UNIVERSIDADE FEDERAL DE SANTA CATARINA

APLICAÇÃO WEB PARA GERAÇÃO DE LAUDOS MÉDICOS

Fernando Longo Beduin

FLORIANÓPOLIS - SC

2023/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Aplicação web para geração de laudos médicos

Fernando Longo Beduin

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.
Orientador: Prof. Frank Augusto Siqueira

FLORIANÓPOLIS - SC
2023/1

AGRADECIMENTOS

Em primeiro lugar, gostaria de expressar meu profundo agradecimento à minha mãe, Eletícia Longo, que sempre foi minha maior incentivadora e fonte de inspiração. Sua dedicação, apoio incondicional e amor inabalável foram fundamentais para que eu chegasse até aqui. Sua presença constante em minha vida é um presente que valorizo imensamente.

À minha esposa, Mariana Horn André, agradeço por todo o amor, compreensão e suporte durante esse período desafiador. Você foi meu porto seguro, sempre me encorajando e acreditando em mim, mesmo nos momentos em que eu duvidava de mim mesmo. Sua paciência e incentivo foram essenciais para que eu persistisse e alcançasse meus objetivos.

Meu melhor amigo, Flávio Fusuma, merece uma menção especial, pois esteve ao meu lado desde o início dessa (longa) jornada acadêmica. Agradeço por todas as conversas, amizade, parceria e inspiração ao longo desses anos. O teu apoio foi fundamental para superar esse grande desafio.

Agradeço profundamente ao meu orientador, Frank Augusto Siqueira, pelo seu acolhimento e sua orientação excepcional. Seu conhecimento, dedicação e disponibilidade contribuíram significativamente para o aprimoramento e conclusão deste trabalho. Obrigado pela oportunidade de aprender com você e por todo o suporte que me ofereceu.

Por fim, gostaria de expressar minha gratidão a todas as pessoas que não foram mencionadas individualmente, mas que têm um papel significativo nessa jornada, como aos demais amigos e familiares.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BSON	Binary Javascript Object Notation
CRM	Conselho Regional de Medicina
CSS	Cascading Style Sheets
DAO	Data Access Object
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
JWT	JSON Web Token
LGPD	Lei Geral de Proteção de Dados
MS	Ministério da Saúde
NPM	Node Package Manager
SGBD	Sistema de Gerenciamento de Banco de Dados
SO	Sistema Operacional
SPA	Single Page Application

RESUMO

Este trabalho apresenta o desenvolvimento do *laudosweb*, um aplicativo web para a geração de laudos médicos que auxilie na criação destes documentos numa clínica médica de maneira prática e padronizada - oferecendo uma alternativa eficiente em contrapartida à geração de laudos de forma manual. A solução proposta consiste em uma ferramenta que utiliza templates pré-definidos de texto organizados por categorias para facilitar a seleção de itens e garantir eficiência na padronização dos laudos. Além disso, os documentos gerados são disponibilizados por e-mail para os pacientes e armazenados em banco de dados. As principais tecnologias utilizadas para o desenvolvimento foram o framework Vue.js para a camada de apresentação, o ambiente de execução Node.js para a camada de aplicativo na forma de uma API REST, além do banco de dados NoSQL MongoDB.

Palavras-chave: Aplicação web. Laudos médicos. Node.js. Vue.js. API REST.

ABSTRACT

This paper presents the development of *laudosweb*, a web application for generating medical reports that aims to facilitate the creation of these documents in a medical clinic in a practical and standardized manner, offering an efficient alternative to manual report generation. The proposed solution is a tool that utilizes pre-defined text templates organized by categories to facilitate item selection and ensure efficiency in report standardization. Additionally, the generated documents are made available to patients via email and stored in a database.The main technologies used for development include the Vue.js framework for the presentation layer, the Node.js runtime environment for the application layer in the form of a REST API, and the MongoDB NoSQL database.

Keywords: Web application. Medical reports. Node.js. Vue.js. REST API.

1 Introdução

O setor da saúde vem passando por constantes transformações tecnológicas, buscando sempre aperfeiçoar seus processos e oferecer um atendimento de qualidade aos pacientes. De acordo com a Estratégia de Saúde Digital para o Brasil do Ministério da Saúde, “a incorporação de tecnologias, métodos, modelos e processos inovadores é a essência das transformações que o mundo vive hoje.” [MS 2020] Nesse contexto, a utilização de aplicações web tem se tornado cada vez mais relevante para otimizar e agilizar as atividades desenvolvidas em clínicas médicas.

A geração de laudos médicos é uma etapa fundamental no processo de diagnóstico e tratamento dos pacientes. No entanto, muitas clínicas ainda seguem métodos tradicionais como a confecção manual em papel ou em sistemas baseados em processos manuais, enfrentando assim desafios relacionados à lentidão e falta de padronização na confecção de laudos. Essas limitações impactam a eficiência dos serviços de saúde e podem comprometer a qualidade do atendimento oferecido aos pacientes.

A aplicação web proposta neste trabalho busca superar esses obstáculos, oferecendo um sistema de fácil utilização que permite aos médicos registrar, editar e compartilhar laudos de forma rápida e precisa. Ainda, a aplicação promove a padronização dos laudos, garantindo maior qualidade e consistência nas informações geradas.

1.1 Objetivo geral

Este Trabalho de Conclusão de Curso visa desenvolver uma aplicação web que permita a geração de laudos com base em trechos de texto pré definidos, chamados de templates, de acordo com o(s) tipo(s) de exame(s) realizado(s) na área de atuação específica de uma clínica médica.

1.2 Objetivos específicos

São objetivos específicos deste trabalho:

- Desenvolver uma aplicação utilizando algumas das mais recentes tecnologias web disponíveis;

- Realizar a integração da aplicação a uma base externa, da qual serão obtidos dados dos exames realizados em uma clínica médica;

- Desenvolver uma aplicação que seja de fácil customização por qualquer clínica, independentemente da área de atuação da mesma.

1.3 Metodologia

Este trabalho adotará uma abordagem que compreenderá as seguintes etapas: levantamento de requisitos, definição da arquitetura, implementação da solução tecnológica e avaliação.

Na primeira etapa, será realizado o levantamento de requisitos baseados nas necessidades e expectativas do corpo médico de uma clínica médica. Tais requisitos serão documentados de forma clara e precisa para orientar a sequência do projeto.

Em seguida, será definida a arquitetura do sistema. Serão considerados os requisitos levantados, bem como as melhores práticas no desenvolvimento de aplicações web. Além disso, serão definidas as tecnologias a serem utilizadas, como linguagem de programação, frameworks, banco de dados e outras ferramentas relevantes. A escolha da arquitetura terá como princípio o desenvolvimento de uma solução que seja escalável e segura.

Com a arquitetura definida, dar-se-á início à implementação do aplicativo. Nessa etapa serão apresentados detalhes da modelagem de dados e do desenvolvimento das camadas de aplicação e apresentação do sistema.

Por último, será feita a avaliação do uso da ferramenta numa clínica médica real, permitindo que os médicos utilizem a aplicação no dia a dia. Durante esse período, será coletado o feedback dos usuários a fim de avaliar a usabilidade e eficiência da solução tecnológica.

1.4 Estrutura do Texto

Este trabalho está organizado em seis capítulos. Este primeiro capítulo introduziu o problema e a necessidade da solução a ser apresentada, assim como objetivos e metodologia do trabalho. O capítulo 2 apresenta a fundamentação tecnológica do sistema, explicando resumidamente os principais conceitos relacionados às tecnologias aplicadas. O terceiro capítulo descreve os trabalhos relacionados, comparando-os à solução proposta. No capítulo 4 é apresentado o detalhamento da solução proposta, incluindo detalhes de implementação. O capítulo 5 apresenta um caso de uso adaptado do **laudosweb** em uma clínica médica. Por fim, são apresentadas as conclusões do trabalho e perspectivas de evolução e aprimoramento da aplicação.

2 Fundamentação tecnológica

Este capítulo apresenta a fundamentação dos conceitos e tecnologias aplicados no desenvolvimento da solução proposta neste projeto. Serão abordados os seguintes tópicos: aplicações web, arquitetura de aplicações web, Javascript, Vue.js, Node.js e MongoDB.

2.1 Aplicações Web

De acordo com a definição fornecida pela Webopedia (2023), uma aplicação web é uma aplicação baseada na web que pode ser acessada por meio de um navegador web. Tais aplicações são projetadas para serem acessadas de qualquer dispositivo com conexão à internet, oferecendo flexibilidade e comodidade aos usuários.

Uma das principais vantagens desse tipo de aplicação é fornecer uma interface intuitiva e de fácil uso, permitindo o acesso a todo tipo de público. Outra importante característica é a atualização centralizada, podendo disponibilizar novos recursos aos usuários sem a necessidade de atualizar o software individualmente em cada dispositivo. [AWS 2023]

2.2 Arquitetura de aplicações web

Ao desenvolver aplicações web, é importante escolher uma arquitetura adequada que garanta qualidades como eficiência, escalabilidade e segurança. De acordo com a IBM (2023), a arquitetura em três camadas é um modelo amplamente utilizado no desenvolvimento de aplicações web para garantir os três requisitos citados acima.

As três camadas – camada de apresentação (*view*), camada de aplicativo (*control*) e camada de dados (*model*) – serão descritas na sequência. Podemos ver na Figura 2.1 um modelo simplificado dessa arquitetura. A comunicação entre *view* e *control* ocorre através de requisições e respostas HTTP, enquanto a persistência dos dados é feita entre o *control* e o *model*.

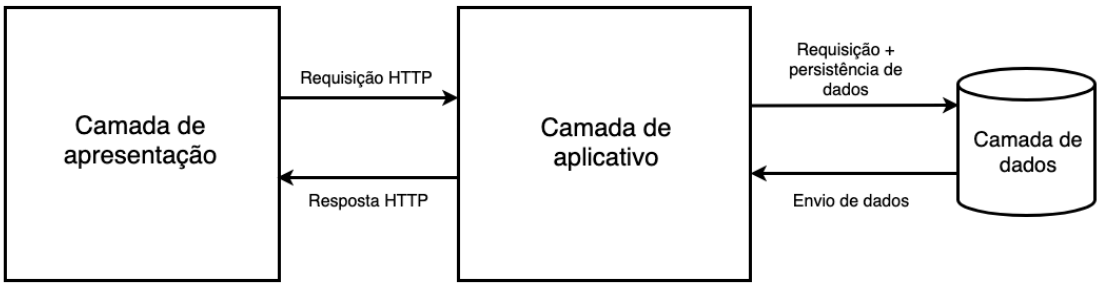


Figura 2.1: Exemplo de arquitetura web

Fonte: Autor

2.2.1 Camada de apresentação

A camada de apresentação é a camada apresentada no navegador Web, a partir da qual o usuário interage com o sistema. Podemos dividi-la em 3 partes: conteúdo (HTML), aparência (CSS) e interação (Javascript). O uso de frameworks modernos e reativos, como Vue.js e React, agiliza o desenvolvimento dessa camada, organizando cada uma das 3 partes descritas acima.

2.2.2 Camada de aplicativo

A camada de aplicativo é a parte do sistema que fica hospedada no servidor da aplicação, responsável pela implementação e orquestração da lógica de negócio, interação com banco de dados e acesso a APIs. Pode ser desenvolvida utilizando diversas linguagens de programação, como PHP, Javascript, C#, Python, Java entre outras.

2.2.3 Camada de dados

A camada de dados é responsável pelo armazenamento dos dados da aplicação, empregando algum Sistema de Gerenciamento de Banco de Dados (SGBD), como por exemplo MySQL, PostgreSQL, SQL Server, MongoDB, MariaDB, etc.

2.3 Javascript

Javascript é uma linguagem de programação estruturada, interpretada e de multiparadigma que forma, juntamente com o HTML e o CSS, um dos tripés tecnológicos no qual é sustentada a World Wide Web. Foi originalmente concebida em 1995 apenas como uma linguagem de script responsável por tornar o HTML, única ferramenta disponível para a construção de sites, mais dinâmico. Destes tempos até os dias de hoje, ganhou popularidade e tornou-se a linguagem padrão para o desenvolvimento web - não somente no desenvolvimento *client-side*, como também no lado do servidor através do popular ambiente de execução Node.js. [Mozilla 2023] [Brasil.js 2023]

Segundo a mais recente pesquisa realizada pelo Stack Overflow, um dos mais populares sites de perguntas e respostas na área da computação, Javascript segue sendo a linguagem de programação mais utilizada atualmente, pelo décimo ano consecutivo, conforme gráfico na Figura 2.2. [Stack Overflow 2022]

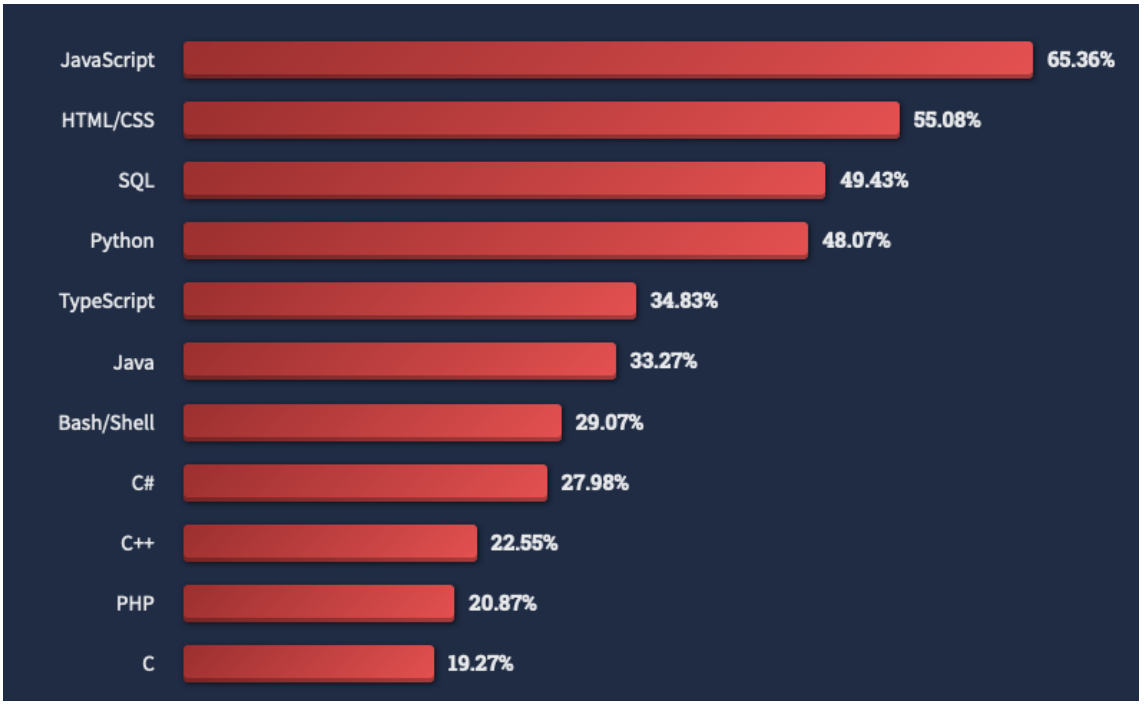


Figura 2.2: Lista de linguagens mais populares.
Fonte: [Stack Overflow 2022]

Uma das características mais notáveis do Javascript é a capacidade de manipular o DOM (*Document Object Model*), que representa o documento exibido pelo navegador Web, permitindo a alteração de elementos dinamicamente. Tais alterações incluem adição e remoção de elementos, alteração de estilos e manipulação de eventos. Nesse sentido, a linguagem possui um grande ecossistema de bibliotecas e frameworks que facilitam o desenvolvimento e otimizam o seu desempenho, como jQuery, React.js, Angular e Vue.js.

2.4 Vue.js

O Vue.js (2023) é um framework Javascript progressivo que tem como finalidade a construção de interfaces de usuário reativas com base em uma arquitetura de componentes reutilizáveis e dinâmicos. Tais características permitem a criação de SPAs (Single Page Applications, ou aplicações de página única), que se notabilizam por atualizar dinamicamente o conteúdo de uma página sem precisar carregá-la por inteiro a cada interação do usuário. Requisições dinâmicas são feitas para o servidor em segundo plano, normalmente quando demandado pelo usuário, para adicionar conteúdo sensível ao domínio da aplicação. Este tipo de framework é amplamente utilizado no desenvolvimento web hoje em dia, a exemplo de outras ferramentas similares como o React.js e o Angular.

O sistema de renderização do Vue.js é baseado em um DOM virtual, permitindo que o framework otimize a sua manipulação em memória e carregue o conteúdo dos componentes no DOM da página de forma mais eficiente. Dessa forma, o desenvolvedor pode focar em escrever os componentes de forma declarativa, deixando a parte **reativa** e a renderização a cargo do Vue. A Figura 2.3 ilustra esse processo, onde o template do componente é compilado em uma função de renderização responsável por gerar o DOM virtual e criar o DOM real a partir dele.

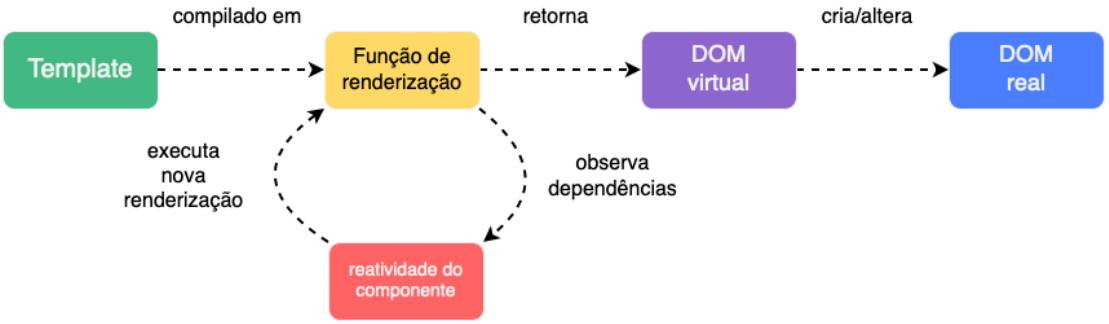


Figura 2.3: Ciclo de renderização de um componente Vue.js
Fonte: adaptado e traduzido de [Vue.js 2023]

A estrutura de um componente Vue.js é normalmente dividida em 3 partes, equivalentes ao tripé tecnológico citado anteriormente (HTML, CSS e Javascript) - dentro das tags *template*, *style* e *script,* respectivamente. O template é a parte visual do componente, onde são definidos estrutura e layout do documento a ser apresentado ao usuário. A sintaxe é similar à do HTML, contendo também elementos de interpolação e eventos. No exemplo mostrado na Figura 2.4, temos um componente responsável por exibir um botão que, ao ser clicado, incrementa o contador que é exibido como parte do texto do mesmo botão. O contador (variável *count*) é uma variável reativa atualizada automaticamente sempre que o seu valor é modificado.

```vue
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Figura 2.4: Exemplo de um componente Vue.js
Fonte: [Vue.js 2023]

Por ser um framework bem amadurecido e com amplo uso pela comunidade de desenvolvedores web, o Vue.js possui uma ampla gama de bibliotecas e componentes de terceiros disponíveis para a utilização em novos projetos.

2.5 Node.js

O Node.js (2023) é um ambiente multiplataforma que permite a execução de código Javascript no servidor. Ele utiliza a *engine* de Javascript *V8*, a mesma desenvolvida pelo Google para ser utilizada dentro do navegador Chrome. Uma das características que o diferencia de outras soluções é a sua execução de eventos numa única thread, chamada *Event Loop*, que permite ao Node.js tratar várias solicitações simultaneamente - como pode ser visto na Figura 2.5 abaixo. Operações bloqueantes, como por exemplo leitura de arquivos, acesso a banco de dados ou outras operações com maior demanda do SO, são

gerenciadas de maneira assíncrona pela biblioteca *libuv,* e informadas ao *Event Loop* assim que concluídas para que este possa notificar o Node.js.

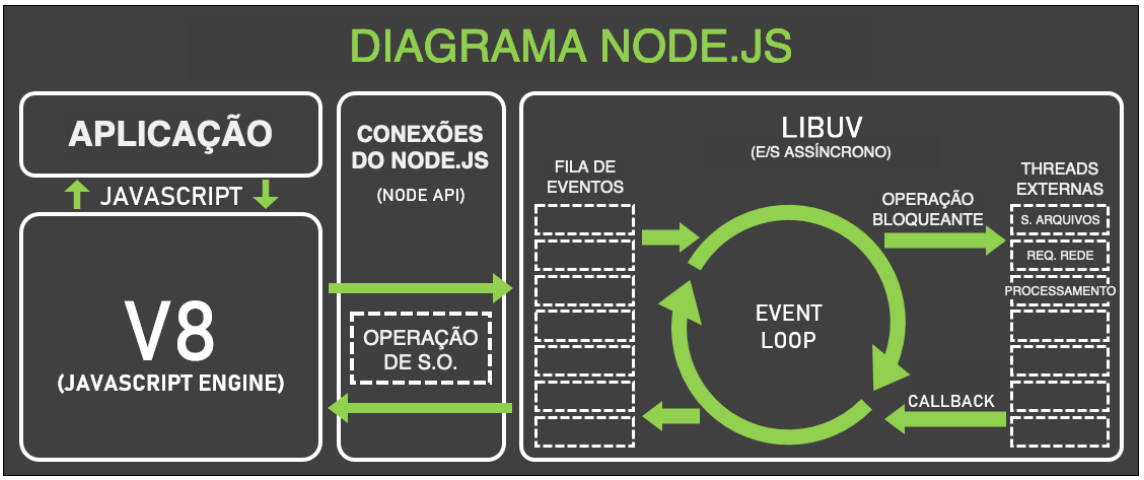


Figura 2.5: Diagrama da arquitetura de funcionamento do Node.js
Fonte: adaptado e traduzido de [Tutorial and Example 2020]

Essa dinâmica permite que o Node.js seja altamente escalável e eficiente em termos de uso de recursos. Em vez de alocar uma thread separada para cada solicitação, o Node.js pode lidar com várias solicitações usando uma única thread, aproveitando ao máximo os recursos do sistema.

Além disso, o Node.js possui um vasto ecossistema de pacotes e bibliotecas, gerenciado pelo NPM (*Node Package Manager*), facilitando a inclusão de funcionalidades adicionais em projetos. Isso permite que os desenvolvedores aproveitem as bibliotecas existentes e acelerem o processo de desenvolvimento.

2.6 MongoDB

O MongoDB (2016) é um banco de dados NoSQL orientado a documentos amplamente usado em aplicativos web. Ao contrário dos bancos de dados relacionais, ele armazena seus dados em um formato de documentos semelhante ao JSON (BSON), facilitando desta forma a manipulação e o armazenamento não-estruturado. Essa abordagem lhe confere uma maior flexibilidade, tornando-o uma opção mais adequada a aplicações web que lidam com dados variáveis e complexos.

Uma das principais vantagens do MongoDB é a escalabilidade horizontal. Como ele oferece suporte à distribuição dos dados em múltiplos servidores, permite que aplicativos web redimensionem o armazenamento e o desempenho à medida que o tráfego e as demandas de dados aumentam. Além disso, possui recursos avançados de indexação e consultas que conferem uma opção moderna e atraente para o desenvolvimento de aplicações web robustas e de alto desempenho.

Outra grande vantagem, principalmente no contexto de desenvolvimento em Node.js (que é o caso deste projeto), é a utilização do JavaScript como linguagem de consulta no MongoDB, permitindo que os dados sejam manipulados de forma intuitiva e familiar ao utilizar a mesma linguagem de programação.

3 Trabalhos relacionados

Neste capítulo serão apresentadas diferentes ferramentas disponíveis no mercado que possuem características e objetivos relacionados a este trabalho, com o propósito de realizar uma análise comparativa entre eles e o aplicativo proposto neste projeto.

3.1 Turing

A empresa Queo Tecnologia (2023), desenvolvedora da ferramenta Turing, promete em sua página agilidade, facilidade e alto grau de customização na confecção de laudos médicos. Apesar de ser disponibilizada como um serviço, a aplicação é disponibilizada apenas numa versão instalável para Windows, conferindo menor flexibilidade ao médico quanto à elaboração dos documentos a partir de qualquer computador.

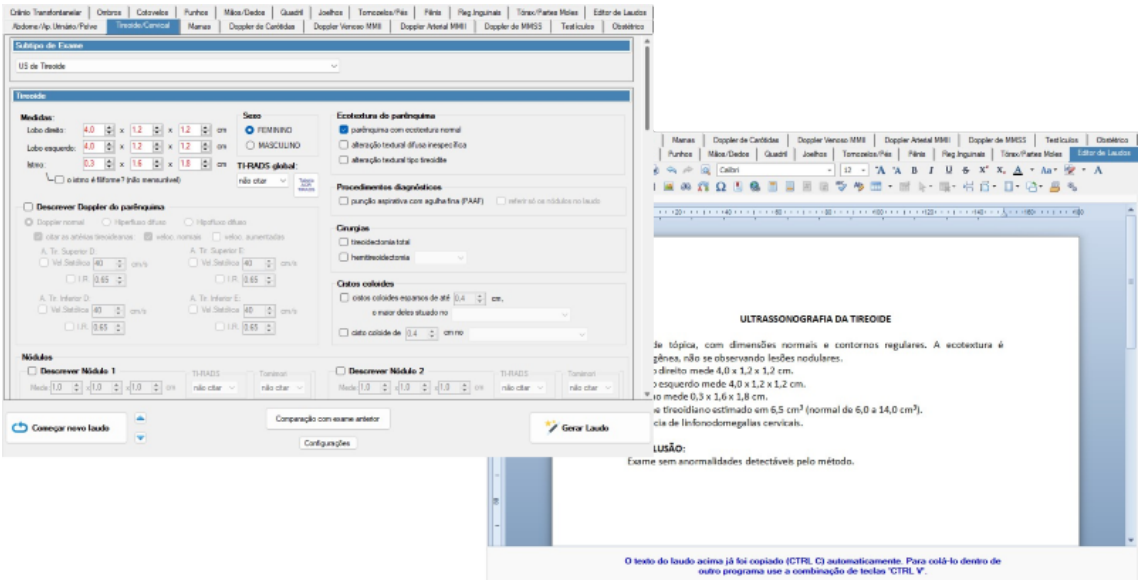


Figura 3.1: Telas do software Turing

Fonte: [Queo 2023]

A ferramenta também não permite integração com outros sistemas, além de não emitir laudos eletrônicos. Segundo a empresa, “Os documentos criados no Turing tem como propósito serem impressos e assinados em forma física. [...] O Turing é apenas um editor de texto”.

Existem 6 versões da aplicação, uma para cada tipo de exame: Turing US (ultrassom), Turing IM (mamografia), Turing RX (raios X), Turing ED (endoscopia), Turing GC (colposcopia), Turing EC (ecocardiografia). Cada versão possui um preço por mês e por máquina instalada.

3.2 Laudomatic 2.0

O software desenvolvido pela FBF Sistemas (2023) tem como finalidade a confecção de laudos médicos periciais. Assim como o Turing, é disponibilizado apenas na forma de software instalável para o Windows. Entre algumas das qualidades citadas pela empresa, destacam-se a existência de campos pré-determinados com todos os itens de uma perícia médica, além da exportação do laudo diretamente em formato Word.

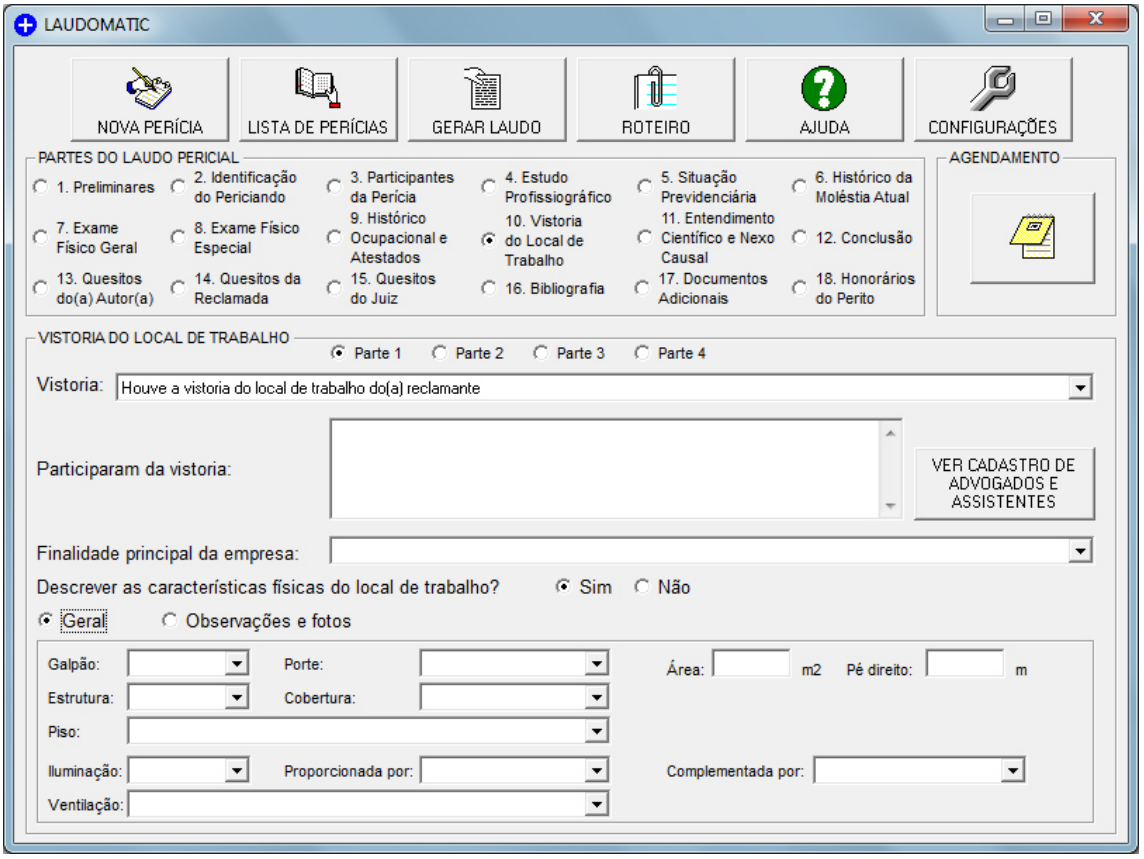


Figura 3.2: Tela principal do Laudomatic
Fonte: [FBF 2023]

Entretanto, a ferramenta não possui nenhuma integração com quaisquer outras ferramentas, sendo necessária a inclusão individual de informações sensíveis em cada laudo criado, como dados do paciente e do médico.

Outra característica, que pode ser vista como uma vantagem do ponto de vista financeiro, é a licença única para a obtenção do software sem a necessidade de pagamentos mensais ou anuais.

3.3 Laudite

O aplicativo Laudite (2021), da empresa VR Health & Tech Serviços de Tecnologia da Iinformação LTDA, promete rapidez e qualidade na geração de

laudos radiológicos. Apesar de ser apresentada como uma ferramenta de nicho, a empresa garante que a ferramenta pode ser utilizada por outros especialistas. Possui um diferencial importante, que é a possibilidade de geração de documentos a partir do reconhecimento de voz.

Ao contrário das outras soluções apresentadas, ela tem o formato de aplicação web, garantindo flexibilidade ao seu uso de diferentes computadores. Também permite, segundo o site, a integração a outros sistemas mediante consulta à empresa.

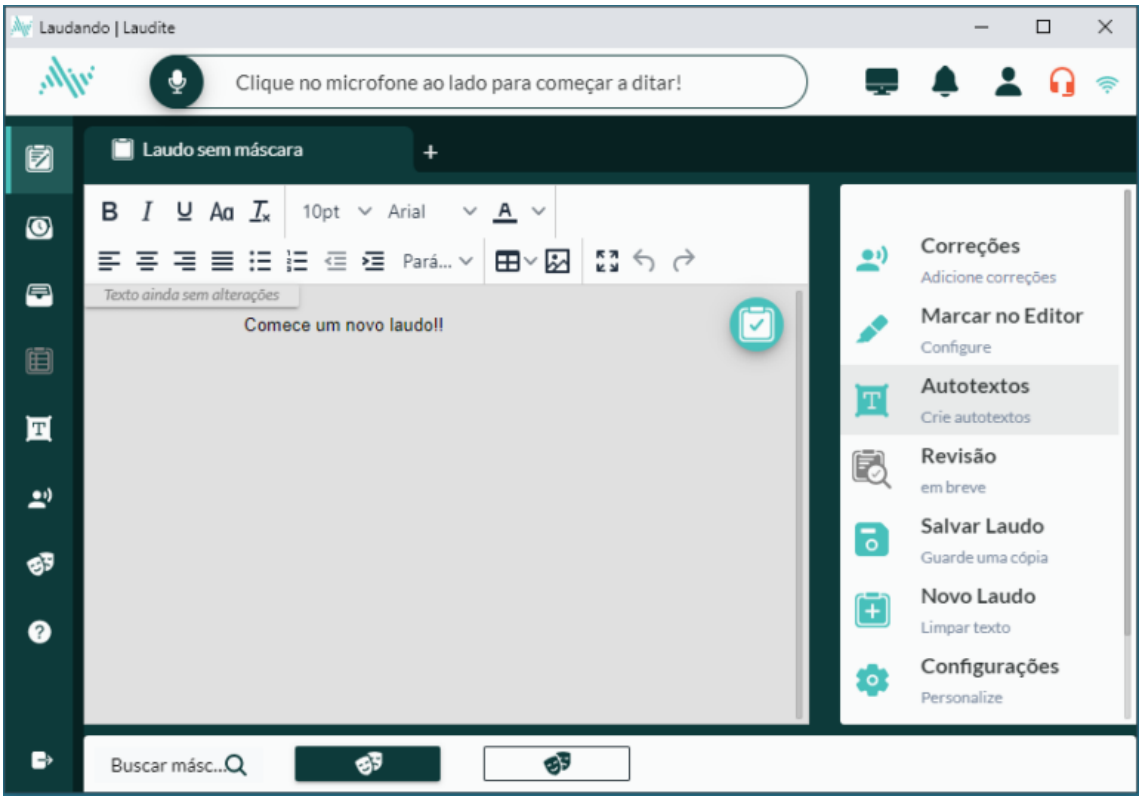


Figura 3.3: Tela principal do Laudite
Fonte: [Laudite 2021]

A aplicação também adota o modelo de negócios baseado em serviço, permitindo o acesso à ferramenta através de planos mensais de pagamento.

3.4 Análise comparativa

Após a etapa de pesquisa das ferramentas relacionadas, é possível destacar as diferenças entre elas de acordo com a tabela abaixo.

Sistema	Meio de distribuição	Há integração com outros sistemas	Permite o uso em qualquer tipo de exame
Turing	software instalável	Não	Não
Laudomatic 2.0	software instalável	Não	Não
Laudite	aplicação web	Sim	Talvez
laudosweb	aplicação web	Sim	Sim

Tabela 3.1: Análise comparativa entre as ferramentas
Fonte: Autor

Levando em consideração os objetivos propostos no desenvolvimento da nossa ferramenta (**laudosweb)**, podemos avaliar que os atributos destacados na Tabela 3.1 conferem algumas vantagens com relação a duas das soluções, como ser disponibilizada como um aplicativo web e permitir a integração com outros sistemas. Também pode ser observado como um diferencial o uso em qualquer tipo de exame, oferecendo maior flexibilidade de uso à ferramenta.

Como diferencial do Laudite, vale destacar a possibilidade de geração de laudos a partir do reconhecimento de voz como uma possível ideia de melhorias futuras na ferramenta.

4 Solução proposta

A solução proposta neste Trabalho de Conclusão de Curso será o desenvolvimento de uma aplicação web, no formato SPA, com o nome de **laudosweb**. Ela permitirá a criação de laudos médicos a partir de exames previamente realizados por uma clínica, baseados em templates pré-definidos de textos organizados por categorias de forma a facilitar a seleção dos itens e padronizar os documentos gerados - que serão posteriormente disponibilizados em formato digital aos pacientes.

4.1 Requisitos da aplicação

Serão apresentados a seguir os requisitos funcionais e não funcionais identificados para a solução proposta. A partir dessa lista, será possível seguir com o desenvolvimento da aplicação que atenda às necessidades e aos objetivos definidos.

4.1.1 Requisitos Funcionais

RF 1 Cadastrar médicos
Descrição: O usuário administrador deve ser capaz de cadastrar usuários médicos no sistema.

RF 2 Acessar o sistema
Descrição: Os usuários devem ser capazes de acessar o sistema com suas credenciais (número de CRM ou e-mail) e uma senha.

RF 3 Receber e armazenar registros de exames realizados

Descrição: A aplicação deve ser capaz de receber e armazenar dados de outros sistemas sobre os exames realizados na clínica, tais como tipo de exame, data, médico responsável, médico solicitante e imagens.

RF 4 Listar exames não laudados

Descrição: O usuário médico deve ser capaz de visualizar a lista dos exames por ele realizados e ainda não laudados.

RF 5 Exibir detalhes do exame/paciente

Descrição: O usuário médico poderá visualizar os detalhes de um exame por ele realizado.

RF 6 Criar laudos

Descrição: O usuário médico poderá criar um laudo a partir da escolha de itens de texto previamente cadastrados pelo corpo médico da clínica. O laudo deverá ser salvo na ferramenta.

RF 7 Criar templates de texto

Descrição: O usuário médico poderá criar e editar templates de texto padronizados para posterior utilização em **RF6,** de acordo com as necessidades do corpo médico da clínica.

RF 8 Disponibilizar laudos aos pacientes

Descrição: O sistema deve disponibilizar um link para acesso a um laudo criado, através de e-mail enviado ao paciente.

RF 9 Consultar laudos

Descrição: O médico poderá consultar laudos emitidos na clínica médica, baseado em filtros como data do exame e nome do paciente.

RF 10 Visualizar laudo médico

Descrição: O usuário paciente poderá visualizar o laudo de seu exame a partir de link recebido por e-mail.

4.1.2 Requisitos Não Funcionais

RNF 1 Usabilidade

Descrição: A ferramenta deve ter uma interface intuitiva e amigável, facilitando o uso dos profissionais da clínica, mesmo aqueles com pouca experiência em tecnologia.

RNF 2 Disponibilidade

Descrição: A ferramenta deve estar disponível em qualquer computador com acesso à internet, permitindo aos usuários a sua utilização sempre que for necessário.

RNF 3 Segurança

Descrição: A ferramenta deve garantir a segurança dos dados sob sua responsabilidade, em consonância com a Lei Geral de Proteção de Dados e do sigilo Médico-Paciente.

RNF 4 Integração

Descrição: A aplicação deve ser capaz de se integrar a outros sistemas da clínica recebendo informações dos mesmos.

4.2 Casos de uso

Com base no levantamento dos requisitos funcionais, serão apresentados os casos de uso da ferramenta, descrevendo as principais funcionalidades e interações esperadas. Os atores identificados foram administrador, médico e paciente.

Caso de Uso #1: Efetuar login
Descrição: Permite o acesso no sistema.
Atores: Administrador e médico

Caso de Uso #2: Efetuar logout
Descrição: Permite a finalização da sessão e saída do sistema.
Atores: Administrador e médico

Caso de Uso #3: Cadastrar usuário
Descrição: Permite o cadastro de um novo usuário para acesso ao sistema.
Ator: Administrador

Caso de Uso #4: Listar exames
Descrição: Permite ao médico exibir a lista de seus exames realizados e não laudados.
Ator: Médico

Caso de Uso #5: Exibir detalhes do exame
Descrição: Permite ao médico exibir os detalhes de um exame realizado.
Ator: Médico

Caso de Uso #6: Criar laudo
Descrição: Permite ao médico criar um laudo a partir da escolha de templates de texto previamente cadastrados.
Ator: Médico

Caso de Uso #7: Editar laudo
Descrição: Permite ao médico editar e personalizar um laudo após a escolha de templates de texto previamente cadastrados.
Ator: Médico

Caso de Uso #8: Criar categoria
Descrição: Permite ao médico criar uma nova categoria de exame.
Ator: Médico

Caso de Uso #9: Editar templates de uma categoria
Descrição: Permite ao médico editar os templates de texto de uma categoria de exame.
Ator: Médico

Caso de Uso #10: Listar laudos
Descrição: Permite ao médico listar os laudos criados.
Ator: Médico

Caso de Uso #11: Consultar laudo
Descrição: Permite ao médico consultar um laudo criado.
Ator: Médico

Caso de Uso #12: Visualizar laudo
Descrição: Permite ao paciente visualizar o laudo de seu exame a partir de um link enviado por e-mail.
Ator: Paciente

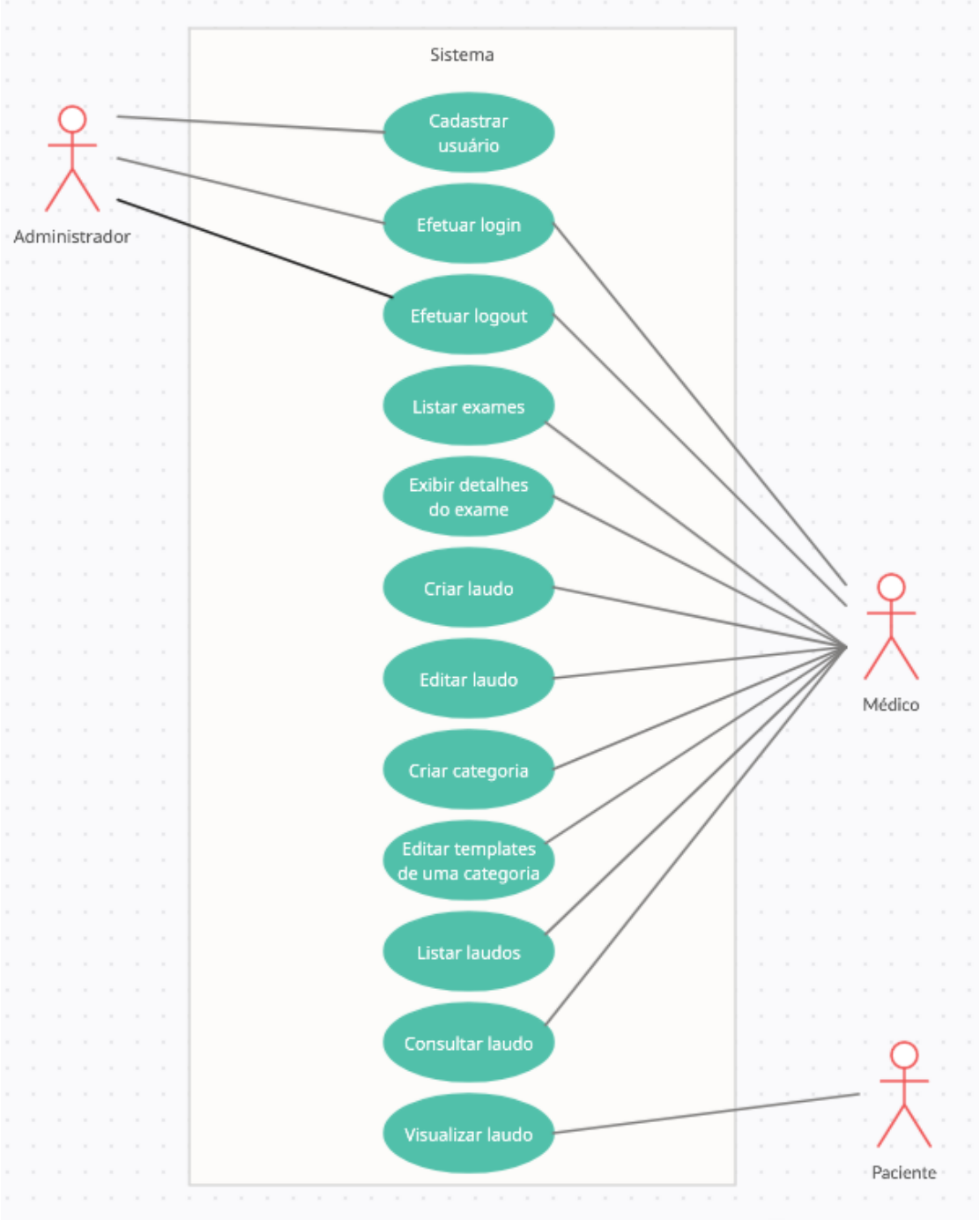


Figura 4.1: Diagrama de Casos de Uso

Fonte: Autor

4.3 Visão geral da arquitetura da solução

O **laudosweb** foi implementado usando a *stack* **MEVN**, uma popular combinação de tecnologias para o desenvolvimento de aplicações web. Ela é muito utilizada no desenvolvimento de aplicações SPA (aplicações *single-page*), caracterizadas pelo uso de uma arquitetura baseada em componentes. Uma das principais vantagens dessa *stack* é a utilização de Javascript em todas elas, tornando o desenvolvimento mais padronizado. Seu nome vem das iniciais das tecnologias utilizadas: **M**ongoDB, **E**xpress.js, **V**ue.js e **N**ode.js - citadas entre os capítulos 2.3 a 2.6.

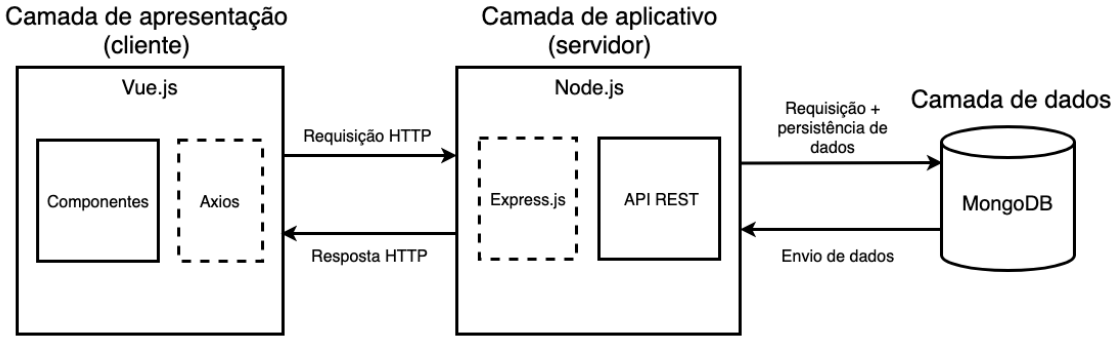


Figura 4.2: Arquitetura simplificada do sistema
Fonte: Autor

Como demonstrado na Figura 4.2, para realizar as requisições HTTP e interagir com a API, o **Axios** foi a biblioteca escolhida para ser usada no lado *cliente* pelos componentes do **Vue.js** no envio e recebimento de solicitações do servidor. No lado *servidor*, o **Express.js** será o responsável por fornecer uma infraestrutura escalável juntamente com o **Node.js**, enquanto o **MongoDB** será utilizado como o banco de dados NoSQL para armazenar os dados da aplicação.

5 Implementação da solução

A implementação da solução proposta foi dividida em 3 etapas principais: modelagem dos dados, desenvolvimento da API REST (camada de aplicação) e criação da interface com o usuário (camada de apresentação).

5.1 Modelagem dos dados

O MongoDB, sendo um banco de dados não relacional, difere dos tradicionais bancos relacionais em termos de representação visual dos dados. Enquanto os bancos relacionais são conhecidos por sua estrutura tabular, o MongoDB utiliza uma abordagem diferente ao armazenar os dados em documentos JSON. Em vez de seguir uma estrutura tabular rígida como os bancos relacionais, o MongoDB utiliza um formato mais flexível de dados. Neste documento, optamos por apresentar os dados em formato de documentos JSON, destacando suas características estruturais e de dados.

No projeto **laudosweb** foram mapeados os seguintes modelos de documentos, chamados de “coleções” em MongoDB:

- Usuários: representa os usuários do sistema, conforme o exemplo apresentado na Figura 5.1;
- Exames: contém os dados referentes aos exames realizados; um exemplo ilustrativo é mostrado na Figura 5.2;
- Templates: descreve os templates utilizados para compor os laudos de exames; a Figura 5.3 apresenta um exemplo de template;
- Laudos: contém as informações referentes aos laudos criados com o laudosweb; um exemplo de documento desta coleção é apresentado na Figura 5.4.

```
{
  "_id": "6477747cce574fd3cabe26a9", //id do usuário (criado pelo MongoDB)
  "no_usuario": "Fulana de Tal", //nome do usuário
  "co_crm": "11111", //essa propriedade só existirá quando o usuário for 'Médico'
  "no_email": "fulana.tal@clinicatcc.com.br", //e-mail do usuário
  "ic_tipo": "M", //tipo de usuário: A - Administrador; M - Médico
  "hash": "..." //senha transformada em hash
}
```

Figura 5.1: Exemplo de documento da coleção de usuários

Fonte: Autor

```
{
  "_id": "648277e8903e2aed9084b0e8",
  "id_exame": "22d30caa-19b1-46b5-a36e-67e78c904eee",
  "dt_exame": "2023-06-07T09:30:00.120Z",
  "no_especialidade": "eda",
  "ar_imagens": [
    "url 1", "url 2", ...
  ],
  "o_paciente": {
    "nome": "Morgana Braga",
    "dt_nascimento": "1990-06-07T00:00:00.000Z",
    "sexo": "Feminino",
    "email": "Morgana_Braga@live.com",
    "prontuario": "..."
  },
  "o_medico_solicitante": {
    "nome": "Fulano da Silva",
    "crm": "22222",
    "email": "fulano.silva@clinicatcc.com.br"
  },
  "o_medico_exame": {
    "nome": "Fulano da Silva",
    "crm": "22222",
    "email": "fulano.silva@clinicatcc.com.br"
  }
}
```

Figura 5.2: Exemplo de documento da coleção de exames

Fonte: Autor

```
{
  "_id": "64777e4fce574fd3cabe26b0",
  "especialidade": "eda",
  "nome": "Endoscopia Digestiva Alta",
  "data": [
    {
      "nome": "Aparelho",
      "menu": [
        {
          "nome": "FICE",
          "texto": "<p>...</p>",
          "conclusao": "<p>...</p>",
          "obs": "<p>...</p>",
          "ref": "<p>...</p>",
        },
        {
          "nome": "Outros",
          "menu": [...]
        }
      ]
    }
  ]
}
```


Figura 5.3: Exemplo de documento da coleção de templates

Fonte: Autor

A coleção de templates, conforme demonstrado na Figura 5.3, caracteriza-se pela recursividade, onde cada objeto dentro da propriedade “data” possui uma propriedade “nome” e pode ser definida de duas formas: com a propriedade “menu” como um *array* de objetos caracterizando este como um ramo da árvore de informações; ou pelo menos uma das propriedades entre “texto”, “conclusão”, “obs”, “ref”, significando que tal objeto é um template de texto a ser utilizado na criação dos laudos.

```
{
  "_id": "6482792357c20ea67fea224c",
  "id_exame": "52486478-614c-489b-8663-ac9ef66c177d",
  "o_medico_laudo": {
    "nome": "Fulana de Tal",
    "crm": "11111",
    "email": "fulana.tal@clinicatcc.com.br"
  },
  "ar_texto": [
    {
      "nome": "Normal",
      "texto": "<p>...</p>",
      "_id": "sbP9TxynO33bIq6N"
    }
  ],
  "dt_laudo": "2023-06-09T00:58:11.387Z"
}
```


Figura 5.4: Exemplo de documento da coleção de laudos
Fonte: Autor

5.2 Camada de aplicativo (na forma de uma API REST)

Neste capítulo será apresentada a implementação da camada de aplicativo (back-end) do sistema. Foi utilizado o **Node.js** com **Express.js** (responsável por tratar as requisições HTTP), além das bibliotecas **Joi** (validação das requisições), **bcrypt** (responsável por encriptar e decriptar as senhas em *hash*), **jsonwebtoken** (autenticação com JWT) e **mongodb** (interface com o MongoDB Server). A arquitetura do back-end foi dividida em camadas, notadamente rotas, controllers, services e DAO, com o objetivo de criar uma API REST eficiente e escalável.

5.2.1 Estrutura de pastas e arquivos

A organização da estrutura de pastas e arquivos, mostrada na Figura 5.5, foi criada de forma a auxiliar na compreensão e manutenção de código.

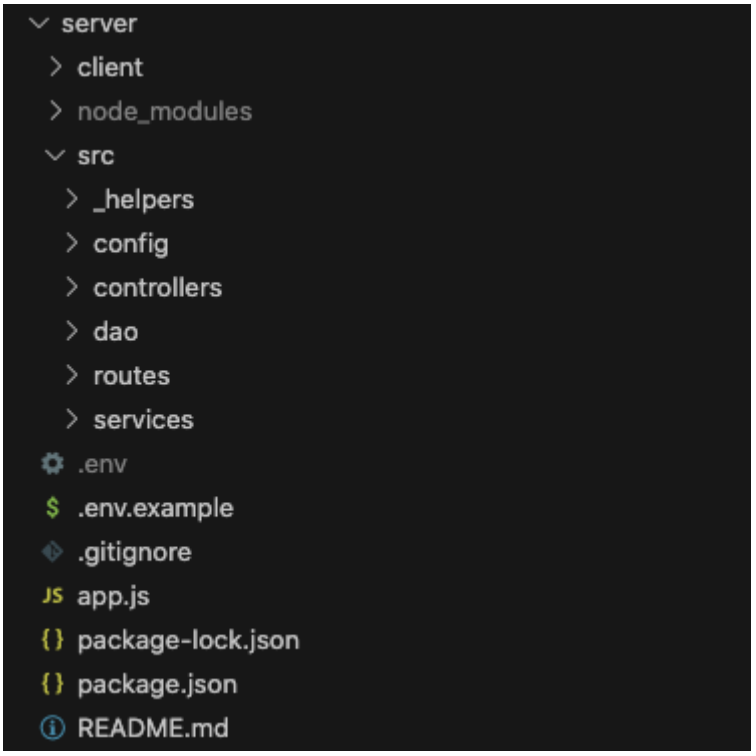


Figura 5.5: Estrutura de pastas do servidor
Fonte: Autor

A pasta **src** contém todo o código-fonte da aplicação, separados nas seguintes sub-pastas:

- Pasta **routes**: Nesta pasta estão os arquivos responsáveis pela definição das rotas da API. As rotas são mapeadas para os arquivos das respectivas entidades/coleções, facilitando a organização e separação das funcionalidades.
- Pasta **controllers**: Aqui são armazenados os controladores da aplicação, responsáveis por receber as requisições das rotas e coordenar as ações a serem executadas.

- Pasta **services**: Aqui estão os serviços da aplicação, contendo as lógicas de negócio que serão chamadas pelos controladores.
- Pasta **dao**: Contém os DAOs (*Data Access Objects*), responsáveis pela interação com o banco de dados da aplicação. Cada entidade possui seu próprio arquivo DAO, garantindo em todos os níveis de organização física a separação das responsabilidades.
- Pasta **helpers**: Aqui estão os códigos auxiliares utilizados para validação da autenticação, autorização, tratamento de erros, etc.

5.2.2 Definição das rotas

Com o levantamento de requisitos e a modelagem de dados concluídas, podemos definir as rotas da API que servirão à aplicação. No nosso escopo, além da rota de login, temos 4 entidades mapeadas: usuário, exame, laudo e template. Abaixo, seguem as rotas atribuídas a cada uma delas:

Rota: api/login/
Método: POST
Descrição: Rota responsável pelo login do usuário.
Parâmetros:
username: CRM do médico ou e-mail do administrador do sistema
password: senha do usuário

Rota: api/exame
Método: GET
Descrição: Rota responsável por trazer a lista de exames não laudados do médico responsável pelo exame.
Parâmetros:
Sem parâmetros

Rota: api/exame/{id}

Método: GET

Descrição: Rota responsável por trazer um documento exame pelo id do mesmo.

Parâmetros:

id: número de identificação do exame

Rota: api/laudo/?{dt_laudo}

Método: GET

Descrição: Rota responsável por trazer a lista de laudos criados por data.

Parâmetros:

dt_laudo: data do(s) laudo(s) a serem retornados

Rota: api/laudo/{id}

Método: GET

Descrição: Rota responsável por trazer um laudo pelo id do mesmo.

Parâmetros:

id: número de identificação do laudo

Rota: api/laudo/

Método: POST

Descrição: Rota responsável por criar um laudo.

Parâmetros:

id_exame: número de identificação do exame

o_medico_laudo: dados do médico responsável pelo laudo

ar_texto: array com os itens de template selecionados durante a confecção do laudo

dt_laudo: data de criação do laudo

Rota: api/template/{especialidade}

Método: GET

Descrição: Rota responsável por trazer os templates associados a uma especialidade.

Parâmetros:

especialidade: nome da especialidade

Rota: api/template/{especialidade}

Método: PATCH

Descrição: Rota responsável por modificar templates associados a uma especialidade.

Parâmetros:

especialidade: nome da especialidade

alterações: array de objetos no formato application/json-patch+json, seguindo o padrão rfc6902.

5.2.3 Validação dos dados

A validação dos dados recebidos por uma API é essencial para garantir a integridade das informações manipuladas pela aplicação. Ao responder às solicitações, é importante garantir que os dados recebidos estejam em consonância com os requisitos definidos.

Nesse sentido, a utilização de mecanismos de validação é fundamental. Utilizamos nesse trabalho a biblioteca **Joi** para validar os dados enviados nas requisições da API. O Joi é uma popular biblioteca de validação de dados Javascript, com média acima de 7 milhões de downloads semanais [NPM 2023], que permite descrever seus dados usando uma linguagem simples, intuitiva e legível [Joi 2023].

```
1  const schemaNovo = Joi.object({
2      id_exame: Joi.string().guid({ version: 'uuidv4'}).required(),
3
4      o_medico_laudo: Joi.object({
5          nome: Joi.string().required(),
6          crm: Joi.string().alphanum().required(),
7          email: Joi.string().email().required()
8      }),
9
10     ar_texto: Joi.array().items(
11         Joi.object({
12             _id: Joi.string().required(),
13             nome: Joi.string().required(),
14             texto: Joi.string(),
15             conclusao: Joi.string(),
16             ref: Joi.string(),
17             obs: Joi.string()
18         }).or('texto', 'conclusao', 'ref', 'obs')
19     ),
20
21     dt_laudo: Joi.date().required()
22 });
23
24 exports.validateNovoLaudo = async (req, res, next) => {
25     await schemaNovo.validateAsync(req.body);
26     next()
27 };
```

Figura 5.6: Reprodução parcial do módulo **laudo.validator**
Fonte: Autor

Como demonstrado na imagem acima (Figura 5.6), a função **validateNovoLaudo** (linha 24) será executada em **laudo.route** como um middleware de validação do Express.js. A linha 25 da referida função faz a chamada de **validateAsync** (Joi), comparando o corpo da requisição com o esquema esperado. Sendo essa comparação bem sucedida, o Express.js seguirá com o fluxo de execução da requisição. Caso contrário, devolverá uma mensagem de erro para o cliente.

5.2.4 Autenticação e segurança

Autenticação e segurança possuem um papel importante para a proteção de dados e no controle do acesso a recursos disponibilizados por uma API. A proteção contra o acesso não autorizado e a prevenção de ataques garantem ao usuário um ambiente de confiança e em conformidade com os regulamentos de proteção de dados, como a Lei Geral de Proteção de Dados. [LGPD 2018]

A necessidade de autenticação e segurança em aplicações web é amplamente reconhecida na literatura. Segundo a descrição de um dos itens na lista Top 10 das maiores vulnerabilidades na web da OWASP, “a confirmação da identidade do usuário, autenticação e gerenciamento de sessão são essenciais para proteção contra ataques relacionados à autenticação.” [OWASP 2021]

Nesse contexto, a nossa aplicação utilizará as bibliotecas **bcrypt** para encriptar as senhas dos usuários antes de salvá-las no banco, além do uso da biblioteca **jsonwebtoken**, que será responsável por gerar tokens de acesso que serão verificados em cada requisição feita à API, também como um middleware do Express.js.

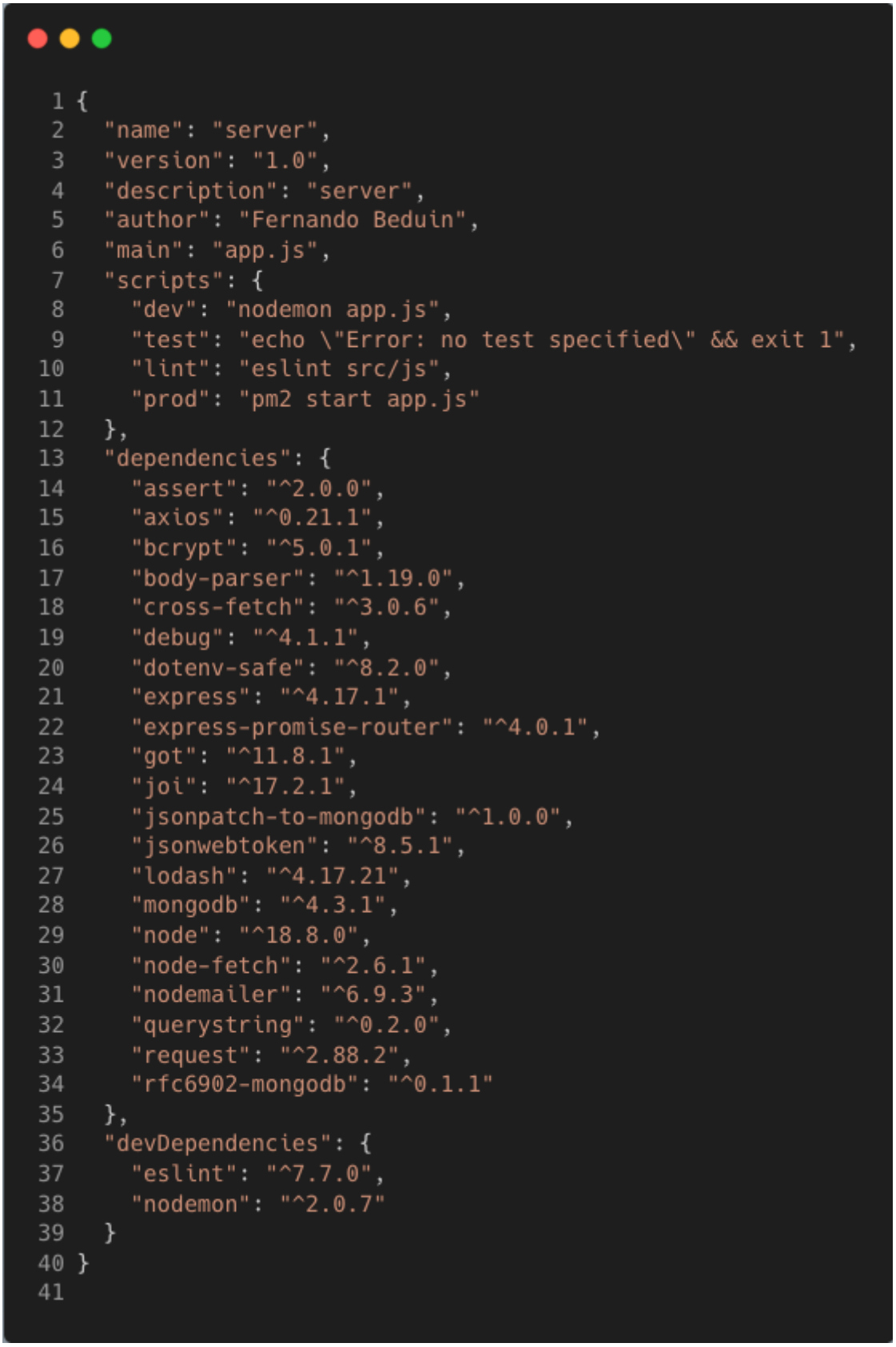
A biblioteca **bcrypt** será responsável por criar um hash a partir da senha do usuário recebida pela API no momento do seu cadastro, antes de salvá-la no banco de dados. Durante a autenticação, a senha enviada é convertida em hash e comparada com a que está salva no banco, garantindo assim a segurança de um dado crítico do sistema.

Já a biblioteca **jsonwebtoken** enviará um token de autorização quando o usuário fizer a correta autenticação no sistema utilizando seu CRM ou e-mail e senha. Esse token será enviado em todas as requisições feitas pelo usuário para garantir a segurança dos recursos acessados.

5.2.5 Desenvolvimento da API

Nessa seção serão abordados os detalhes de desenvolvimento da API conforme as especificações e tecnologias citadas anteriormente.

Uma aplicação Node.js é criada a partir do terminal com o comando “npm init”, o qual cria uma estrutura de arquivos inicial do projeto com base em configurações padrão. Nessa estrutura, dois arquivos se destacam inicialmente: **package.json** e **app.js**. O primeiro é um arquivo de detalhes e configurações, contendo informações sobre scripts de execução da aplicação e pacotes instalados.



```
{
  "name": "server",
  "version": "1.0",
  "description": "server",
  "author": "Fernando Beduin",
  "main": "app.js",
  "scripts": {
    "dev": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1",
    "lint": "eslint src/js",
    "prod": "pm2 start app.js"
  },
  "dependencies": {
    "assert": "^2.0.0",
    "axios": "^0.21.1",
    "bcrypt": "^5.0.1",
    "body-parser": "^1.19.0",
    "cross-fetch": "^3.0.6",
    "debug": "^4.1.1",
    "dotenv-safe": "^8.2.0",
    "express": "^4.17.1",
    "express-promise-router": "^4.0.1",
    "got": "^11.8.1",
    "joi": "^17.2.1",
    "jsonpatch-to-mongodb": "^1.0.0",
    "jsonwebtoken": "^8.5.1",
    "lodash": "^4.17.21",
    "mongodb": "^4.3.1",
    "node": "^18.8.0",
    "node-fetch": "^2.6.1",
    "nodemailer": "^6.9.3",
    "querystring": "^0.2.0",
    "request": "^2.88.2",
    "rfc6902-mongodb": "^0.1.1"
  },
  "devDependencies": {
    "eslint": "^7.7.0",
    "nodemon": "^2.0.7"
  }
}

```

Figura 5.7: Conteúdo do arquivo **package.json**

Fonte: Autor

A propriedade **dependencies** (Figura 5.7, linha 13) lista as bibliotecas que foram instaladas utilizando o gerenciador de pacotes NPM com o comando “npm install <nome da biblioteca>”. Outra importante propriedade é a **main**, que define o ponto de partida da aplicação - no nosso caso, o arquivo **app.js**.

```
1  const express = require('express');
2  const app = express();
3  const bodyParser = require('body-parser');
4
5  const dotenv = require('dotenv-safe');
6  dotenv.config();
7
8  const routes = require('./src/routes/index');
9
10 app.use(bodyParser.json());
11 app.use(express.static('client'));
12 app.use('/api', routes);
13
14 const path = require('path');
15 app.route('/*').get(function (req, res) {
16   res.sendFile(path.join(__dirname + '/client/index.html'));
17 });
18
19 app.use(require('./src/_helpers/error.handler'));
20
21 app.listen(process.env.PORT, function () {
22   console.log(`Application listening on port ${process.env.PORT}`);
23 });
```

Figura 5.8: Conteúdo do arquivo **app.js**
Fonte: Autor

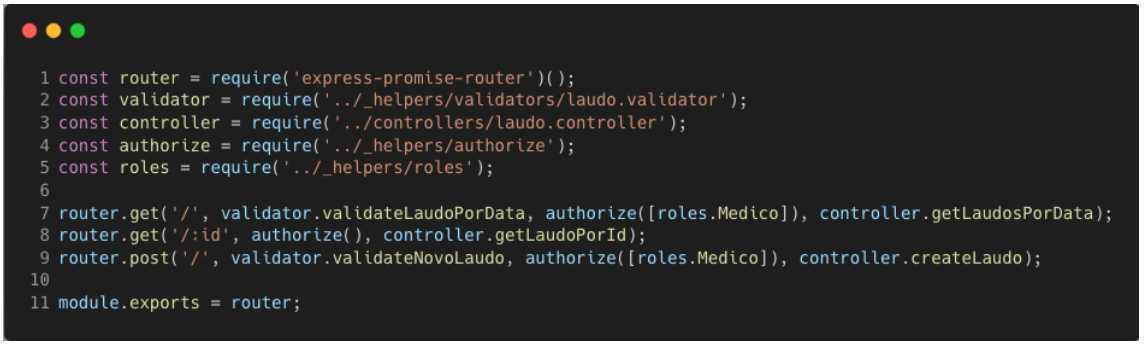
Neste arquivo, como demonstrado na Figura 5.8, é definida a variável **app** (linha 2) a partir da biblioteca Express.js, que será o ponto de partida da API REST. O arquivo **index.js** da pasta **routes** é carregado (linha 8) e utilizado como um middleware da aplicação (linha 12), indicando que toda requisição HTTP recebida pela API que comece com “/api” seja por ele interceptada e processada.

```
1  const express = require('express');
2  const router = express.Router();
3
4  const authenticate = require('../_helpers/authenticate');
5  const signRoute = require('./usuario.route');
6
7  const examesRoute = require('./exame.route');
8  const laudoRoute = require('./laudo.route');
9  const templateRoute = require('./template.route');
10
11 router.post('/login', authenticate);
12 router.use('/sign', signRoute);
13
14 router.use('/usuario', signRoute);
15 router.use('/exame', examesRoute);
16 router.use('/laudo', laudoRoute);
17 router.use('/template', templateRoute);
18
19 module.exports = router;
```

Figura 5.9: Conteúdo do arquivo **/src/routes/index.js**

Fonte: Autor

Na Figura 5.9 podemos ver como o fluxo de execução é direcionado para o arquivo de rota correspondente. Usando como exemplo uma requisição POST “api/laudo/”, responsável pela criação de um novo laudo, esta será interceptada pelo *middleware* definido na linha 16, direcionando ao arquivo **laudo.route.**



```
1  const router = require('express-promise-router')();
2  const validator = require('../_helpers/validators/laudo.validator');
3  const controller = require('../controllers/laudo.controller');
4  const authorize = require('../_helpers/authorize');
5  const roles = require('../_helpers/roles');
6
7  router.get('/', validator.validateLaudoPorData, authorize([roles.Medico]), controller.getLaudosPorData);
8  router.get('/:id', authorize(), controller.getLaudoPorId);
9  router.post('/', validator.validateNovoLaudo, authorize([roles.Medico]), controller.createLaudo);
10
11 module.exports = router;
```

Figura 5.10: Código-fonte do arquivo **laudo.route**

Fonte: Autor

A Figura 5.10 exemplifica o uso da rota pela API. Na linha 9 temos a função **router** do Express.js que, quando receber a nossa requisição POST "api/laudo/"**,** executará dois *middlewares:* o validador da requisição e o helper de autorização. Caso seja, respectivamente, validada e autorizada, a requisição segue o fluxo chamando a função **createLaudo** de **laudo.controller.**

```
1  const service = require('../services/laudo.service');
2
3  exports.getLaudosPorData = async (req, res) => {
4      const result = await service.getLaudosPorData(req.query.dt_laudo);
5      res.status(200).send(result);
6  };
7
8  exports.getLaudoPorId = async (req, res) => {
9      const result = await service.getLaudoPorId(req.params.id);
10     res.status(200).send(result);
11 };
12
13 exports.createLaudo = async (req, res) => {
14     const laudo = {
15         id_exame: req.body.id_exame,
16         o_paciente: req.body.o_paciente,
17         o_medico_laudo: req.body.o_medico_laudo,
18         ar_texto: req.body.ar_texto,
19         dt_laudo: new Date(req.body.dt_laudo)
20     }
21
22     const result = await service.createLaudo(laudo);
23     res.status(200).send(result);
24 };
```

Figura 5.11: Código-fonte do arquivo **laudo.controller**

Fonte: Autor

O controlador demonstrado na Figura 5.11 define o objeto (linhas 14-20) a ser repassado para a função **createLaudo** de **laudo.services** (linha 22).

```
1 const dao = require('../dao/laudo.dao');
2 const mailService = require('../services/mail.service');
3
4 [...]
5
6 exports.createLaudo = async (laudo) => {
7     const retorno = await dao.createLaudo(laudo);
8     const email = laudo.o_paciente.email;
9     const assunto = 'Seu laudo está disponível!';
10    const mensagem = `Olá, ${laudo.o_paciente.nome}!<p> Seu laudo já está disponível, acesse clicando no link
  abaixo: <p>${process.env.URL}:${process.env.CLIENT_PORT}/laudo/${laudo.id_exame}/pdf`;
11    mailService.sendEmail(email, assunto, mensagem);
12    return retorno;
13 };
```

Figura 5.12: Código-fonte parcial do arquivo **laudo.service**
Fonte: Autor

Conforme demonstrado na Figura 5.12, a função **createLaudo** chama a função equivalente em **laudo.dao** (Figura 5.13), que irá salvar o documento no banco de dados e em seguida chamará o serviço **SendEmail** para enviar o e-mail ao cliente disponibilizando o link de acesso ao laudo.

```
1 const { connect } = require('../_helpers/mongodb');
2
3 [...]
4
5 exports.createLaudo = async (laudo) => {
6     try {
7         const db = await connect();
8         await db.collection("col_laudos").insertOne(laudo);
9     } catch(e) {
10        return e;
11    }
12 }
```

Figura 5.13: Código-fonte parcial do arquivo **laudo.dao**
Fonte: Autor

Seguindo esse padrão, foram implementadas todas as rotas descritas anteriormente. Essa organização da API tem como objetivo garantir a escalabilidade e a facilidade na manutenção do código.

5.2.6 Integração com uma API externa

A integração da aplicação web com uma API externa que disponibilizará dados de exames será realizada através de um serviço criado dentro da nossa API REST. O referido serviço será executado a partir do serviço chamado a partir da rota **api/exame,** conforme demonstrado na Figura 5.14.

```
1  const axios = require("axios").default;
2  const dao = require('../dao/exame.dao');
3
4  exports.getExamePorMedico = async (user) => {
5      await getExamesExternos();
6      const retorno = await dao.getExamePorMedico(user);
7      return retorno;
8  };
9
10 exports.getExamePorId = async (id) => {
11     const retorno = await dao.getExamePorId(id);
12     return retorno;
13 };
14
15 const getExamesExternos = async () => {
16     const dados = (await axios.get(process.env.API_ENDPOINT)).data;
17     await dao.saveExamesExternos(dados);
18     return;
19 };
```

Figura 5.14: Código-fonte do arquivo **exame.service**
Fonte: Autor

Na Figura 5.14, podemos observar a função **getExamePorMedico** executando uma outra função chamada **getExamesExternos** (linha 5), que é a responsável por acessar a API externa (linha 16) e, com os dados obtidos, salvá-los no banco de dados (linha 17). Somente após a execução dessa função, é que a função **getExamePorMedico** buscará os exames no banco de dados (linha 6), garantindo assim a entrega de uma lista atualizada de exames à camada de apresentação (cliente) da aplicação.

Nesse contexto, é importante destacar que a API externa utilizada no desenvolvimento da ferramenta é apenas ilustrativa, trazendo dados de exames fictícios com o objetivo de demonstrar a integração descrita neste capítulo.

5.3 Camada de apresentação

A camada de apresentação (front-end) do projeto foi iniciada através da CLI (Interface de Linha de Comando) do próprio framework **Vue.js,** executada com o comando “vue create client” no terminal a partir da pasta raiz da aplicação (laudosweb). O mesmo criou uma subpasta com o nome “client” e uma estrutura padrão, à qual foram adicionados os componentes específicos da aplicação.

Nessa estrutura, é a partir do arquivo **main.js** que a aplicação é executada. Nele são definidos os componentes que serão utilizados pelo Vue e o componente raiz do aplicativo.

```
1  import '@babel/polyfill';
2  import 'mutationobserver-shim';
3  import '@/_plugins/fontawesome';
4  import 'element-ui/lib/theme-chalk/index.css';
5  import '@/_plugins/bootstrap-vue';
6
7  import Vue from 'vue';
8  import App from './App.vue';
9  import ElementUI from 'element-ui';
10 import locale from 'element-ui/lib/locale/lang/pt-br';
11 import Api from '@/_plugins/axios';
12 import Sjcl from '@/_plugins/sjcl';
13 import Vuelidate from 'vuelidate';
14 import { BootstrapVue } from 'bootstrap-vue';
15
16 import router from '@/router';
17 import store from '@/store';
18
19 import CompModalForm from '@/components/CompModalForm';
20 import CompTree from '@/components/CompTree';
21 import CompTreeNode from '@/components/CompTreeNode';
22 import CompLaudo from '@/components/CompLaudo';
23
24 Vue.config.productionTip = false;
25 Vue.prototype.$project = require('@/project.json');
26 Vue.prototype.$http = Api;
27 Vue.prototype.$encr = Sjcl;
28 Vue.component('cp-modal-form', CompModalForm);
29 Vue.component('cp-laudos-tree', CompTree);
30 Vue.component('cp-laudos-tree-node', CompTreeNode);
31 Vue.component('cp-laudo', CompLaudo);
32 Vue.use(Vuelidate);
33 Vue.use(BootstrapVue);
34 Vue.use(ElementUI, {locale});
35
36 new Vue({
37   router,
38   store,
39   render: h => h(App)
40 }).$mount('#app')
```

Figura 5.15: Código-fonte do arquivo **main.js**

Fonte: Autor

A parte inicial do arquivo (Figura 5.15) contém as importações e registros dos arquivos, bibliotecas e componentes que serão utilizados. Na linha 36 é instanciado um objeto Vue que irá renderizar a aplicação a partir do componente raiz **App.vue.**

Serão abordadas nos próximos tópicos algumas dessas bibliotecas usadas juntamente com o Vue.js para facilitar o desenvolvimento da aplicação.

5.3.1 Vue Router

O Vue Router é a biblioteca oficial do Vue responsável pela criação das rotas em aplicações SPA, permitindo a transição entre diferentes componentes da aplicação de forma similar às rotas utilizadas numa API. Entre suas características, destacam-se a utilização de rotas dinâmicas, definição de rotas aninhadas e redirecionamentos a partir de outros componentes.

```
import Vue from 'vue';
...
import Exames from '@/views/Exames.vue';
...

Vue.use(VueRouter);

const routes = [
    ...
     {
        path: '/exame',
        name: 'Exames',
        component: Exames,
        props: true
    },
    ...
];

const router = new VueRouter({
    mode: 'history',
    base: project.context,
    routes
});

export default router;

```

Figura 5.16: Código-fonte parcial do arquivo **router/index.js**
Fonte: Autor

Na Figura 5.16, podemos observar a criação da instância VueRouter (linha 19) com as rotas definidas (linha 8) e sua inclusão dentro do Vue (linha 6).

5.3.2 Axios

Axios é uma biblioteca Javascript amplamente utilizada para realizar requisições HTTP, facilitando a interação entre o front-end e o back-end. Possui recursos como suporte a interceptores para manipular requisições e respostas, e gerenciamento de erros.

```
1 async getExames() {
2     ...
3     const items = (await this.$http.get(`/api/exame/`)).data;
4     ...
5 }
```

Figura 5.17: Exemplo de requisição HTTP feita pelo Axios
Fonte: Autor

Na Figura 5.17 (acima), podemos ver um exemplo de requisição GET feita com o auxílio do Axios para a rota “/api/exame” (linha 3).

5.3.3 Telas do sistema

A tela de login (Figura 5.18) é o ponto de partida da aplicação. Os dados de autenticação são enviados ao servidor e, caso estejam corretos, a aplicação receberá um *token* de acesso que será utilizado durante todas as interações entre cliente e servidor. Quaisquer solicitações a outros *endpoints* da aplicação sem o *token* de acesso, serão direcionadas para esta tela.

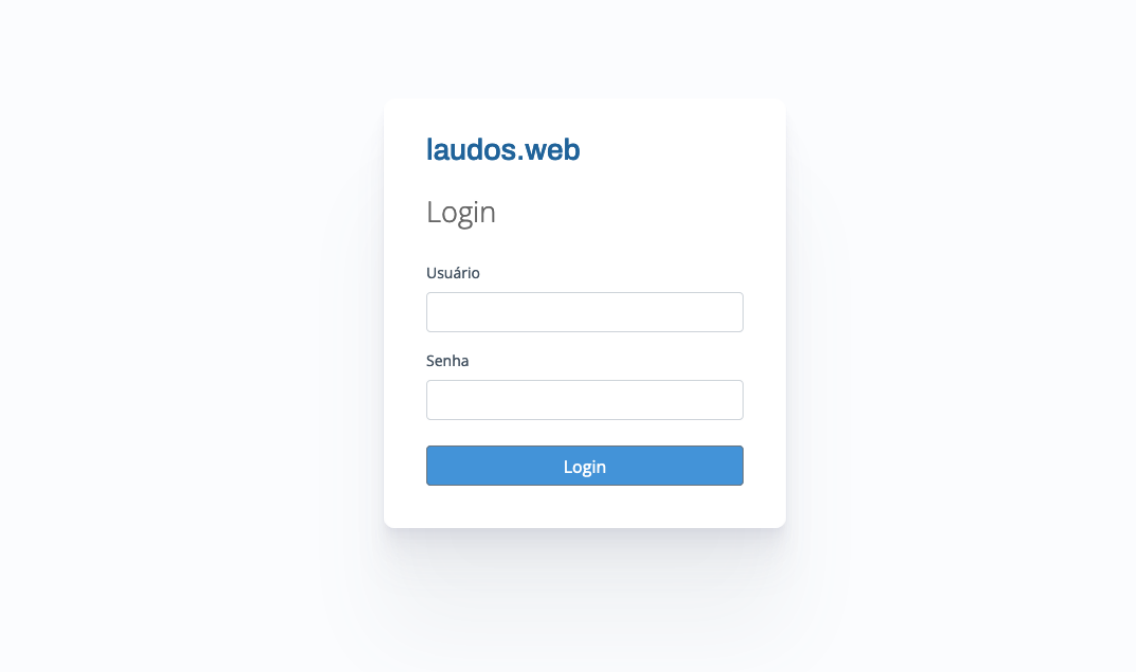


Figura 5.18: Tela de Login do **laudosweb**
Fonte: Autor

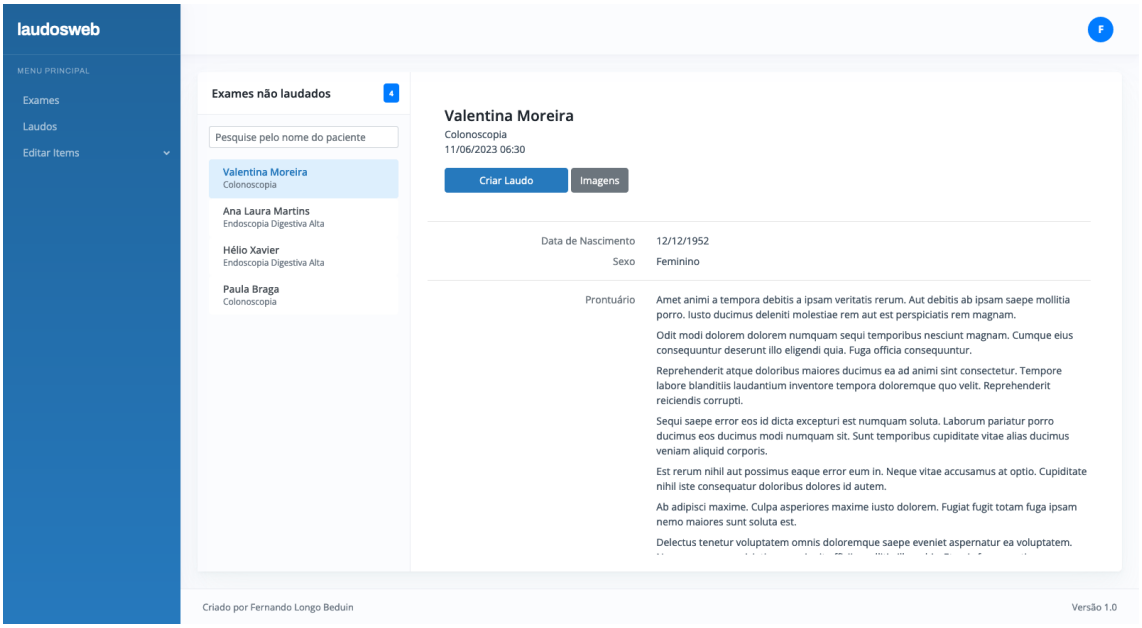


Figura 5.19: Tela de Exames do **laudosweb**

Fonte: Autor

A primeira tela a ser exibida após o login é a lista de exames realizados e ainda não laudados (Figura 5.19). Quando o médico clica em um item, são exibidas as informações do exame e do paciente, além de disponibilizar duas funcionalidades: **Imagens** e **Criar Laudo**. A primeira exibirá um modal com as imagens do exame realizado, enquanto a segunda permitirá ao médico a criação do laudo para o exame.

```
async mounted() {
    await this.getExames();
}
...
async getExames() {
  try {
    const items = (await this.$http.get(`/api/exame/`)).data;
    items.forEach(it => it.nomeExame = this.nomeExame(it.no_especialidade));
    this.items_exames = items.sort((a, b) =>
        this.ts(a.dt_exame) -  this.ts(b.dt_exame)
    );
  } catch (error) {
      this.toast(
        error.data ? error.data.message || error.data : error.statusText,
      false);
  }
},
async criarLaudo(){
    this.$router.push({
        path: `/laudo/${this.exame_selecionado.id_exame}/`
    });
}
```

Figura 5.20: Código-fonte parcial do componente **Exames.vue**
Fonte: Autor

Na Figura 5.20, podemos ver parte do código do componente **Exames.vue**. O método **getExames** é chamado dentro da função **mounted**, parte integrante do ciclo de vida de um componente Vue e executada assim que um componente é renderizado na aplicação. Já o método **criarLaudo** envia a rota declarada em Vue Router, que será responsável por abrir a tela de criação de um novo laudo.

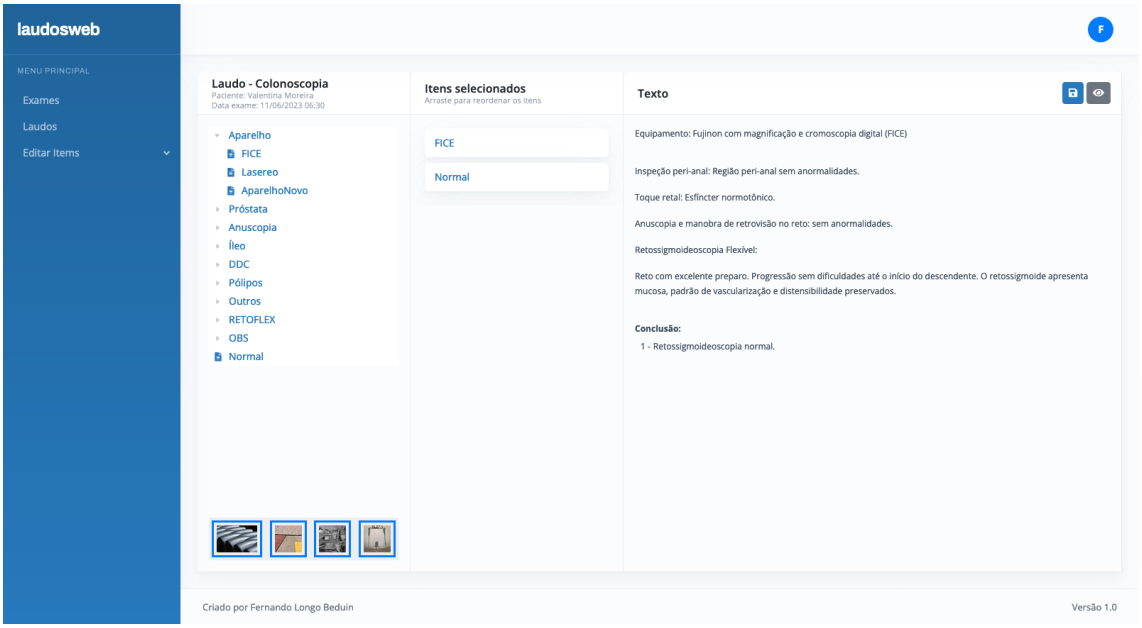


Figura 5.21: Tela de criação de laudo do **laudosweb**
Fonte: Autor

A tela de criação do laudo do exame, mostrada na Figura 5.21, é composta por 3 colunas: árvore com os templates de texto organizados por categorias, itens selecionados e texto do laudo. Ao clicar em um template de texto na primeira coluna, o mesmo aparecerá numa lista na segunda coluna, e o texto correspondente aparecerá na terceira coluna, podendo ser editado livremente pelo médico antes de salvar a versão final do laudo. Podemos observar no rodapé da primeira coluna as miniaturas das imagens obtidas no exame, permitindo ao médico o acesso às imagens enquanto edita o texto. Ao finalizar o laudo, o médico pode visualizá-lo num modal para assegurar-se que o conteúdo está satisfatório antes de salvar o laudo no sistema. Como demonstrado no capítulo sobre o back-end, ao finalizar a gravação no banco de dados, o servidor enviará um e-mail ao paciente com um link para acesso ao laudo.

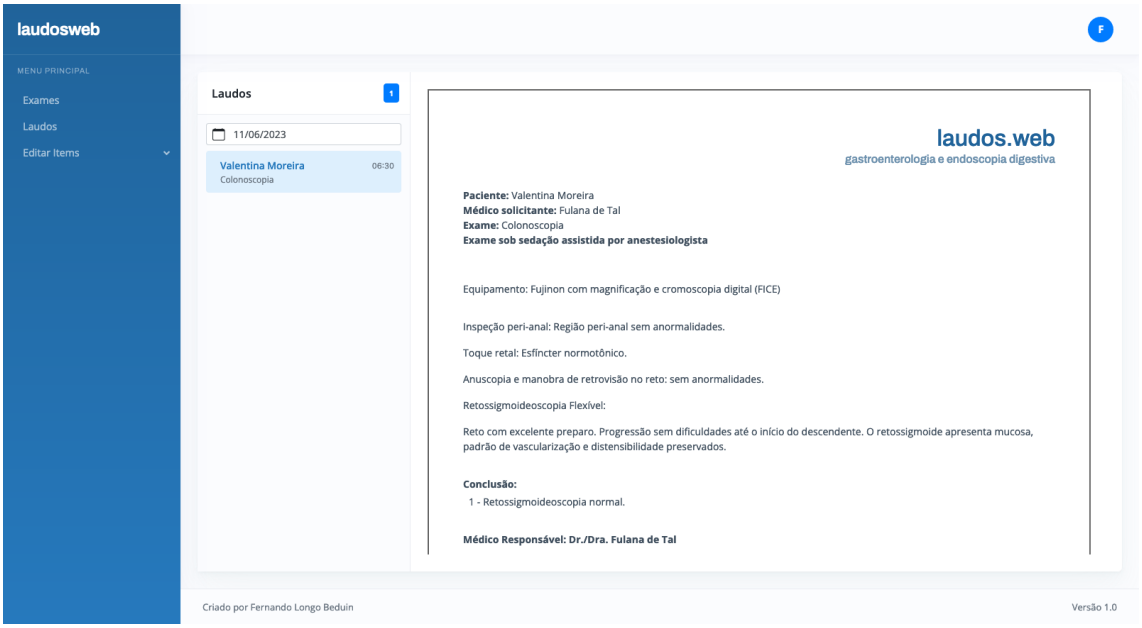


Figura 5.22: Tela laudos do **laudosweb**

Fonte: Autor

Caso seja necessário, o médico pode consultar laudos já criados a partir da opção **Laudos** no menu à esquerda. Na Figura 5.22, podemos ver um laudo criado a partir da aplicação. Nesse caso, o componente responsável pela renderização do documento é o mesmo utilizado na visualização da prévia do laudo durante a confecção e na página disponibilizada ao paciente - caracterizando a reutilização de componentes, uma das grandes vantagens no uso do Vue.js.

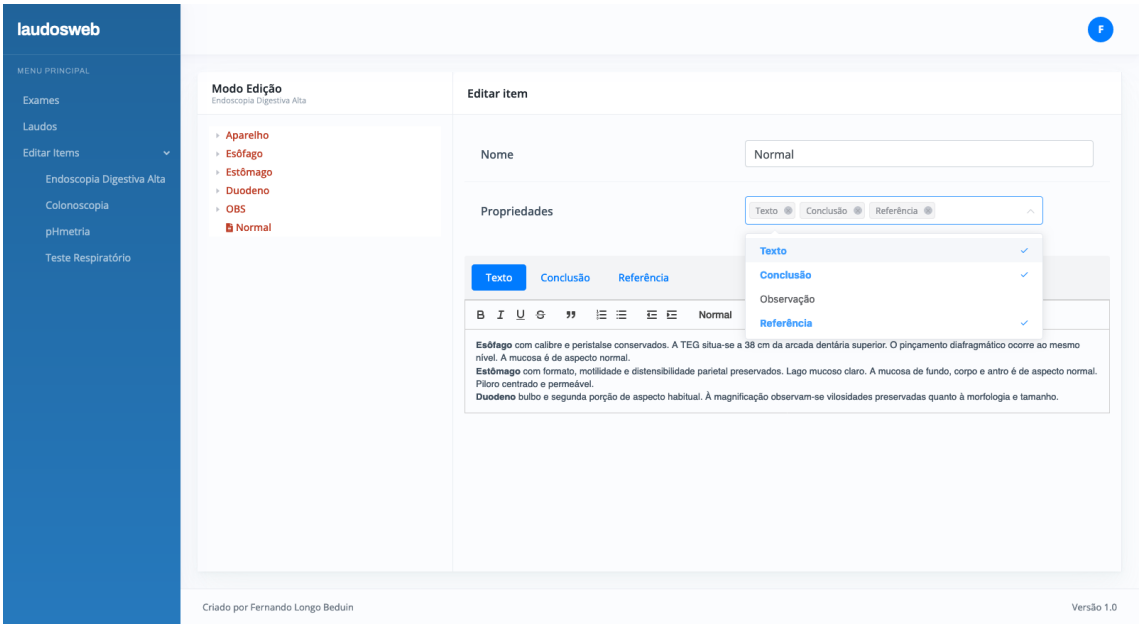


Figura 5.23: Tela edição de templates do **laudosweb**

Fonte: Autor

Por último, mas não menos importante, a Figura 5.23 mostra a tela de edição de templates, onde é possível criar, modificar e excluir itens de texto que são utilizados na confecção dos laudos. O componente árvore exibido à esquerda é muito parecido com o da criação dos laudos, porém incrementado com menus de contexto que permitem realizar as operações de edição acima descritas. Na Figura 5.23, podemos observar um item do template sendo editado.

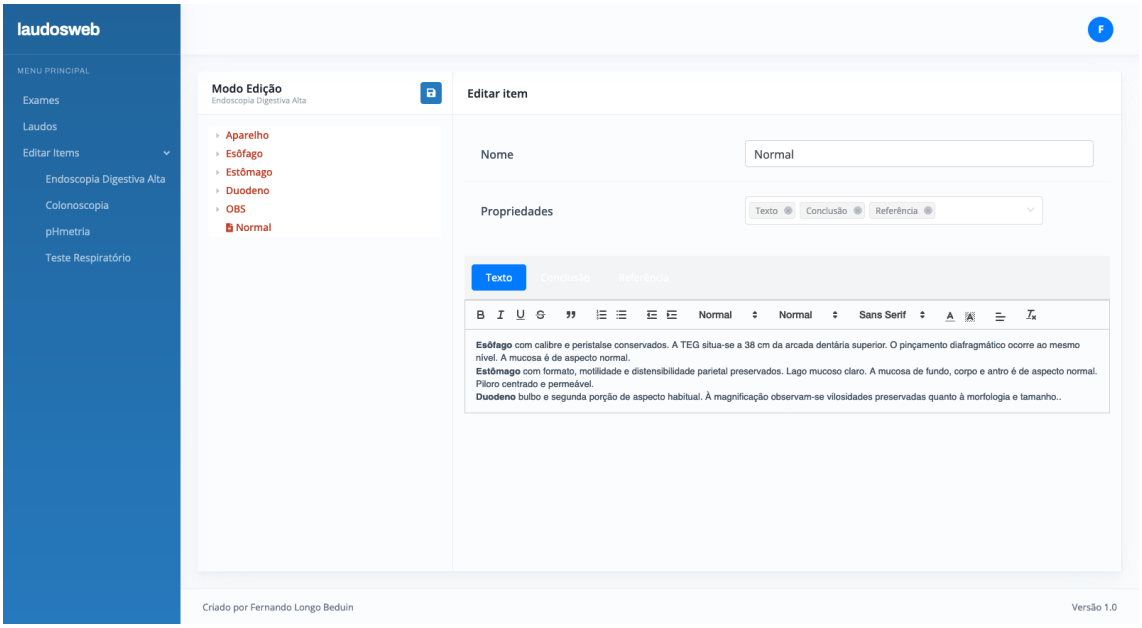


Figura 5.24: Tela edição de templates do **laudosweb**

Fonte: Autor

Ao realizar alguma alteração, o botão para salvá-la aparecerá logo ao lado do cabeçalho, como mostrado na Figura 5.24. O componente, ao comparar a árvore original com a modificada, cria um JSON Patch (Figura 5.25) que será enviado ao servidor para alterar o documento no banco de dados utilizando o método HTTP PATCH.

```
[
  { "op": "test", "path": "/a/b/c", "value": "foo" },
  { "op": "remove", "path": "/a/b/c" },
  { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },
  { "op": "replace", "path": "/a/b/c", "value": 42 },
  { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },
  { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }
]
```

Figura 5.25: Exemplo de arquivo JSON Patch

Fonte: [IETF 2013]

O formato JSON Patch é um padrão de arquivo que contém uma sequência de operações a serem aplicadas a um documento com o objetivo de atualizá-lo parcialmente sem a necessidade de substituir o documento inteiro. Dessa forma, é possível reduzir significativamente o *payload* de uma requisição, otimizando o tráfego de dados na rede.

No nosso caso, como citado anteriormente, as duas árvores são comparadas e, utilizando os recursos de reatividade do Vue.js, geram o referido JSON PATCH a partir de uma biblioteca chamada **rfc6902**, que é justamente o nome da especificação do JSON Patch criado pela Internet Engineering Task Force (IETF).

```
1  const { connect } = require('../_helpers/mongodb');
2  const { updatesForPatch } = require('../_helpers/mongodb-patch');
3
4  ...
5
6  exports.applyPatch = async (especialidade, patches) => {
7      const db = await connect();
8      const original = await db.collection("col_template").findOne({"especialidade": especialidade});
9      const updates = updatesForPatch(patches, original);
10
11     for await (const update of updates) {
12         await db.collection("col_template").updateOne({"especialidade": especialidade}, update);
13     }
14 };
```

Figura 5.26: Código da função **applyPatch**
Fonte: Autor

Como demonstrado na Figura 5.26 (acima), ao receber o referido arquivo, o servidor o converte em um array de operações compatíveis com o MongoDB (linha 9) através da biblioteca **mongodb-patch**, e realiza as operações necessárias até transformar o objeto na camada de persistência equivalente ao objeto alterado pelo usuário (linhas 11 à 13).

6 Caso de uso do **laudosweb** em uma clínica médica

Este capítulo apresenta o caso de uso da aplicação desenvolvida neste trabalho em uma clínica médica de gastroenterologia de renome na cidade de Florianópolis, com o objetivo de testar a eficiência da ferramenta.

Em virtude de limitações de acesso aos sistemas da referida clínica, não foi possível integrar a nossa ferramenta aos mesmos, uma das características relevantes no contexto da aplicação desenvolvida. Nesse caso, foi utilizado somente o componente de criação de laudos de exames com base nos templates de texto predefinidos, com o texto gerado pela nossa aplicação sendo copiado e colado dentro da ferramenta usada pela clínica.

Mesmo considerando essa limitação no uso de suas funcionalidades, a ferramenta se mostrou eficiente se compararmos aos processos adotados anteriormente pelos médicos - os laudos eram digitados diretamente na ferramenta de gestão da clínica, sem nenhuma padronização ou ferramenta auxiliar.

Com base no depoimento de duas médicas que utilizaram o sistema durante o período de testes, o tempo médio necessário para a elaboração dos laudos diminuiu significativamente. Antes, os laudos levavam entre 5 a 10 minutos para serem concluídos, e agora são finalizados em cerca de 2 minutos. Foram destacadas como características positivas pelas usuárias a interface intuitiva e de fácil navegação, e a possibilidade de customização - tanto do texto produzido, quanto dos templates de texto utilizados como padrão.

Vale ressaltar que todos os templates utilizados pela ferramenta foram criados pelo próprio corpo médico da clínica, pois trata-se de conteúdo específico na área de conhecimento da mesma.

7 Conclusão

Neste trabalho de conclusão de curso foi apresentada uma aplicação web desenvolvida com o propósito de auxiliar uma clínica médica na geração de laudos médicos, tendo assim alcançado êxito no cumprimento do objetivo principal. O aplicativo foi testado no contexto de uma clínica, e os resultados obtidos foram satisfatórios quando comparados com as práticas realizadas anteriormente, onde foi possível observar que a padronização e facilidade na seleção de templates tornou o processo de criação de laudos mais eficiente.

Entre os objetivos específicos, cabe destacar que a utilização de tecnologias web recentes facilitou muito o desenvolvimento do protótipo da aplicação, pois estas oferecem um vasto ecossistema de ferramentas e bibliotecas bem consolidados pela comunidade de desenvolvedores. Tal prática também demonstrou um compromisso com a inovação tecnológica, garantindo a relevância da ferramenta em possíveis melhorias futuras. O uso de um framework reativo, no caso o Vue.js, permitiu oferecer uma interface intuitiva e interativa ao usuário, criada com poucas linhas de código - o que acelerou o processo de criação.

A facilidade de customização também foi destaque no caso de uso citado no capítulo anterior, já que a própria clínica médica conseguiu criar os seus templates baseados no conhecimento específico da área de atuação da mesma.

Como limitação, não conseguimos integrar o aplicativo a uma base externa por falta de acesso à ferramenta comercial utilizada pela clínica onde foram realizados os testes da ferramenta.

7.1 Trabalhos Futuros

Durante o desenvolvimento deste trabalho, foram abordados os aspectos considerados essenciais para a eficiência e usabilidade do aplicativo web proposto. No entanto, sempre há espaço para futuras expansões e melhorias no sistema.

Uma grande melhoria seria a inclusão de uma versão responsiva da interface do aplicativo, tornando-o acessível também em plataformas móveis e assim permitindo uma maior flexibilidade ao médico na confecção dos laudos. Funcionalidades de customização do layout do laudo também poderiam ser desenvolvidas com o intuito de oferecer maior autonomia à clínica em personalizações. Outros tipos de filtros também poderiam ser agregados às telas de laudos realizados e exames, tais como pesquisas por especialidade, médico, etc. Além disso, poderiam ser gerados relatórios que proporcionassem aos gestores uma visão consolidada e detalhada para ajudar na tomada de decisões.

Como citado no capítulo 3.3 em um dos softwares relacionados (Laudite), a possibilidade de geração de documentos a partir do reconhecimento de voz poderia agregar valor no aspecto da eficiência na geração dos laudos.

8 Referências

Amazon Web Services. "What is a Web Application?"
Disponível em: https://aws.amazon.com/what-is/web-application/
Acessado em: maio/2023

Brasil. Presidência da República. (2018).
"Lei nº 13.709. Lei Geral de Proteção de Dados Pessoais (LGPD)."
Disponível em:https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm
Acessado em: maio/2023

Brasil.js. “História do JavaScript.”
Disponível em: https://brasil.js.org/#historia
Acessado em: junho/2023

FBF Sistemas. “Laudomatic 2.0”
Disponível em: https://www.fbfsistemas.com/pericias.html
Acessado em: maio/2023

IBM. "Three-tier architecture."
Disponível em: https://www.ibm.com/topics/three-tier-architecture/
Acessado em: maio/2023

Internet Engineering Task Force. (2013).
"RFC 6902: JavaScript Object Notation (JSON) Patch."
Disponível em: https://datatracker.ietf.org/doc/html/rfc6902
Acessado em: abril/2023

Joi API Documentation. (2021).
Disponível em: https://joi.dev/api/?v=17.9.1
Acessado em: maio/2023

Laudite. (2021)
Disponível em: https://laudite.com.br/
Acessado em: maio/2023

Ministério da Saúde. (2020). "Estratégia de Saúde Digital para o Brasil."
Disponível em:
https://bvsms.saude.gov.br/bvs/publicacoes/estrategia_saude_digital_Brasil.pdf
Acessado em: maio/2023

MongoDB, Inc. (2016). "MongoDB: The Definitive Guide."
Disponível em:
https://pepa.holla.cz/wp-content/uploads/2016/07/MongoDB-The-Definitive-Guide.pdf
Acessado em: maio/2023

Mozilla Developer Network (MDN). “Glossário JavaScript.”
Disponível em: https://developer.mozilla.org/pt-BR/docs/Glossary/JavaScript
Acessado em: junho/2023

Node.js. "Event Loop."
Disponível em: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/
Acessado em: maio/2023

npm. Joi.
Disponível em: https://www.npmjs.com/package/joi
Acessado em: maio/2023

OWASP. (2021). “A07:2021-Identification and Authentication Failures”
Disponível em:
https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures
Acessado em: abril/2023

Queo Sistemas Ltda. “Turing”
Disponível em: https://www.queo.com.br/wp/
Acessado em: maio/2023

Stack Overflow (2022). "Most Popular Technologies: Language."
Disponível em:
https://survey.stackoverflow.co/2022/#most-popular-technologies-language
Acessado em: maio/2023

Tutorial and Example (2020). "Node.js Event Loop."
Disponível em: https://www.tutorialandexample.com/node-js-event-loop/
Acessado em: maio/2023

Viana, Jesiel (2022). "Arquitetura de Aplicações Web."
Disponível em:
https://jesielviana.gitbook.io/webdev/introducao/arquitetura-de-aplicacoes-web
Acessado em: maio/2023

Vue.js. "Introdução - Guia do Vue.js."
Disponível em: https://vuejs.org/guide/introduction.html
Acessado em: maio/2023

Webopedia (2021). “Web Application”
Disponível em: https://www.webopedia.com/definitions/web-application/
Acessado em: maio/2023

APÊNDICE A – ARTIGO NO FORMATO SBC

Aplicação web para geração de laudos médicos

Fernando L. Beduin[1], Frank A. Siqueira[1]

[1]Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística — INE, Florianópolis, SC — Brasil

fernando.beduin@gmail.com, frank.siqueira@ufsc.br

Abstract. *This paper presents the development of a web application for the generation of medical reports that facilitated the work of the doctor and standardized the documents generated in the medical clinic. As the main functionalities of the application, it is possible to highlight the creation of reports from predefined text templates and the editing of these templates by the clinic's medical staff.*

Resumo. *Este trabalho apresenta o desenvolvimento de um aplicativo web para a geração de laudos médicos que permita facilitar o trabalho do médico e padronizar os documentos gerados numa clínica médica. Como principais funcionalidades da ferramenta, é possível destacar a criação de laudos a partir de modelos de textos pré-definidos e a edição desses modelos pelo corpo médico da clínica.*

1. Introdução

O setor da saúde tem se beneficiado das constantes transformações tecnológicas, visando aprimorar os processos e oferecer um atendimento de qualidade aos pacientes [MS 2020]. Nesse contexto, as aplicações web têm se tornado cada vez mais relevantes para otimizar as atividades nas clínicas médicas. No entanto, a geração de laudos médicos de forma manual enfrenta desafios relacionados à lentidão e falta de padronização.

Este trabalho propõe o desenvolvimento de uma aplicação web que permita aos médicos registrar, editar e compartilhar laudos de forma rápida e precisa, superando esses obstáculos. A aplicação promove a padronização dos laudos, garantindo maior qualidade e consistência nas informações geradas. Os objetivos específicos incluem o uso das mais recentes tecnologias web, a integração com uma base de dados externa e a customização fácil da aplicação para qualquer clínica, independentemente da área de atuação.

2. Metodologia

A metodologia adotada neste trabalho consistirá nas seguintes etapas: levantamento de requisitos, definição da arquitetura, implementação da solução tecnológica e avaliação.

O levantamento de requisitos será realizado com base nas necessidades do corpo médico de uma clínica médica, documentando-os de forma clara e precisa. Em seguida, a arquitetura do sistema será definida, considerando os requisitos levantados e as

melhores práticas no desenvolvimento de aplicações web, visando a escalabilidade e segurança da solução. A implementação do aplicativo envolverá a modelagem de dados e o desenvolvimento das camadas de aplicação e apresentação. Por fim, a ferramenta será avaliada em uma clínica médica real, coletando feedback dos usuários para avaliar a usabilidade e eficiência da solução tecnológica.

3. Solução proposta

A solução proposta neste trabalho é o desenvolvimento de uma aplicação web chamada **laudosweb**, no formato SPA (Single-Page Application). A aplicação permitirá a criação de laudos médicos com base em exames realizados por uma clínica, utilizando templates pré-definidos de textos organizados por categorias para facilitar a seleção de itens e padronizar os documentos gerados. Os laudos serão disponibilizados em formato digital aos pacientes.

A lista de requisitos funcionais inclui o cadastro de médicos, o acesso ao sistema, o recebimento e armazenamento de registros de exames, a criação de laudos, a criação e edição de templates de texto, a disponibilização de laudos aos pacientes, a consulta de laudos e a visualização de laudos pelos pacientes. Além disso, foram identificados requisitos não funcionais relacionados à usabilidade, disponibilidade, segurança e integração da aplicação.

A arquitetura da solução utiliza a stack MEVN (MongoDB, Express.js, Vue.js e Node.js), proporcionando uma implementação eficiente e escalável.

4. Implementação da solução

A implementação da solução proposta foi dividida em quatro etapas principais: modelagem dos dados, desenvolvimento da API REST (camada de aplicação), a integração com uma API de dados externa e a criação da interface com o usuário (camada de apresentação).

4.1. Modelagem dos dados

Na etapa de modelagem dos dados, foram utilizados documentos JSON para representar as coleções do banco de dados MongoDB. Foram definidos modelos de documentos para as entidades de Usuários, Exames, Templates e Laudos. A estrutura dos documentos foi apresentada, mostrando exemplos de cada coleção.

4.2. Desenvolvimento da API REST

Na camada de aplicação, foi utilizado o Node.js com o framework Express.js para criar a API REST. A estrutura de pastas e arquivos foi organizada de forma a separar as responsabilidades, incluindo as pastas de rotas, controladores, services e DAO. Foram utilizadas bibliotecas como Joi para validação de dados, bcrypt para encriptação de senhas, jsonwebtoken para autenticação com JWT e mongodb para a interface com o banco de dados MongoDB.

Foram definidas as rotas da API para as entidades de login, exame, laudo e template, com os respectivos métodos HTTP e parâmetros, conforme Tabela 1 (abaixo).

Tabela 1. Rotas da API REST

Rota	Método	Descrição
api/login/	POST	Rota responsável pelo login do usuário.
api/exame	GET	Rota responsável por trazer a lista de exames não laudados do médico responsável pelo exame.
api/exame/{id}	GET	Rota responsável por trazer um documento exame pelo id do mesmo.
api/laudo/?{dt_laudo}	GET	Rota responsável por trazer a lista de laudos criados por data.
api/laudo/{id}	GET	Rota responsável por trazer um laudo pelo id do mesmo.
api/laudo/	POST	Rota responsável por criar um laudo.
api/template/{especialidade}	GET	Rota responsável por trazer os templates associados a uma especialidade.
api/template/{especialidade}	PATCH	Rota responsável por modificar templates associados a uma especialidade.

A validação dos dados recebidos nas requisições foi realizada utilizando a biblioteca Joi, visando garantir a integridade das informações.

Para a autenticação e segurança, foram adotadas medidas para proteger os dados e controlar o acesso aos recursos da API. Isso inclui a utilização de autenticação com JWT, criptografia de senhas e conformidade com regulamentos de proteção de dados, como a Lei Geral de Proteção de Dados [LGPD 2018].

Essas etapas de implementação visaram criar uma API REST eficiente e escalável, seguindo as melhores práticas e utilizando tecnologias adequadas para cada componente da solução.

4.3. Integração com API externa

A integração da aplicação web com uma API externa que disponibiliza dados de exames é realizada por meio de um serviço dentro da API REST, chamado a partir da rota "api/exame".

A função responsável por retornar os exames (getExamePorMedico) invoca outra função (getExamesExternos) responsável por acessar a API externa e salvar os dados obtidos no banco de dados. Somente após essa execução, a função que busca os exames no banco de dados (getExamePorMedico) é executada, garantindo que a lista de exames entregue à camada de apresentação da aplicação seja atualizada.

É importante ressaltar que a API externa utilizada é apenas ilustrativa, fornecendo dados fictícios de exames para demonstrar a integração descrita neste capítulo.

4.4. Criação da interface com o usuário

A camada de apresentação (front-end) do projeto foi desenvolvida utilizando o framework Vue.js, na forma de uma SPA (Single Page Aplication). Foi utilizada a biblioteca Vue Router para criar as rotas da aplicação, permitindo assim a transição entre seus diferentes componentes. O Axios foi utilizado para realizar as requisições HTTP, facilitando dessa forma a interação entre as camadas de apresentação e de aplicação.

As telas do aplicativo incluem a tela de login, lista de exames não laudados - onde é possível ver detalhes dos exames, tela de criação de laudo, lista de laudos criados com a visualização do documento gerado e tela de edição de templates de texto.

A tela de criação de laudo permite ao médico selecionar templates de texto organizados em uma árvore por categorias, editar o texto do laudo e visualizar uma prévia antes de salvá-lo no sistema (Figura 1).

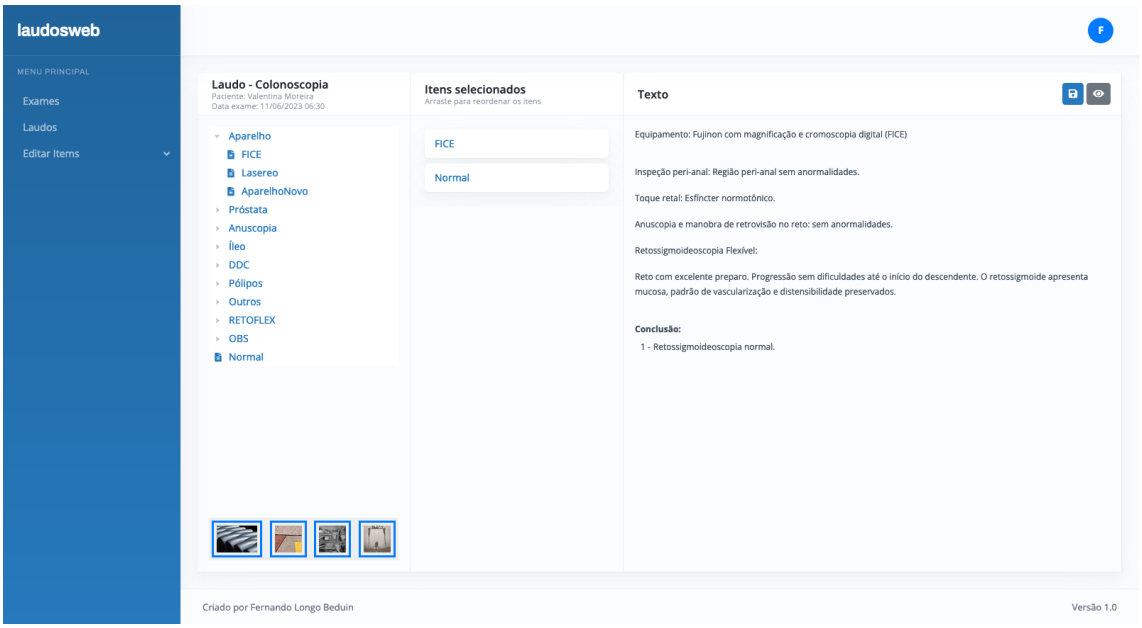


Figura 1. Tela de criação de laudo

A tela de edição de templates permite ao médico criar, modificar e excluir itens de texto utilizados na confecção dos laudos (Figura 2).

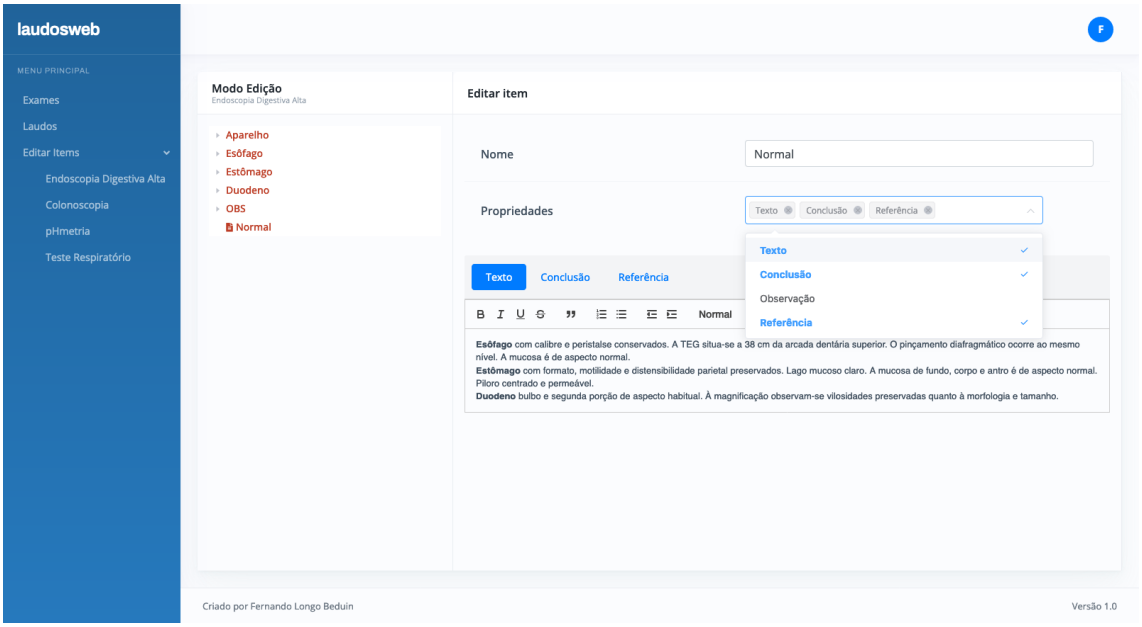


Figura 2. Tela de edição de templates de texto

No processo de edição de templates, é utilizado o formato JSON Patch para realizar alterações parciais no documento, reduzindo o tamanho das requisições e otimizando o tráfego de dados na rede. O servidor recebe o JSON Patch e aplica as operações correspondentes ao documento no banco de dados.

A camada de apresentação foi desenvolvida com o objetivo de proporcionar uma interface intuitiva e amigável aos usuários, utilizando recursos do Vue.js e bibliotecas complementares para facilitar o desenvolvimento da aplicação e garantir uma experiência de usuário agradável.

5. Caso de uso em uma clínica médica

Este capítulo descreve o caso de uso da aplicação **laudosweb** em uma clínica médica de gastroenterologia em Florianópolis. Devido a limitações de acesso aos sistemas da clínica, a integração da ferramenta não foi possível, o que limitou seu uso às funcionalidades de criação de laudos com base nos templates de texto predefinidos. O texto gerado pela aplicação foi copiado e colado na ferramenta utilizada pela clínica.

Apesar dessa limitação, a ferramenta demonstrou eficiência em comparação aos processos anteriores adotados pelos médicos. Antes, os laudos eram digitados diretamente na ferramenta de gestão de exames da clínica, sem padronização ou ferramentas auxiliares. Com a utilização do **laudosweb**, o tempo médio para a elaboração dos laudos diminuiu consideravelmente, passando de 5 a 10 minutos para cerca de 2 minutos.

As médicas que utilizaram o sistema durante os testes também destacaram a interface intuitiva e de fácil navegação como pontos positivos da aplicação, assim como a

possibilidade de personalização do texto produzido e dos templates de texto utilizados como padrão. É importante ressaltar que todos os templates foram criados pelo próprio corpo médico da clínica, levando em consideração o conhecimento específico da sua área de atuação.

6. Conclusão e trabalhos futuros

Neste trabalho, foi desenvolvida uma aplicação web para auxiliar uma clínica médica na geração de laudos médicos, alcançando sucesso em seu objetivo principal. A aplicação foi testada em uma clínica e os resultados foram satisfatórios, demonstrando maior eficiência em comparação com as práticas anteriores. A padronização e facilidade na seleção de templates tornaram o processo de criação de laudos mais eficiente.

A utilização de tecnologias web modernas facilitou o desenvolvimento do aplicativo, aproveitando um ecossistema consolidado de ferramentas e bibliotecas. O uso do framework Vue.js possibilitou a criação de uma interface intuitiva e interativa com poucas linhas de código.

Para trabalhos futuros, é sugerido desenvolver uma versão responsiva da interface para tornar o aplicativo acessível em dispositivos móveis. Funcionalidades de customização do layout do laudo e a inclusão de filtros adicionais também podem ser consideradas. Além disso, a geração de relatórios consolidados e a possibilidade de gerar documentos através do reconhecimento de voz podem agregar valor ao sistema proposto.

Referências

Ministério da Saúde. (2020) “Estratégia de Saúde Digital para o Brasil”, Em: https://bvsms.saude.gov.br/bvs/publicacoes/estrategia_saude_digital_Brasil.pdf

Brasil.Presidência da República. (2018) “Lei nº 13.709. Lei Geral de Proteção de Dados Pessoais (LGPD).”, https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm

APÊNDICE B – CÓDIGO FONTE

O código fonte da aplicação está disponível em:
https://github.com/fbeduin/tcclaudosweb