UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
Bacharelado em Sistemas de Informação
Mariana Morais Santiago Moreira

COMPARATIVO ENTRE DIFERENTES ABORDAGENS DE RENDERIZAÇÃO E EFEITOS NO *SEARCH ENGINE OPTIMIZATION* TENDO O DESENVOLVIMENTO DE UM E-COMMERCE COMO ESTUDO DE CASO

Diamantina
2025

Mariana Morais Santiago Moreira

COMPARATIVO ENTRE DIFERENTES ABORDAGENS DE RENDERIZAÇÃO E EFEITOS NO *SEARCH ENGINE OPTIMIZATION* TENDO O DESENVOLVIMENTO DE UM E-COMMERCE COMO ESTUDO DE CASO

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri (UFVJM), como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientadora: Profª. Claudia Beatriz Berti

Diamantina

2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI

FOLHA DE APROVAÇÃO

Mariana Morais Santiago Moreira

COMPARATIVO ENTRE DIFERENTES ABORDAGENS DE RENDERIZAÇÃO E EFEITOS NO *SEARCH ENGINE OPTIMIZATION* TENDO O DESENVOLVIMENTO DE UM E-COMMERCE COMO ESTUDO DE CASO

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri, como requisitos parcial para conclusão do curso.

Orientadora: Prof.ª Dr.ª Claudia Beatriz Berti

Aprovado em 4 de Dezembro de 2025

BANCA EXAMINADORA

Profª. Drª. Claudia Beatriz Berti
Faculdade de Ciências Exatas - DECOM - UFVJM

Prof. Dr. Alessandro Vivas Andrade
Faculdade de Ciências Exatas - DECOM - UFVJM

Profª. Drª. Kattiana Fernandes Constantino
Faculdade de Ciências Exatas - DECOM - UFVJM

Documento assinado eletronicamente por **Claudia Beatriz Berti**, **Servidor(a)**, em 11/12/2025, às 15:29, 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 **Alessandro Vivas Andrade**, **Servidor(a)**, em 11/12/2025, às 15:41, 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 **Kattiana Fernandes Constantino**, **Servidor(a)**, em 12/12/2025, às 21:27, 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.ufvjm.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1976585** e o código CRC **779A61CA**.

Referência: Processo nº 23086.000729/2023-83 SEI nº 1976585

AGRADECIMENTOS

Primeiramente, agradeço aos meus professores, pois sem eles nada disso seria possível. Agradeço pela paciência, dedicação e paixão pelo trabalho que inspiram a mim e a tantos outros estudantes. Um agradecimento especial aos docentes Claudia Berti, Cinthya Rocha, Alessandro Vivas, Marcus Guelpeli, Kattiana Constantino – poderia mencionar todos aqueles que fizeram parte da minha trajetória. Muito obrigada pelos ensinamentos e conhecimentos que, com certeza, levarei para toda a minha vida.

Aos meus amigos e colegas Jhonathan, Pedro, Augusto, Ian e Victor, agradeço de coração por todos os momentos que compartilhamos. Vocês permitiram que esse caminho fosse mais leve e divertido e, por essa razão, meus mais sinceros agradecimentos.

Agradeço à minha família por todo o apoio. Agradeço ao meu irmão por se fazer presente mesmo estando distante. Agradeço a Deus por me dar força e perseverança, mesmo em momentos difíceis.

RESUMO

A maior parte dos sistemas atualmente são desenvolvidos especificamente para a *Internet*, sendo moldados com base nos protocolos, na infraestrutura e no ecossistema da *Internet*. Com base nisso, é essencial compreender o quanto uma escolha de arquitetura pode impactar não somente na performance do sistema em si, mas em todo o negócio de uma empresa – uma vez que a visibilidade orgânica (SEO) e a experiência do usuário (UX) são fatores decisivos para as vendas, e que estão intrinsecamente ligados ao potencial lucro da empresa. Nesse contexto, o presente trabalho tem como objetivo traçar um comparativo entre as diferentes arquiteturas de renderização de conteúdo (*Client-Side Rendering*, *Server-Side Rendering*, *Static Site Generation*, *Incremental Static Regeneration* e renderização híbrida) e descobrir qual é a melhor abordagem para um sistema de *e-commerce*, tendo o desenvolvimento do site da empresa fictícia Alto do Vale como estudo de caso. Para atingir esse objetivo, o *e-commerce* foi desenvolvido utilizando as tecnologias Django REST framework, Nuxt, SASS e PostgreSQL e aplicando a arquitetura distribuída baseada em microsserviços. A metodologia Kanban foi aplicada durante todo o processo, para garantir a organização e agilidade no desenvolvimento. Além disso, o comportamento de cada arquitetura foi demonstrado de forma prática, o que permitiu, como resultado, escolher a abordagem ideal para o sistema.

Palavras-chave: *Search Engine Optimization* (SEO), Arquitetura de *Software*, Renderização, Comércio Eletrônico, *Client-Side Rendering* (CSR), *Server-Side Rendering* (SSR), *Static Site Generation* (SSG), *Incremental Static Regeneration* (ISR).

ABSTRACT

The majority of current systems are developed specifically for the Internet, being molded by the Internet’s protocols, infrastructure, and ecosystem. Given this premise, it is essential to understand the extent to which an architectural choice can impact not only the system's performance but also the company's entire business, considering that organic visibility (SEO) and user experience (UX) are decisive factors for sales and are intrinsically linked to the company's potential profit. In this context, the present work aims to draw a comparison between the different content rendering architectures (Client-Side Rendering, Server-Side Rendering, Static Site Generation, Incremental Static Regeneration, and hybrid rendering) and identify the optimal approach for an e-commerce system, using the development of the fictional company Alto do Vale's website as a case study. To achieve this objective, the e-commerce platform was developed utilizing the technologies Django REST framework, Nuxt, SASS, and PostgreSQL, and applying a distributed, microservices-based architecture. The Kanban methodology was applied throughout the entire process to ensure organization and agility in the development workflow. Furthermore, the behavior of each architecture was demonstrated practically, which resulted in the selection of the ideal approach for the system.

Keywords: Search Engine Optimization (SEO), Software Architecture, Rendering, Electronic Commerce, Client-Side Rendering (CSR), Server-Side Rendering (SSR), Static Site Generation (SSG), Incremental Static Regeneration (ISR).

LISTA DE IMAGENS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SEO	*Search Engine Optimization*
UI	*User Interface*
UX	*User Experience*
CSR	*Client-Side Rendering*
SSR	*Server-Side Rendering*
SSG	*Static Site Generation*
ISR	*Incremental Static Regeneration*
SASS	*Syntactically Awesome Style Sheets*
API	*Application Programming Interface*
B2C	*Business to Consumer*
HTML	*HyperText Markup Language*
CSS	*Cascading Style Sheets*
Web	*World Wide Web*
JSON	JavaScript *Object Notation*
DOM	*Document Object Model*
IDE	*Integrated Development Environment*
DRY	*Don’t Repeat Yourself*
JWT	*JSON Web Token*
ORM	*Object Relational Mapper*
HTTP	*Hypertext Transfer Protocol*

SUMÁRIO

1 INTRODUÇÃO

Existem inúmeros fatores que determinam o sucesso de um negócio digital, como um *e-commerce*. Entre eles, um dos mais relevantes é o *Search Engine Optimization* (SEO), que consiste na otimização do site com a finalidade de garantir uma boa posição nos mecanismos de busca como o Google. Quanto melhor o posicionamento no *ranking* de busca, maior é a quantidade de visitantes e, consequentemente, maiores são as taxas de vendas.

Nesse contexto, surge o conceito de renderização. A renderização consiste no processo de conversão de um código, escrito em linguagens como HTML, CSS e JavaScript, em uma página *web* visual e interativa. A forma como essa renderização é programada tem impacto direto no SEO, na posição em *rankings* de busca, na experiência do usuário, na performance e no desempenho do site. Diante disso, comprova-se que a escolha da estratégia de renderização é um dos fatores que impactam, mesmo que indiretamente, no sucesso de um negócio que depende de visualizações, vendas, notoriedade e tráfego orgânico.

Entre os tipos de renderização existem: o *Client-Side Rendering* (CSR); o *Server-Side Rendering* (SSR); o *Static Site Generation* (SSG); o *Incremental Static Regeneration* (ISR) e a renderização híbrida, que consiste na utilização de duas ou mais abordagens em um mesmo site. A escolha do tipo de renderização faz-se crítica especialmente em sistemas distribuídos, onde há comunicação via API. Esse tipo de arquitetura de *software* tem se tornado cada vez mais predominante, o que ressalta a necessidade de se compreender em totalidade o funcionamento da renderização de conteúdo.

De maneira resumida, o *Client-Side Rendering* (CSR), como o próprio nome sugere, consiste em um tipo de renderização em que o conteúdo do site é carregado apenas do lado do cliente via JavaScript. Ou seja, antes que o cliente acesse o *link*, o site é apenas uma página em branco. Essa é a abordagem padrão utilizada por ferramentas comuns hoje em dia, como o React, Angular e Vue. Contudo, uma vez que as páginas armazenadas nos servidores são vazias, os mecanismos de busca como o Google são incapazes de ler o conteúdo do site, o que dificulta a indexação no Google, faz com que o site tenha uma péssima posição em *rankings* de busca e dificilmente seja encontrado por potenciais clientes.

De maneira contrária, o SSR, o SSG, o ISR e a abordagem híbrida surgem como alternativas para a resolução da problemática causada pelo CSR. Cada um desses padrões de renderização possui suas próprias vantagens e desvantagens e são recomendados para casos específicos.

Nesse cenário, busca-se entender cada um dos padrões e encontrar a melhor opção para um *e-commerce* ou sistema semelhante, a fim de garantir boa visibilidade e desempenho. A Alto do Vale surge como empresa fictícia de bebidas e desempenha o papel de estudo de caso para a presente pesquisa.

1.1 Objetivos

O objetivo geral deste trabalho consiste em estudar e comparar as diferentes arquiteturas de renderização – a fim de solucionar o problema causado pelo CSR – e encontrar a melhor solução para um sistema de *e-commerce*, tendo o site da empresa fictícia Alto do Vale como estudo de caso.

Os objetivos específicos são:

1. Estudar e comparar as principais arquiteturas de renderização *web*.
2. Desenvolver o *e-commerce* da Alto do Vale de forma que atenda a seus requisitos funcionais e não-funcionais.
3. Aplicar cada uma das técnicas de renderização e analisar os seus efeitos.

1.2 Justificativa

Este trabalho representa um estudo de caso em relação às possíveis abordagens de renderização na *web*, sendo assim relevante academicamente ao demonstrar uma aplicação prática sobre o impacto da escolha arquitetural em um site. Além disso, o presente trabalho traz novas perspectivas sobre uma arquitetura recente e promissora – o *Incremental Static Regeneration* (ISR) – sendo o primeiro trabalho acadêmico brasileiro a apresentar de maneira prática o potencial dessa nova abordagem.

Além da importância acadêmica, o estudo também contribui para a construção deste e de outros sistemas, de forma a garantir boa visibilidade, performance e melhor custo benefício.

1.3 Estrutura do Trabalho

No segundo capítulo, cada arquitetura é discutida de maneira mais aprofundada, sendo seguido de um terceiro capítulo que traz a perspectiva de diferentes autores. Por conseguinte, o quarto capítulo aborda as metodologias e tecnologias que guiaram o desenvolvimento do presente trabalho. O capítulo quinto apresenta detalhes da implementação do *e-commerce* e comparação prática dos comportamentos das arquiteturas, sendo seguido do capítulo sexto, o qual traz as considerações finais.

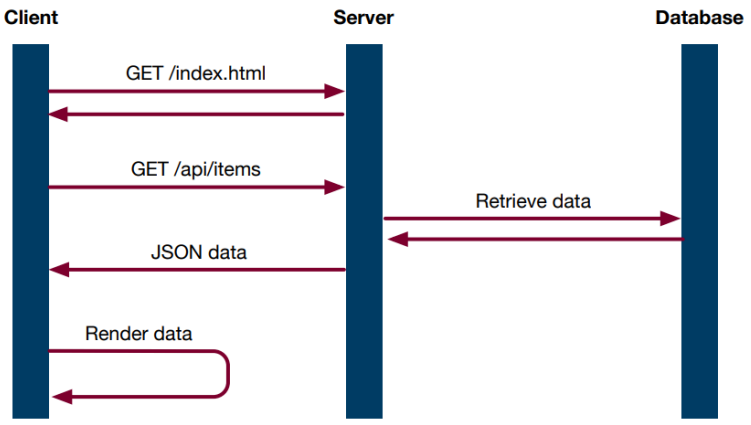
2 REFERENCIAL TEÓRICO

Com o crescimento exponencial da tecnologia, os motores de busca tornaram-se ferramentas indispensáveis, não apenas para as empresas, como também para consumidores que os utilizam a fim de obter informações e conteúdos (Paiva, 2018). Dessa forma, é fundamental que as empresas ocupem lugar de destaque nas primeiras posições nos *rankings* de pesquisa, uma vez que o comportamento padrão dos consumidores é não ir além dos primeiros resultados (Paiva, 2018). Nesse contexto, é importante compreender que a abordagem de renderização tem impacto direto no SEO de um site. Além disso, inúmeras pesquisas comprovam que o CSR apresenta incompatibilidade com motores de busca, resultando em um SEO reduzido (Bezerra, 2024), o que ressalta a importância do estudo de diferentes alternativas de renderização.

2.1 Client-Side Rendering

Segundo M. Beke (2018), o CSR consiste em um tipo de renderização comum em aplicações modernas. Nesse tipo de arquitetura, a máquina do cliente faz uma requisição ao servidor do *front-end*, que retorna dados no formato JSON, e esse conteúdo é renderizado no navegador do cliente através de JavaScript. Esse é o padrão aplicado por frameworks comuns como Angular, React e Vue. A Figura 1 demonstra o fluxo de interações dessa abordagem.

Figura 1 – Fluxo do *Client-Side Rendering*.



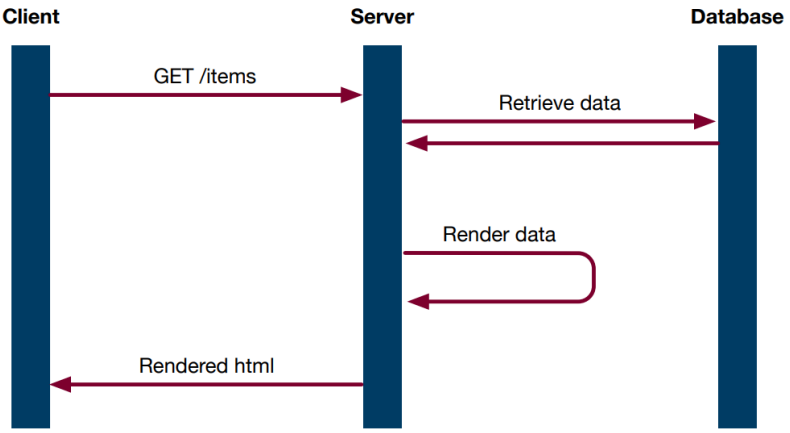
Fonte: (M. Beke, 2018)

J. Santos (2024) reforça que o CSR pode resultar em uma experiência de usuário mais fluida e carregamento mais rápido, uma vez que a manipulação do *Document Object Model* (DOM) ocorre diretamente no navegador do usuário. Contudo, essa prática resulta, também, em um SEO reduzido, pois os motores de busca não conseguem ler o conteúdo das páginas, uma vez que o conteúdo é carregado na máquina do cliente e não no servidor. (Bezerra, 2024).

2.2 Server-Side Rendering

Ainda de acordo com M. Beke (2018), o *Server-Side Rendering* (SSR) consiste em uma abordagem distinta. Nessa arquitetura, o cliente faz requisições ao servidor e o próprio servidor renderiza os dados e retorna ao cliente a página HTML renderizada ao invés de um documento JSON, como pode-se ver na Figura 2.

Figura 2 – Fluxo do *Server-Side Rendering*.



Fonte: (M. Beke, 2018)

Essa abordagem resulta em um melhor SEO em comparação ao CSR, uma vez que os motores de busca agora conseguem interpretar o conteúdo dos sites. Entretanto, considerando que o conteúdo do servidor é atualizado a cada nova requisição, essa arquitetura pode ocasionar em uma sobrecarga dos servidores, elevando drasticamente o custo de produção do sistema (Bezerra, 2024).

2.3 Static Site Generation

O *Static Site Generation* (SSG) também surge como uma proposta para otimizar o SEO. Da mesma forma que o SSR, a renderização do conteúdo das páginas é feita do lado do servidor, com uma grande diferença: essa renderização ocorre uma única vez, em tempo de *build* (construção do site), e não a cada requisição (Bezerra, 2024). Sendo assim, o custo de produção é reduzido significativamente em relação ao SSR.

Contudo, tendo em vista que a renderização ocorre uma única vez, caso os dados armazenados ou o próprio conteúdo do site mudem, o que fica salvo no servidor é o conteúdo antigo, da primeira versão. Consequentemente, o SSG torna-se uma boa opção para sites estáticos, mas uma opção desfavorável para sistemas cujo conteúdo está continuamente em mudanças. Sendo assim, o SSG deve ser recompilado sempre que houver uma mudança no conteúdo (Vepsäläinen et al., 2023).

2.4 Incremental Static Regeneration

Diante dos problemas apresentados pelas abordagens anteriores, surgiu, nos últimos anos, uma nova arquitetura – o *Incremental Static Regeneration* (ISR) – que combina o dinamismo do SSR e o custo-benefício do SSG. Trata-se de uma abordagem recente, popularizada por frameworks como Next.js, que permite a atualização dos conteúdos das páginas no servidor com base em cache. Dessa forma, o conteúdo do servidor é atualizado recorrentemente – uma vez por semana, por exemplo – e não há a necessidade de recompilar o servidor a cada mudança no conteúdo do site (Vepsäläinen et al., 2023).

2.5 Renderização Híbrida

Considerando cada uma das abordagens anteriores, é comum que grandes aplicações e sistemas complexos tenham diferentes tipos de páginas, sendo que cada página pode ter uma necessidade arquitetural diferente. Um mesmo site pode ter páginas estáticas, que demandem o SSG, enquanto outras páginas podem requerer o dinamismo do SSR ou do ISR. Nesses casos, é possível aplicar a renderização híbrida (Vepsäläinen et al., 2023).

3 TRABALHOS RELACIONADOS

Sem entrar em detalhes técnicos sobre renderização, Paiva (2018) ressalta a relevância do SEO não apenas para o sucesso de um negócio, como também para uma melhor experiência dos clientes.

Além disso, diversas pesquisas comparam as diferentes arquiteturas de renderização. Entre elas, a pesquisa de Beke (2018) oferece um estudo completo sobre as diferenças entre o CSR e o SSR em relação a atributos como performance, escalabilidade, custo de desenvolvimento e segurança, tendo como base métricas mensuráveis, metodologias e resultados.

De maneira complementar, Bezerra (2024) também compara as diferentes arquiteturas e contribui significativamente ao trazer novas perspectivas em relação aos efeitos da escolha da abordagem de renderização para o SEO, também com base em métricas e resultados.

O ISR consiste em uma arquitetura extremamente recente, portanto há poucas pesquisas que demonstram a sua real eficiência. Nesse sentido, Vepsäläinen et al. (2023) comparam o SSR e o SSG tradicional com o ISR e comprovam o quanto essa nova arquitetura é promissora, visto que combina as melhores características das outras duas abordagens. Não há na literatura brasileira, até o momento, nenhum artigo que se aprofunde no ISR.

Nesse sentido, o presente trabalho contribui ao trazer uma aplicação prática que implementa e compara cada um dos padrões arquiteturais supracitados.

4 METODOLOGIA

A seção de metodologia é dividida em duas partes: a metodologia aplicada para o desenvolvimento do *e-commerce* e a metodologia utilizada para comparar o comportamento das diferentes arquiteturas de renderização.

4.1 Desenvolvimento do Sistema

O desenvolvimento do e-commerce foi dividido nas etapas:

- Levantamento de requisitos;
- Prototipação da interface;
- Modelagem do banco de dados;
- Implementação do *front-end*;
- Implementação do *back-end*;
- Implantação.

Com o intuito de agilizar o processo, foi utilizada a metodologia Kanban, que consiste em dividir as tarefas em um quadro visual, de forma que seja fácil identificar quais são as tarefas pendentes, quais estão em progresso, quais já foram concluídas e também o nível de prioridade de cada tarefa. Sendo assim, através da plataforma de gerenciamento de projetos ClickUp[1], foi criado um quadro Kanban para cada etapa do desenvolvimento, o que permitiu organizar e acompanhar cada atividade. Além disso, foi utilizado o Google Sheets para documentar o que foi feito no decorrer dos dias e quanto tempo em média levou cada atividade.

4.2 Comparação das Arquiteturas

Diferentemente das pesquisas mencionadas, a comparação das arquiteturas não foi feita com base na análise de métricas, mas sim em comportamentos, características e efeitos visíveis de cada abordagem.

[1] Disponível em: https://clickup.com. Acesso em: 11 dez. 2025.

4.3 Tecnologias

As ferramentas utilizadas para planejamento e organização foram o Clickup e o Google Sheets. Em relação à concepção do projeto em si, a iniciar pela etapa de prototipação, a principal ferramenta utilizada foi o Figma[2], que consiste em uma plataforma de design com foco na criação da interface (UI) e na experiência do usuário (UX).

Sobre o ambiente de desenvolvimento, foi utilizada a IDE VSCode para programação do sistema, tanto do *front-end* quanto do *back-end*. Além disso, foi utilizado o Github para versionamento de código, o que garante maior segurança ao desenvolvimento e previne contra possíveis perdas.

Para a construção da interface no *front-end*, foi utilizado o Nuxt, *framework* do Vue que, além da reatividade, componentização e várias outras funcionalidades padrões do Vue, permite ainda o uso de diferentes abordagens de renderização, o que garante maior desempenho e flexibilidade. Além do Nuxt e do JavaScript para programação, foi usado o SASS, um pré-processador de CSS que facilita na estilização da interface. Já no *back-end*, foi utilizado o Django REST framework para a construção da API do *e-commerce*.

A arquitetura aplicada na construção do sistema foi a arquitetura distribuída com base em microsserviços. Foram utilizadas APIs externas para a integração com sistemas de terceiros – como a API do Mercado Pago[3] para integração com sistema de pagamentos e a API do Melhor Envio[4] para geração de etiquetas de transporte de pacotes.

Quanto à hospedagem do sistema, utilizou-se um servidor na plataforma Heroku[5] para disponibilização do banco de dados PostgreSQL e da API. Já para a hospedagem do *front-end*, foi utilizado um segundo servidor no Netlify[6], o qual faz as requisições à API e disponibiliza no site todas as informações do *e-commerce*.

---

[2] Disponível em: https://figma.com. Acesso em: 11 dez. 2025.
[3] Disponível em: https://www.mercadopago.com.br/developers/pt. Acesso em: 11 dez. 2025.
[4] Disponível em: https://docs.melhorenvio.com.br/docs/introducao-a-api. Acesso em: 11 dez. 2025.
[5] Disponível em: https://heroku.com. Acesso em: 11 dez. 2025.
[6] Disponível em: https://netlify.com. Acesso em: 11 dez. 2025.

5 DESENVOLVIMENTO

O presente capítulo é dividido em duas partes principais – uma relativa ao desenvolvimento do *e-commerce* e outra relativa à análise do comportamento das diferentes arquiteturas.

5.1 Desenvolvimento do Sistema

O sistema apresenta dois tipos de atores principais: o cliente, que executa as compras no site, e o administrador, que gerencia produtos, estoque e pedidos. Portanto, o site possui duas interfaces, uma para o cliente e outra para o administrador.

5.1.1 Requisitos Funcionais

Os requisitos funcionais descrevem as ações do *software*, com foco nas funcionalidades e nas interações dos usuários com o sistema. Quanto às ações executadas pelos atores, os clientes devem poder:

- Navegar pelo site e visualizar as coleções e produtos da loja;
- Visualizar produtos em destaque e/ou mais vendidos;
- Buscar por produtos específicos com base no nome ou código;
- Cadastrar, fazer *login* e *logout* no sistema;
- Recuperar senha caso tenha esquecido;
- Adicionar, remover, alterar quantidade ou visualizar produtos no carrinho;
- Adicionar, remover ou selecionar endereços de entrega;
- Efetuar compra de produtos por meio dos métodos de pagamento Pix ou cartão de crédito;
- Adicionar, remover ou visualizar produtos favoritos;
- Checar ou editar dados pessoais;
- Checar pedidos anteriores e detalhes como status de entrega;
- Encontrar informações sobre a empresa, tais como endereços de contato, políticas, termos e ajudas com dúvidas;

Além disso, não deve ser possível cadastrar ou fazer compra caso seja menor de idade.

Já os administradores devem poder:

- Fazer *login* e *logout* no sistema de administração;
- Ter acesso a um *dashboard* com estatísticas de vendas e informações relevantes como quantidade de pedidos pendentes;
- Visualizar pedidos de clientes, bem como alterar o status dos pedidos e gerar etiquetas de transporte;
- Adicionar, remover, editar ou visualizar catálogo de coleções;
- Adicionar, remover, editar ou visualizar produtos de cada coleção;
- Adicionar, remover, editar ou visualizar descontos associados a um produto;
- Organizar ordem de relevância das coleções;
- Gerenciar estoque de cada produto;
- Gerenciar produtos em destaque;
- Gerenciar produtos favoritos dos clientes.

5.1.2 Requisitos não Funcionais

Enquanto os requisitos funcionais focam nos comportamentos e interações do sistema, os requisitos não funcionais descrevem as qualidades e atributos do *e-commerce*. São eles:

- Desempenho e velocidade;
- Boa performance SEO;
- Interfaces intuitivas que aplicam práticas do UI/UX;
- Segurança em relação aos pagamentos e dados dos clientes;
- Manutenibilidade;
- Escalabilidade.

5.1.3 Prototipação

A prototipação consiste em transformar os requisitos em um modelo próximo ao que seria o sistema final, a fim de testar e validar as ideias antes de executar o desenvolvimento.

5.1.3.1 Idealização das Páginas

Antes da execução do *wireframe*, por se tratar de um sistema complexo, foi necessário idealizar quais seriam as páginas do site. Na interface do cliente, as páginas idealizadas foram:

Tabela 1 – Descrição das páginas do lado do cliente.

Página	Descrição
Página inicial	Apresentação geral da loja e dos produtos
Sobre	Página dedicada a contar a história da empresa
Catálogo de coleções	Um painel contendo várias coleções, onde é possível ordenar por nome, data de lançamento ou ordem de relevância
Página da coleção	Painel contendo os produtos de cada coleção
Produto específico	Um álbum de imagens do produto e detalhes como nome, preço descrição e botão de adicionar ao carrinho
Login	Formulário com email e senha
Cadastro	Formulário onde o cliente informa dados pessoais para cadastro
Carrinho	Uma lista dos produtos de interesse do cliente, com subtotal e botão de finalizar compra
Checkout	Finalização de um pedido, onde são colocados detalhes de entrega e é efetuado o pagamento
Perfil do usuário	Seção do site em que o cliente pode conferir dados pessoais, endereços, pedidos e produtos favoritos
Termos e condições	Termos da empresa
Contato	Detalhes e endereços de contato
Central de ajuda	Parte dedicada a tirar dúvidas

Fonte: O autor, 2025.

Já na interface do administrador, as páginas idealizadas foram:

Tabela 2 – Descrição das páginas do lado do administrador.

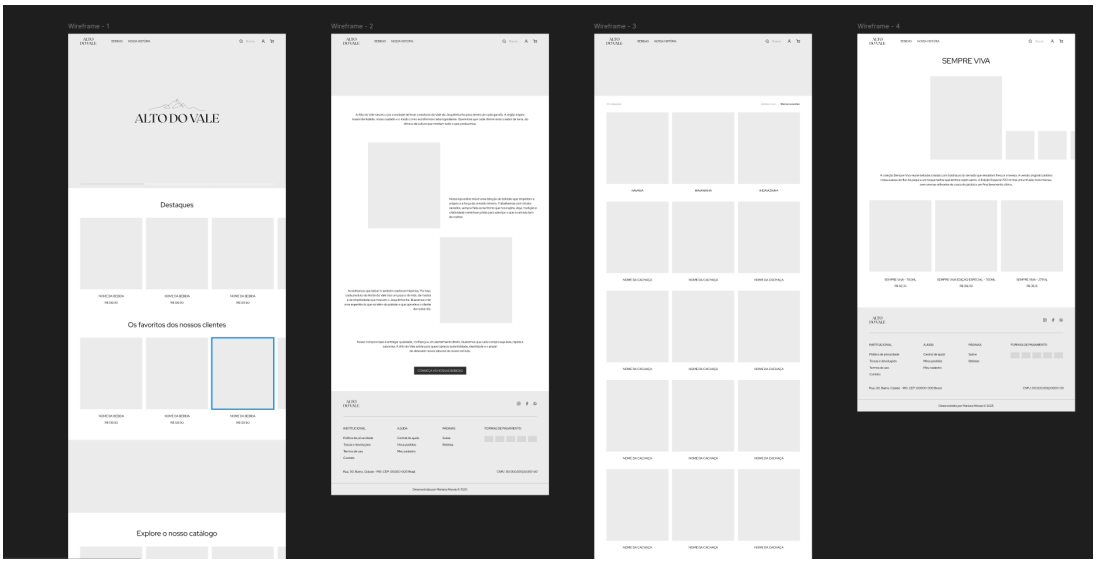
Página	Descrição
Início	*Link* para diferentes seções da administração
Dashboard	Página com estatísticas de vendas e detalhes relevantes
Pedidos	Verificação de pedidos dos clientes
Produtos	Seção onde é possível gerenciar catálogo, estoque, produtos em destaque e favoritos dos clientes
Login	Formulário com email e senha

Fonte: O autor, 2025.

5.1.3.2 Wireframe e Experiência do Usuário

Após a idealização, a próxima etapa foi a criação de um *wireframe*, que pode ser observado na Figura 3. Um *wireframe* consiste em um rascunho da estrutura de textos e imagens de forma a facilitar na criação do protótipo final. Tudo isso foi feito com base na interface de outros *e-commerces* e em convenções comumente utilizadas para a experiência do usuário (UX). Por exemplo, é uma convenção que, ao acessar um produto específico, as imagens do produto estejam à esquerda, enquanto detalhes como nome e preço ficam à direita. Outra convenção comum é que o botão de carrinho esteja acessível no menu, normalmente no canto superior direito. Dessa forma, garante-se que, ao acessar o site, o visitante encontre *layouts* aos quais já está familiarizado, o que torna a plataforma mais intuitiva.

Figura 3 – *Wireframe*.

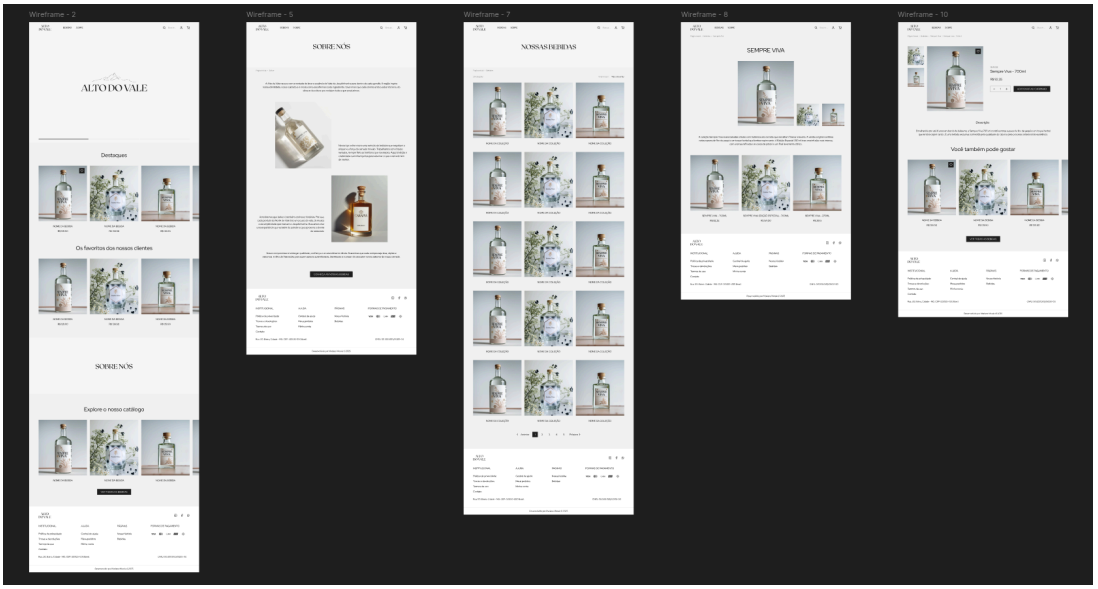


Fonte: O autor, 2025.

5.1.3.3 Protótipo Final

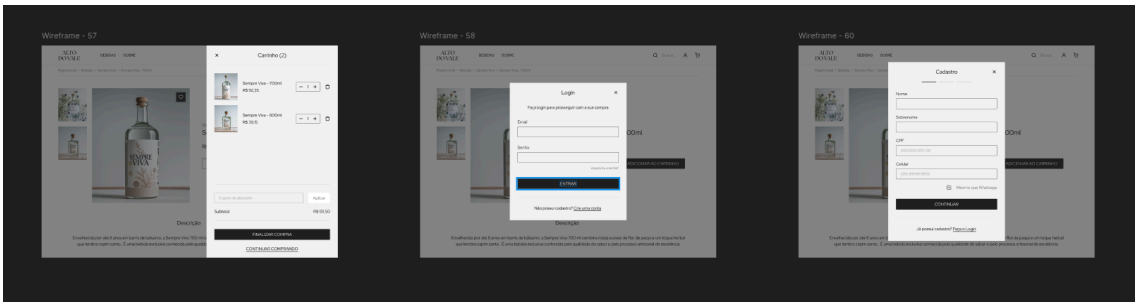
Tendo o *wireframe* pronto, para finalizar o protótipo foi necessário apenas adicionar as cores e imagens. Foram criadas cerca de 35 páginas de protótipo para a interface do cliente e 16 páginas para a interface do administrador, abrangendo todas as funcionalidades do *e-commerce*. Algumas dessas páginas podem ser visualizadas nas Figuras 4 e 5.

Figura 4 – Protótipo das páginas de produtos e sobre.



Fonte: O autor, 2025.

Figura 5 – Protótipo de carrinho, login e cadastro.

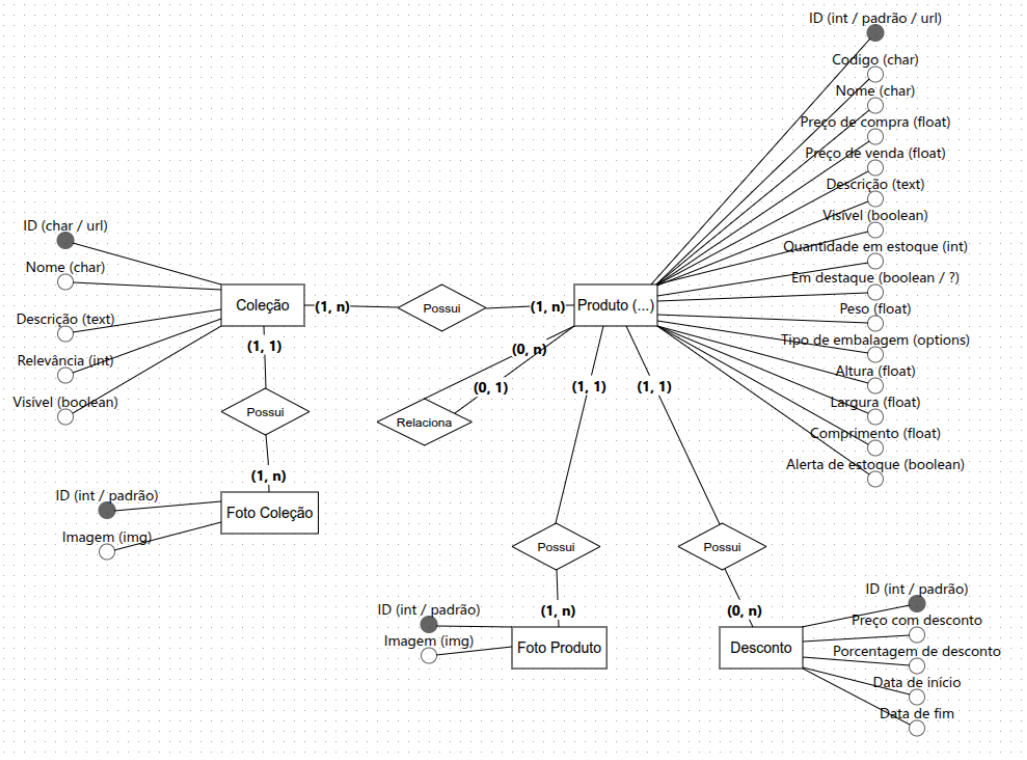


Fonte: O autor, 2025.

5.1.4 Modelagem

Foi realizada a modelagem conceitual do sistema utilizando a ferramenta brModelo Web[7], como pode-se ver nas Figuras 6 e 7. Algumas partes do modelo foram posteriormente adaptadas para melhor implementação do sistema.

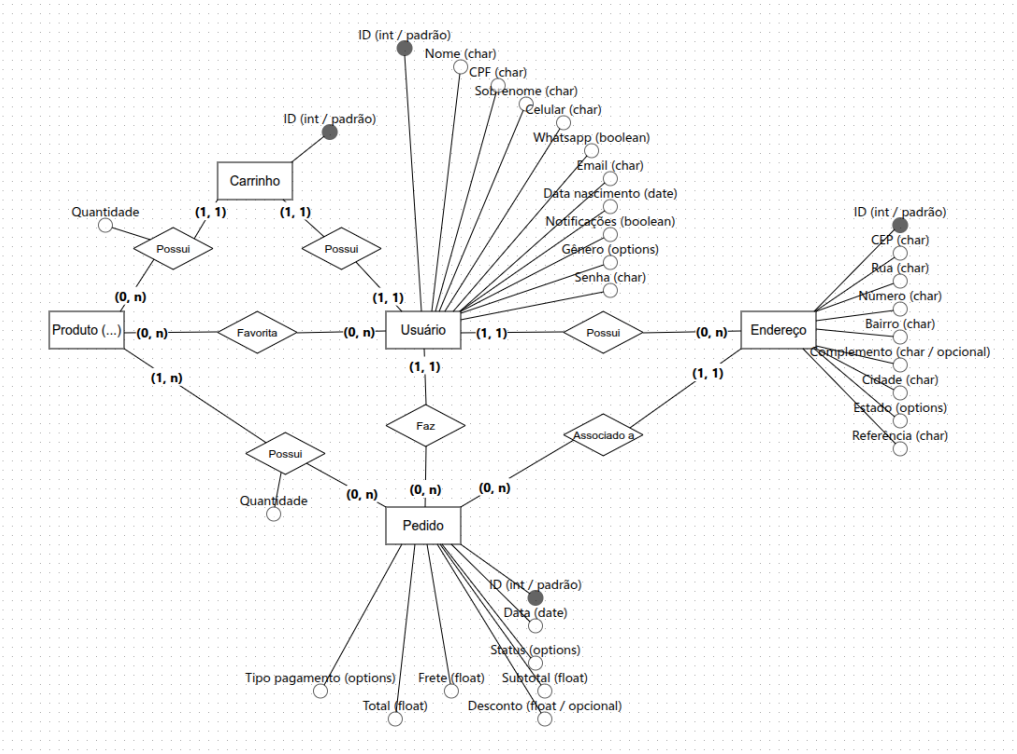
Figura 6 – Modelagem conceitual da estrutura de pedidos.



Fonte: O autor, 2025.

[7] Disponível em: https://app.brmodeloweb.com/. Acesso em: 11 dez. 2025.

Figura 7 – Modelagem conceitual da estrutura de usuários e pedidos.



Fonte: O autor, 2025.

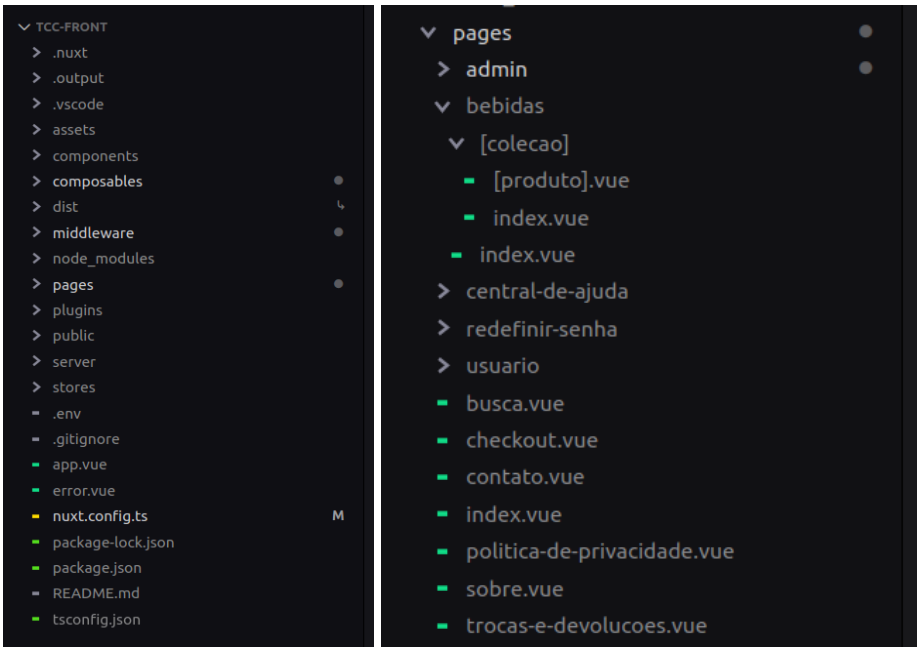
5.1.5 Implementação do Front-end

Essa seção tem como objetivo mostrar uma visão geral sobre a estrutura e funcionamento do projeto no lado do *front-end*.

A renderização padrão do Vue e do React é o CSR, não sendo possível implementar, apenas com eles, as demais arquiteturas. Por essa razão, a escolha da ferramenta de desenvolvimento do *front-end* foi pensada nessas limitações e na necessidade de se comparar as diferentes arquiteturas. O Nuxt 3 foi escolhido, portanto, como ferramenta principal.

O Nuxt é um *framework* opinativo. Isso significa que ele traz automaticamente convenções, estruturas de pastas e de arquivos que facilitam tanto o desenvolvimento quanto as manutenções. Essa estrutura pode ser observada na Figura 8.

Figura 8 – Estrutura de arquivos e diretórios do Nuxt.



Fonte: O autor, 2025.

Todas as páginas do site são localizadas no diretório *pages*, e são escritas no formato de arquivo *.vue*. Rotas dinâmicas – aquelas que consistem em um *template* cujo conteúdo é carregado via API – têm nome escrito entre colchetes. Um exemplo de rota dinâmica é a página de produto, que consiste em um único *template* para todos os produtos da loja.

Além do diretório *pages*, outras partes importantes de um projeto Nuxt são os diretórios *components* e *composables*. O diretório *components* permite criar trechos de *templates* que podem ser reutilizados em diferentes partes do site, como, por exemplo, os componentes de menu e rodapé. E também, o diretório *composables* permite a reutilização de trechos de códigos JavaScript em diferentes partes do sistema. Sendo assim, ambos os diretórios evitam o problema de duplicação de código, que dificulta o processo de manutenção, e permitem aplicar a prática *Don't Repeat Yourself* (DRY).

Quanto à estrutura dos arquivos, um arquivo do tipo *.vue* é composto por 3 partes:

- Template: Consiste em uma estrutura semelhante ao HTML. Contudo, nele é possível executar lógicas de programação como iterações e condicionais, trazendo interatividade e reatividade às páginas do site.

- Script: Nessa seção, é possível escrever código JavaScript, interagir com o HTML e, inclusive, fazer requisições a APIs.
- Style: Seção dedicada à estilização da página, sendo possível aplicar CSS ou SASS.

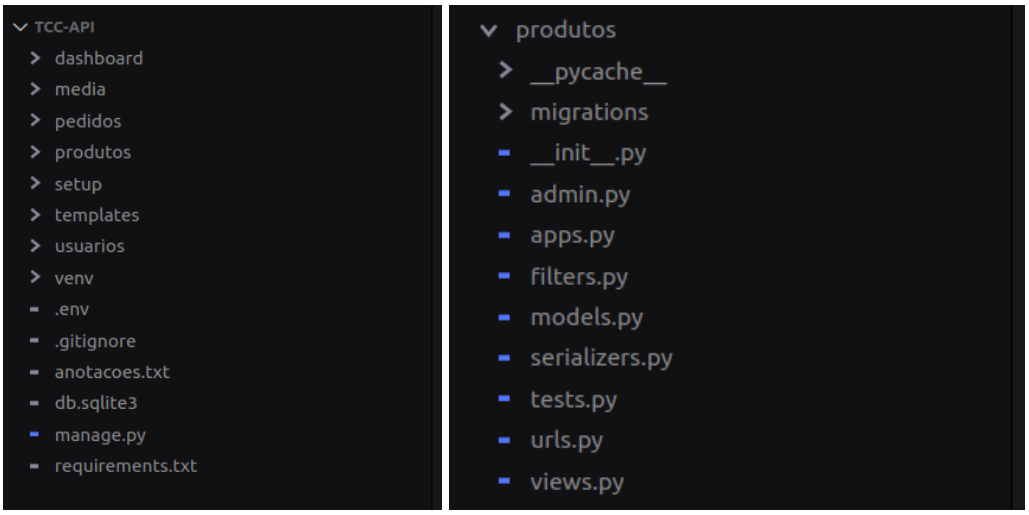
No sistema de *e-commerce*, o *front-end* interage, de diversas formas, com o servidor do *back-end* através de requisições do tipo GET, POST, PUT, PATCH e DELETE, utilizando a biblioteca Axios. Outros fatores relevantes para o sistema são a automatização de autenticação, implementada através de *tokens* JWT, e o armazenamento de informações em *Cookies* e *LocalStorage*. Além dos *tokens* de sessão, são armazenados localmente detalhes de produtos de carrinho para usuários anônimos e a informação de maioridade do cliente.

Outro arquivo de extrema relevância é o *nuxt.config.ts*, inserido na raiz do projeto. Ele contém inúmeras configurações relativas à aplicação e, inclusive, é nele em que se define a abordagem de renderização, sendo possível escolher o CSR, o SSR, o SSG, o ISR ou a arquitetura híbrida. O padrão aplicado pelo Nuxt é o SSR.

5.1.6 Implementação do Back-end

No *back-end*, a principal ferramenta foi o Django REST framework, utilizado para a criação da API em Python. Esse *framework* também é opinativo e implica em estruturas de pastas pré-definidas, como pode-se ver na Figura 9.

Figura 9 – Estrutura de arquivos e diretórios do Django REST framework.



Fonte: O autor, 2025.

O *framework* permite criar diferentes *apps*, ou partes do sistema, o que garante maior isolamento e baixo acoplamento de código. Cada *app* possui seus próprios *models*, *serializers*, *views* e outros códigos. Ou seja, pode-se dizer que cada *app* atua de forma independente mas que, juntos, compõem todo o sistema. Os *apps* criados para o projeto foram: *dashboard*, pedidos, produtos e usuários. Sendo assim, a lógica de cada *app* permanece isolada.

Cada arquivo dentro do *app* é responsável por alguma ação. O arquivo *models* é um dos mais importantes, pois nele são escritas as classes que serão convertidas em tabelas do banco de dados, através da ORM do Django. Uma ORM consiste em um sistema que traduz código orientado a objetos em SQL, permitindo interagir com dados e esquemas usando linguagens como Python ao invés de SQL.

Outros arquivos relevantes no ecossistema do Django Rest framework são o *serializers* e o *views*. O arquivo *serializers* permite serializar e desserializar dados, complexos, isto é, converter objetos do banco de dados em estruturas do formato JSON e vice-versa, a fim de estabelecer comunicação com o *front-end*. Enquanto isso, o *views* determina as ações a serem executadas ao receber requisições HTTP (GET, POST, PUT, PATCH e DELETE).

O arquivo *settings* define variáveis globais de configuração do sistema, e o arquivo *.env*, que permite a definição de chaves secretas, tokens, e outras variáveis que não podem ser expostas, mesmo para sistemas de versionamento de repositórios como o Github, por razões de segurança.

O projeto da Alto do Vale exigiu a comunicação com diferentes serviços externos, e essa comunicação foi feita pelo *back-end*, também por razões de segurança e para ocultar a complexidade da camada com a qual o usuário interage. Foi necessário, por exemplo, no *app* pedidos, fazer a comunicação com o serviço de Checkout Transparente, do Mercado Pago, para garantir que as transações financeiras fossem realizadas antes de efetuar a confirmação de um pedido. Também, foi feita a comunicação com o serviço do Melhor Envio para geração de etiquetas de transporte. Ambas as integrações foram feitas com os ambientes de Sandbox das respectivas empresas. Isto é, ambientes de testes seguros e isolados que simulam interações reais.

5.1.7 Implantação

Parte dos testes de renderização não precisam ser realizados em ambiente de produção, pois os resultados são visíveis mesmo em ambiente de desenvolvimento ou em máquinas locais. Contudo, ainda assim, o sistema foi mantido em produção durante 30 dias para se analisar os efeitos de algumas arquiteturas específicas. Os comportamentos de cada arquitetura serão analisados no tópico 5.2.

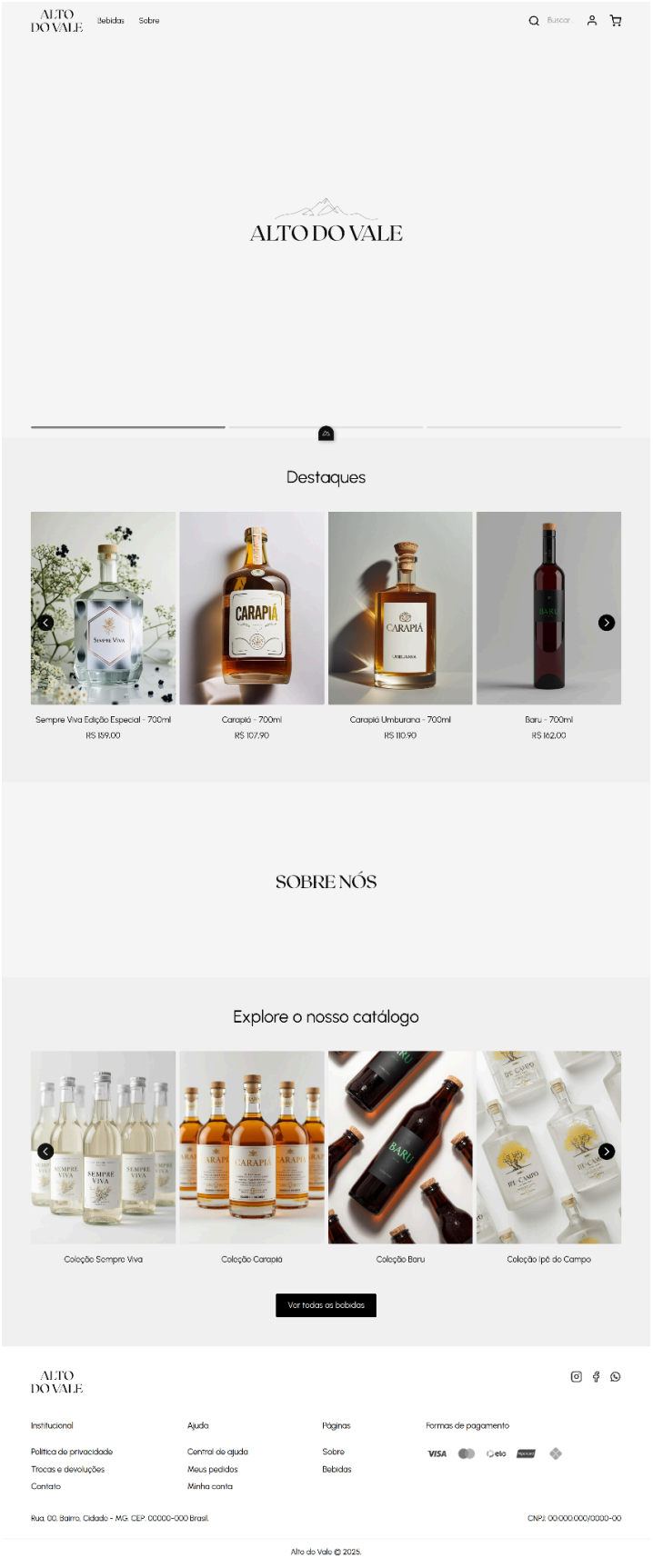
A implantação da API foi realizada no Heroku, utilizando o banco de dados PostgreSQL, enquanto o *front-end* foi hospedado na Netlify, que permite a hospedagem gratuita desde que não ultrapasse os limites da plataforma.

5.1.8 Apresentação do Sistema

Uma vez tendo sido finalizadas a prototipação e a implementação do sistema, é possível acessá-lo como cliente ou administrador. O presente tópico apresenta uma visão geral sobre a interface final e as possíveis interações que o usuário pode executar.

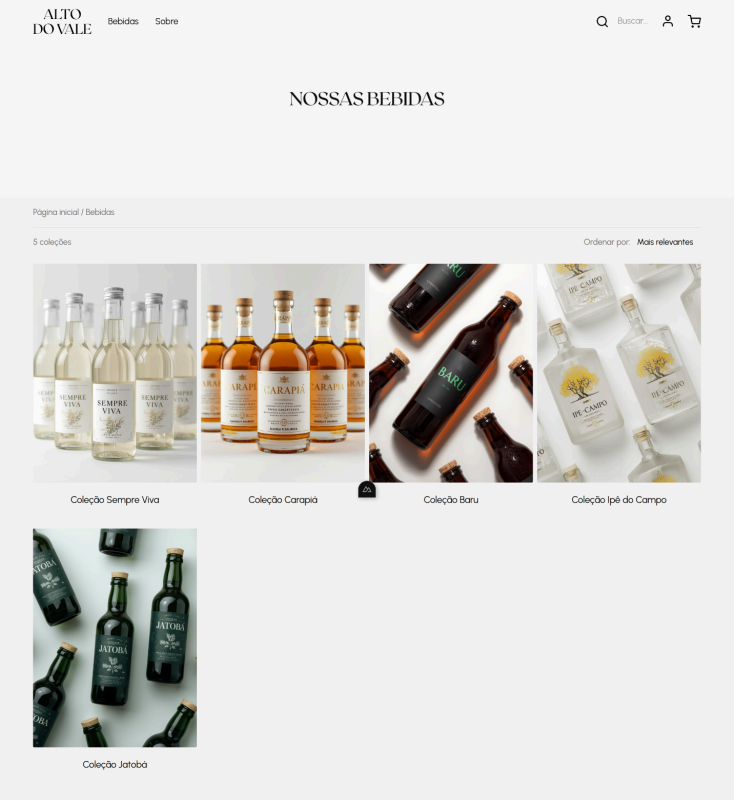
Na interface do cliente, o usuário pode navegar pelas coleções e produtos da loja de forma a encontrar os seus itens de interesse, conforme ilustram as Figuras 10, 11, 12 e 13.

Figura 10 – Página inicial do *e-commerce*.



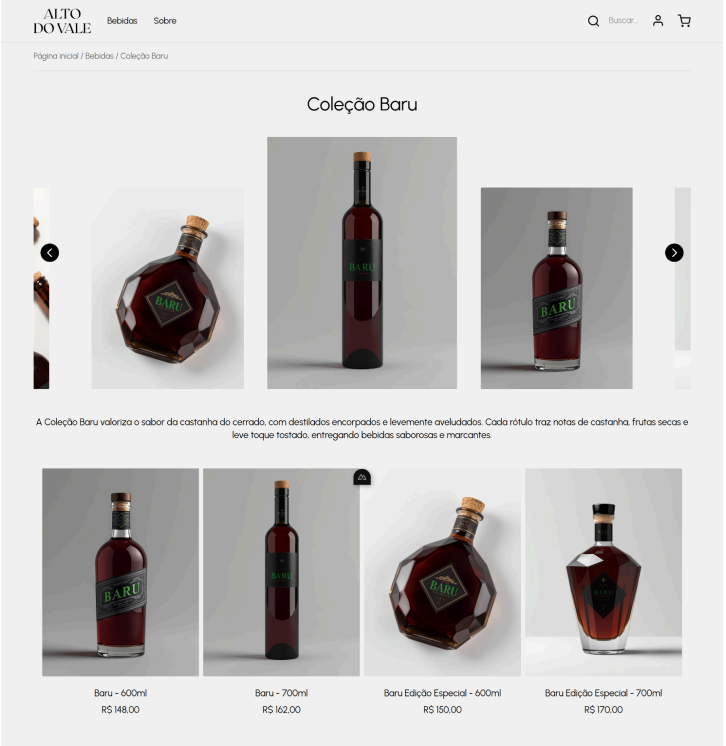
Fonte: O autor, 2025.

Figura 11 – Página de coleções.



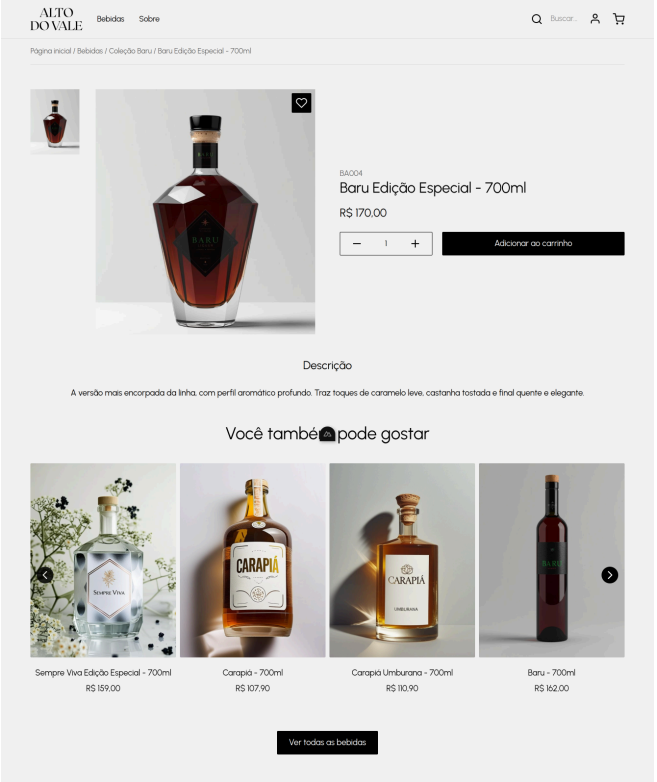
Fonte: O autor, 2025.

Figura 12 – Página de uma coleção específica.



Fonte: O autor, 2025.

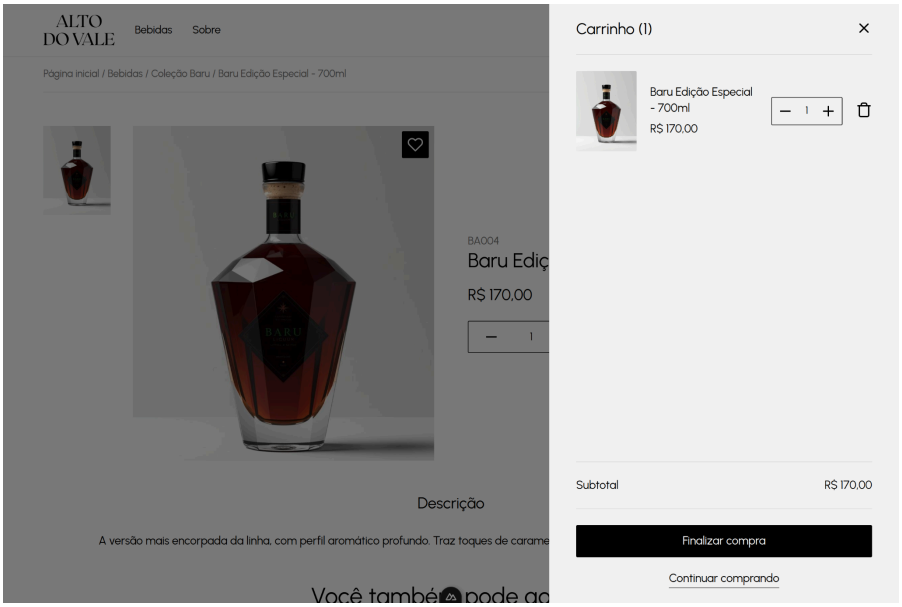
Figura 13 – Página de um produto específico.



Fonte: O autor, 2025.

O cliente pode visualizar, adicionar, remover e alterar quantidade de produtos no carrinho, como na Figura 14, estando autenticado ou não.

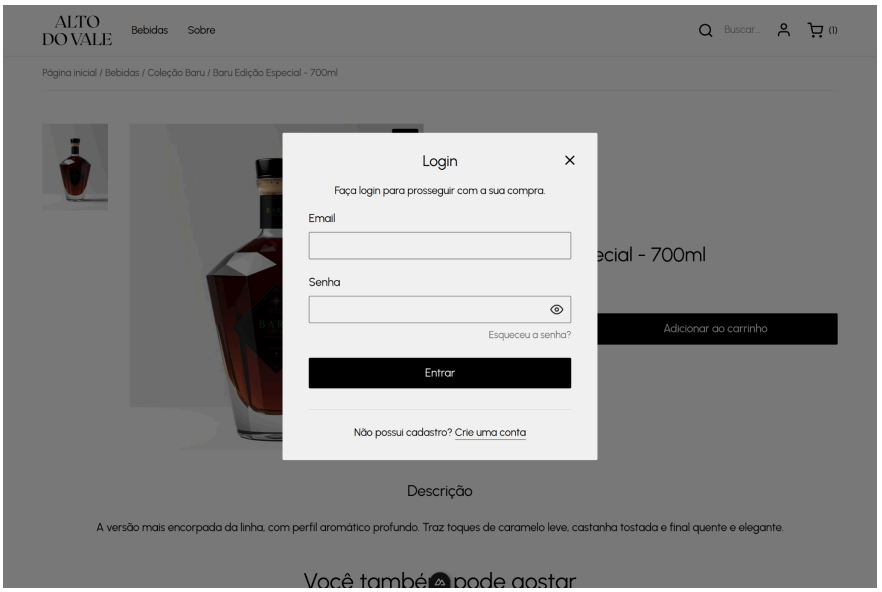
Figura 14 – Carrinho de produtos.



Fonte: O autor, 2025.

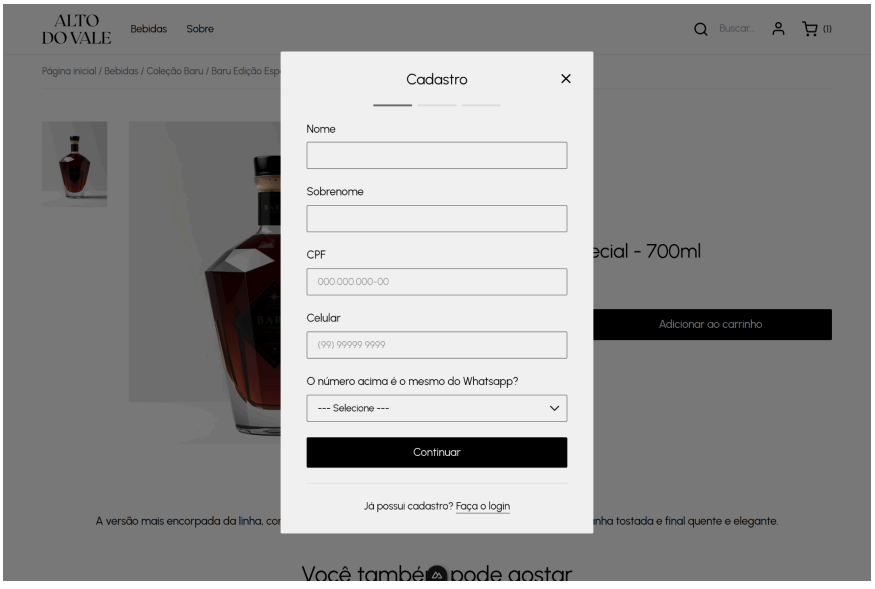
Para finalizar uma compra, o cliente deve estar cadastrado e autenticado, conforme ilustrado nas Figuras 15 e 16.

Figura 15 – Login.



Fonte: O autor, 2025.

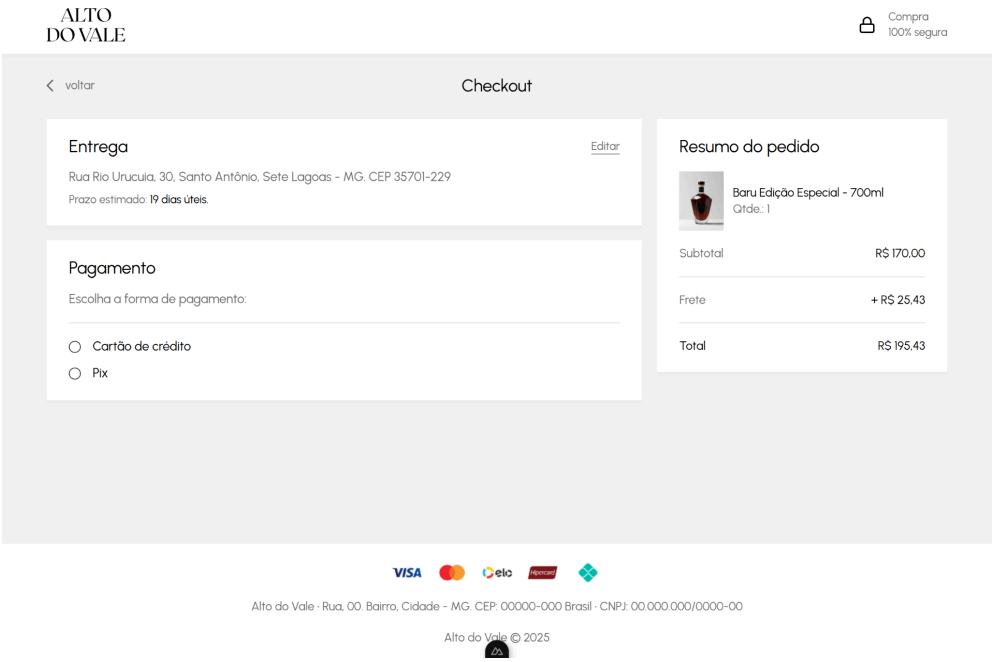
Figura 16 – Cadastro.



Fonte: O autor, 2025.

Para finalizar uma compra, o cliente deve inserir o seu endereço e selecionar a forma de pagamento no *checkout* (Figura 17). O frete e o prazo estimado são calculados automaticamente no *back-end* utilizando a API do Melhor Envio.

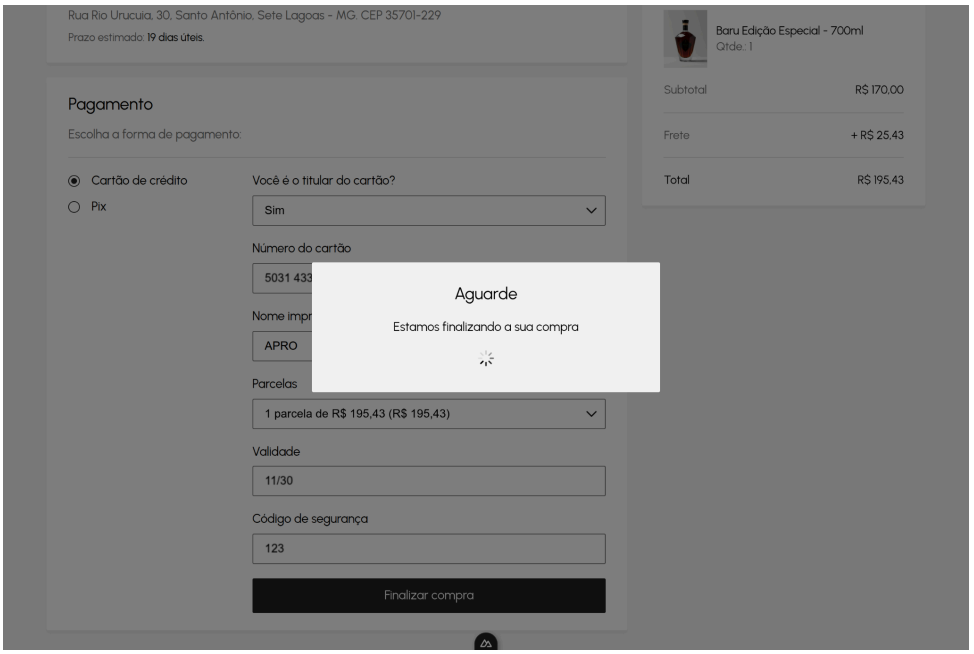
Figura 17 – Checkout.



Fonte: O autor, 2025.

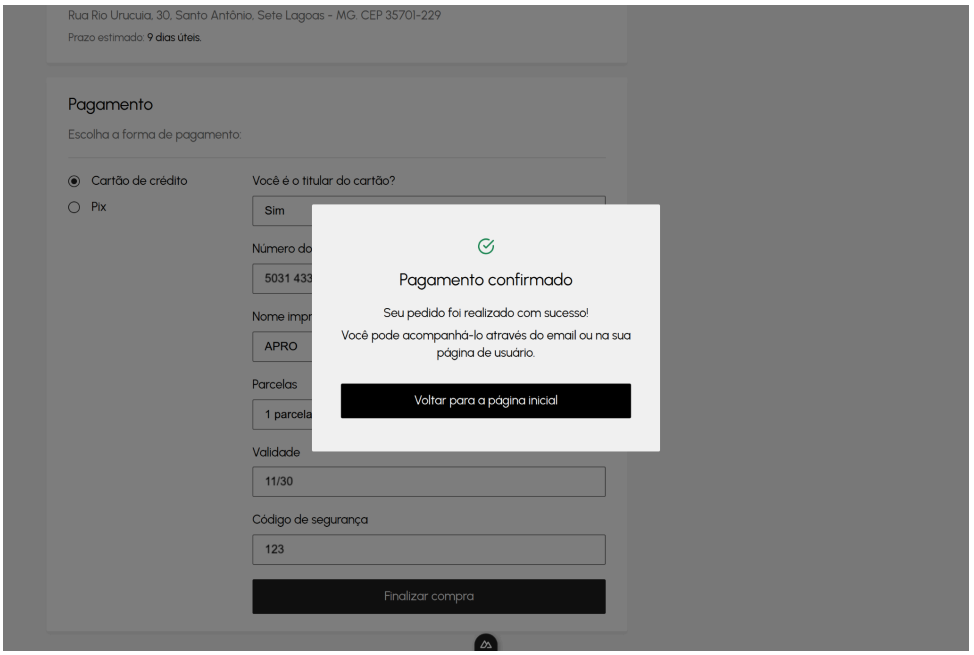
Caso a opção escolhida seja cartão de crédito, basta preencher o formulário e finalizar o pagamento. No caso das Figuras 18 e 19, foi utilizado o cartão de teste do Mercado Pago para simular uma transação verdadeira.

Figura 18 – Finalizando compra com cartão.



Fonte: O autor, 2025.

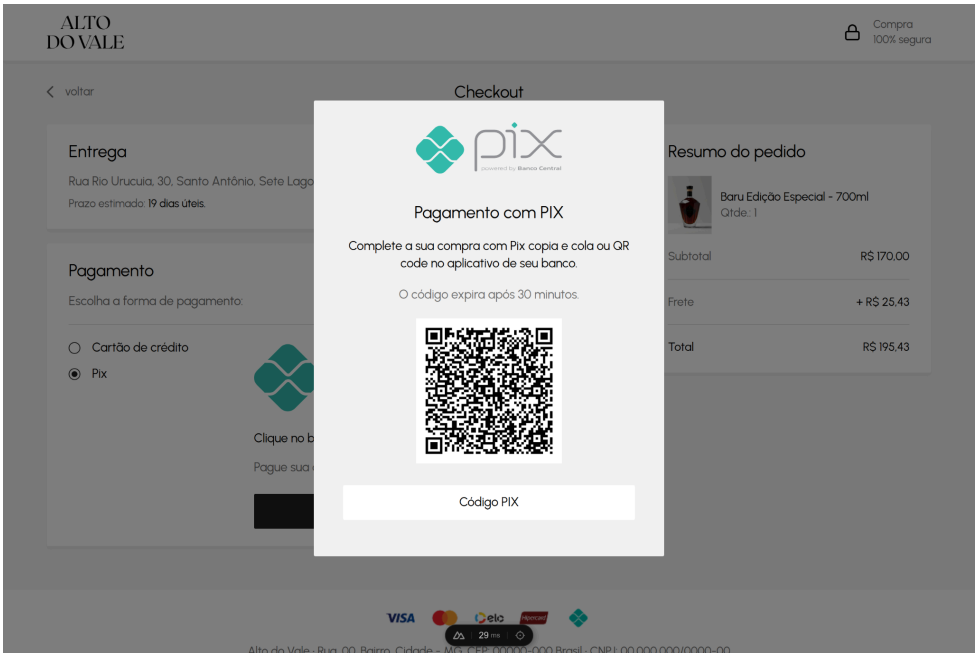
Figura 19 – Compra concluída com cartão.



Fonte: O autor, 2025.

Caso a opção escolhida seja Pix, basta copiar o QR Code ou código para finalizar a compra (Figura 20).

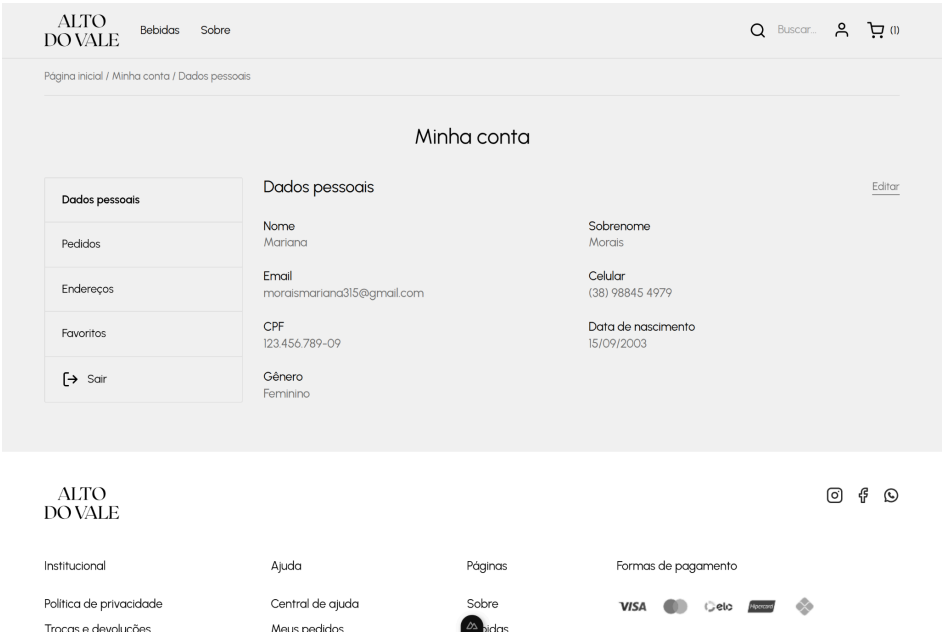
Figura 20 – Compra com Pix.



Fonte: O autor, 2025.

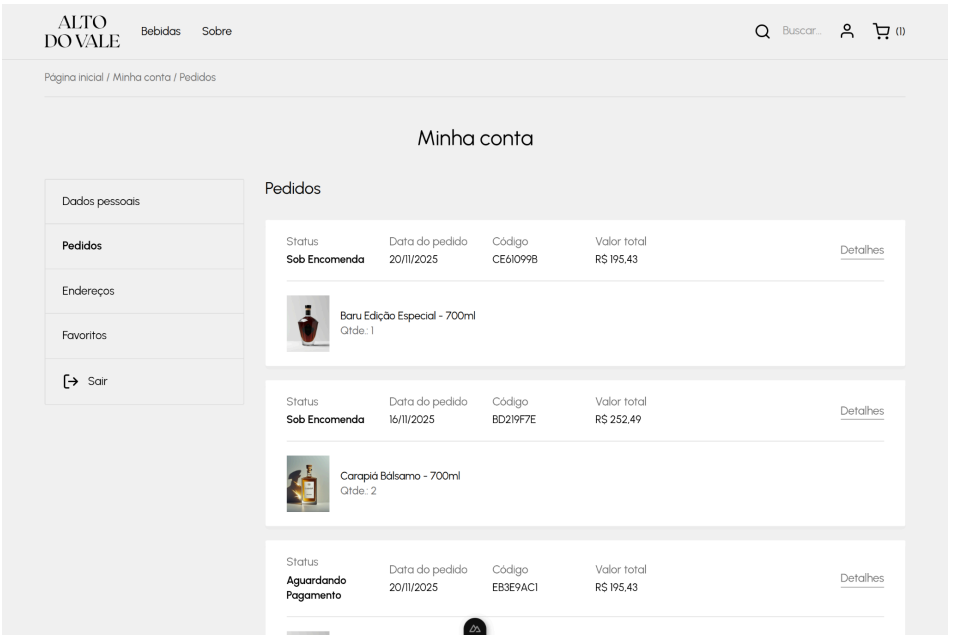
O cliente ainda consegue gerenciar detalhes da conta, visualizar pedidos anteriores, gerenciar endereços e produtos favoritos, como pode-se ver nas Figuras 21, 22, 23 e 24.

Figura 21 – Dados pessoais.



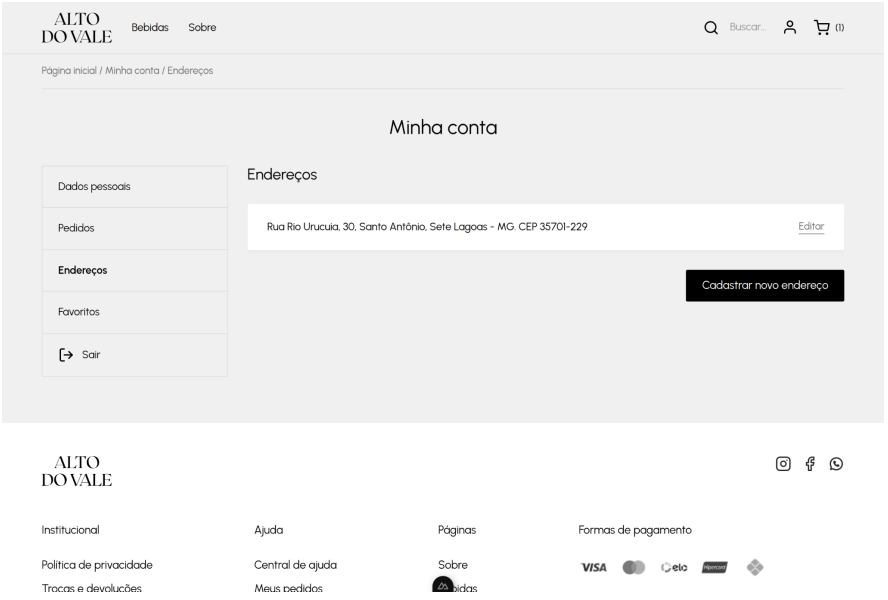
Fonte: O autor, 2025.

Figura 22 – Pedidos.



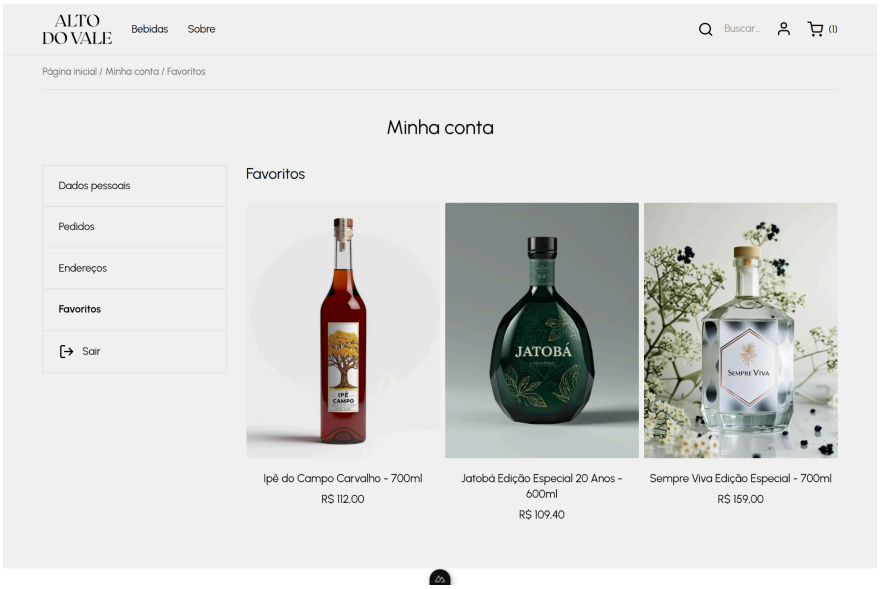
Fonte: O autor, 2025.

Figura 23 – Endereços.



Fonte: O autor, 2025.

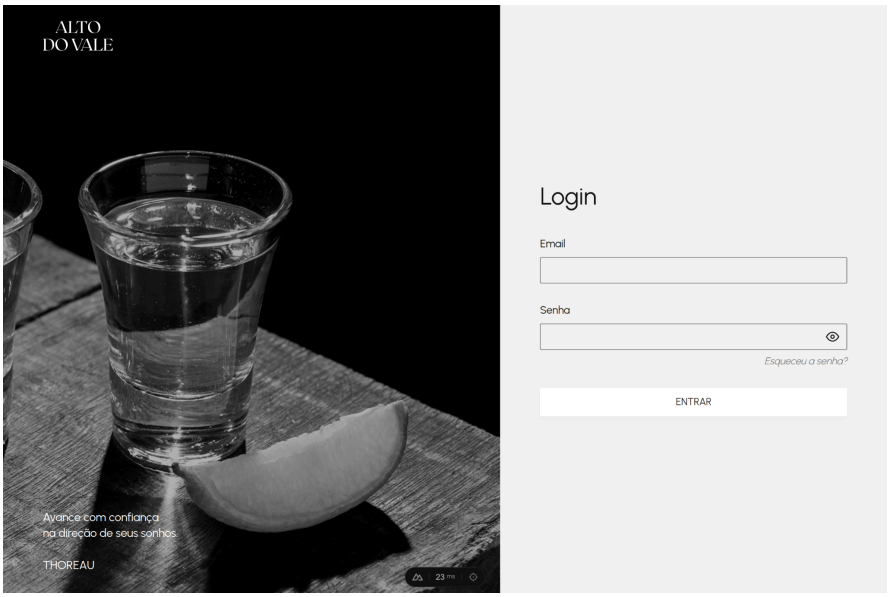
Figura 24 – Favoritos.



Fonte: O autor, 2025.

Na interface de administração, o usuário consegue efetuar outras ações, relativas à gerência do negócio. É possível fazer *login* como administrador através da página ilustrada abaixo.

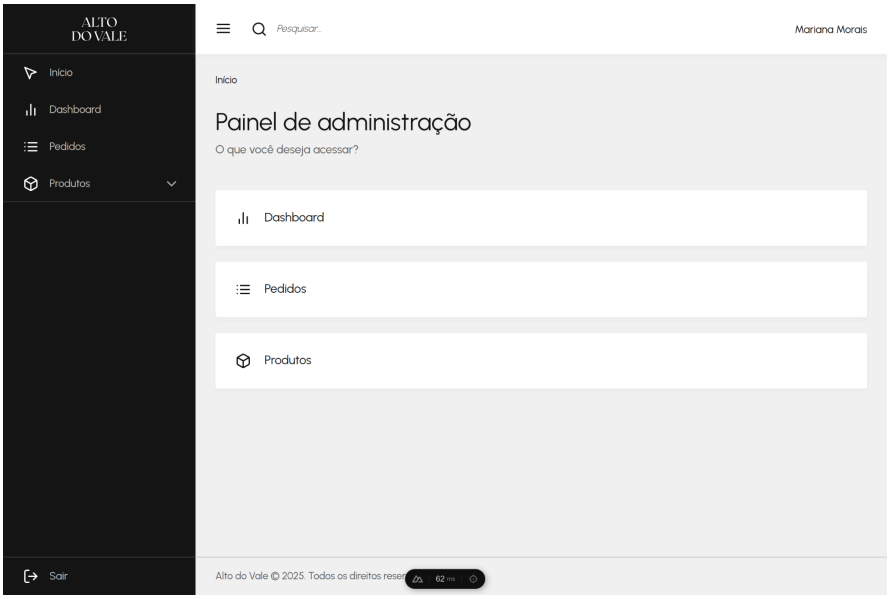
Figura 25 – Login na interface de administração.



Fonte: O autor, 2025.

A interface de administração é dividida em três partes principais: *dashboard* de estatísticas de vendas; gerenciamento de pedidos e gerenciamento de produtos. A página inicial apresenta *links* que direcionam para cada uma dessas partes, como pode-se ver na figura abaixo.

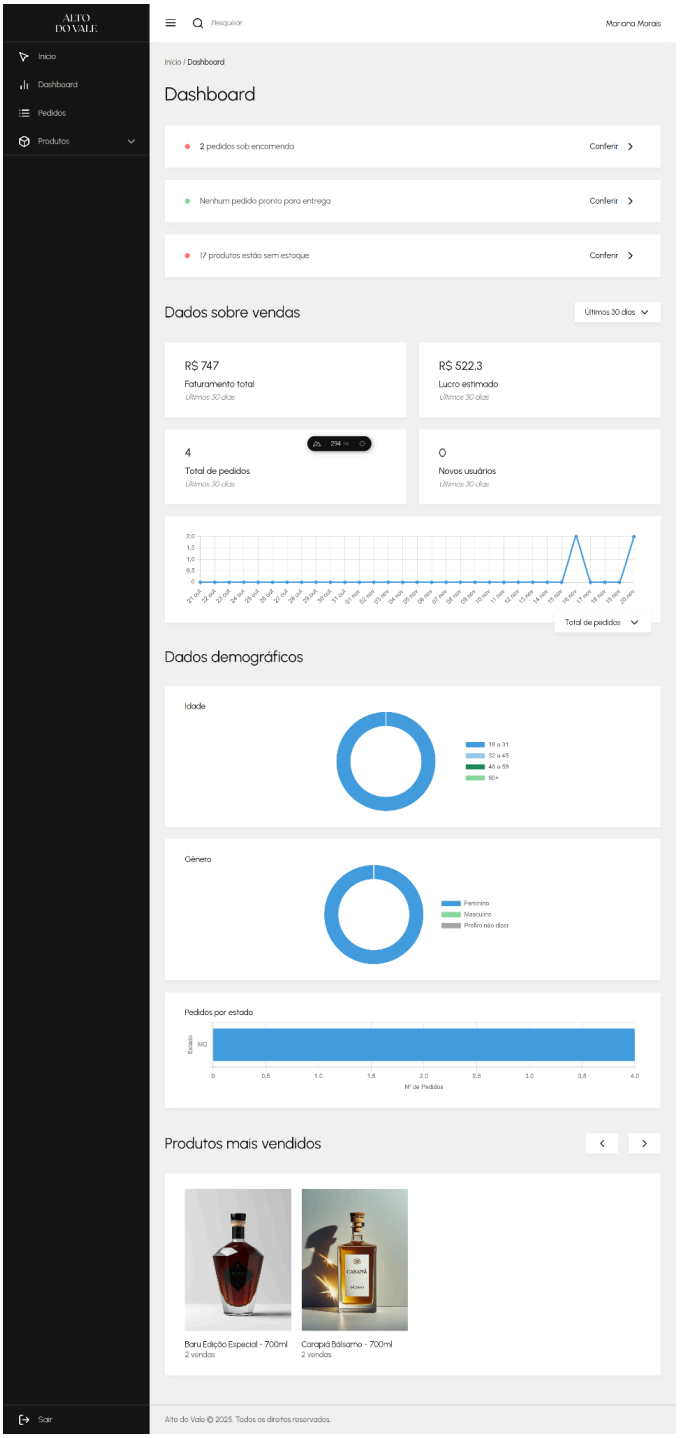
Figura 26 – Página inicial da interface de administração.



Fonte: O autor, 2025.

O *dashboard* apresenta algumas informações relevantes para a tomada de decisão dos gerentes. Nesse painel é possível visualizar detalhes de faturamento, lucro, quantidade de pedidos, alguns dados gerais dos clientes e produtos mais vendidos de um determinado período de tempo (Figura 27).

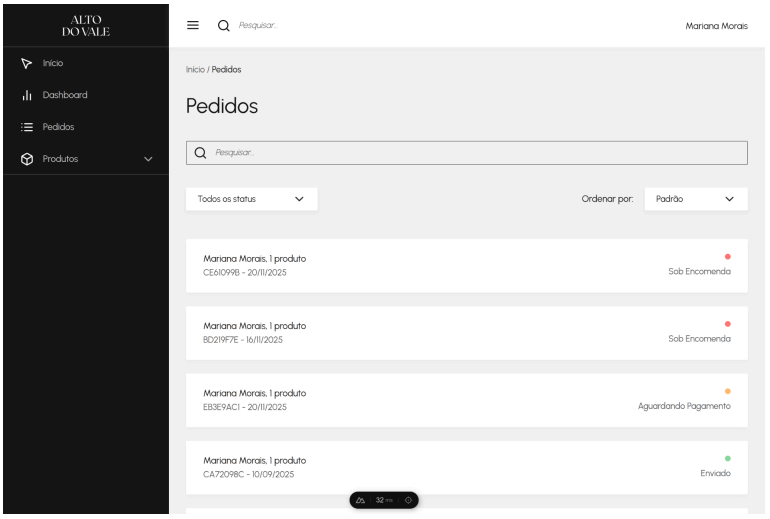
Figura 27 – *Dashboard*.



Fonte: O autor, 2025.

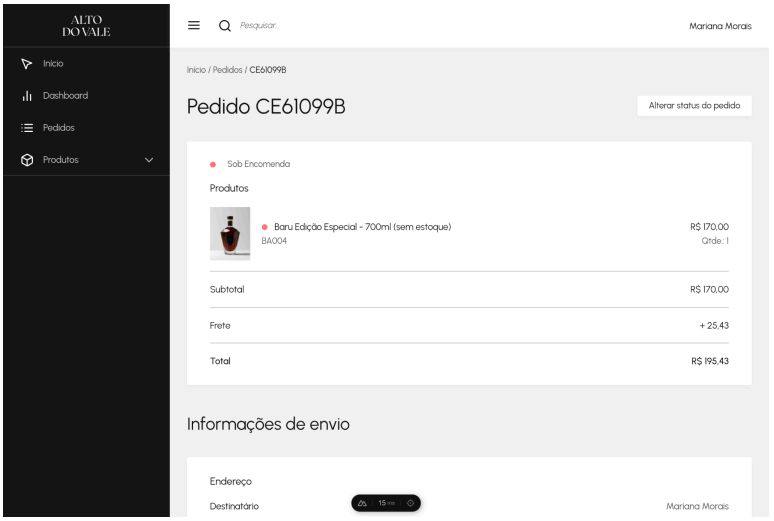
Na seção de gerenciamento de pedidos, é possível visualizar a lista de todos os pedidos, sendo possível ordenar e filtrar essa lista com base em diferentes parâmetros (Figura 28), e também é possível visualizar detalhes de cada pedido (Figura 29).

Figura 28 – Pedidos.



Fonte: O autor, 2025.

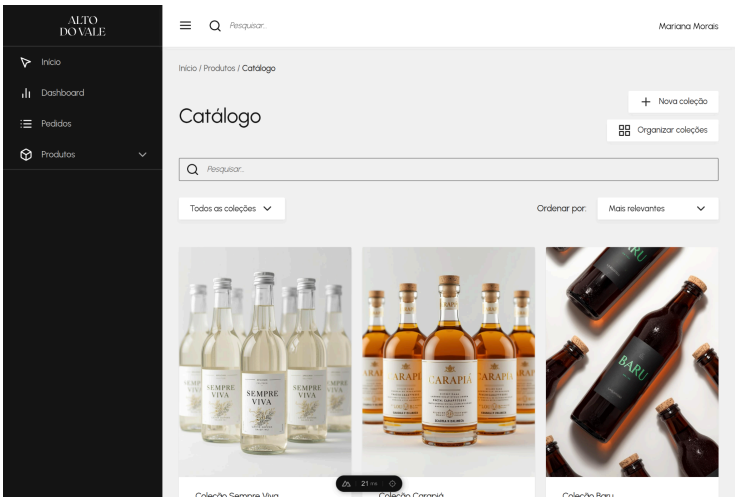
Figura 29 – Pedido específico.



Fonte: O autor, 2025.

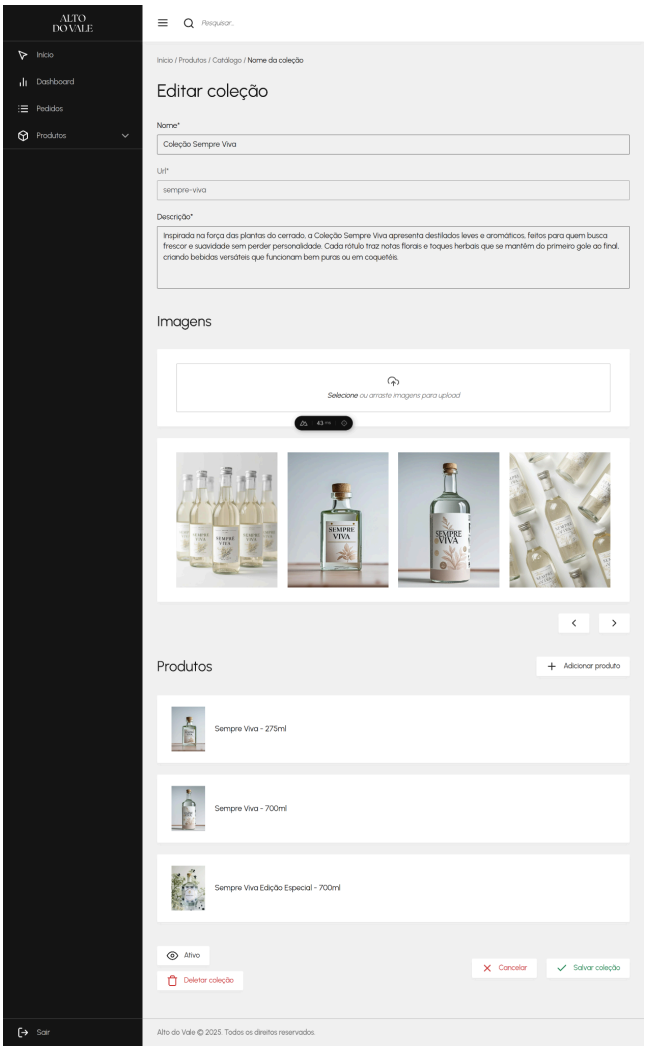
Na seção de produtos, é possível gerenciar o catálogo de coleções da loja (Figuras 30 e 31), e também o catálogo de produtos de cada coleção (Figura 32). Há uma seção separada para gerenciar o estoque de cada produto (Figura 33).

Figura 30 – Catálogo de coleções.



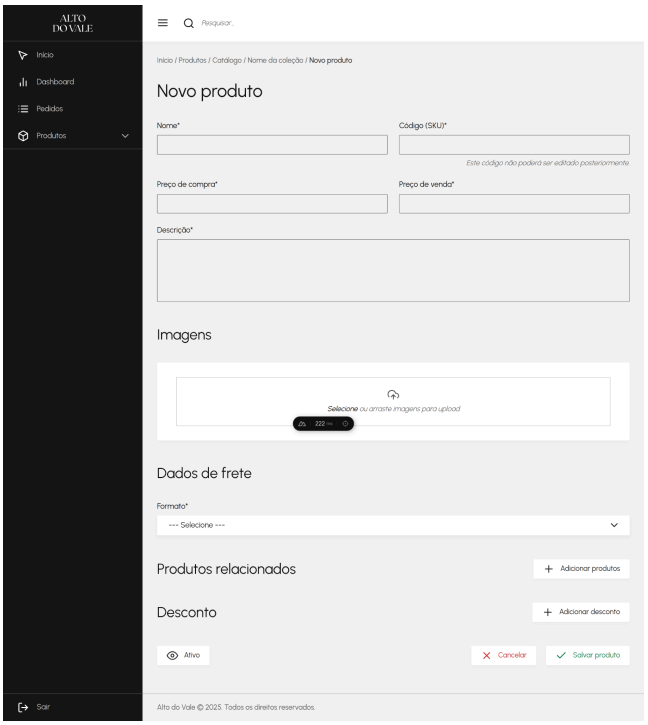
Fonte: O autor, 2025.

Figura 31 – Editar coleção.



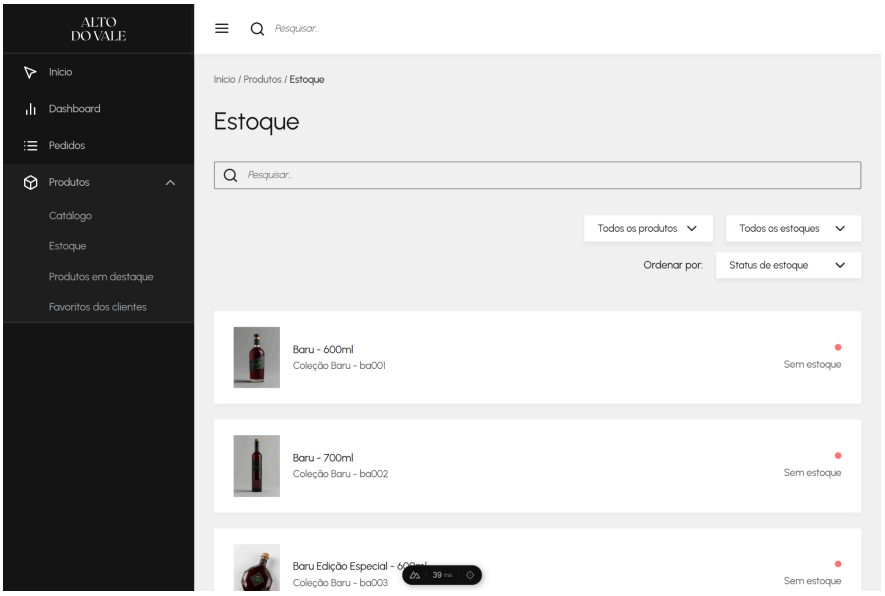
Fonte: O autor, 2025.

Figura 32 – Adicionar produto.



Fonte: O autor, 2025.

Figura 33 – Gerenciar estoque.



Fonte: O autor, 2025.

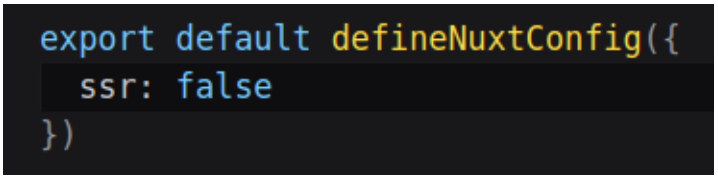
5.2 Comparações e Comportamentos das Arquiteturas de Renderização

A presente seção tem como objetivo demonstrar, de maneira prática, as características principais de cada uma das arquiteturas de renderização, com o propósito de, ao final da análise, concluir qual é a renderização ideal para o e-commerce desenvolvido.

5.2.1 Client-Side Rendering

O principal problema do CSR fundamenta-se no princípio de que o site indexado aos motores de busca não apresenta nenhum conteúdo, ou seja, consiste em página vazia, visto que os títulos, textos, imagens e todo o HTML são carregados no navegador do cliente, e não no servidor. Comprovar esse comportamento é simples, pois o Nuxt oferece uma forma fácil de aplicar o CSR. Basta adicionar o seguinte código ao arquivo de configuração *nuxt.config.ts*, que desativa o SSR padrão do *framework*:

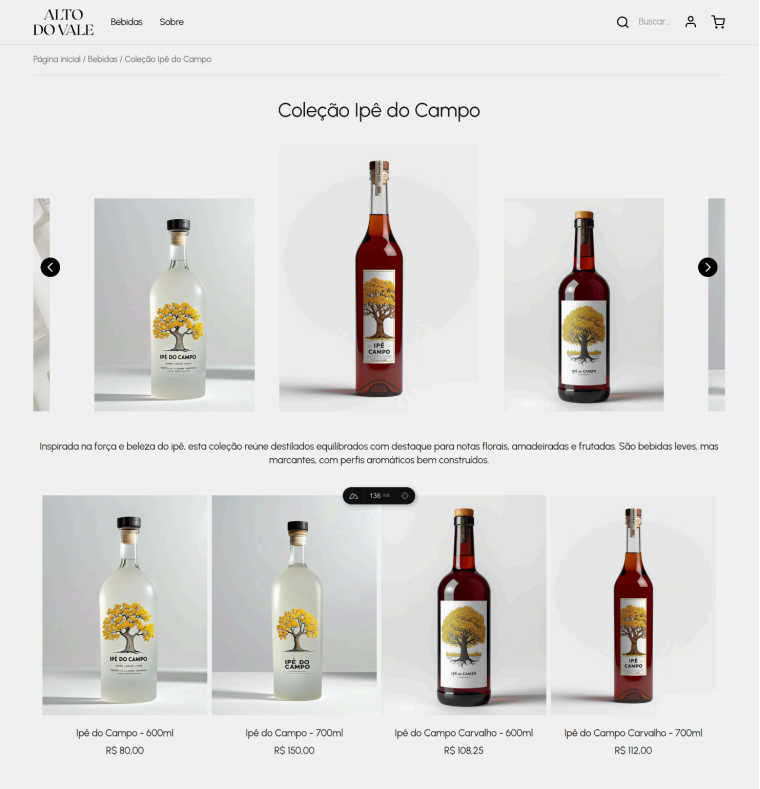
Figura 34 – Código de aplicação do CSR no Nuxt.



Fonte: O autor, 2025.

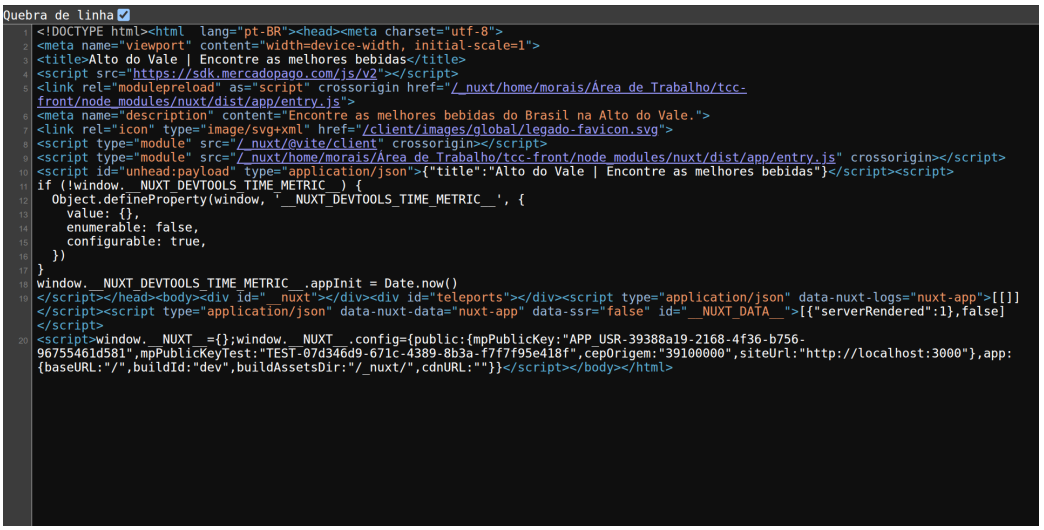
Com base na definição dessa arquitetura, acessa-se uma página qualquer do site. Na visão do cliente (Figura 35), essa página funciona normalmente e contém todos os conteúdos. Contudo, ao selecionar o botão “Inspecionar código fonte da página” no navegador, é possível perceber que o código fonte não contém nenhum título, texto ou imagem em seu HTML, apenas alguns códigos padrões do *framework* (Figura 36).

Figura 35 – Visão do cliente com CSR.



Fonte: O autor, 2025.

Figura 36 – Código fonte da página com CSR.



Fonte: O autor, 2025.

Em contrapartida, caso o código fonte da mesma página seja inspecionado utilizando qualquer outra abordagem de renderização – SSR, SSG ou ISR –, o

código retornado contém, agora, todos os conteúdos da página, até mesmo aqueles carregados via API.

Figura 37 – Código fonte da página com SSR.

Fonte: O autor, 2025.

Fica, assim, demonstrado o comportamento principal do CSR. Uma vez que o código fonte armazenado nos servidores é vazio, o conteúdo é invisível para mecanismos de buscas, o que prejudica o SEO do site.

5.2.2 Static Site Generation

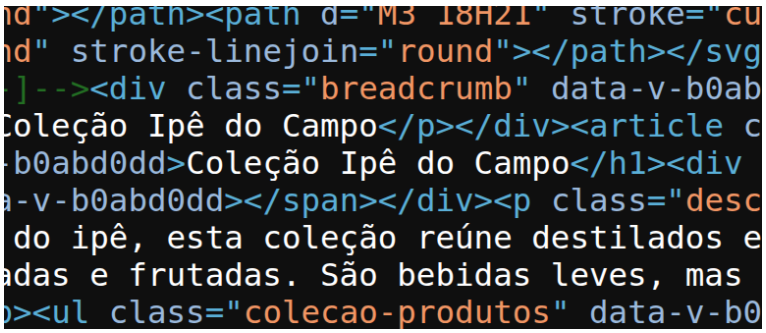
A principal característica do SSG é a natureza estática do conteúdo. Isto é, mesmo caso um texto seja atualizado, por exemplo, o que fica armazenado nos mecanismos de busca é o texto antigo.

Além disso, o Nuxt com SSG não lida muito bem com rotas dinâmicas. Caso queira gerar uma página específica com base em algum objeto da base de dados, é necessário especificar a rota que será gerada através do arquivo de configuração. Neste exemplo, foi gerada a rota “/bebidas/ipe-do-campo” para que seja possível analisar o comportamento do SSG.

Para aplicar essa arquitetura efetivamente, o principal fator está no comando de *build*. Ao invés de utilizar o comando de *build* padrão (*npm run build*), aplica-se o comando *npm run generate*.

Acessando a página da “Coleção Ipê do Campo” e selecionando o botão “Inspecionar código fonte da página”, é possível perceber que o título salvo no código fonte é “Coleção Ipê do Campo” (Figura 38).

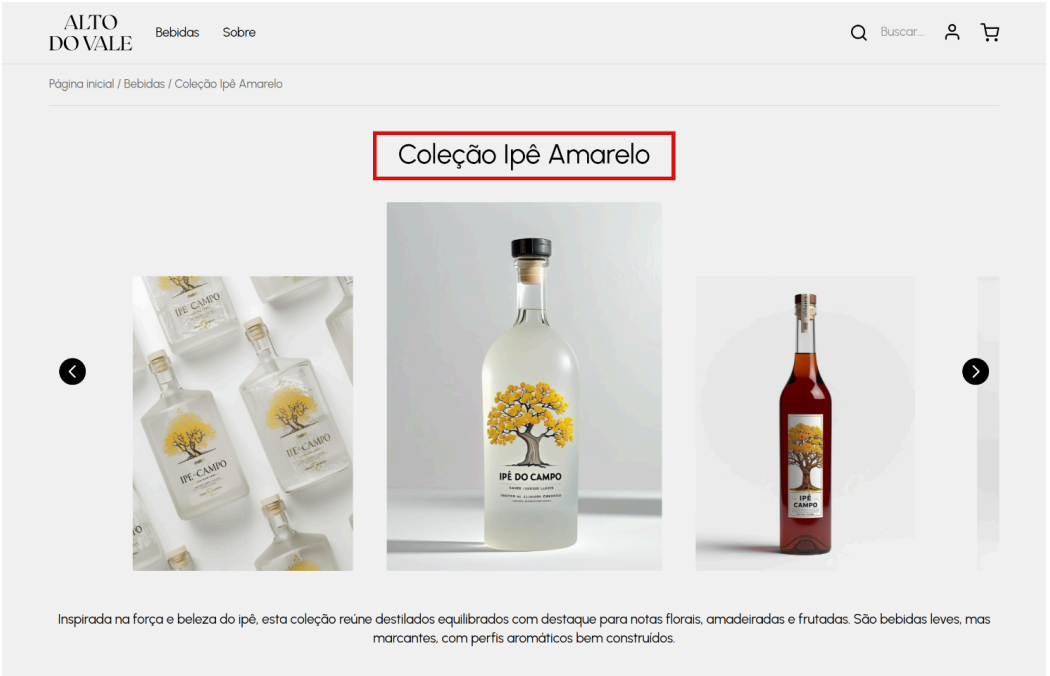
Figura 38 – Código fonte da página com SSG antes da alteração.



Fonte: O autor, 2025.

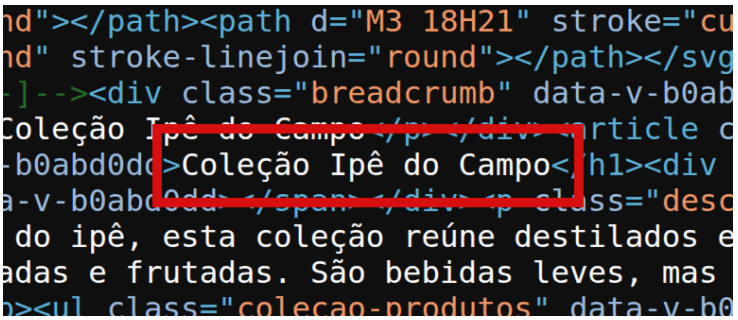
Caso o administrador altere o nome da coleção para “Coleção Ipê Amarelo”, é possível perceber que, mesmo que a mudança tenha ocorrido na interface do cliente (Figura 39), o código fonte permanece desatualizado (Figura 40).

Figura 39 – Visão do cliente com SSG após alteração.



Fonte: O autor, 2025.

Figura 40 – Código fonte da página com SSG após alteração.



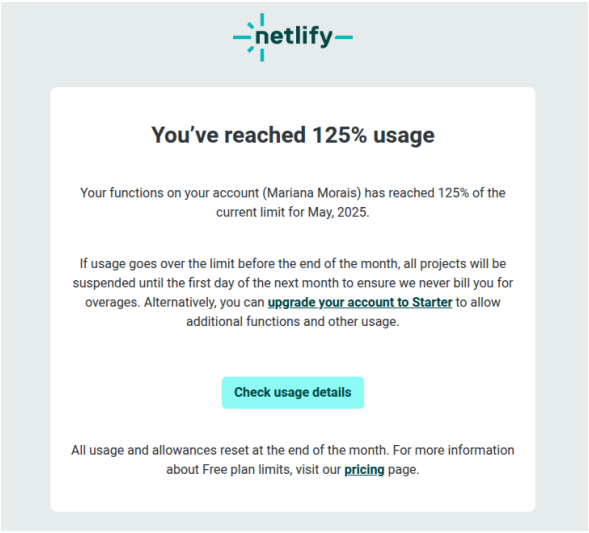
Fonte: O autor, 2025.

Portanto, fica demonstrado o comportamento estático do SSG. Sendo assim, é possível perceber que essa arquitetura não é ideal para sites cujo conteúdo sofre mudanças recorrentes.

5.2.3 Server-Side Rendering

O *Server-Side Rendering*, ao contrário do SSG, atualiza automaticamente os conteúdos no servidor do *front-end* sempre que sofrem alterações. Contudo, foi realizado um teste ao manter o site em produção na plataforma de hospedagem Netlify durante 15 dias. O resultado do teste foi que, em menos de 1 mês e com poucos acessos, o sistema ultrapassou em 125% o limite gratuito da plataforma (Figura 41).

Figura 41 – Notificação da Netlify.



Fonte: O autor, 2025.

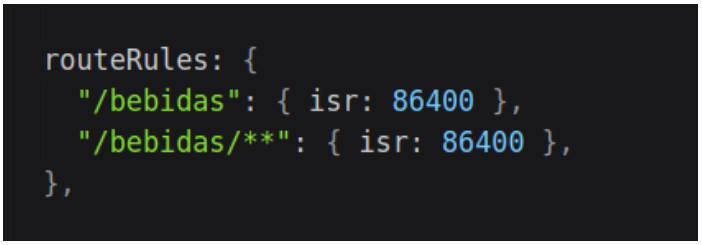
Essa informação indica que o *Server-Side Rendering*, por realizar a renderização a cada nova requisição, resulta em uma sobrecarga do servidor, o que pode ser mais custoso economicamente. Enquanto isso, o plano gratuito de hospedagem da Netlify atende às demais arquiteturas – CSR, SSG e ISR –, sem custos adicionais.

5.2.4 Incremental Static Regeneration

Enquanto cada uma das arquiteturas anteriores apresentam desvantagens consideráveis, o ISR soluciona cada uma delas ao oferecer um sistema baseado em cache.

A principal forma de implementar o ISR no Nuxt é aplicando o seguinte código no arquivo de configurações:

Figura 42 – Implementação do ISR no Nuxt.



Fonte: O autor, 2025.

Nesse caso, indica-se quais são as rotas nas quais o ISR será aplicado e quanto tempo dura o cache, em segundos. Sendo assim, ao invés de renderizar o conteúdo no servidor a cada nova requisição – o que gera sobrecarga e custo elevado –, o servidor retorna uma resposta com base em cache a cada requisição. Ao final do período pré-definido no código, ele renderiza novamente e atualiza o cache.

Essa abordagem **reduz o custo** consideravelmente, sendo uma boa opção em relação ao SSR. Além disso, **garante o dinamismo e a atualização dos conteúdos**, ao contrário do SSG. E, também, **é compatível com motores de busca**, em oposição ao CSR.

Como forma de teste, o sistema com ISR foi mantido em produção durante 15 dias, e como resultado exigiu muito menos recursos da plataforma de hospedagem

em comparação ao SSR. O ISR mostrou-se, portanto, uma boa opção para conteúdos que são atualizados com frequência e que dependem de boa visibilidade e bom custo-benefício.

5.2.5 Arquitetura Híbrida

A abordagem híbrida é ideal para sistemas grandes e complexos, que contém diferentes páginas com diferentes necessidades. A implementação dessa abordagem no Nuxt também é feita de forma simples, utilizando o mesmo atributo *routeRules*. Em uma mesma aplicação, é possível aplicar diversas arquiteturas ao mesmo tempo, conforme na Figura 43.

Figura 43 – Implementação da arquitetura híbrida no Nuxt.

Fonte: O autor, 2025.

Tendo em vista que um sistema de *e-commerce* é composto por diversas partes, cada uma com diferentes características e necessidades, essa foi a arquitetura escolhida para o sistema da Alto do Vale.

Nas rotas de administração, detalhes de usuário e finalização de compra, por se tratarem de páginas críticas que exigem segurança extra, foi aplicada a arquitetura CSR. Afinal, não há interesse em indexar o conteúdo dessas páginas aos

motores de busca. Nas rotas relativas ao catálogo de bebidas, a fim de garantir SEO satisfatório, atualização de conteúdos e bom custo-benefício, foi mantido o ISR. Nas rotas que são raramente atualizadas, foi aplicado o SSG para construção do conteúdo em tempo de *build*.

6 CONCLUSÃO

Este trabalho ofereceu uma perspectiva abrangente em relação aos padrões de renderização *web* modernos, aplicados pelas tecnologias mais utilizadas no desenvolvimento de sites. O estudo permitiu solucionar a principal problemática causada pelo CSR – a dificuldade dos mecanismos de busca em interpretar o conteúdo dos sites feitos puramente com React, Angular ou Vue, o que gera SEO reduzido e pode ocasionar em consequências negativas para todo o negócio de uma empresa.

Da mesma forma, as pesquisas, as aplicações práticas e a análise sobre o efeito de cada arquitetura permitiram a escolha de uma abordagem eficiente para um sistema de *e-commerce*, tendo o site da Alto do Vale como estudo de caso.

Essa pesquisa foi voltada especificamente aos efeitos da escolha da abordagem de renderização em relação à forma como o conteúdo é interpretado pelos motores de busca. Entretanto, o impacto na escolha de renderização vai muito além de apenas o SEO. O tipo de arquitetura pode ter, também, consequências no desempenho, na performance, na usabilidade e na experiência do usuário. Além disso, cada um desses componentes não é afetado apenas pela abordagem de renderização, mas por um conjunto de fatores e práticas de desenvolvimento. Isso quer dizer que, mesmo escolhendo a arquitetura certa, o SEO pode, ainda, ser afetado por outras variáveis. Ou seja, o ecossistema de renderização e de SEO são extremamente abrangentes e abertos a diversos tipos de pesquisa.

Para trabalhos futuros, sugere-se a análise de outros fatores que também impactam no SEO além da renderização. Ou, também, a comparação das diferentes técnicas de renderização com base em métricas que permitem mensurar performance, desempenho, e outros aspectos relevantes de um site. O *Incremental Static Regeneration* também é um ótimo tópico para pesquisas, visto que, por ser uma arquitetura recente até o presente trabalho, ainda carece de estudos acadêmicos.

7 REFERÊNCIAS

BEZERRA, Leonardo. **O impacto da abordagem de renderização na otimização para motores de busca.** 2024. Disponível em: <https://ric.cps.sp.gov.br/bitstream/123456789/21533/1/20241S_Leonardo%20de%20Souza%20Bezerra_OD2105.pdf>. Acesso em: 16 dez. 2025.

PAIVA, Camila. **A importância do Search Engine Optimization para uma melhor experiência do consumidor: caso de estudo Impacting Digital.** 2018. Disponível em: <https://www.proquest.com/openview/c3f46475c596f1fe665c58dae04e58ed/1?pq-origsite=gscholar&cbl=2026366&diss=y>. Acesso em: 11 dez. 2025.

BEKE, Mathias. **On the Comparison of Software Quality Attributes for Client-side and Server-side Rendering.** 2018. Disponível em: <https://denbeke.be/thesis/versions/mathias-beke-final.pdf>. Acesso em: 11 dez. 2025.

SANTOS, Joey. **Client-side vs Server-side Rendering: Impactos no Desempenho e Experiência do Usuário em Aplicações Web.** 2024. Disponível em: <https://bib.pucminas.br/pergamumweb/downloadArquivo?vinculo=ZTk5ODIyM1kyOWtSVzF3Y21WellUMHlOU1poWTJWeWRtODlOVFkyTlRZMUpuTmxjVkJoY21GbmNtRm1iejB5Sm5ObGNWTmxZMkZ2UFRjbWEyRnlaR1Y0UFU0bWJHOWpZV3hCY25GMWFYWnZQVU5QVFZCQlVsUkpURWhCVFVWT1ZFOG1ibTl0WlVOaGJXbHVhRzg5TURBd01HUTBMekF3TURCa05EWmpMbkJrWmc9PWFmMDkzM2U=&nomeExtensao=.pdf>. Acesso em: 16 dez. 2025.

VEPSÄLÄINEN, Juho. et al. **Implications of the Edge Computing for Static Site Generation.** 2023. Disponível em: <https://arxiv.org/pdf/2309.05669>. Acesso em: 11 dez. 2025.