UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PATRICK ALEXIS PEROTTI

PORTFÓLIO E GESTÃO DE UM COMÉRCIO ELETRÔNICO DE PRODUTOS DE UMA CERVEJARIA ARTESANAL

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2020

PATRICK ALEXIS PEROTTI

PORTFÓLIO E GESTÃO DE UM COMÉRCIO ELETRÔNICO DE PRODUTOS DE UMA CERVEJARIA ARTESANAL

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO
2020

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
UTFPR - CAMPUS PATO BRANCO
DIRETORIA GERAL - CÂMPUS PATO BRANCO
DIR. DE GRAD. E EDUCAÇÃO PROFISSIONAL-PB
DEP. ACADÊMICO DE INFORMÁTICA DAINF-PB

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

Portfólio e Gestão de um Comércio Eletrônico de Produtos de uma Cervejaria Artesanal

POR

Patrick Alexis Perotti

Este trabalho de conclusão de curso foi apresentado no dia 25 de setembro de 2020, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Prof. MSc Vinicius Pegorini	**Profa. MSc Andreia Scariot Beulke**	**Prof. Dr. Ives Rene Venturini Pola**
Professor orientador	Professora convidada	Professor convidado

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profa. Dra. Mariza Miola Dosciatti
Responsável pela Atividade de Trabalho de Conclusão de Curso

Documento assinado eletronicamente por **ANDREIA SCARIOT BEULKE, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 06/10/2020, às 18:22, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **VINICIUS PEGORINI, PROFESSOR(A) ORIENTADOR(A)**, em 06/10/2020, às 19:41, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **IVES RENE VENTURINI POLA, PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/10/2020, às 19:44, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **MARIZA MIOLA DOSCIATTI, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 07/10/2020, às 19:51, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **EDILSON PONTAROLO, COORDENADOR(A) DE CURSO/PROGRAMA**, em 08/10/2020, às 09:11, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.utfpr.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1674624** e o código CRC **75ABC747**.

RESUMO

O comércio eletrônico tem substituído ou agregado à forma tradicional de comércio em exponencial crescente. Esse tipo de comércio teve início com a venda de bens tangíveis como livros. Ao adquirir um bem desse tipo, o usuário não necessita provar ou atestar a sua qualidade, desde que a compra seja de um item novo. Então, eles se mostraram mais propícios à forma eletrônica de comércio. Porém, com a evolução das tecnologias de informação e comunicação que permitem visualizações bem realistas de produtos, manequins de prova com as medidas do usuário e outros recursos, os mais diversificados produtos começaram a ser vendidos por meio da Internet. Além disso, formas alternativas de entrega e mais adequadas aos diversos tipos de produtos, formas alternativas de pagamento e mais seguras, também ajudam a ampliar e diversificar os itens comercializados de forma *online*. O sistema de comércio eletrônico desenvolvido para este trabalho é para uma cervejaria artesanal que fará o comércio do seu produto produzido em loja física, mas o sistema *web* será utilizado para vender produtos relacionados (souvenires). Os dados armazenados relacionados às vendas serão utilizados para fornecer informações que possam auxiliar o gestor na tomada de decisão. Para o desenvolvimento foram utilizadas diversas tecnologias para *web* como JavaScript, React e Node.js.

Palavras-chave: React. Node.js. E-commerce.

ABSTRACT

The e-commerce has replaced or agregated to the traditional form of commerce in exponential growth. This type of commerce has started with the sale of tangible goods such as books. When acquiring a good of this type, the user does not need to prove or certify its quality, as long as the purchase is for a new item. So, they showed to be more propitious to the electronic form of commerce. However, with the evolution of information and communication technologies that allow very realistic product visualizations, test mannequins with user measurements and other resources, the most diversified products started to be sold through the Internet. Besides, alternative forms of delivery and more adequate for different types of products, alternative and safer payment methods, also help to expand and diversify the items sold online. The e-commerce system developed for this work is to a craft brewery that will sell its product produced in a physical store, but the web system will be used to sell related products (souvenirs). The stored data related to sales will be used to provide information that can assist the manager in the decision making. For the development, several web technologies were used, such as JavaScript, React and Node.js.

Keywords: React. Node.js. E-commerce.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

AJAX	*Asynchronous JavaScript and X ML*
API	*Application Programming Interface*
CSS	*Cascading Style Sheet*
HTML	*Hypertext Markup Language*
HTTP	*HyperText Transfer Protocol*
NPM	*Node.js Package Manager*
RF	Requisitos Funcionais
RIA	*Rich Internet Applications*
RNF	Requisitos Não Funcionais
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais que definem o contexto no qual se insere o aplicativo desenvolvido como resultado da realização deste trabalho. Neste capítulo também são apresentados os objetivos e a justificativa do trabalho. E, por fim, está a organização do texto por meio da apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

Os aplicativos *web* caracterizados por processamento realizado no lado do servidor, com o cliente responsável apenas por solicitar e apresentar conteúdo gera interações limitadas com o usuário, é necessário recarregar a página inteira para executar navegação, os tempos de resposta são longos e há dificuldade para exibir animações e conteúdo multimídia (NUÑEZ *at al*., 2017). Esse tipo de aplicativo caracteriza os primeiros sistemas ou aplicativos *web*. Contudo, com a grande expansão e aumento crescente de uso e a diversidade de aplicações para esses sistemas, os usuários passaram a exigir maior rapidez nas transações e no carregamento da página e com elementos de interface (interação) que possuíssem recursos e funcionalidades apresentados pelas aplicações *desktop*, que eles estavam acostumados a utilizar.

Visando reduzir o tráfego de rede - pela possibilidade de realização de parte do processamento no cliente e até armazenamento de dados e pelo carregamento seletivo de partes da página (somente as que sofrem alterações) – e oferecendo elementos de interação com funcionalidades e recursos bem mais ricos que os existentes nos formulários desenvolvidos com *Hypertext Markup Language* (HTML), surgem as aplicações Internet com Interface rica. As *Rich Internet Applications* (RIA) vieram com o propósito de solucionar ou minimizar as limitações de interação e reduzir o volume de dados que trafega entre cliente e servidor *web* (FRATERNALI; ROSSI; SANCHEZ-FIGUEROA, 2010).

As RIAs permitiram melhorar significativamente a experiência do usuário, principalmente em termos de apresentação e interação. Essas aplicações possuem quatro características principais (FRATERNALI; ROSSI; SANCHEZ-FIGUEROA, 2010, BUSCH; KOCH, 2009): a distribuição de dados, a distribuição da lógica de negócios, a comunicação assíncrona entre cliente e servidor e a melhoria da interface do usuário.

Atualmente, independentemente de seguirem uma arquitetura RIA específica ou não, uma porcentagem crescente de aplicações *web* apresenta entre suas características a possibilidade para distribuir seus dados e sua lógica de negócios entre cliente e servidor,

permitindo, ainda, comunicação assíncrona entre eles. Muitas tecnologias foram propostas visando facilitar o desenvolvimento dessas aplicações com o objetivo de aumentar a produtividade do desenvolvedor (NUÑEZ *et al*., 2017).

Nesse contexto de aplicações *web*, destaca-se o comércio eletrônico que permite transações entre compradores e vendedores entre si e uns com os outros. Além das operações relacionadas à compra nesse tipo de aplicação que exigem recursos de interface melhorados, estão as operações de gerenciamento que fornecem apoio à tomada de decisão. Por meio da realização deste trabalho foi desenvolvido um portal gerencial dos dados de um sistema de comércio eletrônico para atender uma cervejaria artesanal.

1.2 OBJETIVOS

A seguir são apresentados os objetivos do sistema proposto neste trabalho, sendo que o objetivo geral está relacionado com o resultado principal e os objetivos específicos ilustram funcionalidades do sistema e complementam o objetivo geral.

1.2.1 Objetivo Geral

Implementar funcionalidades gerenciais de um comércio eletrônico de produtos relacionados a uma cervejaria artesanal.

1.2.2 Objetivos Específicos

- Implementar gráficos e relatórios para suporte à gestão e também apresentá-los por meio de *dashboard*.
- Implementar um sistema de gerenciamento de cupons de desconto.

1.3 JUSTIFICATIVA

O fornecimento de dados que suportem o processo de tomada de decisão e ofereçam uma visão real do momento do negócio é importante para que o gestor possa realizar ações que efetivamente contribuam para a sustentabilidade e a melhoria do negócio. Uma visão sintetizada dos dados do negócio pode ser apresentada na forma de *dashboard*, que pode ser complementada por gráficos e relatórios sintetizados. É importante que o tomador de decisão,

o gestor, tenha acesso rápido e facilitado a esses dados e que eles sejam apresentados visando facilitar o entendimento e a visualização do estado atual do negócio.

O acesso aos dados gerenciais deve ser fornecido aos usuários do sistema que possuam perfil de administrador. Os sistemas devem facilitar a atribuição desse tipo de permissão, mas fornecendo segurança dos dados por meio de controle de acesso.

A solução proposta por meio deste trabalho visa fornecer suporte gerencial para um sistema de comércio. As operações de negócio relacionadas ao comércio em si como a apresentação dos produtos e o carrinho de compra não fazem parte do escopo deste trabalho, mas elas geram dados que são utilizados pela solução aqui proposta.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico sobre aplicações internet com interface caracterizada como ricas. No Capítulo 3 são descritas as ferramentas e as tecnologias utilizadas na modelagem do sistema e que serão utilizadas na implementação subsequente do sistema. No Capítulo 4 é apresentado o resultado da realização do trabalho que é a modelagem do sistema. Por fim estão as referências utilizadas no texto.

2 APLICAÇÕES INTERNET RICAS

Aplicações web tradicionais (1.0) baseadas em hipertexto referem-se às aplicações cliente-servidor construídas com base nas tecnologias *Transmission Control Protocol* (TCP), *HyperText Transfer Protocol* (HTTP) e *Hypertext Markup Language* (HTML) (ASADI; DALIRI; ALIPOUR, 2018). Nessas aplicações a maioria do processamento da informação encontra-se no lado servidor enquanto o lado cliente tem a responsabilidade de apresentar conteúdo estático (BERNARDI; DI LUCCA; DISTANTE, 2014).

Com a evolução da *web* e seus recursos, aplicações *web* baseadas em HTML começaram a apresentar as suas limitações, especialmente quando elas integram atividades complexas para serem realizadas por meio de interface gráfica com o usuário (ROUBI; ERRAMDANI; MBARKI, 2016).

O surgimento de técnicas como *Asynchronous JavaScript and XML* (AJAX), a possibilidade de comunicação em tempo real e de renderização dinâmica lidam com o problema da atualização das páginas, oferecendo aplicações com interface semelhante às aplicações *desktop* (FRATERNALI; ROSSI; SANCHEZ-FIGUEROA, 2010, PRECIADO et al., 2007). A tendência de aproximar as aplicações *web* das aplicações *desktop* em termos de recursos continuou pela introdução de *frameworks* de interação rica com o usuário como jQuery, Angular e React, entre outros, que conduzem a uma nova classe de aplicações *web* conhecidas como aplicações Internet Ricas, as *Rich Internet Applications* (RIA) (PRECIADO *et al.,* 2007). Utilizando essas novas tecnologias, as RIAs combinam a arquitetura distribuída da *web* com a interatividade de interface e a capacidade computacional das aplicações *desktop* (ASADI; DALIRI; ALIPOUR, 2018).

Uma RIA é uma aplicação cliente-servidor caracterizada por uma combinação de vantagens do modelo de distribuição da *web* com a interatividade e a riqueza de interface das aplicações *desktop* (ROUBI; ERRAMDANI; MBARKI, 2016, MELIÁ et al., 2010). Essas aplicações provêem a maioria dos benefícios de manutenibilidade e implantação das aplicações *web* ao mesmo tempo em que elas oferecem uma interface de usuário com muitos recursos (MELIÁ et al., 2010). Esses autores também destacam que essas aplicações introduzem novas características arquiteturais às aplicações *web* tradicionais, como interface do usuário com estado (conectado e desconectado) e comunicação entre cliente e servidor inteligente com requisições assíncronas.

Busch e Koch (2009) definem RIAs como aplicações *web* que usam dados que podem ser processados tanto pelo cliente quanto pelo servidor. Além disso, a troca de dados é realizada

de maneira assíncrona de forma que o cliente permaneça responsivo enquanto continuamente recalcula ou atualiza partes da interface com o usuário. No cliente, as RIAs provêm uma interface similar a olhar e sentir (*look-and-feel*) como ocorre nas aplicações *desktop* e a palavra "rica" significa particularmente a diferença para a geração anterior das aplicações *web* com interface baseada somente em elementos simples de HTML em sua primeira versão como linguagem de marcação de hipertexto. Para essas autoras, as RIAs são basicamente caracterizadas por uma variedade de controle de operações interativas, que possibilitam uso da aplicação *online* e *offline*, o uso transparente da capacidade computacional do cliente e do servidor e da conexão de rede.

As RIAs são caracterizadas por apresentar interação mais ampla e melhor com o usuário final que não tem um papel passivo como um consumidor de conteúdo armazenado e processado no servidor da aplicação. Ao invés disso, o usuário tem assumido um papel ativo que contribui, pela escolha (seleção), para a definição ou processamento do conteúdo a ser fornecido no cliente. As RIAS, adicionalmente a apresentar usabilidade melhor, permitem que a aplicação tenha um desempenho melhor porque elas reduzem significativamente o número de requisições ao servidor pelo processamento que ocorre no cliente e a manipulação de dados requisitados pelo usuário, reduzindo, assim o recarregamento da página e o tráfego entre cliente e servidor. Assim, as RIAs são baseadas em um estilo arquitetural cliente-servidor que usa requisições assíncronas compondo pequenos blocos de dados (BERNARDI; DI LUCCA; DISTANTE, 2014).

Nesse contexto, as RIAs são similares às aplicações *desktop*, mas com hibridismo em relação às aplicações *web* porque elas combinam a arquitetura de distribuição leve das aplicações *web* com a interatividade de interface e a capacidade computacional das aplicações *desktop*. A combinação resultante melhora todos os elementos de uma aplicação *web* (dados, lógica de negócio, comunicação e apresentação) (FRATERNALI; ROSSI, 2010). Sumariamente, uma RIA pode ser vista como uma aplicação *web* na qual a interface com o usuário é processada no lado cliente e a lógica de negócio é definida por *backend* de serviços (VALVERDE; PASTOR, 2009).

3 MATERIAIS E MÉTODO

A seguir estão os materiais e o método utilizados para a modelagem e a implementação do sistema obtido resultado deste trabalho.

3.1 MATERIAIS

O Quadro 1 apresenta a lista de ferramentas e tecnologias utilizadas para o desenvolvimento deste trabalho.

Quadro 1 - Lista de ferramentas e tecnologias

Ferramenta / Tecnologia	Versão	Finalidade
HTML	5	Linguagem de marcação de hipertexto
CSS	4	Linguagem de folhas de estilo em cascata
Visual Studio Code	1.49.3	*Integrated Development Environment* (IDE) de desenvolvimento
JavaScript	1.8.5	Linguagem de programação
Yarn	1.22.4	Gerenciador de pacotes para JavaScript
Node.js	12.18.2	*Framework* para o lado servidor
PostgreSQL	11.3	Gerenciador de banco de dados
pgAdmin	4.6	Administrador de banco de dados
Express	4.17.1	*Framework* para implementação de APIs
Sequelize	5.5.1	Mapeamento de objeto-relacional
Multer	1.4.2	*Framework* para carregamento de arquivos
JWT	8.5.1	Método de autenticação
React	16.13.1	*Framework* de interface de usuário para o lado cliente
Redux	4.0.5	*Framework* de gerenciamento de estado
Redux-Saga	1.1.3	*Framework* de efeitos colaterais (assíncronos) no estado
Material-UI	4.11.0	Componentes estilizados para o lado cliente
material-table	1.67.1	Componente de tabela estilizada para o lado cliente
Formik	2.1.4	*Framework* para validação de formulários
formik-material-ui	2.0.0	*Framework* de ligação entre Formik e Material-UI

Yup	0.29.1	*Framework* para validação de dados
date-fns	2.14.0	*Framework* para manipulação de datas
Recharts	1.8.5	*Framework* de geração de gráficos para o lado cliente
pdfmake	0.1.65	*Framework* para geração de arquivos do tipo PDF
Visual Paradigm	16.1	Ferramenta de modelagem *Unified Modeling Language* (UML)

Fonte: Autoria própria.

3.2 MÉTODO

A plataforma de gestão de *e-commerce* é uma aplicação web que visa atender a necessidade do gerenciamento de cadastros e movimentações do *e-commerce*, bem como o fornecimento de dados aos gestores por meio de *dashboard*, gráficos e relatórios, de maneira que suportem o processo de tomada de decisão. Os dados para essa gestão estão baseados nas operações de negócio realizadas por um produtor de cerveja artesanal. Portanto, os requisitos, são baseados em um negócio real visando, contudo, fornecer dados que suportem a tomada de decisão e ofereçam uma visão mais ampla e fidedigna do negócio em termos de momento atual e de histórico.

A análise e o projeto consistiram na modelagem dos requisitos, representada pela listagem dos requisitos (funcionais e não funcionais), a sua organização em casos de uso com os atores relacionados e o modelo entidades e relacionamentos do banco de dados.

O banco de dados consiste das tabelas necessárias para armazenar os cadastros, controlar permissões e registrar o histórico de movimentações de produtos. Isso porque embora o comércio em si (cadastros como de produtos, carrinho de compras, pagamento) não faça parte da solução proposta por este trabalho, a modelagem do banco de dados considera a persistência gerada por essas funcionalidades. Inclusive, porque, são utilizados dados gerados pelas operações de comércio para os relatórios de gestão.

A implementação da plataforma de gestão de *e-commerce* iniciou-se com a configuração e implementação da estrutura base do servidor, dos modelos do banco de dados, das rotas e regras de negócio para consulta e persistência dos dados.

Em sequência, a configuração e implementação da estrutura base da aplicação cliente, dos componentes principais para uso como base nas páginas, das páginas de listagem e cadastro.

Por fim, a implementação do *dashboard*, gráficos e relatórios, com as rotas e consultas no servidor e das páginas na aplicação cliente.

4 RESULTADOS

Este capítulo apresenta os resultados obtidos durante o desenvolvimento deste trabalho. Na Seção 4.1 é apresentada uma visão geral das funcionalidades e do escopo do sistema. Essas funcionalidades são apresentadas e modeladas como requisitos na Seção 4.2. Nas seções 4.3 e 4.4 são apresentadas a interface e a implementação do sistema, respectivamente.

4.1 ESCOPO DO SISTEMA

O portal gerencial disponibilizará acesso para o usuário administrador, que poderá cadastrar usuários para acessá-lo. Também contará com cadastros para manutenção das informações que serão dispostas para acesso, sendo produtos para vendas no *e-commerce*, cupons de desconto, entre outros. Outras funcionalidades do portal gerencial envolvem a geração de relatórios para suporte ao planejamento e gestão do negócio e facilitar a visualização de informações adicionais sobre o negócio por meio das informações apresentadas em *dashboard*.

4.2 MODELAGEM DO SISTEMA

No Quadro 2 estão listados os Requisitos Funcionais (RF) identificados para o sistema.

Quadro 2 – Requisitos funcionais

Id.	Requisito	Descrição
RF01	Manter usuários	Cadastro dos usuários no sistema, contendo os dados necessários para identificação, *login* e gerenciamento de acesso. A autenticação no sistema, utilizando dados desse cadastro permitirá o acesso ao portal gerencial.
RF02	Manter produtos	Cadastro dos produtos contendo os dados necessários para controle de estoque e venda no *e-commerce*.
RF03	Manter tamanhos	Cadastro de tamanhos contendo os dados referentes a tamanhos para os produtos, bem como sua capacidade em caso de tamanhos de embalagens.
RF04	Manter preços de produtos	Cadastro de programações de preços contendo dados referentes aos períodos de vigência e valores dos produtos.
RF05	Manter movimentações de estoque	Histórico de movimentações de estoque, com dados referentes às operações de entrada e saída de produtos.

RF06	Manter cupons de desconto	Cadastro de cupons de desconto com dados referentes aos períodos de vigência e valores de desconto dos produtos.
RF07	Emitir gráficos e relatórios	Gráficos e relatórios referentes às movimentações de estoque e vendas do *e-commerce*.

Fonte: Autoria própria.

Os Requisitos não Funcionais (RNF) são apresentados no Quadro 3.

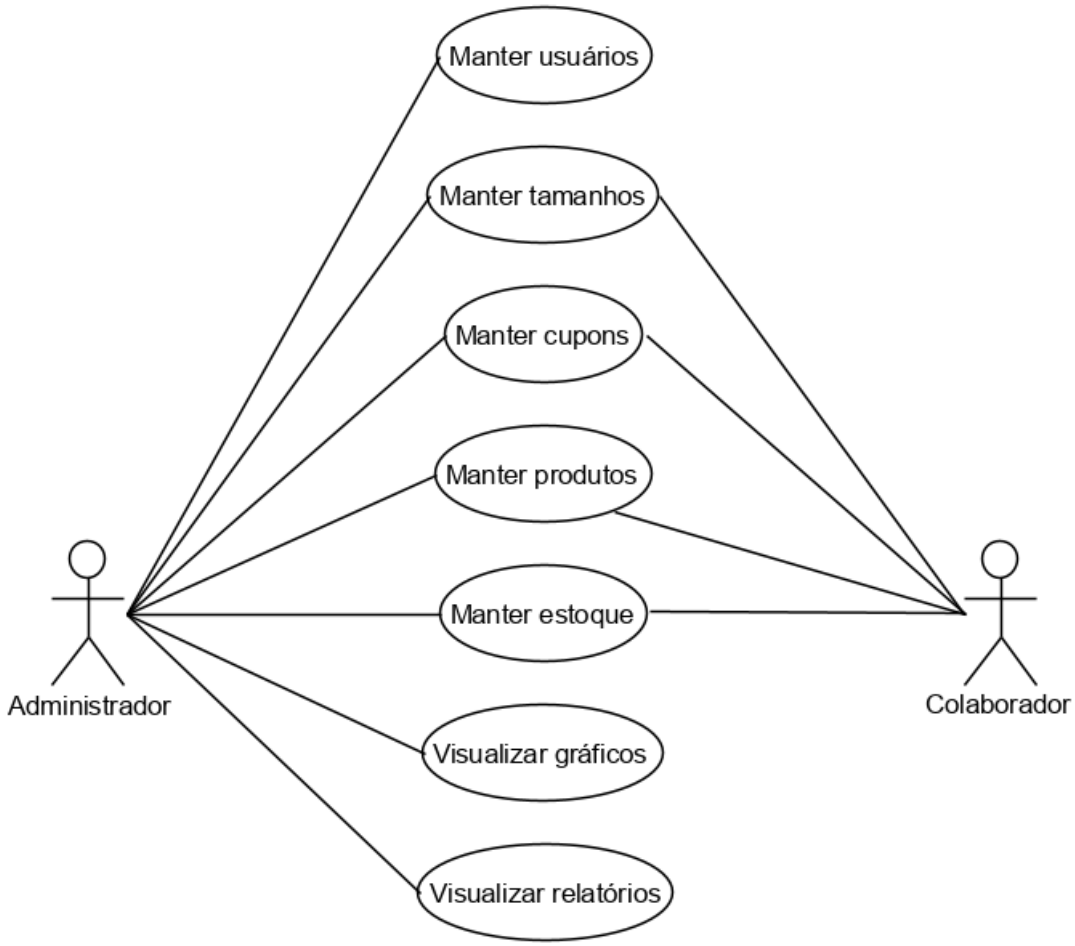
Quadro 3 – Requisitos não funcionais

Id.	Requisito	Descrição
RNF01	Acesso ao portal gerencial	Para a visualização do portal gerencial será necessário o cadastro do usuário e posteriormente ele acessar o sistema utilizando *login* e senha previamente cadastrados.
RNF02	Permissões de usuário	Haverá permissões de usuários para acesso a funcionalidades e rotinas do sistema. Esse controle será realizado pelo tipo de usuário cadastrado.
RNF03	Validação de campos obrigatórios no cadastro do usuário	Os campos nome e senha deverão ser validados com restrições de quantidade mínima e máxima de caracteres. O campo confirmar senha deverá ser preenchido com o mesmo valor do campo senha.

Fonte: Autoria própria.

A Figura 1 apresenta o diagrama de casos de uso que foi composto a partir dos requisitos funcionais identificados para o sistema. As operações que geram dados que serão persistidos e utilizados para a gestão são de usuários cadastrados, que se dividem em administradores e colaboradores (usuários comuns).

Figura 1 – Casos de uso



Fonte: Autoria própria.

Os quadros a seguir apresentam a expansão dos casos de uso da Figura 1. Os casos de uso manter, que se referem aos cadastros são expandidos por operação. Exemplo, a operação inclusão aplica-se a todos os casos de uso de cadastro, definido como “Manter” na Figura 1.

O Quadro 4 apresenta a operação de inclusão dos casos de uso “Manter”.

Quadro 4 - Operação inclusão dos casos de uso “Manter”

Caso de uso: Realizar cadastro. Refere-se à operação de inclusão dos seguintes casos de uso “Manter”. **Descrição:** Ator quer incluir dados cadastrais no sistema. **Atores:** Administradores e colaboradores. **Pré-condição:** Dados cadastrais para serem inseridos no sistema. **Sequência de Eventos:** 1. Ator acessa a página do sistema referente ao cadastro para o qual quer incluir um registro (dados).

<table>
<tr><td colspan="2">2. Ator preenche os campos e confirma a entrada dos dados.
3. O sistema valida as informações, insere no banco de dados e apresenta mensagem de confirmação da operação ao usuário.
Pós-Condição:
Registro inserido no banco de dados.</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td>1. Campos obrigatórios não preenchidos.</td><td>2.1. O ator não preenche os campos obrigatórios e clica no botão “Confirmar”.
2.2. O sistema valida as informações e destaca visualmente os campos obrigatórios que não foram preenchidos e mensagem de notificação é apresentada.</td></tr>
<tr><td>2. Campos preenchidos com formato inválido</td><td>3.1. O ator preenche os campos de forma incorreta e clica no botão “Confirmar”.
3.2. O sistema valida as informações e destaca visualmente os campos que foram preenchidos de forma incorreta e apresenta mensagem de notificação ao usuário.</td></tr>
</table>

Fonte: Autoria própria.

No Quadro 5 está a expansão da operação de alteração dos casos de uso de cadastro.

Quadro 5 - Operação alteração dos casos de uso “Manter”

<table>
<tr><td colspan="2">Caso de uso:
Alterar cadastro. Refere-se à operação de alteração dos seguintes casos de uso “Manter”.
Descrição:
Ator solicitar a alteração de dados de cadastros já inseridos no sistema.
Atores:
Administradores e colaboradores.
Pré-condição:
Ator autenticado no sistema.
Sequência de Eventos:
1. Ator acessa o cadastro para visualizar os dados do registro que pretende alterar.
2. Ator altera os dados do registro e solicita que os dados sejam salvos.
3. Sistema valida as informações, altera o registro no banco de dados e exibe mensagem informando ao usuário o status do procedimento.
Pós-Condição:
Registro alterado no banco de dados.</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td>1. Campos obrigatórios não preenchidos.</td><td>1.1. O ator não preenche campos obrigatórios e confirma a operação de inclusão de dados.
1.2. O sistema valida as informações e destaca visualmente os campos obrigatórios que não foram preenchidos e apresenta mensagem de notificação ao usuário.</td></tr>
<tr><td>2. Campos preenchidos com formato inválido</td><td>2.1. O ator preenche campos de forma incorreta, de acordo com as regras e validação definidas e confirma a operação.
2.2. O sistema valida as informações e destaca visualmente os campos que foram preenchidos de forma incorreta e apresenta mensagem de notificação ao usuário.</td></tr>
</table>

Fonte: Autoria própria.

A operação de cancelamento do caso de uso “Manter estoque” é apresentada no Quadro 6.

Quadro 6 - Operação cancelamento do caso de uso “Manter estoque”

Caso de uso: Cancelar cadastro. Refere-se à operação de cancelamento do caso de uso “Manter estoque”. **Descrição:** Ator solicitar o cancelamento de dados no sistema. **Atores:** Administradores. **Pré-condição:** Ator autenticado no sistema. **Sequência de Eventos:** 1. Ator acessa a página para visualizar os registros já cadastrados. 2. Ator seleciona o registro que deseja cancelar. 3. Ator selecionar a opção para cancelar o registro. 4. O sistema cancela o registro do banco de dados e exibe mensagem notificando o usuário sobre o status do procedimento. **Pós-Condição:** Registro cancelado no banco de dados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Cancelamento de registro que resulta em estoque negativo.	1.1. Ator solicita o cancelamento de registro. 1.2 O sistema verifica se o cancelamento resulta em estoque negativo e se resultar exibe mensagem informando o usuário a impossibilidade de cancelamento do registro.

Fonte: Autoria própria.

A expansão do caso de uso visualizar relatório é apresentada no Quadro 7.

Quadro 7 - Casos de uso “Visualizar gráficos” e “Visualizar relatórios”

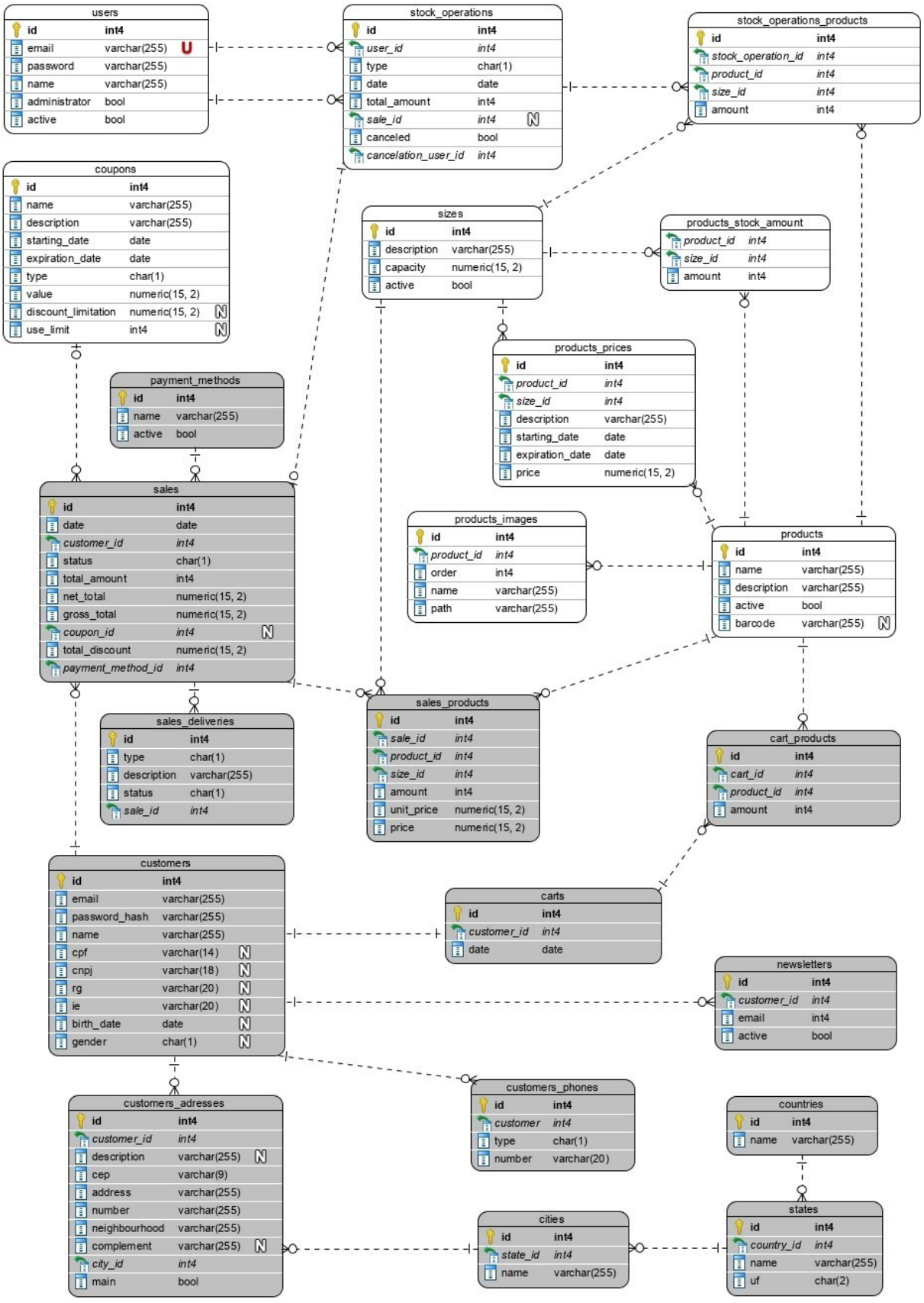
Caso de uso: Visualizar gráficos e relatórios. Se referem aos os dados filtrados e agrupados em forma de gráficos e relatórios, bem como os dados apresentados em *dashboard*. **Descrição:** Ator visualiza dados armazenados no sistema, filtrados e agrupados. **Atores:** Administradores. **Pré-condição:** Ator autenticado no sistema. **Sequência de Eventos:** 1. Ator acessa a página para visualizar gráficos, relatórios ou *dashboard*. 2. Ator seleciona o que quer visualizar. 3. O apresenta os dados de acordo com a opção solicitada. **Pós-Condição:** Dados apresentados.	
Nome do fluxo alternativo (extensão)	**Descrição**
1. Campos obrigatórios não preenchidos.	1.1. O ator não preenche campos obrigatórios e confirma a operação. 1.2. O sistema valida as informações e destaca visualmente os campos obrigatórios que não foram preenchidos e apresenta mensagem de notificação ao usuário.

2. Campos preenchidos com formato inválido	2.1. O ator preenche campos de forma incorreta, de acordo com as regras e validação definidas e confirma a operação. 2.2. O sistema valida as informações e destaca visualmente os campos que foram preenchidos de forma incorreta e apresenta mensagem de notificação ao usuário.

Fonte: Autoria própria.

A Figura 2 apresenta o diagrama de entidades e relacionamentos do banco dados do sistema.

Figura 2 – Diagrama de entidades e relacionamentos do banco de dados



Fonte: Autoria própria.

Na entidade denominada *users* serão registrados os usuários da aplicação. A entidade *products* manterá os produtos e sua descrição, com detalhes nas entidades *products_prices* para armazenar as programações de preços e *products_images* para armazenar as imagens e respectivas ordens de exibição. Na entidade denominada *sizes* serão registrados os tamanhos e a entidade *coupons* manterá os cupons promocionais.

O usuário pode realizar movimentações de estoque de produtos por meio das entidades *stock_operations* e *stock_operations_products*. A entidade *stock_operations* é o mestre da movimentação, que registrará a data de movimentação e o usuário que está realizando a operação, já a entidade *stock_operations_products* é o detalhe da movimentação, registrando os produtos movimentados e suas respectivas quantidades.

As entidades em cinza representam a parte do *e-commerce*, que são utilizadas pelo portal gerencial para obtenção de dados a fim de gerar gráficos e relatórios.

4.3 APRESENTAÇÃO DO SISTEMA

A seguir são apresentadas as funcionalidades desenvolvidas no portal gerencial, para uso dos usuários. Inicialmente é criado um usuário administrador padrão para o uso do portal.

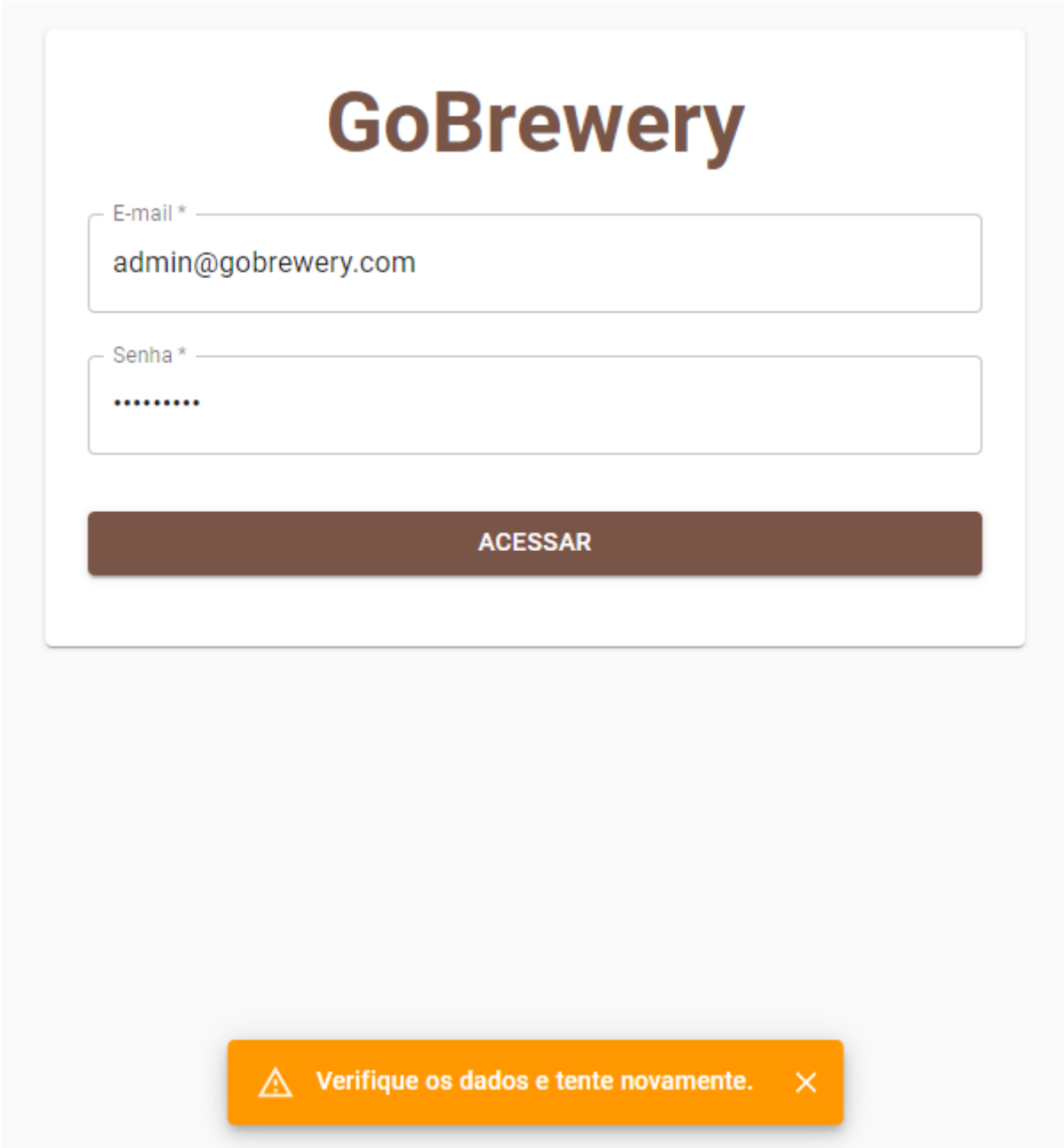
A Figura 3 apresenta a tela de autenticação do sistema, que é realizada utilizando o e-mail e a senha de usuário.

Figura 3 – Tela de autenticação do portal gerencial

Fonte: Autoria própria.

A Figura 4 apresenta o caso de falha na autenticação informada ao usuário visualmente por meio de *snackbar*.

Figura 4 – Falha na autenticação



Fonte: Autoria própria.

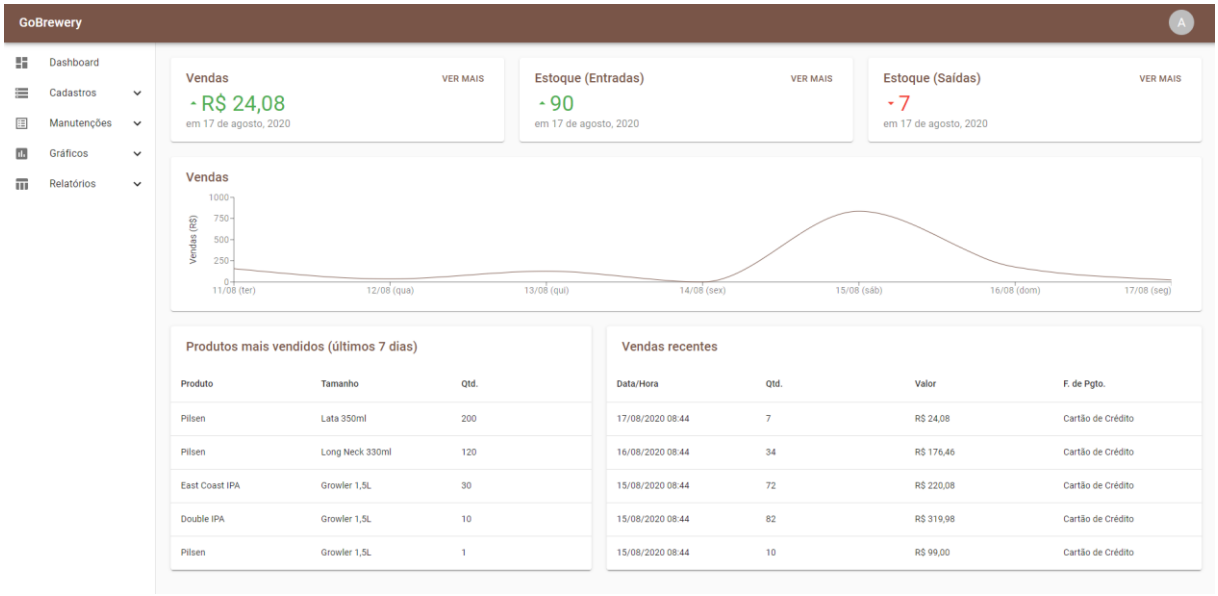
A Figura 5 apresenta o indicador visual de progresso circular durante a autenticação.

Figura 5 – Indicador visual de autenticação em progresso

Fonte: Autoria própria.

A Figura 6 apresenta a tela inicial para usuários administradores após a autenticação ser realizada, sendo o *dashboard* do portal. Essa tela apresenta os dados do total diário de vendas e de entradas e saídas de estoque. Também apresenta um gráfico linear de vendas no período dos últimos sete dias e tabelas com dez registros dos produtos mais vendidos e das vendas recentes. Dados fictícios foram registrados para demonstrar o funcionamento do *dashboard*.

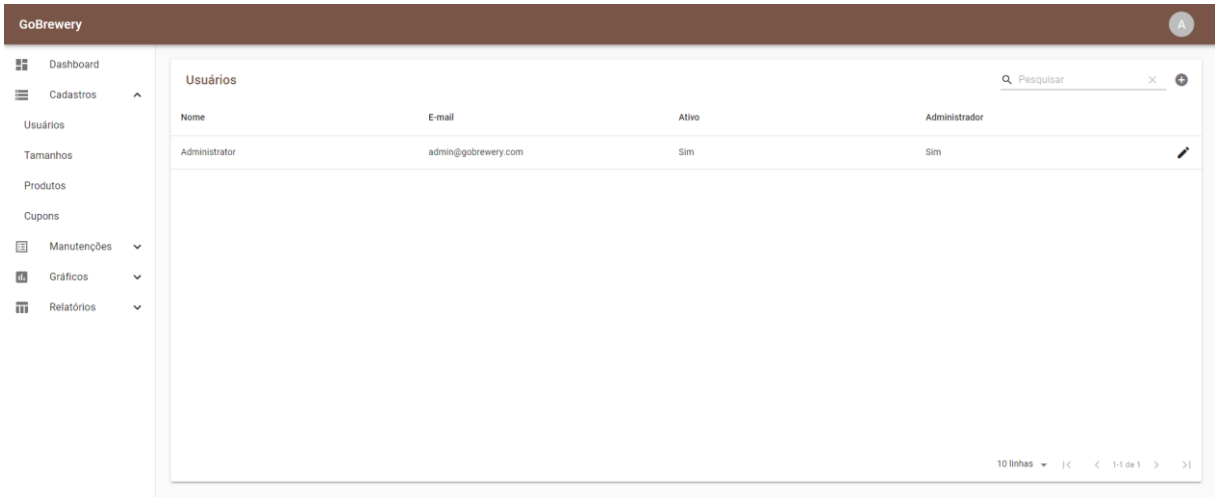
Figura 6 – *Dashboard* do portal gerencial



Fonte: Autoria própria.

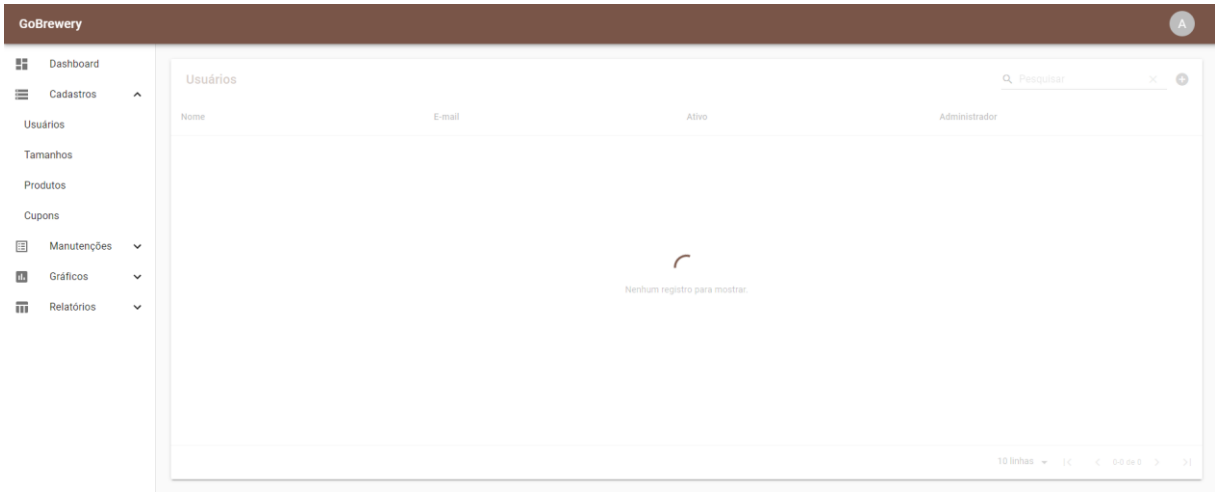
A Figura 7 apresenta a tela de cadastro de usuários. Todas as telas de listagem de cadastros seguem o mesmo padrão, com título na parte superior direita, barra de pesquisa para filtrar registros e botão de adicionar na parte superior esquerda e botão de edição nos registros. Também apresentam um indicador visual circular de progresso durante o carregamento dos registros, visualizado na Figura 8. Em caso de inexistência de registros para listagem, é indicado visualmente em texto no corpo da tabela, como observado na Figura 9. Em caso de falha no carregamento, é indicado visualmente por meio de *snackbar*, visualizado na Figura 10.

Figura 7 – Listagem de usuários cadastrados



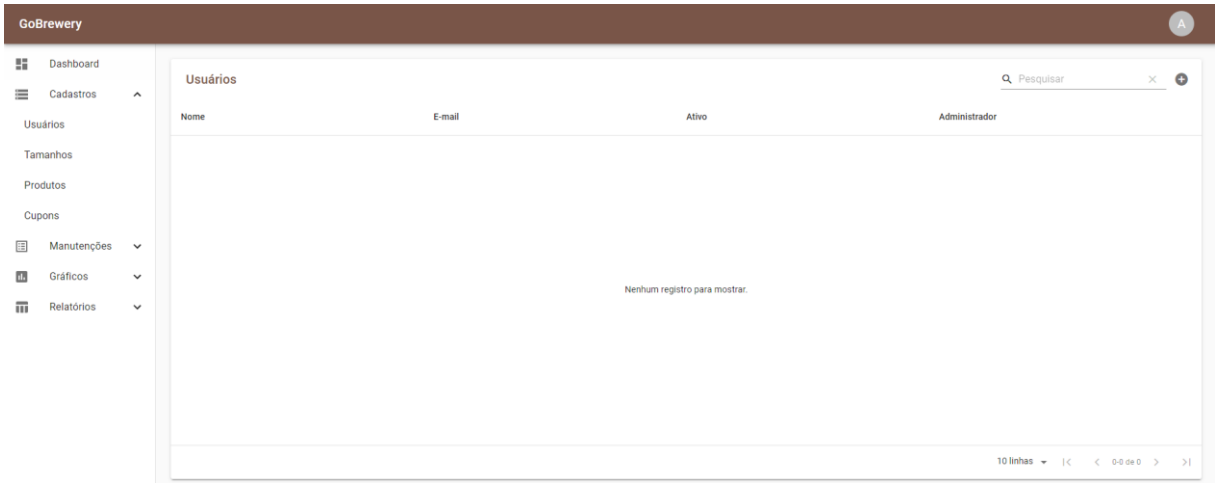
Fonte: Autoria própria.

Figura 8 – Indicador visual de carregamento de cadastros em progresso



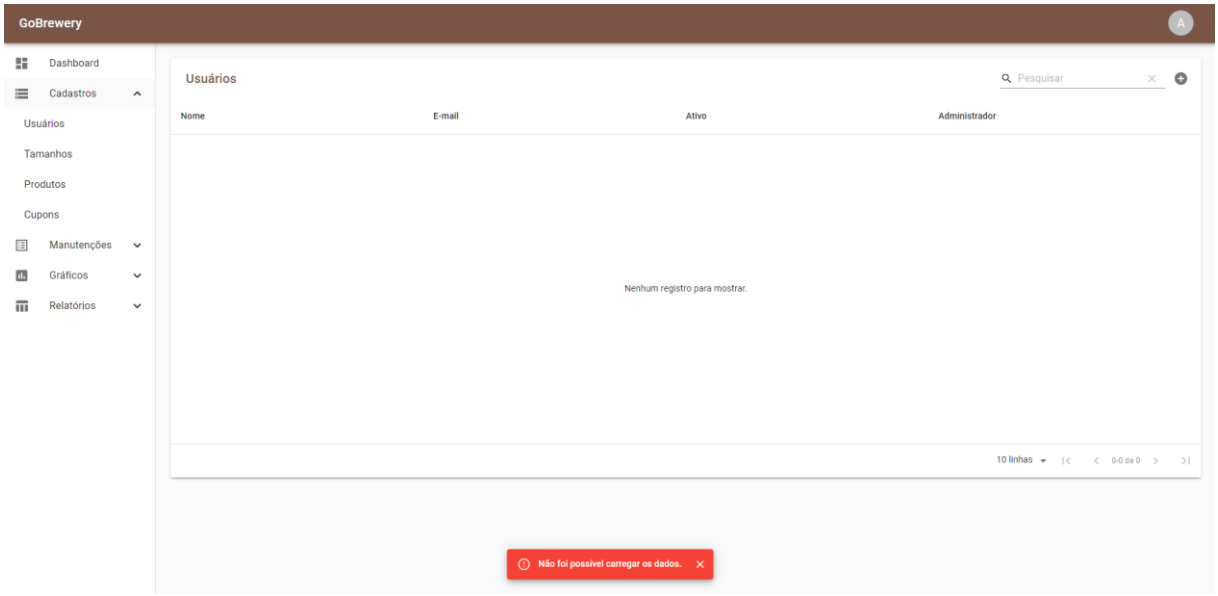
Fonte: Autoria própria.

Figura 9 – Indicador visual de inexistência de cadastros para listagem



Fonte: Autoria própria.

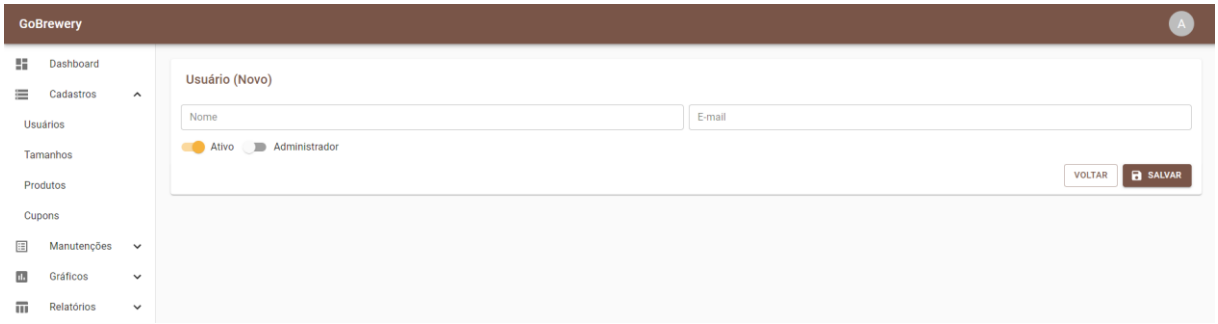
Figura 10 – Indicador visual de falha ao carregar cadastros



Fonte: Autoria própria.

A Figura 11 apresenta o formulário para cadastrar um novo usuário. Todos os formulários de novos cadastros seguem o mesmo padrão, com título na parte superior esquerda indicando novo cadastro, formulário para inserção dos dados do cadastro, um botão para voltar para a tela de listagem e um botão para salvar o cadastro. Novos usuários são automaticamente cadastrados com uma senha padrão, que pode ser alterada após o primeiro acesso ao portal, por meio da edição do perfil.

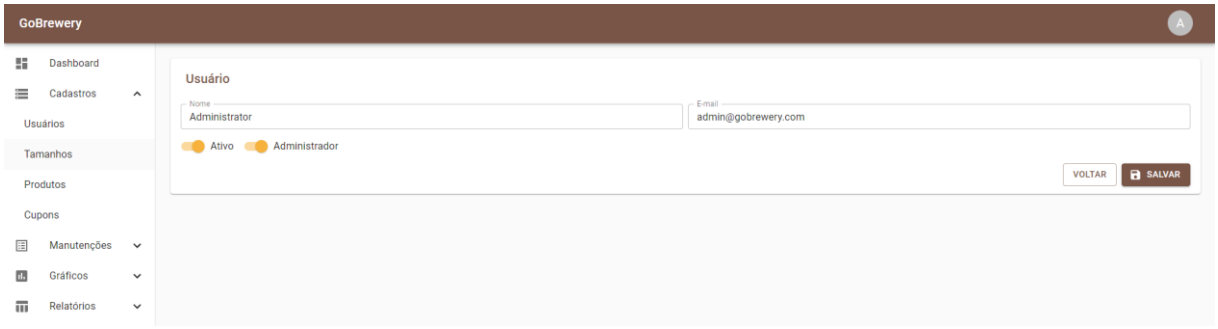
Figura 11 – Novo cadastro de usuário



Fonte: Autoria própria.

A Figura 12 apresenta o formulário de edição de usuário. Todos os formulários de edição de cadastros seguem o mesmo padrão, com título na parte superior esquerda, formulário para edição dos dados, um botão para voltar para a tela de listagem e um botão para salvar o cadastro.

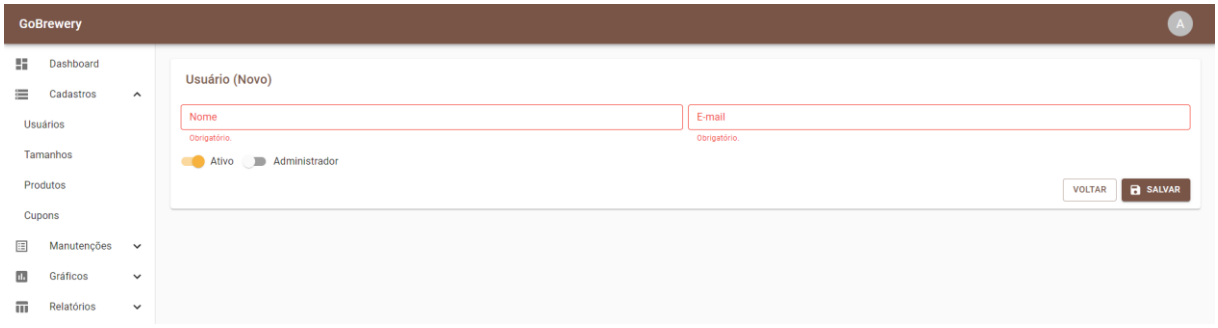
Figura 12 – Edição de usuário



Fonte: Autoria própria.

A Figura 13 apresenta a validação do cadastro de usuário. Todos os formulários seguem o mesmo padrão de validação, com indicadores visuais nos campos do formulário, tanto ao mudar o foco entre componentes do formulário, quanto ao clicar no botão de salvar. A validação de usuário define como obrigatório o preenchimento dos campos de nome e e-mail com formato válido.

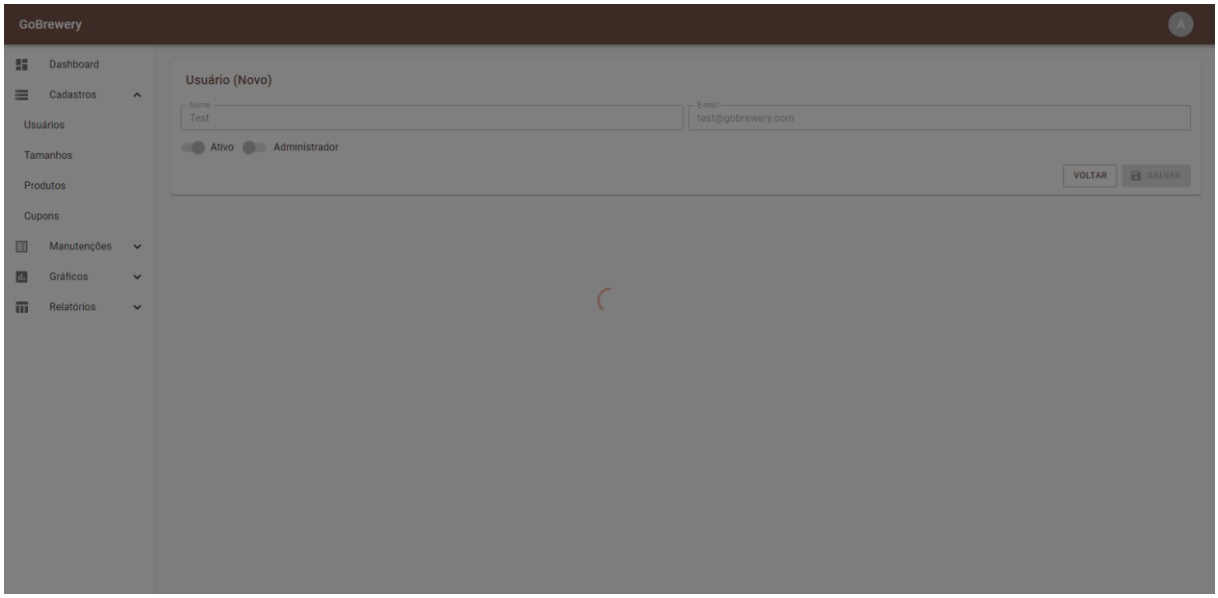
Figura 13 – Validação de formulário de usuário



Fonte: Autoria própria.

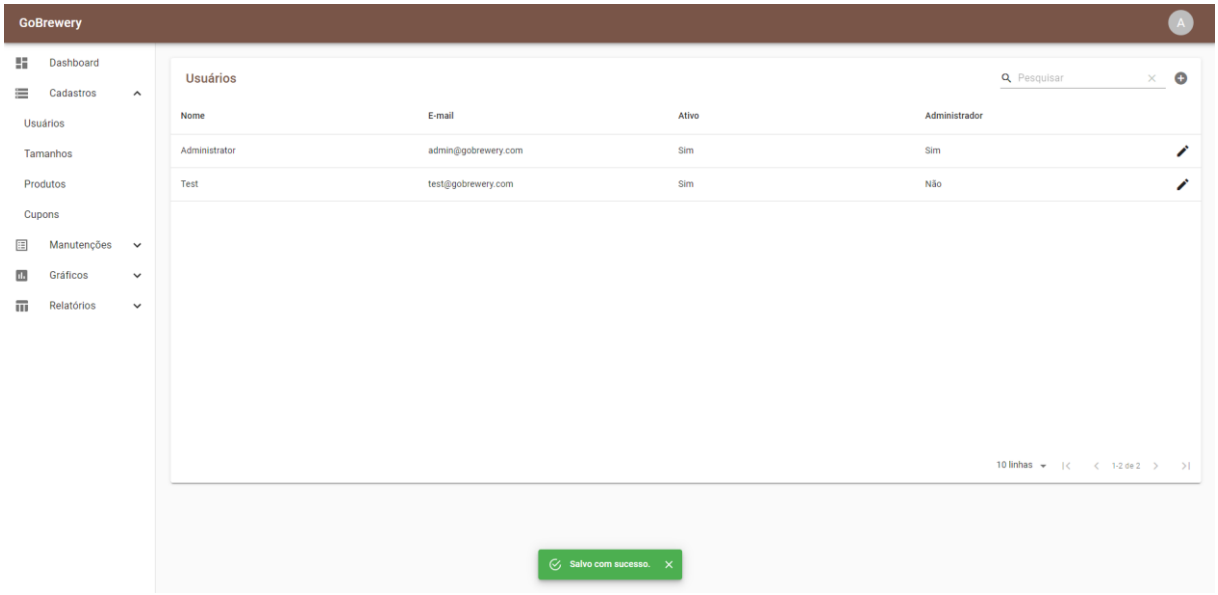
Ao salvar um registro é apresentado ao usuário um *backdrop* com indicador visual circular de progresso, como apresenta a Figura 14, os demais formulários de cadastro do sistema seguem o mesmo padrão. Também é indicado visualmente ao usuário o resultado após salvar um registro por meio de um *snackbar* de sucesso, como apresenta a Figura 15, ou a falha, como pode ser visualizado na Figura 16. Após o registro ser salvo com sucesso, o usuário é automaticamente redirecionado para a página de listagem do cadastro em questão, como pode ser observado na Figura 15.

Figura 14 – Indicador visual de salvamento de cadastro em progresso



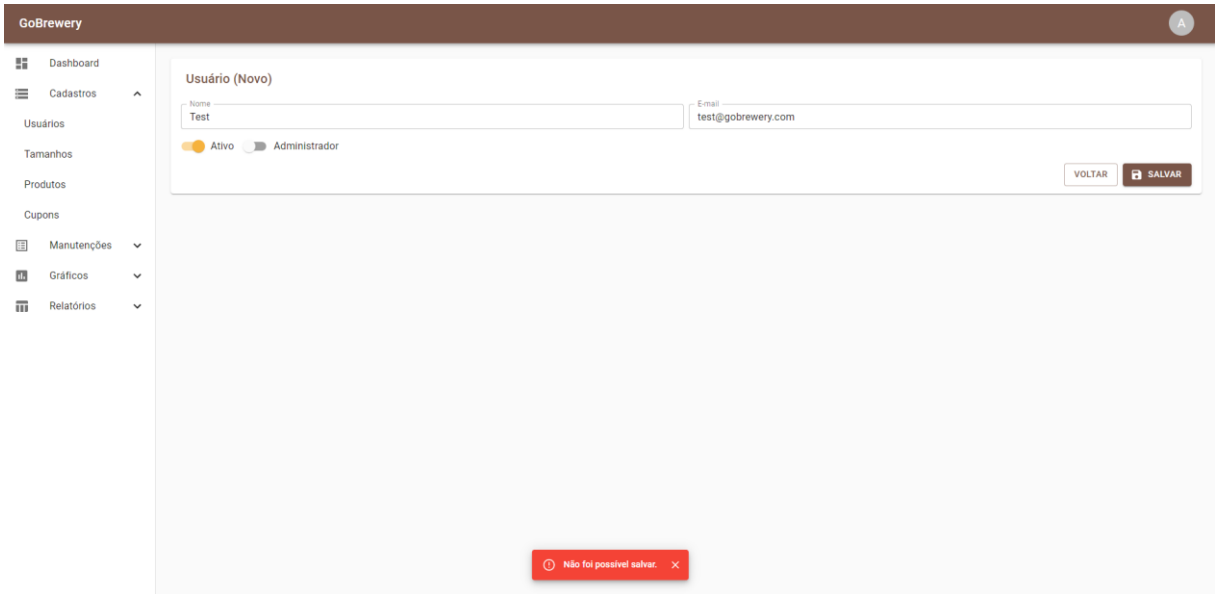
Fonte: Autoria própria.

Figura 15 – Indicador visual de sucesso ao salvar cadastro



Fonte: Autoria própria.

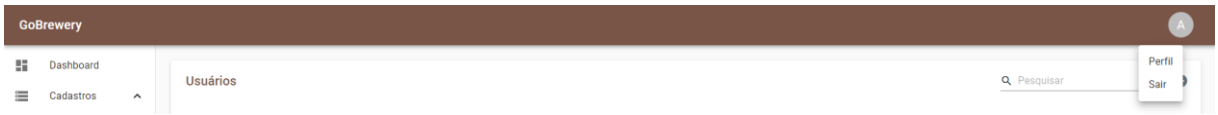
Figura 16 – Indicador visual de falha ao salvar cadastro



Fonte: Autoria própria.

A Figura 17 apresenta o menu do usuário autenticado, localizado na barra do portal gerencial. É possível acessar o seu perfil e sair do portal, sendo necessário realizar nova autenticação para acesso.

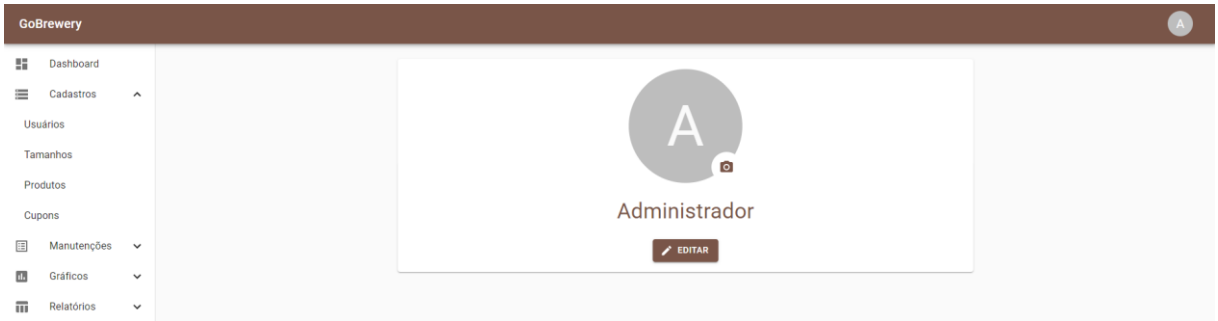
Figura 17 – Menu do usuário autenticado



Fonte: Autoria própria.

A Figura 18 apresenta o perfil do usuário autenticado, sendo possível inserir, alterar ou remover a foto de perfil por meio do botão no canto inferior direito da foto.

Figura 18 – Perfil do usuário autenticado



Fonte: Autoria própria.

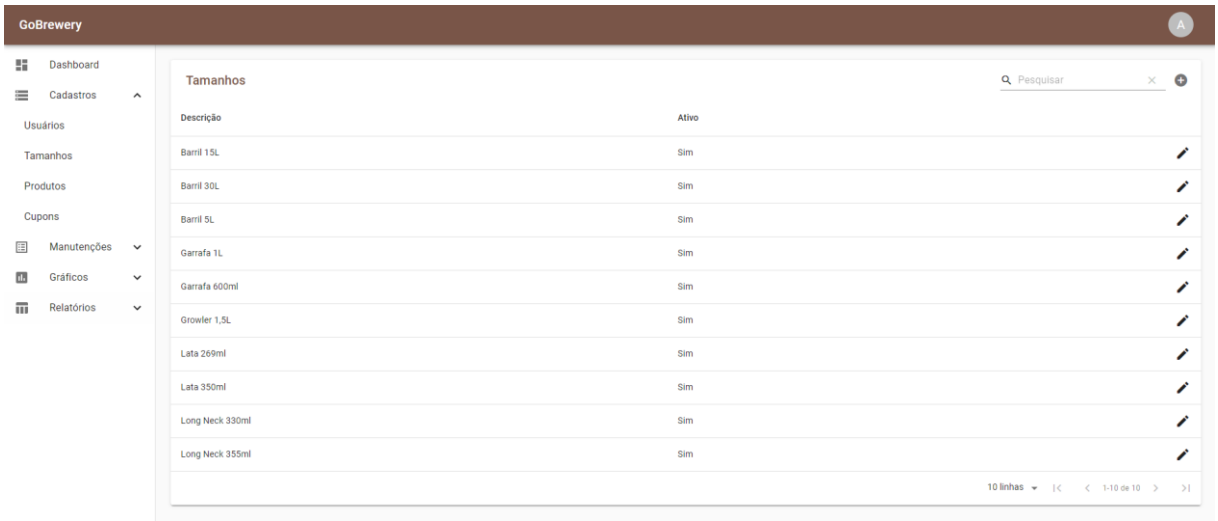
A Figura 19 apresenta a edição do usuário autenticado, sendo possível alterar seu nome, e-mail e senha.

Figura 19 – Edição do usuário autenticado

Fonte: Autoria própria.

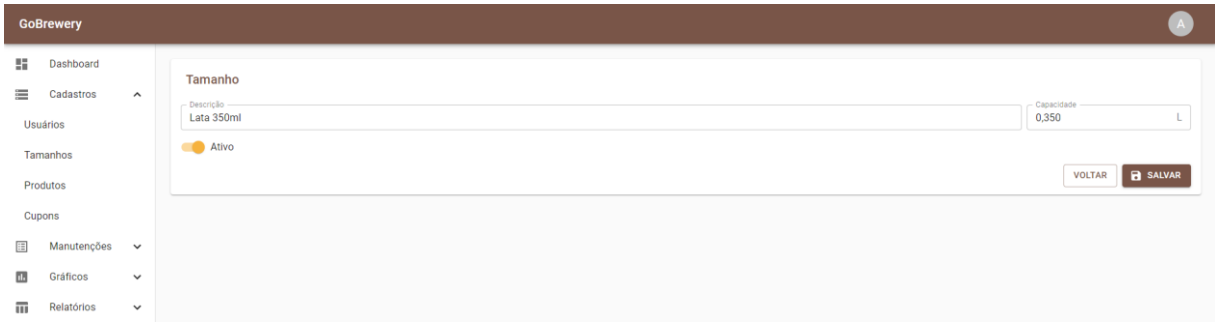
A Figura 20 apresenta a listagem de tamanhos cadastrados e a Figura 21 o formulário de cadastro de tamanho. É possível cadastrar tamanhos simples, ou seja, sem informação de capacidade, ou tamanhos referentes a embalagens, com a capacidade em litros. A capacidade pode ser utilizada em cálculos para disponibilizar informações em gráficos e relatórios.

Figura 20 – Listagem de tamanhos cadastrados



Fonte: Autoria própria.

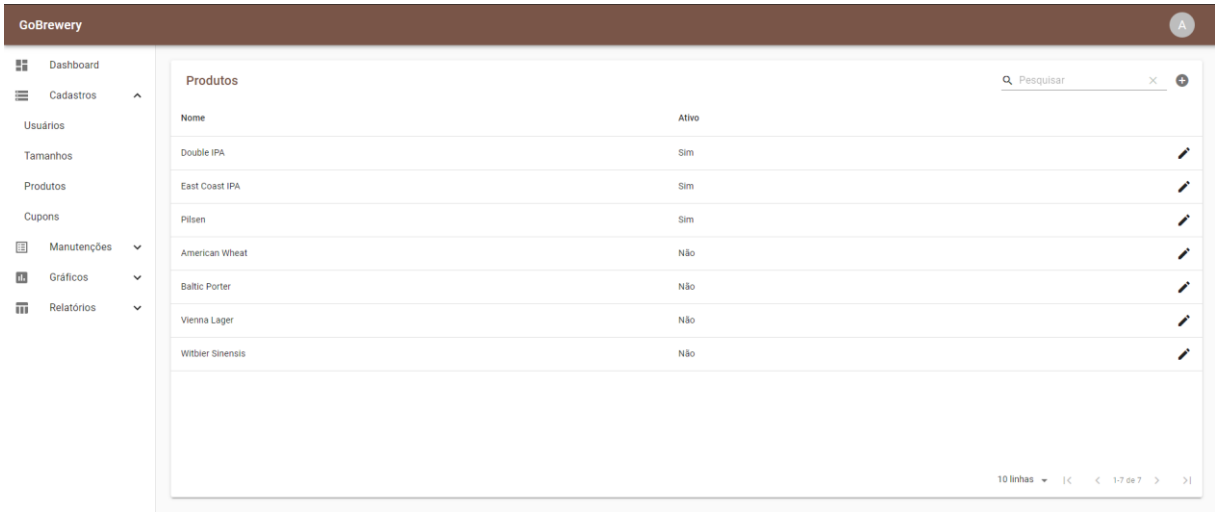
Figura 21 – Formulário de cadastro de tamanho



Fonte: Autoria própria.

A Figura 22 apresenta a listagem de produtos cadastrados e a Figura 23 o formulário de cadastro de produtos. Esse formulário é dividido nas seguintes abas: dados, preços e imagens. A aba de dados abrange os dados gerais do produto.

Figura 22 – Listagem de produtos cadastrados



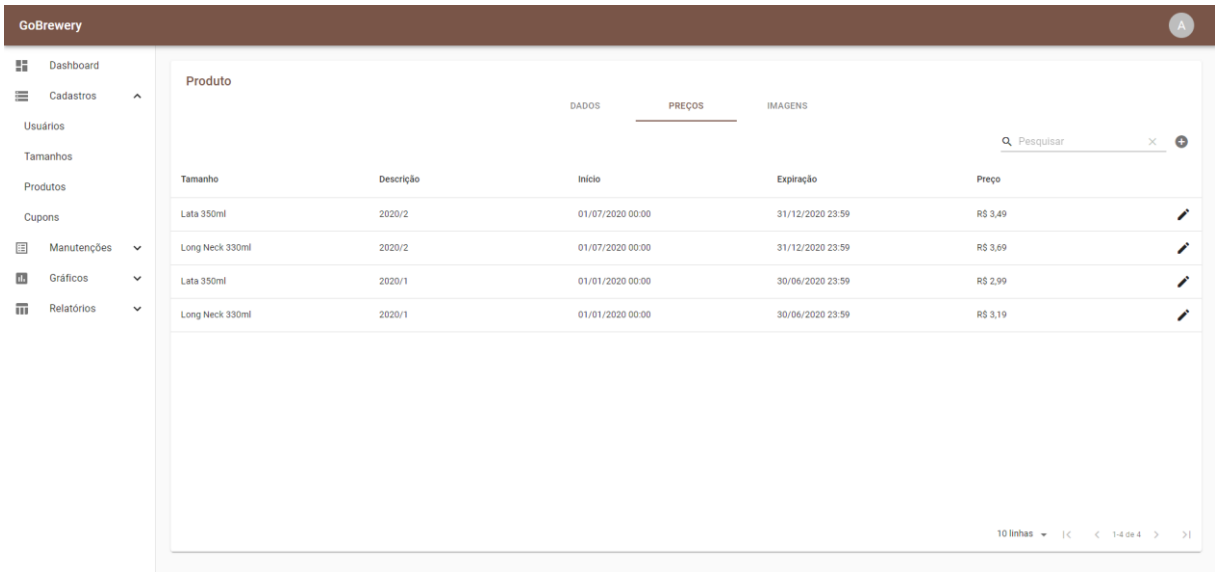
Fonte: Autoria própria.

Figura 23 – Aba de formulário de dados do cadastro de produtos

Fonte: Autoria própria.

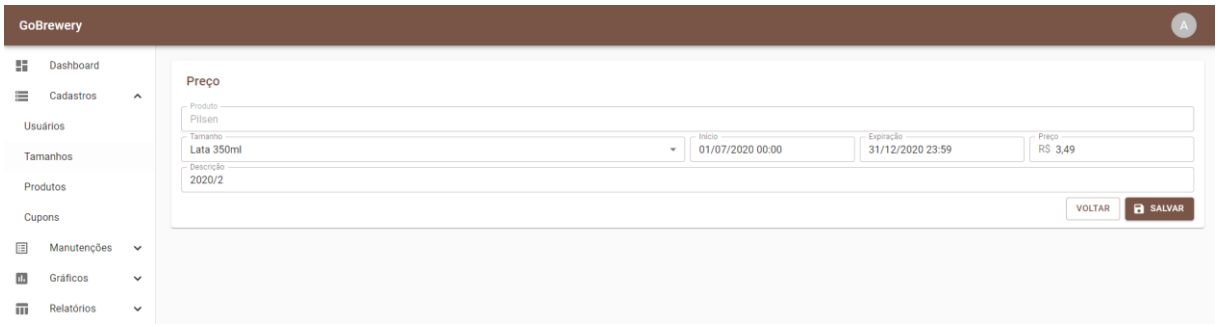
A Figura 24 apresenta a aba de listagem de preços do formulário de cadastro de produtos e a Figura 21 o formulário de cadastro de preços. Preços são divididos por tamanhos e pelo período de vigência do preço, determinados pela data de início e expiração.

Figura 24 – Aba de listagem de cadastro de preços do produto



Fonte: Autoria própria.

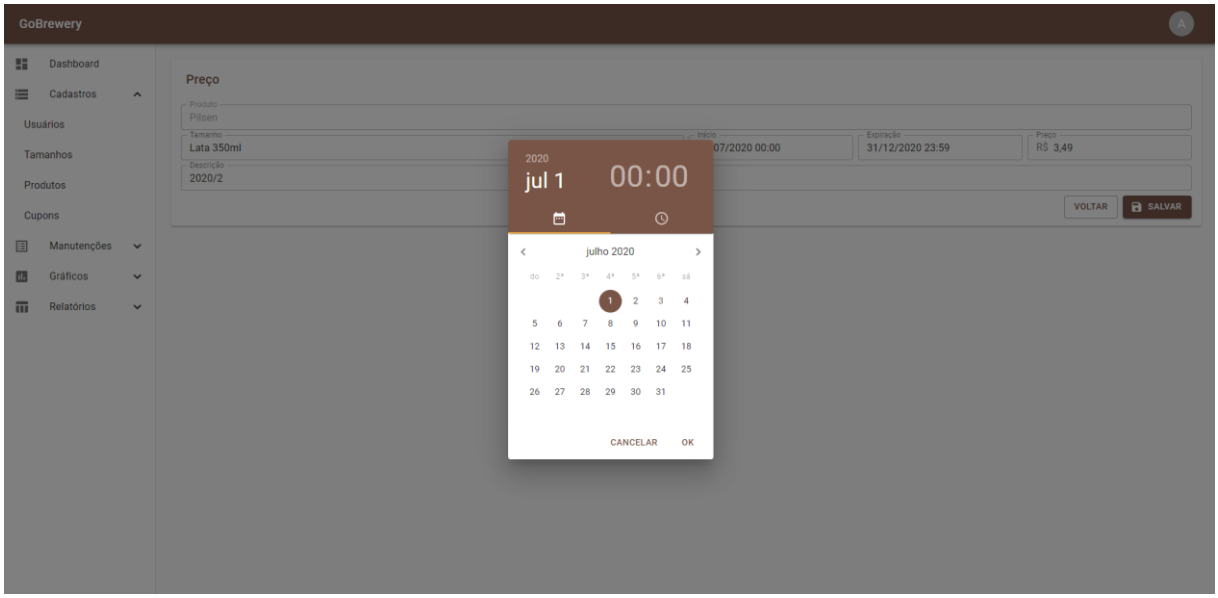
Figura 25 – Formulário de cadastro de preço de produto



Fonte: Autoria própria.

A Figura 26 apresenta o seletor de datas, que é utilizado em todos os formulários em que se faz necessário.

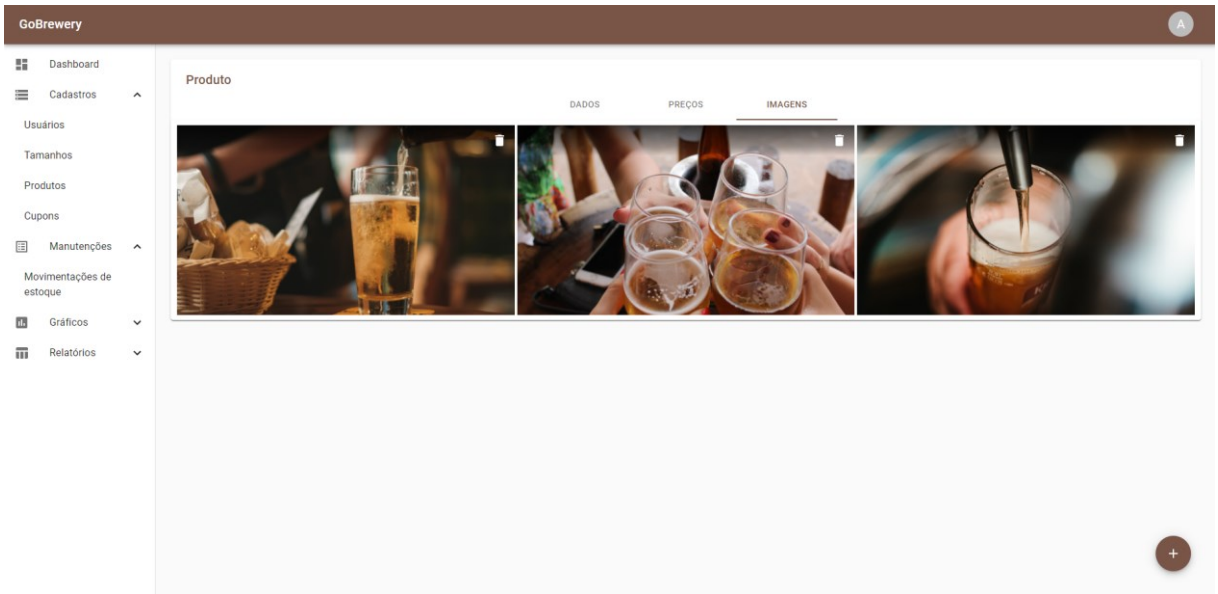
Figura 26 – Seletor de data dos formulários



Fonte: Autoria própria.

A Figura 27 apresenta a aba de imagens do formulário de cadastro de produtos, sendo possível adicionar e remover imagens.

Figura 27 – Aba de listagem e manutenção de imagens do produto

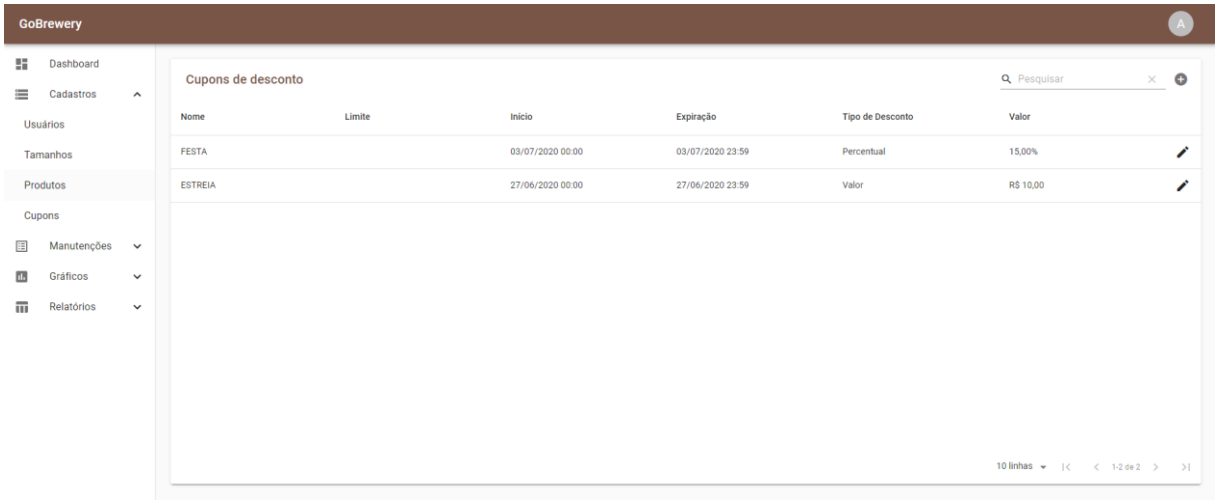


Fonte: Autoria própria.

A Figura 28 apresenta o formulário de cadastro de cupons de desconto. O nome do cupom define o próprio cupom a ser utilizado em vendas. Cupons de desconto tem seu período de durabilidade determinados pela data de início e expiração e podem ter um limite de uso,

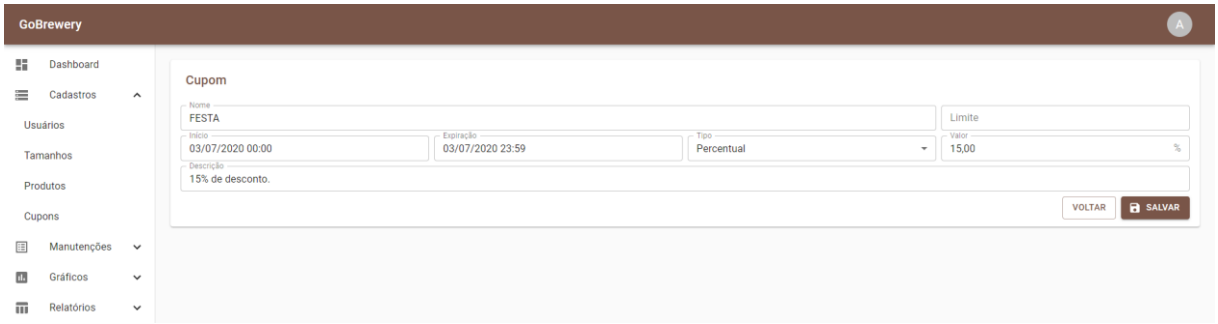
valor mínimo de venda e valor máximo de desconto determinados. É possível cadastrar dois tipos de cupom de desconto, por percentual e por valor, os quais podem ser visualizados na Figura 29 e na Figura 30, respectivamente. Em cupons de desconto percentual, é descontado o percentual do total da venda, em cupons de desconto por valor, é descontado o valor do cupom (em R$) do total da venda.

Figura 28 – Listagem de cupons de desconto cadastrados



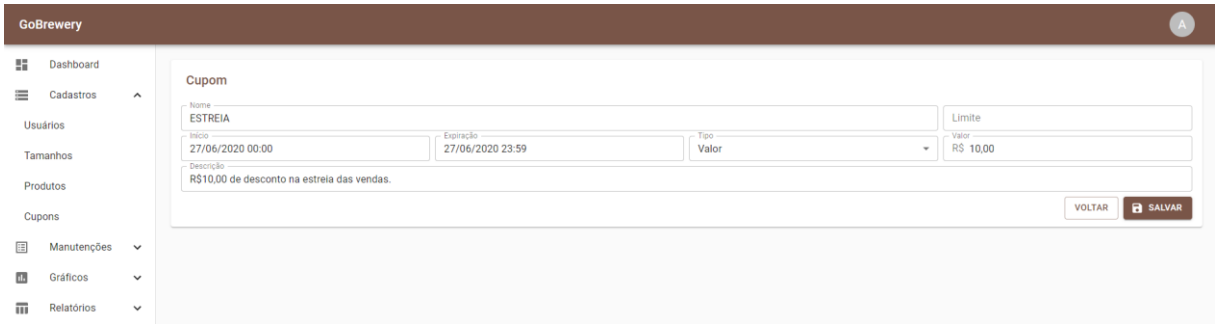
Fonte: Autoria própria.

Figura 29 – Formulário de cadastro de cupons de desconto do tipo percentual



Fonte: Autoria própria.

Figura 30 – Formulário de cadastro de cupons de desconto do tipo valor

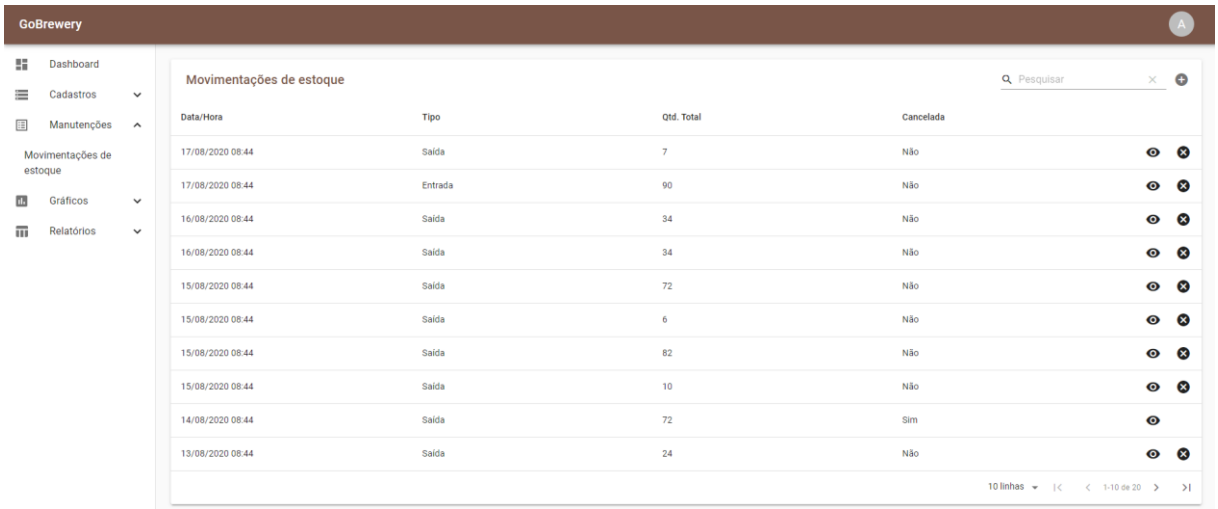


Fonte: Autoria própria.

A Figura 31 apresenta a listagem de movimentações de estoque. É possível realizar novas movimentações, bem como visualizar movimentações existentes, como observado na Figura 33 e na Figura 34, respectivamente.

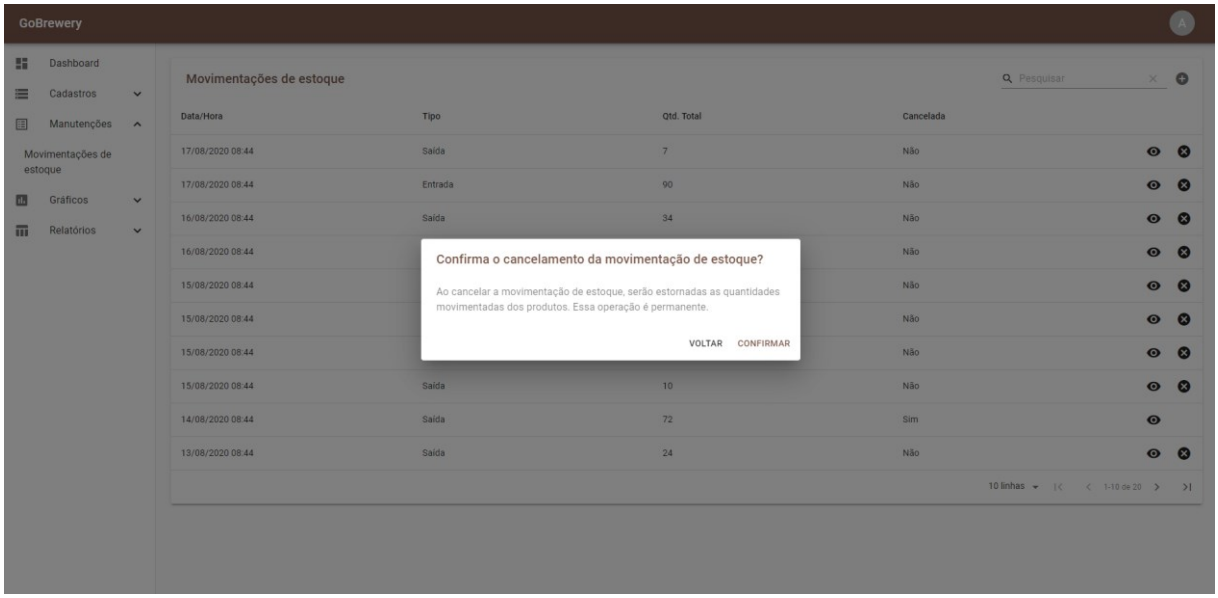
A Figura 32 apresenta a possibilidade, para usuários administradores, de realizar o cancelamento de movimentações de estoque, sendo este cancelamento permanente e estornando as quantidades movimentadas de produtos. Caso algum produto da movimentação a ser cancelada resulte em saldo de estoque negativo, a operação de cancelamento é abortada, sendo necessária a realização de uma nova movimentação para ajustar os saldos de estoque. Movimentações de estoque canceladas possuem um indicador visual em seu título na visualização.

Figura 31 – Listagem de movimentações de estoque



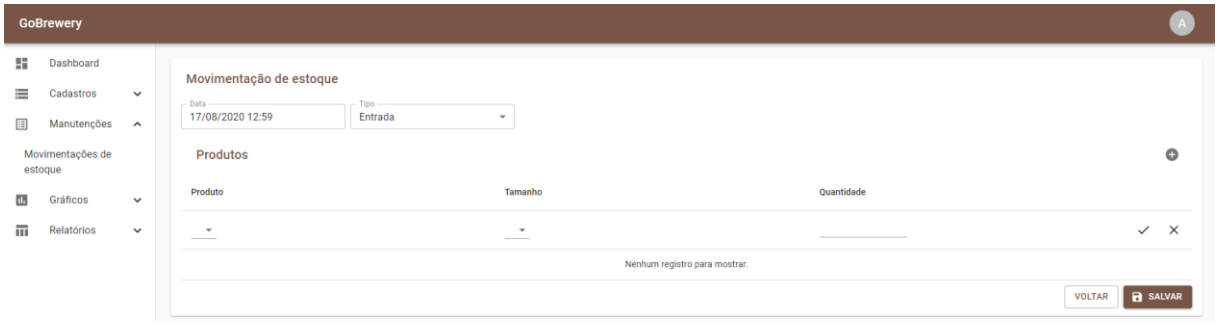
Fonte: Autoria própria.

Figura 32 – Janela de confirmação de cancelamento de movimentação de estoque



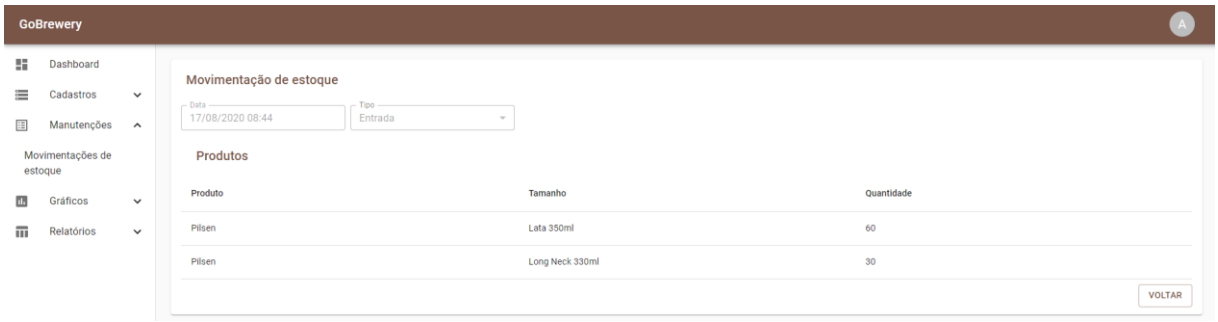
Fonte: Autoria própria.

Figura 33 – Formulário de nova movimentação de estoque



Fonte: Autoria própria.

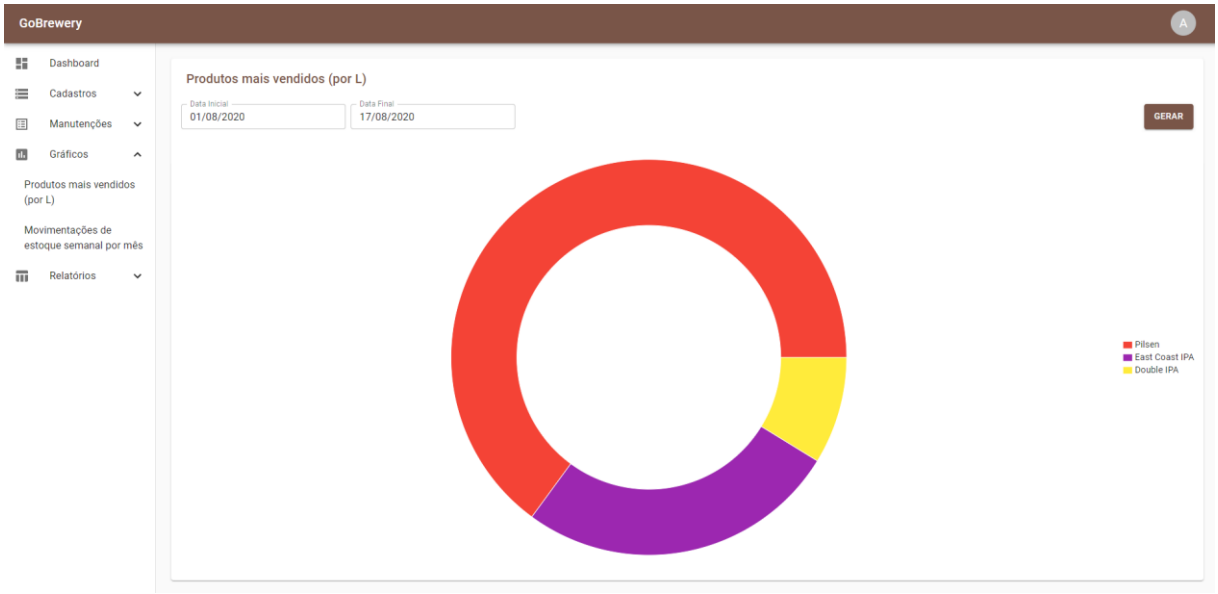
Figura 34 – Visualização de movimentação de estoque



Fonte: Autoria própria.

A Figura 35 apresenta o gráfico de produtos mais vendidos por litro. É possível gerar o gráfico de um determinado período.

Figura 35 – Gráfico de produtos mais vendidos por litro



Fonte: Autoria própria.

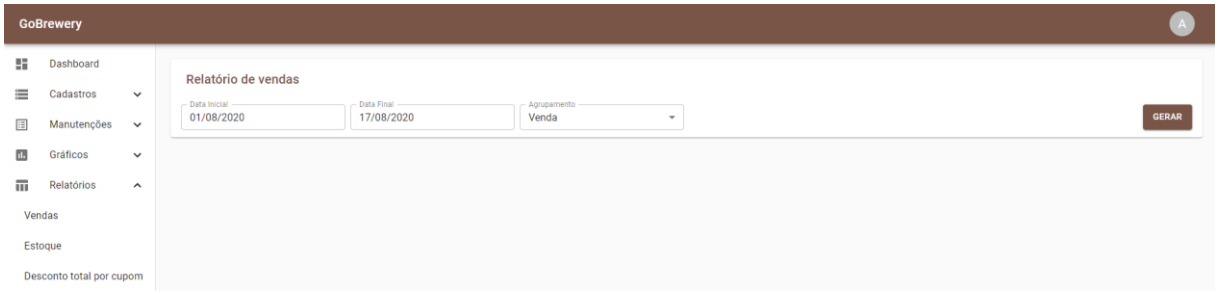
A Figura 36 apresenta o gráfico de movimentações de estoque semanal por mês. Este gráfico agrupa o total de movimentações de entradas e saídas de estoque por semana de um determinado mês.

Figura 36 – Gráfico de movimentações de estoque semanal por mês

Fonte: Autoria própria.

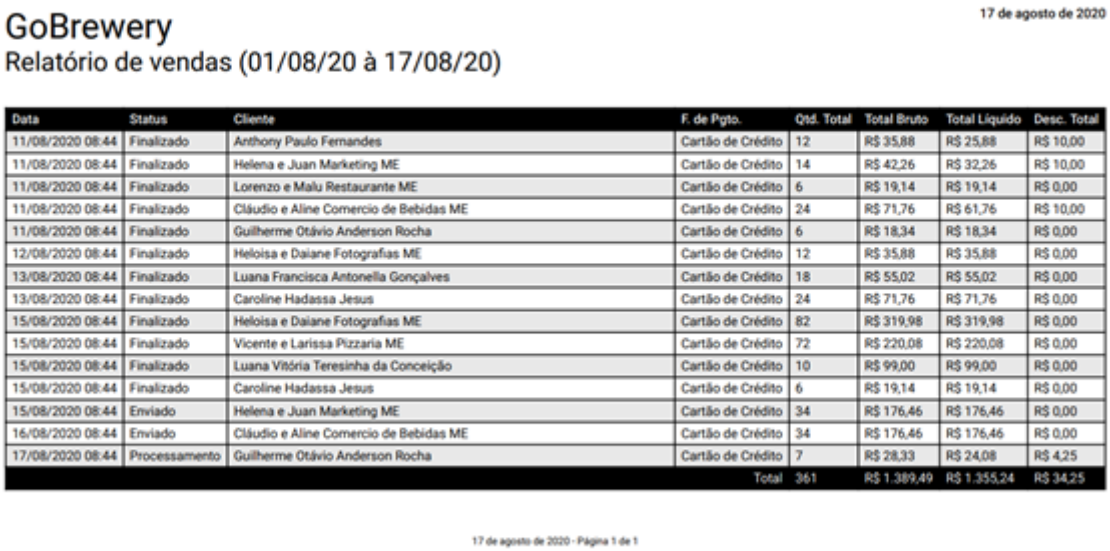
A Figura 37 apresenta os parâmetros de geração de relatório de vendas. É possível gerar o relatório de um determinado período e agrupar tanto por vendas, visualizado na Figura 38, quanto por produtos, visualizado na Figura 39.

Figura 37 – Parâmetros de geração de relatório de vendas



Fonte: Autoria própria.

Figura 38 – Relatório de vendas agrupado por venda



GoBrewery

17 de agosto de 2020

Relatório de vendas (01/08/20 à 17/08/20)

Data	Status	Cliente	F. de Pgto.	Qtd. Total	Total Bruto	Total Líquido	Desc. Total
11/08/2020 08:44	Finalizado	Anthony Paulo Fernandes	Cartão de Crédito	12	R$ 35,88	R$ 25,88	R$ 10,00
11/08/2020 08:44	Finalizado	Helena e Juan Marketing ME	Cartão de Crédito	14	R$ 42,26	R$ 32,26	R$ 10,00
11/08/2020 08:44	Finalizado	Lorenzo e Malu Restaurante ME	Cartão de Crédito	6	R$ 19,14	R$ 19,14	R$ 0,00
11/08/2020 08:44	Finalizado	Cláudio e Aline Comercio de Bebidas ME	Cartão de Crédito	24	R$ 71,76	R$ 61,76	R$ 10,00
11/08/2020 08:44	Finalizado	Guilherme Otávio Anderson Rocha	Cartão de Crédito	6	R$ 18,34	R$ 18,34	R$ 0,00
12/08/2020 08:44	Finalizado	Heloisa e Daiane Fotografias ME	Cartão de Crédito	12	R$ 35,88	R$ 35,88	R$ 0,00
13/08/2020 08:44	Finalizado	Luana Francisca Antonella Gonçalves	Cartão de Crédito	18	R$ 55,02	R$ 55,02	R$ 0,00
13/08/2020 08:44	Finalizado	Caroline Hadassa Jesus	Cartão de Crédito	24	R$ 71,76	R$ 71,76	R$ 0,00
15/08/2020 08:44	Finalizado	Heloisa e Daiane Fotografias ME	Cartão de Crédito	82	R$ 319,98	R$ 319,98	R$ 0,00
15/08/2020 08:44	Finalizado	Vicente e Larissa Pizzaria ME	Cartão de Crédito	72	R$ 220,08	R$ 220,08	R$ 0,00
15/08/2020 08:44	Finalizado	Luana Vitória Teresinha da Conceição	Cartão de Crédito	10	R$ 99,00	R$ 99,00	R$ 0,00
15/08/2020 08:44	Finalizado	Caroline Hadassa Jesus	Cartão de Crédito	6	R$ 19,14	R$ 19,14	R$ 0,00
15/08/2020 08:44	Enviado	Helena e Juan Marketing ME	Cartão de Crédito	34	R$ 176,46	R$ 176,46	R$ 0,00
16/08/2020 08:44	Enviado	Cláudio e Aline Comercio de Bebidas ME	Cartão de Crédito	34	R$ 176,46	R$ 176,46	R$ 0,00
17/08/2020 08:44	Processamento	Guilherme Otávio Anderson Rocha	Cartão de Crédito	7	R$ 28,33	R$ 24,08	R$ 4,25
			Total	361	R$ 1.389,49	R$ 1.355,24	R$ 34,25

17 de agosto de 2020 - Página 1 de 1

Fonte: Autoria própria.

Figura 39 – Relatório de vendas agrupado por produto

GoBrewery

17 de agosto de 2020

Relatório de vendas por produto (01/08/20 à 17/08/20)

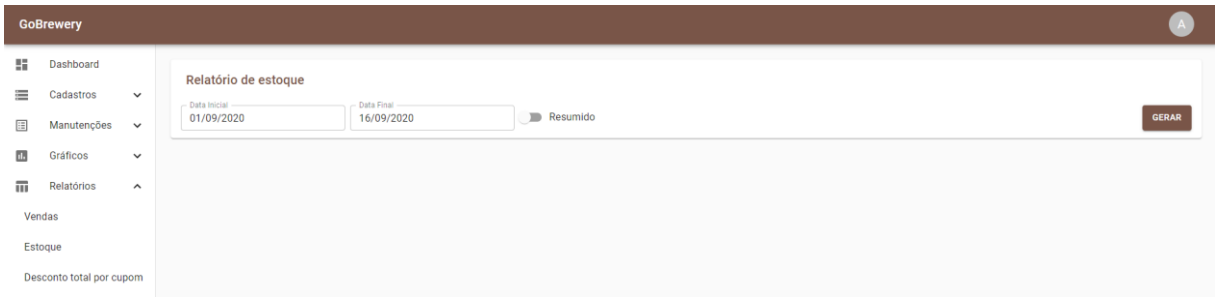
Produto	Tamanho	Preço	Quantidade	Total Bruto
Double IPA	Growler 1,5L	R$ 9,99	10	R$ 99,00
East Coast IPA	Growler 1,5L	R$ 9,99	30	R$ 299,70
Pilsen	Growler 1,5L	R$ 9,99	1	R$ 9,99
Pilsen	Lata 350ml	R$ 2,99	200	R$ 598,00
Pilsen	Long Neck 330ml	R$ 3,19	120	R$ 382,80
		Total	361	R$ 1.389,49

17 de agosto de 2020 - Página 1 de 1

Fonte: Autoria própria.

A Figura 40 apresenta os parâmetros de geração de relatório de estoque. É possível gerar o relatório de um determinado período tanto resumido quanto detalhado. O relatório resumido, visualizado na Figura 41, utiliza como fonte a tabela de sintetização dos saldos de estoque e apresenta apenas o saldo atual, para obter resultados rapidamente. O relatório detalhado, visualizado na Figura 42, além do saldo atual, apresenta o saldo anterior, entradas e saídas contabilizando todas as movimentações realizadas anteriormente para obter o saldo anterior e as movimentações abrangidas no período filtrado, dessa forma, levando mais tempo para obtenção dos resultados.

Figura 40 – Parâmetros de geração de relatório de estoque



Fonte: Autoria própria.

Figura 41 – Relatório resumido de movimentações de estoque

16 de setembro de 2020

GoBrewery

Relatório resumido de estoque (em 16/09/20)

Produto	Tamanho	Saldo Atual
Double IPA	Growler 1,5L	40
East Coast IPA	Growler 1,5L	20
Pilsen	Growler 1,5L	9
Pilsen	Lata 350ml	100
Pilsen	Long Neck 330ml	30
	Total	199

16 de setembro de 2020 - Página 1 de 1

Fonte: Autoria própria.

Figura 42 – Relatório detalhado de movimentações de estoque

17 de agosto de 2020

GoBrewery

Relatório de estoque (01/08/20 à 17/08/20)

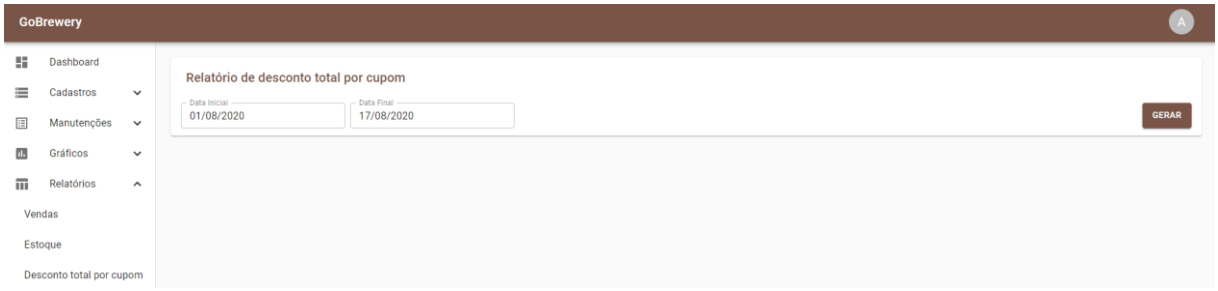
Produto	Tamanho	Saldo Anterior	Entradas	Saídas	Saldo Atual
Double IPA	Growler 1,5L	0	50	10	40
East Coast IPA	Growler 1,5L	0	50	30	20
Pilsen	Growler 1,5L	0	10	1	9
Pilsen	Lata 350ml	0	300	200	100
Pilsen	Long Neck 330ml	0	150	120	30
	Total	0	560	361	199

17 de agosto de 2020 - Página 1 de 1

Fonte: Autoria própria.

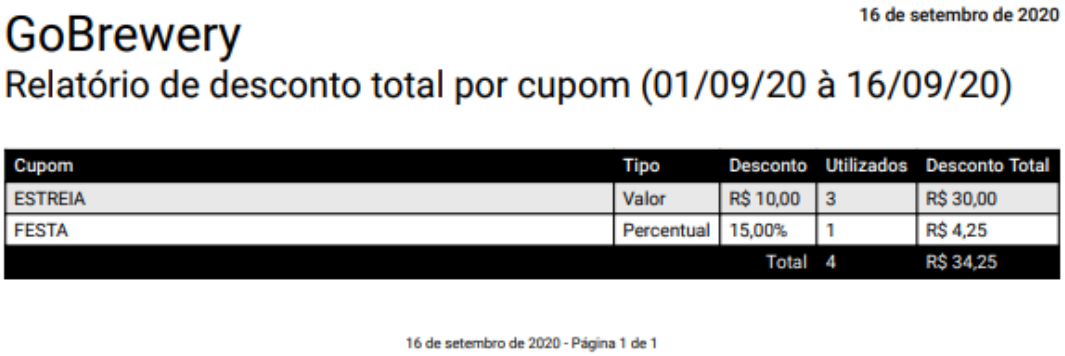
A Figura 43 apresenta os parâmetros de geração de relatório de desconto total por cupom. É possível gerar o relatório de um determinado período, como observado na Figura 44.

Figura 43 – Parâmetros de geração de relatório de desconto total por cupom



Fonte: Autoria própria.

Figura 44 – Relatório de desconto total por cupom



GoBrewery

16 de setembro de 2020

Relatório de desconto total por cupom (01/09/20 à 16/09/20)

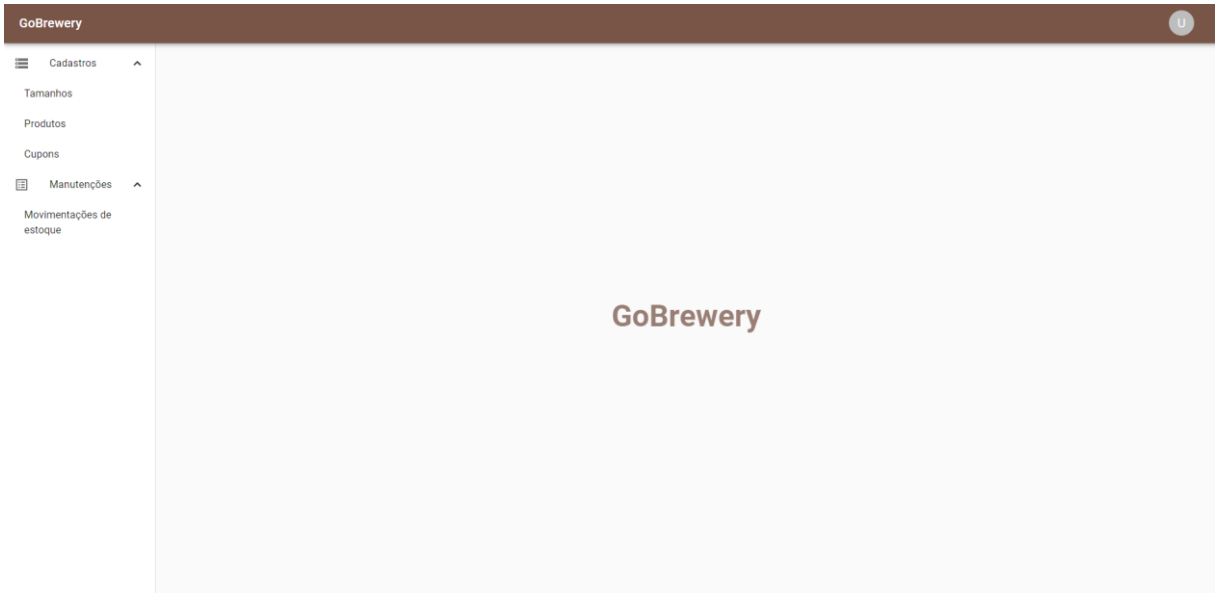
Cupom	Tipo	Desconto	Utilizados	Desconto Total
ESTREIA	Valor	R$ 10,00	3	R$ 30,00
FESTA	Percentual	15,00%	1	R$ 4,25
		Total	4	R$ 34,25

16 de setembro de 2020 - Página 1 de 1

Fonte: Autoria própria.

A Figura 45 apresenta as opções do menu e tela de início para usuários sem permissão de administrador.

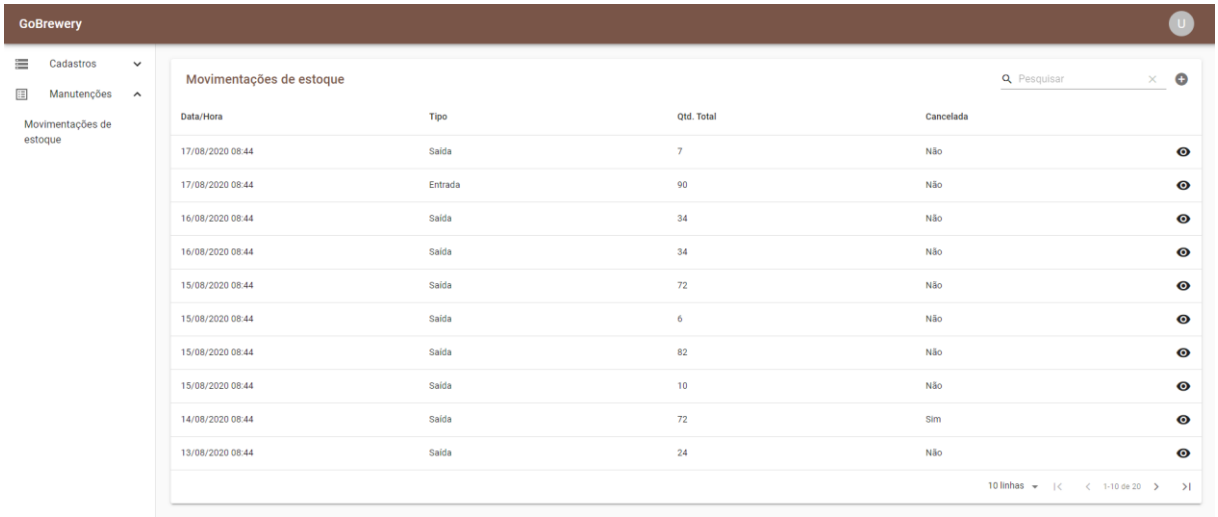
Figura 45 – Menu e início para usuários sem permissão de administrador



Fonte: Autoria própria.

A Figura 46 apresenta a listagem de movimentações de estoque para usuários sem permissão de administrador, onde pode se observar que também não possui a opção de cancelamento de movimentações.

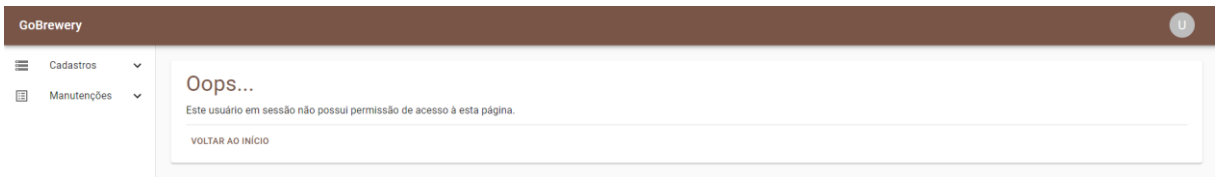
Figura 46 – Listagem de movimentações de estoque para usuários sem permissão de administrador



Fonte: Autoria própria.

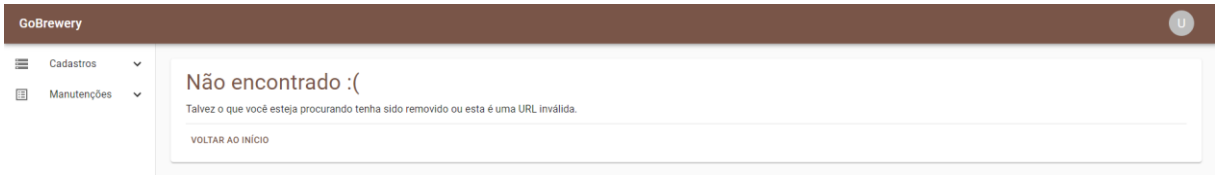
A Figura 47 apresenta a tela de acesso negado e a Figura 48 a tela de conteúdo não encontrado.

Figura 47 – Acesso negado



Fonte: Autoria própria.

Figura 48 – Conteúdo não encontrado



Fonte: Autoria própria.

4.4 IMPLEMENTAÇÃO DO SISTEMA

A implementação do sistema foi dividida em duas seções, sendo a primeira delas o lado servidor da aplicação e a segunda o lado cliente.

4.4.1 Servidor

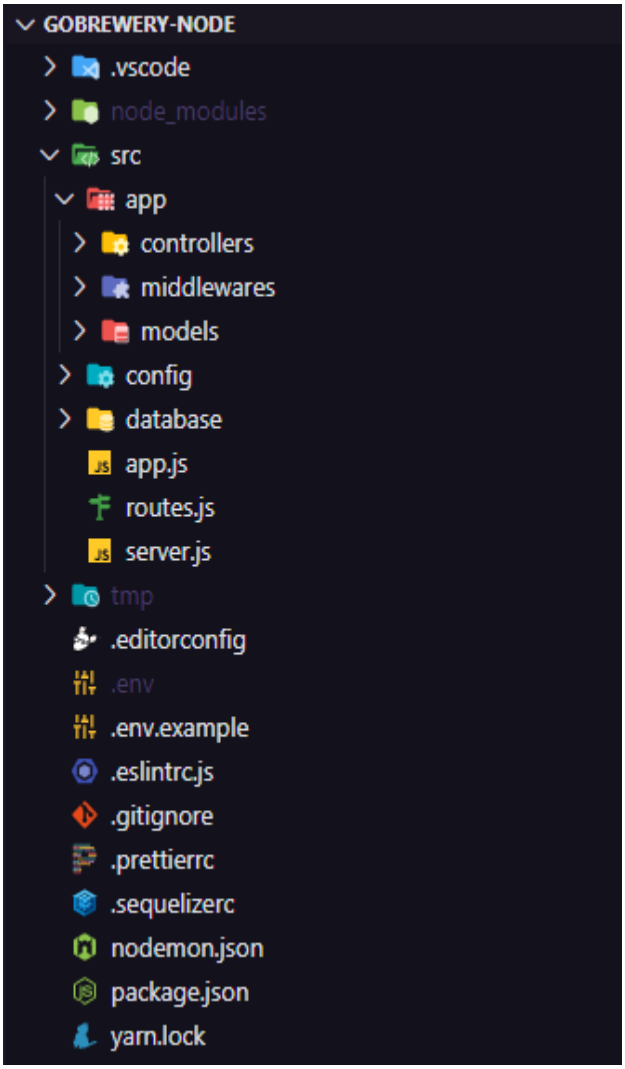
As principais ferramentas utilizadas para o desenvolvimento do servidor foram o *framework Node.js*, juntamente com a biblioteca *Express.js*, facilitadora da implementação de APIs, que foi o formato utilizado neste projeto.

O projeto do servidor foi criado seguindo a estrutura de pastas apresentada na Figura 49. A pasta *config* contém arquivos de configurações gerais do aplicativo, como o de parâmetros de conexão com o banco de dados.

Foi utilizada a biblioteca *Sequelize* para realizar o mapeamento objeto-relacional no projeto, tendo sua configuração centralizada na pasta *database*. Nesta pasta estão contidas as definições das *migrations*, que realizam o controle de versão do banco de dados. Através das *migrations* foi implementada a estrutura do banco de dados do sistema. Também contém as definições das *seeds*, utilizadas para inserção de registros padrão e dados fictícios para demonstração do sistema.

Na pasta *app*, estão contidos os *models*, *controllers* e *middlewares* do servidor. Para cada entidade do banco de dados, existe um arquivo em *models* que a representa. Através dos arquivos de *models*, também é possível realizar a associação entre entidade e até a implementação de *hooks*, que são pequenas funções que podem ser executadas antes ou depois de operações com a entidade.

Figura 49 – Estrutura de pastas do projeto do servidor



Fonte: Autoria própria.

O projeto tem como principais arquivos estruturais: *app.js*, *server.js* e *routes.js*. O arquivo *app.js*, apresentado na Listagem 1, é responsável pela estrutura base do servidor, definindo a instância do próprio servidor, bem como seus *middlewares*, rotas e manipulador de exceções.

Listagem 1 – Estrutura do servidor

```
import 'dotenv/config';

import express from 'express';
import cors from 'cors';
import path from 'path';
import Youch from 'youch';
import 'express-async-errors';
import routes from './routes';

import './database';

class App {
  constructor() {
    this.server = express();
    this.middlewares();
    this.routes();
    this.exceptionHandler();
  }

  middlewares() {
    this.server.use(cors());
    this.server.use(express.json());
    this.server.use(
      '/files',
      express.static(path.resolve(__dirname, '..', 'tmp', 'uploads'))
    );
  }

  routes() {
    this.server.use(routes);
  }

  exceptionHandler() {
    this.server.use(async (err, req, res, next) => {
      if (process.env.NODE_ENV === 'development') {
        const errors = await new Youch(err, req).toJSON();

        return res.status(500).json(errors);
      }

      return res.status(500).json({ error: 'Internal server error.' });
    });
  }
}

export default new App().server;
```

Fonte: Autoria própria.

O arquivo *server.js* define a porta em que o servidor atenderá requisições e o arquivo *routes.js*, visualizado em trecho na Listagem 2, define a estrutura de rotas do servidor. Após a rota de sessão, todas as rotas passam pela validação de autenticação, realizada por meio de *middleware*. Existem rotas para listagem, consultas, cadastro, edição, exclusão e cancelamento, todas seguindo o padrão apresentado. Em algumas rotas são utilizados parâmetros, para indicar

qual registro será manipulado ou para filtragem de dados, por exemplo. Para o carregamento de arquivos é utilizada a biblioteca *Multer*, como *middleware* entre requisição.

Listagem 2 – Trecho de estrutura de rotas do servidor

```
const routes = Router();
const upload = multer(multerConfig);

routes.post('/sessions', SessionController.store);
routes.use(authMiddleware);

routes.get('/profile', ProfileController.index);
routes.put('/profile', ProfileController.update);

routes.post(
  '/profile/avatar',
  upload.single('file'),
  ProfileAvatarController.store
);
routes.delete('/profile/avatar', ProfileAvatarController.delete);

routes.get('/users', administratorMiddleware, UserController.index);
routes.get('/users/:id', administratorMiddleware, UserController.index);
routes.post('/users', administratorMiddleware, UserController.store);
routes.put('/users/:id', administratorMiddleware, UserController.update);
```

Fonte: Autoria própria.

A Listagem 3 apresenta a implementação do *model* de usuário. Todos os *models* do projeto seguem a mesma estrutura.

Listagem 3 – Implementação do *model* de usuário

```
import Sequelize, { Model } from 'sequelize';
import bcrypt from 'bcryptjs';

class User extends Model {
  static init(sequelize) {
    super.init(
      {
        email: Sequelize.STRING,
        password: Sequelize.VIRTUAL,
        password_hash: Sequelize.STRING,
        name: Sequelize.STRING,
        administrator: Sequelize.BOOLEAN,
        active: Sequelize.BOOLEAN,
      },
      { sequelize }
    );

    this.addHook('beforeSave', async user => {
      if (user.password) {
        user.password_hash = await bcrypt.hash(user.password, 8);
      }
    });
```

```
    return this;
  }

  checkPassword(password) {
    return bcrypt.compare(password, this.password_hash);
  }

  static associate(models) {
    this.hasOne(models.UserAvatar, {
      foreignKey: 'user_id',
      as: 'avatar',
    });
  }
}

export default User;
```

Fonte: Autoria própria.

Para cada rota do servidor, existe um arquivo em *controllers* que a implementa. Os *controllers* deste projeto implementam os métodos *index*, *store*, *update* e *delete* que respondem, respectivamente, aos métodos de requisição *get*, *post*, *put* e *delete* da API. Não são implementados todos os métodos em todos os *controllers* necessariamente, a implementação varia de acordo com a necessidade. A Listagem 4 apresenta um exemplo de implementação do método *index*. Todos os *controllers* do projeto seguem a mesma estrutura de implementação de métodos.

Listagem 4 – Implementação do método de busca de registros de tamanhos

```
async index(req, res) {
    if (req.params.id) {
      const productSize = await Size.findByPk(req.params.id, {
        attributes: ['active', 'description'],
      });
      return res.json(productSize);
    }

    const { active } = req.query;

    const where = active ? { active } : {};

    const attributes = active
      ? ['id', 'description']
      : ['id', 'description', 'capacity', 'active'];

    const order = active
      ? ['description']
      : [['active', 'DESC'], 'description'];

    const sizes = await Size.findAll({
      attributes,
      where,
      order,
```

```
    });

    return res.json(sizes);
  }
```

Fonte: Autoria própria.

A Listagem 5 apresenta um exemplo de implementação do método *store* e a Listagem 6 um exemplo de implementação do método *update*. A validação de dados recebidos via requisição é realizada com o auxílio da biblioteca *Yup*, que permite a definição de um formato de objeto base com validações específicas de suas propriedades.

Listagem 5 – Implementação do método de criação de registro de tamanho

```
async store(req, res) {
    const schema = Yup.object().shape({
      description: Yup.string().required(),
      capacity: Yup.number().moreThan(0),
      active: Yup.boolean().required(),
    });

    if (!(await schema.isValid(req.body))) {
      return res.status(400).json({ error: 'Validation fails.' });
    }

    const { id, description, capacity, active } = await
Size.create(req.body);

    return res.json({
      id,
      description,
      capacity,
      active,
    });
  }
```

Fonte: Autoria própria.

Listagem 6 – Implementação do método de atualização de registro de tamanho

```
async update(req, res) {
    const productSize = await Size.findByPk(req.params.id);

    if (!productSize) {
      return res.status(404).json({
        error: 'Size with this given ID was not found.',
      });
    }

    const schema = Yup.object().shape({
      description: Yup.string().required(),
      capacity: Yup.number().moreThan(0),
      active: Yup.boolean(),
    });

    if (!(await schema.isValid(req.body))) {
      return res.status(400).json({ error: 'Validation fails.' });
```

```
    }

    const { id, description, capacity, active } = await productSize.update(
      req.body
    );

    return res.json({
      id,
      description,
      capacity,
      active,
    });
  }
```

Fonte: Autoria própria.

A Listagem 7 apresenta a implementação base de armazenamento de arquivos, onde foi utilizada a biblioteca *Multer* e os arquivos são armazenados em pasta temporária no projeto.

Listagem 7 – Implementação base de armazenamento de arquivos

```
import multer from 'multer';
import crypto from 'crypto';
import { extname, resolve } from 'path';

export default {
  storage: multer.diskStorage({
    destination: resolve(__dirname, '..', '..', 'tmp', 'uploads'),
    filename: (req, file, cb) => {
      crypto.randomBytes(16, (err, res) => {
        if (err) return cb(err);

        return cb(null, res.toString('hex') + extname(file.originalname));
      });
    },
  }),
};
```

Fonte: Autoria própria.

A Listagem 8 apresenta um exemplo de implementação de armazenamento de arquivos de imagens.

Listagem 8 – Implementação do armazenamento de imagens de produtos

```
import { unlinkSync } from 'fs';
import { resolve } from 'path';
import ProductImage from '../models/ProductImage';
import Product from '../models/Product';

class ProductImageController {
  async index(req, res) {
    const images = await ProductImage.findAll({
      where: { product_id: req.params.productId },
      attributes: ['id', 'path', 'url'],
      order: ['id'],
    });
```

```
    return res.json(images);
  }
  async store(req, res) {
    const { id: productId } = await Product.findByPk(req.params.productId);

    if (!productId) {
      return res.status(404).json({ error: 'This product was not found.'
});
    }
    if (!req.file) {
      return res.status(404).json({ error: 'Must upload a file.' });
    }
    const { originalname, filename } = req.file;
    const { id, product_id, url, name, path } = await ProductImage.create({
      product_id: productId,
      name: originalname,
      path: filename,
    });

    return res.json({ id, product_id, url, name, path });
  }
  async delete(req, res) {
    const product = await Product.findByPk(req.params.productId);

    if (!product) {
      return res.status(404).json({ error: 'This product was not found.'
});
    }

    const image = await ProductImage.findByPk(req.params.id);

    if (image) {
      unlinkSync(
        resolve(__dirname, '..', '..', '..', 'tmp', 'uploads', image.path)
      );
      await image.destroy();
    } else {
      return res.status(404).json({
        error: 'Image with this given ID was not found.',
      });
    }
    return res.json();
  }
}
export default new ProductImageController();
```

Fonte: Autoria própria.

O controle de sessão de usuário da aplicação é realizado por meio do arquivo *SessionController.js*, visualizado na Listagem 9. Neste projeto é utilizada a biblioteca *JWT* para a autenticação de usuários.

Listagem 9 – Implementação do método de sessão de usuário

```
async store(req, res) {
    const { email, password } = req.body;

    const user = await User.findOne({
```

```
      where: { email },
      include: {
        model: UserAvatar,
        as: 'avatar',
        attributes: ['id', 'url', 'path'],
      },
    });

    if (!user) {
      return res.status(401).json({ error: 'User not found.' });
    }

    if (!(await user.checkPassword(password))) {
      return res.status(401).json({ error: 'Password does not match.' });
    }

    const { id, name, avatar, administrator } = user;

    return res.json({
      user: {
        id,
        name,
        email,
        avatar,
        administrator,
      },
      token: jwt.sign({ id, administrator }, authConfig.secret),
    });
  }
```

Fonte: Autoria própria.

Os *controllers* de gráficos, relatórios e *dashboard* foram divididos em subpastas separadas. A Listagem 10 apresenta a implementação de um método de busca sintetizada de registros.

Listagem 10 – Implementação de busca para relatório de desconto total por cupom

```
async index(req, res) {
    if (!req.query.startingDate || !req.query.endingDate) {
      return res.status(400).json({
        error: 'Starting and ending date request params must have value.',
      });
    }
    const startingDate = startOfDay(parseISO(req.query.startingDate));
    const endingDate = endOfDay(parseISO(req.query.endingDate));
    if (isAfter(startingDate, endingDate)) {
      return res
        .status(400)
        .json({ error: 'Starting date must be before ending date.' });
    }
    if (isBefore(endingDate, startingDate)) {
      return res
        .status(400)
        .json({ error: 'Ending date must be after starting date.' });
    }
    const coupons = await Sale.findAll({
      include: [
```

```
        {
          model: Coupon,
          as: 'coupon',
          attributes: ['id', 'name', 'type', 'value', 'limit'],
          required: true,
        },
      ],
      attributes: [
        [Database.connection.fn('count', '*'), 'used'],
        [
          Database.connection.fn(
            'sum',
            Database.connection.col('total_discount')
          ),
          'total_discount',
        ],
      ],
      where: {
        date: {
          [Op.between]: [startingDate, endingDate],
        },
        status: {
          [Op.not]: 'C',
        },
      },
      group: ['coupon.id'],
    });

    return res.json(coupons);
  }
```

Fonte: Autoria própria.

Os *middlewares* deste projeto são apenas de verificação de autenticação de usuário e de verificação de privilégios de administrador. A Listagem 11 apresenta o *middleware* de verificação de autenticação de usuário. Todos os *middlewares* do projeto seguem a mesma estrutura.

Listagem 11 – Implementação de *middleware* de verificação de autenticação de usuário

```
import jwt from 'jsonwebtoken';
import { promisify } from 'util';
import authConfig from '../../config/auth';

export default async (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    return res.status(401).json({ error: 'Token not provided.' });
  }

  const [, token] = authHeader.split(' ');

  try {
    const { id } = await promisify(jwt.verify)(token, authConfig.secret);
    req.userId = id;
    return next();
```

```
  } catch (error) {
    return res.status(401).json({ error: 'Invalid token.' });
  }
};
```

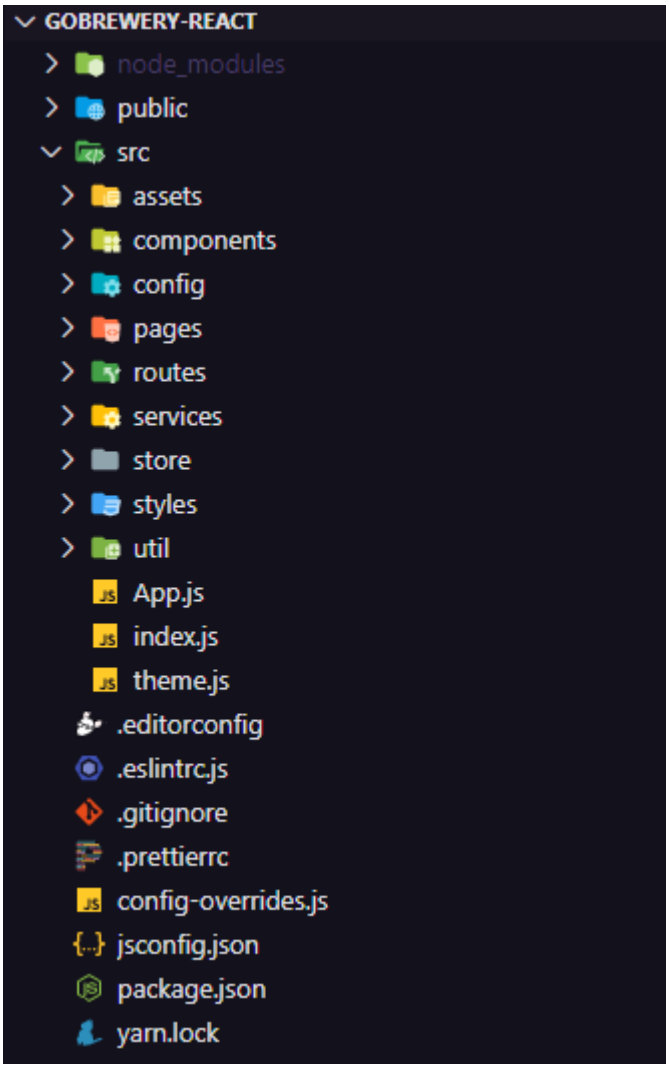
Fonte: Autoria própria.

4.4.2 Aplicação Cliente

As principais ferramentas utilizadas para o desenvolvimento da aplicação *web* foram a biblioteca de interfaces de usuário *React*, juntamente com a biblioteca de componentes visuais *MaterialUI*.

O projeto da aplicação cliente foi criado seguindo a estrutura de pastas apresentada na Figura 50.

Figura 50 – Estrutura de pastas do projeto da aplicação cliente



Fonte: Autoria própria.

O projeto tem como principais arquivos estruturais: *index.html, index.js* e *App.js*. O arquivo *index.html,* localizado na pasta *public,* define a estrutura HTML base para a aplicação cliente, visualizado na Listagem 12. O arquivo *index.js*, visualizado na Listagem 13, renderiza nesta estrutura por meio do *React*, a aplicação definida no arquivo *App.js*, visualizado na Listagem 14.

Listagem 12 – Estrutura HTML base da aplicação cliente

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta   name="viewport"   content="minimum-scale=1,   initial-scale=1,
width=device-width"/>
    <title>GoBrewery</title>
    <link                                                    rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&disp
lay=swap" />
    <link                                                    rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons" />
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Fonte: Autoria própria.

Listagem 13 – Estrutura base de renderização da aplicação cliente

```
import React from 'react';
import ReactDOM from 'react-dom';
import CssBaseline from '@material-ui/core/CssBaseline';
import { ThemeProvider } from '@material-ui/core/styles';
import App from './App';
import GlobalStyle from './styles/global';
import theme from './theme';

ReactDOM.render(
  <React.StrictMode>
    <GlobalStyle />
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <App />
    </ThemeProvider>
  </React.StrictMode>,
  document.getElementById('root')
);
```

Fonte: Autoria própria.

Listagem 14 – Implementação da base da aplicação cliente

```
import React from 'react';
import { Provider } from 'react-redux';
```

```
import { PersistGate } from 'redux-persist/integration/react';
import { Router } from 'react-router-dom';

import './config/ReactotronConfig';

import { store, persistor } from './store';
import Routes from './routes';
import history from './services/history';

import SnackbarProvider from '~/components/SnackbarProvider';

function App() {
  return (
    <Provider store={store}>
      <PersistGate persistor={persistor}>
        <SnackbarProvider />
        <Router history={history}>
          <Routes />
        </Router>
      </PersistGate>
    </Provider>
  );
}

export default App;
```

Fonte: Autoria própria.

A pasta *services* contém os arquivos responsáveis pela conexão com a API do servidor e do histórico de sessão para facilitar a navegação da aplicação.

A pasta *routes* contém as rotas da aplicação, tendo sua implementação visualizada em trecho na Listagem 16, bem como a verificação de sessão de usuário, realizada em toda requisição de página, apresentada na Listagem 15. Caso o usuário não tenha iniciado sessão, é redirecionado para a página de autenticação, indiferente da página requisitada.

Após realizar a autenticação, os usuários com privilégio de administrador, são redirecionados diretamente para a página de *dashboard*, já usuários sem este privilégio, são redirecionados para a página inicial. Usuários sem privilégios de administrador, ao requisitarem páginas que não possuem acesso, são redirecionados para a página de acesso negado. Ao requisitarem rotas inexistentes, são redirecionados para a página de não encontrado.

Listagem 15 – Implementação das validações de sessão e permissões de usuário das rotas

```
import React, { Fragment } from 'react';
import PropTypes from 'prop-types';
import { Route, Redirect } from 'react-router-dom';

import AppContainer from '~/components/AppContainer';

import { store } from '~/store';

export default function RouteWrapper({
```

```
  component: Component,
  isPrivate,
  isAdminRestricted,
  ...rest
}) {
  const { signed } = store.getState().auth;
  const { administrator } = store.getState().user;

  if (!signed && isPrivate) {
    return <Redirect to="/" />;
  }

  if (signed && !isPrivate) {
    if (administrator) {
      return <Redirect to="/dashboard" />;
    }
    return <Redirect to="/home" />;
  }

  if (!administrator && isAdminRestricted) {
    return <Redirect to="/restricted" />;
  }

  const AppLayout = signed ? AppContainer : Fragment;

  return (
    <Route
      {...rest}
      render={(props) => (
        <AppLayout>
          <Component {...props} />
        </AppLayout>
      )}
    />
  );
}

RouteWrapper.propTypes = {
  isPrivate: PropTypes.bool,
  isAdminRestricted: PropTypes.bool,
  component: PropTypes.oneOfType([PropTypes.element, PropTypes.func])
    .isRequired,
};

RouteWrapper.defaultProps = {
  isPrivate: false,
  isAdminRestricted: false,
};
```

Fonte: Autoria própria.

Listagem 16 – Trecho de implementação das rotas da aplicação cliente

```
      <Route path="/" exact component={SignIn} />

      <Route path="/home" component={Home} isPrivate />

      <Route path="/profile" exact component={Profile} isPrivate />
      <Route path="/profile/edit" exact component={ProfileForm} isPrivate
/>
```

```
      <Route
        path="/dashboard"
        component={Dashboard}
        isPrivate
        isAdminRestricted
      />

      <Route
        path="/users"
        exact
        component={Users}
        isPrivate
        isAdminRestricted
      />
      <Route
        path="/users/new"
        exact
        component={UserForm}
        isPrivate
        isAdminRestricted
      />
      <Route
        path="/users/:id"
        exact
        component={UserForm}
        isPrivate
        isAdminRestricted
      />
```

Fonte: Autoria própria.

O estado geral da aplicação é gerenciado via biblioteca *Redux*, tendo sua implementação base apresentada na Listagem 17. O uso dessa biblioteca é por meio de implementação de *reducers* na aplicação, que estão contidos na pasta *store*. *Reducers* desta aplicação estão divididos em módulos e que, em seguida, são combinados para disponibilização na aplicação, como apresenta a Listagem 18. Por meio dos *reducers* são apenas gerenciadas a sessão do usuário com suas respectivas informações e *snackbars*, mensagens de retorno visual ao usuário.

Listagem 17 – Implementação base de controle de estado da aplicação

```
import createSagaMiddleware from 'redux-saga';
import { persistStore } from 'redux-persist';
import createStore from './createStore';
import persistReducers from './persistReducers';

import rootReducer from './modules/rootReducer';
import rootSaga from './modules/rootSaga';

const sagaMonitor =
  process.env.NODE_ENV === 'development'
    ? console.tron.createSagaMonitor()
    : null;

const sagaMiddleware = createSagaMiddleware({ sagaMonitor });
```

```
const middlewares = [sagaMiddleware];

const store = createStore(persistReducers(rootReducer), middlewares);

const persistor = persistStore(store);

sagaMiddleware.run(rootSaga);

export { store, persistor };
```

Fonte: Autoria própria.

Listagem 18 – Implementação base de *reducers*

```
import { combineReducers } from 'redux';

import ui from './ui/reducers';
import auth from './auth/reducers';
import user from './user/reducers';

export default combineReducers({ ui, auth, user });
```

Fonte: Autoria própria.

Também é utilizada a biblioteca *Redux Saga*, que permite a execução de métodos assíncronos em *reducers*, estes nomeados de *sagas* na aplicação, que se fazem necessários para o módulo de autenticação. *Sagas* desta aplicação estão divididos em módulos e que, em seguida, são combinados para disponibilização na aplicação, como apresenta a Listagem 19. Para que esse estado geral não seja perdido ao se atualizar a página ou fechar o navegador, este é persistido no *local storage*, por meio da biblioteca *Redux Persist*.

Listagem 19 – Implementação base de *sagas*

```
import { all } from 'redux-saga/effects';

import auth from './auth/sagas';
import user from './user/sagas';

export default function* rootSaga() {
  return yield all([auth, user]);
}
```

Fonte: Autoria própria.

As ações de manipulação do estado geral desta aplicação foram divididas em *actions*, com seus nomes únicos. Para executar ações, utiliza-se o método *dispatch* com uma *action* como parâmetro, em qualquer página da aplicação. Estas *actions* podem contemplar a execução tanto de *reducers* quanto de *sagas*. A Listagem 20 apresenta um exemplo de implementação de *actions*, a Listagem 21 de *reducers* e a Listagem 22 de *sagas*.

Listagem 20 – Implementação de *actions* de autenticação

```
export function signInRequest(email, password) {
  return {
    type: '@auth/SIGN_IN_REQUEST',
    payload: { email, password },
  };
}

export function signInSuccess(token, user) {
  return {
    type: '@auth/SIGN_IN_SUCCESS',
    payload: { token, user },
  };
}

export function signFailure() {
  return {
    type: '@auth/SIGN_FAILURE',
  };
}

export function signOut() {
  return {
    type: '@auth/SIGN_OUT',
  };
}
```

Fonte: Autoria própria.

Listagem 21 – Implementação de *reducers* de autenticação

```
import produce from 'immer';

const INITIAL_STATE = {
  token: null,
  signed: false,
  loading: false,
};

export default function auth(state = INITIAL_STATE, action) {
  return produce(state, (draft) => {
    switch (action.type) {
      case '@auth/SIGN_IN_REQUEST': {
        draft.loading = true;
        break;
      }
      case '@auth/SIGN_IN_SUCCESS': {
        draft.token = action.payload.token;
        draft.signed = true;
        draft.loading = false;
        break;
      }
      case '@auth/SIGN_FAILURE': {
        draft.loading = false;
        break;
      }
      case '@auth/SIGN_OUT': {
        draft.token = null;
        draft.signed = false;
        break;
```

```
      }
      default:
    }
  });
}
```

Fonte: Autoria própria.

Listagem 22 – Implementação de *sagas* de autenticação

```
import { takeLatest, call, put, all } from 'redux-saga/effects';

import { signInSuccess, signFailure } from './actions';
import { showSnackbar } from '~/store/modules/ui/actions';

import history from '~/services/history';
import api from '~/services/api';

export function* signIn({ payload }) {
  try {
    const { email, password } = payload;

    const response = yield call(api.post, 'sessions', {
      email,
      password,
    });

    const { token, user } = response.data;

    api.defaults.headers.Authorization = `Bearer ${token}`;

    yield put(signInSuccess(token, user));

    history.push('/');
  } catch (error) {
    yield  put(showSnackbar('warning',  'Verifique  os  dados  e  tente
novamente.'));
    yield put(signFailure());
  }
}

export function setToken({ payload }) {
  if (!payload) return;

  const { token } = payload.auth;

  if (token) {
    api.defaults.headers.Authorization = `Bearer ${token}`;
  }
}

export function signOut() {
  history.push('/');
}

export default all([
  takeLatest('persist/REHYDRATE', setToken),
  takeLatest('@auth/SIGN_IN_REQUEST', signIn),
  takeLatest('@auth/SIGN_OUT', signOut),
]);
```

Fonte: Autoria própria.

Na pasta *pages* estão contidos os arquivos de implementação de todas as páginas por rotas da aplicação. As páginas de cadastros foram divididas estruturalmente entre listagens e formulários. A Listagem 23 apresenta a implementação de uma página de listagem de registros e a Listagem 24 a implementação de uma página de formulário. As páginas de formulário possuem dois estados: criação e edição de registro. Também utilizam a biblioteca *Yup* para validação de dados do formulário. Para auxiliar no gerenciamento de estado dos formulários, foi utilizada a biblioteca *Formik*.

Listagem 23 – Implementação de página de listagem de usuários

```
import React, { useState, useEffect } from 'react';
import { useDispatch } from 'react-redux';
import Typography from '@material-ui/core/Typography';
import MaterialTable from 'material-table';
import { options, localization } from '~/config/MaterialTableConfig';

import api from '~/services/api';
import history from '~/services/history';

import { showSnackbar } from '~/store/modules/ui/actions';

function Users() {
  const dispatch = useDispatch();

  const [data, setData] = useState([]);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    (async function loadData() {
      try {
        const response = await api.get('users');
        setData(response.data);
      } catch (error) {
        dispatch(showSnackbar('error', 'Não foi possível carregar os
dados.'));
      }
      setIsLoading(false);
    })();
  }, [dispatch]);

  return (
    <MaterialTable
      title={
        <Typography variant="h6" color="primary">
          Usuários
        </Typography>
      }
      columns={[
        { title: 'Nome', field: 'name' },
        { title: 'E-mail', field: 'email' },
        {
          title: 'Ativo',
          field: 'active',
```

```
          render: (rowData) => (rowData.active ? 'Sim' : 'Não'),
        },
        {
          title: 'Administrador',
          field: 'administrator',
          render: (rowData) => (rowData.administrator ? 'Sim' : 'Não'),
        },
      ]}
      data={data}
      isLoading={isLoading}
      localization={localization.ptBR}
      options={options}
      actions={[
        {
          icon: 'add_circle',
          tooltip: 'Adicionar',
          isFreeAction: true,
          onClick: (_event) => history.push('/users/new'),
        },
        {
          icon: 'edit',
          tooltip: 'Editar',
          onClick:            (_event,            rowData)            =>
history.push(`/users/${rowData.id}`),
        },
      ]}
    />
  );
}

export default Users;
```

Fonte: Autoria própria.

Listagem 24 – Implementação de página de formulário de usuário

```
import React, { useState, useCallback } from 'react';
import { useParams, Link } from 'react-router-dom';
import { useDispatch } from 'react-redux';
import {
  Paper,
  Typography,
  Grid,
  FormControlLabel,
  Button,
  Backdrop,
  CircularProgress,
} from '@material-ui/core';
import Save from '@material-ui/icons/Save';
import { Formik, Field, Form } from 'formik';
import { TextField, Switch } from 'formik-material-ui';
import * as Yup from 'yup';

import Loader from '~/components/Loader';

import api from '~/services/api';
import history from '~/services/history';

import { showSnackbar } from '~/store/modules/ui/actions';

import style from './styles';
```

```
function UserForm() {
  const classes = style();

  const dispatch = useDispatch();

  const { id } = useParams();

  const [isSubmitting, setIsSubmitting] = useState(false);

  const [initialValues, setInitialValues] = useState({
    name: '',
    email: '',
    active: true,
    administrator: false,
  });

  const loadValues = useCallback(async () => {
    if (id) {
      const res = await api.get(`users/${id}`);
      if (res.data) {
        const { name, email, active, administrator } = res.data;
        setInitialValues({ name, email, active, administrator });
      }
    }
  }, [id]);

  const validationSchema = Yup.object().shape({
    name: Yup.string().required('Obrigatório.'),
    email:                                    Yup.string().email('E-mail
inválido.').required('Obrigatório.'),
  });

  const handleSubmit = async (values) => {
    try {
      setIsSubmitting(true);
      if (id) {
        await api.put(`users/${id}`, values);
      } else {
        const data = { ...values, password: 'gobrewery' };
        await api.post('users', data);
      }
      setIsSubmitting(false);
      dispatch(showSnackbar('success', 'Salvo com sucesso.'));
      history.push('/users');
    } catch (error) {
      setIsSubmitting(false);
      dispatch(showSnackbar('error', 'Não foi possível salvar.'));
    }
  };

  return (
    <Paper>
      <Loader loadFunction={loadValues}>
        <Formik
          enableReinitialize
          initialValues={initialValues}
          validationSchema={validationSchema}
          onSubmit={handleSubmit}
        >
          <Form>
```

```
            <Typography            variant="h6"            color="primary"
className={classes.title}>
              Usuário {!id && '(Novo)'}
            </Typography>
            <Grid container spacing={1} className={classes.container}>
              <Grid item xs={6}>
                <Field
                  component={TextField}
                  type="text"
                  label="Nome"
                  name="name"
                  variant="outlined"
                  size="small"
                  fullWidth
                />
              </Grid>
              <Grid item xs={6}>
                <Field
                  component={TextField}
                  type="text"
                  label="E-mail"
                  name="email"
                  variant="outlined"
                  size="small"
                  fullWidth
                />
              </Grid>
              <Grid item xs={12}>
                <FormControlLabel
                  control={
                    <Field        component={Switch}        name="active"
type="checkbox" />
                  }
                  label="Ativo"
                />
                <FormControlLabel
                  control={
                    <Field
                      component={Switch}
                      name="administrator"
                      type="checkbox"
                    />
                  }
                  label="Administrador"
                />
              </Grid>
              <Grid item xs={12} className={classes.buttons}>
                <Button
                  type="submit"
                  variant="contained"
                  color="primary"
                  disabled={isSubmitting}
                  className={classes.button}
                  startIcon={<Save />}
                >
                  Salvar
                </Button>
                <Button
                  type="button"
                  variant="outlined"
                  color="primary"
```

```
                  component={Link}
                  to="/users"
                >
                  Voltar
                </Button>
              </Grid>
            </Grid>
          </Form>
        </Formik>
      </Loader>
      <Backdrop open={isSubmitting} className={classes.backdrop}>
        <CircularProgress color="primary" />
      </Backdrop>
    </Paper>
  );
}

export default UserForm;
```

Fonte: Autoria própria.

Também foram divididas em pastas as páginas de gráficos e relatórios. Para a implementação dos gráficos foi utilizada a biblioteca *Recharts*. A Listagem 25 apresenta um exemplo de implementação de gráfico.

Listagem 25 – Implementação gráfico de produtos mais vendidos por litro

```
import React, { useState, useCallback, useEffect } from 'react';
import { useDispatch } from 'react-redux';
import { Paper, Grid, Typography, Button } from '@material-ui/core';
import { PieChart, Pie, ResponsiveContainer, Legend, Tooltip } from
'recharts';
import { getColor } from 'random-material-color';
import { MuiPickersUtilsProvider, DatePicker } from '@material-ui/pickers';
import { startOfMonth } from 'date-fns';
import DateFnsUtils from '@date-io/date-fns';
import ptBR from 'date-fns/locale/pt-BR';

import api from '~/services/api';

import { showSnackbar } from '~/store/modules/ui/actions';

import style from './styles';

function BestSellersByLiter() {
  const classes = style();

  const dispatch = useDispatch();

  const [startingDateParam, setStartingDateParam] = useState(
    startOfMonth(new Date())
  );
  const [endingDateParam, setEndingDateParam] = useState(new Date());
  const [chartData, setChartData] = useState([{}]);

  const generateChart = useCallback(
```

```
    async (startingDate, endingDate) => {
      try {
        const res = await api.get('charts/best-sellers-by-liter', {
          params: { startingDate, endingDate },
        });

        if (res.data) {
          const formattedData = res.data.map((r) => {
            return {
              name: r.product.name,
              value: Number(r.liters),
              fill: getColor({
                shades: ['500'],
                text: r.product.id.toString(),
              }),
            };
          });

          setChartData(formattedData);
        }
      } catch (error) {
        dispatch(showSnackbar('error',    'Não    foi    possível    gerar
gráfico.'));
      }
    },
    [dispatch]
  );

  useEffect(() => {
    generateChart(startOfMonth(new Date()), new Date());
  }, [generateChart]);

  function handleGenerate() {
    generateChart(startingDateParam, endingDateParam);
  }

  return (
    <Paper className={classes.paper}>
      <Typography variant="h6" color="primary" className={classes.title}>
        Produtos mais vendidos (por L)
      </Typography>
      <MuiPickersUtilsProvider utils={DateFnsUtils} locale={ptBR}>
        <Grid container spacing={1} className={classes.container}>
          <Grid item xs={2}>
            <DatePicker
              label="Data Inicial"
              inputVariant="outlined"
              size="small"
              fullWidth
              ampm={false}
              format="dd/MM/yyyy"
              cancelLabel="Cancelar"
              maxDate={endingDateParam}
              maxDateMessage="Data  inicial  deve  ser  menor  que  a  data
final."
              value={startingDateParam}
              onChange={setStartingDateParam}
            />
          </Grid>
          <Grid item xs={2}>
            <DatePicker
```

```
              label="Data Final"
              inputVariant="outlined"
              size="small"
              fullWidth
              ampm={false}
              format="dd/MM/yyyy"
              cancelLabel="Cancelar"
              minDate={startingDateParam}
              minDateMessage="Data final deve ser maior que a data
inicial."
              value={endingDateParam}
              onChange={setEndingDateParam}
            />
          </Grid>
          <Grid item xs={8} className={classes.buttons}>
            <Button
              type="button"
              variant="contained"
              color="primary"
              className={classes.button}
              onClick={handleGenerate}
            >
              Gerar
            </Button>
          </Grid>
        </Grid>
      </MuiPickersUtilsProvider>
      <ResponsiveContainer>
        <PieChart>
          <Pie
            data={chartData}
            dataKey="value"
            innerRadius="60%"
            outerRadius="90%"
          />
          <Legend layout="vertical" align="right" verticalAlign="middle"
/>
          <Tooltip />
        </PieChart>
      </ResponsiveContainer>
    </Paper>
  );
}

export default BestSellersByLiter;
```

Fonte: Autoria própria.

Para a implementação dos relatórios foi utilizada a biblioteca *PDFMake* e com esta foi desenvolvido um gerador, como observado na Listagem 26, que é utilizado por todos os relatórios da aplicação. A Listagem 27 apresenta um exemplo de relatório.

Listagem 26 – Implementação do gerador de relatórios

```
import pdfMake from 'pdfmake/build/pdfmake';
import vfsFonts from 'pdfmake/build/vfs_fonts';
import { format } from 'date-fns';
import ptBR from 'date-fns/locale/pt-BR';
```

```
function makeCell(content, rowIndex = -1, options = {}) {
  return {
    text: content,
    fillColor: rowIndex % 2 ? 'white' : '#e8e8e8',
    ...options,
  };
}

function th(content, rowIndex = -1, options = {}) {
  return makeCell(content, rowIndex, {
    bold: true,
    fontSize: 9,
    fillColor: 'black',
    color: 'white',
    ...options,
  });
}

function td(content, rowIndex = -1, options = {}) {
  return makeCell(content, rowIndex, {
    bold: false,
    fontSize: 9,
    ...options,
  });
}

function generateReport(
  reportTitle,
  pageOrientation,
  columnsWidth,
  reportRows
) {
  const { vfs } = vfsFonts.pdfMake;
  pdfMake.vfs = vfs;

  const createDocumentDefinition = (
    reportDate,
    subHeading,
    ...contentParts
  ) => {
    const baseDocDefinition = {
      pageSize: 'A4',
      pageOrientation,
      footer: (currentPage, pageCount) => {
        return {
          text: `${reportDate} - Página ${currentPage.toString()} de
${pageCount.toString()}`,
          alignment: 'center',
          fontSize: 7,
        };
      },
      styles: {
        title: {
          fontSize: 24,
        },
        titleSub: {
          fontSize: 18,
        },
        titleDate: {
          fontSize: 9,
```

```
          alignment: 'right',
          bold: true,
        },
      },
      content: [
        {
          columns: [
            {
              text: 'GoBrewery',
              style: 'title',
              width: '*',
            },
            { text: reportDate, style: 'titleDate', width: '160' },
          ],
        },
        { text: `${subHeading}\n\n`, style: 'titleSub' },
      ],
    };
    const docDefinition = JSON.parse(JSON.stringify(baseDocDefinition));
    docDefinition.footer = baseDocDefinition.footer;
    if (contentParts) {
      docDefinition.content.push(...contentParts);
    }
    return docDefinition;
  };

  const tableData = {
    table: {
      headerRows: 1,
      widths: columnsWidth,
      body: reportRows,
    },
  };

  const documentDefinition = createDocumentDefinition(
    format(new Date(), "d 'de' MMMM 'de' yyyy", { locale: ptBR }),
    reportTitle,
    tableData
  );

  pdfMake.createPdf(documentDefinition).open();
}

export { th, td, generateReport };
```

Fonte: Autoria própria.

Listagem 27 – Implementação de relatório de desconto total por cupom

```
import React, { useState } from 'react';
import { useDispatch } from 'react-redux';
import { Paper, Typography, Grid, Button } from '@material-ui/core';
import { MuiPickersUtilsProvider, DatePicker } from '@material-ui/pickers';
import { startOfMonth, format } from 'date-fns';
import DateFnsUtils from '@date-io/date-fns';
import ptBR from 'date-fns/locale/pt-BR';

import { formatCurrency, formatPercentage } from '~/util/format';

import api from '~/services/api';
```

```
import { showSnackbar } from '~/store/modules/ui/actions';

import { th, td, generateReport } from '~/util/report';

import style from './styles';

function TotalDiscountByCoupon() {
  const classes = style();

  const dispatch = useDispatch();

  const [startingDate, setStartingDate] = useState(startOfMonth(new
Date()));
  const [endingDate, setEndingDate] = useState(new Date());

  function report(data) {
    const formatType = (type) => {
      switch (type) {
        case 'P':
          return 'Percentual';
        case 'V':
          return 'Valor';
        default:
          return '';
      }
    };

    const formatValue = (type, value) => {
      switch (type) {
        case 'P':
          return formatPercentage(value);
        case 'V':
          return formatCurrency(value);
        default:
          return '';
      }
    };

    const generateReportBody = (rows) => {
      const body = [
        [
          th('Cupom'),
          th('Tipo'),
          th('Desconto'),
          th('Disponibilizados'),
          th('Utilizados'),
          th('Desconto Total'),
        ],
      ];

      let totalLimit = 0;
      let totalUsed = 0;
      let totalDiscount = 0;

      rows.forEach((row, index) => {
        const tableRow = [];
        tableRow.push(
          td(row.coupon.name, index),
          td(formatType(row.coupon.type), index),
          td(formatValue(row.coupon.type, row.coupon.value), index),
          td(row.coupon.limit, index),
```

```
          td(row.used, index),
          td(formatCurrency(row.total_discount), index)
        );
        body.push(tableRow);

        totalLimit += Number(row.coupon.limit);
        totalUsed += Number(row.used);
        totalDiscount += Number(row.total_discount);
      });

      body[rows.length + 1] = [
        td('Total', -1, {
          colSpan: 3,
          fillColor: 'black',
          color: 'white',
          alignment: 'right',
        }),
        td(''),
        td(''),
        td(totalLimit, -1, {
          fillColor: 'black',
          color: 'white',
        }),
        td(totalUsed, -1, {
          fillColor: 'black',
          color: 'white',
        }),
        td(formatCurrency(totalDiscount), -1, {
          fillColor: 'black',
          color: 'white',
        }),
      ];

      return body;
    };

    const periodStart = format(startingDate, 'dd/MM/yy');
    const periodEnd = format(endingDate, 'dd/MM/yy');

    generateReport(
      `Relatório de desconto total por cupom (${periodStart} à
${periodEnd})`,
      'portrait',
      ['*', 'auto', 'auto', 'auto', 'auto', 'auto'],
      generateReportBody(data)
    );
  }

  const handleGenerate = async () => {
    try {
      const response = await api.get('reports/total-discount-by-coupon', {
        params: { startingDate, endingDate },
      });

      if (response.data) {
        report(response.data);
      }
    } catch (error) {
      dispatch(showSnackbar('error',     'Não     foi     possível     gerar
relatório.'));
    }
```

```
  };

  return (
    <Paper>
      <Typography variant="h6" color="primary" className={classes.title}>
        Relatório de desconto total por cupom
      </Typography>
      <MuiPickersUtilsProvider utils={DateFnsUtils} locale={ptBR}>
        <Grid container spacing={1} className={classes.container}>
          <Grid item xs={2}>
            <DatePicker
              label="Data Inicial"
              inputVariant="outlined"
              size="small"
              fullWidth
              ampm={false}
              format="dd/MM/yyyy"
              cancelLabel="Cancelar"
              maxDate={endingDate}
              maxDateMessage="Data inicial deve ser menor que a data
final."
              value={startingDate}
              onChange={setStartingDate}
            />
          </Grid>
          <Grid item xs={2}>
            <DatePicker
              label="Data Final"
              inputVariant="outlined"
              size="small"
              fullWidth
              ampm={false}
              format="dd/MM/yyyy"
              cancelLabel="Cancelar"
              minDate={startingDate}
              minDateMessage="Data final deve ser maior que a data
inicial."
              value={endingDate}
              onChange={setEndingDate}
            />
          </Grid>
          <Grid item xs={8} className={classes.buttons}>
            <Button
              type="button"
              variant="contained"
              color="primary"
              className={classes.button}
              onClick={handleGenerate}
            >
              Gerar
            </Button>
          </Grid>
        </Grid>
      </MuiPickersUtilsProvider>
    </Paper>
  );
}

export default TotalDiscountByCoupon;
```

Fonte: Autoria própria.

Todas as páginas são compostas por componentes, tanto da biblioteca de componentes visuais *MaterialUI*, quanto de componentes desenvolvidos para esta aplicação que se localizam na pasta *components*. A Listagem 28 apresenta o componente de carregador de listagens, que indica visualmente, com progresso circular, ao usuário o carregamento de registros.

Listagem 28 – Implementação de componente de carregador de listagens

```
import React, { useState, useEffect, useCallback } from 'react';
import PropTypes from 'prop-types';
import { CircularProgress, Typography, Button } from '@material-ui/core';
import Refresh from '@material-ui/icons/Refresh';

import style from './styles';

function Loader({ loadFunction, children }) {
  const classes = style();

  const [state, setState] = useState({});

  const handleLoad = useCallback(async () => {
    try {
      setState({ isLoading: true, loadError: false });
      await loadFunction();
      setState({ isLoading: false, loadError: false });
    } catch (error) {
      setState({ isLoading: false, loadError: true });
    }
  }, [loadFunction]);

  useEffect(() => {
    handleLoad();
  }, [handleLoad]);

  return (
    <>
      {state.isLoading && (
        <div className={classes.wrapper}>
          <CircularProgress className={classes.item} />
        </div>
      )}
      {state.loadError && (
        <div className={classes.wrapper}>
          <Typography variant="body1" className={classes.item}>
            Algo deu errado.
          </Typography>
          <Button
            type="button"
            variant="outlined"
            color="primary"
            className={classes.item}
            onClick={handleLoad}
          >
            <Refresh /> Tentar novamente
          </Button>
        </div>
      )}
```

```
      {!state.isLoading && !state.loadError && children}
    </>
  );
}

Loader.propTypes = {
  loadFunction: PropTypes.func.isRequired,
  children: PropTypes.element.isRequired,
};

export default Loader;
```

Fonte: Autoria própria.

A pasta *config* contém arquivos de configurações gerais do aplicativo, como as configurações visuais padrão das tabelas da aplicação. A pasta *utils* contém arquivos úteis, como o formatador de números.

5 CONCLUSÃO

Foi desenvolvido neste trabalho um sistema de gestão de comércio eletrônico de produtos de uma cervejaria artesanal, com o objetivo de prover suporte ao gestor e auxiliar na tomada de decisões, fornecendo acesso rápido e facilitado à dados sintetizados, visando facilitar o entendimento e a visualização do estado atual do negócio.

Para este trabalho foi utilizado o ecossistema ao redor do *JavaScript*, com *Node.js* no lado servidor e *React* na aplicação cliente para o usuário final, sendo ambas ferramentas extremamente eficazes no cumprimento do projeto e gerando um grande valor em aprendizado.

Com um grande número de bibliotecas *open source* facilitadoras de implementação de funcionalidades cobertas por este trabalho, juntamente com o tempo de necessário para o aprendizado das mesmas, não foi possível a implementação do portfólio que também constava como um objetivo deste projeto durante o desenvolvimento do trabalho de conclusão de curso 1. Entretanto foi realizado a implementação de carregamento e visualização de imagens no cadastro de produtos. Também foi desenvolvido um controle de perfil de usuário para usufruir das bibliotecas à disposição que já se faziam necessárias na aplicação, que não estava previsto no escopo inicial do projeto.

Por fim, certamente podem ser implementadas melhorias para este projeto, como a autenticação de dois fatores, recuperação de senha via *e-mail*, opção de lembrar usuário e expiração de sessão, relatórios e gráficos específicos e dinâmicos, e maior número de mensagens informativas ao usuário.

REFERÊNCIAS

ASADI, Mosen; DALIRI, Mohammad-Reza; ALIPOUR, Navid. **Managing product lines variability in Rich Internet Applications**. In: 2nd International Enterprise Distributed Object Computing Conference. IEEE. 2018, p. 208-217.

BERNARDI, Mario Luca; DI LUCCA, Giuseppe Antonio; DISTANTE, Damiano. **Model-driven fast prototyping of RIAs**: from conceptual models to running applications. IEEE, 2014, p. 250-258.

BUSCH, Marianne; KOCH, Nora. **Rich internet applications**: state-of-the-art. Technical Report 0902. Programming and Software Engineering Unit (PST). Institute for Informatics. Ludwig-Maximilians-Universität München, Germany. December 2009, p. 1-18.

FRATERNALI, Piero; ROSSI, Gustavo; SANCHEZ-FIGUEROA, Fernando. Rich Internet Applications. **IEEE Internet Computing**, v. 9, n. 12, may/june 2010, p. 9-12.

MELIÁ, Santiago; GÓMEZ, Jaime; PÉREZ, Sandy; DÍAZ, Oscar. Architectural and Technological Variability in Rich Internet Applications. **IEEE Internet Computing**. 2010. p. 24-32.

NUÑEZ, Guido; GONZÁLEZ, Magali; AQUINO, Nathalie; CERNUZZI, Luca. **A Model-Driven Approach to develop Rich Web Applications**. In: XLIII Latin American Computer Conference (CLEI) 2017, p. 1-10.

PRECIADO, Juan Carlos; LINAJE, Marino; COMAI, Sara; SANCHEZ-FIGUEROA, Fernando. **Designing rich internet applications with web engineering methodologies**. In: 9th IEEE Workshop Web Site Evolution (WSE 07), IEEE CS Press, 2007, p. 23-30.

ROUBI, Sarra; ERRAMDANI, Mohammed; MBARKI, Samir. **A model driven approach to generate graphical user interfaces for Rich Internet Applications using interaction flow modeling language.** In: International Conference on Intelligent Systems Design and Applications (ISDA), 2015, p. 272-276

VALVERDE, Francisco; PASTOR, Oscar. **Facing the technological challenges of Web 2. 0**: a RIA model-driven engineering approach**.** In: International Conference on WISE, 2009, p. 1-14.