PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FASHIONBOOST: SISTEMA DE GESTÃO E FIDELIZAÇÃO PARA LOJAS DE MODA

KALLEL BRAGA BEZERRA

GOIÂNIA
2026

KALLEL BRAGA BEZERRA

FASHIONBOOST: SISTEMA DE GESTÃO E FIDELIZAÇÃO PARA LOJAS DE MODA

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes, do Curso de Ciência da Computação, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Ernesto Fonseca Veiga

GOIÂNIA
2026

AGRADECIMENTOS

Primeiramente, gostaria de expressar minha gratidão à minha família, pelo afeto, cuidado, apoio e motivação em todos os momentos. Cada realização ao longo da minha jornada, seja na faculdade ou fora dela, só foi viável devido ao suporte contínuo e à confiança que sempre tiveram em mim.

Também quero agradecer ao meu orientador, Prof. Ernesto Fonseca Veiga, pela orientação cuidadosa, pelas valiosas contribuições e pela dedicação em cada fase deste projeto.

Sou grato ao corpo docente da Pontifícia Universidade Católica de Goiás, que, com dedicação e compromisso, ofereceu não apenas conhecimento técnico, mas igualmente valores éticos e humanos que levarei comigo em minha carreira profissional.

Por último, faço um agradecimento especial aos meus amigos e colegas de turma, pela parceria, compreensão e incentivo durante toda a trajetória acadêmica. O apoio e a presença de cada um foram fundamentais para tornar esta experiência mais leve.

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de um sistema SaaS (*Software as a Service*) voltado para proprietários de lojas de moda de pequeno e médio porte, denominado *FashionBoost*. A proposta surge da constatação de que pequenos lojistas do setor de moda, em geral, não dispõem de acesso a ferramentas acessíveis de gestão e relacionamento com clientes, como sistemas de CRM e programas de fidelização. O sistema foi desenvolvido com NestJS e TypeScript no *back-end*, TypeORM e PostgreSQL na camada de persistência, e Next.js 15 com Tailwind CSS v4 no *front-end*. A arquitetura adotada segue o modelo multi-tenant, no qual cada lojista opera em um ambiente completamente isolado dentro da mesma plataforma. Entre as funcionalidades implementadas destacam-se: gestão de produtos, estoque e categorias; registro de vendas com aplicação de cupons de desconto; programa de fidelidade com acúmulo de pontos e níveis automáticos; e um painel de inteligência artificial integrado à API Groq com o modelo Llama 3.3 70B, capaz de analisar os dados da loja e sugerir ações estratégicas categorizadas. A prototipação inicial da interface foi conduzida com o auxílio da plataforma Lovable, ferramenta de geração de interfaces por inteligência artificial. Os resultados obtidos demonstram que o *FashionBoost* é capaz de apoiar a gestão operacional e a fidelização de clientes em lojas independentes de moda, respondendo à questão de pesquisa proposta com uma solução técnica viável e acessível.

Palavras-chave: Sistemas Web. SaaS. Gestão de Lojas. Programa de Fidelidade. Inteligência Artificial. NestJS. Next.js.

ABSTRACT

This work aims to develop a SaaS (Software as a Service) system for small and medium-sized fashion store owners, called *FashionBoost*. The proposal arises from the observation that small retailers in the fashion sector generally do not have access to affordable customer management and loyalty tools, such as CRM systems and loyalty programs. The system was developed with NestJS and TypeScript on the back-end, TypeORM and PostgreSQL on the persistence layer, and Next.js 15 with Tailwind CSS v4 on the front-end. The adopted architecture follows the multi-tenant model, in which each retailer operates in a completely isolated environment within the same platform. Key implemented features include: product, inventory and category management; sales recording with discount coupon application; a loyalty program with point accumulation and automatic tier progression; and an artificial intelligence dashboard integrated with the Groq API using the Llama 3.3 70B model. The results demonstrate that *FashionBoost* is capable of supporting operational management and customer loyalty in independent fashion stores, answering the proposed research question with a viable and accessible technical solution.

Keywords: Web Systems. SaaS. Store Management. Loyalty Program. Artificial Intelligence. NestJS. Next.js.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
DOM	Document Object Model
ERP	Enterprise Resource Planning
HTML	HyperText Markup Language
IA	Inteligência Artificial
JSON	JavaScript Object Notation
JWT	JSON Web Token
LGPD	Lei Geral de Proteção de Dados
LLM	Large Language Model
MER	Modelo Entidade-Relacionamento
MVCC	Multi-Version Concurrency Control
NF	Requisito Não Funcional
ORM	Object-Relational Mapping
POS	Point of Sale
REST	Representational State Transfer
RF	Requisito Funcional
RN	Regra de Negócio
SaaS	Software as a Service
SGBD	Sistema Gerenciador de Banco de Dados

SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SRS	Software Requirements Specification
SSG	Static Site Generation
SSR	Server-Side Rendering
UI	User Interface
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
UX	User Experience

SUMÁRIO

1 INTRODUÇÃO

O setor de moda no varejo brasileiro é composto majoritariamente por pequenos e médios empreendedores que conduzem suas operações com recursos limitados e acesso restrito a ferramentas tecnológicas de gestão (SEBRAE, 2024). Ao contrário das grandes redes varejistas, que dispõem de sistemas integrados de gestão empresarial, plataformas de CRM (*Customer Relationship Management*) e programas de fidelização sofisticados, os lojistas independentes frequentemente dependem de processos manuais ou de soluções genéricas que não atendem às especificidades do segmento de moda.

Esse cenário representa um gargalo competitivo relevante. A ausência de ferramentas de controle de estoque atualizadas em tempo real, a dificuldade em registrar e analisar o comportamento de compra dos clientes e a falta de mecanismos automáticos de fidelização são fatores que comprometem tanto a eficiência operacional quanto a capacidade de retenção de clientes nesses estabelecimentos. Estudos do Sebrae apontam que a mortalidade de pequenas empresas do varejo está fortemente associada à ausência de processos organizados e ao uso insuficiente de tecnologia na gestão (SEBRAE, 2024).

Paralelamente, o mercado de tecnologia como serviço (*Software as a Service* — SaaS) tem se consolidado como um modelo de distribuição de software especialmente adequado para pequenos negócios, por eliminar a necessidade de infraestrutura própria, reduzir custos de implantação e permitir acesso via navegador a qualquer momento. Nesse contexto, uma plataforma SaaS voltada especificamente para o varejo de moda de pequeno porte pode representar um instrumento relevante de modernização e profissionalização desses negócios.

Diante desse contexto, este Trabalho de Conclusão de Curso propõe o desenvolvimento do *FashionBoost*, um sistema SaaS destinado a proprietários de lojas de moda de pequeno e médio porte. A plataforma centraliza as principais operações da loja — gestão de produtos, controle de estoque, registro de vendas, aplicação de cupons de desconto e programa de fidelidade com níveis automáticos — em um único ambiente. Para que múltiplos lojistas possam utilizar a mesma infraestrutura de software sem que os dados de um interfiram nos dados de outro, o *FashionBoost* adota o modelo de multi-tenancy: uma arquitetura na qual uma única instância da aplicação atende a vários clientes simultaneamente,

mantendo o isolamento lógico completo entre eles. Cada lojista, ao se cadastrar, passa a operar em seu próprio espaço isolado dentro da plataforma — com seus produtos, clientes, vendas e configurações completamente separados dos demais —, sem a necessidade de servidores ou bancos de dados dedicados para cada um (BEZEMER; ZAIDMAN, 2010). Além disso, o sistema conta com um painel de inteligência artificial que analisa os dados da loja e sugere ações estratégicas nas áreas de promoções, reposição de estoque, fidelização e operações.

O código-fonte completo do *FashionBoost* está disponível publicamente no repositório: https://github.com/KallelBrg/FashionBoost .

Questão de pesquisa: Como o *FashionBoost*, enquanto um sistema SaaS, pode apoiar a gestão operacional e a fidelização de clientes em lojas de moda de pequeno e médio porte?

1.1 Justificativa

A justificativa para o desenvolvimento do *FashionBoost* fundamenta-se em duas constatações complementares. A primeira é a lacuna tecnológica observada no varejo de moda independente: ferramentas de CRM, programas de fidelização e sistemas de gestão integrada existem no mercado, mas em sua maioria são caras, complexas de implementar ou projetadas para grandes redes, tornando-se inacessíveis para lojistas de pequeno porte. A segunda constatação é que a fidelização de clientes representa um dos fatores mais determinantes para a sustentabilidade de pequenos negócios de varejo, dado que o custo de retenção de um cliente existente é significativamente menor do que o custo de aquisição de um novo cliente.

O *FashionBoost* surge, portanto, como uma resposta direta a essas necessidades, propondo uma solução acessível, de interface moderna e funcionalidades automatizadas — incluindo sugestões geradas por inteligência artificial — que até então estavam disponíveis apenas para grandes operações do varejo.

1.2 Objetivos

1.2.1 Geral

- Desenvolver uma plataforma SaaS multi-tenant para gestão operacional e fidelização de clientes voltada a proprietários de lojas de moda de pequeno e médio porte, com funcionalidades de inteligência artificial integradas.

1.2.2 Objetivos Específicos

- Realizar o levantamento e a especificação formal de requisitos do sistema seguindo o padrão IEEE para *Software Requirements Specification* (SRS);
- Implementar arquitetura multi-tenant com isolamento completo de dados por lojista;
- Desenvolver módulo de gestão completa da loja, contemplando cadastro de produtos, controle automático de estoque, categorias e clientes;
- Implementar fluxo completo de vendas com aplicação de cupons e reversão automática de estoque e pontos em caso de cancelamento;
- Criar programa de fidelidade com acúmulo de pontos configurável e progressão automática por níveis;
- Integrar painel de inteligência artificial com sugestões estratégicas categorizadas, utilizando o modelo Llama 3.3 70B via API Groq;
- Utilizar ferramentas modernas de prototipação e assistência baseadas em inteligência artificial para acelerar o desenvolvimento e a definição visual da interface.

1.3 Resultados Esperados

O principal resultado esperado é um protótipo funcional do *FashionBoost* que responda à questão de pesquisa proposta, demonstrando como um sistema SaaS pode apoiar concretamente a gestão operacional e a fidelização de clientes no contexto do varejo de moda independente. Espera-se que o sistema permita que o lojista realize, em um único ambiente, todas as operações essenciais do dia a dia, desde o cadastro de produtos até o acompanhamento do engajamento dos clientes no programa de fidelidade. A integração com

inteligência artificial representa um diferencial esperado, ao oferecer ao lojista sugestões automatizadas baseadas nos próprios dados da sua loja.

1.4 Organização do Trabalho

Este trabalho está organizado em seis capítulos. O presente capítulo introduz o tema, apresenta o problema abordado, a justificativa, os objetivos e os resultados esperados.

O Capítulo 2 apresenta a fundamentação teórica, abordando os conceitos e tecnologias que sustentam o desenvolvimento do *FashionBoost*, incluindo o modelo SaaS, multi-tenancy, programas de fidelidade, inteligência artificial aplicada ao varejo e as principais tecnologias utilizadas na implementação.

O Capítulo 3 descreve os materiais e procedimentos metodológicos, detalhando as ferramentas utilizadas, a abordagem de desenvolvimento adotada e os critérios seguidos para implementação e validação do sistema.

O Capítulo 4, denominado Desenvolvimento do Projeto, apresenta em detalhes o processo de construção do *FashionBoost*: levantamento de requisitos, modelagem do banco de dados com o Modelo de Entidade-Relacionamento completo, arquitetura da aplicação, decisões técnicas e implementação das principais funcionalidades.

O Capítulo 5 apresenta os resultados obtidos, respondendo à questão de pesquisa e comparando o *FashionBoost* com soluções relacionadas disponíveis no mercado.

Por fim, o Capítulo 6 traz as considerações finais, relacionando os resultados alcançados com os objetivos propostos e apresentando sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os fundamentos teóricos e tecnológicos que sustentam o desenvolvimento do *FashionBoost*, abordando desde o modelo de negócio SaaS e os conceitos de multi-tenancy e programas de fidelidade, até as linguagens, frameworks e ferramentas utilizadas na construção do sistema.

2.1 Modelo SaaS e Multi-tenancy

2.1.1 Software as a Service (SaaS)

O modelo SaaS (*Software as a Service*) representa uma forma de distribuição de software na qual a aplicação é hospedada e mantida pelo provedor e acessada pelo usuário final por meio de um navegador web ou aplicativo, mediante assinatura ou pagamento por uso. Diferentemente dos modelos tradicionais de licenciamento, o SaaS elimina a necessidade de instalação local, manutenção de infraestrutura própria e atualização manual de versões por parte do cliente (WARD; BARKER, 2014).

Essa abordagem é especialmente adequada para pequenos e médios negócios, pois reduz significativamente o custo de entrada na adoção de tecnologia. O lojista acessa a plataforma pelo navegador, sem precisar adquirir servidores ou contratar equipes de TI, o que democratiza o acesso a ferramentas que antes eram exclusivas de grandes organizações. No *FashionBoost*, o modelo SaaS se manifesta na possibilidade de qualquer lojista se cadastrar, criar sua loja e começar a operar imediatamente, sem qualquer configuração de infraestrutura.

2.1.2 Multi-tenancy

Multi-tenancy é o padrão arquitetural pelo qual uma única instância de um software atende múltiplos clientes — denominados tenants —, mantendo o isolamento lógico dos dados de cada um. Em uma arquitetura multi-tenant, todos os lojistas cadastrados no *FashionBoost* utilizam a mesma infraestrutura de servidor e banco de dados, mas cada tenant possui acesso exclusivo aos seus próprios dados — produtos, clientes, vendas e configurações — sem que haja qualquer intersecção com os dados de outros lojistas (BEZEMER; ZAIDMAN, 2010).

Existem diferentes estratégias para implementar o isolamento em uma arquitetura multi-tenant. Uma delas é a utilização de bancos de dados separados por cliente; outra é a utilização de schemas distintos dentro do mesmo banco; e uma terceira, adotada no *FashionBoost*, é a separação por linha — isto é, todas as entidades do banco de dados compartilham as mesmas tabelas, mas cada registro possui uma chave estrangeira referenciando o tenant ao qual pertence. Toda requisição autenticada carrega o identificador do tenant no token JWT, e todas as consultas são filtradas automaticamente por esse identificador, garantindo o isolamento de forma transparente para o usuário.

2.2 Programas de Fidelidade no Varejo

Programas de fidelidade são mecanismos estruturados de relacionamento com o cliente que têm como objetivo incentivar compras recorrentes por meio de recompensas acumuladas ao longo do tempo. No contexto do varejo, esses programas geralmente se baseiam na concessão de pontos a cada compra, que podem ser trocados por descontos, brindes ou outros benefícios, e na segmentação de clientes em níveis de fidelidade com vantagens progressivas (KOTLER; KELLER, 2018).

A lógica subjacente é a do custo diferencial de retenção: manter um cliente ativo tende a ser economicamente mais eficiente do que conquistar um novo. Ao oferecer benefícios crescentes para clientes mais engajados, o programa cria um incentivo para que eles concentrem suas compras em um único estabelecimento. Para pequenas lojas de moda, que muitas vezes dependem de uma base de clientes recorrentes para manter o faturamento, esse tipo de mecanismo pode ter impacto direto na sustentabilidade do negócio.

No *FashionBoost*, o programa de fidelidade é inteiramente configurável por loja: o lojista define a regra de acúmulo de pontos por produto e os limiares de pontos que determinam a progressão entre níveis. O sistema calcula e atribui os pontos automaticamente a cada venda confirmada e reverte o saldo em caso de cancelamento, mantendo a consistência dos dados de fidelidade sem necessidade de intervenção manual.

2.3 Inteligência Artificial Aplicada ao Varejo

A aplicação de inteligência artificial no varejo tem se expandido rapidamente, abrangendo desde a personalização de recomendações de produtos até a previsão de demanda

e a análise de comportamento do consumidor. No contexto de pequenos negócios, uma das aplicações mais acessíveis e impactantes é o uso de modelos de linguagem de grande escala (*Large Language Models* — LLMs) para análise de dados operacionais e geração de sugestões estratégicas em linguagem natural (KUMAR; REINARTZ, 2018).

2.3.1 Modelos de Linguagem de Grande Escala (LLMs)

Os LLMs são modelos de inteligência artificial treinados em grandes volumes de texto, capazes de compreender e gerar linguagem natural com alto grau de coerência e precisão contextual. Sua capacidade de processar contextos extensos e responder a instruções detalhadas em linguagem natural os torna adequados para tarefas de análise, síntese e geração de recomendações baseadas em dados estruturados (BROWN et al., 2020).

Uma característica fundamental dos LLMs para aplicações empresariais é a possibilidade de utilização via API, sem a necessidade de treinamento ou hospedagem do modelo por parte do desenvolvedor. Isso permite que sistemas como o *FashionBoost* integrem capacidades analíticas avançadas com custo e complexidade reduzidos.

2.3.2 API Groq e Modelo Llama 3.3 70B

O Groq é um provedor de inferência de LLMs que oferece acesso a modelos de linguagem de alto desempenho via API, com latências significativamente menores do que outras plataformas equivalentes, graças ao uso de hardware especializado para inferência de modelos de linguagem. O modelo Llama 3.3 70B, desenvolvido pela Meta e disponibilizado via Groq, é um modelo de linguagem de 70 bilhões de parâmetros, adequado para tarefas de análise e geração de texto que exigem raciocínio contextual aprofundado (META AI, 2024).

No *FashionBoost*, os dados operacionais da loja — incluindo produtos com seus níveis de estoque, vendas recentes, distribuição de clientes por nível de fidelidade e histórico de utilização de cupons — são serializados em formato estruturado e enviados como contexto para o modelo Llama 3.3 70B via API Groq. O modelo analisa essas informações e retorna sugestões organizadas em quatro categorias: promoções, estoque, fidelização e operacional. Cada sugestão é acompanhada de um nível de impacto estimado — alto, médio ou baixo — e de uma justificativa fundamentada nos dados reais da loja.

2.4 Linguagens e Frameworks

2.4.1 TypeScript

O TypeScript é uma linguagem de programação de código aberto desenvolvida pela Microsoft que estende o JavaScript com um sistema de tipagem estática opcional. Por ser um superconjunto do JavaScript, todo código JavaScript válido é também válido em TypeScript. A tipagem estática permite a identificação de erros em tempo de desenvolvimento, antes da execução do programa, aumentando a segurança e a previsibilidade do código em projetos de maior escala (MICROSOFT, 2024).

O TypeScript oferece suporte a conceitos de programação orientada a objetos, como classes, interfaces e herança, além de ser transpilado para JavaScript padrão, preservando a compatibilidade com todo o ecossistema JavaScript. No *FashionBoost*, o TypeScript foi adotado em toda a *stack* — tanto no *back-end* quanto no *front-end* — garantindo consistência e segurança de tipos em todos os módulos da aplicação.

Figura 1 – Exemplo de código TypeScript com tipagem estática

```typescript
class Student {
  fullName: string;
  constructor(
    public firstName: string,
    public middleInitial: string,
    public lastName: string
  ) {
    this.fullName = firstName + " " + middleInitial + " " + lastName;
  }
}

interface Person {
  firstName: string;
  lastName: string;
}

function greeter(person: Person) {
  return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("Jane", "M.", "User");

document.body.textContent = greeter(user);
```

Fonte: O autor.

2.4.2 NestJS

O NestJS é um framework para Node.js que utiliza TypeScript como linguagem base e incorpora conceitos de arquitetura modular, injeção de dependência e programação orientada a objetos. Inspirado na estrutura do Angular, o NestJS organiza a aplicação em

módulos que agrupam controladores, serviços e provedores, resultando em uma estrutura altamente coesa e de fácil manutenção (NESTJS, 2025).

O framework oferece suporte nativo à criação de APIs REST, integração com ORMs como o TypeORM, além de *guards*, *interceptors* e *pipes* que facilitam a implementação de autenticação, autorização e validação de dados. No *FashionBoost*, essas abstrações foram amplamente utilizadas: *guards* para verificação do token JWT e do tenant em cada requisição, *pipes* para validação automática dos dados de entrada, e *interceptors* para padronização das respostas da API.

2.4.3 Next.js

O Next.js é um framework React para desenvolvimento de aplicações web que oferece suporte nativo a renderização do lado do servidor (*Server-Side Rendering* — SSR), geração de páginas estáticas (*Static Site Generation* — SSG) e o modelo de *App Router*, introduzido na versão 13 e consolidado na versão 15. O *App Router* adota componentes de servidor React por padrão, permitindo a busca de dados diretamente nos componentes e a composição de layouts hierárquicos de forma declarativa (VERCEL, 2024).

No *FashionBoost*, o Next.js 15 foi utilizado para o desenvolvimento do *front-end*, com o *App Router* organizando as rotas da aplicação de forma hierárquica. Cada rota do painel do lojista é protegida por um mecanismo de autenticação que verifica a presença e validade do token JWT antes de renderizar qualquer conteúdo.

2.4.4 Tailwind CSS

O Tailwind CSS é um framework de utilitários CSS que permite a estilização de interfaces diretamente no HTML ou JSX por meio de classes pré-definidas de baixo nível. Diferentemente de frameworks baseados em componentes prontos, o Tailwind não impõe um conjunto de componentes visuais, mas oferece primitivos de estilização que permitem construir qualquer design sem sair do arquivo de marcação. Isso resulta em maior produtividade, eliminação de código CSS morto e total liberdade na definição visual da interface (TAILWIND CSS, 2025). A versão 4, utilizada no *FashionBoost*, introduziu melhorias significativas na performance de compilação e na forma como as variáveis de design são configuradas.

Figura 2 – Exemplo de estilização com Tailwind CSS v4

index.html app.css package.json

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Tailwind CSS</title>
    <link rel="stylesheet" href="/build.css" />
  </head>
  <body>
    <button class="flex items-center bg-blue-500"></button>
  </body>
</html>
```

build.css

```css
@layer utilities {
  .flex {
    display: flex;
  }
  .items-center {
    align-items: center;
  }
  .bg-blue-500 {
    background-color: var(--color-blue-500, oklch(62.3% 0.214 259.815));
  }
}
```

Fonte: O autor.

2.5 Persistência de Dados

2.5.1 TypeORM

O TypeORM é um framework de mapeamento objeto-relacional para TypeScript e JavaScript que permite definir entidades do banco de dados como classes TypeScript decoradas, mapeando automaticamente os atributos da classe para colunas da tabela correspondente. O TypeORM suporta os padrões de repositório e *data mapper*, oferece geração de migrações automáticas a partir das entidades definidas e integra-se nativamente com o NestJS por meio de um módulo dedicado (TYPEORM, 2025).

No *FashionBoost*, cada entidade do sistema foi definida como uma classe TypeScript decorada com os metadados do TypeORM, incluindo chaves primárias do tipo UUID, chaves estrangeiras para os relacionamentos e índices para os campos de filtragem mais utilizados, como o `storeId` presente em praticamente todas as entidades operacionais.

2.5.2 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados relacional de código aberto, reconhecido por sua estabilidade, conformidade com o padrão SQL e suporte completo a transações ACID (*Atomicity, Consistency, Isolation, Durability*). Essas propriedades garantem que operações compostas — como o cancelamento de uma venda, que envolve a reversão simultânea do estoque, dos pontos de fidelidade e do status da venda — sejam executadas de forma atômica, sem risco de estados parcialmente aplicados em caso de falha (POSTGRESQL, 2025).

Dentre as características mais relevantes para o *FashionBoost*, destaca-se também o mecanismo MVCC (*Multi-Version Concurrency Control*), que permite que múltiplos usuários

acessem e modifiquem dados simultaneamente sem bloqueios desnecessários. Em um ambiente SaaS com múltiplos tenants operando em paralelo, essa característica é fundamental para garantir performance e consistência.

2.6 Autenticação e Segurança

2.6.1 JSON Web Token (JWT)

O JSON Web Token é um padrão aberto (RFC 7519) para a criação de tokens de acesso compactos e autocontidos, usados para transmitir informações verificáveis entre partes por meio de uma assinatura digital. No contexto de autenticação em APIs REST, o JWT é gerado no servidor no momento do login e enviado pelo cliente em todas as requisições subsequentes, tipicamente no cabeçalho `Authorization`. O servidor valida a assinatura criptográfica do token sem necessidade de consultar o banco de dados, tornando o processo *stateless* e horizontalmente escalável (JONES; BRADLEY; SAKIMURA, 2015).

No *FashionBoost*, o payload do token JWT carrega o identificador do usuário, o identificador do tenant e o papel do usuário na plataforma. Essas informações são utilizadas em todos os *guards* da API para autenticação e para o filtro automático de dados por tenant, eliminando a necessidade de qualquer parâmetro explícito de tenant nas requisições.

2.6.2 Bcrypt

O Bcrypt é um algoritmo de hashing adaptativo especialmente projetado para o armazenamento seguro de senhas. Ao contrário de funções de hash genéricas como MD5 ou SHA-1, o Bcrypt incorpora um fator de custo configurável que determina o número de iterações do algoritmo, tornando o cálculo deliberadamente lento e, portanto, resistente a ataques de força bruta e de dicionário. No *FashionBoost*, todas as senhas dos usuários são armazenadas exclusivamente como hashes Bcrypt no banco de dados.

2.6.3 Nodemailer e Gmail SMTP

O Nodemailer é uma biblioteca para Node.js que permite o envio de e-mails a partir de aplicações *back-end*, com suporte a múltiplos provedores de transporte, incluindo servidores SMTP externos. No *FashionBoost*, o Nodemailer foi configurado com o Gmail

SMTP como provedor de transporte, permitindo o envio automático de e-mails de verificação de conta no momento do cadastro de um novo lojista.

2.7 Ferramentas de Prototipação e Assistência com Inteligência Artificial

2.7.1 Lovable

O Lovable é uma plataforma de desenvolvimento de interfaces baseada em inteligência artificial que permite a criação de telas e componentes web a partir de descrições em linguagem natural. A ferramenta interpreta as instruções fornecidas pelo usuário e gera código funcional em React com estilização em Tailwind CSS, produzindo protótipos navigáveis de forma rápida e iterativa, sem necessidade de codificação manual nessa etapa (LOVABLE, 2024).

No contexto do *FashionBoost*, o Lovable foi utilizado na fase inicial de prototipação do *front-end* como ferramenta de exploração visual e estrutural. Por meio de prompts descritivos das telas planejadas para a plataforma, foi possível gerar rapidamente versões preliminares da interface, que serviram como referência visual para a implementação definitiva em Next.js.

A contribuição do Lovable foi especialmente relevante na definição dos fluxos de navegação entre as telas da plataforma. Antes de escrever qualquer linha de código no projeto definitivo, foi possível simular o caminho que o lojista percorreria desde o login até o registro de uma venda, passando pelo cadastro de produtos, configuração dos níveis de fidelidade e acesso ao painel de inteligência artificial. Essa simulação antecipada permitiu identificar inconsistências no fluxo e corrigi-las ainda na fase de prototipação, sem custo de retrabalho no desenvolvimento real.

A ferramenta também foi fundamental para a definição da identidade visual da plataforma. Por meio de prompts que descreviam o público-alvo, o setor de moda e o tom de marca desejado, o Lovable gerou propostas de paleta de cores, tipografia e organização dos elementos visuais que serviram como ponto de partida para as decisões de design adotadas na implementação definitiva com Tailwind CSS v4. A parametrização da cor principal por tenant — um dos diferenciais visuais do *FashionBoost* — também foi testada conceitualmente nos protótipos gerados, validando a viabilidade da abordagem antes da implementação.

Do ponto de vista metodológico, o uso do Lovable representou uma mudança significativa no processo tradicional de prototipação. Em vez de criar wireframes estáticos em ferramentas de design, foi possível trabalhar diretamente com protótipos navigáveis e funcionais, o que aproximou a fase de exploração da experiência real de uso do sistema.

2.7.2 Claude (Anthropic)

O Claude, desenvolvido pela Anthropic, é um assistente de inteligência artificial baseado em modelos de linguagem de grande escala, capaz de compreender e gerar texto com alto grau de coerência contextual e raciocínio técnico. No desenvolvimento do *FashionBoost*, o Claude foi utilizado como ferramenta de assistência ao longo de todo o ciclo de desenvolvimento, desempenhando um papel complementar ao trabalho do desenvolvedor em diferentes frentes (ANTHROPIC, 2024).

No contexto do *back-end*, o Claude auxiliou na geração e revisão de trechos de código NestJS, especialmente na implementação de *guards* de autenticação, na estruturação dos módulos e na lógica de transações atômicas do cancelamento de vendas. A capacidade do modelo de compreender o contexto acumulado do projeto permitiu sugestões consistentes com as decisões arquiteturais já tomadas, reduzindo o tempo de implementação de funcionalidades individuais e acelerando a identificação de erros lógicos antes da execução.

No contexto do *front-end*, o Claude contribuiu na estruturação de componentes React reutilizáveis, na organização das rotas com o *App Router* do Next.js 15 e na configuração da instância Axios com injeção automática do token JWT. Também foi utilizado para revisar a consistência entre os contratos da API e os tipos TypeScript definidos no *front-end*, antecipando erros de integração antes da execução.

Além da assistência ao código, o Claude foi utilizado como ferramenta de suporte à documentação técnica deste trabalho, auxiliando na estruturação e revisão do texto acadêmico em LaTeX, na organização das seções e na consistência terminológica ao longo dos capítulos. Essa utilização reflete uma tendência crescente no desenvolvimento de software moderno: a integração de assistentes baseados em LLMs como ferramentas de produtividade no ciclo completo de desenvolvimento, desde a concepção até a documentação.

É importante destacar que o uso do Claude não substituiu o raciocínio e as decisões técnicas do desenvolvedor, mas funcionou como um acelerador do processo — reduzindo o

tempo gasto em tarefas repetitivas e ampliando a capacidade de revisão e refinamento do código produzido.

3 MATERIAIS E PROCEDIMENTOS METODOLÓGICOS

Este capítulo descreve os procedimentos adotados e as ferramentas utilizadas no desenvolvimento do *FashionBoost*.

3.1 Metodologia

Quanto à sua natureza, esta pesquisa é classificada como aplicada, pois tem como objetivo gerar conhecimento voltado à solução de um problema específico e concreto: a ausência de ferramentas de gestão e fidelização acessíveis para pequenos lojistas de moda. Do ponto de vista dos objetivos, a pesquisa é exploratória e descritiva. Exploratória porque investiga um domínio ainda pouco explorado academicamente — a aplicação de plataformas SaaS com inteligência artificial integrada no varejo de moda independente — e descritiva porque documenta com precisão as características, funcionalidades e resultados do sistema desenvolvido (WAZLAWICK, 2014).

Quanto aos procedimentos técnicos, a pesquisa é bibliográfica e experimental. Bibliográfica por apoiar-se em fontes já publicadas para embasar as escolhas tecnológicas e conceituais, e experimental pelo desenvolvimento e validação prática do protótipo funcional do *FashionBoost* (GIL, 2017).

O processo de desenvolvimento seguiu uma abordagem iterativa e incremental, na qual as funcionalidades foram implementadas progressivamente. A sequência adotada iniciou-se pelo levantamento de requisitos e modelagem do banco de dados, avançou para a construção da API *back-end* com NestJS, passou pela prototipação visual com o Lovable e culminou no desenvolvimento definitivo do *front-end* em Next.js e na integração com o serviço de inteligência artificial via API Groq.

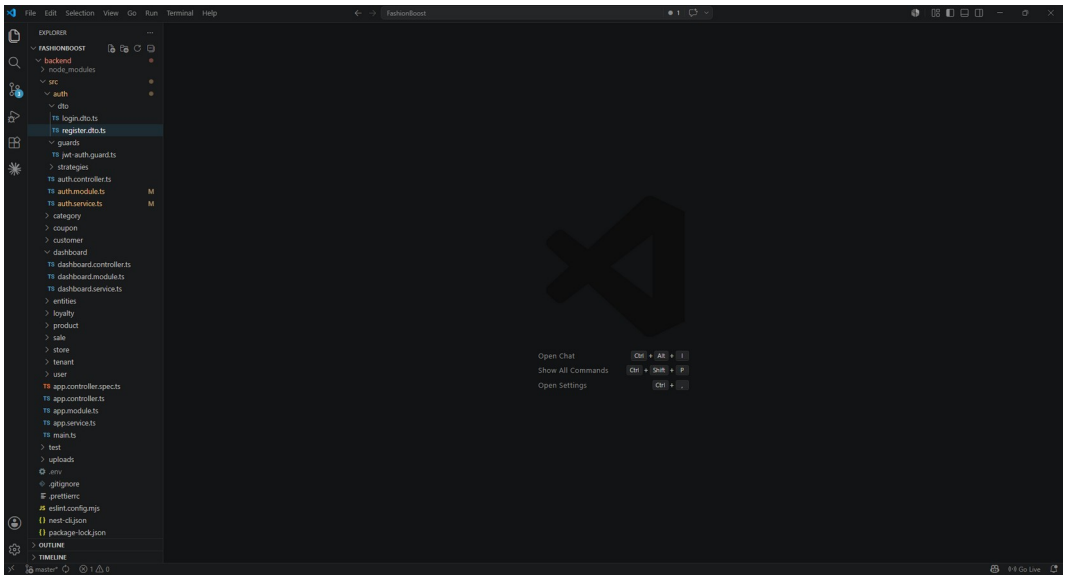
3.2 Ferramentas de Desenvolvimento

3.2.1 Visual Studio Code

O Visual Studio Code é um editor de código-fonte desenvolvido pela Microsoft, compatível com múltiplas linguagens e plataformas. Sua ampla biblioteca de extensões — incluindo suporte nativo a TypeScript, integração com Git, depuração em tempo real e plugins específicos para NestJS e Next.js — o tornou a ferramenta central de desenvolvimento em

todas as etapas deste projeto. A Figura 3 apresenta a estrutura de pastas do *back-end* do *FashionBoost* aberta no editor, evidenciando a organização modular adotada, com cada domínio funcional isolado em seu próprio diretório.

Figura 3 – Visual Studio Code com a estrutura modular do back-end do *FashionBoost*



Fonte: O autor.

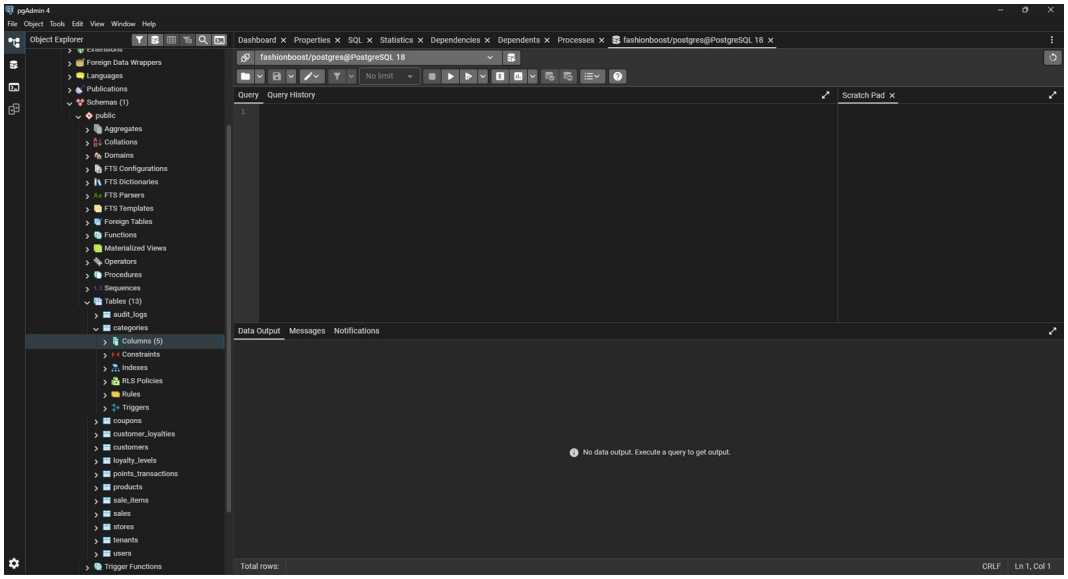
3.2.2 Lovable

O Lovable foi utilizado como ferramenta de prototipação inteligente do *front-end*. Por meio de descrições em linguagem natural das telas e funcionalidades desejadas, a plataforma gerou protótipos navigáveis em React e Tailwind CSS, que serviram como referência visual e estrutural para o desenvolvimento definitivo da interface do *FashionBoost*.

3.2.3 pgAdmin

O pgAdmin é uma ferramenta gráfica para administração de bancos de dados PostgreSQL. Durante o desenvolvimento do *FashionBoost*, foi utilizado para criação, visualização e manutenção das tabelas do banco, além do monitoramento das relações entre entidades e execução de consultas SQL de verificação. A Figura 4 ilustra a interface do pgAdmin durante o desenvolvimento, exibindo as tabelas criadas pelo TypeORM.

Figura 4 – Interface do pgAdmin durante o desenvolvimento do *FashionBoost*

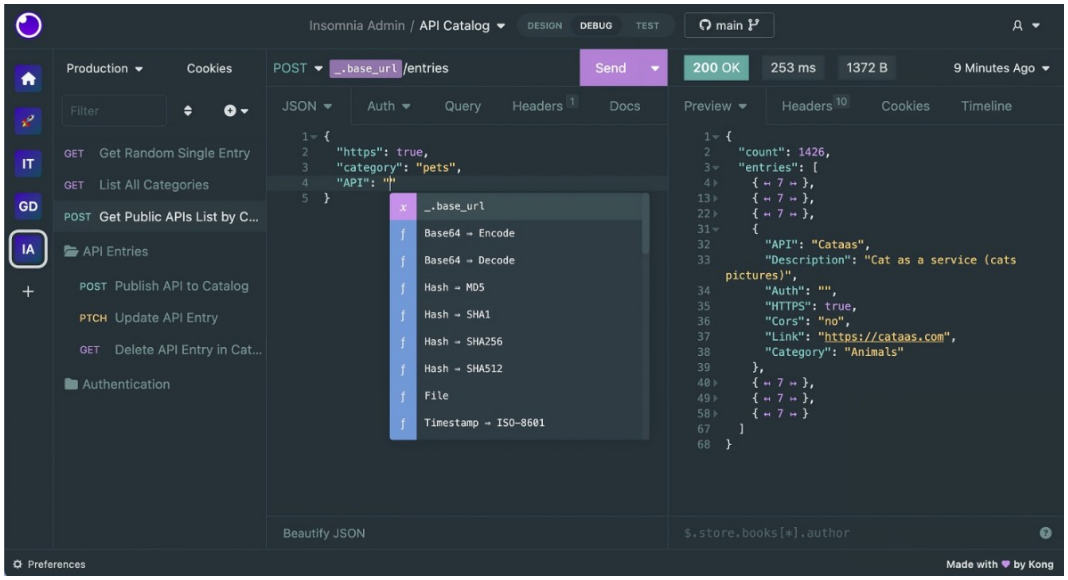


Fonte: O autor.

3.2.4 Insomnia e Swagger

Os testes das rotas da API foram realizados por meio do Insomnia, cliente HTTP que permite o envio de requisições com configuração detalhada de cabeçalhos, parâmetros e corpo da requisição. O Swagger foi integrado diretamente ao NestJS para geração automática da documentação interativa da API. A Figura 5 ilustra uma requisição realizada durante o desenvolvimento, exibindo o corpo enviado e a resposta retornada pela API com status 200 OK

Figura 5 – Teste de endpoint da API no Insomnia



Fonte: O autor.

3.2.5 dbdiagram.io

O dbdiagram.io é uma ferramenta online para modelagem e visualização de bancos de dados relacionais. No *FashionBoost*, a ferramenta foi utilizada para produzir o MER completo do sistema, que serviu tanto como documentação técnica quanto como guia durante a implementação das entidades no TypeORM.

3.3 Materiais

Tabela 1 – Características do computador utilizado no desenvolvimento

CAMPO	ESPECIFICAÇÃO
SISTEMA OPERACIONAL	Windows 11
PROCESSADOR	Ryzen 7 5700G
MEMÓRIA RAM	32 GB
TIPO DE SISTEMA	64 BITS
ARMAZENAMENTO	SSD 1 TB

Fonte: O autor.

4 DESENVOLVIMENTO DO PROJETO

4.1 Prototipação

4.1.1 Levantamento de Requisitos

Antes de iniciar qualquer etapa de modelagem ou implementação, foi realizado o levantamento formal de requisitos do *FashionBoost*. Essa etapa teve como objetivo identificar e documentar, de forma estruturada, o que o sistema deveria fazer, quais regras de negócio deveriam ser respeitadas e quais atributos de qualidade seriam necessários. O documento produzido seguiu a estrutura recomendada pelo padrão IEEE para *Software Requirements Specification* (SRS), garantindo clareza, rastreabilidade e organização dos requisitos ao longo de todo o desenvolvimento. O documento completo de especificação de requisitos encontra-se disponível no Anexo — Protótipo de Requisitos de Software deste trabalho.

O levantamento identificou dois atores principais no sistema: o **Lojista**, responsável pela administração da loja, configuração do programa de fidelidade e visualização de relatórios; e o **Funcionário da Loja**, usuário autorizado a registrar vendas presenciais, consultar clientes e validar cupons de recompensa. Essa distinção de papéis orientou diretamente as decisões de controle de acesso implementadas no *back-end*.

Os requisitos funcionais levantados cobrem todo o ciclo operacional da plataforma, desde o cadastro do lojista até a geração de relatórios de vendas. Os principais são: cadastro e autenticação segura de usuários (RF001, RF002); cadastro e configuração de lojas (RF003); gerenciamento de categorias e produtos com preço, estoque e pontuação (RF004, RF005); cadastro de clientes por CPF (RF006); registro de vendas presenciais com cálculo automático de pontos (RF007, RF008); configuração de níveis de fidelidade (RF009); geração e validação de cupons por pontos (RF010, RF011); e geração de relatórios de vendas (RF012).

As regras de negócio formalizadas incluem: identificação única de clientes por CPF dentro da loja (RN001); cálculo de pontos baseado na pontuação definida por produto (RN002); níveis de fidelidade configuráveis pelo lojista, incluindo Bronze, Prata e Ouro (RN003); progressão de nível condicionada à pontuação mínima (RN004); uso único de cupons após geração (RN005); e restrição de operações críticas a usuários autenticados

(RN006).

Os requisitos não funcionais estabeleceram: autenticação segura (NF001); desempenho adequado para operações comerciais em tempo real (NF002); interface simples e intuitiva (NF003); e suporte a múltiplas lojas simultaneamente por meio de arquitetura multi-tenant (NF004).

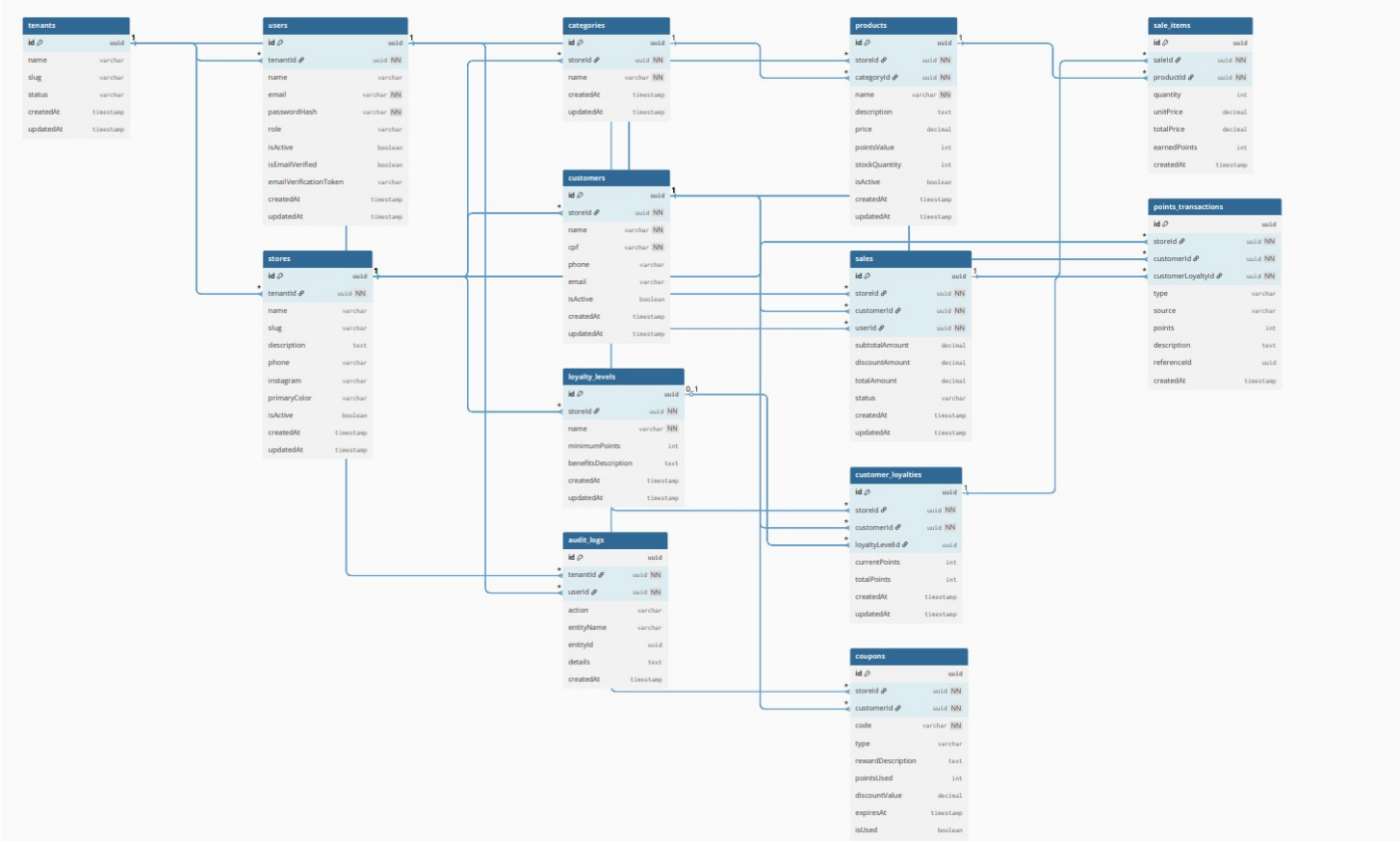
Esse levantamento foi determinante para que o desenvolvimento seguisse um caminho claro e rastreável, evitando a implementação de funcionalidades sem respaldo nos requisitos e garantindo que cada decisão técnica pudesse ser justificada em relação às necessidades reais do sistema.

4.1.2 Modelo de Entidade-Relacionamento

A partir dos requisitos levantados, a próxima etapa concreta do desenvolvimento foi a elaboração do Modelo de Entidade-Relacionamento (MER). Essa etapa teve como objetivo mapear com precisão quais informações o sistema precisaria armazenar, quais atributos cada entidade deveria ter e como as diferentes partes do domínio se relacionariam entre si. As entidades do MER correspondem diretamente ao modelo de dados descrito no SRS, validando a coerência entre os requisitos levantados e a estrutura de persistência implementada.

O MER do *FashionBoost* foi desenvolvido com o auxílio da ferramenta dbdiagram.io e é composto por 13 entidades que cobrem todos os domínios funcionais da plataforma: identidade e acesso, catálogo de produtos, clientes, vendas, descontos, fidelidade e auditoria. A Figura 6 apresenta o diagrama completo com todos os relacionamentos entre as entidades.

Figura 6 – Modelo de Entidade-Relacionamento do *FashionBoost*



Fonte: O autor.

O diagrama é organizado a partir de sua âncora central: a entidade `stores`, que representa a loja de cada lojista cadastrado na plataforma. Praticamente todas as demais entidades operacionais se relacionam diretamente com `stores` por meio de uma chave estrangeira `storeId`, o que garante o isolamento de dados entre lojistas diferentes — princípio fundamental da arquitetura multi-tenant adotada no sistema.

Domínio de identidade e acesso: A entidade `tenants` é a raiz do modelo multi-tenant e representa o lojista como titular da conta na plataforma, carregando atributos como nome, slug, status e *timestamps*. A entidade `users` representa os operadores do sistema dentro de um tenant, armazenando nome, e-mail, hash da senha, papel (*role*), status de ativação e status de verificação de e-mail. A entidade `stores` vincula-se ao tenant e concentra os dados públicos e de configuração da loja: nome, slug, descrição, telefone, Instagram, cor principal do painel (`primaryColor`) e status de ativação.

Domínio de catálogo: A entidade `categories` organiza os produtos por tipo. A enti-

dade products armazena os itens do catálogo com nome, descrição, preço unitário, pontos que a compra gera (pointsValue), quantidade em estoque (stockQuantity), categoria e status de ativação.

Domínio de clientes e vendas: A entidade customersregistra os clientes por loja, identificados por CPF — conforme estabelecido na regra de negócio RN001. A entidade sales representa cada venda realizada, armazenando subtotal, desconto aplicado, total final e status (*active*, *cancelled* ou *exchanged*). A entidadesale_items resolve o relacionamento muitos-para-muitos entre vendas e produtos, armazenando quantidade, preço unitário praticado e pontos ganhos por item.

Domínio de descontos: A entidade couponsarmazena os cupons de desconto com código único, tipo, valor, data de expiração e indicador de uso, atendendo à regra de negócio RN005.

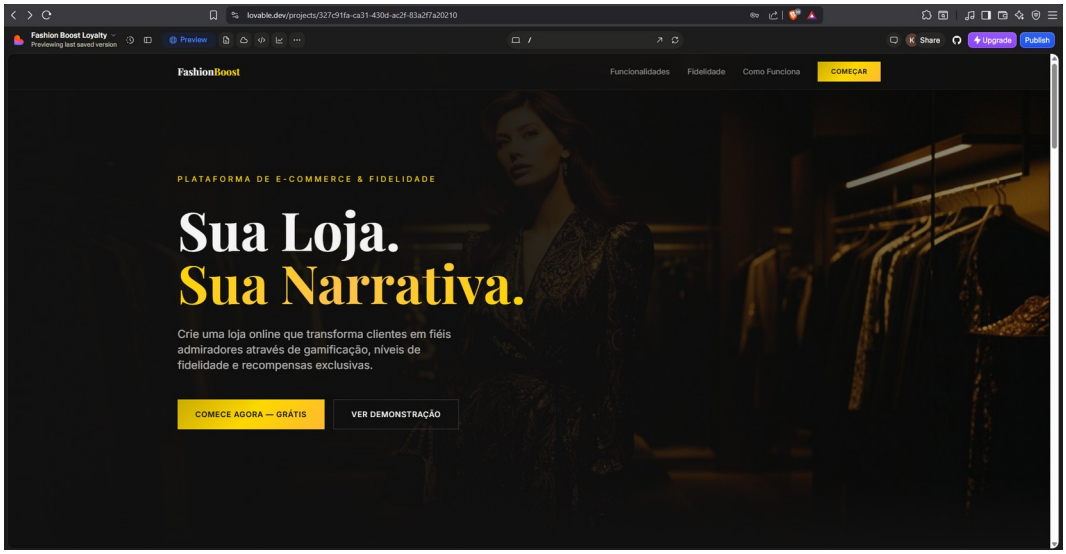
Domínio de fidelidade: A entidade loyalty_levels define os níveis configurados pelo lojista, com pontuação mínima e descrição dos benefícios — correspondendo diretamente às regras RN003 e RN004. A entidade customer_loyalties mantém o estado atual de fidelidade de cada cliente, com pontos disponíveis e total histórico acumulado. A entidade points_transactions registra cada movimentação de pontos com tipo, fonte, quantidade e referência à operação de origem.

Domínio de auditoria: A entidade audit_logs registra todas as ações relevantes executadas no sistema, fornecendo rastreabilidade completa das modificações efetuadas na plataforma.

4.1.3 Prototipação com Lovable

Após a definição da estrutura de dados, a etapa seguinte foi a prototipação da interface do *front-end* com o auxílio da plataforma Lovable. Por meio de prompts detalhados descrevendo as telas planejadas, foram geradas versões navigáveis da interface que serviram como referência visual para a implementação definitiva em Next.js 15. A Figura 7 apresenta um dos protótipos gerados, evidenciando a identidade visual inicial adotada para a plataforma.

Figura 7 – Protótipo gerado pelo Lovable para o *FashionBoost*



Fonte: O autor.

Essa abordagem acelerou a fase de definição visual e reduziu o retrabalho ao permitir testar diferentes organizações de layout antes de comprometer tempo de desenvolvimento na implementação real. Os protótipos não foram utilizados diretamente no produto final, mas serviram como referência clara para as decisões de design.

4.2 Construção da Aplicação

4.2.1 Desenvolvimento do Back-End

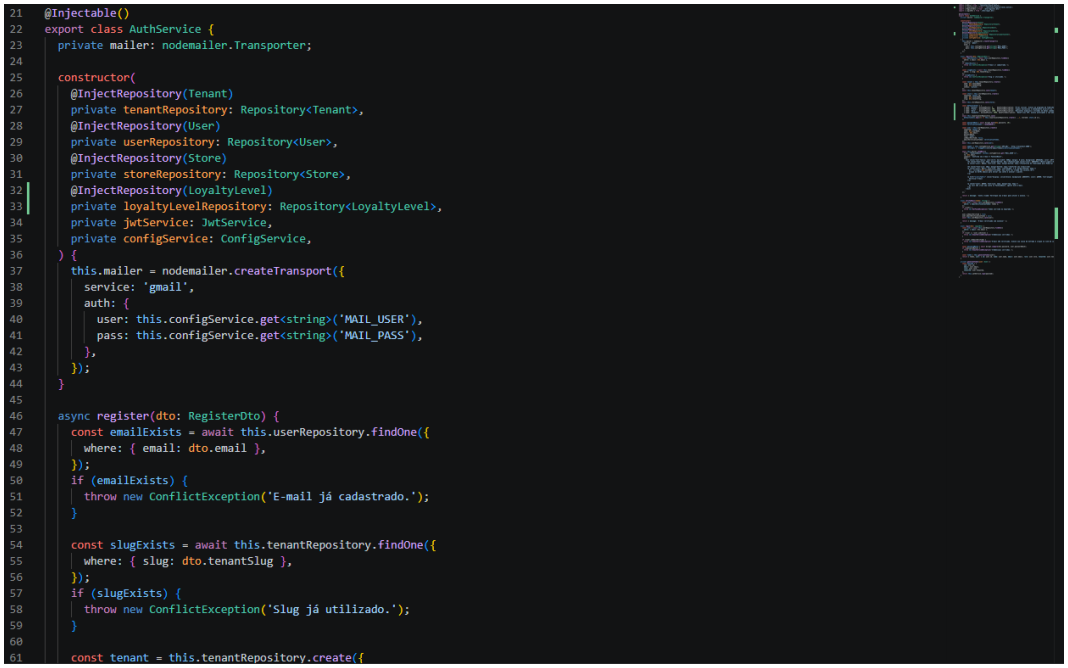
O *back-end* do *FashionBoost* foi desenvolvido integralmente com o NestJS, em TypeScript, utilizando o TypeORM para comunicação com o PostgreSQL. A arquitetura modular do NestJS orientou a organização do código: cada domínio funcional — autenticação, tenants, lojas, produtos, categorias, clientes, vendas, fidelidade, cupons, inteligência artificial e auditoria — foi implementado como um módulo independente, com seu próprio *controller*, *service* e conjunto de entidades.

4.2.1.1 Autenticação, Multi-tenancy e Verificação de E-mail

O fluxo de cadastro e autenticação implementa os requisitos RF001 e RF002. O processo se inicia com o registro do tenant, que dispara automaticamente um e-mail de verificação via Nodemailer integrado ao Gmail SMTP. As senhas são armazenadas como hashes Bcrypt e o login gera um token JWT contendo o identificador do usuário, do tenant e

o papel do usuário. O isolamento multi-tenant é aplicado em todas as rotas por um *guard* customizado que extrai o `tenantId` do token e filtra automaticamente todas as consultas ao banco de dados. A Figura 8 ilustra parte do código para realizar autenticação na plataforma, e a Figura 9 demonstra a implementação do isolamento por tenant.

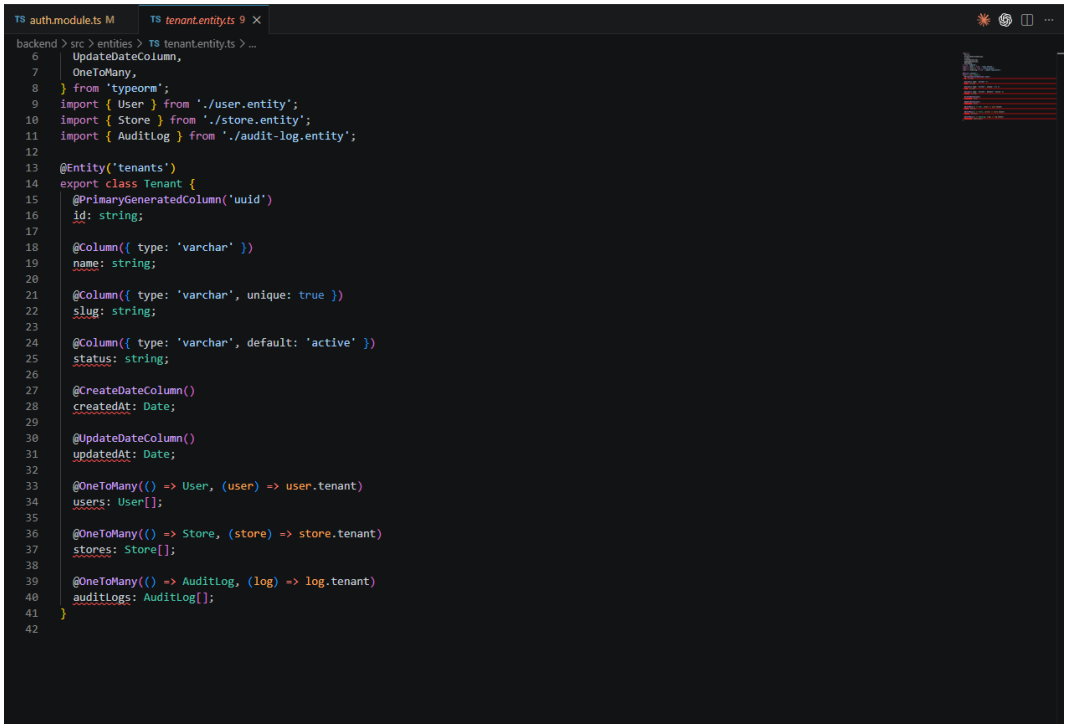
Figura 8 – Código de autenticação do *FashionBoost*



```
@Injectable()
export class AuthService {
  private mailer: nodemailer.Transporter;

  constructor(
    @InjectRepository(Tenant)
    private tenantRepository: Repository<Tenant>,
    @InjectRepository(User)
    private userRepository: Repository<User>,
    @InjectRepository(Store)
    private storeRepository: Repository<Store>,
    @InjectRepository(LoyaltyLevel)
    private loyaltyLevelRepository: Repository<LoyaltyLevel>,
    private jwtService: JwtService,
    private configService: ConfigService,
  ) {
    this.mailer = nodemailer.createTransport({
      service: 'gmail',
      auth: {
        user: this.configService.get<string>('MAIL_USER'),
        pass: this.configService.get<string>('MAIL_PASS'),
      },
    });
  }

  async register(dto: RegisterDto) {
    const emailExists = await this.userRepository.findOne({
      where: { email: dto.email },
    });
    if (emailExists) {
      throw new ConflictException('E-mail já cadastrado.');
    }

    const slugExists = await this.tenantRepository.findOne({
      where: { slug: dto.tenantSlug },
    });
    if (slugExists) {
      throw new ConflictException('Slug já utilizado.');
    }

    const tenant = this.tenantRepository.create({
```

Fonte: O autor.

Figura 9 – Código de isolamento por tenant no *FashionBoost*

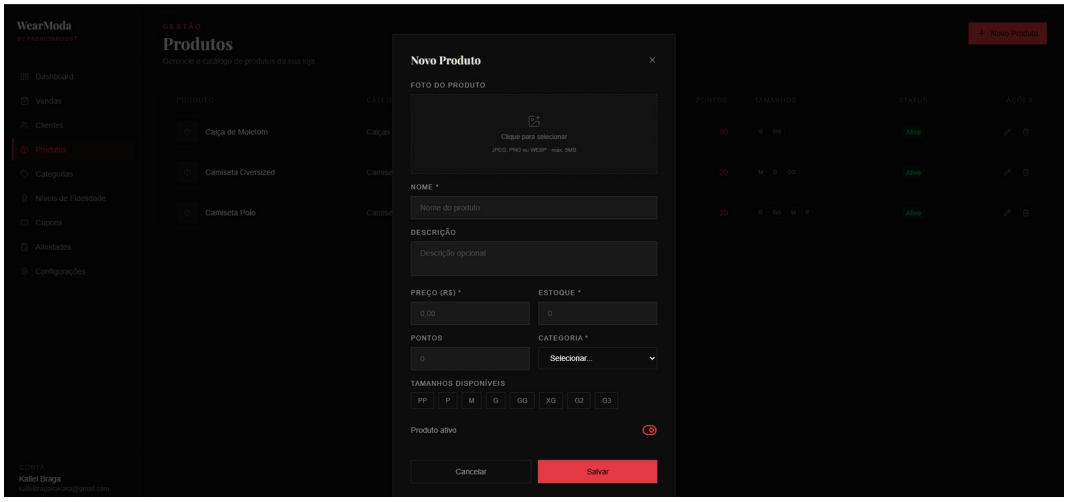


Fonte: O autor.

4.2.1.2 Gestão de Produtos, Categorias e Estoque

Implementando os requisitos RF004 e RF005, o módulo de produtos permite o cadastro de itens com nome, descrição, preço unitário, categoria, pontos e quantidade em estoque. O controle de estoque é automatizado: a cada venda confirmada as quantidades são deduzidas; em cancelamento, são restauradas na mesma operação. A Figura 10 apresenta a interface de gestão de produtos da plataforma, e a Figura 11 ilustra o módulo de categorias.

Figura 10 – Interface de gestão de produtos do *FashionBoost*



Fonte: O autor.

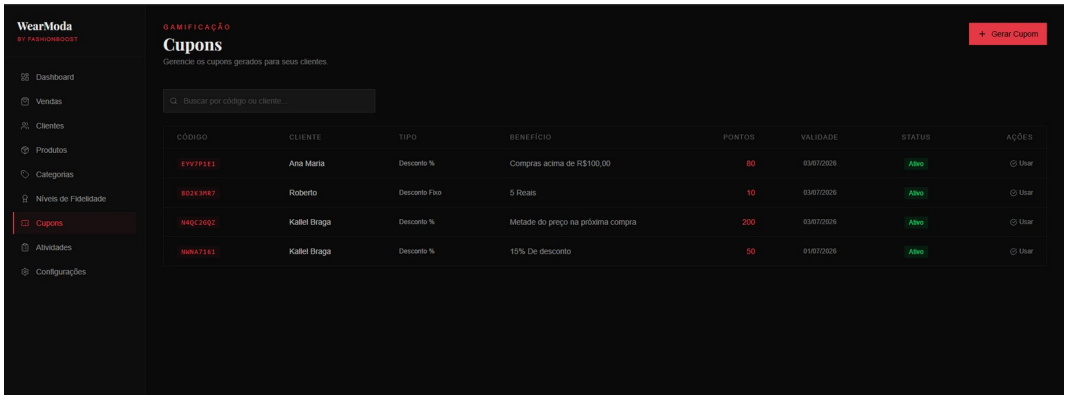
Figura 11 – Módulo de categorias do *FashionBoost*

Fonte: O autor.

4.2.1.3 Fluxo de Vendas e Cupons

Atendendo aos requisitos RF007, RF010 e RF011, o módulo de vendas permite a criação de pedidos com múltiplos itens e aplicação opcional de cupom. O cancelamento executa dentro de uma única transação atômica: restaura estoque, estorna pontos com registro em `points_transactions` e atualiza o status da venda — garantindo consistência total dos dados. A Figura 12 apresenta a interface de gestão de cupons do *FashionBoost*.

Figura 12 – Interface de gestão de cupons do *FashionBoost*



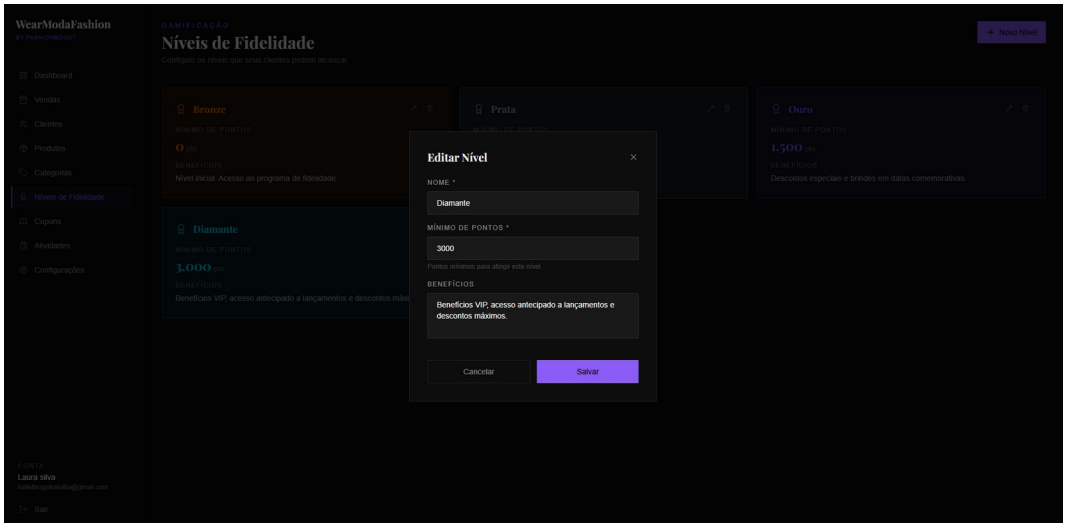
Fonte: O autor.

4.2.1.4 Programa de Fidelidade

Implementando os requisitos RF008 e RF009 e as regras RN002, RN003 e RN004, o programa de fidelidade do *FashionBoost* foi projetado para ser totalmente configurável pelo lojista, adaptando-se às particularidades de cada estabelecimento sem exigir conhecimento técnico para sua operação.

A configuração do programa ocorre em dois níveis. O primeiro é a definição dos níveis de fidelidade: o lojista cria os níveis que desejar — por exemplo, Bronze, Prata e Ouro — e estabelece para cada um a pontuação histórica mínima necessária para que o cliente seja promovido, além de uma descrição dos benefícios associados a cada categoria. A Figura 13 ilustra a tela de configuração de níveis de fidelidade no painel do lojista.

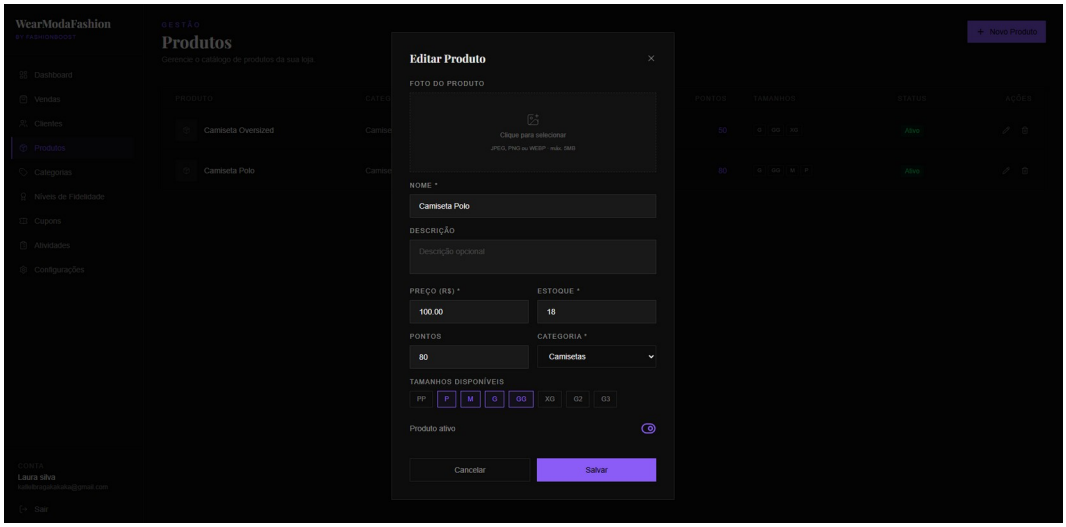
Figura 13 – Tela de configuração de níveis de fidelidade no *FashionBoost*



Fonte: O autor.

O segundo nível é a definição da pontuação por produto: cada item do catálogo possui um campo pointsValue que determina quantos pontos o cliente recebe ao adquiri-lo, permitindo que o lojista crie incentivos diferenciados. A Figura 14 apresenta a configuração de pontuação diretamente no cadastro de um produto.

Figura 14 – Configuração de pontuação por produto no *FashionBoost*



Fonte: O autor.

A cada venda confirmada, o sistema executa automaticamente a seguinte sequência: calcula o total de pontos ganhos com base nos itens adquiridos e nos seus respectivos pointsValue; registra a transação em points_transactions ; atualiza os campos currentPoints e totalPoints do registro customer_loyalties do cliente; e verifica se o novo total histórico ultrapassa o limiar do próximo nível de fidelidade — promovendo o cliente automaticamente quando esse critério é atingido.

Em caso de cancelamento de venda, os pontos creditados são estornados na mesma transação atômica que reverte o estoque, garantindo que o saldo de fidelidade do cliente reflita sempre apenas compras efetivamente concluídas.

4.2.1.5 Painel de Inteligência Artificial

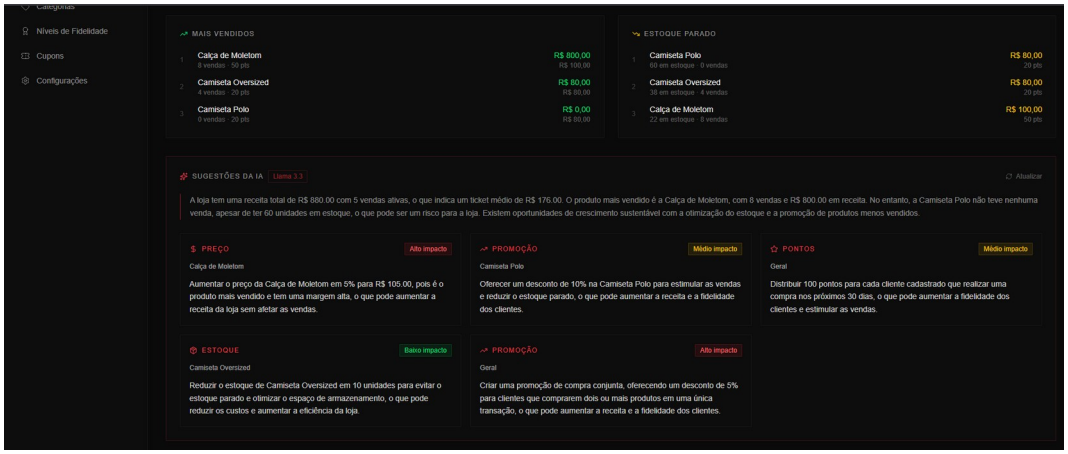
O painel de inteligência artificial foi desenvolvido com o objetivo de oferecer ao lojista um suporte estratégico baseado nos próprios dados da sua loja. Pequenos varejistas de moda, em geral, não dispõem de equipes de análise de dados ou ferramentas de *business intelligence* para interpretar o comportamento das suas vendas, o nível de engajamento

dos clientes no programa de fidelidade ou os produtos com risco de ruptura de estoque. O painel de IA preenche essa lacuna ao transformar os dados operacionais da loja em recomendações práticas, apresentadas em linguagem natural e organizadas por área de impacto.

A cada acesso ao painel de IA, o sistema coleta e serializa os dados operacionais da loja e os envia como contexto ao modelo Llama 3.3 70B via API Groq. O modelo retorna sugestões estruturadas em quatro categorias: promoções, estoque, fidelização e operacional, cada uma com nível de impacto e justificativa baseada nos dados reais da loja. A Figura 15 apresenta o painel de sugestões gerado pela IA.

Com as informações exibidas no painel, o lojista pode tomar decisões direcionadas sem precisar cruzar dados manualmente. Uma sugestão de promoção pode indicar que determinado produto acumula estoque há semanas e se beneficiaria de um desconto; uma sugestão de fidelização pode apontar clientes próximos de atingir um novo nível de pontos; e sugestões operacionais podem alertar para produtos com estoque crítico.

Figura 15 – Painel de sugestões estratégicas geradas pela IA no *FashionBoost*



Fonte: O autor.

Embora o painel de IA represente o componente mais inovador do *FashionBoost*, sua implementação envolve limitações e riscos técnicos que merecem discussão.

O primeiro diz respeito à **alucinação** (*hallucination*), fenômeno pelo qual modelos de linguagem geram informações plausíveis mas factualmente incorretas ou desconexas do contexto fornecido (BROWN et al., 2020). Para mitigar esse risco, o prompt estruturado instrui o modelo a basear cada sugestão exclusivamente nos dados fornecidos e a apresentar uma

justificativa explícita.

O segundo ponto é a **dependência de API externa**. O painel de IA só funciona enquanto a API Groq estiver disponível. Como o *FashionBoost* não hospeda o modelo localmente, não há controle sobre a disponibilidade e a evolução do serviço de inferência, o que representa um ponto de fragilidade arquitetural.

O terceiro aspecto é o **custo operacional**. A API Groq opera com modelo de cobrança por tokens processados. Em escala, com muitos tenants acessando o painel frequentemente, esse custo pode se tornar relevante e precisa ser incorporado ao modelo de precificação da plataforma.

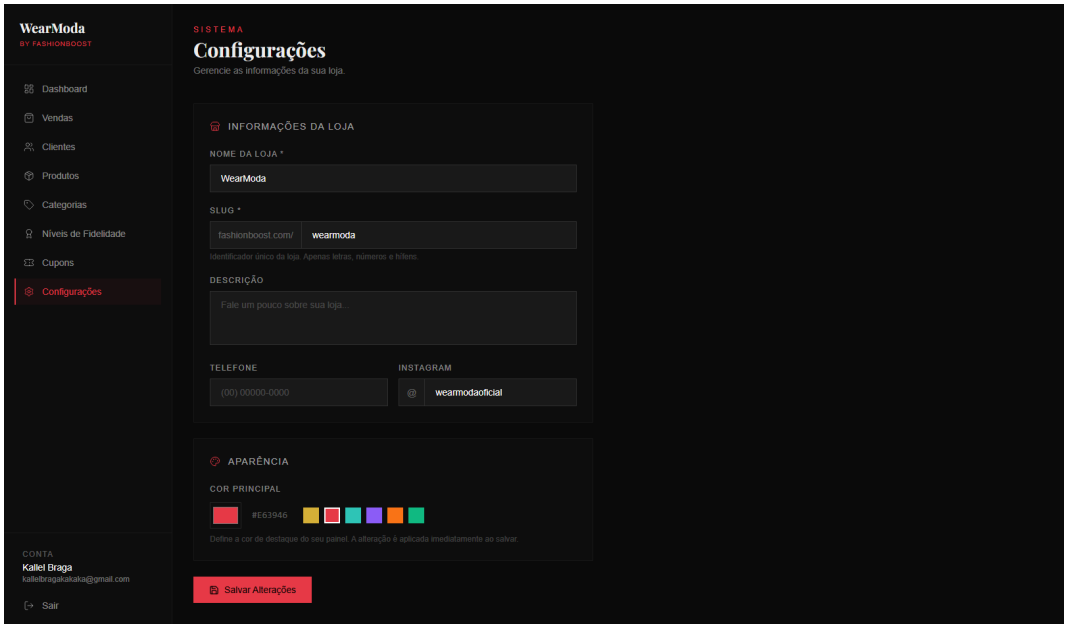
O quarto é a **qualidade variável das recomendações**. Lojas recém-cadastradas, com poucos produtos ou poucas vendas registradas, tendem a receber sugestões genéricas e de menor utilidade prática.

Por fim, a **privacidade dos dados** é uma questão crítica. Ao serializar e transmitir dados operacionais da loja para um serviço externo de inferência, o *FashionBoost* precisa garantir conformidade com a Lei Geral de Proteção de Dados (LGPD) (BRASIL, 2018). Na implementação atual, os dados enviados ao modelo são agregados e não incluem informações de identificação pessoal direta.

4.2.1.6 Configurações da Loja e Audit Logs

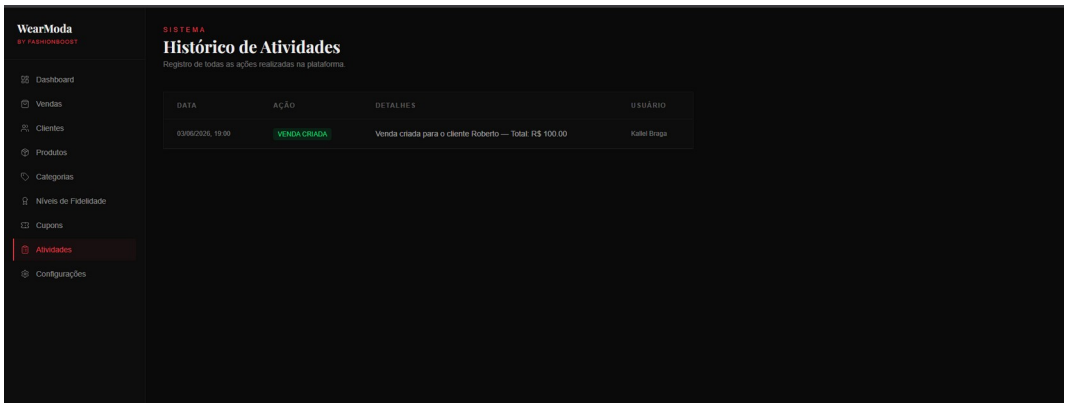
O módulo de configurações atende ao requisito RF003, permitindo a personalização do nome, slug, telefone, Instagram e cor principal do painel. A Figura 16 apresenta a tela de configurações da loja. Todas as operações de escrita geram automaticamente registros em `audit_logs`, garantindo rastreabilidade completa. A Figura 17 ilustra o painel de auditoria.

Figura 16 – Tela de configurações da loja no *FashionBoost*



Fonte: O autor.

Figura 17 – Painel de audit logs do *FashionBoost*



Fonte: O autor.

4.2.2 Desenvolvimento do Front-End

O *front-end* do *FashionBoost* foi desenvolvido com Next.js 15, TypeScript e Tailwind CSS v4, utilizando o *App Router* para organização hierárquica das rotas. A comunicação com o *back-end* ocorre via Axios, com uma instância configurada que injeta automaticamente o token JWT em todas as requisições autenticadas. A biblioteca Lucide React foi utilizada para iconografia, e os componentes foram desenvolvidos com foco na reutilização. A cor principal da loja é aplicada dinamicamente como variável CSS global, criando experiência visual personalizada para cada tenant.

4.3 Principais Funcionalidades Implementadas

4.3.1 Autenticação e Isolamento por Tenant

Registro com verificação de e-mail, login com JWT e proteção de rotas. Cada lojista acessa painel completamente isolado, com dados filtrados automaticamente pelo `tenantId`.

4.3.2 Gestão de Produtos, Estoque e Categorias

Cadastro completo de produtos com controle automático e bidirecional de estoque, decrementado na venda e restaurado no cancelamento de forma atômica.

4.3.3 Gestão de Clientes

Cadastro com CPF, telefone e e-mail, com visualização do nível de fidelidade atual, pontos disponíveis, total histórico acumulado e histórico completo de transações de pontos.

4.3.4 Vendas com Cupons e Cancelamento Atômico

Criação de pedidos com múltiplos itens, aplicação de cupons e cancelamento processado em transação atômica que reverte estoque, pontos e status simultaneamente.

4.3.5 Programa de Fidelidade Configurável

Pontos por produto, limiares de nível e progressão automática, tudo configurável pelo lojista. Histórico completo de transações para auditoria.

4.3.6 Cupons de Desconto

Suporte a valor fixo (R$) e percentual (%), com uso único por cliente e data de expiração configurável.

4.3.7 Painel de IA com Sugestões Estratégicas

Análise dos dados operacionais da loja pelo modelo Llama 3.3 70B via API Groq, com sugestões categorizadas por área, nível de impacto e justificativa baseada nos dados reais.

4.3.8 Configurações e Personalização da Loja

Personalização completa do perfil e cor do painel, aplicada dinamicamente em toda a interface daquele tenant.

5 RESULTADOS E DISCUSSÃO

5.1 Visão Geral dos Resultados

O desenvolvimento do *FashionBoost* resultou em uma plataforma SaaS funcional, coerente com os requisitos definidos no SRS e capaz de atender às principais demandas operacionais de uma loja de moda de pequeno e médio porte. Retomando a questão de pesquisa que orienta este trabalho — *Como o FashionBoost, enquanto um sistema SaaS, pode apoiar a gestão operacional e a fidelização de clientes em lojas de moda de pequeno e médio porte?* — os resultados obtidos demonstram que o sistema responde a essa pergunta em três dimensões concretas.

Na dimensão da **gestão operacional**, o *FashionBoost* centraliza em um único ambiente o cadastro de produtos com controle de estoque automatizado, o registro de vendas presenciais e o acompanhamento financeiro por meio de relatórios. Essas funcionalidades eliminam a necessidade de planilhas ou registros manuais, reduzindo erros e acelerando o atendimento ao cliente.

Na dimensão da **fidelização**, o programa de pontos configurável com progressão automática de níveis substitui processos manuais por um mecanismo preciso, auditável e reversível. O lojista passa a ter visibilidade sobre quais clientes estão mais engajados e pode criar cupons direcionados, tornando a fidelização uma prática acessível e estruturada.

Na dimensão da **inteligência estratégica**, o painel de IA transforma dados operacionais brutos em sugestões acionáveis em linguagem natural, oferecendo ao lojista acesso a um tipo de análise consultiva que antes exigiria contratação de especialistas ou investimento em ferramentas de *business intelligence* de alto custo.

A arquitetura multi-tenant demonstrou-se tecnicamente viável: o isolamento de dados por tenant é garantido em todas as camadas da aplicação, sem impacto perceptível na performance. O modelo de banco de dados com 13 entidades atendeu à complexidade do domínio sem redundâncias significativas, e todos os 12 requisitos funcionais e as 6 regras de negócio formalizados no SRS foram implementados e verificados.

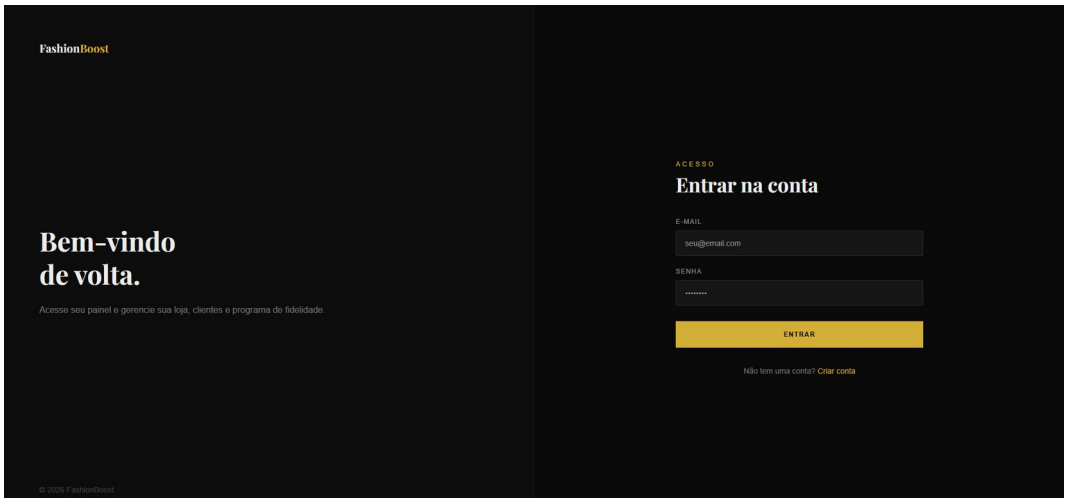
5.2 Principais Telas do Sistema

Esta seção apresenta as principais telas do *FashionBoost*, ilustrando a interface entregue ao lojista ao final do desenvolvimento. As telas demonstram a consistência visual proporcionada pelo uso do Tailwind CSS v4 em conjunto com a parametrização de cor por tenant.

5.2.1 Tela de Login

A tela de login é o ponto de entrada do *FashionBoost* para o lojista. Nela, o usuário informa seu e-mail e senha para acessar o painel da sua loja. O layout foi projetado para ser direto e sem elementos superflúos, dado que essa é uma tela de uso recorrente no dia a dia do estabelecimento. Após a validação das credenciais, o sistema gera o token JWT e redireciona o usuário ao painel principal da sua loja, já filtrado pelo seu identificador de tenant.

Figura 18 – Tela de login do *FashionBoost*

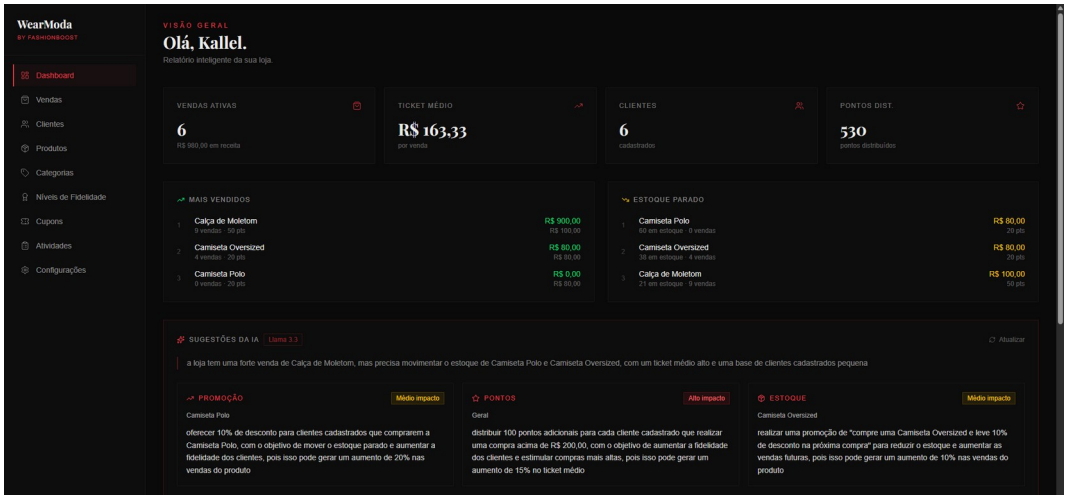


Fonte: O autor.

5.2.2 Dashboard

O dashboard é a tela principal do lojista após o login. Ele concentra os indicadores operacionais mais relevantes da loja — como total de vendas do período, número de clientes ativos, produtos com estoque baixo e distribuição dos clientes por nível de fidelidade —, oferecendo uma visão geral do desempenho do negócio sem a necessidade de navegar por múltiplas telas.

Figura 19 – Dashboard principal do *FashionBoost*



Fonte: O autor.

5.2.3 Gestão de Categorias

A tela de categorias permite ao lojista organizar o catálogo da loja por tipo de produto. Cada categoria pode ser criada, editada ou removida diretamente pelo painel, e os produtos cadastrados são vinculados a ela no momento do cadastro.

Figura 20 – Tela de gestão de categorias do *FashionBoost*

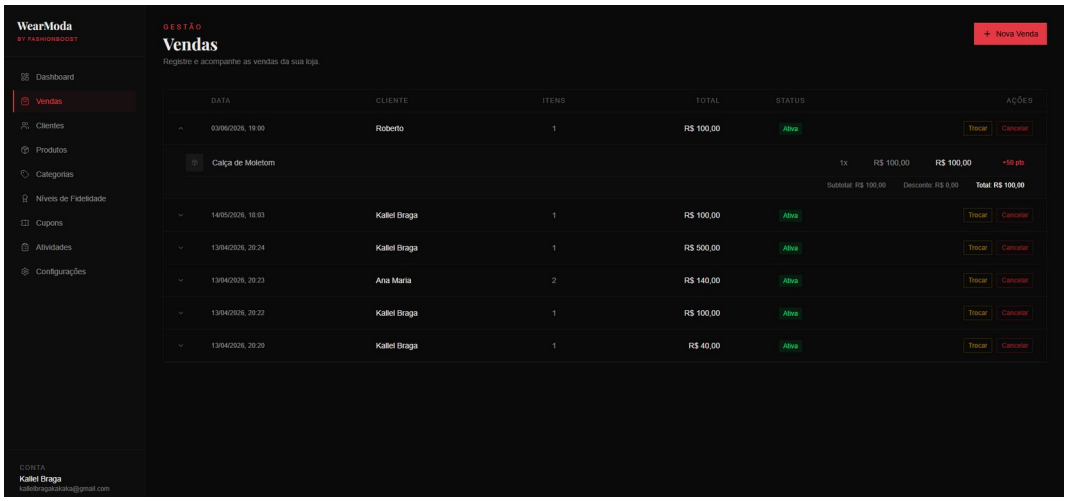
Fonte: O autor.

5.2.4 Registro de Vendas

A tela de vendas é o núcleo operacional do *FashionBoost* para o lojista ou funcionário. Por ela é possível selecionar os produtos adquiridos pelo cliente, aplicar um cupom de

desconto quando disponível, confirmar a venda e registrar automaticamente a atualização de estoque e o acúmulo de pontos no programa de fidelidade.

Figura 21 – Tela de registro de vendas do *FashionBoost*

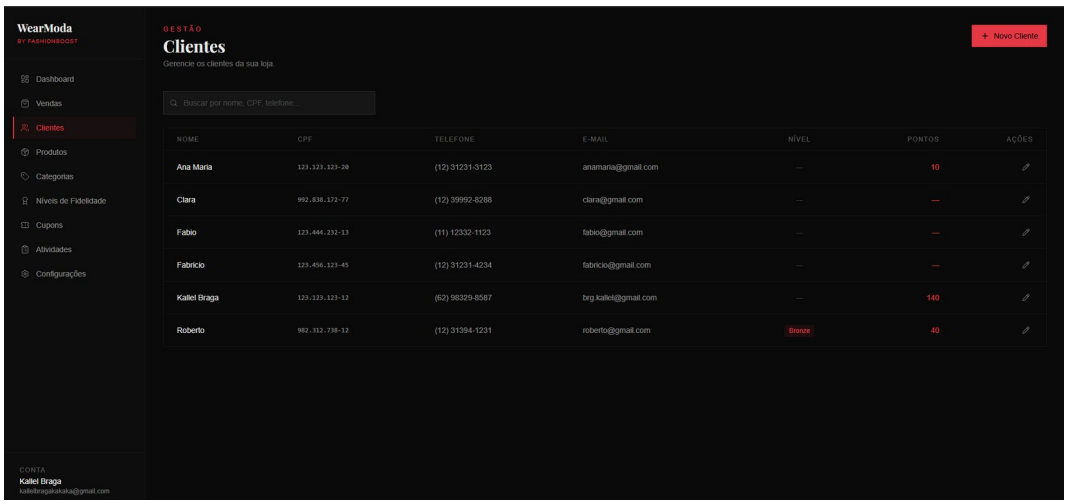


Fonte: O autor.

5.2.5 Gestão de Clientes

A tela de clientes exibe a base de clientes cadastrados na loja, com informações como nome, CPF, telefone, nível de fidelidade atual e pontuação acumulada. A partir dela, o lojista pode consultar o histórico de compras e de movimentações de pontos de cada cliente.

Figura 22 – Tela de gestão de clientes do *FashionBoost*



Fonte: O autor.

5.3 Comparativo com Trabalho Relacionado

Para contextualizar o posicionamento do *FashionBoost*, é relevante compará-lo com uma solução já consolidada no mercado que apresenta sobreposição funcional significativa com o sistema desenvolvido. O **Shopify POS** foi selecionado como referência de comparação por ser, entre as soluções analisadas, a que mais se aproxima do escopo do *FashionBoost*: ambos atendem ao varejo físico presencial, oferecem gestão de produtos e estoque, suportam registro de vendas e dispõem de mecanismos de fidelização de clientes (SHOPIFY, 2025).

O Shopify POS é uma solução internacional de ponto de venda desenvolvida pela Shopify, projetada para integrar operações de varejo físico e e-commerce em uma única plataforma. Oferece recursos avançados de gestão de produtos, controle de estoque, registro de vendas presenciais e acesso a programas de fidelidade por meio de aplicações de terceiros disponíveis em seu marketplace. A plataforma opera no modelo SaaS com planos mensais e é utilizada por lojistas em mais de 175 países (SHOPIFY, 2025).

Apesar da abrangência funcional, a comparação revela diferenças relevantes em relação ao *FashionBoost*. Em primeiro lugar, o programa de fidelidade no Shopify POS não é nativo: ele depende da instalação e configuração de aplicações de terceiros, como Smile.io ou Loyalty Lion, que possuem cobranças adicionais. No *FashionBoost*, o programa de fidelidade é parte integrante da plataforma, com configuração de níveis, regras de pontuação por produto, progressão automática e reversão de pontos no cancelamento.

Em segundo lugar, o Shopify POS não oferece um painel de inteligência artificial nativo. O *FashionBoost*, ao integrar o modelo Llama 3.3 70B via API Groq, oferece sugestões em linguagem natural categorizadas por área de impacto, reduzindo a carga interpretativa sobre o lojista.

Em terceiro lugar, há uma diferença significativa de acessibilidade. O Shopify POS exige a assinatura de um plano base cujos valores partem de aproximadamente US$ 29 mensais, acrescidos do custo das aplicações de fidelidade. Para lojistas de pequeno porte no contexto brasileiro, esse modelo de precificação em dólar representa uma barreira de entrada relevante.

Por fim, o Shopify POS é uma plataforma horizontal, voltada para o varejo em geral.

O *FashionBoost* é verticalizado para lojas de vestuário, com estruturas de dados orientadas às necessidades desse setor específico.

Tabela 2 – Comparação entre *FashionBoost* e Shopify POS

Critério	***FashionBoost***	**Shopify POS**
Gestão de produtos e estoque	Sim	Sim
Registro de vendas presenciais	Sim	Sim
Programa de fidelidade nativo	Sim	Não (via plugin)
Níveis de fidelidade configuráveis	Sim	Parcial
Reversão automática de pontos	Sim	Não
Painel de IA com sugestões	Sim	Não
Foco no setor de moda	Sim	Não
Precificação em moeda local	Sim	Não (dólar)
Complexidade de *onboarding*	Baixa	Alta

Fonte: O autor.

O comparativo evidencia que, embora o Shopify POS seja uma plataforma madura e amplamente adotada, ele não foi projetado para o perfil específico de uma pequena loja de moda física brasileira. O *FashionBoost* endereça diretamente esse conjunto de necessidades, demonstrando que um sistema SaaS verticalizado e acessível pode oferecer, em uma única plataforma, o que as soluções existentes entregam de forma fragmentada ou a um custo proibitivo.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o *FashionBoost*, uma plataforma SaaS multi-tenant voltada para proprietários de lojas de moda de pequeno e médio porte, que centraliza a gestão operacional — produtos, estoque, vendas e clientes — e a fidelização de clientes por meio de um programa de pontos configurável, com suporte de inteligência artificial para geração de sugestões estratégicas baseadas nos dados reais da loja.

O desenvolvimento do *FashionBoost* representou uma oportunidade concreta de aplicar, de forma integrada, conceitos modernos de engenharia de software — SaaS, multi-tenancy, autenticação *stateless* com JWT, programas de fidelidade configuráveis e inteligência artificial aplicada ao varejo — em um contexto de problema real e relevante para o varejo independente brasileiro.

Resposta à Questão de Pesquisa

A questão de pesquisa que orientou este trabalho — *Como o FashionBoost, enquanto um sistema SaaS, pode apoiar a gestão operacional e a fidelização de clientes em lojas de moda de pequeno e médio porte?* — pode ser respondida de forma afirmativa e analiticamente estruturada em três eixos.

O primeiro eixo é o da **viabilidade técnica**. O *FashionBoost* demonstrou que é possível construir, com tecnologias modernas e de código aberto, uma plataforma SaaS multi-tenant funcional que atende simultaneamente a múltiplos lojistas com isolamento completo de dados. A arquitetura adotada — separação por linha com filtragem automática por `tenantId` via JWT — provou-se eficiente e escalável, sem exigir infraestrutura dedicada por cliente. Todos os 12 requisitos funcionais e as 6 regras de negócio formalizados no SRS foram implementados e verificados, confirmando a viabilidade técnica da solução proposta.

O segundo eixo é o da **aderência ao problema**. O sistema responde diretamente às três características centrais do problema identificado na justificativa: a ausência de controle operacional, resolvida pelo módulo de gestão de produtos, estoque e vendas; a falta de mecanismos de fidelização, resolvida pelo programa de pontos com progressão automática e reversão atômica; e a inacessibilidade de ferramentas analíticas, resolvida pelo painel de IA que transforma dados operacionais em sugestões estratégicas em linguagem

natural. Cada funcionalidade implementada possui correspondência direta com um requisito levantado e com uma necessidade real do público-alvo.

O terceiro eixo é o da **diferenciação frente ao mercado**. O comparativo com o Shopify POS evidenciou que o *FashionBoost* ocupa uma lacuna concreta: é a única solução analisada que combina, em uma única plataforma acessível e verticalizada para o setor de moda, gestão operacional, programa de fidelidade nativo configurável e painel de inteligência artificial — sem dependência de *plugins* de terceiros e sem precificação em moeda estrangeira.

Portanto, o *FashionBoost* demonstrou, de forma concreta, que um sistema SaaS pode apoiar efetivamente a gestão operacional e a fidelização de clientes em lojas de moda de pequeno e médio porte, respondendo afirmativamente à questão de pesquisa proposta.

O objetivo geral foi alcançado: foi desenvolvida uma plataforma SaaS multi-tenant funcional, com interface moderna e conjunto completo de funcionalidades de gestão e fidelização. Os objetivos específicos também foram atendidos: o levantamento formal de requisitos seguindo o padrão IEEE foi realizado e documentado no SRS; o isolamento multi-tenant foi implementado em todas as camadas; o módulo de gestão de produtos e estoque opera de forma automatizada e reversível; o fluxo de vendas com cupons e cancelamento atômico garante consistência em qualquer cenário; o programa de fidelidade configurável funciona sem intervenção manual; e o painel de IA gerou sugestões contextualmente relevantes a partir dos dados operacionais reais.

A utilização do Lovable como ferramenta de prototipação e do Claude como assistente ao desenvolvimento mostrou-se uma abordagem eficaz, acelerando tanto a definição visual da interface quanto a implementação e revisão do código. O comparativo com o Shopify POS reforçou o posicionamento diferenciado do *FashionBoost*, que reúne em uma única plataforma acessível funcionalidades que na solução analisada estão fragmentadas, ausentes ou disponíveis apenas a um custo proibitivo para o público-alvo.

Limitações

Apesar dos resultados positivos, o trabalho apresenta limitações que devem ser reconhecidas. A primeira diz respeito à ausência de validação com usuários reais: o sistema foi desenvolvido e testado em ambiente controlado, sem a realização de testes de campo

com lojistas em operação, o que impede afirmar com certeza que a interface e os fluxos atendem plenamente às necessidades do público-alvo em condições reais de uso.

A segunda refere-se ao painel de inteligência artificial: a qualidade das sugestões geradas depende diretamente do volume e da maturidade dos dados da loja, tornando o recurso menos útil para estabelecimentos recém-cadastrados. Além disso, a dependência da API Groq representa um ponto de fragilidade arquitetural.

A terceira é a ausência de uma interface para o cliente final da loja: o programa de fidelidade é gerenciado exclusivamente pelo lojista, sem que o consumidor tenha acesso direto ao seu saldo de pontos ou histórico de compras.

Por fim, o sistema não contempla, na versão atual, integração com meios de pagamento, emissão de notas fiscais ou suporte a múltiplas unidades por lojista.

Uma limitação adicional diz respeito à **ausência de testes automatizados**. O *FashionBoost* não conta, na versão atual, com uma suíte de testes unitários ou de integração implementada. A validação das funcionalidades foi realizada de forma manual, por meio do Insomnia para os endpoints da API e da navegação direta no painel para o *front-end*. A implementação de testes automatizados — utilizando ferramentas como Jest para o *back-end* e Cypress ou Playwright para o *front-end* — representa uma necessidade relevante antes de qualquer implantação em ambiente de produção, garantindo rastreabilidade de regressões e maior confiabilidade nas evoluções futuras do sistema.

Trabalhos Futuros

Como trabalhos futuros, destacam-se: implementação de módulo de relatórios exportáveis em PDF e CSV; integração com gateways de pagamento para registro de transações financeiras; desenvolvimento de aplicativo móvel para o cliente final consultar seu saldo de fidelidade e histórico de pontos — atualmente mediado pelo lojista —, ampliando o alcance do programa de fidelidade para além do painel administrativo; suporte a múltiplas lojas por tenant; e testes de campo com lojistas reais para validação da experiência de uso em ambiente de produção.

O *FashionBoost* demonstrou-se tecnicamente consistente e viável enquanto projeto de TCC, estabelecendo uma base sólida para uma plataforma que, com evolução contínua, possui potencial real de aplicação no mercado de varejo de moda independente. O código-

fonte completo está disponível em: https://github.com/KallelBrg/FashionBoost .

REFERÊNCIAS

ANTHROPIC. **Claude: AI Assistant**. 2024. Disponível em: https://www.anthropic.com/claude. Acesso em: maio 2026.

BEZEMER, Cor-Paul; ZAIDMAN, Andy. Multi-tenant SaaS applications: maintenance dream or nightmare? In: **Proceedings of the Joint ERCIM Workshop on Software Evolution**. 2010.

BRASIL. **Lei n. 13.709, de 14 de agosto de 2018**. Lei Geral de Proteção de Dados Pessoais (LGPD). Brasília: Presidência da República, 2018. Disponível em: https://www.planalto.gov.br/ccivil_-03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: maio 2026.

BROWN, Tom et al. Language Models are Few-Shot Learners. **Advances in Neural Information Processing Systems**, v. 33, 2020.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa**. 6. ed. São Paulo: Atlas, 2017.

JONES, Michael B.; BRADLEY, John; SAKIMURA, Nat. **JSON Web Token (JWT) — RFC 7519**. IETF, 2015. Disponível em: https://datatracker.ietf.org/doc/html/rfc7519. Acesso em: nov. 2025.

KOTLER, Philip; KELLER, Kevin Lane. **Administração de Marketing**. 15. ed. São Paulo: Pearson, 2018.

KUMAR, V.; REINARTZ, Werner. **Customer Relationship Management: Concept, Strategy, and Tools**. 3. ed. Springer, 2018.

LOVABLE. **Lovable Documentation**. 2024. Disponível em: https://docs.lovable.dev. Acesso em: nov. 2025.

META AI. **Llama 3 Model Card**. 2024. Disponível em: https://ai.meta.com/research/publications/. Acesso em: nov. 2025.

MICROSOFT. **TypeScript Documentation**. Disponível em: https://www.typescriptlang.org/docs/. Acesso em: nov. 2025.

NESTJS. **Documentation**. Disponível em: https://docs.nestjs.com/. Acesso em: nov. 2025.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL Documentation**. Disponível em: https://www.postgresql.org/docs/. Acesso em: nov. 2025.

SEBRAE. **Inteligência Setorial: Varejo de Moda**. Brasília: Serviço Brasileiro de Apoio às Micro e Pequenas Empresas, 2024.

SHOPIFY. **Shopify POS**. 2025. Disponível em: https://www.shopify.com/pos. Acesso em: maio 2026.

TAILWIND CSS. **Documentation**. Disponível em: https://tailwindcss.com/docs. Acesso em: nov. 2025.

TYPEORM. **Documentation**. Disponível em: https://typeorm.io/. Acesso em: nov. 2025.

VERCEL. **Next.js Documentation**. Disponível em: https://nextjs.org/docs. Acesso em: nov. 2025.

WARD, Jonathan; BARKER, Adam. Undefined by data: a survey of big data definitions. **arXiv preprint**, 2014.

WAZLAWICK, Raul Sidnei. **Metodologia de pesquisa para ciência da computação**. Rio de Janeiro: Elsevier, 2014.

ANEXO — PROTÓTIPO DE REQUISITOS DE SOFTWARE

O documento a seguir apresenta uma prototipação de requisitos do *FashionBoost*, elaborada seguindo o padrão IEEE para *Software Requirements Specification* (SRS).

Protótipo de Requisitos para o FashionBoost

FASHIONBOOST: APLICAÇÃO PARA GESTÃO E FIDELIZAÇÃO PARA LOJAS

1. Introdução

Este documento descreve a especificação de requisitos do sistema FashionBoost, uma plataforma SaaS voltada para lojistas do setor de vestuário. O sistema tem como objetivo auxiliar lojas físicas no gerenciamento de produtos, registro de vendas presenciais e administração de programas de fidelidade baseados em pontos e níveis de recompensa.

Este documento segue a estrutura recomendada pelo padrão IEEE para especificação de requisitos de software, fornecendo uma visão clara das funcionalidades do sistema, regras de negócio e modelo de dados.

2. Descrição Geral

O FashionBoost é uma aplicação web que permite que lojistas criem e gerenciem suas lojas em um ambiente SaaS. A plataforma permite cadastrar produtos, registrar vendas presenciais, gerenciar clientes e administrar programas de fidelidade com pontuação acumulativa.

Clientes acumulam pontos ao realizar compras e podem atingir diferentes níveis de fidelidade (bronze, prata e ouro). Com base nesses pontos, cupons de benefício podem ser gerados e posteriormente validados durante o atendimento na loja.

3. Atores do Sistema

Lojista: responsável pela administração da loja dentro da plataforma, incluindo cadastro de produtos, configuração do programa de fidelidade e visualização de relatórios.

Funcionário da Loja: usuário autorizado a registrar vendas presenciais, consultar clientes e validar cupons de recompensa.

4. Requisitos Funcionais

1 **RF01** – O sistema deve permitir o cadastro de lojistas na plataforma.

2 **RF02** – O sistema deve permitir autenticação segura de usuários.

3 **RF03** – O sistema deve permitir o cadastro e configuração de lojas.

4 **RF04** – O sistema deve permitir o cadastro de categorias de produtos.

5 **RF05** – O sistema deve permitir o cadastro de produtos com preço, estoque e pontuação.

6 **RF06** – O sistema deve permitir o cadastro de clientes utilizando CPF como identificador.

7 **RF07** – O sistema deve permitir o registro de vendas presenciais.

8 **RF08** – O sistema deve calcular automaticamente os pontos gerados por cada venda.

9 **RF09** – O sistema deve permitir configurar níveis de fidelidade.

10 **RF10** – O sistema deve permitir gerar cupons utilizando pontos acumulados.

11 **RF11** – O sistema deve permitir validar cupons apresentados pelos clientes.

12 **RF12** – O sistema deve permitir a geração de relatórios de vendas.

5. Regras de Negócio

1 **RN01** – Cada cliente deve ser identificado por um CPF único dentro da loja.

2 **RN02** – Os pontos recebidos por uma compra devem ser calculados com base na pontuação definida para cada produto.

3 **RN03** – Os níveis de fidelidade devem ser definidos pelo lojista e podem incluir Bronze, Prata e Ouro.

4 **RN04** – Um cliente só pode atingir um nível de fidelidade quando atingir a pontuação mínima configurada.

5 **RN05** – Cupons só podem ser utilizados uma única vez após sua geração.

6 **RN06** – Apenas usuários autenticados podem registrar vendas ou validar cupons.

6. Requisitos Não Funcionais

1 **NF01** – O sistema deve garantir autenticação segura de usuários.

2 **NF02** – O sistema deve apresentar desempenho adequado para operações comerciais.

3 **NF03** – A interface deve ser simples e intuitiva.

4 **NF04** – O sistema deve suportar múltiplas lojas simultaneamente (multi-tenant).

7. Diagrama de Casos de Uso (Descrição)

Os principais casos de uso do sistema incluem:

- Gerenciar Loja
- Cadastrar Produtos
- Cadastrar Clientes
- Registrar Venda
- Calcular Pontos de Fidelidade
- Gerar Cupom de Recompensa
- Validar Cupom
- Consultar Relatórios de Vendas

Os atores Lojista e Funcionário da Loja interagem com essas funcionalidades por meio do painel administrativo do sistema.

8. Modelo de Dados Resumido

Entidade	Descrição
Tenant	Conta do lojista dentro da plataforma SaaS.
User	Usuários que acessam o painel administrativo.
Store	Informações da loja gerenciada pelo lojista.

Category	Categorias utilizadas para organizar produtos.
Product	Produtos cadastrados pela loja.
Customer	Clientes da loja física identificados por CPF.
Sale	Registro de vendas realizadas presencialmente.
SaleItem	Itens associados a cada venda.
LoyaltyLevel	Níveis de fidelidade configurados pela loja.
CustomerLoyalty	Conta de fidelidade do cliente.
Coupon	Cupom gerado a partir da troca de pontos.
PointsTransaction	Histórico de movimentação de pontos.