UNIVERSIDADE FEDERAL DE SANTA CATARINA

BRUNO DUARTE BARRETO BORGES

APLICAÇÃO PWA PARA GERENCIAMENTO DE ESTOQUE

Florianópolis - SC

2022

Bruno Duarte Barreto Borges

APLICAÇÃO PWA PARA GERENCIAMENTO DE ESTOQUE

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Ricardo Pereira e Silva

Florianópolis - SC

2022

RESUMO

Toda empresa que possui um estoque necessita manter o controle sobre os produtos que possui e sua quantidade. Uma maneira simples de fazer isso é usar tabelas, normalmente a partir de programas cujo objetivo não é específico para a situação, como o Excel do Microsoft Office ou o Sheets do Google. Porém, apesar de servir o propósito, em uma alta escala, o controle da entrada e saída pode ficar mais complicado e descobrir o que tem de sobra e falta fica difícil. Este trabalho tem por objetivo desenvolver uma aplicação web que permita a criação de listas de estoque onde o usuário possa adicionar produtos e controlar a quantidade, além de conseguir compartilhar suas listas com outras pessoas. Adicionalmente, é possível criar configurações específicas para colocar produtos em uma lista de compras separada quando a quantidade estiver menor que um valor pré-cadastrado. Além disso, o aplicativo será um PWA, o que permite que ele possa ser acessado tanto em computadores quanto em dispositivos móveis sem a necessidade de instalar o programa.

Palavras-chaves: PWA, Controle de Estoque, Aplicação Web.

ABSTRACT

Every company that has an inventory needs to keep control over the products it has and their quantity. A simple way to do this is to use tables, usually from programs whose purpose is not specific to the situation, such as Microsoft Office Excel or Google Sheets. But while it serves the purpose, on a large scale, controlling input and output can get more complicated and figuring out what is missing and what is not is hard. This paper aims to develop a web application that allows the creation of stock lists where the user can add products and control its quantity, even being able to share these lists with other people. Additionally, it is possible to create specific configurations to place products in a separate shopping list when the quantity is less than a pre-registered value. In addition, the application will be a PWA, which allows it to be accessed both on computers and mobile devices without the need to download the program.

Keywords: PWA, Inventory control, Web Application.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PWA	Progressive Web Application
SPA	Single Page Application
DOM	Modelo de Objeto de Documento
ES	ECMAScript
NPM	Node Package Manager
JSON	JavaScript Object Notation
W3C	World Wide Web Convention
ERP	Enterprise Resource Planning
SaaS	Software as a Service
CI/CD	Continuous Integration / Continuous Delivery

SUMÁRIO

1. INTRODUÇÃO

Há muitos anos organização e logística são sinônimos de lucro quando se trata de controle de estoque. Um controle alto significa que uma empresa consegue usar seu estoque com eficiência e economizar o máximo possível na hora de repor seus produtos.

A logística surgiu no meio militar e era comumente utilizada para armazenar os armamentos e mantimentos das tropas dos exércitos, bem como para transportar soldados de um território para outro[1]. Após isso, a técnica começou a ser usada por empresas para manter o controle de sua própria estrutura.

Entre os benefícios de um bom controle de estoque e uma boa logística, estão o aumento da produtividade, a redução do desperdício, a diminuição dos custos (por consequência), entre outros.

O aumento da produtividade quando se usa um bom sistema de controle de estoque se deve à automação de atividades cotidianas. Consequentemente, os erros humanos e a necessidade de retrabalho diminuem.

Além disso, o controle dos itens no estoque permite descobrir aqueles produtos que possuem uma baixa rotatividade e assim diminuir o desperdício comprando algo que não cria lucro.

1.1. PROBLEMA

Apesar da implementação de um sistema de controle de estoque poder variar em complexidade desde uma tabela no excel até um sistema específico para empresas, normalmente o acesso a esses sistemas pode ser complicado.

Uma tabela simples pode ser boa para mostrar todos os produtos de um estoque, mas peca quando o assunto seria a automação de atividades repetitivas e controle de entradas e saídas.

Já os sistemas mais complexos com propósito específico de controle de estoque normalmente são específicos para empresas de médio/grande porte e estão atrás de barreiras monetárias que podem não ser alcançadas por pequenas empresas. Além

disso, caso uma pessoa física queira controlar apenas os produtos que possui na sua casa, esses sistemas podem ser muito mais que o necessário para suprir suas necessidades.

1.2. MOTIVAÇÃO

Por esses motivos esse projeto possui o objetivo de implementar um sistema simples o bastante para que possa ser usado tanto por empresas, de pequeno a grande porte quanto por indivíduos que queriam apenas organizar sua casa. Mas que também tenha um nível de complexidade que permita um alto nível de controle e gestão do estoque. Nesse sentido, é possível mencionar o cadastro de regras de negócio que, no sistema, permitirá a geração de listas de compras com os produtos que estiverem adequados às regras criadas. Ou também, a possibilidade de criar listas compartilhadas com outras pessoas para permitir atualizações conjuntas nas listas.

1.3. JUSTIFICATIVA

O desenvolvimento de uma aplicação de gestão de estoque permite estender as opções de alguém por organização. Disponibilizando on-line como um PWA (Progressive Web App), seria possível qualquer um com acesso à internet usar o programa e começar a controlar seus próprios produtos. Junto com a possibilidade de usar o aplicativo PWA no celular, aumentaria a mobilidade e o gerenciamento em tempo real. E com o armazenamento dos dados nas nuvens tornaria possível compartilhar os mesmos dados entre dispositivos sem esforço. Aproveitando do programa ser um PWA, também é possível permitir ao usuário a alteração dos dados enquanto o celular estiver offline, guardando os dados em cache e atualizando no banco de dados quando o dispositivo voltasse a possuir conexão com a internet.

1.4. OBJETIVOS

1.4.1. Objetivos gerais

O objetivo geral do projeto é criar uma aplicação web funcional que consiga fazer a conexão com o banco de dados e o manuseio dos dados, estando online, e que possa ser instalada como PWA em qualquer dispositivo móvel com acesso à internet. Também permitindo o uso posterior offline, guardando os dados em cache para serem atualizados ao se reconectar.

1.4.2. Objetivos específicos

- Criar uma aplicação web com front-end PWA.
- Conectar e gerenciar o banco de dados proporcionado pelo firebase (firestore).
- Permitir o controle dos dados tanto online quanto offline.
- Implementar uma política de controle de mudanças para evitar conflitos de dados.
- Hospedar a aplicação em algum serviço compatível com acesso https.

1.5. MÉTODO DE PESQUISA

1.5.1. Natureza: Aplicada

Visando criar conhecimento a fim de minimizar um problema existente. No caso desse trabalho, facilitar o controle de produtos em um estoque de maneira simples, utilizando como meio de implementação uma aplicativo web.

1.5.2. Abordagem: Qualitativo

As soluções já utilizadas em aplicações similares a esse já são uma fonte de dados para o problema que está sendo abordado, não necessitando de cálculos específicos para capturar as informações necessárias.

1.5.3. Objetivo: exploratório

Almejando a compreensão do problema através de trabalhos relacionados eliciados nesse trabalho usando estudo de soluções passadas para problemas similares ao que está sendo abordado.

1.5.4. Procedimentos técnicos: Bibliográficos, Estudo de Caso

O desenvolvimento de aplicações que auxiliem no controle de estoque para computadores e dispositivos mobile já foi uma ideia previamente explorada em estudos passados. Busca-se replicar a mesma ideia disponível para navegadores e fornecendo a possibilidade de uso offline através de funcionalidades disponíveis de um PWA.

1.6. ORGANIZAÇÃO DO TEXTO

Dada uma introdução ao tema, os problemas encontrados, a motivação que leva a execução deste projeto, os objetivos do projeto e a metodologia adotada para atingi-los, as próximas seções deste documento decorrem sobre a fundamentação teórica e tecnológica do mesmo, o desenvolvimento da solução para o problema identificado, os resultados obtidos com a proposta e por fim uma análise destes resultados.

No capítulo 2, são apresentados os principais conceitos teóricos e tecnológicos relacionados ao tema central do projeto, discorrendo desde dados históricos a tecnologias em específico. Tais conceitos foram utilizados tanto para a fundamentação do desenvolvimento e implementação do projeto, quanto para o entendimento da solução proposta.

O capítulo 3 apresenta a revisão sistemática da literatura, que teve por objetivo identificar soluções já presentes no mercado. Neste é descrito o protocolo de revisão utilizado, contendo as fontes de pesquisa e termos de busca utilizados, além de apresentar os principais resultados encontrados e uma conclusão a respeito dos mesmos.

A solução proposta é descrita em detalhes no capítulo 4, e nele é apresentada a arquitetura e as funcionalidades da solução, seus requisitos e como está sendo implementada, além disso, descreve todo o desenvolvimento da aplicação. O último capítulo é a conclusão, sumarizando tudo o que foi e o que será feito no futuro.

2. FUNDAMENTAÇÃO TEÓRICA E TECNOLÓGICA

Neste capítulo é apresentada uma fundamentação dos conceitos teóricos e tecnológicos fundamentais para o entendimento da proposta estabelecida neste documento, e que foram essenciais para o desenvolvimento do projeto proposto.

2.1. JavaScript

O JavaScript é uma linguagem de programação estruturada - interpretada sequencialmente e focada na decisão e iteração - com script em alto nível, possui tipagem dinâmica e suporta diferentes paradigmas, como imperativo, funcional, orientação a objetos e outros). Atualmente a linguagem possui mais de um milhão de bibliotecas disponíveis para uso. Isso se deve ao fato da linguagem ser a mais popular do mundo, com mais de 12 milhões de usuários ativos em todo o mundo[10].

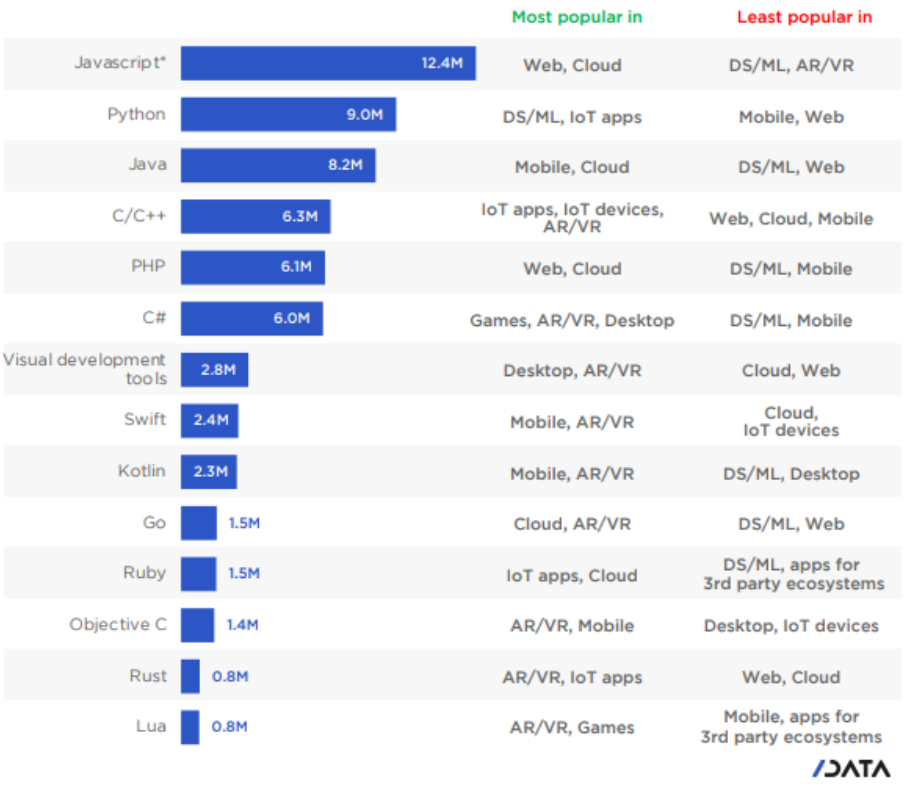


Figura 2.1 - Tabela de popularidade entre linguagens

Fonte: [10]

A popularidade se deve ao fato de ser uma linguagem leve e simples de usar. Além disso, junto com HTML e CSS, o JavaScript é uma das principais tecnologias da World Wide Web. Enquanto o HTML é responsável pelo esqueleto de uma aplicação, o CSS dos estilos, o JavaScript permite a criação de diferentes comportamentos que transformam as páginas da Web em dinâmicas.

A linguagem utiliza o padrão ECMAScript, e desde 2012, todos os navegadores modernos possuem acesso total ao ECMAScript 5.1. Porém, em 2015, foi lançado o padrão ES6, e a partir daí, começou a ser publicado anualmente. Apesar das atualizações do ECMAScript serem anuais, nem todos os navegadores acompanham as mudanças, para isso é usado o Babel, um compilador de Javascript capaz de transformar um código usando a versão mais recente da linguagem em uma versão que os navegadores possam entender.

```
var person = { name: "Susan", happy: true };
person.happy ?
    `I'm ${person?.name} and I'm happy` :
    `I'm ${person?.name} and I'm sad`;
```

Código 2.1 - Exemplo de código usando a versão mais atual do JavaScript
Fonte: O Autor (2022)

```
var person = {
    name: "Susan",
    happy: true
};
person.happy ? “I'm ”  + (person == null ? void 0 : person.name) +
“ I'm happy” : “I'm ”  + (person == null ? void 0 : person.name) +
“ I'm sad”;
```

Código 2.2 - Retorno do compilador Babel
Fonte: [14]

Outro modo muito comum de se usar o JavaScript é com base nos protótipos. Protótipos são usados para definir herança entre objetos. A possibilidade de objetos de protótipo possuírem protótipos permite a criação de uma cadeia de protótipos. Além disso, usando a propriedade `.prototype`, é possível adicionar métodos

customizados a classes que já existem, caso haja a necessidade de um comportamento personalizado a cada situação. Por exemplo:

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}

const person1 = new Person('Peter', 23);
Person.prototype.printPerson = function() {
      console.log(this.name, this.age);
}
```

Código 2.3 - Exemplo de uso de protótipos

Fonte: O Autor (2022)

Como é possível ver no Código 2.3, ocorre a criação de um protótipo `Person` (equivalente ao que seria uma classe), e após isso é criada uma instância `person1` que possui como protótipo a própria função Person, e logo depois é adicionada a função `printPerson` ao protótipo de `Person`, assim, caso seja chamado `person1.printPerson()`, o navegador primeiro irá checar se `person1` possui o valor `printPerson`, então procurará em Person e aí sim irá executar o método encontrado.

Figura 2.2 - Diagrama de herança de protótipo

Fonte: [9]

2.2. Aplicações Web

Aplicações Web são sistemas ou programas rodando diretamente no navegador. Elas estão disponíveis ao usuário a qualquer momento e em qualquer lugar dependendo apenas da conexão com a internet e de um navegador Web, sem a necessidade de baixar nada a mais.

Essas aplicações possuem um componente muito importante, o protocolo HTTP. Com ele são feitas as requisições entre cliente e servidor e obtidos os dados que serão mostrados na tela.

Apesar de parecidos, existe uma principal diferença entre sites e aplicações web: enquanto sites são estáticos, sem a possibilidade de adicionar ou remover conteúdos, aplicações web são interativas, permitindo ao usuário alterar valores, criar e remover dados.

Entre as vantagens das aplicações web estão:

- **Usabilidade simples:** devido a não necessidade de baixar outros programas ou arquivos para o uso das aplicações, torna-se simples e rápido ao usuário o acesso ao programa;
- **Atualização dinâmica:** por estar armazenada em um servidor, usuários conseguem desfrutar da última versão do programa sem a necessidade de instalar ou executar nenhum outro processo;
- **Compatibilidade:** as aplicações web não possuem problemas de instabilidade ou incompatibilidade entre computadores. Se uma aplicação roda sem problemas em um navegador específico, qualquer outro navegador com a mesma versão, não importando o *hardware* do cliente, vai conseguir executar o problema sem nenhum problema.

Normalmente aplicações web são divididas em duas partes, o front-end e o back-end. O primeiro se refere a interface e aquilo que o usuário vê e que está acessível no browser, enquanto o segundo ao que acontece “por de trás das câmeras” e dá suporte a interface, com o código presente apenas no servidor. As principais linguagens usadas no front-end são o HTML e CSS (geralmente JavaScript), o back-end também pode usar o JavaScript, mas está aberto a outras linguagens, como Python, PHP, etc. Atualmente existem diversos frameworks que

permitem a criação de aplicações web juntando HTML aos scripts em um. Como o Django para Python, ou o React Js ou Vue Js que utilizam o Javascript.

2.2.1. PWA

Um PWA é um tipo de aplicação web que age como um híbrido entre web e mobile. Em um navegador um PWA funciona como qualquer aplicação web tradicional, porém quando acessado pelo celular, ou até possível através de alguns navegadores para computador, é perguntado se o usuário deseja adicionar o site na tela inicial. Ao fazer isso, o atalho para o site se torna um aplicativo e começa a agir como um.

Um PWA seria um meio termo entre uma aplicação web padrão e um aplicativo nativo. Isso permite que usuários possam acessar aplicações normalmente desenvolvidas para a web em seus celulares, sentindo como se estivessem usando um aplicativo, com uma vantagem de não ter que baixar muitos dados como de aplicativos da Play Store, dado que PWAs geralmente são leves, por volta de 1MB.

A Tabela 2.1 mostra uma comparação de PWAs com aplicativos nativos e aplicações web padrão. Uma parte importante é que além das vantagens, o desenvolvedor deve criar uma aplicação responsiva que consiga se adaptar tanto a tela de um computador quanto de celular, levando em conta tamanho de fonte, dos containers, formato para que quando exista a troca de dispositivo não haja um desconforto pela parte do usuário.

	Aplicação Nativa	**PWA**	**Aplicação Web Padrão**
Instalação	Necessário baixar da Play Store/Apple Store	Necessário apenas clicar para adicionar a tela inicial (Android)	Instalação não requerida
Atualizações	Envio para loja de aplicativos e download pelo usuário necessários	Atualizações são instantâneas	Atualizações são instantâneas

Tamanho	Maioria são pesados. Podem levar algum tempo para baixar nos dispositivos	Leve e rápido	Leve e rápido
Acesso Offline	Disponível	Necessita o acesso online apenas na primeira vez, depois o conteúdo salvo em cache é acessado	Não requerido
Experiência do Usuário	Excelente quando a aplicação é bem projetada	Confuso devido aos menus duplos (menu do aplicativo e do navegador)	O mesmo de PWA
Notificações	Sim	Sim (apenas Android)	Sim (Depende de serviços de terceiros)
Descoberta	Não é boa. Necessário trabalhar na otimização da loja de aplicativos	Boa. Para aparecer nos resultados de pesquisa é necessário otimizar para o SEO	Não requerido

Tabela 2.1 - Comparação entre PWA, Aplicações nativas e Aplicações Web

Fonte: [28] (Traduzido)

Em relação ao programa em si, muitos dos Frameworks para desenvolvimento web apresentam suporte a PWAs. O Vue Js por exemplo, durante a criação da aplicação, oferece como um “plugin” adicional o PWA, marcando essa opção, já são criados e programados automaticamente o *Manifest* e o *Service Worker*, permitindo ao usuário aproveitar tudo que o PWA tem a oferecer (apesar disso algumas funções, como a capacidade de desenvolver aplicações offline first - programas que armazenam alterações enquanto offline e atualizam o banco de dados quando se reconecta - precisam de módulos à parte que podem ser instalados com o NPM). Em relação aos arquivos citados, o primeiro representa todas as informações que o aplicativo terá, nome, ícone, cores de fundo, splash screen entre outras configurações. Já o segundo é o responsável por guardar os dados do aplicativo na cache quando ele é

aberto pela primeira vez, permitindo o acesso offline. Com isso feito a aplicação criada já pode ser executada como um PWA automaticamente.

2.3. Node.js

O Node.js foi criado por Ryan Dahl e lançado em 27 de maio de 2009. Inspirado pela inabilidade do navegador de saber que um arquivo havia sido carregado sem ter que consultar o servidor web percebida após observar a barra de upload de arquivos no Flickr, Ryan criou um software de código aberto que permitia a execução de códigos JavaScript fora do navegador web.

Uma característica importante do Node é sua capacidade de, diferente de outras tecnologias como Java e C#, executar o código JavaScript em uma única thread sem usar muitos recursos computacionais nem fila de espera. Entre as vantagens do uso do Node.js estão:

- **Escalabilidade:** O Node possui alta escalabilidade devido ao grande potencial de suportar grandes números de conexões simultâneas em relação aos servidores tradicionais;
- **Flexibilidade:** O NPM (Node Package Manager) provê ao usuário do Node um gerenciador com uma quantidade enorme e constantemente crescente de pacotes que podem ser instalados a mando do programador. Um exemplo de pacote é o Express.js, um framework para desenvolvimento Web, ou axios, uma biblioteca que permite a execução de requisições Web com muita facilidade;
- **Leveza:** Um ambiente Node.js não exige muitos recursos. Ainda é possível integrar a aplicação a um container como Docker para aproveitar todas as vantagens da conteinerização, principalmente o aumento da eficiência em ambientes escaláveis;
- **Suporte:** Empresas como AWS e Google Cloud provêm suporte nativo ao Node.js
- **Produtividade:** Como já citado anteriormente, o NPM fornece pacotes para diversas funcionalidades diferentes e distribuídos gratuitamente. Além disso

com o Node, a mesma linguagem usada no frontend é usada no backend, aumentando o fator de reutilização de código.

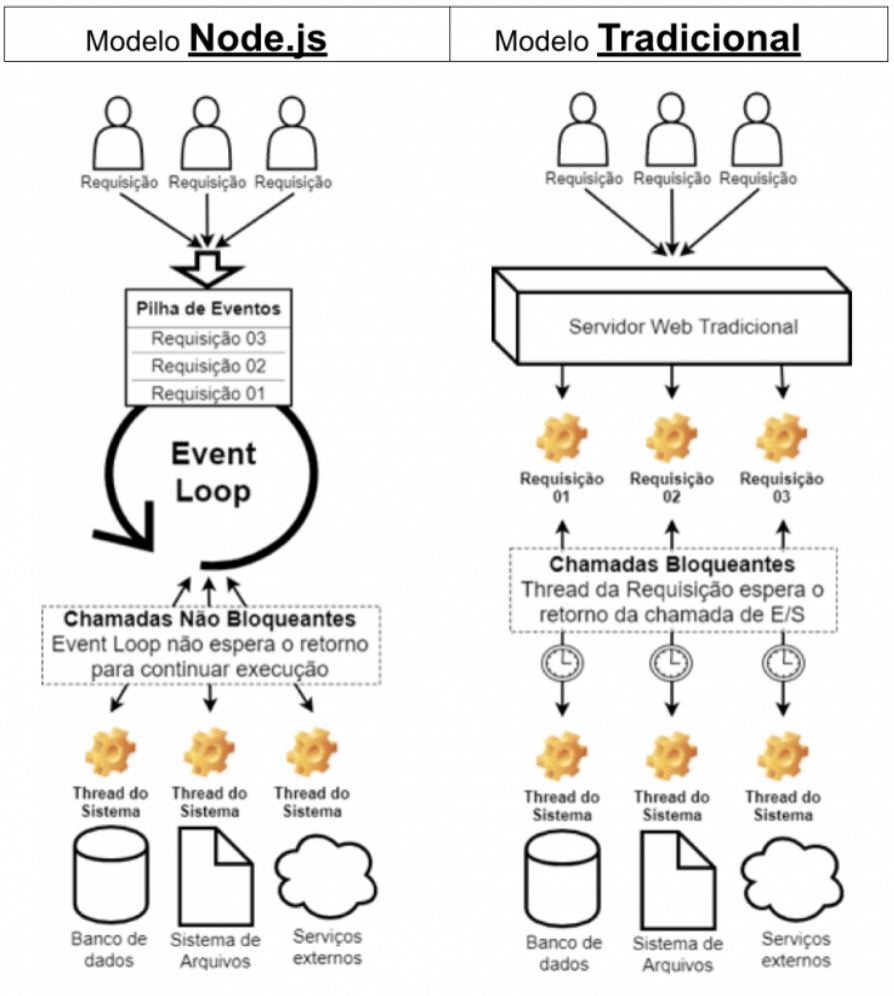


Figura 2.3 - Comparação de funcionamento entre o servidor Node.js e Tradicional

Fonte: [15]

Os casos mais comuns do uso do Node são em aplicações em tempo real, que dependem principalmente da execução de requisições com o servidor. Outros exemplos são APIs que usam noSQL como base de dados; por ser desenvolvido em JavaScript, é mais simples obter os dados desse tipo de banco por serem baseados em JSON, logo não necessitando a conversão ou tratamento quando são requisitados.

O Código 2.4 é um exemplo de aplicação Node que ouve as conexões na porta 3000 e retorna “Hello World”. A cada conexão o método é chamado, mas o programa entra em modo sleep quando não há nada acontecendo.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Código 2.4 - Exemplo de aplicação Node.js

Fonte: [16]

2.4. DOM

O DOM ou Modelo de Documento por Objetos é uma convenção usada na representação e interação com objetos escritos em HTML, XHTML e XML. Fiscalizada pela W3C, esse modelo permite que scripts, normalmente escritos em JavaScript alterem a estrutura de uma página Web sem alterar diretamente o código HTML e CSS.

Acessado no JavaScript a partir do objeto `document`, é possível obter informações e alterar diferentes tipos de dados da árvore DOM de uma página. A figura 2.3.1 mostra um exemplo de árvore DOM, e se caso fosse necessário obter todos os elementos `<h1>`, o método `document.getElementsByTagName("h1")` retornaria um array com todos os elementos cuja tag corresponde ao parâmetro.

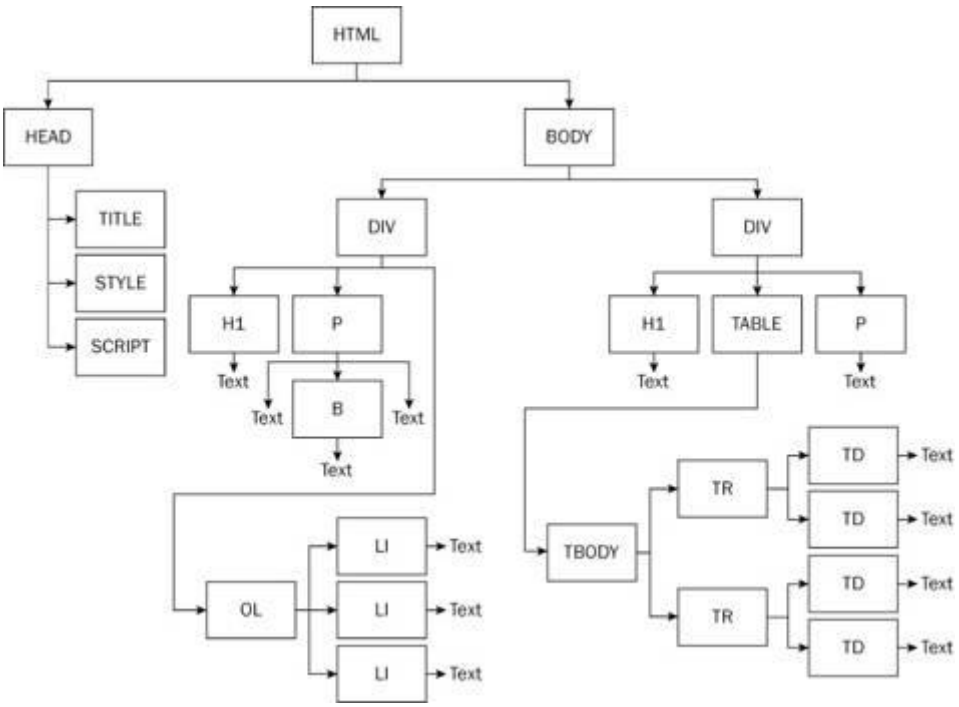


Figura 2.4 - Exemplo de árvore DOM

Fonte: [18]

Além do comando mostrado anteriormente, a Tabela 2.2 mostra outros métodos e propriedades que o objeto `document` possui que também podem ser usados para obter e alterar dados da página.

Propriedade	Descrição
`documentElement`	Retorna o elemento raiz <html> de um documento HTML
`getElementById`	Busca um elemento da página Web com o uso do atributo id do elemento.
`createElement`	Cria um novo elemento na página
`createAttribute`	Cria um novo atributo na página
`createTextNode`	Cria um nodo de texto na página
`getElementsByTagName`	Retorna um array dos elementos com a mesma tag
`appendChild`	Insere um novo elemento filho
`removeChild`	Remove um elemento filho

`parentNode`	Retorna o nodo pai de um nodo
`getElementsByClassName`	Retorna um array dos elementos com a mesma classe de estilo CSS

Tabela 2.2 - Algumas propriedades do objeto `document`

Fonte: [18]

Vale ressaltar que as propriedades que retornam um elemento de dentro do DOM, retornam protótipos do mesmo, ou seja, é possível usar os mesmos métodos e atributos citados para alterar e adicionar elementos aninhados no elemento anteriormente selecionado. Além disso, elementos específicos podem adicionar propriedades que funcionam apenas naquele tipo de elemento. Por exemplo, elementos do tipo `HTMLMediaElement`, como vídeos, possuem a propriedade `playbackRate`, a qual define a velocidade que a mídia é exibida, podendo ser alterada para valores mais altos que 1 para aumentar a velocidade de um vídeo, ou menores para obter o efeito contrário.

2.5. Vue.js

Quando se refere ao desenvolvimento de aplicações para web, geralmente o foco principal se volta para um projeto cujo esqueleto é feito em HTML, a aparência gerenciada pelo CSS e o comportamento do site com JavaScript. Porém, já existem frameworks que possibilitam a criação de aplicativos mais complexos de forma mais simples. É aí que o Vue.js entra. Desenvolvido por Evan You, o Vue é um framework progressivo baseado em componentes feito especialmente para criação de SPAs (mas permite a criação de aplicações com mais de uma página com auxílio de uma biblioteca fornecida oficialmente pela própria Vue, o Vue Router).

O framework permite o desenvolvimento em javascript ou typescript e em um único arquivo consegue juntar o HTML, CSS e os scripts, permitindo a interação entre os três aspectos de maneira fácil.

Entre outras funcionalidades do Vue.js estão:

- **Reatividade:** No Vue, cada componente consegue controlar suas próprias propriedades e dependências e assim, consegue otimizar a re-renderização

para ocorrer apenas nos componentes que realmente precisam ser atualizados;

- **Modelos:** O Vue usa uma sintaxe de modelo baseada em HTML que permite vincular o DOM renderizado aos dados contidos na instância subjacente do Vue[3]. Quando os modelos são compilados, eles são transformados em funções de renderização em um DOM virtual, e assim, consegue calcular o número mínimo de alterações a serem feitas e alterar manualmente no DOM verdadeiro quando o aplicativo é alterado.
- **Modularização:** Outro benefício importante do Vue é a possibilidade alta de modularização. Além da própria desenvolvedora, a comunidade também pode fornecer módulos que podem ser adicionados no projeto que desempenham funções diferentes. Entre elas existem oficiais como Vuex, um módulo para gerenciamento centralizado de estado, o Vue Router, que como citado anteriormente permite o encaminhamento de páginas e remove a limitação de um SPA; E os não oficiais, um exemplo de módulo não oficial que é muito usado é o Vuetify, que fornece uma variedade de componentes extremamente customizáveis prontas para uso.

O Código 2.5 demonstra o corpo de um elemento feito com Vue.js. Um arquivo .vue é dividido em três partes principais, a primeira é o `<template>`, a parte do código responsável pela aparência da página, escrita com tags HTML e permitindo o uso de outros componentes criados com o Vue.js. Nessa parte também é possível inserir códigos em Javascript, como é usado o message, presente no objeto data, presente no exemplo.

A segunda parte é o `<script>`, assim como num arquivo HTML normal, é a tag onde o código JavaScript é inserido; a diferença é que no Vue, o comportamento é definido exportando um objeto com propriedades que mudam a página de jeitos diferentes. Alguns exemplos são:

- **name:** Define o nome para o componente sendo desenvolvido;
- **data:** Define estados para o componente. Eles podem ser acessados na área do <template> diretamente ou dentro do <script> usando o objeto this;
- **computed:** Um diferencial do Vue.js; Assim como o data sempre retorna um valor de estado, mas sendo possível adicionar uma lógica que altera o resultado final a partir de outros dados;

- **methods:** Define métodos que podem ser usados dentro do componente;
- **watch:** Permite criar funções que são executadas quando a variável com o mesmo nome que a função criada é alterada.

Além dessas propriedades, é possível usar funções prontas que são executadas em momentos específicos do ciclo de vida do componente, como o `created`, que é executada quando o componente acabou de ser criado, ou o `beforeDestroy` que é o último método executado antes do componente ser destruído.

```
<template>
    <div class=”container”>
        {{ message }}
    </div>
</template>

<script>
export default {
  name: "Hello World",
  data: () => ({
      message: "Hello World"
  }),
};
</script>

<style scoped>
.container {
    background-color: red
}
</style>
```

Código 2.5 - Exemplo de um component Vue

Fonte: O Autor (2022)

2.6. Firebase/Firestore

Firebase é uma plataforma desenvolvida pelo Google para auxiliar na criação de aplicativos móveis e web. O serviço é inicialmente gratuito, mas com possibilidade de pagamento para melhorar o uso, com mais funcionalidades.

A plataforma fornece opções para autenticação, hosting, funções para os aplicativos, machine learning e, principalmente, um sistema de banco de dados, o Firestore.

O Firestore permite ao desenvolvedor criar um sistema de banco de dados não relacional - conhecidos documentos, armazenados em coleções - e acessar esses dados facilmente a partir do código usando o próprio módulo da plataforma.

A Firebase foi originada da startup Envolve, criada por James Tamplin e Andrew Lee em 2011. Inicialmente seu objetivo era providenciar uma API para desenvolvedores para possibilitar a integração da funcionalidade de chat nos websites. Seu primeiro produto foi um banco de dados em tempo real armazenado na nuvem. Em 2014, Firebase foi posicionado como *Mobile Backend as a Service* com o lançamento do Firebase Hosting e Firebase Authentication. E no mesmo ano, a empresa foi comprada pela Google, e junto com a aquisição da Divshot, uma plataforma de hospedagem web HTML5 foi lançada.

Então, em 2017, a Cloud Firestore foi lançada como sucessora do banco de dados em tempo real original da empresa.

Dentro do Firestore, os dados são chamados de documentos e são armazenados em coleções. Além disso, o sistema permite que os documentos armazenem coleções dentro de si, permitindo uma relação de “parentesco” entre elas.

2.7. Sistemas ERP

Sistemas ERP ou Sistema de Gestão Integrado (da sigla em inglês, *Enterprise Resource Planning*) são sistemas que abrangem diversas áreas para o controle e unificação de diferentes setores de empresas. Alguns tipos de sistemas ERP são:

- Controle de estoque;
- Controle financeiro;
- Controle de vendas;
- Fluxo de caixa;
- Emissão de boletos;
- Emissão de nota fiscal;
- Emissão de relatórios.

Esses sistemas surgiram em 1950, quando foi usado o primeiro *mainstream* para automatizar o controle de estoques. Três décadas depois, as redes de computadores possibilitaram o uso de servidores, reduzindo custos em relação aos *mainframes*, e estreitando a comunicação entre departamentos. A partir de então, os sistemas ERP evoluíram se adaptando a novas tecnologias como machine learning, internet das coisas, sempre em constante melhora.

As principais vantagens para o uso desses sistemas envolvem a automação e otimização dos processos, redução do risco de erros e o maior controle e conhecimento sobre o funcionamento da empresa.

Esse projeto foi desenvolvido a partir dos temas abordados nessa seção. O Vue Js foi usado devido a sua simplicidade, popularidade e familiaridade do uso. Também foi decidido criar um PWA para permitir a facilidade do usuário de usar não importando o dispositivo; sempre focando principalmente no desenvolvimento de um aplicativo leve e simples de se usar. Pelo mesmo motivo foi escolhido o uso do Firebase no sistema. Ele é uma plataforma fácil de se desenvolver com, e também prática pois já supre a necessidade de dois sistemas diferentes, banco de dados e autenticação.

3. ESTADO DA ARTE

3.1. SOLUÇÕES EXISTENTES

Para identificar soluções existentes e semelhantes aos objetivos desse projeto, foi realizada uma revisão sistemática da literatura tomando como base o método descrito por Kitchenham [21].

3.1.1. Protocolo de revisão

O objetivo da revisão sistemática era analisar o estado atual do mercado em relação ao desenvolvimento de aplicações que, assim como o objetivo desse trabalho, pretendem fornecer serviços de gerenciamento de estoque ou que usam tecnologias semelhantes às propostas para esse projeto.

Para isso, foram usadas como fontes para pesquisa o Google e o Google Scholar dentro do período de dezembro de 2021 até fevereiro de 2022. Os termos de buscas usados e combinados são apresentados na Tabela 3.1.

Termo de busca	**Termo relacionado**	**Tradução (Inglês)**
PWA	Progressive Web Application	Não se Aplica
Software	Código, Aplicação, Programa	Não se Aplica
Gestão	Gerenciamento	Management
Estoque	Inventário	Stock/Inventory

Tabela 3.1 - Termos de busca utilizados

Fonte: O Autor (2022)

3.1.2. Detalhamento das soluções existentes

A seguir é apresentado um detalhamento de algumas das mais relevantes soluções existentes encontradas durante a revisão sistemática descrita na seção 3.1.1. Dos

resultados obtidos, foram separados dois sistemas que obtiveram destaque nas pesquisas.

3.1.2.1. Bling

O Bling é uma ERP, ou sistema de gestão integrado, que dentre suas diversas funcionalidades possui o módulo de gestão de estoque. A plataforma é um SaaS desenvolvido em Bento Gonçalves (RS) e já presta serviços para milhares de clientes. O uso do software é limitado a um sistema de planos com mensalidades que começam em 25 reais com funções limitadas e crescem à medida que mais funcionalidades são adicionadas até o maior plano de 100 reais ao mês. Apesar disso, pessoas físicas podem usar o aplicativo de graça limitados a 3mb de uso de dados.

Dentre os pontos positivos estão a facilidade de cadastro e uso, a existência de uma aplicação para sistemas móveis e a interface clara e intuitiva. E como já citado anteriormente o principal ponto negativo seria a limitação do sistema aos planos mais baratos.

O plano grátis pode ser bom para alguém que quer gerenciar um estoque para um negócio pequeno, mas à medida que o negócio cresce os planos também devem aumentar.

Outro ponto que pode ser tanto positivo como negativo é o detalhamento nas telas do sistemas. Para negócios maiores o detalhamento pode ajudar no controle, porém caso alguém queira controlar os produtos que possui, passar por tantas telas de cadastro, por exemplo, com tantas informações pode ser cansativo e uma perda de tempo.

3.1.2.2. Odoo

Assim como o Bling, o Odoo é um sistema ERP, porém com o diferencial de ser um software de código aberto. Desenvolvido sob o modelo MVC, a plataforma apresenta diversos módulos grátis para qualquer um usar, porém apenas isolados. Caso o usuário queira aproveitar mais de um módulo integrado e mais algumas funcionalidades, como o uso de aplicativo para celulares, o plano passa a ser pago.

Além disso, o site conta com uma biblioteca enorme de documentações em seu site para auxiliar qualquer um que queira aproveitar e desenvolver usando a plataforma. Também é fornecida uma API que permite o desenvolvedor de obter diversas informações e acessar os dados de sua conta através de uma ligação por chave de API.

A arquitetura do sistema se baseia em três camadas, a base de dados armazenada no PostgreSQL, o servidor escrito em Python e o cliente (dividido em três tipos: GTK+, web, e Qt) também em Python, sendo o cliente GTK+ é baseado na plataforma PyGTK e o último indisponível desde 2009. A Figura 3.1 demonstra um esquema da arquitetura descrita.

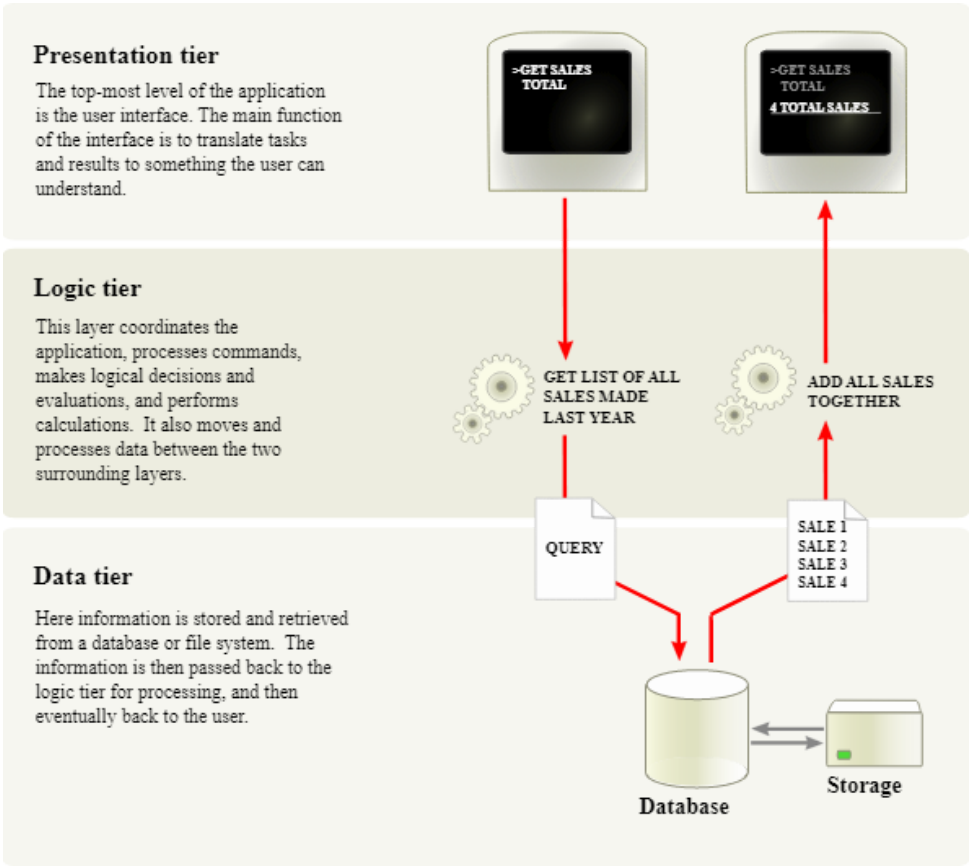


Figura 3.1 - Esquema da arquitetura de três níveis

Fonte: [24]

3.1.2.3. Zoho Inventory

Assim como os ERPs citados anteriormente, o Zoho Inventory é um sistema pago focado no gerenciamento das empresas em diversas áreas, estoque, vendas, pedidos, etc. Ele também é um serviço pago baseado em mensalidades, porém com a possibilidade de pagar valores separados para aumentar certas características como o número de usuários ou número de depósitos cadastrados no sistema.

A plataforma também oferece integração com diversos serviços. Para lojas online existe a integração com Etsy, um site de comércio famoso principalmente nos Estados Unidos; FedEx, para serviços de entrega e PayPal para pagamentos são alguns exemplos de integrações.

O site do Zoho também oferece aos usuários uma conta demo para o usuário explorar antes de tomar a decisão de aderir ou não ao serviço. Assim é possível ver como é o sistema por dentro, principalmente para esse projeto, a parte de estoque. A tela de cadastro de itens é relativamente complexa; para fazer um cadastro de item é preciso informar códigos, dimensões, tamanho, custo, e outros. Apesar de alguns não serem campos obrigatórios, o usuário ainda tem esses campos obstruindo uma visão otimizada o que gera uma perda de tempo na hora do cadastro.

3.2. Análise Comparativa

Analisando as soluções existentes apresentadas é possível notar que já existem muitas soluções para o assunto apresentado neste projeto. Assim, o objetivo desse trabalho era criar uma aplicação semelhante a essas que consiga fornecer um controle parecido porém que não possua tantos campos desnecessários e que seja disponibilizada gratuitamente, sem os limites que os serviços anteriores propunham. Além de permitir aos usuários fácil acesso a aplicação sem a necessidade de baixar ou guardar dados desnecessários em seus dispositivos. Outro diferencial é a possibilidade do usuário alterar os dados de seus estoques enquanto offline (possível através do uso da cache do PWA), atualizando as modificações no banco de dados quando a conexão com a internet for estabelecida.

4. DESENVOLVIMENTO

Neste capítulo é descrita a proposta de solução para o desenvolvimento de uma aplicação de gerenciamento de estoque para Web. Nas seções a seguir serão apresentados os requisitos funcionais e não funcionais, os casos de uso e as tecnologias utilizadas para o desenvolvimento da plataforma.

4.1. SOLUÇÃO PROPOSTA

A aplicação proposta como solução será um aplicativo web PWA desenvolvido em Vue.js. Com nome de iStash, ela será um aplicativo onde usuários poderão criar estoques - ou “stashes”, do substantivo em inglês, que significa um suprimento, ou estoque de algo - onde será possível adicionar produtos e controlar a quantidade de cada um e criar uma lista de compra com os produtos que estiverem em baixa quantidade.

4.1.1. Requisitos do sistema

Para o projeto, foram definidos os requisitos funcionais e não funcionais mínimos descritos na tabela 4.1 para a implementação da solução proposta. Estes requisitos foram obtidos com base no estudo de soluções existentes apresentadas no capítulo 3 e no problema descrito na seção 1.1.

Código	**Descrição**
RF01	A ferramenta deve permitir o início de sessão pelos usuários (Login e Logout)
RF02	A ferramenta deve permitir o cadastro (Inclusão, alteração e exclusão) de usuários no sistema pelos próprios usuários
RF03	A ferramenta deve permitir o cadastro (Inclusão, alteração e exclusão) de estoques de mercadorias pelos usuários
RF04	A ferramenta deve permitir a criação de regras para avisos em

	relação a quantidade de itens no estoque pelos usuários
RF05	A ferramenta deve permitir o acesso aos estoques pelos usuários
RF06	A ferramenta deve permitir o cadastro (Inclusão, alteração e exclusão) de produtos dentro dos estoques pelos usuários
RF07	A ferramenta deve permitir o compartilhamentos de estoques entre usuários e alteração sincronizada entre os envolvidos
RF08	A ferramenta deve permitir a criação e exportação de listas de compra baseadas nas regras criadas pelos usuários
RNF01	A ferramenta deve ser compatível com os navegadores populares recentes (Chrome, Firefox, etc.)
RNF02	A ferramenta deve ser desenvolvida com a linguagem javascript usando o framework Vue.js
RNF03	A aplicação deve ser desenvolvida dentro do tempo de duração das disciplinas de TCC I e II
RNF04	O sistema deve funcionar corretamento tanto online quanto offline
RNF05	O sistema deve ser conectado com o sistema do Firebase, autenticação e banco de dados (Firestore)

Tabela 4.1 - Requisitos funcionais e não funcionais levantados

Fonte: O Autor (2022)

4.1.2. Casos de Uso

Os casos de uso apresentados na tabela 4.2 descrevem as funcionalidades levantadas para implementação da solução e o escopo da mesma.

A figura 4.1 apresenta os casos de uso em forma de um diagrama de casos de uso UML. Nela, é possível visualizar que o projeto foi separado em módulos, referentes à autenticação e ao estoque. Além disso, existe a relação entre estoque e produtos, e produtos e estoque, pois cada estoque poderá ter um número variado de produtos, e um produto poderá ter uma ou nenhuma regra. Deve-se ressaltar que os casos de uso levantados foram elaborados levando em consideração requisitos apresentados na seção 4.1.1.

Código	Nome	Descrição
UC01	Realizar Login	Permite iniciar a sessão do sistema
UC02	Realizar Logout	Permite finalizar a sessão do sistema e desconectar o usuário
UC03	Cadastrar Usuário	Permite o registro de uma nova conta de usuário ao sistema para acesso à plataforma
UC04	Criar Estoque	Permite ao usuário criar um registro referente a lista de produtos e associar a sua conta
UC05	Editar Estoque	Permite alterar os dados referentes ao estoque
UC06	Remover Estoque	Permite remover um estoque associado a conta do usuário
UC07	Cadastrar Produto	Permite criar o registro de um produto e associar a um estoque
UC08	Editar Produto	Permite editar os dados de um produto
UC09	Remover Produto	Permite remover um produto associado a um estoque
UC10	Criar Regra	Permite o usuário criar um regra para um produto baseado na quantidade em que ele está presente em um estoque
UC11	Editar Regra	Permite alterar os valores de uma regra
UC12	Remover Regra	Permite remover uma regra
UC13	Solicitar Criação Lista de Compras	Permite o usuário solicitar ao sistema a criação de uma lista de compras baseada nas regras criadas pelo usuário
UC14	Compartilhar Estoque	Permite o usuário compartilhar um estoque com outro usuário
UC15	Exportar Lista de Compras	Permite o usuário exportar a lista de compras como um arquivo pdf

Tabela 4.2 - Casos de Uso

Fonte: O Autor (2022)

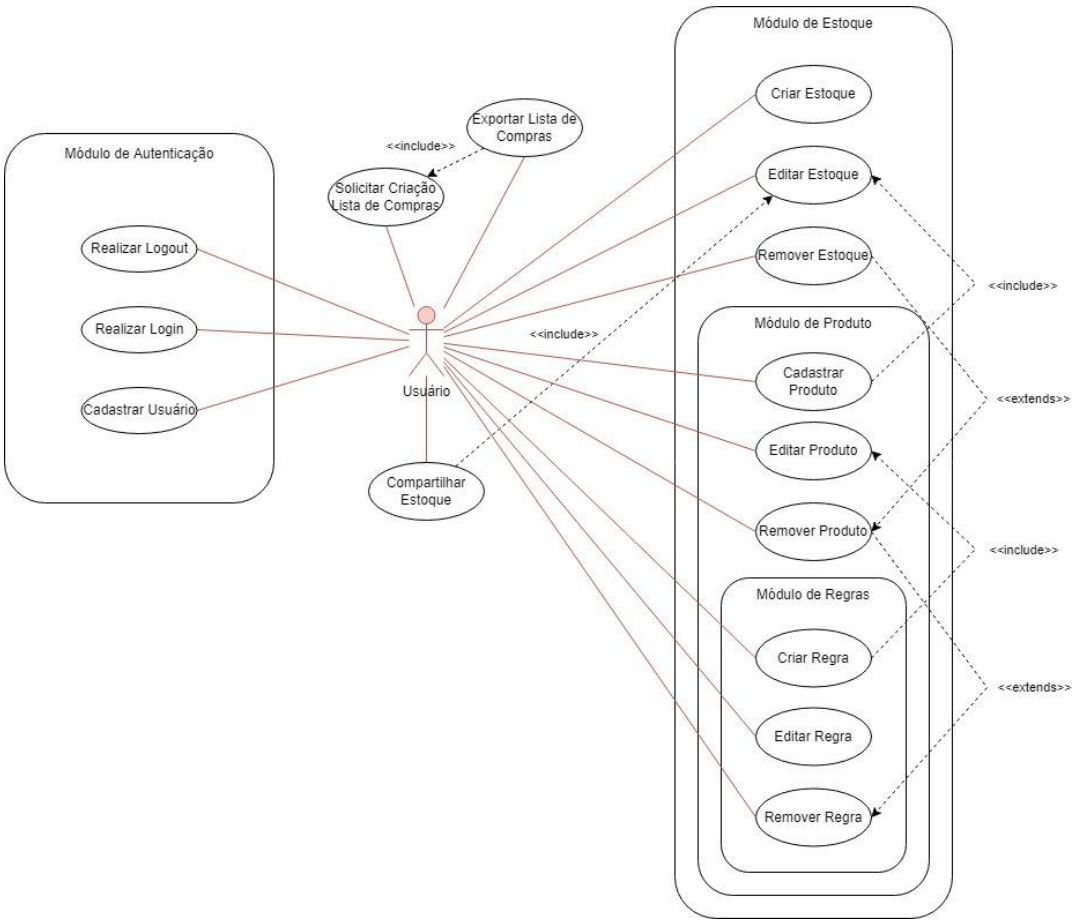


Figura 4.1 - Diagrama de Casos de Uso

Fonte: O Autor (2022)

4.2. ARQUITETURA DO SISTEMA

O modelo da arquitetura do sistema consiste da aplicação hospedada na web, o serviço de autenticação do Firebase e o banco de dados presente do Firestore. As requisições para o banco de dados são feitas diretamente pela aplicação usando os métodos fornecidos pelo próprio módulo do Firebase.

O banco de dados é responsável por armazenar todas as informações referentes aos usuários e seus estoques. Por ser usado um banco de dados não relacional, seria possível armazenar os estoques como uma coleção dentro do documento de cada usuário, mas devido ao UC16 foi decidido que o melhor é armazená-los em uma coleção separada e referenciar os usuários que possuem acesso pelo código deles.

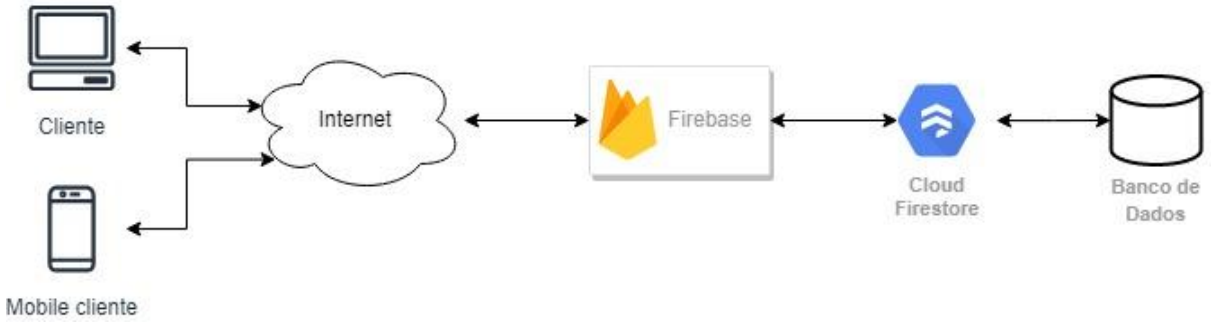


Figura 4.2 - Arquitetura do sistema proposto

Fonte: O Autor (2022)

4.2.1. Projeto de telas

Para o desenvolvimento das interfaces do projeto foram desenvolvidos protótipos das telas para então o desenvolvimento final no programa com HTML e CSS junto ao Vue.js.

4.2.1.1. Tela de Login

A primeira tela que o usuário presencia quando entra no sistema é a tela de login. Como é possível ver na figura 4.3, a tela de login é um simples formulário com usuário e senha. Não é mostrado no protótipo, mas também haverá hyperlinks para a tela de cadastro e para tela de “Esqueci minha senha”, ambas similares a essa.

Figura 4.3 - Protótipo da Tela de Login

Fonte: O Autor (2022)

4.2.1.2. Tela de Estoques

A tela de estoques é a principal tela do sistema, nela será possível consultar e criar novos estoques. A figura 4.4 mostra como ela deverá ser, uma lista de cada estoque criado pelo usuário. Também será possível ver se um estoque está compartilhado com alguém, ou se há produtos em falta com base em ícones e avisos dentro de cada item da lista.

Figura 4.4 - Protótipo da Tela de Estoques

Fonte: O Autor (2022)

Clicar em um item da lista levará o usuário para a tela de detalhes do estoque, onde haverá uma lista com os produtos cadastrados, e a possibilidade de cadastrar novos produtos.

4.2.1.3. Tela da Lista de Compras

A última tela importante do sistema é a lista de compras. A tela será uma lista com os produtos do estoque escolhido pelo usuário que estão abaixo da quantidade também cadastrada pelo usuário. Além disso, como é possível observar na figura 4.5, deverá haver um botão de exportar, que criará um pdf com o conteúdo da lista.

Figura 4.5 - Protótipo da Tela de Lista de Compras

Fonte: O Autor (2022)

Funcionalmente, o fato de ser um PWA tornará a aplicação mais leve, por não necessitar de muito espaço no dispositivo, devido a grande parte da aplicação estar na internet e não diretamente instalado no computador ou celular (outra vantagem, a mobilidade entre dispositivos). Outro ponto que favorece ao pouco uso de armazenamento é o banco de dados, que ao invés de ser armazenado no dispositivo, também está na nuvem. O uso de tecnologias offline first permitirá a usabilidade da aplicação enquanto não houver conexão com a internet, armazenando as alterações na cache. Apesar do Vue js não fornecer essa função diretamente, existem inúmeros módulos que permitem essas operações.

4.2.2. Modelagem de Classes

Para os dados da aplicação, foi decidido seguir uma modelagem de classes que centralizariam os métodos para que fosse possível de usá-los em diferentes partes do sistema. Com isso, foi desenvolvido o diagrama de classes mostrado na Figura 4.6.

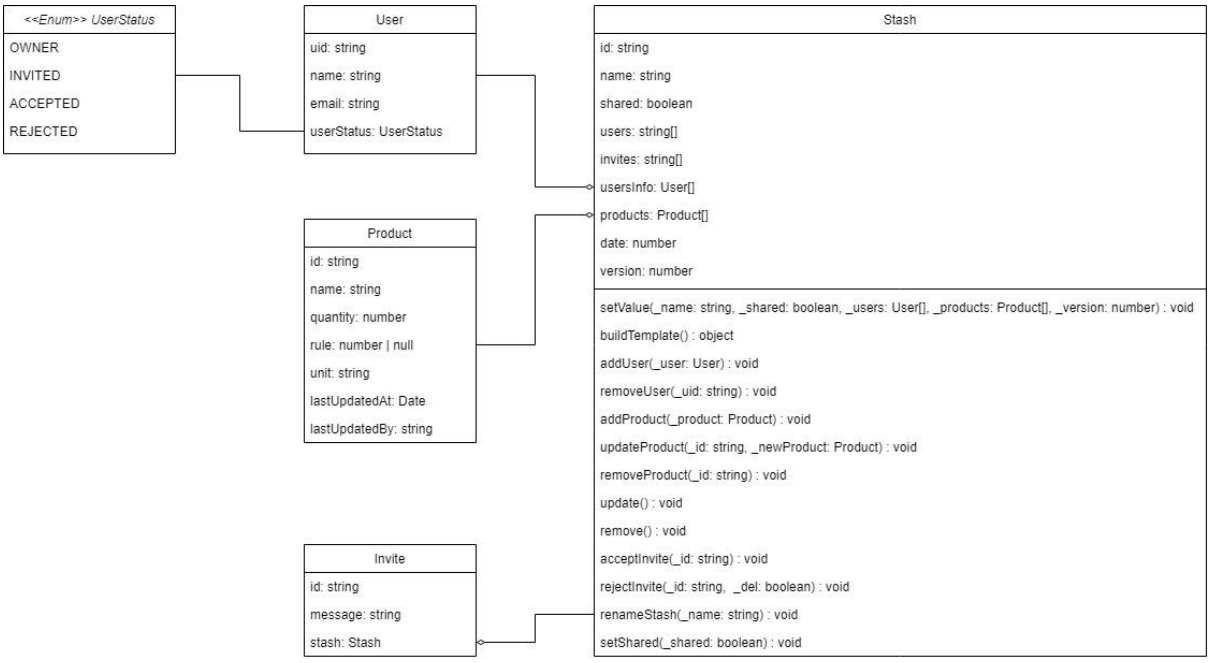


Figura 4.6 - Diagrama de classes

Fonte: O Autor (2022)

4.2.3. Desenvolvimento do aplicativo

Existem diversas maneiras de se criar um projeto em Vue, entre eles existe a criação manual a partir do comando `npm init` e instalação posterior dos pacotes desejados, ou também o uso de scripts prontos que a partir de informações dadas (como nome do projeto e até padrões pré-estabelecidos para criar o projeto já configurado com o que o usuário precisa para rodar uma aplicação.

Um exemplo desse último é o comando vue create, fornecido pelo CLI (Command Line Interface) do próprio Vue. Rodando o comando no terminal pedindo como parâmetro obrigatório o nome do projeto, guia o usuário através de diversas configurações pra deixar o projeto o mais completo possível, como pacotes adicionais, como o Vue Router (para navegação) e o Vuex (para armazenamento compartilhado de dados), ou outras funções como configuração de PWA, etc.

Porém, para o desenvolvimento desse projeto, foi escolhido usar outro pacote para a criação, o Vite. O Vite, do francês para rápido, é um CLI que fornece, além de para Vue, também para React, uma forma mais rápida e mais leve de rodar as aplicações. Um dos motivos é que, enquanto o CLI da Vue baixa diversos pacotes desnecessário e ainda cria arquivos de configuração que muitas vezes são inúteis para o usuário, o Vite baixa apenas os pacotes realmente necessários e junta todas as configurações padrão internamente, podendo alterá-las adicionando as mudanças em seu próprio arquivo de configuração.

Com isso, foi decidido usar a versão de criação de projeto Vue junto com Typescript, porém, não é possível instalar direto o template typescript para versão do Vue 2.x, que é a versão do projeto, então foi necessário instalar junto um plugin do Vite que permitia essa integração entre versões.

O decorrer do desenvolvimento do projeto será descrito nas seções a seguir.

4.2.3.1. Dependências e o package.json

O arquivo mais importante em qualquer projeto baseado em Node.js é o package.json, nele as informações básicas do projeto, como nome, scripts e as

dependências usadas no projeto são armazenadas. A Figura 4.6 mostra a versão do arquivo no final do desenvolvimento do projeto.

Na figura é possível observar diferentes seções, as primeiras, como o nome das propriedades sugerem, definem o nome, a versão e se é um projeto privado ou não (essa última com pouco importância para esse projeto). As seguintes definem os “atalhos” para comandos com `npm run` (a primeira), e as dependências, padrões e de desenvolvimento (segunda e terceira, respectivamente).

A diferença entre dependências normais e de desenvolvimento é que a última é usada apenas durante o desenvolvimento e não são levadas em conta durante o processo de build e quando o projeto é rodado com em produção.

Entre as dependências de desenvolvimento, estão principalmente as bibliotecas de tipos, para o typescript, de pacotes que normalmente não fornecem os tipos customizados então a comunidade cria e publica em uma biblioteca compartilhada. Também estão plugins do Vue e do Vite, e diferentes loaders para CSS e SASS.

Para as dependências normais, a Tabela 4.2 descreve o que cada uma das mais importantes faz e o porquê de ela ter sido escolhida para o projeto.

```
{
  "name": "istash",
  "private": true,
  "version": "0.0.0",
  "scripts": {
    "serve": "vite",
    "build": "vue-tsc --noEmit && vite build",
    "preview": "vite preview",
    "host": "vite preview --host"
  },
  "dependencies": {
    "firebase": "^8.6.7",
    "jspdf": "^2.5.1",
    "jspdf-autotable": "^3.5.23",
    "localforage": "^1.10.0",
    "node-sass": "^7.0.1",
    "register-service-worker": "^1.7.1",
    "velocity-animate": "^1.5.2",
    "vue": "^2.6.14",
    "vue-i18n": "^8.25.0",
    "vue-notification": "^1.3.20",
    "vue-router": "^3.2.0",
    "vuetify": "^2.6.4",
    "vuex": "^3.4.0"
  },
  "devDependencies": {
    "@types/node": "^17.0.35",
    "@types/velocity-animate": "^2.0.1",
    "@types/vue": "^2.0.0",
    "@vitejs/plugin-vue": "^2.3.1",
    "@vue/cli-plugin-babel": "^4.5.0",
    "@vue/cli-plugin-pwa": "^4.5.0",
    "@vue/cli-plugin-router": "^4.5.0",
    "@vue/cli-plugin-vuex": "^4.5.0",
    "@vue/eslint-config-prettier": "^6.0.0",
    "@vue/runtime-dom": "^3.2.33",
    "sass": "~1.32",
    "sass-loader": "^12.6.0",
    "style-loader": "^2.0.0",
    "typescript": "^4.5.4",
    "vite": "^2.9.7",
    "vite-plugin-pwa": "^0.12.0",
    "vite-plugin-vue2": "^2.0.0",
    "vue-tsc": "^0.34.7"
  }
}

```

Figura 4.7 - Arquivo package.json

Fonte: O Autor (2022)

Dependência	**Motivo**
vue-router	Permite o controle de navegação entre páginas
vuex	Permite a criação e controle de estados compartilhado com toda a aplicação
vue-i18n	Usada para a implementação da internacionalização do projeto
vuetify	Fornece componentes e estilos personalizáveis mais avançados que o html padrão
vue-notification / velocity-animate	Usado para gerar notificações flutuantes na tela
local-forage	Permite o armazenamento dos dados offline na cache
jspdf / jspdf-autotable	Criação de tabelas em pdf
firebase	Responsável pela autenticação e pelo banco de dados

Tabela 4.2 - Principais dependências do projeto

Fonte: O Autor (2022)

4.2.3.2. Estrutura Básica

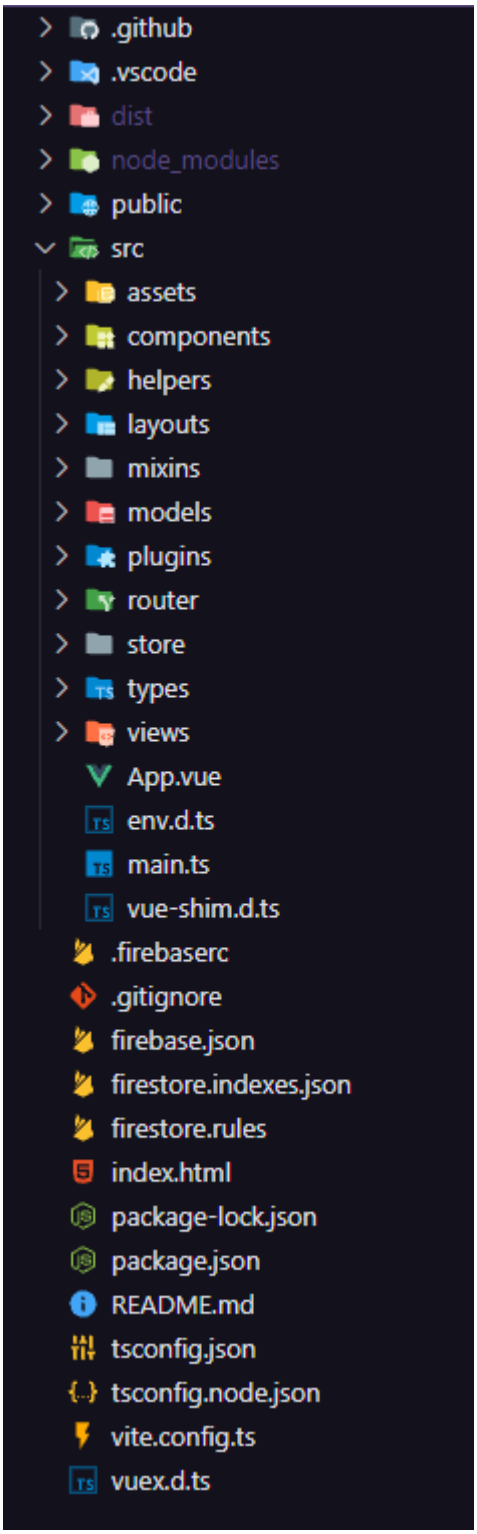


Figura 4.8 - Estrutura de pastas

Fonte: O Autor (2022)

A Figura 4.8 mostra a estrutura de pastas usada para o projeto.

As pastas .github e .vscode são criadas automaticamente pela configuração do Git e pelas configurações de depuração do editor usado para o desenvolvimento (Visual Code).

A pasta *dist* guarda o resultado da compilação do projeto, e a pasta *public* guarda dados compartilhados e que são levados em conta durante essa compilação, como imagens e ícones.

A pasta *node_modules* é gerada automaticamente e armazena todos as dependências e as dependências da mesma, e por isso é normalmente a maior pasta de todo o projeto.

Por fim, a pasta *src* é onde todo o código do projeto fica. Os arquivos ficam separados em pastas de acordo com sua função, sendo as principais as páginas, na pasta *views* e os componentes na pasta de mesmo nome.

Os demais arquivos são as configurações geradas por outras dependências, como firestore e o vite, e o index.html base para a geração do produto final.

4.2.3.3. Configurações básicas

Além das configurações básicas criadas junto ao projeto, algumas inclusões e alterações foram necessárias. Entre elas, estavam a configuração do plugin de PWA, Vuetify, Firebase, armazenamento e rotas.

4.2.3.3.1. PWA

Para a criação de aplicações progressivas, o Vite fornece um plugin que facilita a criação de todas as configurações necessárias, como o manifest, e suporte ao controle de mudanças na cache.

O primeiro passo é a configuração do manifest, a Figura 4.9 mostra a definição do manifest, como o nome da aplicação, uma descrição, e principalmente as imagens referentes ao ícone e “splash page” do produto final.

```
VitePWA({
      includeAssets: ["favicon.ico", "robots.txt", "apple-touch-icon.png"],
      manifest: {
        name: "iStash",
        short_name: "iStash",
        description: "An app for controlling inventory",
        theme_color: "#303F9F",
        icons: [
          {
            src: "img/icons/android-chrome-192x192.png",
            sizes: "192x192",
            type: "image/png",
          },
          {
            src: "img/icons/android-chrome-512x512.png",
            sizes: "512x512",
            type: "image/png",
          },
          {
            src: "img/icons/android-chrome-maskable-512x512.png",
            sizes: "512x512",
            type: "image/png",
            purpose: "any maskable",
          },
        ],
      },
    })
```

Figura 4.9 - Configuração do manifest da aplicação

Fonte: O Autor (2022)

Além disso, as imagens também devem ser importadas dentro do arquivo index.html. Esse arquivo define a base da página onde o código resultante da compilação do projeto será injetado.

```html
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,initial-scale=1.0" />
    <link
      href="https://fonts.googleapis.com/css?family=Roboto:100,300,400,500,700,900"
      rel="stylesheet"
    />
    <link
      href="https://cdn.jsdelivr.net/npm/@mdi/font@5.x/css/materialdesignicons.min.css"
      rel="stylesheet"
    />
    <title>iStash</title>
    <meta name="description" content="Your app description" />
    <link rel="icon" href="/favicon.ico" type="image/svg+xml" />
    <link
      rel="alternate icon"
      href="/favicon.ico"
      type="image/png"
      sizes="16x16"
    />
    <link
      rel="apple-touch-icon"
      href="img/icons/apple-touch-icon.png"
      sizes="180x180"
    />
    <link rel="mask-icon" href="/favicon.ico" color="#303F9F" />
    <meta name="theme-color" content="#303F9F" />
  </head>

  <body>
    <noscript>
      <strong>
        We're sorry but iStash doesn't work properly without JavaScript enabled.
        Please enable it to continue.
      </strong>
    </noscript>
    <div id="app"></div>
    <script type="module" src="/src/main.ts"></script>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Figura 4.10 - Arquivo index.html

Fonte: O Autor (2022)

Um ponto importante na importação de ícones para um PWA é a necessidade de fornecer imagens de diferentes tamanhos, para diferentes resoluções de dispositivos. Nesse projeto, foram incluídos ícones com tamanhos que variam de 57x57 até 512x512.

Figura 4.11 - Ícone do aplicativo

Fonte: O Autor (2022)

O último aspecto para finalizar a configuração do PWA é o controle automático de mudanças para o usuário. Felizmente o plugin do Vite já fornece tudo que é necessário para concluir tal tarefa. Na documentação do plugin é fornecido um componente que pode ser adicionado ao projeto que mostra um pop-up sempre que há mudanças na aplicação. Porém, para a versão do Vue 2.X, os comandos que são usados não estão presentes, então a documentação também fornece os métodos necessários para serem adicionados ao código como um mixin (Um mixin seria como um componente, porém contendo apenas a parte de código javascript. Ele fornece métodos e estado como um componente normal e pode ser importado de qualquer componente Vue, podendo até criar overrides para comportamentos customizados).

As Figuras 4.12 e 4.13 mostram a implementação desse componente e do mixin, respectivamente.

```vue
<template>
  <div v-if="offlineReady || needRefresh" class="pwa-toast" role="alert">
    <div class="message">
      <span v-if="offlineReady"> App ready to work offline </span>
      <span v-else>
        New content available, click on reload button to update.
      </span>
    </div>
    <button v-if="needRefresh" @click="updateServiceWorker">Reload</button>
    <button @click="closePromptUpdateSW">Close</button>
  </div>
</template>

<script>
import Vue from "vue";
import useRegisterSW from "../mixins/useRegisterSW";

const intervalMS = 60 * 60 * 1000;

export default Vue.extend({
  name: "reload-prompt",
  mixins: [useRegisterSW],
  methods: {
    handleSWManualUpdates(r) {
      r &&
        setInterval(() => {
          r.update();
        }, intervalMS);
    },
  },
});
</script>

<style>
.pwa-toast {
  position: fixed;
  right: 0;
  bottom: 0;
  margin: 16px;
  padding: 12px;
  border: 1px solid #8885;
  border-radius: 4px;
  z-index: 1;
  text-align: left;
  box-shadow: 3px 4px 5px 0 #8885;
}
.pwa-toast .message {
  margin-bottom: 8px;
}
.pwa-toast button {
  border: 1px solid #8885;
  outline: none;
  margin-right: 5px;
  border-radius: 2px;
  padding: 3px 10px;
}
</style>
```

Figura 4.12 - Componente reloadPrompt.vue

Fonte: [31]

```
import Vue from "vue"

export default Vue.extend({
  name: "useRegisterSW",
  data() {
    return {
      updateSW: undefined as any,
      offlineReady: false,
      needRefresh: false
    }
  },
  async mounted() {
    try {
      const { registerSW } = await import("virtual:pwa-register")
      const vm = this
      this.updateSW = registerSW({
        immediate: true,
        onOfflineReady() {
          vm.offlineReady = true
          vm.onOfflineReadyFn()
        },
        onNeedRefresh() {
          vm.needRefresh = true
          vm.onNeedRefreshFn()
        },
        onRegistered(swRegistration: any) {
          swRegistration && vm.handleSWManualUpdates(swRegistration)
        },
        onRegisterError(e: any) {
          vm.handleSWRegisterError(e)
        }
      })
    } catch {
      console.log("PWA disabled.")
    }

  },
  methods: {
    async closePromptUpdateSW() {
      this.offlineReady = false
      this.needRefresh = false
    },
    onOfflineReadyFn() {
      console.log("onOfflineReady")
    },
    onNeedRefreshFn() {
      console.log("onNeedRefresh")
    },
    updateServiceWorker() {
      this.updateSW && (this.updateSW as (val: boolean) => void)(true)
      location.reload()
    },
    handleSWManualUpdates(swRegistration: any) {},
    handleSWRegisterError(error: any) {}
  }
})
```

Figura 4.13 - Mixin useRegisterSW.ts

Fonte: [31]

4.2.3.3.2. Firebase e Firestore

Para o controle de autenticação e o armazenamento de dados da aplicação, foi decidido usar o Firebase, uma ferramenta fornecida pela Google que entre outras funcionalidades, oferece as funcionalidades citadas anteriormente, necessárias para essa aplicação, além da hospedagem (que será discutida nas seções a seguir), e sem custos.

A configuração dos módulos começa criando um projeto no site oficial do Firebase, então é gerado uma chave para usar como configuração no código do programa. Então, após adicionar a biblioteca a aplicação é preciso adicionar os dados do projeto como configuração, como mostrado na Figura 4.14.

```
import firebase from "firebase/app";
import "firebase/auth";
import "firebase/firestore";

// firebase init - add your own config here
const firebaseConfig = {
  apiKey: "<API-KEY>",
  authDomain: "istash.firebaseapp.com",
  projectId: "istash",
  storageBucket: "istash.appspot.com",
  messagingSenderId: "838654125998",
  appId: "1:838654125998:web:df6fe17c410c5632b2ca3f",
  measurementId: "G-0GBS97EVTZ",
};

const app = firebase.initializeApp(firebaseConfig);

export default app;

```

Figura 4.14 - Configuração do Firebase

Fonte: O Autor (2022)

4.2.3.3.3. Vuetify e Estilização

O Vuetify foi a biblioteca escolhida para ajudar na questão dos estilos. Além de adicionar componentes extras, a biblioteca também adiciona suportes a temas de cores e funções auxiliares para detectar breakpoints que facilitam a responsividade.

Além disso, foi usado o SASS como auxiliar do css para facilitar a criação de estilos mais complexos, pois adiciona mais funcionalidades ao css padrão.

4.2.3.3.4. Internacionalização

A biblioteca padrão usada para adicionar suporte adicionar à internacionalização em aplicações Vue é o vueI18n. Ela fornece métodos que facilitam as funcionalidades de localização de textos, formatos de datas e formatação de números.

O método mais comum de uso é o "`t(key: string, values?: object)`", que recebe uma chave configurada como mensagem e então retorna o valor correspondente ao locale selecionado.

```
const messages = {
  en: {
    message: {
      hello: 'hello world'
    }
  },
  pt: {
    message: {
      hello: 'olá mundo'
    }
  }
}

// Create VueI18n instance with options
const i18n = new VueI18n({
  locale: 'pt', // set locale
  messages, // set locale messages
})

i18n.t('message.hello') // 'olá mundo'
```

Código 4.1 - Exemplo de uso do vueI18n

Fonte: [32]

Além dessa função, existem outras variações de métodos, como a “`tc(key: string, count: number)`” que permite a pluralização, dado um número como parâmetro adicional. Também são adicionados métodos para a localização de datas e números, caso os atributos “`dateTimeFormats`” e “`numberFormats`” sejam fornecidos.

Para o projeto foram adicionados suporte para português e inglês. A Figura 4.15 mostra a configuração usada para o projeto, e nela existe o atributo “`fallbackLocale`”. Por padrão, o vuel18n usa o locale fornecido pelo browser como locale inicial, caso não o browser não suporte esse valor, a bilbioteca usa o locale fornecido nesse atributo.

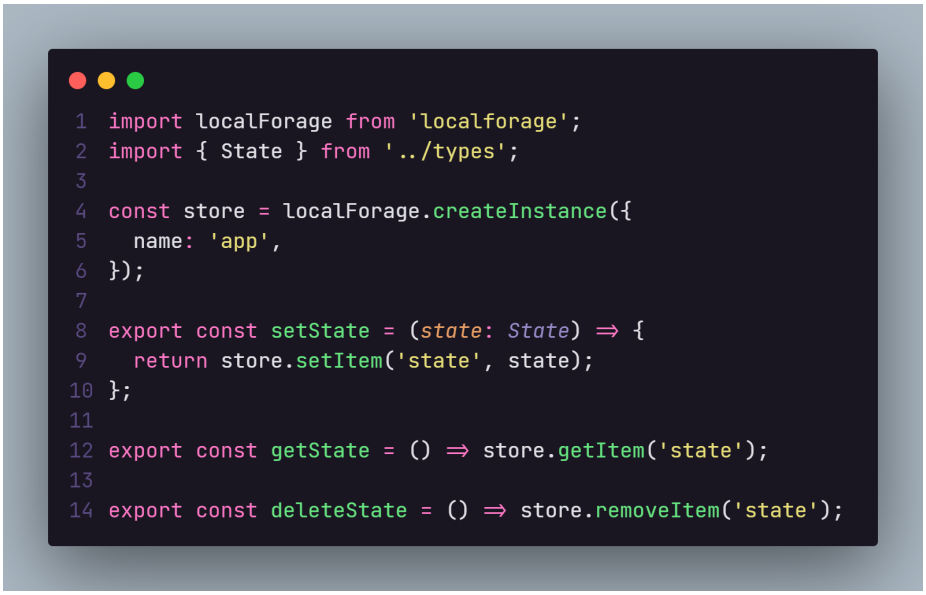
```ts
import Vue from "vue";
import VueI18n from "vue-i18n";

Vue.use(VueI18n);

const messages = {...};

const dateTimeFormats = {...};

const numberFormats = {...};

const i18n = new VueI18n({
  locale: window.navigator.language,
  fallbackLocale: "en-US",
  messages,
  dateTimeFormats: dateTimeFormats as VueI18n.DateTimeFormats,
  numberFormats,
});

export default i18n;

export function changeLocale(newLocale: "pt-BR" | "en-US") {
  console.log(newLocale);
  i18n.locale = newLocale;
}

export function getLocale() {
  return i18n.locale;
}

```

Figura 4.15 - Configuração Vue I18n do projeto

Fonte: O Autor (2022)

4.2.3.3.5. Store

Além do uso de estados internos para cada componente, a aplicação também contém estados compartilhados na forma da Store criada pelo Vuex. Com essa biblioteca, é possível criar um único conjunto de dados que é compartilhado a todos os componentes da aplicação e que podem ser alterados através de mutations e actions (funções mais complexas que mutations, suportando a chamada das mesmas e também o uso de funções assíncronas). Além disso, para suportar o armazenamento dos dados enquanto offline foi usado o Local Forage. Essa é uma biblioteca que armazena dados no browser, similar ao Local Storage, nativo de muitos navegadores, porém na cache.



```
1  import localForage from 'localforage';
2  import { State } from '../types';
3
4  const store = localForage.createInstance({
5    name: 'app',
6  });
7
8  export const setState = (state: State) => {
9    return store.setItem('state', state);
10 };
11
12 export const getState = () => store.getItem('state');
13
14 export const deleteState = () => store.removeItem('state');
```

Figura 4.16 - Configuração da instância do Local Forage

Fonte: O Autor (2022)

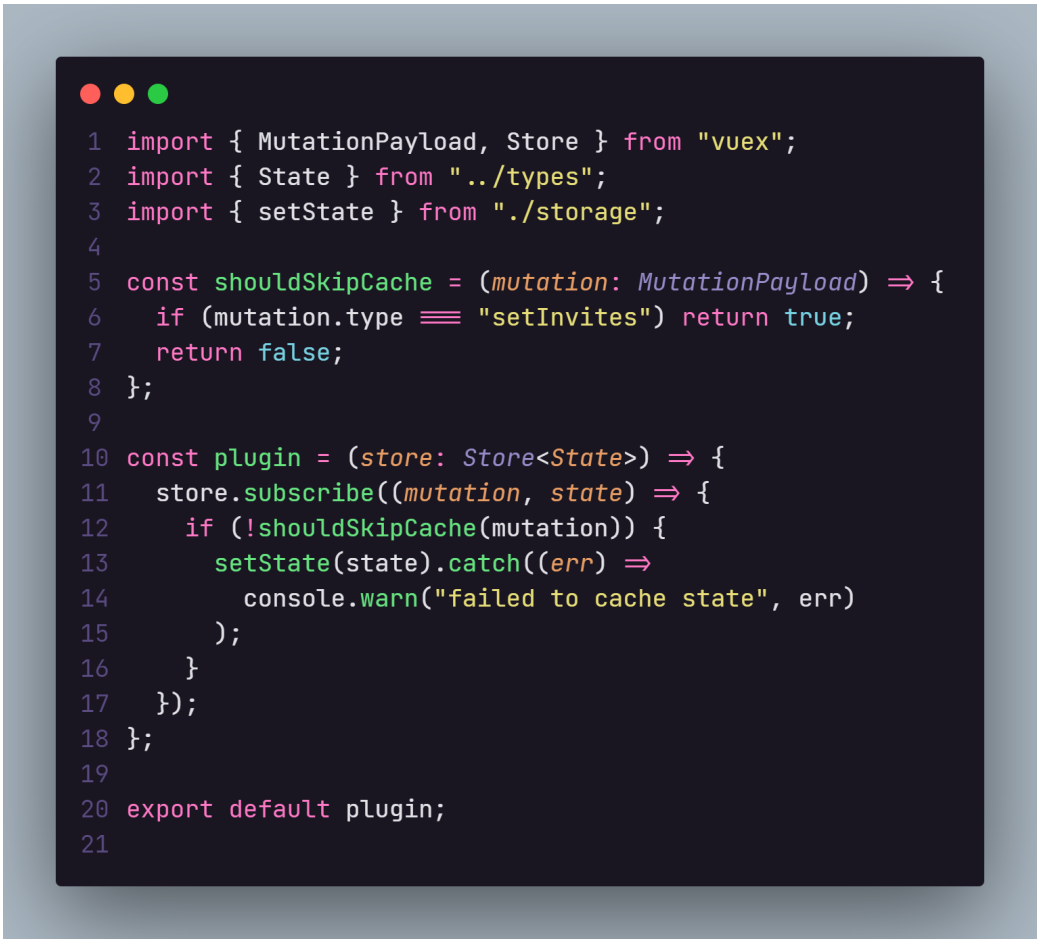
Para tal tarefa foram necessários criar três plugins para a Store.

```
import { Store } from "vuex";
import { State } from "../types";
import { getState } from "./storage";

export default function (store: Store<State>) {
  if (store.state.initialized) {
    return Promise.resolve();
  }

  return getState().then((state) => {
    store.commit("setSavedData", state);
  });
}

```

Figura 4.17 - Plugin load

Fonte: O Autor (2022)

O primeiro plugin criado é o “load” que roda sempre que a aplicação inicia e recupera os dados salvas na cache.



```
import { MutationPayload, Store } from "vuex";
import { State } from "../types";
import { setState } from "./storage";

const shouldSkipCache = (mutation: MutationPayload) => {
  if (mutation.type === "setInvites") return true;
  return false;
};

const plugin = (store: Store<State>) => {
  store.subscribe((mutation, state) => {
    if (!shouldSkipCache(mutation)) {
      setState(state).catch((err) =>
        console.warn("failed to cache state", err)
      );
    }
  });
};

export default plugin;

```

Figura 4.18 - Plugin cache

Fonte: O Autor (2022)

O segundo é o “cache”, esse plugin é rodado após cada mutação que acontece na Store e atualiza os dados do Local Forage.

```
import { Store } from "vuex";
import { removeValue, updateValue } from "../plugins/firebase/firestore";
import { Mutation, State } from "../types";

const plugin = (store: Store<State>) => {
  store.subscribe((mutation, state) => {
    switch (mutation.type) {
      case "updateStash":
        updateValue(
          "stashes",
          mutation.payload["id"],
          mutation.payload["value"]
        )
          .catch()
          .finally(() => {
            state.updateData = true;
          });
        break;
      case "removeStash":
        removeValue("stashes", mutation.payload["id"])
          .catch()
          .finally(() => {
            state.updateData = true;
          });
        break;
      default:
      // Ignore default case
    }
  });
};

export default plugin;

```

Figura 4.19 - Plugin sync

Fonte: O Autor (2022)

E por último o plugin “sync”, assim como o anterior, ele roda após cada mutação e gera uma alteração na API, que irá alterar os dados guardados no banco de dados, apenas quando o usuário estiver online.

No produto final, o estado final contém:

- `initialized`: Uma flag para controlar se os dados já foram carregados da cache;
- `logged`, `currentUser` e `userId`: Dados referentes ao usuário conectado;
- `myStashes` e `myInvites`: Referentes aos dados dos stashes e convites de outros stashes, carregados do banco de dados;

- `stashesLoaded` e `invitesLoaded`: Flags que indicam se os dados do banco de dados já foram carregados do banco de dados pela primeira vez;
- `updateData`: Habilita ou desabilita a atualização dos dados temporariamente;
- `diffs` e `newData`: Usados para o controle de versão e conflitos.

Na parte de mutações, estão apenas funções para alterar esses dados, e nas actions, estão login e removeStash, como na Figura 4.16. Como é possível ver na figura, na função removeStash é usado um getter. Getters são definidos dentro da própria store que selecionam dados e retornam eles de maneira mais otimizada. Nesse caso, ele retorna o stash dado o id dele como parâmetro.

```
actions: {
  login({ commit }, currentUser: string) {
    commit("login", currentUser);

    if (currentUser) {
      db.collection("users")
        .doc(currentUser)
        .get()
        .then((doc) ⇒ doc.exists && commit("setUserData", doc.data()));
    }
  },
  removeStash({ getters }, id: string) {
    const element = getters.getStash(id);
    element.remove();
  },
},
```

Figura 4.20 - Actions da Store

Fonte: O Autor (2022)

4.2.3.3.6. Router

A última configuração necessária é a do Router, que define cada rota e o componente que será renderizado em cada uma.

Figura 4.21 - Configuração do router

Fonte: O Autor (2022)

No objeto de configuração são passados o modo do router, a base e um vetor com objetos definindo as rotas.

O modo define como será feito o armazenamento das rotas. O modo history é o mais comum em páginas web, onde a URL é gerada como um caminho de arquivos através de pastas, já o outro modo disponivel é o hash, onde após a URL base da aplicação, existe um sinal de hashtag (#) e depois são definidas as sub-rotas. Esse último é mais comum em aplicações Single-Page por não precisar de configurações extras para redirecionar todas as chamadas do servidor para um único arquivo enquanto no outro modo, requisições para a url (localhost/home) procuraria por um arquivo index dentro da pasta home na pasta raíz da aplicação. Porém, atualmente vários servidores já contam com configurações para suportar o modo history em SPA's e por isso esse modo foi escolhido como o modo do Router.

O atributo base define a URL base para o Router e o ponto apenas diz que a URL base será a mesma onde a aplicação está hospedada.

E o routes consiste da lista de rotas passadas que seguem o padrão mostrado na Figura 4.22.

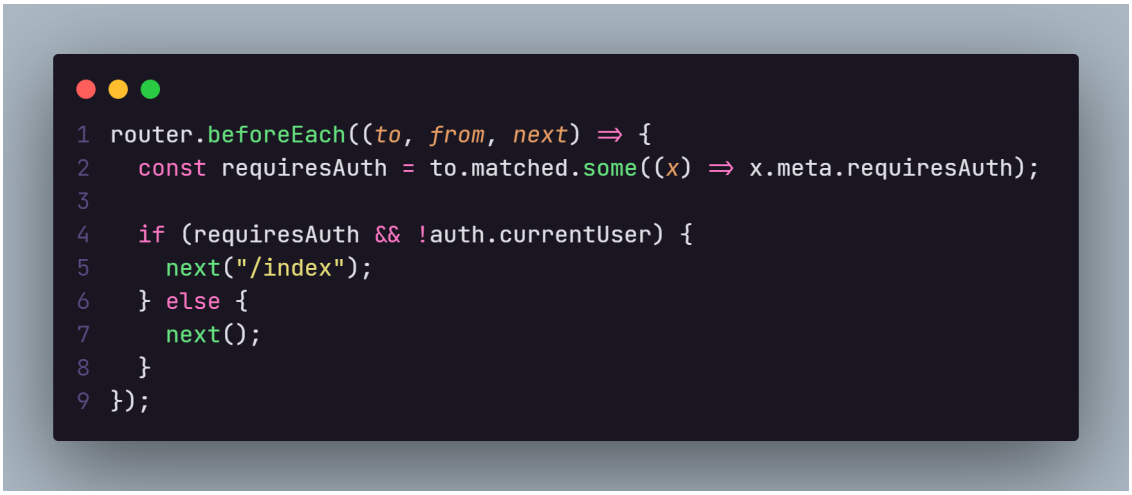
```
{
  path: "/",
  name: "home",
  component: StashesList,
  meta: {
    requiresAuth: true,
  },
}
```

Figura 4.22 - Exemplo de rota do Router

Fonte: O Autor (2022)

O objeto define o caminho para a rota, o nome, o componente, e uma série de parâmetros que podem ser o código em outros lugares do código para diferentes comportamentos no programa. Nesse caso, o parâmetro adicionado é o “requiresAuth”, que define quais telas podem ser acessadas enquanto o usuário está conectado em uma conta ou não.

Para esse fim foi criado um método que roda antes do navegador entrar em cada rota que checa se o usuário está logado e a página precisa de tal credencial.



```
router.beforeEach((to, from, next) => {
  const requiresAuth = to.matched.some((x) => x.meta.requiresAuth);

  if (requiresAuth && !auth.currentUser) {
    next("/index");
  } else {
    next();
  }
});
```

Figura 4.23 - Checagem da autenticação

Fonte: O Autor (2022)

4.2.3.4. Telas

Essa sessão irá descrever cada tela da aplicação e os componentes principais usados em cada uma. Porém, antes de qualquer tela ser renderizada um componente renderizado é o App.vue. Esse componente é o núcleo de toda a aplicação, e nele estão os componentes que devem aparecer na tela não importando em qual página o usuário estiver. No caso desse projeto são renderizados o cabeçalho (Header.vue), o componente de notificações (Notification.vue) e o Drawer, um container lateral com algumas funções extras. Também é no App que os dados são carregados do banco de dados.

```
<template>
  <v-app>
    <Header :openDrawer="openDrawer" />
    <v-main id="app">
      <Notification group="center" />
      <router-view></router-view>
      <div
        v-if="Object.keys($store.state.diffs).length > 0"
        style="position: fixed; bottom: 0.5rem; left: 0.5rem"
      >
        <v-icon>mdi-alert-circle</v-icon>
        {{ $t("message.pendingchanges") }}
      </div>
    </v-main>

    <reload-prompt />
    <app-drawer :model="drawer" ref="drawer" v-if="logged"></app-drawer>
  </v-app>
</template>
```

Figura 4.24 - Template - App.vue

Fonte: O Autor (2022)

Todos os componentes que começam com “v-” são disponibilizados pelo Vuetify. E além dos outros componentes citados anteriormente, o “router-view” define o componente que irá renderizar as telas dependendo da página, o “div” é apenas um aviso para o usuário caso exista qualquer mudança no controle de versão e o “reload-prompt” é o componente do PWA para avisar caso exista mudanças na aplicação necessitando de atualização da página.

```
computed: {
    logged(): boolean {
      return this.$store.state.logged;
    },
    stashes(): Document {
      return firestore.collection("stashes");
    },
    userStashes(): Query {
      return this.stashes
        .where("users", "array-contains", this.$store.state.userId)
        .orderBy("date");
    },
    userInvites(): Query {
      return this.stashes.where(
        "invites",
        "array-contains",
        this.$store.state.userId
      );
    },
  },
```

Figura 4.25 - Computed props - App.vue

Fonte: O Autor (2022)

As propriedades computadas usadas nesse componente são um “atalho” para a propriedade “logged” de dentro da Store e um para a coleção de stashes do Firestore. Os dois últimos são as queries para obter os stashes do usuários e os convites para os stashes de outros checando se o id do usuário está no vetor de usuário ou de convites respectivamente.

Dos métodos desse componente, os mais importantes são os referentes ao carregamento de dados. O método “load” primeiramente checa se o navegador está online; se sim, são criadas duas Promises para carregar os Stashes e o convites separadamente; se não, apenas pega os dados salvos na cache pelo Local Forage e atualiza a Store. Não importando o resultado, o método cria um timeout para si mesmo para atualizar os dados a cada 5 segundos, alterando os valores para os novos.

```
async load() {
      if (navigator.onLine) {
        let promises = [];

        if (this.$store.state.logged && this.$store.state.updateData) {
          promises.push(new Promise(this.loadStashes));
          promises.push(new Promise(this.loadInvites));
        }

        Promise.all(promises).then(() => setTimeout(this.load, 5 * 1000));
      } else {
        getState().then((state) => {
          this.$store.commit("setSavedData", state);

          setTimeout(this.load, 5 * 1000);
        });
      }
    }
```

Figura 4.26 - Método load - App.vue

Fonte: O Autor (2022)

O método loadInvites apenas carrega os dados que satisfazem a query citada anteriormente e formata os dados separando o id, construindo a mensagem do convite a partir do método do Vue I18n.

```
async loadInvites(resolve: Resolve): Promise<void> {
      const stashes = await this.userInvites.get();

      if (stashes) {
        this.$store.commit(
          "setInvites",
          stashes.docs
            .filter((el) => el.data().shared)
            .map((el) => {
              const data = el.data();
              return {
                id: el.id,
                message: this.$t("message.invitemessage", {
                  user: data.usersInfo.find(
                    (el: User) => el.userStatus === OWNER
                  ).name,
                  stash: data.name,
                }),
                stash: createStash(data as Stash),
              };
            })
        );
      }

      resolve();
    }
```

Figura 4.27 - Método loadInvites - App.vue

Fonte: O Autor (2022)

Já o método loadStashes é mais complexo. Primeiro, após obter os dados do banco de dados, os stashes armazenados na Store com versão negativa (atributo usado para diferenciar os dados criados enquanto o aplicativo estava offline) são criados dentro do banco de dados; então os objetos do banco de dados são convertidos em classes e então começa o processo de checagem de diferenças de versões: Se a versão do stash salvo localmente é mais do que a do banco de dados, então os dados do banco de dados são atualizados, se não, os dados são comparados e armazenados em um objeto separados pelo id de cada stash. Após a checagem, caso não exista nenhum conflito, os dados sobrescrevem os armazenados na Store e caso exista, os conflitos são salvos no atributo “diffs” e os novos dados são salvos no atributo “newData”, para que seja possível controlar qual dados serão salvos posteriormente.

```
async loadStashes(resolve: Resolve): Promise<void> {
      const stashes = await this.userStashes.get();

      this.$store.state.myStashes
        ?.filter((el: Stash) => el.version < 0)
        .forEach((item: Stash) => {
          updateValue("stashes", item.id, item.buildTemplate());
        });

      if (stashes) {
        const parsedData = stashes.docs.map((el) => {
          const data = el.data();
          return createStash(data as Stash);
        });

        const diffs = parsedData.reduce((prev, el) => {
          if (this.$store.state.myStashes.length > 0) {
            let localData = this.$store.getters.getStash(el.id);

            if (localData.version < el.version) {
              const newDiff = diff(el.products, localData.products);

              const next = {
                ...prev,
                [el.id]: newDiff,
              };

              if (localData.name !== el.name) {
                (next as any)[el.id]["name"] = {
                  newName: el.name,
                  oldName: localData.name,
                };
              }

              if (Object.keys((next as any)[el.id]).length === 0)
                delete (next as any)[el.id];

              return next;
            } else if (localData.version > el.version) {
              if (!("buildTemplate" in localData))
                localData = createStash(localData);

              updateValue("stashes", el.id, localData.buildTemplate());
            }
          }
          return prev;
        }, {});

        if (Object.keys(diffs).length === 0)
          this.$store.commit("setStashes", parsedData);
        else this.$store.commit("setDiffs", { diffs, data: parsedData });
      }

      resolve();
    }
```

Figura 4.28 - Método loadStashes - App.vue

Fonte: O Autor (2022)

```
const COMPARE_KEYS = ["name", "quantity", "rule", "unit"];

export function diff(data: Product[], local: Product[]) {
  if (!local) {
    return {};
  }

  var diffs: { [id: string]: any } = {};
  var key: string;

  var compare = function (
    item1: unknown,
    item2: unknown,
    id: string,
    name: string,
    key: string
  ) {
    if (item1 !== item2) {
      if (!(id in diffs)) diffs[id] = { id: id, name: name };
      diffs[id][key] = { oldValue: item2, newValue: item1 };
    }
  };

  if (data.length > local.length) {
    data.forEach((el) => {
      const localEl = local.find((x) => x.id === el.id);

      if (!localEl) return;

      COMPARE_KEYS.forEach((propName) => {
        compare(
          el[propName as keyof Product],
          localEl[propName as keyof Product],
          el.id,
          el.name,
          propName
        );
      });
    });
  } else {
    local.forEach((el) => {
      const dataEl = data.find((x) => x.id === el.id);

      if (!dataEl) return;

      COMPARE_KEYS.forEach((propName) => {
        compare(
          dataEl[propName as keyof Product],
          el[propName as keyof Product],
          el.id,
          el.name,
          propName
        );
      });
    });
  }

  // Return the object of differences
  return diffs;
}
```

Figura 4.29 - Método diff - diff.ts

Fonte: O Autor (2022)

Dos componentes usados dentro do App.vue o Notification.vue apenas inicializa o componente do Vue Notification configurando animações e posicionamento. O Header.vue define os elementos do cabeçalho, como o botão de voltar, que só aparece caso a rota possua um parâmetro de rota anterior; o título que se altera para o nome do stash aberto e um componente para alterar a linguagem do aplicativo. E por último, o Drawer que cria um container extensível que mostra o nome do usuário conectado e dá acesso às telas de notificações (e convites) e de alterações (conflitos).

```
<template>
  <v-app-bar app color="primary" id="app-bar" dark>
    <v-btn :class="!route.meta.backRoute && 'hidden'" icon @click="handleHome">
      <v-icon>mdi-arrow-left</v-icon>
    </v-btn>

    <v-spacer></v-spacer>

    <v-toolbar-title class="text-bold">{{ title }}</v-toolbar-title>

    <v-spacer></v-spacer>

    <v-app-bar-nav-icon
      v-if="logged"
      :class="!logged && 'hidden'"
      @click.stop="openDrawer"
    ></v-app-bar-nav-icon>
    <v-menu offset-y v-else>
      <template v-slot:activator="{ on, attrs }">
        <v-btn dark v-bind="attrs" v-on="on" plain small class="pa-1">
          {{ selectedLocale }}
        </v-btn>
      </template>
      <v-list>
        <v-list-item
          v-for="(item, index) in locales"
          :key="index"
          @click="selectedLocale = item"
        >
          <v-list-item-title>{{ item }}</v-list-item-title>
        </v-list-item>
      </v-list>
    </v-menu>
  </v-app-bar>
</template>
```

Figura 4.30 - Template do cabeçalho - Header.vue

Fonte: O Autor (2022)

```vue
<template>
  <v-navigation-drawer v-model="drawer" absolute temporary right>
    <v-list nav dense>
      <v-list-item-content class="pl-2">
        <v-list-item-title
          class="text-h6"
          style="text-transform: capitalize"
          v-if="$store.state.currentUser"
        >
          {{ $store.state.currentUser.name }}
        </v-list-item-title>
      </v-list-item-content>

      <v-divider></v-divider>

      <v-list-item
        link
        @click="handleNotifications"
        :disabled="route.name === 'notifications'"
        v-if="isOnline"
      >
        <v-list-item-icon>
          <v-icon> mdi-bell </v-icon>
          <div class="badge" v-if="$store.state.myInvites.length > 0"></div>
        </v-list-item-icon>
        <v-list-item-title>{{ $t("keys.notifications") }}</v-list-item-title>
      </v-list-item>

      <v-list-item
        link
        @click="handleChanges"
        :disabled="$store.state.newData.length === 0"
        v-if="isOnline"
      >
        <v-list-item-icon>
          <v-icon> mdi-clipboard-search </v-icon>
          <div
            class="badge"
            v-if="Object.values($store.state.diffs).length > 0"
          ></div>
        </v-list-item-icon>
        <v-list-item-title>{{ $t("keys.changes") }}</v-list-item-title>
      </v-list-item>

      <v-divider></v-divider>

      <v-list-item link @click="handleSignOut">
        <v-list-item-title>{{ $t("button.logout") }}</v-list-item-title>
      </v-list-item>

      <v-divider></v-divider>

      <v-list-item v-if="$store.state.logged" style="padding: 0.2rem 1rem">
        <v-list-item-title>{{ $t("keys.language") }}</v-list-item-title>
        <v-spacer></v-spacer>
        <v-menu offset-y>
          <template v-slot:activator="{ on, attrs }">
            <v-btn v-bind="attrs" v-on="on" plain small class="pa-1">
              {{ selectedLocale }}
            </v-btn>
          </template>
          <v-list>
            <v-list-item
              v-for="(item, index) in locales"
              :key="index"
              @click="changeLanguage(item)"
            >
              <v-list-item-title>{{ item }}</v-list-item-title>
            </v-list-item>
          </v-list>
        </v-menu>
      </v-list-item>
    </v-list>
  </v-navigation-drawer>
</template>
```

Figura 4.31 - Template do drawer - Drawer.vue

Fonte: O Autor (2022)

4.2.3.4.1. Tela inicial

A primeira página que o usuário vê ao entrar no aplicativo pela primeira vez, é a tela de usuário não conectado. Ela é apenas um texto pedindo que o usuário se conecte a uma conta oferencendo as opções de fazer login ou criar uma conta nova.

Nessa tela também é a primeira ocorrência dos métodos de localização do Vue I18n e dos métodos de troca de rotas do Vue Router.

O componente do botão é disponibilizado pelo Vuetify e o único componente customizado é o “Main”, que é apenas um container (também do Vuetify) configurado com classes para ocupar toda a tela e centralizar o conteúdo.

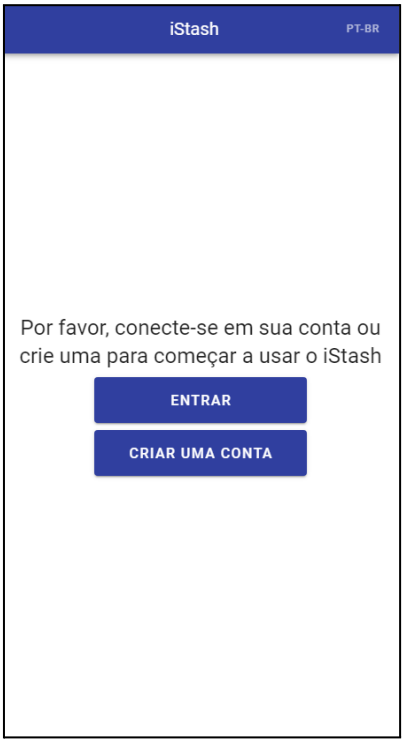


Figura 4.32 - Tela inicial

Fonte: O Autor (2022)

```vue
<template>
  <Main>
    <span class="headline text-center">{{ $t("message.welcome") }}</span>
    <v-btn x-large color="primary" class="button mt-2" @click="handleLogin">{{
      $t("button.login")
    }}</v-btn>
    <v-btn x-large color="primary" class="button mt-2" @click="handleSignUp">{{
      $t("button.signup")
    }}</v-btn>
  </Main>
</template>

<script>
import Main from "../../layouts/Main.vue";

export default {
  name: "NotLogged",
  components: { Main },
  methods: {
    handleLogin() {
      this.$router.replace("/login");
    },
    handleSignUp() {
      this.$router.replace("/signup");
    },
  },
};
</script>
```

Figura 4.33 - Código da tela inicial - NotLogged.vue

Fonte: O Autor (2022)

4.2.3.4.2. Login e Criação de conta

Ambas telas de login e criação de conta são formulários pedindo informações básicas para exercerem as suas funções. Para criar uma conta são necessários o nome, o email e a senha, sendo os dois ultimos, os dados usados para a conexão posteriormente. Além disso, na tela de login, também existe um link direcionando para a tela de criação, e outro que altera para o modo de “esqueci minha senha”, que pede ao usuário o email e manda para o mesmo a senha de sua conta.

Nas duas telas são usados os mesmos componentes que foram criados baseados no “v-text-field” do Vuetify, mas como possuiam muitos parâmetros em comum, foram separados em um único componente. Os componentes “`TextInput`” e “`PasswordInput`” são caixas de textos estilizadas com algumas regras de validação (mostrando mensagens de erros caso condições passadas não sejam satisfeitas), sendo o segundo uma variação da primeira trazendo características específicas de caixas de senha, como a opção de esconder e mostrar os caractéres.

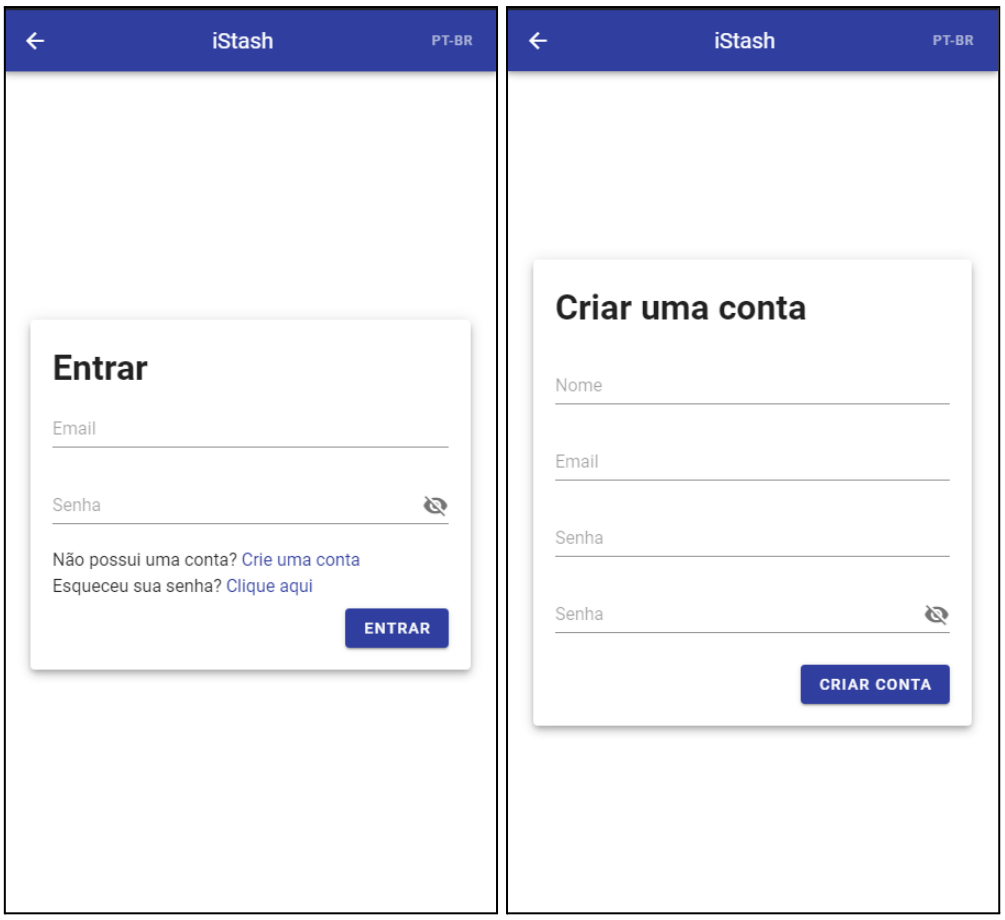


Figura 4.34 - Tela de login e criação de conta

Fonte: O Autor (2022)

```vue
<TextInput
  v-model="email"
  placeholder="Email"
  required
  email
  :rules="[emailErrorMessage]"
/>
<PasswordInput
  v-model="password"
  required
  :rules="[passwordErrorMessage]"
/>
```

Figura 4.35 - Componentes TextInput e PasswordInput

Fonte: O Autor (2022)

Como é possível observar nos componentes mostrados na figura 4.29, um parâmetro que é muito comum em componentes Vue e aparece muito durante o desenvolvimento do projeto, é o “v-model”. Esse atributo age como uma via de mão dupla em questão de obter e atualizar os dados. A variável que está em um v-model fornece os seus dados para o componente, e caso o componente altera os seus dados, também alterará o valor da variável. Isso tira a necessidade de criação de eventos “onchange” para a alterar o valor manualmente.

Para a manutenção dos dados de usuário são usados os próprios comandos do módulo de autenticação do Firebase:

- A função usada, “`createUserWithEmailAndPassword(email: string, password: string)`”, cria a conta no módulo de autenticação, mas para acessar os dados como o nome e email no resto da aplicação, posteriormente eram adicionados tais informações a coleção de usuários do Firestore;

```
handleSignUp() {
  if (!this.name || !this.email || !this.password || !this.confirmPassword)
    return this.$notify({
      group: "center",
      title: this.$t("keys.error"),
      text: this.$t("message.fillrequiredfields"),
      type: "error",
    });

  auth
    .createUserWithEmailAndPassword(this.email.toLowerCase(), this.password)
    .then(
      (a) => this.handleSignUpSuccess(a.user),
      (error) => this.parseError(error)
    );
},
handleSignUpSuccess(userAuth) {
  const newUser = {
    name: this.name,
    email: userAuth.email,
    uid: userAuth.uid,
  };

  firestore.collection("users").doc(userAuth.uid).set(newUser);

  this.$router.replace("/");
}
```

Figura 4.36 - Métodos de criação de conta - SignUp.vue

Fonte: O Autor (2022)

- Para fazer o login foi usado o “`signInWithEmailAndPassword(email: string, password: string)`”;

```
handleLogin() {
  if (!this.email || !this.password)
    return this.$notify({
      group: "center",
      title: this.$t("keys.error"),
      text: this.$t("message.fillrequiredfields"),
      type: "error",
    });

  auth
    .signInWithEmailAndPassword(this.email.toLowerCase(), this.password)
    .then(
      () => this.$router.push("/"),
      (error) => this.parseError(error)
    );
}
```

Figura 4.37 - Método de login - Login.vue

Fonte: O Autor (2022)

- E por último, para recuperar a senha foi usado o método “`sendPasswordResetEmail(email: string)`”.

```
handleForgotPasswordSubmit() {
  auth
    .sendPasswordResetEmail(this.email)
    .catch((error) => this.parseError(error));
}
```

Figura 4.38 - Método para recriar a senha - Login.vue

Fonte: O Autor (2022)

4.2.3.4.3. Lista de Stashes

Essa tela é a principal e a que o usuário chegará na aplicação caso ele já esteja conectado. Ela consiste da lista de todos os Stashes que o usuário tem acesso, tanto as criadas por ele ou as que ele faz parte pois foi convidado.

A topo da tela possui uma barra de pesquisa, para pesquisar pelo nome do Stash e um botão para criar um novo Stash. Abaixo estão os botões para acessar a tela de detalhes do stash e no fim da lista, outro botão, também com a função de criação de Stash.

Um botão de Stash consiste do seu nome no topo esquerdo, seguidos abaixo do nome do criador (caso não seja o usuário conectado), da quantidade de produtos no Stash e a quantidade de produtos que são controlados (quando um produto é cadastrado, é possível colocar uma quantidade que será controlada, ou seja, caso o produto tenha uma quantidade menor que o controle, ele será adicionado à lista de compras). No lado direito estão, em ordem, um menu flutuante com a opção de apagar o Stash, e dois ícones que apareceram, o primeiro se o Stash possuir qualquer quantidade de produtos abaixo do controlado, e o segundo sinalizando que o Stash está com o compartilhamento ativo.

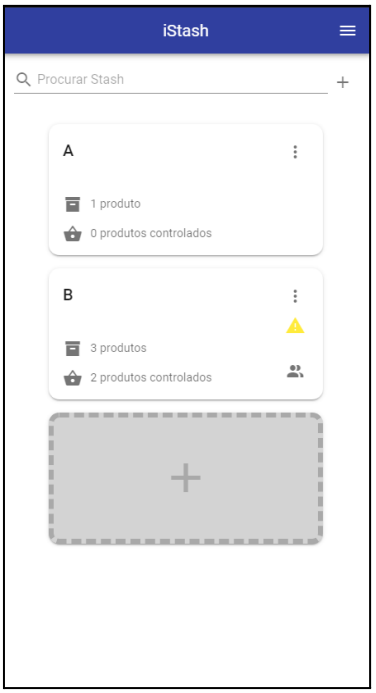


Figura 4.39 - Tela da lista de Stashes

Fonte: O Autor (2022)

```vue
<template>
  <v-card class="v-btn stash-button" @click="onclick">
    <v-card-title class="pb-0">{{ stash.name }}</v-card-title>
    <div
      :class="['top-right', $vuetify.breakpoint.xs && 'small-menu-container']"
    >
      <v-btn icon v-on:click.stop v-if="showWarning">
        <v-icon color="yellow">mdi-alert</v-icon>
      </v-btn>
      <v-menu offset-y>
        <template v-slot:activator="{ on, attrs }">
          <v-btn icon v-on:click.stop v-on="on" v-bind="attrs">
            <v-icon>mdi-dots-vertical</v-icon>
          </v-btn>
        </template>
        <v-list>
          <v-list-item>
            <v-list-item @click="handleRemove">Excluir</v-list-item>
          </v-list-item>
        </v-list>
      </v-menu>
    </div>

    <div class="bottom-right" v-if="stash.shared">
      <v-btn icon v-on:click.stop @click="handleShowUsers">
        <v-icon>mdi-account-multiple</v-icon>
      </v-btn>
    </div>

    <v-card-subtitle
      class="pt-0 text-capitalize"
      :style="!showCreator && 'visibility: hidden'"
    >
      {{ $t("message.createdby", { name: owner.name }) }}
    </v-card-subtitle>

    <v-card-text class="pb-2 text-lowercase product-label">
      <v-icon>mdi-archive</v-icon>
      {{ stash.products.length }}
      {{ $tc("keys.product", stash.products.length) }}
    </v-card-text>
    <v-card-text class="pt-1 text-lowercase product-label">
      <v-icon>mdi-basket</v-icon>
      {{ controlledProducts.length }}
      {{ $tc("keys.controlledproduct", controlledProducts.length) }}
    </v-card-text>
  </v-card>
</template>
```

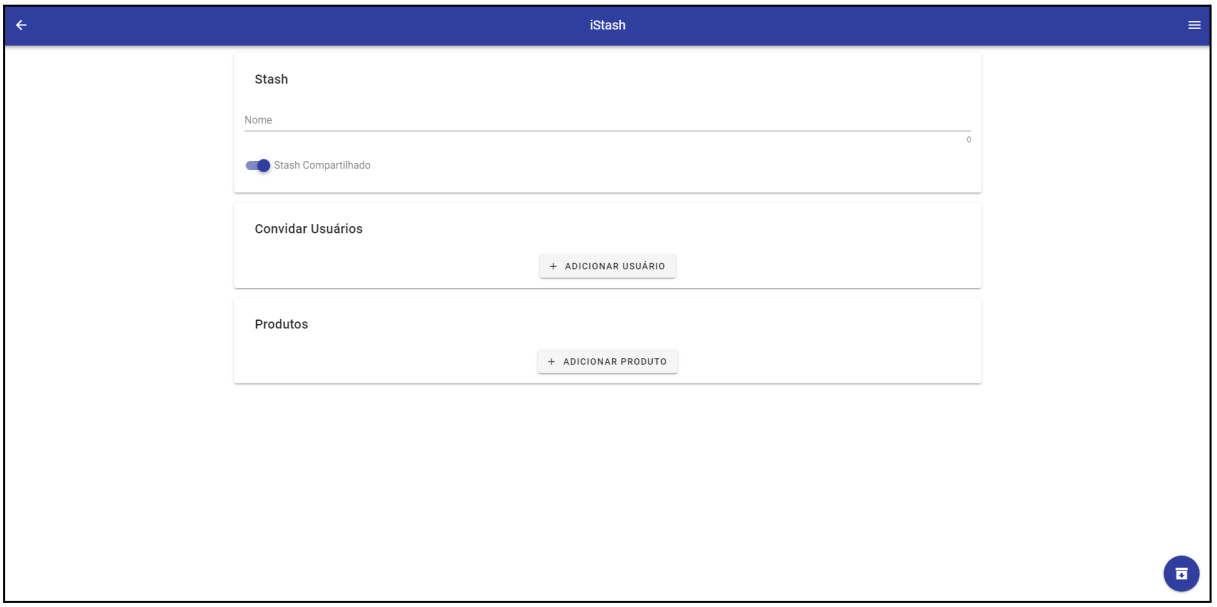
Figura 4.40 - Componente StashButton.vue

Fonte: O Autor (2022)

O botão de Stash, mostrado na Figura 4.40 usa um “v-card” do Vuetify como container, e usa os próprios subcomponentes como “v-card-title”, “v-card-subtitle” e “v-card-text” para renderizar os diferentes tipos de texto e o componente “v-menu” para mostrar o menu flutuante.

4.2.3.4.4. Tela de criação de Stash

A tela de criação normalmente é exibida como uma partição só, mas em telas menores é divida em três partes. A primeira define o nome do Stash e se ele será compartilhado ou não. Caso afirmativo, ele passa para a segunda tela, onde o usuário pode procurar pelo email de outros usuários cadastrados para dar acesso a eles, caso não, o usuário pulará direto para a última partição, onde é possível adicionar produtos antes da criação do Stash.



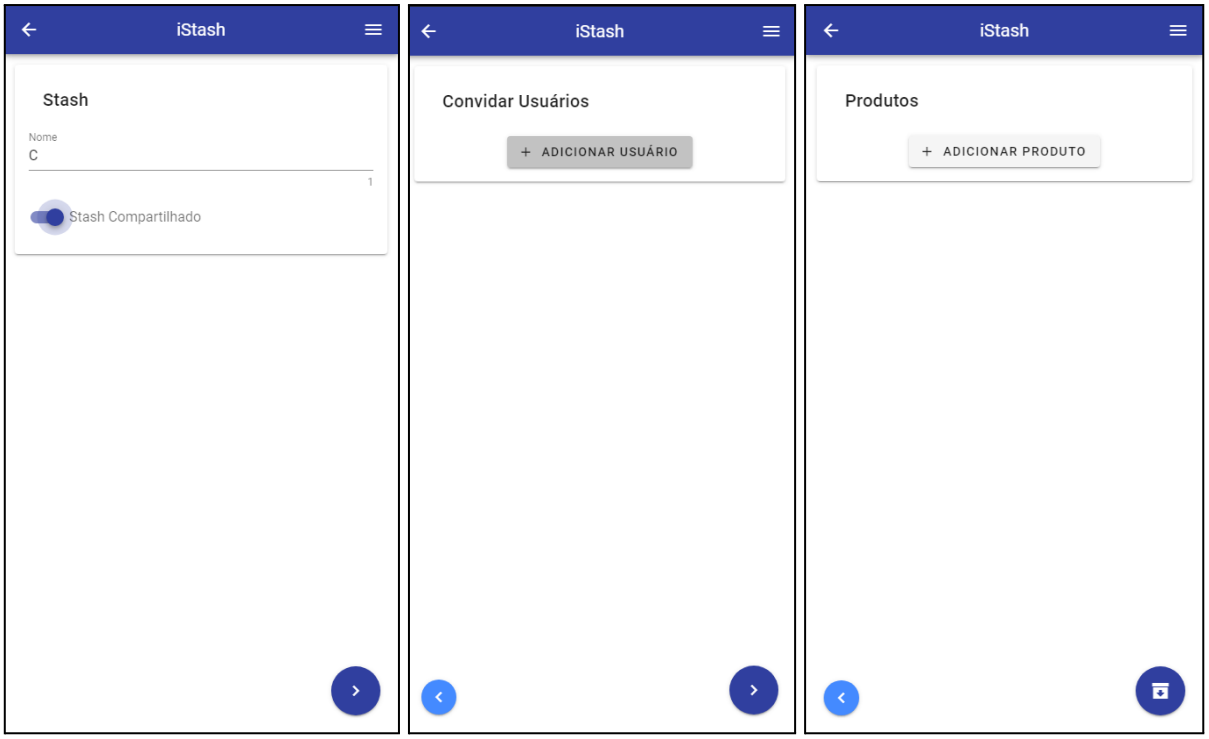


Figura 4.41 - Tela da criação de Stash (PC e Mobile)

Fonte: O Autor (2022)

Componentes novos do Vuetify que foram usados nessa tela são o “v-switch” que adiciona uma chave de ativação/desativação, e o “v-dialog”, que permite a adição de diálogos, ou modais, para mostrar conteúdo sobre a tela atual. Foram nesses diálogos a primeira vez que se foi usado outro conceito importante do Vue, os slots/templates.

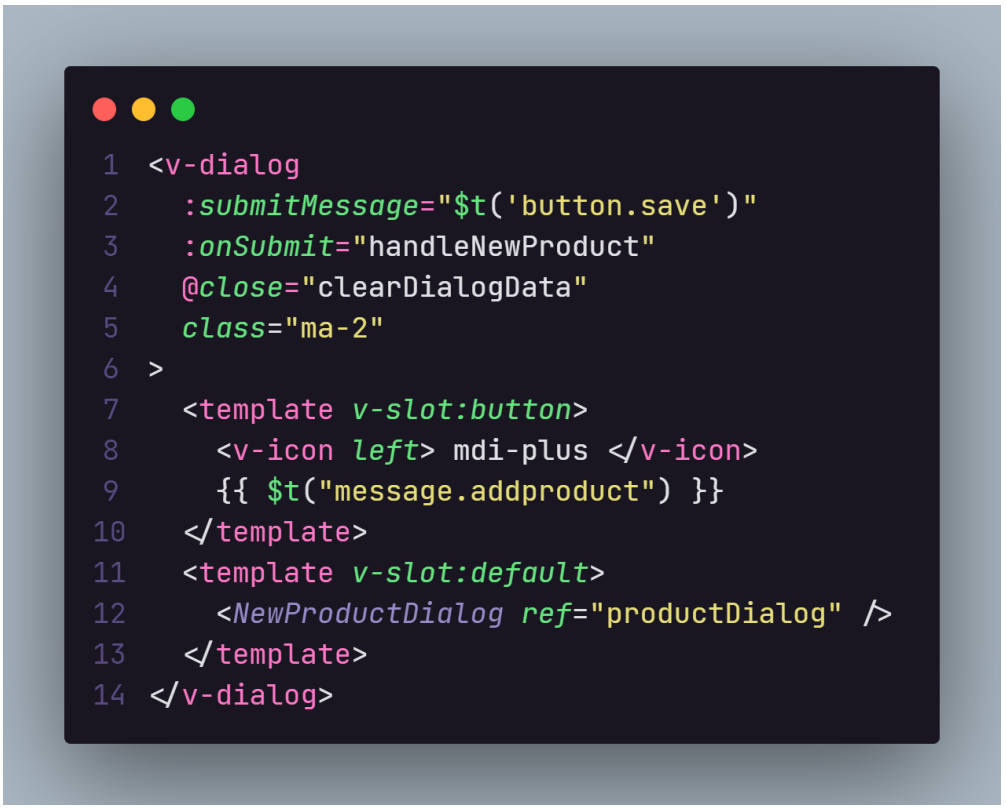
Os slots são tags especiais do Vue que permitem que algum outro conteúdo, de fora do componente, seja adicionado no lugar do slot. Eles também podem ser nomeados, permitindo o uso de mais de um slot dentro de um único componente.

No caso do “v-dialog” das figuras 4.42 e 4.43, são utilizados o “v-slot:button” para definir o componente que será o ativador do diálogo e o “v-slot:default” para o conteúdo do diálogo.

```
<v-dialog
  :submitMessage="$t('button.save')"
  :onSubmit="handleAddUser"
  @close="clearDialogData"
  class="ma-2"
>
  <template v-slot:button>
    <v-icon left> mdi-plus </v-icon>
    {{ $t("message.adduser") }}
  </template>
  <template v-slot:default>
    <InviteUserDialog :usersList="users" ref="userDialog" />
  </template>
</v-dialog>
```

Figura 4.42 - Diálogo InviteUserDialog - CreateStash.vue

Fonte: O Autor (2022)



```
<v-dialog
  :submitMessage="$t('button.save')"
  :onSubmit="handleNewProduct"
  @close="clearDialogData"
  class="ma-2"
>
  <template v-slot:button>
    <v-icon left> mdi-plus </v-icon>
    {{ $t("message.addproduct") }}
  </template>
  <template v-slot:default>
    <NewProductDialog ref="productDialog" />
  </template>
</v-dialog>
```

Figura 4.43 - Diálogo NewProductDialog - CreateStash.vue

Fonte: O Autor (2022)

O primeiro diálogo, para convidar usuários para o Stash, consiste apenas de uma caixa de texto onde o usuário pode escrever o email da pessoa que esse quer convidar. Ao valor da entrada ser alterado, o programa busca por usuários salvos na coleção “users” cujo email coincide com o procurado.

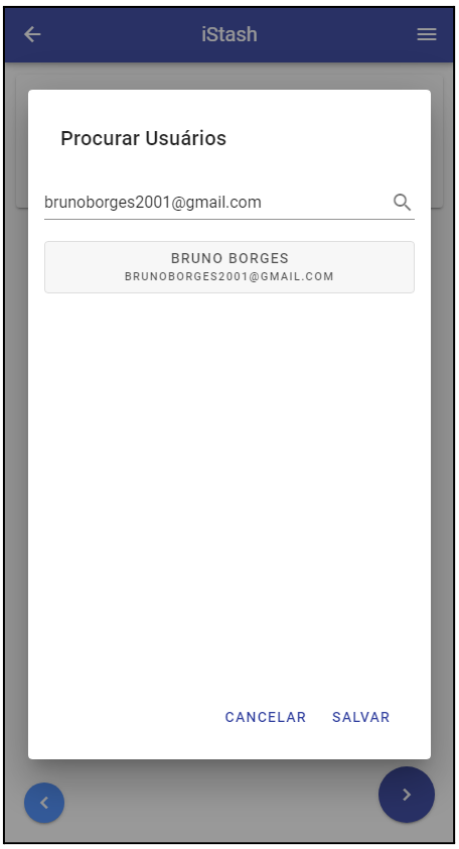


Figura 4.44 - Diálogo de procura de usuários

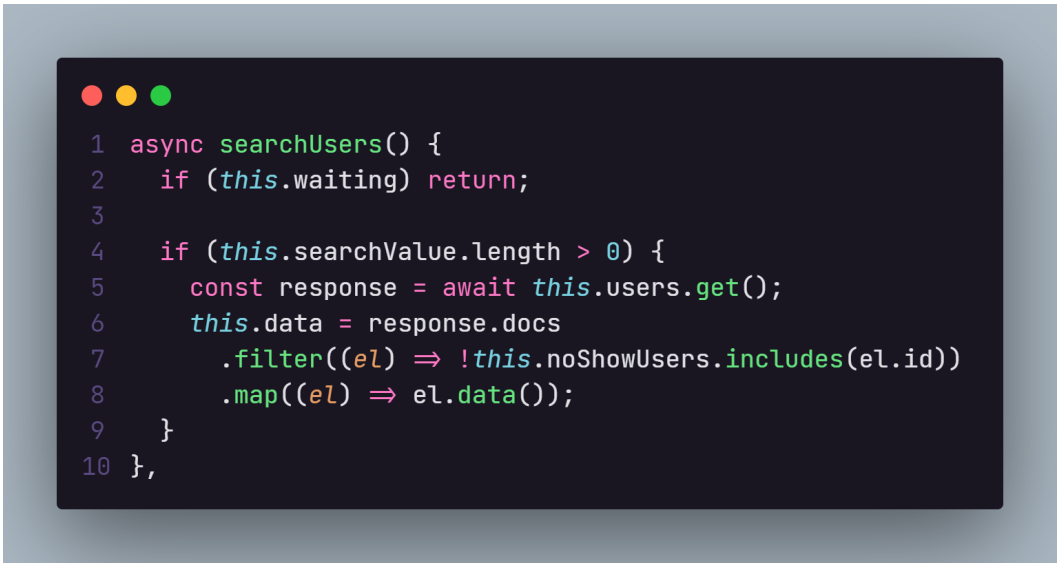
Fonte: O Autor (2022)

Para evitar um excesso de consultas ao alterar a entrada muito rápido, foi implementado uma estratégia de debounce que limita o número de consultas a uma requisição a cada três segundos. Essa lógica foi implementada dentro de um watch do searchValue. Os watches são métodos executados sempre que a variável de mesmo nome tem seu valor alterado.

```
searchValue(value) {
  if (value.length === 0) {
    this.data = [];
  }

  this.searchUsers();

  if (!this.waiting) {
    setTimeout(() => {
      this.waiting = false;
      this.searchUsers();
    }, 3000);
  }

  this.waiting = true;
}
```

Figura 4.45 - Watch searchValue - InviteUserDialog.vue

Fonte: O Autor (2022)



```
async searchUsers() {
  if (this.waiting) return;

  if (this.searchValue.length > 0) {
    const response = await this.users.get();
    this.data = response.docs
      .filter((el) => !this.noShowUsers.includes(el.id))
      .map((el) => el.data());
  }
},
```

Figura 4.46 - Método searchUsers - InviteUserDialog.vue

Fonte: O Autor (2022)

O segundo diálogo usado nessa tela é o diálogo de adicionar produto. Ele é um formulário que pede o nome, quantidade, unidade e se o produto será controlado ou não; pedindo a quantidade de controle caso positivo. Ao salvar as ações em ambos diálogos, os dados são recuperados através do método getData() que pode ser acessado pelas refs que são definidas no componente. Em componentes Vue, o uso de “ref” permite o acesso a todos os dados, métodos e valores computed do componente referenciado pelo componente pai.

Para o componente InviteUserDialog.vue, o getData apenas retorna os dados do usuário selecionado junto com userStatus 1 (valor de usuário convidado no Enum). Porém, no NewProductDialog.vue, os valores são adicionados a um id criado pelo método uuidv4() do módulo uuid e os atributos lastUpdatedBy (com o nome do usuário atual) e lastUpdatedBy (com a data atual).

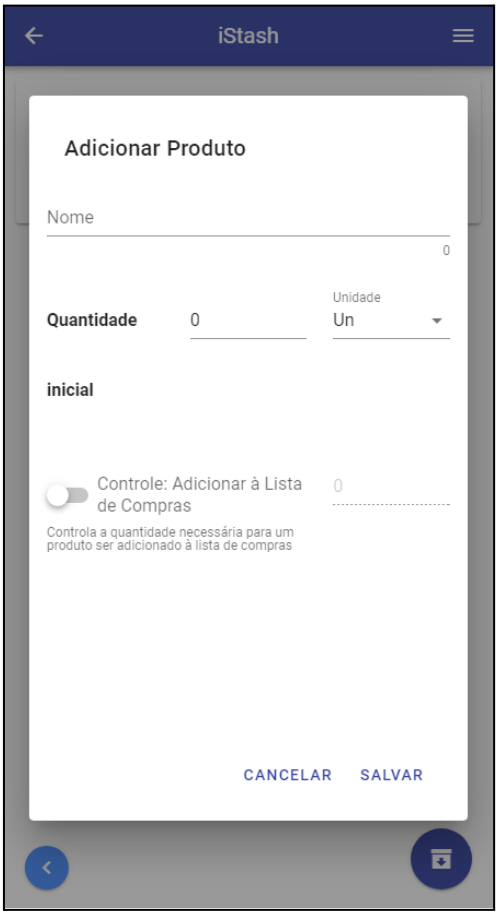


Figura 4.47 - Diálogo de criação de produto

Fonte: O Autor (2022)

```
getData() {
  if (this.validateData()) {
    const { name, quantity, unit, slControl, slQuantity } = this;

    let convertedQuantity = 0;
    let convertedSlQuantity = 0;

    if (unit === "Un") {
      convertedQuantity = parseInt(quantity);
      convertedSlQuantity = parseInt(slQuantity);
    } else {
      convertedQuantity = parseFloat(quantity);
      convertedSlQuantity = parseFloat(slQuantity);
    }

    return {
      id: this.id ?? uuidv4(),
      name,
      quantity: convertedQuantity,
      unit,
      rule: slControl ? convertedSlQuantity : null,
      lastUpdatedBy: this.$store.state.currentUser.name,
      lastUpdatedAt: new Date(),
    };
  }

  return false;
},
```

Figura 4.48 - Método getData - NewProductDialog.vue

Fonte: O Autor (2022)

Ao final da criação, quando o usuário clica no botão para salvar, um novo Stash é instanciado e então é usado o método setValues para passar os valores que foram passados durante o processo, e então, dentro do método setValues, o Stash é adicionado à store e posteriormente passado para o banco de dados, além de redirecionar o usuário direto para a tela de detalhes do novo Stash.

```
saveData() {
  const { uid, name } = this.currentUser;
  const { name: stashName, shared, users, products } = this;

  const newStash = new Stash();

  newStash.setValues(
    stashName,
    shared,
    [{ uid, name, userStatus: -1 }, ...users],
    products,
    navigator.onLine ? 0 : -1
  );
},
```

Figura 4.49 - Método saveData - CreateStash.vue

Fonte: O Autor (2022)

```
setValues(
  _name = "",
  _shared = false,
  _users: User[] = [],
  _products: Product[] = [],
  _version = 0
) {
  this.name = _name;
  this.shared = _shared;
  this.usersInfo = _users;

  this.users = _users
    .filter((el) => el.userStatus !== INVITED && el.userStatus !== REJECTED)
    .map((el) => el.uid);

  this.invites = _users
    .filter((el) => el.userStatus === INVITED)
    .map((el) => el.uid);

  this.products = _products;

  this.version = _version;

  this.update();

  setTimeout(() => router.replace("/stash/" + this.id));
}
```

Figura 4.50 - Método setValues - Stash.ts

Fonte: O Autor (2022)

Uma característica importante em todos os métodos da classe Stash é o método update. Esse método atualiza os dados da classe na store e, caso o usuário esteja online, também incrementa a versão do Stash usando o método debounce, para agrupar mudanças sucessivas em uma única versão, e não aumentar o valor pra cada uma.

```
update() {
  store.commit("updateStash", { id: this.id, value: this.buildTemplate() });
  if (navigator.onLine) {
    debounce(() => {
      if (this.version < 0) this.version - 1;
      else this.version += 1;
    }, 1000);
  }
}
```

Figura 4.51 - Método update - Stash.ts

Fonte: O Autor (2022)

4.2.3.4.5. Tela de detalhes

Quando um Stash é criado, o usuário é redirecionado para a tela de detalhes. Uma página dividida em quatro abas: Produtos, Lista de compras, Usuários e Configurações.

As abas são adicionadas através dos componentes “`v-tabs-items`” e “`v-tab-item`” renderizando cada componente como conteúdo do “`v-tab-item`”. Os botões são criados a partir do array tabsList, que possui o label (nome mostrado no botão) e o icon (o ícone renderizado). Além disso, como é possível observar na figura 4.45, o nome da aba de lista de compras é alterado apenas para “Compras” caso o tamanho da tela do dispositivo seja muito pequena para caber o nome inteiro.

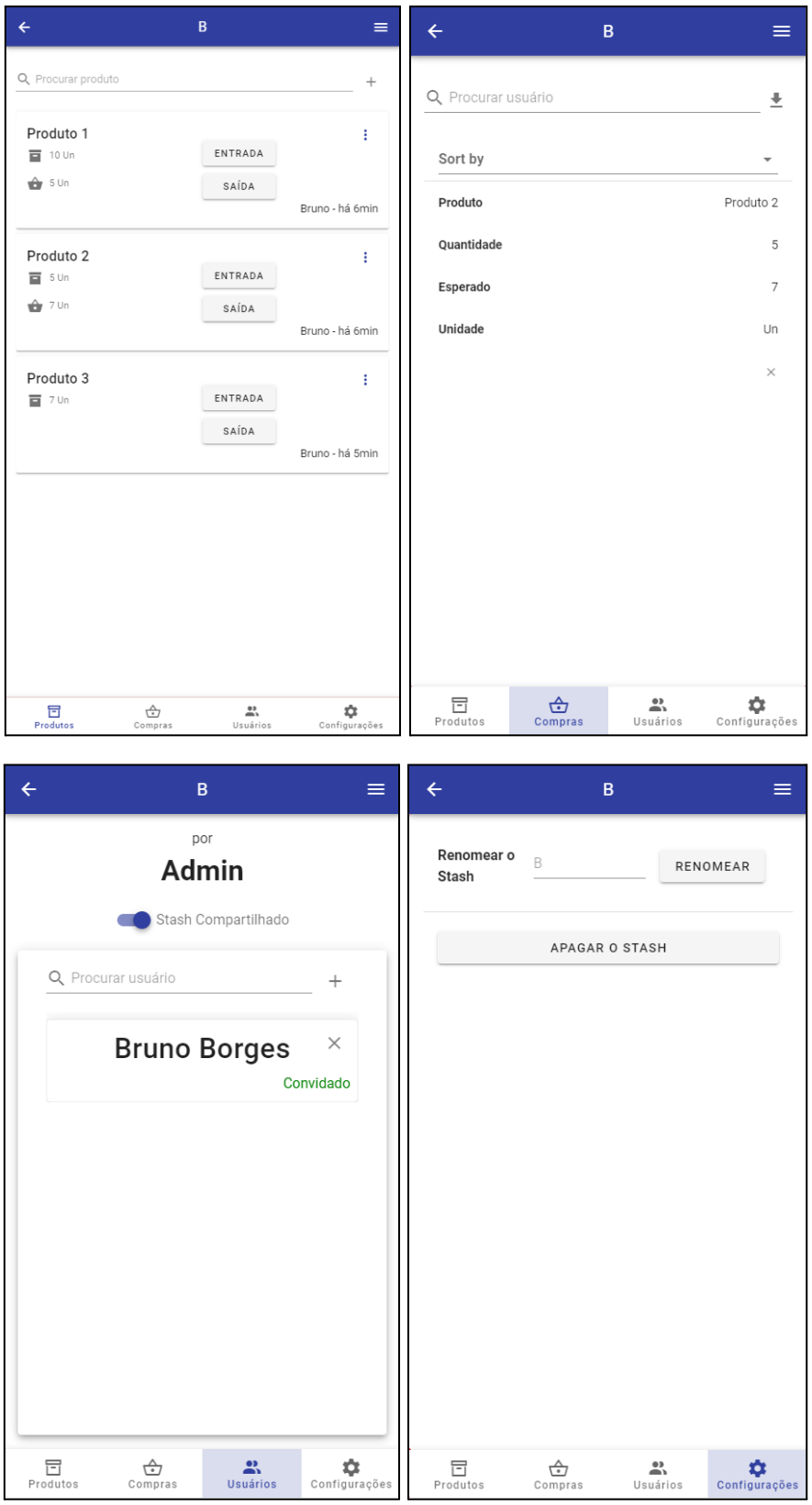


Figura 4.52 - Tela de detalhes do Stash

Fonte: O Autor (2022)

```
tabsList(): { label: string; icon: string }[] {
  return [
    { label: "product", icon: "mdi-archive-outline" },
    { label: this.shoppingListLabel, icon: "mdi-basket-outline" },
    { label: "user", icon: "mdi-account-multiple" },
    { label: "setting", icon: "mdi-cog" },
  ];
},
shoppingListLabel(): string {
  if (this.$vuetify.breakpoint.xs) return "shoppinglistshort";
  return "shoppinglist";
},
```

Figura 4.53 - Computed prop tabsList e shoppingListLabel - Stash/index.vue

Fonte: O Autor (2022)

```
<v-card class="content">
  <v-tabs-items class="full-height" v-model="tab" v-if="stash">
    <v-tab-item :key="tabsList[0].label" class="full-height">
      <product-view :stash="stash" class="full-height" />
    </v-tab-item>
    <v-tab-item :key="tabsList[1].label" class="full-height">
      <shopping-list :stash="stash" class="full-height" />
    </v-tab-item>
    <v-tab-item :key="tabsList[2].label" class="full-height">
      <user-view :stash="stash" class="full-height" />
    </v-tab-item>
    <v-tab-item :key="tabsList[3].label" class="full-height">
      <settings-view :stash="stash" class="full-height" />
    </v-tab-item>
  </v-tabs-items>
  <LoadingIndicator v-else />
</v-card>
<v-bottom-navigation v-model="tab" grow color="primary">
  <v-btn v-for="item in tabsList" :key="item.label">
    <span>{{ $tc(`keys.${item.label}`, 2) }}</span>

    <v-icon>{{ item.icon }}</v-icon>
  </v-btn>
</v-bottom-navigation>
```

Figura 4.54 - Template das abas da tela de Stash - Stash/index.vue

Fonte: O Autor (2022)

4.2.3.4.5.1. Aba de Produtos

A primeira aba da tela de detalhes é a de produtos, nela serão mostrados todos os produtos cadastrados com suas informações. A aba consiste de uma caixa de procura para filtrar produtos junto a um botão para adicionar novos. Assim como na criação de Stash, esse botão também abre o diálogo de criação.

Na lista de produtos, cada um é renderizado com o seu nome, a quantidade do produto e a de controle, também é mostrado o último usuário que editou cada produto e a quanto tempo. O botão também possui botões para registrar entrada e saída de produtos. Caso desejado, o usuário poderá mudar a quantidade diretamente na opção de edição presente no menu flutuante, junto a opção de apagar o produto, no canto direito superior do produto.

Figura 4.55 - Renderização dos produtos

Fonte: O Autor (2022)

A figura 4.55 mostra uma outra propriedade de componentes Vue, o “v-for”: como o nome sugere, ele serve para renderizar um componente repetidas vezes baseado em um vetor dado. Nesse caso, o componente renderizado é o card com as informações do produto.

Para evitar conflitos e carregar informações antigas antes de atualizar as novas no banco de dados, foram criados dois sistemas que garantem que o usuário verá sempre a informação mais atualizada. A primeira foi a criação da flag updateData,

que desabilita temporariamente o método loadStashes até que a atualização dos valores aconteça. Outra foi a implementação do objeto overrideData, que é um objeto com todos os atributos que são visíveis ao usuário, ele é atualizado com valores editados localmente e zerado quando novos dados são carregados; enquanto o objeto possuir valores não nulos, os mesmos são renderizados no lugar dos valores carregados. O primeiro é usado quando o usuário abre o diálogo de criação de produtos e o segundo a qualquer momento que os dados forem alterados, principalmente quando o usuário salva os dados do diálogo de edição (que é o mesmo que o diálogo de criação, porém com os valores atuais preenchidos). O método handleEdit, na figura 4.49, ativa o diálogo de edição e usa do método "`$nextTick`" para usar o método "`setEditData`" do ref de NewProductDialog para colocar os dados salvos nos campos apenas depois que o diálogo foi renderizado.

```
handleOpenDialog() {
  this.$store.commit("disableUpdateData");
},
handleNewProduct() {
  const value = (this.$refs.productDialog as any).getData();

  if (!value) return false;

  this.stash?.addProduct(value);

  this.$store.commit("enableUpdateData");

  return true;
}
```

Figura 4.56 - Ativação e desativação da flag updateData - ProductView.vue

Fonte: O Autor (2022)

```
handleEdit() {
  this.editDialog = true;
  this.$nextTick(() => this.$refs.productDialog.setEditData(this.product));
}
```

Figura 4.57 - Método handleEdit - ProductCard.vue

Fonte: O Autor (2022)

```
handleEditSubmit() {
  const value = this.$refs.productDialog.getData();

  if (!value) return false;
  this.overrideData = {
    ...this.overrideData,
    ...value,
    lastUpdatedAt: new Date(),
  };

  this.stashRef.updateProduct(this.product.id, value);

  this.$refs.productCounter?.override(value.quantity);

  this.editDialog = false;
  return true;
}
```

Figura 4.58 - Método handleEditSubmit - ProductCard.vue

Fonte: O Autor (2022)

4.2.3.4.5.2. Aba de Compras

A segunda aba mostra todos os produtos cadastrados com quantidade menor que a quantidade de controle cadastrada; caso ela seja maior que a cadastrada, ou o controle esteja desativado, os produtos não são mostrados. Além disso, a tabela pode ser exportada como PDF, através do botão ao lado da barra de pesquisa, para ser compartilhada como o usuário desejar.

A tabela foi renderizada usando o componente de tabela do Vuetify: “`v-data-table`”. Dados o array de valores e um objeto de cabeçalhos com a string demonstrando o texto do cabeçalho e outra que define a chave do valor no objeto, o componente automaticamente seleciona os dados corretos de cada item do vetor e mostra em uma tabela customizada e com suporte a filtros, ordenação e paginação.

```vue
<v-data-table
  :headers="headers"
  :items="shoppingList"
>
  <template v-slot:[`item.actions`]="{ item }">
    <v-icon
      class="text-xs-right"
      small
      @click="ignoreProductList.push(item.id)"
    >
      mdi-close
    </v-icon>
  </template>
  <template v-slot:[`footer.page-text`]="items">
    {{ items.pageStart }} - {{ items.pageStop }} {{ $t("keys.of") }}
    {{ items.itemsLength }}
  </template>
</v-data-table>
```

Figura 4.59 - Componente da tabela de produtos - ShoppingList.vue

Fonte: O Autor (2022)

```vue
headers(): DataTableHeader[] {
  return [
    { text: this.$tc("keys.product", 1) as string, value: "name" },
    { text: this.$t("keys.quantity") as string, value: "quantity" },
    { text: this.$t("keys.expected") as string, value: "rule" },
    { text: this.$t("keys.unit") as string, value: "unit" },
    { text: "", value: "actions", sortable: false },
  ];
}
```

Figura 4.60 - Objeto de cabeçalhos - ShoppingList.vue

Fonte: O Autor (2022)

Já a criação da tabela como PDF foi feita usando a biblioteca “`jspdf`” e o seu plugin “`jspdf-autotable`”, específico para criação de tabelas. O módulo recebe um array de strings para o cabeçalho e um segundo array com os itens selecionados, além das estilizações, e cria automaticamente um arquivo .pdf e faz o download do mesmo.

```
createPDF() {
  const pdfName = this.$t("keys.shoppinglist");
  const head = [this.headers.map((el) => el.text)];
  const body = this.shoppingList.map((el) => [
    el.name,
    el.quantity,
    el.rule,
    el.unit,
  ]);
  var doc = new jsPDF();
  autoTable(doc, {
    head,
    body,
    startY: 10,
    styles: { fontSize: 14, cellPadding: 4 },
  });
  doc.save(pdfName + ".pdf");
}
```

Figura 4.61 - Método createPDF - ShoppingList.vue

Fonte: O Autor (2022)

4.2.3.4.5.3. Aba de Usuários

A terceira aba controla o acesso de usuários, nela estão o nome do criador e a lista de usuários convidados e que já aceitaram o convite. Apenas para o dono do Stash, estão as opções para ativar ou desativar o compartilhamento e também para convidar novos usuário ou remover existentes.

Para a opção de convidar, é usado o mesmo diálogo InviteUserDialog.vue que foi usado na criação do Stash. A troca entre compartilhamento ativo ou não é feita através do método setShared da própria classe Stash.

```
1 setShared(_shared: boolean) {
2   store.commit("disableUpdateData");
3
4   this.shared = _shared;
5
6   this.update();
7 }
```

Figura 4.62 - Método setShared - Stash.ts

Fonte: O Autor (2022)

4.2.3.4.5.4. Aba de Configurações

A última aba apresenta apenas algumas configurações gerais do Stash. No caso, as configurações são as opções para renomear o Stash e para apagar o mesmo. Os componentes usados já foram discutidos em seções anteriores, porém foi adicionado um comportamento específico para o botão de apagar o Stash: foi adicionado um passo de confirmação ao clicar no botão, antes de realmente apagar o Stash. Também é usado um novo atributo dos componentes Vue, o "v-click-outside", o método atribuído a esse atributo é executado sempre que o usuário clica em qualquer lugar fora do componente.

```
<v-btn
  class="flex-grow-1"
  @click="handleDelete"
  v-click-outside="cancelDelete"
>
  {{
    deleteConfirm ? $t("message.areyousure") : $t("message.deletestash")
  }}
</v-btn>
```

Figura 4.63 - Botão para apagar Stash - SettingsView.vue

Fonte: O Autor (2022)

```
handleDelete() {
  if (this.deleteConfirm) {
    this.stash.remove();
    this.$router.replace("/");
  } else {
    this.deleteConfirm = true;
  }
}
```

Figura 4.64 - Método handleDelete - SettingsView.vue

Fonte: O Autor (2022)

```
remove() {
  store.commit("removeStash", { id: this.id });
}
```

Figura 4.65 - Método remove - Stash.ts

Fonte: O Autor (2022)

O processo para renomear o Stash é mais simples. O valor da caixa de texto, ele é checado, com as regras de tamanho de string e então renomeado através do método “`renameStash`” da classe Stash.

```
handleRename() {
  if (!this.newStashName) {
    return this.$notify({
      group: "center",
      title: this.$t("keys.error") as string,
      text: this.$t("message.fillrequiredfields") as string,
      type: "error",
    });
  }

  if (this.newStashName.length > 25) return;

  this.stash.renameStash(this.newStashName);

  this.newStashName = "";
}
```

Figura 4.66 - Método handleRename - SettingsView.vue

Fonte: O Autor (2022)

Figura 4.67 - Método renameStash - Stash.ts

Fonte: O Autor (2022)

4.2.3.4.6. Convites

A tela de notificações/convites é habilitada no Drawer apenas quando existe algum disponível para o usuário. Ele mostra uma lista com a mensagem de convite de cada um e um botão para aceitar e outro para recusar, definidos no componente InviteCard.vue.

Figura 4.68 - Tela de controle de mudanças

Fonte: O Autor (2022)

Os métodos de aceite e rejeite dos convites fazem checagens para possíveis erros e retornar um código numérico para cada um - que posteriormente retornam uma mensagem para o usuário de acordo com o erro - e alteram o estado do usuário no Stash.

```
acceptInvite(_id: string) {
  if (!this.invites.includes(_id)) return 999;

  if (!this.usersInfo.find((el) => el.uid === _id)) return 1;

  store.commit("disableUpdateData");

  this.invites = this.invites.filter((el) => el !== _id);
  this.users.push(_id);

  const userIndex = this.usersInfo.findIndex((el) => el.uid === _id);
  this.usersInfo[userIndex].userStatus = ACCEPTED;

  this.update();

  return 0;
}
```

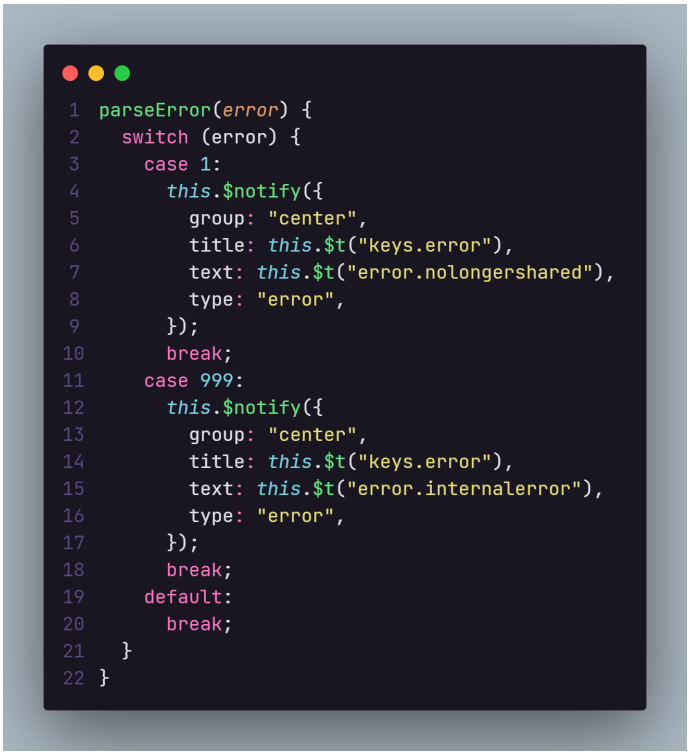
Figura 4.69 - Método acceptInvite - Stash.ts

Fonte: O Autor (2022)

```
rejectInvite(_id: string, _del = false) {
  if (!this.invites.includes(_id)) return 999;

  if (!this.usersInfo.find((el) => el.uid === _id)) return 1;

  store.commit("disableUpdateData");

  this.invites = this.invites.filter((el) => el !== _id);

  if (_del) {
    this.usersInfo = this.usersInfo.filter((el) => el.uid !== _id);
  } else {
    const userIndex = this.usersInfo.findIndex((el) => el.uid === _id);
    this.usersInfo[userIndex].userStatus = REJECTED;
  }

  this.update();

  return 0;
}
```

Figura 4.70 - Método rejectInvite - Stash.ts

Fonte: O Autor (2022)



```
parseError(error) {
  switch (error) {
    case 1:
      this.$notify({
        group: "center",
        title: this.$t("keys.error"),
        text: this.$t("error.nolongershared"),
        type: "error",
      });
      break;
    case 999:
      this.$notify({
        group: "center",
        title: this.$t("keys.error"),
        text: this.$t("error.internalerror"),
        type: "error",
      });
      break;
    default:
      break;
  }
}
```

Figura 4.71 - Método parseError - InviteCard.vue

Fonte: O Autor (2022)

4.2.3.4.7. Alterações e Conflitos

A tela de alterações usa dos valores obtidos e salvos na variável diffs (na store) para mostrar as alterações conflitantes ocorridas entre versões e separadas por Stashes. Cada Stash definido no diff é renderizado como uma instância do componente ChangeGroupCard.vue.

Nesse componente são mostradas todas as alterações com o nome do atributo seguidos dos valores antigos e valores novos separados por uma seta (►). Em cada um desses grupos, o usuário pode escolher entre manter as alterações locais (antigas) e sobrescrever as informações do banco de dados (botão esquerdo), ou substituir os dados locais e salvar os dados novos (botão direito).

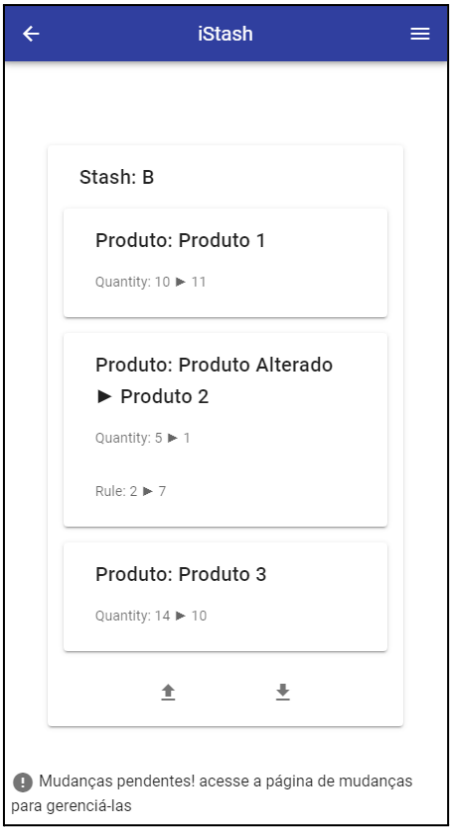


Figura 4.72 - Tela de controle de mudanças

Fonte: O Autor (2022)

Caso o usuário escolha o botão da direita, o método “`handleOverride`” é acionado e os valores locais são recuperados da store e salvos novamente, para que os dados do banco de dados sejam alterados para os locais.

```
handleOverride() {
  const newData = { ...this.$store.getters.getStash(this.id) };
  newData.version = this.stash.version + 1;

  this.$store.commit("updateStash", { id: this.id, value: newData });
  this.clearChange();
}
```

Figura 4.73 - Método handleOverride - ChangeGroupCard.vue

Fonte: O Autor (2022)

Porém, se o usuário escolhe a segunda opção, o método “handleSave” é chamado e os dados salvos no vetor newData (na store, que foi passado como o atributo “stash” do componente) é atualizado na store.

Figura 4.74 - Método handleSave - ChangeGroupCard.vue

Fonte: O Autor (2022)

Em ambos os casos, após a atualização, o método clearChange é chamado para apagar as mudanças do vetor diffs e os dados do vetor newData.

4.2.3.5. Hospedagem e Desenvolvimento CI/CD

Manter uma aplicação no ar e sempre na sua última versão é uma parte crucial em um programa de sucesso. Para isso, o Firebase fornece um serviço de hospedagem gratuito e ainda permite a integração com o GitHub para que sempre que alterações sejam commitadas para o repositório, uma compilação automática seja executada e seja realizada a atualização da aplicação automaticamente, criando um fluxo CI/CD (ou integração contínua e entrega contínua).

Atualmente a aplicação está hospedada e pode ser acessada pelo link https://istash.web.app e o seu código fonte está no repositório público do github que pode ser acessado através do link https://github.com/bruno-borges-2001/iStash.

5. CONCLUSÕES E TRABALHOS FUTUROS

A proposta inicial do projeto de desenvolver um aplicativo que auxilie no controle de estoque foi finalizada com sucesso. O aplicativo foi disponibilizado em um link público e aberto para testes. Quando este trabalho foi iniciado, já se imaginava dar acesso irrestrito e ser uma aplicação gratuita para o uso de qualquer pessoa.

Apesar de o aplicativo estar funcional, é preciso ressaltar que o aplicativo não está totalmente completo e ainda possui funcionamento limitado. A estética do aplicativo não ficou exatamente como desejado e precisaria de uma revisão para ficar perfeita, principalmente em relação a experiência do usuário. Além disso, a internacionalização não está totalmente perfeita e muitos bugs ainda são possíveis de ocorrer.

Contudo, seria interessante testar o aplicativo com usuários reais e receber o feedback dos mesmos. Além disso, caso qualquer problema seja encontrado é necessário gerar novas versões para resolvê-los, uma vez que o aplicativo só foi testado em dispositivos android, com um único modelo de smartphone, e no browser, também em um único computador. Também é importante continuar o desenvolvimento e melhoria da aplicação, talvez até adicionando novas funcionalidades no futuro.

REFERÊNCIAS

[1] PROVIN, Diego Telles; SELLITTO, Miguel Afonso. POLÍTICA DE COMPRA E REPOSIÇÃO DE ESTOQUES EM UMA EMPRESA DE PEQUENO PORTE DO RAMO ATACADISTA DE MATERIAIS PARA CONSTRUÇÃO CIVIL. Revista Gestão Industrial, Campus Ponta Grossa - Paraná - Brasil, ano 2011, v. 07, n. 02, ed. 1808-0448, p. 187-200, 20 jun. 2011. DOI 10.3895/S1808-04482011000200010. Disponível em:

https://periodicos.utfpr.edu.br/revistagi/article/viewFile/631/674. Acesso em: 30 nov. 2021.

[2] Edna Lúcia da Silva e Estera Muszkat Menezes. “Metodologia da pesquisa e elaboração de dissertação”, 2001.

[3] INTRODUCTION - Vue.js. [S. l.], 201-. Disponível em:

https://vuejs.org/v2/guide/#What-is-Vue-js. Acesso em: 22 dez. 2021.

[4] GOOGLE Acquires Firebase To Help Developers Build Better Real-Time Apps | TechCrunch. [S. l.], 21 out. 2014. Disponível em:

https://techcrunch.com/2014/10/21/google-acquires-firebase-to-help-developers-build-better-realtime-apps/. Acesso em: 11 jan. 2022.

[5] Firebase Adds Web Hosting To Its Database Platform | TechCrunch. [S. l.], 13 mai. 2014. Disponível em:

https://techcrunch.com/2014/05/13/firebase-adds-web-hosting-to-its-database-platform/. Acesso em: 11 jan. 2022.

[6] Firebase Authentication | Firebase Documentation. [S. l.], 201-. Disponível em:

https://firebase.google.com/docs/auth/. Acesso em: 11 jan. 2022.

[7] The Firebase Blog: Firebase expands to become a unified app platform. [S. l.], mar. 2016. Disponível em: Acesso em: 11 jan. 2022.

https://firebase.googleblog.com/2016/05/firebase-expands-to-become-unified-app-platform.html. Acesso em: 11 jan. 2022.

[8] Google Announces Firestore, a Document Database. [S. l.], out. 2017. Disponível em:

https://www.infoq.com/news/2017/10/google-firestore/. Acesso em: 11 jan. 2022.

[9] PROTÓTIPOS de objetos. In: MDN Web Docs. [S. l.], 30 jan. 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/Object_prototypes . Acesso em: 31 jan. 2022.

[10] VOSKOGLOU, Christina; STEPHENS, Jed; KORAKITIS, Konstantinos; MUIR, Richard; IACOZZA, Sara. Programming Language Community: An Update. State of The Developer Nation 19th Edition: The latest trends from our Q3 2020 survey of 17,000+ developers, SlashData, ed. 19, p. 16-20, Outubro 2020. Disponível em: https://slashdata-website-cms.s3.amazonaws.com/sample_reports/y7fzAZ8e5XuKCL1Q.pdf. Acesso em: 31 jan. 2022.

[11] JAVASCRIPT.COM. [S. l.], [201-]. Disponível em: https://www.javascript.com. Acesso em: 31 jan. 2022.

[12] JavaScript. In: MDN Web Docs. [S. l.], 30 jan. 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 31 jan. 2022.

[13] SILVA, Giancarlo. O que é e como funciona a linguagem JavaScript?. Canaltech, 28 jan. 2015. Disponível em:

https://canaltech.com.br/internet/O-que-e-e-como-funciona-a-linguagem-JavaScript/. Acesso em: 31 jan. 2022.

[14] BABEL: The compiler for next generation JavaScript. [S. l.], [201-].

Disponível em: https://babeljs.io. Acesso em: 31 jan. 2022.

[15] LENON. Node.js: O que é, como funciona e quais as vantagens. [S. l.], 5 set. 2018. Disponível em: https://www.opus-software.com.br/node-js/. Acesso em: 31 jan. 2022.

[16] NODE.JS. [S. l.], [201-]. Disponível em: https://nodejs.org/en/. Acesso em: 31 jan. 2022.

[17] README.MD. In: Nodejs/node. GitHub, [2009-]. Disponível em:

https://github.com/nodejs/node/blob/master/README.md. Acesso em: 31 jan. 2022.

[18] HIGOR. Trabalhando com DOM em JavaScript. [S. l.], 2013. Disponível em: https://www.devmedia.com.br/trabalhando-com-dom-em-javascript/29039. Acesso em: 1 fev. 2022.

[19] INTRODUÇÃO ao DOM. [S. l.], 30 jan. 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction. Acesso em: 1 fev. 2022.

[20] HTMLMediaElement.playbackRate. [S. l.], 14 set. 2021. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement/playbackRate. Acesso em: 1 fev. 2022.

[21] KITCHENHAM, Barbara. “Procedures for performing systematic reviews”. Em: Keele, UK, Keele University 33.2004 (2004), pp. 1–26. Disponível em: https://www.researchgate.net/profile/Barbara-Kitchenham/publication/228756057_Procedures_for_Performing_Systematic_Reviews/links/618cfae961f09877207f8471/Procedures-for-Performing-Systematic-Reviews.pdf. Acesso em: 8 fev. 2022.

[22] BLING: Sistema ERP Online para empresas e lojas virtuais. [S. l.], [201-]. Disponível em: https://www.bling.com.br. Acesso em: 8 fev. 2022.

[23] ODOO: Open Source ERP and CRM. [S. l.], [201-]. Disponível em: https://www.odoo.com. Acesso em: 9 fev. 2022.

[24] ODOO: Chapter 1: Architecture Overview. [S. l.], [201-]. Disponível em: https://www.odoo.com/documentation/15.0/developer/howtos/rdtraining/01_architecture.html. Acesso em: 10 fev. 2022.

[25] SENIOR SISTEMAS: Sistema ERP o que é e como funciona. [S. l.], [201-]. Disponível em: https://www.senior.com.br/sistema-erp-o-que-e-e-como-funciona. Acesso em: 9 fev. 2022.

[26] NOLETO, Cairo. Blog da Trybe: Aplicações web: entenda o que são e como funcionam! [S. *l.*], 18 mar. 2020. Disponível em: https://blog.betrybe.com/desenvolvimento-web/aplicacoes-web/. Acesso em: 17 fev. 2022.

[27] NATIONS, Daniel. What Is a Web Application?. [S. l.], 24 jun. 2021. Disponível em: https://www.lifewire.com/what-is-a-web-application-3486637. Acesso em: 17 fev. 2022.

[28] TANDEL, Sayali Sunil; JAMADAR, Abhishek. Impact of Progressive Web Apps on Web App Development. International Journal of Innovative Research in Science, Engineering and Technology, Maharashtra, India, v. 7, n. 9, p. 9439-9444, set. 2018. DOI 10.15680/IJIRSET.2018.0709021. Disponível em:

https://www.researchgate.net/profile/Sayali-Tandel-2/publication/330834334_Impact_of_Progressive_Web_Apps_on_Web_App_Development/links/5c5605d3a6fdccd6b5dde018/Impact-of-Progressive-Web-Apps-on-Web-App-Development.pdf. Acesso em: 17 fev. 2022.

[29] ZOHO INVENTORY: Inventory Control Software. [S. l.], [201-]. Disponível em: https://www.zoho.com. Acesso em: 18 fev. 2022.

[30] VITE. [*S. l.*], 2020. Disponível em: https://vitejs.dev. Acesso em: 29 maio 2022.

[31] VITE Plugin PWA. [*S. l.*], 2020.

Disponível em: https://vite-plugin-pwa.netlify.app. Acesso em: 29 maio 2022.

[32] VUE I18n. [*S. l.*], 2020.

Disponível em: https://kazupon.github.io/vue-i18n/. Acesso em: 04 junho 2022.

Aplicação PWA para Gerenciamento de Estoque

Bruno D. B. Borges

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

brunoborges2001@gmail.com

Abstract. *This article describes the development process of the graduation conclusion project which objective is to create a PWA application for inventory management. System requirements, use cases, source code modeling and development, as well as research methods and technologies used will be discussed.*

Resumo. *Este artigo descreve o processo de desenvolvimento do projeto de trabalho de conclusão de curso (TCC) cujo objetivo é a criação de uma aplicação PWA para gerenciamento de estoque. Serão discutidos os requisitos do sistema, os casos de usos, a modelagem e o desenvolvimento do código fonte, além dos métodos de pesquisa e as tecnologias usadas.*

1. Introdução

Há muitos anos organização e logística são sinônimos de lucro quando se trata de controle de estoque. Um controle alto significa que uma empresa consegue usar seu estoque com eficiência e economizar o máximo possível na hora de repor seus produtos.

Apesar da implementação de um sistema de controle de estoque poder variar em complexidade desde uma tabela no excel até um sistema específico para empresas, normalmente o acesso a esses sistemas pode ser complicado.

Por esses motivos esse projeto possui o objetivo de implementar um sistema simples o bastante para que possa ser usado tanto por empresas quanto por pessoas físicas. Mas que também tenha um nível de complexidade que permita um alto nível de controle e gestão do estoque. Nesse sentido, é possível mencionar o cadastro de regras de negócio que, no sistema, permitirá a geração de listas de compras com os produtos que estiverem adequados às regras criadas. Ou também, a possibilidade de criar listas compartilhadas com outras pessoas para permitir atualizações conjuntas nas listas.

O desenvolvimento de uma aplicação de gestão de estoque permite estender as opções de alguém por organização. Disponibilizando on-line como um pwa, seria possível qualquer um com acesso à internet usar o programa e começar a controlar seus próprios produtos. Junto com a possibilidade de usar o aplicativo PWA no celular, aumentaria a mobilidade e o gerenciamento em tempo real. E com o armazenamento dos dados nas nuvens tornaria possível compartilhar os mesmos dados entre dispositivos sem esforço. Aproveitando do programa ser um PWA, também é possível permitir ao usuário a alteração dos dados enquanto o celular estiver offline, guardando os dados em cache e atualizando no banco de dados quando o dispositivo voltasse a possuir conexão com a internet.

2. Fundamentação

O projeto foi desenvolvido usando a linguagem Javascript – a linguagem mais usada no desenvolvimento web e com a maior comunidade do planeta [Voskoglou et al. 2020] – usando um framework específico para criação de aplicações web: Vue.Js.

O framework provê cracterísticas que facilitam o desenvolvimento, como a componetização e a reatividade. O primeiro permite a reutilização de partes de código com funcionalidades semelhantes em diferentes partes da aplicação; e o segundo permite que cada componente controle seu próprio estado e propriedades. Além disso, o Vue.Js junta em um único arquivo os códigos HTML, Javascript e CSS e também suporta a mistura deles para facilitar o uso dos dados das variáveis Javascript diretamente dentro do template HTML.

A aplicação web foi desenvolvido com o front-end PWA, permitindo as usuários o acesso em computadores e dispositivos móveis, possibilitando a instalação semelhante a aplicativos nativos, sem a necessidade de instalar grandes quantidades de dados. Felizmente o framework suporta a criação de aplicações com esse suporte e ainda fornece um plugin que faz toda a configuração automaticamente.

Além disso, aplicações PWA permitem o uso da cache para possibilitar o funcionamento do aplicativo enquanto offline. Esse foi outro recurso usado para que os dados alterados no aplicativo, enquanto não houvesse conexão, fossem armazenados até que a conexão fosse reestabelecida para efetuar a atualziação.

Para o controle dos dados e da autenticação foi usado o Firebase/Firestore – uma plataforma fornecida gratuitamente pela google que oferece serviços de banco de dados não relacional e autenticação, além do serviço de hospedagem, que também foi usado para colocar a aplicação no ar.

3. Estado da Arte

Para o estudo do estado da arte desse projeto, foram pesquisadas soluções existentes semelhantes à ideia da aplicação desenvolvida, seguindo com o método de Kitchenham [Kitchenham 2004] como base.

Com esse objetivo, foram usados o Google e Google Scholar como fonte de pesquisa e os termos usados são mostrados na tabela 1.

Termo de busca	Termo relacionado	Tradução (Inglês)
VueJs	Vue, Vue JS	Não se Aplica
PWA	Progressive Web Application	Não se Aplica
Software	Código, Aplicação, Programa	Não se Aplica
Gestão	Gerenciamento	Management
Estoque	Inventário	Stock/Inventory

Tabela 1. Termos de busca

Das soluções encontradas, a maior parte dos softwares eram pagos e também eram muito complexos para algum usuário que desejaria apenas controlar a quantidade de alguns itens e sua lista de compras – muitas das plataformas testadas necessitavam

de muitas informações extras dos produtos selecionados que fundamentalmente não seriam necessárias para o objetivo de controle de quantidades. Por esse motivo, o objetivo desse projeto era fornecer uma aplicação simples de se usar, com a necessidade de poucos detalhes para criar um produto e facilidade no controle dos valores. Além disso, como citado em sessões anteriores, a possibilidade do uso offline é outro diferencial a favor desse projeto.

4. Desenvolvimento

4.1. Requisitos do sistema

O primeiro passo para o desenvolvimento do projeto foi o levantamento dos requisitos do sistema. Com os requisitos funcionais e não funcionais em mente foi mais fácil visualizar todas as características que a aplicação final deveria ter. A Tabela 2 define todos os requisitos funcionais e não funcionais levantados acompanhados de uma breve descrição para cada um.

Código	Descrição
RF01	A ferramenta deve permitir o início de sessão pelos usuários (Login e Logout)
RF02	A ferramenta deve permitir o cadastro (Inclusão, alteração e exclusão) de usuários no sistema pelos próprios usuários
RF03	A ferramenta deve permitir o cadastro (Inclusão, alteração e exclusão) de estoques de mercadorias pelos usuários
RF04	A ferramenta deve permitir a criação de regras para avisos em relação a quantidade de itens no estoque pelos usuários
RF05	A ferramenta deve permitir o acesso aos estoques pelos usuários
RF06	A ferramenta deve permitir o cadastro (Inclusão, alteração e exclusão) de produtos dentro dos estoques pelos usuários
RF07	A ferramenta deve permitir o compartilhamentos de estoques entre usuários e alteração sincronizada entre os envolvidos
RF08	A ferramenta deve permitir a criação e exportação de listas de compra baseadas nas regras criadas pelos usuários
RNF01	A ferramenta deve ser compatível com os navegadores populares recentes (Chrome, Firefox, etc.)
RNF02	A ferramenta deve ser desenvolvida com a linguagem javascript usando o framework Vue.js
RNF03	A aplicação deve ser desenvolvida dentro do tempo de duração das disciplinas de TCC I e II
RNF04	O sistema deve funcionar corretamento tanto online quanto offline
RNF05	O sistema deve ser conectado com o sistema do Firebase, autenticação e banco de dados (Firestore)

Tabela 2. Requisitos do sistema

4.2. Casos de uso

Apesar de os requisitos do sistema fornecerem uma boa ideia de como a aplicação deveria ser, o processo de detalhamento não acabou aí. A partir dos requisitos, foram criados casos de uso que separam eles em funcionalidades mais definidas.

Código	Nome	Descrição
UC01	Realizar Login	Permite iniciar a sessão do sistema
UC02	Realizar Logout	Permite finalizar a sessão do sistema e desconectar o usuário
UC03	Cadastrar Usuário	Permite o registro de uma nova conta de usuário ao sistema para acesso à plataforma
UC04	Criar Estoque	Permite ao usuário criar um registro referente a lista de produtos e associar a sua conta
UC05	Editar Estoque	Permite alterar os dados referentes ao estoque
UC06	Remover Estoque	Permite remover um estoque associado a conta do usuário
UC07	Cadastrar Produto	Permite criar o registro de um produto e associar a um estoque
UC08	Editar Produto	Permite editar os dados de um produto
UC09	Remover Produto	Permite remover um produto associado a um estoque
UC10	Criar Regra	Permite o usuário criar um regra para um produto baseado na quantidade em que ele está presente em um estoque
UC11	Editar Regra	Permite alterar os valores de uma regra
UC12	Remover Regra	Permite remover uma regra
UC13	Solicitar Criação Lista de Compras	Permite o usuário solicitar ao sistema a criação de uma lista de compras baseada nas regras criadas pelo usuário
UC14	Compartilhar Estoque	Permite o usuário compartilhar um estoque com outro usuário

Tabela 3. Casos de uso

Com os casos de uso definidos na Tabela 3, foi definido o diagrama de casos de uso mostrado na Figura 1 mostrando os atores responsáveis por cada um dos casos. O diagrama ainda é separado em módulos para facilitar o entendimento e separar os casos de uso de acordo com a semelhança das funcionalidades e ação dos atores.

Figura 1. Diagrama de Casos de uso

4.3. Modelagem de classes

Para o manuseamento dos dados da aplicação, foi decidido seguir uma modelagem de classes que centralizariam os métodos para que fosse possível de usá-los em diferentes partes do sistema.

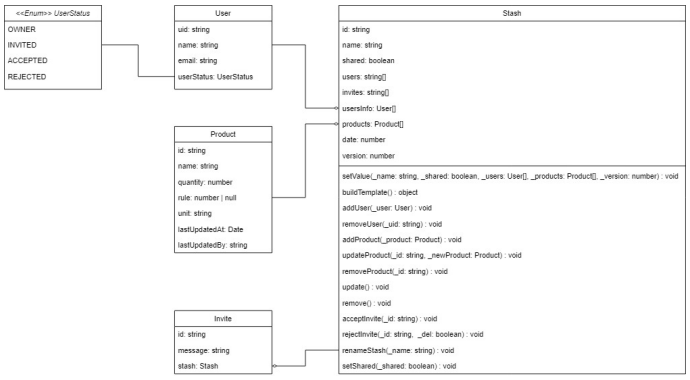


Figura 2. Diagrama de Classes

4.4. Dependências

Depois de todo detalhamento descrito nas seções anteriores, o desenvolimento do código iniciou com a decisão de quais módulos e pacotes seriam usados no decorrer de todo o projeto. Como já foi citado, o pacote principal e o framework que praticamente define a estrutura de todo o código: o Vue foi a escolha. Esse framework de Javascript, focado no desenvolvimento de aplicações web foi escolhido por já existirem experiências passadas, e por ser um framework simples e eficiente de se usar.

Para gerenciar todos os comandos, de desenvolvimento, compilação e controle de pacotes, foi usado o npm junto com o Vite. O primeiro é o gerenciador de pacotes do

Node.Js (Node Packages Manager) e o segundo, é uma CLI que permite ao Vue uma execução rápida e mais leve, principalmente por deixar de baixar muitos pacotes inúteis.

Na Tabela 4 estão definidas as principais dependências usadas no projeto e uma breve descrição do motivo de uso de cada uma.

Dependência	**Motivo**
vue-router	Permite o controle de navegação entre páginas
vuex	Permite a criação e controle de estados compartilhado com toda a aplicação
vue-i18n	Usada para a implementação da internacionalização do projeto
vuetify	Fornece componentes e estilos personalizáveis mais avançados que o html padrão
vue-notification / velocity-animate	Usado para gerar notificações flutuantes na tela
local-forage	Permite o armazenamento dos dados offline na cache
jspdf / jspdf-autotable	Criação de tabelas em pdf
firebase	Responsável pela autenticação e pelo banco de dados

Tabela 4. Principais dependências do projeto

4.5. Interfaces e estruturação da aplicação

A criação das interfaces em uma aplicação Vue segue semelhante a uma aplicação com HTML e CSS puros. A diferença se deve a tanto os códigos HTML, CSS e Javascript estarem no mesmo arquivo, permitindo a interação entre eles sem a necessidade de alteração direto na DOM e também aumentando o controle dos estados e das propriedades computadas (propriedades definidas por uma lógica definida pelo programador que atualiza automaticamente quando os valores usados são atualizados).

Outra diferença é a possibilidade de componentização que muitos frameworks, além do Vue, permitem. Isso permite a reutilização de código em diferentes partes da aplicação. Para esse projeto muitos dos componentes usados foram fornecidos pelo Vuetify, que além de prover vários componentes com funcionalidades diversas, também permite que eles sejam customizados com facilidade.

4.6. PWA e Manifest

Para a criação de aplicações progressivas, o Vite fornece um plugin que facilita a criação de todas as configurações necessárias, como o manifest, e suporte ao controle de mudanças na cache.

O Manifest consiste da definição de todas as informações que uma aplicação precisa para ser considerado um aplicativo ”baixável”. O nome do aplicativo quando é adicionado ao celular, ou computador, do usuário, uma descrição, cor do tema, ícone, todas essas informações são definidas nesse objeto.

```
VitePWA({
      includeAssets: ["favicon.ico", "robots.txt", "apple-touch-icon.
   png"],
      manifest: {
        name: "iStash",
        short_name: "iStash",
        description: "An app for controlling inventory",
        theme_color: "#303F9F",
        icons: [
          {
            src: "img/icons/android-chrome-192x192.png",
            sizes: "192x192",
            type: "image/png",
          },
          {
            src: "img/icons/android-chrome-512x512.png",
            sizes: "512x512",
            type: "image/png",
          },
          {
            src: "img/icons/android-chrome-maskable-512x512.png",
            sizes: "512x512",
            type: "image/png",
            purpose: "any maskable",
          },
        ],
      },
    })
```

Código 1. Manifest

Além do Manifest, o plugin de PWA do Vite fornece o controle automático de mudanças na aplicação que mostra um diálogo automaticamente para o usuário poder atualizar os dados do aplicativos salvos na cache para a última versão disponível.

4.7. Firebase e Firestore

Para o controle de autenticação e o armazenamento de dados da aplicação, foi decidido usar o Firebase, uma ferramenta fornecida pela Google que entre outras funcionalidades, oferece as duas necessárias para essa aplicação, além da hospedagem (que será discutida nas seções a seguir), e sem custos.

A configuração dos módulos começa criando um projeto no site oficial do Firebase, então é gerado uma chave para usar como configuração no código do programa. Então, após adicionar a biblioteca a aplicação é preciso adicionar os dados do projeto como configuração, como mostrado no Código 2.

```
1  import firebase from "firebase/app";
2  import "firebase/auth";
3  import "firebase/firestore";
4
5  // firebase init - add your own config here
6  const firebaseConfig = {
7    apiKey: "<API-KEY>",
8    authDomain: "istash.firebaseapp.com",
9    projectId: "istash",
10   storageBucket: "istash.appspot.com",
11   messagingSenderId: "838654125998",
12   appId: "1:838654125998:web:df6fe17c410c5632b2ca3f",
13   measurementId: "G-0GBS97EVTZ",
14 };
15
16 const app = firebase.initializeApp(firebaseConfig);
17
18 export default app;
```

Código 2. Configuração do Firebase

4.8. Store

Para controlar mais eficientemente os dados através de todos os componentes da aplicação, foi usado uma Store fornecida pelo pacote Vuex (dependência oficial ligada ao Vue). Com uma store, é possível armazenar os dados no *state* e esses dados podem ser alterados em *mutations*. Lógicas complexas, agrupamentos de mutations seguindo um algorítmo definido, são possíveis devido às *actions*, que além do descrito anteriormente, permite o uso de funções assíncronas. Além disso, é possível adicionar *plugins*, que são fragmentos de código executados durante o ciclo de vida da store.

Aproveitando da possibilidade de PWAs serem acessados offline, a configuração na aplicação foi feita usando a dependência Local Forage. O módulo permite o armazenamento de dados na cache do navegador. Junto com alguns plugins, foi possível configurar para que os dados da store fossem salvos na cache permitindo o funcionamento da aplicação sem acesso à internet.

Esses plugins foram 3: ”load”, ”cache”e ”sync”.

- ”load”: Esse plugin, executado apenas uma vez no início da execução do programa, carrega os dados salvos na cache e salva novamente na store.

```
import auth from "../plugins/firebase/auth";
import { getState } from "./storage";

export default function (store) {
  if (auth.currentUser && store.state.initialized) {
    return Promise.resolve();
  }

  return getState().then((state) => {
    store.commit("setSavedData", state);
  });
}
```

Código 3. Plugin: load

- ”cache”: O plugin, executado após toda mutação, atualiza os dados na cache.

```
import { setState } from "./storage";

const shouldSkipCache = (mutation) => {
  if (mutation.type === "setInvites") return true;
  return false;
};

const plugin = (store) => {
  store.subscribe((mutation, state) => {
    if (!shouldSkipCache(mutation)) {
      setState(state).catch((err) =>
        console.warn("failed to cache state", err)
      );
    }
  });
};

export default plugin;
```

Código 4. Plugin: cache

- ”sync”: Esse plugin é executado após toda mutation e, caso o usuário esteja online, atualiza os dados na base de dados.

```
import { removeValue, updateValue } from "../plugins/firebase/
    firestore";

const plugin = (store) => {
  store.subscribe((mutation, state) => {
    switch (mutation.type) {
      case "updateStash":
        updateValue(
          "stashes",
          mutation.payload["id"],
          mutation.payload["value"]
        )
          .catch()
          .finally(() => {
            state.updateData = true;
          });
        break;
      case "removeStash":
        removeValue("stashes", mutation.payload["id"])
          .catch()
          .finally(() => {
            state.updateData = true;
          });
        break;
      default:
      // Ignore default case
    }
  });
};

export default plugin;
```

Código 5. Plugin: sync

4.9. Router

O controle de rotas é feito pelo router fornecido pelo módulo vue-router (também oficial, ligado ao Vue). Com ele é possível definir os componentes mostrados em cada rota, além disso, foi possível criar uma espécie de middleware que checa se o usuário está conectado antes de mostrar algumas telas. Caso contrário o mesmo é redirecionado para a tela inicial.

```
router.beforeEach((to, from, next) => {
  const requiresAuth = to.matched.some((x) => x.meta.requiresAuth);

  if (requiresAuth && !auth.currentUser) {
    next("/index");
  } else {
    next();
  }
});
```

Código 6. Checagem da autenticação

4.10. Internacionalização

O conceito de Internacionalização foi incluído por consequência de um projeto para a disciplina de mesmo nome no início do desenvolvimento do programa, e mesmo após a finalização da matéria, foi decidido continuar o desenvolvimento seguindo o conceito. A aplicação tem suporte para dois locales, português brasileiro e inglês.

Isso foi possível graças à dependência vue-I18n. Ela fornece suporte a localização de textos, números, datas, entre outras funcionalidades. Nas configurações são definidas cada texto nos diferentes locales em uma key, então, quando o programador quiser mostrar o texto, basta chamar o método `t(key: string, values?: object)`, que define o texto mostrado baseado no locale selecionado. Ou então o método `tc(key: string, count: number)`, que semelhante ao anterior, retorna o texto correspondente, mas também recebe um número como parâmetro para definir a pluralidade do texto selecionado.

```
1  import Vue from "vue";
2  import VueI18n from "vue-i18n";
3  Vue.use(VueI18n);
4
5  const messages = {
6    "en-US": { ... },
7    "pt-BR": { ... },
8  };
9
10 const dateTimeFormats = {
11   "en-US": { ... },
12   "pt-BR": { ... },
13 };
14
15 const numberFormats = {
16   "en-US": { ... },
17   "pt-BR": { ... },
18 };
19
20 const i18n = new VueI18n({
21   locale: window.navigator.language,
22   fallbackLocale: "en-US",
23   messages,
24   dateTimeFormats: dateTimeFormats,
25   numberFormats,
26 });
27
28 export default i18n;
29
30 export function changeLocale(newLocale) {
31   i18n.locale = newLocale;
32 }
33
34 export function getLocale() {
35   return i18n.locale;
36 }
```

Código 7. Configuração do vue-I18n

4.11. CI/CD

Para manter a aplicação no ar e sempre na sua última versão foi usado o serviço de hospedagem gratuito fornecido pelo Firebase. Uma *action* foi criada no GitHub para que sempre que alterações fossem *commitadas* para o repositório, uma compilação automática é executada e já atualiza a aplicação no ar automaticamente, criando um fluxo CI/CD (ou integração contínua e entrega contínua).

Atualmente a aplicação está hospedada e pode ser acessada pelo link https://istash.web.app e o seu código fonte está no repositório público do github que pode ser acessado através do link https://github.com/bruno-borges-2001/iStash.

5. Conclusão

A proposta inicial do projeto de desenvolver um aplicativo que auxilie no controle de estoque foi finalizada com sucesso. O aplicativo foi disponibilizado em um link público e aberto para testes Quando este trabalho foi iniciado, já se imaginava dar acesso a qualquer um e ser uma aplicação de graça para o uso de qualquer um.

Apesar de o aplicativo estar funcional, é preciso ressaltar que o aplicativo não está totalmente completo e ainda possui funcionamento limitado. A estética do aplicativo não ficou exatamente como desejado e precisaria de uma revisão para ficar perfeita, principalmente em relação a experiência do usuário. Além disso, a internacionalização não está totalmente perfeita e muitos bugs ainda são possíveis de ocorrer.

Contudo, seria interessante testar o aplicativo com usuários reais e receber o feedback dos mesmos. Além disso, caso qualquer problema seja encontrado é necessário gerar novas versões para resolvê-los, uma vez que o aplicativo só foi testado em dispositivos android, com um único modelo de smartphone, e no browser, também em um único computador. Também é importante continuar o desenvolvimento e melhoria da aplicação, talvez até adicionando novas funcionalidades no futuro.

Referências

[Kitchenham 2004] Kitchenham, B. (2004). *Procedures for Performing Systematic Reviews*. Empirical Software Engineering National ICT Australia Ltd.

[Voskoglou et al. 2020] Voskoglou, C., Iacozza, S., Muir, R., Korakitis, K., and Stephens, J. (2020). State of the developer nation 19th edition: The latest trends from our q3 2020 survey of 17,000+ developers. *Programming Language Community: An Update*.