Gabriela Helena Demori

NEXOERP: INTEGRAÇÃO E AUTOMAÇÃO PARA PMES ATRAVÉS DE UM SISTEMA ERP SIMPLIFICADO

Amparo

2025

Gabriela Helena Demori

NEXOERP: INTEGRAÇÃO E AUTOMAÇÃO PARA PMES ATRAVÉS DE UM SISTEMA ERP SIMPLIFICADO

Monografia apresentada ao curso de Desenvolvimento de Sistemas do Serviço Nacional de Aprendizagem Industrial como requisito parcial à obtenção do título de Desenvolvedor(a) de Sistemas Junior.
Área de concentração: Tecnologia da Informação e Gestão
Orientador: Welington F. O. Martins

Amparo

2025

DEDICATÓRIA

Dedico este Trabalho de Conclusão de Curso àqueles que contribuíram de forma significativa para sua realização, seja por meio de apoio intelectual, colaboração direta ou incentivo durante o processo.

À minha mãe, Tamara, por ser uma referência de determinação, disciplina e conhecimento. Sua orientação, tanto pessoal quanto acadêmica, foi fundamental para o desenvolvimento e finalização deste trabalho.

Ao colega Guilherme, pelo suporte técnico na área de programação e banco de dados, mesmo diante de seus próprios compromissos acadêmicos. Sua colaboração foi essencial em momentos decisivos do projeto.

Dedico ainda a todos que, de maneira direta ou indireta, influenciaram este percurso, mesmo que por meio de críticas, desafios ou circunstâncias adversas. Cada contribuição, em sua medida, fez parte da construção deste trabalho.

RESUMO

A competitividade do mercado exige que empresas, especialmente as de pequeno e médio porte (PMEs), busquem soluções tecnológicas que otimizem sua gestão e melhorem seus processos. Nesse cenário, os sistemas ERP (*Enterprise Resource Planning*) têm se destacado por integrar diversas áreas da empresa em uma única plataforma, facilitando o controle e a tomada de decisões. No entanto, muitos ERPs disponíveis no mercado são complexos, caros ou não se adaptam à realidade dessas empresas menores.

Este trabalho apresenta o desenvolvimento de um ERP simplificado, voltado especificamente para PMEs, com foco na automação de processos e na agilidade operacional. O sistema permite a gestão de usuários, clientes, produtos, vendas, estoque e geração de relatórios de forma integrada e intuitiva. A metodologia utilizada no desenvolvimento foi a ágil, com ênfase no *framework Scrum*, permitindo uma construção iterativa, colaborativa e eficiente da aplicação.

Ao final, o projeto busca entregar uma solução prática e acessível, que contribua diretamente para a organização e produtividade das pequenas empresas, reduzindo tarefas manuais e melhorando o fluxo de informações no ambiente empresarial.

Palavras-chave: ERP; PMEs; Gestão; Automação; Scrum.

ABSTRACT

The competitiveness of the market requires companies, especially small and medium-sized ones (SMEs), to look for technological solutions that optimize their management and improve their processes. In this scenario, ERP (Enterprise Resource Planning) systems have stood out for integrating various areas of the company into a single platform, facilitating control and decision-making. However, many ERPs available on the market are complex, expensive or do not adapt to the reality of these smaller companies.

This paper presents the development of a simplified ERP, aimed specifically at SMEs, with a focus on process automation and operational agility. The system allows users, customers, products, sales, stock and reports to be managed in an integrated and intuitive way. The methodology used in the development was agile, with an emphasis on the Scrum framework, allowing for an iterative, collaborative and efficient construction of the application.

In the end, the project aims to deliver a practical and accessible solution that contributes directly to the organization and productivity of small businesses, reducing manual tasks and improving the flow of information in the business environment.

Keywords: ERP; SMEs; Management; Automation; Scrum.

LISTA DE ILUSTRAÇÕES

FIGURAS

DIAGRAMAS

TABELAS

SUMÁRIO

1 INTRODUÇÃO

No dinâmico e implacável cenário do século XXI, a competitividade de mercado não é apenas um desafio, mas um imperativo que redefine as estratégias empresariais. Em meio a essa efervescência, as Pequenas e Médias Empresas (PMEs) emergem como protagonistas vitais da economia, embora frequentemente se deparem com a necessidade urgente de otimizar sua gestão e aprimorar seus processos para garantir sua sobrevivência e expansão. É nesse panorama que a tecnologia se posiciona não como um luxo, mas como uma ferramenta indispensável. Nesse contexto, os sistemas ERP (*Enterprise Resource Planning*) têm se consolidado como pilares da gestão moderna, prometendo a integração holística de diversas áreas de uma organização em uma única plataforma, o que, em teoria, facilita o controle operacional e qualifica a tomada de decisões estratégicas (Davenport, 1993; Porter, 1985).

Contudo, a realidade para muitas PMEs é paradoxal. Embora o valor intrínseco dos ERPs seja inegável, o mercado é dominado por soluções que, em sua maioria, são intrinsecamente complexas, proibitivamente caras ou excessivamente robustas, não se adaptando à agilidade e às especificidades do cotidiano dessas empresas de menor porte. Essa lacuna tecnológica impede que um vasto contingente de negócios brasileiros e globais alcance seu potencial máximo, perpetuando a dependência de processos manuais, a fragmentação de dados e a ineficiência operacional, que se traduzem em perda de tempo e recursos valiosos.

Diante desse desafio premente, este trabalho apresenta o desenvolvimento de um ERP simplificado, meticulosamente concebido para as necessidades e a escala das PMEs. O foco central reside na automação de processos e na agilidade operacional, buscando desmistificar a implementação de sistemas de gestão integrada. O sistema proposto oferece uma interface intuitiva para a gestão de usuários, clientes, produtos, vendas, estoque e a geração de relatórios cruciais, consolidando informações que antes estariam dispersas. Para garantir a robustez, a adaptabilidade e a eficiência do desenvolvimento, a metodologia adotada foi a ágil, com uma forte ênfase no *framework Scrum* (Schwaber & Sutherland, 2017). Essa abordagem iterativa e colaborativa permitiu uma construção flexível, responsiva às mudanças e focada na entrega contínua de valor.

Em sua essência, este projeto transcende a mera criação de um software; ele aspira a ser um catalisador de transformação para as pequenas empresas. Ao entregar uma solução prática, acessível e verdadeiramente alinhada à realidade das PMEs, o NexoERP busca contribuir diretamente para a organização interna e o aumento da produtividade. A redução de tarefas manuais e a melhoria substancial do fluxo de informações no ambiente empresarial não apenas otimizam o dia a dia, mas também capacitam essas empresas a se posicionarem de forma mais competitiva no mercado**,** pavimentando o caminho para um crescimento sustentável e inovador.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas Integrados De Gestão Empresarial (ERP)

Sistemas Integrados de Gestão Empresarial, ou *Enterprise Resource Planning* (ERP), são plataformas de *software* projetadas para gerenciar e integrar as principais áreas e processos de uma

empresa em um único sistema unificado. O objetivo central de um ERP é fornecer uma visão holística e em tempo real das operações, permitindo que os gestores tomem decisões mais informadas e otimizem o desempenho organizacional.

A integração é o princípio fundamental do ERP. Ao consolidar dados de diversas áreas — como vendas, finanças, estoque e recursos humanos — em um banco de dados centralizado, o sistema elimina a duplicação de informações e garante a consistência e a rastreabilidade dos dados em toda a cadeia de valor.

2.1.1 Histórico e evolução dos ERPs

A trajetória dos ERPs é marcada pela expansão progressiva das funcionalidades de gestão. O ponto de partida foi na década de 1960, com o MRP (*Material Requirements Planning*), focado exclusivamente no planejamento e controle de estoque e produção.

Na década de 1980, o conceito evoluiu para o MRP II (*Manufacturing Resource Planning*), que adicionou a capacidade de integrar planejamento financeiro e capacidade de produção. O termo "ERP" surgiu no início dos anos 90, cunhado pelo Gartner Group, para descrever sistemas que expandiram a funcionalidade do MRP II para cobrir todos os aspectos do negócio, incluindo vendas, *marketing* e serviços.

Com o advento da *internet*, os ERPs passaram a adotar arquiteturas baseadas em navegadores, facilitando a acessibilidade e a redução dos custos de infraestrutura. Mais recentemente, a evolução se concentra em soluções em nuvem (*Cloud ERP*) e sistemas modulares, permitindo maior flexibilidade e adaptação às necessidades específicas de cada segmento de mercado.

2.1.2 Arquiteturas de sistemas ERP

A arquitetura de um sistema ERP define como os componentes de *software* são organizados e como eles interagem. As arquiteturas mais relevantes incluem:

1. Arquitetura Cliente-Servidor (Legada): Onde a interface do usuário (cliente) se comunica diretamente com o servidor de aplicação e o servidor de banco de dados. Embora robusta, é complexa de manter e pouco flexível para acesso remoto.
2. Arquitetura em Três Camadas (3-*Tier*): É o padrão mais adotado em soluções modernas. É dividida em:
    - Camada de Apresentação (*Frontend*): Responsável pela interface do usuário.
    - Camada de Aplicação (*Backend*): Contém a lógica de negócios e processamento.
    - Camada de Dados (Banco de Dados): Onde os dados são armazenados. Essa separação facilita a manutenção, a escalabilidade e a distribuição do sistema, sendo a base da arquitetura *Web* utilizada neste projeto.

2.2 A gestão empresarial em pequenas e médias empresas (PME)

As PMEs representam uma parcela vital da economia, mas enfrentam desafios de gestão distintos das grandes corporações, exigindo soluções de *software* mais adequadas à sua realidade.

2.2.1 Desafios Comuns De Gestão

A adoção de sistemas de gestão em PMEs é frequentemente dificultada por:

- Custos: O alto custo de licença e implantação de ERPs tradicionais é proibitivo.
- Complexidade: Sistemas complexos exigem treinamento intensivo e recursos de TI que as PMEs geralmente não possuem.
- Decisão Fragmentada: O uso de planilhas e softwares não integrados (sistemas departamentais) leva a erros manuais, duplicação de informações e dificuldade na consolidação de relatórios.
- Foco no *Core Business*: O tempo da equipe é limitado, e soluções que exigem muita manutenção desviam o foco das atividades principais da empresa.

2.2.2 Requisitos de um ERP Simplificado para PMEs

Para ser eficaz em uma PME, um ERP deve priorizar:

1. Modularidade e Foco: Concentrar-se nos módulos essenciais para a sobrevivência do negócio (Vendas, Clientes e Estoque), evitando funcionalidades desnecessárias.
2. Usabilidade (UI/UX): Ter uma interface de usuário intuitiva e um baixo custo de curva de aprendizado.
3. Acessibilidade: Ser baseado em tecnologia web para acesso de qualquer dispositivo e local (*Cloud-Ready*).
4. Custo-Benefício: Utilizar tecnologias de código aberto (*Open Source*) ou de baixo custo para tornar a solução economicamente viável.
5. Controle Operacional: Garantir rastreabilidade completa das transações (quem vendeu, o que foi vendido, para quem).

3 METODOLOGIA E ARQUITETURA DO SISTEMA

Este capítulo descreve a abordagem metodológica utilizada na pesquisa e no desenvolvimento do ERP Simplificado e detalha a estrutura lógica e física da solução.

3.1 Tipo e Natureza da Pesquisa

A pesquisa que embasa este trabalho possui uma natureza aplicada, pois tem como objetivo principal o desenvolvimento de uma solução tecnológica prática para um problema real de gestão em PMEs. Em relação à forma de abordagem, é predominantemente quali-quantitativa,

utilizando dados qualitativos (necessidades de PMEs) para o *design* e dados quantitativos (testes e validação) para a avaliação da eficácia.

Quanto aos objetivos, trata-se de um projeto de desenvolvimento experimental, que culmina na criação de um artefato de *software*, sendo também descritiva na etapa de análise de requisitos e tecnologias.

3.2 Processo de Desenvolvimento De Software

Para o gerenciamento e execução do projeto, foi adotado um Modelo de Desenvolvimento Ágil, especificamente o *Scrum*, devido à sua capacidade de lidar com requisitos emergentes e entregar valor em ciclos curtos (*Sprints*).

3.2.1 Modelo de Desenvolvimento Adotado (Scrum)

O uso do Scrum permitiu que o desenvolvimento fosse dividido em fases iterativas e incrementais, com foco na priorização das funcionalidades mais críticas (Módulo de Vendas e Cadastro Básico) e na rápida obtenção de *feedback*. Isso garante que o produto final seja prático e útil para o público-alvo (PMEs), ao invés de ser excessivamente complexo.

3.2.2 Ferramentas de Gerenciamento de Projeto e Versionamento

O controle do código-fonte e o histórico de alterações foram mantidos utilizando o Git como sistema de versionamento distribuído. Os repositórios foram hospedados em plataformas como o GitHub/GitLab, garantindo a segurança do código e facilitando o controle de versões.

3.3 Stack Tecnológico e Ambiente de Desenvolvimento

O ambiente de desenvolvimento e as tecnologias selecionadas buscam o equilíbrio entre performance, produtividade e baixo custo de infraestrutura, ideal para o contexto das PMEs.

3.3.1 Linguagens e Frameworks (Node.js, JavaScript, HTML/CSS)

O sistema foi construído sobre o JavaScript no modelo *Full-Stack*, utilizando o Visual Studio Code (VSCode) como Ambiente de Desenvolvimento Integrado (IDE) principal para codificação e depuração.

- *Backend*: A camada de aplicação foi desenvolvida utilizando o ambiente de execução Node.js. Essa escolha permite a criação de uma *API RESTful* de alta performance, capaz de gerenciar a lógica de negócios de forma assíncrona e eficiente, utilizando o JavaScript como linguagem de programação no servidor.
- *Frontend*: A interface do usuário (UI) foi desenvolvida utilizando HTML e estilizada com CSS. Essa abordagem de código limpo e nativo garante a máxima compatibilidade e leveza da aplicação, sendo ideal para um ERP simplificado que prioriza a velocidade e a baixa dependência de bibliotecas complexas no lado do cliente. O JavaScript puro é empregado para adicionar a interatividade e manipular o DOM.

3.3.2 Ferramentas de Bancos de Dados (MySQL, Prisma)

- Banco de Dados: O MySQL foi escolhido como o SGBD relacional, sendo amplamente conhecido por sua velocidade, confiabilidade e ser uma solução de código aberto, ideal para o cenário de PMEs.
- Ambiente Local: Durante o desenvolvimento local, foi utilizado o XAMPP para facilitar a configuração e gerenciamento de um servidor MySQL, além de fornecer um ambiente de testes local para a aplicação *backend*.
- ORM: O *framework* Prisma é utilizado como Object-Relational Mapper, conectando *backend* Node.js ao MySQL. O Prisma simplifica as operações de banco de dados e mantém o modelo de dados definido de forma declarativa.

3.3.3 Ferramentas de Infraestrutura e Implantação

Para o controle de ambiente e a entrega contínua do *software*, foi utilizada a plataforma Vercel. Esta plataforma oferece hospedagem otimizada para o *Frontend* (HTML/CSS/JS) e a gestão de funções *serverless* para a aplicação *backend* (Node.js), permitindo uma implantação rápida e eficiente do sistema para os ambientes de teste e produção.

3.4 Arquitetura da Solução Proposta

O ERP Simplificado adota a Arquitetura em Três Camadas (3-*Tier*) em um modelo Cliente-Servidor distribuído, ideal para aplicações *web* que exigem escalabilidade e separação de responsabilidades.

3.4.1 Arquitetura em Camadas (Frontend, Backend, Banco De Dados)

1. Camada de Apresentação (*Frontend*): Implementada com HTML, CSS e JavaScript, é responsável por toda a interação com o usuário, exibição de dados e captura de *inputs*. Ela se comunica exclusivamente com a Camada de Aplicação por meio de requisições HTTP (AJAX/Fetch).
2. Camada de Aplicação (*Backend*): Implementada com Node.js/Express, contém toda a lógica de negócio do ERP. Processa requisições, aplica regras (ex: baixa de estoque) e gerencia a comunicação com a Camada de Dados.
3. Camada de Dados (Banco de Dados): Composta pelo MySQL e gerenciada pelo Prisma. É responsável pela persistência, integridade e recuperação dos dados.

3.4.2 Apis e Comunicação De Dados

A comunicação entre o *Frontend* e o *Backend* é realizada por meio de uma API (*Application Programming Interface*) *RESTful*. Esta API define um conjunto de rotas (*endpoints*) que utilizam verbos HTTP (GET, POST, PUT, DELETE) para realizar operações CRUD (*Create, Read, Update, Delete*) nas entidades do sistema (Usuário, Cliente, Produto, Venda). O formato de troca de dados é o JSON (*JavaScript Object Notation*), amplamente aceito e eficiente.

4 MODELAGEM, IMPLEMENTAÇÃO E DESENVOLVIMENTO DO BACKEND

4.1 Modelagem e Estrutura do Banco de Dados

4.1.1 Estrutura das Tabelas e Relacionamentos

O modelo de dados do sistema foi cuidadosamente estruturado para garantir integridade, rastreabilidade e flexibilidade, atendendo às necessidades de pequenas e médias empresas. O design segue o modelo relacional, onde as entidades do negócio são representadas por tabelas interconectadas por chaves primárias e estrangeiras.

Tabela Usuario Armazena informações dos usuários, incluindo perfil para controle de acesso.

- Campos Chave e de Identificação: id (PK), email (Único), nome, senha.

Tabela Cliente Registra os dados dos clientes, físicos ou jurídicos.

- Campos Chave e de Identificação: id (PK), nome, email (Único), telefone, cpf (Único) e cnpj.
- Vinculação: usuarioId (FK) estabelece uma relação 1:N com Usuario.

Tabela Produto Armazena os produtos ou serviços oferecidos pela empresa.

- Campos Chave e de Identificação: id (PK), nome, descricao, preco e estoque.

Tabela Venda Representa cada transação comercial realizada pela empresa.

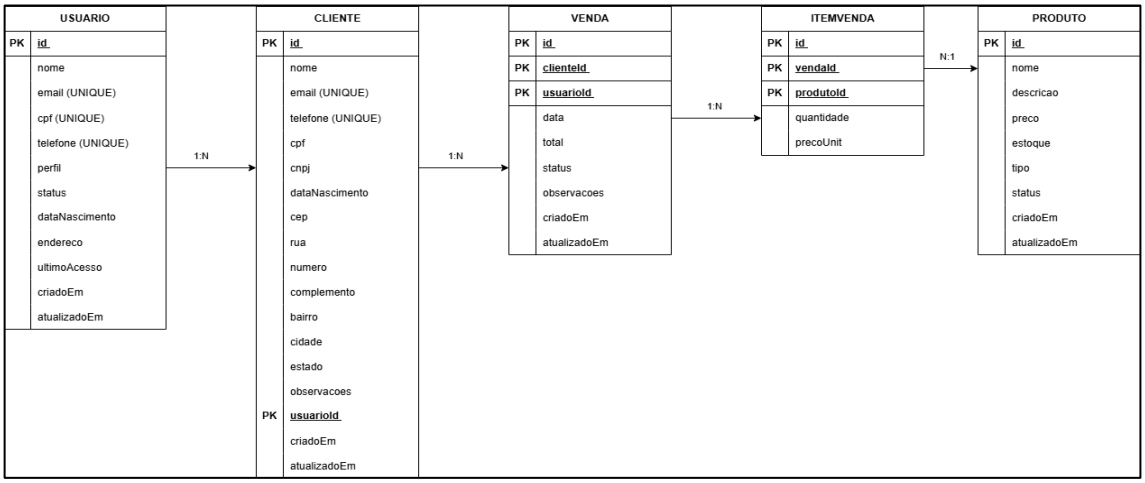
- Campos Chave e de Identificação: id (PK), data e total.
- Relacionamentos: clienteId (FK) e usuarioId (FK) estabelecem relações 1:N com Cliente e Usuario, respectivamente.

Tabela ItemVenda Tabela associativa que detalha os itens de uma venda, resolvendo o relacionamento N:M (Muitos para Muitos) entre Venda e Produto.

- Campos Chave e de Identificação: id (PK), quantidade e precoUnit.
- Relacionamentos: vendaId (FK) e produtoId (FK) estabelecem relações 1:N com Venda e Produto, respectivamente.

A modelagem proposta permite que o sistema gerencie de forma eficiente e segura todas as informações essenciais para o funcionamento de uma PME.

DIAGRAMA 1 - *Estrutura e Relacionamentos das Entidades do Banco de Dados*



4.1.2 Detalhamento do Schema Físico

O *schema* físico detalha a implementação das entidades no banco de dados, garantindo a aplicação das regras de integridade conforme especificado no Prisma.

4.1.3 Scripts de Criação e Migração

O *framework* Prisma é utilizado como ORM (*Object-Relational Mapper*) para gerenciar o ciclo de vida do banco de dados, gerando automaticamente os scripts SQL necessários (DDL - *Data Definition Language*) a partir do arquivo schema.prisma.

A seguir, um trecho simplificado do DDL gerado pelo Prisma que ilustra a criação de tabelas e a definição de chaves estrangeiras:

FIGURA 1 – *Trecho gerado pelo prisma para criação de tabelas*

Para aplicar essas alterações ao banco de dados, o desenvolvedor executa o seguinte comando na CLI do Prisma:

FIGURA 2 – *Comando para definir/modificar o arquivo schema.prisma*

```
npx prisma migrate dev
```

Este comando gera e executa automaticamente as migrações necessárias, atualizando as tabelas e relacionamentos conforme o modelo definido, e mantendo um histórico das alterações.

4.1.4 Garantia De Integridade E Unicidade

O uso de *constraints* (restrições) no modelo de dados é fundamental para garantir a integridade e a confiabilidade das informações, seguindo os princípios de integridade de entidade, referencial e de domínio.

- Integridade de Entidade e Unicidade: As Chaves Primárias (*PRIMARY KEY*) garantem a unicidade de cada registro. Os Índices Únicos (*UNIQUE*) são aplicados a campos críticos, como email (em Usuario e Cliente) e cpf (em Cliente), para evitar duplicidade de dados cadastrais.
- Integridade Referencial: As Chaves Estrangeiras (*FOREIGN KEY*) estabelecem os vínculos essenciais entre as tabelas (usuarioId em Cliente, clienteId e usuarioId em Venda, etc.). As regras de exclusão e atualização são configuradas para manter a coerência, sendo restritas (*RESTRICT*) em pontos que não podem perder o histórico de dados.
- Integridade de Domínio: É garantida pelo uso de tipos de dados adequados (ex: DECIMAL para valores monetários) e a definição de Valores Não Nulos (*NOT NULL*) para campos obrigatórios, como chaves estrangeiras.

4.2 Desenvolvimento do Projeto Backend

4.2.1 Estrutura do Projeto Backend

A organização das pastas do projeto *backend* (Node.js) segue uma estrutura fundamental para clareza, manutenção e escalabilidade:

- *controllers*: Centraliza a lógica de negócio, processando requisições, validando dados e interagindo com o banco de dados (via Prisma). Ex: usuarioController, produtoController.
- *routes*: Define os *endpoints* da API e associa cada rota a um *controller* específico.
- *models*: Contém as definições das entidades do sistema no schema.prisma.
- config: Armazena arquivos de configuração essenciais, como variáveis de ambiente e parâmetros de inicialização do servidor.

4.2.2 Implementação dos Endpoints Restful

A estrutura de *endpoints* da API RESTful é baseada no conceito CRUD e organizada por entidade:

Usuário

- *POST* /usuarios – Cria um novo usuário.
- *GET* /usuarios – Retorna a lista de todos os usuários cadastrados.
- *GET* /usuarios/:id – Retorna os dados de um usuário específico.
- *PUT* /usuarios/:id – Atualiza as informações de um usuário existente.
- *DELETE* /usuarios/:id – Remove um usuário do sistema.

Cliente

- *POST* /clientes – Cadastra um novo cliente.
- *GET* /clientes – Lista todos os clientes registrados.
- *GET* /clientes/:id – Retorna os dados de um cliente específico.
- *PUT* /clientes/:id – Atualiza as informações de um cliente existente.
- *DELETE* /clientes/:id – Exclui um cliente do sistema.

Produto

- *POST* /produtos – Cadastra um novo produto.
- *GET* /produtos – Lista todos os produtos cadastrados.
- *GET* /produtos/:id – Retorna os dados de um produto específico.
- *PUT* /produtos/:id – Atualiza as informações de um produto.
- *DELETE* /produtos/:id – Remove um produto do sistema.

Venda

- *POST* /vendas – Registra uma nova venda.
- *GET* /vendas – Lista todas as vendas realizadas.
- *GET* /vendas/:id – Retorna os dados de uma venda específica.
- *PUT* /vendas/:id – Atualiza os dados de uma venda.
- *DELETE* /vendas/:id – Remove uma venda do sistema.

ItemVenda

- *POST* /itensvenda – Adiciona um item a uma venda.
- *GET* /itensvenda – Lista todos os itens de venda registrados.
- *GET* /itensvenda/:id – Retorna os dados de um item de venda específico.
- *PUT* /itensvenda/:id – Atualiza as informações de um item de venda.

- *DELETE* /itensvenda/:id – Remove um item de venda do sistema.

4.3 Testes de Api com Insomnia

O sistema foi testado extensivamente utilizando a ferramenta Insomnia para simular operações reais, garantindo que os *endpoints* da API RESTful implementados no *backend* estivessem funcionais e aderentes aos requisitos de negócio.

4.3.1 Metodologia de Teste e Agrupamento

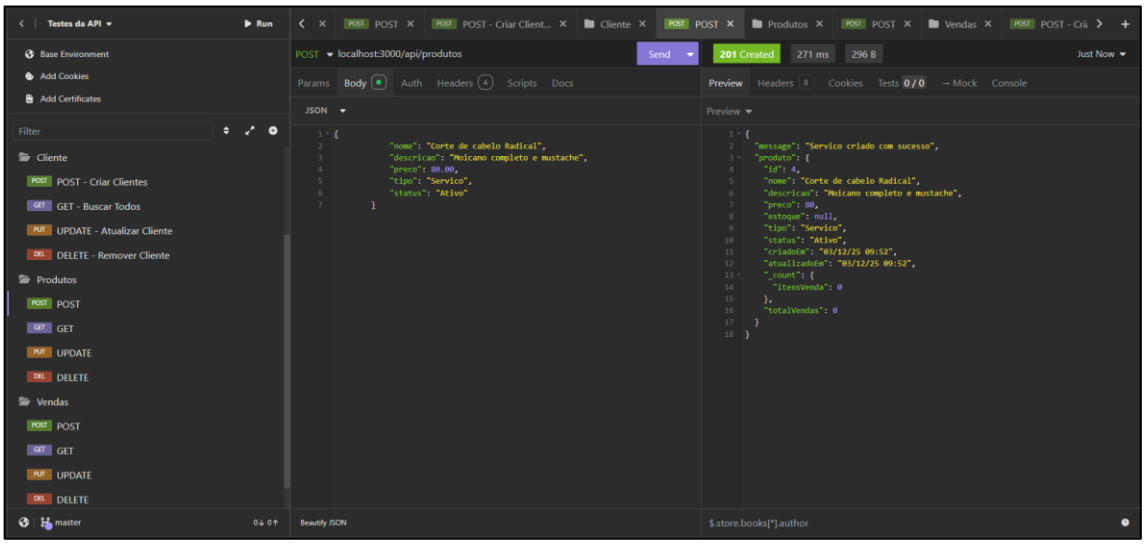
As requisições foram agrupadas modularmente por entidade (Usuário, Cliente, Produto, Venda) dentro do Insomnia. Cada grupo continha as quatro operações básicas (CRUD): POST para criação, GET para leitura, PUT para atualização e DELETE para remoção. Os testes focaram na validação do código de status HTTP e no corpo da resposta (JSON) em diferentes cenários (sucesso e falha).

4.3.2 Testes da Entidade Produto (Crud)

O teste da entidade Produto validou a gestão completa do inventário.

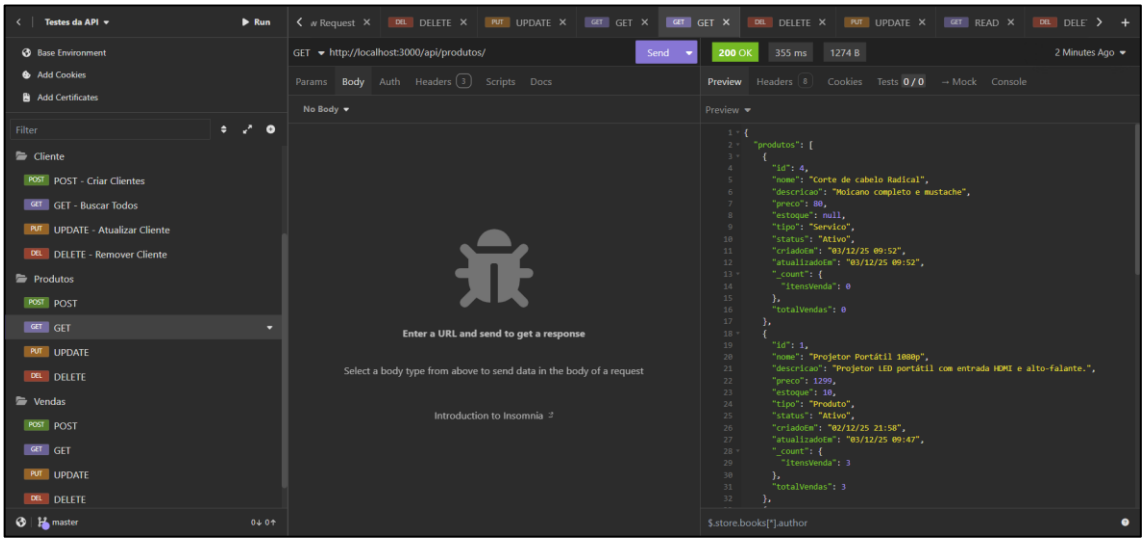
1. Criação (POST /produtos): Validação da inserção de um novo produto no banco. Resultado Esperado: Código de status HTTP 201 Created.

FIGURA 3 – *Requisição de Cadastro de Produto via Insomnia (POST)*



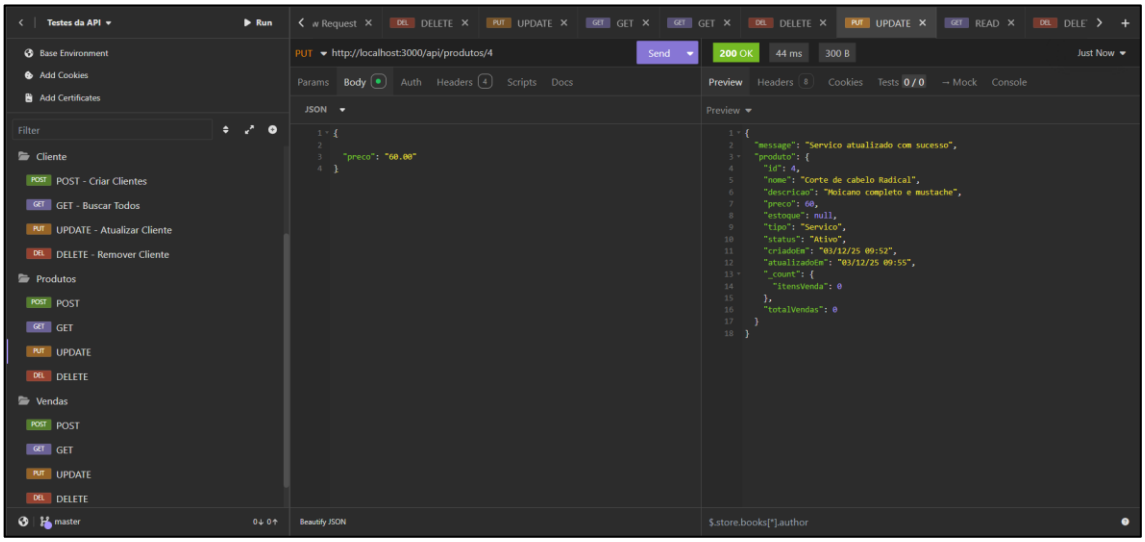
2. Leitura (GET/produtos): Validação da recuperação de um registro específico. Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 4 – *Requisição de Leitura de Produto via Insomnia (GET)*



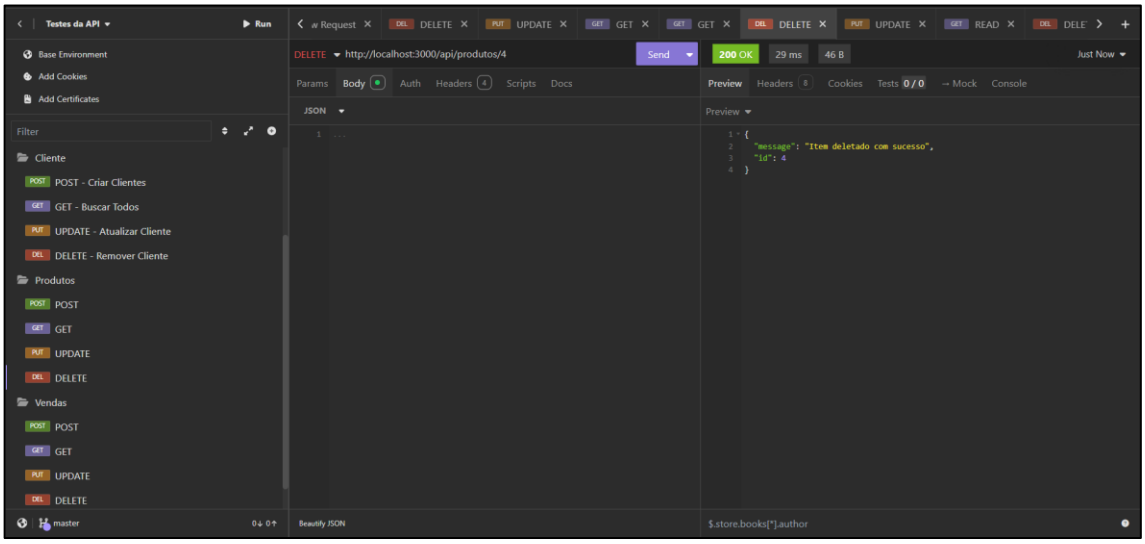
3. Atualização (PUT /produtos/:id): Validação da modificação de campos (ex: preço e estoque) de um produto existente. Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 5 – *Requisição de Atualização de Produto via Insomnia (PUT)*



4. Exclusão (DELETE /produtos/:id): Validação da remoção lógica ou física do produto. Resultado Esperado: Código de status HTTP 200 OK (ou 204 No Content).

FIGURA 5 – *Requisição de Exclusão de Produto via Insomnia (DELETE)*

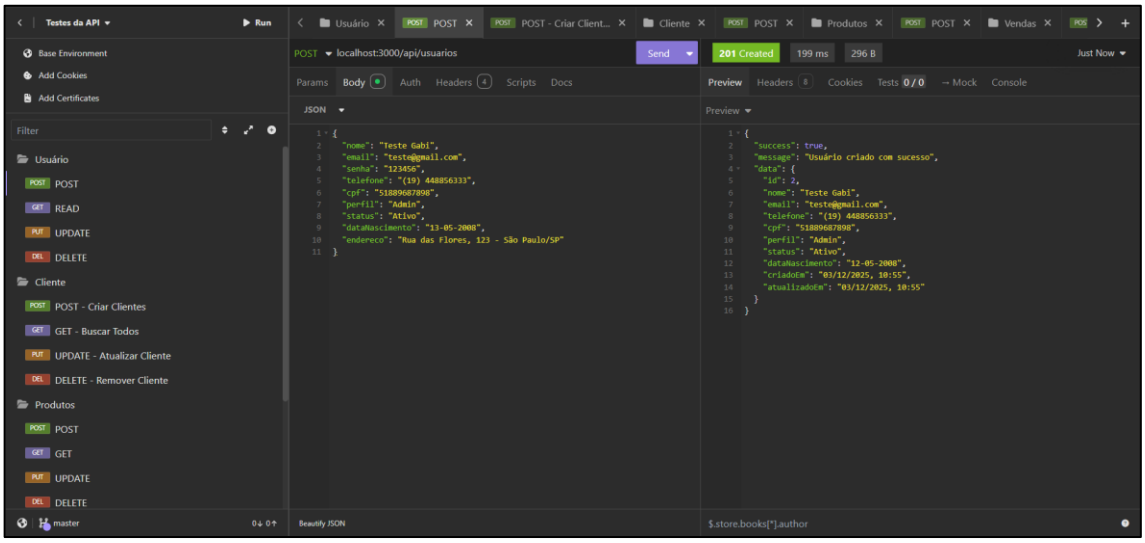


4.3.3 Testes Da Entidade Usuário (Crud)

A entidade Usuario foi testada para garantir a segurança e o controle de acesso, simulando o cadastro de novos vendedores ou administradores.

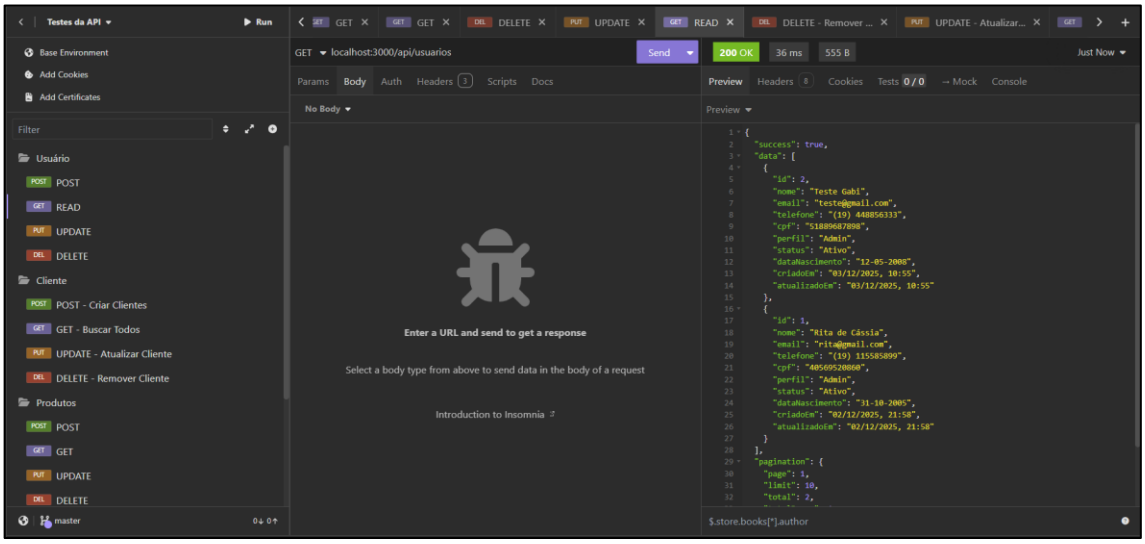
1. Criação (POST /usuarios): Validação da criação de um novo usuário. Resultado Esperado: Código de status HTTP 201 Created.

FIGURA 6 – *Requisição de Cadastro de Usuário via Insomnia (POST)*



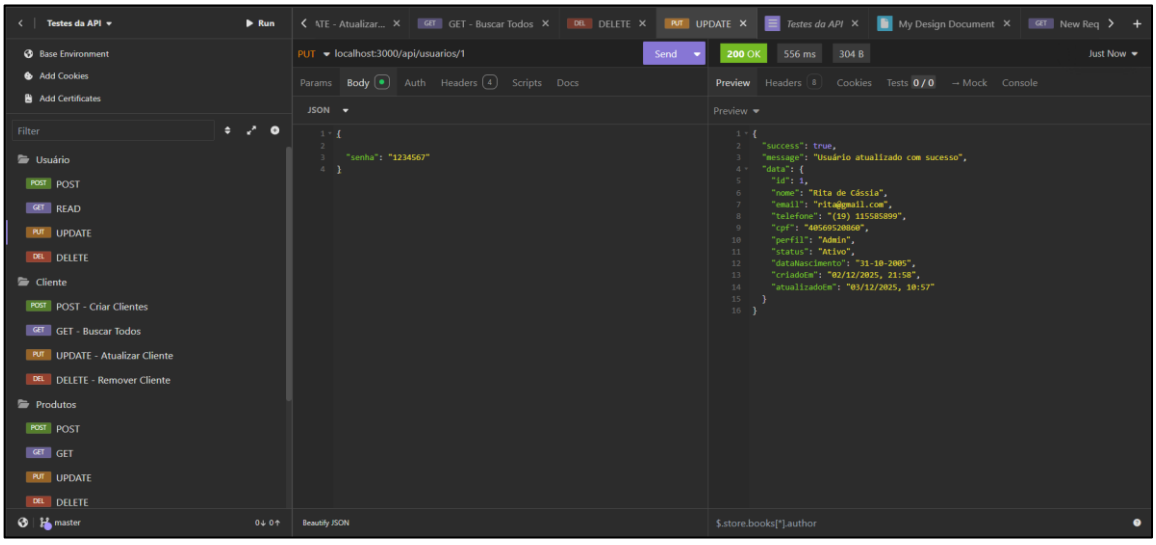
2. Leitura (GET /usuarios): Recuperação dos dados do usuário. Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 7 – *Requisição de Leitura de Usuário via Insomnia (GET)*



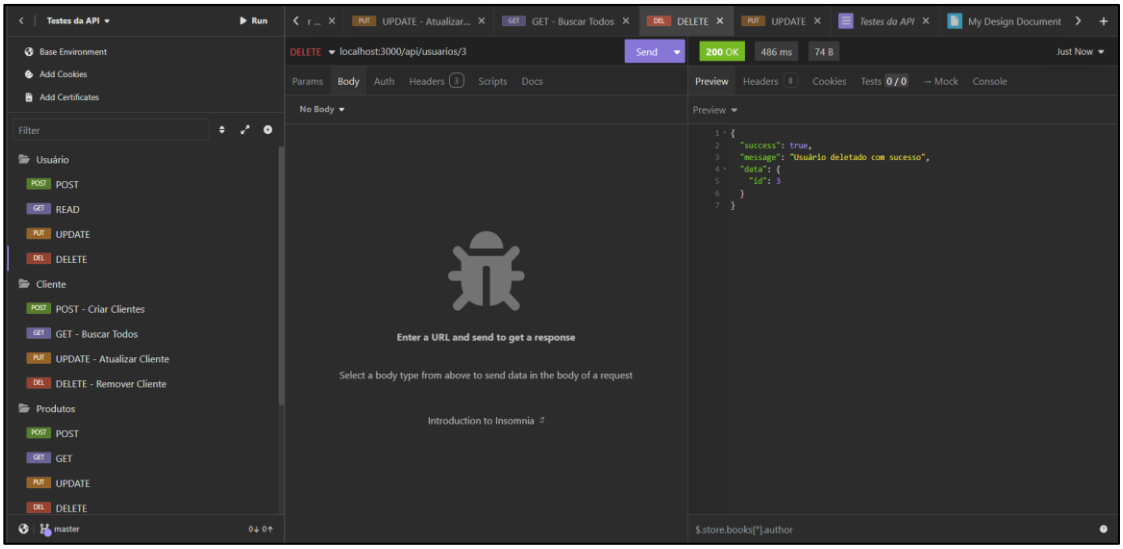
3. Atualização (PUT /usuarios/:id): Modificação de dados do usuário (ex: nome, perfil). Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 8 - *Requisição de Atualização de Usuário via Insomnia (PUT)*



4. Exclusão (DELETE /usuarios/:id): Remoção de um registro de usuário. Resultado Esperado: Código de status HTTP 200 OK (ou 204 No Content).

FIGURA 9 – *Requisição de Exclusão de Usuário via Insomnia (DELETE)*

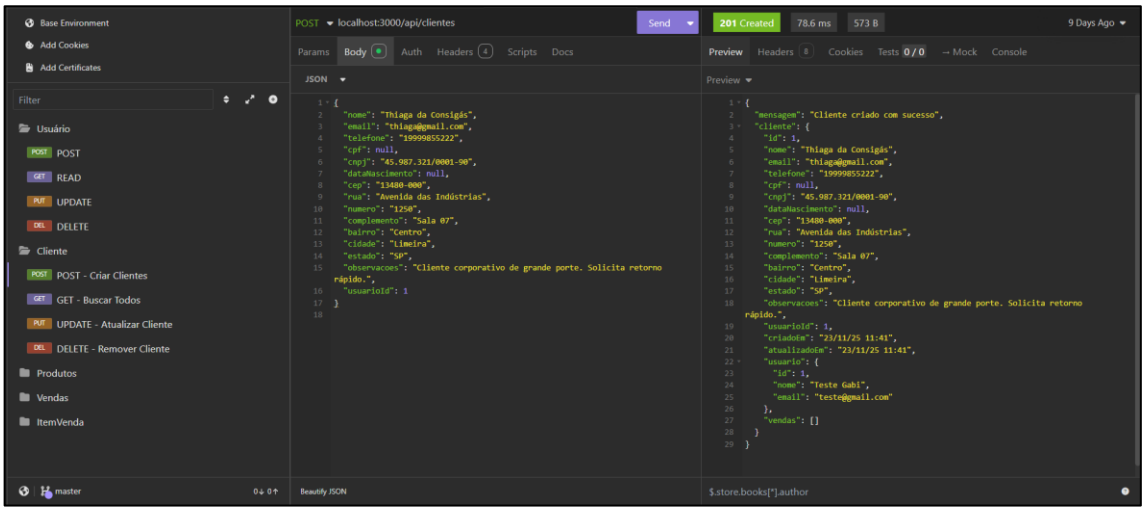


4.3.4 Testes da Entidade Cliente (CRUD)

Os testes de Cliente garantiram a funcionalidade de cadastro de novos clientes, fundamental para as operações de venda.

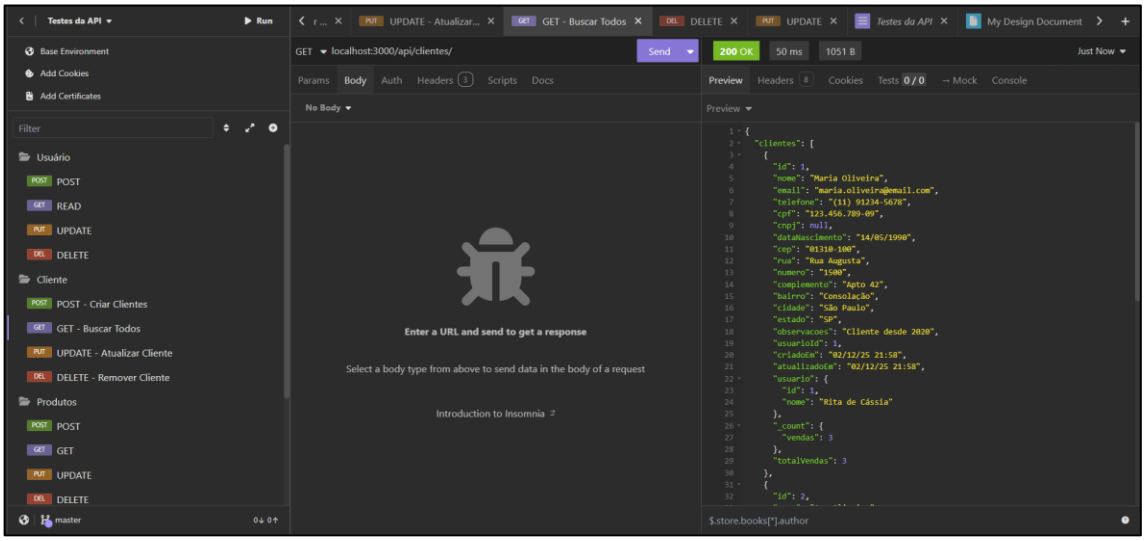
1. Criação (POST /clientes): Validação do cadastro de um novo cliente. Resultado Esperado: Código de status HTTP 201 Created.

FIGURA 10 – *Requisição de Cadastro de Cliente via Insomnia (POST)*



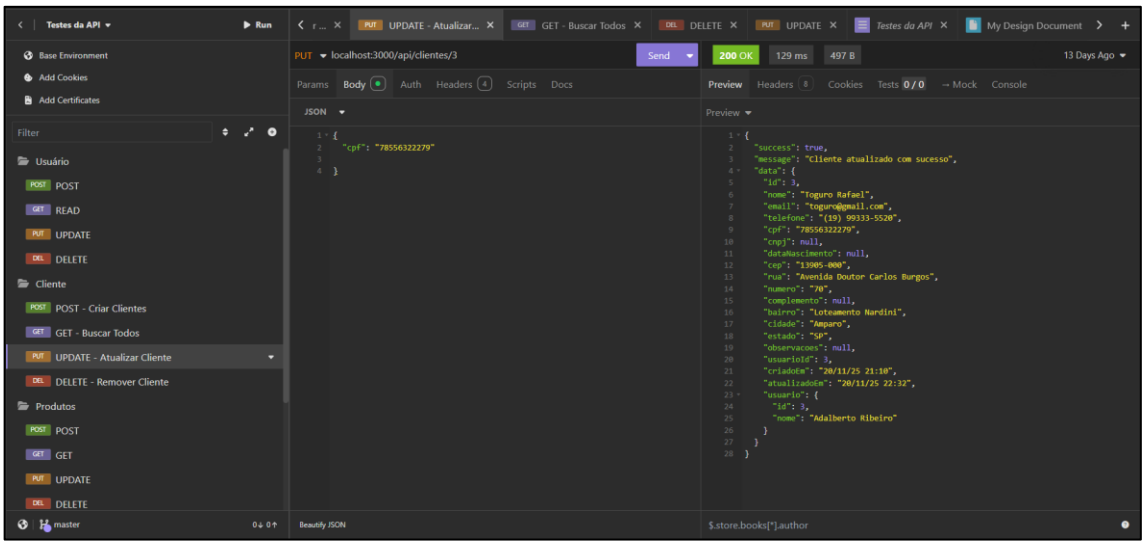
2. Leitura (GET /clientes): Recuperação dos dados cadastrais do cliente. Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 11 – *Requisição de Leitura de Cliente via Insomnia (GET por ID)*



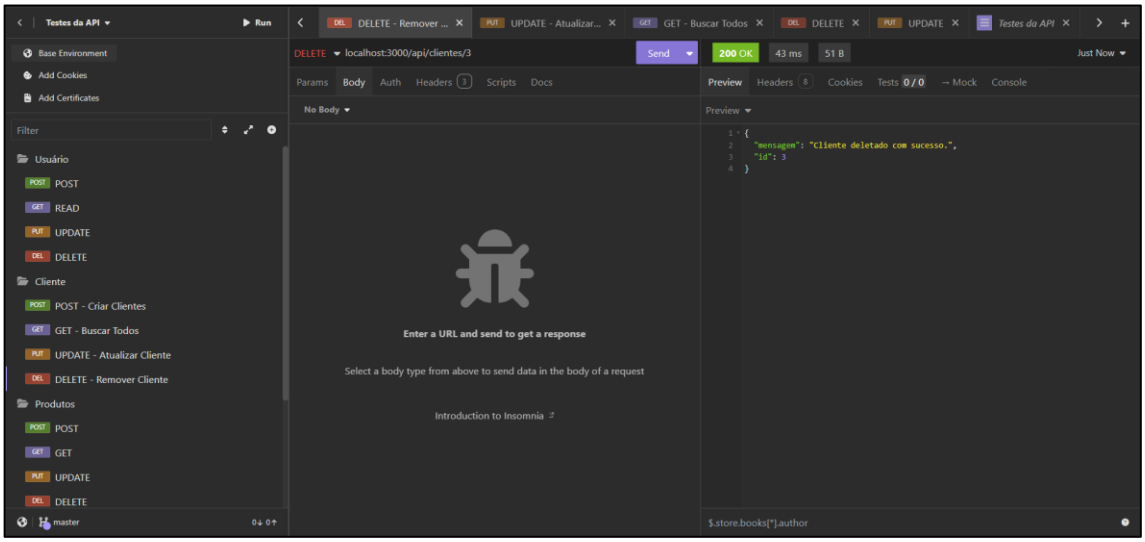
3. Atualização (PUT /clientes/:id): Modificação do telefone ou endereço do cliente. Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 12 – *Requisição de Atualização de Cliente via Insomnia (PUT)*



4. Exclusão (DELETE /clientes/:id): Remoção de um registro de cliente. Resultado Esperado: Código de status HTTP 200 OK (ou 204 No Content).

FIGURA 13 – *Requisição de Exclusão de Cliente via Insomnia (DELETE)*

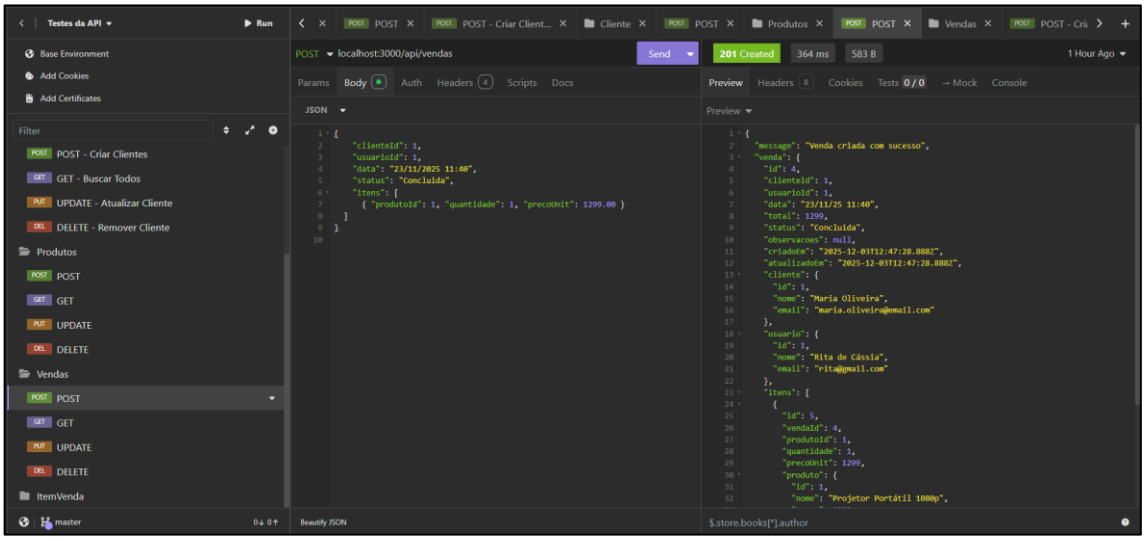


4.3.5 Testes da Entidade Venda (CRUD)

Os testes de Venda validaram o ciclo de vida da transação, desde o registro até a exclusão.

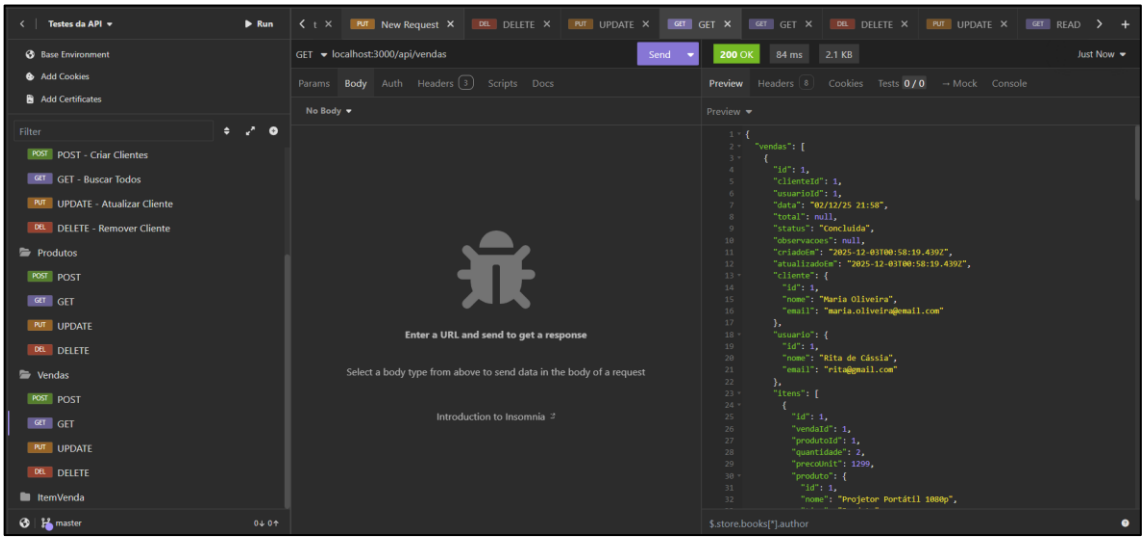
1. Criação (POST /vendas): Validação do registro de uma nova transação Resultado Esperado: Código de status HTTP 201 Created.

FIGURA 14 – *Requisição de Cadastro de Venda via Insomnia (POST)*



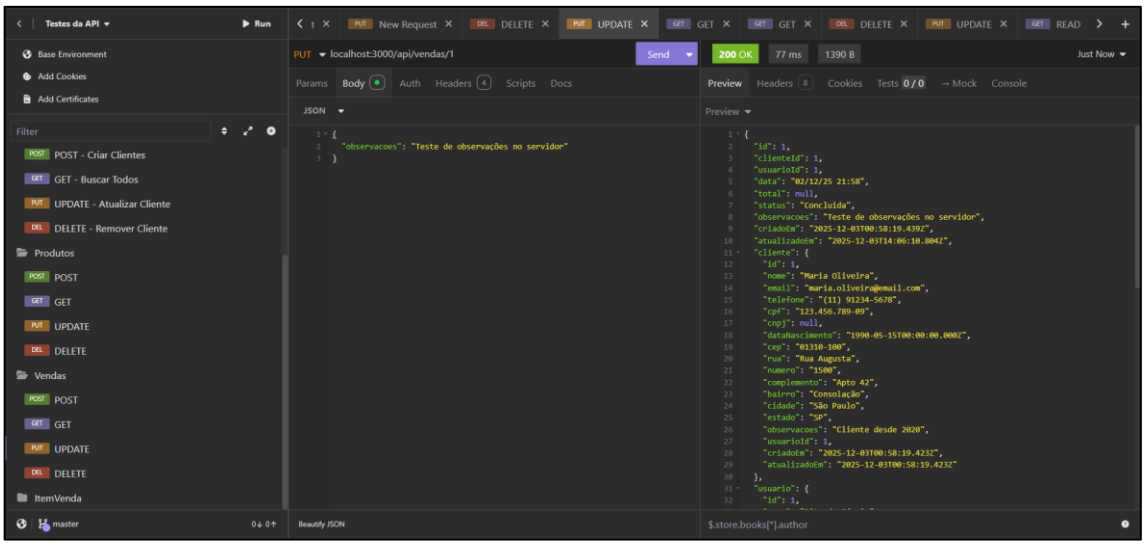
2. Leitura por (GET /vendas): Recuperação dos detalhes da venda (incluindo os itens de venda relacionados, se o *endpoint* estiver configurado para isso). Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 15 – *Requisição de Leitura de Venda via Insomnia (GET por ID)*



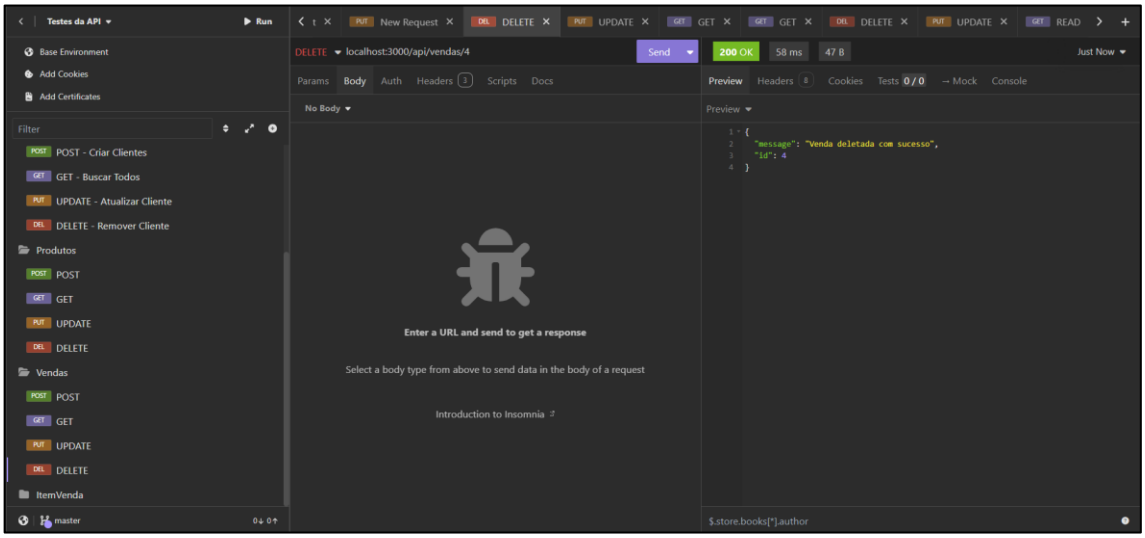
3. Atualização (PUT /vendas/:id): Atualização do status da venda (Ex: de Pendente para Concluída). Resultado Esperado: Código de status HTTP 200 OK.

FIGURA 16 – *Requisição de Atualização de Venda via Insomnia (PUT)*



4. Exclusão (DELETE /vendas/:id): Cancelamento e remoção da transação de venda. Resultado Esperado: Código de status HTTP 200 OK (ou 204 No Content).

FIGURA 17 – *Requisição de Exclusão de Venda via Insomnia (DELETE)*



4.3.6 Validação dos Dados no Banco (MySQL/MariaDB)

Após a realização das operações de escrita (POST, PUT e DELETE) pela API, foi realizada a validação direta no SGBD MariaDB para confirmar se os dados foram corretamente persistidos no banco. Essa conferência foi feita através do terminal do XAMPP, utilizando comandos SQL executados diretamente no banco NexoERPBanco.

Para verificar a consistência das informações, utilizou-se a seguinte consulta:

FIGURA 18 – *Comando para retornar todas as vendas.*

```
SELECT * FROM venda ORDER BY id DESC;
```

Esse comando retorna todas as vendas cadastradas, ordenadas da mais recente para a mais antiga, permitindo comparar os valores do banco com os retornos da API**.**

FIGURA 19 – *Validação no Banco de Dados (phpMyAdmin/MySQL)*

```
MariaDB [nexoerpbanco]> SELECT * FROM venda ORDER BY id DESC;
+----+-----------+-----------+---------------------+-------+-----------+-------------------------------+---------------------+---------------------
--------+
| id | clienteId | usuarioId | data                | total | status    | observacoes                   | criadoEm            | atualizadoEm
        |
+----+-----------+-----------+---------------------+-------+-----------+-------------------------------+---------------------+---------------------
--------+
|  3 |         1 |         1 | 2025-11-23 14:40:00.000 |  1299 | Concluida | NULL                          | 2025-12-03 12:47:25.729 | 2025-12-03 12:47
:25.729 |
|  2 |         2 |         1 | 2025-12-03 00:58:19.439 |  NULL | Concluida | NULL                          | 2025-12-03 00:58:19.439 | 2025-12-03 00:58
:19.439 |
|  1 |         1 |         1 | 2025-12-03 00:58:19.439 |  NULL | Concluida | Teste de observa├º├Áes no servidor  | 2025-12-03 00:58:19.439 | 2025-12-03 14:
06:10.804 |
+----+-----------+-----------+---------------------+-------+-----------+-------------------------------+---------------------+---------------------
--------+
```

4.4 Relatório Final de Testes e Conclusão

Os testes de API executados na ferramenta Insomnia abrangeram um total de 16 casos de teste fundamentais, cobrindo todas as quatro operações CRUD para as quatro entidades principais do sistema (Usuário, Cliente, Produto e Venda). O objetivo primário de validar a integridade e a funcionalidade da API RESTful foi atingido com sucesso.

Tabelas 1- Resultado das operações CRUD

Entidade	Total de Casos Testados	Casos de Sucesso	Casos de Falha	Taxa de Sucesso
Produto	4	4	0	100%
Usuário	4	4	0	100%
Cliente	4	4	0	100%
Venda	4	4	0	100%
Total	**16**	**16**	**0**	**100%**

Todas as requisições, incluindo *POST* (criação), *GET* (leitura), *PUT* (atualização) e *DELETE* (exclusão), retornaram os códigos de status HTTP esperados (201 ou 200 OK) e o corpo de resposta JSON validado conforme o esperado. Isso comprova que a Camada de Aplicação (Backend Node.js/Express) está corretamente integrada com a Camada de Dados (MySQL/Prisma) e que a lógica de negócio básica (CRUD) está implementada de forma robusta e funcional.

5 INTRODUÇÃO AO DESENVOLVIMENTO FRONTEND PARA WEB

O desenvolvimento *frontend* refere-se à parte visual e interativa de um sistema ou aplicação web — ou seja, tudo aquilo que o usuário final vê e com o que interage diretamente. Também conhecido como camada de apresentação, o *frontend* é responsável por transmitir as informações do sistema de forma clara, acessível e intuitiva, estabelecendo a ponte entre o usuário e as funcionalidades do *software*.

O papel do *frontend* vai além da simples exibição de dados. Ele é fundamental para garantir uma Experiência de Usuário (UX) eficiente e agradável, contribuindo para a usabilidade, acessibilidade e desempenho do sistema. Uma interface mal estruturada, visualmente confusa ou com elementos pouco responsivos pode comprometer a compreensão do usuário e até mesmo dificultar o uso da aplicação, impactando diretamente nos resultados e na satisfação do público-alvo.

Nesse contexto, o desenvolvimento *frontend* utiliza três tecnologias essenciais:

- HTML (*HyperText Markup Language*): é a linguagem base de estruturação do conteúdo de uma página *web*. Ele define os elementos visuais e semânticos da interface, como títulos, parágrafos, botões, tabelas, formulários, entre outros.
- CSS (*Cascading Style Sheets*): é a linguagem de estilização responsável pela aparência visual da aplicação. Por meio do CSS, define-se o *layout*, cores, espaçamento, fontes, animações e a responsividade da interface.
- JavaScript: é a linguagem de programação utilizada no *frontend* para tornar as páginas interativas e dinâmicas. Com JavaScript, é possível implementar comportamentos como validações de formulários, requisições assíncronas (AJAX), animações, manipulação do DOM e muito mais.

A combinação dessas três tecnologias constitui a base para o desenvolvimento moderno de interfaces *web*. Além disso, com o avanço da tecnologia, surgiram diversos *frameworks* e bibliotecas (como React, Vue.js e Angular) que otimizam o processo de desenvolvimento, promovem reutilização de componentes e melhoram a escalabilidade dos projetos.

Portanto, o domínio do desenvolvimento *frontend* é essencial para qualquer profissional da área de sistemas, especialmente quando se busca desenvolver aplicações eficientes, intuitivas e centradas no usuário.

6 ESTRUTURA E FUNCIONAMENTO DO HTML

O HTML (*HyperText Markup Language*) é a linguagem padrão utilizada para estruturar o conteúdo de páginas *web*. Sua função principal é organizar os elementos que serão exibidos no navegador, como textos, imagens, links, formulários e demais componentes visuais. Ao contrário das linguagens de programação, o HTML é uma linguagem de marcação, composta por *tags* que indicam a função semântica de cada elemento dentro da estrutura da página.

Um documento HTML segue uma hierarquia bem definida. A estrutura básica inclui:

- <!DOCTYPE html>: declaração do tipo de documento, informando ao navegador que se trata de um HTML5.
- <html>: elemento raiz do documento.
- <head>: seção destinada a metadados, como título da página (<title>), codificação de caracteres (<meta charset="UTF-8">), *links* para arquivos CSS, entre outros.
- <body>: corpo do documento, onde são inseridos os elementos visíveis e interativos da página.

Entre os principais elementos estruturais utilizados na organização de uma página moderna, destacam-se:

- <header>: cabeçalho da página ou de uma seção.
- <nav>: agrupamento de *links* de navegação.

- <main>: conteúdo principal da página.
- <section> e <article>: blocos de conteúdo tematicamente agrupados.
- <footer>: rodapé da página ou seção.

O uso correto dessas *tags* contribui para a semântica do HTML, promovendo acessibilidade (facilitando o uso por leitores de tela e dispositivos assistivos), melhor indexação em motores de busca (SEO) e maior organização e manutenibilidade do código.

7 ORGANIZAÇÃO DE PROJETOS FRONTEND

A organização de projetos *frontend* é essencial para manter a escalabilidade, clareza e eficiência do desenvolvimento. Uma boa estrutura facilita a colaboração entre desenvolvedores e reduz o tempo de manutenção e evolução do sistema.

7.1 Estruturação de Diretórios e Arquivos

Em projetos de médio e grande porte, recomenda-se dividir os arquivos em pastas com responsabilidades específicas, como:

- /assets: imagens, fontes e arquivos estáticos.
- /css ou /styles: arquivos de estilos (CSS).
- /js: *scripts* JavaScript.
- /components ou /partials: partes reutilizáveis da interface.
- /views ou /pages: páginas completas da aplicação.

7.2 Separação de Responsabilidades

Cada tecnologia possui um papel distinto no *frontend*, e o princípio da Separação de Responsabilidades (*Separation of Concerns*) deve ser rigorosamente seguido:

- HTML: define a estrutura do conteúdo.
- CSS: aplica a estilização visual aos elementos HTML.
- JavaScript: adiciona lógica, interatividade e dinamismo à interface.

Algumas recomendações importantes incluem:

- Utilização de nomes de arquivos e pastas padronizados e descritivos.
- Modularização do código, facilitando o reaproveitamento de componentes.
- Manter os arquivos organizados por funcionalidade e não apenas por tipo.

8 FUNCIONALIDADES E COMPOSIÇÃO DAS TELAS DO SISTEMA

O *frontend* de um sistema web é composto por várias telas que atendem a diferentes funcionalidades, cada uma com seus elementos visuais e comportamentos específicos. As principais telas incluem:

- Tela de Login: permite a autenticação do usuário. Deve conter campos para usuário e senha, validações básicas e redirecionamento seguro após o *login* bem-sucedido.
- Dashboard: apresenta uma visão geral do sistema com informações resumidas, gráficos e indicadores-chave (KPIs), como número de vendas, clientes ativos, ou status de estoque.
- Telas de CRUD: contemplam operações de Cadastro, Leitura, Atualização e Exclusão de dados. Podem ser aplicadas a entidades como usuários, clientes, produtos, vendas e estoque. Geralmente incluem formulários, tabelas e botões de ação.
- Tela de Relatórios: permite ao usuário aplicar filtros personalizados (por data, categoria, status, etc.) e gerar relatórios de forma dinâmica. Os dados podem ser apresentados em formato tabular, gráfico ou exportável.
- Design Responsivo: todas as telas devem ser construídas com atenção à responsividade, garantindo o bom funcionamento em diferentes tamanhos de tela (*desktops*, *tablets* e *smartphones*), aplicando princípios como *mobile-first* e *media queries* no CSS.

8.1 Prototipação e Design da Interface (Ui/Ux)

A interface do ERP foi projetada com foco na simplicidade e usabilidade (*User Experience - UX*), requisitos essenciais para PMEs.

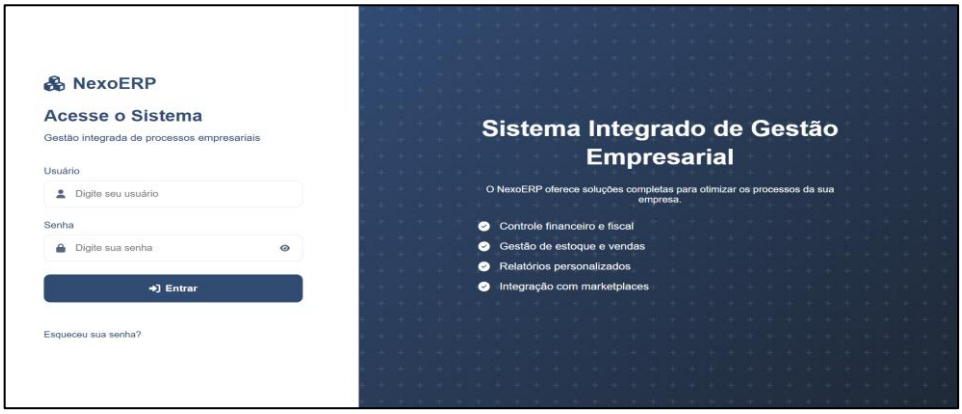
9 FUNCIONALIDADES E COMPOSIÇÃO DAS TELAS DO SISTEMA

9.1 TELAS PRINCIPAIS DO SISTEMA

1. Tela de Login e Autenticação

Esta tela é o ponto de entrada do sistema. Ela exige o fornecimento de credenciais válidas (e-mail e senha) e é responsável por iniciar o processo de autenticação do usuário, obtendo o *token* de acesso para as requisições subsequentes.

FIGURA 20 – *Tela de Login do ERP Simplificado*



2. **Dashboard (Painel de Controle)**

Após o *login*, o usuário é direcionado ao *Dashboard*, que serve como uma visão panorâmica das operações da empresa. Exibe métricas de desempenho em tempo real, como volume de vendas, status de estoque e resumo financeiro.

FIGURA 21 – *Dashboard do ERP Simplificado*

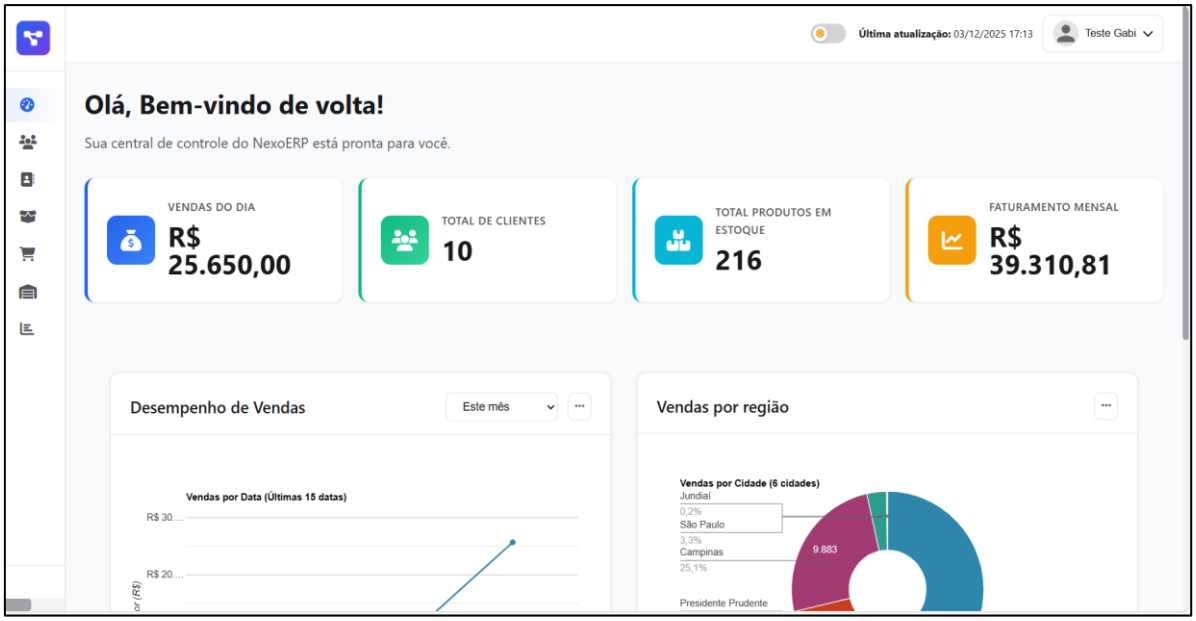
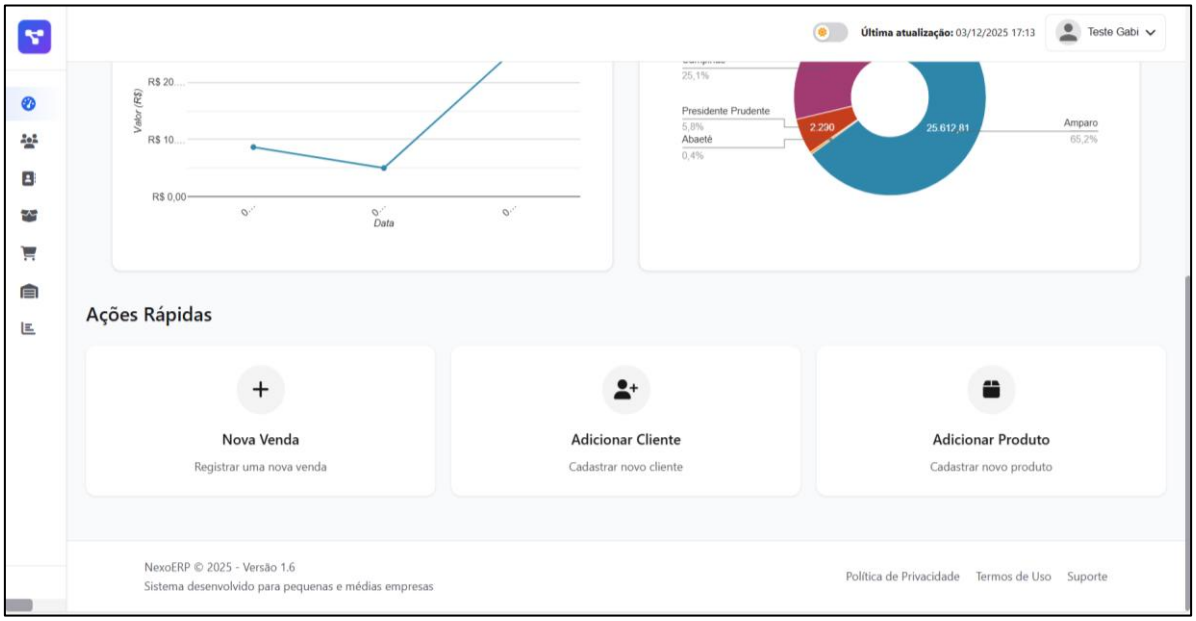


FIGURA 22 – *Dashboard do ERP Simplificado 2*



3. **Tela de Gerenciamento de Usuários (CRUD)**

Esta tela é crucial para a segurança do sistema. Permite que administradores gerenciem os registros de acesso, incluindo a criação, edição e exclusão de contas, bem como a definição de perfis (Admin ou Operador).

FIGURA 23 – *Tela de Gerenciamento de Usuários - Listagem e Ações CRUD*

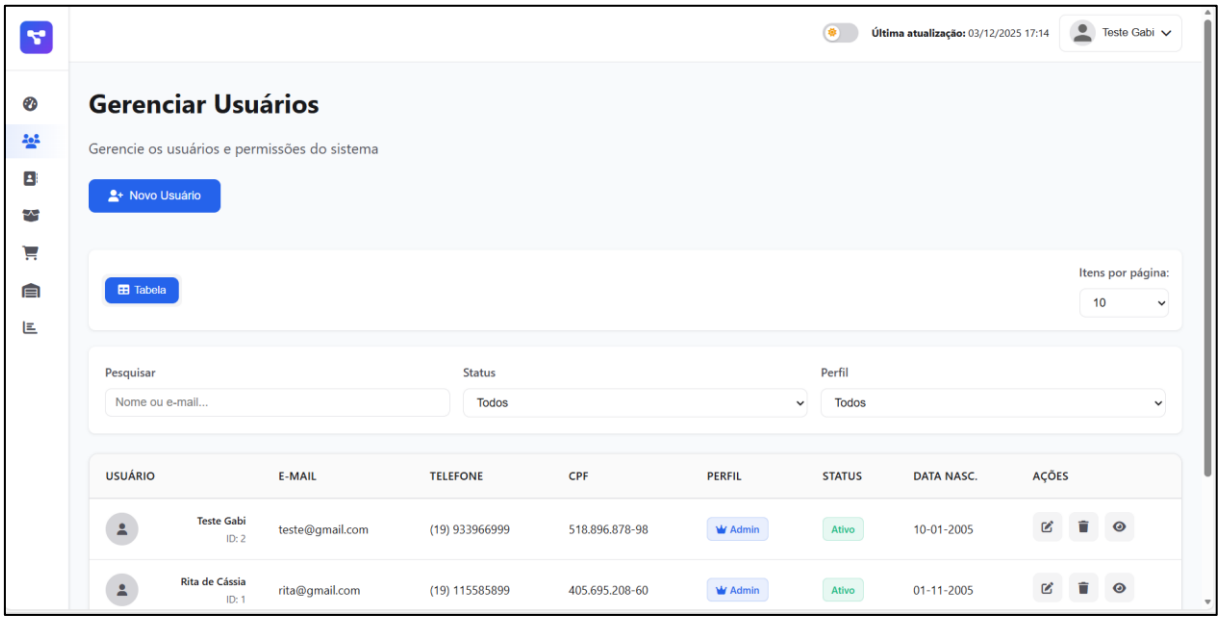
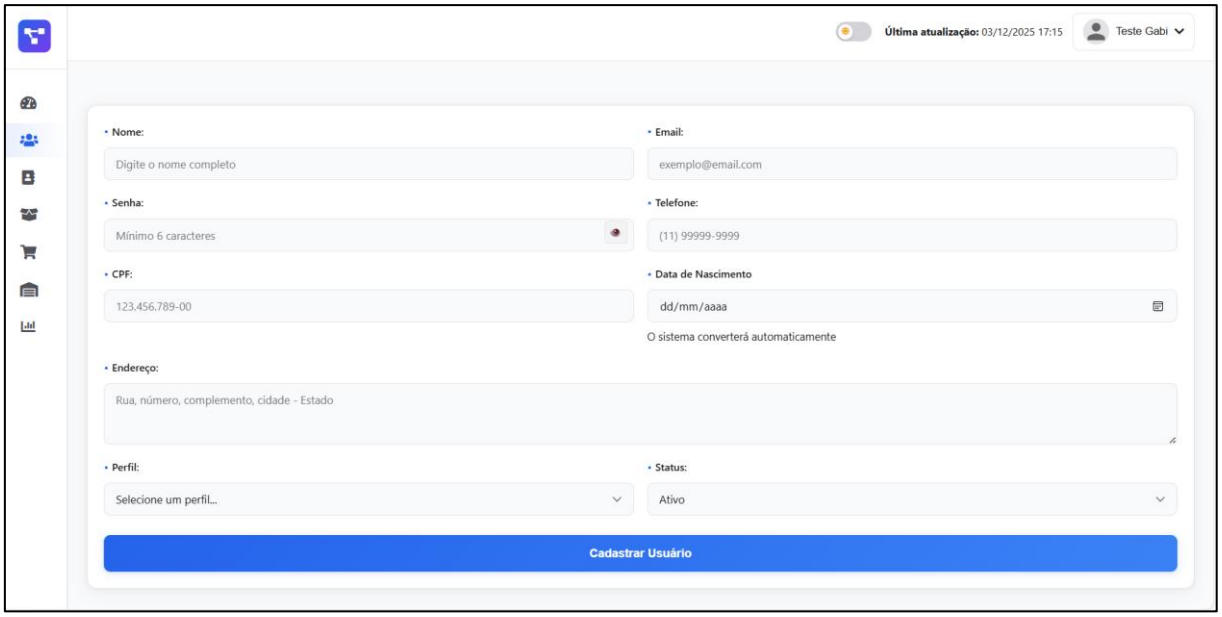


FIGURA 24 – *Tela de Gerenciamento de Usuários (CREATE)*



4. **Tela de Gerenciamento de Clientes (CRUD)**

Dedica-se à gestão da base de clientes, permitindo listar, adicionar, editar e excluir informações cadastrais (nome, CPF/CNPJ, contato). Um cadastro atualizado é vital para o módulo de vendas.

FIGURA 25 – *Tela de Gerenciamento de Clientes - Listagem e Ações CRUD*

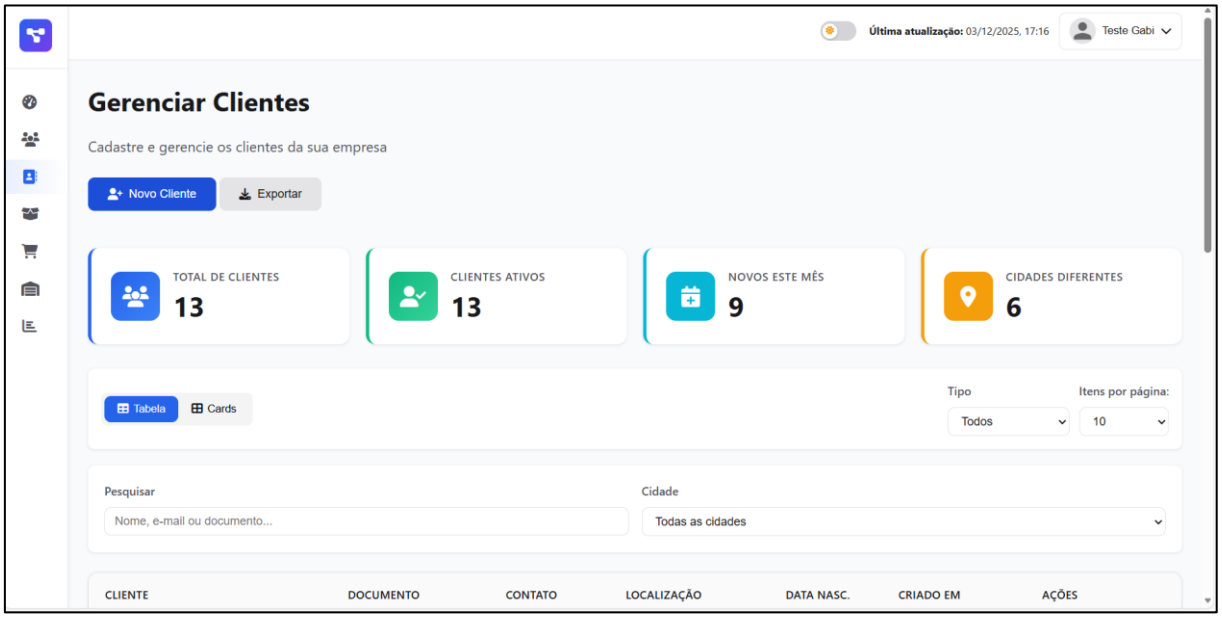


FIGURA 26 – *Tela de Gerenciamento de Clientes (Visualização dos clientes)*

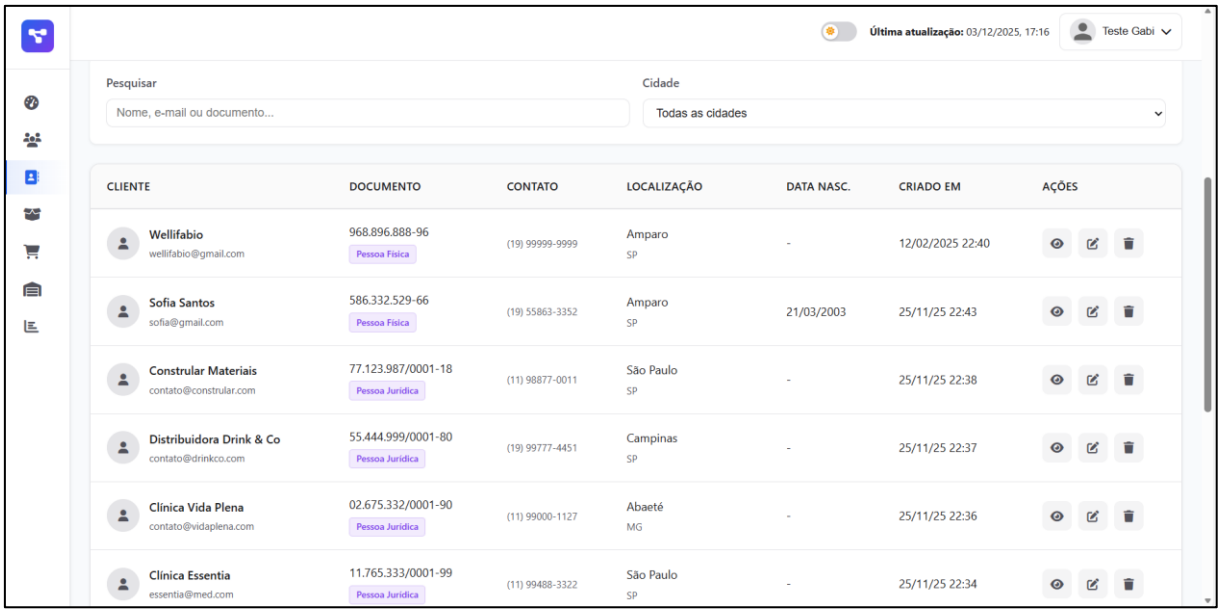
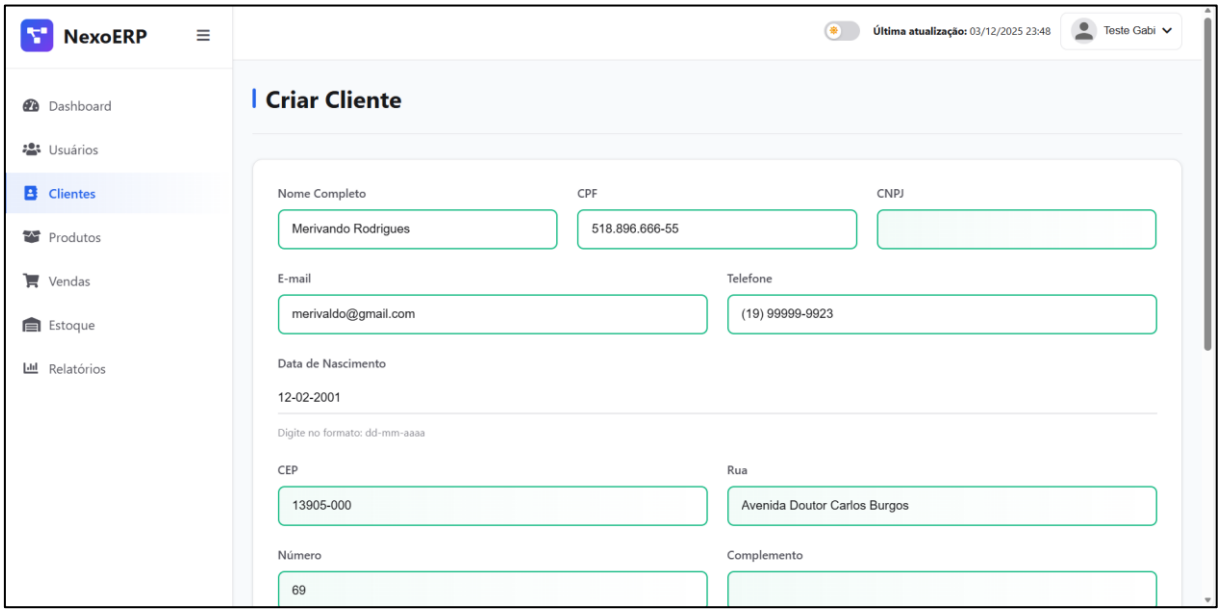


FIGURA 27 – *Tela de Gerenciamento de Clientes (CREATE)*



5. Tela de Gerenciamento de Produtos (CRUD)

Esta tela é dedicada à gestão do inventário. Permite listar todos os produtos e serviços cadastrados, adicionar novos, editar informações (como preço e estoque) e inativar registros. A interface tabular facilita a visualização e busca de itens.

FIGURA 28 – *Tela de Gerenciamento de Produtos - Listagem e Ações CRUD*

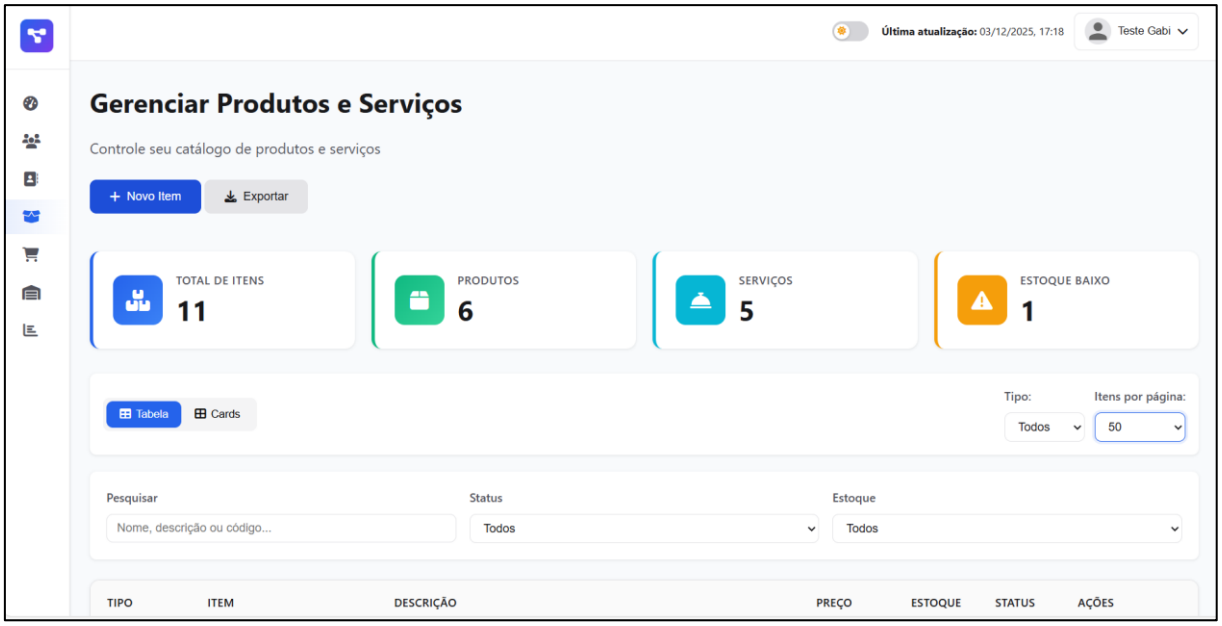


FIGURA 29 – *Tela de Gerenciamento de Produtos (Visualização dos produtos)*

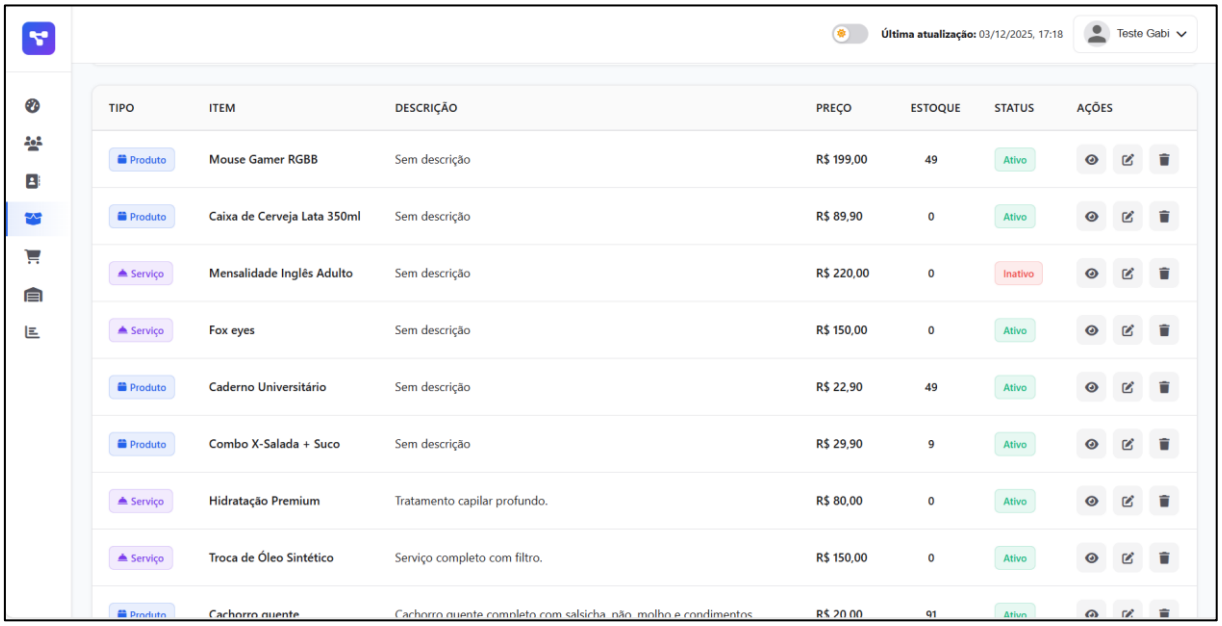
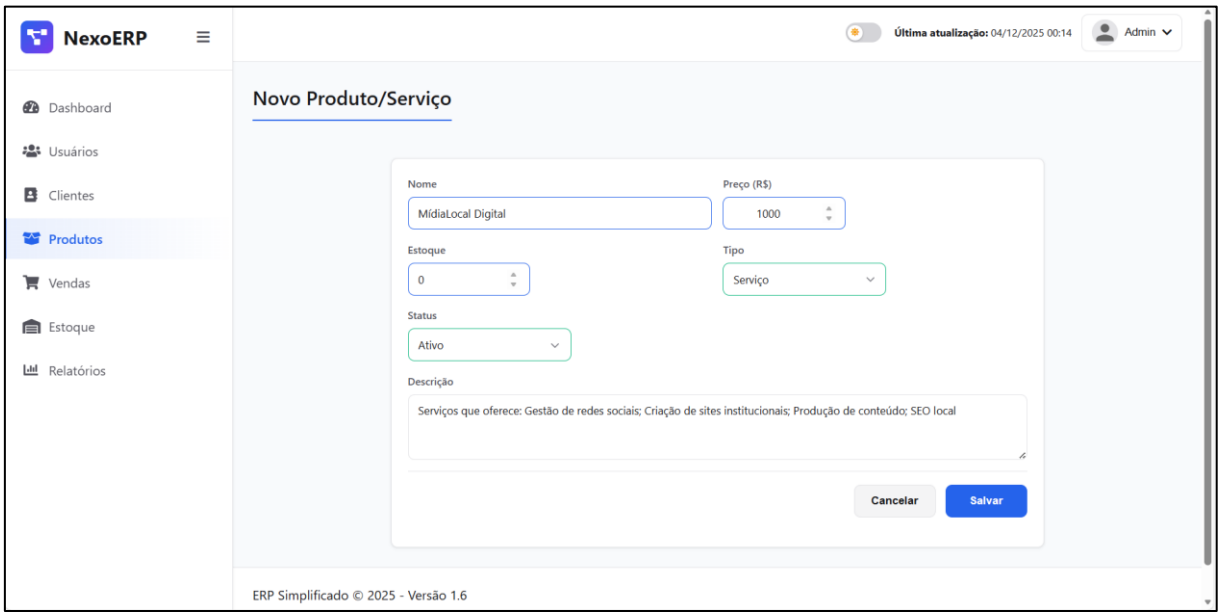


FIGURA 30 – *Tela de Gerenciamento de Produtos (CREATE)*



6. **Tela de Lançamento de Vendas**

Este é o módulo central do sistema, projetado para ser intuitivo e rápido. Ele permite selecionar o cliente, adicionar múltiplos produtos ao carrinho, calcular o total automaticamente e registrar a transação no *backend*.

FIGURA 31 – *Tela de Gerenciamento de Vendas - Listagem e Ações CRUD*

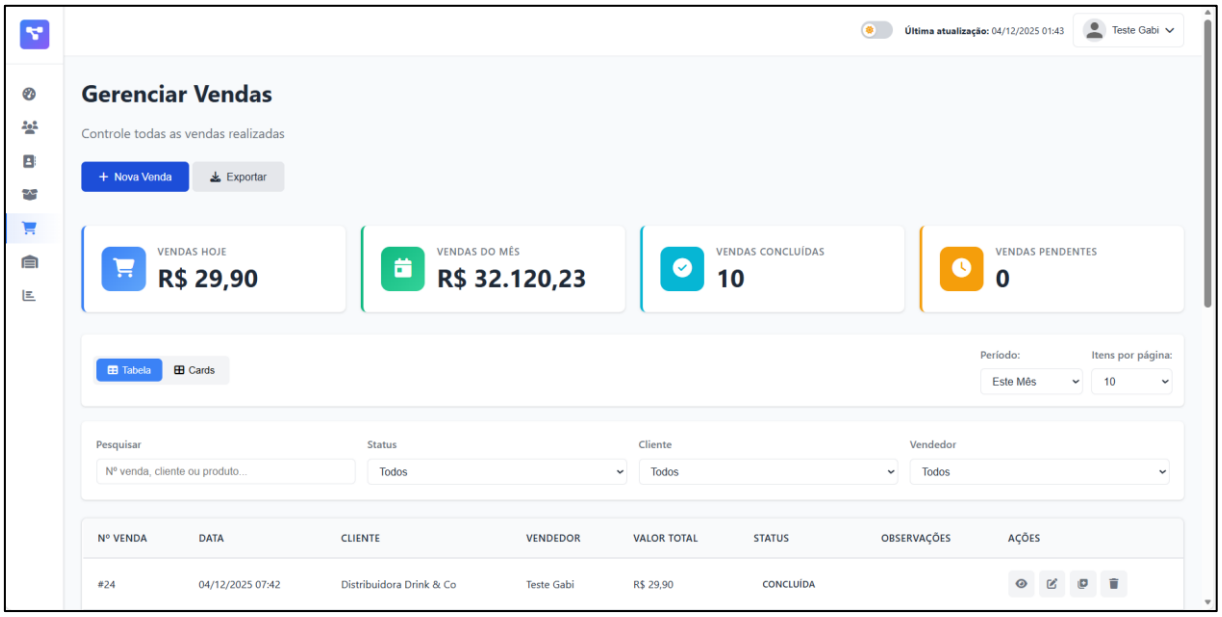
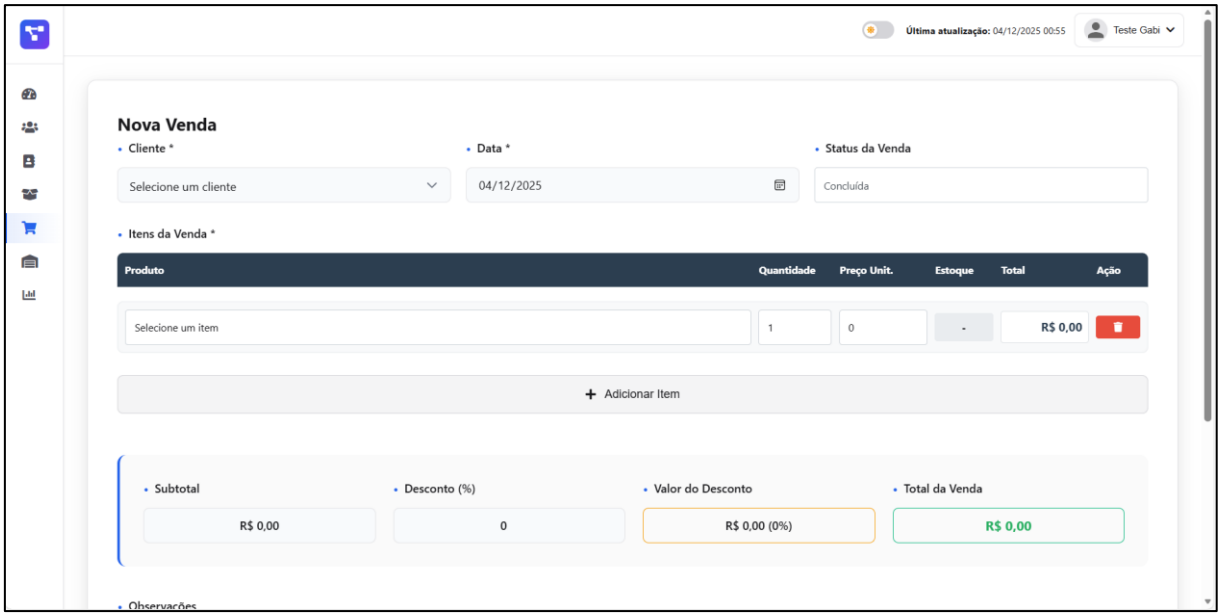


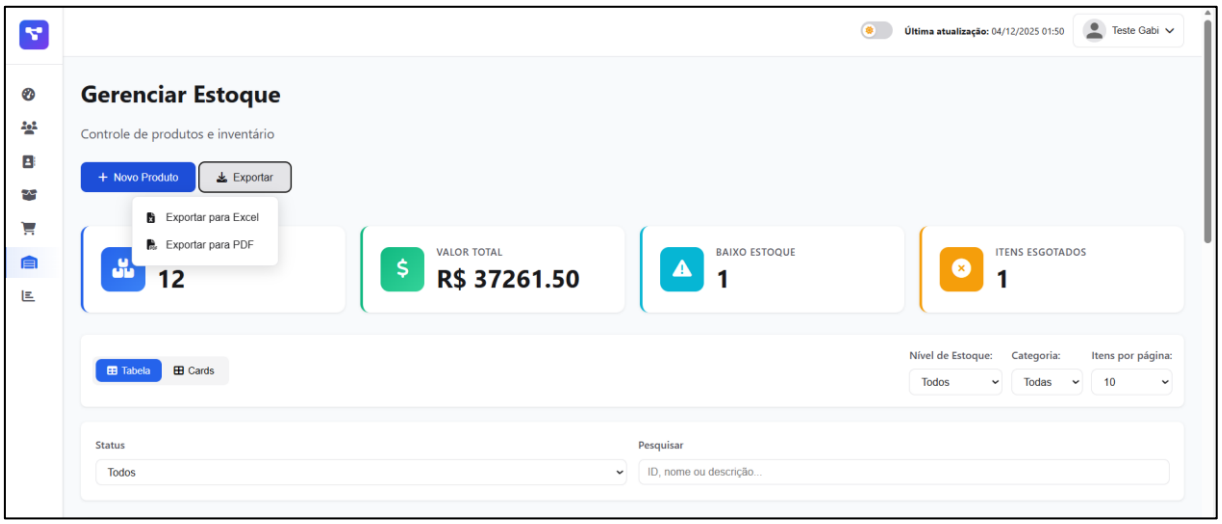
FIGURA 32 – *Tela de Gerenciamento de Vendas (CREATE)*



7. Tela de Gerenciamento de Estoque

Este módulo nos traz as mesmas funcionalidades da tela de produtos e serviços, mas dessa vez nos proporcionando a possibilidade de exportar os dados, sendo possível a exportação em Excel e PDF.

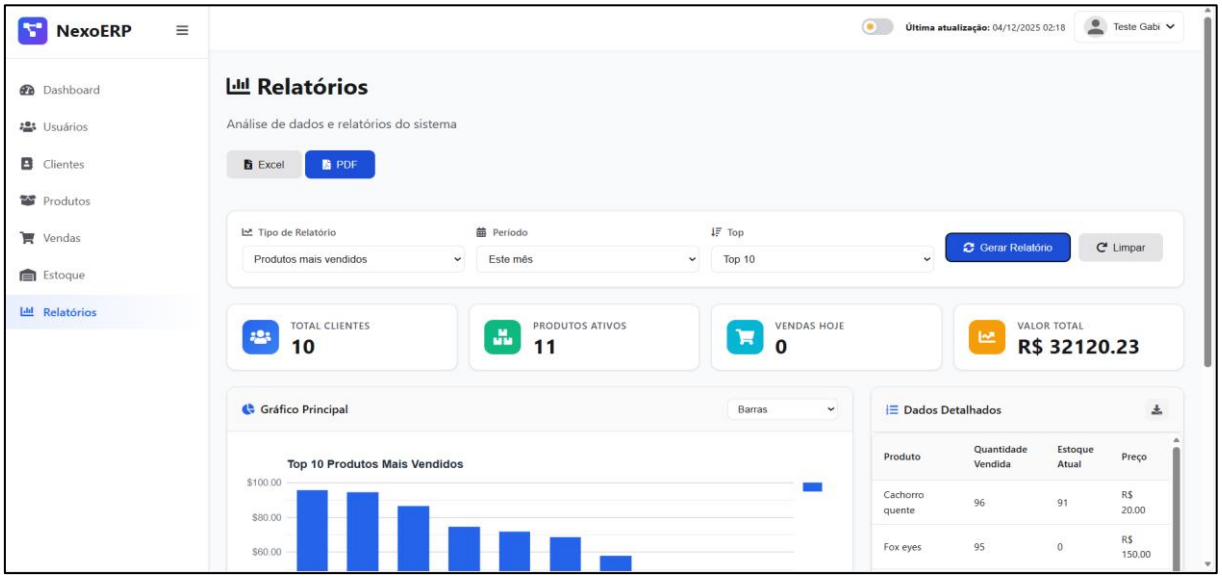
FIGURA 33 – *Tela de Gerenciamento de Estoque (CREATE)*



8. Tela de Relatórios

Este módulo nos trás uma visão panorâmica de todo o sistema, o relatório das principais operações feitas com a possibilidade escolher o seu tipo (Ex.: produtos mais vendidos, cliente que mais comprou), o período e o Top que implica na quantidade de itens que você espera ser gerado. Além de relatórios gerados para produtos, vendas e clientes.

FIGURA 34 – *Tela de Relatórios*



10 INTEGRAÇÃO DA CAMADA DE APRESENTAÇÃO COM A API RESTful

Este capítulo detalha o fluxo de comunicação e o mecanismo de troca de dados entre a Camada de Apresentação (*Frontend*) e a Camada de Aplicação (*Backend*/API RESTful), sem a necessidade de expor o código fonte detalhado.

10.1 FLUXO DE COMUNICAÇÃO (AJAX e Fetch API)

A comunicação entre o *Frontend* (HTML/JavaScript) e o *Backend* (Node.js API) é realizada de forma assíncrona, utilizando o protocolo HTTP e o recurso nativo *Fetch* API do JavaScript. Essa abordagem é essencial para a criação de uma *Single-Page Application* (SPA) ou uma aplicação com carregamento de conteúdo dinâmico, que evita o recarregamento total da página.

10.1.1 Requisição de Leitura de Dados (GET)

O processo de leitura de dados, como a carga da lista de produtos na tela de CRUD ou o carregamento de métricas no *Dashboard* e Relatórios, segue o fluxo:

1. Gatilho: O JavaScript é acionado (ex: ao carregar a página ou clicar em um botão).
2. Requisição: Uma requisição GET é enviada para o *endpoint* específico da API (ex: /produtos ou /relatorios/vendas), incluindo o Token de Acesso (Bearer Token) no cabeçalho HTTP para validação.
3. Resposta: A API retorna os dados solicitados no formato JSON e um código de status HTTP 200 OK.
4. Processamento: O JavaScript recebe a *string* JSON, a converte em um objeto JavaScript (JSON *parsing*).
5. Renderização: O *script* JavaScript manipula o DOM (*Document Object Model*) da página, inserindo os dados recebidos (ex: criando linhas na tabela de produtos, ou renderizando gráficos nos relatórios).

10.1.2 Envio e Atualização de Dados (POST, PUT, DELETE)

Para operações de escrita e modificação (CRUD), o fluxo de comunicação requer o envio de dados no corpo da requisição:

1. Captura de Dados: O JavaScript coleta os dados do formulário HTML (ex: nome, preço e estoque de um novo produto).
2. Serialização: Os dados são formatados em um objeto JavaScript e depois convertidos em uma *string* JSON (*JSON.stringify*).
3. Requisição: A requisição (POST para criação, PUT para atualização, DELETE para exclusão) é enviada para a API, com o corpo JSON anexado e o cabeçalho Content-Type: application/json.
4. Confirmação: A API processa a lógica de negócio, interage com o banco de dados e retorna um código de status de sucesso (ex: 201 *Created* ou 200 OK).

5. *Feedback*: O *Frontend* exibe uma mensagem de sucesso para o usuário e atualiza a interface, geralmente recarregando a lista de dados afetada.

10.2 LÓGICA DE NEGÓCIO IMPLEMENTADA NO FRONTEND

Embora a maior parte da lógica de negócio resida no *Backend* (garantindo segurança e integridade), o *Frontend* é responsável por lógicas essenciais que otimizam a experiência do usuário:

- Validação de Formulários: Implementação de validações em tempo real (ex: campos obrigatórios, formato de e-mail/CPF/CNPJ) para reduzir requisições inválidas ao servidor.
- Cálculo de Vendas: No módulo de vendas, o JavaScript executa a lógica de cálculo do subtotal e do total da transação à medida que o usuário adiciona itens ou altera quantidades. Isso fornece *feedback* imediato e minimiza o processamento do *Backend* até o momento da finalização da compra.
- Controle de Acesso Visual: Após o login, o perfil do usuário é identificado. O JavaScript é responsável por ocultar ou exibir elementos da interface (ex: botões de "Excluir" ou *links* para telas administrativas) que não são permitidos para o perfil logado (ex: um Vendedor não pode acessar o CRUD de Usuários Administradores).

11 CONCLUSÃO

O presente trabalho de conclusão de curso teve como objetivo principal o desenvolvimento e a implementação de um Sistema ERP Simplificado (NexoERP) para atender às necessidades de gestão básicas de Pequenas e Médias Empresas (PMEs), com foco na eficiência, baixo custo de infraestrutura e usabilidade.

Através de uma metodologia ágil (*Scrum*) e uma arquitetura robusta de três camadas, o projeto demonstrou a viabilidade de construir um sistema completo utilizando o Stack Node.js (*Backend*), MySQL (Banco de Dados) e HTML/CSS/JavaScript puro (*Frontend*). A escolha por tecnologias *open-source* e leves foi crucial para alcançar o requisito de baixo custo operacional.

Os objetivos específicos foram integralmente cumpridos:

1. Modelagem de Dados e API RESTful: Foi estabelecida uma estrutura de banco de dados relacional (MySQL) eficiente, e a API RESTful foi implementada com sucesso (Capítulo 4), conforme demonstrado pelos testes Insomnia, que apresentaram 100% de sucesso nas operações CRUD (Criação, Leitura, Atualização e Exclusão) para as entidades cruciais: Usuário, Cliente, Produto e Venda.
2. Desenvolvimento Frontend e UX: A camada de apresentação (Capítulos 9 e 10) resultou em uma interface intuitiva e responsiva, com fluxo de trabalho otimizado para as PMEs (Gerenciamento de Usuários, Clientes, Produtos e o Módulo de Vendas). A integração assíncrona com a API (Fetch API) garantiu um carregamento dinâmico e rápido das informações.

Em suma, o NexoERP se estabelece como um protótipo funcional e validado, que resolve o desafio de fornecer uma ferramenta de gestão acessível e eficiente para o público-alvo, provando que soluções simples e bem arquitetadas podem oferecer grande valor de negócio.

Para expandir o valor e a funcionalidade, sugere-se os seguintes trabalhos e módulos futuros:

1. Módulo Financeiro Avançado: Implementação de contas a pagar, contas a receber e fluxo de caixa detalhado, permitindo uma visão financeira mais completa para o gestor.
2. Módulo de Inventário Otimizado: Introdução de funcionalidades de controle de lote, validade e entradas de estoque (compras), além da baixa automática por código de barras.
3. Segurança e Autenticação Reforçada: Migração do sistema de autenticação para um padrão mais robusto, como OAuth 2.0, e a inclusão de controle de acesso baseado em funções (Role-Based Access Control - RBAC) mais granular, não apenas por perfis.
4. Notificações em Tempo Real: Implementação de *WebSockets* para notificar usuários sobre eventos críticos, como estoque mínimo ou novas vendas realizadas, sem a necessidade de recarregamento da página.

REFERÊNCIAS

As referências a seguir listam as principais obras acadêmicas, guias de desenvolvimento e documentações técnicas que nortearam a metodologia, arquitetura e implementação deste projeto de TCC.

NORMAS, PADRÕES E METODOLOGIAS

ABNT NBR 14724:2011. **Informação e documentação – Numeração progressiva das seções de um documento**. Rio de Janeiro: ABNT, 2011.

ABNT NBR 6023:2018. **Informação e documentação – Referências – Elaboração**. Rio de Janeiro: ABNT, 2018.

SCHWABER, Ken; BEEDLE, Mike. **Scrum: guia prático de projetos ágeis**. Tradução: Lúcia A. M. Leão. Rio de Janeiro: Elsevier, 2004.

LIVROS E FONTES ACADÊMICAS

PRESSMAN, Roger S. **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: AMGH, 2016. (Utilizado para conceitos de ciclo de vida e testes de software).

TANENBAUM, Andrew S.; WETHERALL, David J. **Redes de Computadores**. 5. ed. Pearson Education, 2011. (Consultado para princípios de Arquitetura Cliente-Servidor e Protocolo HTTP).

DOCUMENTAÇÃO TÉCNICA E FERRAMENTAS ONLINE

MDN Web Docs. **HTML: HyperText Markup Language**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em: 04 dez. 2025.

MDN Web Docs. **CSS: Cascading Style Sheets**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS. Acesso em: 04 dez. 2025.

MDN Web Docs. **JavaScript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 04 dez. 2025.

NODE.JS. **Documentação Oficial**. Disponível em: https://nodejs.org/. Acesso em: 04 dez. 2025.

PRISMA. **Documentação Oficial (ORM)**. Disponível em: https://www.prisma.io/docs/. Acesso em: 04 dez. 2025.

MySQL. **MySQL 8.0 Reference Manual**. Disponível em: https://dev.mysql.com/doc/refman/8.0/en/. Acesso em: 04 dez. 2025.

W3C. **Web Content Accessibility Guidelines (WCAG) 2.1**. Disponível em: https://www.w3.org/TR/WCAG21/. Acesso em: 04 dez. 2025. (Referência para princípios de usabilidade e acessibilidade).