Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Desenvolvimento de Templates de Aplicações com Vue.js e Suas Variantes: Chameleon Stack

Victor de Oliveira Ferreira Polito

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
George Henrique Godim da Fonseca

Outubro, 2024
João Monlevade–MG

Victor de Oliveira Ferreira Polito

Desenvolvimento de Templates de Aplicações com Vue.js e Suas Variantes: Chameleon Stack

Orientador: George Henrique Godim da Fonseca

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Outubro de 2024

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Victor de Oliveira Ferreira Polito

Desenvolvimento de templates de aplicações com Vue.js e suas variantes: Chameleon Stack

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação

Aprovada em 16 de outubro de 2024

Membros da banca

Dr. George Henrique Godim da Fonseca - Orientador(a) - Universidade Federal de Ouro Preto
Dr. Fernando Bernardes de Oliveira - Universidade Federal de Ouro Preto
Dr. Igor Muzetti Pereira - Universidade Federal de Ouro Preto

George Henrique Godim da Fonseca, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 19/10/2024

Documento assinado eletronicamente por **George Henrique Godim da Fonseca**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 19/10/2024, às 12:12, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0797904** e o código CRC **2AAE4F48**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Este trabalho é dedicado primeiramente à Deus por me dar a oportunidade de poder estar realizando este trabalho; meus avós, que sempre foram meus pilares e minha família, que sempre me apoiou!

Agradecimentos

A jornada foi longa, mas a recompensa valeu cada esforço. Agradeço à minha namorada, pelo suporte e estar sempre ali, aos amigos e colegas, por compartilharem momentos inesquecíveis e aos professores, pelo conhecimento e orientação. Essa conquista é o resultado da soma de cada um de vocês em minha vida. Obrigado!

"Great things are done by a series of small things brought together."

— Vincent Van Gogh (1853 – 1890),
in: letter to his brother, Theo.

Resumo

Este trabalho tem como objetivo criar uma aplicação *web* para auxiliar equipes de tecnologia e programadores no planejamento e organização do uso das ferramentas e tecnologias do projeto e seu andamento. O projeto Chameleon visa simplificar não apenas a seleção das tecnologias a serem usadas, mas também facilitar a implementação de funcionalidades como autenticação, operações CRUD e outras. O Chameleon foi desenvolvido utilizando três bibliotecas de componentes do *Vue.js*: *Vuetify*, *Element+* e *Ant-Design*, permitindo a criação de três variantes visuais distintas da aplicação. As vantagens dessas bibliotecas incluem a vasta gama de componentes pré-construídos e customizáveis, além da integração nativa com o ecossistema *Vue.js*, o que acelera o desenvolvimento. Contudo, algumas desvantagens foram observadas, como a curva de aprendizado para a personalização entre elas e as diferenças no estilo de design entre as bibliotecas. Esse projeto é parte integrante de um esforço mais amplo, cujo objetivo será incorporar diversas tecnologias; entretanto, o foco deste trabalho está em Vue.js e suas variantes.

Palavras-chaves: Chameleon. CRUD. vue.js. element+. Vuetify. Ant-Design

Abstract

This project aims to create a web application to assist technology teams and developers in planning and organizing the use of project tools, technologies, and its progress. The Chameleon project seeks to simplify not only the selection of technologies to be used but also to facilitate the implementation of functionalities such as authentication, CRUD operations, and more. Chameleon was developed using three Vue.js component libraries: Vuetify, Element UI, and Ant-Design, allowing the creation of three distinct visual variants of the application. The advantages of these libraries include a wide range of pre-built and customizable components, along with native integration with the Vue.js ecosystem, which speeds up development. However, some disadvantages were observed, such as the learning curve for customization and differences in design style between the libraries. This project is part of a broader effort to incorporate various technologies, but the focus of this work is on Vue.js and its variants.

Key-words: Chameleon. CRUD. vue.js. element+. Vuetify. Ant-Design

Lista de ilustrações

Figura 1 – Dados sobre a popularidade do Vue.js na pesquisa *Stack Overflow Developer Survey*. 14
Figura 2 – Usabilidade dos *frameworks front-end*. 18
Figura 3 – Plataforma T3 Stack. 19
Figura 4 – Plataforma Bunny Shell. 20
Figura 5 – Plataforma IDX. 21
Figura 6 – Modelo Relacional do funcionamento do *back-end*. 27
Figura 7 – Tela inicial da plataforma *Vuetify*. 30
Figura 8 – Tela inicial *Element+*. 31
Figura 9 – Tela inicial *Ant-Design*. 32
Figura 10 – Protótipo *Landing Page*. 33
Figura 11 – Modelo Inicial do Projeto. 35
Figura 12 – Tela de *login*. 36
Figura 13 – Tela de criação de conta para novos usuários. 37
Figura 14 – Tela inicial. 38
Figura 15 – Modal para criação de uma nova tarefa. 39
Figura 16 – Modal para criação de uma nova categoria. 39
Figura 17 – Modal para edição de uma tarefa. 40
Figura 18 – Tela inicial atualizada. 41
Figura 19 – Modal para excluir de uma tarefa. 41

Lista de tabelas

Tabela 1 – Requisitos não funcionais do sistema. 28
Tabela 2 – Comparação entre bibliotecas de componentes Vue.js 32
Tabela 3 – Comparação entre uso de linguagens para personalização das bibliotecas de componentes Vue.js 42
Tabela 4 – Requisitos Funcionais da Tela de Login 51
Tabela 5 – Requisitos Funcionais da Tela Home Page 51

Lista de abreviaturas e siglas

API *Application Programming Interface*

CI/CD Integração Contínua/Entrega Contínua

CRUD *Create*, *Read*, *Update* e *Delete*

CSS *Cascading Style Sheets*

DOM *Document Object Model*

HTML *Hypertext Markup Language*

HTTP *Hypertext Transfer Protocol*

MIT Massachusetts Institute of Technology

MVC *Model, View, Controller*

SPA *Single-Page Applications*

UI *User Interface*

VSCode *Visual Studio Code*

Sumário

1 INTRODUÇÃO 13
1.1 O problema de pesquisa 15
1.2 Objetivos 15
1.3 Metodologia 16

2 REVISÃO BIBLIOGRÁFICA 17
2.1 Trabalhos Correlatos 17
2.1.1 Plataformas de Geração Automática de *Templates* 19
2.2 Importância da Padronização e Boas Práticas 21
2.3 Considerações 21

3 DESENVOLVIMENTO E REQUISITOS 23
3.1 Conceitos Básicos 23
3.1.1 Editor de Texto 23
3.1.2 Boas Práticas de Desenvolvimento: ESLint 23
3.1.3 Git/Github 24
3.1.4 Back-end 24
3.1.5 API 24
3.2 Arquitetura do Projeto 25
3.2.1 Modelo Relacional 25
3.2.1.1 Entidades 25
3.2.1.2 Relacionamentos 25
3.2.1.3 Funcionamento dos Relacionamentos 26
3.2.1.4 Exemplo de Uso 26
3.3 Especificações do Sistema 27
3.3.1 Requisitos Funcionais 27
3.3.2 Requisitos Não Funcionais 28
3.4 Desenvolvimento 28
3.4.1 Funcionamento do Modelo 29
3.5 Variações 30
3.5.1 Vuetify 30
3.5.2 Element+ 30
3.5.3 Ant-Design 31
3.5.4 Comparações 32
3.6 Considerações 33

4 RESULTADOS 35
4.1 Tela de Login 35
4.1.1 Criação de Usuário 36
4.2 Tela Home Page 37
4.2.1 Cards 38
4.3 Modais 38
4.3.1 Criação de Nova Tarefa 38
4.3.2 Criação de Nova Categoria 39
4.3.3 Edição de Cards 40
4.4 Tela Home Page - Com cards 40
4.4.1 Exclusão de Cards 41
4.5 Considerações Finais 41
4.5.1 Discussão e Recomendação Geral 44

5 CONCLUSÃO 46
5.1 Trabalhos Futuros 46

REFERÊNCIAS 48

APÊNDICES 50

APÊNDICE A – REQUISITOS FUNCIONAIS 51
A.1 Tela de Login 51
A.2 Tela Home Page 51

1 Introdução

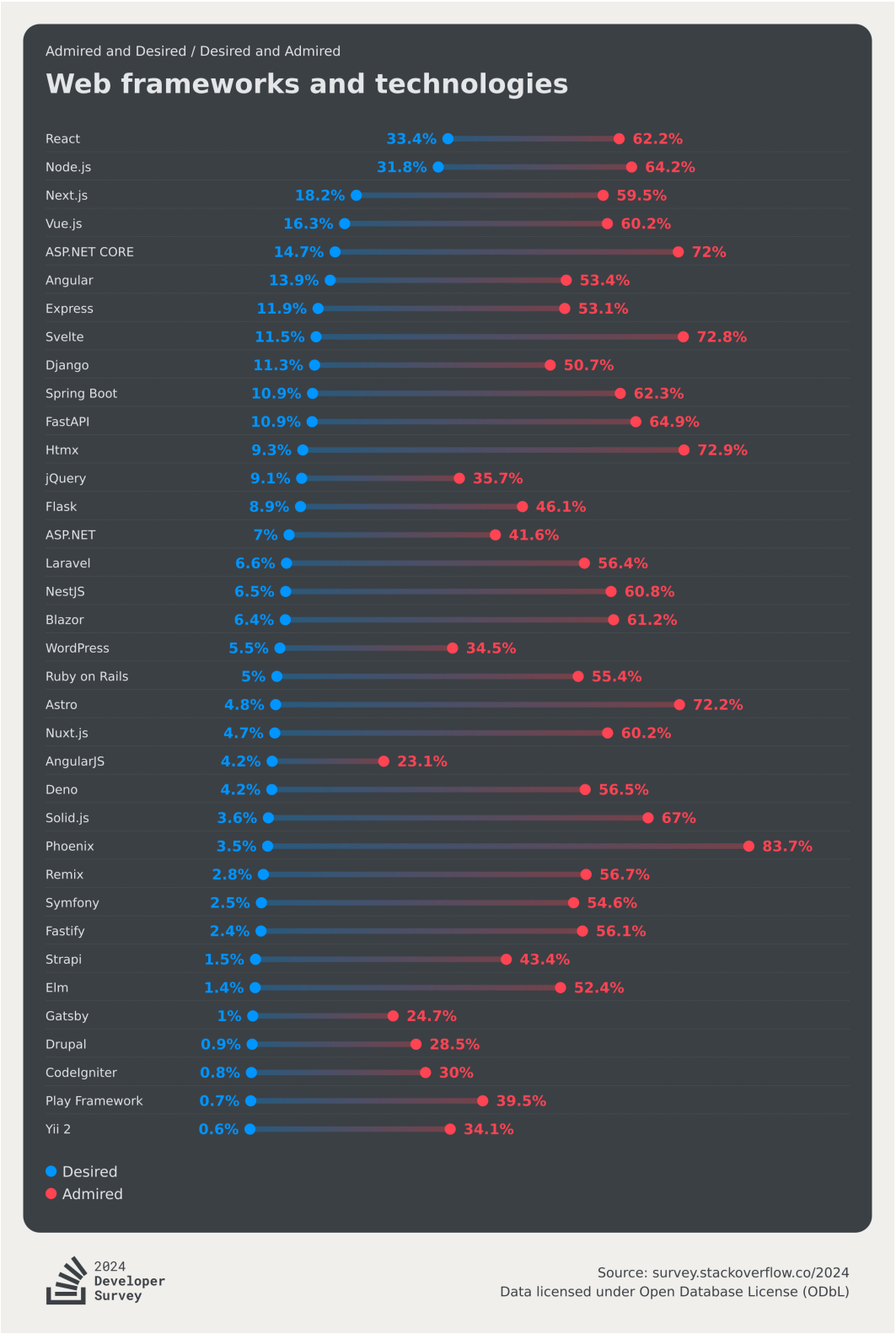
O planejamento de aplicações, sejam elas *mobile*, *web* ou *desktop*, pode ser uma tarefa complexa, especialmente ao definir quais tecnologias e *frameworks* serão utilizadas no projeto. Existem diversas combinações de tecnologias que atendem a diferentes objetivos. Essas combinações são conhecidas como *stacks*.

Nesse contexto, a criação de *templates* para operações *Create*, *Read*, *Update* e *Delete* (CRUD) se apresenta como uma estratégia eficiente para agilizar o desenvolvimento e assegurar a consistência entre projetos. Este trabalho apresenta o desenvolvimento de um gerador de *templates* CRUD utilizando o *framework* *Vue.js*, com o suporte de três distintas bibliotecas de componentes: *Element+*, *Ant-Design* e *Vuetify*.

A escolha do *Vue.js* como base para este gerador se fundamenta em sua flexibilidade e popularidade crescente (conforme a pesquisa do *Stack Overflow*[1] de 2024, exibida na Figura 1), permitindo a criação de interfaces de usuário reativas e de alta performance. As bibliotecas de componentes selecionadas oferecem um conjunto abrangente de elementos pré-construídos, facilitando a implementação de interfaces visuais robustas e consistentes. *Element+*, *Ant-Design* e *Vuetify* foram escolhidas por suas características específicas e ampla aceitação na comunidade de desenvolvedores.

[1] Este "*Admired and Desired Technologies Webframe*" é disponibilizado sob a Open Database License: <http://opendatacommons.org/licenses/odbl/1.0/>. Quaisquer direitos sobre conteúdos individuais do banco de dados são licenciados sob a Database Contents License: <http://opendatacommons.org/licenses/dbcl/1.0/>

Figura 1 – Dados sobre a popularidade do Vue.js na pesquisa *Stack Overflow Developer Survey*.



Fonte: Adaptado de *Stack Overflow Developer Survey* (2024).

O desenvolvimento desta ferramenta visa facilitar o trabalho das equipes de desenvolvimento, além de fomentar a adoção de boas práticas de programação, padronizando e organizando o código gerado. Espera-se que a ferramenta se torne um recurso útil no cotidiano dos desenvolvedores, promovendo eficiência e qualidade nos projetos de software.

1.1 O problema de pesquisa

No cenário atual de desenvolvimento de software, equipes enfrentam desafios que podem comprometer a eficiência e a qualidade dos projetos. Um dos mais recorrentes é a demora para iniciar o trabalho de forma produtiva, muitas vezes decorrente de um planejamento inadequado. A falta de clareza nas etapas iniciais pode gerar retrabalho, aumento de custos e atrasos no cronograma. Brooks (1975), em "*The Mythical Man-Month*", alerta sobre os perigos de adicionar mais pessoas a um projeto atrasado, enfatizando que essa prática pode agravar os problemas.

A importância de um bom planejamento é destacada por McConnell (1996), que aponta que a falta de uma visão clara do projeto desde o início pode resultar em cronogramas irrealistas e expectativas desalinhadas. Além disso, Boehm (1988), com o Modelo Espiral, argumenta que um planejamento iterativo e incremental é essencial para a minimização de riscos e para a melhoria contínua durante o desenvolvimento.

O movimento ágil, surgido no início dos anos 2000, mudou a abordagem tradicional de desenvolvimento de software. Segundo o *Manifesto Ágil*, a colaboração constante com o cliente, a resposta rápida a mudanças e a entrega contínua de software em funcionamento são elementos fundamentais para o sucesso dos projetos (BECK, 2001). Metodologias ágeis, como *Scrum* e *Kanban*, enfatizam ciclos curtos de desenvolvimento, *feedback* rápido e flexibilidade para ajustar o planejamento, o que torna o desenvolvimento mais adaptável.

Relatórios como o *CHAOS Report* do *Standish Group* apontam que uma parte significativa dos projetos de TI falha devido a problemas de planejamento e gerenciamento inadequado, reforçando a necessidade de estratégias eficazes desde o início (GROUP, 1994). Hunt e Thomas (1999), em "*The Pragmatic Programmer*", também abordam a importância de práticas estruturadas para evitar desafios comuns no início dos projetos.

Diante desses pontos, o objetivo geral deste trabalho é desenvolver um gerador de *templates* CRUD utilizando o *framework Vue.js*.

1.2 Objetivos

Como apresentado na Seção 1.1, o trabalho consiste no desenvolvimento de um gerador de *templates*. O objetivo principal é facilitar o trabalho das equipes de desenvolvi-

mento de software, padronizando e agilizando a criação de interfaces CRUD, essenciais para a maioria das aplicações *web*.

Os objetivos específicos deste trabalho são:

- Desenvolver diferentes geradores de *templates* CRUD para que o usuário escolha entre as bibliotecas de componentes *Element+*, *Ant-Design* e *Vuetify*.
- Avaliar as três bibliotecas de componentes utilizadas, destacando suas características, vantagens e desvantagens em termos de usabilidade e performance.
- Documentar todo o processo de desenvolvimento, desde a concepção até a implementação, fornecendo um manual para a utilização da ferramenta.

1.3 Metodologia

Os passos para execução deste trabalho são definidos da seguinte forma:

- **Revisão da literatura**: Realizar uma análise de obras e artigos sobre desenvolvimento de software, *frameworks front-end* e bibliotecas de componentes. Essa revisão visa embasar teoricamente o desenvolvimento do gerador de *templates* e fornecer uma compreensão clara dos desafios enfrentados pelas equipes no início dos projetos.
- **Desenvolvimento do gerador de *templates* CRUD**: Implementar a ferramenta utilizando o *framework Vue.js*, incorporando as bibliotecas de componentes *Element+*, *Ant-Design* e *Vuetify*. Essa etapa envolve a codificação, integração e customização dos *templates* para atender a diferentes necessidades de projetos.
- **Validação do gerador:** Aplicar a ferramenta desenvolvida em estudos de caso e realizar testes práticos para verificar sua funcionalidade, eficiência e usabilidade.

2 Revisão bibliográfica

Este capítulo apresenta uma revisão de literatura sobre os conceitos fundamentais que embasam o desenvolvimento de sistemas CRUD, *frameworks front-end* e bibliotecas de componentes, além de examinar trabalhos correlatos e ferramentas semelhantes, como a plataforma "*T3 Stack*". A revisão também incluirá discussões sobre a importância de boas práticas de desenvolvimento e a padronização de código para garantir a eficiência de desenvolvimento de software.

- *Frameworks Front-End*

  Os *frameworks front-end* são ferramentas que permitem a criação de interfaces de usuário (*User Interface* - UI) mais ágeis, padronizadas e de fácil manutenção devido a sua divisão e renderização em componentes. O *Vue.js*, um dos principais *frameworks* nos dias de hoje, tem se destacado por sua flexibilidade e desempenho, sendo amplamente utilizado em aplicações modernas. Sua arquitetura baseada em componentes modulares possibilita a reutilização de código, tornando o desenvolvimento mais eficiente (YOU, 2014).

- Bibliotecas de Componentes

  As bibliotecas de componentes, como *Element+*, *Ant-Design* e *Vuetify*, complementam os *frameworks*, como o *Vue.js*, ao fornecer um conjunto de elementos pré-construídos e estilizados para facilitar a criação de interfaces ricas e responsivas. Essas bibliotecas permitem que os desenvolvedores se concentrem na lógica de negócios, enquanto garantem que a interface visual siga padrões de design consistentes e de alta qualidade.

  Element+ é conhecida por seu design minimalista e extensível, sendo ideal para projetos com necessidades de customização detalhada (ELEMENT+, 2024). *Ant-Design*, por sua vez, se destaca pela quantidade de componentes robustos e pela documentação extensiva, muito utilizada em ambientes corporativos (ANT-DESIGN, 2024). Já *Vuetify*, construído com base no *Material Design*, é popular por sua simplicidade e por seguir rigorosamente as diretrizes de *design* da Google, sendo amplamente aceito pela comunidade de desenvolvedores (VUETIFY, 2024).

2.1 Trabalhos Correlatos

Diversos estudos sobre automação no desenvolvimento de software e o uso de *frameworks front-end* apontam para a importância da padronização e da automação como formas de melhorar a eficiência e reduzir os riscos associados ao desenvolvimento manual.

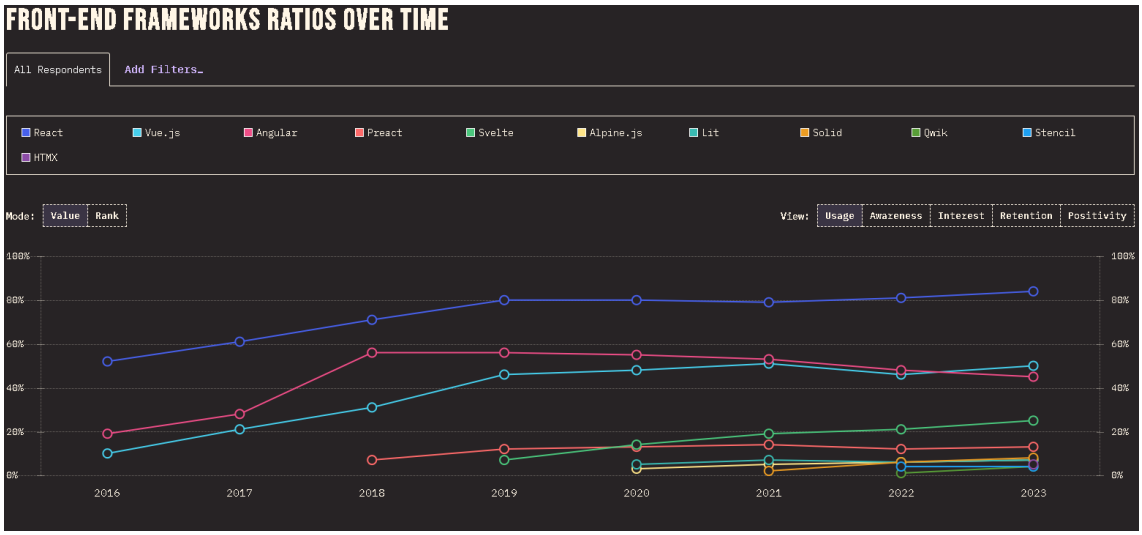
Segundo Kitchenham (2004), a adoção de ferramentas automatizadas pode ajudar as equipes a evitarem inconsistências comuns, permitindo uma abordagem mais sistemática na implementação de interfaces e operações CRUD.

Além disso, plataformas de automação no desenvolvimento, como a *T3 Stack*, têm ganhado destaque por sua capacidade de acelerar o processo de criação de aplicações *full-stack*. Ao integrar tecnologias modernas, como *TypeScript* e *Tailwind CSS*, essas ferramentas não só padronizam o código, mas também garantem uma maior escalabilidade e manutenção, conforme descrito por Forsgren (2019).

Outro ponto relevante abordado na literatura é a questão da flexibilidade em ambientes de desenvolvimento, um fator ressaltado por Fowler (2004) em "*Patterns of Enterprise Application Architecture*", que argumenta que a automação deve ser adaptável às necessidades específicas de cada equipe e projeto. Nesse sentido, geradores automáticos de templates como os discutidos neste trabalho permitem uma customização sem comprometer a padronização, oferecendo um equilíbrio entre consistência e flexibilidade.

Estudos mais recentes, como o StateOfFrontend (2023), identificam que a automação no desenvolvimento *front-end* é uma tendência crescente, e o *Vue.js* vem crescendo em uso entre desenvolvedores, como apresentado em . Além disso, plataformas como *T3 Stack*, *Bunny Shell* e *Google IDX* oferecem soluções eficazes para otimizar o desenvolvimento de interfaces dinâmicas e operações CRUD, superando desafios relacionados à padronização e ao retrabalho.

Figura 2 – Usabilidade dos *frameworks front-end*.



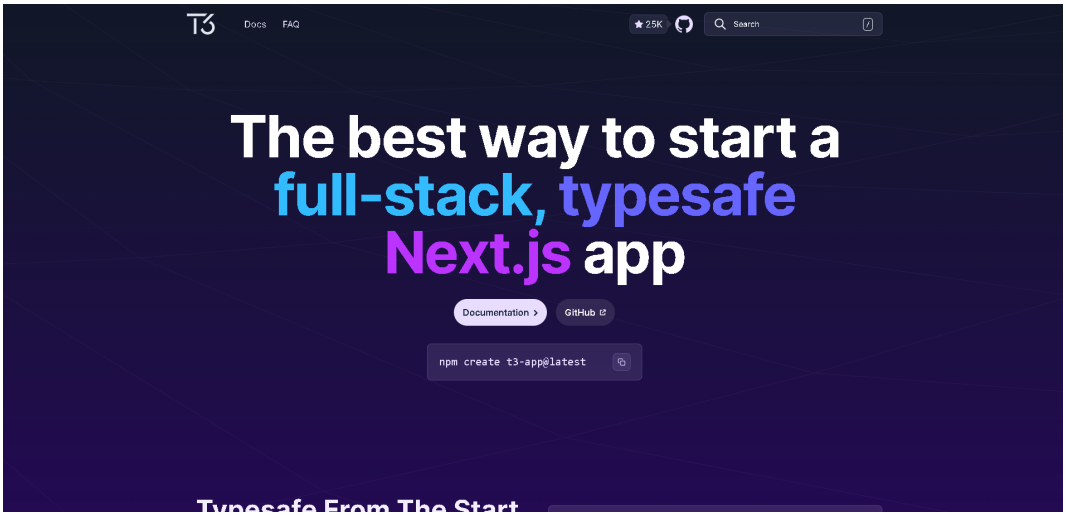
Fonte: stateofjs, 2024.

2.1.1 Plataformas de Geração Automática de *Templates*

A geração automática de templates é uma abordagem que busca otimizar o processo de desenvolvimento, especialmente em relação à criação de sistemas CRUD. O conceito de geradores de código tem sido amplamente explorado, com ferramentas que proporcionam a automação de processos repetitivos, permitindo que as equipes foquem em tarefas mais complexas e personalizadas.

Um exemplo relevante é o *T3 Stack*[1], uma plataforma que combina tecnologias modernas como *TypeScript*, *Tailwind CSS* e o *framework Next.js* para criar aplicações *full-stack* com eficiência e padronização. O *T3 Stack* se alinha à proposta deste trabalho por ter uma ideia semelhante e, por isso, ser a principal referência, integrando componentes de diferentes bibliotecas e *frameworks* de forma ágil. A proposta dessa plataforma é oferecer um conjunto de ferramentas configuráveis para criar aplicações escaláveis e com alta performance desde o início, sem a necessidade de configurar manualmente cada parte do projeto (T3 STACK, 2024).

Figura 3 – Plataforma T3 Stack.



Fonte: T3 Stack, 2024.

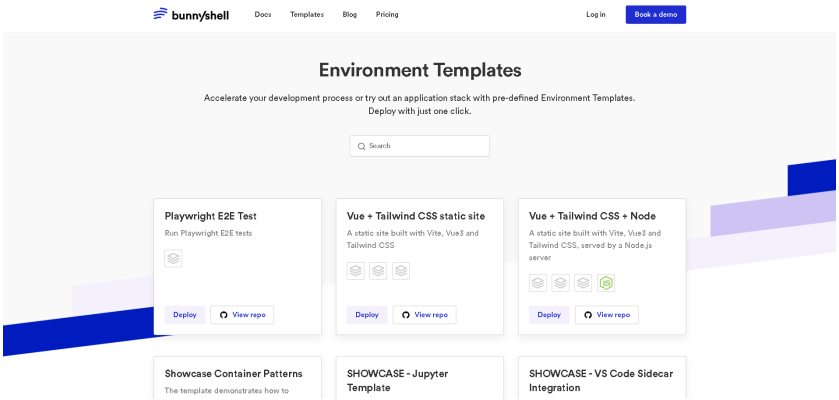
O conceito por trás do *T3 Stack* reflete a tendência de plataformas que buscam acelerar o processo de desenvolvimento ao oferecer estruturas base que podem ser adaptadas às necessidades específicas de cada projeto. Ferramentas como essa possibilitam uma maior colaboração entre equipes e minimiza erros (T3 STACK, 2024).

Outro exemplo atual é a plataforma *Bunny Shell*[2], que visa proporcionar automação no gerenciamento de infraestrutura e ambientes de desenvolvimento. O *Bunny Shell* permite

[1] Documentação Oficial em: <https://create.t3.gg/en/introduction>
[2] Documentação Oficial em: <https://documentation.bunnyshell.com/docs/introduction-to-bunnyshell>

a criação de ambientes de desenvolvimento e produção automaticamente, além de integrar sistemas de Integração Contínua/Entrega Contínua (CI/CD), ajudando as equipes a focarem no desenvolvimento de código sem se preocupar com a configuração manual de ambientes (BUNNY SHELL, 2024).

Figura 4 – Plataforma Bunny Shell.

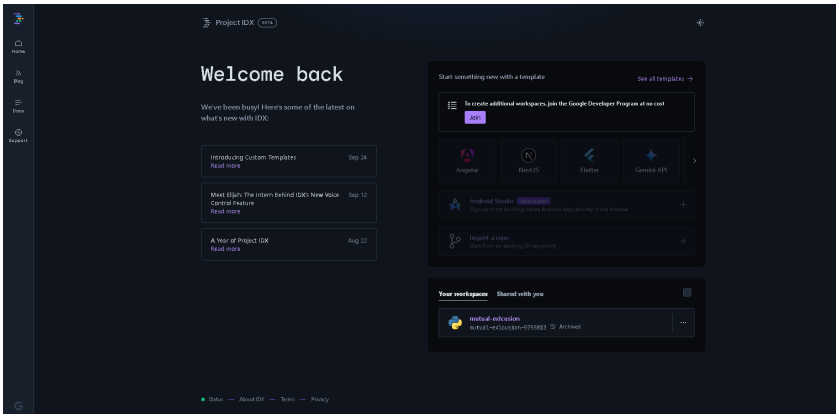


Fonte: Bunny Shell, 2024.

O exemplo mais recente é o *Google IDX*[3], uma plataforma voltada para o desenvolvimento de aplicações baseadas na nuvem, focando em experiências de desenvolvedores com automação e integração de ferramentas. A *Google IDX* permite que desenvolvedores construam aplicações web diretamente no navegador com o suporte de ferramentas poderosas, incluindo um ambiente de desenvolvimento colaborativo em tempo real e integração profunda com as APIs da Google. Além disso, a *Google IDX* utiliza *machine learning* para auxiliar os desenvolvedores a escreverem código mais rapidamente, sugerindo implementações otimizadas e corrigindo erros comuns automaticamente (GOOGLE IDX, 2024). Esse tipo de tecnologia reflete o futuro da geração de *templates* e desenvolvimento automatizado, com um foco crescente em inteligência artificial para facilitar o trabalho dos desenvolvedores e aumentar a produtividade das equipes.

[3] Documentação Oficial em: <https://idx.google.com/>

Figura 5 – Plataforma IDX.



Fonte: Google IDX, 2024.

2.2 Importância da Padronização e Boas Práticas

A padronização de código e a adoção de boas práticas no desenvolvimento de software são essenciais para a eficiência e manutenção de projetos a longo prazo. Segundo Hunt e Thomas, em "*The Pragmatic Programmer*", a consistência no código facilita a colaboração entre programadores, melhora a legibilidade e reduz a ocorrência de erros. O uso de *frameworks front-end* e bibliotecas de componentes pré-definidos segue essa linha de raciocínio, promovendo uma base comum que todos os membros da equipe podem seguir.

McConnell reforça essa ideia ao afirmar que a falta de padrões no início de um projeto pode resultar em dificuldades na escalabilidade e na manutenção do software. Ao automatizar a criação de templates, como ocorre no *T3 Stack*, as equipes podem garantir que o código gerado siga os mesmos padrões de qualidade em todos os componentes do sistema.

2.3 Considerações

Com as referências e objetivos expostos, este trabalho visa contribuir para a linha de pesquisa voltada ao desenvolvimento de ferramentas que otimizam o processo de criação de aplicações *web*, especialmente no contexto de sistemas CRUD.

Inspirado pela proposta de plataformas como o *T3 Stack*, o projeto busca reduzir a curva de aprendizado inicial, permitindo que os desenvolvedores foquem em aspectos mais complexos e personalizados do projeto, economizando tempo e recursos.

Por fim, este trabalho contribui com a comunidade de desenvolvedores ao proporcionar um ambiente de desenvolvimento que combina flexibilidade, automação e modernidade,

oferecendo uma plataforma capaz de se adaptar às rápidas mudanças do setor de tecnologia, mantendo-se relevante e útil para projetos futuros.

3 Desenvolvimento e Requisitos

Este projeto é parte integrante de um esforço mais amplo que incorpora diversas tecnologias, como *Angular*, *React*, *Flutter*, *Django*, *Node.js*, *Vue.js*, entre outras. Entretanto, neste trabalho, é apresentado o desenvolvimento da aplicação *front-end* utilizando o *framework Vue.js* consumindo a *Application Programming Interface* (API) do projeto *back-end*[1] para manipulação dos dados.

3.1 Conceitos Básicos

Antes de aprofundar ao desenvolvimento do trabalho, esta seção será dedicada a expor as ferramentas, conceitos básicos e modelagens que foram empregados desde as fases de planejamento e desenvolvimento deste.

3.1.1 Editor de Texto

O *Visual Studio Code* (VSCode)[2] é um editor de código-fonte amplamente utilizado no desenvolvimento de software, conhecido por sua leveza, flexibilidade e vasto ecossistema de extensões. Desenvolvido pela *Microsoft*, o VSCode oferece suporte a diversas linguagens de programação, incluindo *JavaScript*, *TypeScript*, *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS), entre várias outras, tornando-o uma excelente escolha para o desenvolvimento de aplicações *web* com *frameworks* como, no caso deste trabalho, o *Vue.js*. Uma de suas principais vantagens é a integração com sistemas de controle de versão, como o *Git*, facilitando o gerenciamento de código em equipe. Além disso, o editor conta com funcionalidades avançadas, como depuração, *IntelliSense* (autocompletar inteligente), e integração nativa com terminais, o que torna o fluxo de trabalho mais eficiente e produtivo para desenvolvedores. A flexibilidade do VSCode também se destaca pelo vasto repositório de extensões, permitindo personalizações e integrações específicas.

3.1.2 Boas Práticas de Desenvolvimento: ESLint

Para garantir a consistência e qualidade do código, o *ESLint*[3] foi utilizado no projeto. O *ESLint* é uma ferramenta estática de análise de código para identificar e corrigir padrões problemáticos encontrados no *JavaScript*. Ele foi configurado com as regras específicas para o *Vue.js* e integrado ao fluxo de desenvolvimento através do VSCode.

[1] Link para o repositório do projeto *back-end* utilizado neste trabalho <https://github.com/Chameleon-Stack/template-nodejs-express-mongo-prisma>
[2] Site Oficial VSCode: <https://code.visualstudio.com/>
[3] Documentação oficial do *ESLint*: <https://eslint.org/>

Dessa forma, sempre que uma nova parte do código era escrita, o *ESLint* apontava erros de estilo, garantindo um código mais limpo, padronizado e de fácil manutenção.

3.1.3 Git/Github

O *Git*[4] é um sistema de controle de versão distribuído amplamente utilizado no desenvolvimento de software para rastrear mudanças no código-fonte de forma eficiente e colaborativa. Ele permite que desenvolvedores trabalhem em conjunto em projetos, mantendo um histórico detalhado de todas as alterações, possibilitando reverter mudanças ou corrigir erros facilmente. O *GitHub*, por sua vez, é uma plataforma baseada em nuvem que hospeda repositórios *Git*, facilitando o compartilhamento e a colaboração em projetos de software. Ele oferece funcionalidades adicionais, como *pull requests*, revisão de código, integração contínua, o que torna o processo de desenvolvimento mais organizado e transparente. A integração do *Git* com plataformas como o *GitHub* simplifica a gestão de projetos, garantindo que equipes possam colaborar em tempo real e manter a integridade do código de maneira centralizada e acessível.

3.1.4 Back-end

No desenvolvimento de software, o *back-end* refere-se à parte do sistema que lida com a lógica de negócios, armazenamento de dados e comunicação com o *front-end*. Ele é responsável por processar requisições, manipular dados em bancos de dados e garantir que as informações sejam entregues ao usuário de maneira segura e eficiente. O *back-end* é geralmente desenvolvido em linguagens como *Node.js*[5], *Python*, *Java*, entre outras, e atua como a camada de processamento e armazenamento que sustenta as operações de uma aplicação.

3.1.5 API

As APIs são interfaces que permitem a comunicação entre o *back-end* e o *front-end*, ou entre diferentes serviços e sistemas. Elas fornecem um conjunto de regras e definições para que o software possa interagir de forma padronizada, garantindo que dados e funcionalidades sejam acessados e utilizados de maneira controlada. No contexto de desenvolvimento *web*, APIs RESTful (baseadas no protocolo *Hypertext Transfer Protocol* (HTTP)) são comumente utilizadas para permitir que o *front-end* consuma dados e funcionalidades do *back-end*, facilitando a integração e escalabilidade de aplicações.

[4] Plataforma Oficial do Github: <https://github.com/>
[5] Documentação Oficial: <https://nodejs.org/en>

3.2 Arquitetura do Projeto

A aplicação foi desenvolvida seguindo a arquitetura de separação entre o *front-end* e o *back-end,* onde o *front-end*, desenvolvido em *Vue.js*, consome a API. O sistema utiliza o padrão de componentes reativos no *front-end*, enquanto o *back-end* foi construído para fornecer dados estruturados por meio da API. A integração entre as camadas ocorre por meio de requisições HTTP, utilizando a biblioteca *Axios*[6] para facilitar o consumo da API.

Antes de seguir com os detalhes da implementação *front-end*, é crucial compreender a estrutura relacional que guia o funcionamento dos dados no *back-end*. A API consumida pelo *Vue.js* depende diretamente do modelo de banco de dados para a manipulação eficiente dos dados, o que inclui a criação, edição e exclusão de tarefas e categorias associadas ao usuário. A Figura 6 ilustra as relações entre essas entidades e como elas se comunicam dentro do sistema.

3.2.1 Modelo Relacional

O foco deste trabalho não é o *back-end*, entretanto, vale ressaltar o modelo relacional de como é o funcionamento das relações do *back-end*/banco de dados para entender a maneira que o *front-end* responde.

No banco de dados, temos uma estrutura que inclui informações sobre usuários, tarefas, categorias de tarefas e suas relações. As relações estão ilustradas na Figura 6.

3.2.1.1 Entidades

- **User**: Representa um usuário do sistema. Possui atributos como *id*, *name*, *password*, *email*, etc.
- ***cards***: Representa um *card* criado por um usuário. Contém informações como *status*, *title*, *description*, etc.
- **Category**: Representa uma categoria que pode ser associada a um *card*. Possui atributos como *name* e *color*.
- **CardCategory**: É uma entidade associativa que representa o relacionamento muitos-para-muitos entre *cards* e categorias. Isso significa que um *card* pode pertencer a várias categorias e uma categoria pode ter vários "*cards*".

3.2.1.2 Relacionamentos

- **User** 1:N ***cards***: Um usuário pode criar vários *cards*. O atributo *user_id* na entidade ***cards*** representa a chave estrangeira que estabelece esse relacionamento.

[6] Documentação oficial do *Axios*: <https://axios-http.com/docs/intro>

- ***cards*** N:N **Category**: Um *card* pode pertencer a várias categorias e uma categoria pode ter vários *cards*. Esse relacionamento é implementado através da entidade associativa **CardCategory**, que possui as chaves estrangeiras *card_id* e *category_id* para conectar as entidades ***cards*** e **Category**.

3.2.1.3 Funcionamento dos Relacionamentos

As entidades se relacionam para permitir a organização e categorização dos *cards* dentro do sistema.

- O relacionamento entre **User** e ***cards*** permite que o sistema identifique quem criou cada *card*.
- O relacionamento entre ***cards*** e **Category** (através de **CardCategory**) permite que os *cards* sejam classificados em diferentes categorias, facilitando a organização.

3.2.1.4 Exemplo de Uso

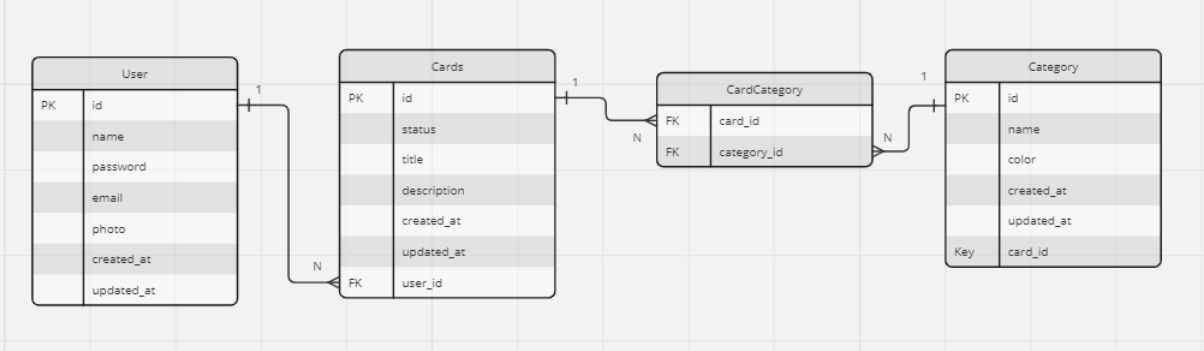
Imagine um sistema de gerenciamento de tarefas.

- **Usuários**: Pessoas que usam o sistema para organizar suas tarefas.
- ***cards***: Tarefas a serem realizadas.
- **Categorias**: Categorias para organizar as tarefas, como “Trabalho”, “Pessoal”, “Urgente”, etc.

Um usuário pode criar várias tarefas (*cards*). Cada tarefa pode ser associada a uma ou mais categorias. Por exemplo, uma tarefa "Fazer relatório"pode pertencer às categorias "Trabalho"e "Urgente".

Esse modelo de dados permite que o sistema armazene e organize as informações de forma eficiente, permitindo que os usuários encontrem e gerenciem suas tarefas facilmente.

Figura 6 – Modelo Relacional do funcionamento do *back-end*.



Fonte: figura do autor.

3.3 Especificações do Sistema

Nesta seção são apresentados os requisitos funcionais e não funcionais do sistema *front-end* do projeto. A primeira etapa no ciclo de desenvolvimento do software proposto neste projeto consistiu em como o planejamento de aplicações pode afetar futuramente, causando retrabalhos e atrasos de entregas. Sendo assim, nesta etapa, o levantamento de requisitos consiste em facilitar a etapa de seleção de *stacks* destas aplicações.

3.3.1 Requisitos Funcionais

Em resumo, os requisitos funcionais que o sistema *front-end* apresentam quando está devidamente instalado, isto é, quando o usuário tiver sua aplicação executando, são:

1. Logar no sistema ou criar um novo usuário;
2. Gerenciar os *cards* (CRUD);
    - *Create*: criar novos *cards* no sistema;
    - *Read*: listar os *cards* ligados ao usuário;
    - *Update*: editar os *cards*;
    - *Delete*: deletar *cards* existentes.
3. Gerenciar categorias;
4. Ajustar a prioridade/categoria dos itens em tempo real.

Uma descrição mais detalhada sobre os requisitos funcionais do sistema está disposta no Apêndice A

3.3.2 Requisitos Não Funcionais

Requisitos não funcionais são critérios que especificam características de um sistema que não estão diretamente relacionadas com as suas funcionalidades. Eles são importantes para o desempenho, segurança e disponibilidade do sistema, e influenciam a experiência do usuário. Para o sistema, estes, estão dispostos na Tabela 1:

Tabela 1 – Requisitos não funcionais do sistema.

Identificador	**Requisito**	**Descrição**
NF01	Usabilidade	O sistema deve ser de fácil entendimento e manipulação; deve, também, permitir ao usuário realizar tarefas de forma fácil, com telas intuitivas. Além disso, é necessário que a aplicação emita mensagens de confirmação para operações de alta importância, como, deletar uma tarefa
NF02	Compatibilidade	O sistema deve ser multiplataforma e executar normalmente com sucesso em diferentes sistemas operacionais.
NF03	Segurança	O sistema deve fornecer mecanismos de autenticação que garantam acesso personalizado para diferentes usuários.
NF04	Disponibilidade	O sistema deve estar disponível online e para que os usuários possam acessá-lo de diferentes locais com acesso a Internet.
NF05	Licença	O sistema deve estar disponível de acordo com a MIT *License*.

3.4 Desenvolvimento

A escolha do *framework Vue.js* para o desenvolvimento deste projeto, justificada no Capítulo 1, deve-se à sua natureza progressiva e foco na construção de interfaces de usuário (*User Interface* (UI)) e aplicações de página única (*Single-Page Applications* (SPA)). Essa decisão se baseia nas seguintes vantagens[7]:

1. **Simplicidade e Flexibilidade:** *Vue.js* é conhecido por sua simplicidade e facilidade de aprendizado, e, como visto em Figura 1, é um *framework* que está numa crescente entre os programadores. Ele adota uma abordagem incremental, o que significa que, ao começar, adiciona pequenas partes do projeto e expande conforme necesário. Isso torna o *Vue.js* uma escolha atraente para qualquer nível de desenvolvedor, desde os iniciantes aos mais experientes, e, também, para projetos de diferentes tamanhos.

2. **Performance:** *Vue.js* é performático. Ele utiliza uma arquitetura reativa que torna o gerenciamento de estado eficiente e facilita a atualização da UI conforme os dados mudam. Além disso, o *Vue.js* oferece recursos de otimização, como a renderização virtual, que reduz o tempo necessário para atualizar o *Document Object Model* (DOM), resultando em uma experiência do usuário mais rápida e fluida.

[7] De acordo com a documentação oficial do Vue.js: <https://vuejs.org/>

3. **Componentização:** *Vue.js* promove o desenvolvimento baseado em componentes, o que significa que pode dividir a interface de usuário em pequenos componentes reutilizáveis. Isso facilita a manutenção do código, promove a reutilização e permite uma melhor organização do projeto.

3.4.1 Funcionamento do Modelo

Seguindo o modelo da Figura 6, nesta seção é abordado de forma mais técnica como foi feita a integração do *back-end* com o *front-end*.

1. **Integração com API**

   - ***Vuex Store***
     - *Vuex* é um padrão de gerenciamento de estado *Vue.js*. Ele serve como um armazenamento centralizado para todos os componentes em um aplicativo, com regras, garantindo que o estado só pode ser mudado de forma previsível. De acordo com a documentação oficial, o *Vuex* serve como um padrão de gerenciamento de estado centralizado para aplicativos *Vue.js*, facilitando a manutenção e previsibilidade do estado da aplicação (VUEX, 2015).
     - A API é responsável por servir os dados de cada tabela do modelo relacional e suas relações. Por exemplo, ao criar uma tarefa/*card*, o *Vue.js* envia uma requisição POST contendo o ID do usuário e as categorias associadas ao cartão.
   - **Gerenciamento de Estado**
     - O estado da aplicação é armazenado em *Vuex*, mantendo as tarefas e categorias em *store* (*store* pode ter uma comparação de funcionamento como um gaveteiro, onde, quando eu quiser adicionar/modificar/remover algo, basta abrir e decidir o que for fazer). As alterações feitas nos dados seriam sincronizadas com a API *back-end* através de ações do *Vuex*.
     - Por exemplo, ao adicionar uma nova categoria a uma tarefa, a *store* seria atualizada com a nova associação via chamada à API.

2. **Rotas e Navegação**

   - Foi utilizado o *Vue Router* para definir diferentes rotas para os componentes, como tela de *login*, *homepage* e cadastro de usuário, facilitando a navegação entre as diferentes funcionalidades da aplicação.

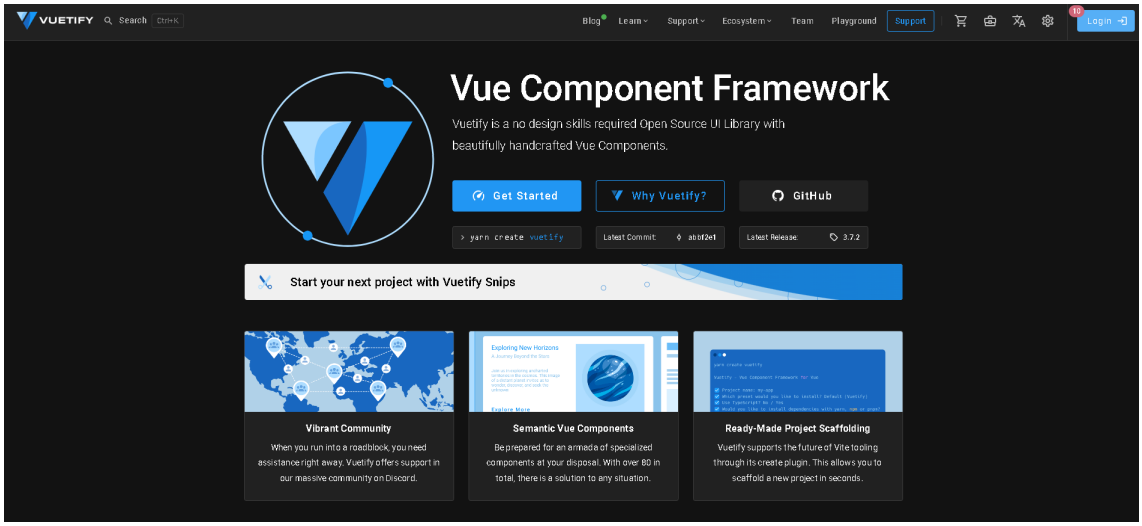
3.5 Variações

Em conjunto, durante o desenvolvimento da aplicação, foram integrados três bibliotecas de componentes que oferecessem suporte para a aplicação. Essas bibliotecas foram selecionadas para compor as variantes da aplicação, cada uma trazendo recursos e estilizações específicos para atender às necessidades da interface do usuário:

3.5.1 Vuetify

O *Vuetify*[8] é uma das bibliotecas mais populares para o desenvolvimento de interfaces utilizando o *Vue.js*. Ele oferece uma gama de componentes prontos, baseados no *Material Design*, o que facilitou a criação de interfaces responsivas e modernas. A biblioteca trouxe uniformidade ao design e acelerou o processo de desenvolvimento, especialmente para componentes como botões, modais e formulários, que seguiram um padrão estético coerente. O *link* do repositório desta variante se encontra em Variante Vuetify

Figura 7 – Tela inicial da plataforma *Vuetify*.



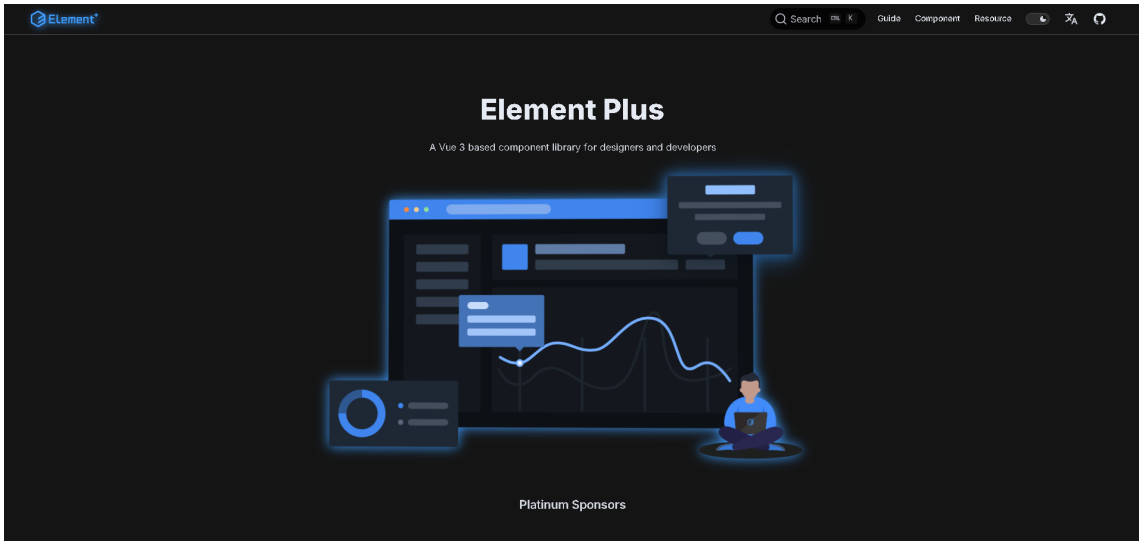
Fonte: *Vuetify, 2024.*

3.5.2 Element+

Element+[9] é outra biblioteca que foi utilizada para criação de componentes, focando na simplicidade e facilidade de uso. Ela trouxe uma abordagem mais minimalista, com componentes leves e de fácil customização. Essa biblioteca foi particularmente mais fácil de manipular devido a documentação mais clara. O *link* do repositório desta variante se encontra em Variante Element+

[8] Documentação oficial do *Vuetify*: <https://vuetifyjs.com/en/>
[9] Documentação oficial do *Element+*: <https://element-plus.org/en-US/>

Figura 8 – Tela inicial *Element+*.



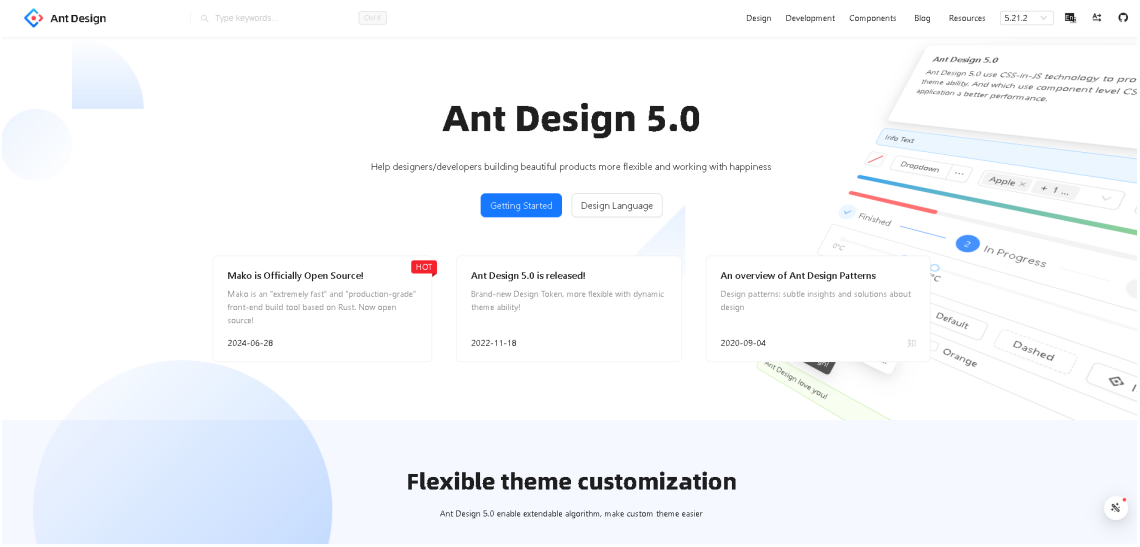
Fonte: *Element+*

3.5.3 Ant-Design

Ant-Design[10] é uma biblioteca de componentes que fornece uma ampla gama de elementos para a construção de interfaces voltadas para aplicações empresariais. Com uma documentação detalhada e diversos exemplos práticos, ele oferece componentes robustos como *cards*, tabelas, menus e *modals*, facilitando a implementação de funcionalidades mais complexas e assegurando uma boa experiência de usuário. Na aplicação, o *Ant-Design* foi utilizado principalmente para componentes de tabelas e modais, permitindo a criação de interfaces intuitivas e responsivas. O *link* do repositório desta variante se encontra em Variante Ant-Design

[10] Documentação oficial do *Ant-Design*: <https://ant.design/>

Figura 9 – Tela inicial *Ant-Design*.



Fonte: *Ant-Design, 2024.*

3.5.4 Comparações

A fim de comparação entre as bibliotecas, na Tabela 2 há algumas comparações qualitativas e quantitativas de cada uma delas, evidênciando algumas de suas diferenças.

Tabela 2 – Comparação entre bibliotecas de componentes Vue.js

Característica	**Ant-Design**	**Element+**	**Vuetify**
Design	Estilo empresarial, foco em eficiência e densidade de informação	Minimalista, moderno	Material Design
Filosofia	Componentes ricos em funcionalidades	Componentes básicos e flexíveis	Componentes com visual pré-definido
Maturidade	Maduro, amplamente utilizado em aplicações empresariais	Relativamente novo, mas em rápido crescimento	Maduro, comunidade ativa
Comunidade	Comunidade ativa, principalmente na China	Comunidade crescente	Comunidade grande e ativa
Customização	Altamente customizável	Flexível, permite personalização	Customização moderada, mantém estilo do Material Design
Downloads npm (semanais)	≈ 101.4k	≈ 255.3k	≈ 596.6k
Estrelas no Github	20.2k	24.4k	39.8k
Documentação	Bem documentada, com exemplos e tutoriais	Documentação completa e bem organizada	Documentação abrangente e bem organizada

Os dados do npm foram retirados dos sites <https://www.npmjs.com/> e <https://www.github.com/> e pesquisando suas respesctivas bibliotecas/pacotes.

3.6 Considerações

Neste capítulo, foram apresentados os principais aspectos relacionados ao desenvolvimento da aplicação, incluindo a arquitetura escolhida, o processo de integração com a API, a utilização de bibliotecas, bibliotecas de componentes e *frameworks*, bem como as práticas de qualidade de código adotadas. É válido mencionar, também, que o padrão arquitetural do projeto seguiu o modelo *Model, View, Controller* (MVC), que é, basicamente, um padrão de arquitetura de software que divide a aplicação em três camadas:

- manipulação dos dados (*model*);
- interação do usuário (*view*);
- camada de controle (*controller*).

Além disso, o protótipo da *Landing Page* está disponível, onde, é possível ver a prévia de como a página inicial do projeto está, como mostrado na Figura 10. Os repositórios do projeto podem ser encontrados em Chameleon Stack.

Figura 10 – Protótipo *Landing Page*.

Fonte: figura do autor.

No próximo capítulo, serão apresentados os resultados obtidos com a implementação da aplicação, detalhando as interfaces desenvolvidas e demonstrando o funcionamento

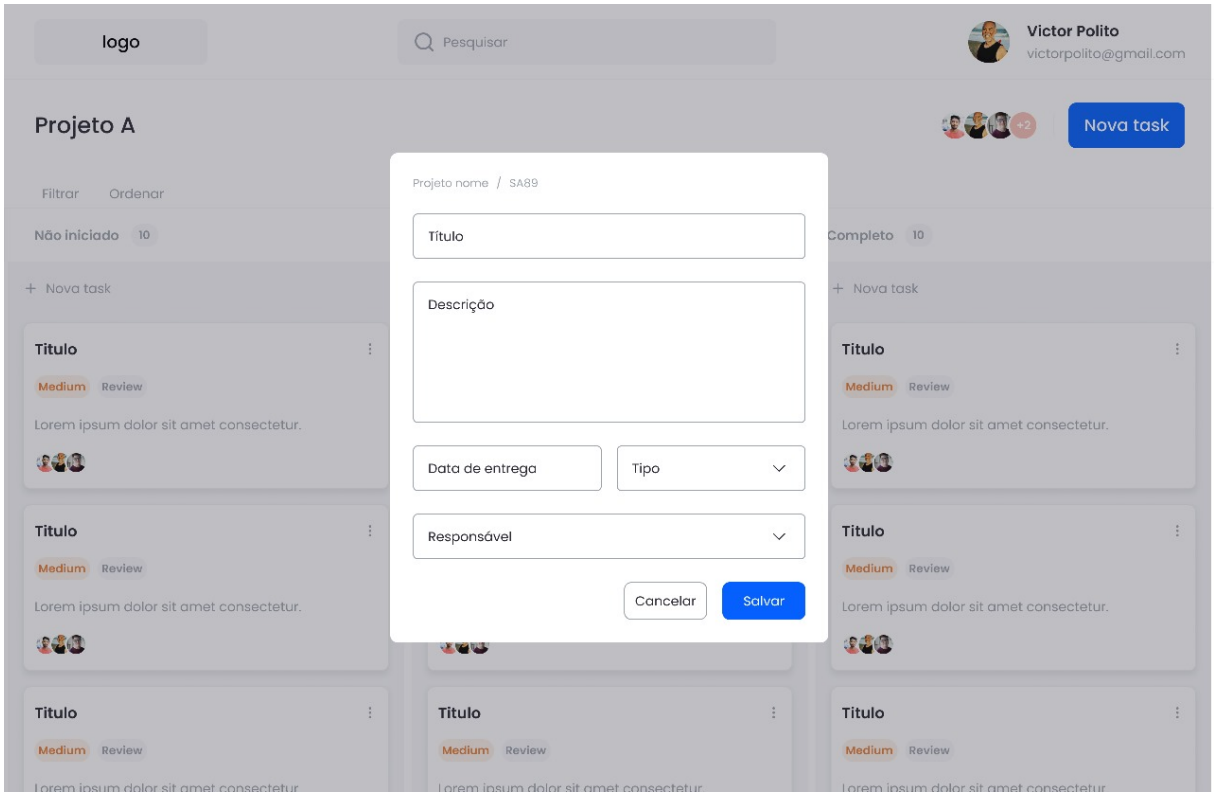
do software. As telas e funcionalidades serão analisadas de acordo com os requisitos estabelecidos no início do projeto, evidenciando como as decisões de desenvolvimento discutidas neste capítulo se refletem na versão atual do sistema.

4 Resultados

Neste capítulo, as telas e o funcionamento geral do software desenvolvido serão exibidas, evidenciando como suas funcionalidades foram implementadas e como cada uma delas contribui para o objetivo do projeto. A interface segue uma estrutura intuitiva e responsiva, projetada para facilitar a experiência do usuário no gerenciamento de tarefas, alinhada com as necessidades de organização pessoal e profissional.

Inicialmente, foi estabelecido um modelo base a ser seguido por todas as variantes, a fim de fazer a comparação de qual variante necessitava mais de ajustes para chegar o mais próximo possível deste. O modelo é apresentado na Figura 11.

Figura 11 – Modelo Inicial do Projeto.



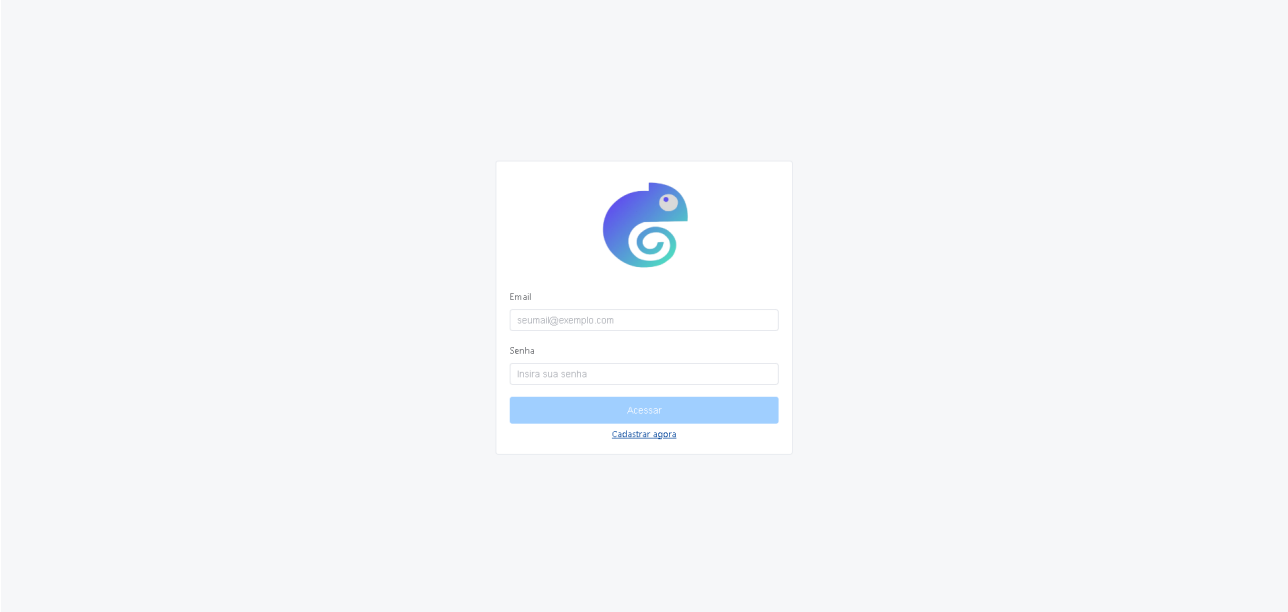
Fonte: figura do autor.

4.1 Tela de Login

A tela inicial do sistema é a de *login*, onde o usuário deve se identificar fornecendo suas credenciais (*e-mail* e senha). Caso ainda não possua uma conta, há a opção de criação

de um novo usuário. A interface dessa tela é minimalista, com campos claros e organizados, facilitando a experiência de autenticação do usuário.

Figura 12 – Tela de *login*.

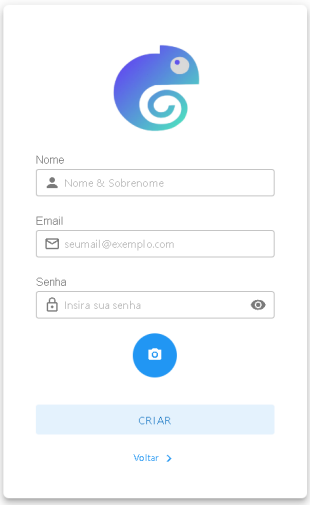


Fonte: figura do autor.

4.1.1 Criação de Usuário

Ao clicar na opção de criação de conta, o usuário é direcionado para um formulário simples, que solicita nome, *e-mail*, senha e uma foto de perfil (opcional). É um design pensado para ser amigável e prático, promovendo a criação rápida de contas.

Figura 13 – Tela de criação de conta para novos usuários.

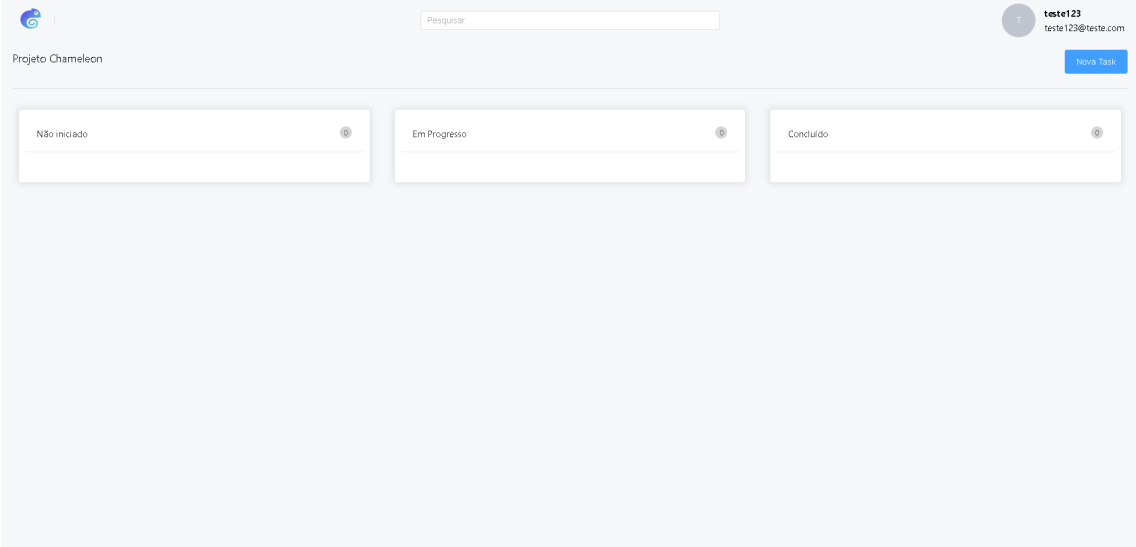


Fonte: figura do autor.

4.2 Tela Home Page

Após o *login*, o usuário é redirecionado para a página principal, apresentado na Figura 18, onde estão organizadas suas tarefas em três colunas: *backlog*, em progresso e concluído. Nesta tela, o usuário também tem acesso às suas informações de perfil (nome, foto (ou as inicias do nome do usuário caso não tenha foto e e-mail) que aparecem no topo da página. Ao clicar na foto, o usuário tem a opção de *deslogar* do sistema.

Figura 14 – Tela inicial.



Fonte: figura do autor.

4.2.1 Cards

As tarefas são apresentadas em formato de *cards*, que podem ser alternados entre as colunas para alterar seu status de *backlog* para em progresso ou concluído. Esta interface visual facilita a organização do fluxo de trabalho.

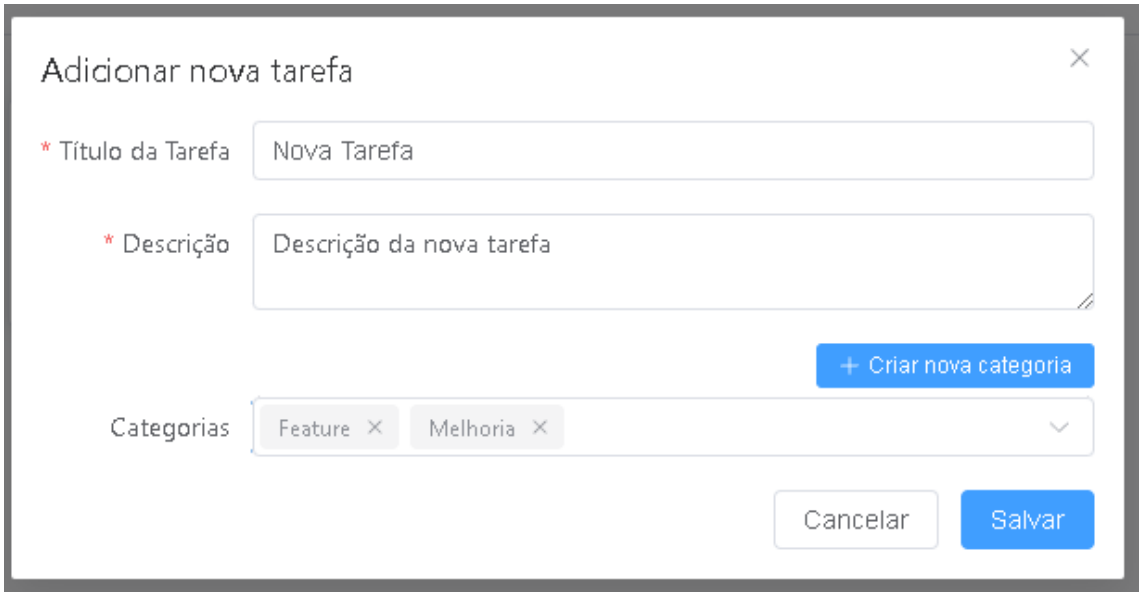
4.3 Modais

Para facilitar a interação com as funcionalidades de criação, edição e exclusão de tarefas, o sistema utiliza modais para realizar essas ações sem sair da tela principal. O uso de modais foi implementado para não interromper o fluxo de uso do software, mantendo a experiência do usuário fluida e eficiente.

4.3.1 Criação de Nova Tarefa

Ao clicar no botão para adicionar uma nova tarefa, exibido na Figura 18, é exibido um modal que permite ao usuário inserir o título, a descrição e as categorias da tarefa. O design do modal foi pensado para ser intuitivo e eficiente, com campos claros e botões bem posicionados.

Figura 15 – Modal para criação de uma nova tarefa.

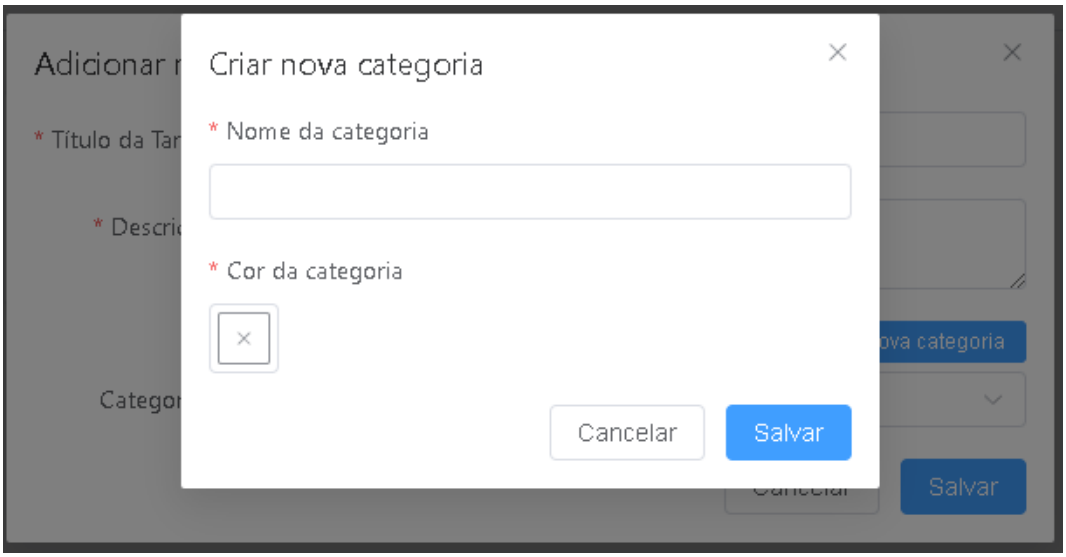


Fonte: figura do autor.

4.3.2 Criação de Nova Categoria

Esta *feature* está presente no modal de Criação de Nova Tarefa e Edição de Cards. É um botão que abre um modal separado permitindo a criação de novas categorias, com um campo para nome e um seletor de cor para a *label* da categoria. A interface permite uma personalização rápida, dando ao usuário flexibilidade na organização.

Figura 16 – Modal para criação de uma nova categoria.

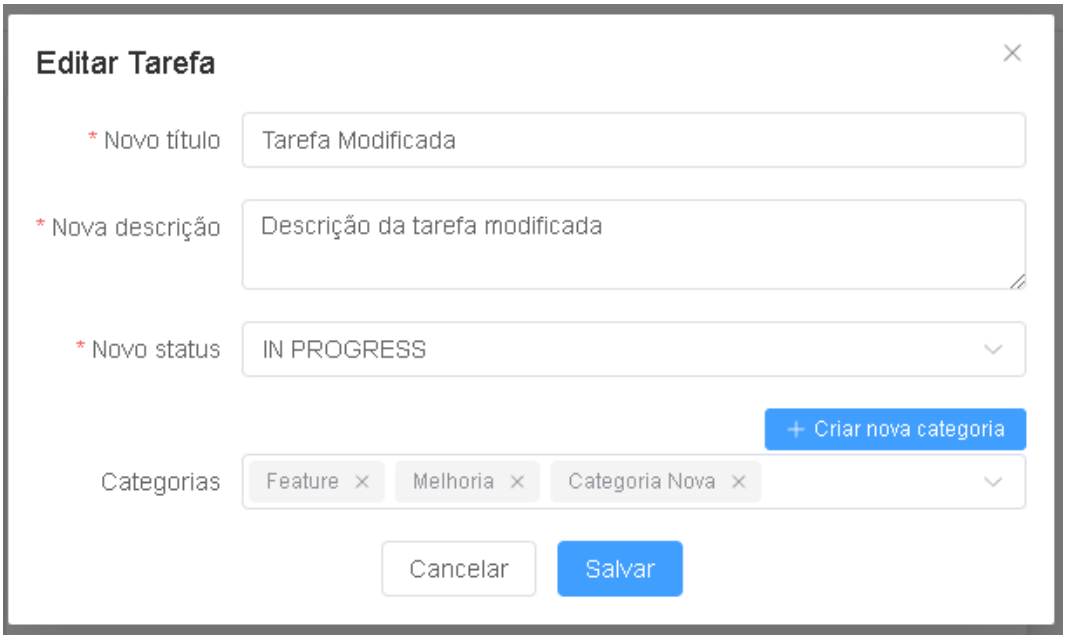


Fonte: figura do autor.

4.3.3 Edição de Cards

Cada card possui um botão para edição, que abre um modal onde o usuário pode alterar o título, descrição, status e categorias. A possibilidade de adicionar categorias já existentes ou criar novas a partir deste modal reforça a praticidade do sistema.

Figura 17 – Modal para edição de uma tarefa.

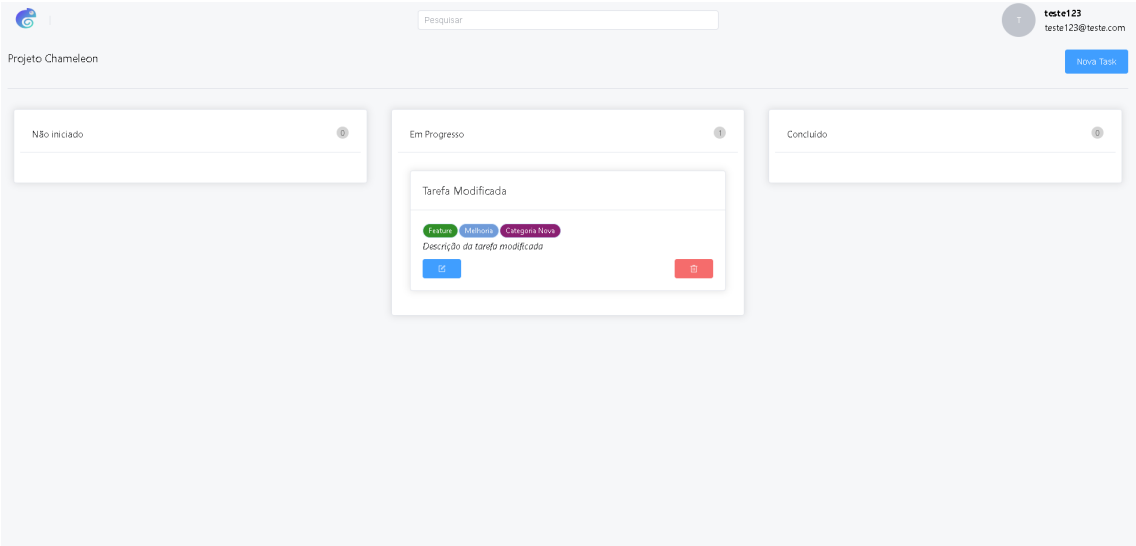


Fonte: figura do autor.

4.4 Tela Home Page - Com cards

Após a criação e edição de um novo *card*, é possível visualizá-lo, em tempo real, na *home page* com seu título e descrição atualizados, além das categorias que foram criadas ou removidas.

Figura 18 – Tela inicial atualizada.



Fonte: figura do autor.

4.4.1 Exclusão de Cards

O usuário pode excluir qualquer card ao clicar em um ícone específico, que abre um modal de confirmação. Esta ação permite que o usuário revise sua decisão antes de excluir uma tarefa permanentemente.

Figura 19 – Modal para excluir de uma tarefa.

Fonte: figura do autor.

4.5 Considerações Finais

Com a interface finalizada, fica visível que o software cumpre o objetivo de ser uma ferramenta para a organização de tarefas *Kanban*. O uso de modais para operações-chave como criação, edição e exclusão, combinado com a organização visual dos *cards*, contribui para uma experiência intuitiva e fluida. A possibilidade de criar e categorizar as tarefas e

mover os *cards* entre diferentes estágios proporciona flexibilidade e controle sobre o fluxo de trabalho.

Como o objetivo central do trabalho era deixar com que as variantes se assemelhassem ao máximo com a Figura 11, não há um porquê exibi-lás, visto que as diferenças entre os componentes modificados são mínimas. Entretanto, vale ressaltar como foi a experiência de desenvolvimento com cada uma delas. Analisando uma *feature* fornecida pelo Github (subseção 3.1.3), é possível ver a quantidade de "diferentes códigos" que foram utilizados para chegar no resultado almejado.

Tabela 3 – Comparação entre uso de linguagens para personalização das bibliotecas de componentes Vue.js

Percentual de código de cada variante			
Linguagem/Biblioteca	**Ant-Design**	**Element+**	**Vuetify**
Vue.js	77,3%	74,8%	72,8%
CSS	17,7%	7%	7,4%
HTML	2,5%	1,4%	1,4%

Com base na Tabela 3 e nas bibliotecas de UI discutidas podemos destacar as principais diferenças e a experiência de desenvolvimento com cada uma delas.

- **Ant-Design**

  Experiência de Desenvolvimento: O *Ant-Design* é amplamente utilizado em projetos corporativos devido ao seu foco em interfaces limpas e padronizadas. Ele segue um *design system* rigoroso, o que facilita a criação de interfaces consistentes, porém pode ser menos flexível para personalizações mais avançadas. A integração com *Vue.js* foi bastante direta, com uma curva de aprendizado relativamente suave para desenvolvedores que já estão familiarizados com padrões de *design* mais rígidos.

  – Vantagens:

    1. Excelente documentação.
    2. Ótimo para projetos corporativos e com necessidade de uma interface altamente padronizada.
    3. Grande número de componentes prontos.

  – Desvantagens:

    1. Menor flexibilidade em termos de personalização.
    2. O código gerado pode ser mais extenso, como indicado pelos 77,3% de código *Vue.js* comparado às outras bibliotecas.

- Ajustes Necessários:

  O Ant-Design exigiu uma maior quantidade de código CSS para alcançar o mesmo resultado das outras bibliotecas, possivelmente devido ao seu *design* mais rígido, que pode requerer ajustes finos para adequar-se a necessidades específicas.

- Recomendação:

  Recomendado para projetos empresariais ou aplicações que precisam seguir um padrão de interface específico. É ideal quando a padronização e a consistência são mais importantes que a flexibilidade.

- **Element+**

  Experiência de Desenvolvimento: *Element+* oferece uma abordagem mais balanceada, permitindo uma boa quantidade de personalizações, mas ainda mantendo um nível razoável de padronização. O *framework* é popular na comunidade *Vue.js*, como visto na Tabela 2 e apresenta uma integração sólida com esse ecossistema.

  - Vantagens:
    1. Fácil de aprender e usar com *Vue.js*.
    2. Boa documentação e comunidade ativa.
    3. Flexível, permitindo personalizações sem sair muito do padrão.
  - Desvantagens:
    1. Pode não ser tão enxuto quanto bibliotecas mais minimalistas.
    2. Menor quantidade de componentes prontos se comparado ao *Ant-Design*.
  - Ajustes Necessários:

    Embora tenha uma quantidade considerável de código Vue.js (74,8%), os ajustes para alcançar o modelo final não foram tão exigentes quanto no *Ant-Design*, tornando *Element+* uma escolha equilibrada para quem deseja flexibilidade e consistência.

  - Recomendação:

    Recomendado para projetos onde a flexibilidade e a personalização são tão importantes quanto a padronização. É uma boa escolha para aplicações de médio porte que precisam de uma interface consistente, mas com margem para adaptações.

- **Vuetify**

  Experiência de Desenvolvimento: *Vuetify* é amplamente conhecido por ser a escolha principal quando se trabalha com o ecossistema *Vue.js*, graças à sua integração

nativa com a plataforma. Ele segue as diretrizes do *Material Design*, proporcionando uma interface moderna e responsiva. Particularmente, o meu preferido entre os três.

- Vantagens:
  1. Excelente integração com *Vue.js*.
  2. Interface moderna seguindo o *Material Design*.
  3. Grande número de componentes e extensões disponíveis.
  4. Redução no código necessário para implementar componentes padrões.
- Desvantagens:
  1. Segue de forma rigorosa o *Material Design*, o que pode não ser desejável em todos os projetos.
  2. Menor flexibilidade de *design* fora das diretrizes do *Material Design* - até para uso de ícones, algumas vezes.
- Ajustes Necessários:

  A menor quantidade de código *Vue.js* (72,8%) reflete a eficiência do *Vuetify* em lidar com a padronização, o que significa menos ajustes e configurações manuais para alcançar o mesmo resultado final, mesmo que tenha 0,4% a mais com relação ao *Element+*.
- Recomendação:

  Ideal para projetos que valorizam uma interface moderna e aderente ao *Material Design*, como aplicativos focados em experiências de usuário inovadoras. *Vuetify* é recomendado para aplicações com alta demanda visual e que necessitam de menos personalizações no *layout*, pois o *framework* já oferece uma série de componentes prontos. É recomendado também para desenvolvedores *web* iniciantes, por ter uma documentação e componentes mais amigáveis.

4.5.1 Discussão e Recomendação Geral

Analisando as três bibliotecas, cada uma se destaca em contextos específicos:

- **Ant-Design** é recomendado para grandes aplicações empresariais, onde a padronização é crítica, mesmo que isso signifique um aumento na quantidade de código e ajustes.
- **Element+** serve bem para projetos que exigem um equilíbrio entre flexibilidade e padronização, sendo adequado para uma ampla gama de aplicações *web*, de médio a grande porte.
- **Vuetify** é ideal para projetos que seguem o *Material Design* e requerem uma interface moderna e responsiva com o mínimo de personalização necessária.

A quantidade de ajustes necessários em cada biblioteca é diretamente refletida na quantidade de código gerado. A escolha final depende das necessidades específicas do projeto, seja ele mais corporativo, flexível ou focado em *design* visual. Isso acabou refletindo na forma com que a curva de aprendizado se acentuava cada vez mais quando as bibliotecas ficavam mais nichadas, isto é, por exemplo, o *Ant-Design* que tem esse estilo mais corporativo.

5 Conclusão

Na conclusão desta monografia, é possível refletir sobre os principais desafios enfrentados e as soluções adotadas ao longo do desenvolvimento de um CRUD utilizando três bibliotecas diferentes de componentes: *Vuetify*, *Element+* e *Ant-Design*. O uso dessas bibliotecas possibilitou a criação de três variantes da aplicação, cada uma com um design e usabilidade próprios, o que foi fundamental para compreender as peculiaridades e potencialidades de cada uma dessas ferramentas.

Um dos aspectos mais enriquecedores deste trabalho foi a oportunidade de entender como diferentes bibliotecas lidam com problemas comuns em interfaces de usuário, como responsividade, customização de estilos e acessibilidade. A implementação de modais, formulários dinâmicos e a manipulação de *cards* nas três variantes do CRUD mostrou que, apesar das semelhanças nas funcionalidades, cada biblioteca oferece abordagens e soluções próprias.

Outro ponto relevante foi o uso de boas práticas de desenvolvimento, como a utilização do ESLint, que contribuiu para a padronização do código e a manutenção da qualidade ao longo do projeto. O processo de desenvolvimento se manteve focado em garantir uma base sólida e facilmente extensível. Além disso, a conectividade entre o *front-end* e *back-end* também é um aspecto muito enriquecedor.

Por fim, a criação das três variantes do CRUD permitiu uma análise comparativa, tanto quantitativa quanto qualitativa entre as bibliotecas e, ao mesmo tempo, expandiu o entendimento sobre a construção de interfaces *web* modernas. Esse aprendizado será valioso para futuros projetos e reforça a importância da escolha adequada de ferramentas de acordo com as necessidades específicas de cada aplicação.

5.1 Trabalhos Futuros

Apesar dos resultados alcançados, há diversas possibilidades para a evolução desta ferramenta. Algumas sugestões de trabalhos futuros incluem:

- **Incluir novos templates**: A adição de templates utilizando outras tecnologias, como *TailwindCSS*, *shadcn/ui*, *SCSS*, entre outros, podendo expandir as possibilidades de personalização e oferecer mais opções para diferentes necessidades.
- **Hospedagem na Web**: Um passo importante seria hospedar a aplicação desenvolvida, tornando-a acessível para desenvolvedores que desejam utilizar a ferramenta como uma solução prática para criação de *templates* de sistemas CRUD.

- **Criação de templates para outros sistemas**: A plataforma pode ser expandida para gerar templates para outros tipos de sistemas, como *e-commerce*, sistemas de gestão de usuários, entre outros. Esses templates podem incluir *features* mais avançadas, como autenticação em dois fatores, gerenciamento de permissões e relatórios dinâmicos.
- **Integração com geradores de *back-end***: Outra possível melhoria seria o desenvolvimento de templates que integram o *front-end* com geradores automáticos de *back-end*, facilitando ainda mais a implementação completa de aplicações web.
- **Internacionalização (i18n)**: A adição de suporte para internacionalização (i18n) seria um avanço importante para tornar os templates mais adaptáveis a diferentes públicos e mercados, especialmente em projetos com alcance global.

Essas sugestões visam não apenas aprimorar a plataforma desenvolvida, mas também facilitar e agilizar o desenvolvimento de sistemas em contextos diversos, fortalecendo a base construída ao longo deste trabalho.

Referências

ANT-DESIGN. *Ant Design - The World's Second Most Popular React UI Framework*. 2024. Disponível em: <https://ant.design/>. Acesso em: 20 set. 2024. Citado na página 17.

BECK, K. *Manifesto for Agile Software Development*. 2001. Disponível em: <https://agilemanifesto.org/>. Acesso em: 25 ago. 2024. Citado na página 15.

BOEHM, B. W. A spiral model of software development and enhancement. In: *ACM SIGSOFT Software Engineering Notes*. [S.l.]: ACM, 1988. v. 11, n. 4, p. 14–24. Citado na página 15.

BROOKS, F. P. *The Mythical Man-Month: Essays on Software Engineering*. [S.l.]: Addison-Wesley, 1975. ISBN 978-0201835953. Citado na página 15.

BUNNY SHELL. *Bunny Shell Documentation*. 2024. Disponível em: <https://documentation.bunnyshell.com/docs/introduction-to-bunnyshell>. Acesso em: 09 out. 2024. Citado na página 20.

ELEMENT+. *Element Plus - A Vue 3 UI Framework*. 2024. Disponível em: <https://element-plus.org/>. Acesso em: 18 set. 2024. Citado na página 17.

FORSGREN, N. *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. [S.l.]: O'Reilly Media, Inc., 2019. Citado na página 18.

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.]: Addison-Wesley, 2004. Citado na página 18.

GOOGLE IDX. *Google IDX Documentation*. 2024. Disponível em: <https://idx.google.com/>. Acesso em: 09 out. 2024. Citado na página 20.

GROUP, T. S. *The CHAOS Report*. 1994. Disponível em: <https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf>. Acesso em: 25 ago. 2024. Citado na página 15.

HUNT, A.; THOMAS, D. *The Pragmatic Programmer: Your Journey to Mastery*. [S.l.]: Addison-Wesley, 1999. ISBN 978-0201616224. Citado na página 15.

KITCHENHAM, B. *Procedures for performing systematic reviews*. 2004. Keele, UK, Keele University. Citado na página 18.

MCCONNELL, S. *Rapid Development: Taming Wild Software Schedules*. [S.l.]: Microsoft Press, 1996. ISBN 978-1556159008. Citado na página 15.

STATEOFFRONTEND. 2023. Disponível em: <https://2023.stateofjs.com/en-US/libraries/front-end-frameworks/>. Acesso em: 15 set. 2024. Citado na página 18.

T3 STACK. *T3 Stack*. [S.l.], 2024. Disponível em: <https://create.t3.gg/en/introduction>. Acesso em: 15 set. 2024. Citado na página 19.

VUETIFY. *Vuetify - Vue.js Material Component Framework*. 2024. Disponível em: <https://vuetifyjs.com/>. Acesso em: 2 set. 2024. Citado na página 17.

YOU, E. *Vue.js - The Progressive JavaScript Framework*. 2014. Disponível em: <https://vuejs.org/>. Acesso em: 15 set. 2024. Citado na página 17.

Apêndices

APÊNDICE A – Requisitos Funcionais

A.1 Tela de Login

Tabela 4 – Requisitos Funcionais da Tela de Login

Identificador	Requisito	Descrição	Prioridade
RF01	Login de Usuário	O sistema deve permitir que o usuário consiga fazer seu login através do seu e-mail e senha cadastrados.	Essencial
RF02	Cadastro de Usuário	O sistema deve permitir que um novo usuário consiga criar uma nova conta por um e-mail não existente no banco de dados	Essencial

Fonte: autor

A.2 Tela Home Page

Tabela 5 – Requisitos Funcionais da Tela Home Page

Identificador	Requisito	Descrição	Prioridade
RF03	Logoff de Usuário	O sistema deve permitir que o usuário que está logado consiga sair da sessão.	Essencial
RF04	Criação de Cards	O sistema deve permitir que o usuário consiga criar novos cards	Essencial
RF05	Edição de Cards	O sistema deve permitir que o usuário possa editar os cards previamente criados.	Essencial
RF06	Exclusão de Cards	O sistema deve permitir que o usuário possa excluir cards previamente criados.	Essencial

Fonte: autor