PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

FOODMANAGER: APLICAÇÃO PARA GERENCIAMENTO DE PEDIDOS DE RESTAURANTES

MATEUS DEL CAMPO QUEZADA

GOIÂNIA
2025

MATEUS DEL CAMPO QUEZADA

FOODMANAGER: APLICAÇÃO PARA GERENCIAMENTO DE PEDIDOS DE RESTAURANTES

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

Orientador: Prof. Me. Fernando Gonçalves Abadia

GOIÂNIA
2025

AGRADECIMENTOS

Agradeço, primeiramente, aos meus pais e à minha irmã, pelo amor, paciência e incentivo incondicional. Cada conquista ao longo da minha trajetória, dentro e fora da faculdade, só foi possível graças ao apoio constante e à confiança que sempre depositaram em mim.

Expresso também minha gratidão ao meu orientador, Prof. Me. Fernando Gonçalves Abadia, pela orientação atenciosa, pelas contribuições valiosas e pela dedicação em cada etapa deste trabalho.

Agradeço ao corpo docente da Pontifícia Universidade Católica de Goiás, que, com empenho e compromisso, proporcionou não apenas conhecimento técnico, mas também valores éticos e humanos que levarei para minha vida profissional.

Por fim, deixo um agradecimento especial aos meus amigos e colegas de classe, pela parceria, compreensão e incentivo durante toda a caminhada acadêmica. O apoio e a presença de cada um foram essenciais para tornar esta jornada mais leve.

RESUMO

O presente projeto de pesquisa tem como objetivo desenvolver um protótipo de aplicativo voltado ao gerenciamento de pedidos em restaurantes, com foco na otimização de processos operacionais e administrativos. A aplicação, denominada *FoodManager*, foi desenvolvida utilizando React e Vite no front-end, TypeScript em toda a stack, NestJS no back-end, PostgreSQL como sistema gerenciador de banco de dados e Prisma ORM na camada de persistência. A arquitetura adotada segue o modelo de cliente-servidor e API REST, fundamentada em boas práticas de modularização, separação de responsabilidades e comunicação entre serviços. A pesquisa é de natureza aplicada e de caráter exploratório-descritivo, buscando propor uma solução tecnológica que integre diferentes etapas do atendimento — desde o registro do pedido até o fluxo de caixa. O sistema visa reduzir erros humanos, agilizar o atendimento e aprimorar a gestão interna dos estabelecimentos. Os resultados obtidos reforçam a relevância das soluções digitais como instrumentos de transformação no setor gastronômico, evidenciando o papel da tecnologia na modernização e eficiência dos negócios de alimentação.

Palavras-chave: Sistemas Web. Gerenciamento de Pedidos. NestJS. React. PostgreSQL.

ABSTRACT

This research project aims to develop a prototype application for managing restaurant orders, with a focus on optimizing operational and administrative processes. The application, called *FoodManager*, was developed using React and Vite on the front-end, TypeScript throughout the stack, NestJS on the back-end, PostgreSQL as the database management system, and Prisma ORM on the persistence layer. The architecture adopted follows the client-server and REST API model, based on best practices for modularization, separation of responsibilities, and communication between services. The research is applied and exploratory-descriptive in nature, seeking to propose a technological solution that integrates different stages of service—from order entry to cash flow. The system aims to reduce human error, streamline service, and improve the internal management of establishments. The results obtained reinforce the relevance of digital solutions as instruments of transformation in the gastronomic sector, highlighting the role of technology in the modernization and efficiency of food businesses.

Keywords: Web Systems. Order Management. NestJS. React. PostgreSQL.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ORM	Object-Relational Mapping
API	Application Programming Interface
MEI	Microempreendedor Individual
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
JSX	JavaScript XML
DOM	Document Object Model
MVC	Model-View-Controller
SQL	Structured Query Language
SGBD	Sistema Gerenciador de Banco de Dados
MVCC	Multi-Version Concurrency Control
IDE	Integrated Development Environment
UX	User Experience
UI	User Interface
UML	Unified Modeling Language
MER	Modelo Entidade-Relacionamento
JSON	JavaScript Object Notation
JWT	JSON Web Token
URL	Uniform Resource Locator
SUS	System Usability Scale

SUMÁRIO

1 INTRODUÇÃO

Com o passar dos anos por consequência da globalização, o avanço tecnológico e o aumento do consumo de conteúdo audiovisual fizeram com que o ser humano se acostumasse à praticidade e à agilidade proporcionadas pelos produtos modernos. Não só nos ambientes tecnológicos e internet, mas também em ambientes como escolas, faculdades, bancos, e até mesmo em restaurantes. Por conta disso, a necessidade da utilização da tecnologia nesses ambientes a fim de atender as tendências humanas atualmente é algo comum. Como por exemplo, na rede de restaurantes, é normal a utilização de sistemas que auxiliem a sua produção e organização. (CASTELLS, 2006 ; JUNGER, 2020)

Em âmbitos tecnológicos relacionados a redes de restaurantes, o uso das comandas papel é quase nulo, já que é extremamente ineficiente. Garçons e caixas, no momento do pedido ou na hora do pagamento estão submetidos a erros simples ou até mesmo problemas na legibilidade. Na região da cozinha, os chefs estão propensos a cometer erros também na legibilidade, e interpretação incorreta dos dados escritos. Assim como diversos outros fatores que podem ocorrer com a comanda, como rasgos, perda etc. Problemas que entram em desacordo com a sociedade atual, que busca o prático e imediato.

Então por isso, é evidente as vantagens que uma aplicação gerenciadora pode trazer. As funcionalidades de criação de pedidos, a possibilidade de utilizar um cardápio digital, movimento e entrada do caixa, controle de produtos, controle de funcionários, acesso múltiplo de usuários, entre outras diversas funções que o sistema pode proporcionar. Segundo a consultora Karyna Muniz na feira "Restaurante de Sucesso", uma das cinco tendências no mercado de alimentos é otimização de processos, ela afirma que os empreendedores busquem novas tecnologias que auxiliem na produção. Assim podendo utilizar um sistema como uma nova solução tecnológica. (JULIO, 2017).

A partir de pesquisas do grupo NPD (empresa estadunidense de pesquisa de mercado) pode-se afirmar que após a pandemia do covid-19 os pedidos online se tornaram uma "norma"e tendem a crescer cada a vez mais. De acordo com eles, os pedidos digitais de restaurantes com serviço completo aumentaram 237% em 2021. Além disso, David Portalatin, consultor da indústria de alimentos da NPD, comenta que o pedido digital tem tudo a ver com conveniência e facilidade elevadas, dizendo que é isso que faz os

consumidores funcionarem (The NPD Group, 2020 ; BOSTON UNIVERSITY, 2023).

Nos últimos anos é possível acompanhar alguns tipos de softwares que trouxe benefícios a várias redes de restaurantes, como a 'Toast', um sistema de pedidos móveis que transforma o processo de pedidos em uma experiência ótima e rápida aos clientes e gerentes. O cliente a partir de um QR code e alguns cliques faz pedidos da forma que quiser sem ter que se estressar em grandes filas ou conversas longas. (BOSTON UNIVERSITY, 2023)

No âmbito operacional, restaurantes enfrentam constantes dificuldades na gestão logística, no abastecimento de insumos e na manutenção da qualidade, fatores diretamente relacionados à sobrevivência do negócio. A gestão financeira e o controle de custos são desafios centrais e permanentes para o empreendedor gastronômico . A necessidade de processos organizados tornou-se ainda mais evidente após a pandemia de COVID-19. É possível observar que entre 2020 e 2021, o setor enfrentou retração de 27,5% no faturamento, com isso, sendo forçado a utilizar outras maneiras para suprir a falta de clientes, como digitalização, delivery e reorganização de operações internas. Contexto que fez surgir práticas que permaneceram como pilares da competitividade atual (SEBRAE, 2024).

Nesse cenário, a tecnologia assume papel essencial. A digitalização tem mudado a forma como consumidores interagem com estabelecimentos, tornando aplicações de gestão parte indispensável da operação moderna. Além disso, a tecnologia auxilia de maneira inquestionável os negócios, ao automatizar fluxos, reduzir erros, otimizar atendimentos e melhorar o controle gerencial (SEBRAE, 2024).

Quando observados os indicadores de sobrevivência, o impacto da ausência de gestão é ainda mais claro: 50% dos MEIs gastronômicos fecham com apenas 10 meses de operação, reforçando que falhas organizacionais e ausência de ferramentas de controle estão entre as principais causas de mortalidade do setor (FGV ; ABRASEL, 2024).

É relevante estudar esse tema porque é perceptível a necessidade de novas implementações tecnológicas em vários ambientes, inclusive nas redes de restaurantes. A partir de um sistema gerenciador e ferramentas tecnológicas que fortaleçam sua gestão, reduzam vulnerabilidades, modernizem processos internos e contribuam para a continuidade e profissionalização dos restaurantes no Brasil.

Diante esse contexto, este projeto de pesquisa visa responder a seguinte questão:

Quais as vantagens de utilizar o FoodManger em sua rede de restaurante?

1.1 Justificativa

A justificativa para o estudo do impacto da implementação de sistemas gerenciadores, como o FoodManager, se fundamenta na evidência de que a utilização de tecnologia em restaurantes não é apenas uma opção, mas uma necessidade para garantir a competitividade e a satisfação do cliente. A pesquisa apresenta argumentos sólidos sobre como a digitalização pode mitigar problemas comuns enfrentados no setor, como erros de legibilidade, ineficiências nos processos de pedidos e pagamentos, controle de custos, entre outros. Além disso, a crescente aceitação de pedidos digitais, que aumentaram significativamente durante a pandemia, ressalta a relevância de um sistema que permita aos restaurantes se adaptarem a essas novas demandas do mercado.

1.2 Objetivos

1.2.1 Geral

- Desenvolver uma aplicação para gerenciamento de restaurantes, implementando funções que visam o melhor desempenho e produtividade fornecendo serviços aos gerentes, chefes, garçons e aos clientes usuários.

1.2.2 Objetivos Específicos

- Envio direto do pedido do cliente para área da cozinha, sem o uso de comandas de papel;
- Implementação de Autenticação (Administrador e Funcionário);
- Verificação de fluxo de caixa através de relatórios.

1.3 Objetivos esperados

O desenvolvimento do FoodManager visa comprovar a viabilidade técnica e funcional de um sistema web que pode centralizar e as operações fundamentais de pedidos de um restaurante. O principal resultado esperado é o desenvolvimento de um protótipo

funcional que reúna em uma única plataforma os módulos de pedidos, gestão de produtos e funcionários. Isso permitirá que diferentes usuários, como gerentes, garçons. Assim trazendo maior produtividade e qualidade de atendimento aos clientes, mostrando as vantagens que FoodManger, um sistema gerenciador de pedidos, pode trazer para os restaurantes que o utilizam.

1.4 Organização do trabalho

Este trabalho foi dividido em cinco capítulos, estruturados da seguinte maneira. No presente capítulo foi apresentada a introdução do trabalho, identificando o tema relacionado ao gerenciamento de pedidos e processos operacionais em restaurantes, bem como a motivação para o desenvolvimento do protótipo FoodManager, um sistema web destinado a otimizar o atendimento em estabelecimentos gastronômicos.

No Capítulo 2, será apresentada a fundamentação teórica, na qual são definidos os conceitos e abordagens que sustentam a proposta deste Trabalho de Conclusão de Curso.

O Capítulo 3, denominado materiais e métodos, detalha a metodologia utilizada no estudo, incluindo as fases de desenvolvimento, as ferramentas e equipamentos empregados, bem como os critérios e procedimentos seguidos para a implementação e validação do protótipo.

No Capítulo 4, intitulado desenvolvimento, é detalhado o processo de construção do sistema FoodManager, apresentando o desenho da arquitetura, os protótipos de interface, o ambiente de desenvolvimento, e as principais funcionalidades implementadas.

Por fim, o Capítulo 5, referente às considerações finais, traz as conclusões do trabalho, discutindo a relação entre os resultados obtidos e os objetivos propostos. Também são apresentadas sugestões de melhorias e propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar os fundamentos teóricos e as tecnologias que sustentam o desenvolvimento do FoodManager, detalhando as linguagens, frameworks e padrões de arquitetura utilizados na construção do sistema.

2.1 Linguagens

2.1.1 Typescript

O TypeScript é uma linguagem de programação de código aberto desenvolvida pela Microsoft com o objetivo de tornar o JavaScript mais robusto e escalável.O TypeScript é um superconjunto do JavaScript, o que significa que qualquer código JavaScript é também um código válido em TypeScript. A principal diferença é a adição de um sistema de tipagem estática opcional, que permite declarar explicitamente o tipo de variáveis, parâmetros e retornos de funções (MICROSOFT, 2024).

Essa tipagem torna possível identificar erros ainda na fase de desenvolvimento, antes da execução do programa, aumentando a segurança e a previsibilidade do código. Esse recurso é especialmente importante em aplicações de grande porte, nas quais múltiplos módulos e desenvolvedores trabalham simultaneamente. Além da tipagem, o TypeScript oferece suporte a conceitos de programação orientada a objetos, como classes, interfaces e herança, recursos que favorecem a organização e manutenção do código em sistemas complexos (MICROSOFT, 2024).

O código escrito em TypeScript é transpilado para JavaScript padrão, processo que converte as funcionalidades avançadas da linguagem para versões compatíveis com navegadores e outros ambientes de execução. Assim, o TypeScript preserva a compatibilidade com o ecossistema JavaScript, ao mesmo tempo em que adiciona ferramentas de produtividade e segurança que o tornam uma escolha amplamente adotada em projetos modernos de desenvolvimento web (ECMAScript, 2025).

Figura 1 – Exemplo de código Typescript

```typescript
interface Account {
  id: number
  displayName: string
  version: 1
}

function welcome(user: Account) {
  console.log(user.id)
}
```

```typescript
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Fonte: https://www.typescriptlang.org

2.1.2 CSS com Tailwind

O Cascading Style Sheets (CSS) é uma linguagem de marcação empregada para estabelecer a aparência e o layout de páginas web, possibilitando a distinção entre a camada de apresentação e a camada de conteúdo. O CSS é um componente fundamental das tecnologias básicas da Web, juntamente com HTML e JavaScript. Ele é encarregado de estabelecer propriedades visuais, como cores, espaçamentos, fontes e dimensões. Além de permitir a criação de interfaces adaptáveis a diferentes dispositivos e tamanhos de tela, essa separação simplifica a manutenção e a padronização visual das aplicações (W3C, 2023).

Nesse cenário, o Tailwind CSS se apresenta como um framework de utilitários que torna a estilização muito mais simples, sem a necessidade de criar longas e repetitivas regras CSS. Ele é fundamentado no uso de classes pré-definidas e na composição de utilitários, o que possibilita ao desenvolvedor aplicar estilos diretamente no HTML ou JSX, mantendo um controle total sobre o design, aumentando a produtividade, consistência visual, menos redundância (TAILWIND CSS, 2025).

Figura 2 – Exemplo de código Tailwind

```
<div class="flex flex-col items-center p-7 rounded-2xl">
  <div>
    <img class="size-48 shadow-xl rounded-md" alt="" src="/img/cover.png" />
  </div>
  <div class="flex">
    <span class="text-2xl font-medium">Class Warfare</span>
    <span class="font-medium text-sky-500">The Anti-Patterns</span>
    <span class="flex gap-2 font-medium text-gray-600 dark:text-gray-400">
      <span>No. 4</span>
      <span>·</span>
      <span>2025</span>
    </span>
  </div>
</div>
```


Fonte: https://tailwindcss.com

2.2 Frameworks e ferramentas

2.2.1 Vite

Vite é uma ferramenta de desenvolvimento que vem sendo cada vez mais utilizada por conta de facilitar o trabalho com aplicações web. Basicamento ele tem como objetivo simplificar o trabalho do desenvolvedor e deixar tudo menos burocrático. A documentação oficial do Vite descreve ele como um build tool pensado para aproveitar os recursos nativos dos navegadores atuais, especialmente os módulos ECMAScript (ESM), o que permite iniciar um servidor de desenvolvimento quase instantaneamente mesmo em projetos grandes (VITE, 2023).

Uma das maiores vantagens do Vite é que não é necessário empacotar todo o projeto antes de começar a rodar, ele modulariza os arquivos e envia para o navegador somente aquilo que ele precisa, por isso atende mais rapidamente alterações no código e não precisa fazer build em tudo de novo em caso de mudança.

2.2.2 React

O React é uma biblioteca JavaScript de código aberto, projetada principalmente para criar interfaces que melhoram a eficiência da experiência do usuário. Basicamente, ela permite que cada componente da aplicação seja autônomo e reutilizável. Ao substituir a manipulação direta do DOM (Document Object Model) pelo uso do Virtual DOM (uma representação na memória), o processo de renderização se torna muito mais eficiente, otimizando as atualizações na tela (BANKS, 2020).

Um dos grandes diferenciais do React é a sua arquitetura baseada em componentes, que promove uma divisão lógica da interface em pequenas partes coesas e reutilizáveis. Além disso, o uso do JSX (JavaScript XML) que uma sintaxe própria dele, torna a escrita do código mais intuitiva, unindo lógica e estrutura visual em um mesmo arquivo (WIE-RUCH,2018).

No contexto deste trabalho, o React foi utilizado no frontend do FoodManager por sua capacidade de lidar com atualizações dinâmicas de dados sem comprometer o desempenho da aplicação.

2.2.3 NestJS

O NestJS é um framework avançado para Node.js que usa TypeScript como base e incorpora conceitos de arquitetura modular, injeção de dependência e programação orientada a objetos. O framework mescla conceitos do Angular com práticas consolidadas da engenharia de software, levando a uma metodologia fortemente baseada no padrão Model-View-Controller (MVC) (NESTJS, 2025).

Desde o início do projeto, o NestJS visa fomentar a padronização e a coesão estrutural. Ele é fundamentado em módulos que reúnem controladores, serviços e entidades, possibilitando que cada componente da aplicação seja autônomo e de fácil reutilização.Essa configuração modular torna o código mais testável e fácil de manter, características que são necessárias em sistemas mais complexos (FOWLER, 2019).

Além disso, o NestJS oferece suporte nativo à criação de APIs, e por isso foi uma das escolhas para utilzação nesse projeto, sendo essencial no backend e para criação de uma API organizada.

2.2.4 Prisma ORM

O Prisma ORM é uma ferramenta contemporânea de mapeamento objeto-relacional (Object-Relational Mapping) criada para otimizar a interação entre aplicações desenvolvidas em TypeScript e bancos de dados relacionais. Seu objetivo é simplificar o acesso aos dados, proporcionando uma camada de abstração segura e intuitiva em relação aos comandos SQL convencionais.(PRISMA,2025)

O ORM funciona como um intermediário entre a aplicação e o banco de dados, convertendo objetos e métodos de alto nível em consultas SQL eficientes. Nesse sentido, o uso de ORMs é uma prática estabelecida em sistemas corporativos, pois possibilita o desacoplamento da camada de persistência da lógica de negócios, o que torna o sistema mais coeso e fácil de manter (FLOWER, 2019).

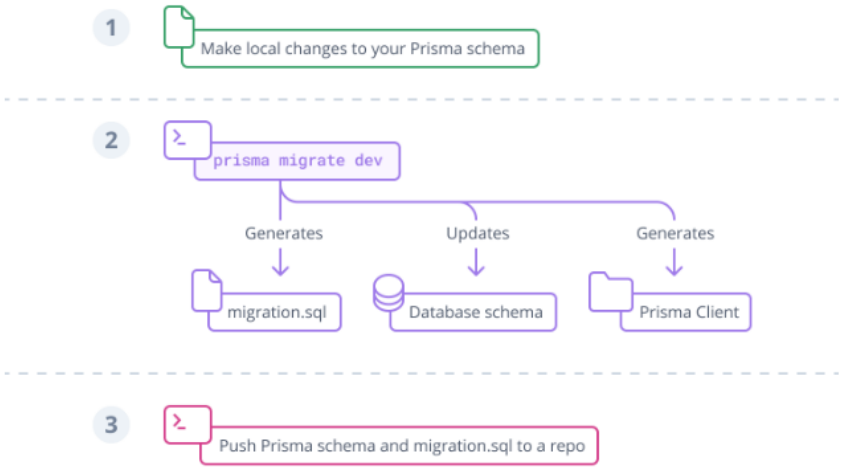
Além disso possui integração nativa com diversos bancos de dados, como PostgreSQL, que foi o escolhido nesse projeto, oferecendo recursos como migrações automáticas, autocompletar de queries e validação de esquema em tempo de compilação.

Figura 3 – Exemplo de métodos de alto nível

```
import { PrismaClient } from '@prisma/client'
// or
// const { PrismaClient } = require('@prisma/client')

const prisma = new PrismaClient()

// use inside an `async` function to `await` the result
await prisma.user.findUnique(...)
await prisma.user.findMany(...)
await prisma.user.create(...)
await prisma.user.update(...)
await prisma.user.delete(...)
await prisma.user.upsert(...)
```

Fonte: https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma

Figura 4 – Funcionamento de Migrate



Fonte: https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma

2.2.5 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados relacional (SGBD) de código aberto, reconhecido por sua estabilidade, segurança e aderência aos princípios do modelo relacional. Eles abrangem a organização dos dados em tabelas conectadas, o uso de chaves primárias e estrangeiras, bem como a garantia da integridade referencial. O PostgreSQL adota esses princípios de forma ampla, assegurando consistência lógica e previsibilidade na gestão dos dados.

Um dos principais diferenciais do PostgreSQL está na sua capacidade de lidar com grandes volumes de dados e na implementação do mecanismo MVCC (Multiversion Concurrency Control), que permite que múltiplos usuários acessem o banco simultaneamente sem bloqueios significativos. Além disso, oferece suporte a transações ACID (Atomicity, Consistency, Isolation, and Durability), funções armazenadas, índices avançados e suporte nativo a JSON, tornando-se adequado tanto para dados estruturados quanto semiestruturados (POSTGRESQL, 2025).

Visto que esse projeto tem a ideia de utilizar vários tablets nos momentos de atendimento ao cliente a tecnologia do MVCC é essencial, já que várias queries serão feitas ao mesmo tempo.

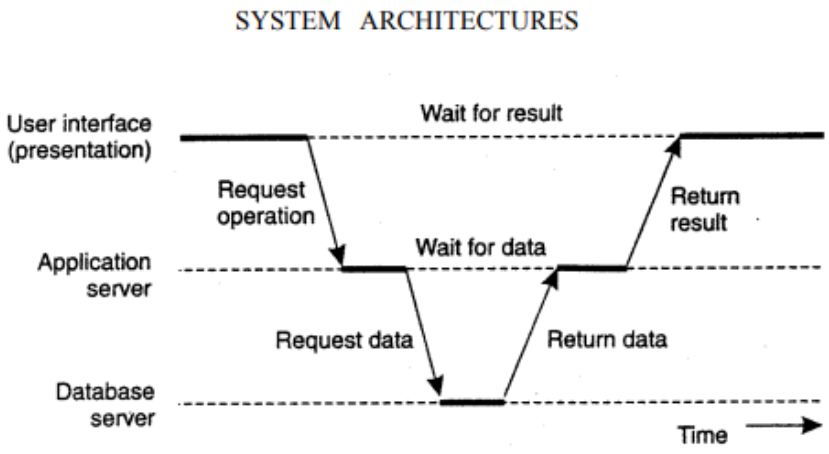
2.3 Arquitetura

2.3.1 Cliente-Servidor

A arquitetura cliente-servidor é um dos modelos mais tradicionais e consolidados no desenvolvimento de sistemas distribuídos. Basicamente a arquitetura se baseia na divisão das responsabilidades, onde o cliente é responsável por fazer a interação com o usuário, atendê-lo e levar as requisições para o servidor, que é encarregado de processar essas solicitações, calcula se necessário e tem o dever de responder as chamadas do cliente . Essa separação permite que cada parte execute papéis distintos dentro da aplicação, promovendo maior organização e facilitando a manutenção do sistema (TANENBAUM ; VAN STEEN, 2007)

Um dos motivos para a ampla adoção desse modelo é sua capacidade de distribuir o processamento de forma eficiente. O servidor concentra operações mais rigorosas e quem não podem ter falhas, como acesso ao banco de dados e regras de negócio, enquanto o cliente lida apenas com a interface, comunicação com usuário e a apresentação das informações. Isso faz com que a carga computacional do usuário final seja menor e possibilita que múltiplos clientes acessem simultaneamente os serviços oferecidos pelo servidor(TANENBAUM ; VAN STEEN, 2007).

Figura 5 – Funcionamento Básico da Arquitetura Cliente-Servidor



Fonte: TANENBAUM ; VANSTEEN, 2007

2.3.2 API Rest

As APIs REST (Representational State Transfer) são uma forma de comunicação amplamente utilizada no desenvolvimento de sistemas web. Sem usar estruturas complexas o REST utiliza de requisições HTTP para fazer o CRUD(Create, Read, Update, Dele) de informações. Basicamente essa abordagem faz com que qualquer elemento relevante do sistema, seja um produto, usuário ou comanda possam ser acessados facilmente por de URLs bem definidas junto com os verbos do protocolo HTTP como GET, POST, PUT e DELETE (RICHARDSON ; RUBY, 2013).

Além disso, o que é imporante destacar é que o servidor não precisa manter informações sobre o estado do cliente entre uma requisição e outra, que é chamado de *stateless* ou seja, cada requisição tem tudo o que o servidor precisa para responder e qualquer requisição pode ir para qualquer instância do servidor, que torna o sistema mais previsível e mais fácil de escalar.

No projeto essa é arquitetura que suntenta aplicação, o frontend em React trabalha como se fosse o cliente, já o backend em NestJS funciona como servidor, processando requisições, verificando e manipulando dados no Postgress e respondendo de forma estrutura por meio das chamadas à API REST.

3 MATERIAIS E PROCEDIMENTOS METODOLÓGICOS

Este capítulo tem como propósito descrever os procedimentos adotados e as estratégias aplicadas no desenvolvimento deste trabalho, correspondente ao Trabalho de Conclusão de Curso II. Serão apresentadas as principais ferramentas utilizadas na construção do FoodManager, sistema web destinado ao gerenciamento de pedidos e processos internos de restaurantes. A implementação foi conduzida com base nos fundamentos teóricos abordados no capítulo anterior, transformando os conceitos de arquitetura web em soluções práticas que sustentam o funcionamento do protótipo proposto.

3.1 Metodologia

Esta pesquisa, segundo sua natureza é um resumo de assunto, buscando explicar a área do conhecimento do projeto, indicando sua evolução histórica, como resultado da investigação das informações obtidas, levando ao entendimento de suas causas e explicações (WAZLAWICK, 2014).

Segundo os objetivos é uma pesquisa exploratória e descritiva. Sendo a exploratória, uma pesquisa onde o autor não necessariamente possui uma hipótese ou objetivos definidos, sendo geralmente o primeiro passo de um processo de pesquisa longo. Nela o autor examina alguns fenômenos buscando resultados e pontos fora da curva ou ainda não são conhecidos (WAZLAWICK, 2014).

Já a pesquisa descritiva, pode-se trazer dados mais conectados com a realidade, mesmo assim não possuem interferência entre pesquisador e teorias que explicam fenômenos, todavia ainda tem objetivo de descrever os fatos justamente como eles são (WAZLAWICK, 2014).

E segundo seus procedimentos técnicos, esta pesquisa é bibliográfica. A pesquisa bibliográfica basicamente não produz conhecimento novo, mas sim aplica ao pesquisador conhecimento que ele ainda não possuía (WAZLAWICK, 2014).

A pesquisa bibliográfica, será realizada a partir de materiais que já foram publicados, teses, livros, revista, materiais da internet, entre outros. (GIL, 2017).

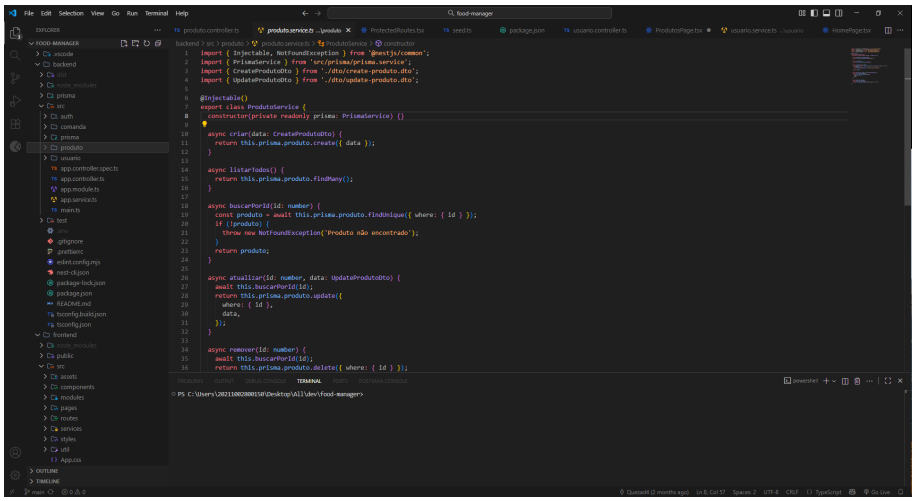
3.2 Materiais

3.2.1 Visual Studio Code

Desenvolvido pela Microsoft, o Visual Studio Code é um ambiente de desenvolvimento integrado (IDE) leve e compatível com várias plataformas. Ele oferece suporte a várias linguagens de programação e extensões que expandem suas capacidades, como gerenciamento de versões com Git, depuração em tempo real e integração com frameworks modernos, como React e NestJS.

A interface do VS Code é simples e personalizável, e sua ampla comunidade de extensões faz com que seja uma das ferramentas mais usadas por desenvolvedores web, oferecendo produtividade, organização e suporte para projetos de várias escalas.

Figura 6 – Visual Studio Code



Fonte: O autor.

3.2.2 Figma

O Figma é uma ferramenta online de design de interface e prototipação colaborativa, amplamente utilizada para o desenvolvimento de layouts e fluxos de navegação de aplicações web e mobile. Por ser baseada em nuvem, permite que vários usuários trabalhem simultaneamente no mesmo projeto, facilitando o processo de criação e validação visual.

No contexto do FoodManager, o Figma foi utilizado para o desenho das telas e para a simulação da experiência do usuário (UX/UI), garantindo consistência visual e facilitando a comunicação entre as etapas de design e implementação. As telas do protótipo serão

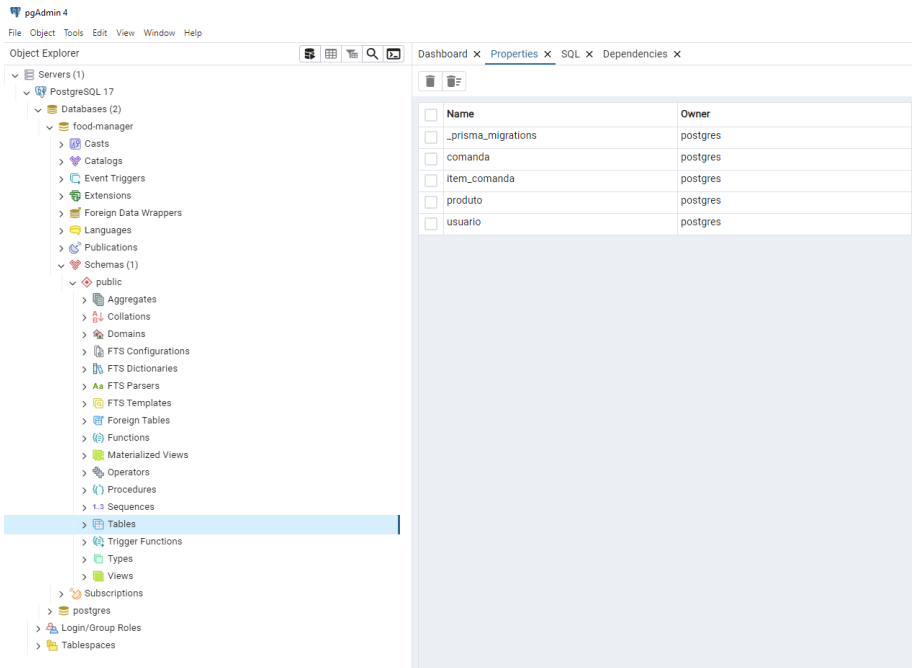
apresentadas e melhor descritas no capítulo posterior.

3.2.3 pgAdmin

O pgAdmin é uma ferramenta gráfica desenvolvida para facilitar a administração e o gerenciamento de bancos de dados PostgreSQL. Ele oferece uma interface visual que permite executar de forma prática diversas tarefas que normalmente exigiriam o uso de comandos no terminal. Com o pgAdmin, é possível criar e modificar bancos de dados, tabelas além de executar consultas SQL com destaque de sintaxe e acompanhar o histórico de execução.

No projeto FoodManager, o pgAdmin foi utilizado como ferramenta de apoio para o gerenciamento do banco de dados PostgreSQL, que serve como base do sistema. Ele foi essencial durante o processo de desenvolvimento e testes, permitindo a criação, visualização e manutenção das tabelas do banco, além do monitoramento das relações entre entidades, como Produto, Comanda, Usuário.

Figura 7 – pgAdmin



Fonte: O autor.

3.2.4 LucideChart e Draw.io

O Lucidchart e o Draw.io são ferramentas de modelagem visual voltadas à criação de diagramas e fluxogramas, amplamente utilizadas para representar arquiteturas de software,

bancos de dados e casos de uso. Ambas permitem a construção intuitiva de diagramas UML, ER (Entidade-Relacionamento) e estruturas lógicas de sistemas.

Durante o desenvolvimento do FoodManager, essas ferramentas foram empregadas para a modelagem dos diagramas de caso de uso, modelos de banco de dados, auxiliando na documentação técnica do projeto.

3.2.5 Computadores

Tabela 1 – Características do desktop utilizado

MODELO	DESKTOP
SISTEMA OPERACIONAL	Windows 10 Pro
PROCESSADOR	AMD Ryzen 5700
MEMÓRIA RAM	32 GB
TIPO DE SISTEMA	64 BITS
PLACA DE VÍDEO	NVIDIA GeForce GTX1050 Ti
DISCOS RÍGIDOS	Toshiba 1TB

Fonte: O autor.

Tabela 2 – Características do notebook utilizado

MARCA	LENOVO
MODELO	LENOVO IdeaPad 3
SISTEMA OPERACIONAL	Linux Mint
PROCESSADOR	AMD Ryzen 7 5700U
MEMÓRIA RAM	12 GB
TIPO DE SISTEMA	64 BITS
PLACA DE VÍDEO	Integrated AMD Graphics
DISCOS RÍGIDOS	512GB SSD

Fonte: O autor.

3.2.6 Tablet

Tabela 3 – Características do tablet utilizado

MARCA	SAMSUNG
MODELO	SAMSUNG Galaxy Tab S8 Lite
SISTEMA OPERACIONAL	Android 12
PROCESSADOR	Snapdragon 8 Gen1 Qualcomm
MEMÓRIA RAM	8 GB
DISCOS RÍGIDOS	256GB

Fonte: O autor.

4 DESENVOLVIMENTO DO PROJETO

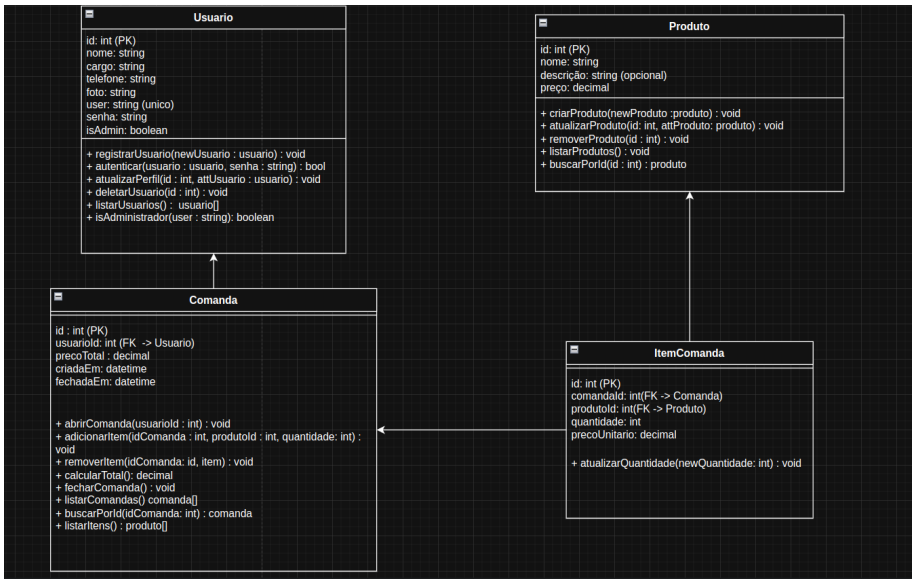
4.1 Prototipação

4.1.1 Modelo de Entidade Relacionamento

O Modelo de Entidade Relacionamento (MER) foi a primeira etapa concreta na definição da estrutura lógica do FoodManager. Ele foi útil na parte das informações que sistemas deveria armezenar, como atributos principais e a relações entre elas. A construção do MER facilitou a a visualização de como os usuários, produtos e comandas devem interagir dentro da aplicação garantindo desde o início que o banco de dados fosse consistente e capaz de atender às necessidades reais de um restaurante.

Além disso, já nessa fase surgiram discussões sobre a diferença entre os perfis de usuário. Desde o princípio, ficou evidente a necessidade de um administrador com mais permissões que os funcionários comuns, pois o gerente de um restaurante geralmente precisa ter acesso a cadastro de produtos, gerenciamento de funcionários enquanto garçons precisam apenas das funcionalidades relacionadas ao atendimento. Essa premissa influenciou a modelagem inicial e guiou a construção posterior de regras de acesso e autenticação na API.

Figura 8 – Modelo de Entidade Relacionamento



Fonte: O autor.

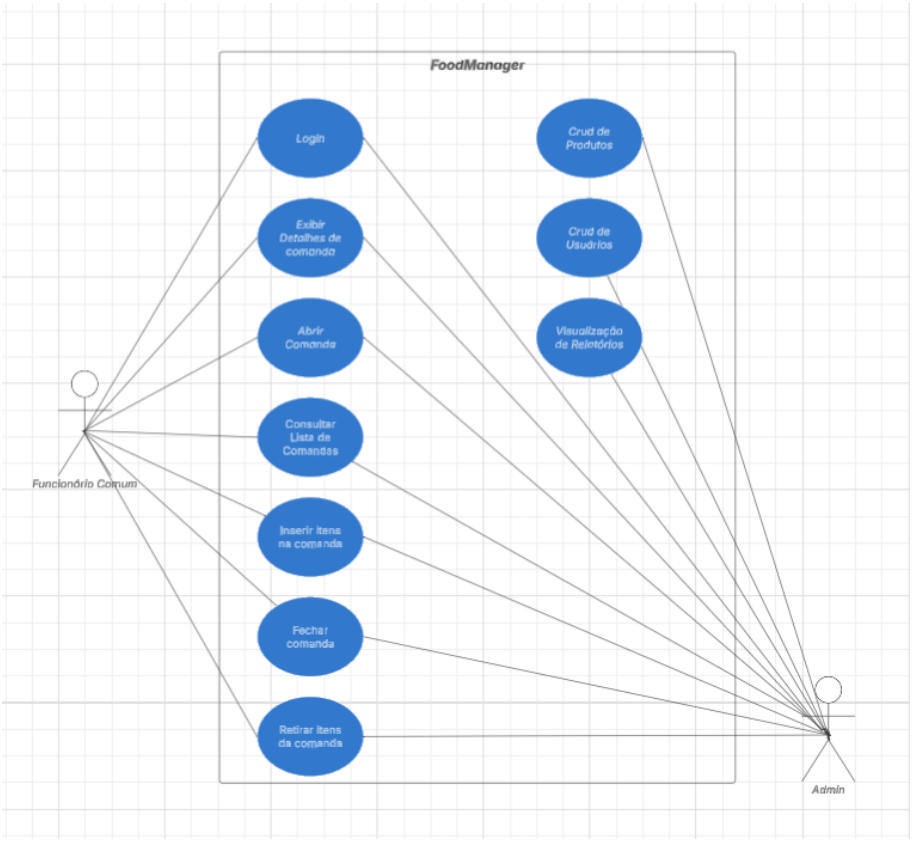
Na figura pode se observar as tabelas de Usuários, Produtos, Comandas e ItemCo-

manda todas com seus respectivos atributos e relacionamentos.

4.1.2 Diagrama de Casos de Uso

O Diagrama de Casos de Uso foi desenvolvido com o objetivo de representar funcionalmente o que cada tipo de usuário pode fazer dentro do FoodManager. Ele serviu como uma visão geral do sistema antes da etapa de implementação, ajudando a estruturar de maneira organizada as interações entre os usuários e as funcionalidades planejadas.

Figura 9 – Diagrama de Casos de Uso



Fonte: O autor.

Na figura, a separação entre Administrador e Funcionário aparece de forma explícita. O funcionário possui um conjunto mais direto de ações: criar comandas, registrar pedidos e atualizar status. Já o administrador, além das ações do funcionário, possui um conjunto ampliado de casos de uso, como visualização de relatórios, criação e manipulaçao de produtos e funcionários. Essa distinção não apenas facilitou a documentação, mas também guiou o trabalho no backend.

4.1.3 Protótipo Figma

O protótipo produzido no Figma representou a versão visual inicial do FoodManager. Ele estabeleceu as cores, identidade visual, componentes principais e layout das telas mais importantes do sistema.

Figura 10 – Projeto Inicial do Figma

Fonte: O autor.

4.1.3.1 Tela de Loading

No protótipo inicial, foi criada uma tela de loading simples, que tinha como propósito introduzir a identidade visual do sistema enquanto os primeiros componentes eram carregados. Essa versão inicial apresentava apenas o logotipo do projeto acompanhado do slogan, funcionando mais como um elemento conceitual do que como uma funcionalidade definitiva.

Essa tela representava apenas uma primeira hipótese de design. Com o avanço do projeto e a consolidação das decisões técnicas e de usabilidade, ela acabou sendo alterada e, em algumas etapas, até mesmo descartada.

Figura 11 – Tela de Loading

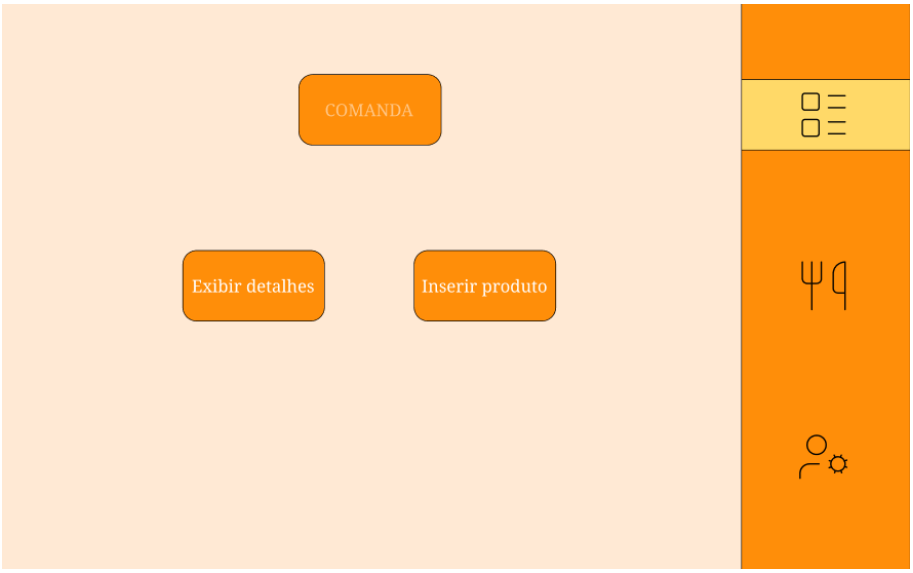
Fonte: O autor.

4.1.3.2 Tela Principal

A tela principal apresentada nesta primeira versão representa um rascunho inicial da estrutura de navegação do sistema. Nesta versão, a interface se concentra em três elementos centrais, o campo destinado ao número da comanda e dois botões principais “Exibir detalhes” e “Inserir produto”. A proposta inicial era oferecer um acesso direto às duas funções mais recorrentes dentro de um restaurante.

No lado direito da tela, uma barra lateral com ícones representando o menu geral do sistema. Embora ainda não completa neste estágio, essa barra já antecipava a futura organização das categorias de navegação.

Figura 12 – Tela Principal



Fonte: O autor.

4.1.3.3 Tela de Categorias

A tela de categorias surgiu como ideia inicial para que não seja necessário toda vez buscar o mesmo produto, e sim escolhendo a categoria dele, facilitando a busca.

O topo da tela apresenta o número da comanda, seguido de um campo de busca para facilitar o acesso direto a produtos específicos. Logo abaixo, cada categoria é exibida como um bloco visual amplo, com foto ao fundo e título em destaque, permitindo ao usuário identificar rapidamente o tipo de item que deseja adicionar. Entre as categorias previstas no protótipo estão “Petiscos”, “Pratos”, “Sanduíches”, “Pizzas”, “Espetinhos”, “Sobremesas”, “Refrigerantes”, “Drinks/Bebidas”, “Cervejas”, entre outras.

Figura 13 – Tela de Categorias

Fonte: O autor.

Embora essa tela represente uma etapa importante no processo de design, diversas ideias presentes no protótipo inicial foram ajustadas ou substituídas durante a implementação real.

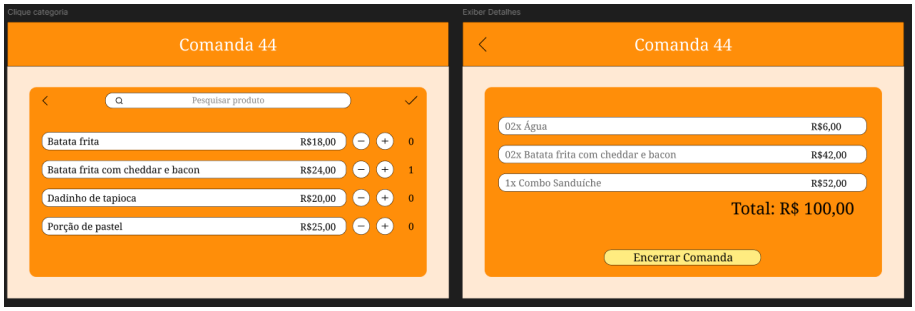
4.1.3.4 Tela de Detalhes/Adicionar Item

As telas apresentadas representam um estágio intermediário do protótipo inicial, no qual o fluxo de adição de itens à comanda e a visualização dos detalhes foram estruturados de maneira mais clara e próxima do funcionamento real do sistema.

No lado esquerdo da imagem, observa-se a tela destinada à seleção de produtos. O cabeçalho exibe o número da comanda e logo abaixo há um campo de pesquisa, permitindo filtrar itens rapidamente. Cada produto é apresentado juntamente com seu valor e acompanhado pelos botões de incremento e decremento, possibilitando ao atendente ajustar quantidades antes de confirmar a inclusão.

Já no lado direito aparece a tela de Exibir Detalhes, que funciona como um resumo da comanda selecionada. Nela, o usuário visualiza claramente a quantidade de cada item, suas descrições e o valor correspondente, além do total parcial calculado automaticamente. A presença do botão “Encerrar Comanda” representa a etapa final desse fluxo.

Figura 14 – Tela de Detalhes/Adicionar Item



Fonte: O autor.

4.1.3.5 Tela de Produtos

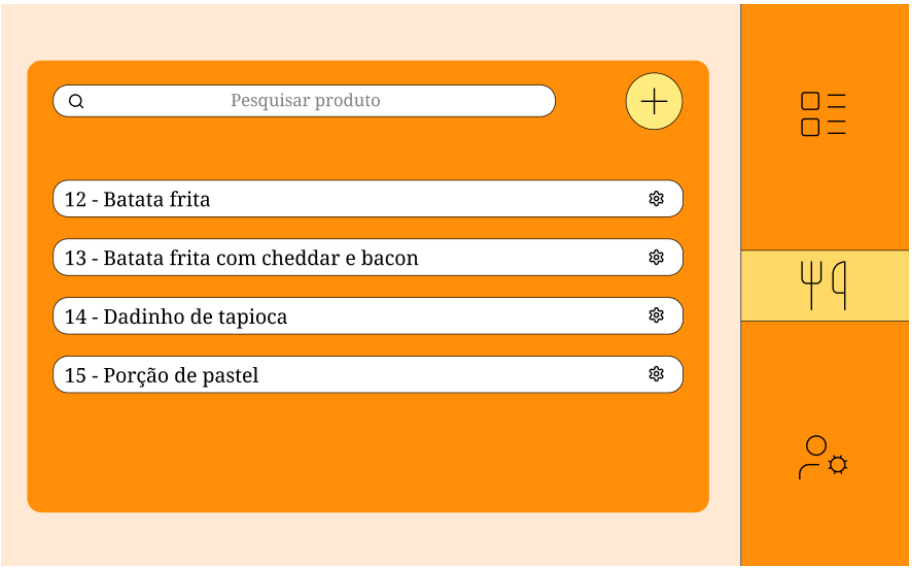
A tela apresentada corresponde a uma das primeiras versões do protótipo responsável pela visualização e gerenciamento dos produtos cadastrados no sistema.

O layout organiza os produtos em uma lista vertical simples, exibindo o código e o nome de cada item (“12 – Batata frita”, “13 – Batata frita com cheddar e bacon”, etc.). Essa estrutura foi pensada para facilitar a leitura rápida e permitir ao administrador identificar, de imediato, quais itens já estavam cadastrados. Ao lado de cada produto há um ícone de engrenagem, representando a ação de configurar ou editar.

Na parte superior, a barra de busca permite filtrar itens evitando assim, a dificuldade em localizar rapidamente um produto específico. Logo ao lado desse campo, o botão circular com símbolo de “+” representa a função de cadastrar um novo produto.

À direita, o menu lateral , assim como na tela principal, se destacam os ícones que representavam as principais rotas do sistema.

Figura 15 – Tela de Produtos



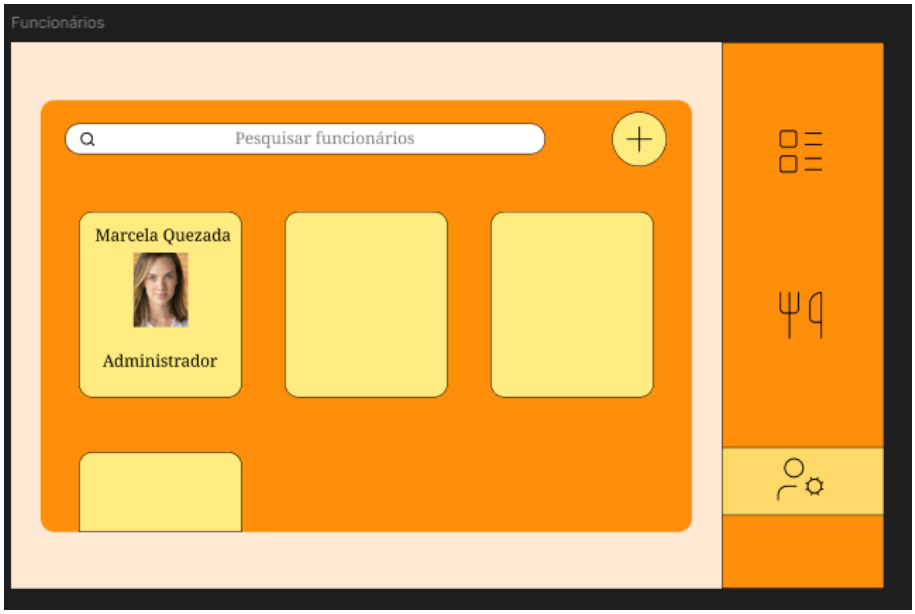
Fonte: O autor.

4.1.3.6 Tela de Funcionários

A imagem apresentada corresponde à primeira versão prototipada da tela de gerenciamento de funcionários da aplicação. Muito semelhante a tela de produtos, apresenta ao topo uma barra de pesquisa, facilitando a busca dos funcionários e também o botao circular de "+, representando a função de criação e cadastro de funcionários.

A área central é composta por cartões destinados a exibir cada funcionário cadastrado. No protótipo exibido, apenas um cartão está preenchido, exibindo uma foto, o nome “Marcela Quezada” e o cargo “Administrador”. A ideia era que cada cartão apresentasse informações visuais e objetivas, facilitando a identificação e permitindo acesso rápido às ações de edição e exclusão. Os demais cartões vazios representam espaços reservados para testar o comportamento da interface quando vários funcionários fossem adicionados.

Figura 16 – Tela de Funcionários



Fonte: O autor.

Embora grande parte das telas já estivesse definida nessa etapa inicial como login, página de produtos, página de comandos e parte da página de funcionários, muitas outras surgiram somente durante o desenvolvimento da aplicação. À medida que as regras de negócio foram implementadas e a API começou a tomar forma, novas necessidades apareceram naturalmente.

4.2 Construção da aplicação

A construção da aplicação foi guiada pelo uso de tecnologias amplamente adotadas no mercado corporativo. React, Vite, NestJS, TypeScript, Prisma e PostgreSQL, todas são ferramentas consolidadas no desenvolvimento moderno de sistemas web e oferecem facilidade tanto quanto no desenvolvimento quanto na manutenção.

4.2.1 Modelagem do Banco de Dados

O projeto do Food-Manager se iniciou através da Modelagem do Banco de Dados, a partir de uma ideia já estruturada anteriomente na prototipação e na criação de um Modelo Entidade Relacionamento.

A modelagem foi guiada pelas necessidade reais de um restaurante, registrar pedidos, controlar comandas e organizar os produtos. O banco foi estruturado no PostgreSQL, sempre

mantendo a integridade nas relações entre as entidades tentando ao máximo que os dados refletissem um fluxo normal de restaurante.

4.2.1.1 Entidades

A entidade Usuário carrega os atributos comuns para funcionários e administradores, como nome, cargo, telefone e foto que são apenas para controle de quem utiliza a aplicação. User e senha são os atributos responsáveis para autenticação, e já isAdmin é um atributo booleano que diferencia um usúario comum de um administrador, campo que também atua na camada de autenticação e autorização que seram mostrados posteriormente.

A entidade Produto armazena os itens do cardápio do restaurante, carrega o nome, uma breve descrição e também o preço unitário daquele produto.

As Comandas representam as solicitações feitas pelos garçons para aquela comanda específica, carregam os Status (se aberta ou fechada) além de atributos de controle, como horários que foram criadas e fechadas, quem foi o autor da comanda etc. Para permitir que cada pedido tenha vários itens, foi criada uma entidade intermediária ItemComanda que a armzanena a quantidade daquele item e também o subtotal daqueles produtos. Essa abordagem segue a lógica de um relacionamento muitos-para-muitos entre comandas e produtos, resolvido com uma tabela adicional.

Do ponto de vista de implementação, toda essa modelagem foi traduzida para o código por meio do Prisma ORM, que atuou como ponte entre o NestJS e o PostgreSQL. Cada entidade do MER se tornou um modelo Prisma, com suas respectivas chaves primárias, chaves estrangeiras e relacionamentos. O Prisma foi responsável por gerar as migrações do banco de dados, garantindo que a estrutura física estivesse sempre alinhada com o modelo lógico definido no projeto.

Figura 17 – Código Schema-Prisma

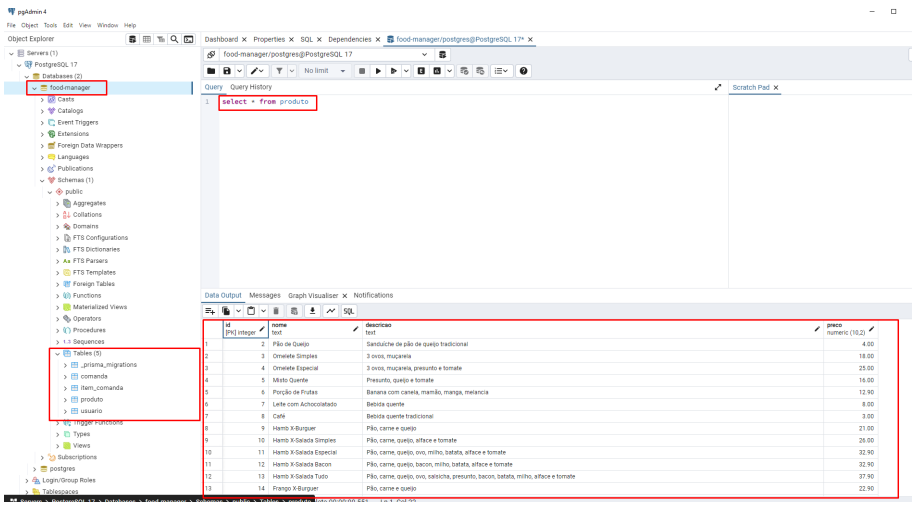
```
model Produto {
  id        Int     @id @default(autoincrement())
  nome      String
  descricao String?
  preco     Decimal @db.Decimal(10, 2)

  itens ItemComanda[]

  @@map("produto")
}

model Usuario {
  id       Int     @id @default(autoincrement())
  nome     String
  cargo    String
  telefone String?
  foto     String?
  user     String  @unique // login único
  senha    String // armazenada com hash (bcrypt)
  isAdmin  Boolean @default(false)

  comandas Comanda[]

  @@map("usuario")
}

model Comanda {
  id        Int           @id @default(autoincrement())
  usuarioId Int
  status    StatusComanda @default(ABERTA)
  criadaEm  DateTime      @default(now())
  fechadaEm DateTime?
  total     Decimal       @default(0) @db.Decimal(12, 2)

  usuario Usuario       @relation(fields: [usuarioId], references: [id])
  itens   ItemComanda[]

  @@map("comanda")
}

model ItemComanda {
  id            Int     @id @default(autoincrement())
  comandaId     Int
  produtoId     Int
  quantidade    Int
  precoUnitario Decimal @db.Decimal(10, 2)
  subtotal      Decimal @db.Decimal(12, 2)

  comanda Comanda @relation(fields: [comandaId], references: [id])
  produto Produto @relation(fields: [produtoId], references: [id])

  @@unique([comandaId, produtoId])
  @@index([comandaId])
  @@index([produtoId])
  @@map("item_comanda")
}
```

Fonte: O autor.

4.2.1.2 Tabelas e Sistema Gerenciador de Banco de Dados

Como falado anteriomente o banco de dados foi estruturado utilizando PostgreSQL, sendo que todas as tabelas podiam ser visualizadas e manipuladas pelo pgAdmin.

Figura 18 – Esquema das Tabelas e Query no pgAdmin



Fonte: O autor.

Nessa interface, é possível visualizar a organização das tabelas criadas pelo Prisma ORM, bem como executar consultas SQL para inspecionar o conteúdo armazenado durante o desenvolvimento.

A imagem destaca a execução de um simples SELECT * FROM produto, exibindo todos os registros cadastrados na tabela produto. A visualização no pgAdmin foi fundamental tanto para validar a consistência dos dados durante a implementação quanto para testar rapidamente o comportamento das rotas da API que dependem dessas informações.

4.2.2 Desenvolvimento Back-End

O backend do FoodManager foi desenvolvido utilizando o NestJS. Desde o começo do projeto a ideia era a construção de uma API escalável e fácil de manter, e o Nest, por apresentar uma estrutura clara foi uma ótima escolha.

Uma das primeiras funcionalidades implementadas foi o processo de autenticação. Todas as senhas dos usuários são criptografadas utilizando o **bcrypt**, fazendo com que nenhum dado sensível seja armezenado de forma pura no banco de dados. Quando o usuário realiza o login, o sistema valida a senha encriptada e, em seguida, gera um token

JWT (JSON Web Token) contendo informações essenciais, como o ID do usuário e seu nível de permissão. Esse token é devolvido ao frontend e utilizado em todas as requisições posteriores para manter o usuário autenticado.

Figura 19 – auth.service.ts

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcrypt';
import { UsuarioService } from 'src/usuario/usuario.service';

@Injectable()
export class AuthService {
  constructor(private users: UsuarioService, private jwt: JwtService) {}

  async validateUser(user: string, senha: string) {
    const found = await this.users.findByUser(user);
    if (!found) throw new UnauthorizedException('Credenciais inválidas');
    const ok = await bcrypt.compare(senha, found.senha);
    if (!ok) throw new UnauthorizedException('Credenciais inválidas');
    const { senha: _, ...pub } = found;
    return pub;
  }

  async login(user: string, senha: string) {
    const pub = await this.validateUser(user, senha);
    const payload = { sub: pub.id, user: pub.user, isAdmin: pub.isAdmin };
    return {
      access_token: await this.jwt.signAsync(payload),
      user: pub,
    };
  }
}
```

Fonte: O autor.

Já a autorização de quem pode de quem pode fazer determinadas requisições foi tratada com criação dos Role Guards, as rotas que apenas os administrados podem entrar possuem um guard que verifica se o token apresenta o papel correto. Dessa forma, mesmo que um usuário comum tente acessar uma rota de administrador, a API simplesmente bloqueia a requisição e retorna uma resposta de acesso negado.

Figura 20 – role.guard.ts

```
import { CanActivate, ExecutionContext, ForbiddenException, Injectable } from '@nestjs/common';
import { Reflector } from '@nestjs/core';

@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(ctx: ExecutionContext): boolean {
    const req = ctx.switchToHttp().getRequest();
    const handler = ctx.getHandler();

    const requiredRoles = this.reflector.get<string[]>('roles', handler);
    if (!requiredRoles || requiredRoles.length === 0) return true;

    const user = req.user;
    if (!user) throw new ForbiddenException('Não autenticado');

    const roleDoUsuario = user.isAdmin ? 'admin' : 'user';
    if (!requiredRoles.includes(roleDoUsuario)) {
      throw new ForbiddenException('Acesso negado: permissão insuficiente');
    }
    return true;
  }
}
```

Fonte: O autor.

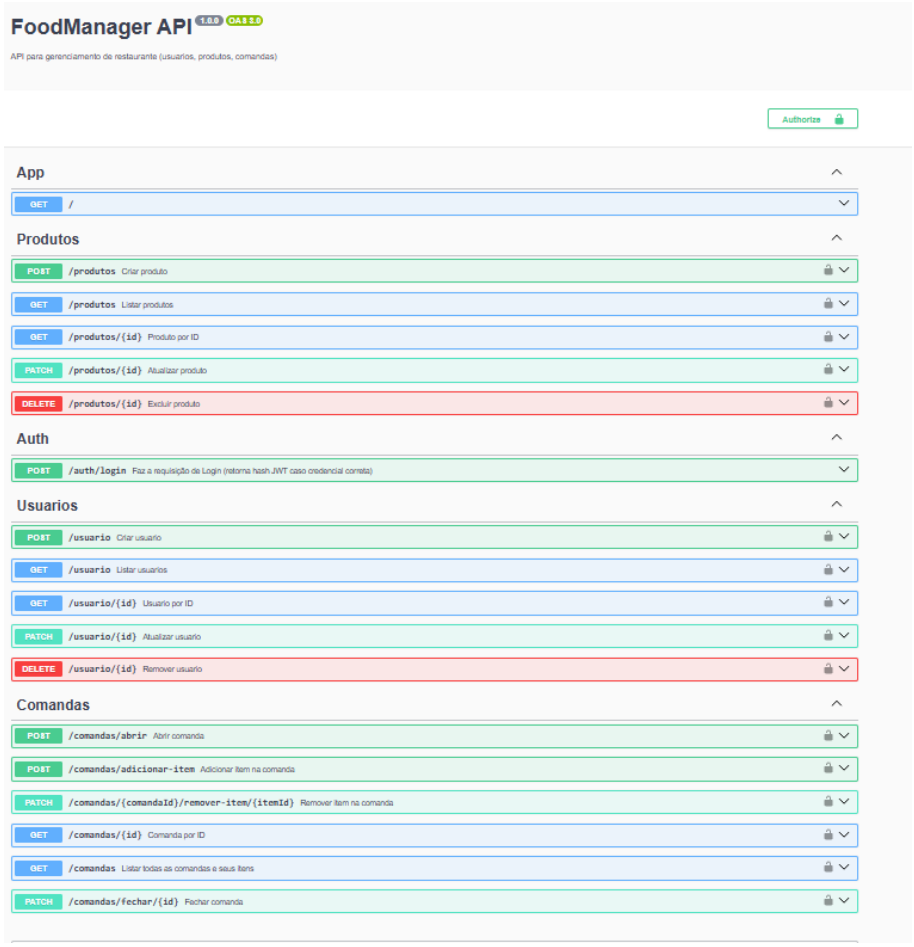
A comunicação com o banco de dados PostgreSQL foi realizada por meio do Prisma ORM, que serviu como ponte entre o backend e o modelo relacional definido previamente.

4.2.2.1 Swagger

Durante o desenvolvimento do backend, um recurso que ajudou bastante foi a integração do Swagger, uma ferramenta que permite documentar e visualizar interativamente todos os endpoints da API. Na prática a utilização do Swagger foi essencial, ele atualiza automaticamente a documentação fazendo com que os testes sejam feitos de maneira muito mais eficiente sem ter que fazer manualmente pelo Postman ou escrever JSON várias vezes.

Além disso o Swagger facilitou a manter um padrão nas respostas da API, ajudando a achar problemas nas requisições de maneira mais tempestiva. Por fim, acabou servindo tanto como documentação quanto como ferramenta de teste rápido.

Figura 21 – Endpoints do Swagger

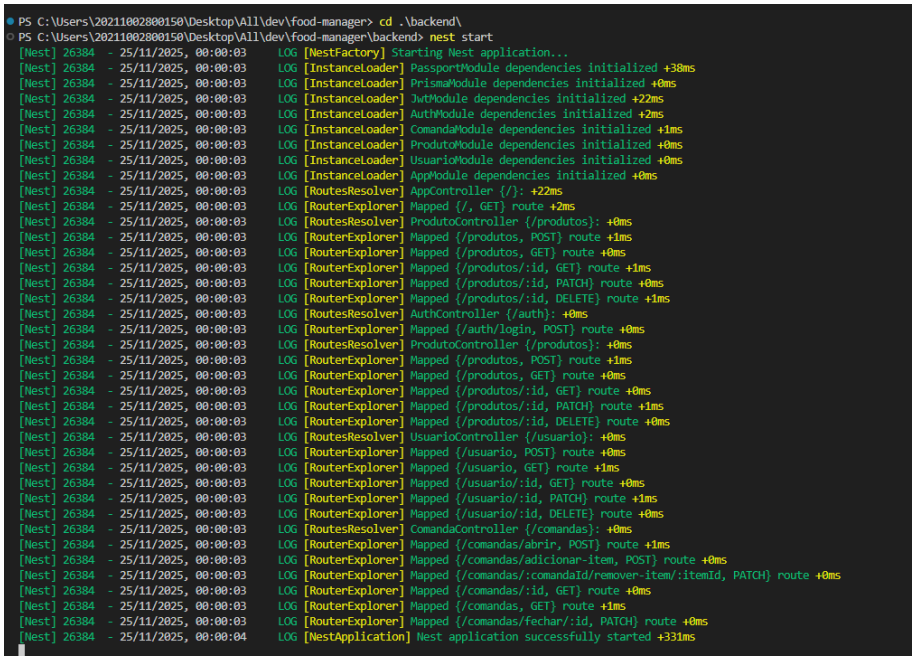


Fonte: O autor.

4.2.3 Funcionamento e Testes

Durante o desenvolvimento do back-end, antes de iniciar a parte do front, a realização de testes e verificação dos módulos, serviços e rotas foi essencial.

Figura 22 – Endpoints - Logs do Nest

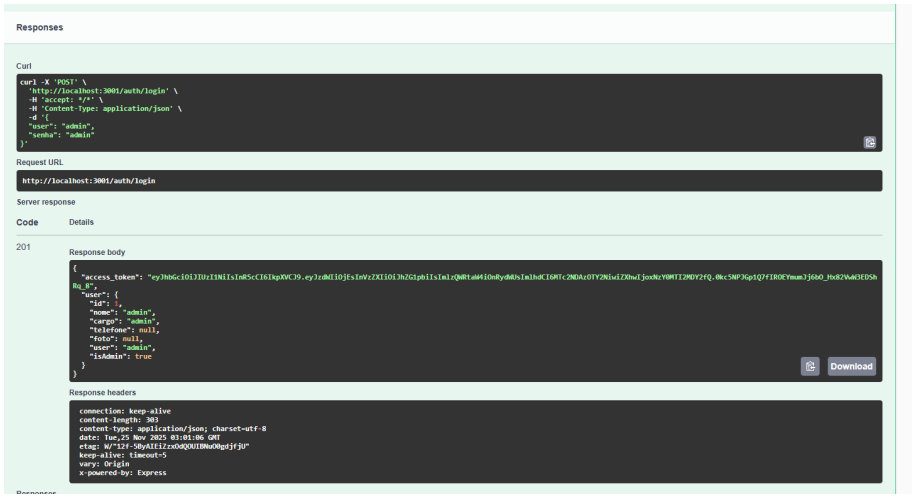


Fonte: O autor.

A primeira imagem mostra o terminal o terminal logo após a execução do comando nest start. Cada linha do log exibido pelo NestJS confirma o carregamento de dependências, inicialização de módulos e mapeamento das rotas HTTP definidas no projeto.

O NestJS, por seguir uma arquitetura modular, exibe no console o registro de cada controller. Isso inclui rotas referentes a produtos, usuários, autenticação e comandas.

Figura 23 – Exemplo de Requisição - /auth/login, POST



Fonte: O autor.

Após o servidor NestJS iniciar, o Swagger gera automaticamente uma interface interativa de documentação da API, permitindo testar cada rota sem a necessidade de ferramentas externas. O exemplo exibido refere-se à rota de autenticação (POST /auth/login), onde, ao enviar as credenciais, o sistema retorna um token JWT válido, acompanhado dos dados do usuário autenticado.

Em conjunto, as duas imagens mostram como o fluxo de desenvolvimento do backend foi feito, primeiro o servidor inicia validando as rotas e em seguida cada endpoint foi testado, garantido a respota enviada para o frontend seja correta.

4.2.4 Desenvolvimento Front-End

O desenvolvimento do frontend do FoodManager foi feito utilizando React, em conjunto com TypeScript e o ambiente otimizado do Vite. Essa combinação ajudou muito para que o projeto tenha uma interface rápida que com decorrer do desenvolvimento ficasse mais organizada e bonita que o protótipo inicial. Além disso, foi pensado em um design que fosse fácil de navegar para funcionários e administradores, mantendo uma diferença clara entre esses dois perfis, algo que já estava previsto nos diagramas de caso de uso iniciais.

Uma das primeiras preocupações foi a criação do fluxo de autenticação no lado do cliente. Foi utilizado um sistema chamado de ProtectedRoutes que garante que apenas usuários logados consigam acessar as telas internas. Essa função funciona tanto visualmente quanto internamente, mesmo que alguém tente acessar uma URL diretamente pelo navegador, a aplicação reconhece a ausência do token JWT e redireciona automaticamente para a tela de login.

Figura 24 – ProtectedRoutes.tsx

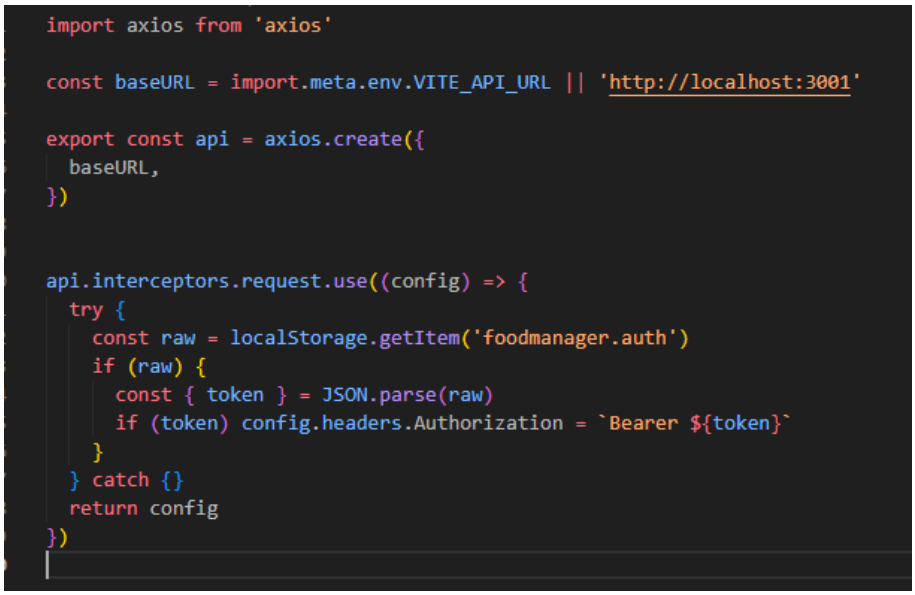
```tsx
import { type ReactNode } from 'react'
import { Navigate, useLocation } from 'react-router-dom'
import { useAuth } from '../modules/auth/AuthContext'

export function ProtectedRoute({ children }: { children: ReactNode }) {
  const { token } = useAuth()
  const location = useLocation()
  if (!token) return <Navigate to="/login" state={{ from: location }} replace />
  return <>{children}</>
}
```

Fonte: O autor.

Outra parte esssencial para o projeto foi a comunicação com o backend. Para isso

foi utilizado a biblioteca do Axios, que fica responsável por realizar todas as requisições HTTP para a API desenvolvida em NestJS. Cada cahamada passa por uma camada que coloca automaticamente o token JWT no cabeçalho.

Figura 25 – api.ts

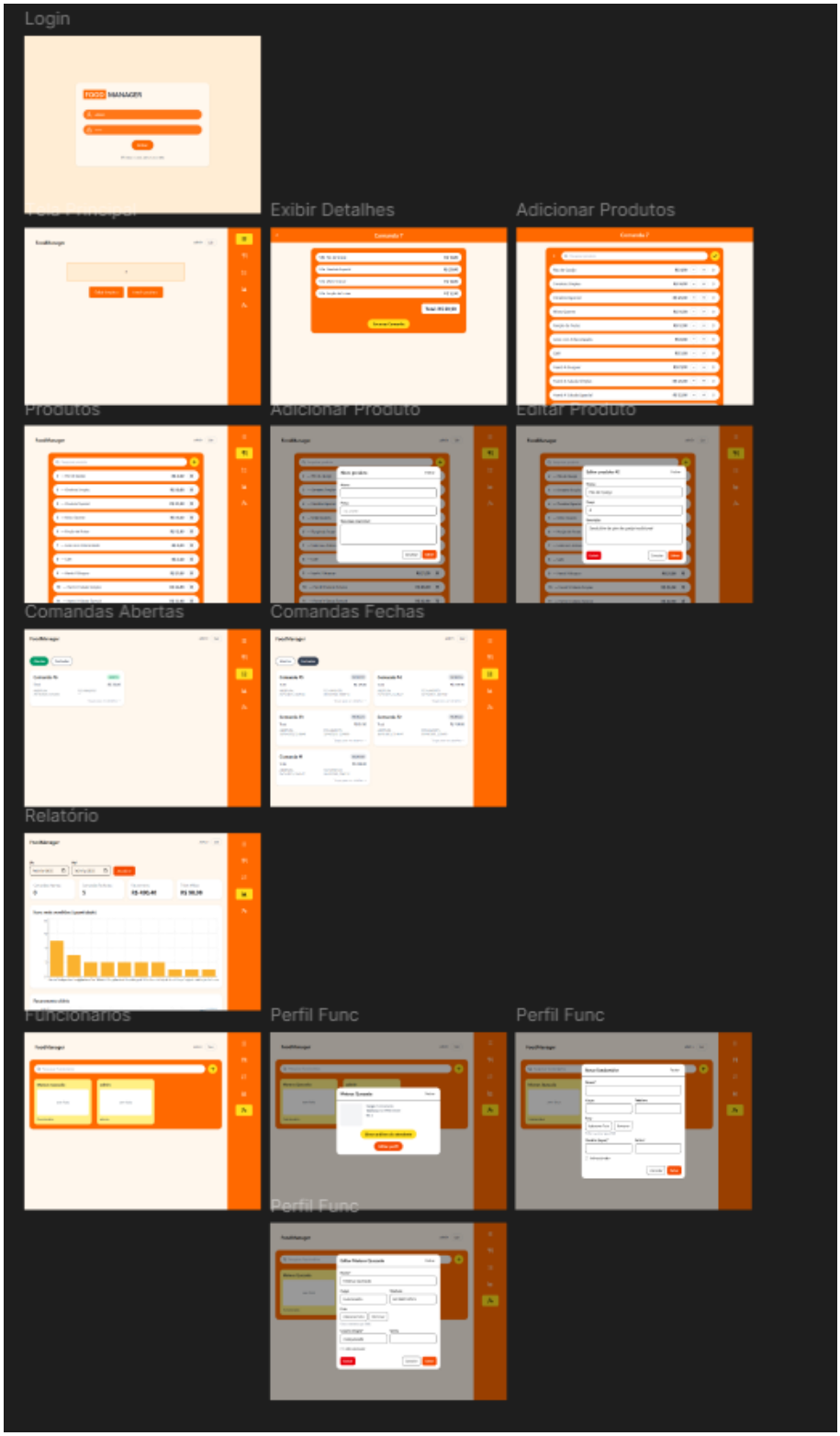


```
import axios from 'axios'

const baseURL = import.meta.env.VITE_API_URL || 'http://localhost:3001'

export const api = axios.create({
  baseURL,
})


api.interceptors.request.use((config) => {
  try {
    const raw = localStorage.getItem('foodmanager.auth')
    if (raw) {
      const { token } = JSON.parse(raw)
      if (token) config.headers.Authorization = `Bearer ${token}`
    }
  } catch {}
  return config
})
```

Fonte: O autor.

Naturalmente, muitas das telas pensadas no protótipo deixaram de existir e outras surgiram pois semente ficaram claras durante a implementação. Processo que mostra como o projeto amadureceu ao longo do desenvolvimento, resultando em uma interface mais bonita, intuitiva e organizada.

Figura 26 – Resultado final do Figma



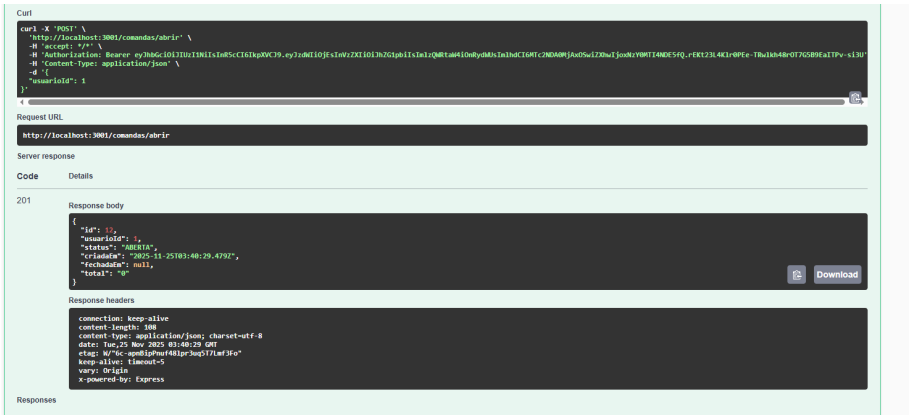
Fonte: O autor.

4.3 Principais funcionalidades

4.3.1 Abertura, gerenciamento e fechamento de comandas

A aplicação permite que o funcionário ou administrador abra uma nova comanda de forma rápida.

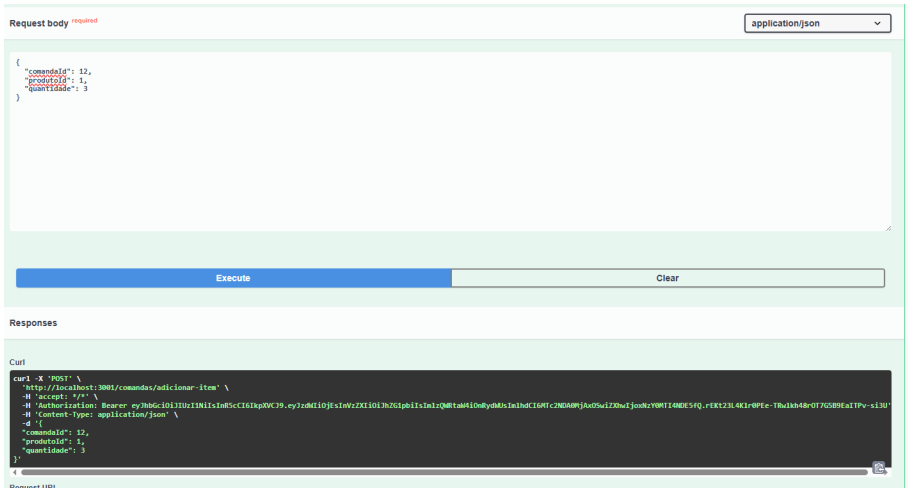
Figura 27 – Requisição - /comandas/abrir, POST



Fonte: O autor.

Uma vez criada, a comanda pode ser atualizada continuamente, recebendo novos itens conforme o atendimento evolui. O sistema mantém todas as informações sincronizadas com o banco de dados, garantindo que cada alteração seja registrada em tempo real.

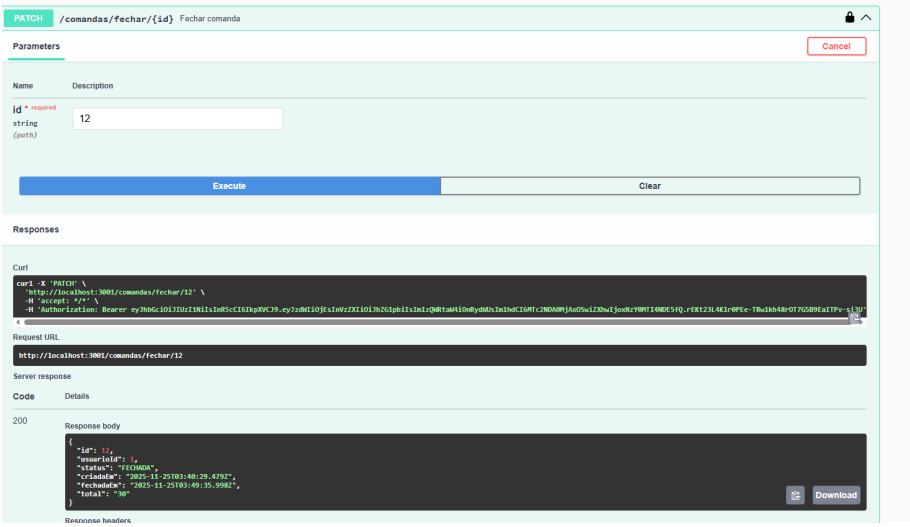
Figura 28 – Requisição - /comandas/adicionar-item, POST



Fonte: O autor.

O fechamento da comanda também é automatizado, ao encerrar, o sistema calcula o total com base nos itens registrados e registra o horário de encerramento.

Figura 29 – Requisição - /comandas/fechar/id, PATCH



Fonte: O autor.

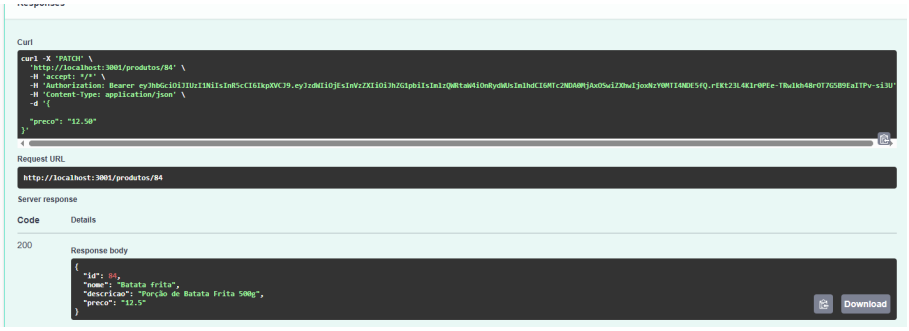
4.3.2 Gerenciamento de Produtos

O módulo de produtos foi desenvolvido especialmente para o administrador, que possui acesso completo às operações de criação, edição e exclusão de itens. O sistema exibe uma lista atualizada de todos os produtos, incluindo preço, nome e descrição, além de oferecer busca em tempo real para facilitar o gerenciamento.

Figura 30 – Requisição - /produto, POST

Fonte: O autor.

Figura 31 – Requisição - /produto/id, PATCH



Fonte: O autor.

4.3.3 Relatórios e acompanhamento financeiro

A aplicação fornece uma área dedicada a relatórios, onde o administrador pode acompanhar indicadores importantes de desempenho, como:

- número de comandas abertas e fechadas no período selecionado;
- itens mais vendidos por quantidade;
- faturamento total;
- ticket médio diário.

Essas informações são apresentadas por meio de gráficos e tabelas gerados diretamente no frontend.

4.4 Pesquisa de usabilidade

4.4.1 SUS - System Usability Score

A avaliação de usabilidade foi conduzida utilizando o System Usability Scale (SUS), um dos instrumentos para mensurar a experiência do usuário de forma rápida, objetiva e estatisticamente confiável.

Para esta pesquisa, foram coletados 14 formulários válidos que respondiam as seguintes perguntas:

- 1. Eu usaria o sistema FoodManger com frequência.
- 2. O sistema é desnecessariamente complexo.
- 3. O sistema foi fácil de usar.
- 4. Eu precisaria de ajuda técnica para conseguir usar o sistema.
- 5. As diversas funções do sistema estão bem integradas.
- 6. Há muita inconsistência no sistema.
- 7. Eu imagino que a maioria das pessoas aprenderia a usar o sistema rapidamente.
- 8. O sistema é muito confuso de se utilizar.
- 9. Eu me senti confiante ao usar o sistema.
- 10. Precisei aprender várias coisas novas antes de conseguir usar o sistema.

Usaremos uma resposta do formulário como exemplo para mostrar o cálculo:

Pergunta	**Resposta**
1	4
2	3
3	4
4	2
5	5
6	1
7	4
8	4
9	3
10	5

Tabela 4 – Respostas do Questionário SUS

- Para as perguntas ímpares (1, 3, 5, 7, 9), subtraímos 1 da resposta. - Para as perguntas pares (2, 4, 6, 8, 10), subtraímos a resposta de 5.

4.4.1.1 Pontuação das Perguntas Ímpares

Para as perguntas ímpares (1, 3, 5, 7, 9), a pontuação é dada por *(Resposta - 1)*. O cálculo é apresentado na tabela abaixo:

Pergunta	Resposta	Pontuação (Resposta - 1)
1	4	3
3	4	3
5	5	4
7	4	3
9	3	2

Tabela 5 – Pontuação das Perguntas Ímpares

4.4.1.2 Pontuação das Perguntas Pares

Para as perguntas pares (2, 4, 6, 8, 10), a pontuação é dada por *(5 - Resposta)*. O cálculo é apresentado na tabela abaixo:

Pergunta	Resposta	Pontuação (5 - Resposta)
2	3	2
4	2	3
6	1	4
8	4	1
10	5	0

Tabela 6 – Pontuação das Perguntas Pares

4.4.2 Somando as Pontuações

Agora, somamos as pontuações obtidas nas perguntas ímpares e pares:

Pergunta	Pontuação Final
1	3
2	2
3	3
4	3
5	4
6	4
7	3
8	1
9	2
10	0

Tabela 7 – Pontuação Final de Cada Pergunta

A soma das pontuações é:

$$3 + 2 + 3 + 3 + 4 + 4 + 3 + 1 + 2 + 0 = 25$$

Agora que temos a soma das pontuações (25), multiplicamos por 2,5 para calcular o SUS Score:

$$\text{SUS Score} = 25 \times 2,5 = 62,5$$

Portanto, o SUS Score para esta única resposta é 62.5.

4.4.3 Resultado Completo

Apesar de uma grande amplitude (25 a 100), foi possível observar um desempenho consistentemente elevado na maior parte das respostas, 11 dos 14 participantes atribuíram uma pontuação acima de 80, indicando uma percepção positiva bastante consolidada sobre a usabilidade do sistema. No final obteve-se uma média geral de 83,93 pontos, valor que é considerado alto e que se encaixa como "Excellent", indicando que a interface do FoodManger é amigável e de fácil uso para a maioria das pessoas que testaram (Bangor, Kortum e Miller, 2009).

A presença de um único valor extremamente baixo (25 pontos) merece atenção, mas não compromete o conjunto da análise, sendo interpretado como um outlier estatístico. Variações isoladas podem ocorrer devido a diferenças individuais de experiência, familiaridade com tecnologia ou até mesmo expectativas particulares sobre o sistema. (Bangor et al., 2009)

5 RESULTADOS ESPERADOS

5.1 Visão geral dos resultados

O desenvolvimento do FoodManager resultou em uma aplicação web funcional, coerente com os requisitos definidos e capaz de atender às principais rotinas operacionais de um restaurante. A aplicação implementada contempla todos os módulos essenciais para o funcionamento do estabelecimento: gerenciamento de produtos, cadastro de usuários, abertura e acompanhamento de comandas e controle básico de movimentações de caixa.

De forma geral, o FoodManager atingiu um nível de maturidade que o torna adequado para uso real em pequenos e médios estabelecimentos. Os resultados obtidos demonstram que o sistema melhora o fluxo de trabalho interno, reduz retrabalho, centraliza informações importantes para a gestão e oferece ao usuário final uma interface simples, limpa e eficiente.

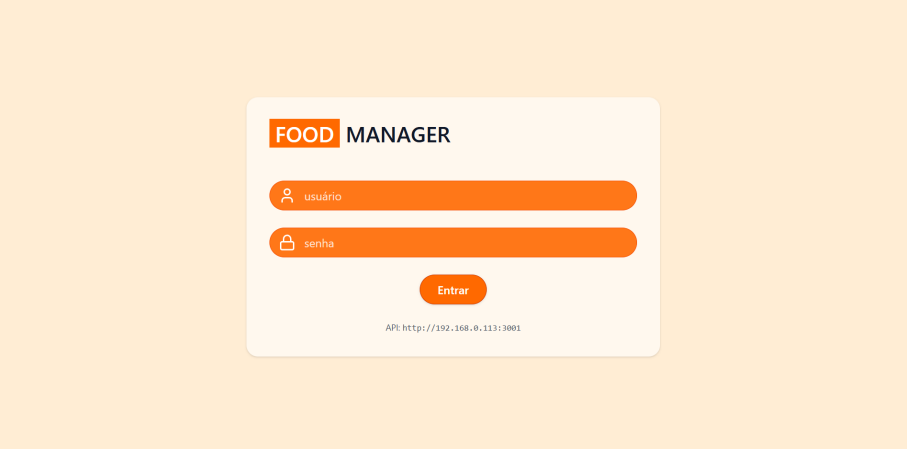
5.1.1 Principais Telas

5.1.1.1 Funcionário

5.1.1.1.1 Login

A tela de login representa o ponto de entrada do FoodManager. Nessa interface, o usuário informa seu nome de usuário e senha para acessar o sistema. A intenção foi manter o layout limpo e direto, já que essa é uma rotina que será executada diversas vezes ao longo do dia pelos funcionários do restaurante. O design utiliza cores quentes e contrastantes, reforçando a identidade visual definida durante o processo de prototipação no Figma.

Figura 32 – Tela de Login



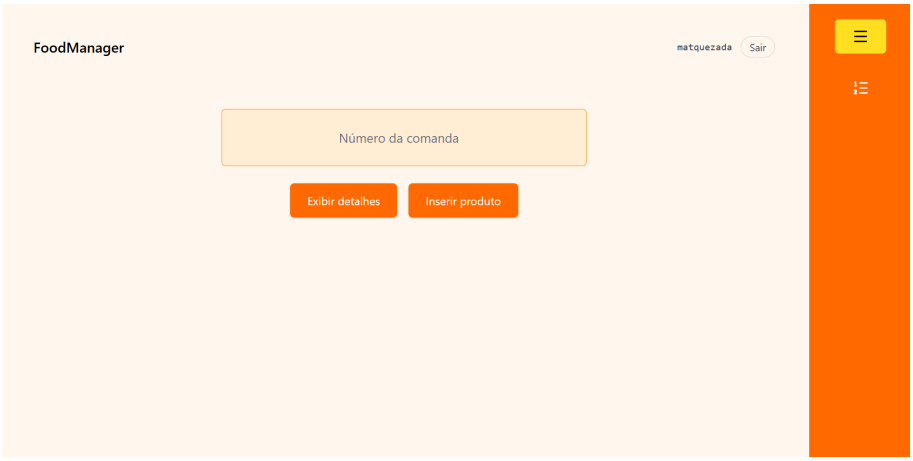
Fonte: O autor.

5.1.1.1.2 Principal

Depois do login, chegamos a tela principal, o usuário é direcionado para essa interface, onde deve informar o número da comanda que deseja manipular. A interface apresenta duas ações principais: Exibir detalhes e Inserir produto. A primeira permite visualizar imediatamente os pedidos já associados àquela comanda, enquanto a segunda conduz diretamente para a seleção de produtos. A proposta foi construir uma tela minimalista, assim acelerando o uso no dia a dia do restaurante.

No canto superior direito, o usuário tem acesso ao botão de “Sair”, que encerra a sessão e remove o token de autenticação, além de um menu lateral que complementa a navegação. Menu qual é composto por ícones simples, fazendo com que a pessoa que utilize a aplicação não tenha dificuldade em alternar entre as diferentes áreas do sistema.

Figura 33 – Tela Principal

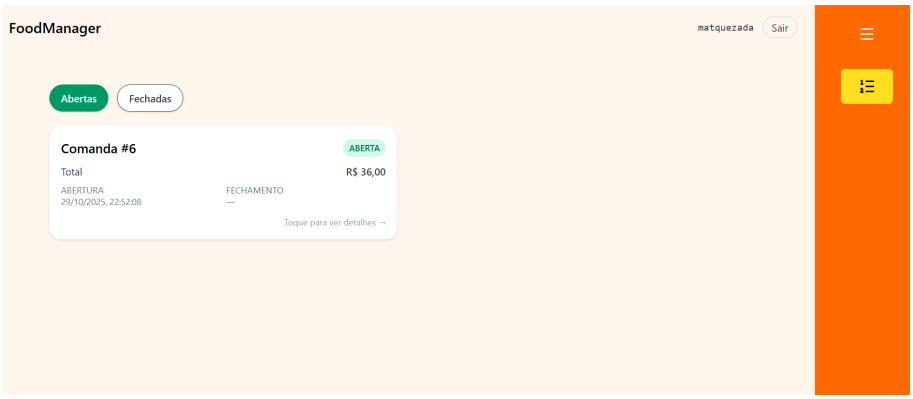


Fonte: O autor.

5.1.1.1.3 Comandas

A tela de listagem de comandas apresenta ao funcionário uma visão geral de todos os atendimentos em andamento no restaurante. Assim que o usuário acessa essa área do sistema, ele pode alternar entre duas categorias principais: comandas abertas e comandas fechadas. Todas as comandas são exibidas por um card individual que mostram um resumo das principais informações, horários de abertura e fechamento e saldo acumulado até o momento.

Figura 34 – Tela das Comandas



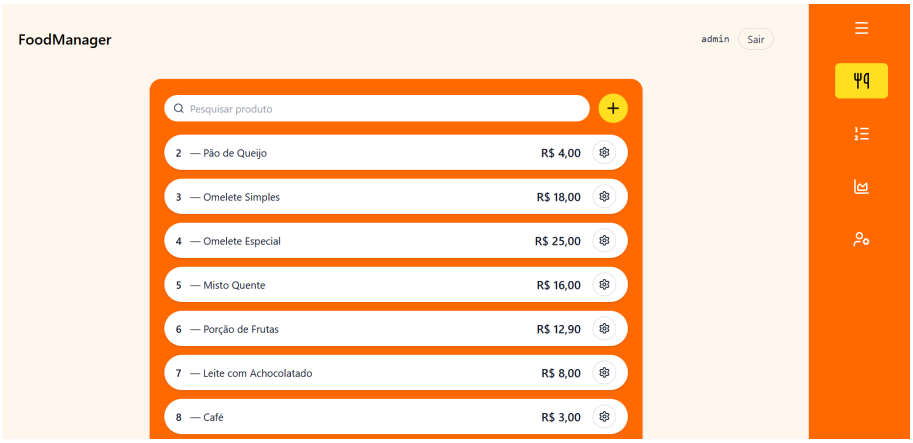
Fonte: O autor.

5.1.1.2 Admin

5.1.1.2.1 Produtos

A tela de gerenciamento de produtos é uma das funcionalidades exclusivas do administrador. Nessa interface, o administrador pode visualizar todos os itens cadastrados, incluindo todos seus atributos, indentificador, nome, preço e descrição. No topo da tela, há um campo de pesquisa que facilita a localização de produtos específicos, ao lado da barra de busca, encontra-se o botão de adição, que direciona para a tela de cadastro de novos produtos. No canto direito de cada card que mostra um produto, há um ícone de engrenagem que abre as opções de edição, permitindo a alteração de qualquer atributo daquele produto, menos o identificador.

Figura 35 – Tela dos Produtos



Fonte: O autor.

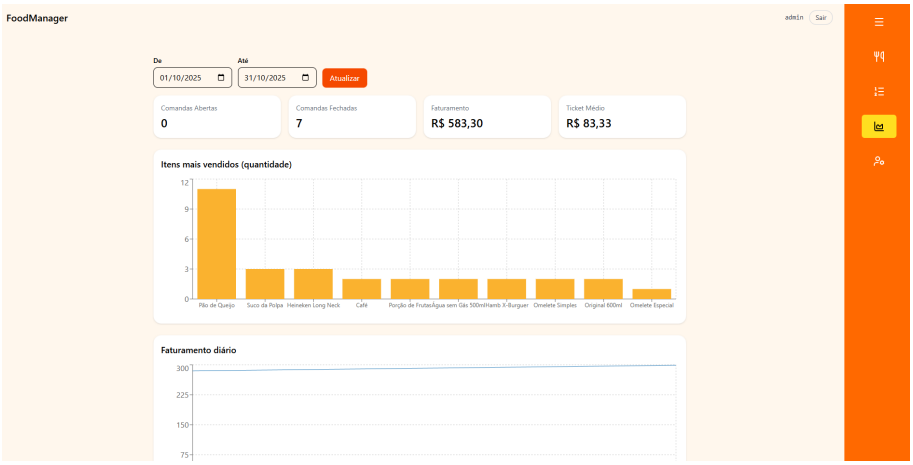
O menu lateral à direita destaca visualmente quais seções pertencem ao administrador, reforçando ainda mais a separação entre perfis.

5.1.1.2.2 Relatórios

A tela de relatórios concentra as informações analíticas do FoodManager. Logo no topo, o usuário pode selecionar um intervalo de datas para filtrar os resultados desejados, caso a analíse queira ser feita mais especificadamente. Logo abaixo pode ser observada o número de comandas abertas e fechadas, assim como outros indicadores essenciais para gestão como o faturamento total e o ticket médio (média de quanto um cliente gasta em uma ida ao restaurante).

A parte central da interface é composta por gráficos, que representam visualmente o comportamento das vendas. O primeiro gráfico mostra os itens mais vendidos, organizados em barras que destacam a quantidade de pedidos de cada produto. Logo abaixo, um gráfico adicional apresenta o faturamento diário, permitindo acompanhar a variação de receita ao longo do tempo.

Figura 36 – Tela dos Relatórios

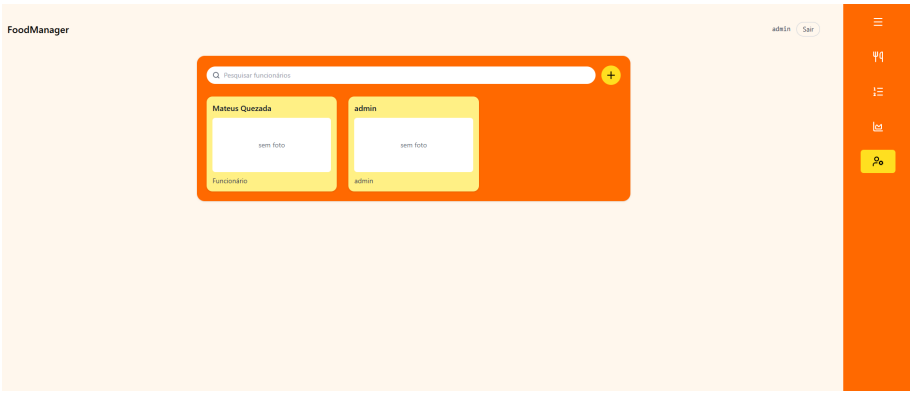


Fonte: O autor.

5.1.1.2.3 Funcionários

A tela de gerenciamento de funcionários foi projetada para facilitar o controle sobre os usuários cadastrados no sistema. Nessa interface, o usuário consegue visualizar todos os colaboradores registrados, cada um organizado em cartões individuais que exibem nome, foto (quando disponível) e o papel atribuído ao usuário. Na parte superior da tela, assim como na tela de produtos, possui um campo de pesquisa, facilitando a busca, como a de um ícone de mais, que abre a tela de cadastro de novos usuários.

Figura 37 – Tela dos Funcionários



Fonte: O autor.

Durante o desenvolvimento do FoodManager, a utilização do React foi essencial por conta da sua capacidade de componentização da interface. Muitas telas do sistema compartilham elementos em comum como barra de navegação, cartões de listagem, botões, campos de entrada e até pequenos componentes visuais usados repetidamente em diferentes contextos.

Com o React, foi possível transformar esses elementos recorrentes em componentes reutilizáveis, sem a necessidade de copiar e colar código, deixando a implementação mais coesa, além disso garantindo que qualquer ajuste visual ou funcional pudesse ser aplicado em um único ponto, refletindo automaticamente em todas as telas que utilizavam aquele componente, facilitando assim a manutenção.

5.1.2 Fluxograma de Criação e Manipulação de Pedidos

O fluxo de criação e gerenciamento de pedidos no FoodManager segue uma sequência simples e direta, pensada para garantir rapidez no atendimento e reduzir erros operacionais. A Figura 38 apresenta o fluxograma geral desse processo, desde o momento em que o usuário acessa o sistema até o encerramento de uma comanda.

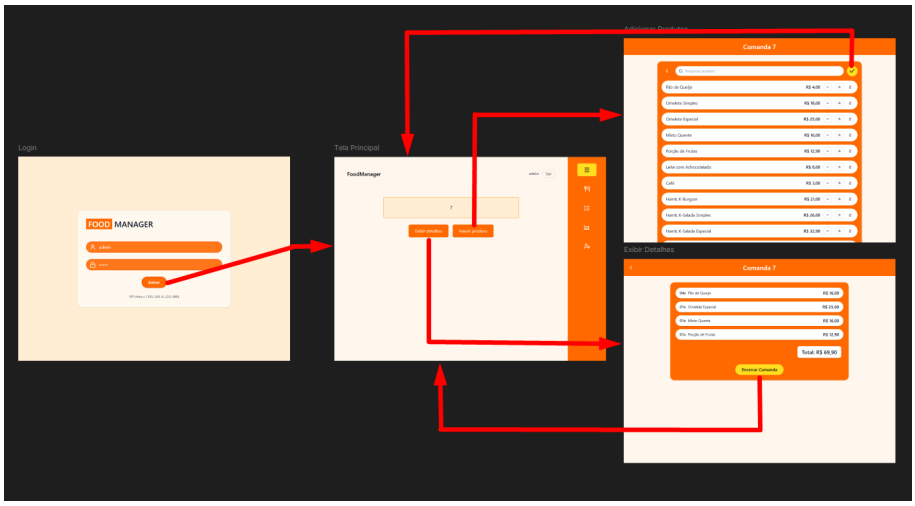
Tudo começa na tela de login, onde o funcionário realiza a autenticação. Após a validação das credenciais, o sistema o direciona para a tela principal, que funciona como ponto de partida para qualquer operação relacionada às comandas. Nela, o usuário informa o número da comanda desejada, que pode ser uma comanda já existente ou uma nova, que será aberta automaticamente na primeira interação.

A partir dessa tela, o funcionário possui duas opções: exibir detalhes da comanda ou inserir produtos. Ao escolher inserir produtos, o sistema direciona para a tela de adição de itens, onde é possível pesquisar produtos, visualizar valores e inserir ou remover itens rapidamente.

Ao selecionar "exibir detalhes", o usuário acessa um painel com o resumo da comanda: lista de itens registrados, quantidades, valores individuais e total parcial. A partir dali, é possível finalizar o pedido clicando em “Encerrar Comanda”, que registra o fechamento e calcula o valor final. Após a finalização, o fluxo retorna automaticamente à tela principal.

Esse fluxo foi pensado para ser direto e intuitivo, já que a operação de comandas costuma ser repetitiva ao longo do dia. A organização das telas e a lógica de transição garantem que o funcionário consiga navegar rapidamente entre as etapas.

Figura 38 – Fluxograma de Pedidos



Fonte: O autor.

5.1.3 Fluxograma de Criação e Manipulação de Prudutos

.O fluxo se inicia na tela de produtos, onde todos os itens cadastrados são exibidos, acompanhados de seus respectivos valores e IDs. Essa interface funciona como o ponto central de gerenciamento: nela, o administrador pode pesquisar itens específicos, visualizar o preço de cada produto e ter acesso imediato às ações de edição ou criação. A inclusão de um novo item ocorre por meio do botão de adição (“+”), localizado no canto superior direito da listagem. Ao selecioná-lo, o sistema abre uma janela modal destinada ao cadastro do

produto, permitindo informar nome, preço e, opcionalmente, uma descrição. Após preencher os campos necessários e salvar, o sistema atualiza automaticamente a lista principal.

Ao clicar no ícone de configurações presente ao lado de cada produto, o administrador acessa o modal de edição, onde é possível alterar nome, valor ou descrição do item. Essa janela também oferece a possibilidade de excluir o produto, caso ele não faça mais parte do cardápio. Após a confirmação das alterações, o sistema atualiza o item e retorna automaticamente à listagem de produtos, garantindo continuidade no fluxo de trabalho.

Figura 39 – Fluxograma de Produtos

Fonte: O autor.

5.1.4 Entregas Atuais

A aplicação já apresenta um conjunto consistente de funcionalidades que permitem a operação completa e integrada de um restaurante. Entre a principais, autenticação e controle de funções baseadas no nível do usuário, abertura e gerencialmente completo dos pedidos e comandas, sendo possível visualizar detalhes e o valor total de cada atendimento. Além disso, o cadastro e gerenciamento de produtos, sendo possível inserir e alterar as características de cada item. O controle e cadastro de funcionários esta presente, sendo possível adicionar novos usuários e também altera-los da forma que for necessária.

Comparando o FoodManager com o processo tradicional, baseado em registros

manuais e comunicação fragmentada entre salão e cozinha, nota-se ganhos significativos. A eliminação das comandas físicas reduz falhas de leitura, extravios e inconsistências, ao mesmo tempo em que acelera o envio das informações. O controle financeiro passa a ser mais preciso, já que o sistema contabiliza automaticamente cada operação realizada. A centralização das informações também facilita a organização interna, evitando duplicidade de dados e reduzindo a necessidade de conferências manuais. Com isso, o restaurante passa a operar com maior agilidade, menos erros e um acompanhamento mais claro das suas operações.

Considerações Finais

O desenvolvimento do FoodManager representou uma oportunidade concreta de aplicar, de forma integrada, conceitos teóricos e práticas adotadas na indústria de software.

O objetivo geral deste trabalho, construir um protótipo funcional para gerenciamento de pedidos em restaurantes foi plenamente alcançada, a melhora para os gerentes, chefes, garçons e clientes é evidente. A aplicação permite cadastrar produtos, abrir comandas, inserir itens, acompanhar valores em tempo real, encerrar atendimentos e acessar relatórios gerenciais.

A transmissão direta dos pedidos para a cozinha fez com que as comandas em papel não existissem mais , reduzindo erros operacionais e agilizando a comunicação entre os funcionários. A implementação de autenticação com perfis distintos (Administrador e Funcionário) garantiu maior controle de acesso, permitindo que ações administrativas realizadas apenas por usuários autorizados. Outrossim, a página de relatórios ofereceu aos administradores uma forma de acompanhar o fluxo de caixa, visualizar itens mais vendidos, analisar períodos específicos e faturamento, evidenciando que os objetivos específicos também foram alcançados.

Apesar dos resultados positivos, o sistema ainda pode melhorar. Como trabalho futuro, destaca-se a possibilidade de implementar funcionalidades como integração com sistemas de pagamento, dashboards mais completos, relacionados também aos funcionários, geração automática de notas e relatórios exportáveis, além de suporte a múltiplas unidades e personalização de cardápios. Além disso, seria interessante a realização de testes de campo, ou seja, com pessoas reais, permitindo a validação do sistema e experiência do usuário de maneira mais profunda.

No geral, o FoodManager mostrou-se um projeto tecnicamente consistente e viável, confirmando a importância de soluções digitais para otimizar processos no setor gastronômico. O sistema cumpre seus propósitos e estabelece uma base sólida para aplicações futuras mais robustas e completas.

REFERÊNCIAS

BANGOR, Aaron; KORTUM, Philip; MILLER, James. Determining what individual SUS scores mean: adding an adjective rating scale. **Journal of Usability Studies**, 2009.

BANKS, Alex. **Learning React: Modern Patterns for Developing React Apps**. O'Reilly Media, 2020.

BOSTON UNIVERSITY. **Technology trends in hospitality**. Boston University, 26 jan. 2023. Disponível em: https://www.bu.edu/hospitality/2023/01/26/technology-trends-in-hospitality/. Acesso em: 03 out. 2025.

CASTELLS, Manuel. **A sociedade em rede**. 10. ed. São Paulo: Paz e Terra, 2006. v. 1.

DATE, C. J. **An Introduction to Database Systems**. 8. ed. Boston: Addison-Wesley, 2004.

ECMA INTERNATIONAL. **ECMAScript Language Specification (ECMA-262)**. Disponível em: https://262.ecma-international.org/. Acesso em: nov. 2025.

FGV; ABRASEL. **Bares e Restaurantes no Brasil: Diagnóstico e Iniciativas**. Rio de Janeiro: Fundação Getulio Vargas, 2024.

FOWLER, Martin. **Patterns of Enterprise Application Architecture**. Addison-Wesley, 2019.

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

JUNGER, Alex Paubel et al. A geração imediatista e a comunicação audiovisual. **Research, Society and Development**, v. 9, n. 7, p. e24897441, 2020. Disponível em: https://rsdjournal.org/index.ph Acesso em: 13 set. 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.

PRISMA. **Documentation**. Disponível em: https://www.prisma.io/docs/. Acesso em: nov. 2025.

RICHARDSON, Leonard; RUBY, Sam. **RESTful Web APIs**. O'Reilly Media, 2013.

SEBRAE/DF. **Gastronomia: Painel de Inteligência Setorial**. Brasília: Serviço Brasileiro de Apoio às Micro e Pequenas Empresas – SEBRAE/DF, 2024.

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

TANENBAUM, Andrew S.; VAN STEEN, Maarten. **Distributed Systems: Principles and Paradigms**. 2. ed. Pearson, 2007.

THE NPD GROUP. **Restaurant carry-out and delivery digital orders soar during the pandemic; full-service restaurant digital orders jump by 237%**. PRWeb, 2020. Disponível em: https://www.prweb.com/releases/restaurant-carry-out-and-delivery-digital-orders-soar-during-the-pandemic-full-service-restaurant-digital-orders-jump-by-237–824018134.html. Acesso em: 09 set. 2025.

VITE. Documentation. Disponível em: https://vitejs.dev/guide/. Acesso em: nov. 2025.

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

WIERUCH, Robin. **The Road to Learn React**. Leanpub, 2018.

WORLD WIDE WEB CONSORTIUM (W3C). **Cascading Style Sheets (CSS) Snapshot 2023**. Disponível em: https://www.w3.org/TR/css-2023/. Acesso em: nov. 2025.