UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BEATRIZ DE ALMEIDA FERNANDES DE LIMA

PILAS: UM SISTEMA WEB PARA GERENCIAMENTO DO CONSUMO DE GULOSEIMAS EM UMA EMPRESA DE TECNOLOGIA

GUARAPUAVA

2023

BEATRIZ DE ALMEIDA FERNANDES DE LIMA

PILAS: UM SISTEMA WEB PARA GERENCIAMENTO DO CONSUMO DE GULOSEIMAS EM UMA EMPRESA DE TECNOLOGIA

Pilas: a web system to manage treats consumption at a technology company

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Andres Jessé Porfirio

Coorientadores: Prof. Henrique Fernando de Oliveira Rodrigues e Prof. Maurício Barfknecht

GUARAPUAVA

2023

BEATRIZ DE ALMEIDA FERNANDES DE LIMA

PILAS: UM SISTEMA WEB PARA GERENCIAMENTO DO CONSUMO DE GULOSEIMAS EM UMA EMPRESA DE TECNOLOGIA.

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnológo em Sistemas para Internet do Curso de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná (UTFPR).

Data da aprovação: 6/Julho/2023

Prof. Andres Jessé Porfirio
Doutor
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Guilherme da Costa Silva
Mestre
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Roni Fabio Banaszewski
Doutor
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

GUARAPUAVA
2023

Para Sarita, minha mãe e melhor amiga.

AGRADECIMENTOS

Esse trabalho é resultado não somente de esforço próprio, ele se deve também a todos que de alguma forma contribuíram na minha trajetória até aqui. Aos que não menciono nomes, tenham certeza que os carrego em meu coração com gratidão.

Começo agradecendo a minha mãe Sarita, mulher que me inspira, que me da forças e me entende como ninguém mais no mundo entenderia. Aquela que permaneceu ao meu lado nos momentos em que eu realmente achei que não conseguiria, passou noites em claro me fazendo companhia e se preocupou em sempre nos preparar uma refeição gostosa enquanto eu não tinha tempo. Não existem palavras que expressem minha gratidão.

Agradeço aos meus familiares e amigos próximos que entenderam minhas ausências e sempre me ofereceram carinho, conforto e palavras de incentivo. Saibam que sem vocês, nada disso seria possível.

Agradeço também ao meu orientador, Prof. Dr. Andres, que não mediu esforços para me ajudar no desenvolvimento desse projeto. Ele que é também um grande aliado na prática de conversas em inglês.

Agradeço a todos os meus colegas de trabalho, que seguraram as pontas quando eu precisei me ausentar e que me ofereceram carinho e apoio. Vocês são incríveis! E um agradecimento especial aos meus chefes e coorientadores Maurício Barfknecht e Henrique Rodrigues, que muito mais que isso, são meu mentores, líderes, amigos e grandes inspirações pessoais e profissionais. Obrigada por sempre me apoiarem e encorajarem.

Meu agradecimento também a UTFPR, toda sua equipe e todos os excelentes professores que tive o prazer de conhecer e conviver e os quais compartilharam comigo algo que não pode ser precificado, seu conhecimento.

A todos vocês, meu mais sincero obrigada!

Muitas das coisas mais importantes do mundo
foram conseguidas por pessoas que
continuaram tentando quando parecia não
haver mais nenhuma esperança de sucesso.
- Dale Carnegie

RESUMO

Cultura empresarial ou organizacional é o conjunto de valores, costumes e ideais inseridos no dia a dia de uma empresa com intuito de engajar seus colaboradores e proporcionar bem estar e qualidade de vida à equipe. Tendo isso em vista, uma empresa de tecnologia implantou um projeto denominado *Pilas*, onde seus colaboradores tem acesso a diversos itens alimentícios em troca de uma moeda fictícia. Seu controle se da atualmente através de planilhas que são manipuladas pelos próprios usuários, o que as torna suscetíveis a falhas, não havendo um sistema computacional capaz de suprir tais demandas, dadas suas características únicas e específicas. Posto isto, com a finalidade de auxiliar esse projeto, o presente trabalho busca desenvolver um sistema web com versão em PWA[1] que gerencia e automatiza os processos envolvidos, como cadastro de produtos para compra e as movimentações de aquisição dos mesmos, bem como gerenciamento de usuários com diferentes níveis de permissões.

Palavras-chave: cultura organizacional; empresas; tecnologia; programação para internet.

[1] Progressive Web App: aplicação híbrida entre as páginas da web regulares e um aplicativo móvel.

ABSTRACT

Organizational culture is the set of values, customs and ideals that are a everyday life part of a company in order to engage its employees and provide well-being and quality of life to the team. Thinking of this, a technology company implanted a project called *Pilas*, where its employees have access to several kinds of food in exchange for a fictitious coin. As there is no computational system capable to fulfill this project demands, given their unique and specific characteristics, its control currently takes place through spreadsheets that are manipulated by users themselves, which makes them likely to failures. With the aim of helping this project, the present work seeks to develop a web system with a version in PWA, which will be able to manage and automate the procedures of *Pilas*, such as the registration of products and their purchase transactions, as well as management of users with different levels of permissions.

Keywords: organizational culture; company; technology; internet programming.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE SIGLAS E ACRÔNIMOS

Siglas

UTFPR	Universidade Tecnológica Federal do Paraná
PWA	Progressive Web App
ERP	Enterprise Resource Planning
ORM	Object Relational Mapper
SQL	Structured Query Language
iOS	iPhone Operating System
PR	Pull Request
CDN	Content Delivery Network
IP	Internet Protocol
DER	Diagrama Entidade Relacionamento
ER	Entidade Relacionamento
UI	User Interface
UX	User Experience
MVP	Minimum Viable Product
TCC1	Trabalho de Conclusão de Curso 1
DB	Data Base
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
DNS	Domain Name System

Acrônimos

MoSCoW	Must, Should, Could and Wouldn’t
CRUD	Create, Read, Update and Delete

SUMÁRIO

1 INTRODUÇÃO

Cultura empresarial ou organizacional é o conjunto de valores, costumes e ideais inseridos no dia a dia de uma empresa. Sua importância está no fato de ser uma excelente forma de engajar os colaboradores e torná-los mais unidos e motivados em prol do sucesso da organização, além de proporcionar bem estar e qualidade de vida à equipe. Alguns exemplos de atividades, atitudes ou rituais que podem compor uma cultura organizacional são: um ambiente descontraído com música e liberdade para conversar, *happy hours*[1], comemorações de aniversários e conquistas da equipe, premiações e refeições gratuitas (MOTTA; VASCONCELOS, 2002). O bem estar de um colaborador em uma empresa é fundamental, primeiro porque incentiva a produtividade, e depois, porque ajuda a amenizar o estresse que muitas vezes algumas atividades do trabalho podem trazer (MASLACH; SCHAUFELI; LEITER, 2001).

Diante disso, uma empresa de tecnologia, que preza a construção de uma forte cultura organizacional, deu início a um controle interno de consumo de produtos, batizado de *Pilas*. São exemplos desses produtos: balinhas, paçocas, biscoitos, chocolates, frutas, iogurtes, entre outros. Para consumi-los, cada colaborador recebe, por mês, um determinado saldo de uma moeda fictícia chamada *Pila*.

O abastecimento do estoque e a distribuição da moeda, é responsabilidade de um membro da equipe, eleito mês a mês de forma democrática como “prefeito”. Atualmente, o controle de itens no estoque, de saldo dos consumidores e movimentações de compra, são feitos através de uma planilha compartilhada no Google Sheets[2]. O projeto *Pilas*, conta também com um agregado chamado de “Fome Zero”, através do qual, a equipe tem livre acesso a alimentos básicos para lanches, como café, leite, pão, manteiga, doce de leite, bolacha, açúcar, dentre outras opções.

Dessa forma, o objetivo com o *Pilas* é disponibilizar todo mês **guloseimas**[3] **adicionais**, e que normalmente uma empresa não provê gratuitamente. Contudo, se esse tipo de produto fosse de livre consumo, haveria um grande risco de todo o estoque ser consumido compulsivamente em poucos dias e de forma desproporcional, mais por alguns do que por outros. Adicionalmente, existe a preocupação com a saúde dos colaboradores, pois a ingestão exagerada desses alimentos ultraprocessados cotidianamente, pode ser prejudicial (BIELEMANN *et al.*, 2015).

O modelo atual em execução é falho, pois existem muitas brechas para inconsistências, visto que cada um anota seus gastos, e por engano ou mesmo má fé, pode acabar fazendo anotações incorretas, não havendo uma forma de fiscalizar isso. Por conseguinte, destaca-se a necessidade de implementação de uma solução computacional com maior poder de gerência

[1] Reunião ou comemoração informal, normalmente envolvendo colegas de trabalho ou estudo, realizada após expediente de trabalho com petiscos e bebidas.

[2] Serviço de planilhas online, website oficial: https://sheets.google.com/

[3] Gíria utilizada para se referir a gulas, doces.

em relação às atuais planilhas, promovendo controle e segurança nas transações de consumo de guloseimas da empresa.

Dito isso, o presente documento propõe o desenvolvimento de um sistema web com versão também em PWA, que fará toda essa gestão, hoje realizada através de planilhas. Por meio deste sistema, será possível o cadastro de usuários, a verificação de saldo e extrato de *Pilas* e a realização de compra dos itens cadastrados. Além de possuir uma área exclusiva para o prefeito e administrador do sistema, que farão a gestão dos produtos, saldos, usuários e permissões.

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver um sistema para gerenciar o consumo de guloseimas pelos colaboradores de uma empresa de tecnologia.

1.1.2 Objetivos específicos

- Desenvolver um sistema web que possibilite o cadastro de usuários com diferentes níveis de permissão;
- Permitir que o prefeito gerencie produtos;
- Permitir que o prefeito controle os saldos dos usuários;
- Permitir ao usuário a consulta de saldo e extrato de consumo de *Pila*;
- Possibilitar aos usuários realizarem movimentações de compra de itens disponíveis;
- Possuir compatibilidade com diversos tipos de sistemas operacionais, incluindo os de dispositivos móveis;
- Publicar a aplicação como PWA;
- Promover facilidade e agilidade na localização e compra de itens através de interface minimalista e intuitiva;
- Permitir que o sistema esteja disponível exclusivamente de forma online, evitando que o usuário precise realizar instalações e configurações.

1.2 Pilas

No primeiro semestre de 2022, foi iniciado na disciplina de Programação para Dispositivos Móveis, na UTFPR Campus Guarapuava, o desenvolvimento de um aplicativo com a ideia de suprir as necessidades apontadas no presente trabalho. Para desenvolvimento inicial desse aplicativo, foram utilizadas tecnologias como React Native, Node.js, Expo e Firebase, proporcionando a autenticação de usuários com diferentes permissões, cadastros de produtos e opção para compra dos mesmos utilizando o saldo disponível. A Figura 1 traz exemplos de telas do aplicativo que foi desenvolvido.

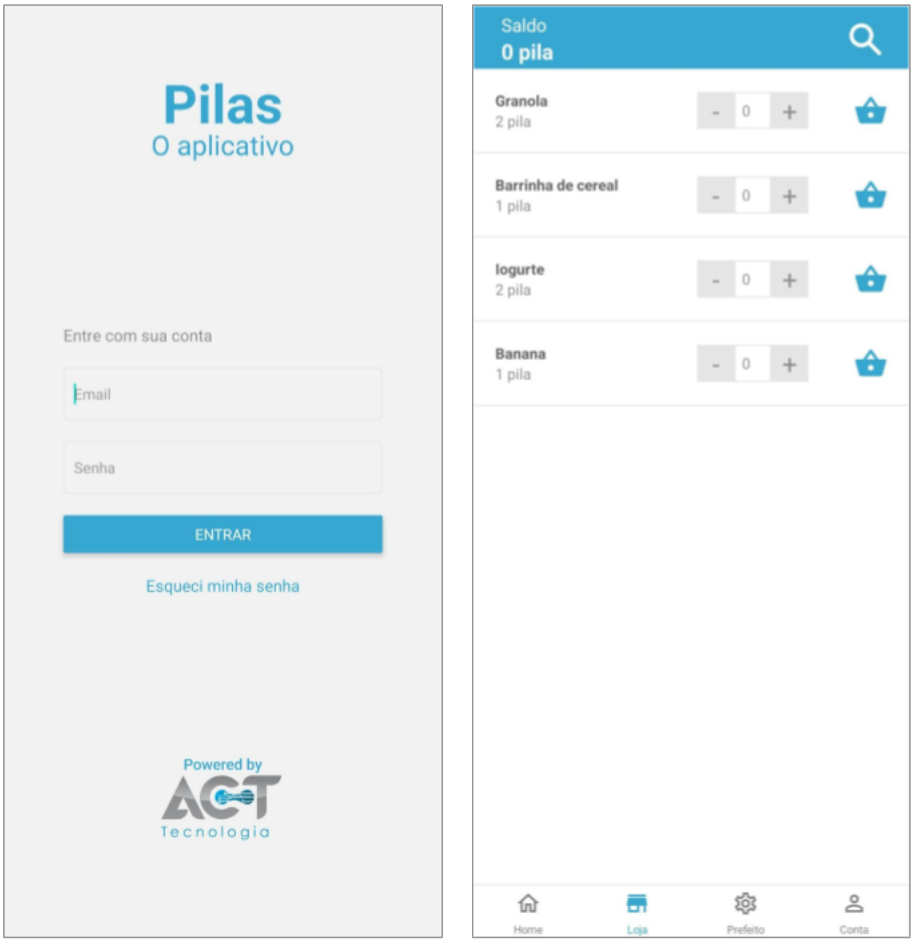


Figura 1 – Telas do aplicativo Pilas
Fonte: Autoria própria (2023).

Após o término da disciplina, foi considerada a possibilidade de dar continuidade ao aplicativo, tendo em vista o interesse em automatizar o processo do projeto Pilas, mas foram observadas algumas limitações. Este teria que ser implantado para utilização dos colaboradores da empresa, sendo disponibilizado nas lojas de aplicativos, tanto de sistemas Android, quanto iOS. Analisando-se os termos dessas lojas, observou-se que ambas as plataformas exigem o

pagamento de licenças e o custo para disponibilizá-lo para iOS[4] seria inviável em um primeiro momento.

Além disso, a disponibilização do sistema como um aplicativo móvel nativo nas lojas de Android e iOS não permite o teste instantâneo de novas funcionalidades e/ou correção de bugs, visto que cada plataforma possui um fluxo de *deploy*[5] próprio, que muitas vezes requer um período de espera até a aprovação de novas versões. Diante disso, cada atualização no sistema pode levar desde algumas horas, até dias para que todos os usuários recebam a notificação da nova versão e atualizem em seus dispositivos. Em contrapartida, uma aplicação web não requer essa espera, dado que feito o *deploy*, todos os usuários ficam imediatamente aptos a utilizarem o sistema atualizado.

Tendo em vista essas limitações, descartou-se a continuidade do aplicativo, dando início à ideia de desenvolvimento de um sistema web que inclui todos os tipos de usuários no que diz respeito aos seus dispositivos e fornece atualizações de maneira transparente.

[4] Apple Developer Program: https://developer.apple.com/support/compare-memberships/

[5] Do inglês "implantar", é um termo utilizado em programação em seu sentido literal, para uma aplicação que é implantada.

2 SISTEMAS SIMILARES

Dada a necessidade de um sistema para o controle supracitado, foram identificadas e analisadas algumas soluções existentes a fim de verificar se estas satisfazem os objetivos almejados.

2.1 Nex

O Nex[1], é um sistema para gestão de pequenos comércios, que permite o cadastro de produtos, fornecedores, histórico de movimentação, conferência de estoque, entre outros recursos relacionados. Embora possua funções semelhantes as buscadas, o método utilizado é pensado exclusivamente para comércios, não se encaixando com o *Pilas*, onde cada colaborador poderá agir autonomamente, comprando seus próprios itens dentro do sistema. O sistema Nex como um todo é complexo para o propósito desejado e contém vários processos desnecessários para o *Pilas*, além de não possuir uma gestão de usuários com saldo. A Figura 2 traz um exemplo de tela do sistema Nex.

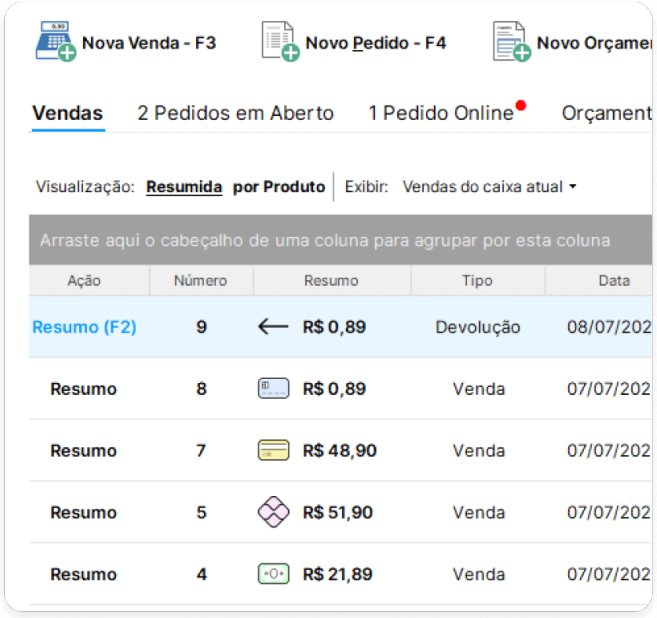


Figura 2 – Tela de resumo de vendas do sistema Nex
Fonte: Nex (2022).

[1] https://www.nextar.com.br

2.2 Contestoque

O Contestoque[2] foi desenvolvido de forma exclusiva para controlar estoque. Ele permite o cadastro de produtos com data de vencimento e código de barras, contando com um leitor que utiliza a câmera do celular e permitindo também o controle de quantidade dos itens, conforme mostra a Figura 3. Esse aplicativo atende principalmente aqueles que possuem estoque de alimentos, dos quais precisam ter completo controle, como comércios, restaurantes, ou cozinhas de instituições em geral. O sistema pode também atender a necessidade de organizações que possuem outros tipos de estoque, como de ferramentas e equipamentos.

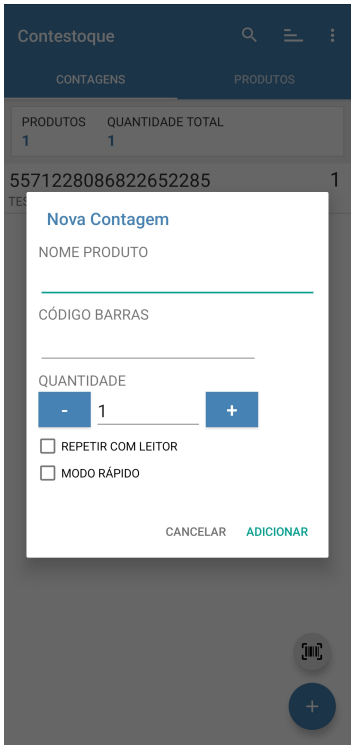


Figura 3 – Tela de registro de um produto no aplicativo Contestoque
Fonte: Contestoque (2022).

2.3 Tiny ERP

O Tiny[3] é um sistema web do tipo ERP (Enterprise Resource Planning) que une diversas funcionalidades úteis para as operações de uma empresa. Através dele é possível acesso a alguns recursos como de vendas (Figura 4), cadastro e controle de produtos, controle de estoque e emissão de boletos e notas fiscais, além de possibilitar integração com diversos marketplaces para unificar as vendas da empresa. O sistema é pago e trabalha no formato modular, permitindo uma contratação personalizada com apenas as funções desejadas.

[2] http://www.contestoque.com.br/
[3] https://www.tiny.com.br/

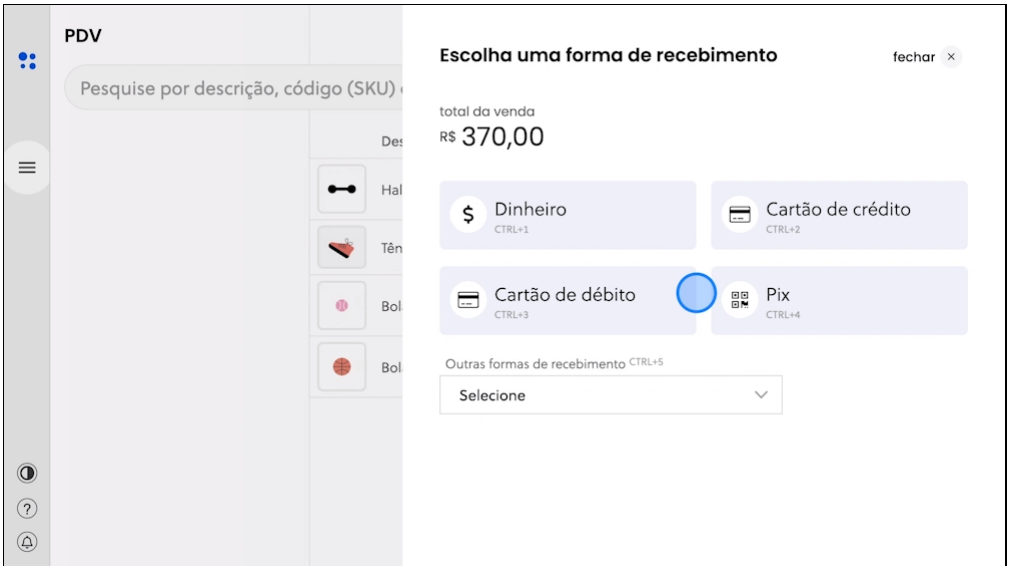


Figura 4 – Tela de movimentação de venda no sistema Tiny
Fonte: Tiny (2022).

Os sistemas anteriormente descritos possuem funções semelhantes as buscadas para o *Pilas*, como cadastro de produtos e histórico de transações, presentes no Nex e no Tiny e controle de estoque que está presente nos 3 sistemas. No entanto, em nenhum destes é possível fazer gestão de usuários com diferentes níveis de permissões, onde cada um possa fazer login e possua um saldo para realizar suas próprias movimentações de compras de produtos, acompanhando também seu extrato individual. O que pretende-se com este trabalho, é justamente unir em um só projeto funções como as que foram citadas, incluindo adicionalmente os requisitos mais específicos do *Pilas*, os quais não são satisfeitos por nenhum outro sistema encontrado, resultando em uma plataforma completa.

3 METODOLOGIA

Este capítulo é dividido em duas seções, 3.1 Materiais e 3.2 Métodos. Ele tem ênfase em discriminar as ferramentas utilizadas no presente projeto, bem como a abordagem utilizada para organização e execução do mesmo.

3.1 Materiais

Nesta seção estão apontadas e descritas as ferramentas e tecnologias utilizadas no desenvolvimento do sistema em questão.

3.1.1 TypeScript

A **linguagem de programação** utilizada é o TypeScript[1], um superconjunto de JavaScript que o agrega novos recursos, como a **tipagem estática** que consiste em definir regras para a forma em que cada tipo de valor pode ser utilizado. Sendo assim, enquanto para o JavaScript variáveis *string* ou numéricas podem ser tratadas da mesma forma, para o TypeScript o tipo dessas variáveis precisa ser devidamente declarado para que o código seja compilado corretamente (MICROSOFT, 2022).

A principal vantagem em utilizar uma linguagem com tipagem estática é a redução de erros da aplicação em tempo de execução, o que auxilia e facilita o trabalho do programador (MEIJER; DRAYTON, 2004).

Apesar de aumentar um pouco a complexidade de desenvolvimento do sistema, optar pelo uso do TypeScript em detrimento de JavaScript proporciona uma estrutura mais escalável, o que é interessante para este projeto, já que poderão ser percebidas novas necessidades conforme o crescimento do *Pilas* e o uso da aplicação.

3.1.2 Node.js

Para o *backend*, o sistema utiliza Node.js[2], solução *open source*, que se trata justamente de um ambiente para execução de JavaScript em *server side* (lado do servidor). Isso significa que ele permite a execução de código JavaScript sem necessidade de um navegador (NODE.JS, 2023). A vantagem do uso de Node.js nessa aplicação está no fato de utilizar a mesma linguagem de programação do *client side* (lado do cliente), o que facilita na aprendizagem das tecnologias para o desenvolvimento.

[1] https://www.typescriptlang.org/

[2] https://nodejs.org/

3.1.3 PostgreSQL e Prisma ORM

Para **base de dados** optou-se pelo PostgreSQL[3], uma ferramenta de código aberto do tipo **objeto relacional** que possui inúmeros recursos avançados para armazenar e escalar os dados de forma segura (POSTGRESQL, 2022), sendo ideal para aplicações de pequeno, médio ou grande porte e estando entre os bancos mais utilizados quando há busca por maior estabilidade (MILANI, 2008).

Um banco de dados do tipo objeto relacional une características de bancos relacionais e orientados a objetos. Isso significa que proporciona maior escalabilidade juntamente com suporte para tipos de dados complexos (DEVARAKONDA, 2001).

Além dos pontos supracitados, o fato de a empresa onde o sistema será implementado já utilizar PostgreSQL para suas bases de dados, foi um ponto de grande influência na escolha do mesmo para execução do projeto, favorecendo a manutenção do sistema após sua implantação.

O Prisma[4] trata-se de uma **ferramenta ORM** (Object Relational Mapper) também de código aberto. Seu objetivo é facilitar para o desenvolvedor, a comunicação entre os objetos TypeScript e os dados de banco que os representam, através de métodos que dispensam a necessidade de codificação de elaboradas consultas em SQL. Outra grande vantagem está no fato de que o Prima conta com um sistema de migração para o banco (PRISMA, 2022).

3.1.4 Docker Engine e Docker Compose

Docker Engine[5] é uma tecnologia *open source* a qual possibilita implementar aplicações através de contêineres (DOCKERINC, 2023b), os quais por sua vez se tratam de uma forma de virtualização. Diferentemente de máquinas virtuais que fazem a virtualização a nível de hardware, os contêineres compartilham o mesmo *kernel* do sistema operacional (FELTER *et al.*, 2015), o que os torna mais leves e gera uma economia de uso de recursos (PAHL *et al.*, 2017).

Enquanto isso, Docker Compose[6] nada mais é do que uma ferramenta que permite rodar múltiplos contêineres através do uso de um arquivo do tipo YAML (DOCKERINC, 2023a).

Para o presente sistema, o banco de dados é executado em um *container* do Docker, com o objetivo de diminuir a probabilidade de erros e facilitar a implementação em diferentes servidores, já que não exige configurações complexas. Essa condição se aplica tanto para o ambiente de desenvolvimento, quanto o de produção.

[3] https://www.postgresql.org/
[4] https://www.prisma.io/
[5] https://docs.docker.com/engine/
[6] https://docs.docker.com/compose/

3.1.5 React.js e Next.js

React[7] é uma **biblioteca JavaScript** de código aberto que foi desenvolvida por engenheiros do Facebook em 2013, a fim de resolver alguns desafios relacionados a interfaces de usuário que mudam constantemente (GACKENHEIMER, 2015). Sendo assim, seu objetivo é a criação de interfaces interativas de usuário (META, 2022).

Dentre as principais vantagens que o uso dessa ferramenta oferece, estão a experiência eficiente de usuário e a facilidade de escrita e reutilização de componentes (FEDOSEJEV, 2015). Mas apesar de ter o intuito de auxiliar no desenvolvimento, React é apenas uma biblioteca, ou seja, ela oferece recursos úteis para construção de interfaces, mas fica a critério do desenvolvedor a forma de utilizá-los, exigindo certo esforço para desenvolver uma aplicação do zero.

Diante disso, com o intuito de facilitar ainda mais o desenvolvimento do *Pilas*, utiliza-se o Next.js[8], um **framework de React** que provê diversos recursos prontos, como de roteamento, busca de dados, integração com outros serviços, infraestrutura, performance e escalabilidade (VERCEL, 2022). Uma das principais vantagens da utilização do Next.js neste projeto é a sua capacidade de execução de código no lado servidor, eliminando a necessidade de um sistema auxiliar (como uma API externa) para fornecimento de recursos à interface React.js, que é nativamente executada no lado cliente. É utilizado também o NextAuth.js[9], projeto *open source* destinado para automatizar a configuração de autenticação no Next.js, entende-se por autenticação, *login* e *logout* no sistema.

3.1.6 Mantine UI

Trata-se de uma biblioteca de componentes React muito completa que conta com diversos componentes personalizáveis. Além de ser open source e possuir uma comunidade bem ativa, Mantine[10] proporciona uma interface moderna, amigável e limpa, com uma documentação robusta que facilita sua aplicação nos mais diversos formatos de sistemas web e mobile.

3.1.7 PWA

Um PWA (Progressive Web App) trata-se de uma **aplicação híbrida** que proporciona uma experiência semelhante ao uso de um aplicativo nativo, mas que na realidade é executada no navegador do dispositivo móvel. Algumas das vantagem de sua utilização são a possibilidade

[7] https://pt-br.reactjs.org/
[8] https://nextjs.org/
[9] https://next-auth.js.org/
[10] https://mantine.dev/

de ocultar a barra de endereço e demais botões do navegador e o fato de não ocupar espaço no armazenamento do dispositivo, já que não exige instalação.

Apesar de ser uma forma atrativa de disponibilizar aplicações, o PWA pode fornecer limitações quando se trata do acesso à recursos nativos do dispositivo (como restrições de espaço de armazenamento e ausência de notificações push no iOS). No entanto, embora existam desvantagens em restringir o acesso completo de alguns recursos dos dispositivos móveis, o fato de ser compatível com qualquer sistema operacional, acabou encorajando a escolha dessa tecnologia para desenvolvimento do presente sistema. Assim proporcionando um meio uniforme para a disponibilização da aplicação a todos os colaboradores da empresa, independente do tipo de dispositivo e sistema operacional utilizado, englobando desde celulares e tablets até computadores de mesa (ADETUNJI *et al.*, 2020).

3.1.8 Git e GitHub

O Git é um sistema de controle de versões utilizado para versionamento de códigos de programação, possibilitando manter um histórico de todas as alterações realizadas, bem como fácil recuperação das mesmas. O GitHub por sua vez, trata-se de uma plataforma online onde são hospedados os repositórios do Git. Essas duas ferramentas formam juntas um grande aliado em projetos desenvolvidos em time, pois permitem que vários desenvolvedores trabalhem simultaneamente em um mesmo sistema sem gerar conflitos (BLISCHAK; DAVENPORT; WILSON, 2016).

Uma das grandes motivações para utilização do Git e GitHub no desenvolvimento deste projeto, é o fato de que com elas é possível trabalhar com diferentes *branchs* (ramificações) de um mesmo projeto, onde cada uma pode ser direcionada para o desenvolvimento de uma funcionalidade específica. Além disso o desenvolvedor pode criar *PRs* (Pull Requests) no GitHub para que suas branchs sejam revisadas e aprovadas pelo restante da equipe, ou por um responsável (neste caso, o professor orientador), antes da mesma ser mesclada a versão oficial do código.

3.1.9 Cloudflare

Cloudflare[11] é atualmente um pacote de soluções voltado para sites e serviços da internet, que oferece segurança e ótimo desempenho em termos de velocidade de acesso, por se tratar também de um CDN (Content Delivery Network) (CLOUDFLARE, 2023a).

Servidores CDN são computadores espalhados por toda a rede mundial com intuito de armazenar conteúdos de forma a torná-los mais próximos possível de quem os requisita, reduzindo a latência e aprimorando o tempo de carregamento de páginas (CLOUDFLARE, 2023b).

[11] https://www.cloudflare.com

Além disso, através da Cloudflare é possível fazer a gestão do seu domínio, criando subdomínios e apontando para os endereços desejados. Nesse sentido, a Cloudflare também possibilita publicar com segurança aplicações de uma rede interna através de Túneis (no inglês *Tunnel*). Ou seja, com o *Cloudflare Tunnel* não é necessário possuir um endereço de IP público para acesso a uma aplicação.

Tendo em vista que a empresa onde o sistema Pilas será implantado utiliza Cloudflare para gestão de seus domínios, foi identificada a possibilidade de configuração de um *Tunnel* para sua publicação com um subdomínio próprio.

3.2 Métodos

No decorrer do desenvolvimento deste projeto, foram planejadas e executadas algumas fases, as quais seguem nesta seção.

3.2.1 Levantamento de requisitos

O primeiro passo foi o levantamento dos requisitos, sendo esses funcionais e não funcionais, onde os requisitos funcionais foram classificados através do método Must, Should, Could e Wouldn't (MoSCoW) (KUHN, 2009), conforme suas prioridades. O nome MoSCoW é um acrônimo das palavras em inglês **M**ust, **S**hould, **C**ould e **W**ouldn't, onde as letras "o", foram adicionadas apenas para tornar o acrônimo mais pronunciável. Este método traz quatro tipos de classificações, sendo elas, requisitos que o projeto:

- Precisa ter (Must have): aqueles requisitos que são essenciais e obrigatórios para o funcionamento do projeto;
- Deveria ter (Should have): requisitos importantes, mas não essenciais;
- Poderia ter (Could have): são os requisitos desejáveis, mas não vitais, eles podem ou não ser atendidos, conforme disponibilidade de tempo e recursos;
- Não teria (Wouldn't have): requisitos que estão fora do atual escopo.

Para chegar nos requisitos do sistema, observou-se a dinâmica do projeto Pilas dentro da empresa e como é a relação das pessoas com o mesmo. Ademais, buscou-se ter conversas com colaboradores para entender melhor suas necessidades e coletar suas ideias. Isso posto, houve uma preocupação em analisar os seguintes questionamentos:

- Quais as maiores dificuldades dos colaboradores que precisam comprar itens;
- Quais as maiores dificuldades do colaborador no papel de prefeito do mês;

- Quais as principais ações práticas requiridas pelos sujeitos envolvidos no projeto e como isso pode ser melhorado e automatizado;
- De que forma o dia a dia dos colaboradores poderia melhorar no que diz respeito aos Pilas.

A partir de um compilado de informações coletadas, definiram-se quais os requisitos para desenvolver o sistema mais adequado, que atenda as demandas existentes sem complicar ou burocratizar ainda mais todo o processo.

3.2.2 Planejamento do Banco de Dados

No planejamento do banco de dados, foi utilizado o Diagrama Entidade Relacionamento (DER ou modelo ER) ou no inglês Entity-Relationship Diagram (ERD). Este tipo de diagrama é comumente utilizado para representar de forma abstrata bancos de dados relacionais. O DER é composto por entidade, atributo e relacionamento. A **entidade** representa algo no sistema, como um objeto ou uma pessoa, por exemplo, que por sua vez possui **atributos** próprios, como nome e endereço. Estas entidades tem entre si **relacionamentos**, os quais mostram a quantidade de dados que flui entre uma entidade e outra, esse conceito é definido por **cardinalidade**. Existem três tipos de relacionamentos cardinais, relacionamento um-para-um, um-para-muitos e muitos-para-muitos. No diagrama, a cardinalidade é representada pelos símbolos ilustrados na Figura 5 (SONG; EVANS; PARK, 1995).

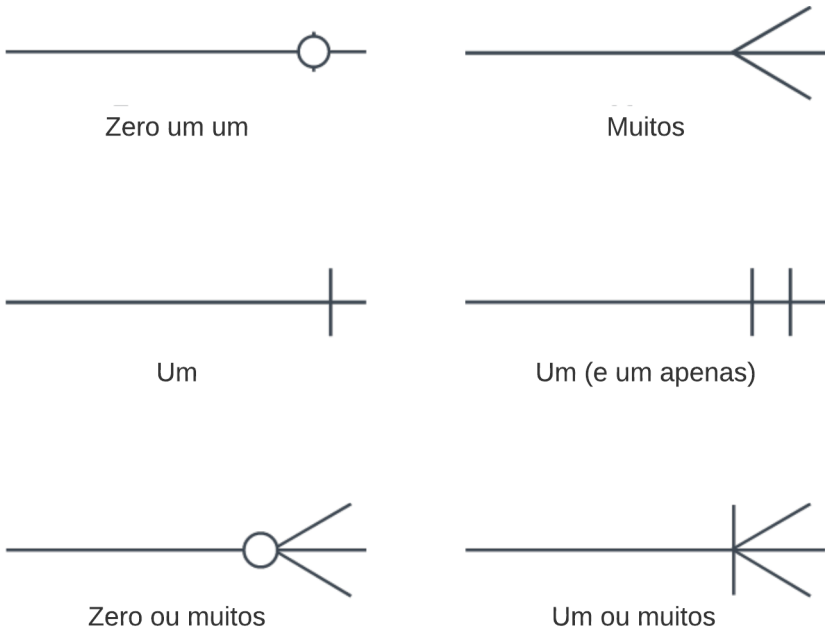


Figura 5 – Representações de cardinalidades
Fonte: Autoria própria (2023).

Para diagramar o modelo ER, utilizou-se a ferramenta online LucidChart[12] a qual possibilita que mais de uma pessoa possam fazer edições ao mesmo tempo, o que foi útil no trabalho em conjunto com o orientador.

3.2.3 Construção do Design System

O Design System consiste na definição de padrões de layout (tal como paleta de cores) que se esperam ser seguidos no desenvolvimento de um sistema (MACDONALD, 2019), constituindo assim uma identidade visual. Esse tipo de recurso também se faz útil para o desenvolvimento *front-end* em momentos em que o sistema precisa ser escalado. Para criação do Design System dos Pilas foi utilizado o Figma[13], ferramenta que tem sido amplamente utilizada por UI/UX (User Interface / User Experience) designers (STAIANO, 2022).

3.2.4 Fluxo de desenvolvimento

Uma vez que todas as definições inicias foram feitas, realizou-se uma análise em cima dos requisitos levantados e partir disto foram criadas as *features*[14] iniciais do sistema, logo após, aluna e orientador, fizeram a definição dos prazos. Foram definidos prazos precisos para os passos iniciais do projeto, enquanto para o restante das tarefas, foram definidos prazos estimados, sendo que esses foram sendo melhor calculados e ajustados conforme a produtividade percebida ao longo do processo.

Para organizar a codificação, optou-se por utilizar o modelo **Feature Branch Workflow**, através da ferramenta Git. Este modelo consiste em um desenvolvimento guiado pelas *features* que seguem sempre um mesmo fluxo, o qual funciona conforme ilustrado no fluxograma da Figura 6 e explicado logo a seguir.

O fluxo tem início na escolha da *feature* a ser desenvolvida. É neste momento que os prazos definidos podem ser revisados caso haja necessidade. Em seguida é criada uma nova *branch* local para que se inicie a escrita dos códigos de programação, realizando novos *commits* a cada etapa até que a mesma seja concluída. Essa divisão em vários *commits*, auxilia para que haja um versionamento do código, facilitando que alterações sejam revertidas se necessário. Uma vez que a tarefa tenha sido desenvolvida, é feito o *push* dessas alterações para a *branch* remota e através do GitHub é criada uma *Pull Request* (PR). Neste ponto, é julgado se há necessidade de revisão dos códigos por parte do orientador, caso positivo o mesmo faz os apontamentos necessários e se houverem correções ou alterações a serem realizadas, estas são cumpridas e enviadas para a *branch* remota novamente para aguardarem nova revisão.

[12] https://www.lucidchart.com/

[13] https://www.figma.com/

[14] Termo utilizado para se referir a funcionalidades ou recursos a serem desenvolvidos para um sistema computacional, por exemplo.

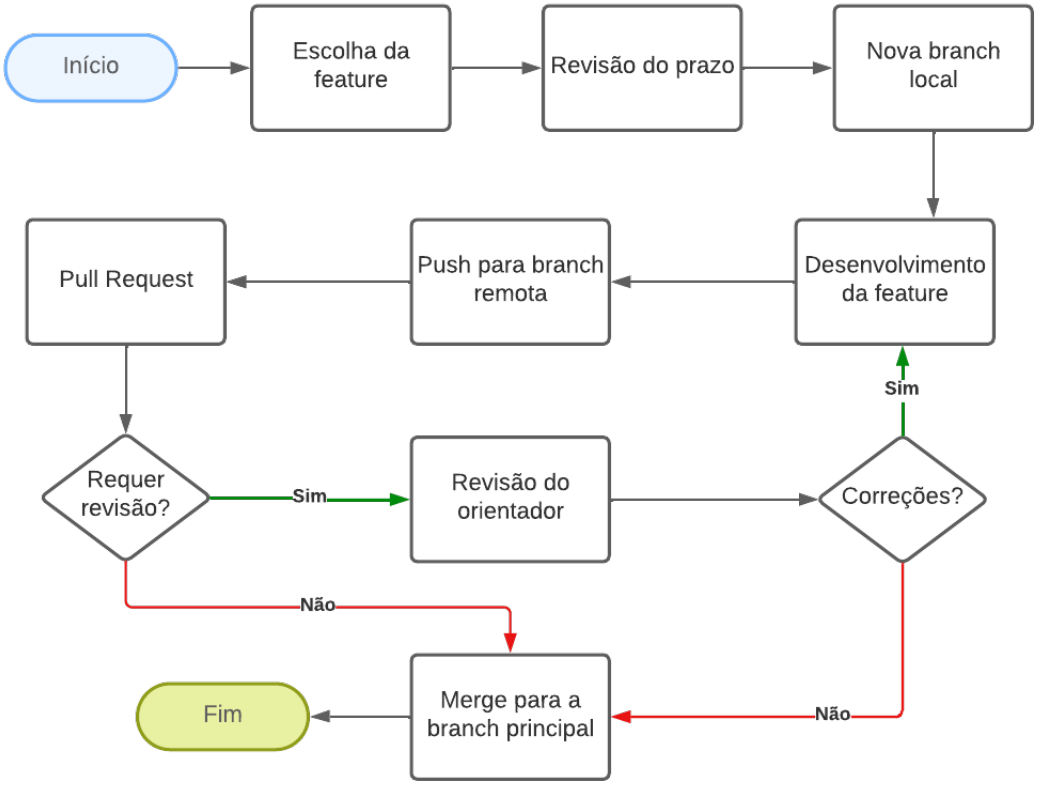


Figura 6 – Fluxograma de desenvolvimento da feature
Fonte: Autoria própria (2023).

Caso contrário, a etapa da revisão é pulada e é feito o *merge* para a *branch* principal finalizando assim, o fluxo da *feature*.

Vale ressaltar que todo o desenvolvimento *frontend* foi pensado e planejado utilizando a abordagem mobile-first (ROTH, 2019) visando a publicação da aplicação como PWA e um uso amigável da mesma em dispositivos móveis.

3.2.5 Organização do desenvolvimento

Na organização das *features*, utilizou-se a ferramenta *Projects* do GitHub. O *Projects* trata-se de um quadro adaptável de tarefas que permite listá-las em forma de *cards*, os quais podem ser vinculados com *issues* e *pull requestes* de um repositório (GITHUB, 2023). Dessa forma, as tarefas ficaram organizadas no quadro conforme seu estado: *To Do* (A fazer), *In Progess* (Em progresso), *Review* (Revisão) e *Done* (Feito). A Figura 7 exibe o quadro de tarefas do projeto Pilas no estado em que ele se encontra no momento em que este documento está sendo escrito.

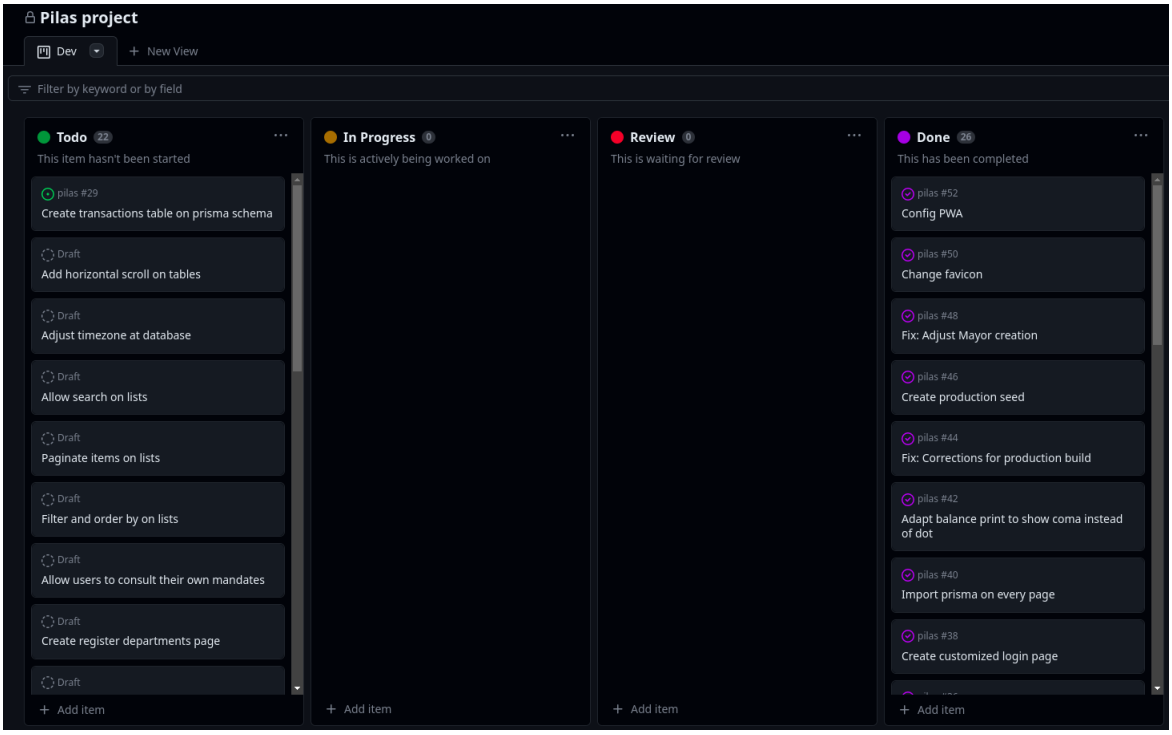


Figura 7 – Quadro de tarefas GitHub Projects
Fonte: Autoria própria (2023).

3.2.6 Implantação

A partir do momento em que as principais *features* foram desenvolvidas, obtendo-se um Produto Mínimo Viável – em inglês *Minimum Viable Product* ou MVP (LENARDUZZI; TAIBI, 2016) – , o sistema foi implantado na empresa em caráter de testes, para que os colaboradores pudessem dar suas opiniões e queixas a respeito da usabilidade e possíveis conflitos que pudessem surgir.

4 RESULTADOS

Este capítulo tem seu foco em expor os resultados obtidos a partir da execução da metodologia descrita no no **Capítulo 3**, **seção 3.2**.

4.1 Escopo do sistema

O presente sistema deve fazer a gestão do projeto Pilas, desde a parte de usuários até os produtos consumidos. Os colaboradores da empresa deverão possuir acesso ao sistema web com possibilidade de uso do PWA instalado em seus dispositivos, como computadores desktop e celulares. O sistema deve ser responsivo para uso em dispositivos móveis.

Deverá ser possível atribuir papel de administrador e prefeito para qualquer usuário, sendo que com o papel de administrador será possível criar usuários e atribuir-lhes papéis. Enquanto com o papel de prefeito será possível cadastrar os produtos disponíveis para compra e atribuir saldo em Pila para qualquer usuário.

O sistema não deve permitir que mais de um usuário possua papel de prefeito simultaneamente e deverá manter um histórico de todos os mandatos de prefeitos que sejam cadastrados, dessa forma usuários com papel de administrador devem ter acesso a esse histórico de mandatos.

Os usuários deverão ter a possibilidade de fazer *login* e *logout* no sistema. E além de visualizar quem é o atual prefeito, eles também poderão consultar os dados de sua conta e trocar sua própria senha.

O sistema deve permitir que qualquer usuário, independente de seu papel, possa comprar os itens disponíveis caso possua saldo suficiente em Pila. O usuário também deverá ter possibilidade de consultar seu histórico de transações, as quais se definem pelas movimentações de compras de itens dos usuários, contendo os detalhes dessas compras, tais como data da transação, produto comprado, valor do produto, quantidade e total da compra.

A compra de produtos se dará pela seguinte dinâmica: o usuário listará os produtos disponíveis para compra e ao clicar sobre o produto desejado poderá indicar a quantidade que deseja e por fim finalizar a compra com somente um clique. O sistema deve permitir apenas uma variedade de produto por compra, não disponibilizando um carrinho de compras.

4.2 Modelagem do sistema

4.2.1 Requisitos Funcionais e Não Funcionais

Com base nos estudos e na metodologia apresentados na **subseção 3.2.1**, obtiveram-se os resultados apresentados na **Tabela 1** para os Requisitos Não Funcionais e na **Tabela 2** para os Requisitos Funcionais.

Requisito	Descrição
RNF01	Possuir compatibilidade com diversos tipos de sistemas operacionais
RNF02	Garantir que o sistema possa ser acessado em plataformas móveis, como Android e iOS, com instalação de um atalho pro sistema e sem gerar custos excessivos à empresa
RNF03	Promover facilidade e agilidade na localização e compra de itens através de interface minimalista e intuitiva
RNF04	Permitir que o sistema esteja disponível exclusivamente de forma online, evitando que o usuário precise realizar instalações e configurações

Tabela 1 – Requisitos não funcionais do sistema.
Fonte: Autoria própria (2023).

4.2.2 Banco de Dados

Para projetar o Banco de Dados, utilizou-se o modelo de Diagrama Entidade Relacionamento, conforme explicado na **subseção 3.2.2**, na qual é possível também encontrar a definição para as cardinalidades utilizadas. O resultado do DER pode ser observado na **Figura 8**.

4.2.3 Design System

Ainda em fase de projeto do sistema, na disciplina de TCC1, definiu-se um Design System contendo as cores do tema e modelos para os componentes, este pode ser visualizado na **Figura 9**.

No entanto, a medida que o sistema começou a ser desenvolvido, foi identificada a possibilidade de basear seu layout no site já existente da respectiva empresa, visto que se trata da atual identidade visual da mesma. A **Figura 10** traz uma captura de tela do site.

Dessa forma um novo Design System foi elaborado, observa-se-o na **Figura 11**. É possível notar que as paletas de cores do Design System e do site são bastante semelhantes e

Requisito	Descrição	Prioridade
RF01	Possuir um usuário nativo administrador	Must
RF02	Permitir que um usuário faça logon	Must
RF03	Permitir que um usuário logado faça logoff	Must
RF04	Permitir que o administrador faça cadastro de usuários	Must
RF05	Permitir que o administrador edite usuários cadastrados	Must
RF06	Permitir que que o administrador liste os usuários cadastrados	Must
RF07	Disponibilizar função para atribuir o papel de prefeito à um usuário cadastrado	Must
RF08	Disponibilizar função para remover o papel de prefeito de um usuário cadastrado	Must
RF09	Disponibilizar função para desativar um usuário ativo	Must
RF10	Disponibilizar função para ativar um usuário desativado	Must
RF11	Permitir que o prefeito faça cadastro de produtos	Must
RF12	Permitir que o prefeito edite produtos cadastrados	Must
RF13	Permitir que o prefeito desative produtos cadastrados	Must
RF14	Permitir que um usuário liste os produtos disponíveis	Must
RF15	Permitir que um usuário compre produtos	Must
RF16	Permitir que um usuário verifique seu saldo	Must
RF17	Permitir que um usuário edite detalhes de sua própria conta	Should
RF18	Permitir que um usuário pesquise por produtos específicos	Should
RF19	Permitir que o prefeito faça cadastro de categorias de produtos	Should
RF20	Permitir que o prefeito edite categorias de produtos cadastradas	Should
RF21	Permitir que o prefeito remova categorias de produtos, desde que sem produto vinculado	Should
RF22	Permitir que um usuário visualize seu extrato de movimentações	Should
RF23	Permitir que o prefeito gere relatórios	Could
RF24	Disponibilizar função para que o prefeito atribua Pilas à um usuário	Could
RF25	Disponibilizar função para que o prefeito confisque Pilas de um usuário	Could
RF26	Permitir que o administrador faça cadastro de grupos de usuários	Could
RF27	Permitir que o administrador edite grupos de usuários cadastrados	Could
RF28	Permitir que o administrador remova grupos de usuários, desde que sem usuários vinculados	Could
RF29	Possuir um "Portal da transparência"onde os usuários podem acompanhar o uso da verba por parte do prefeito	Would Not
RF30	Possuir uma funcionalidade onde os usuários podem avaliar o desempenho de cada prefeito ao fim do mandato	Would Not
RF31	Disponibilizar função para leitura de código de barras de produtos	Would Not

Tabela 2 – Requisitos funcionais do sistema.

Fonte: Autoria própria (2023).

o arredondamento e gradiente utilizados no botão são também uma reprodução de um para o outro.

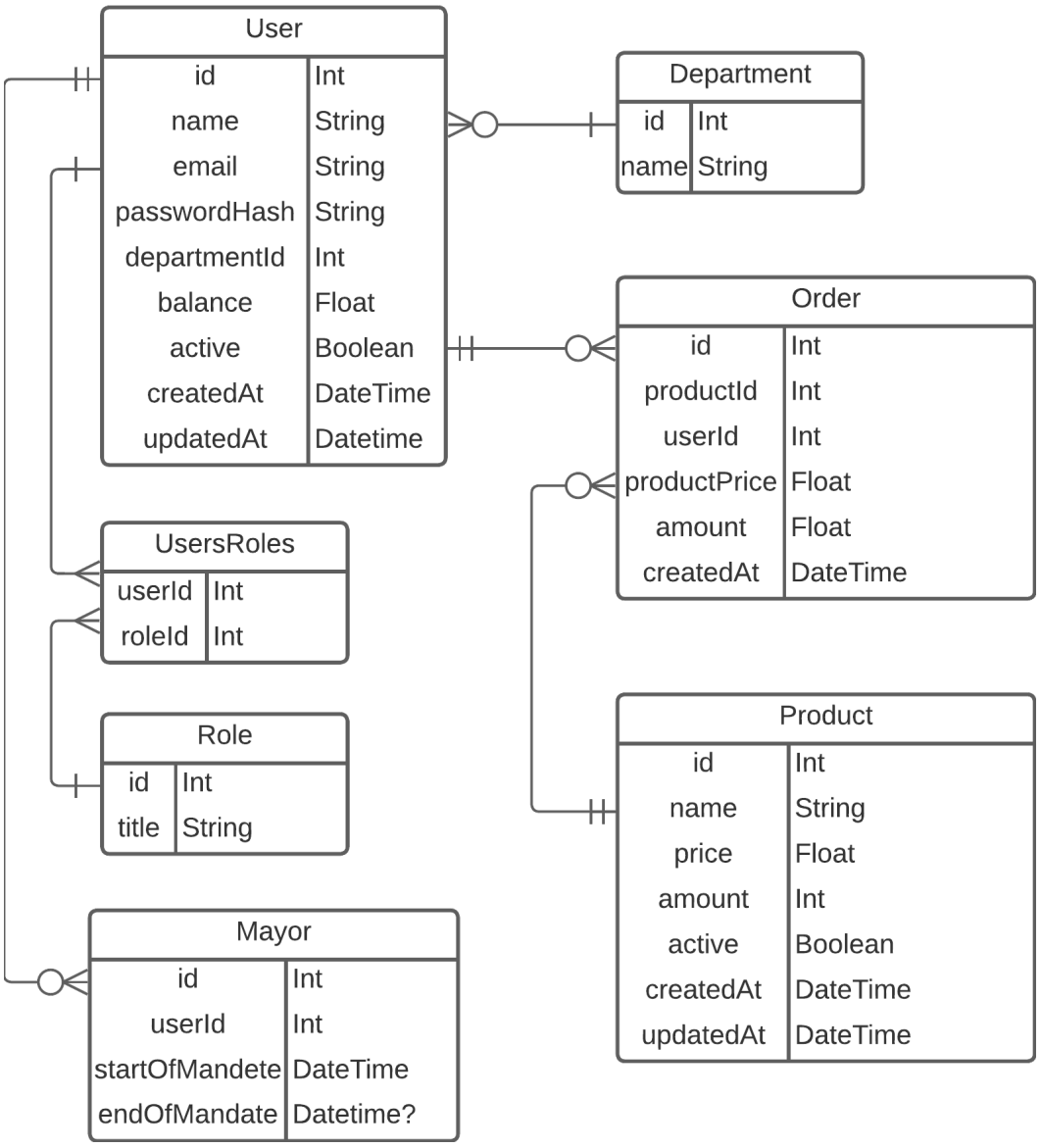


Figura 8 – Diagrama Entidade Relacionamento

Fonte: Autoria própria (2023).

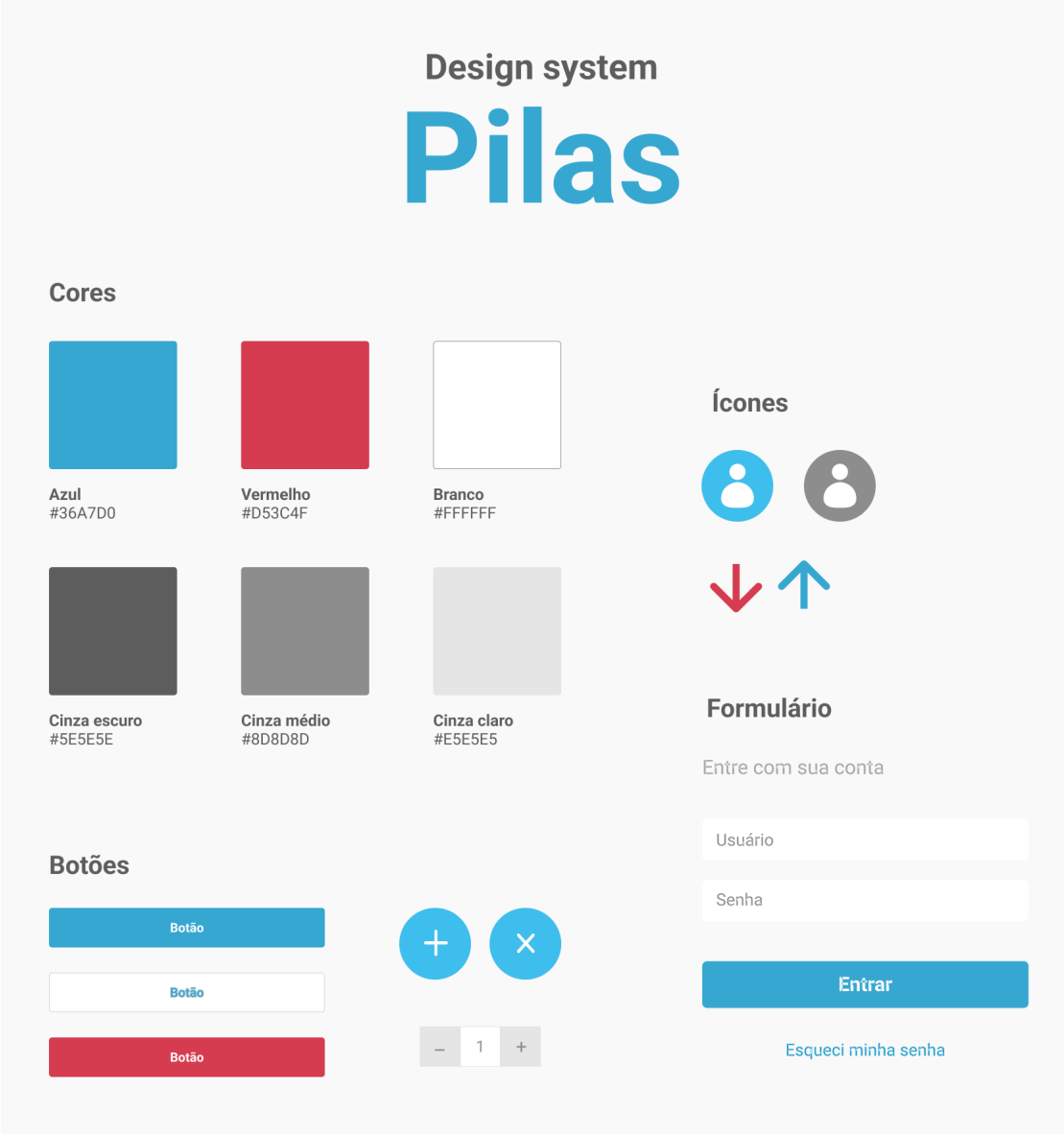


Figura 9 – Versão antiga do Design System dos Pilas
Fonte: Autoria própria (2023).

Figura 10 – Captura de tela do site que inspira o Design System
Fonte: ActTecnologia (2023).

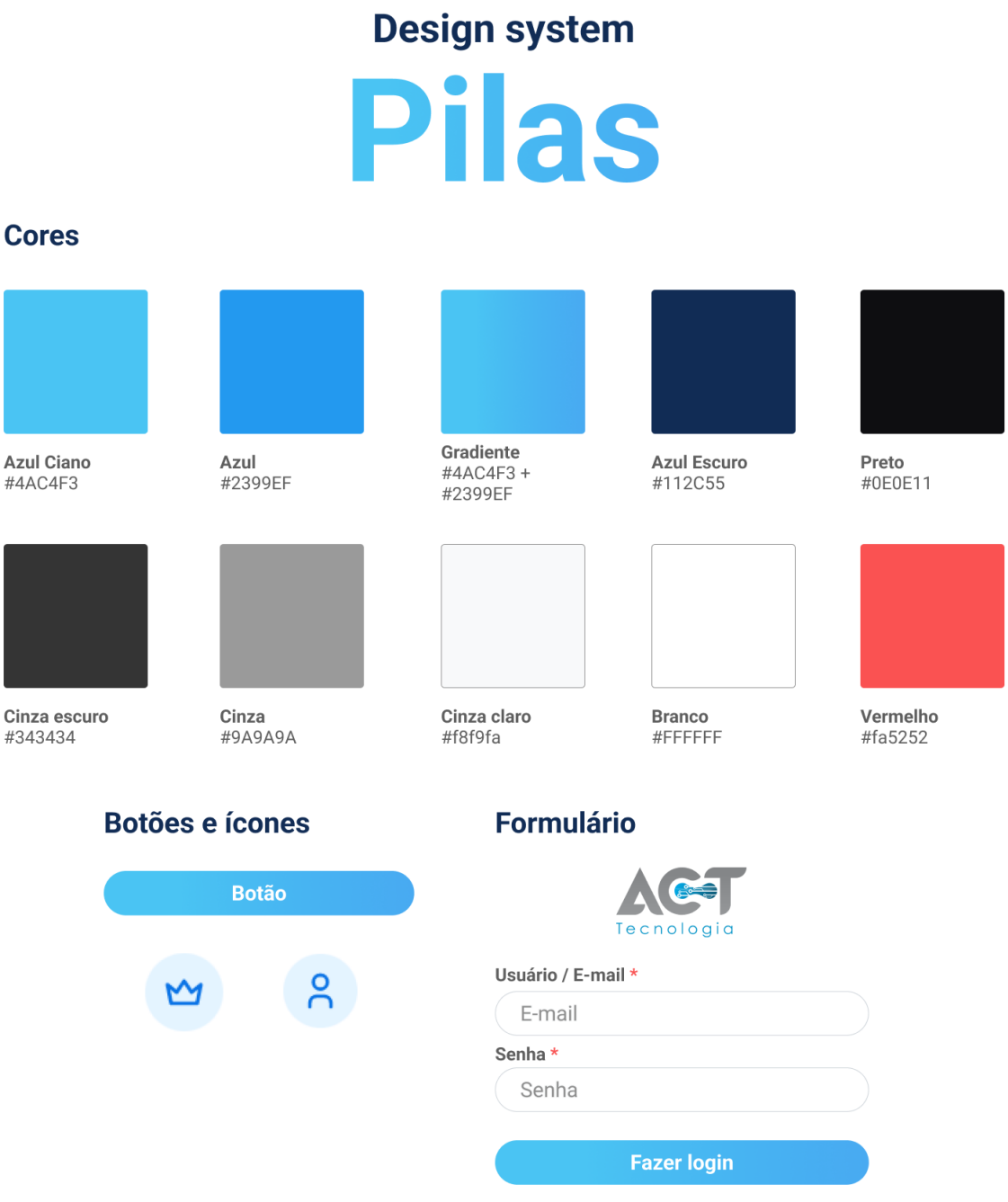


Figura 11 – Design System dos Pilas
Fonte: Autoria própria (2023).

4.3 Apresentação do sistema

Nesta seção serão apresentadas capturas das principais telas do sistema que foi desenvolvido já em produção. Durante a criação das telas buscou-se construir uma interface limpa, agradável e que seguisse o padrão definido no Design System apresentado na subseção 4.2.3. Houve também uma preocupação em deixá-las objetivas de forma a simplificar os processos a serem executados.

Algumas informações sensíveis foram censuradas nas capturas através de uma tarja de borrão, tendo em vista que as imagens foram feitas do sistema em produção.

4.3.1 Telas de usuário comum

Entende-se por usuário comum, aqueles que não possuem nenhum papel, seja de administrador ou prefeito. A **Figura 12** exibe a Página Inicial após ser feito o login com um usuário comum. Logo em seguida, na **Figura 13**, é possível visualizar a mesma tela aberta em um dispositivo móvel. Salienta-se que todas as páginas seguem um design responsivo.

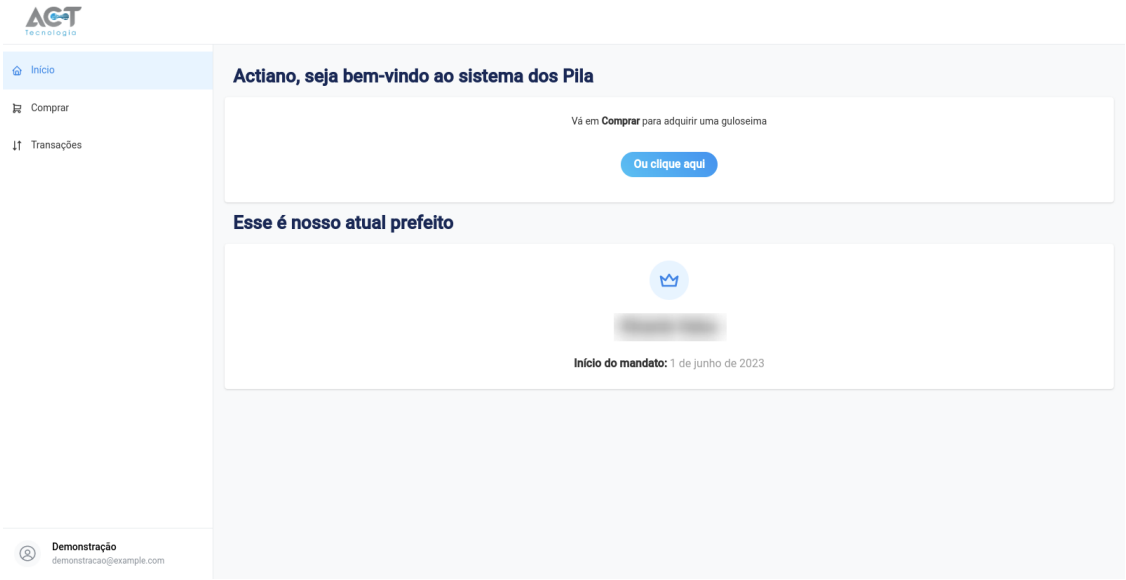
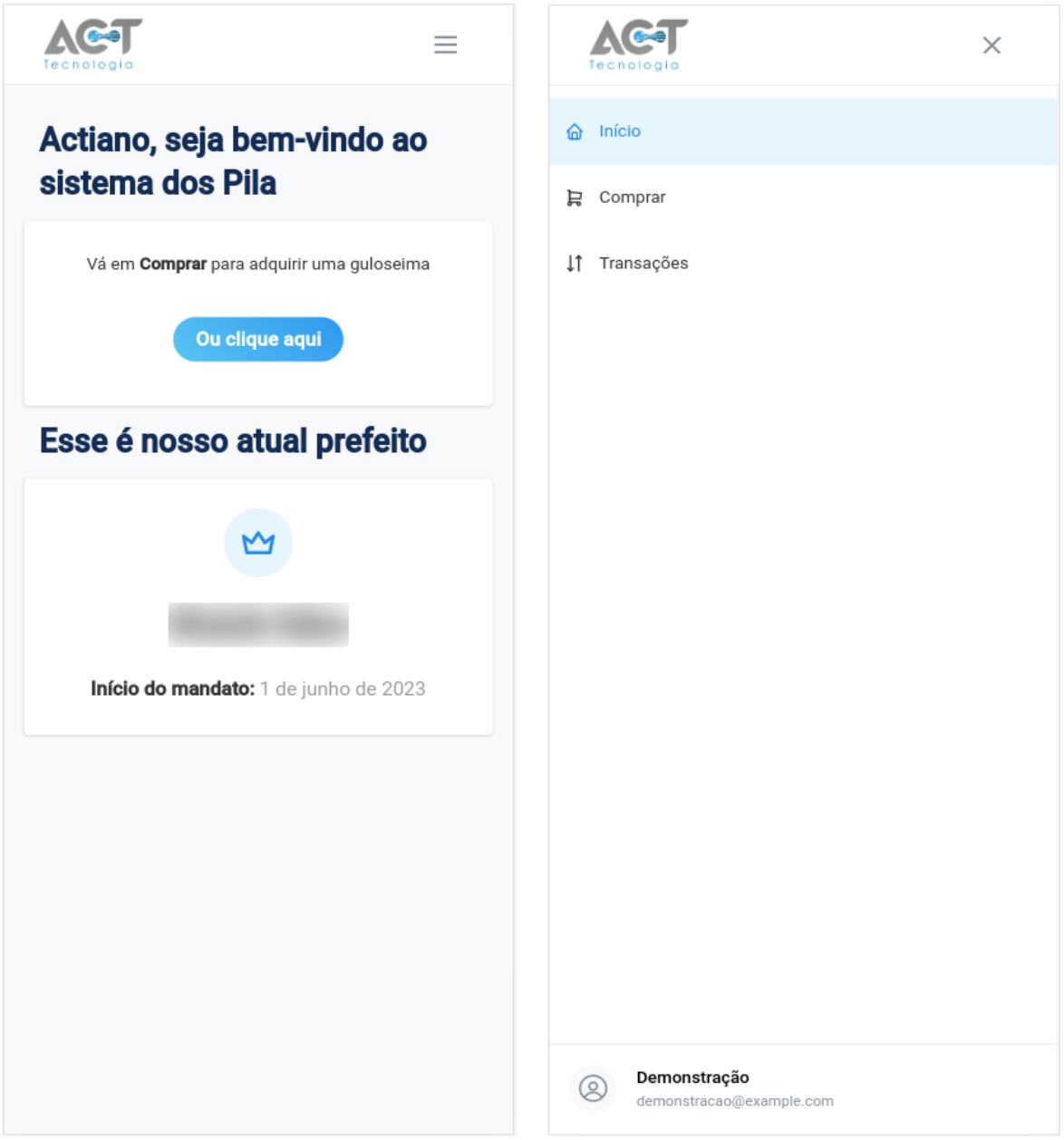


Figura 12 – Captura da Página Inicial
Fonte: Autoria própria (2023).

No momento da compra de um produto, o usuário deve abrir a opção **Comprar** do menu lateral. Nesse momento ele verá uma página conforme mostrado na **Figura 14**. Ao clicar sobre o nome de um produto para comprá-lo, será exibido um modal (**Figura 15**) para que o usuário defina a quantidade desejada e finaliza-se clicando sobre o botão **Comprar**.

Para visualizar seu histórico de compras, o usuário deverá ir até a opção **Transações** no menu lateral. Os dados do usuário podem ser consultados na opção **Minha conta**.



(a) Página Inicial **(b) Menu lateral**

Figura 13 – Capturas em dispositivo móvel: (a) Página Inicial, (b) Menu lateral

Fonte: Autoria própria (2023).

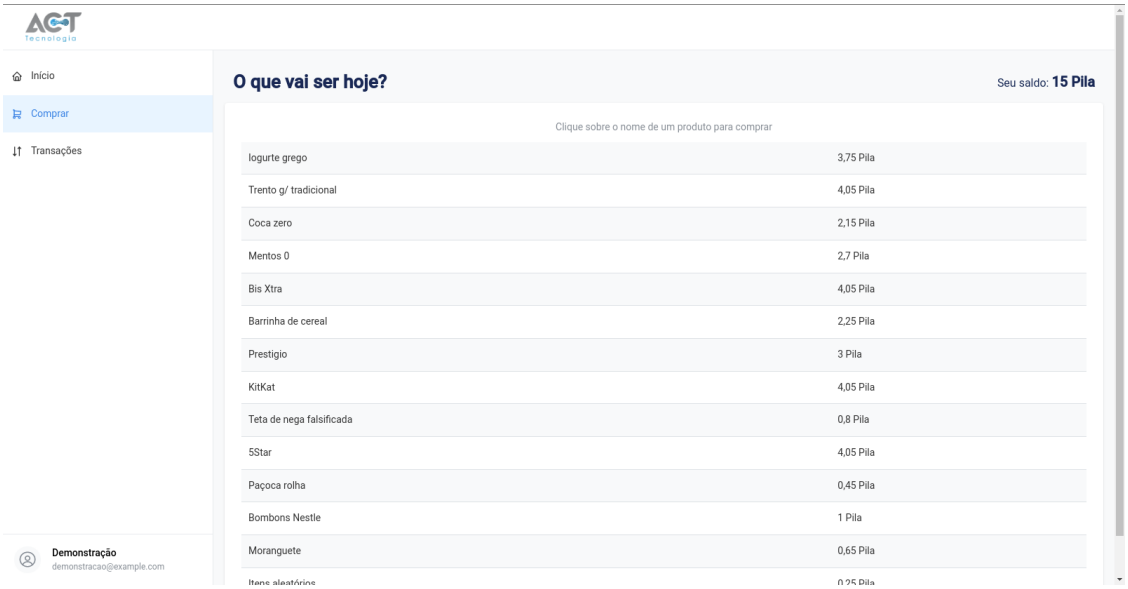


Figura 14 – Captura da Página de Produtos para Compra

Fonte: Autoria própria (2023).

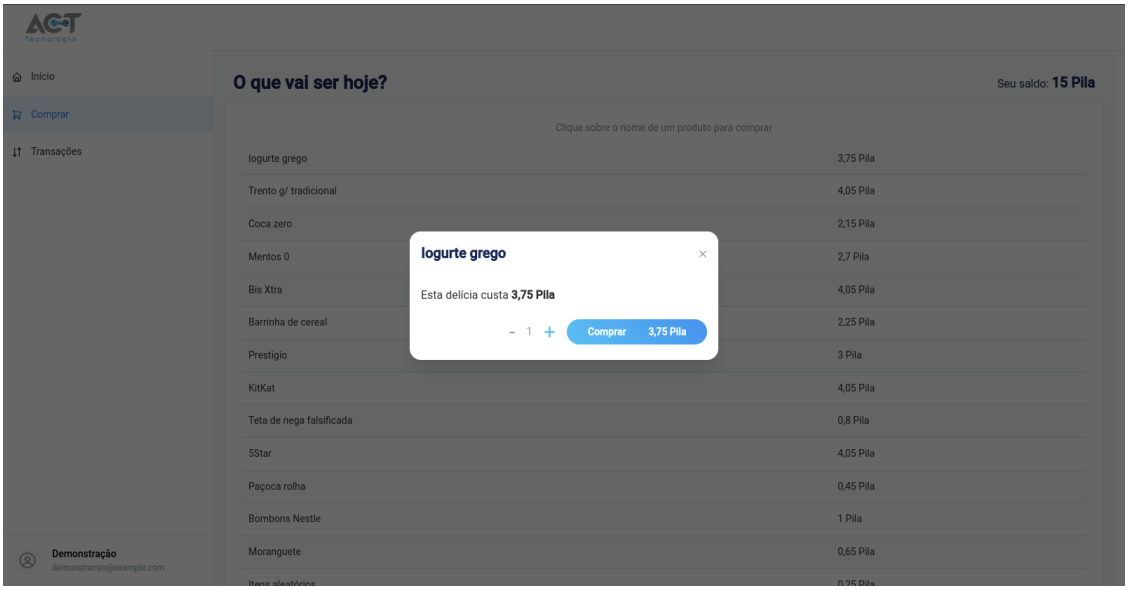


Figura 15 – Captura do Modal de Compra de Produto

Fonte: Autoria própria (2023).

4.3.2 Telas de usuário administrador

Um usuário com papel de **administrador** tem permissão nas páginas de administrador e também de prefeito, mesmo que não possua esse papel. A **Figura 16** apresenta a página onde o administrador faz a gestão de mandatos de prefeitos, ela exibe o prefeito atual e o histórico de mandatos.

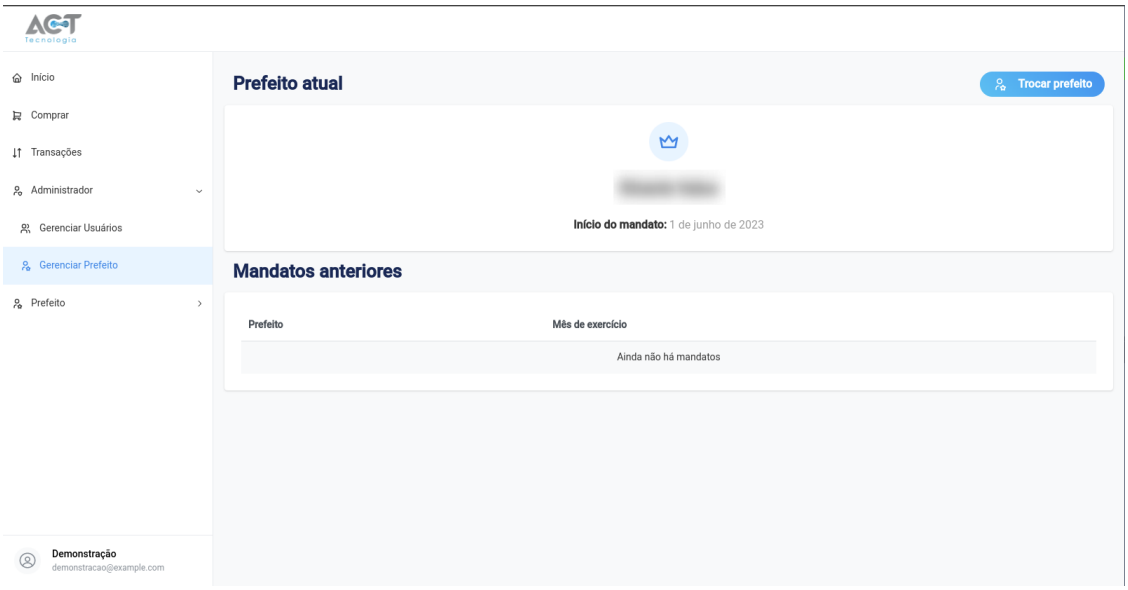


Figura 16 – Captura da Página de Gerenciamento de Mandatos de Prefeitos
Fonte: Autoria própria (2023).

Ao clicar no botão **Trocar prefeito** é aberto um *drawer* (gaveta) na lateral direita para que o administrador possa definir um novo prefeito, isso pode ser visto na **Figura 17**.

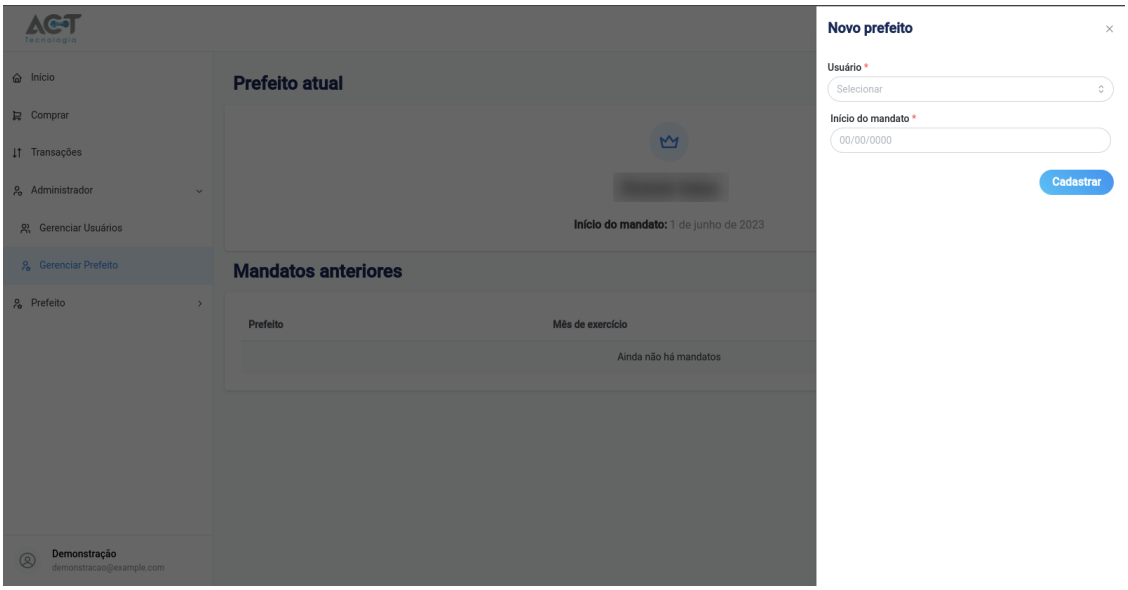


Figura 17 – Captura do Drawer de Troca de Prefeito
Fonte: Autoria própria (2023).

4.3.3 Telas de usuário prefeito

O usuário com papel de **prefeito** pode cadastrar os produtos disponíveis para compra através de um *drawer* conforme o mostrado na **Figura 18**. Para editar um produto basta clicar sobre ele já cadastrado que um *drawer* de edição é aberto.

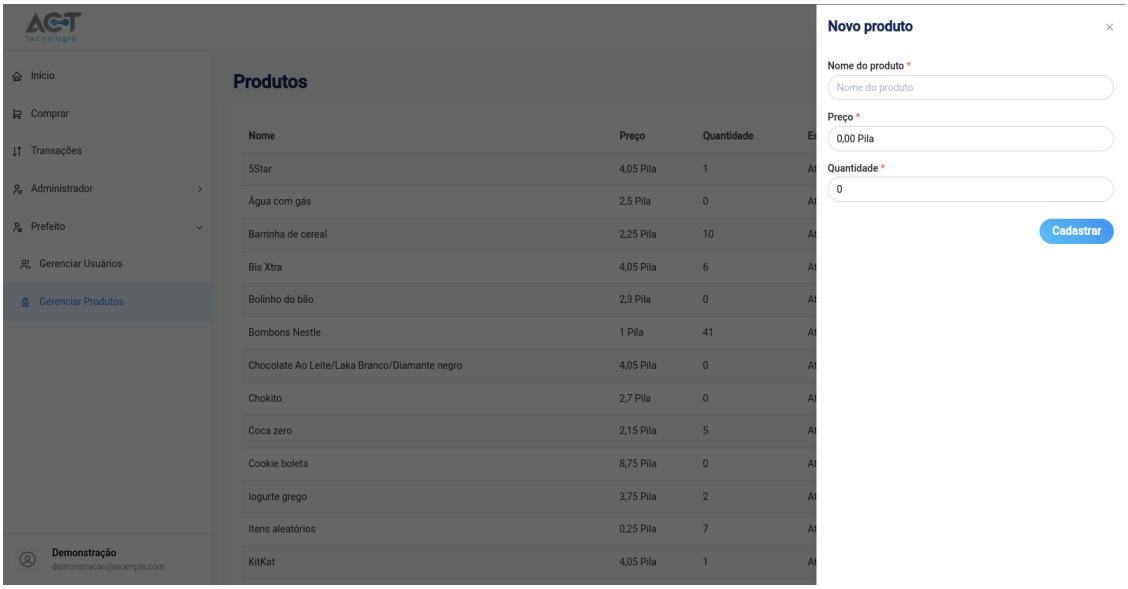


Figura 18 – Captura do Drawer de Cadastro de Produto
Fonte: Autoria própria (2023).

O **prefeito** também faz a gestão dos saldos dos usuários, dessa forma ele tem acesso a uma página como a exibida na **Figura 19**.

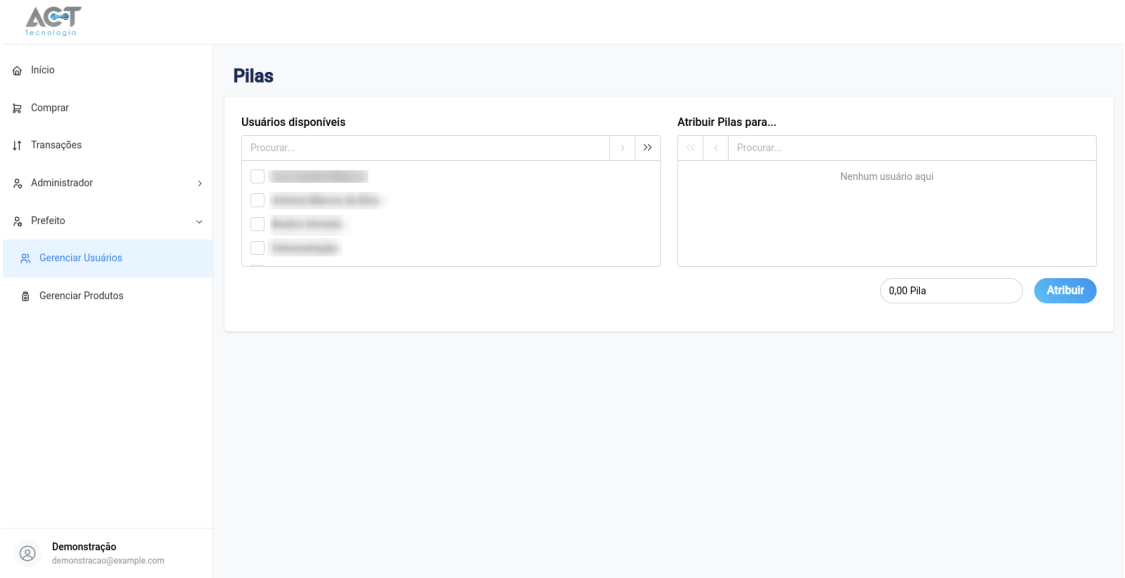


Figura 19 – Captura da Página de Gerenciamento de Saldo em Pilas
Fonte: Autoria própria (2023).

4.4 Implementação do sistema

A seguir, estão listadas, em ordem cronológica de execução, as principais tarefas classificadas como Feitas (*Done*) no quadro Project do GitHub citado na subseção 3.2.5. Cada tarefa indica uma nova *feature* implementada no sistema e conta com uma descrição do que foi desenvolvido nela.

1. **Inicialização do projeto**

   A inicialização do projeto consistiu na criação da aplicação **Next.js** conforme a própria documentação do framework orienta[1], neste momento é que foi definido que seria utilizado **TypeScript**. A partir disso a estrutura base do sistema é automaticamente criada já adequada ao **TypeScript** e tem-se uma aplicação inicial pronta para rodar com algumas páginas de demonstração.

   Além disso, o **Next.js** possui um sistema de roteamento baseado em páginas, ou seja, a forma como as páginas são nomeadas e organizadas, definirão as rotas da aplicação. Por exemplo, para criar uma rota exclusiva para administradores, de listagem de usuários, basta estruturar dentro da pasta ***pages*** os arquivos conforme mostrado na Figura 20. Assim sendo, no navegador o acesso a essa página se dará por uma url como essa *"url.do.sitema/admin/users"*.

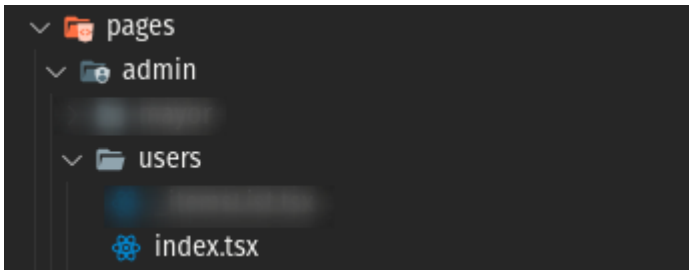


Figura 20 – Estrutura de páginas para roteamento
Fonte: Autoria própria (2023).

2. **Prototipação do modelo da base de dados**

   Nesta fase foi executada a prototipação da base de dados. Para tanto, utilizou-se do método descrito na subseção 3.2.2 e o resultado foi exposto na subseção 4.2.2.

3. **Configuração do PostgreSQL em Container do Docker**

   Conforme mencionado na subseção 3.1.4, optou-se por rodar o banco de dados PostgreSQL em um container do **Docker**. Assim sendo, foi criado um arquivo yaml que foi executado com o docker compose.

[1] Documentação para inicialização de aplicação Next.js: https://nextjs.org/docs/getting-started/installation

4. **Implementação do esquema de base de dados no Prisma**

   O próximo passo foi fazer a inicialização do **Prisma** na aplicação e em seguida a criação do esquema da base de dados, a partir disso foi rodada a primeira migração da *Data Base* (DB). O **Prisma** possibilita que isso seja feito através de modelos para as entidades, um exemplo pode ser visto na Listagem 1, onde é exibida a primeira versão do modelo criado para a entidade Usuário. Este trecho de código gera uma tabela no banco de dados com os relacionamentos adequados através da migração, a Listagem 2 mostra um fragmento dos comandos SQL gerados pela migração do **Prisma**.

```
1   model User {
2     id             Int          @id @default(autoincrement())
3     name           String
4     email          String       @unique
5     passwordHash   String
6     department     Department   @relation
7                                 (
8                                     fields: [departmentId], references: [id]
9                                 )
10    departmentId   Int
11    balance        Float?
12    roles          Role[]
13    createdAt      DateTime     @default(now())
14    updatedAt      DateTime     @updatedAt
15    admin          Admin?
16    mayor          Mayor[]
17    orders         Order[]
18  }
```

Listagem 1 – Primeira Versão do Modelo Usuário no Prisma Schema
Fonte: Autoria própria (2023).

5. **Implementação de seeders**

   Com o intuito de facilitar o desenvolvimento, as tabelas da DB foram populadas através de seeds, funcionalidade também proporcionada pelo **Prisma**[2]. A tabela de Usuário foi populada com alguns dados fictícios e também com o usuário padrão administrador que foi mantido futuramente em produção, isso satisfez o **RF01** listado na **Tabela de Requisitos Funcionais** (Tabela 2).

6. **Configuração do NextAuth**

   Para autenticação no sistema, utilizou-se o **NextAuth.js**. Através dessa ferramenta é possível automatizar o login e logout de usuários cadastrados na DB. O **NextAuth** fornece funções para armazenar a sessão e fazer a verificação da mesma para validar

[2] Documentação para utilização de seeds no Prisma: https://www.prisma.io/docs/guides/migrate/seed-database

```sql
-- CreateTable
CREATE TABLE "User" (
    "id" SERIAL NOT NULL,
    "name" TEXT NOT NULL,
    "email" TEXT NOT NULL,
    "passwordHash" TEXT NOT NULL,
    "departmentId" INTEGER NOT NULL,
    "balance" DOUBLE PRECISION,
    "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" TIMESTAMP(3) NOT NULL,

    CONSTRAINT "User_pkey" PRIMARY KEY ("id")
);

-- CreateIndex
CREATE UNIQUE INDEX "User_email_key" ON "User"("email");

-- AddForeignKey
ALTER TABLE "User" ADD CONSTRAINT "User_departmentId_fkey"
FOREIGN KEY ("departmentId") REFERENCES "Department"("id")
ON DELETE RESTRICT ON UPDATE CASCADE;

-- Other ForeignKeys
[...]
```

Listagem 2 – Primeira migração de Usuário gerada pelo Prisma
Fonte: Autoria própria (2023).

se há um usuário logado. Dessa forma, satisfazendo os requisitos funcionais **RF02** e **RF03**.

A contar dessa situação, foi possível também criar uma função que verifica se o usuário armazenado na sessão possui os privilégios adequados, ou seja, se ele possui os papéis (*roles*) de Administrador (*ADMIN*) ou Prefeito (*MAYOR*), retornando *true* caso positivo e *false* caso negativo. O trecho de código da Listagem 3 exibe a função mencionada.

Se o usuário acessa uma página que exige determinado papel que ele não possui, o mesmo é redirecionado para a tela de login.

7. **Criação do layout do sistema**

   Nesta fase deu-se início a criação da UI dos Pilas e conforme mencionado na subseção 3.1.6, utilizou-se a biblioteca **Mantine** para este fim. Dessa forma, os primeiros componentes da interface do sistema foram os do layout principal, tal como o menu lateral através do qual é proporcionada a navegação entre as páginas. Vale ressaltar que este menu lateral exibe apenas os links para páginas as quais o usuário tem privilégios conforme os seus papéis no sistema.

```
1   import { Session } from "next-auth";
2
3   export default function containsRole(
4     user: Session["user"] | undefined,
5     role: "ADMIN" | "MAYOR"
6   ) {
7     if (!user) return false;
8     return user.roles.find
9     ((userRole) =>
10      userRole.title === role
11    ) != undefined;
12  }
```

Listagem 3 – Função para validação de papel de usuário
Fonte: Autoria própria (2023).

8. **Criação da página de produtos**

A página que lista os produtos disponíveis para compra foi a primeira a ser desenvolvida. Nesta tarefa foram criados tanto a UI (User Interface) quanto a lógica do lado servidor, englobando a validação dos dados da compra e a persistência na DB.

Por consequência deu-se início a criação da Application Programming Interface (API). O **Next.js** disponibiliza além do supracitado roteamento baseado em páginas, o roteamento de API, dessa forma, basta criar dentro da pasta ***pages*** uma segunda pasta chamada ***api*** e todo arquivo criado dentro dela será traduzido para uma rota de API[3]. Estes arquivos serão pacotes exclusivos do lado servidor, não se tornando acessíveis como uma página do lado cliente.

Para chamar as rotas de API, utilizou-se nessa aplicação o *client* HTTP ***Axios***[4], bastando instalar suas dependências e importá-lo nas páginas. Veja um exemplo de como fica a estrutura das páginas na Figura 21 e como fica uma requisição *POST* para essa mesma rota de API no trecho de código da Listagem 4.

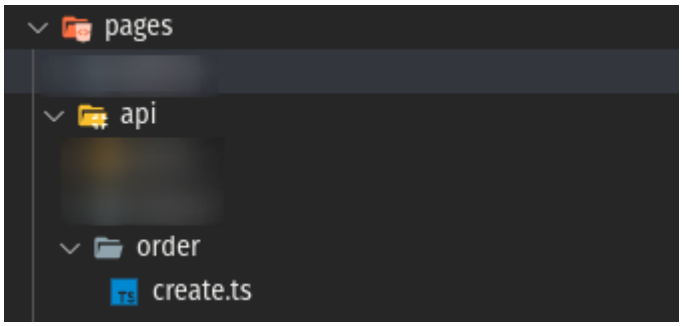


Figura 21 – Estrutura de páginas para API
Fonte: Autoria própria (2023).

[3] Documentação sobre o Next API routes: https://nextjs.org/docs/pages/building-your-application/routing/api-routes
[4] https://axios-http.com/

```
1    axiosApi
2      .post('/api/order/create', values)
3      .then((res) => {
4        // Tratamento para a resposta
5      })
6      .catch((e) => {
7        // Tratamento de erros
8      })
```

Listagem 4 – Requisição POST para a API
Fonte: Autoria própria (2023).

9. **Criação das páginas de gerenciamento**

   Na sequência foram criadas as páginas de gerenciamento, como criação de usuários, definição de prefeito, criação de produtos e gestão de pilas. As páginas seguem um padrão similar na sua construção.

   Para exibir ao usuário as listagens utilizaram-se tabelas e para alimentá-las foram coletados os dados da base de dados. As requisições para a base foram feitas de dentro de uma função do Next.js chamada **getServerSideProps**. Esta função pré renderiza os dados em *server side* toda vez que é feita uma requisição[5].

   O trecho de código da Listagem 5 exibe uma *query* do Prisma que busca todos os usuários por ordem alfabética dos nomes, onde o *include* faz o join com as tabelas de papéis e departamento. Conforme observado, para busca de vários resultados com o Prisma utiliza-se o *findMany*. Destaca-se que essa requisição está sendo feita de dentro de uma função getServerSideProps.

10. **Configuração do PWA**

    Visando melhor usabilidade da aplicação conforme já mencionado na subseção 3.1.7, finalizou-se o desenvolvimento do Minimum Viable Product (MVP) do sistema com sua publicação como **PWA**.

    Para tanto, foi instalada a dependência do plugin de **PWA** para **Next.js** e realizadas as configurações necessárias conforme instruído da documentação do **next-pwa**[6]. Essa configuração é rápida e não exige que sejam feitas grandes adaptações, sobretudo em casos onde a aplicação tem *layout* responsivo, o que corresponde a situação do que foi desenvolvido no presente trabalho.

11. **Implantação do sistema**

[5] Documentação do Next.js sobre getServerSideProps: https://nextjs.org/docs/pages/building-your-application/data-fetching/get-server-side-props

[6] Repositório no GitHub com documentação para configuração do PWA com Next.js: https://github.com/shadowwalker/next-pwa

```
export const getServerSideProps: GetServerSideProps = async (context) => {
    // ...
    const users = await prisma.user.findMany({
        include: {
          roles: true,
          department: true
        },
        orderBy: {
          name: 'asc',
        },
    })

    // ...

    return {
        props: {
            users,
        },
    };
}
```

Listagem 5 – Busca de usuários com Prisma em server side
Fonte: Autoria própria (2023).

A implantação, também chamada de *deploy*, foi executada em ambiente virtualizado no servidor da empresa[7]. Manteve-se o PostgreSQL em *Docker container*, mas com configurações específicas para ambiente de produção. Também houve uma adaptação nos seeder da aplicação de modo que não gerassem mais dados fictícios, somente o usuário administrador e os departamentos, os quais ainda não possuem CRUD no sistema.

Para disponibilizar o sistema na internet, foi configurado um túnel da CloudFlare, conceito já explicado na subseção 3.1.9, utilizando o domínio da própria empresa. Isso permitiu acesso ao sistema de qualquer lugar do mundo através de um subdomínio sem a necessidade de um IP público próprio para o servidor dos Pilas.

Uma vez implantado, foram criados no sistema os acessos individuais de cada colaborador. Inicialmente criou-se acesso apenas para o atual prefeito, o qual teria que fazer o cadastro dos produtos em estoque. Em seguida, foram criados acessos a pessoas chaves que testariam a funcionalidade de gerenciamento de usuários criando os acessos do restante da equipe e só então o prefeito fez a distribuição de pilas conforme saldos que estavam registrados na até então planilha de gestão.

Por fim, todos puderam acessar o sistema com suas próprias credenciais e a compra de produtos passou a ser toda centralizada nele.

[7] O passo a passo para fazer o *deploy* está documentado no repositório dos Pilas no GitHub: https://github.com/BeaFernandes/pilas

4.5 Formulário para Coleta de Feedback

Após uma semana de implantação do sistema na empresa, foi disponibilizado um formulário do Google[8] com algumas perguntar relacionadas sobre a percepção dos usuários a respeito do sistema implantado. O sistema foi testado por um total de dezesseis colaboradores e todos eles responderam ao formulário, as respostas eram anônimas para que fosse obtido um retorno mais honesto possível. Nesta seção são apresentados os resultados para algumas das perguntas da pesquisa (Figuras 22 a 25).

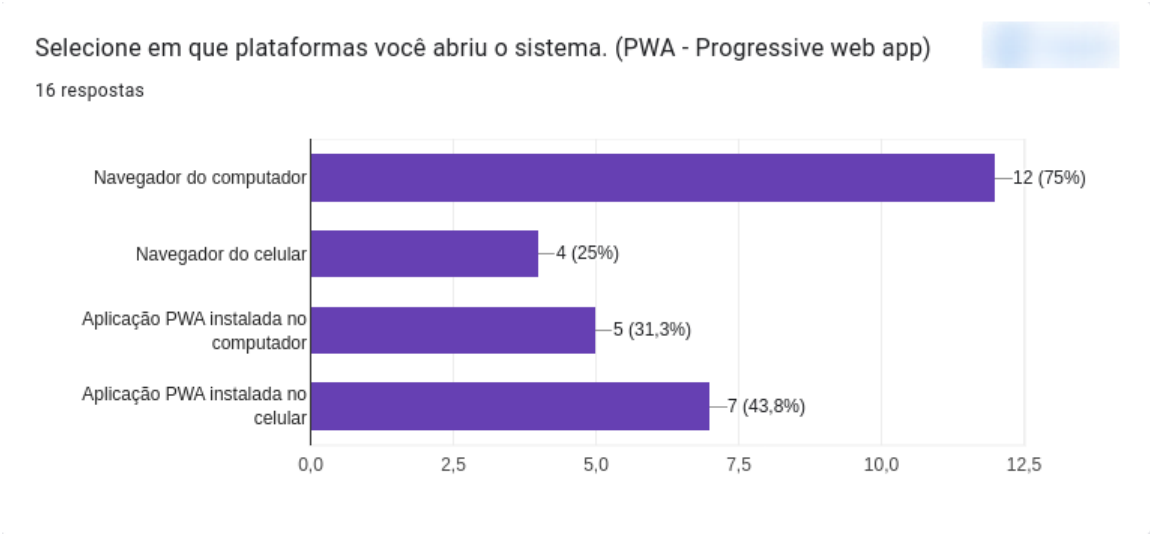


Figura 22 – Primeira pergunta
Fonte: Autoria própria (2023).

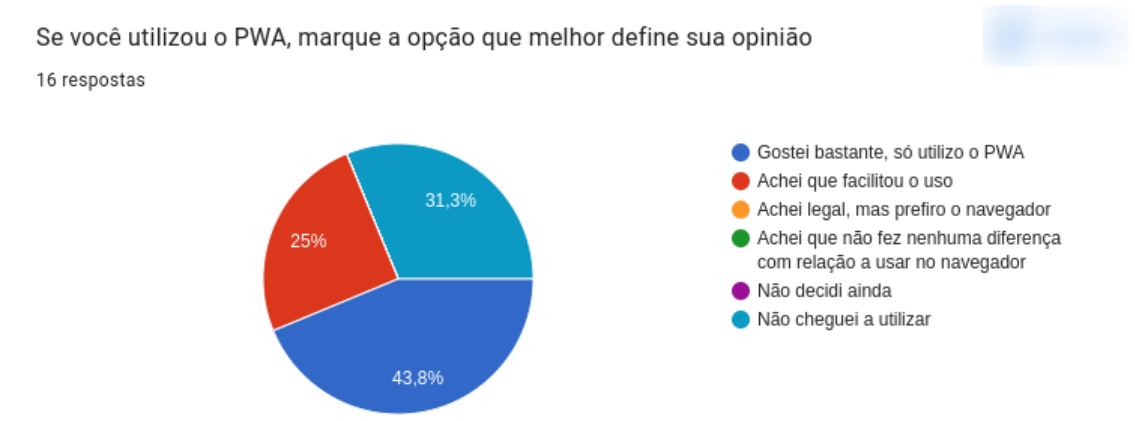


Figura 23 – Segunda pergunta
Fonte: Autoria própria (2023).

[8] https://docs.google.com/forms

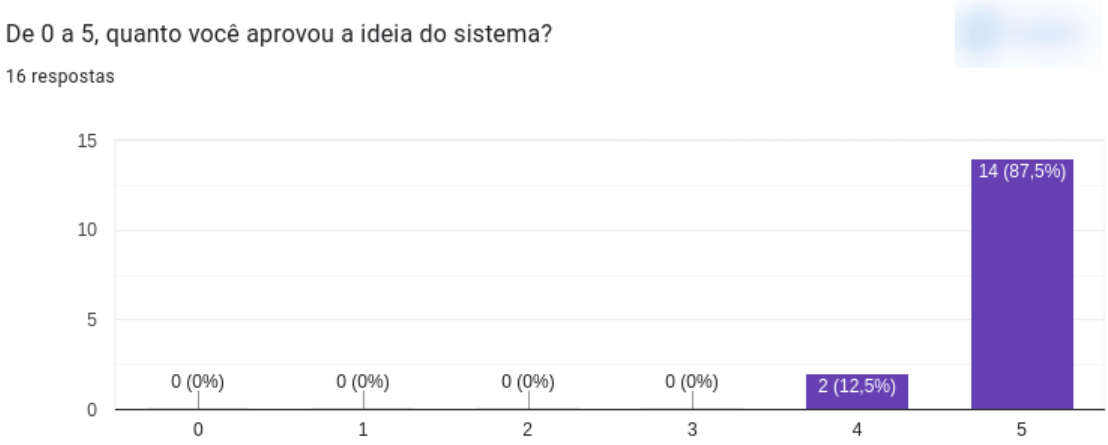


Figura 24 – Terceira pergunta
Fonte: Autoria própria (2023).

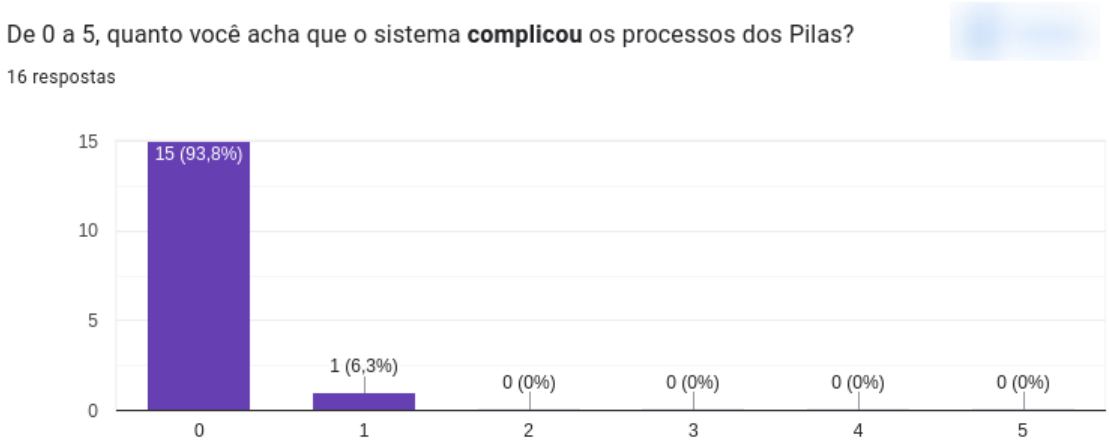


Figura 25 – Quarta pergunta
Fonte: Autoria própria (2023).

Quando perguntado se encontraram **problemas ou *bugs***, algumas das respostas foram:

- "Apenas dos valores não arredondarem, e algumas vezes ficam tipo 5,99999999999";
- "Após uma alteração, é necessário recarregar a página para que atualize as modificações";
- "No meu celular ele cortou o menu do usuário, tanto no navegador quanto no pwa, utilizo um iPhone 13";
- "Tabelas com informações cortadas no celular".

E ao perguntar sobre **sugestões de novas funcionalidades**, essas foram algumas das respostas:

- "Acredito que seja bacana ter um campo de pesquisa para facilitar a filtragem na compra de um item";
- "Alerta de desconto";
- "Saldo dos pilas na página inicial";
- "Gerar um relatório após a finalização de um mandato";
- "Mostrar controlar os gastos da prefeitura (portal da transparência)";
- "Imagem dos itens";
- "Empréstimo de pilas (para caso alguem queira comprar produtos com o coleguinha)=)".

Também alguns **comentários positivos**:

- "Muito bacana e funcional o sistema vai ajudar muito";
- "Ficou excelente!!!";
- "Eu acho que está perfeito";
- "Tudo perfeito!".

4.6 Discussões

Nesta seção são apresentadas algumas discussões a respeito dos resultados obtidos na pesquisa.

O sistema teve uma boa aceitação da equipe. Mais de 60% dos usuários instalaram o PWA em celulares ou computadores e 30% deles disse estar utilizando apenas o PWA em detrimento do próprio navegador. Isso indica que o objetivo de entregar praticidade através do PWA foi alcançado.

Houve um usuário que reportou ter tido problemas com o menu da aplicação, onde a parte de informações do usuário estaria ficando cortada em seu dispositivo móvel, que ainda segundo o usuário, se trata de um *iPhone 13*. Alguns usuários também reportaram que os dados das tabelas estariam cortando quando abertas em dispositivos móveis. Apesar desses relatos, o sistema se adaptou bem aos diferentes tamanhos de telas em um modo geral, sendo que a experiência em celulares ficou melhor quando com o PWA instalado.

Vários outros usuários reportaram um problema na exibição do seu saldo de Pilas, onde em certos momentos os número ficam com muitas casas decimais. Foi percebido que o problema ocorre por falta de tratamento adequado no momento do cálculo das compras de itens, sendo que ao subtrair o valor dos produtos comprados do saldo do usuário não é feito um arredondamento do número.

Outro problema percebido foi que as páginas estão precisando ser recarregadas após alguma alteração. Por exemplo, ao comprar um item o saldo exibido na tela não atualiza automaticamente, exigindo que o usuário recarregue a página. Esta situação ocorre apenas em ambiente de produção e se deve provavelmente a uma otimização do **Next.js**, onde ele faz *cache*[9] dos dados para aprimorar o tempo de resposta e fazer menos requisições ao servidor[10].

[9] Cache: armazenamento de dados em uma memória temporária para acesso rápido.

[10] Explicação dessa otimização na documentação no Next.js: https://nextjs.org/docs/pages/building-your-application/deploying/production-checklist

5 CONCLUSÃO

Esse trabalho propôs o desenvolvimento de um sistema web para controle do consumo de guloseimas pelos colaboradores de uma empresa de tecnologia em um projeto interno denominado *Pilas*, tendo a finalidade de substituir o método até então utilizado, onde esse controle se dava por meio de planilhas estáticas suscetíveis a falhas.

O *Pilas* é um projeto descontraído mas com propósito sério de fortalecimento da cultura organizacional interna da empresa em questão, sendo uma iniciativa que visa gerar engajamento e proporcionar bem estar à toda equipe. Dadas suas características e demandas únicas, não foi encontrado um sistema existente que suprisse por completo as necessidades do *Pilas*, encorajando dessa forma a idealização do presente trabalho.

O objetivo foi desenvolver durante o semestre, um MVP que atendesse as demandas essenciais do *Pilas* ao ponto deste ser implantado na empresa. Esperava-se também que fosse possível obter *feedbacks* dos usuários.

Pode-se dizer que o objetivo principal foi alcançado, obteve-se um sistema apto para uso real, apesar de terem sido encontrados em produção pequenos problemas a serem corrigidos. Foram satisfeitos os requisitos funcionais de prioridade ***Must have*** (requisitos obrigatórios) do **RF01** ao **RF16** e também alguns de prioridade ***Should have*** (importantes, não essenciais) e ***Could have*** (desejáveis). Quanto aos requisitos não funcionais, estes foram todos atendidos. Houve uma boa aceitação dos usuários para com o sistema, dessa forma seu uso será continuado e ele permanecerá em produção.

5.1 Trabalhos futuros

Há alguns trabalhos a serem realizados futuramente para aprimorar o sistema. Primeiramente, é interessante que os requisitos funcionais de prioridade ***Should have*** e ***Could have*** que não foram satisfeitos durante o desenvolvimento do presente trabalho, sejam executados, pois mesmo não sendo essenciais, se mostram muito úteis.

Outro trabalho futuro a ser desenvolvido, é a possibilidade de gerar diversos tipos de relatórios sobre as transações realizadas, principalmente com intuito de facilitar o trabalho do prefeito e possuir controle sobre as possíveis inadimplências.

Uma outra importante necessidade observada, é a viabilização de persistência de toda e qualquer transação realizada no sistema, atualmente mantém-se apenas registro de itens comprados. O que pode ser desenvolvido é o armazenamento em banco de movimentações de Pilas atribuídos ou confiscados pelo prefeito, para que dessa forma, garanta-se uma maior integridade dos dados e proporcione-se relatórios mais precisos.

A partir dos testes com os usuários em ambiente de produção foi possível identificar também alguns problemas que deverão ser resolvidos. Além do mais, percebeu-se que o não

desenvolvimento de testes automatizados foi uma considerável lacuna para este projeto, deste modo, é uma funcionalidade interessante a ser implementada.

É ainda valioso que os problemas de *cache* de dados por parte do Next.js percebidos em ambiente de produção sejam estudados. Percebeu-se que esse problema incomodou a muitos usuários e que realmente torna a aplicação menos pática. Assim sendo, deverá ser analisada uma possível solução.

Pretende-se também, aprimorar a segurança da aplicação. Um ponto já levantado nesse sentido, é a necessidade de restringir as requisições feitas para a API, de forma a aceitar apenas requisições vindas de fontes confiáveis, ou mesmo, apenas vindas do *client side* do sistema.

REFERÊNCIAS

ACTTECNOLOGIA. **Site Act Tecnologia**. [*S.l.*], 2023. Disponível em: https://acttecnologia.com.br/. Acesso em: 20 de junho de 2023.

ADETUNJI, O. *et al.* Dawning of progressive web applications (pwa): Edging out the pitfalls of traditional mobile development. **American Academic Scientific Research Journal for Engineering, Technology, and Sciences**, v. 68, n. 1, p. 85–99, 2020.

BIELEMANN, R. M. *et al.* Consumo de alimentos ultraprocessados e impacto na dieta de adultos jovens. **Revista de Saúde Pública**, SciELO Public Health, v. 49, p. 28, 2015.

BLISCHAK, J. D.; DAVENPORT, E. R.; WILSON, G. A quick introduction to version control with git and github. **PLoS computational biology**, Public Library of Science, v. 12, n. 1, p. e1004668, 2016.

CLOUDFLARE. **O que é a Cloudflare?** [*S.l.*], 2023. Disponível em: https://www.cloudflare.com/pt-br/learning/what-is-cloudflare/. Acesso em: 27 de junho de 2023.

CLOUDFLARE. **O que é um servidor de borda da CDN?** [*S.l.*], 2023. Disponível em: https://www.cloudflare.com/pt-br/learning/cdn/glossary/edge-server/. Acesso em: 27 de junho de 2023.

CONTESTOQUE. [*S.l.*], 2022. Disponível em: http://www.contestoque.com.br/. Acesso em: 12 de outubro de 2022.

DEVARAKONDA, R. S. Object-relational database systems—the road ahead. **XRDS: Crossroads, The ACM Magazine for Students**, ACM New York, NY, USA, v. 7, n. 3, p. 15–18, 2001.

DOCKERINC. **Docker Compose**. [*S.l.*], 2023. Disponível em: https://docs.docker.com/compose/. Acesso em: 26 de junho de 2023.

DOCKERINC. **Docker Engine**. [*S.l.*], 2023. Disponível em: https://docs.docker.com/engine/. Acesso em: 26 de junho de 2023.

FEDOSEJEV, A. **React. js essentials**. [*S.l.*]: Packt Publishing Ltd, 2015.

FELTER, W. *et al.* An updated performance comparison of virtual machines and linux containers. *In*: IEEE. **2015 IEEE international symposium on performance analysis of systems and software (ISPASS)**. [*S.l.*], 2015. p. 171–172.

GACKENHEIMER, C. **Introduction to React**. [*S.l.*]: Springer, 2015. v. 52.

GITHUB. **About Projects**. [*S.l.*], 2023. Disponível em: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects. Acesso em: 20 de junho de 2023.

KUHN, J. Decrypting the moscow analysis. **The workable, practical guide to Do IT Yourself**, v. 5, 2009.

LENARDUZZI, V.; TAIBI, D. Mvp explained: A systematic mapping study on the definitions of minimal viable product. *In*: IEEE. **2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)**. [*S.l.*], 2016. p. 112–119.

MACDONALD, D. **Practical ui patterns for design systems: Fast-track interaction design for a seamless user experience**. [*S.l.*]: Apress, 2019.

MASLACH, C.; SCHAUFELI, W. B.; LEITER, M. P. Job burnout. **Annual review of psychology**, Annual Reviews 4139 El Camino Way, PO Box 10139, Palo Alto, CA 94303-0139, USA, v. 52, n. 1, p. 397–422, 2001.

MEIJER, E.; DRAYTON, P. Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages. *In*: CITESEER. [*S.l.*], 2004.

META, P. **Documentação**. [*S.l.*], 2022. Disponível em: https://pt-br.reactjs.org/. Acesso em: 05 de dezembro de 2022.

MICROSOFT. **TypeScript for the New Programmer**. [*S.l.*], 2022. Disponível em: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html. Acesso em: 01 de dezembro de 2022.

MILANI, A. **PostgreSQL-Guia do Programador**. [*S.l.*]: Novatec Editora, 2008.

MOTTA, F. C. P.; VASCONCELOS, I. F. G. A cultura organizacional. **MOTTA, Fernando C. Prestes. Teoria geral da administração**, v. 3, 2002.

NEX. [*S.l.*], 2022. Disponível em: http://www.nextar.com.br/. Acesso em: 05 de outubro de 2022.

NODE.JS. **Node.js**. [*S.l.*], 2023. Disponível em: https://nodejs.org/. Acesso em: 26 de junho de 2023.

PAHL, C. *et al.* Cloud container technologies: a state-of-the-art review. **IEEE Transactions on Cloud Computing**, IEEE, v. 7, n. 3, p. 677–692, 2017.

POSTGRESQL. **About**. [*S.l.*], 2022. Disponível em: https://www.postgresql.org/about/. Acesso em: 05 de dezembro de 2022.

PRISMA. **Documentation**. [*S.l.*], 2022. Disponível em: https://www.prisma.io/docs/concepts/overview. Acesso em: 05 de dezembro de 2022.

ROTH, R. What is mobile first cartographic design. *In*: INTERNATIONAL CARTOGRAPHIC ASSOCIATION BERN, SWITZERLAND. **ICA Joint Workshop on User Experience Design for Mobile Cartography**. [*S.l.*], 2019.

SONG, I.-Y.; EVANS, M.; PARK, E. K. A comparative analysis of entity-relationship diagrams. **Journal of Computer and Software Engineering**, v. 3, n. 4, p. 427–459, 1995.

STAIANO, F. **Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop**. [*S.l.*]: Packt Publishing Ltd, 2022.

TINY. **Tiny**. [*S.l.*], 2022. Disponível em: https://www.tiny.com.br/. Acesso em: 06 de dezembro de 2022.

VERCEL. **About Next.js**. [*S.l.*], 2022. Disponível em: https://nextjs.org/learn/foundations/about-nextjs. Acesso em: 06 de dezembro de 2022.