INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS PICOS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JOSÉ RYAN DE LIMA OLIVEIRA

VITRINEJÁ: PLATAFORMA WEB DE DIVULGAÇÃO E COMERCIALIZAÇÃO DE PRODUTOS REGIONAIS

PICOS, PIAUÍ
2026

JOSÉ RYAN DE LIMA OLIVEIRA

VITRINEJÁ: PLATAFORMA WEB DE DIVULGAÇÃO E COMERCIALIZAÇÃO DE PRODUTOS REGIONAIS

Trabalho de Conclusão de Curso (Relatório Técnico de Software) apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Picos.

Orientador(a): Prof. Mestre Ciro Ferreira de Carvalho Júnior

PICOS, PIAUÍ
2026

JOSÉ RYAN DE LIMA OLIVEIRA

VitrineJá: Plataforma Web de Divulgação e Comercialização de Produtos Regionais

Trabalho de Conclusão de Curso (Relatório Técnico de Software) apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Picos.

Aprovada em: 10 / 08 / 26.

BANCA EXAMINADORA

_____

Prof. Mestre Ciro Ferreira de Carvalho Júnior (Orientador)
Instituto Federal do Piauí (IFPI)

_____

Profa Laís Nicoly Moreira Nunes da Silva
Instituto Federal do Piauí (IFPI)

_____

Prof. Esp José Carlos dos Santos Silva
Instituto Federal do Piauí (IFPI)

PICOS, PIAUÍ
2026

RESUMO

A ausência de canais estruturados para a inserção digital do comércio de proximidade, agravada pelos altos custos de plataformas tradicionais e pela baixa maturidade técnica dos comerciantes, reduz a competitividade de pequenos lojistas e restringe o acesso dos consumidores aos produtos de sua própria região, cenário evidente no Centro-Sul Piauiense. Este relatório técnico apresenta o desenvolvimento da plataforma VitrineJá, um sistema de software *full-stack* projetado para centralizar a divulgação de catálogos comerciais locais, atuando como uma vitrine informativa para pequenas empresas e microempreendedores individuais. A metodologia aplicada baseia-se no ciclo de vida iterativo e incremental da engenharia de software, separando rigorosamente as responsabilidades do sistema através de uma arquitetura desacoplada (*headless*). O ecossistema técnico compreende uma interface responsiva desenvolvida em *Next.js* e *React* no *frontend*, comunicando-se assincronamente com uma *API RESTful* independente construída em *Node.js* e *Express* no *backend*, cujos dados são persistidos em uma infraestrutura relacional *PostgreSQL* via *Supabase*. A validação do sistema deu-se por meio de testes funcionais orientados aos requisitos e fluxos operacionais da aplicação. Conclui-se que a solução estabelece um canal viável e escalável para atenuar as barreiras técnicas e financeiras de digitalização, promovendo a modernização e o fortalecimento do comércio de proximidade regional.

Palavras-chave: Engenharia de Software; Desenvolvimento Web; Vitrine Digital; Comércio Local; Next.js; API RESTful.

ABSTRACT

The absence of structured channels for the digital integration of local commerce, exacerbated by the high costs of traditional platforms and the limited technical maturity of merchants, reduces the competitiveness of small retailers and restricts consumers' access to regional products—a scenario clearly observed in the Center-South region of Piauí. This technical report presents the development of the VitrineJá platform, a full-stack software system designed to centralize the showcase of local commercial catalogs, operating as an informative digital storefront for small businesses and individual micro-entrepreneurs. The applied methodology relies on the iterative and incremental software engineering lifecycle, strictly separating system concerns through a headless, decoupled architecture. The technical ecosystem comprises a responsive interface built with Next.js and React on the frontend, asynchronously communicating with an independent RESTful API developed in Node.js and Express on the backend, whose data is persisted in a PostgreSQL relational infrastructure via Supabase. System validation was conducted through functional testing oriented to application requirements and core operational flows. It is concluded that the solution establishes a viable and scalable channel to mitigate technical and financial barriers to digitization, fostering the modernization and strengthening of regional local commerce.

Keywords: Software Engineering; Web Development; Digital Showcase; Local Commerce; Next.js; RESTful API.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface* (Interface de Programação de Aplicações)

CI/CD – *Continuous Integration / Continuous Deployment* (Integração Contínua / Implantação Contínua)

CRUD – *Create, Read, Update, Delete* (Criar, Ler, Atualizar, Excluir)

DER – Diagrama de Entidades e Relacionamentos

JSONB – *JavaScript Object Notation Binary* (Notação de Objetos JavaScript Binária)

JWT – *JSON Web Token*

MEI – Microempreendedor Individual

MFA – *Multi-Factor Authentication* (Autenticação Multifator)

POSTGRESQL – Sistema Gerenciador de Banco de Dados Relacional e Objeto-Relacional

REST – *Representational State Transfer* (Transferência de Estado Representacional)

RF – Requisito Funcional

RNF – Requisito Não Funcional

RTS – Relatório Técnico de Software

TCC – Trabalho de Conclusão de Curso

UAT – *User Acceptance Testing* (Teste de Aceitação do Usuário)

UML – *Unified Modeling Language* (Linguagem de Modelagem Unificada)

UX – *User Experience* (Experiência do Usuário)

SUMÁRIO

1 INTRODUÇÃO

A evolução contemporânea das Tecnologias de Informação e Comunicação (TICs) remodelou as dinâmicas de consumo global, consolidando um ambiente mercadológico intensamente dependente da presença digital (LAUDON; LAUDON, 2014). A expansão contínua do comércio eletrônico (*e-commerce*) e a hegemonia de grandes *marketplaces* transacionais transformaram o comportamento dos consumidores, que passaram a priorizar conveniência, agilidade e centralização na descoberta de bens de consumo (TURBAN *et al.*, 2017). No entanto, o avanço dessa digitalização corporativa exibe uma assimetria socioespacial marcante, concentrando vantagens competitivas e operacionais em polos metropolitanos (CASTELLS, 2003).

No cenário socioeconômico brasileiro, o comércio de proximidade — formado por microempresas, empresas de pequeno porte e Microempreendedores Individuais (MEIs) — atua como sustentáculo econômico indispensável para municípios interioranos e bairros periféricos. Segundo o Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE, 2024), os pequenos negócios representam aproximadamente 97% dos empreendimentos ativos no país, respondendo por mais de 26% do Produto Interno Bruto (PIB) e concentrando 70% dos novos postos de trabalho formais. Apesar de sua relevância estrutural, esses estabelecimentos enfrentam graves entraves para viabilizar sua transformação digital nos moldes convencionais.

Em microrregiões interioranas, como o Centro-Sul Piauiense, as elevadas taxas de comissão praticadas por intermediadores corporativos (que variam habitualmente entre 12% e 25% por transação), somadas aos custos de licenciamento de plataformas proprietárias e à baixa maturidade técnica dos comerciantes, inviabilizam o ingresso competitivo no comércio eletrônico tradicional (SEBRAE, 2023). Como estratégia de contingência para contornar esse isolamento, expressiva parcela dos lojistas recorre ao uso informal de mídias sociais e mensageiros instantâneos (CETIC.BR, 2024).

De acordo com a pesquisa *TIC Empresas* (CETIC.BR, 2024), cerca de 74% dos pequenos negócios brasileiros utilizam aplicativos de mensagens como o WhatsApp e 71% adotam redes sociais como canal primário de contato e divulgação. Todavia, sob a perspectiva de modelagem de sistemas abordada por Sommerville (2011) e Pressman e Maxim (2021), canais conversacionais diretos não foram

projetados para funcionar como sistemas de informação gerenciais estruturados. A ausência de persistência relacional de atributos como preço e estoque gera fragmentação e impõe alta sobrecarga cognitiva aos usuários.

Embora existam soluções no mercado nacional focadas na criação de catálogos simplificados (como *Kyte*, *Meloja* e *Stoqui*), tais ferramentas operam predominantemente sob o paradigma de lojas individuais isoladas. Nelas, cada estabelecimento gera seu próprio endereço eletrônico, cabendo exclusivamente ao lojista o ônus de divulgá-lo de forma dispersa. Esse formato organiza o catálogo particular do lojista, mas não atende à necessidade de um portal centralizado de descoberta regional para os cidadãos do município.

Diante dessa lacuna de centralização e visibilidade no varejo de proximidade, este trabalho apresenta o desenvolvimento da plataforma web VitrineJá. O sistema foi concebido como uma solução de software *full-stack* de baixo custo de implantação, projetada para consolidar, estruturar e divulgar catálogos comerciais locais. A plataforma atua como uma vitrine digital informativa que disponibiliza aos lojistas um painel simplificado de gestão de estoques e oferece aos consumidores um ponto unificado de busca e descoberta geolocalizada.

A concepção da solução fundamenta-se nos preceitos da Engenharia de Software moderna, adotando uma arquitetura desacoplada (*headless*) para garantir escalabilidade e independência entre camadas. O ecossistema técnico integra uma interface web reativa desenvolvida com Next.js e *React* no *frontend*, conectada assincronamente a uma *API RESTful* em Node.js e Express no *backend*, com persistência relacional no PostgreSQL através do *Supabase*. Com essa estrutura, o projeto busca atenuar as barreiras tecnológicas e financeiras de digitalização, promovendo o fortalecimento da economia regional.

1.1 Objetivos

Ao organizar este estudo, fez-se importante apresentar os objetivos dessa pesquisa. Nesta subseção, são definidos o objetivo geral e os objetivos específicos.

1.1.1 Objetivo geral

Desenvolver uma plataforma web baseada em uma arquitetura desacoplada e modular, com o intuito de centralizar e otimizar a divulgação de catálogos de produtos e estabelecimentos do comércio local, promovendo a inclusão digital de pequenos lojistas e simplificando a busca de produtos por consumidores regionais.

1.1.2 Objetivos específicos

O trabalho busca mapear os requisitos funcionais e não funcionais necessários para estruturar um catálogo digital de produtos adaptado a usuários não técnicos, projetar e construir uma *API RESTful* robusta e segura para o gerenciamento centralizado de vendedores, produtos, categorias e permissões de acesso, desenvolver uma interface *frontend* rica, responsiva e otimizada, garantindo usabilidade tanto para dispositivos móveis quanto *desktop*, implementar um modelo de dados relacional que garanta a integridade, consistência e rastreabilidade dos registros de estabelecimentos e produtos, e configurar uma esteira de integração e entrega contínua para garantir implantações automatizadas e evolução incremental estável do *software*.

2 FUNDAMENTAÇÃO TEÓRICA

O desenvolvimento de plataformas de software voltadas para o comércio e divulgação digital assenta-se sobre preceitos metodológicos rígidos de Engenharia de Software. Segundo Sommerville (2011), o ciclo de vida de um sistema computacional corporativo deve ser norteado pela modularidade e extensibilidade, garantindo que novos requisitos possam ser acoplados sem degradação do núcleo arquitetural existente. Esse conceito é vital para o desenvolvimento da plataforma, cujo planejamento prevê uma transição suave de uma vitrine informativa para um *marketplace* transacional completo.

A arquitetura Cliente-Servidor distribuída e o padrão *REST* (*Representational State Transfer*), propostos por Fielding (2000), constituem a base para a criação de

APIs modernas que trocam dados estruturados por meio do protocolo HTTP sob o formato JSON. Conforme aponta Pressman e Maxim (2021), a separação clara entre a interface de apresentação e as regras de controle de negócio resguarda a estabilidade sistêmica, permitindo que falhas na camada visual não comprometam a persistência e a integridade do banco de dados relacional subjacente.

No âmbito da usabilidade e Experiência do Usuário (UX), Garrett (2010) enfatiza que sistemas voltados a públicos heterogêneos e de variados níveis de letramento digital devem mitigar a carga cognitiva. Plataformas de comércio eletrônico locais necessitam de interfaces autoexplicativas, buscas tolerantes a falhas e carregamento assíncrono otimizado para evitar o abandono do canal. Paralelamente, a Segurança da Informação atua como pilar de sustentação não funcional; de acordo com Stallings (2014), o controle de acesso baseado em perfis (RBAC) e o uso de criptografia robusta na persistência de credenciais são imperativos éticos e técnicos para resguardar dados corporativos e a identidade dos usuários em redes públicas.

3 TECNOLOGIAS ENVOLVIDAS

As tecnologias integradas foram selecionadas de forma a garantir alto desempenho, custo operacional nulo ou mínimo de infraestrutura e conformidade com os padrões de engenharia *web* vigentes.

3.1 Front-end

A camada de visualização e interação com o usuário opera com componentes funcionais reativos, gerenciamento avançado de rotas e otimização de renderização.

Quadro 1 - Especificação técnica da stack do Frontend

Tecnologia	Versão	Descrição
Next.JS	16.2.9	*Framework full-stack* baseado em *React*. Justifica-se pela implementação de roteamento declarativo e compilação otimizada de páginas. No contexto do sistema, garante que as vitrines dos vendedores locais carreguem de forma quase instantânea e possuam indexação nativa em motores de busca.
JavaScript	Nativo	Linguagem executada no lado do cliente, responsável pela manipulação do *DOM*, orquestração de estados locais e validação imediata de formulários de cadastro de produtos.
Axios	1.9.0	Cliente HTTP baseado em promessas. Utilizado para efetuar chamadas assíncronas assinaladas à *API* do

		backend, viabilizando a configuração centralizada de interceptores para injeção do *token JWT* nas requisições autenticadas.
Tailwind CSS	4.0.9	*Framework CSS* utilitário. Permite construir uma interface responsiva, fluida e consistente sem a sobrecarga de arquivos de estilo volumosos, acelerando a renderização em dispositivos móveis.

Fonte: Elaborado pelo autor com base em Costa, Silva e Nunes (2026) e Tilkov e Vinoski (2010).

3.2 Back-end

A API de controle e persistência de regras corporativas processa os dados de forma assíncrona, assegurando segurança no tráfego de payloads e isolamento de escopo, operando em conformidade com as diretrizes lógicas estruturadas por Moraes (2023).

Quadro 2 - Especificação técnica da stack do Backend.

Tecnologia	**Versão**	**Descrição**
Node.JS	LTS	Ambiente de *runtime JavaScript* assíncrono orientado a eventos. Garante alta vazão no tratamento de requisições concorrentes sem bloqueio de I/O, ideal para suportar múltiplos lojistas atualizando estoques simultaneamente.
Express	5.1.0	Framework minimalista focado em microsserviços e rotas HTTP. Utilizado para mapear os *endpoints REST* da plataforma, estruturar o fluxo de controle de dados e gerenciar barramentos de tratamento de exceções.
JWT	9.0.2	Implementação de *JSON Web Token*. Utilizada para viabilizar autenticação apátrida (*stateless*), gerando tokens criptografados de curta duração após o login do vendedor, mantendo as sessões do servidor seguras.
Cloudinary	SDK Nativo	Serviço de gerenciamento de mídia em nuvem. Utilizado para processar, otimizar, armazenar e servir as imagens dos produtos enviadas pelos lojistas, aliviando o tráfego de dados do servidor principal de banco de dados através de uma CDN nativa.

Fonte: Elaborado pelo autor com base em Moraes (2023) e Sadalage e Fowler (2013).

3.3 Infraestrutura

A infraestrutura física e de nuvem apoia-se em conceitos modernos de fornecimento sob demanda e arquitetura distribuída escalável.

Quadro 3 - Topologia de serviços de Infraestrutura.

Serviço / Ferramenta	**Descrição**
Vercel	Hospedagem *serveless* otimizada para o ecossistema *Next.js*. Garante a distribuição global dos arquivos estáticos da interface através de redes de borda (*Edge Network*), otimizando a entrega final ao usuário.
Render	Plataforma baseada em nuvem para execução contínua de serviços de *backend*. Gerencia a execução do contêiner *Node.js/Express*, oferecendo *deploys* automáticos baseados em ganchos.
Supabase	Infraestrutura cloud baseada em *PostgreSQL* relacional. Fornece transacionalidade ACID, isolamento de dados e suporte a consultas relacionais complexas de catálogo de produtos com alta consistência.

Fonte: Elaborado pelo autor com base em Valente (2020) e Fielding (2000).

3.4 Ferramentas de desenvolvimento, testes e comunicação

O ambiente de engenharia apoia-se em ferramentas de produtividade, controle de versão distribuído, automação de testes de *endpoints* e documentação estruturada do ecossistema de software.

Quadro 4 - Ferramentas auxiliares de desenvolvimento.

Ferramenta	**Versão**	**Descrição**
VSCode	1.126	*Visual Studio Code*. Ambiente de Desenvolvimento Integrado (IDE) de código aberto amplamente otimizado para o ecossistema *JavaScript/Node.js*. Utilizado como editor de texto principal por sua agilidade e ecossistema de extensões para engenharia de *software*.
Postman	Desktop Client	Plataforma de desenvolvimento e testes de *APIs*. Utilizada para simular requisições *HTTP (GET, POST, PUT, DELETE),* validar o comportamento dos *controllers*, inspecionar *payloads JSON* e testar o tempo de resposta dos *middlewares* de autenticação antes da integração com o *frontend*.
Git	2.x	Sistema de controle de versão distribuído. Empregado para o gerenciamento de histórico do código-fonte, ramificação de escopos operacionais (*branching*) e garantia de rastreabilidade das modificações realizadas pela equipe de desenvolvimento.

Github	Cloud Service	Plataforma de hospedagem de repositórios baseada em *Git*. Atua como o nó centralizador do código-fonte do projeto, provendo ferramentas de revisão de código (*Pull Requests*) e servindo como gatilho disparador para a esteira automática de CI/CD.

Fonte: Elaborado pelo autor com base em Pressman e Maxim (2021) e Sommerville (2011).

4 MODELAGEM DO PROJETO

Para o ciclo de engenharia, optou-se pela adoção de um processo de desenvolvimento de software de caráter iterativo e incremental. Essa escolha foi determinada pela premissa de escopo evolutivo da plataforma, estabelecendo uma primeira fase focada exclusivamente em catálogo e visibilidade (Vitrine) para, em ciclos subsequentes, incorporar mecanismos transacionais financeiros (Marketplace).

As iterações foram delimitadas em quatro fases cíclicas:

- **Análise e Elicitação:** Mapeamento conceitual do domínio de comércio de rua e MEIs, refinando as interações em fluxos lógicos;
- **Projeto Lógico:** Engenharia do banco de dados relacional e definição da matriz de rotas da *API RESTful*;
- **Construção Modular:** Codificação paralela e independente das camadas *frontend* e *backend*, amparada por contratos estáveis de *endpoints*;
- **Esteira de CI/CD:** Configuração de processos automatizados onde cada submissão estável de código no *branch* principal do *GitHub* dispara rotinas de compilação e publicação imediata nos servidores da *Vercel* e *Render*.

4.1 Levantamento de requisitos

O estabelecimento dos requisitos de um sistema de *software* constitui a etapa basilar para a delimitação do escopo operacional, arquitetural e técnico da solução. Segundo Wiegers e Beatty (2013), a engenharia de requisitos de *software* deve atuar na extração ativa e na tradução das necessidades dos usuários reais, mitigando lacunas de comunicação e alinhando os objetivos de negócio às restrições do sistema.

Para garantir o alinhamento da plataforma com as demandas socioeconômicas do varejo de proximidade, o projeto adotou uma abordagem empírica de descoberta,

partindo da investigação direta com o mercado consumidor e microempreendedores locais.

4.1.1 Levantamento de dados

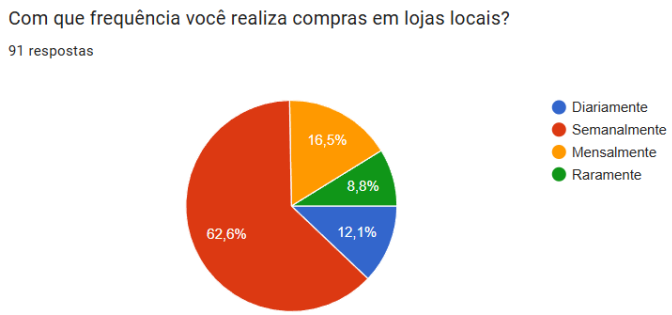
A aplicação de questionários estruturados digitais como ferramenta exploratória quantitativa permite mapear com precisão o comportamento de amostras heterogêneas, reduzindo distorções de interpretação e fornecendo os insumos estatísticos necessários para a posterior tradução em requisitos de software, em conformidade com as diretrizes metodológicas de Gil (2022).

Conforme apontado por Kotonya e Sommerville (1998), a triangulação de dados coletados junto a diferentes grupos de interesse (*stakeholders*) é um requisito de qualidade indispensável para validar a viabilidade de um novo *software*, garantindo que as interfaces desenvolvidas e os barramentos de microsserviços resolvam problemas reais de comunicação e assimetria de informação.

4.1.1.1 Pesquisa com usuários

A primeira fase da pesquisa concentrou-se em mapear a jornada de compra e os pontos de fricção informacional vivenciados por **91 consumidores** do comércio de proximidade. Inicialmente, buscou-se mensurar a recorrência de consumo local, evidenciando que **62,6%** realizam compras semanalmente e **12,1%** diariamente, totalizando **74,7%** de engajamento recorrente, conforme ilustrado no Gráfico 1.

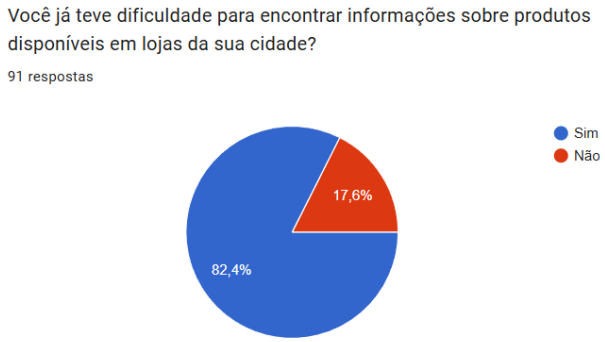
Gráfico 1 - Frequência de compras em lojas locais por consumidores



Fonte: Elaborado pelo autor (2026)

Apesar da alta relevância do varejo físico para a economia regional, a investigação identificou uma barreira crítica na acessibilidade a metadados de produtos: **82,4%** dos participantes relataram dificuldades severas para localizar dados sobre a disponibilidade de mercadorias na cidade, conforme exposto no Gráfico 2.

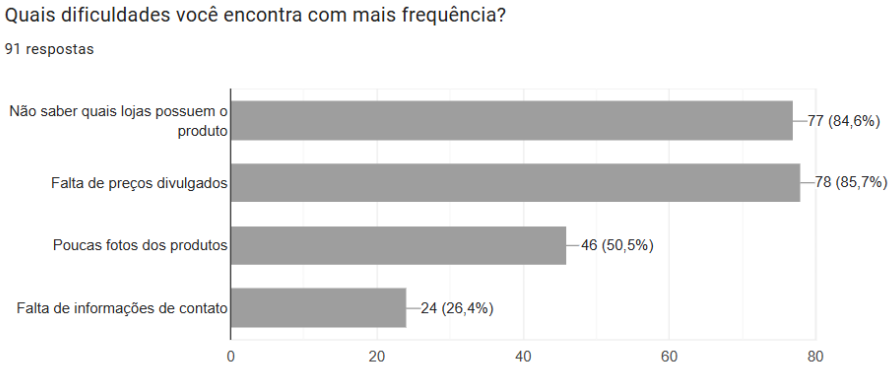
Gráfico 2 - Acesso a informações dos produtos



Fonte: Elaborado pelo autor (2026)

Ao estratificar a natureza dessas dificuldades (Gráfico 3), o diagnóstico estatístico revelou que as maiores causas de fricção concentrando-se na falta de preços divulgados (85,7%) e na ausência de centralização para saber qual estabelecimento possui o produto buscado (84,6%).

Gráfico 3 - Principais dificuldades informacionais listadas pelo consumidor.



Fonte: Elaborado pelo autor (2026)

Essa obscuridade informacional impacta negativamente a conversão financeira das lojas: **42,9%** dos consumidores desistem de comprar frequentemente e outros

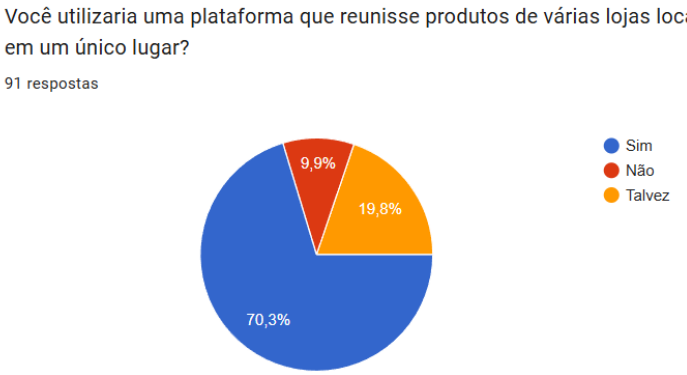
42,9% abandonam a intenção ocasionalmente (somando **85,8%** de desistência motivada unicamente por lacunas de dados), conforme ilustra o Gráfico 4.

Gráfico 4 - Frequência de desistência de compra por ausência de dados.

Fonte: Elaborado pelo autor (2026)

Esse cenário fundamenta a necessidade da criação do VitrineJá: **70,3%** dos entrevistados afirmaram que utilizariam prontamente uma plataforma centralizada e **19,8%** indicaram interesse potencial (Gráfico 5).

Gráfico 5 - Intenção de uso de uma plataforma unificada de vitrines.



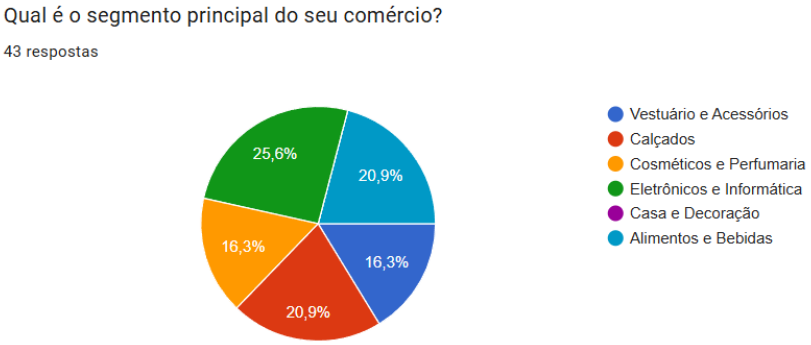
Fonte: Elaborado pelo autor (2026)

4.1.1.2 Pesquisa com público-alvo

De forma paralela à sondagem dos consumidores, a segunda vertente da pesquisa de campo investigou as dores operacionais e de gestão de **43**

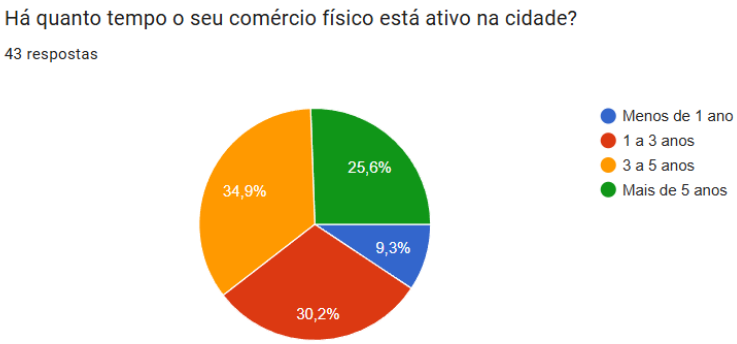
microempreendedores locais (o público-alvo central para a retaguarda do sistema). Este grupo está distribuído em segmentos competitivos como Eletrônicos (**25,6%**), Calçados (**20,9%**), Alimentos (**20,9%**), Vestuário (**16,3%**) e Cosméticos (**16,3%**), cuja maturidade de ponto físico majoritariamente supera os 3 anos ativos na praça (Gráficos 6 e 7).

Gráfico 6 - Distribuição por segmentos do comércio local estudado.



Fonte: Elaborado pelo autor (2026)

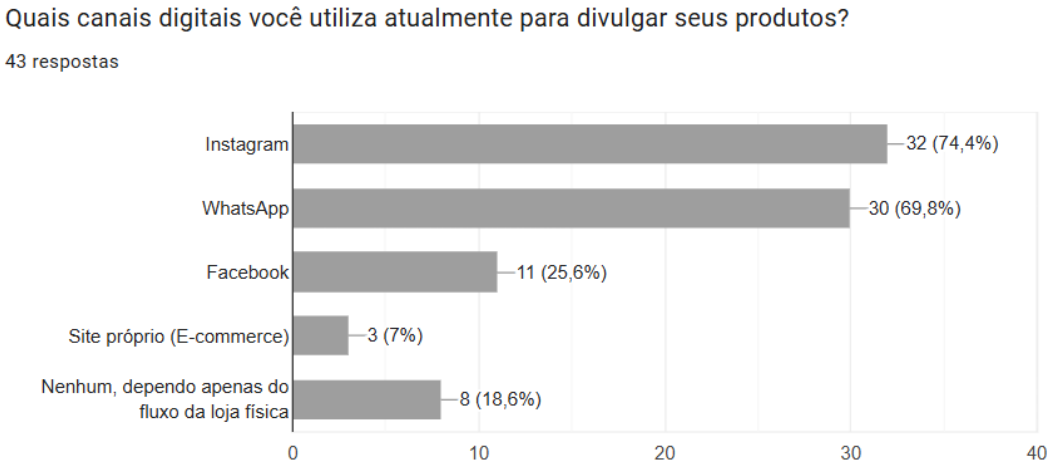
Gráfico 7 - Tempo de atividade do estabelecimento físico na cidade.



Fonte: Elaborado pelo autor (2026)

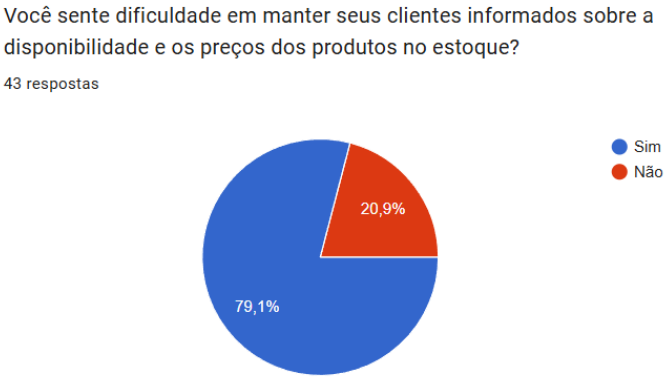
Embora utilizem canais digitais de divulgação pulverizados como Instagram (**74,4%**) e WhatsApp (**69,8%**) de forma massiva (Gráfico 8), os comerciantes enfrentam um grave gargalo de manutenção síncrona de dados: **79,1%** assumem dificuldades crônicas em manter os clientes devidamente informados sobre preços e estoque em tempo real (Gráfico 9).

Gráfico 8 - Canais digitais utilizados atualmente para publicidade.



Fonte: Elaborado pelo autor (2026)

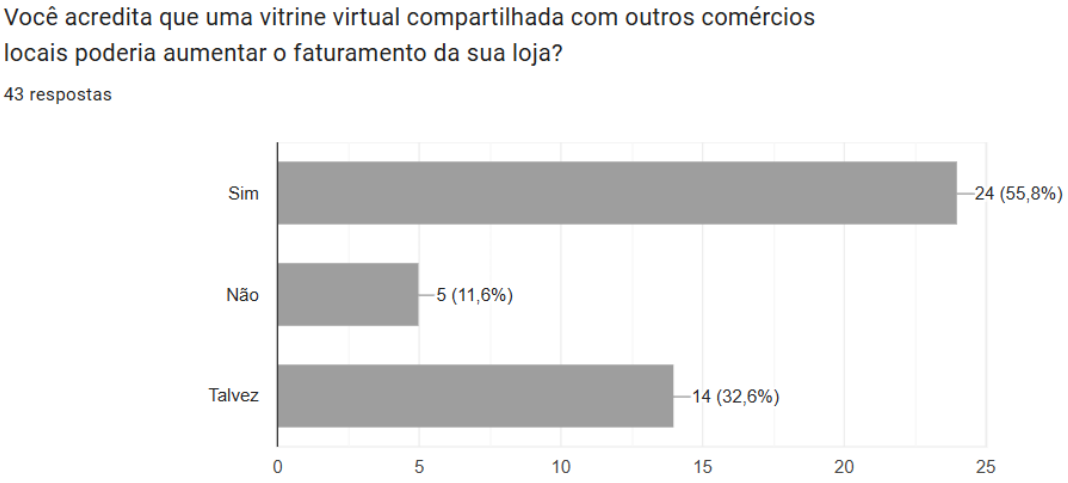
Gráfico 9 - Dificuldade na manutenção de preços e estoques.



Fonte: Elaborado pelo autor (2026)

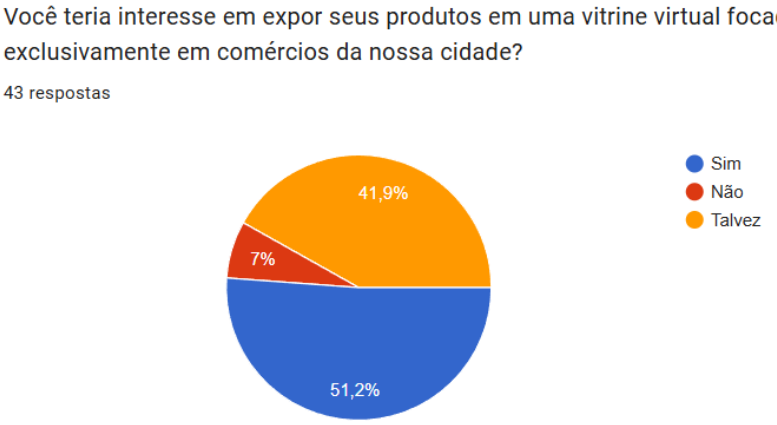
Para solucionar essa dor de gestão de estoque sem adicionar custos operacionais ou burocracias financeiras elevadas, **55,8%** dos lojistas acreditam que a vitrine compartilhada elevará o faturamento (Gráfico 10), e **51,2%** afirmam intenção imediata de expor seus catálogos na plataforma, além disso, **41,9%** responderam "Talvez", demonstrando abertura à proposta, embora condicionada a fatores como custos, funcionalidades ou benefícios oferecidos. Apenas 7,0% dos participantes afirmaram não possuir interesse, evidenciando baixa rejeição à solução (Gráfico 11).

Gráfico 10 - Expectativa de aumento de faturamento via vitrine virtual.



Fonte: Elaborado pelo autor (2026)

Gráfico 11 - Interesse em expor produtos na plataforma regional.



Fonte: Elaborado pelo autor (2026)

A consolidação estatística destes dados justifica empiricamente o escopo técnico construído para o VitrineJá.

4.1.2 Requisitos funcionais

Os Requisitos Funcionais (RF) mapeiam as ações que o sistema deve executar para responder às interações disparadas pelos atores cadastrados. Conforme

explicitado por Sommerville (2011), estes requisitos descrevem detalhadamente o comportamento esperado do software, as reações do sistema a entradas específicas e as diretrizes de computação que determinam como os serviços lógicos da aplicação devem operar frente às requisições do usuário.

Quadro 5 - Especificação dos Requisitos Funcionais.

Código	Requisito	Descrição
RF1	Autenticação de Usuários	O sistema deve autenticar vendedores e administradores através de e-mail e senha, emitindo um *token* de sessão estável para rotas restritas.
RF2	Cadastro de Produtos	O sistema deve viabilizar a inserção de produtos no catálogo do estabelecimento, exigindo nome, descrição, preço e status de estoque.
RF3	Upload de Imagens	O sistema deve permitir que o lojista realize o *upload* de uma imagem representativa para cada produto cadastrado.
RF4	Organização por Categorias	O sistema deve agrupar os produtos por categorias lógicas para simplificar a navegação do cliente.
RF5	Pesquisa de Produtos	O sistema deve fornecer uma barra de busca pública na tela inicial que filtre produtos disponíveis na cidade por palavras-chave ou nome.
RF6	Painel Administrativo	O sistema deve disponibilizar uma área de controle centralizada para administradores gerenciarem categorias e bloquearem cadastros irregulares.

Fonte: Elaborado pelo autor (2026)

4.1.3 Requisitos não-funcionais

Os Requisitos Não Funcionais (RNF) definem as restrições comportamentais, critérios de desempenho e propriedades técnicas de qualidade que regulam a operação do ecossistema de software, não se limitando a funções específicas, mas condicionando o comportamento do sistema como um todo (SOMMERVILLE, 2011).

Quadro 6 - Especificação dos Requisitos Não Funcionais.

Código	Requisito	Descrição
RNF01	Desempenho de Resposta	A API do *backend* deve responder a requisições de consulta pública de catálogo em um tempo mínimo de segundos em condições normais de tráfego.
RNF02	Mitigação de Hibernação	O microsserviço de *backend* na Render deve conter rotinas de tráfego artificial periódico para neutralizar o modo de suspensão de servidores gratuitos.
RNF03	Integridade Referencial	O banco de dados *PostgreSQL* deve forçar restrições de chaves estrangeiras, impedindo a exclusão de estabelecimentos com produtos vinculados.

Fonte: Elaborado pelo autor (2026)

4.2 Diagramas de casos de uso

A modelagem de casos de uso constitui uma etapa fundamental no processo de Engenharia de Requisitos, atuando como um elemento de transição entre a especificação abstrata das necessidades dos usuários e o projeto arquitetural do sistema. Segundo Sommerville (2011), um caso de uso descreve de forma abstrata uma interação entre um ator externo e o próprio sistema de *software*, mapeando explicitamente o escopo das transações permitidas e definindo as fronteiras físicas e lógicas da aplicação. Na modelagem estruturada pela *Unified Modeling Language* (UML), essa técnica permite documentar o comportamento dinâmico do software sem expor detalhes internos de codificação ou estruturas de arquivos do banco de dados.

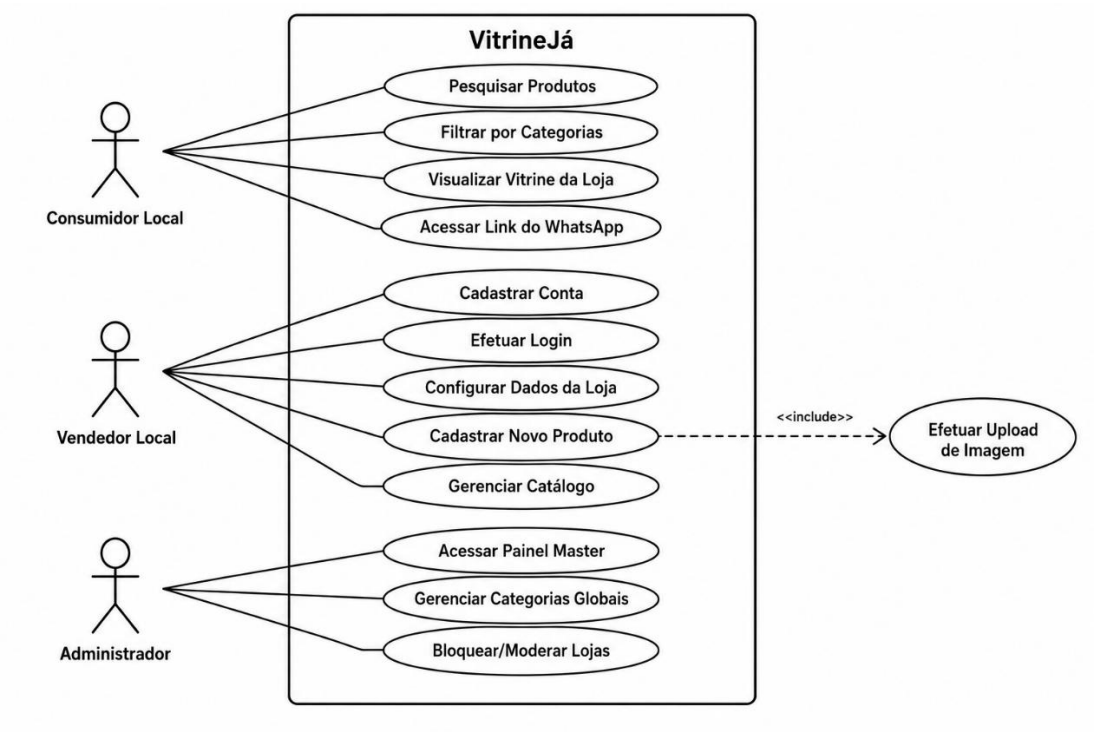
No contexto da plataforma, a aplicação deste artefato UML viabiliza a segregação clara de responsabilidades e níveis de visibilidade mercadológica para as entidades de dados cadastrados. Conforme apontam Pressman e Maxim (2021), a

definição precisa dos atores — entendidos como entidades externas que interagem diretamente com o *software* de forma ativa ou passiva — é crucial para o estabelecimento de regras de validação consistentes e para o projeto de interfaces orientadas à experiência do usuário (UX).

O diagrama mapeia formalmente os fluxos operacionais divididos entre as áreas públicas de navegação livre, destinadas à descoberta de produtos, e os módulos restritos protegidos por criptografia e controle de acesso baseado em papéis (RBAC).

Abaixo, a Figura 1 ilustra graficamente o mapeamento de casos de uso da plataforma, explicitando as conexões lógicas entre os atores e os balões operacionais do sistema.

Figura 1 - Diagrama de casos de uso



Fonte: Elaborado pelo autor (2026)

4.3 Diagrama de classe

Enquanto os diagramas de comportamento e interação como o de Casos de Uso concentram-se em mapear as ações e as fronteiras lógicas sob a ótica dos atores operacionais, a modelagem estrutural cumpre o papel de delimitar a arquitetura

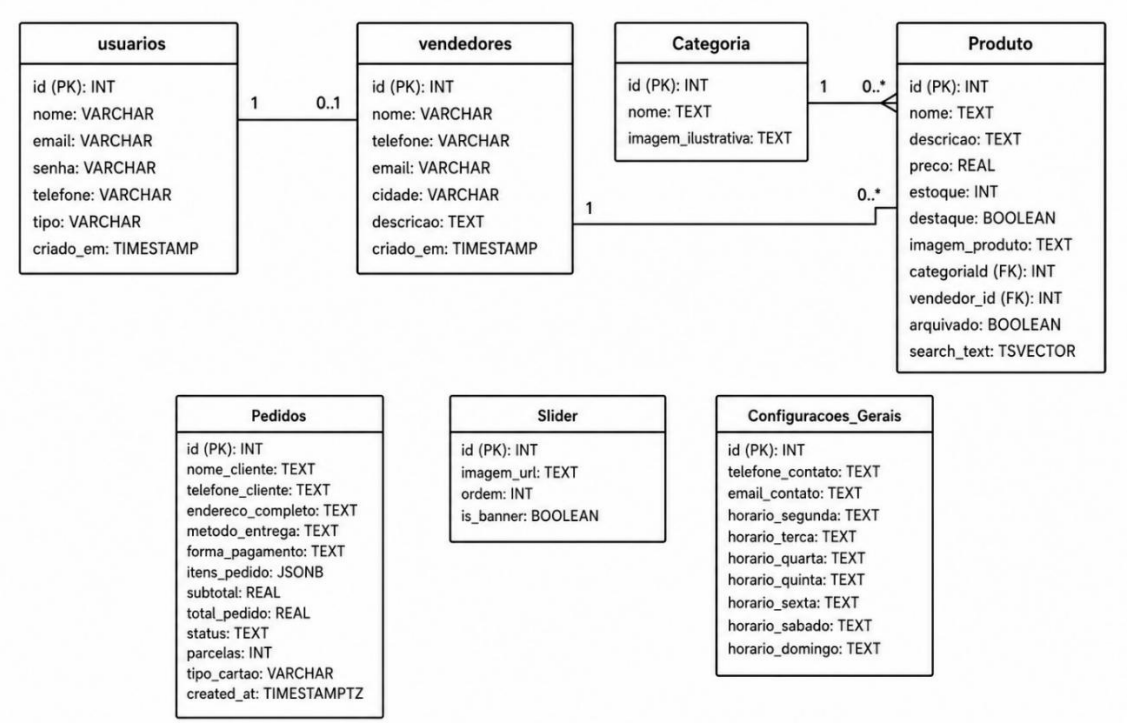
estática e a topologia de dados que dão sustentação ao ecossistema de *software*. No escopo da *Unified Modeling Language* (UML), o Diagrama de Classes atua como o artefato central para a transição entre a modelagem conceitual e a implementação do esquema físico de armazenamento. Segundo Pressman e Maxim (2021), este diagrama descreve formalmente a estrutura de classes de um sistema, detalhando a tipagem de seus atributos internos, suas restrições de integridade e a natureza das associações que governam o tráfego e a persistência das informações.

A modelagem estrutural reflete diretamente a engenharia do banco de dados relacional implementado sobre o *PostgreSQL*. Conforme preconizado por Sommerville (2011), o estabelecimento rigoroso de chaves estrangeiras (*Foreign Keys*) e restrições de cardinalidade é um requisito de qualidade indispensável para assegurar transações seguras e evitar a ocorrência de anomalias ou inconsistências de dados em ambientes multiusuário.

A estrutura de classes do sistema evidencia relacionamentos fundamentais de dependência e herança lógica para o comércio local. A associação entre as entidades vendedores e Produto, bem como entre Categoria e Produto, opera sob uma cardinalidade de um para muitos (1:0..^*), garantindo o isolamento de estoques e a organização indexada do catálogo de mercadorias. Adicionalmente, o ecossistema incorpora classes estruturadas de forma modular e independente como Pedidos, projetadas com tipos de dados semiestruturados (*JSONB*) para encapsular o histórico imutável das transações.

Abaixo, a Figura 2 ilustra a modelagem de classes e a distribuição estrutural das entidades que compõem o núcleo de persistência do *software*.

Figura 2 - Diagrama de classe



Fonte: Elaborado pelo autor (2026)

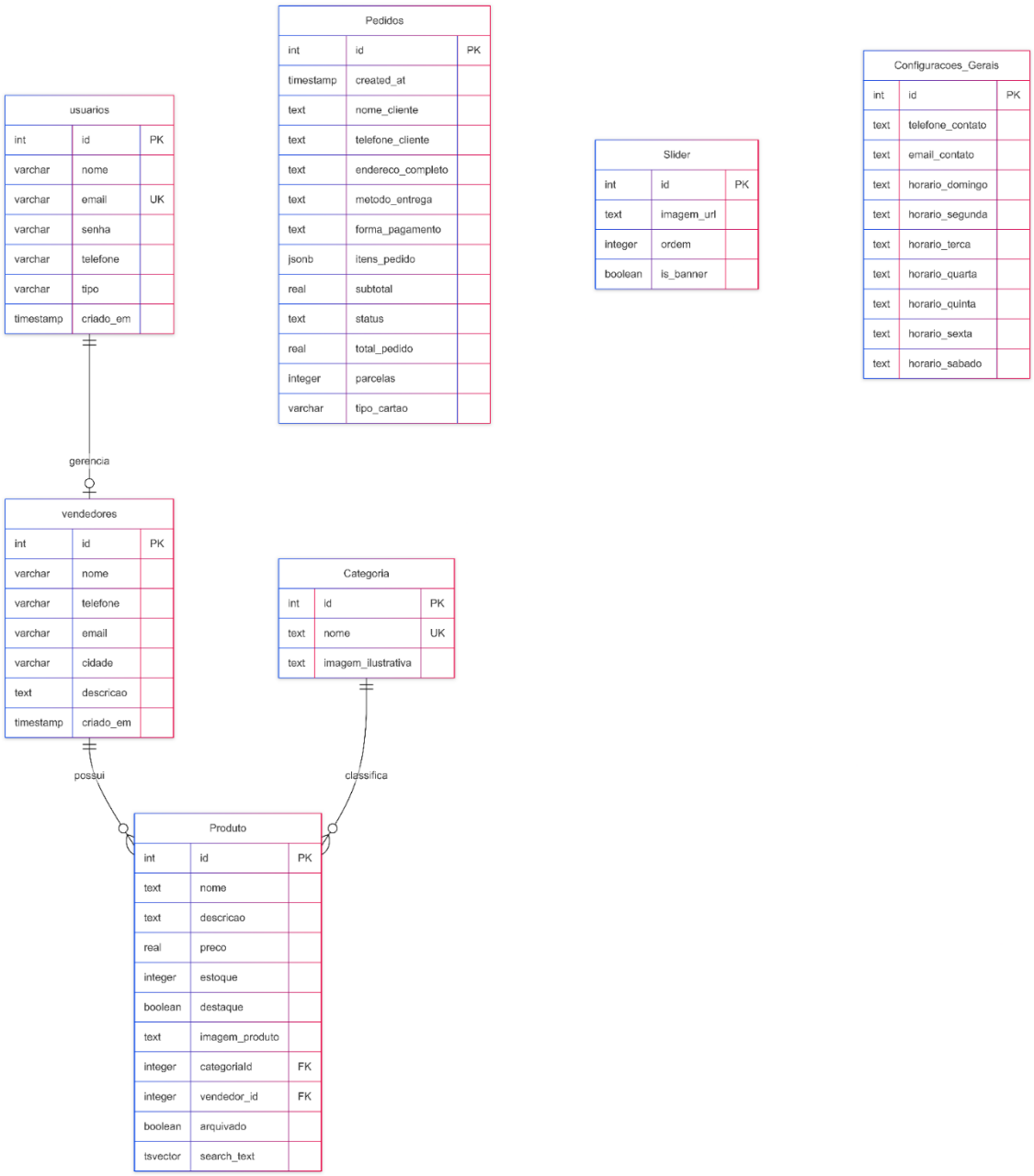
4.4 Diagrama de entidade-relacionamento

Enquanto o Diagrama de Classes aborda a visão estática do *software* sob a ótica da orientação a objetos, a modelagem de banco de dados exige uma representação formal centrada estritamente na persistência física dos dados e suas restrições transacionais. A modelagem por meio do Diagrama de Entidades-Relacionamentos (*DER*) traduz as regras de negócio em tabelas, atributos, chaves primárias (*Primary Keys*) e chaves estrangeiras (*Foreign Keys*). Conforme discutido por Pressman e Maxim (2021), o mapeamento explícito do DER fornece a abstração necessária para validar o esquema físico do SGBD, garantindo a aplicação de chaves de integridade que blindam o banco contra redundâncias informacionais.

O *DER* reflete o esquema *SQL* do *PostgreSQL* hospedado no *Supabase*. O diagrama explicita os vínculos de cardinalidade clássicos do varejo de proximidade: o relacionamento de um para muitos (1:N) mapeia como os produtos estão associados às suas respectivas categorias lógicas e como os estoques individuais estão

vinculados de forma exclusiva a cada comerciante da tabela de vendedores. Adicionalmente, as entidades de suporte operacional — como Pedidos e Configuracoes_Gerais - aparecem de forma desacoplada, preservando a imutabilidade de dados semiestruturados em formato *JSONB*. A Figura 3 ilustra a modelagem física do banco de dados.

Figura 3 - Diagrama entidades-relacionamentos



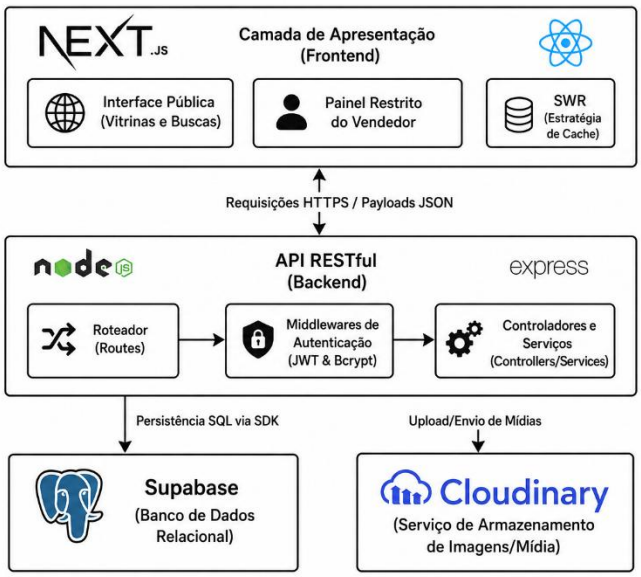
Fonte: Elaborado pelo autor (2026)

4.5 Arquitetura do sistema

A arquitetura lógica estrutura-se sob o modelo de desacoplamento completo entre a interface de interação (*Frontend*) e o motor de persistência de regras de negócio (*Backend API*). O tráfego de dados é efetuado por canais seguros *HTTPS* utilizando verbos formais do protocolo *REST*.

O cliente *Next.js* faz requisições assíncronas enviando *payloads* padronizados em *JSON*. Nas rotas protegidas por autenticação, o *token JWT* gerado pelo servidor é injetado diretamente nos cabeçalhos da chamada *HTTP*. No *backend*, o *Express* intercepta a requisição, submete o fluxo de *bytes* aos *middlewares* de segurança e delega a transação lógica à camada de serviço, que opera de forma integrada com as tabelas relacionais do *PostgreSQL* hospedadas no *Supabase*.

Figura 4 - Arquitetura do Sistema



Fonte: Elaborado pelo autor (2026)

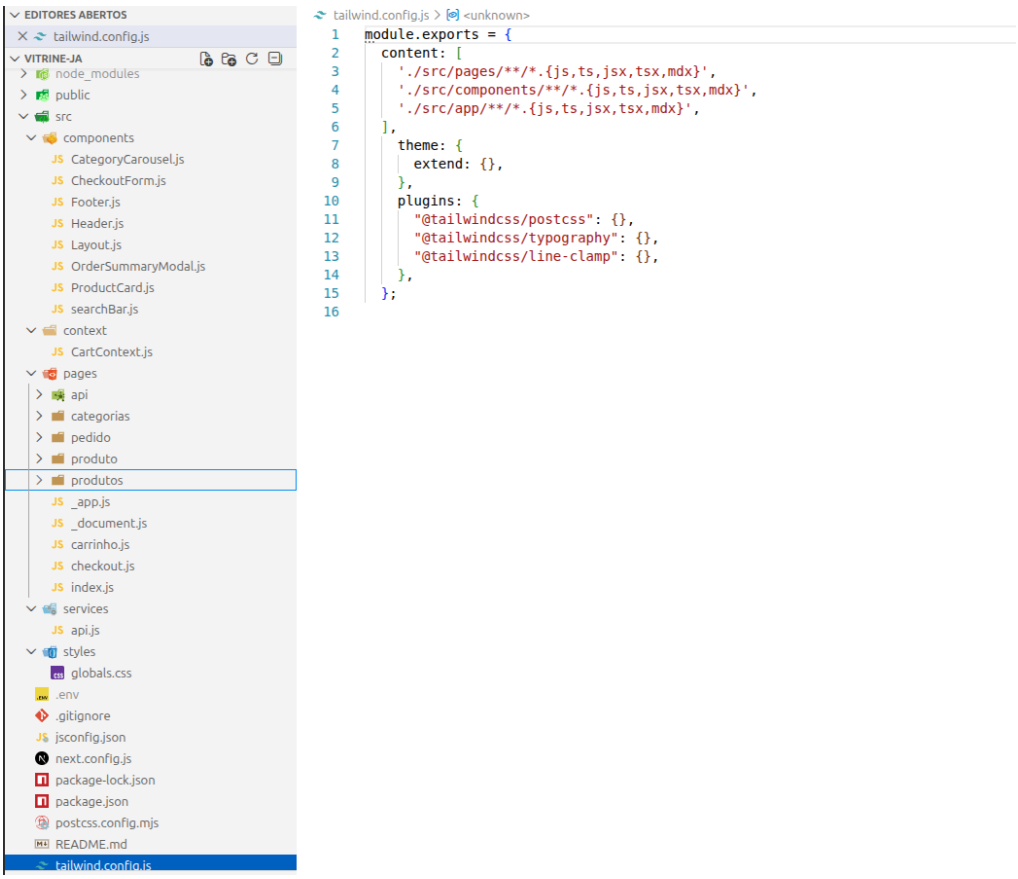
4.5.1 Padrão utilizado no frontend

A organização arquitetural do lado do cliente e a divisão modular de componentes foram projetadas seguindo os preceitos de separação de conceitos estabelecidos por Costa, Silva e Nunes (2026), garantindo que as interfaces reativas operem de forma otimizada e fluida junto ao usuário final. A distribuição do código-

fonte dentro do diretório /src baseia-se no princípio da separação de conceitos, dividindo a aplicação entre componentes visuais atômicos, gerenciamento de estado global e roteamento de páginas. Conforme aponta Sommerville (2011), a modularização de uma interface *client-side* reduz o acoplamento do sistema e eleva a manutenibilidade, permitindo alterações visuais isoladas sem impacto nas regras de controle de negócio.

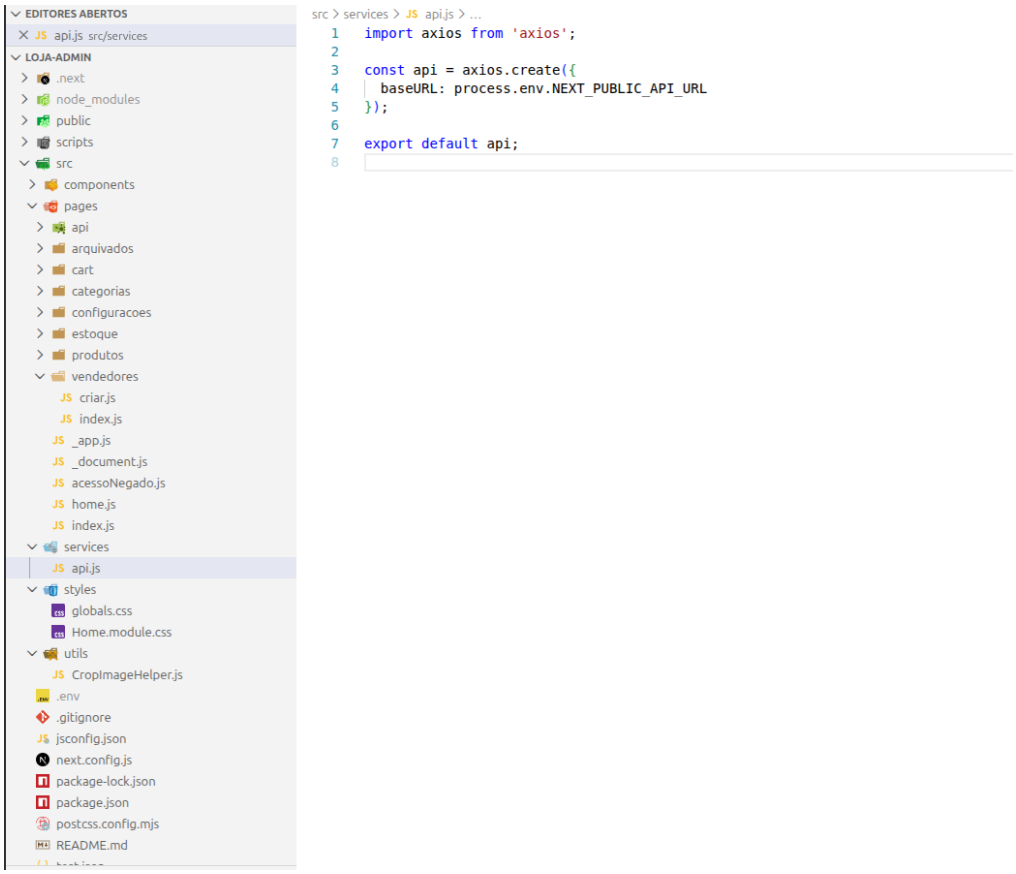
No ecossistema, o *Next.js* gerencia de forma unificada tanto a área pública de navegação (vitrine e buscas) quanto a área restrita (painel administrativo do lojista). Abaixo, a Figura 16 ilustra a organização física de diretórios e arquivos que estruturam o *frontend* do sistema.

Figura 5 - Organização das pastas da vitrine - Frontend



Fonte: Elaborado pelo autor (2026)

Figura 6 - Organização das pastas do painel administrativo – Frontend



Fonte: Elaborado pelo autor (2026)

A anatomia funcional da árvore de diretórios do *frontend* é detalhada a seguir:

- ***src/components/:*** Concentra as unidades visuais reutilizáveis da interface com o usuário. Conforme Garrett (2010), a consistência dos elementos de interface é um pilar indispensável para mitigar a carga cognitiva do usuário. Este diretório encapsula arquivos fundamentais para a experiência de uso (UX), tais como:
    - *CategoryCarousel.js*: Componente dinâmico de carrossel para filtragem por segmentos de mercado;
    - *SearchBar.js*: Barra de pesquisa baseada em entrada de texto para consultas rápidas;
    - *ProductCard.js*: Cartão visual que renderiza as informações consolidadas de cada item;

  - *CheckoutForm.js* e *OrderSummaryModal.js*: Interfaces responsáveis por estruturar o encerramento do fluxo de compras e confirmação de pedidos.
- ***src/context/ (CartContext.js):*** Camada dedicada ao gerenciamento de estado global da aplicação utilizando a *Context API* do *React*. O arquivo *CartContext.js* desempenha a função de unificar e compartilhar o estado síncrono do carrinho de compras (adição, remoção e cálculo de subtotal de itens) entre componentes visuais isolados (como listagens, cabeçalhos e *checkout*), eliminando a necessidade de propagação manual de propriedades (*prop drilling*), técnica recomendada por Flavio Almeida (2023) para a escalabilidade de aplicações reativas.
- ***src/pages/:*** Define as rotas lógicas da aplicação mapeadas a partir do sistema de arquivos do *Next.js.* O ecossistema segmenta de forma segura a experiência pública da administrativa através de arquivos e subdiretórios específicos:
  - *index.js*: Arquivo correspondente à rota raiz pública (/), encarregado de orquestrar a página inicial do catalogo digital;
  - *_app.js* e *_document.js*: Arquivos estruturais de inicialização e injeção de estilos globais;
  - carrinho.js e *checkout.js*: Rotas responsáveis pela consolidação dos dados informados pelo consumidor final;
  - Subdiretórios /categorias, /pedido e /produto: Rotas parametrizadas que realizam a exibição focada de registros e o tratamento dinâmico de chaves lógicas.
- ***src/services/ (api.js):*** Módulo centralizador de consumo de recursos de rede. O arquivo api.js encapsula a instância cliente da biblioteca ***Axios***, definindo a *URL* base apontada para a *API* remota de *backend* hospedada na Render, além de injetar interceptores para o tratamento padronizado de códigos de erro *HTTP*, seguindo as boas práticas de comunicação cliente-servidor documentadas por Tilkov e Vinoski (2010).
- ***tailwind.config.js*:** Arquivo de configuração que especifica as regras de compilação da folha de estilo utilitária *Tailwind CSS*. O script gerencia a injeção dos plugins de qualidade visual.

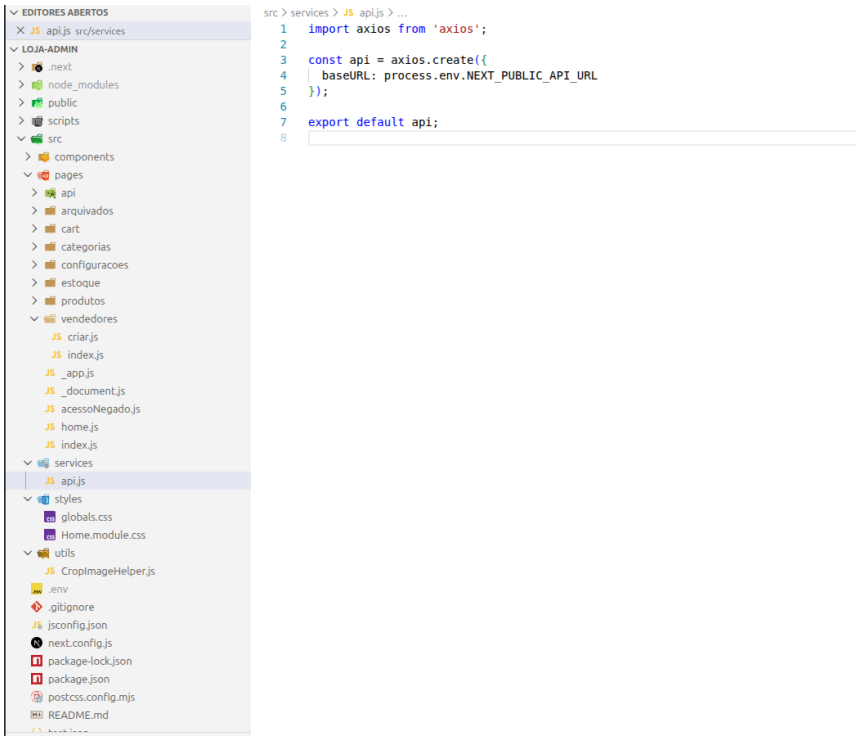
A viabilidade das requisições assíncronas e a arquitetura cliente-servidor distribuída que dão suporte ao tráfego do Axios e do Tailwind CSS apoiam-se nos conceitos de modularização de ecossistemas web mapeados por Tilkov e Vinoski (2010), garantindo alto desempenho na renderização do lado do cliente.

4.5.2 Padrão utilizado no backend

A arquitetura do servidor de aplicação foi projetada sob o amparo do modelo de camadas distribuídas, separando as responsabilidades de mapeamento de chamadas, aplicação de filtros de segurança e execução de regras transacionais. Conforme discutido por Valente (2024), a estruturação modular em camadas mitiga a complexidade no desenvolvimento de microsserviços, isolando o fluxo de dados e simplificando a manutenção e a escalabilidade global do ecossistema.

Abaixo, a Figura 7 ilustra a organização física do código-fonte e o arquivo de inicialização principal da API do *backend*.

Figura 7 - Organização das pastas – Backend



Fonte: Elaborado pelo autor (2026)

O fluxo de processamento de requisições e a organização de diretórios do servidor atendem aos seguintes módulos de engenharia de software:

- ***src/index.js* (Ponto de Entrada Global):** Atua como o núcleo de inicialização (*bootstrap*) do servidor de aplicação *Express*. O script é encarregado de carregar as configurações de ambiente (*dotenv.config()*), instanciar a aplicação e configurar os middlewares globais indispensáveis para a integridade de rede:
    - *app.use(cors()):* Controla as políticas de compartilhamento de recursos de origens cruzadas, garantindo que apenas o domínio *frontend* homologado consiga ler os dados expostos;
    - *app.use(express.json()):* Configura o interpretador (*parser*) nativo para tratamento e conversão automática de *payloads* textuais trafegados sob o formato padronizado *JSON*.
- **src/routes/ (Camada de Roteamento Detalhado):** Isola os caminhos lógicos expostos pela API HTTP. O barramento principal descentraliza a aplicação através de arquivos de roteamento específicos atrelados aos seus respectivos controladores:
    - app.use('/auth', authRoutes): Endpoints de segurança encarregados de interceptar tentativas de login e registro, invocando bibliotecas de cifragem (bcryptjs) e assinaturas tokenizadas (JWT);
    - app.use('/vendedores', vendedoresRoutes) e app.use('/produtos', produtosRoutes): Gerenciam os fluxos de persistência e leitura das entidades comerciais públicas e do catálogo estruturado;
    - app.use('/pedidos', pedidosRoutes): Orquestra o barramento transacional de vendas e o recebimento de snapshots de carrinhos em formato JSONB;
    - app.use('/categorias', categoriasRoutes), app.use('/configuracoes', configuracoesRoutes) e app.use('/slider', sliderRoutes): Gerenciam as tabelas utilitárias de taxonomia e customização da interface da vitrine.
- ***src/services/* (Abstração de Persistência e Integrações):** Camada responsável por isolar a lógica transacional corporativa e gerenciar a comunicação externa. O subdiretório encapsula o arquivo *supabase.js*, que inicializa o cliente SDK oficial para conectar e executar comandos transacionais no banco de dados relacional *PostgreSQL* hospedado na nuvem, abstraindo

consultas SQL manuais. É nesta camada que se integram serviços de mídia (*Cloudinary*) para o processamento de imagens de produtos.

- ***src/controllers.js* (Orquestração de *Payloads*):** Camada intermediária encarregada de recepcionar as requisições HTTP validadas pelos roteadores, extrair parâmetros contidos na URL (*params*) ou no corpo da mensagem (*body*), delegar o processamento à camada de serviço e devolver a resposta estruturada ao cliente juntamente com o código de status HTTP correspondente (como 200 OK ou 201 Created).

A camada responsável por isolar a lógica transacional corporativa inicializa o cliente SDK oficial para conectar e executar comandos relacionais, atendendo às boas práticas de persistência poliglota em nuvem propostas por Sadalage e Fowler (2013).

4.6 Interface

A camada de interface com o usuário (*User Interface* - UI) representa a materialização física das funcionalidades do sistema, atuando como o canal de comunicação direto entre os atores humanos e as regras de negócio processadas no *backend*. Segundo Garrett (2010), o projeto de interface em sistemas voltados à web deve alinhar de forma harmônica a arquitetura da informação e o design visual, mitigando a carga cognitiva do usuário e otimizando a jornada de navegação. As interfaces foram concebidas sob os preceitos do design responsivo e da consistência visual, garantindo usabilidade tanto em dispositivos móveis (foco do consumidor local) quanto em desktops (foco do comerciante).

Conforme discutido por Pressman e Maxim (2021), a validação de uma interface de software deve demonstrar como os requisitos funcionais e os casos de uso mapeados são operados em telas reais. A seguir, são apresentadas as principais interfaces que compõem o ecossistema da aplicação, acompanhadas de seus memoriais descritivos e fluxos de interação.

4.6.1 Interface da vitrine digital (visão do consumidor)

A interface pública foi projetada sob o preceito do livre acesso (acesso sem autenticação), permitindo rotinas de busca, triagem taxonômica e montagem de carrinho dinâmico sem que o visitante precise preencher cadastros prévios. Essa

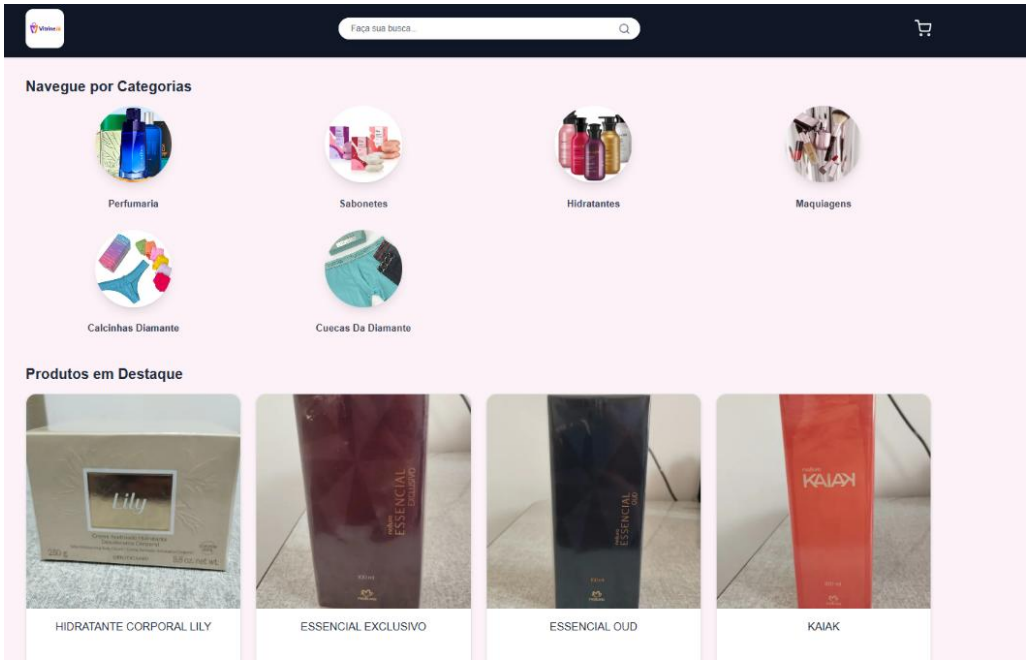
abordagem mitiga a taxa de rejeição e replica a fluidez de uma vitrine física tradicional no ambiente digital, servindo como um catálogo aberto de alta performance que consome diretamente os dados públicos expostos via API.

Abaixo, detalham-se os cenários e fluxos visuais que compõem a jornada de compra do consumidor.

4.6.1.1 Tela inicial (home page)

A interface principal da plataforma constitui o ambiente de descoberta de produtos, projetada para ser leve, intuitiva e indexável. Ela permite que o visitante realize buscas textuais e filtre o catálogo sem a necessidade de autenticação prévia.

Figura 8 - Interface de Tela Inicial (Home)



Fonte: Elaborado pelo autor (2026)

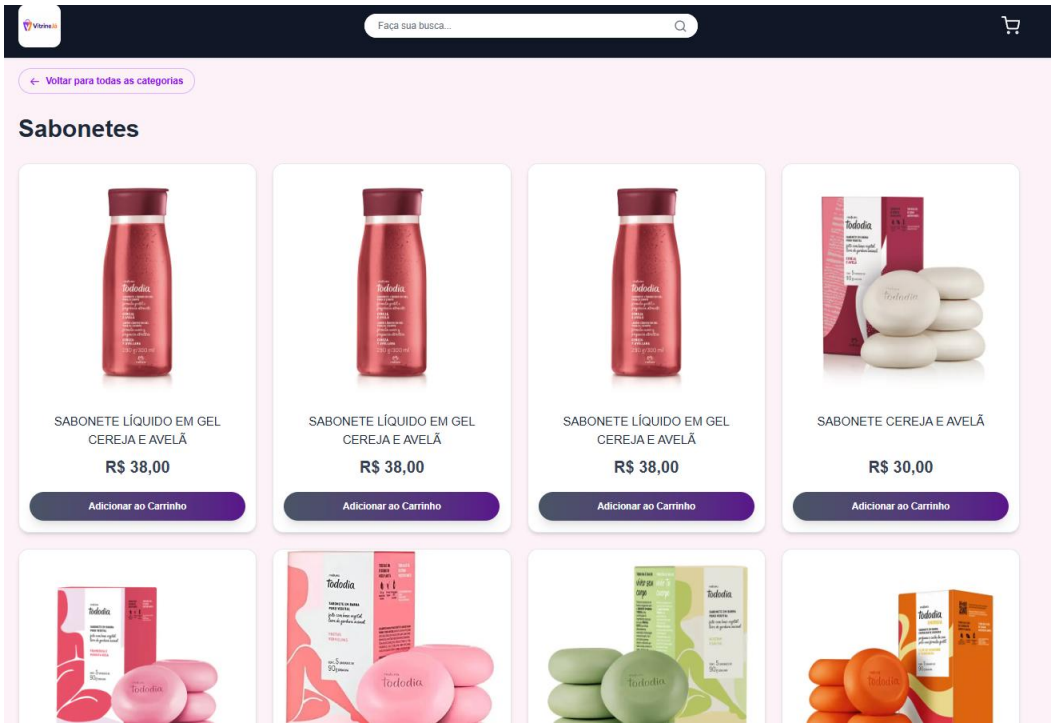
- **Barra de Navegação Superior (*Header*):** Componente estático que apresenta a identidade visual da plataforma, o campo de entrada textual de buscas (*SearchBar*) e o atalho responsivo para o carrinho de compras com contador volumétrico reativo baseado no *CartContext*. A barra de pesquisa dispara rotas que fazem a correspondência léxica com a coluna *search_text* da tabela Produto.

- **Seção Navegue por Categorias:** Listagem de distribuição horizontal que renderiza os registros persistidos na tabela Categoria. Cada item exibe dinamicamente o campo nome (ex: Perfumaria, Sabonetes, Hidratantes) e a respectiva imagem_ilustrativa moldada em formato circular (*avatar viewport*).
- **Seção Produtos em Destaque:** Grade de cards assíncronos alimentada via requisição HTTP que filtra no banco de dados os registros.

4.6.1.2 Tela de listagem por categoria selecionada

Ao interagir com o carrossel ou grade de categorias, a aplicação redireciona o fluxo de navegação para uma rota parametrizada baseada no identificador correspondente.

Figura 9 - Interface de Filtragem por categorias



Fonte: Elaborado pelo autor (2026)

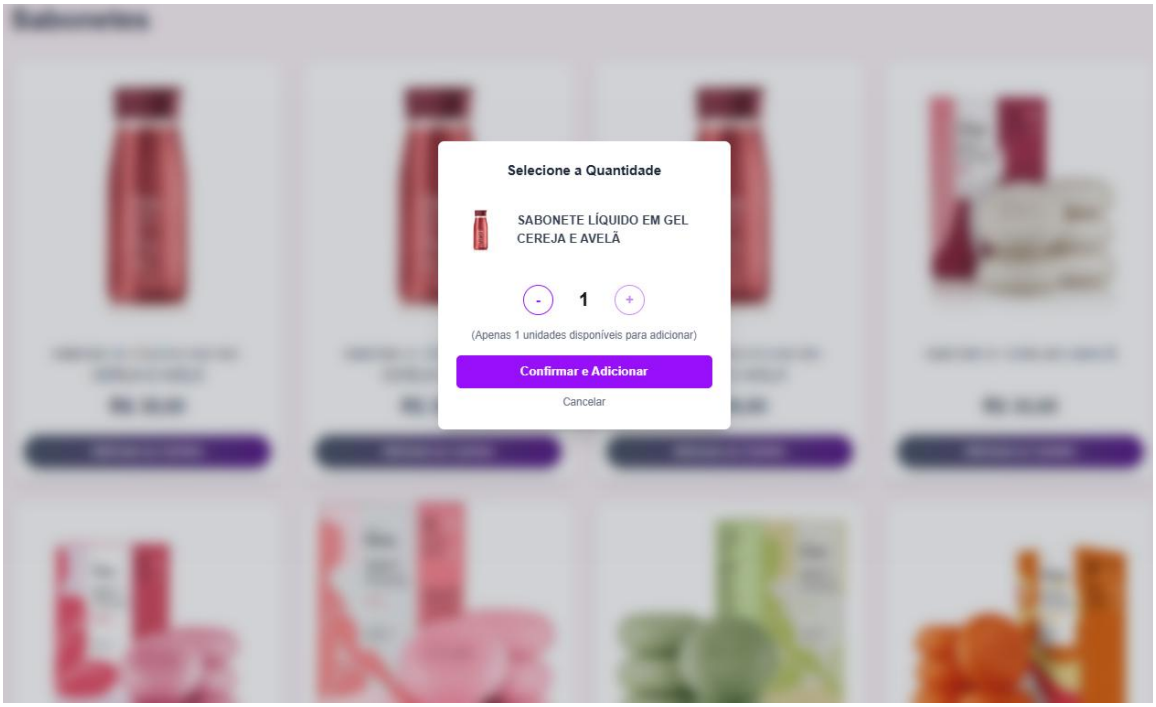
- **Controle de Retorno:** Um botão funcional permite a limpeza de estado e o retorno seguro à rota mãe do sistema.
- **Grade Específica de Catálogo:** Interface acionada pelo *endpoint*. O sistema atualiza o título dinâmico da página com o nome da categoria recuperada e

renderiza os componentes. Cada card exibe o nome do produto, o campo preço devidamente tratado com máscara monetária local (R$) e o gatilho "Adicionar ao Carrinho".

4.6.1.3 Modal de ajuste volumétrico de item

Componente interceptor projetado para refinar a intenção de compra antes da inserção definitiva dos dados na sessão local da aplicação.

Figura 10 - Modal informando quantidade de itens no estoque



Fonte: Elaborado pelo autor (2026)

- **Feedback de Disponibilidade:** O modal exibe de forma clara um descritivo informando o limite superior baseado no campo estoque (ex: "Apenas 1 unidades disponíveis para adicionar"), impedindo que o cliente adicione volumes indisponíveis no estoque real do lojista.
- **Controles Incrementais:** Botões reativos de soma e subtração ajustam o estado local do contador numérico. O clique no botão "Confirmar e Adicionar" serializa o snapshot do produto e envia ao estado global controlado pela API de Contexto.

4.6.1.4 Tela detalhada do produto

Acessada quando o consumidor clica diretamente no card do produto para obter informações detalhadas e dados institucionais do lojista proprietário.

Figura 11 - Interface detalhada dos itens

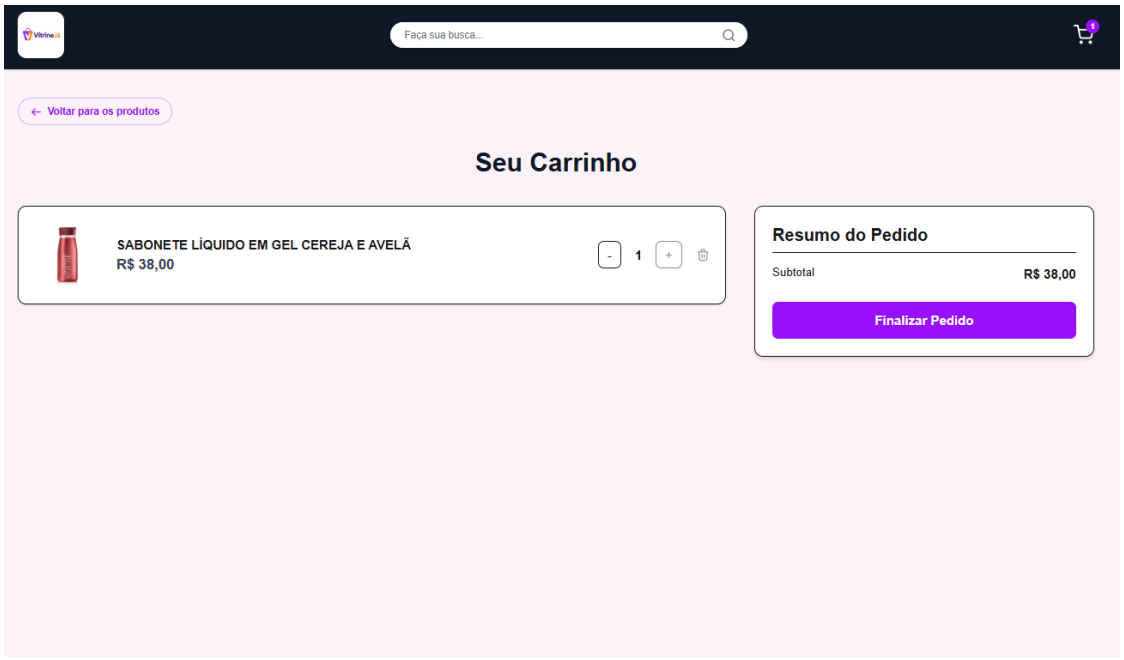
Fonte: Elaborado pelo autor (2026)

- **Detalhamento de Propriedade:** O painel cinza-claro em destaque expõe as informações recuperadas diretamente da tabela vinculada vendedores, exibindo o Nome Fantasia do comerciante e o canal de contato associado ao registro.
- **Tratamento de Exceções de Estoque:** Caso a coluna estoque da tabela Produto seja igual a zero no banco de dados, o botão principal da interface é modificado e bloqueado nativamente pela regra de renderização condicional do *React*, exibindo o rótulo desativado "Sem Estoque", o que blinda o sistema contra pedidos inconsistentes.

4.6.1.5 Interface do carrinho de compras ativo

Ambiente centralizador que consolida todos os itens selecionados pelo consumidor antes da fase de faturamento e fechamento de pedidos.

Figura 12 - Interface do carrinho com os pedidos



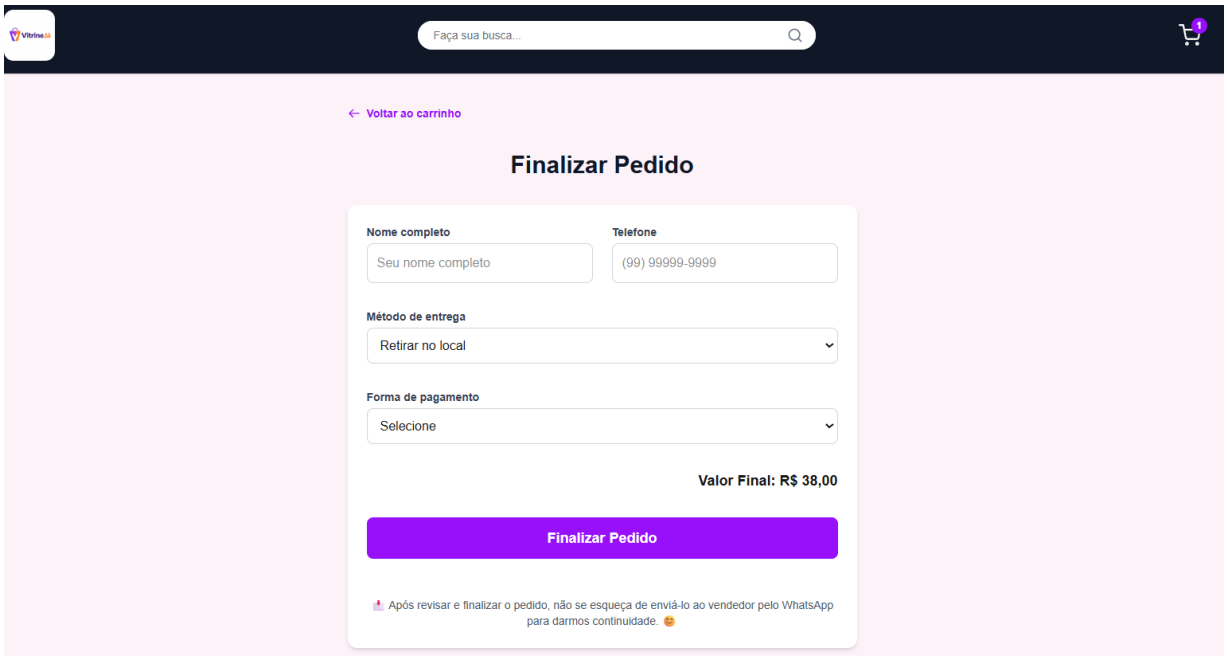
Fonte: Elaborado pelo autor (2026)

- **Listagem de Linhas de Registro:** Renderiza individualmente cada produto armazenado no *array* global. O cliente pode alterar a quantidade ou ejetar o item do carrinho clicando no ícone da lixeira, disparando eventos síncronos que recalculam o subtotal da operação.
- **Card de Resumo Financeiro:** Painel fixado à direita que soma os valores dos itens multiplicados por suas respectivas quantidades, exibindo de forma clara o valor bruto (subtotal) a ser transferido para a próxima tela por meio do botão "Finalizar Pedido".

4.6.1.6 Formulário de checkout e dados de entrega

Interface de coleta de dados que prepara o *payload* do cliente final para a geração do pedido físico e envio logístico.

Figura 13 - Formulário de coleta e entrega de dados



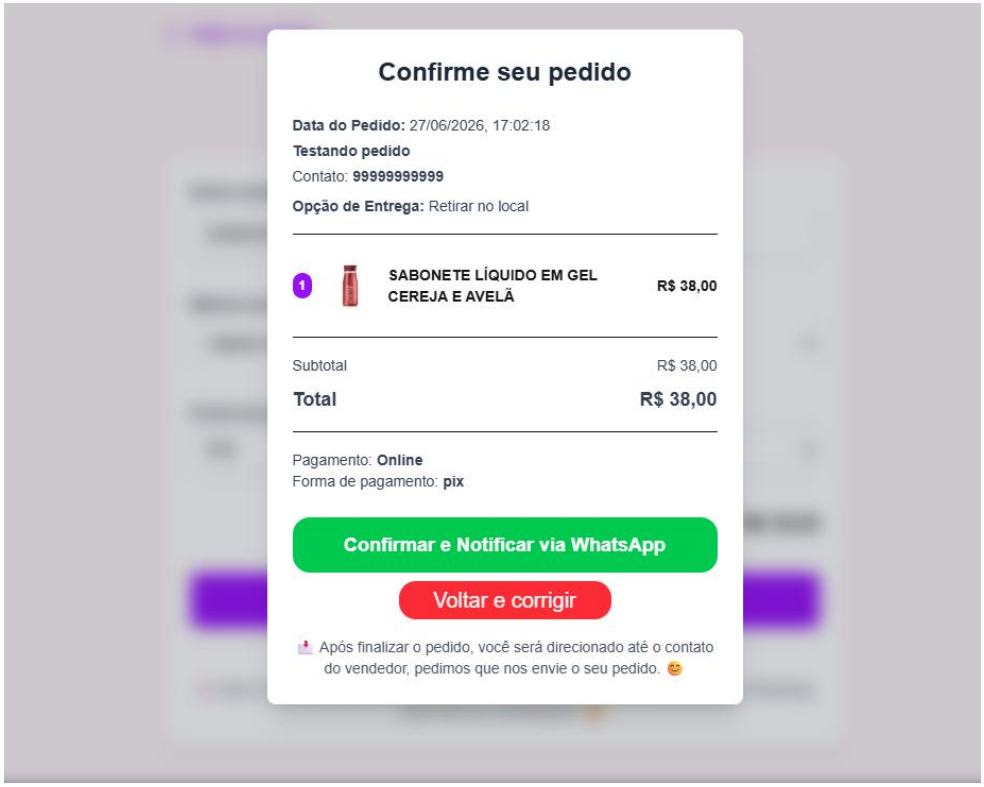
Fonte: Elaborado pelo autor (2026)

- **Campos de Entrada Estruturados:** Coleta o nome completo do comprador e o telefone de contato com máscaras de validação *Regex*.
- **Seletores de Regras Logísticas:** Caixas de seleção suspensas estruturadas para alimentar as colunas metodo_entrega (ex: Retirada no local, Entrega a domicílio) e forma_pagamento (ex: Pix, Dinheiro, Cartão), atualizando em tempo real o valor final do pedido na base do formulário.

4.6.1.7 Modal de confirmação final e integração com Whatsapp

Etapa crucial que realiza a persistência atômica no banco de dados PostgreSQL e orquestra o redirecionamento externo (*deep linking*).

Figura 14 - Modal de validação do pedido



Fonte: Elaborado pelo autor (2026)

- **Consolidação de Dados:** Exibe o laudo completo do pedido com a estampa de data e hora coletada no momento exato do encerramento da jornada.
- **Gatilho Transacional ("Confirmar e Notificar via WhatsApp"):** Ao acionar este botão, a aplicação *frontend* executa duas ações coordenadas de engenharia:
    - Efetua uma requisição HTTP POST para a rota do *backend* /pedidos, salvando na tabela Pedidos os dados do cliente, os totais financeiros e a *string* estruturada em formato JSONB dentro do campo itens_pedido.
    - Gera uma *string* de texto codificada para URL (*URL Encoding*) contendo o resumo legível do pedido e executa a função de redirecionamento nativa enviando o cliente para o contato de *whatsapp* do vendedor.

4.6.2 Interfaces do painel administrativo

O acesso a essas telas é estritamente condicionado à autenticação prévia do usuário, cujas rotas no *frontend* são protegidas por verificações de estado e de validação de *tokens JWT* no *backend*. A interface foi estruturada a partir de um design modular e limpo, focado em painéis (*dashboards*) operacionais. Esse arranjo permite que o lojista gerencie seu perfil institucional, controle o inventário físico de mercadorias em tempo real e faça a triagem transacional dos pedidos recebidos sem a necessidade de conhecimentos técnicos avançados em bancos de dados.

4.6.2.1 Menu principal / painel administrativo geral

A interface inicial do ambiente restrito atua como o centro de comando do usuário autenticado (tipo = 'vendedor' ou tipo = 'admin'), organizando o acesso aos módulos operacionais por meio de uma arquitetura limpa de cartões (*dashboard cards*).

Figura 15 - Página inicial do painel

Fonte: Elaborado pelo autor (2026)

- **Arquitetura Modular Grid:** Organiza as permissões de acesso em blocos simétricos. Cada botão direciona o operador para uma sub-rota específica monitorada por validação de *token* de sessão:
    - Pedidos Recebidos: Acesso à triagem transacional;
    - Categorias: Gerenciamento taxonômico do sistema;
    - Produtos / Controle de Estoque: Manutenção do catálogo ativo;

4.6.2.2 Interface de monitoramento de pedidos recebidos

Esta tela gerencia as vendas consolidadas e o fluxo de atendimento logístico, servindo como a interface de consumo dos dados transacionais persistidos na tabela Pedidos. Sob a ótica de Pressman e Maxim (2021), sistemas que manipulam registros comerciais exigem exibições sumarizadas que permitam inspeções detalhadas sob demanda, garantindo a rastreabilidade do processo sem sobrecarregar a tela principal.

Figura 16 - Interface dos pedidos recebidos

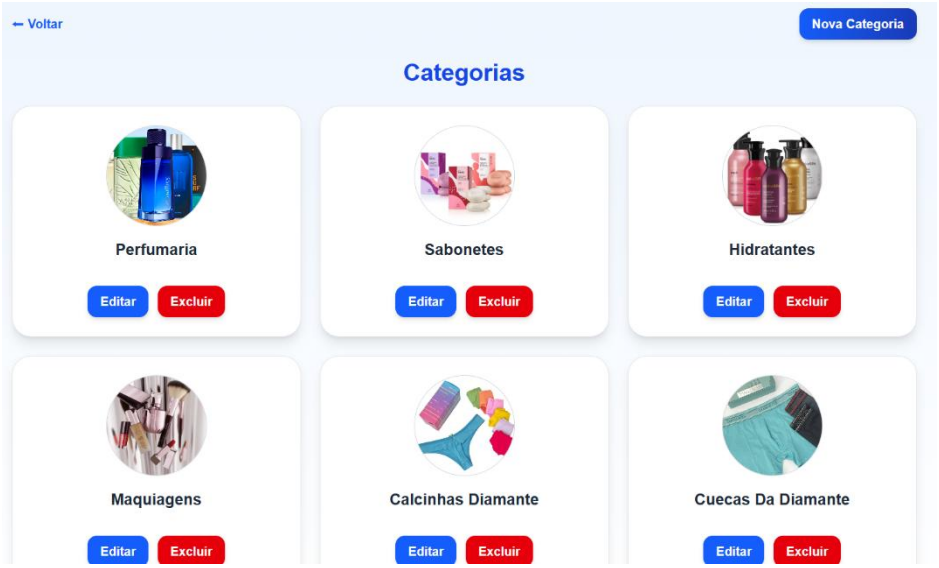
Fonte: Elaborado pelo autor (2026)

- **Listagem Transacional Completa:** Renderiza os registros vindos do *endpoint* /pedidos. Apresenta cabeçalhos contendo o número incremental do pedido (ex: Pedido #12), nome do cliente, data da transação e o valor total acumulado.
- **Componente Modal "Ver Detalhes":** Interceptor visual acionado dinamicamente. Ele realiza o mapeamento e a leitura do campo semiestruturado direto do banco de dados, quebrando o objeto binário na tela para exibir de forma legível a quantidade e a discriminação de cada produto, além do método de entrega e a forma de pagamento selecionada (pix).
- **Sinalizador de Status Reativo:** Exibe etiquetas visuais estilizadas via *Tailwind CSS* baseadas na coluna status da tabela, auxiliando no controle de fluxo de entrega.

4.6.2.3 Interface de gestão de categorias globais

Esta interface dá suporte ao gerenciamento taxonômico do sistema, permitindo que os administradores organizem a árvore de produtos que alimenta a vitrine pública. Segundo Sommerville (2011), interfaces voltadas ao gerenciamento de entidades estruturadas devem fornecer *feedback* claro sobre ações que possam impactar o relacionamento entre tabelas no banco de dados.

Figura 17 - Interface contendo as categorias criadas



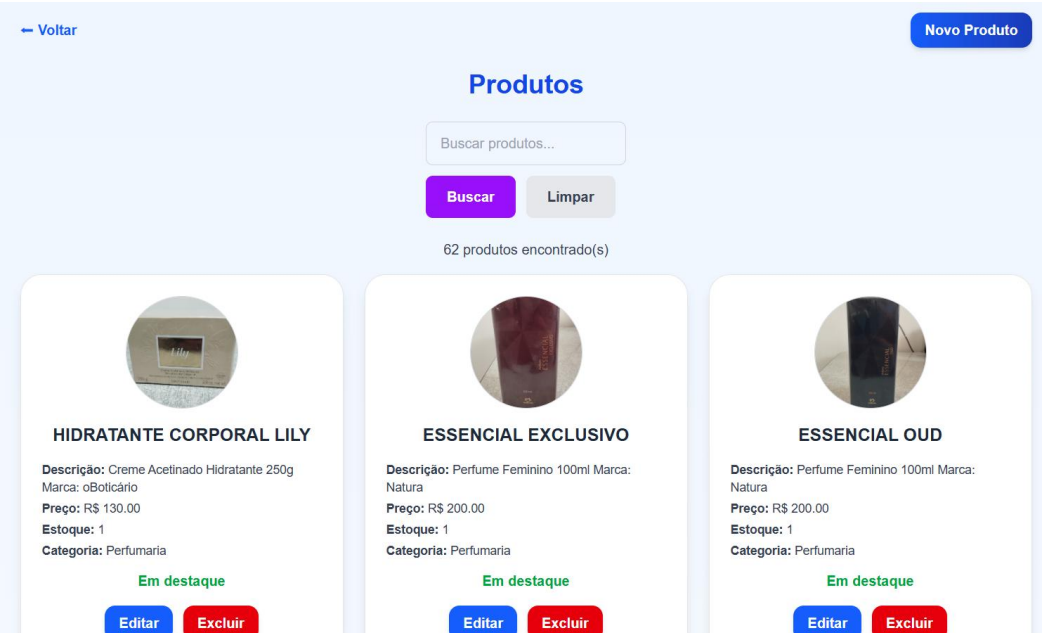
Fonte: Elaborado pelo autor (2026)

- **Ação de Escrita ("Nova Categoria"):** Gatilho posicionado no topo direito que invoca um formulário de inserção para alimentar a tabela Categoria.
- **Mesa de Operações (*CRUD Matrix*):** Cada categoria cadastrada exibe sua respectiva imagem em miniatura e título. Na base de cada bloco, existem botões de ação imediata:
    - Editar: Abre o escopo de atualização do registro correspondente;
    - Excluir: Dispara uma requisição *HTTP DELETE* que remove a linha no banco de dados, desde que respeitadas as restrições de integridade referencial com a tabela de produtos.

4.6.2.4 Manutenção do catálogo de produtos

O painel de produtos concentra a manutenção do catálogo visível aos clientes, servindo de interface para o controle de preços, descrições, marcas e imagens processadas por serviços de nuvem. Conforme apontado por Marco Tulio Valente (2024), painéis de gerenciamento de inventário moderno demandam componentes de filtragem ágeis e indicadores de estado para otimizar operações em catálogos com alto volume de registros.

Figura 18 - Interface com todos os produtos listados



Fonte: Elaborado pelo autor (2026)

- **Indicador de Volumetria:** Exibe no topo a contagem dinâmica de itens retornados pela consulta SQL (62 produtos encontrado(s)).
- **Filtro Avançado de Busca Interna:** Permite ao lojista filtrar sua lista de forma ágil através de entradas lógicas de texto.
- **Ficha de Ações do Item:** Cada *card* exibe de forma detalhada o preço configurado, estoque, marca, categoria e um indicador condicional verde de status ("Em destaque"), que reflete se a coluna destaque está marcada como verdadeira. Os botões "Editar" e "Excluir" gerenciam os ciclos de alteração e deleção lógica do catálogo.

4.6.2.5 Módulo de controle rápido de estoque

Diferente das interfaces anteriores focadas no detalhamento de dados, este módulo foi projetado especificamente para operações de alta rotatividade. O design de interface prioriza tarefas repetitivas focadas em um único atributo numérico, aplicando o conceito de design focado na produtividade para mitigar erros operacionais de digitação.

Figura 19 - Controle de estoque

Fonte: Elaborado pelo autor (2026)

- **Layout em Linha Simples (*Row Layout*):** Diferente da visualização em grade, este módulo adota linhas limpas com foco exclusivo no campo numérico estoque.
- **Gatilhos Incrementais Diretos:** À direita de cada registro, encontram-se botões reativos de decremento e incremento (- e +). Ao clicar nestes botões, a aplicação *frontend* efetua chamadas assíncronas em segundo plano do tipo *HTTP PATCH* para atualizar imediatamente o saldo de estoque da ID correspondente no *PostgreSQL*, sem a necessidade de recarregar toda a interface de trabalho.

5 SOFTWARE

Este capítulo apresenta como o sistema foi colocado em funcionamento no ambiente real e detalha as etapas de testes realizadas no *software*. O objetivo dessas fases foi localizar possíveis falhas e avaliar o uso das interfaces, garantindo que a plataforma opere com segurança, qualidade e eficiência para os usuários.

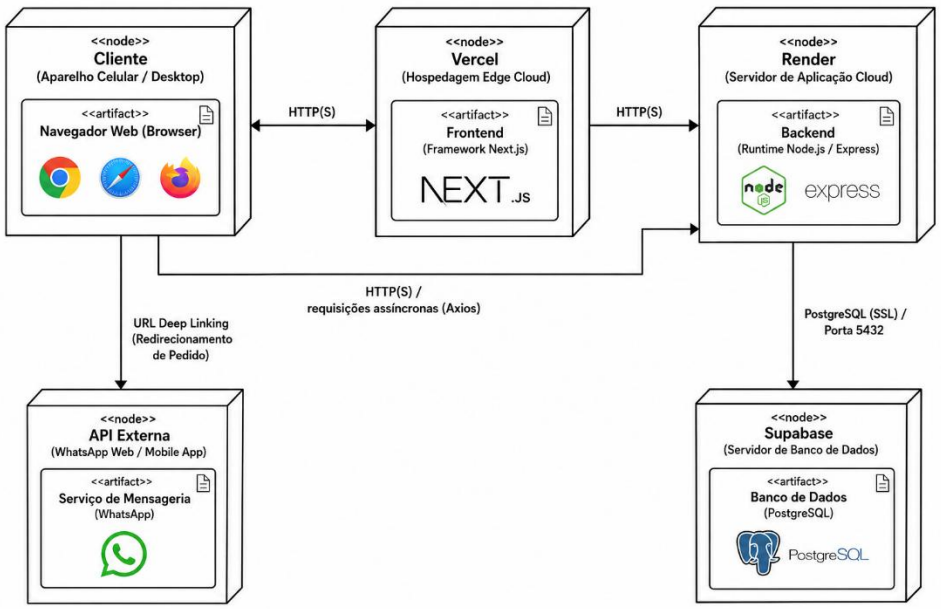
5.1 Implantação do sistema

A distribuição física e a entrada em produção do ecossistema foram conduzidas por meio de uma estratégia de Implantação Piloto de caráter gradual. Conforme preconiza Larman (2007), essa abordagem estratégica consiste na liberação controlada do sistema para um subconjunto restrito de usuários e ambientes reais, funcionando como uma fase de amortecimento de riscos operacionais antes da transição global definitiva.

Na prática, a arquitetura foi disponibilizada inicialmente em estabelecimentos parceiros com menor volume de movimentação diária, o que permitiu monitorar o comportamento do *software* sob condições reais de rede sem comprometer fluxos comerciais de alta criticidade. A metodologia envolveu a abordagem direta e presencial dos consumidores logo após o encerramento de seus atendimentos convencionais na loja física. Os usuários foram convidados a interagir com o catálogo digital e a simular a jornada completa de montagem de carrinho e emissão de intenção de compra. Esse monitoramento assistido forneceu dados qualitativos imediatos sobre a aderência das interfaces, o tempo de resposta das requisições e a intuitividade dos fluxos projetados.

Para representar a topologia física dos componentes de *hardware* e os artefatos de *software* que dão sustentação ao ecossistema em ambiente de produção, elaborou-se o Diagrama de Implantação baseado na especificação UML, conforme detalhado na Figura 20.

Figura 20 - Diagrama de Implantação



Fonte: Elaborado pelo autor (2026)

5.2 Testes

A fase de testes constitui um elemento crítico da Engenharia de *Software* para assegurar que a aplicação atenda aos requisitos funcionais levantados e opere livre de anomalias críticas. Segundo Pressman e Maxim (2021), a validação de um sistema deve seguir uma abordagem estruturada, confrontando as entradas fornecidas com as respostas geradas pela aplicação.

Para a validação da plataforma, utilizou-se a metodologia de **Teste de Caixa Preta**, concentrando-se nos fluxos de entrada e saída das interfaces, na integridade das rotas protegidas pelo *backend* e no comportamento da persistência de dados no *PostgreSQL*. Dessa forma, o confronto sistemático entre as entradas fornecidas nas interfaces e as saídas lógicas geradas no banco de dados valida os limites operacionais da aplicação. Essa abordagem de verificação funcional blinda o

ecossistema contra anomalias severas e atende aos critérios de garantia de qualidade de software defendidos por Rios e Moreira (2020). O quadro 7 descreve o plano de testes executado e os resultados observados.

Quadro 7 - Matriz de Testes Funcionais do Sistema

ID	Cenário de teste	Procedimento executado	Resultado esperado	Resultado obtido	Status
CT01	Autenticação de Usuário	Inserir credenciais válidas no formulário de login do painel administrativo.	Geração do *token JWT*, armazenamento no cliente e redirecionamento para o dashboard.	*Token* gerado e armazenado com sucesso; redirecionamento imediato.	Aprovado
CT02	Proteção de Rota Privada	Tentar acessar o *endpoint* /pedidos diretamente via URL sem *token* de autenticação.	Bloqueio de acesso pelo *middleware* do *Express* e retorno do código *HTTP* 401 *Unauthorized*.	Acesso negado com retorno do status de segurança esperado.	Aprovado
CT03	Consumo do Catálogo	Abrir a *Home Page* pública e interagir com o componente de categorias.	Disparo de requisição assíncrona e renderização imediata dos *cards* correspondentes.	Interface atualizada dinamicamente via requisição à *API*.	Aprovado
CT04	Validação de Estoque	Tentar adicionar ao carrinho um produto cuja coluna estoque seja igual a zero.	Bloqueio do gatilho na interface e exibição visual do botão desativado "Sem Estoque".	Botão desabilitado pelo *React* conforme regra condicional.	Aprovado
CT05	Persistência e Integração	Finalizar a coleta de dados no *checkout* e	Inserção do *snapshot* no banco (*JSONB*), registro do	Registro efetuado com sucesso no *Supabase* e	Aprovado

		acionar a confirmação do pedido.	pedido e redirecionamento para o *WhatsApp* do lojista.	link do *WhatsApp* gerado com *payload* correto.	

Elaborado pelo autor (2026)

6 CONSIDERAÇÕES FINAIS

O desenvolvimento e a implementação da plataforma cumpriram integralmente o objetivo de conceber uma solução de catálogo digital e retaguarda operacional voltada ao fomento e à transformação digital do comércio local de proximidade. Através da adoção de uma arquitetura desacoplada (*headless architecture*), unindo o ecossistema reativo do *Next.js* no *frontend* à resiliência de uma *API RESTful* estruturada em *Node.js* e *Express* no *backend*, o projeto demonstrou viabilidade técnica na construção de um ecossistema estável, performático e de baixo custo de manutenção.

A decisão de engenharia de desatrelar a plataforma de *gateways* ou intermediários de pagamento tradicionais provou-se o maior acerto estratégico e arquitetural para o público-alvo do sistema. Em vez de onerar o microempreendedor local com taxas transacionais complexas ou retocar fluxos burocráticos de saques, o sistema opera como um facilitador de conexões: o banco de dados *PostgreSQL* (via *Supabase*) cumpre o papel de persistir o snapshot imutável do pedido em formato JSONB para fins de auditoria e controle do lojista, enquanto a interface do cliente delega e transfere a fase final de faturamento, combinação de frete e liquidação financeira para o canal de comunicação direta que ambos os atores (vendedor e consumidor) já dominam e utilizam diariamente: o *WhatsApp*. Esse modelo preserva a autonomia do lojista, que assume o controle total do pedido dali em diante, replicando a dinâmica flexível do comércio de rua no ambiente digital.

6.1 Limitações do projeto

Embora o ecossistema tenha sido homologado com sucesso, o ciclo de desenvolvimento enfrentou restrições técnicas e de infraestrutura inerentes ao escopo atual da aplicação. A primeira limitação estrutural reside no desacoplamento transacional externo. Como a plataforma não processa nativamente a liquidação

financeira — seja por cartão, Pix ou boleto — e não se integra a *APIs* de rastreio logístico terceirizadas, como a dos Correios ou operadoras de frete, ocorre uma perda na rastreabilidade do ciclo de vida do pedido logo após o disparo para o WhatsApp. Consequentemente, o fechamento e a confirmação de entrega passam a depender inteiramente do preenchimento e da atualização manual de status por parte do lojista dentro do painel administrativo.

Ademais, a escolha pela infraestrutura de hospedagem gratuita para a publicação da API de *backend*, utilizando as instâncias de nível *free tier* da plataforma *Render*, introduziu a limitação técnica conhecida como inicialização fria (*cold start*). Devido a essa configuração, após períodos de inatividade, o servidor entra automaticamente em estado de hibernação. Esse comportamento faz com que a primeira requisição *HTTP* disparada pelo *frontend* sofra um atraso (*delay*) perceptível até que ocorra o provisionamento e a inicialização completa do contêiner em nuvem.

Por fim, manifesta-se uma dependência estrita em relação a *APIs* de terceiros, uma vez que a entrega final da intenção de compra está totalmente condicionada à disponibilidade e à estabilidade da *API* pública de links do *WhatsApp*. Sob essa ótica, quaisquer instabilidades na infraestrutura de rede da empresa *Meta* ou eventuais alterações unilaterais na política de redirecionamento dinâmico (*deep linking*) geram impactos diretos no fluxo principal de encerramento de pedidos do *software*.

6.2 Trabalhos futuros e novas funcionalidades

Para ciclos futuros de evolução, e visando transformar a plataforma em um produto comercial escalável a nível regional, propõe-se uma série de implementações e melhorias estruturais no ecossistema do sistema. A primeira evolução planejada consiste na introdução de um módulo de mensagens em tempo real baseado em *Websockets*. O objetivo é substituir o modelo atual de requisições repetidas por conexões persistentes via *Socket.io* no *backend*, permitindo que o painel administrativo do lojista receba alertas sonoros e notificações visuais instantâneas na tela no exato momento em que um cliente persistir um pedido na tabela do banco de dados, eliminando completamente a necessidade de atualizar a página.

Em paralelo, projeta-se o desenvolvimento de um módulo focado em Inteligência de Negócios de forma integrada ao *dashboard* do vendedor. Essa funcionalidade consumirá os dados agregados de vendas por períodos específicos.

REFERÊNCIAS

COSTA, Luan Barbosa da; SILVA, Luan Vieira da; NUNES, Leonardo Silva. **Estudo de caso:** o impacto do ecossistema Next.js na produtividade e desenvolvimento de sistema Needuk. Lumen et Virtus, v. 17, n. 61, p. 112-125, jan. 2026.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. 162 f. Tese (Doutorado em Ciência da Computação) - University of California, Irvine, 2000.

GARRETT, Jesse James. **The Elements of User Experience**: User-Centered Design for the Web and Beyond. 2. ed. Berkeley: New Riders, 2010.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. 7. ed. São Paulo: Atlas, 2022.

KOTONYA, Gerald; SOMMERVILLE, Ian. **Requirements Engineering**: Processes and Techniques. Chichester: John Wiley & Sons, 1998.

LARMAN, Craig. **Utilizando UML e Padrões**: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

MORAES, William Bruno. **Construindo aplicações com NodeJS**. 4. ed. São Paulo: Novatec Editora, 2023.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software**: uma abordagem profissional. 9. ed. Porto Alegre: AMGH, 2021.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de software**. 3. ed. Rio de Janeiro: Alta Books, 2020.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL Distilled**: a brief guide to the emerging world of polyglot persistence. Upper Saddle River: Pearson Education, 2013.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Addison-Wesley, 2011.

STALLINGS, William. **Cryptography and Network Security**: Principles and Practice. 6. ed. Boston: Pearson, 2014.

TILKOV, S.; VINOSKI, S. **Node.js in Action**. Greenwich, CT: Manning Publications, 2010.

VALENTE, Marco Tulio. **Engenharia de Software Moderna:** Princípios e Práticas para o Desenvolvimento de Software com Produtividade. Belo Horizonte: Edição do Autor, 2020.
Disponível em: https://engsoftmoderna.info/. Acesso em: 15 mar. 2026.

WIEGERS, Karl; BEATTY, Joy. **Software Requirements**. 3. ed. Redmond: Microsoft Press, 2013.