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

Desenvolvimento de Templates de Aplicações com Variantes Node.js e ReactJs

Ibsiany Dias Godinho

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
George Henrique Godim da Fonseca

Março, 2024
João Monlevade–MG

Ibsiany Dias Godinho

Desenvolvimento de Templates de Aplicações com Variantes Node.js e ReactJs

Orientador: George Henrique Godim da Fonseca

Monografia apresentada ao curso de Engenharia da 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

Março 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

Ibsiany Dias Godinho

Desenvolvimento de templates de aplicações com variantes Node.js e ReactJs

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 26 de setembro 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 12:05, 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 **0797903** e o código CRC **BEBCB119**.

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

"Dedico este trabalho à minha mãe que é minha motivação, pelo apoio constante em minha jornada. Sua presença inspiradora moldou não apenas quem sou como pessoa, mas também a trajetória realizada no ambiente acadêmico."

Agradecimentos

Agradeço primeiro a Deus por ter me mantido na trilha certa durante este projeto com saúde e forças para chegar até o final e por me guiar nesse caminho.

Minha imensa gratidão à minha família, que tem sido minha base e meu ponto de apoio ao longo desta jornada acadêmica. Seu constante incentivo e encorajamento foram essenciais para que eu seguisse adiante nos estudos. Agradeço por todo apoio incondicional ao longo desses anos, por estarem sempre ao meu lado nesta trajetória desafiadora. Em especial, quero dedicar um agradecimento à minha mãe, Walquiria, cujo amor e apoio inabaláveis são fontes de inspiração constantes. A ela, meu mais profundo agradecimento e admiração.

Expresso minha enorme gratidão ao meu orientador, George Henrique Godim da Fonseca, por sua prontidão e apoio incansáveis, pela orientação excepcional e pelo conhecimento transmitido ao longo do projeto. Sua dedicação e orientação foram fundamentais para o desenvolvimento e sucesso deste trabalho.

Aos meus amigos, que estiveram sempre ao meu lado, torcendo por mim e se tornando parte essencial nesta jornada acadêmica. Agradeço imensamente pelo apoio constante e pelas risadas compartilhadas, que melhoraram os momentos mais difíceis. O suporte e a apoio dessas amizades foram fundamentais para me manter motivada nos momentos mais desafioadores.

Por último, quero expressar minha mais sincera gratidão a todos que, de diversas maneiras, contribuíram para a realização deste trabalho. Cada colaboração, por mais pequena que possa parecer, desempenhou um papel crucial para chegar até aqui. Valorizo imensamente o esforço coletivo e a dedicação de todos os envolvidos, que foram essenciais para o êxito deste projeto. Este trabalho é verdadeiramente o resultado do trabalho em equipe e da colaboração mútua. Obrigada a cada um pelo seu valioso contributo.

"O importante é não parar de questionar. A curiosidade tem sua própria razão de existir"

— Albert Einstein (1879 – 1955)

Resumo

Iniciar um novo projeto em uma nova tecnologia muitas vezes representa um desafio, especialmente ao buscar documentação e exemplos em diversos lugares. O objetivo deste projeto consiste em desenvolver uma plataforma que disponha uma extensa variedade de templates destinados a diversas linguagens de programação, abarcando tanto o *backend* quanto o *frontend*. O intuito é fornecer acesso ao código fonte diretamente no GitHub, simplificando o processo de iniciação e acelerando o desenvolvimento em novos ambientes de programação. A partir essa plataforma, os usuários poderão escolher a tecnologia desejada tanto para o *backend* quanto para o *frontend*, possibilitando assim o acesso ao código-fonte específico das tecnologias selecionadas. O Chameleon Stack aborda tecnologias como Python e JavaScript para o *backend*, utilizando frameworks como Express e Nest, e frameworks *frontend* como Vue, Angular e React. Este trabalho concentra-se principalmente nas variações do Node.js com os *frameworks* Express e Nest, utilizando os bancos de dados PostgreSQL e MongoDB. Os templates são projetados para uma aplicação de gerenciamento de projetos e controle de tarefas no estilo Kanban, semelhante ao Trello. A proposta é criar uma solução simples que demonstre todos os exemplos de um Create, Read, Update and Delete (CRUD), consultas em banco de dados e manipulação de componentes no *frontend*. No *backend*, as conexões ao banco de dados serão realizadas utilizando tanto o TypeORM quanto o Prisma, com suporte a PostgreSQL e MongoDB. Além disso, foi incluído um template React para complementar os templates que foram gerados para o *frontend* do projeto.

Palavras-chaves: Chameleon Stack. Desenvolvimento Web. *backend*

Abstract

Starting a new project with a new technology often presents a challenge, especially when searching for documentation and examples across various sources. The goal of this project is to develop a platform that offers a wide variety of templates for different programming languages, encompassing both the backend and frontend. The intention is to provide access to the source code directly on GitHub, simplifying the initiation process and accelerating development in new programming environments. Through this platform, users will be able to choose the desired technology for both the backend and frontend, enabling access to the specific source code of the selected technologies. The Chameleon Stack addresses technologies such as Python and JavaScript for the backend, utilizing frameworks like Express and Nest, as well as frontend frameworks like Vue, Angular, and React. This work focuses primarily on variations of Node.js with the Express and Nest frameworks, using PostgreSQL and MongoDB databases. The templates are designed for a project management and task control application in a Kanban style, similar to Trello. The proposal is to create a simple solution that demonstrates all examples of a CRUD operation, database queries, and component manipulation on the frontend. On the backend, database connections will be handled using both TypeORM and Prisma, with support for PostgreSQL and MongoDB. Additionally, a React template has been included to complement the templates generated for the frontend of the project.

Keywords: Chameleon Stack. Web Development. Backend

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API Interface de Programação de Aplicação

CLI Command Line Interface

CRUD Create, Read, Update and Delete

CI/CD integração contínua e entrega contínua

ID Identificador

JWT JSON Web Token

ORM Object Relational Mapper

SGBD Sistema Gerenciador de Banco de Dados

SOLID *Single Responsibility Principle; Open-Closed Principle; Liskov Substitution Principle; Interface Segregation Principle; Dependency Inversion Principle*

URL Uniform Resource Locator

Sumário

1 Introdução

Segundo GONÇALVES (2002), desde os primórdios dos microcomputadores, os softwares têm incessantemente proporcionado benefícios e comodidades, uma realidade palpável no cotidiano. No cenário dinâmico e em constante evolução do desenvolvimento de software, a eficiência e produtividade tornam-se imperativos cruciais. Esse projeto aborda a criação do projeto *Chameleon Stack*, uma iniciativa que busca aprimorar a maneira como os estudantes e profissionais de desenvolvimento de software iniciam seus projetos, um site que armazena diversos códigos fonte para *frontend* e *backend*, eliminando a necessidade de começar um novo projeto do zero.

1.1 Problema

O surgimento de diversas tecnologias tem tornado a migração para a área de desenvolvimento de software cada vez mais desafiadora. Tanto para novos quanto para experientes desenvolvedores que buscam dominar novas tecnologias, a questão persiste: como iniciar um projeto nesse contexto? Com o propósito de aprimorar a eficiência dos desenvolvedores de software, planeja-se criar uma aplicação que sirva como ponto de partida para programadores em uma ampla gama de linguagens de programação e ferramentas. O objetivo é acelerar significativamente o processo de desenvolvimento ao fornecer código-fonte pronto para diversas linguagens, incluindo Python, Node.js, React, Vue e Angular.

Para a construção desse trabalho o foco será na linguagem de programação JavaScript. A escolha de linguagens de programação, como o JavaScript com a biblioteca Node.js, foi realizazda pela sua crescente popularidade como pode ser notado pela pesquisa do Overflow (2023a) e pelas oportunidades de emprego na área de programação como mostra a Figura 1, onde tem uma pesquisa realizada pelo TV (2024) que no *ranking* salarial o Node.js está em quarto lugar dentre todas as tecnologias no mercado. Um estudo conduzido por Costa e Oliveira (2020) revelou que, das 6935 vagas de emprego disponíveis no mercado naquele ano, aproximadamente 23% demandavam conhecimento em JavaScript, enquanto 7% eram especificamente voltadas para o Node.js. Em contraste, a taxa de inclusão dessas linguagens nos currículos das faculdades era de apenas 11% para JavaScript e 2% para Node.js, com base nesses dados, a inclusão da linguagem de programação JavaScript nas grades curriculares das faculdades deveria ser discutida de maneira mais aprofundada. Portanto, o objetivo principal é destacar o desenvolvimento dessas linguagens, que, embora sub-representadas no ensino superior, são altamente valorizadas e requisitadas no mercado de trabalho.

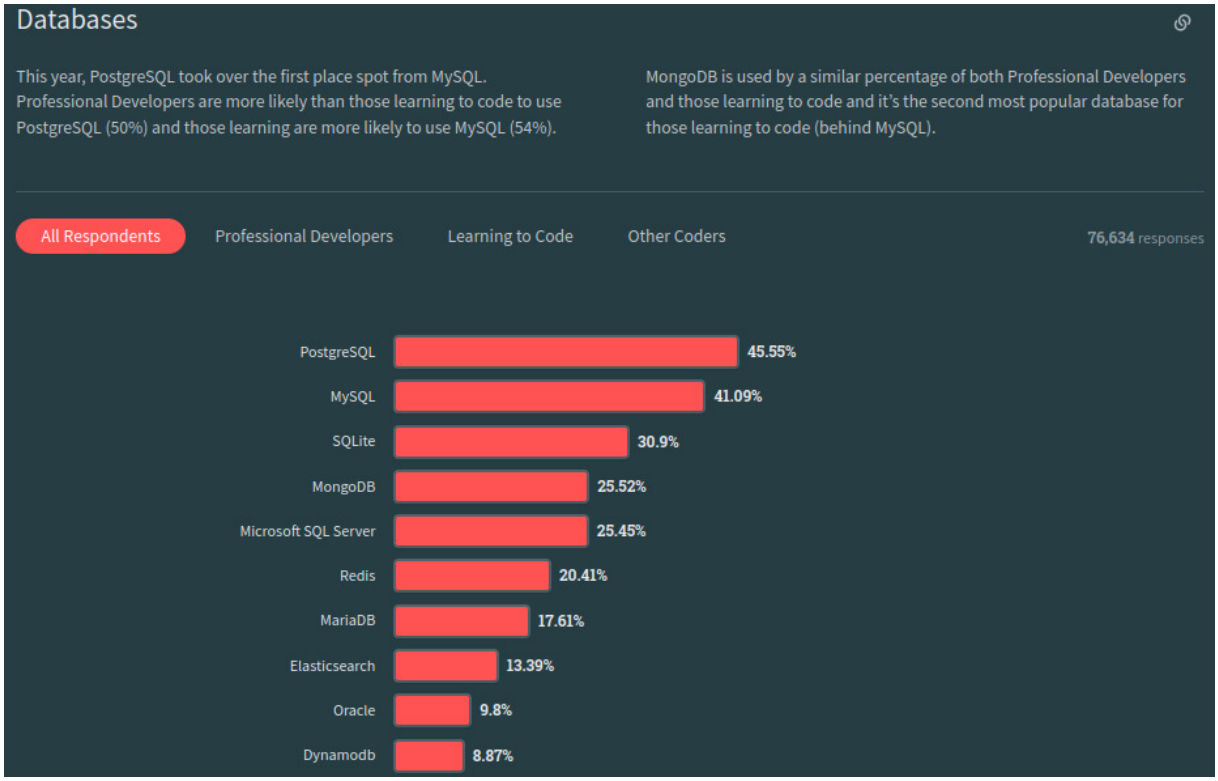
Figura 1 – Pesquisa salarial de Programadores Brasileiros 2024

Fonte: TV (2024)

Para a seleção do Sistema Gerenciador de Banco de Dados (SGBD), uma abordagem foi adotada visando a inclusão de um sistema relacional e um sistema não relacional, para que seja analisada as diferenças entre os diferentes SGBDs. Tal decisão se justifica pela grande gama de estratégias de armazenamento de dados usados em sistemas modernos. Com o intuito de abranger uma variedade de perfis de usuários, optou-se por integrar sistemas tanto relacionais quanto não relacionais. O PostgreSQL[1] foi selecionado como representante dos SGBD relacionais, sendo reconhecido como a principal escolha no cenário atual, conforme corroborado por pesquisas recentes realizadas por Overflow (2023a). Enquanto isso, para o segmento dos SGBDs não relacionais, o MongoDB foi escolhido, posicionando-se como a segunda opção mais prevalente de acordo com estudos conduzidos também pelo Overflow (2023a). A Figura 2 apresenta um gráfico que mostra os bancos de dados relacionais e não relacionais mais utilizados pelos desenvolvedores, desde iniciantes até profissionais experientes, o PostgreSQL lidera a posição de ferramenta mais utilizada representando 45% de um total de 76.634 pessoas entrevistadas para o formulário da pesquisa.

[1] <https://www.postgresql.org>

Figura 2 – Sistemas Gerenciadores de Bancos de Dados mais utilizados



Fonte: Overflow (2023a)

O Prisma[2] foi empregado como um Mapeador Objeto-Relacional *Object Relational Mapper (ORM)*, em virtude de sua capacidade de integração com ambos os paradigmas de banco de dados, tanto relacional quanto não relacional. Por sua vez, o TypeORM[3] foi selecionado devido à sua documentação abrangente e meticulosa, fornecendo recursos detalhados e abrangentes para o desenvolvimento e a integração.

1.2 Objetivo

O principal objetivo deste trabalho é conceber e implementar o projeto *Chameleon Stack*. O intuito é fornecer acesso ao código fonte diretamente no link[4], simplificando o processo de iniciação e acelerando o desenvolvimento em novos ambientes de programação. O foco principal desse trabalho abrangerá diversas variações do Node.js com TypeScript, incluindo *Frameworks* como Express e Nest, diferentes conexões de dados como TypeORM e Prisma, e bancos de dados, como PostgreSQL e MongoDB. Adicionalmente, também está disponível um codigo-fonte em ReactJS para a aumentar as variações de templates para o *frontend*. Outros membros do grupo *Chameleon Stack* estarão responsáveis por

[2] <https://www.prisma.io/docs>
[3] <https://typeorm.io>
[4] <https://github.com/Chameleon-Stack>

entregas que abrangem uma variedade de tecnologias tanto no *backend* quanto no *frontend*, incluindo o uso de Python, Vue e Angular.

A proposta é eliminar a necessidade de iniciar um novo projeto a partir do zero, proporcionando aos desenvolvedores um ponto de partida eficiente e flexível. Ao focar nas tecnologias populares e altamente demandadas no mercado de trabalho, como Node.js, o projeto busca acelerar significativamente o processo de desenvolvimento e aprimorar a eficiência dos desenvolvedores.

O trabalho também busca resolver o problema crescente da complexidade na migração para novas linguagens de programação, oferecendo uma solução prática para a questão de como iniciar um projeto nesse contexto desafiador. Embora Bunnyshell[5] ofereça opções pagas para esse fim, o foco reside em transformar esse projeto em uma iniciativa universitária e de código aberto.

Como objetivos específicos tem-se:

- Implementar aplicação de gerenciamento de projetos e controle de tarefas no estilo Kanban, com variadas tecnologias *backend*;
- Implementar *frontend* utilizando o *framework* React;
- Implementar documentação e testes nas aplicações;
- Implementar dockerização nos projetos;
- Disponibilizar a aplicação em um serviço de hospedagem, utilizando conceitos de integração contínua e entrega contínua (CI/CD).

1.3 Organização do trabalho

A estrutura subsequente deste trabalho é delineada como segue. O Capítulo 2 oferece uma análise detalhada da fundamentação teórica subjacente ao desenvolvimento do *Chameleon Stack*, assim como a seleção criteriosa das tecnologias empregadas nos templates. O Capítulo 3 discorre sobre os procedimentos adotados durante o processo de desenvolvimento, abordando igualmente os padrões de implementação estabelecidos para as distintas tecnologias envolvidas. No Capítulo 4, serão discutidos os resultados referentes ao desempenho dos templates desenvolvidos para as tecnologias abordadas neste projeto. Por último, o Capítulo 5 engloba as considerações finais, acompanhado de sugestões para trabalhos futuros.

[5] <https://template-nodejs-nest-postgresql.onrender.com/api>

2 Fundamentação Teórica

Neste capítulo, será discutida a relevância da eficiência no desenvolvimento de software, além de abordar as tecnologias empregadas na construção dos templates. Dentre essas tecnologias, incluem-se a escolha entre MongoDB e PostgreSQL, bem como o uso de frameworks como Express, Nest e Node.js, juntamente com bibliotecas como Prisma, TypeORM, React.js, Material-UI e para hospedagem online foi utilizado o Render e Supabase.

2.1 Eficiência no desenvolvimento de *software*

A necessidade de eficiência na área de desenvolvimento de software foi historicamente subestimada, como indicado por Alcantara e Sant'Anna (2002). O foco predominante estava na expansão quantitativa de sistemas, em detrimento da qualidade e da otimização dos processos. Alcantara e Sant'Anna (2002) também inclui uma análise da eficiência dos programadores, considerando diversas variáveis, entre as quais se destaca a motivação da equipe. A medição da eficiência dos programadores é uma questão crucial no contexto do desenvolvimento de software, uma vez que está diretamente ligada à qualidade e à produtividade do trabalho realizado. No entanto, conforme destacado por Silva (2016), houve uma mudança significativa nessa perspectiva ao longo do tempo. O aprimoramento da eficiência dos programadores emergiu como um tema central de estudo, refletindo-se no reconhecimento do impacto do ambiente de trabalho e dos processos adotados na criação de novos sistemas.

Nesse contexto, o Chameleon Stack surge como uma ferramenta que visa promover a eficiência tanto para programadores iniciantes quanto para aqueles mais experientes, que buscam soluções prontas para reutilização em seus projetos. Ao oferecer uma estrutura flexível e adaptável, o Chameleon Stack facilita a criação e a integração de componentes de software, contribuindo para a agilidade e a qualidade do desenvolvimento.

Essa mudança de paradigma, no qual a eficiência torna-se um objetivo central, reflete não apenas uma evolução na compreensão da prática de desenvolvimento de software, mas também uma resposta às demandas crescentes por sistemas mais robustos e ágeis. Portanto, iniciativas como o Chameleon Stack representam um passo significativo em direção à construção de um ecossistema de desenvolvimento mais eficiente e colaborativo.

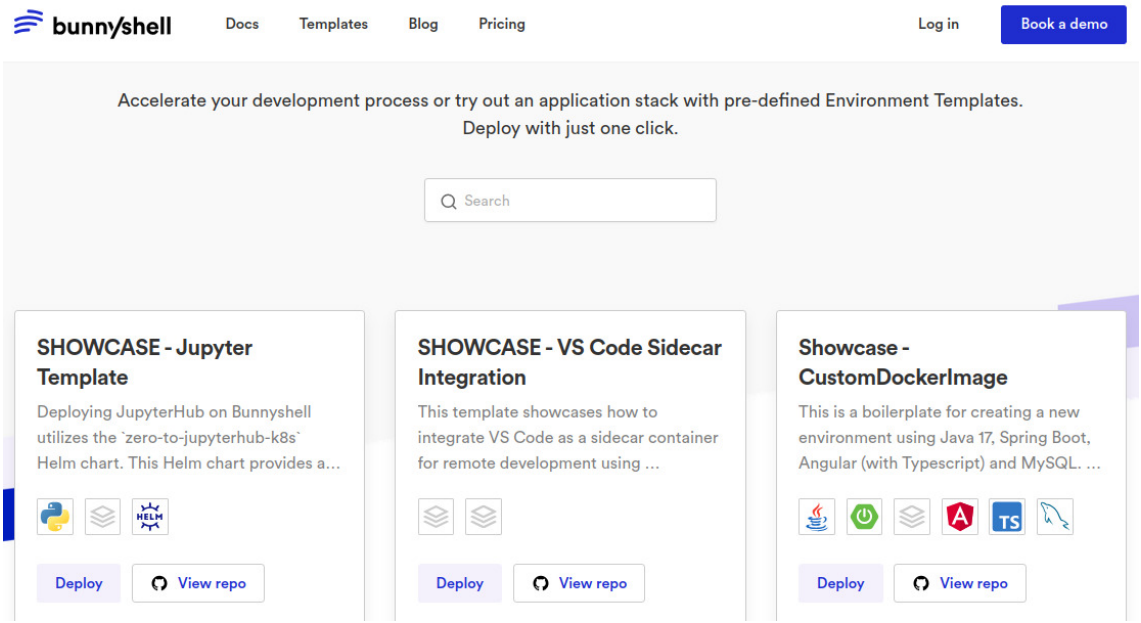
2.2 Trabalhos relacionados

Para a criação do projeto, foi realizada uma pesquisa e estudo de ferramentas já existentes. Muitas dessas ferramentas possuem premissas semelhantes às deste trabalho, cada uma com seus diferenciais.

2.2.1 Bunnyshell

A primeira plataforma analisada foi o Bunnyshell[1], que oferece um website com exemplos de código-fonte, porém com versões limitadas na versão gratuita. A manipulação do site é limitada ao registro, e o processo de instalação dos templates pode ser um pouco complexo para usuários sem experiência nas tecnologias. A Figura 3 mostra os templates que estão disponíveis no site.

Figura 3 – Imagem do website Bunnyshell tela de templates

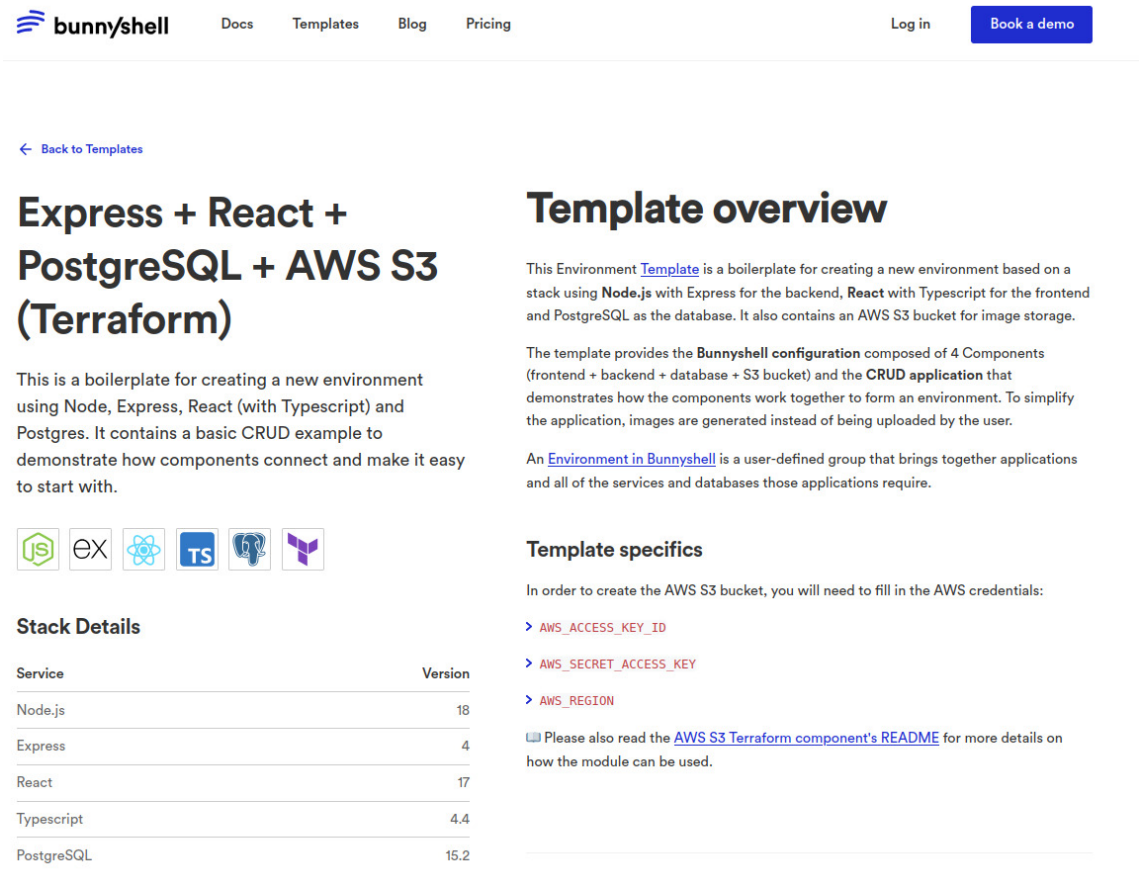


Fonte: produzido pela autora, retirado de BUNNYSHELL (2024)

A Figura 4 de uma aba do site apresenta uma explicação de como utilizar a ferramenta e caminhos para a instação do serviço.

[1] <https://template-nodejs-nest-postgresql.onrender.com/api>

Figura 4 – Template Express + React + PostgreSQL + AWS S3 no Bunnyshell

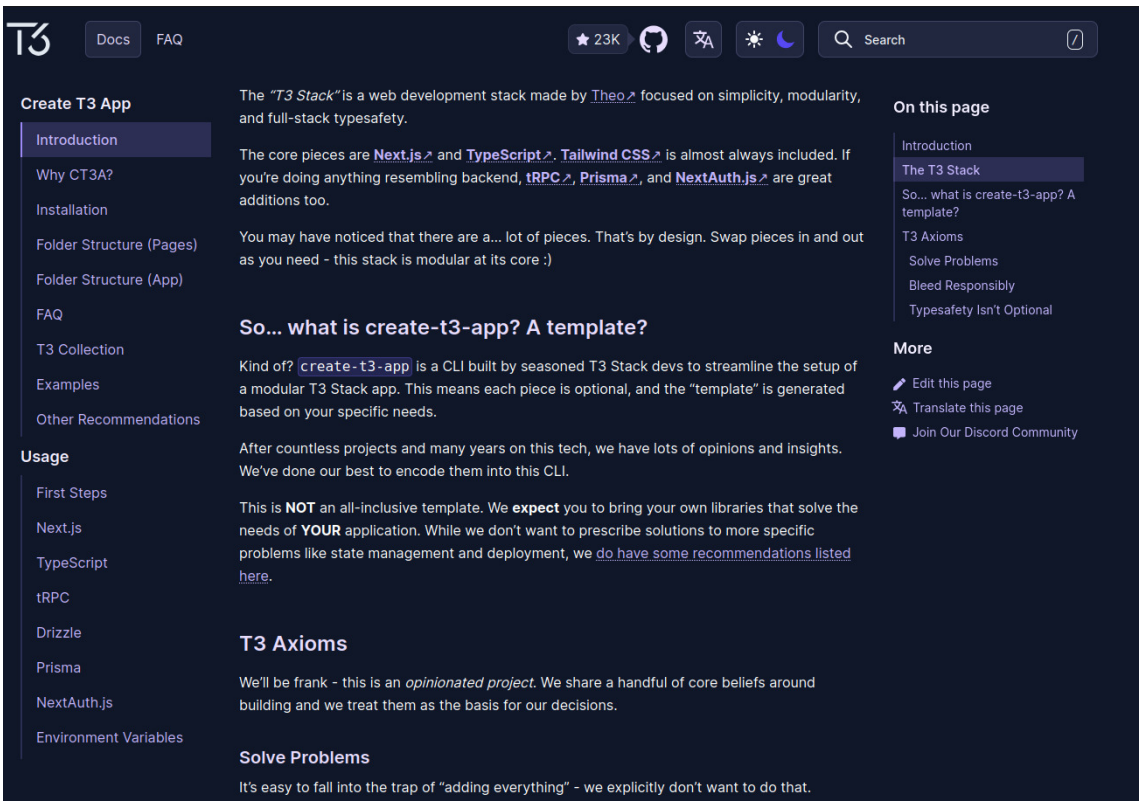


Fonte: produzido pela autora, retirado de BUNNYSHELL (2024)

2.2.2 T3 Stack

O T3[2] Stack foi outra ferramenta estudada e possui um conjunto de ferramentas e linguagens acessíveis via interface de linha de comando *Command Line Interface (CLI)* para facilitar o início do desenvolvimento de um aplicativo modular, conforme descrito no próprio site e como mostra a Figura 5, seu diferencial é justamente a interface por linha de comando.

Figura 5 – Dashboard do T3 Stack



Fonte: produzido pela autora

O diferencial oferecido pelo Chameleon Stack é a interface intuitiva, além da disponibilização de aplicações online com documentação completa e ferramentas integradas para testes. Sua abordagem de código aberto também abre portas para colaborações e personalizações.

2.3 Ferramentas utilizadas

A seleção das ferramentas certas é fundamental no desenvolvimento de software, afetando diretamente a eficiência e a qualidade do produto final. Dada a variedade de tecnologias disponíveis para frontend e backend, foi optado por utilizar Node.js no *backend*

[2] <https://create.t3.gg/en/introduction>

e React no *frontend*. O GitHub foi selecionado para o controle de versão devido à sua acessibilidade e ao uso gratuito para todos os membros da equipe. Além disso, ferramentas como Render[3] e Supabase[4] foram escolhidas pela gratuidade e facilidade de uso. A seguir, serão detalhadas as tecnologias utilizadas.

2.3.1 Node.js

Node.js é uma ferramenta para a construção de Interfaces de Programação de Aplicação (APIs), criada em 2009. Trata-se de um ambiente de execução JavaScript baseado no motor V8 do Google Chrome, que permite aos desenvolvedores criarem aplicações do lado do servidor. Rosa (2023) fala sobre a utilização do V8 no JavaScript, e como o V8 age implementando uma linguagem de script chamada ECMAScript, conforme especificado na ECMA-262. Além disso, o V8 é uma *engine* de tempo de execução de código aberto que é escrita na linguagem de programação C++.

A utilização do Node.js como aplicação *backend* traz diversas vantagens. Conforme destacado por Nunes (2018), o Node.js possui um interpretador C++ que torna a linguagem mais rápida e eficiente. Além disso, permite a utilização do JavaScript no *backend*, proporcionando uma maior integração entre o desenvolvimento *frontend* e *backend*. Outra vantagem é sua leveza, o que o torna viável para qualquer sistema operacional. Por fim, vale destacar a escalabilidade dos serviços construídos com Node.js.

O Node.js pode ser utilizado com diversos frameworks, como Nest e Express. Este projeto visa ilustrar a codificação em ambos os frameworks, utilizando diferentes tipos de bancos de dados: relacionais e não relacionais. Desse modo, quem iniciar o desenvolvimento com base neste projeto, o projeto atenderá a diversas necessidades específicas.

A pesquisa realizada pelo Overflow (2023b) e pelo TV (2024) mostra a alta demanda do Node.js, conforme mostrado na Figura 6 e Figura 1. Em 2023, o Node.js lidera a lista de popularidade entre *frameworks* e tecnologias utilizadas, com de 42.65% de respostas de pessoas que utilizam a ferramenta de um total de 71.802 usuários que responderam a pesquisa do StackOverflow.

[3] <https://render.com/>
[4] <https://supabase.com/open-source>

Figura 6 – *Frameworks* e tecnologias de desenvolvimento de software mais utilizados

Fonte: Overflow (2023b)

2.3.2 Express

O Express é um *framework* criado em 2010 utilizado no Node.js para facilitar o processo de desenvolvimento de aplicações web. O Express[5] surgiu em 2010 com o objetivo de simplificar a configuração de servidores backend. Antes do seu surgimento, os desenvolvedores precisavam configurar servidores usando apenas o módulo HTTP nativo do Node.js, tornando o processo mais complexo e trabalhoso.

Além de facilitar o processo de configuração do servidor, atualmente o Express também auxilia na criação de Interfaces de Programação de Aplicação API RESTful, gerenciamento de rotas, integração de middleware para autenticação, validação de dados, manipulação de sessões, e suporte a templates para renderização de páginas dinâmicas.

2.3.3 Nest

O Nest é um *framework* lançado em 2017 que, assim como o Express, veio para facilitar o processo de desenvolvimento de aplicações do lado do servidor. De acordo com o artigo (SABO, 2020), o Nest oferece escalabilidade graças à sua estrutura modular, permitindo que os desenvolvedores construam aplicações organizadas e eficientes. Embora inspirado pelo Express, o Nest se destaca por sua arquitetura flexível, que permite a configuração do *framework* para utilizar diferentes estruturas conforme necessário. Além disso, o Nest atua como uma camada de abstração sobre as estruturas existentes, simplificando a escrita de código e promovendo boas práticas de desenvolvimento.

[5] <https://expressjs.com/pt-br/>

Neste trabalho, o Nest foi utilizado em conjunto com o Node.js e diferentes variações de banco de dados. O projeto desenvolvido com Nest foi disponibilizado online e pode ser acessado por meio do link[6].

2.3.4 React

React é uma biblioteca de código aberto que utiliza a linguagem de programação JavaScript, desenvolvido pelo Facebook, e é amplamente utilizado para construir interfaces de usuário como uma tecnologia *frontend*. Criado em 2013, React se destaca por facilitar a criação de interfaces complexas de modo eficiente e dinâmico. A reatividade e a componentização proporcionadas pelo React revolucionaram a maneira como as interfaces de usuário são criadas, tornando-as mais dinâmicas, interativas e responsivas.

Conforme mencionado por Augusto e Carvalho (2023), o React destaca-se como uma das tecnologias mais utilizadas no desenvolvimento web atualmente. Sua popularidade deve-se à sua escalabilidade, facilitada pelo fato de ser um *framework* de código aberto, beneficiando-se da contribuição contínua de uma grande comunidade de desenvolvedores.

O React se destaca como um *framework* amplamente utilizado e requisitado no mercado de desenvolvimento *web*, liderando as tecnologias e *frameworks* mais populares para o *frontend*, conforme pesquisa realizada pelo Stack Overflow Overflow (2023b). A Figura 6 citada anteriormente mostra a colocação do React em comparação com outras tecnologias utilizados pelos desenvolvedores, desde iniciantes até profissionais experientes.

2.3.5 GitHub

O GitHub é uma plataforma para hospedagem e gerenciamento de código-fonte que é um centralizador no desenvolvimento de software. Ele foi criado em torno do sistema de controle de versão Git, e oferece um ambiente acessível e gratuito para acessar e controlar projetos de programação. Recursos como a criação de *branches*, abertura de *pull requests*, revisão de código e gerenciamento de *issues* tornam o GitHub uma ferramenta indispensável para o desenvolvimento moderno de software.

Durante o projeto, o GitHub foi amplamente utilizado pelos desenvolvedores para colaborar em diversos repositórios e trabalhar com uma variedade de linguagens de programação.

2.3.6 Supabase

De acordo com Asse (2024) o Supabase foi fundado em 2020 por Paul Copplestone e Ant Wilson. A ideia por trás do projeto surgiu da necessidade de uma solução de backend

[6] <https://template-nodejs-nest-postgresql.onrender.com/api>

com código aberto e transparente, que pudesse ser auto-hospedada e que oferecesse recursos semelhantes ao Firebase, mas com a flexibilidade do PostgreSQL. O objetivo era criar uma plataforma que facilitasse o desenvolvimento de aplicativos modernos, sem comprometer a liberdade e o controle sobre a infraestrutura.

O Supabase é uma plataforma de código aberto projetada para fornecer uma alternativa ao Firebase, oferecendo serviços de backend como autenticação, banco de dados em tempo real, armazenamento e funções *serverless*. A história do Supabase é marcada por uma rápida evolução e adoção dentro da comunidade de desenvolvedores.

A escolha do Supabase para o projeto foi motivada por sua interface intuitiva e pela facilidade de manipulação. Ao criar um projeto de banco de dados na plataforma, uma *Uniform Resource Locator (URL)* é disponibilizada, permitindo o acesso por qualquer um dos projetos construídos, incluindo o projeto que está online.

2.3.7 Render

Conforme citado pelo próprio *website* Render[7] é uma plataforma de hospedagem de aplicativos em nuvem que se destaca pela facilidade de uso e eficiência, permitindo que desenvolvedores implementem e escalem seus aplicativos rapidamente. O Render foi lançado em 2018, devido a necessidade de uma solução mais simples e integrada para a implantação de aplicações.

Desde o seu lançamento, Render ganhou popularidade rapidamente na comunidade de desenvolvedores, graças à sua interface amigável e aos recursos poderosos. A plataforma suporta várias ferramentas, como Node.js, Python, Ruby, Docker, e muitos outros, permitindo uma grande flexibilidade para os desenvolvedores.

O Render foi utilizado para hospedar e manter online o projeto desenvolvido em Nest, enquanto foi utilizado a URL gerada pelo Supabase para a utilização do banco de dados PostgreSQL.

2.3.8 Vercel

De acordo com o *website* Vercel[8], é uma plataforma em nuvem como serviço. A estrutura de desenvolvimento web da Vercel é em Next.js. A plataforma é focada principalmente em aplicações *frontend*. A plataforma é simples e possui um *layout* intuitivo, eficiente e escalável para o deploy de sites estáticos e aplicações web dinâmicas.

O Vercel foi utilizado para hospedar e manter online o projeto desenvolvido em React, e está disponível no link[9].

[7] <https://dashboard.render.com/>
[8] <https://vercel.com>
[9] <https://template-react-material-ui.vercel.app/>

3 Desenvolvimento

Este capítulo apresenta o desenvolvimento do projeto, abrangendo desde a implementação dos templates até o processo de deploy. Serão abordadas a arquitetura escolhida para a construção do sistema, os casos de uso, as rotas propostas para o Kanban e a modelagem do banco de dados. Para que os objetivos específicos sejam alcançados, foi seguido a seguinte estrutura para o desenvolvimento:

- Modelagem do banco de dados, como apresentado na Figura 12;
- Definição de variantes a serem construidos com o Node.js;
- Implementação de projetos *backend*, contendo documentação e testes unitários;
- Disponibilização online do repositório utilizando Nest e PostgreSQL no *Render*;
- Desenvolvimento da aplicação frontend com React e MaterialUI.

3.1 Visão geral do sistema

O projeto contemplará construir templates com diversas tecnologias para que o inicio de um projeto seja facilitado. Os templates foram projetados com base em um projeto Kanban, que se baseia em um gerenciador de tarefas, será disponibilizado o código-fonte para utilização na página principal. Os projetos foram construídos com o conceito API rest, utilizando principios de *Single Responsibility Principle; Open-Closed Principle; Liskov Substitution Principle; Interface Segregation Principle; Dependency Inversion Principle* (SOLID).

Observando a Figura 7 a respeito da classe *GetUserByIdService* e a Figura 8, pode-se analisar os 5 princípios do SOLID atendidos, sendo eles:

- Princípio da responsabilidade única (S): Na classe GetUserByIdService ilustrada na Figura 7 a classe tem uma única responsabilidade que é buscar um usuário pelo Identificador (ID).
- Princípio Aberto-Fechado (O): A classe GetUserByIdService está aberta para extensão, mas fechada para modificação. Caso necessário adicionar novas funcionalidades, pode ser estendido a classe ou criar novas classes sem modificar a implementação existente.

- Princípio da substituição de Liskov (L): A classe depende de uma interface sendo ela *IUserRepository* ilustrada na Figura 8. Permitindo que qualquer implementação de *IUserRepository* possa ser usada, e garantindo que a substituição de implementações não quebre a funcionalidade da classe.
- Princípio da Segregação da Interface (I): A interface *IUserRepository* define apenas os métodos necessários para manipulação de usuários como ilustrado na Figura 8, garantindo que GetUserByIdService não dependa de métodos que não utiliza.
- Princípio da inversão da dependência (D): A classe *GetUserByIdService* depende de uma abstração *IUserRepository* e não de uma implementação concreta. Isso é evidenciado pelo uso da injeção de dependência com o decorator @inject.

Figura 7 – Código fonte da classe: Busca de usuário pelo identificador

```
1  @injectable()
2  export class GetUserByIdService {
3    constructor(
4      @inject('UserRepository')
5      private usersRepository: IUserRepository,
6    ) {}
7
8    async execute(id: string): Promise<User> {
9      const user = await this.usersRepository.findById(id);
10
11     if (!user) {
12       throw new LibError('User does not exists!', 404);
13     }
14
15     return user;
16   }
17 }
18
```

Fonte: produzido pela autora

Figura 8 – Código fonte da interface de usuários

```
export interface IUserRepository {
  create(new_user: ICreateUserDTO): Promise<User>;
  update(user: User): Promise<User>;
  findById(id: string): Promise<User | null>;
  findByEmail(email: string): Promise<User | null>;
  delete(user: User): Promise<void>;
}
```

Fonte: produzido pela autora

3.1.1 Página inicial

A página principal foi construída por um membro do grupo e irá conter as aplicações disponíveis, conforme Figura 9 primeiramente é apresentado um pouco sobre o projeto.

Figura 9 – Página Inicial do site

Fonte: produzido pela autora

Logo em seguida, ao clicar no botão “Template”, são apresentados os templates disponíveis. Como o foco deste projeto é o *backend*, a Figura 10 mostra alguns dos templates dessa categoria. Também é possível acessar outras áreas, como *frontend*, mobile, *fullstack*, visualizar todas as tecnologias disponíveis ou até realizar uma busca por uma tecnologia específica.

Figura 10 – Página Inicial do site

Fonte: produzido pela autora

3.2 Levantamento de requisitos

A seguir, serão detalhados os requisitos funcionais e não funcionais do sistemas criados para serem utilizados como template no website, além do diagrama de caso de uso, diagrama de atividades e a modelagem do banco de dados de maneira detalhada. Os requisitos foram definidos para simular os requisitos comuns em aplicações de websites para gerenciamento de quadros Kanban.

3.2.1 Requisitos funcionais

A seguir os requisitos funcionais dos projetos criados com a linguagem de programação JavaScript utilizando a ferramenta Node.js.

1. Permitir a criação de usuários na plataforma;
2. Permitir o gerenciamento de cards, incluindo criar, atualizar e excluir;
3. Permitir o gerenciamento de categorias, incluindo criar e excluir;
4. Permitir a edição do usuário, permitindo alteração de foto e dados pessoais;

5. Fornecer uma documentação completa do sistema;
6. Disponibilizar um guia de instalação e utilização do sistema localmente.

3.2.2 Requisitos não funcionais

A seguir, são apresentados os requisitos não funcionais:

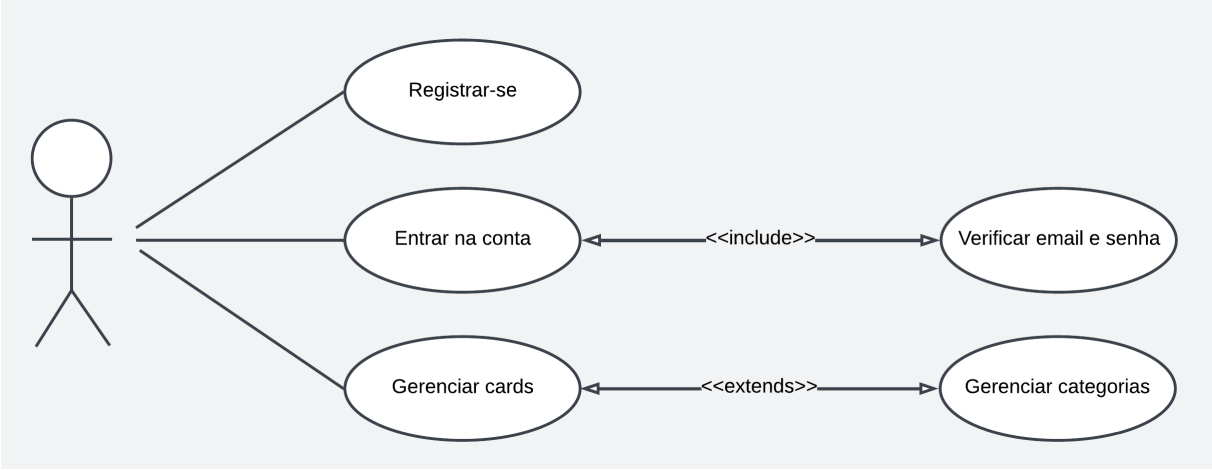
1. Cobertura de testes unitários do sistema, garantindo a confiabilidade das funcionalidades implementadas;
2. Disponibilizar um guia de instalação e utilização do sistema localmente;
3. O projeto suporta um aumento gradual no número de usuários e cards gerenciados, sem perda significativa de desempenho.

3.3 Diagrama de Casos de Uso

A Figura 11 apresenta o diagrama dos templates desenvolvidos neste projeto. O diagrama foi criado utilizando o site Lucid. Nele, foi definido apenas um ator, que é:

- Usuário: Responsável pelo gerenciamento de tarefas por meio do sistema.

Figura 11 – Diagrama de Caso de Uso



Fonte: produzido pela autora

A seguir serão apresentado as histórias referente ao diagrama de caso de uso.

3.3.1 Registrar usuário

Para acessar o sistema, o usuário precisa criar uma conta. A criação da conta é feita na tela inicial, no qual o usuário deve fornecer dados pessoais como email, nome, senha e, opcionalmente, uma foto de perfil.

3.3.2 *Login*

Para fazer o *login* no sistema, o usuário deve fornecer o email e a senha corretamente, conforme informado durante o registro.

3.3.3 Gestão de *card*

Após fazer o *login* na plataforma, o usuário poderá gerenciar suas tarefas utilizando um quadro no estilo Kanban. Ele poderá criar novas tarefas, além de editar e excluir as existentes.

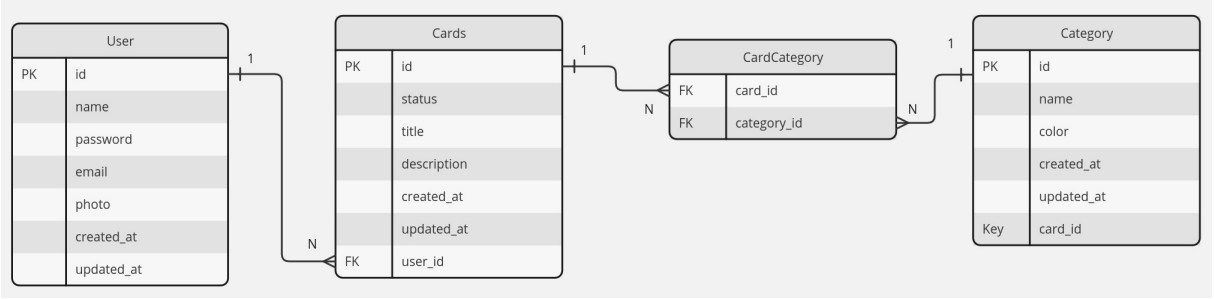
3.3.4 Gestão de categorias

Após cadastrar uma nova tarefa no *board*, o usuário pode adicionar categorias aos *cards*. Também é possível remover essas categorias dos *cards* ou excluí-las da conta. A gestão de *cards* e categorias é realizada pelo usuário logado.

3.4 Banco de dados

O banco de dados possui uma modelagem simples, conforme ilustrado na Figura 12. O objetivo do projeto é ter uma modelagem simples com conexões 1:N e N:N, abrangendo várias linguagens de programação e incluindo bancos de dados não relacionais.

Figura 12 – Modelagem do banco de dados



Fonte: produzido pela autora

A seguir está detalhado as tabelas presentes na modelagem do banco de dados.

3.4.1 Usuários

Esta tabela armazena os dados dos usuários registrados no sistema. Cada linha representa um usuário e inclui informações pessoais, como nome, email, senha e, opcionalmente, uma foto. Além disso, a tabela possui um relacionamento com as tabelas de *cards* e categorias, permitindo controlar quais tarefas e categorias cada usuário pode visualizar.

3.4.2 *Cards*

A tabela de *card* armazena os dados das tarefas criadas no *board* do sistema. Cada linha representa um *card* criado na tela e inclui os seguintes atributos: nome, título, descrição e status. Além disso, há uma coleção de identificadores que relaciona o *card* às categorias correspondentes, facilitando a organização e classificação das tarefas.

3.4.3 Categorias

Esta tabela armazena os dados das categorias no sistema. Cada linha representa uma categoria e inclui os seguintes atributos: nome, cor e identificador do usuário. A cor define a aparência da categoria no *frontend*; por exemplo, uma categoria relacionada a um *bug* pode ter a cor vermelha, indicando criticidade. O identificador do usuário, representa o usuário que criou a categoria, permitindo identificar e gerenciar as categorias específicas de cada usuário.

4 Resultados

Neste capítulo é apresentado testes unitários, documentação das APIs desenvolvidas, bem como as análises e resultados de desempenho ao teste de estresse (*stress testing*), e o as telas desenvolvidas para o frontend.

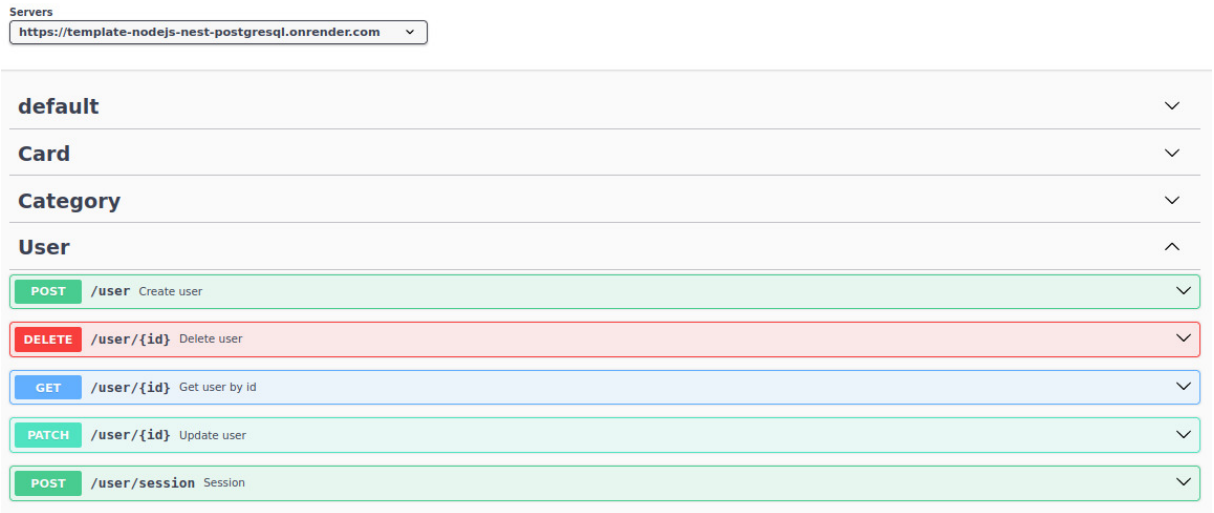
4.1 Documentação

Neste tópico, será apresentada a documentação dos *endpoints* criados para a API que implementa a modelagem descrita no Capítulo 3. A documentação também está disponível online no site [1] para consultas futuras.

4.1.1 Usuário

A Figura 13 apresenta a documentação referente ao CRUD de usuários, contendo *endpoints* para criação, busca, deleção, edição de usuários e até mesmo um método para realizar o *login* do usuário.

Figura 13 – Documentação: CRUD de usuário



Fonte: Produzida pela própria autora

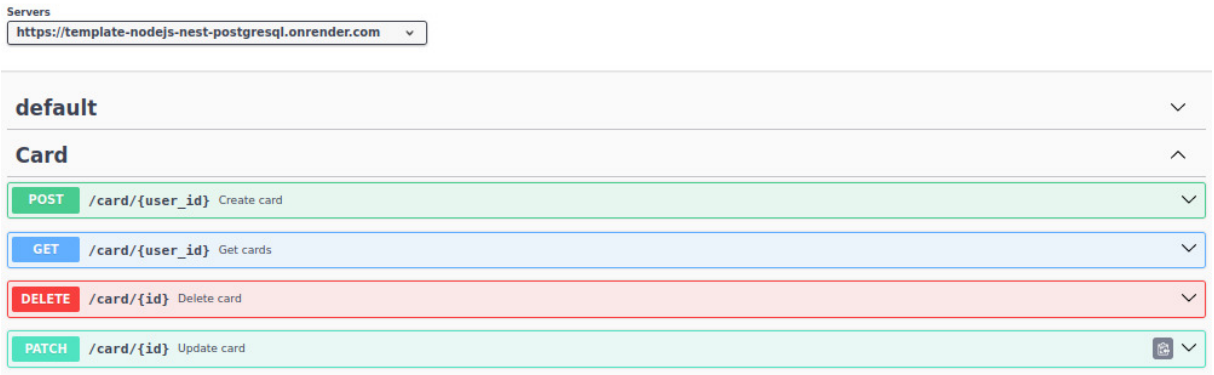
4.1.2 Cards

Para gerenciar os cards pelas rotas disponibilizadas, é necessário ter um usuário previamente registrado. A Figura 14 mostra um CRUD simples, no qual é possível criar,

[1] <https://template-nodejs-nest-postgresql.onrender.com/api>

ler, editar e deletar um card. Conforme a modelagem apresentada na Figura 12, o card está relacionado a um usuário. Portanto, para buscar ou criar os cards, é necessário informar o usuário que está realizando a ação. Já para deletar ou atualizar o card, basta informar o identificador único do card.

Figura 14 – Documentação: CRUD de card



Fonte: Produzida pela própria autora

4.1.3 Categorias

A Figura 15 mostra o CRUD de categorias. Pode-se observar que ele é mais simples, permitindo apenas criar, deletar e ler categorias, sem a opção de editá-las. Isso ocorre devido às necessidades do projeto, conforme será apresentado nos próximos tópicos, no qual não foi necessário editar categorias via *frontend*.

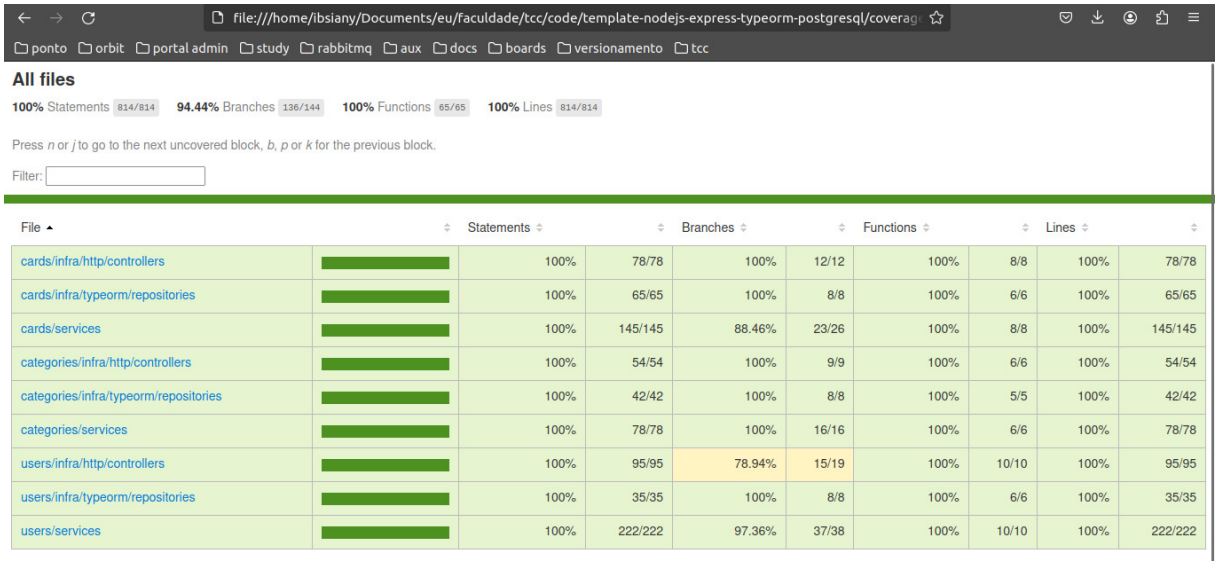
Figura 15 – Documentação: CRUD de categorias

Fonte: Produzida pela própria autora

4.2 Testes Unitários

Com auxilio da biblioteca Jest[2], foi possível realizar testes unitários nas seis aplicações *backend* com cobertura de 100% dos casos em classes e métodos implementados como mostra a Figura 16.

Figura 16 – Teste unitário utilizando JEST (2024)



File	Statements		Branches		Functions		Lines	
cards/infra/http/controllers	100%	78/78	100%	12/12	100%	8/8	100%	78/78
cards/infra/typeorm/repositories	100%	65/65	100%	8/8	100%	6/6	100%	65/65
cards/services	100%	145/145	88.46%	23/26	100%	8/8	100%	145/145
categories/infra/http/controllers	100%	54/54	100%	9/9	100%	6/6	100%	54/54
categories/infra/typeorm/repositories	100%	42/42	100%	8/8	100%	5/5	100%	42/42
categories/services	100%	78/78	100%	16/16	100%	6/6	100%	78/78
users/infra/http/controllers	100%	95/95	78.94%	15/19	100%	10/10	100%	95/95
users/infra/typeorm/repositories	100%	35/35	100%	8/8	100%	6/6	100%	35/35
users/services	100%	222/222	97.36%	37/38	100%	10/10	100%	222/222

Fonte: Produzida pela própria autora

4.3 Testes realizados

Para realizar o *stressing test* na aplicação, foi desenvolvido o seguinte script para realizar diversas chamadas na API. A Figura 17 mostra o código utilizado. Para a variável *request* o valor utilizado foi de 10 mil, ou seja, foi realizado dez mil requisições para criação de dados nas APIs.

[2] <https://jestjs.io/pt-BR/>

Figura 17 – Script de teste de estresse na API: Criação de cards

```
async function stressTest() {
    console.time('Execution time Template Express + Typeorm');
    for (let i = 0; i < requests; i++) {
        const randomNumber = randomInt(0, 10);

        /*
        users -> Usuários pré cadastrados
        */

        try{
            await  axios.post(`http://localhost:3333/card/${users[randomNumber]}`,
            {
                title: `Create card ${i} Title`,
                description: `Create card ${i} Description`,
                status: '10'
            }
            )

        }catch(error){
            console.log(error)
        }
    }
    console.timeEnd('Execution time Template Express + Typeorm')
}

stressTest()
```

Fonte: Produzida pela própria autora

Os testes foram realizados com conexão local. E com o auxílio da documentação do Prisma (2024) foi possível criar um ambiente local para a execução dos projetos que utilizam o bando de dados Mongo. Já para o banco de dados PostgreSQL a integração local foi mais simples, apenas criando um container com o passo a passo que está salvo no próprio projeto[3].

Para os testes foram criados 10 usuários, para os quais foram criados cards de

[3] <https://github.com/Chameleon-Stack>

maneira aleatória. Para os testes de parametrização de busca de dados em ambos as tecnologias foi reutilizado o script da Figura 17, porém alterado a chamada para busca de cards por usuário, conforme a Figura 18.

Figura 18 – Script de teste de estresse na API: Listagem

```
const requests = 10

async function stressTest() {
    console.time('Execution time Template Nest + Typeorm + PostgreSql');

    for (let i = 0; i < requests; i++) {
        try{
            await  axios.get(`http://localhost:3000/card/${users[i]}`)
        }catch(error){
            console.log(error)
        }
    }
    console.timeEnd('Execution time Template Nest + Typeorm + PostgreSql')
}
```

Fonte: Produzida pela própria autora

A Tabela 1 apresenta os resultados da execução dos dois scripts apresentados anteriormente. Para o processo de escrita no banco de dados foram realizadas 10 mil requisições diretas para a API, criando um card por rota, nos 6 diferentes serviços. A coluna “Escrita” apresenta o resultado a respeito desse teste.

Para a coleta de quanto tempo demorava a busca foi utilizado os dados anteriores e mais de 285 mil cards criados em tabelas do banco de dados MongoDB e PostgreSQL. A Tabela 1 mostra o resultado na coluna “Leitura” a respeito da leitura nas seis combinações de tecnologias de backend abordadas.

Pode-se observar que a busca foi significativamente mais eficiente no banco de dados MongoDB em comparação ao banco de dados PostgreSQL. Por outro lado, a criação de dados mostrou-se muito mais eficaz nos bancos de dados PostgreSQL, conforme demonstrado na Tabela 1. Algo que pode influenciar nos testes são as ferramentas utilizadas

Tecnologia	Biblioteca(s)	Tempo de execução (em segundos) Escrita	Leitura
Express	TypeORM + PostgreSQL	67.37	11.47
	Prisma + PostgreSQL	74.72	7.04
	Prisma + Mongo	221.04	2.22
Nest	TypeORM + PostgreSQL	66.31	8.44
	Prisma + PostgreSQL	75.00	7.58
	Prisma + Mongo	302.73	1.99

Tabela 1 – Comparação do tempo de resposta para diferentes tecnologias na criação e leitura de cards

para conexão de banco de dados, o código e a estrutura utilizada foi a mesma para todos os serviços, seguindo os princípios do SOLID.

Outro ponto relevante a ser considerado é a comparação entre as tecnologias Nest e Express. A escolha vai depender do nível de familiaridade técnica com cada ferramenta e das necessidades específicas do projeto. Express é um framework que permite ser mais flexível implementando variados padrões de projetos, já o Nest tende a seguir mais rigorosamente os princípios do SOLID, oferecendo uma arquitetura mais organizada e modular, facilitando a manutenção e escalabilidade de projetos.

Já a escolha entre as bibliotecas de TypeORM e Prisma de ORM, a decisão também pode variar de acordo com a familiaridade técnica e os requisitos do projeto. Pode-se visualizar na Tabela 1 que o Prisma apresenta melhor desempenho em operações de leitura, enquanto o TypeORM se destaca em operações de escrita.

4.4 Apresentação do projeto React

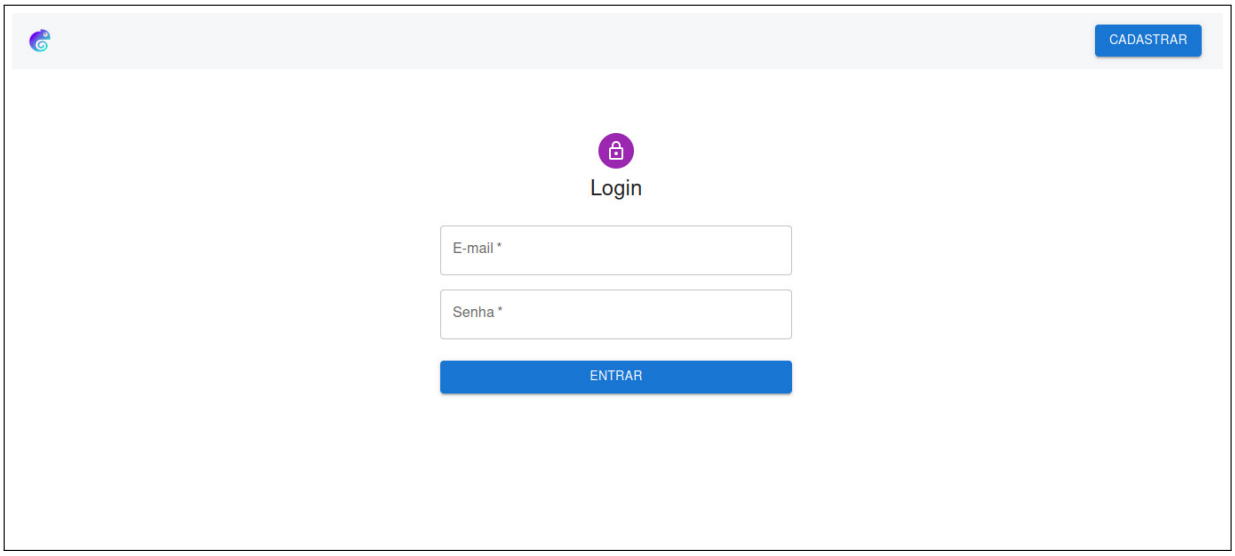
Nesse tópico será apresentado as telas criadas para o template Kanban desenvolvido com React e MaterialUI[4]. Importante ressaltar que esse template não está disponível online, somente com instalação local.

4.4.1 *Login* e cadastro de usuário

A primeira tela conforme mostra a Figura 19 é para realizar o login e conseguir acessar o dashboard da aplicação.

[4] <https://mui.com/material-ui/>

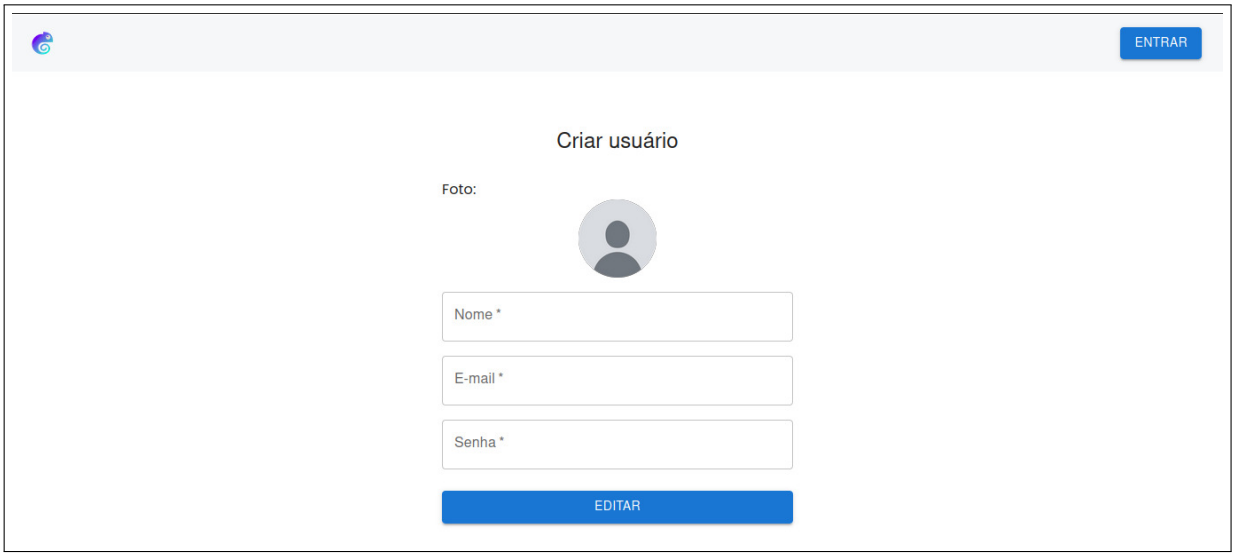
Figura 19 – Tela de *login*



Fonte: Produzida pela própria autora

Caso o usuário deseje acessar o sistema sem um cadastro, é necessário registrar-se clicando no botão "Cadastrar" localizado no cabeçalho, conforme indicado na Figura 19. A Figura 20 ilustra a tela de cadastro de usuário, no qual é possível fornecer nome, e-mail, senha e, opcionalmente, uma foto.

Figura 20 – Tela de cadastro de usuário

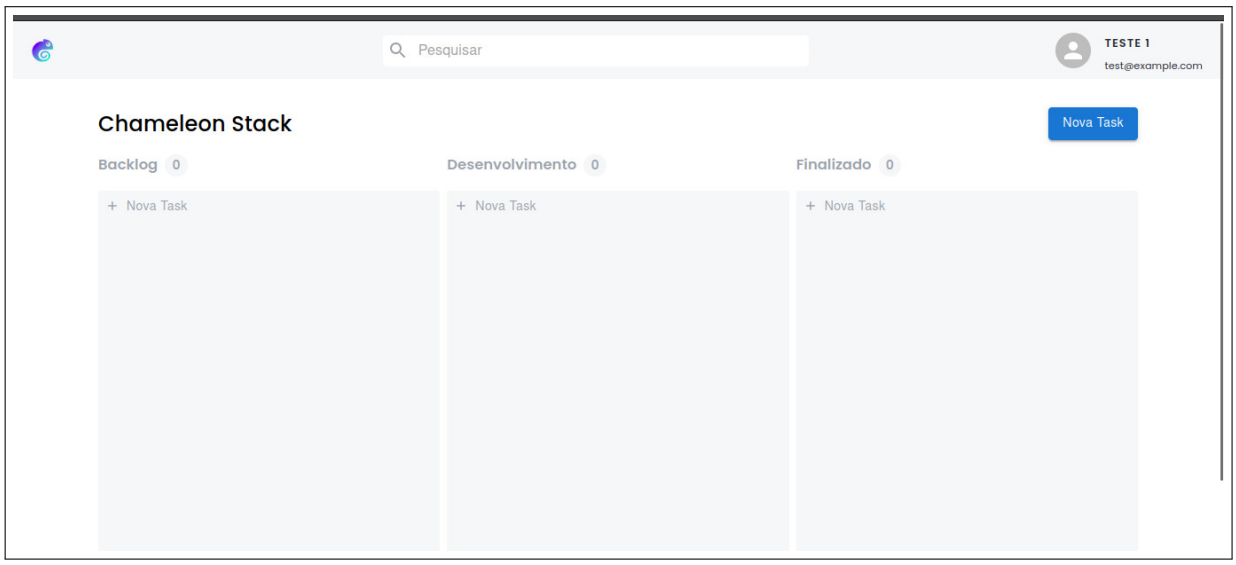


Fonte: Produzida pela própria autora

4.4.2 *Dashboard*

Após realizar o *login*, o usuário terá acesso ao sistema, representado na Figura 21. Esta figura mostra um quadro de gerenciamento de tarefas, no qual é possível organizar e controlar as atividades. As tarefas são distribuídas em colunas: "Backlog" para tarefas pendentes, "Em Desenvolvimento" para tarefas em progresso, e "Finalizadas" para as concluídas.

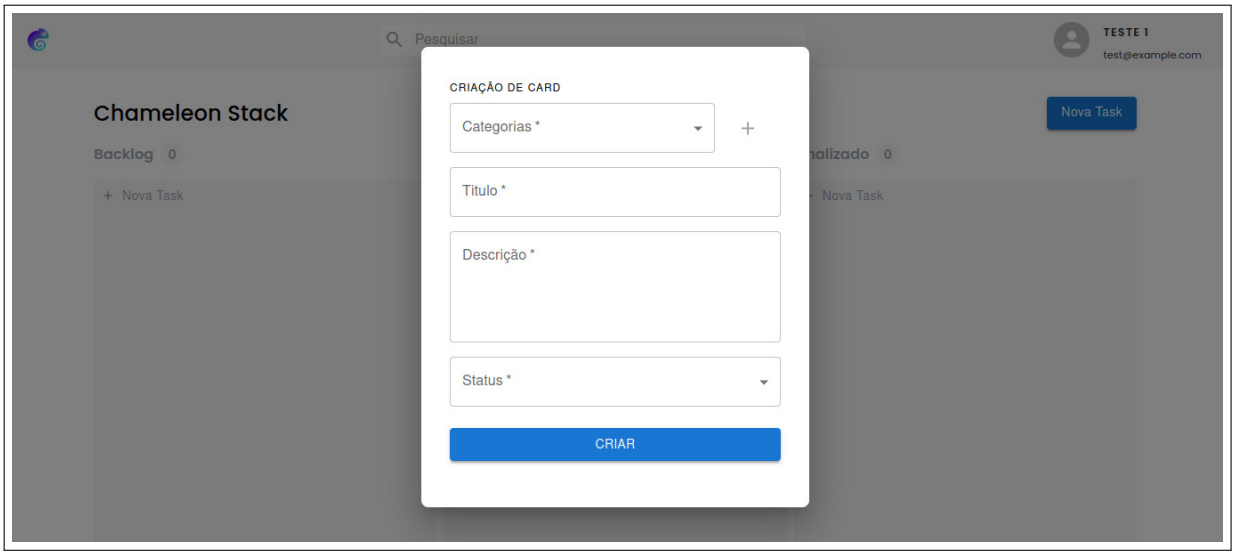
Figura 21 – Dashboard



Fonte: Produzida pela própria autora

Acima do quadro de tarefas, há um botão "NOVA TASK" que permite criar um novo *card*, no qual é possível adicionar um título, uma descrição e uma categoria para a tarefa conforme mostra a Figura 22.

Figura 22 – Tela para cadastrar card



Fonte: Produzida pela própria autora

Além de gerenciar as tarefas e categorias criadas, o sistema permite ao usuário sair ou atualizar suas informações, conforme ilustrado no cabeçalho da Figura 23. A figura também apresenta um campo de busca, no qual é possível localizar um card específico dentro do quadro de tarefas pelo seu título.

Figura 23 – Gestão de usuário e busca de cards

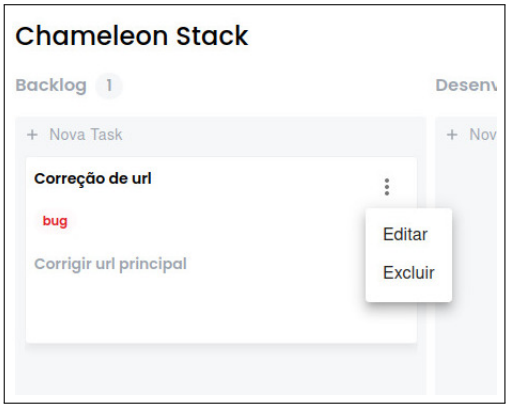
Fonte: Produzida pela própria autora

No botão "+" localizado ao lado da lista de categorias atribuídas ao cartão na Figura 22, também é possível cadastrar uma nova categoria. A Figura 24a mostra um *modal* responsável pelo registro de uma nova categoria, no qual é necessário informar o nome e a cor atribuída a ela.

Figura 24 – Gestão de card e categorias

(a) Tela para cadastrar categoria

(b) Gerenciar card



Fonte: Produzida pela própria autora

5 Considerações Finais

Com base nos projetos desenvolvidos e nos resultados obtidos, pode-se concluir que é crucial considerar o uso pretendido do projeto ao escolher o banco de dados mais eficiente. Por exemplo, no MongoDB, a inserção de dados é mais lenta, mas a busca é muito rápida, sendo ideal para serviços focados em consultas rápidas e que não exigem *queries* complexas. Por outro lado, o PostgreSQL é mais adequado para inserções de dados frequentes e *queries* complexas. Além disso, consultas ao PostgreSQL com o ORM Prisma foram cerca de 1 segundo mais rápidas.

Utilizando Node.js, foi possível implementar as seis tecnologias desejadas, juntamente com suas documentações e testes unitários para cada serviço. Os projetos também foram organizados em containers Docker, e para os projetos utilizando o MongoDB foi utilizado um arquivo docker-compose para configurar réplicas, detalhando o passo a passo de execução. Além das aplicações backend, foi desenvolvido um projeto *frontend* com React.js, conforme apresentado nos capítulos 3 e 4.

Durante o desenvolvimento, foi enfrentado desafios relacionados às diferenças na implementação de código entre os SGBD MongoDB e PostgreSQL. Por mais que tenha o auxílio do ORM essas diferenças acontece pela distinção entre bancos de dados relacionais e não relacionais, resultando em variações no modo de leitura e escrita dos dados.

Como sugestões de trabalhos futuros pode-se citar:

- Disponibilização de tecnologias em outras linguagens, como uma aplicação móvel em Kotlin;
- Exploração de resultados em mais bancos de dados e ORMs.
- Melhorar segurança dos projetos, implementando boa prática da segurança da informação, como *authorization* e *authentication*;
- Implementação de login social;
- Implementação de login via JSON Web Token (JWT);
- Implementação de configuração de ferramentas de qualidade do software como Grafana e Sonaqube;
- Implementar outras aplicações base, como lista de compras, *e-commerce* entre outras.

Referências

ALCANTARA, A. A. M.; SANT'ANNA, A. P. Medindo eficiência em desenvolvimento de sistemas. *Revista produção*, v. 11, n. 2, p. 89–101, 2002. Acesso em: 30 março 2024. Disponível em: <https://www.scielo.br/j/prod/a/XxR5ZqKBXRwspdRnYTPh9gH/?format=pdf&lang=pt>. Citado na página 17.

ASSE, R. *Guia Definitivo para Aprender Supabase*. 2024. Acesso em: 30 de maio de 2024. Disponível em: <https://www.semcodar.com.br/guia-definitivo-para-aprender-supabase/>. Citado na página 23.

AUGUSTO, J.; CARVALHO, C. Análise avaliativa acerca do uso do react para programação web front-end. In: *Encontro de Ensino, Pesquisa e Extensão*. [s.n.], 2023. Acesso em: 28 de maio de 2024. Disponível em: <http://enpe.ptc.iftm.edu.br/index.php/enpe/article/view/370>. Citado na página 23.

BUNNYSHELL. *Looking to deliver real developer productivity?* 2024. Acesso em: 14 janeiro 2024. Disponível em: <https://www.bunnyshell.com/>. Citado 2 vezes nas páginas 18 e 19.

COSTA, M. R. R. da; OLIVEIRA, C. S. de. Comparação entre linguagens de programação no mercado, ensino superior e pesquisas científicas na cidade de são paulo. *Sinergia*, 2020. Acesso em: 14 jan. 2024. Disponível em: <https://ojs.ifsp.edu.br/index.php/sinergia/article/view/1520>. Citado na página 13.

GONÇALVES, E. L. *A evolução dos softwares*. 2002. Acesso em: 14 janeiro 2024. Disponível em: <http://ric.cps.sp.gov.br/handle/123456789/2157>. Citado na página 13.

JEST. *Jest é um poderoso Framework de Testes em JavaScript com um foco na simplicidade*. 2024. Acesso em: 28 janeiro 2024. Disponível em: <https://jestjs.io/pt-BR/>. Citado 2 vezes nas páginas 8 e 34.

NUNES, G. N. *As vantagens do Node.js*. 2018. Acesso em: 27 de maio de 2024. Disponível em: <https://refaqi.faqi.edu.br/index.php/refaqi/article/view/96>. Citado na página 21.

OVERFLOW, S. *2023 Develop Survey*. 2023. Acesso em: 29 de março de 2024. Disponível em: <https://survey.stackoverflow.co/2023/#overview>. Citado 3 vezes nas páginas 13, 14 e 15.

OVERFLOW, S. *2023 Developer Survey*. 2023. Acesso em: 06 de junho de 2024. Disponível em: <https://survey.stackoverflow.co/2023/#section-most-popular-technologies-web-frameworks-and-technologies>. Citado 3 vezes nas páginas 21, 22 e 23.

PRISMA. *Prisma*. 2024. Acesso em: 02 de julho de 2024. Disponível em: <https://github.com/prisma/prisma/tree/4f39f9f5004f0e6df3ae84e473e0941241fd6ffc/docker>. Citado na página 35.

ROSA, D. *O que é exatamente o Node.js?* 2023. Acesso em: 06 de junho de 2024. Disponível em: <https://www.freecodecamp.org/portuguese/news/o-que-e-exatamente-o-node-js/>. Citado na página 21.

SABO, M. *NestJS*. 2020. Disponível em: <https://zir.nsk.hr/islandora/object/mathos:441>. Acesso em: 07 de junho de 2024. Citado na página 22.

SILVA, L. A. L. Edson Coutinho da. Framework scrum: Eficiência em projetos de software. *Dialnet*, 2016. Acesso em: 30 mar. 2024. Disponível em: <https://dialnet.unirioja.es/servlet/articulo?codigo=5632152>. Citado na página 17.

TV, C. F. *Pesquisa Salarial de Programadores Brasileiros 2024*. 2024. Acesso em: 25 de setembro de 2024. Disponível em: <https://pesquisa.codigofonte.com.br/2024>. Citado 3 vezes nas páginas 13, 14 e 21.