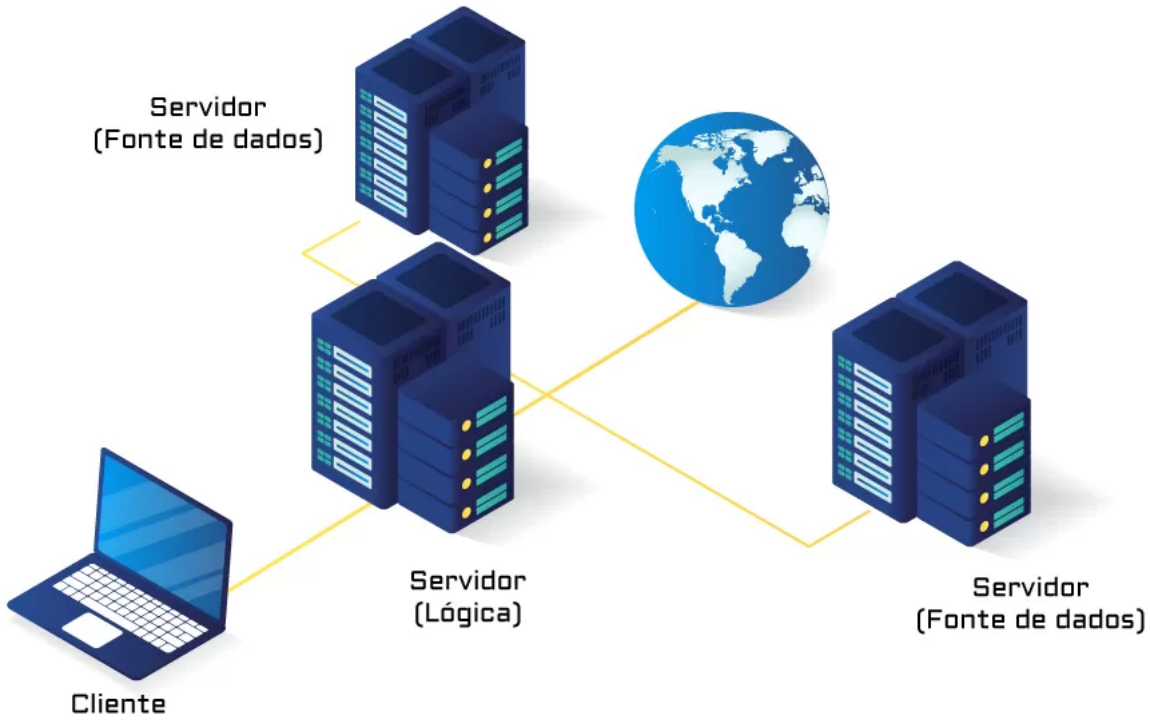


Figura 2.5: Servidores de banco de dados.
Fonte: https://dcomp-it.com.br/solucoes-em-ti/solucoes-em-servidores/. Acesso em 28/07/2024.

2.13 Versionamento de Código

O versionamento de código é uma prática essencial no desenvolvimento de software que envolve o gerenciamento de mudanças no código-fonte ao longo do tempo. Ferramentas de controle de versão, como Git, SVN, e Mercurial, são usadas para rastrear e registrar cada alteração no código, facilitando a colaboração entre desenvolvedores e mantendo um histórico detalhado do projeto. Segundo Fowler (2006), "os sistemas de controle de versão são uma ferramenta essencial para o desenvolvimento de software moderno. Eles permitem que os desenvolvedores rastreiem e gerenciem alterações em sua base de código, colaborem de forma mais eficaz e mantenham um histórico da evolução do projeto".

2.13.1 GIT

Git é um sistema de controle de versão distribuído amplamente utilizado no desenvolvimento de software moderno. Foi criado por Linus Torvalds em 2005 para ajudar no desenvolvimento do kernel do Linux, e rapidamente se tornou a ferramenta preferida para projetos de todos os tamanhos devido à sua eficiência, flexibilidade e robustez (TORVALDS, 2008). A Tabela 2.7 traz as principais características do GIT.

Características	**Descrição**
Distribuído	Cada clone de um repositório Git é um repositório completo com histórico completo e capacidade total de rastreamento de versões. Isso significa que cada desenvolvedor tem uma cópia completa do projeto, incluindo toda a sua história de mudanças
Eficiente	Git é projetado para ser rápido e eficiente no gerenciamento de grandes projetos com muitos arquivos e históricos de commits extensos. Operações comuns, como commit, branch e merge, são extremamente rápidas
Flexível	Suporta vários fluxos de trabalho de desenvolvimento, como branches e merges complexos, permitindo que equipes escolham o fluxo de trabalho que melhor se adapte às suas necessidades
Seguro	Utiliza algoritmos criptográficos para garantir a integridade do histórico e do conteúdo do código. Cada commit é identificado por um hash SHA-1, garantindo que a história do projeto não possa ser alterada sem que isso seja detectado

Tabela 2.7: Tabela de características do GIT

2.13.2 GITHUB

GitHub é uma plataforma de hospedagem de código-fonte baseada na web que utiliza Git para controle de versão. Além de oferecer repositórios Git, GitHub fornece uma gama de funcionalidades adicionais que facilitam a colaboração, revisão de código e integração contínua (GITHUB, 2020). A Tabela 2.8 traz os principais benefícios do GITHUB.

Benefícios	**Descrição**
Colaboração Facilitada	Ferramentas como pull requests, code reviews e issues facilitam a colaboração entre desenvolvedores, promovendo um desenvolvimento de código mais transparente e de alta qualidade.
Integração com Ferramentas Externas	GitHub integra-se facilmente com várias outras ferramentas de desenvolvimento, como CI/CD, ferramentas de monitoramento e IDEs, criando um ecossistema robusto para o desenvolvimento de software.
Comunidade e Visibilidade	Repositórios públicos no GitHub podem ser acessados por uma grande comunidade de desenvolvedores, o que ajuda a obter feedback, contribuições e aumentar a visibilidade dos projetos.
Documentação e Wikis	GitHub permite a criação de documentação e wikis para projetos, facilitando o compartilhamento de informações e conhecimento sobre o código.

Tabela 2.8: Tabela de características do GITHUB

CAPÍTULO 3

LEVANTAMENTO E ANÁLISE DE REQUISITOS

Nesta seção, são detalhados as etapas iniciais para dar inicio ao desenvolvimento do sistema. Por conseguinte, o processo foi estruturado em etapas que incluem o levantamento de requisitos, análise de requisitos, implementação, testes, correções e entrega. O fluxo de desenvolvimento seguiu um passo a passo lógico, conforme ilustrado no fluxograma da Figura 3.1.

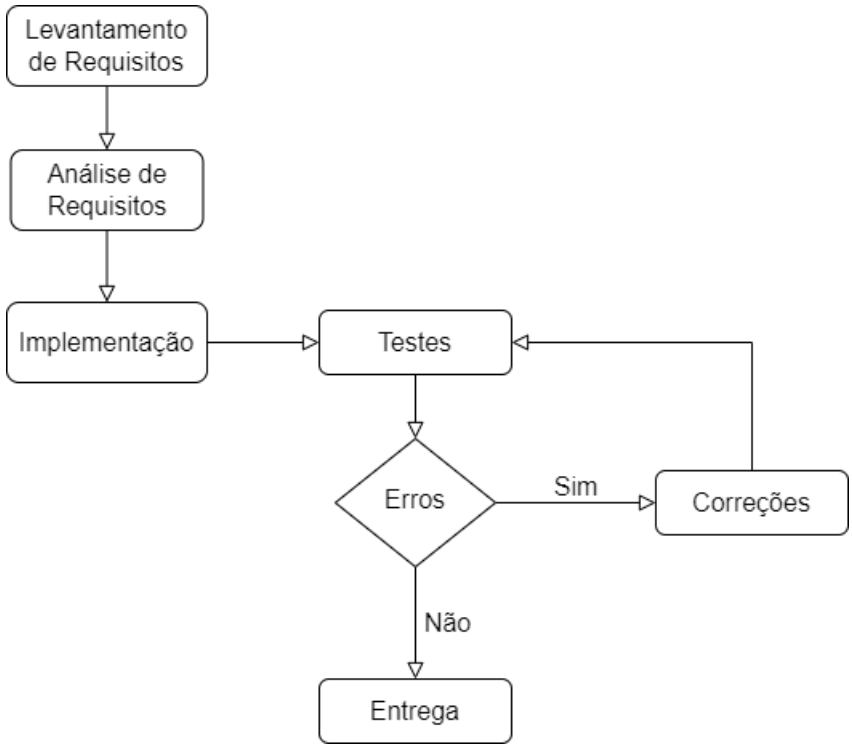


Figura 3.1: Fluxograma dos processos de construção do sistema.

3.1 Levantamento e Análise de Requisitos

O levantamento de requisitos foi realizado junto à equipe de gestão da equoterapia. Durante essa etapa, foram identificadas diversas necessidades e funcionalidades essenciais para melhorar o processo de gestão e utilização dos recursos da equoterapia do Instituto Federal Goiano - Campus Urutaí. O gestor entrevistado expressou a necessidade de substituir o método tradicional de cadastro e armazenamento manual das fichas dos praticantes por uma solução automatizada e mais eficiente.

Foi discutida a importância de implementar um sistema capaz de gerar relatórios detalhados sobre o progresso dos praticantes, gráficos que facilitassem a visualização de dados relevantes, além de funcionalidades robustas para cadastro, consulta e atualização de informações. A necessidade de uma interface intuitiva e acessível foi destacada, visando facilitar o uso por parte dos profissionais envolvidos na *Equoterapia*, garantindo uma melhor experiência e eficiência operacional.

Adicionalmente, foram levantadas expectativas quanto à integração de tecnologias que pudessem facilitar o acompanhamento dos resultados das sessões, o registro de evoluções individuais dos praticantes, e a gestão administrativa geral do programa de Equoterapia.

3.1.1 Níveis de acesso do Usuário

Através da análise de requisitos, foram obtidos os dados necessários para cadastrar usuários no sistema. Estes usuários são pertencentes a equipe de Equoterapia do Instituto Federal Goiano Campus Urutaí. Sendo assim, existem usuários comuns e administradores, dessa forma, ambos devem ter papeis diferentes no uso do sistema. Os usuários administradores possuem acesso total ao sistema, podendo assim, gerenciar usuários e praticantes. Por fim, os usuários comuns, apenas podem gerenciar os praticantes.

3.1.2 Gerenciamento de Usuários

O gerenciamento de usuários envolve todas as operações relacionadas à criação, atualização, listagem, exclusão e autenticação de usuários no sistema. Para adicionar um novo usuário ao sistema, somente um outro usuário com nível administrador poderá fazer o cadastro do mesmo. O mesmo se aplica para outras ações sobre os usuários do sistema, ou seja, para visualizar, atualizar e deletar, será preciso que seja feito por um administrador.

3.1.3 Gerenciamento de Praticantes

O gerenciamento de praticantes refere-se à administração dos indivíduos que estão praticando do programa de Equoterapia. Envolve a criação, atualização, exclusão e visualização dos dados dos praticantes, além de monitorar seu progresso e atividades. Dessa forma, usuários com/sem nível de administrador poderá ter acesso a essas ações dentro do sistema.

3.1.4 Geração de Relatório do Praticante

A geração de relatórios dos praticantes é um processo que permite a coleta e apresentação de dados relevantes sobre o desempenho e progresso dos praticantes. Esses relatórios podem incluir estatísticas, gráficos e análises detalhadas, auxiliando na avaliação e no acompanhamento dos praticantes.

3.1.5 Evolução do Praticante

A evolução do praticante diz respeito ao acompanhamento contínuo do desenvolvimento e progresso dos praticantes ao longo do tempo. Inclui a análise de métricas de desempenho, feedback, e a implementação de planos de melhoria personalizados para cada praticante, com o objetivo de otimizar seus resultados e alcançar metas estabelecidas.

3.1.6 Diagramas UML

Neste projeto, foi escolhido a utilização dos diagramas de casos de uso, sequência e classe, devido aos seguintes motivos:

- **Diagrama de Casos de Uso**
  - **Motivo da Escolha**: O diagrama de casos de uso é fundamental para capturar os requisitos funcionais do sistema do ponto de vista do usuário. Ele representa as interações entre os atores (usuários ou outros sistemas) e o sistema, destacando as funcionalidades que o sistema deve oferecer.
  - **Benefícios**: Ajuda a definir claramente o escopo do sistema, garantindo que todas as funcionalidades necessárias sejam consideradas. Além disso, facilita a comunicação com os stakeholders, que podem facilmente entender e validar os requisitos.

- **Diagrama de Sequência**
  - **Motivo da Escolha**: O diagrama de sequência é utilizado para detalhar a lógica do fluxo de eventos dentro de um caso de uso ou operação específica. Ele mostra como os objetos do sistema interagem entre si ao longo do tempo.
  - **Benefícios**: Proporciona uma visualização clara da interação dinâmica entre diferentes componentes do sistema, ajudando a identificar possíveis problemas de comunicação e sincronização. Isso é crucial para garantir que o sistema funcione conforme o esperado.
- **Diagrama de Classes**
  - **Motivo da Escolha**: O diagrama de classes é essencial para definir a estrutura estática do sistema. Ele mostra as classes, seus atributos e métodos, e os relacionamentos entre elas.
  - **Benefícios**: Serve como base para o design orientado a objetos, garantindo que todos os elementos necessários sejam considerados e organizados corretamente. Além disso, facilita a transição do design para a implementação, pois os desenvolvedores podem usar o diagrama de classes como referência ao escrever o código.

Diagrama	Descrição
Casos de Uso	Representa as interações entre os usuários (atores) e o sistema, destacando as funcionalidades que o sistema deve oferecer
Sequência	Mostra a interação entre objetos ao longo do tempo, detalhando como os processos operam em sequência
Classes	Exibe a estrutura estática do sistema, mostrando classes, atributos, métodos e os relacionamentos entre elas

Tabela 3.1: Tabela de Diagramas UML

3.1.7 Diagrama de Casos de Uso

Foi necessário desenvolver um diagrama de casos de uso, Figura 3.2, como parte do processo de análise de requisitos do sistema de automação para a Equoterapia no Instituto Federal Goiano Campus Urutaí. O diagrama de casos de uso é uma representação visual que descreve as interações entre os usuários (atores) e o sistema, identificando os principais requisitos funcionais do sistema a partir da perspectiva dos usuários finais.

No contexto específico da Equoterapia, o diagrama de casos de uso ajudou a mapear como diferentes atores, como terapeutas, administradores e praticantes, interagem com o sistema. Cada caso de uso no diagrama representa uma função ou tarefa específica que o sistema deve ser capaz de realizar para atender às necessidades dos usuários. Isso inclui funcionalidades como cadastro de praticantes, agendamento de sessões, geração de relatórios, consultas de históricos de sessões, entre outras.

O desenvolvimento do diagrama de casos de uso permitiu uma compreensão clara dos fluxos de interação entre os usuários e o sistema, facilitando a identificação de requisitos essenciais e a definição das funcionalidades prioritárias a serem implementadas.

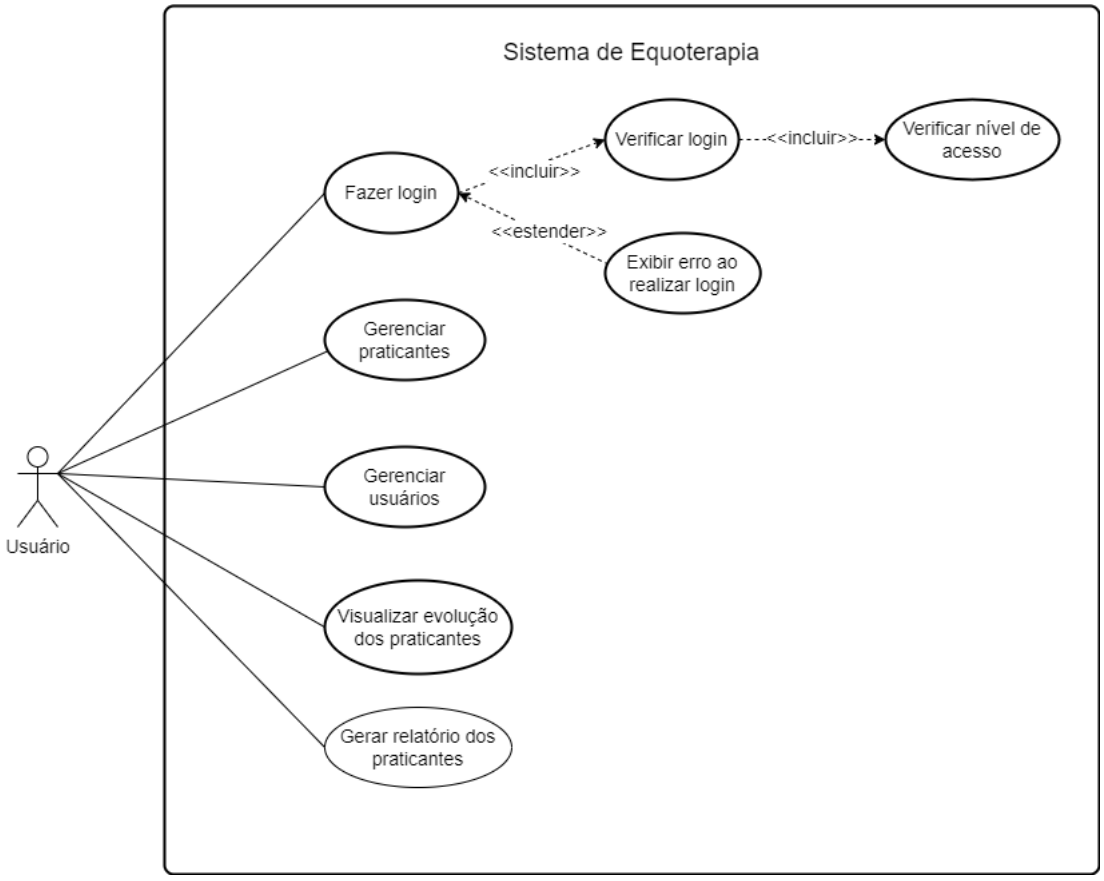


Figura 3.2: Diagrama de casos de uso.

3.1.8 Diagramas de Classes

Após a definição dos casos de uso, que fornecem uma visão de alto nível sobre as interações do usuário com o sistema, o próximo passo no processo de desenvolvimento foi a criação do Diagrama de Classes. Este é uma ferramenta essencial na análise e no design do sistema, pois descreve a estrutura estática do software, representando as classes que compõem o sistema e os relacionamentos entre elas.

No contexto do sistema de automação para a Equoterapia, esse diagrama foi desenvolvido para mapear as principais entidades do domínio do sistema, como usuários, administradores e praticantes. Essas entidades são representadas como classes, cada uma contendo atributos que descrevem suas propriedades e métodos que definem seu comportamento.

Este, por sua vez, é crucial para o projeto, pois fornece uma representação clara e detalhada da estrutura do sistema, ajudando a entender como os dados são organizados e manipulados internamente. Ele auxilia na transição dos requisitos funcionais, identificados nos casos de uso, para uma implementação de software concreta. Suas principais contribuições para o projeto foram:

Contribuições	Descrição
Identificação de Entidades e Relacionamentos	O diagrama destaca as principais entidades (ou classes) do sistema e como elas se relacionam.
Definição de Atributos e Métodos	Para cada classe, o diagrama define atributos que armazenam dados relevantes e métodos que implementam o comportamento esperado.
Relacionamentos Entre Classes	Os relacionamentos, como associações, heranças e agregações, são modelados para mostrar como as classes interagem e dependem umas das outras.

Tabela 3.2: Tabela de contribuições dos Diagramas de Classes.

Foram criados, três (3) diagramas de classes, cada um representando uma parte do sistema. Essa divisão ocorreu devido ao fato do grande tamanho do sistema, que resultaria em um enorme diagrama. Dessa forma, não seria possível exibi-lo neste documento. O primeiro diagrama, Figura 3.3, mostra a relação entre classes responsáveis por segurança e autenticação no sistema. Sendo assim, cada classe possui uma função:

Classe	**Função**
JwtToken	É responsável por gerar e decodificar tokens.
FiltroInterceptador	Realiza a filtragem de cada requisição, validando se o token é válido.
FiltroConfiguracaoWeb	Após a verificação da validade do token, essa classe é responsável por verificar quais endpoint's podem ser acessados a partir das credenciais verificadas do token.
LoginControlador	Recebe as credenciais em seu endpoint e envia para ser processada e validada internamente.
LoginServicoImplementacao	Verifica se o usuário existe a partir das credenciais obtidas. Após isso gera um token através de uma chamada a classe JwtToken. E por fim, devolve uma resposta para o cliente contendo o token gerado.
Message	Devolve mensagens personalizadas caso ocorram algum erro ou falha.
Log	Registra todas as tentativas de login no sistema em um arquivo texto. O mesmo contém informações como: data, ip, navegador, sistema operacional, nome de usuário e senha.

Tabela 3.3: Tabela de funções do Diagrama de Classes da Figura 3.3.

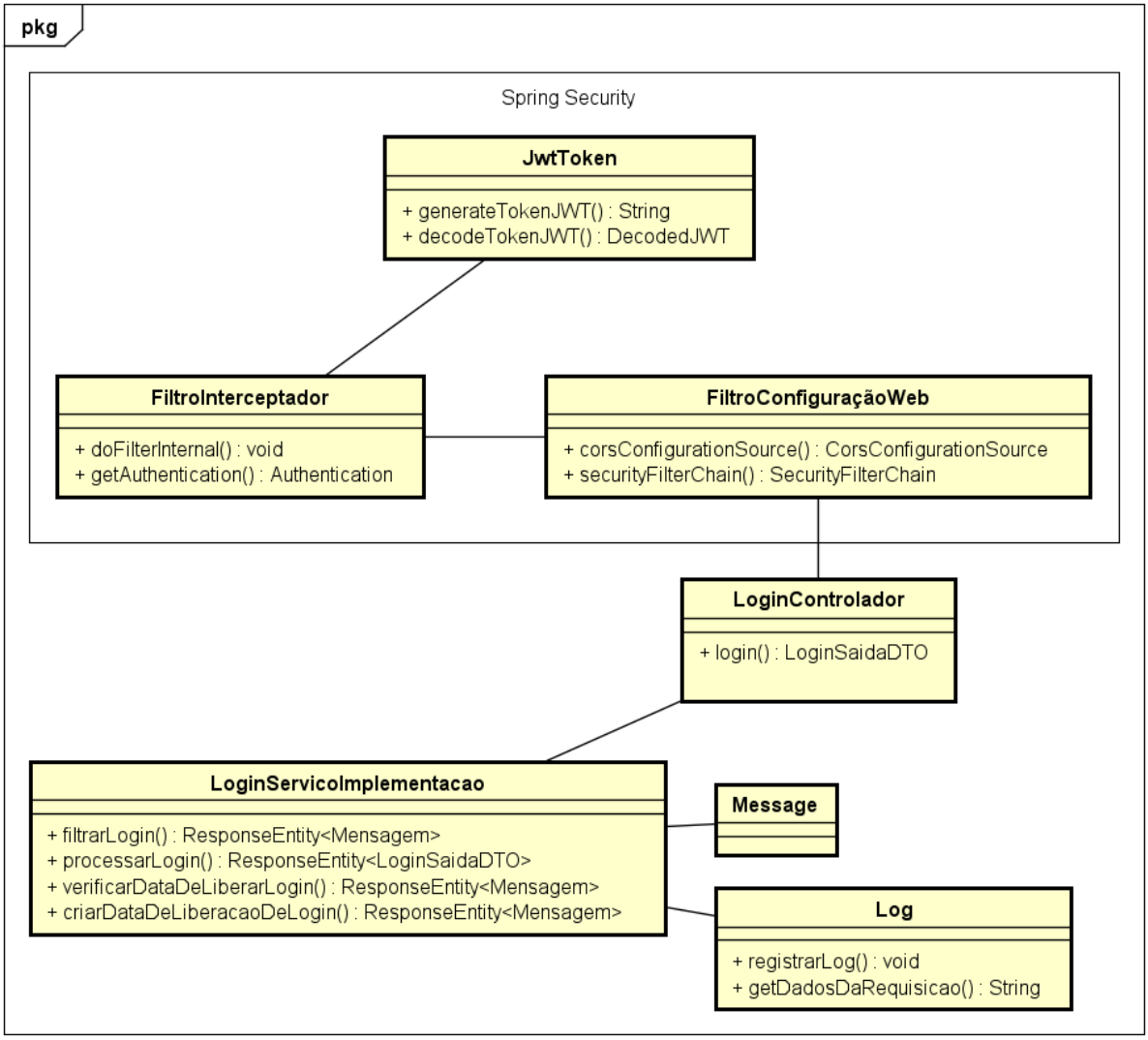


Figura 3.3: Diagrama de classes de login, segurança e autenticação.

O segundo diagrama, Figura 3.4, mostra a relação entre classes de entidades referente ao praticante. As classes foram separadas de acordo com seu tipo, sendo assim, foram divididas em quatro (4) tipos: Ficha de Cadastro Admissional, Avaliação Fisioterapêutica, Avaliação Psicológica e Plano Terapêutico Singular. Cada um desses tipos possui várias classes de entidades que representam tabelas no banco de dados. A partir disso, podemos perceber que existe uma entidade central ou seja, uma classe Praticante que faz relacionamento de composição com todas as demais entidades. Dessa forma, as classes e seus tipos ficaram organizadas da seguinte maneira:

Tipo	**Classes**
Ficha de Cadastro Admissional	CompletudeMatricula, DadosPessoais, EducacaoPraticante, Emergencia, FichaCadastroAdmissional, OutrasAtividades-Manha, OutrasAtividadesTarde, ResponsavelPraticante
Avaliação Fisioterapêutica	AvaliacaoFisioterapeutica, CoordenacaoMotora, EmPe, EquilibrioDinamico, EquilibrioEstatico, FormaDeComunicacao, GruposMusculares, HabilidadesMotorasAVD, MobilidadeArticular, QuadroAtual, SaudeGeralDoPraticante
Avaliação Psicológica	Afetividade, AvaliacaoPsicologica, Comportamento, Compreensao, CuidadosPessoais, HabilidadesSociais, Linguagem, RelacaoFamiliarExaminado, Rotina, Saude, Saude-Mental, SobreACrianca, Socializacao, TracosDePersonalidade
Plano Terapêutico Singular	PlanoTerapeuticoSingular

Tabela 3.4: Tabela de tipos e classes de entidades do praticante.

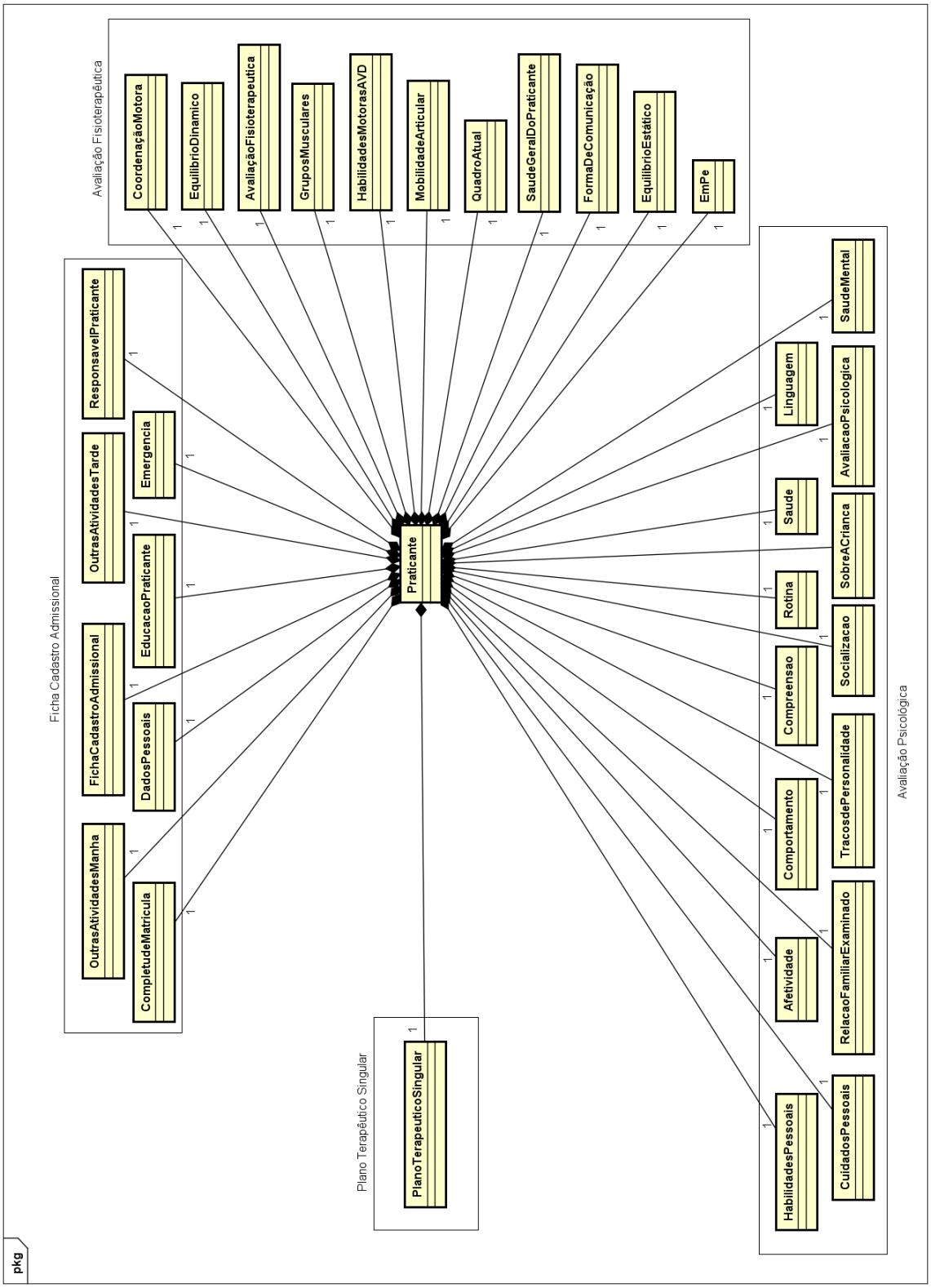


Figura 3.4: Diagrama de classes de modelo do praticante.

O terceiro diagrama, Figura 3.5, mostra como funciona o fluxo de processamento para cadastrar os dados pessoais do praticante. No entanto, essa representação se aplica igualmente para outros modelos, ou seja, para cadastro de usuários e praticantes o fluxo de processo é o mesmo. A primeira etapa consiste na interação do cliente com a página web do sistema de Equoterapia, no qual o mesmo cadastra os dados pessoais de um praticante e os enviam para serem salvos. Os dados enviados chegam em um controlador (DadosPessoaisControlador), que é responsável por atender requisições sobre os dados pessoais do praticante. Em seguida, o controlador utiliza uma classe para mapear o objeto de entrada para objeto de entidade, pois dessa forma, o objeto poderá ser repassado ao serviço (DadosPessoaisServico) para aplicar as regras de negócios e então salva-lo no banco de dados. Após os dados serem salvos, o serviço retorna os dados para o controlador, no qual o mesmo, faz o mapeamento para objeto de saída e assim, envia a resposta para o cliente.

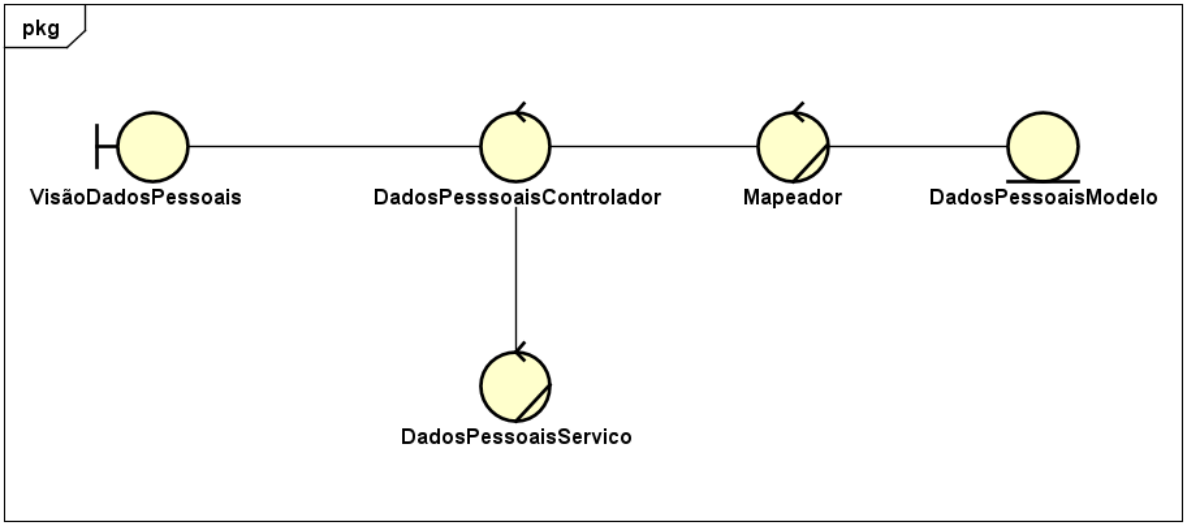


Figura 3.5: Exemplo de fluxo de processamento para cadastrar os dados pessoais do praticante.

3.1.9 Diagramas de Sequências

Após a criação do Diagrama de Casos de Uso Diagrama de Classes, a próxima etapa foi o desenvolvimento do Diagrama de Sequências. Este diagrama é uma ferramenta essencial para capturar a dinâmica das interações entre os diversos componentes do sistema ao longo do tempo. O Diagrama de Sequências é um tipo de diagrama UML (Unified Modeling Language) que descreve como os objetos no sistema interagem em um processo específico, apresentando a ordem das mensagens trocadas entre eles para realizar uma determinada funcionalidade. No contexto do sistema para Equoterapia, este diagrama desempenha um papel crucial na visualização de cenários específicos de uso, permitindo uma compreensão detalhada de como as operações ocorrem sequencialmente. O Diagrama de Sequências é particularmente útil para desenvolver um entendimento mais profundo sobre as interações entre os atores e o sistema, assim como entre os componentes internos do sistema. Algumas das suas principais contribuições incluem:

Contribuições	Descrição
Detalhamento de Interações	Ele ilustra claramente como os diferentes elementos (atores, classes e componentes) se comunicam ao longo do tempo para executar uma tarefa. Isso é fundamental para compreender a lógica de processos complexos.
Visualização do Fluxo de Mensagens	Cada mensagem trocada entre os elementos é representada de forma sequencial, facilitando a identificação de problemas potenciais na ordem ou timing das operações.
Modelagem de Cenários Específicos	O diagrama ajuda a modelar cenários específicos que foram identificados no Diagrama de Casos de Uso, permitindo uma análise detalhada de casos complexos, como o agendamento de sessões ou a geração de relatórios.
Apoio ao Desenvolvimento e Testes	Ao detalhar as interações do sistema, o diagrama oferece suporte valioso para o desenvolvimento de software e para o design de testes, garantindo que todas as etapas críticas dos processos sejam cobertas.

Tabela 3.5: Tabela de contribuições dos Diagramas de Sequências.

Foram criados, três (3) diagramas de sequências, cada um representando uma parte do sistema. Essa divisão ocorreu devido ao fato do grande tamanho do sistema, que resultaria em um enorme diagrama. Dessa forma, não seria possível exibi-lo neste documento. O primeiro diagrama, Figura 3.6, mostra a relação entre classes responsáveis por realizar o login do usuário no sistema. Sendo assim, cada classe possui uma função:

Classe	**Função**
LoginControlador	Recebe as credenciais em seu endpoint e envia para ser processada e validada internamente.
LoginServicoImplementacao	Registra a tentativa de login em um arquivo texto e chama o serviço de usuário para fazer a validação.
UsuarioServicoImplementacao	Realiza as consultas e regras de negócios referente ao usuário com o banco de dados e por fim gera um token relativo ao seu nível de acesso (Usuário ou Administrador). Após isso retorna a resposta para o serviço de login, que retorna para o controlador de login, que retorna a resposta para o cliente.

Tabela 3.6: Tabela de funções do Diagrama de Sequências relativa ao login no sistema.

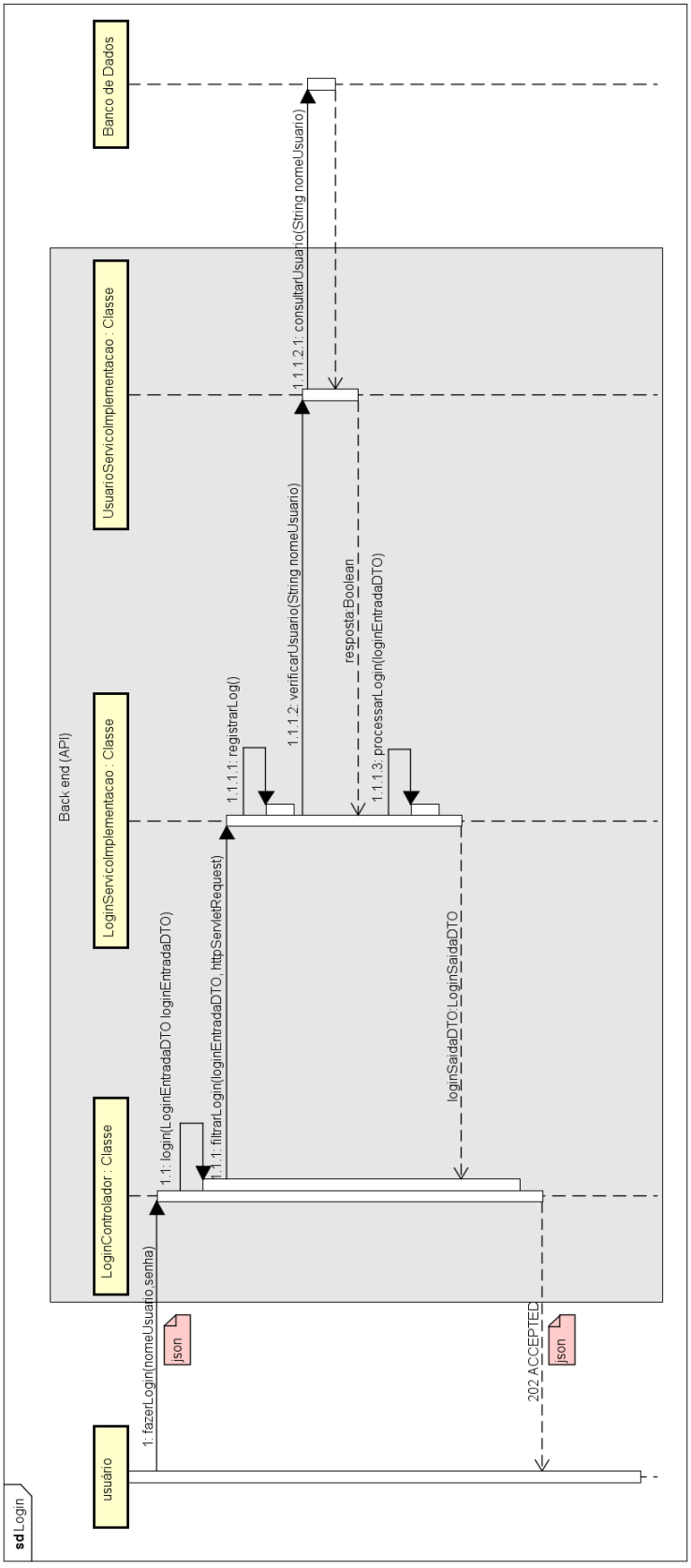


Figura 3.6: Diagrama de sequência de login no sistema.

O segundo diagrama, Figura 3.7, mostra como funciona o fluxo de processamento para cadastrar um praticante. No entanto, essa representação se aplica igualmente para outros modelos, ou seja, para cadastro de usuários e praticantes o fluxo de processo é o mesmo. A primeira etapa consiste na interação do cliente com a página web do sistema de Equoterapia, no qual o mesmo cadastra os dados do praticante e os envia para serem salvos. Os dados enviados chegam em um controlador (PraticanteControlador), que é responsável por atender as requisições sobre o praticante (Pode ser qualquer dado do praticante. Exemplo: Avaliação Psicológica, Fisioterapêutica, etc.). Em seguida, o controlador repassa os dados para o serviço (PraticanteServicoImplementacao) que faz a validação dos dados do praticante e então salva-os no banco de dados. Após isso, retorna os dados salvos para o controlador, que envia uma resposta ao cliente com os dados em JSON e status 201 que indica o sucesso em criar um registro. O mesmo fluxo de processo ocorre com as outras operações, como atualizar e consultar.

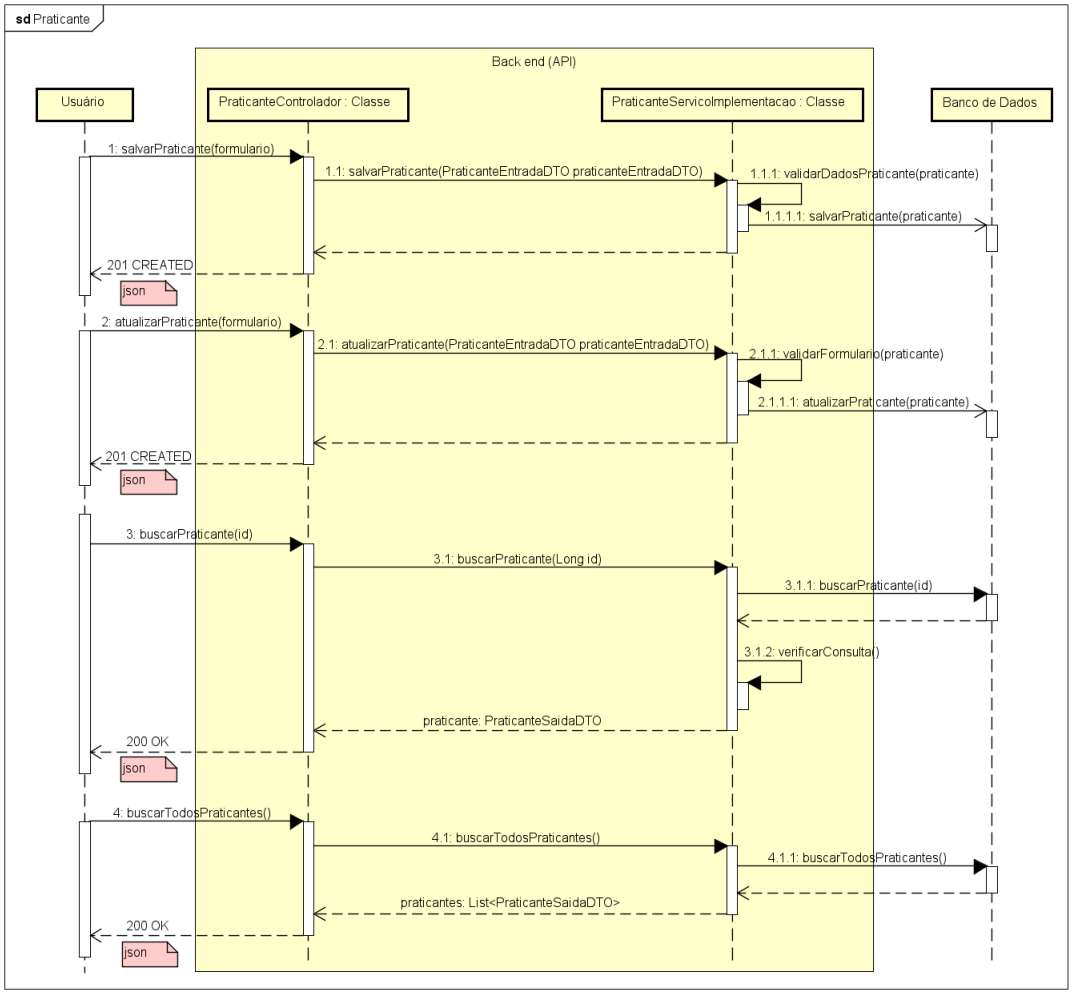


Figura 3.7: Diagrama de sequência de gerenciamento dos praticantes no sistema.

O terceiro diagrama, Figura 3.8, mostra como funciona o fluxo de processamento para cadastrar um usuário. A primeira etapa consiste na interação do cliente com a página web do sistema de Equoterapia, no qual o mesmo cadastra os dados do usuário e os envia para serem salvos. Os dados enviados chegam em um controlador (UsuarioControlador), que é responsável por atender as requisições sobre o usuário. Em seguida, o controlador repassa os dados para o serviço (UsuarioServicoImplementacao) que faz a validação dos dados do usuário e então salva-os no banco de dados. Após isso, retorna os dados salvos para o controlador, que envia uma resposta ao cliente com os dados em JSON e status 201 que indica o sucesso em criar um registro. O mesmo fluxo de processo ocorre com as outras operações, como atualizar, deletar e consultar.

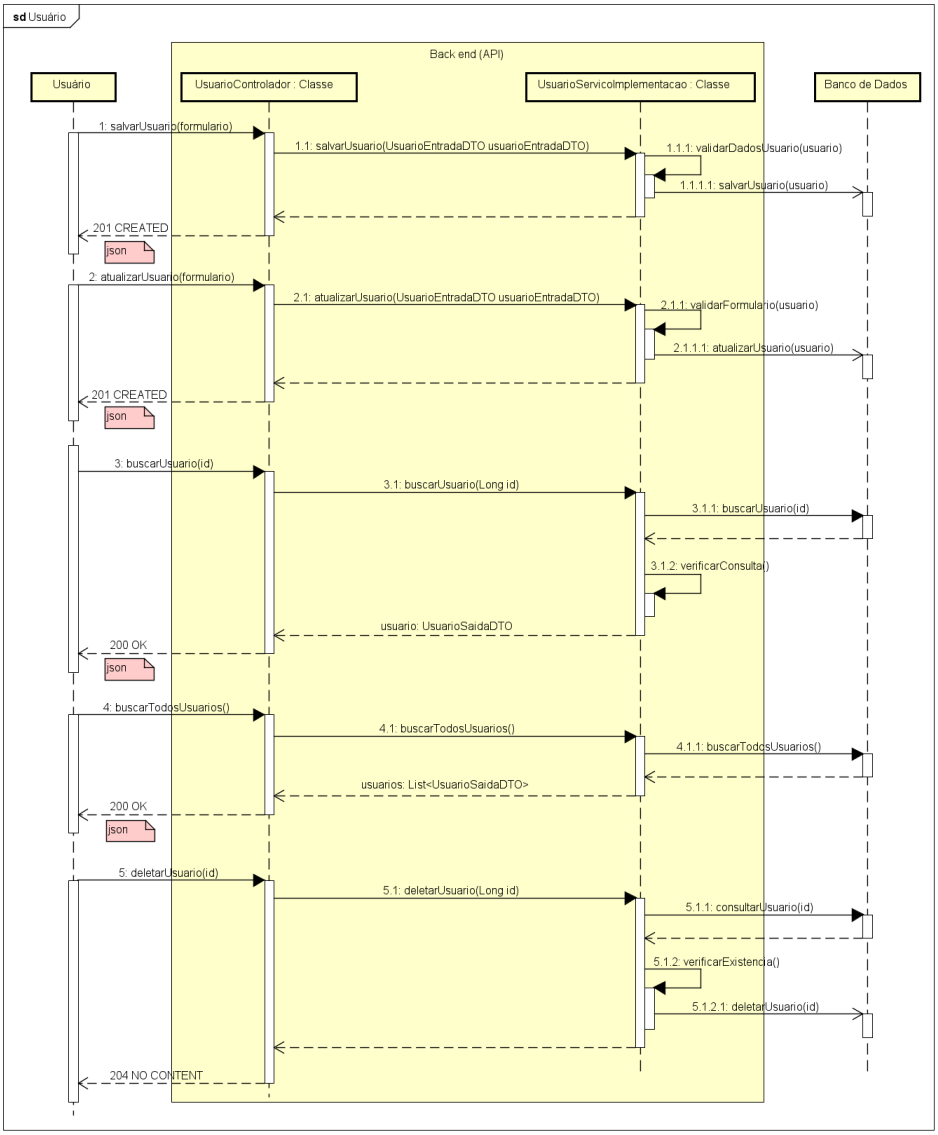


Figura 3.8: Diagrama de sequência de gerenciamento dos usuários no sistema.

3.2 Definição de Tecnologias e Ferramentas

Antes de iniciar a construção do sistema, foi preciso definir quais ferramentas, linguagens de programação, frameworks, banco de dados e IDE seriam utilizados. Sendo assim, a decisão foi a seguinte:

Tecnologia	Aspecto	Descrição
Java	Linguagem	Back end (API REST)
Javascript	Linguagem	Front end
Spring	Framework	Framework Java
React	Framework	Framework Javascript
MySQL	SGBD	Banco de dados relacional
IntelliJ Idea	IDE	Desenvolvimento do Back end
WebStorm	IDE	Desenvolvimento do Front end
MySQL Workbench	IDE	Consulta aos dados
Astah	Modelagem Visual	Criação dos diagramas UML do sistema
GIT	Versionamento de Código	Versionamento de código local da aplicação
GITHUB	Versionamento de Código	Armazenamento em nuvem das modificações do sistema

Tabela 3.7: Tabela consolidada de tecnologias, aspectos e descrições

3.3 Por que Spring?

- **Injeção de Dependências e Inversão de Controle (IoC)**: O Spring facilita a implementação de padrões de projeto como a Inversão de Controle (IoC) através da Injeção de Dependências, permitindo que os desenvolvedores criem aplicações mais modulares, flexíveis e fáceis de testar (WALLS, 2018).

- **Modularidade e Flexibilidade**: Spring é altamente modular, permitindo que os desenvolvedores escolham apenas os componentes necessários para seu projeto. Isso resulta em aplicações leves e bem organizadas, em contraste com frameworks monolíticos que impõem um conjunto fixo de ferramentas (BROWN, 2019).

- **Integração com Outras Tecnologias**: Spring oferece suporte robusto para integração com outras tecnologias e frameworks, como Hibernate para persistência de dados, JMS para mensageria, e muitos outros. Isso torna o Spring uma escolha ideal para projetos que exigem interoperabilidade com diversas tecnologias (WHITE, 2020).

- **Desenvolvimento de Aplicações Web**: Com o Spring MVC, o framework facilita o desenvolvimento de aplicações web, fornecendo uma arquitetura limpa e extensível que

segue o padrão MVC (Model-View-Controller). Isso permite a criação de aplicações web escaláveis e mantíveis (SMITH, 2019b).

- **Comunidade Ativa e Suporte Empresarial**: Spring possui uma comunidade global ativa e é amplamente adotado na indústria, o que significa que há uma vasta quantidade de recursos, bibliotecas e extensões disponíveis. Além disso, o suporte empresarial fornecido pela Pivotal/VMware torna o Spring uma opção confiável para projetos corporativos de grande escala (GARCIA, 2021).

- **Desempenho e Escalabilidade**: Spring é projetado para ser eficiente em termos de recursos e escalável, o que é crucial para aplicações corporativas que precisam lidar com grandes volumes de transações e dados (JOHNSON, 2021a).

- **Suporte a Microsserviços**: Com o Spring Boot, o Spring facilita o desenvolvimento de microsserviços, permitindo a criação de aplicações distribuídas e de alta disponibilidade. O Spring Cloud complementa essa abordagem com ferramentas que simplificam a implementação de padrões como configuração distribuída, balanceamento de carga, e descoberta de serviços (LEE, 2020).

- **Segurança**: O Spring Security é um dos módulos mais robustos para gerenciar autenticação e autorização em aplicações Java, proporcionando uma camada adicional de segurança que pode ser facilmente integrada às aplicações Spring (HARRIS, 2020).

3.4 Por que React?

- **Componentização**: React permite a criação de componentes reutilizáveis e isolados, que podem ser combinados para construir interfaces de usuário complexas. Essa abordagem modular facilita a manutenção e a escalabilidade do código (DOE, 2020).

- **Virtual DOM**: O uso do Virtual DOM pelo React otimiza o desempenho das aplicações ao minimizar as operações de manipulação direta no DOM real. Isso resulta em atualizações de interface mais rápidas e eficientes (SMITH, 2019a).

- **Comunidade e Ecossistema**: React é mantido pelo Facebook e tem uma comunidade de desenvolvedores muito ativa, o que resulta em um vasto ecossistema de bibliotecas, ferramentas e recursos. Isso facilita a integração com outras tecnologias e acelera o desenvolvimento (JOHNSON, 2021b).

- **Unidirectional Data Flow**: React adota um fluxo de dados unidirecional, o que torna a lógica de aplicação mais previsível e mais fácil de depurar. Isso contrasta com frameworks que utilizam *two-way data binding*, onde o gerenciamento de estado pode ser mais complexo (BROWN, 2021).

- **JSX**: React utiliza JSX, uma extensão de sintaxe para JavaScript que permite escrever elementos de interface de forma declarativa dentro do código JavaScript. Isso melhora a legibilidade e mantém a lógica e o layout da interface próximos (WHITE, 2018).

- **Desenvolvimento de SPA (Single Page Applications)**: React é uma excelente escolha para o desenvolvimento de *Single Page Applications* (SPAs), permitindo a criação de interfaces de usuário rápidas e dinâmicas que oferecem uma experiência similar à de aplicações desktop (MARTINEZ, 2020).

- **Atualizações e Suporte Contínuo**: Sendo mantido por uma grande empresa como o Facebook, o React recebe atualizações regulares e suporte contínuo, garantindo que a tecnologia evolua de acordo com as melhores práticas e demandas do mercado (KIM, 2022).

- **Popularidade e Mercado de Trabalho**: React é uma das bibliotecas mais populares no desenvolvimento front-end, e sua adoção generalizada garante que os desenvolvedores tenham acesso a uma ampla variedade de oportunidades de trabalho e projetos na indústria (THOMPSON, 2021).

CAPÍTULO 4

IMPLEMENTAÇÃO DO BACK-END (API RESTFUL)

Após o levantamento e análise dos requisitos, foi inicializado o desenvolvimento do sistema, começando pela implementação do back-end, sendo feito sua construção na IDE IntelliJ Idea, utilizando a linguagem Java e o Framework Spring.

4.1 Configuração Inicial

Inicialmente, um novo projeto Spring foi configurado. Foram adicionadas as dependências necessárias do ecossistema Spring: Spring Boot, Spring Web, Spring DevTools, Spring Data JPA, Spring Security, Spring Validation e Spring Doc. Por fim, algumas outras dependências necessárias, como, MySQL, Lombok, Java JWT, User Agent Utils e Model Mapper. A Tabela 4.1 descreve cada projeto do ecossistema Spring utilizado no desenvolvimento do sistema. Ademais, a Tabela 4.2 descreve outras bibliotecas utilizadas; porém, estas não pertencem ao Spring, mas sim a terceiros.

Framework Spring	Descrição
Spring Boot	Facilita o desenvolvimento de aplicativos Java, automatizando configurações e oferecendo um ambiente de execução pronto para produção.
Spring Web	Fornece suporte para construção de aplicativos web, incluindo desenvolvimento de APIs RESTful usando anotações como @RestController e @RequestMapping.
Spring DevTools	Fornece ferramentas de desenvolvimento, como reinicialização automática do servidor para melhorar a produtividade durante o desenvolvimento.
Spring Data JPA	Facilita o acesso e manipulação de dados em bancos de dados relacionais usando a abordagem de persistência de dados baseada em JPA (Java Persistence API).
Spring Security	Oferece recursos de segurança robustos para proteger aplicativos Spring, incluindo autenticação, autorização e prevenção contra ataques comuns.
Spring Validation	Fornece suporte para validação de dados, permitindo a aplicação de regras de validação em objetos de domínio usando anotações como @Valid e @NotBlank.
Spring Doc	Facilita a geração de documentação automática para APIs RESTful, tornando mais fácil para desenvolvedores e consumidores entenderem como interagir com os endpoints da API.

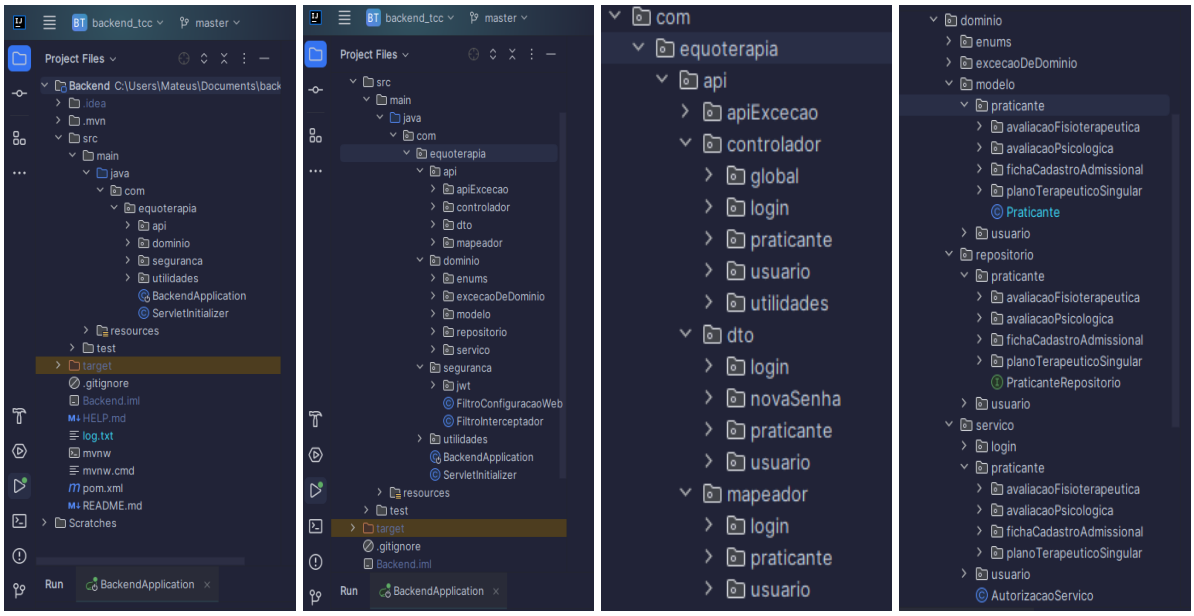
Tabela 4.1: Tabela de Frameworks do Ecossistema Spring e suas descrições

Outras Dependências	Descrição
MySQL	Driver JDBC para MySQL que permite a conexão e interação com bancos de dados MySQL.
Lombok	Biblioteca que simplifica a escrita de código Java, eliminando a necessidade de escrever getters, setters e outros métodos repetitivos através de anotações.
Java JWT	Biblioteca Java para trabalhar com JSON Web Tokens (JWT), usada para criar, verificar e decodificar tokens JWT.
User Agent Utils	Biblioteca Java que fornece utilitários para análise de agentes de usuário (user agents) de navegadores web.
Model Mapper	Biblioteca para mapeamento de objetos em Java, simplificando a transferência de dados entre diferentes modelos de objetos.

Tabela 4.2: Tabela de outras dependências necessárias

4.2 Estrutura de Pacotes do Back-end

A Figura 4.1 mostra a estrutura de pacotes do back-end em que se observa a estutura de pacotes geral e outras mais específicas. A figura (a), mostra os pacotes API, Domínio, Segurança e Utilidades. A figura (b), mostra os subpacotes de cada um desses principais. A figura (c), detalha sobre o pacote API e seus subpacotes. Por fim, a figura (d), detalha o pacote Domínio e seus subpacotes.



(a) Estrutura geral 1. **(b)** Estrutura geral 2. **(c)** Estrutura da API. **(d)** Estrutura do domínio.

Figura 4.1: Estrutura de pacotes do Back-end

4.2.1 API

O pacote *API* contém as definições e implementações das interfaces de API que são expostas pela aplicação. É aqui que as funcionalidades do sistema são disponibilizadas para os clientes através de controladores que respondem a requisições HTTP. Esse pacote inclui:

⇒ **Controladores:** Possui classes responsáveis por receber as requisições HTTP (GET, POST, PUT, DELETE) e encaminhá-las para a lógica de negócios apropriada.

⇒ **Exceção:** Possui classes responsáveis por gerar exceções ocorridas nos controladores.

⇒ **DTO:** Possui classes responsáveis por carregar dados de entrada e saída do sistema.

4.2.2 Domínio

O pacote Domínio inclui as classes de domínio que representam as entidades principais da aplicação. Estas classes geralmente são mapeadas para tabelas de banco de dados e contêm as regras de validação e a lógica de negócio intrínseca às entidades. As principais características do pacote Domínio são:

⇒ **Exceção:** Possui classes responsáveis por geração exceção interna do sistema, ou seja, exceções de regras de negócios.

⇒ **Modelo:** Possui classes responsáveis por fazer mapeamento de entidades para o banco de dados.

⇒ **Repositório:** Possui interfaces responsáveis para acessar ao banco de dados.

⇒ **Serviço:** Possui classes responsáveis por processar toda regra de negócio.

⇒ **Enum:** Possui enums para garantir tipos exatos esperados para serem salvos no banco de dados.

4.2.3 Segurança

O pacote Segurança inclui configurações e classes relacionadas à segurança da aplicação, como autenticação e autorização. A segurança é um aspecto crítico em qualquer sistema, especialmente em aplicações que lidam com dados sensíveis. Este pacote contém classes de filtragem de requisição e geração de token's.

4.2.4 Utilidades

O pacote Utilidades armazena classes utilitárias que fornecem funcionalidades auxiliares usadas em diversas partes do projeto. As classes neste pacote são projetadas para realizar tarefas que são comuns em diferentes partes do sistema, como, validar senhas, realizar log's, formatar datas e gerar mensagens prontas para respostas.

4.3 Definição das Entidades

Nessa etapa foi realizado a implementação das entidades, que representam as estruturas de dados fundamentais no domínio do sistema, mapeando classes Java para tabelas do banco de dados através de anotações do JPA. Cada entidade é responsável por modelar um conjunto específico de dados e suas relações com outras entidades, garantindo que a persistência e recuperação de informações sejam consistentes e eficientes.

⇒ **Representação de Tabelas:** Cada entidade corresponde a uma tabela no banco de dados, com atributos que refletem as colunas dessa tabela.

⇒ **Modelagem de Relacionamentos:** As entidades podem definir relações como one-to-one, one-to-many, ou many-to-many, utilizando anotações como @OneToOne, @OneToMany, e @ManyToMany.

⇒ **Validações de Dados:** Anotações de validação, como @NotNull e @Size, são utilizadas para garantir que os dados armazenados estejam de acordo com as regras de negócio.

4.4 Criação dos Controladores

Nessa etapa foi realizado a implementação dos controladores REST, que são responsáveis por expor os endpoints da API, permitindo que o sistema interaja com clientes externos por meio de requisições HTTP. Eles atuam como uma camada de mediação entre a interface do usuário e a lógica de negócios, assegurando que as solicitações sejam processadas corretamente e retornem as respostas esperadas.

⇒ **Mapeamento de Endpoints:** Os controladores mapeiam requisições HTTP para métodos específicos, usando anotações como @GetMapping, @PostMapping, etc.

⇒ **Processamento de Requisições:** Recebem e processam dados de entrada, delegando a execução de lógica de negócios para serviços especializados.

⇒ **Geração de Respostas HTTP:** Retornam dados para o cliente em formatos como JSON, incluindo códigos de status HTTP adequados.

⇒ **Validação de Entrada:** Realizam validações iniciais dos dados recebidos, garantindo que informações inválidas não sejam processadas.

4.5 Criação dos DTO's

Nessa etapa foi realizado a implementação dos DTO's, que são utilizados para transferir dados entre as diferentes camadas da aplicação, facilitando a comunicação e promovendo a separação entre a lógica de apresentação e a lógica de persistência. Os DTOs ajudam a estruturar os dados de forma eficiente, mantendo a segurança e a consistência nas interações entre cliente e servidor.

⇒ **Entrada e Saída de Dados:** Existem DTO's específicos para entrada e saída de dados, assegurando que apenas as informações necessárias sejam trafegadas.

⇒ **Abstração de Dados Sensíveis:** Os DTO's escondem detalhes internos das entidades, expondo somente os dados relevantes para cada operação.

⇒ **Validações de DTO's:** Anotações de validação podem ser aplicadas aos campos dos DTOs, garantindo que os dados estejam no formato correto antes de serem processados.

⇒ **Consistência na Transferência de Dados:** Asseguram que os dados sejam sempre trocados de maneira uniforme, facilitando a integração com outras partes do sistema ou aplicações externas.

4.6 Criação dos Mapeadores

Nessa etapa foi realizado a implementação dos mapeadores, que são responsáveis por converter entre entidades do domínio e DTO's, permitindo que os dados sejam transformados conforme necessário entre a camada de persistência e a camada de apresentação. Essa conversão é essencial para preservar a integridade e a consistência dos dados ao longo do fluxo de informação.

⇒ **Conversão Automática:** Utilização de bibliotecas como ModelMapper para facilitar a conversão automática entre objetos de entidades e DTO's.

⇒ **Mapeamento Personalizado:** Permite definir regras de mapeamento customizadas, como ignorar certos campos ou aplicar transformações específicas durante a conversão.

⇒ **Isolamento da Lógica de Conversão:** Centraliza a lógica de transformação em classes dedicadas, facilitando manutenção e atualização de mapeamentos complexos.

⇒ **Eficiência e Desempenho:** Otimiza o processo de conversão, minimizando a sobrecarga e melhorando o desempenho da aplicação durante a troca de dados.

4.7 Criação dos Repositórios

Nessa etapa foi realizado a implementação dos repositórios, que fornecem uma interface de acesso aos dados armazenados no banco de dados, estendendo as funcionalidades do Spring Data JPA para operações de persistência. Eles desempenham um papel crucial na camada de persistência, abstraindo a complexidade do acesso direto ao banco de dados e permitindo que a lógica de negócios manipule os dados de forma intuitiva.

⇒ **Abstração de Persistência:** Os repositórios abstraem a lógica de acesso ao banco de dados, permitindo interações mais simples e intuitivas com os dados.

⇒ **Operações CRUD Simplificadas:** Facilita operações de criação, leitura, atualização e exclusão (CRUD) sem a necessidade de código SQL explícito.

⇒ **Consultas Personalizadas:** Permite definir consultas específicas através de métodos personalizados ou pela anotação @Query para cenários complexos.

⇒ **Uso de Convenções:** Aproveita as convenções de nomenclatura do Spring Data JPA para gerar automaticamente consultas baseadas em nomes de métodos, reduzindo o tempo de desenvolvimento.

4.8 Implementação dos Serviços

Nessa etapa foi realizado a implementação dos serviços, que encapsulam a lógica de negócios da aplicação, atuando como intermediários entre os controladores e os repositórios. Eles garantem que a lógica complexa seja executada de forma centralizada e consistente.

⇒ **Centralização da Lógica de Negócios:** Agrupam e executam as regras de negócio, garantindo consistência e aderência às políticas da aplicação.

⇒ **Orquestração de Processos:** Coordenam interações entre diferentes componentes, como chamadas a repositórios, API's externas ou outros serviços, para atingir o resultado desejado.

⇒ **Isolamento de Funções:** Permite que a lógica de negócios seja desenvolvida e testada separadamente, facilitando a manutenção e o teste unitário de funcionalidades.

⇒ **Tratamento de Exceções:** Implementa estratégias robustas para capturar e tratar exceções, garantindo que erros sejam manuseados adequadamente sem interromper o fluxo da aplicação.

4.9 Implementação de Exceções

Nessa etapa foi realizado a implementação das exceções, que é um componente crítico para a robustez e a estabilidade do sistema, permitindo que o back-end lide com erros de maneira controlada e eficaz. A implementação de um gerenciamento abrangente de exceções assegura que o sistema possa continuar operando mesmo quando ocorrem falhas inesperadas.

⇒ **Definição de Exceções Personalizadas:** Criação de exceções específicas para representar erros comuns ou condições excepcionais no sistema, facilitando o diagnóstico e a correção de problemas.

⇒ **Manipuladores de Exceções:** Uso de @ControllerAdvice e @ExceptionHandler para interceptar e tratar exceções, convertendo-as em respostas HTTP apropriadas.

⇒ **Registro de Erros:** Mantém logs detalhados de exceções, permitindo auditoria e análise para identificar padrões de falhas e melhorar a resiliência do sistema.

⇒ **Continuidade Operacional:** Assegura que exceções não interrompam o fluxo da aplicação, aplicando padrões de fallback ou recuperação automática onde aplicável.

4.10 Implementação de Segurança

Nessa etapa foi realizado a implementação da segurança, que é um aspecto essencial em qualquer sistema que lide com informações sensíveis ou dados de usuários. A implementação de um conjunto robusto de medidas de segurança protege o sistema contra acessos não autorizados e ameaças comuns, garantindo a integridade e a confidencialidade dos dados.

⇒ **Autenticação e Autorização:** Gerenciar autenticação de usuários e autorização de acesso a endpoints, assegurando que apenas usuários verificados possam realizar ações específicas.

⇒ **Uso de Tokens JWT:** Implementar para autenticação, permitindo que tokens seguros sejam utilizados para verificar a identidade do usuário em cada requisição.

⇒ **Proteção Contra Ameaças Comuns:** Incorporar defesas contra vulnerabilidades como CSRF (*Cross-Site Request Forgery* ou Falsificação de solicitação entre sites) , XSS (*Cross-Site Scripting* ou Script entre sites) , e injeção de SQL, assegurando que o sistema resista a ataques maliciosos.

⇒ **Configuração de Políticas de Acesso:** Definir políticas detalhadas que determinam quais usuários ou grupos têm permissão para acessar recursos específicos, garantindo que a segurança seja mantida em todos os níveis do sistema.

4.11 Implementação de Login

Nessa etapa foi realizado a implementação do login, que é crítica para a experiência do usuário, proporcionando uma forma segura de autenticação e acesso ao sistema. O Spring Security gerencia o processo de login, assegurando que apenas usuários válidos possam obter acesso. As características da implementação de login incluem:

⇒ **Validação de Credenciais:** Verificar a autenticidade das credenciais fornecidas pelos usuários, garantindo que nomes de usuários e senhas estejam corretos antes de conceder acesso.

⇒ **Geração de Tokens JWT:** Após a validação bem-sucedida, um token JWT é gerado e retornado ao usuário, que deve ser incluído em requisições subsequentes para acessar recursos protegidos.

⇒ **Bloqueio de Conta:** Implementar medidas de segurança, como bloqueio temporário de contas após várias tentativas falhas de login, aumentando a proteção contra ataques de força bruta.

⇒ **Registro de Tentativas de Login:** Manter um log detalhado de todas as tentativas de login, registrando tanto os sucessos quanto as falhas, para monitoramento de segurança e auditoria.

4.12 Trechos de Código

Para toda requisições, o fluxo de processamento dos dados é o mesmo. Sendo assim, os códigos abaixo referente ao cadastro dos dados pessoais do praticante podem exemplificar

bem como ocorre o fluxo de processamento de dados de todo restante dos cadastros, consultas, deletes e atualizações.

4.12.1 Login

A Figura 4.1, mostra o endpoint de login que recebe um JSON e o converte para um objeto do tipo LoginEntradaDTO. Após isso, repassa esse objeto para o serviço de login, que irá processar as credenciais.

```
@RestController
@RequestMapping("/login")
public class LoginControlador {

    @Autowired
    private LoginServico servico;

    @PostMapping("/logar")
    public ResponseEntity<?> login(@RequestBody @Valid
       LoginEntradaDTO loginEntradaDTO, HttpServletRequest request) {
        return servico.filtrarLogin(loginEntradaDTO, request);
    }

}
```

Listing 4.1: Endpoint de Login da API

A Figura 4.2, mostra o DTO de entrada referente a requisição de login. Os atributos possuem anotações para validações de entrada.

```
@Getter
@Setter
@ToString
@NoArgsConstructor
@AllArgsConstructor
public class LoginEntradaDTO {

    @NotBlank(message = Resposta.NOME_USUARIO_LOGIN)
    private String nomeUsuario;

    @NotBlank(message = Resposta.SENHA_LOGIN)
    @Size(min = 6, message = Resposta.SENHA)
    private String senha;
}
```

Listing 4.2: DTO de entrada para login

A Figura 4.3, mostra o DTO de saída referente a resposta de login. Esses dados serão fundamental para o cliente utilizar em suas próximas requisições ao sistema.

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class LoginSaidaDTO {

    @NotNull
    private Long idUsuario;

    @NotBlank
    private String nomeUsuario;

    @NotBlank
    private String token;

    private LocalDateTime validadeToken = LocalDateTime.now().plus
       (60, ChronoUnit.MINUTES);

    @NotNull
    private RoleEnum role;

}
```

Listing 4.3: DTO de saída para login

4.12.2 Praticante

A Figura 4.4, mostra o controlador responsável por gerenciar as requisições e respostas referente aos dados pessoais do praticante. Além disso, destaca o endpoint para salvar os dados pessoais de um novo praticante.

```
@RestController
@RequestMapping("/praticante/dados-pessoais")
public class DadosPessoaisControlador {

    @Autowired
    private DadosPessoaisServico dadosPessoaisServico;

    @PostMapping("/salvar-dados-pessoais-do-praticante")
    public ResponseEntity<?> salvarDadosPessoais(@RequestBody @Valid
       DadosPessoaisEntradaDTO dadosPessoaisEntradaDTO){
        DadosPessoais dadosPessoais = PraticanteMapeador.
           converterDadosPessoaisEntradaDTOParaDadosPessoais(
           dadosPessoaisEntradaDTO);
        DadosPessoais dadosPessoaisSalvo = dadosPessoaisServico.
           salvarDadosPessoais(dadosPessoais);
        DadosPessoaisSaidaDTO dadosPessoaisSaidaDTO =
           PraticanteMapeador.
           converterDadosPessoaisParaDadosPessoaisSaidaDTO(
           dadosPessoaisSalvo);
        return new ResponseEntity<DadosPessoaisSaidaDTO>(
           dadosPessoaisSaidaDTO, HttpStatus.CREATED);
    }

    // Outros endpoints....

}
```

Listing 4.4: Controlador de Dados Pessoais

A Figura 4.5, mostra a classe responsável por fazer conversões de DTO's de entrada e saída do sistema referente ao praticante.

```
1  public class PraticanteMapeador {
2
3      public static DadosPessoais
           converterDadosPessoaisEntradaDTOParaDadosPessoais(
           DadosPessoaisEntradaDTO dadosPessoaisEntradaDTO) {
4          ModelMapper modelMapper = new ModelMapper();
5          return modelMapper.map(dadosPessoaisEntradaDTO, DadosPessoais
               .class);
6      }
7
8      public static DadosPessoais
           converterDadosPessoaisAtualizacaoEntradaDTOParaDadosPessoais(
           DadosPessoaisAtualizacaoEntradaDTO
           dadosPessoaisAtualizacaoEntradaDTO) {
9          ModelMapper modelMapper = new ModelMapper();
10         return modelMapper.map(dadosPessoaisAtualizacaoEntradaDTO,
               DadosPessoais.class);
11     }
12
13     public static DadosPessoaisSaidaDTO
           converterDadosPessoaisParaDadosPessoaisSaidaDTO(DadosPessoais
           dadosPessoais) {
14         ModelMapper modelMapper = new ModelMapper();
15         return modelMapper.map(dadosPessoais, DadosPessoaisSaidaDTO.
               class);
16     }
17
18     public static List<DadosPessoaisSaidaDTO>
           converterListaDeDadosPessoaisParaListaDeDadosPessoaisSaidaDTO(
           List<DadosPessoais> listaDadosPessoais) {
19         ModelMapper modelMapper = new ModelMapper();
20         List<DadosPessoaisSaidaDTO> listaSaidaDTO = new ArrayList<>()
               ;
21         for (DadosPessoais dadosPessoais : listaDadosPessoais) {
22             DadosPessoaisSaidaDTO outputDTO = modelMapper.map(
                   dadosPessoais, DadosPessoaisSaidaDTO.class);
23             listaSaidaDTO.add(outputDTO);
24         }
25         return listaSaidaDTO;
26     }
27
28     // Outros metodos mapeadores....
29
30 }
```

Listing 4.5: Classe mapeadora para dados pessoais do praticante

A Figura 4.6, mostra a classe de modelo referente aos dados pessoais do praticante, que representa uma entidade que resulta em uma tabela no banco de dados.

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class DadosPessoais {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idDadosPessoais;
    private String nomeCompleto;
    private String diagnosticoClinico;
    private String queixaPrincipal;
    private String CID;
    private Date dataNascimento;
    private Double peso;
    private TipoSanguineoEnum tipoSanguineo;
    private String fatorRH;
    private Double altura;
    private SexoEnum sexo;
    private String naturalidade;
    private CorOuRaca corOuRaca;
    private String cpf;
    @Column(unique = true)
    private String cartaoSUS;
    private String enderecoResidencial;
    private String bairro;
    private String cidade;
    private String cep;

    private Boolean finalizado;

    @OneToOne
    @JoinColumn
    private Praticante praticante;
}
```

Listing 4.6: Modelo dos dados pessoais do praticante

A Figura 4.7, mostra um método da classe de serviço de dados pessoais do praticante, cujo objetivo é gerenciar as regras de negócio. Esse método verifica se já existe algum registro na base de dados com o mesmo CPF ou cartão do SUS. Caso um registro com o mesmo CPF ou cartão do SUS seja encontrado, não será possível salvar os novos dados e uma exceção personalizada será lançada, resultando em uma resposta adequada para o cliente que fez a requisição.

```
@Service
public class DadosPessoaisServicoImplementacao implements
    DadosPessoaisServico {

    @Autowired
    private PraticanteRepositorio praticanteRepositorio;

    @Autowired
    private DadosPessoaisRepositorio dadosPessoaisRepositorio;

    @Autowired
    private FichaCadastroAdmissionalServico
        fichaCadastroAdmissionalServico;

    @Override
    public DadosPessoais salvarDadosPessoais(DadosPessoais
        dadosPessoais) {

        if (dadosPessoais.getCartaoSUS().isEmpty()) {
            throw new ExcecaoDeRegrasDeNegocio("Informe o cart o do
                SUS!");
        }

        if (!dadosPessoais.getCpf().isEmpty() &&
            dadosPessoaisRepositorio.findByCpf(dadosPessoais.getCpf())
            .isPresent()) {
            throw new ExcecaoDeRegrasDeNegocio("O praticante com CPF
                "
                    + dadosPessoais.getCpf() + " j  est  cadastrado
                        no sistema!");
        } else if (dadosPessoaisRepositorio.findByCartaoSUS(
            dadosPessoais.getCartaoSUS()).isPresent()) {
            throw new ExcecaoDeRegrasDeNegocio("O praticante com
                cart o do SUS "
                    + dadosPessoais.getCartaoSUS()
                    + " j  est  cadastrado no sistema!");
        } else {
            Praticante praticante = new Praticante();

            praticante = praticanteRepositorio.save(praticante);
            dadosPessoais.setPraticante(praticante);
            FichaCadastroAdmissional fichaCadastroAdmissional = new
                FichaCadastroAdmissional();
            fichaCadastroAdmissional.setPraticante(praticante);
            fichaCadastroAdmissional.setDataAvaliacao(new Date());

            if (fichaCadastroAdmissionalServico.
                salvarFichaCadastroAdmissional(
                fichaCadastroAdmissional) != null) {
                return dadosPessoaisRepositorio.save(dadosPessoais);
            } else {
```

```
39                  throw new ExcecaoDeRegrasDeNegocio("Houve um erro ao
                        salvar os dados pessoais do praticante!");
40              }
41          }
42      }
43
44      // Outros metodos....
45
46  }
```

Listing 4.7: Método da classe de serviço de dados pessoais

4.12.3 Usuário

A Figura 4.8, mostra o controlador de usuário. Todas requisições referente ao mesmo, tem como destino este controlador, no qual possui os endpoints específicos para cada requisição.

```
@RestController
@RequestMapping(path = "/usuario")
public class UsuarioControlador {

    @Autowired
    private UsuarioServico usuarioServico;

    @PostMapping("/salvar-novo-usuario")
    public ResponseEntity<Mensagem> salvarNovoUsuario(@RequestBody
       @Valid UsuarioEntradaDTO usuarioEntradaDTO) {
        ValidadorDeSenha.isStrong(usuarioEntradaDTO.getSenha());
        usuarioServico.salvar(UsuarioMapeador.
           converterUsuarioEntradaDTOEmUsuario(usuarioEntradaDTO));
        return new ResponseEntity<>(new Mensagem(Resposta.
           USUARIO_CAD_OK), HttpStatus.CREATED);
    }

    @DeleteMapping("/deletar-usuario")
    public ResponseEntity<Mensagem> deletarUsuario(@RequestParam("id"
       ) Long id){
        usuarioServico.deletarUsuarioPorId(id);
        return new ResponseEntity<>(new Mensagem("Usu rio deletado
           com sucesso!"), HttpStatus.OK);
    }

    @PutMapping("/atualizar-usuario")
    public ResponseEntity<Mensagem> atualizarUsuario(@RequestBody
       @Valid UsuarioAtualizacaoEntradaDTO
       usuarioAtualizacaoEntradaDTO) {
        usuarioServico.atualizarUsuario(UsuarioMapeador.
           converterUsuarioUpdateEntradaDTOEmUsuario(
           usuarioAtualizacaoEntradaDTO));
        return new ResponseEntity<>(new Mensagem(Resposta.
           USUARIO_UP_OK), HttpStatus.CREATED);
    }

    @GetMapping("/buscar-usuario-por-nome")
    public ResponseEntity<List<UsuarioSaidaDTO>>
       pesquisarUsuarioPorNome(@RequestParam("nome") @Valid @NotBlank
        String nome) {
        List<Usuario> listUsuario = usuarioServico.
           pesquisarUsuarioPorNomeComOperadorLike(nome);
        return new ResponseEntity<>(UsuarioMapeador.
           converterListaUsuarioEmListaUsuarioSaidaDTO(listUsuario),
           HttpStatus.OK);
    }
```

```
32
33      @GetMapping("/buscar-usuario-por-id")
34      public ResponseEntity<UsuarioSaidaDTO> pesquisarUsuarioPorID(
            @RequestParam("id") Long id) {
35          Usuario usuario = usuarioServico.buscarUsuarioPorId(id);
36          return new ResponseEntity<>(UsuarioMapeador.
                converterUsuarioEmUsuarioSaidaDTO(usuario), HttpStatus.OK)
                ;
37      }
38
39  }
```

Listing 4.8: Controlador de usuário

A Figura 4.9, mostra a classe responsável por fazer conversões de DTO's de entrada e saída do sistema referente ao usuário.

```
1  public class UsuarioMapeador {
2
3      public static Usuario converterUsuarioEntradaDTOEmUsuario(
           UsuarioEntradaDTO usuarioEntradaDTO){
4          ModelMapper modelMapper = new ModelMapper();
5          return modelMapper.map(usuarioEntradaDTO, Usuario.class);
6      }
7
8      public static Usuario converterUsuarioUpdateEntradaDTOEmUsuario(
           UsuarioAtualizacaoEntradaDTO usuarioAtualizacaoEntradaDTO){
9          ModelMapper modelMapper = new ModelMapper();
10         return modelMapper.map(usuarioAtualizacaoEntradaDTO, Usuario.
               class);
11     }
12
13     public static UsuarioSaidaDTO converterUsuarioEmUsuarioSaidaDTO(
           Usuario usuario){
14         ModelMapper modelMapper = new ModelMapper();
15         return modelMapper.map(usuario, UsuarioSaidaDTO.class);
16     }
17
18     public static List<UsuarioSaidaDTO>
           converterListaUsuarioEmListaUsuarioSaidaDTO(List<Usuario>
           listaUsuario){
19         ModelMapper modelMapper = new ModelMapper();
20         List<UsuarioSaidaDTO> listaUsuarioSaidaDTO = new ArrayList
               <>();
21         for(Usuario usuario:listaUsuario){
22             UsuarioSaidaDTO usuarioSaidaDTO = modelMapper.map(usuario
                   , UsuarioSaidaDTO.class);
23             listaUsuarioSaidaDTO.add(usuarioSaidaDTO);
24         }
25         return listaUsuarioSaidaDTO;
26     }
27 }
```

Listing 4.9: Classe UsuarioMapeador

A Figura 4.10, mostra a classe de modelo do usuário, que implementa a interface *UserDetails*. Nessa classe modelo, existe o método implementado *getAuthorities()* que serve para obter o tipo de usuário atual. Além disso, contém todos atributos necessários para o usuário, que resultam em uma tabela no banco de dados.

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Entity
@EntityListeners(AuditingEntityListener.class)
public class Usuario implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idUsuario;
    private String nome;
    @Lob
    @Column(columnDefinition = "LONGTEXT")
    private String foto;
    private Date dataNascimento;
    @Column(unique = true)
    private String cpf;
    private EstadoCivilEnum estadoCivil;
    @Column(unique = true)
    private String telefone;
    @Column(unique = true)
    private String email;
    @Column(unique = true)
    private String nomeUsuario;
    private String senha;
    private String detalhesFormacao;
    private String cidade;
    private String bairro;
    private String logradouro;
    private RoleEnum role;
    private VinculoEnum vinculo;
    private SImOuNaoEnum possuiFormacao;
    private String token;
    private boolean status;
    private int tentativasLogin;
    private Date liberarLogin;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        if (this.role == RoleEnum.ROLE_ADMIN) {
            return List.of(new SimpleGrantedAuthority("ROLE_ADMIN"),
                new SimpleGrantedAuthority("ROLE_USER"));
        }
```

```
45
46          if (this.role == RoleEnum.ROLE_USER) {
47              return List.of(new SimpleGrantedAuthority("ROLE_USER"));
48          } else {
49              return null;
50          }
51      }
52
53      @Override
54      public String getPassword() {
55          return senha;
56      }
57
58      @Override
59      public String getUsername() {
60          return nomeUsuario;
61      }
62
63      @Override
64      public boolean isAccountNonExpired() {
65          return true;
66      }
67
68      @Override
69      public boolean isAccountNonLocked() {
70          return true;
71      }
72
73      @Override
74      public boolean isCredentialsNonExpired() {
75          return true;
76      }
77
78      @Override
79      public boolean isEnabled() {
80          return true;
81      }
82
83  }
```

Listing 4.10: Classe de entidade Usuario

A Figura 4.11, mostra a classe de serviço do usuário. Ela possui vários métodos e dentre eles, possui o método de salvar um novo usuário, que faz várias validações, como, verificar se já existe algum usuário cadastrado com o mesmo email, mesmo nome de usuário, mesmo telefone e mesmo CPF. Após essas validações, caso seja um usuário válido, irá ser cadastrado no sistema, do contrário, será lançada uma exceção personalizada para enviar uma resposta ao cliente, dizendo o motivo de não ser possível realizar o cadastro do mesmo.

```
@Service
public class UsuarioServicoImpl implements UsuarioServico {

    @Autowired
    private UsuarioRepositorio repository;

    private final PasswordEncoder encriptadorDeSenha = new
       BCryptPasswordEncoder();

    private final int MINUTOS_PARA_NOVA_TENTATIVA = 60;

    @Transactional(readOnly = false)
    @Override
    public Usuario salvar(Usuario usuario) {
        if (repository.findByEmail(usuario.getEmail()).isEmpty()) {
            if (repository.buscarUsuarioPorNomeDeUsuario(usuario.
               getUsername()).isEmpty()) {
                if (repository.findByTelefone(usuario.getTelefone()).
                   isEmpty()) {
                    if (repository.findByCpf(usuario.getCpf()).
                       isEmpty()) {
                        String tokenDoUsuario = JwtToken.
                           generateTokenJWT(usuario);
                        usuario.setToken(tokenDoUsuario);
                        usuario.setStatus(false);
                        usuario.setRole(usuario.getRole());
                        usuario.setSenha(encriptadorDeSenha.encode(
                           usuario.getPassword()));
                        Usuario usuarioSalvo = repository.save(
                           usuario);
                        if (usuarioSalvo.getIdUsuario() == null) {
                            throw new ExcecaoDeRegrasDeNegocio(
                               Resposta.ERRO_SALVAR_USUARIO + usuario
                               .getNome());
                        } else {
                            return usuarioSalvo;
                        }
                    } else {
                        throw new ExcecaoDeRegrasDeNegocio("J
                           existe um usu rio cadastrado com o cpf "
                           + usuario.getCpf());
                    }
```

```
32                  } else {
33                      throw new ExcecaoDeRegrasDeNegocio("J  existe um
                            usu rio cadastrado com o telefone " +
                           usuario.getTelefone());
34                  }
35              } else {
36                  throw new ExcecaoDeRegrasDeNegocio("J  existe um
                        usu rio cadastrado com o nome de usu rio " +
                        usuario.getUsername());
37              }
38          } else {
39              throw new ExcecaoDeRegrasDeNegocio("J  existe um
                    usu rio cadastrado com o email " + usuario.getEmail()
                    );
40          }
41      }
42
43      // Outros metodos....
44
45  }
```

Listing 4.11: Classe de serviço UsuarioServicoImpl

4.13 Projeto do Banco de Dados

Para criação da base de dados, foi preciso recolher todos os atributos necessários de ambas as entidades - usuários e praticantes - e assim definir o tipo de cada atributo. A geração das tabelas, colunas e relacionamentos foram feitas automaticamente pelo próprio Spring a partir das classes de entidades - pojos - criadas juntamente com as anotações específicas para classes e atributos.

Figura 4.2: Diagrama de classes do banco de dados.

CAPÍTULO 5

IMPLEMENTAÇÃO DO FRONT-END (REACT)

Após o levantamento e análise dos requisitos e implementação do back-end, foi iniciado a implementação do front-end, sendo feito sua construção na IDE WebStorm, utilizando a linguagem Javascript e a Biblioteca React. Desta forma, foi possível fazer a integração e interação com a API REST criada e assim realizar a comunicação entre cliente e servidor.

5.1 Configuração Inicial

Para a construção do projeto, foram necessárias várias dependências de grande importância, conforme mostrado nas Tabelas 5.1, 5.2 e 5.3:

Dependências	**Descrição**
react	Biblioteca principal para construção de interfaces de usuário.
react-dom	Complemento do React para manipulação da árvore de componentes e renderização no DOM.
react-router-dom	Biblioteca para gerenciamento de rotas em aplicações React.
core-js	Biblioteca que fornece funcionalidades JavaScript padrão e polifills para compatibilidade com diferentes navegadores.

Tabela 5.1: Tabela de dependências padrão do React.

Dependências	Descrição
@coreui/chartjs	Integração do CoreUI com Chart.js para criação de gráficos e visualizações de dados.
@coreui/coreui	Biblioteca de componentes UI baseada no Bootstrap, oferecendo estilos e componentes prontos para uso.
@coreui/icons	Conjunto de ícones SVG otimizados para uso com o CoreUI.
@coreui/icons-react	Implementação de ícones CoreUI para uso com React.
@coreui/react	Biblioteca de componentes React para CoreUI, facilitando a criação de interfaces de usuário.
@coreui/react-chartjs	Componentes React do CoreUI para Chart.js, permitindo a criação de gráficos facilmente em aplicações React.
@coreui/utils	Utilitários auxiliares para desenvolvimento com CoreUI.
@emotion/react	Biblioteca para estilização de componentes em React usando CSS-in-JS.
@emotion/styled	API para criar componentes estilizados com @emotion/react.
@mui/material	Biblioteca de componentes React para Material-UI, um popular framework de design.

Tabela 5.2: Tabela de dependências para estilização.

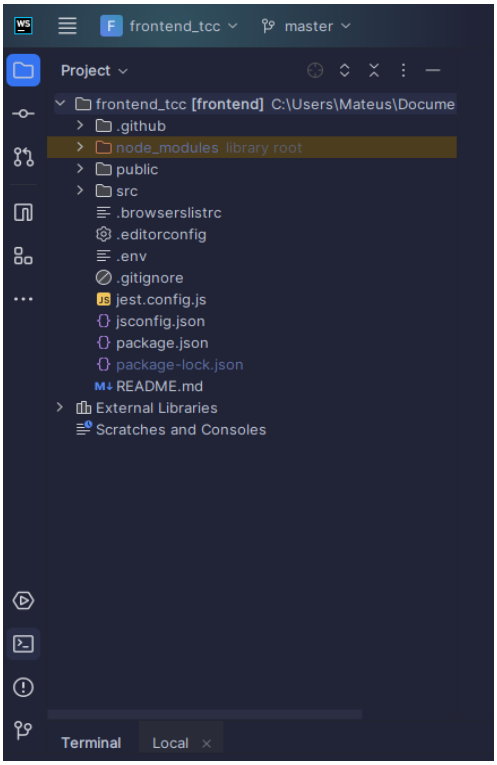
Dependências	Descrição
axios	Cliente HTTP baseado em promessas para fazer requisições HTTP.
chart.js	Biblioteca JavaScript para criar gráficos e visualizações de dados.
classnames	Utilitário para condicionalmente juntar classes CSS.
react-imask	Biblioteca para mascarar entradas em campos de formulário em aplicações React.
react-to-print	Biblioteca para permitir a impressão de componentes em aplicações React.
redux	Biblioteca para gerenciamento de estado previsível em aplicações JavaScript.
simplebar-react	Implementação de barras de rolagem customizadas em React usando SimpleBar.
react-redux	Biblioteca para integração do React com o Redux, facilitando o gerenciamento de estado global.
prop-types	Biblioteca para checagem de tipos em componentes React, ajudando a garantir que os props passados para os componentes sejam do tipo esperado.
react-app-polyfill	Polifills para suporte a navegadores mais antigos em aplicativos React.

Tabela 5.3: Tabela de dependências de terceiros.

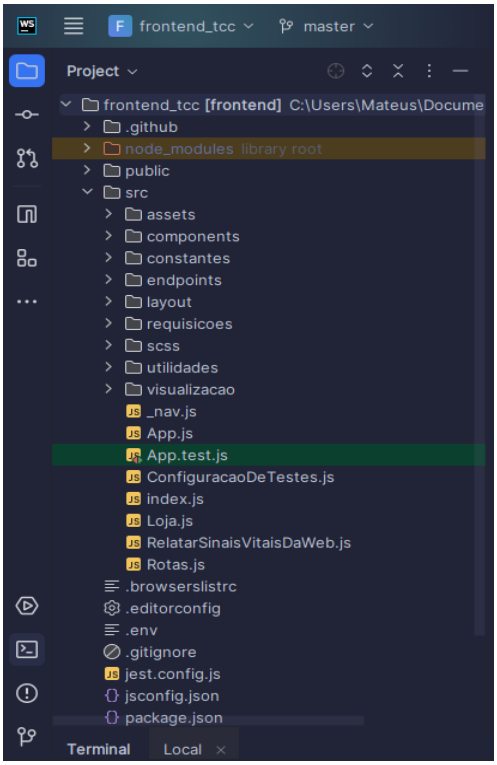
5.2 Organização de Pastas

A organização de pastas em um projeto React é crucial para assegurar que o código seja eficiente, sustentável, e de fácil manutenção. A estruturação correta das pastas facilita o desenvolvimento, colaboração e a expansão do projeto. Abaixo, apresento uma descrição detalhada de cada pasta, destacando a sua importância no contexto geral do desenvolvimento.

A Figura 5.1, detalha a estrutura de pacotes do Front-end em React. Na figura (a) está uma visualização mais geral, e a figura (b), exibe detalhadamente a estrutura de pacotes e pastas.



(a) Estrutura geral 1.



(b) Estrutura geral 2.

Figura 5.1: Estrutura de Pacotes do Front-end

⇒ **Assets** A pasta Imagens é dedicada ao armazenamento de todos os arquivos de imagem que o projeto utiliza, como logos, ícones, e outras imagens estáticas. Manter essas imagens centralizadas proporciona várias vantagens para a equipe de desenvolvimento e design.

⇒ **Componentes** A pasta Componentes contém todos os componentes React que constituem a aplicação. Estes componentes são as unidades fundamentais de construção em React, e sua organização eficaz é essencial para uma aplicação escalável e modular.

⇒ **Constantes** A pasta Constantes armazena valores fixos usados em várias partes do projeto,

como strings de configuração, números mágicos e outros valores que necessitam de centralização para fácil manutenção e clareza.

⇒ **Endpoints** A pasta Endpoints contém arquivos que definem os endpoints da API que a aplicação consome. Manter estas definições centralizadas é crucial para a gestão eficiente das comunicações da API.

⇒ **Layout** A pasta Layout é responsável pelos componentes e arquivos que definem a estrutura visual e o layout da aplicação. Elementos como cabeçalhos, rodapés, e barras laterais são encontrados aqui, assegurando uma experiência de usuário consistente.

⇒ **Requisições** A pasta Requisições armazena funções e módulos responsáveis por realizar requisições HTTP, como chamadas a APIs. A centralização deste código é vital para a manutenção eficiente da comunicação do aplicativo.

⇒ **SCSS** A pasta SCSS contém arquivos de estilo escritos em SCSS (Sass), uma extensão do CSS que oferece funcionalidades avançadas para um desenvolvimento de estilo mais eficiente.

⇒ **Utilidades** A pasta Utilidades é dedicada a funções e classes auxiliares que servem para diferentes partes do projeto, ajudando a centralizar a lógica comum e prevenir a duplicação de código.

⇒ **Formulários** A pasta Formulários abriga componentes e lógicas específicas para a criação e gerenciamento de formulários, incluindo validações, manipulação de estados e interação do usuário.

5.3 Login

Nessa etapa foi realizada a implementação do sistema de Login, garantindo que apenas usuários autorizados possam acessar o sistema, com foco na segurança e usabilidade.

⇒ **Formulário de Autenticação:** Inclui campos para nome de usuário e senha, com validação de entrada.

⇒ **Recuperação de Senha:** Permite aos usuários redefinir suas senhas através de um fluxo de recuperação seguro.

5.4 Dashboard

Nessa etapa foi realizada a implementação do Dashboard, que serve como a página inicial do sistema e fornece uma visão geral das principais métricas e informações relacionadas às atividades de equoterapia.

⇒ **Praticante:** Exibe opções para gerenciar os praticantes

⇒ **Usuário:** Exibe opções para gerenciar os usuários.

5.5 Cadastro, Atualização, Consulta e Exclusão de Usuário

Nessa etapa foi implementado o módulo de gerenciamento de usuários, que permite a administração completa do ciclo de vida dos usuários do sistema.

⇒ **Cadastro de Usuários:** Formulário para adicionar novos usuários com validação de dados.

⇒ **Atualização de Dados:** Permite a edição das informações dos usuários existentes.

⇒ **Consulta e Listagem:** Exibe todos os usuários em uma tabela com opções de busca e filtro.

⇒ **Exclusão de Usuários:** Opção de remover usuários com confirmação para evitar exclusões acidentais.

5.6 Cadastro, Atualização e Consulta de Praticante

Nessa etapa foi realizado o desenvolvimento do módulo de gerenciamento de praticantes, essencial para acompanhar o progresso individual de cada praticante.

⇒ **Cadastro de Praticantes:** Formulário detalhado para inserir informações pessoais e médicas.

⇒ **Atualização de Informações:** Edição dos dados do praticante conforme o progresso no tratamento.

⇒ **Consulta de Praticantes:** Permite visualizar e filtrar praticantes cadastrados no sistema.

5.7 Relatório do Praticante

Nessa etapa foi implementada a funcionalidade de geração de relatórios para os praticantes, permitindo um acompanhamento detalhado de seu desenvolvimento.

⇒ **Geração de Relatórios:** Criação de relatórios personalizados baseados em dados selecionados.

⇒ **Exportação de Dados:** Opção para exportar relatórios em formato PDF.

⇒ **Visualização Detalhada:** Interface para análise dos relatórios gerados.

5.8 Evolução do Praticante

Nessa etapa foi desenvolvida a funcionalidade de monitoramento da evolução dos praticantes ao longo das sessões, auxiliando no planejamento do tratamento.

⇒ **Linha do Tempo de Evolução:** Visualização cronológica dos marcos e progresso dos praticantes.

⇒ **Histórico de Sessões:** Registro detalhado de cada sessão realizada com feedback.

⇒ **Análise Comparativa:** Ferramenta para comparar o progresso entre diferentes períodos.

5.9 Gráficos da Evolução do Praticante

Nessa etapa foi implementada a visualização gráfica do progresso dos praticantes, utilizando gráficos para representar dados complexos de forma clara.

⇒ **Gráficos Interativos:** Visualização de dados de evolução através de gráficos dinâmicos.

⇒ **Personalização de Dados:** Opções para selecionar período de tempo para serem exibidos nos gráficos.

5.10 Trechos de Código

A seguir estão alguns trechos de códigos do front-end, onde mostra códigos de tabelas, campos de entrada, dashboard e entre outros.

A Figura 5.1, mostra o componente pai do dashboard, que possui outros componentes, como, barra lateral, cabeçalho, conteúdo e rodapé.

```
const LayoutPadrao = () => {
    useEffect(() => {
      let login = JSON.parse(localStorage.getItem("login"))
      if (!(login.idUsuario !== "" && login.nomeUsuario !== "" &&
         login.token !== "" && login.role !== "")) {
        // N o  est   logado
        window.location.href = "/"
      }
    }, []);
    return (
        <div>
            <AppSidebar/>
            <div className="wrapper d-flex flex-column min-vh-100 bg-
               light">
                <AppCabecalho/>
                <div className="body flex-grow-1 px-3">
                    <AppContent/>
                </div>
                <AppRodaPe/>
            </div>
        </div>
    )
}

export default LayoutPadrao
```

Listing 5.1: Componente principal do dashboard

A Figura 5.2, mostra o componente padrão para criar campos de entrada de dados dos formulários.

```
const Campo = (props) => {
  const renderField = () => {
    // Verifica o para garantir que `props.valor` n o seja uma RegExp
    if (props.valor instanceof RegExp) {
      console.error(`Invalid prop valor: ${props.valor} is a RegExp object`);
      return null;
    }

    switch (props.tipo) {
      case 'select':
        return (
          <CFormSelect
            id={props.id}
            value={props.valor}
            onChange={props.setar}
            disabled={props.disabled}
          >
            <option value="">Selecionar</option>
            {props.opcoes.map((option, index) => (
              <option key={index} value={option.value}>
                {option.label}
              </option>
            ))}
          </CFormSelect>
        );
      case 'textarea':
        return (
          <CFormTextarea
            id={props.id}
            value={props.valor === "NAO_INFORMADO" ? "" : props.valor}
            onChange={props.setar}
            disabled={props.disabled}
            placeholder={props.descricao}
          />
        );
      case 'file':
        return (
          <CFormInput
            id={props.id}
            type="file"
            onChange={props.setar}
            disabled={props.disabled}
          />
        );
      default:
```

```
46          return (
47            <CFormInput
48              id={props.id}
49              type={props.tipo}
50              value={props.valor === "NAO_INFORMADO" ? "" : props.valor
                    }
51              onChange={props.setar}
52              disabled={props.disabled}
53              placeholder={props.descricao}
54            />
55          );
56      }
57    };
58
59    return (
60      <div className="mb-3">
61        <CFormLabel htmlFor={props.id} style={{display:"block",
              textAlign:"left"}}>{props.legenda}</CFormLabel>
62        {renderField()}
63      </div>
64    );
65  };
66
67  export default Campo;
```

Listing 5.2: Componente de campo de entrada de dados

A Figura 5.3, mostra o componente da tabela de praticante, onde é possível visualizar dados importantes sobre a evolução do praticante.

```
const TabelaPraticante = (props) => {

  return (
    <CTable hover>
      <CTableHead>
        <CTableRow>
          <CTableHeaderCell style={{width: "20px", textAlign: "center
             "}}>C digo </CTableHeaderCell>
          <CTableHeaderCell style={{width: "300px", textAlign: "
             center"}}>Nome completo</CTableHeaderCell>
          <CTableHeaderCell style={{width: "600px", textAlign: "
             center"}}>Diagn stico cl nico </CTableHeaderCell>
          <CTableHeaderCell colSpan="3" style={{width: "25px",
             textAlign: "center"}}>Frequ ncia </CTableHeaderCell>
          <CTableHeaderCell style={{width: "50px", textAlign: "center
             "}}>Relat rio </CTableHeaderCell>
          <CTableHeaderCell style={{width: "80px", textAlign: "center
             "}}>Evoluir</CTableHeaderCell>
        </CTableRow>
      </CTableHead>

      <CTableBody>
        {
          props.lista.map((item, key) => (
              <CTableRow key={key}>
                <CTableDataCell style={{textAlign: "center"}}>{item.
                   praticante.idPraticante}</CTableDataCell>
                <CTableDataCell style={{textAlign: "center"}}>
                  {
                    item.finalizado ?

                        <a href={`${ATUALIZAR_PRATICANTE}?id=${item.
                           praticante.idPraticante}`}
                           style={{textDecoration: "none", color: "
                              green"}}
                           title={"Atualizar cadastro do(a)
                              praticante " + item.nomeCompleto}>{item
                              .nomeCompleto} <strong>
                          <svg xmlns="http://www.w3.org/2000/svg"
                             width="16" height="16" fill="
                             currentColor" className="bi bi-check2-
                             circle" viewBox="0 0 16 16">
                            // Path do svg...
                          </svg>
                        </strong></a>
                    :
                    <p>{item.nomeCompleto} <i>[<a href={`${
                       CADASTRO_PRATICANTE}?id=${item.praticante.
```

```
                idPraticante}`}>Pendente</a>]</i></p>
            }
          </CTableDataCell>
          <CTableDataCell style={{textAlign: "center"}}>{item.
              diagnosticoClinico}</CTableDataCell>
          <CTableDataCell style={{verticalAlign: "middle",
              textAlign: "center"}}>
            <ModalComEvolucaoGraficoDeLinhas nomeCompleto={item
                .nomeCompleto}
                                             idPraticante={item
                                                .praticante.
                                                idPraticante}/>
          </CTableDataCell>
          <CTableDataCell style={{verticalAlign: "middle",
              textAlign: "center"}}>
            <ModalComEvolucaoGraficoDeBarras nomeCompleto={item
                .nomeCompleto}
                                             idPraticante={item
                                                .praticante.
                                                idPraticante}/>
          </CTableDataCell>
          <CTableDataCell style={{verticalAlign: "middle",
              textAlign: "center"}}>
            <ModalComEvolucaoGraficoDeTorta nomeCompleto={item.
                nomeCompleto}
                                            idPraticante={item.
                                               praticante.
                                               idPraticante}/>
          </CTableDataCell>
          <CTableDataCell style={{verticalAlign: "middle",
              textAlign: "center"}}>
            <CButton color="" title={"Gerar relat rio para " +
                item.nomeCompleto} style={{background: "none"}}
                    onClick={
                      () => {
                        window.location.href = `${
                           GERAR_RELATORIO_PRATICANTE}?id=${
                           item.praticante.idPraticante}`
                      }
                    }>
              <svg xmlns="http://www.w3.org/2000/svg" width="20
                  " height="20" fill="currentColor"
                    className="bi bi-file-pdf-fill" viewBox="0 0
                        16 16">
                    // Path do svg...
              </svg>
            </CButton>
          </CTableDataCell>
          <CTableDataCell style={{verticalAlign: "middle",
              textAlign: "center"}}>
            <ModalParaEvoluir nomeCompleto={item.nomeCompleto}
```

```
                        idPraticante={item.praticante.idPraticante}/>
64                  </CTableDataCell>
65                </CTableRow>
66              )
67            )}
68        </CTableBody>
69      </CTable>
70    )
71  }
72  export default TabelaPraticante
```

Listing 5.3: Componente da tabela do praticante

A Figura 5.4, mostra o componente da tabela de usuário, onde é possível visualizar dados importantes sobre eles, e assim, realizar algumas ações, como, atualizações e remoções de usuários.

```
const TabelaDeUsuarios = ({ list , setDisplayModalOpcoes,
   setTituloModalOpcoes, setConteudoModalOpcoes, setIdParaDeletar})
   => {
  return (
    <CTable hover>
      <CTableHead>
        <CTableRow>
          <CTableHeaderCell>Nome</CTableHeaderCell>
          <CTableHeaderCell>Email</CTableHeaderCell>
          <CTableHeaderCell>Telefone</CTableHeaderCell>
          <CTableHeaderCell>Forma  o </CTableHeaderCell>
          <CTableHeaderCell>N vel </CTableHeaderCell>
          <CTableHeaderCell>V nculo </CTableHeaderCell>
          <CTableHeaderCell>A  es </CTableHeaderCell>
        </CTableRow>
      </CTableHead>
      <CTableBody>
        {
          list.map((item) => (
            <CTableRow key={item.idUsuario} style={{cursor:"pointer"
               }}>
              <CTableDataCell style={{ verticalAlign: "middle",
                 minWidth: "150px" }}>{item.nome}</CTableDataCell>
              <CTableDataCell style={{ verticalAlign: "middle",
                 minWidth: "150px" }}>{item.email}</CTableDataCell>
              <CTableDataCell style={{ verticalAlign: "middle",
                 minWidth: "150px" }}>{item.telefone}</CTableDataCell
                 >
              <CTableDataCell style={{ verticalAlign: "middle",
                 minWidth: "150px" }}>{item.possuiFormacao ? 'Sim' :
                 'N o'}</CTableDataCell>
              <CTableDataCell style={{ verticalAlign: "middle",
                 minWidth: "150px", fontWeight:"bold", color: item.
                 role === 'ROLE_USER' ? '#e55353':'#e55353' }}>
                {item.role === 'ROLE_USER' ? 'Usu rio' : '
                   Administrador'}
              </CTableDataCell>
              <CTableDataCell style={{ verticalAlign: "middle",
                 minWidth: "150px" }}>{formatarVinculo(item.vinculo)
                 }</CTableDataCell>
              <CTableDataCell style={{ verticalAlign: "middle",
                 minWidth: "150px" }}>
                <div className="container text-center">
                  <div className="row">
                    <div className="col">
                      <CButton color="" title={`Ser  redirecionado
                         para atualizar os dados de ${item.nome}`}
```

```
                        onClick={() => {
32                        window.location.href = `${ATUALIZAR_USUARIO}?
                            id=${item.idUsuario}`
33                      }}><CIcon icon={cilSettings}/></CButton>
34                    </div>
35                    <div className="col">
36                      <CButton color="" title={`Esta a   o deleta ${
                          item.nome}`} onClick={() => {
37                        apresentarModalDeOpcoes("Aten   o",
38                        "Deseja realmente deletar este usu rio?",
39                        setDisplayModalOpcoes,
40                        setTituloModalOpcoes,
41                        setConteudoModalOpcoes)
42                        setIdParaDeletar(item.idUsuario)
43                      }
44                      }><CIcon icon={cilTrash}/></CButton>
45                    </div>
46                  </div>
47                </div>
48              </CTableDataCell>
49            </CTableRow>
50          ))
51        }
52      </CTableBody>
53    </CTable>
54  );
55 }
56 export default TabelaDeUsuarios;
```

Listing 5.4: Componente da tabela de usuário

5.11 Testes e Considerações Finais do Capítulo

Após a implementação do sistema, ele foi inicialmente disponibilizado para pessoas da minha rede social do Instagram, para que realizassem testes na plataforma, que estava temporariamente acessível em https://equoterapia.vercel.app/. Durante os primeiros testes, surgiram alguns erros de escrita, lógica, entre outros, que foram reportados pelos usuários. Após isso, as correções necessárias foram realizadas, e o sistema foi novamente disponibilizado para testes.

Em seguida, os usuários da minha rede no Instagram acessaram a plataforma novamente, e o feedback obtido foi positivo. Posteriormente, o sistema foi disponibilizado para o representante da equipe de equoterapia do Instituto Federal Goiano Campus Urutaí, que permitiu que os funcionários o utilizassem. O representante retornou com feedback, em uma reunião pelo Google Meet, apontando alguns erros e sugerindo a adição de novas funcionalidades. Após as correções, foi realizada outra reunião para apresentar os novos resultados, e, mais uma vez, foram necessárias algumas correções e adaptações. Finalmente, o sistema foi totalmente corrigido, concluído, aprovado e entregue ao representante da equoterapia.

O sistema será disponibilizado em algum servidor do IF Goiano ou em um servidor particular na nuvem, como a AWS. Por fim, no próximo capítulo, será apresentado o resultado final do sistema após todas as correções realizadas.

CAPÍTULO 6

A APLICAÇÃO

Até então, foi apresentado vários detalhes sobre a implementação do sistema, e neste capítulo, será apresentado o resultado final depois de muito trabalho. A seguir estão as telas do sistema, que mostram todas as funcionalidades, desde login até a geração de relatórios de praticantes.

6.1 Tela de Login

A Figura 6.1, apresenta a tela de login do sistema, onde o usuário pode inserir seu nome de usuário e senha para acessar o sistema. No primeiro acesso, é necessário aceitar os Termos de Uso. Para isso, o usuário deve clicar no link destacado "Termos de Uso", ler o conteúdo e concordar para prosseguir com o uso do sistema. Caso o usuário tente fazer login sem aceitar os termos, uma mensagem de aviso será exibida, informando que é necessário aceitar os Termos de Uso.

Além disso, há um link para recuperação de senha caso o usuário a tenha esquecido. Após preencher as credenciais, basta clicar no botão "Entrar"e aguardar o redirecionamento, caso as informações estejam corretas. Ao lado do formulário, está exibido o logotipo oficial da Equoterapia do IFG - Campus Urutaí.

Figura 6.1: Tela de Login.

A Figura 6.2, apresenta os Termos de Uso do sistema ao usuário. Ao final da leitura, há um botão para aceitar os termos. Caso o usuário não concorde, ele pode simplesmente fechar a janela dos Termos de Uso.

Figura 6.2: Termos de uso.

A Figura 6.3, mostra o botão de aceitação dos Termos de Uso com a opção "Não aceito os termos de uso!"selecionada.

Figura 6.3: Termos de uso não aceito.

A Figura 6.4, mostra uma mensagem ao usuário após a tentativa de realizar o login sem aceitar os termos de uso do sistema.

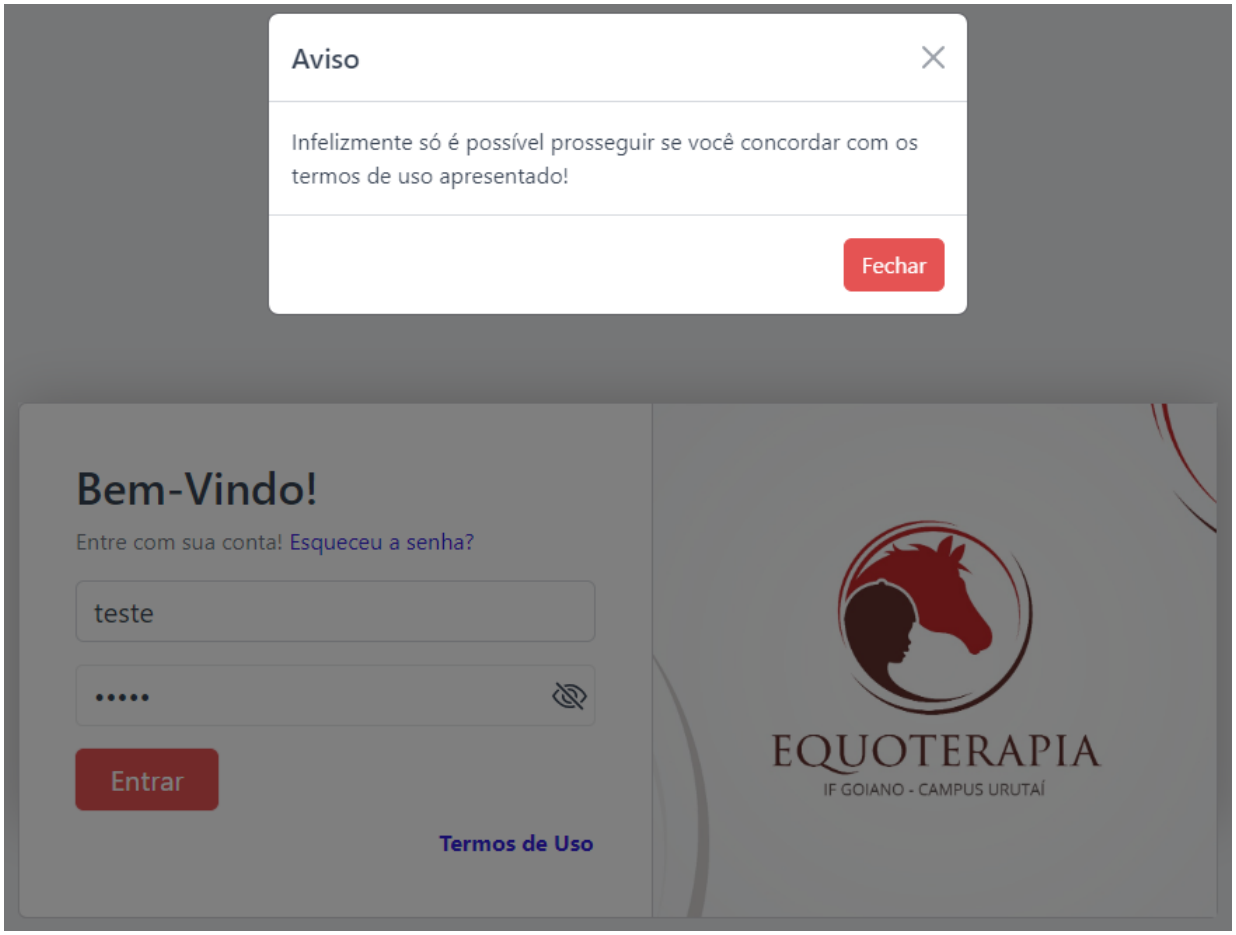


Figura 6.4: Mensagem de login.

A Figura 6.5, mostra o botão de aceitação dos Termos de Uso com a opção "Sim, eu li, concordo e aceito todos os termos de uso!"selecionada. Sendo assim, o usuário conseguirá realizar o login.

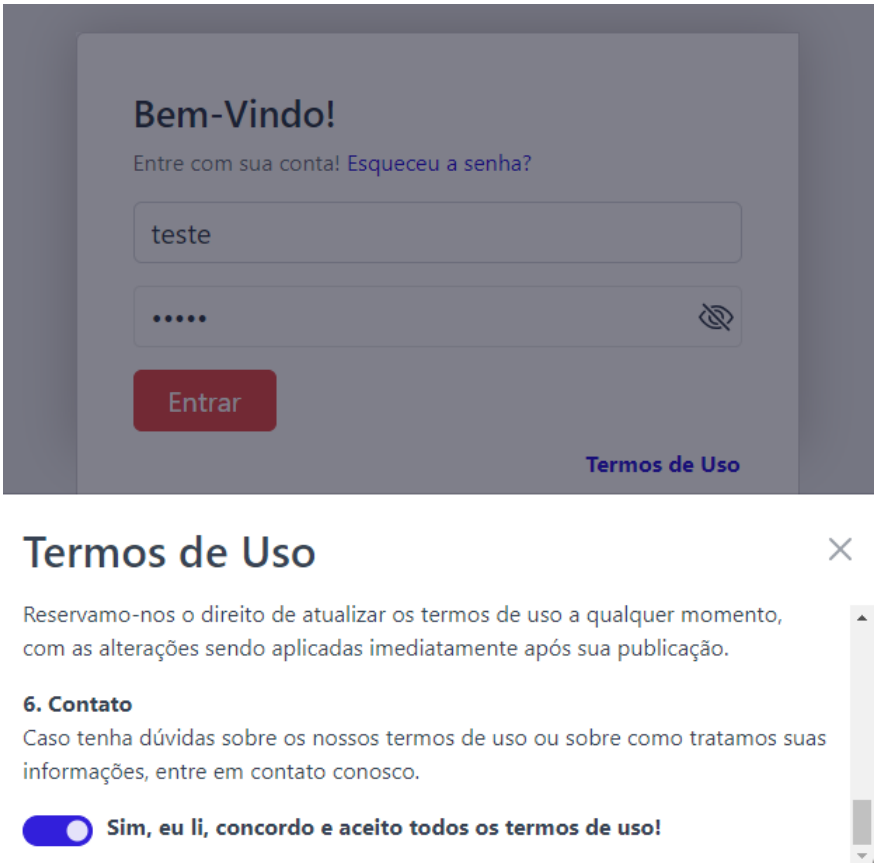


Figura 6.5: Termos de uso aceito.

A Figura 6.6, exibe a tela de login sem a opção "Termos de Uso", uma vez que os termos já foram aceitos. Caso os termos ainda não tenham sido aceitos, essa opção permanecerá disponível para que o usuário possa acessá-los e concordar.

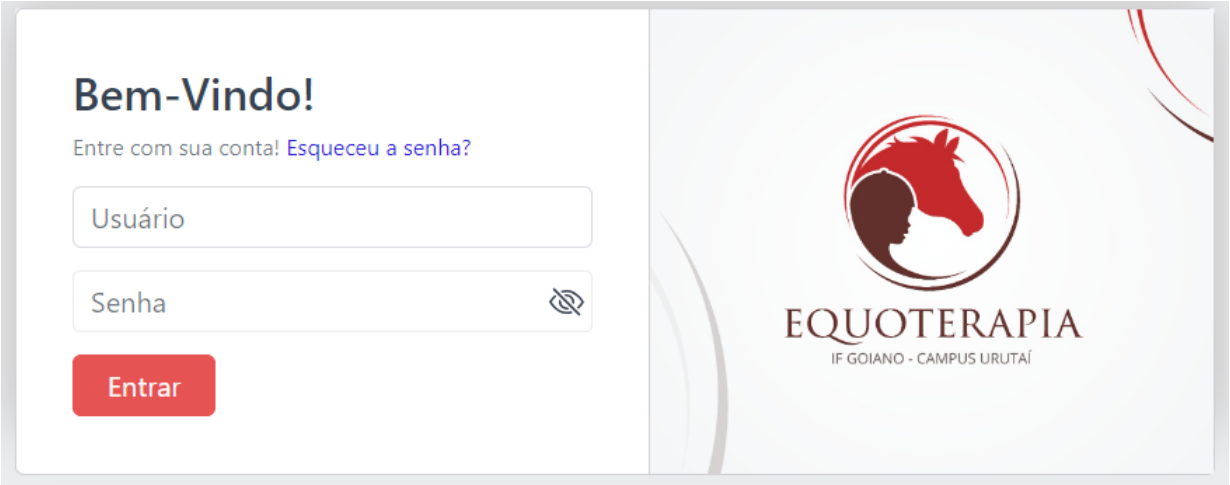


Figura 6.6: Tela de login após aceitação dos termos de uso.

6.2 Recuperação de conta

A Figura 6.7, mostra um formulário de recuperação de conta. O usuário informa seu email e então é enviado um link de recuperação de conta para seu email. Caso o email informado seja inválido, o usuário será notificado.

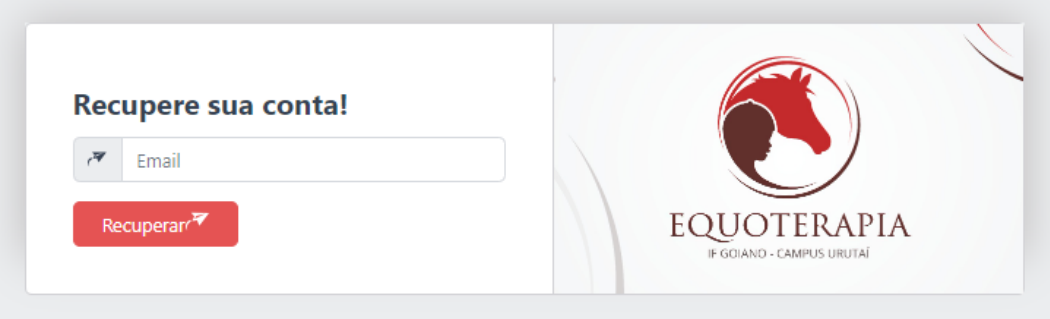


Figura 6.7: Tela de recuperação de conta.

A Figura 6.8, mostra uma animação enquanto o email está sendo enviado para o usuário. Após a conclusão do envio, a animação é desativada.

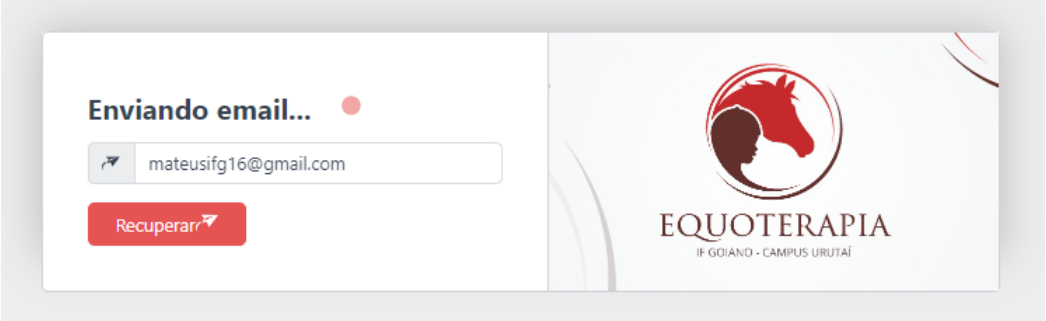


Figura 6.8: Tela de recuperação de conta, enviando email.

A Figura 6.9, mostra uma mensagem informando que o email foi enviado com sucesso. Dessa forma, o usuário pode entrar em seu email e prosseguir com a recuperação de conta.

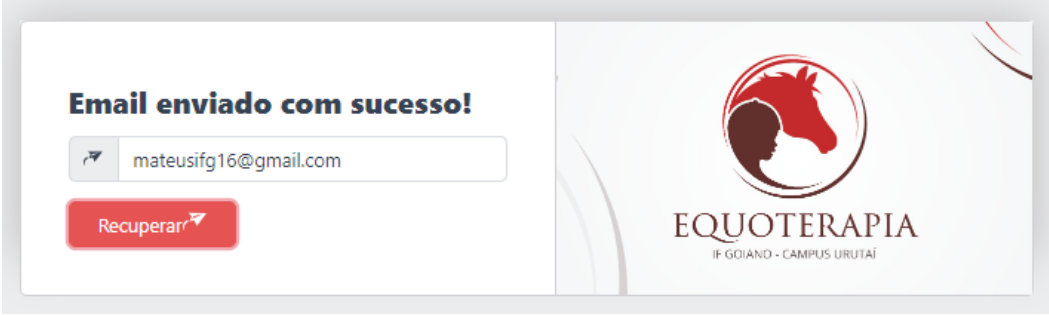


Figura 6.9: Tela de recuperação de conta, email enviado com sucesso.

A Figura 6.10, mostra a notificação no email do usuário. Ao clicar no botão, será redirecionado o usuário para informar sua nova senha.

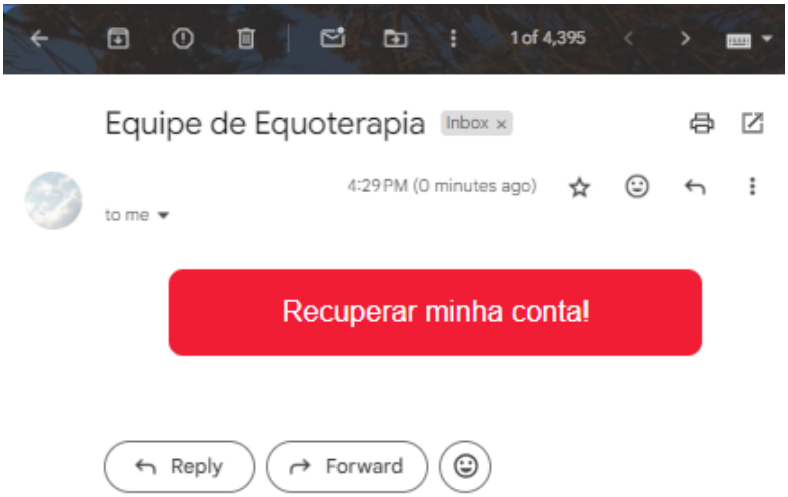


Figura 6.10: Tela de notificação de email.

A Figura 6.11, mostra a tela responsável por permitir ao usuário redefinir sua senha. Dessa forma, o usuário informa a nova senha e, em seguida, a repete para confirmação. Caso a senha seja válida, a troca será concluída, o usuário receberá uma mensagem de sucesso, e por fim, ele será redirecionado para a página de login; caso contrário, o usuário será informado com uma mensagem de erro.

Figura 6.11: Tela para informar a nova senha.

6.3 Mensagem de Login Inválido

A Figura 6.12, mostra a mensagem de aviso gerada no back-end após o usuário fornecer credenciais inválidas.

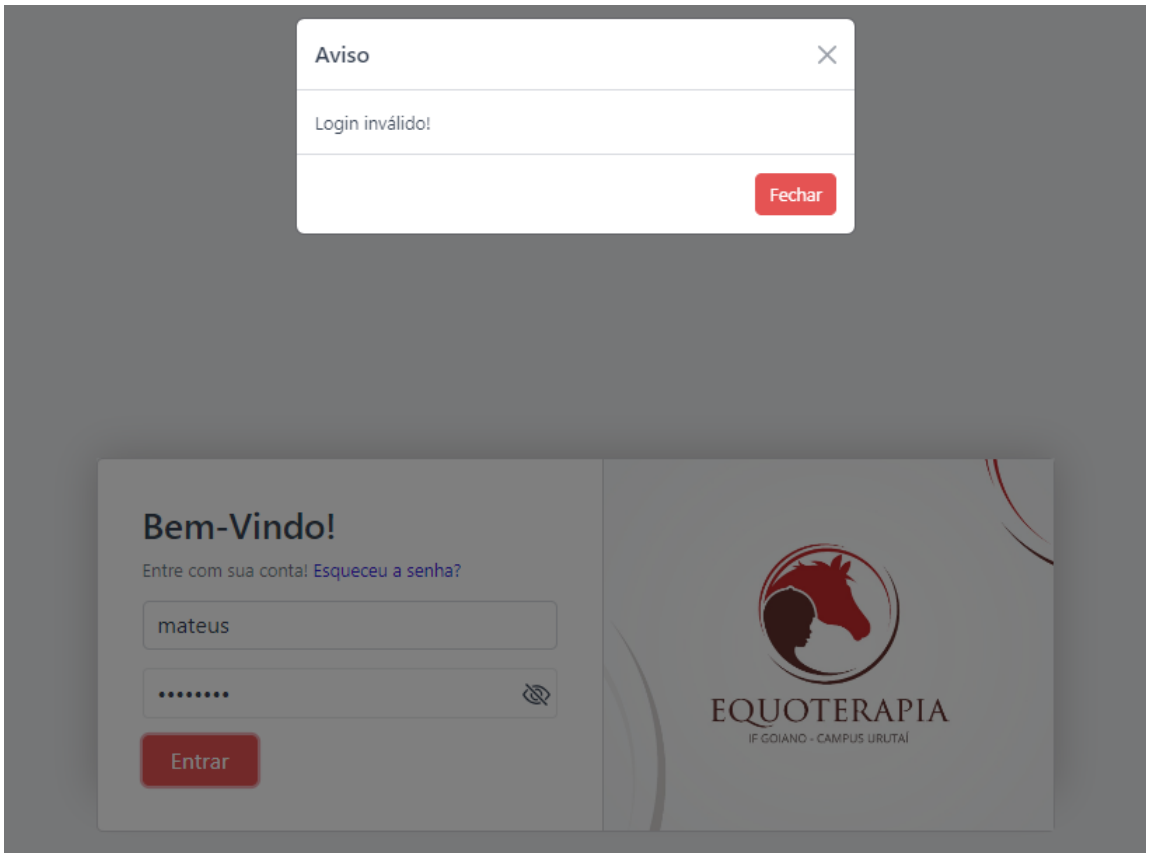


Figura 6.12: Login inválido.

6.4 Mensagem de Login Bloqueado

A Figura 6.13, mostra uma mensagem de bloqueio de login do usuário *mateus10* após três (3) tentativas consecutivas de login inválidas. Sendo assim o usuário terá que esperar por 60 minutos para tentar realizar login com o mesmo usuário.

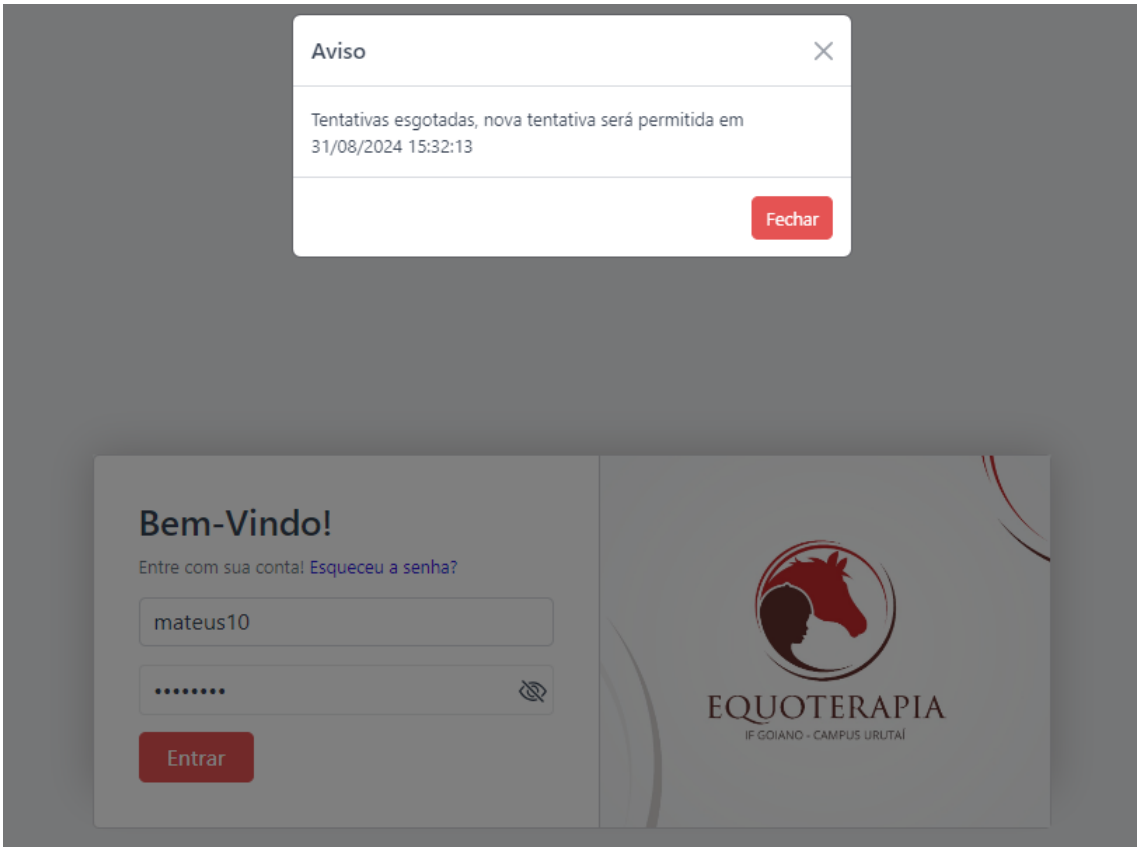


Figura 6.13: Tentativas Esgotadas.

6.5 Tela Inicial

A Figura 6.14, mostra a página carregada ao administrador após realizar o login no sistema. Inicialmente é exibido os formulários para cadastro de praticante.

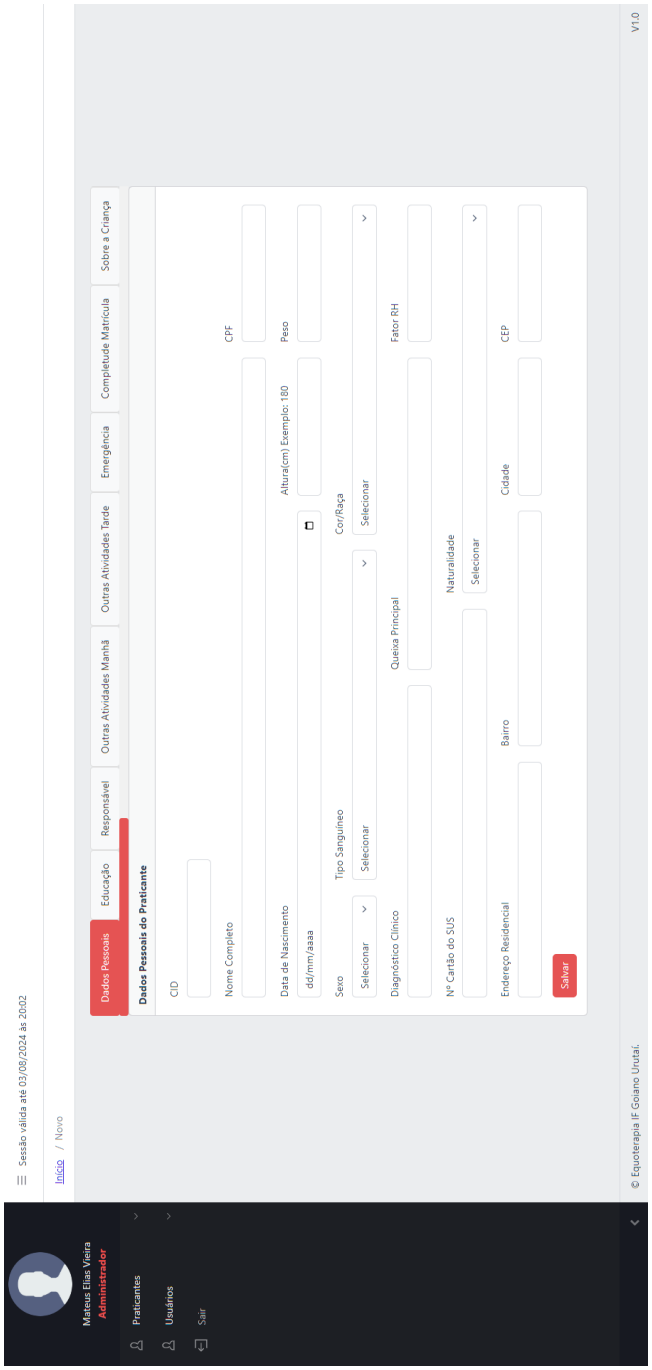
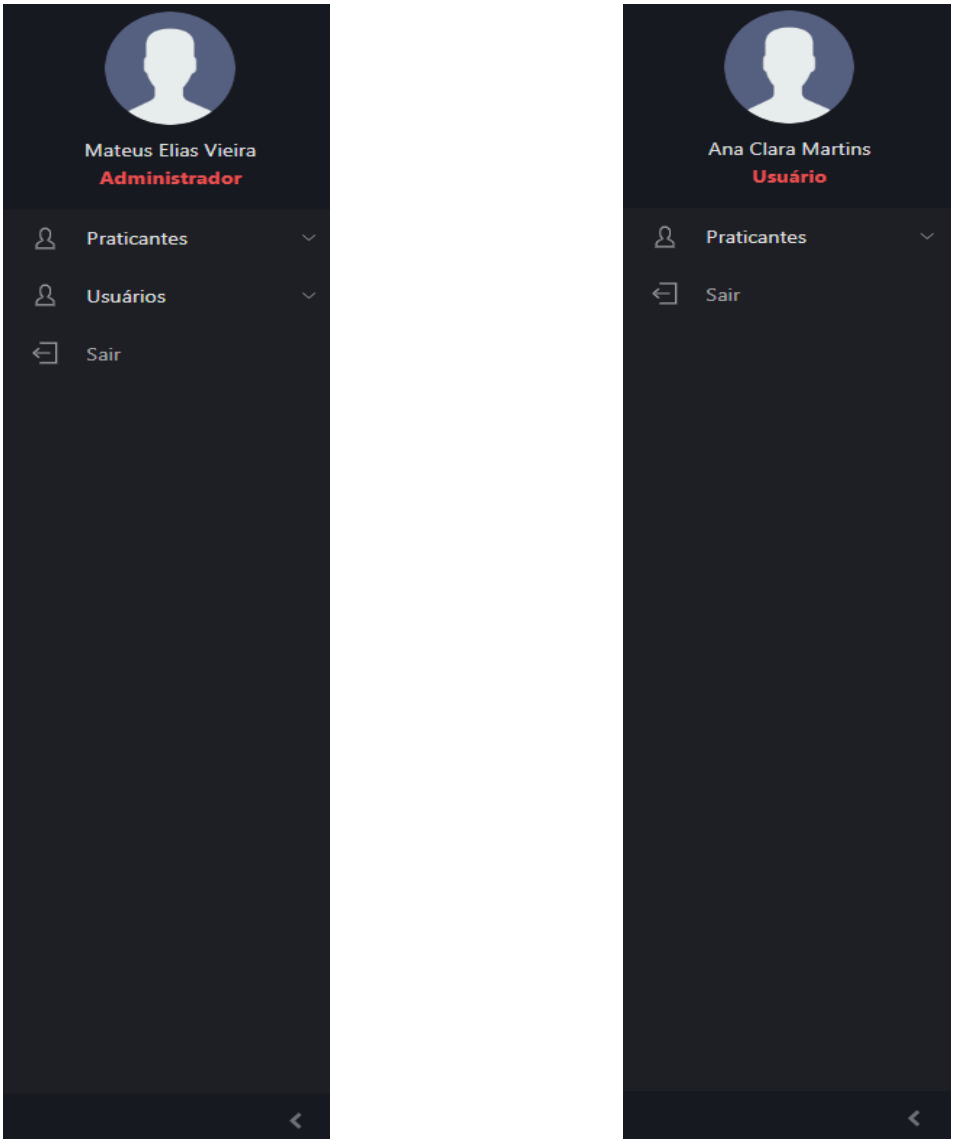


Figura 6.14: Tela inicial do administrador.

6.6 Menu Lateral

A Figura 6.15, mostra a diferença entre funcionalidades disponíveis para diferentes tipos de usuários do sistema. As opções disponíveis para usuário e administrador estão fechadas, sendo assim, apenas serão abertas quando clicar no botão.



(a) Menu Administrador. **(b)** Menu Usuário.

Figura 6.15: Menus de administrador e usuário.

6.7 Pesquisa de Praticante

A Figura 6.16 mostra praticantes fictícios cadastrados no sistema, bem como algumas funcionalidades:

- Pesquisa dinâmica por nome do praticante
- Gráficos de linhas da evolução
- Gráficos de barras da evolução
- Gráficos de pizza da evolução
- Gerar relatório completo
- Evoluir praticante

Ademais, para cada praticante existe uma barra mostrando a porcentagem de cadastro realizado. Aqueles que já concluíram todo o cadastro mostram uma porcentagem de 100% e a barra na cor verde. Dessa forma, o usuário que clicar no nome do praticante já concluído será redirecionado para uma página onde poderá realizar atualizações no cadastro, se necessário.

Também há uma barra vermelha com uma porcentagem inferior a 100%, indicando que o cadastro do praticante ainda precisa ser finalizado. Ao lado do nome do praticante, existe um link que mostra quantos formulários ainda faltam para concluir o cadastro. Ao clicar neste link, o usuário será redirecionado para finalizar o cadastro do praticante. Os praticantes que não concluíram seu cadastro não podem ser atualizados; apenas aqueles que já foram concluídos podem ser atualizados.

Por fim, existe paginação para realizar a listagem dos praticantes. Essa paginação contém cinco registros por página e o uso dessa funcionalidade resulta em vários benefícios, sendo a principal, o melhor desempenho do sistema.

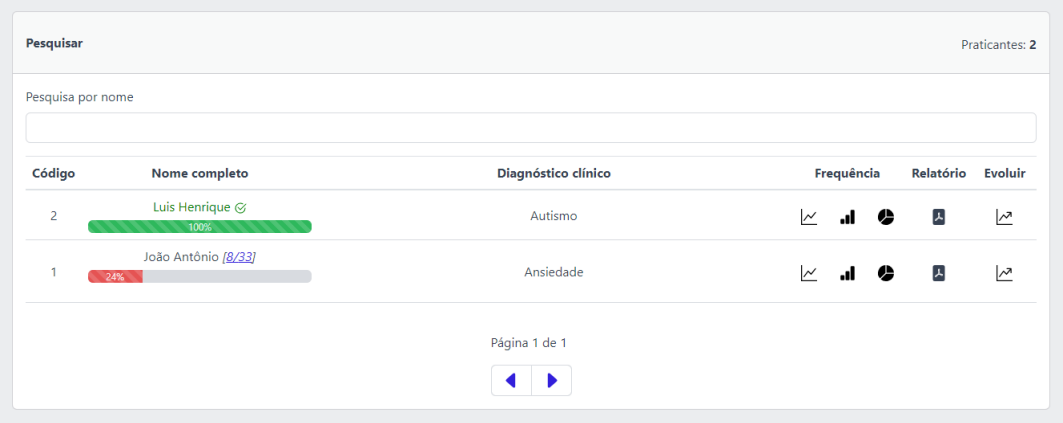


Figura 6.16: Tabela de listagem de praticantes.

A Figura 6.17, mostra uma situação onde o usuário precisa fazer uma pesquisa de um praticante pelo seu nome. Dessa forma, ao começar a pesquisa, o sistema faz a filtragem a cada nova letra digitada, resultando em uma pesquisa clara e objetiva.

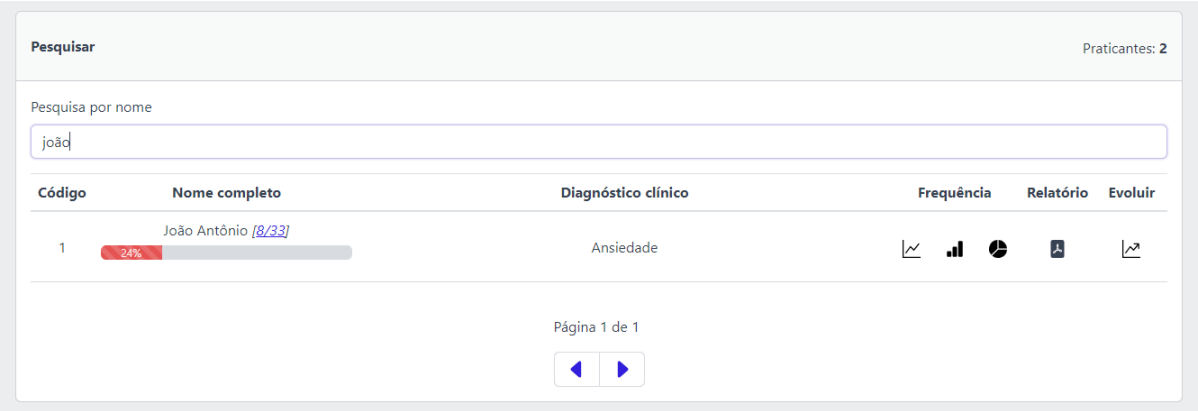


Figura 6.17: Pesquisa rápida.

6.8 Evolução do Praticante

A Figura 6.18 mostra o formulário para evoluir o praticante. Para adicionar uma nova evolução, é necessário selecionar se o praticante estava presente (sim/não), a data do atendimento e fornecer observações (opcional). Não é permitido registrar uma evolução com datas futuras ou datas que já foram aplicadas anteriormente.

Além disso, é possível visualizar e atualizar as observações de cada praticante. Para atualizar a evolução de um praticante, basta clicar no ícone de atualizar localizado na última coluna de cada linha da tabela. Após clicar no ícone, os dados serão carregados nos campos do formulário e o botão de atualizar será desbloqueado. O usuário poderá então fazer as alterações necessárias e clicar em atualizar.

Vale ressaltar que também existem restrições ao tentar atualizar um praticante com uma data inválida, conforme mencionado para realizar o cadastro de uma nova evolução.

Figura 6.18: Evoluir praticante.

A Figura 6.19, mostra uma mensagem de aviso ao usuário caso exista uma tentativa de dar frequência para o praticante mais de 1 vez ao dia.

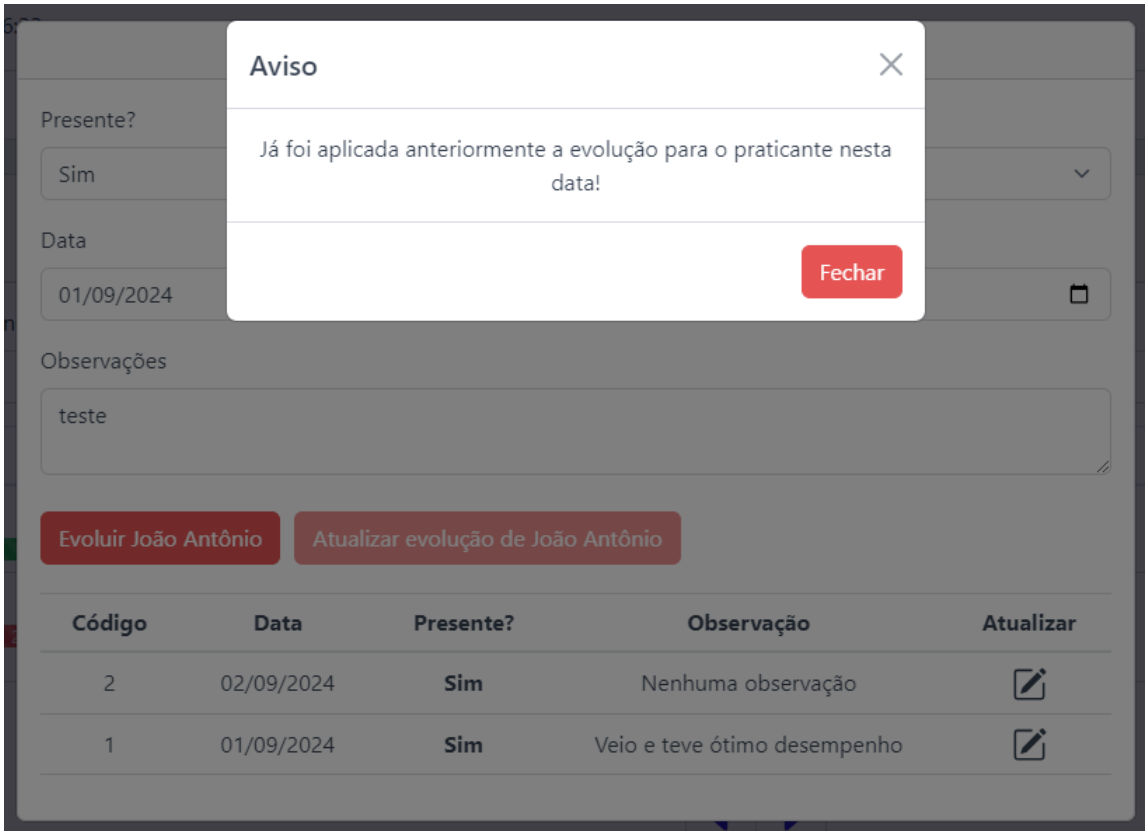


Figura 6.19: Falha ao evoluir praticante.

A Figura 6.20, mostra uma mensagem de aviso ao usuário caso tenha evoluído o praticante com sucesso.

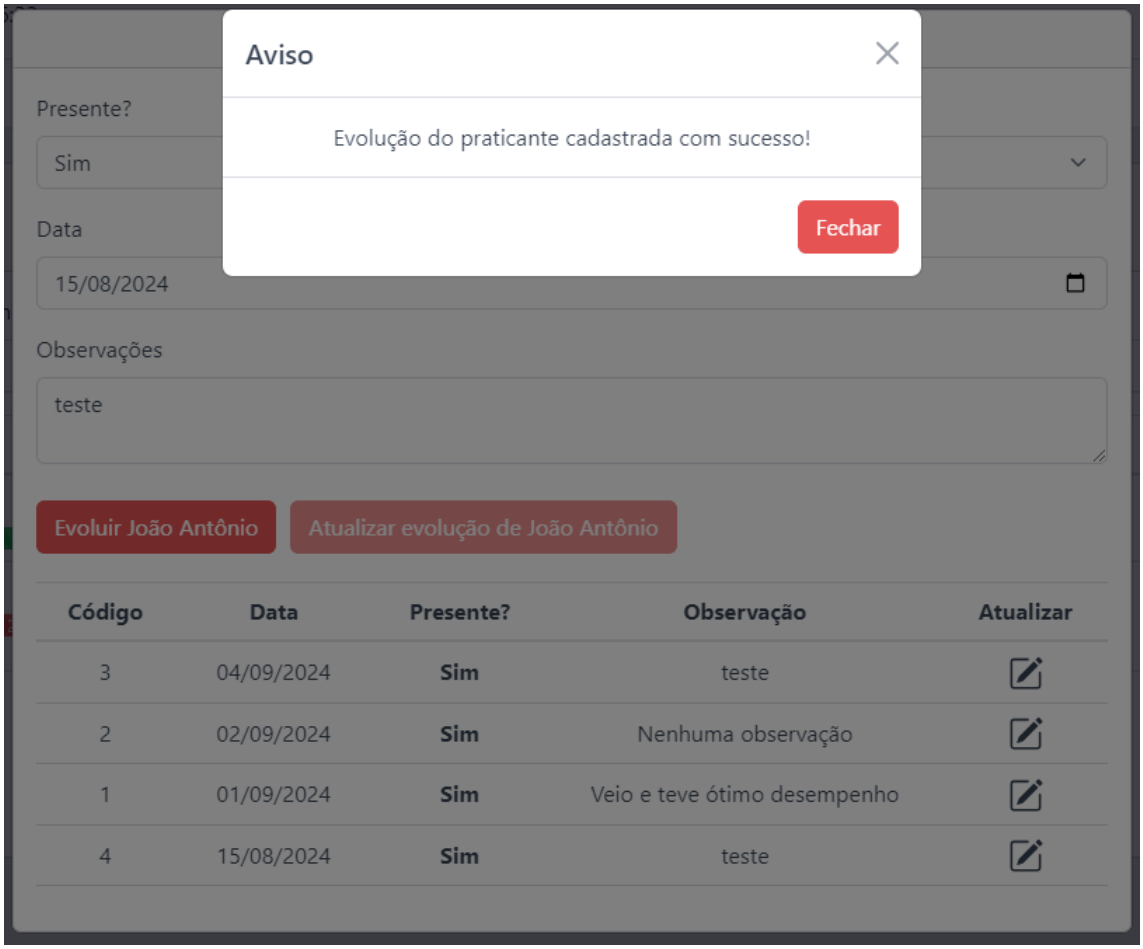


Figura 6.20: Sucesso ao evoluir praticante.

6.9 Gráfico de Linhas

A Figura 6.21, mostra o gráfico de linhas referente a evolução de um praticante fictício no período determinado. Para gerar o gráfico, é preciso informar as datas de busca (datas de início e fim) e após isso clicar no botão de pesquisa, indicado por um símbolo de lupa. Também é possível gerar um arquivo PDF deste gráfico ao clicar no botão de download indicado pelo símbolo de seta para baixo.

Figura 6.21: Gráfico de linhas.

6.10 Arquivo de PDF Gerado com Gráfico de Linhas

A Figura 6.22, mostra o PDF gerado para download ou impressão do gráfico de linhas após clicar do botão de download.

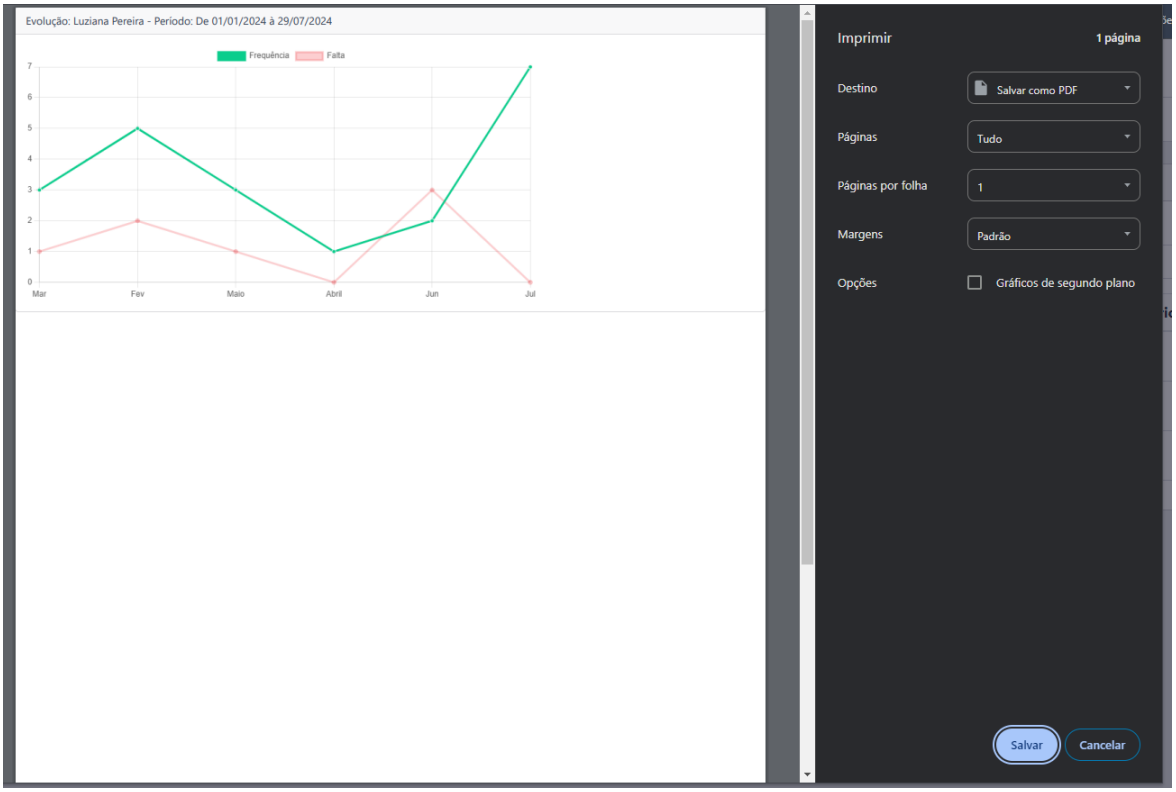


Figura 6.22: Arquivo PDF.

6.11 Gráfico de Barras

A Figura 6.23, mostra o gráfico de barras referente a evolução de um praticante fictício no período determinado. Para gerar o gráfico, é preciso informar as datas de busca (datas de início e fim) e após isso clicar no botão de pesquisa, indicado por um símbolo de lupa. Também é possível gerar um arquivo PDF deste gráfico ao clicar no botão de download indicado pelo símbolo de seta para baixo.

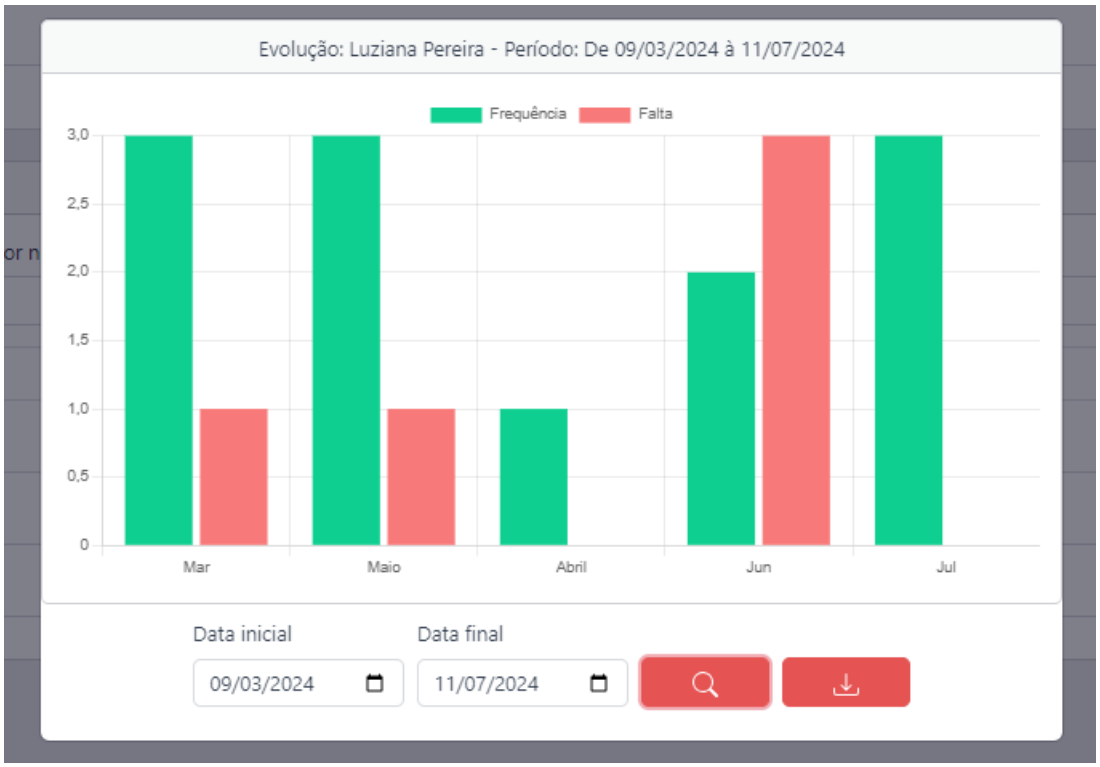


Figura 6.23: Gráfico de barras.

6.12 Arquivo de PDF Gerado com Gráfico de Barras

A Figura 6.24, mostra o PDF gerado para download ou impressão do gráfico de barras após clicar do botão de download.

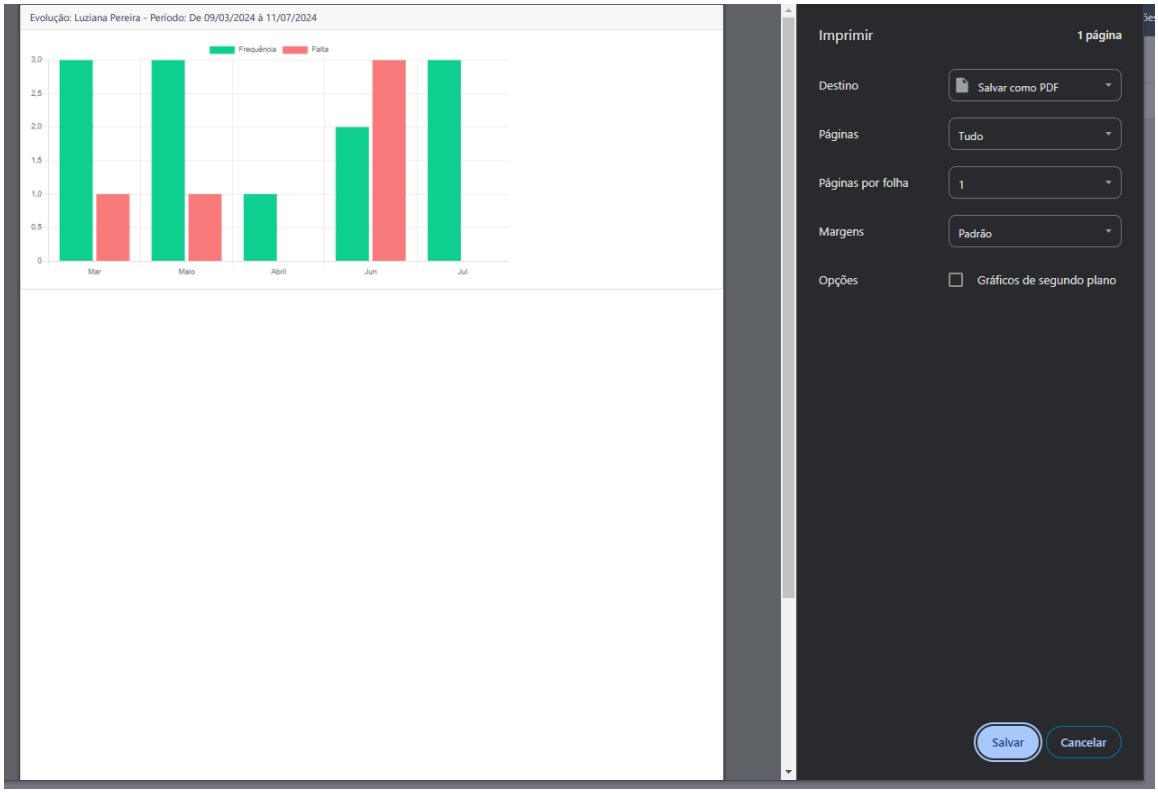


Figura 6.24: Arquivo PDF.

6.13 Gráfico de Torta

A Figura 6.25, mostra o gráfico de torta referente a evolução de um praticante fictício no período determinado. Para gerar o gráfico, é preciso informar as datas de busca (datas de início e fim) e após isso clicar no botão de pesquisa, indicado por um símbolo de lupa. Também é possível gerar um arquivo PDF deste gráfico ao clicar no botão de download indicado pelo símbolo de seta para baixo.

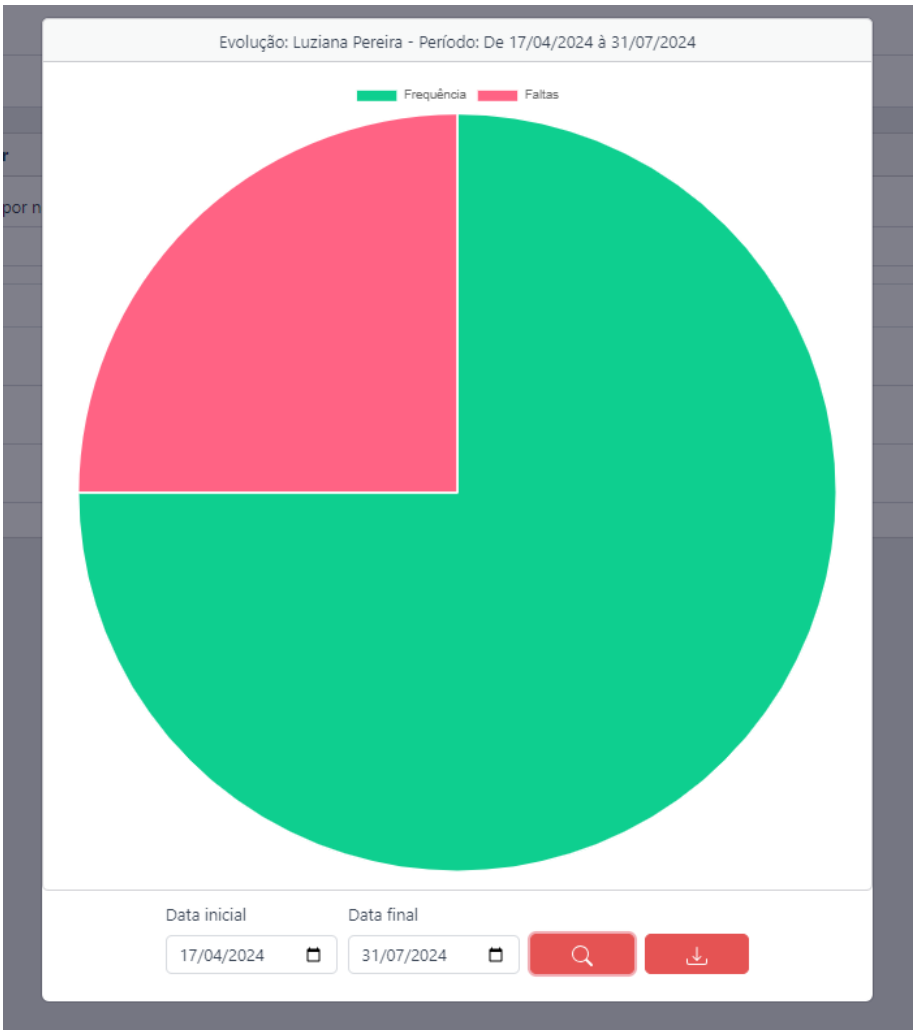


Figura 6.25: Gráfico de torta.

6.14 Arquivo de PDF Gerado com Gráfico de Torta

A Figura 6.26, mostra o PDF gerado para download ou impressão do gráfico de torta após clicar do botão de download.

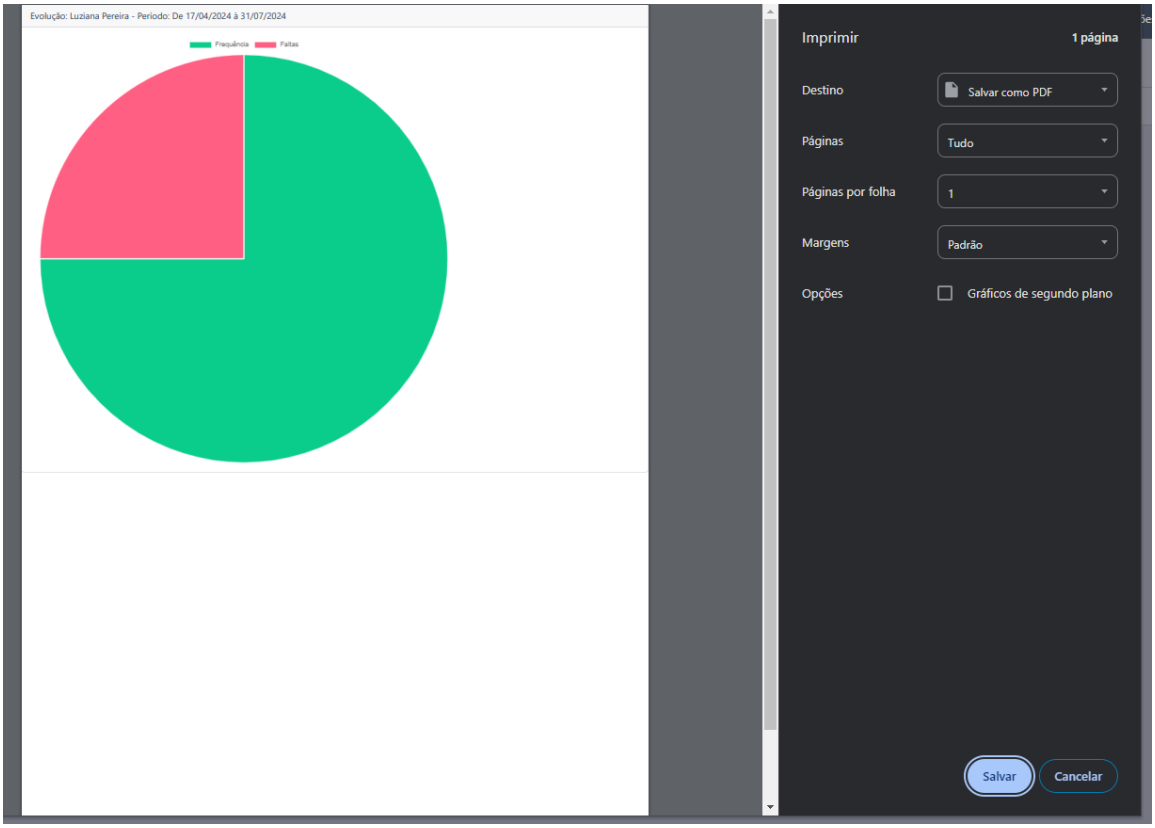
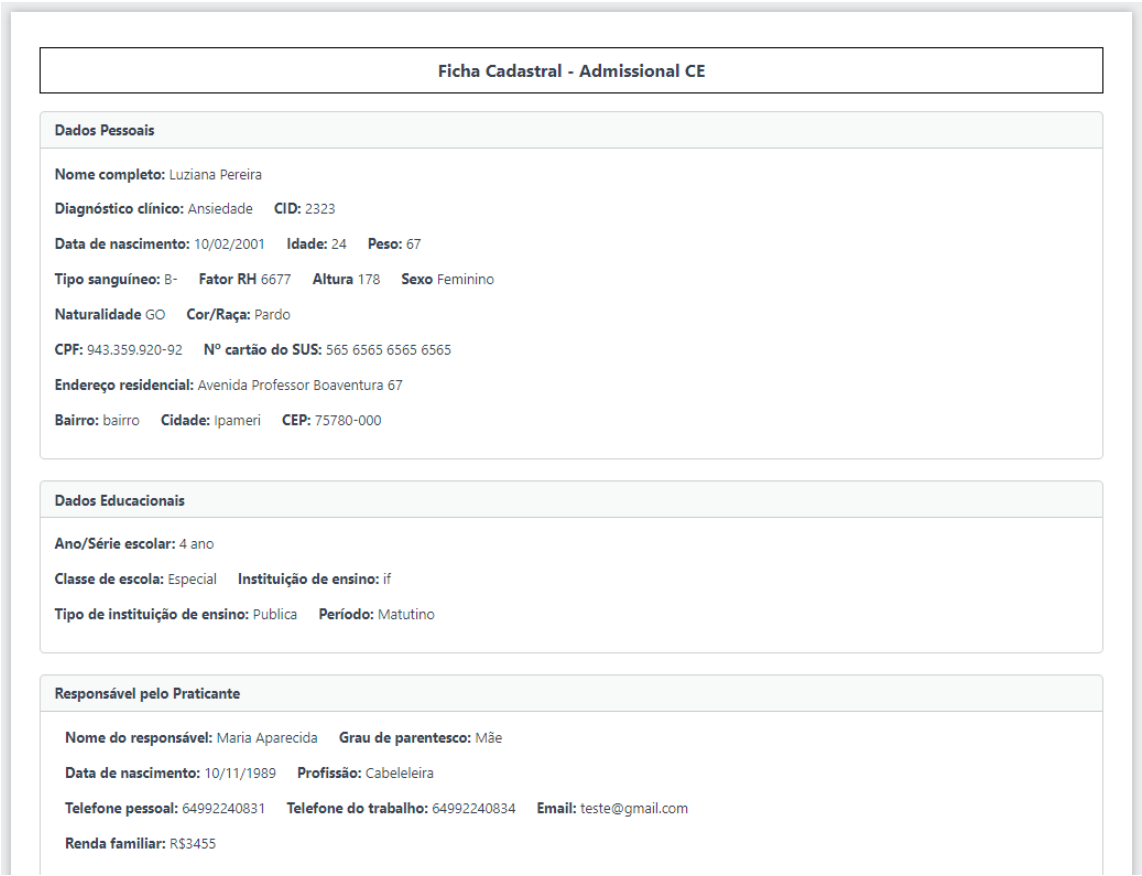


Figura 6.26: Arquivo PDF.

6.15 Relatório do Praticante

A Figura 6.27, mostra uma parte do relatório gerado de um praticante fictício.



Ficha Cadastral - Admissional CE

Dados Pessoais

Nome completo: Luziana Pereira

Diagnóstico clínico: Ansiedade **CID:** 2323

Data de nascimento: 10/02/2001 **Idade:** 24 **Peso:** 67

Tipo sanguíneo: B- **Fator RH** 6677 **Altura** 178 **Sexo** Feminino

Naturalidade GO **Cor/Raça:** Pardo

CPF: 943.359.920-92 **Nº cartão do SUS:** 565 6565 6565 6565

Endereço residencial: Avenida Professor Boaventura 67

Bairro: bairro **Cidade:** Ipameri **CEP:** 75780-000

Dados Educacionais

Ano/Série escolar: 4 ano

Classe de escola: Especial **Instituição de ensino:** if

Tipo de instituição de ensino: Publica **Período:** Matutino

Responsável pelo Praticante

Nome do responsável: Maria Aparecida **Grau de parentesco:** Mãe

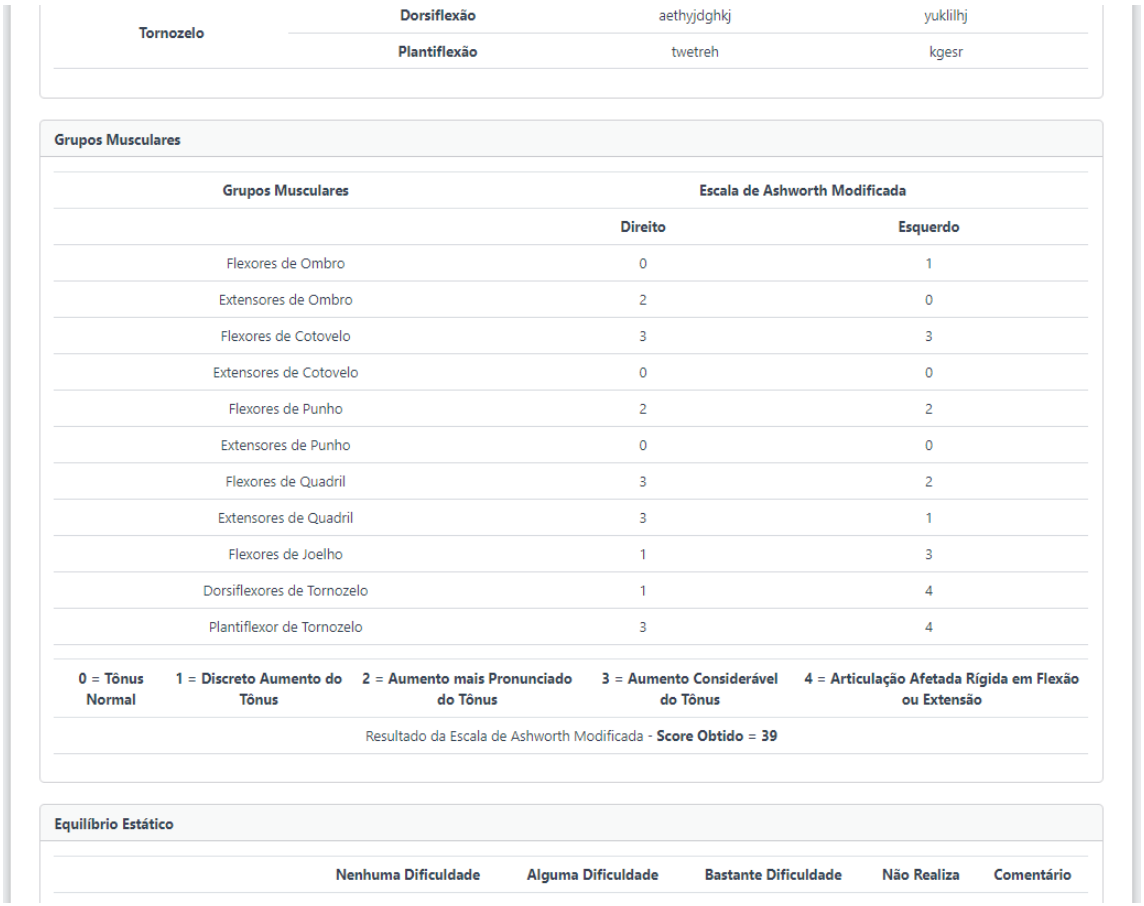
Data de nascimento: 10/11/1989 **Profissão:** Cabeleleira

Telefone pessoal: 64992240831 **Telefone do trabalho:** 64992240834 **Email:** teste@gmail.com

Renda familiar: R$3455

Figura 6.27: Parte do relatório.

A Figura 6.28, mostra uma parte do relatório gerado de um praticante fictício.



Tornozelo	**Dorsiflexão**	aethyjdghkj	yuklilhj
	Plantiflexão	twetreh	kgesr

Grupos Musculares

Grupos Musculares	**Escala de Ashworth Modificada**	
	Direito	**Esquerdo**
Flexores de Ombro	0	1
Extensores de Ombro	2	0
Flexores de Cotovelo	3	3
Extensores de Cotovelo	0	0
Flexores de Punho	2	2
Extensores de Punho	0	0
Flexores de Quadril	3	2
Extensores de Quadril	3	1
Flexores de Joelho	1	3
Dorsiflexores de Tornozelo	1	4
Plantiflexor de Tornozelo	3	4

0 = Tônus Normal	**1 = Discreto Aumento do Tônus**	**2 = Aumento mais Pronunciado do Tônus**	**3 = Aumento Considerável do Tônus**	**4 = Articulação Afetada Rígida em Flexão ou Extensão**

Resultado da Escala de Ashworth Modificada - **Score Obtido = 39**

Equilíbrio Estático

	Nenhuma Dificuldade	**Alguma Dificuldade**	**Bastante Dificuldade**	**Não Realiza**	**Comentário**

Figura 6.28: Parte do relatório.

A Figura 6.29, mostra o relatório de um praticante fictício gerado em arquivo PDF após clicar no botão.

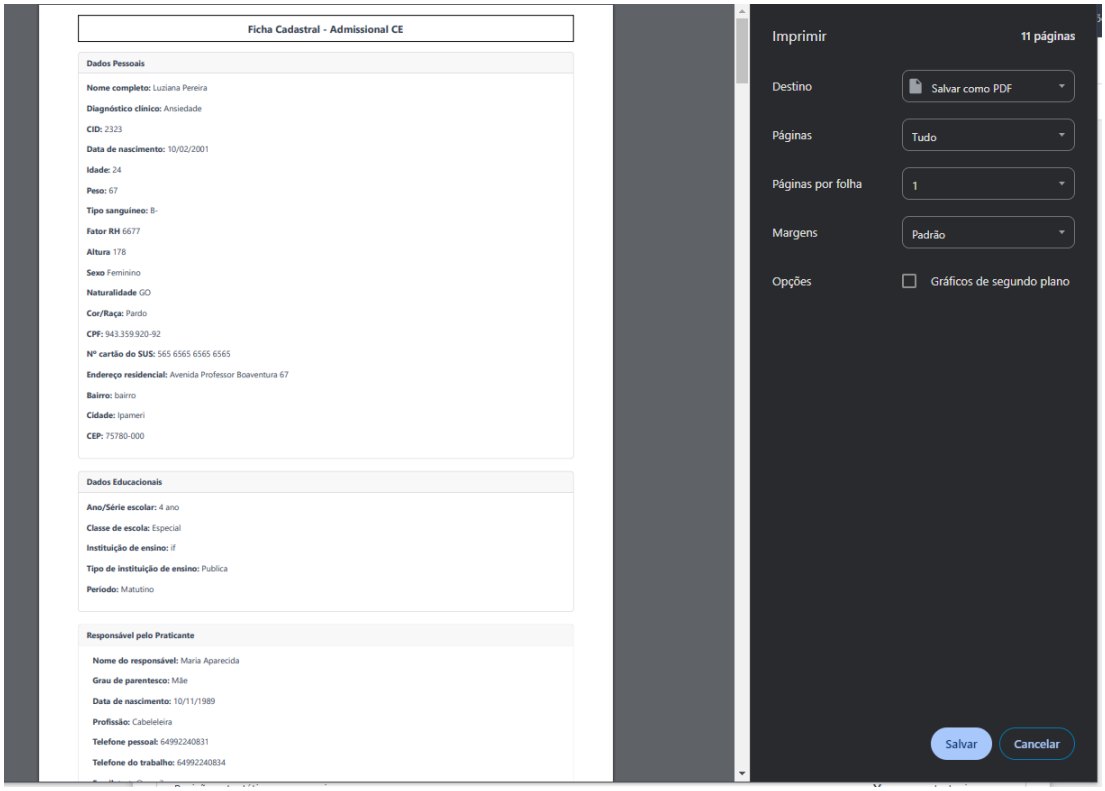


Figura 6.29: Relatório em PDF.

A Figura 6.30, mostra o formulário com dados fictícios para cadastrar os dados pessoais do praticante. Os demais formulários, seguem com o design semelhante.

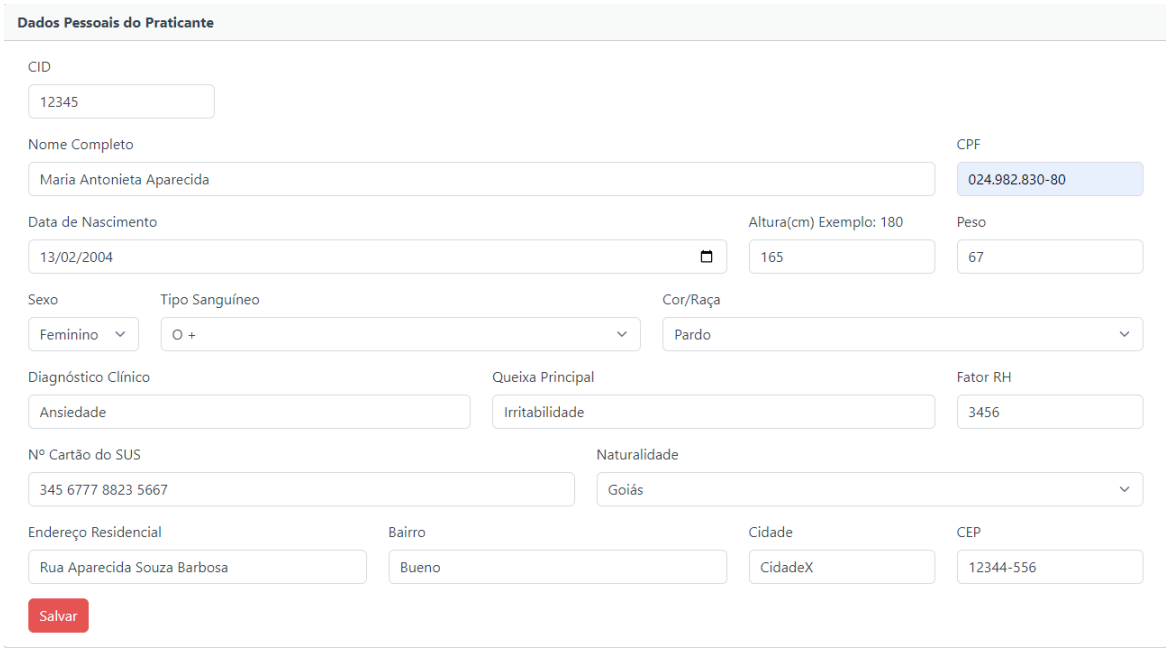


Figura 6.30: Formulário de cadastro de dados pessoais.

6.16 Tela de Listagem de Usuários

A Figura 6.31 mostra a página de visualização de todos os usuários cadastrados no sistema. A tabela exibe várias informações relevantes, como nome, e-mail, telefone, formação, nível, vínculo e possíveis ações, como atualização e exclusão de usuário.

Além disso, é possível clicar no e-mail do usuário para enviar uma mensagem. Ao clicar, será aberto o sistema de envio de e-mail do computador do usuário, permitindo que o usuário conclua o envio do e-mail.

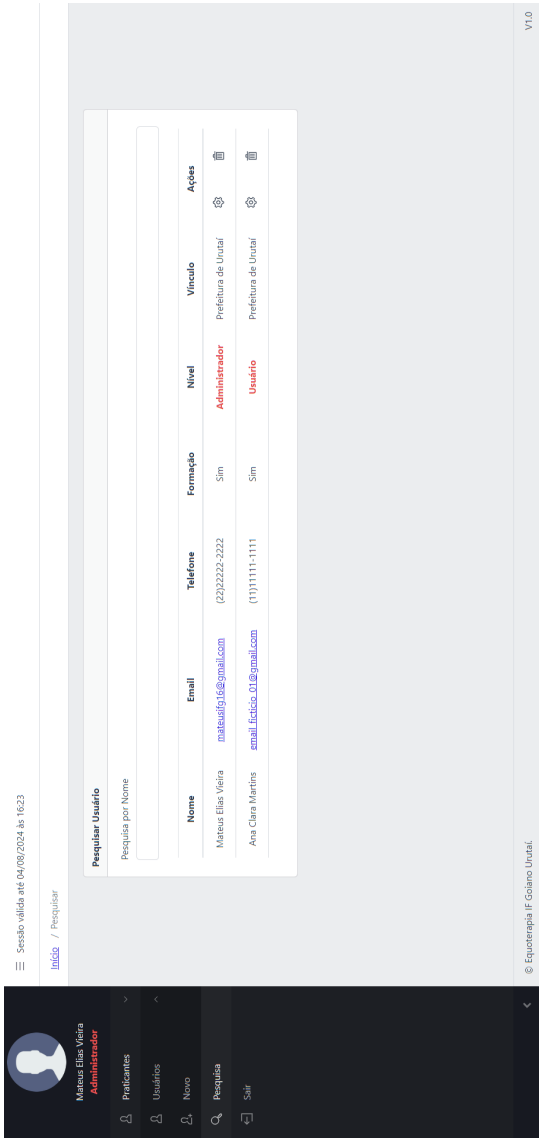


Figura 6.31: Tela inicial do administrador.

6.17 Formulário de Cadastro de Usuário

A Figura 6.32, mostra o formulário com dados fictícios para cadastrar os dados pessoais do novo usuário.

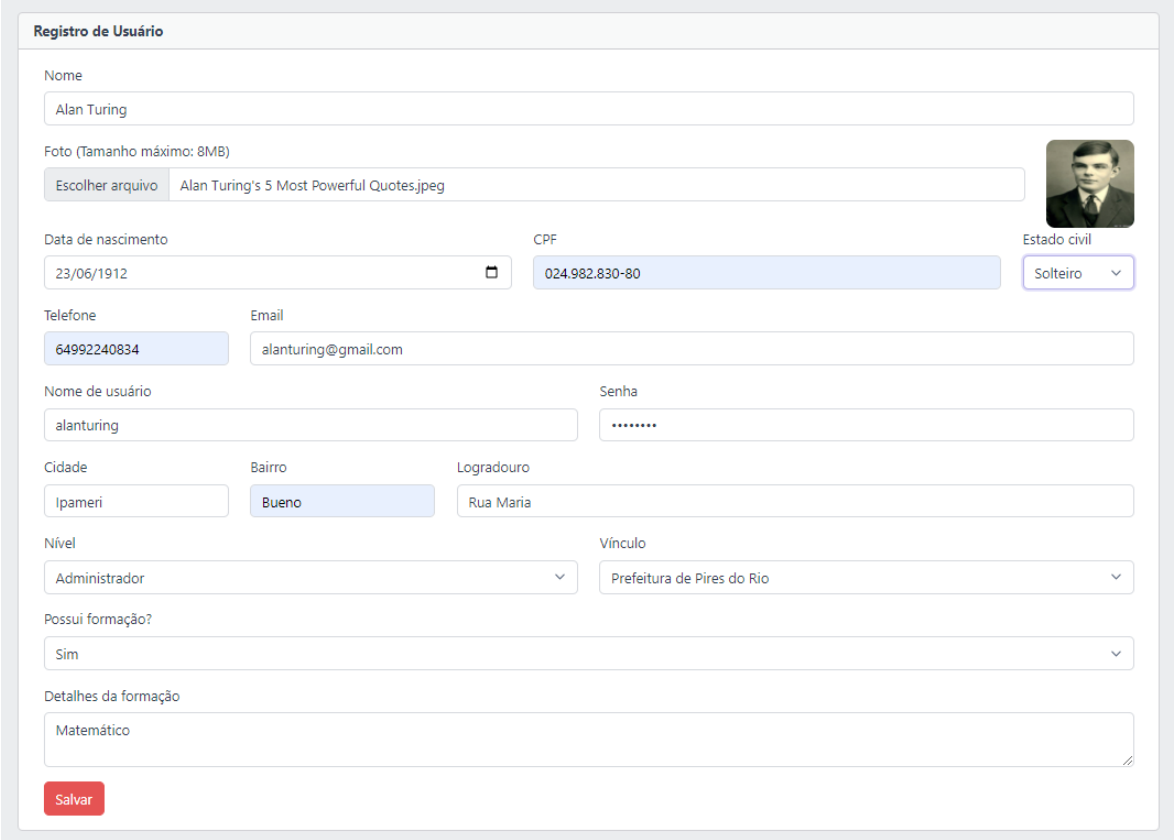


Figura 6.32: Formulário de cadastro de usuário.

A Figura 6.33, mostra uma mensagem de aviso sobre o CPF. O motivo é o fato de já existir na base de dados do sistema outro registro com o mesmo CPF, e por isso não pode ser cadastrado.

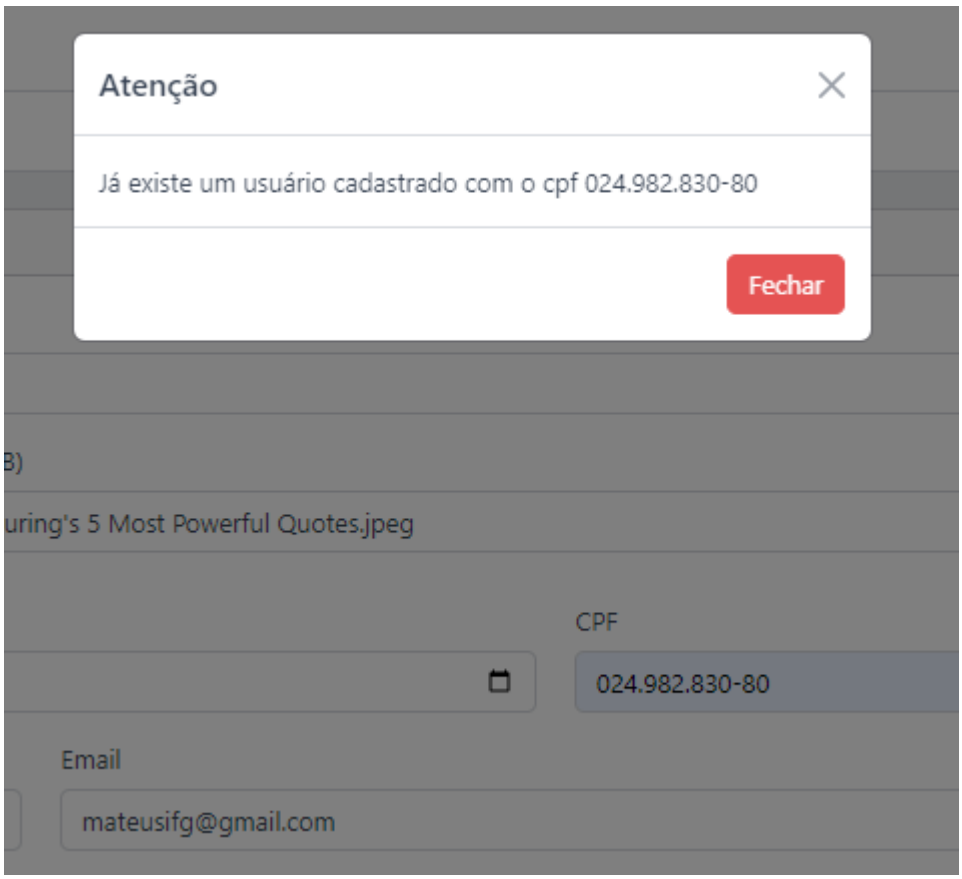


Figura 6.33: Falha de cadastro por CPF.

A Figura 6.34, mostra uma mensagem de aviso sobre o email. O motivo é o fato de já existir na base de dados do sistema outro registro com o mesmo email, e por isso não pode ser cadastrado.

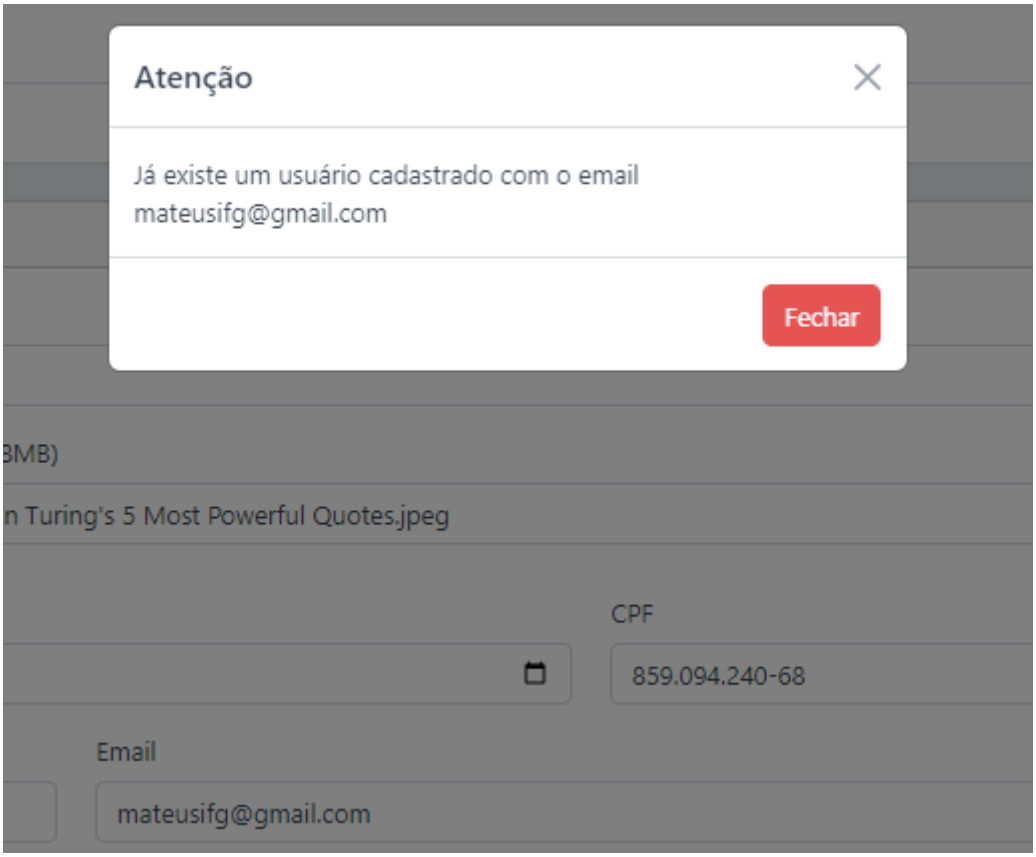


Figura 6.34: Falha de cadastro por email.

A Figura 6.35, mostra uma mensagem de aviso sobre o nome de usuário. O motivo é o fato de já existir na base de dados do sistema outro registro com o mesmo nome de usuário, e por isso não pode ser cadastrado.

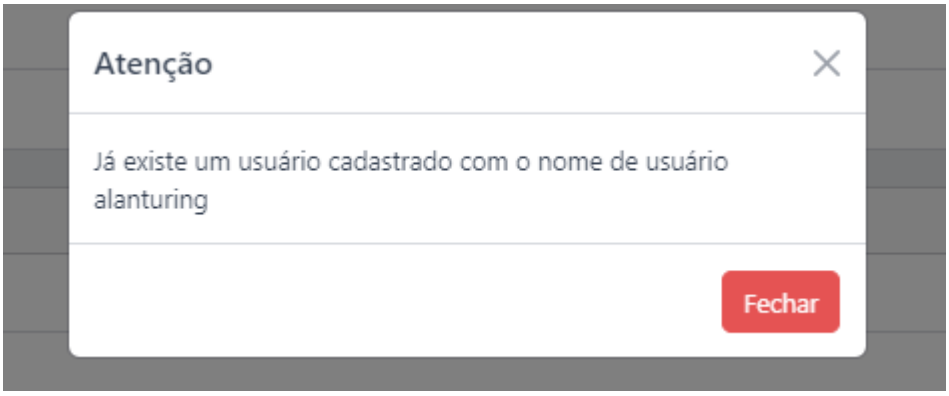


Figura 6.35: Falha de cadastro por nome de usuário.

A Figura 6.36, mostra uma mensagem de sucesso dizendo que o novo usuário foi cadastrado.

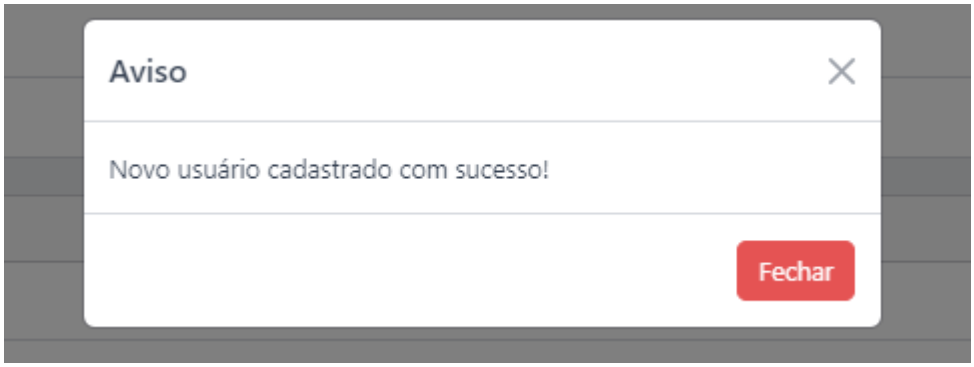


Figura 6.36: Cadastro de usuário realizado com sucesso.

6.18 Formulário de Atualização de Usuário

A Figura 6.37, mostra o formulário com dados fictícios para atualizar os dados pessoais do usuário.

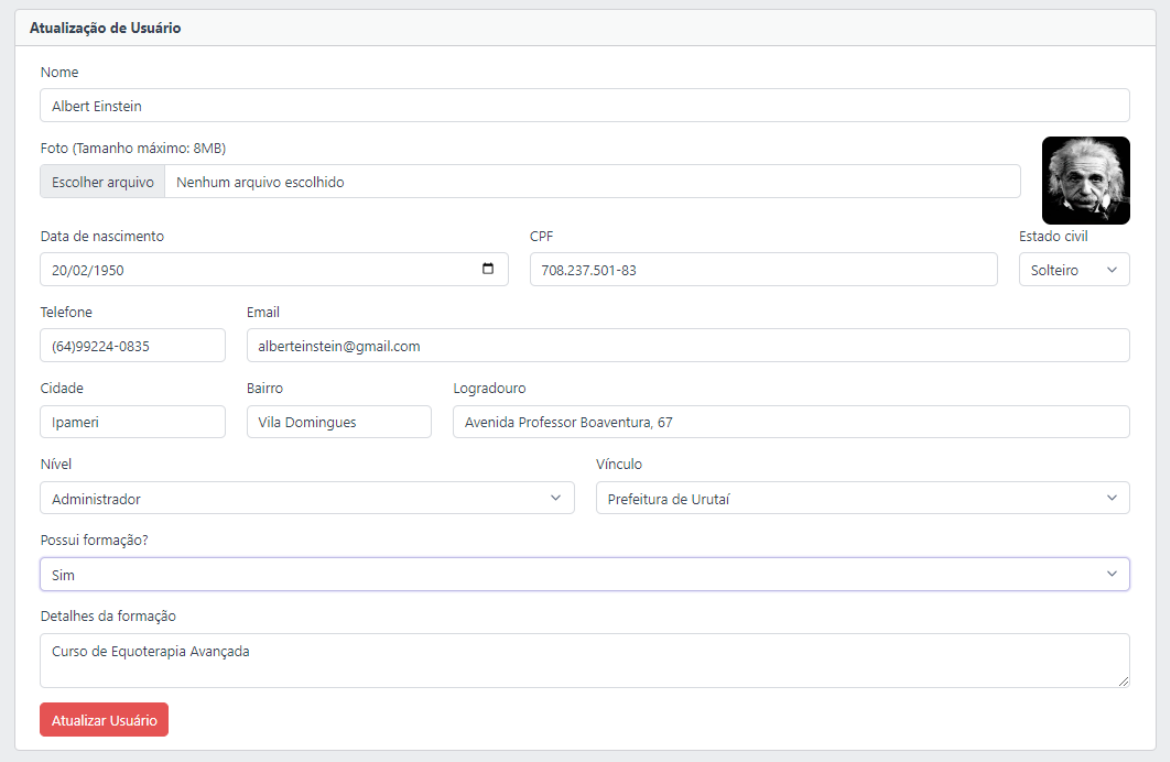


Figura 6.37: Atualização de usuário.

A Figura 6.38, mostra uma mensagem de sucesso dizendo que o usuário foi atualizado.

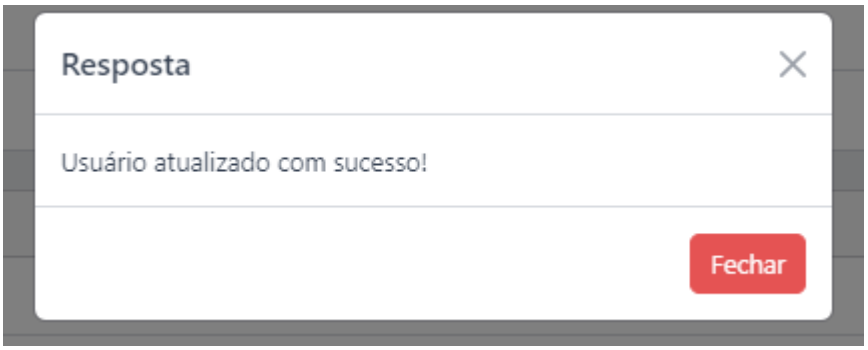


Figura 6.38: Atualização de usuário realizada com sucesso.

6.19 Exclusão de Usuário

A Figura 6.39, mostra uma mensagem perguntando se realmente deseja excluir o usuário. Caso o usuário confirme, o registro será excluído da base de dados. Vale ressaltar, que esta mensagem é uma segurança para evitar que algum usuário seja excluído acidentalmente.

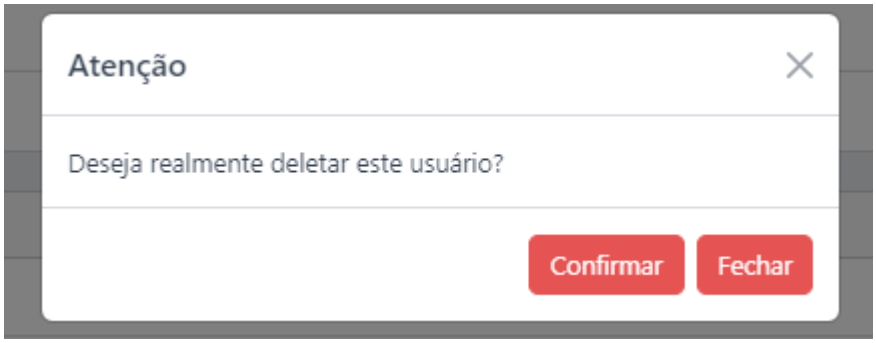


Figura 6.39: Exclusão de usuário.

A Figura 6.40, mostra uma mensagem de aviso, informando que o usuário foi deletado com sucesso.

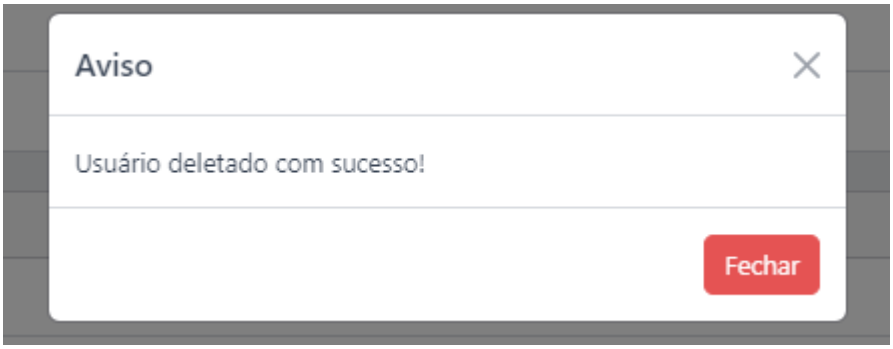


Figura 6.40: Usuário deletado.

CONCLUSÃO

Levando em consideração uma importante demanda da sociedade, este trabalho apresentou o desenvolvimento de um sistema visando a informatização da gestão do centro de equoterapia do Instituto Federal Goiano Campus Urutaí. Por consequinte, foi realizado: coleta e análise de requisitos, diagramação do sistema, desenvolvimento do front-end com React, desenvolvimento do back-end com Spring, implementação do banco de dados com MySql e testes de caixa preta (teste prático do sistema).

Na fase final do desenvolvimento, foi realizado validações juntamente ao representante da equipe gestora da equoterapia, que através de testes com inserções e buscas de dados fictícios na aplicação (no qual foi disponibilizada para testes em ambiente de produção em uma plataforma gratuita e de uso temporáraio https://equoterapia.vercel.app/dashboard), que em primeiro contato satisfez as expectativas da equipe de equoterapia.

Assim, o desenvolvimento e a implementação deste software, têm um grande potencial para causar um impacto significativo nas operações diárias da equipe de equoterapia, possibilitando também obter *insights* detalhados sobre o progresso dos praticantes, o que permitirá que a equipe se concentre mais no aspecto humano da terapia, aumentando a qualidade do atendimento e resultando em uma nova perspectiva de trabalho para a equipe. Isso inclui a eliminação de grandes quantidades de papéis, em um processo inovador, atualizado e preparado para lidar com grandes volumes de dados dos praticantes.

Considerando a importância da equoterapia como uma modalidade terapêutica abrangente e versátil, e a necessidade de inovação tecnológica para aprimorar os processos terapêuticos, o software desenvolvido se destaca. Ele não só otimiza a gestão dos praticantes, como também expande as possibilidades de pesquisa e desenvolvimento no campo da equoterapia. Este projeto estabelece uma base sólida para futuras investigações e melhorias, contribuindo para práticas terapêuticas mais eficazes e centradas no praticante. Além disso, oferece uma significativa contribuição para a sociedade e para indivíduos em situação de vulnerabilidade social que dependem da equoterapia para melhorar sua qualidade de vida.

6.20 Trabalhos Futuros

Como trabalhos futuros, poderá ser realizada a implementação de novas funcionalidades no sistema, como o cadastro de equinos utilizados no desenvolvimento dos praticantes, o acesso dos praticantes ao sistema com a possibilidade de terem uma conta pessoal para acompanhar as informações disponibilizadas pela equipe de equoterapia, e a adição de um blog ao projeto para que as pessoas possam acessar e acompanhar o trabalho da equipe. Além disso, seria interessante explorar a possibilidade de utilizar inteligência artificial para personalizar as sessões de terapia com base em análises de dados preditivos, adaptando as práticas de acordo com a evolução do praticante.

REFERÊNCIAS BIBLIOGRÁFICAS

ASTAH. *Astah User Guide*. Online: Astah, 2020. Disponível em: <https://astah.net/astah-user-guide>. Acesso em: 26 ago. 2024.

BERNERS-LEE, T. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. San Francisco, CA: HarperSanFrancisco, 1999. ISBN 978-0062515862.

BROWN, A. Modularity and flexibility in spring framework. *Journal of Software Engineering*, v. 22, n. 4, p. 123–138, 2019.

BROWN, M. Understanding unidirectional data flow in react applications. *Journal of Modern Web Architecture*, v. 12, n. 1, p. 85–100, 2021.

BáEZ, C. *React: Up and Running: Building Web Applications*. Sebastopol: O'Reilly Media, 2020.

CAMARGO, L. C. *Equoterapia: fundamentos e práticas*. São Paulo: Editora XYZ, 2014.

CONNOLLY, T. M.; BEGG, C. E. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Harlow: Pearson, 2014.

DEITEL, P.; DEITEL, H. *Java: Como Programar*. 10. ed. São Paulo: Pearson, 2017. ISBN 978-85-430-0469-6.

DOE, J. Reusable components in react: Building modular interfaces. *Journal of Web Development*, v. 15, n. 3, p. 45–60, 2020.

DUBOIS, P. *MySQL Cookbook*. Sebastopol: O'Reilly Media, 2016.

ELMASRI, R.; NAVATHE, S. B. *Fundamentals of Database Systems*. Boston: Pearson, 2015.

ERL, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ: Prentice Hall PTR, 2005. ISBN 9780131858589.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Boston, MA: Addison-Wesley, 2015. É no Backend onde se encontra a API e por sua vez o Web service.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, 2000. <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

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

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.]: Addison-Wesley, 2002.

FOWLER, M. *Continuous Integration*. martinfowler.com, 2006. Disponível em: <https://martinfowler.com/articles/continuousIntegration.html>.

FREEMAN, E.; ROBSON, E. *Head First HTML and CSS: A Learner's Guide to Creating Standards-Based Web Pages*. Sebastopol: O'Reilly Media, 2004.

GARCIA, M. The evolution of the spring framework community. *Software Development Review*, v. 14, n. 1, p. 45–58, 2021.

GITHUB, I. *GitHub: Where the World Builds Software*. Online: GitHub, Inc., 2020.

GOURLEY, D.; TOTTY, B. *HTTP: The Definitive Guide*. Sebastopol: O'Reilly Media, 2002.

HARRIS, O. Securing applications with spring security. *Journal of Application Security*, v. 18, n. 4, p. 34–47, 2020.

JOHNSON, D. Performance and scalability in spring applications. *Performance Engineering Journal*, v. 19, n. 3, p. 90–105, 2021.

JOHNSON, E. The growth of the react ecosystem: Community and resources. *Web Developer Monthly*, v. 8, n. 2, p. 25–38, 2021.

JOHNSON, J. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. San Francisco: Morgan Kaufmann Publishers, 2010.

JOHNSON, R. E. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. San Francisco: Morgan Kaufmann Publishers, 2003.

JONES, A. *Effective Software Design with UML*. San Francisco: SoftTech Publishing, 2019.

KIM, D. Maintaining modern react applications: Updates and best practices. *Enterprise Web Development Journal*, v. 19, n. 1, p. 34–49, 2022.

LEE, H. *Spring Boot and Microservices*. 2. ed. Sebastopol: O'Reilly Media, 2020. ISBN 978-1-4920-9348-1.

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.

MARTINEZ, C. Building efficient single page applications with react. *Web Application Design Quarterly*, v. 11, n. 2, p. 55–70, 2020.

RICHARDSON, L.; RUBY, S. *RESTful Web APIs*. Sebastopol, CA: O'Reilly Media, 2013.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. *Database System Concepts*. New York: McGraw-Hill, 2010.

SMITH, J. Optimizing web performance with virtual dom in react. *International Journal of Front-end Development*, v. 10, n. 4, p. 102–117, 2019.

SMITH, J. *Spring MVC: A Tutorial*. 1. ed. Birmingham: Packt Publishing, 2019. ISBN 978-1-78839-063-3.

SMITH, J. *UML and Software Design: A Practical Guide*. New York: TechPress, 2021.

TANENBAUM, A. S.; STEEN, M. V. *Distributed Systems: Principles and Paradigms*. Upper Saddle River: Pearson, 2007.

THOMPSON, L. The rise of react: Market demand and career opportunities. *Developer Trends*, v. 9, n. 3, p. 12–27, 2021.

TILKOV, S.; WOLF, I. *REST: From Research to Practice*. Berlin: Springer Science & Business Media, 2015.

TORVALDS, L. *Git: A Distributed Version Control System*. Online: Kernel.org, 2008.

WALL, C. et al. *Spring in Action*. New York: Manning Publications, 2021.

WALLS, C. *Spring in Action*. 5. ed. New York: Manning Publications, 2018. ISBN 978-1-61729-354-5.

WHITE, A. Jsx: Combining javascript and html in react development. *Journal of JavaScript Programming*, v. 14, n. 3, p. 77–92, 2018.

WHITE, B. Integrating technologies with spring. *Tech Integration Quarterly*, v. 16, n. 2, p. 67–80, 2020.

World Wide Web Consortium (W3C). *XML Inclusions (XInclude) Version 1.0*. [S.l.], 2004. W3C Recommendation. Disponível em: <https://www.w3.org/TR/xinclude/>.