RESIGeT On-Line
Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica
Vol. 15 Nro. 1 2025
Uni-FACEF Centro Universitário Municipal de Franca

Soluções Tecnológicas para a Gestão de Bares e Restaurantes: A Integração de Sistemas e Tecnologias para Agilidade e Precisão

Breno Martins Oliveira
Graduando em Engenharia de Software – Uni-FACEF
24070@unifacef.edu.br

Fábio Medeiros Faria
Docente - Uni-FACEF
fabiomedeiros@facef.br

Resumo

O setor gastronômico enfrenta desafios significativos na gestão de operações, como controle de estoque e processamento de pedidos, que frequentemente resultam em falhas operacionais e queda na qualidade do serviço. Este trabalho apresenta o desenvolvimento do sistema NEXXOS, um sistema integrado de gestão para bares e restaurantes, com o objetivo de automatizar o controle de estoque, vendas, comandas e relatórios. A solução foi projetada com foco em usabilidade, velocidade e escalabilidade, utilizando uma arquitetura *FullStack* moderna com .NET (C#) para o *back-end*, *React* para o *front-end* e *PostgreSQL* para gerenciamento de dados. O sistema resultante busca otimizar os processos operacionais, reduzir custos e melhorar a experiência de clientes e funcionários.

Palavras-chave: gestão de restaurantes, sistemas de comanda, estoque, relatório, desenvolvimento de software.

Abstract

The gastronomic sector faces significant challenges in operations management, such as inventory control and order processing, which often result in operational failures and decreased service quality. This work presents the development of the NEXXOS system, an integrated management system for bars and restaurants, with the objective of automating inventory control, sales, tabs, and reports. The solution was designed with a focus on usability, speed, and scalability, using a modern FullStack architecture with .NET (C#) for the back-end, React for the front-end, and PostgreSQL for data management. The resulting system seeks to optimize operational processes, reduce costs, and improve the experience of both customers and employees.

Keywords: *restaurant management, ordering systems, inventory, reporting, software development.*

1 Introdução

O setor gastronômico vem passando por um processo acelerado de digitalização, impulsionado pela necessidade de aumentar a eficiência e a qualidade no atendimento. Entretanto, bares e restaurantes ainda enfrentam dificuldades na implementação de soluções tecnológicas eficazes, como interfaces pouco intuitivas, lentidão no processamento de pedidos e falhas no controle de estoque. Esses problemas impactam diretamente a experiência do cliente e a gestão financeira dos estabelecimentos.

Nesse cenário, sistemas de gestão apresentam-se como alternativa promissora, permitindo a automação de processos e a integração de diferentes áreas do negócio. A aplicação de tecnologias modernas possibilita não apenas maior agilidade no atendimento, mas também a geração de dados estratégicos para tomada de decisão.

Diante disso, este trabalho tem como objetivo geral o processo de desenvolvimento do NEXXOS, um sistema integrado de gestão para bares e restaurantes, capaz de automatizar o controle de estoque, pedidos, comandas e mesas, além de gerar relatórios. Entre os objetivos específicos, destacam-se:

- Oferecer um *dashboard* interativo para monitoramento de vendas e fluxo financeiro;
- Disponibilizar um aplicativo móvel, por meio da tecnologia de Aplicativos *Web Progressivos* (PWA – *Progressive Web App*), para registro ágil de pedidos;
- Validar a solução em cenários reais, garantindo usabilidade e eficiência.

A proposta visa contribuir para a modernização do setor, oferecendo uma plataforma escalável, intuitiva e acessível, que promova ganhos em produtividade, redução de custos e melhoria na experiência do cliente.

2 Referencial Teórico

A seguir, é apresentado o embasamento teórico que fundamenta este trabalho, abordando os principais conceitos de gestão, as tecnologias aplicáveis ao setor gastronômico e a importância da usabilidade e da experiência do usuário no desenvolvimento de sistemas.

2.1 Gestão de Bares e Restaurantes

A gestão de bares e restaurantes é uma área complexa que exige a combinação de múltiplos fatores, incluindo o controle de estoque, processamento de pedidos e gerenciamento de mesas. Estes fatores impactam diretamente a eficiência operacional e a experiência do cliente. A automatização dessas operações através de tecnologias digitais tem se mostrado uma solução eficaz, permitindo uma gestão integrada e mais fluída.

A transição de processos manuais para digitais otimiza o tempo e reduz erros humanos, resultando em maior satisfação do cliente e redução de custos operacionais. A implementação de sistemas integrados, por sua vez, permite uma visão panorâmica da operação, fundamental para a tomada de decisão estratégica e para mitigar as falhas identificadas no setor (VASCONCELOS, 2013).

2.2 Tecnologias de Gestão para Restaurantes e Bares

A implementação de sistemas de gestão em restaurantes e bares permite a integração de todas as operações em uma plataforma única. A adoção de Tecnologias da Informação (TI) é um fator de transformação na gestão, pois conecta níveis organizacionais, permite o acúmulo de experiências compartilhadas e transforma gestores em trabalhadores do conhecimento (SANTOS; GUIMARÃES JUNIOR, 2023).

Embora *sistemas de Point of Sale* (*POS*) e comandas móveis sejam comuns, a solução proposta neste artigo se diferencia pela arquitetura robusta e pelo foco em usabilidade, velocidade e escalabilidade, para atender de forma abrangente as necessidades específicas de gestão, incluindo controle de estoque em tempo real e relatórios gerenciais interativos.

2.3 Usabilidade e *Design* de Interfaces em Sistemas de Gestão

A usabilidade é um atributo de qualidade atrelado à facilidade de uso de um sistema. Conforme a literatura, a usabilidade se refere à presteza com que os usuários aprendem a utilizar uma determinada solução, estando intrinsecamente ligada à relação que se estabelece entre usuário, tarefa e interface (SANTANA, 2020).

Uma das leis primárias da usabilidade é "Não me faça pensar!", indicando que, durante a interação, o usuário deve entender claramente como agir e qual será

o resultado de sua ação (KRUG, 2008 *apud* SANTANA, 2020). Portanto, a interface precisa ser autoexplicativa e guiar o usuário de forma clara.

2.4 Sistemas de Comandas Móveis e a Experiência do Cliente

A implementação de comandas móveis tem se mostrado uma das principais inovações na gestão de bares e restaurantes. A utilização de dispositivos móveis para registro de pedidos em tempo real reduz significativamente o tempo de espera e melhora a precisão do serviço. A interação direta com o cliente por meio de aplicativos móveis também facilita a personalização do atendimento, permitindo que o cliente possa fazer pedidos diretamente da mesa, sem a necessidade de esperar por um garçom.

Além disso, o uso de comandas digitais ajuda a reduzir erros no processo de pedidos, uma vez que as informações são transmitidas diretamente para a cozinha/bar, sem intermediários.

3 Aplicações Tecnológicas no Setor Gastronômico

A evolução tecnológica impulsionou uma transformação significativa no setor gastronômico, tornando a digitalização uma necessidade competitiva. Esta seção explora as principais aplicações tecnológicas que moldam a gestão de bares e restaurantes, analisando tanto os benefícios da automação e da agilidade operacional quanto as barreiras que podem impedir a adoção. É dada ênfase especial à usabilidade e à experiência do usuário (UX), fatores determinantes para a eficácia e sucesso na implementação de qualquer sistema de gestão.

3.1 A Digitalização de Bares e Restaurantes

Nos últimos anos, o setor gastronômico tem se beneficiado de uma digitalização crescente, com a implementação de sistemas de gestão de pedidos, controle de estoque e pagamentos. A tecnologia oferece ferramentas para automatizar processos, reduzindo erros humanos e agilizando as operações diárias. Plataformas como POS (*Point of Sale*) e sistemas de comandas móveis são exemplos de inovações que otimizam a gestão desses estabelecimentos.

A transição de métodos tradicionais para soluções digitais permite não apenas o controle de pedidos em tempo real, mas também proporciona uma gestão mais eficaz do estoque e do fluxo de caixa. As plataformas digitais oferecem

vantagens como flexibilidade, agilidade e precisão, essenciais para aumentar a eficiência operacional.

3.2 Barreiras Tecnológicas e Culturais

A adoção de tecnologias avançadas como sistemas de gestão digital, Inteligência Artificial (IA) e integração com métodos de pagamento digitais enfrenta barreiras, especialmente em pequenos estabelecimentos. Muitos proprietários de restaurantes ainda resistem à mudança devido ao custo inicial, à complexidade da implementação e à falta de conhecimento sobre as vantagens de tais sistemas.

A resistência à adoção de novas tecnologias muitas vezes ocorre devido ao desconhecimento ou à falta de confiança nas soluções oferecidas. Investir em treinamento e capacitação de funcionários é essencial para garantir o sucesso na implementação e uso eficaz das ferramentas tecnológicas.

3.3 Questões de Usabilidade e Acessibilidade

Outro desafio é garantir que os sistemas de gestão digital sejam fáceis de usar, principalmente quando se trata de funcionários com diferentes níveis de habilidade tecnológica. Sistemas complexos e com interfaces difíceis de navegar podem aumentar o tempo de operação e causar frustração nos funcionários.

Portanto, a usabilidade deve ser uma prioridade na hora de desenvolver um sistema de gestão. A interface do usuário (UI) deve ser intuitiva, permitindo que os funcionários registrem pedidos, gerenciem mesas e verifiquem o estoque rapidamente, sem a necessidade de longos treinamentos.

3.4 *Design* e Experiência do Usuário (UX) no Sistema de Gestão

A experiência do usuário (UX) é um fator determinante no sucesso de sistemas de gestão em restaurantes. Sistemas que proporcionam uma experiência intuitiva e sem fricções têm maior taxa de adoção e satisfação dos usuários.

Portanto, o *design* da interface deve ser desenvolvido com foco na simplicidade e funcionalidade, permitindo que os usuários realizem suas tarefas de maneira rápida e eficiente. A interação do garçom ou do gerente com o sistema deve ser fluída, sem a necessidade de treinamento extensivo.

3.5 O Impacto de Interfaces Simples e Intuitivas

Interfaces simples e intuitivas em sistemas de gestão são cruciais para a eficiência operacional em bares e restaurantes. A facilidade de uso resulta em maior engajamento dos funcionários e satisfação dos clientes. Isso se traduz diretamente na redução de erros na anotação de pedidos e na agilidade do atendimento. Sistemas de fácil navegação e com clareza na interface minimizam a curva de aprendizado dos colaboradores, permitindo que a equipe se concentre primariamente na experiência do cliente, resultando em um serviço mais rápido, preciso e agradável.

4 Materiais e Métodos

O desenvolvimento deste artigo focou na criação de um sistema de gestão para bares e restaurantes, visando a otimização dos processos operacionais e a melhoria da experiência do cliente. O sistema foi desenvolvido utilizando tecnologias modernas que garantem uma interface dinâmica e fluída, integração eficiente e alto desempenho.

4.1 Materiais

A escolha dos materiais e ferramentas utilizadas na criação desta solução foi baseada nas necessidades do projeto e nas características desejadas para a plataforma, como escalabilidade, flexibilidade e facilidade de manutenção. A arquitetura *FullStack* foi adotada com tecnologias contemporâneas e robustas para a construção do sistema.

Abaixo, são apresentados os principais materiais adotados:

Back-end: C# com o *framework .NET*.

Front-end: React com *TypeScript*.

Banco de Dados: *PostgreSQL*.

O uso dessa combinação (C#/*.NET*, *React* e *PostgreSQL*) é justificado por ser uma tendência em sistemas que exigem eficácia na gestão, facilidade de manutenção e um alto grau de confiabilidade no armazenamento e recuperação de dados (OLIVEIRA et al., 2023).

4.1.1 *.NET* (C#)

O *back-end* do sistema foi desenvolvido utilizando o *framework* .NET com a linguagem C#. A escolha se deveu à robustez da plataforma, que oferece alta performance, segurança e ampla biblioteca de recursos para criação de APIs (*Application Programming Interface*). Além disso, a maturidade do ecossistema *.NET* garante escalabilidade e facilidade na manutenção do sistema, atendendo às demandas críticas de processamento de pedidos e controle de estoque.

4.1.2 *ReactJS* com *TypeScript*

No *front-end*, foi adotada a biblioteca *ReactJS* em conjunto com o *TypeScript*. Essa combinação permitiu a criação de interfaces dinâmicas, interativas e seguras, com tipagem estática que minimizou erros de desenvolvimento. A modularidade do *React* facilitou a reutilização de componentes e contribuiu para uma experiência de usuário fluída, essencial em sistemas em tempo real, como o gerenciamento de comandas e mesas.

4.1.3 PostgreSQL

O banco de dados escolhido foi o *PostgreSQL*, acessado por meio da interface *Neon*. A decisão se deu pela confiabilidade, robustez e conformidade do *PostgreSQL* como banco relacional, aliado à escalabilidade e praticidade que a *Neon* oferece em ambientes de nuvem. Essa arquitetura possibilitou a manipulação eficiente de dados estruturados, como comandas, pedidos e movimentações financeiras, garantindo consistência transacional e suporte a consultas complexas.

4.1.4 Trello

Para gerenciar o desenvolvimento do projeto, foi utilizada a ferramenta *Trello*. A plataforma, baseada na metodologia *Kanban*, facilitou o acompanhamento das tarefas e o progresso do projeto, com visualização clara das etapas concluídas, em andamento e pendentes. Isso contribuiu para a organização eficiente do trabalho e o cumprimento dos prazos.

4.2 Métodos

A execução do desenvolvimento seguiu uma metodologia ágil e iterativa, estruturada em *sprints* semanais. Essa abordagem permitiu entregas constantes e validações contínuas, garantindo que as funcionalidades fossem implementadas de acordo com as necessidades do cliente.

4.2.1 Levantamento de Requisitos

O levantamento de requisitos foi baseado em dois pilares: a) a análise de literatura sobre práticas de sucesso em gestão e b) a observação de falhas em experiências reais operacionais. Para a identificação dos problemas, considerou-se a experiência prática do autor, cuja família administra quatro empresas no ramo alimentício (OLIVEIRA, 2024).

Essa análise indicou que a complexidade de sistemas com funcionalidades excessivas e pouco intuitivas é um ponto de falha operacional comum. Desse modo, o objetivo foi focar nos requisitos mínimos essenciais e de alto impacto, eliminando funcionalidades de baixa relevância que pudessem comprometer a usabilidade e a agilidade da operação.

A gestão de insumos, por exemplo, pode ser otimizada por meio de ferramentas analíticas, como a Curva ABC, crucial para a saúde financeira do negócio (EVANGELISTA, 2020).

A partir dos resultados, foi possível definir os seguintes requisitos essenciais:

- Gestão de pedidos: capacidade de registrar e atualizar pedidos em tempo real.
- Controle de estoque: monitoramento do estoque em tempo real, com alertas de produtos com baixa quantidade.
- Fechamento de contas: funcionalidade para fechar contas com diferentes formas de pagamento.
- Gestão de comandas: criação e controle de comandas para mesas, balcões e entregas.

4.2.2 Modelagem e Arquitetura do Sistema

A modelagem do sistema foi realizada com o uso de diagramas *BPMN* (*Business Process Model and Notation*), UML e Entidade-Relacionamento (DER),

que ajudaram a mapear os fluxos de dados e as interações entre as diversas camadas do sistema. Esses diagramas estão apresentados na Seção 4.3. A arquitetura adotada foi *Client-Server*, com o *front-end* e *back-end* separados, comunicando-se via APIs *RESTful*.

4.2.3 Desenvolvimento

O desenvolvimento foi dividido em sprints semanais, cada uma focada em entregar uma funcionalidade específica, como cadastro de produtos, gerenciamento de comandas e controle de estoque. Durante o processo, foram adotadas boas práticas como:

- Versionamento com Git: controle do código-fonte utilizando GitHub.
- Revisões de código: a análise do código foi feita antes da integração de novas funcionalidades.

4.2.4 Validação e Testes

Após o desenvolvimento das funcionalidades, foi realizado um ciclo de testes com funcionários e gerentes de restaurantes, com o objetivo de validar a usabilidade e o desempenho do sistema. A avaliação focou na facilidade de uso, agilidade no atendimento e precisão dos pedidos. A partir dos feedbacks, foram feitas melhorias para garantir que o sistema atendesse às expectativas dos usuários.

4.2.5 Entrega e Documentação

Com a conclusão do levantamento de requisitos do projeto, foi construída a documentação, o que inclui: o código-fonte, diagramas de arquitetura, e manual do usuário, de modo a garantir que tanto os funcionários quanto os gestores pudessem utilizar o sistema de forma eficaz. A documentação também inclui instruções detalhadas para futuras atualizações e manutenção do sistema.

4.3 Diagramas

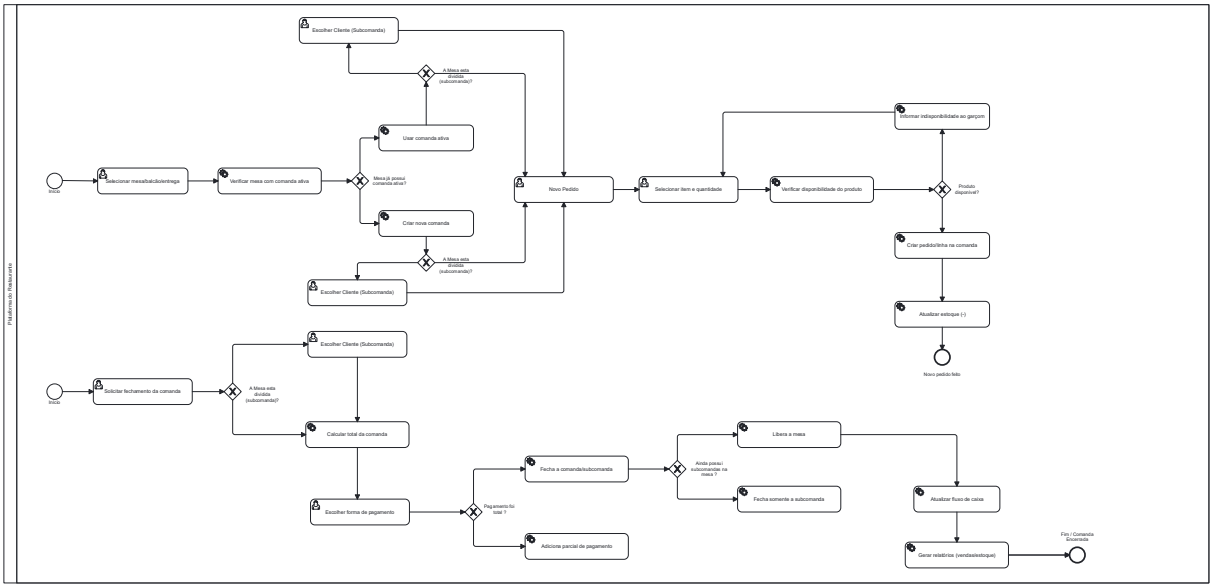
A utilização de diagramas foi fundamental para organizar o sistema. O BPMN representou os fluxos de atendimento e interação dos usuários, desde a abertura da comanda até o fechamento da conta. O Diagrama de Classes (UML) mapeou a estrutura estática do sistema, permitindo visualizar as principais entidades (*classes*),

seus atributos e as relações lógicas entre elas. Este modelo foi crucial para a organização da lógica de programação do *back-end* e para a definição das responsabilidades de cada componente de *software*. Já o DER detalhou a estrutura do banco de dados, contemplando tabelas e seus vínculos. Esses modelos apoiaram o alinhamento do desenvolvimento com os objetivos do projeto.

4.3.1 BPMN (Business Process Model and Notation)

O diagrama em BPMN (Business Process Model and Notation), apresentado na Figura 1, apresenta o fluxo de atividades do garçom e do sistema na plataforma de gestão, desde a abertura da comanda até o fechamento do pagamento. Inclui decisões como disponibilidade de produtos e forma de pagamento, permitindo visualizar os diferentes caminhos do atendimento. Essa modelagem torna mais clara a sequência de etapas e facilita o entendimento do funcionamento da plataforma.

Figura 1 - Diagrama de Processo (BPMN)



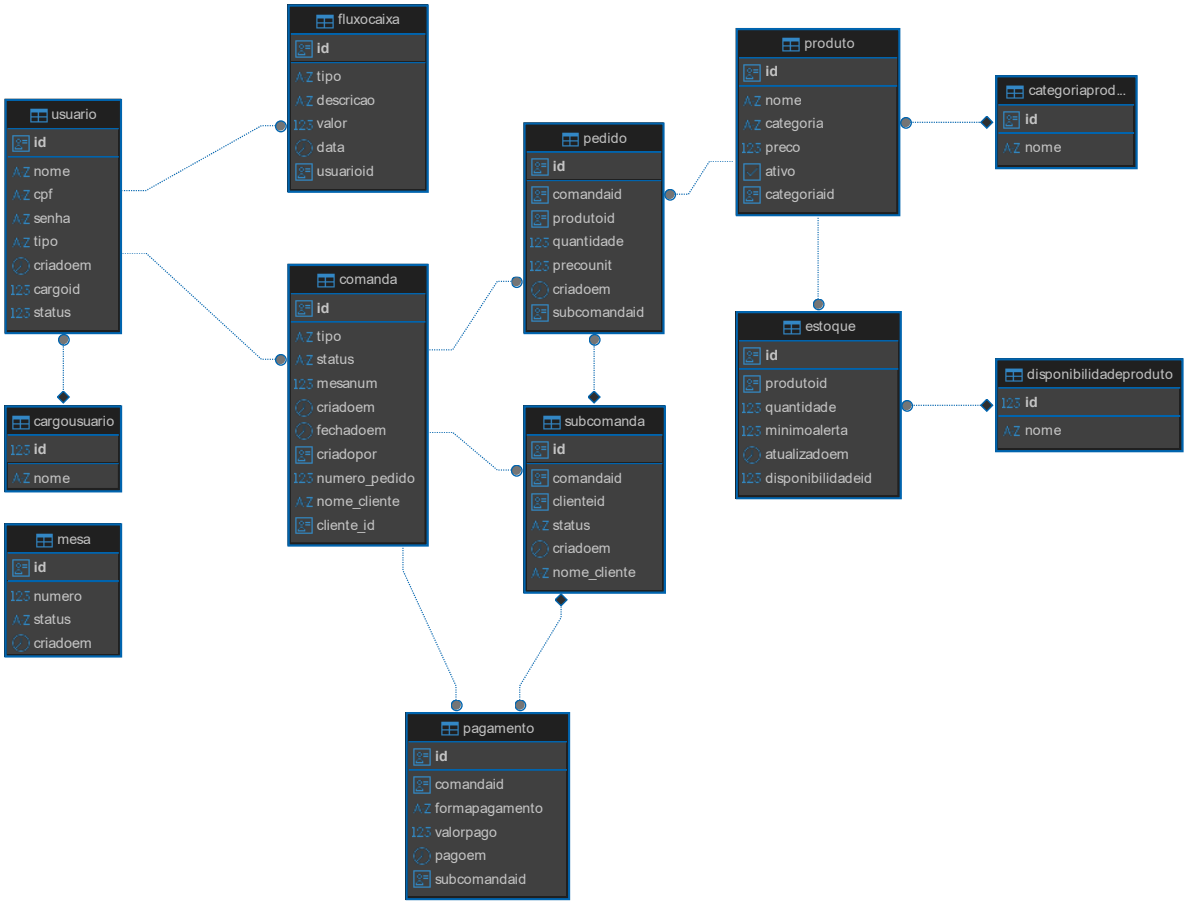
Fonte: O autor

4.3.2 Entidade-Relacionamento (DER)

O diagrama Entidade-Relacionamento (DER), apresentado na Figura 2, descreve a estrutura do banco de dados da plataforma de gestão para bares e restaurantes. Estão modeladas as principais entidades do sistema, como usuário, comanda, subcomanda, pedido, produto, estoque e pagamento, além de tabelas auxiliares, como mesa, categoria de produto e disponibilidade de produto. As relações foram definidas para refletir o fluxo real do atendimento: comandas

agrupam pedidos, que impactam diretamente o estoque; pagamentos estão vinculados a comandas e subcomandas; e usuários, conforme seu cargo, podem criar e gerenciar registros. Essa organização assegura integridade referencial e facilita consultas complexas, como análise de vendas, controle de estoque e acompanhamento do fluxo de caixa.

Figura 2 - Diagrama Entidade-Relacionamento (DER)

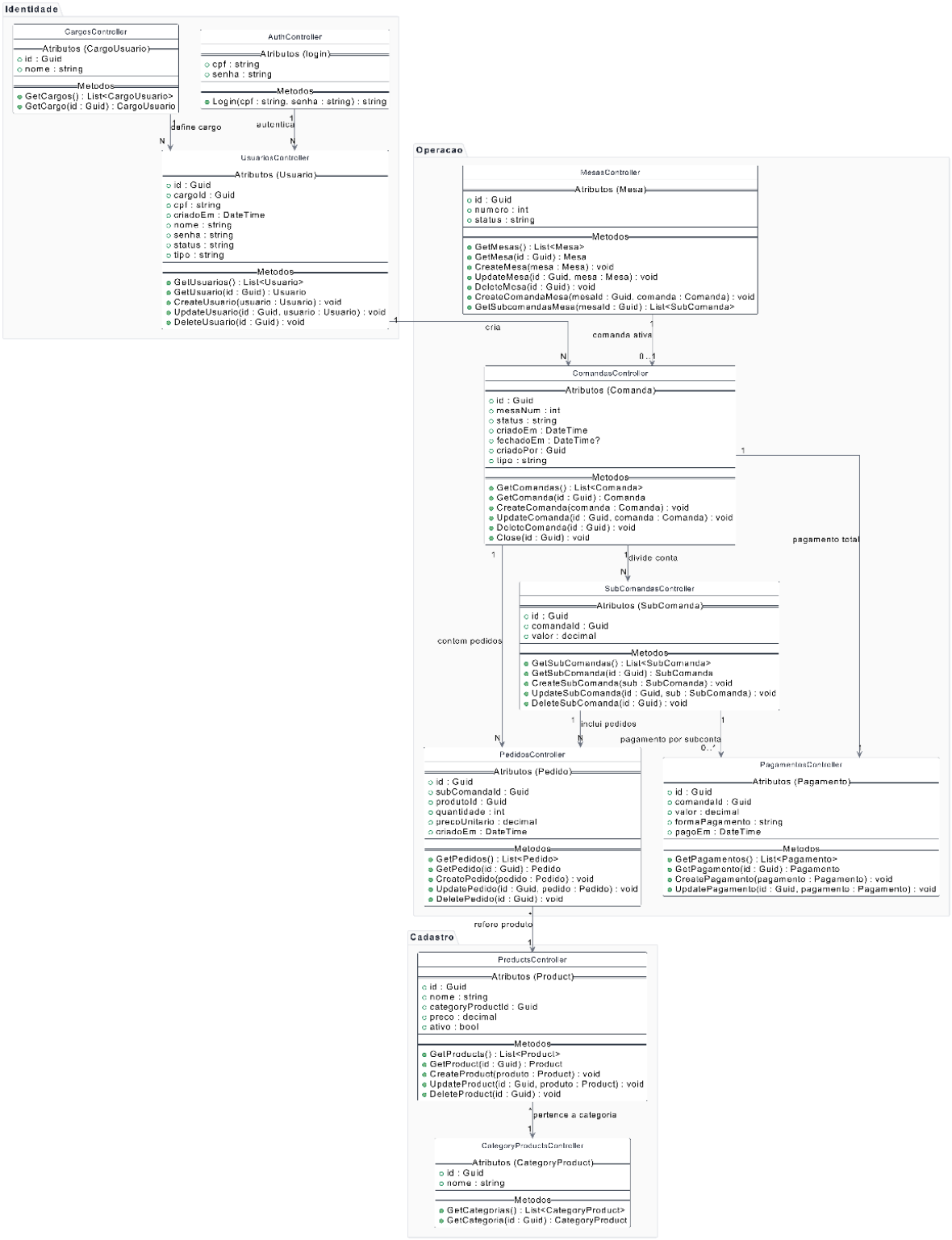


Fonte: O autor

4.3.3 Diagrama de Classes

O diagrama de classes, apresentado na Figura 3, organiza os controladores da aplicação, cada um responsável por uma entidade do sistema. O UserController gerencia usuários, o ComandaController administra as comandas, o PedidoController controla pedidos, o ProdutoController cuida do cardápio e o PagamentoController trata dos registros de pagamento. Essa divisão garante coesão, facilita a manutenção e reflete a lógica de negócio do restaurante.

Figura 3 - Diagrama de Classes



Fonte: O autor

4.4 Arquitetura do Sistema

A arquitetura do sistema foi estruturada no modelo Cliente-Servidor, organizada em três camadas principais: interface do usuário (*front-end*), API de

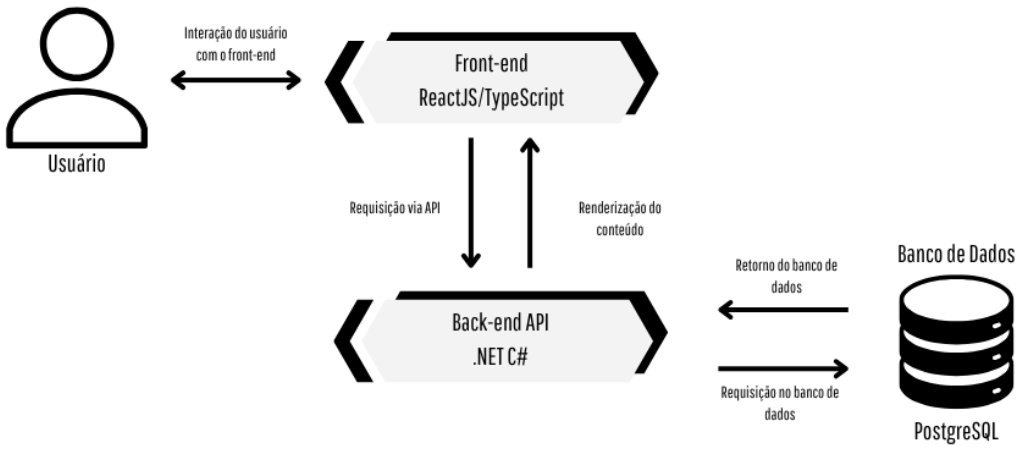
serviços (*back-end*) e banco de dados, representada na Figura 4. Essa divisão permite maior modularidade, escalabilidade e facilidade de manutenção.

O sistema realiza a comunicação entre o *front-end* e o *back-end* por meio de uma API, que funciona como uma interface de programação responsável por expor recursos do servidor para outros sistemas. No projeto, a API foi implementada em *.NET* (C#) e disponibilizada no padrão *REST* utilizando o protocolo HTTP (*Hypertext Transfer Protocol*), o que garante interoperabilidade e permite que qualquer tecnologia capaz de consumir requisições HTTP se conecte ao sistema. Essa abordagem reforça a flexibilidade da solução, viabilizando integrações futuras com outros serviços ou aplicações.

- *Front-end*: desenvolvido em *ReactJS* com *TypeScript*, é responsável pela interface com o usuário, oferecendo telas responsivas e intuitivas para atendentes e administradores.
- *Back-end* (API): implementado em .NET (C#), expõe serviços via APIs *RESTful*, tratando as regras de negócio, autenticação e comunicação com o banco de dados.
- Banco de Dados: o *PostgreSQL* garante consistência transacional, segurança e suporte a consultas complexas, armazenando informações de comandas, pedidos, produtos e relatórios.

Essa arquitetura garante a independência entre camadas, de forma que o *front-end* pode evoluir sem impactar o *back-end*, e novas integrações podem ser adicionadas sem comprometer a estrutura existente.

Figura 4 – Arquitetura do Sistema



Fonte: O autor

5 Desenvolvimento e Prototipação

O sistema foi desenvolvido com foco na integração entre comandas, pedidos, estoque e pagamentos, assegurando agilidade no atendimento e precisão no controle de dados. A prototipação das interfaces orientou a implementação, permitindo validar a usabilidade e o fluxo de uso ainda nas etapas iniciais. A arquitetura modular adotada organizou os controladores por entidades, facilitando manutenção e evolução do software. Além disso, os módulos foram conectados ao banco de dados em tempo real, garantindo consistência nas operações e suporte à geração de relatórios de vendas e estoque para os gestores.

5.1 Gerenciamento de Comandas e Pedidos

O ComandaController e o PedidoController foram responsáveis pelas operações CRUD (*Create, Read, Update* e *Delete*), que representam as ações de criação, leitura, atualização e exclusão de registros, das comandas e pedidos. Esses módulos permitem ao garçom abrir uma comanda vinculada a uma mesa, registrar produtos solicitados e acompanhar o consumo em tempo real. Cada pedido realizado atualiza automaticamente o estoque, garantindo precisão no controle de insumos. Na Figura 5 está apresentado um trecho do código do ComandaController, responsável pelas operações CRUD relacionadas às comandas, ilustrando a lógica utilizada para criação e gerenciamento dessas entidades.

Figura 5 - ComandaController

```
[HttpGet("{id}")]
public async Task<ActionResult<Comanda>> GetComanda(Guid id)
{
    var comanda = await _context.Comanda
        .Include(c => c.pedidos)
        .Include(c => c.pagamentos)
        .Include(c => c.subcomandas.Where(s => s.status != "Fechada"))
        .FirstOrDefaultAsync(c => c.id == id);
    if (comanda == null)
    {
        return NotFound();
    }
    return comanda;
}

[HttpPost]
public async Task<ActionResult<Comanda>> CreateComanda(Comanda comanda)
{
    var userId = HttpContext.GetUserId();
    if (userId is null) return Unauthorized();
    comanda.criadopor = userId.Value;

    if ((comanda.tipo.Equals("Balcao", StringComparison.OrdinalIgnoreCase) ||
         comanda.tipo.Equals("Entrega", StringComparison.OrdinalIgnoreCase)) &&
        string.IsNullOrWhiteSpace(comanda.nome_cliente))
    {
        return BadRequest("Nome do cliente é obrigatório para balcão ou entrega");
    }

    if (comanda.mesanum.HasValue)
    {
        var mesa = await _context.Mesas.FirstOrDefaultAsync(m => m.numero == comanda.mesanum.Value);
        if (mesa != null)
        {
            mesa.status = "Ocupada";
        }
    }

    _context.Comanda.Add(comanda);
    await _context.SaveChangesAsync();
    return CreatedAtAction(nameof(GetComanda), new { id = comanda.id }, comanda);
}
```

Fonte: O autor

5.2 Controle de Estoque em Tempo Real

O StockService foi implementado para monitorar a disponibilidade de produtos e integrar-se diretamente ao registro de pedidos. Esse módulo previne inconsistências, possibilita uma gestão antecipada dos insumos e impede a criação de pedidos quando o item não está disponível, apoiando a tomada de decisão sobre reposições e garantindo maior confiabilidade no atendimento. Na Figura 6 está apresentada a funcionalidade de verificação da disponibilidade de produtos, enquanto na Figura 7, o processo de atualização do estoque após o registro dos pedidos, assegurando que o sistema reflita em tempo real o consumo dos itens.

Figura 6 - StockService: Verificação de Estoque

```
public async Task<StockCheckResult> CheckStockAsync(IEnumerable<StockItemRequest> items, CancellationToken cancellationToken = default)
{
    var groupedItems = items
        ?.Where(i => i.Quantity > 0)
        .GroupBy(i => i.ProductId)
        .Select(g => new StockItemRequest(g.Key, g.Sum(x => x.Quantity)))
        .ToList() ?? [];

    if (groupedItems.Count == 0)
    {
        return new StockCheckResult(true);
    }

    var productIds = groupedItems.Select(g => g.ProductId).ToList();

    var stocks = await _context.Stocks
        .AsNoTracking()
        .Where(s => productIds.Contains(s.produtoid))
        .ToDictionaryAsync(s => s.produtoid, cancellationToken);

    var products = await _context.Products
        .AsNoTracking()
        .Where(p => productIds.Contains(p.id))
        .ToDictionaryAsync(p => p.id, cancellationToken);

    foreach (var item in groupedItems)
    {
        stocks.TryGetValue(item.ProductId, out var stock);
        var available = stock?.quantidade ?? 0;
        if (available < item.Quantity)
        {
            var missing = item.Quantity - available;
            var productName = products.TryGetValue(item.ProductId, out var product)
                ? product.Name
                : item.ProductId.ToString();
            var unitLabel = missing == 1 ? "unidade" : "unidades";
            var message = $"Produto \"{productName}\" sem estoque suficiente. Faltam {missing} {unitLabel}.";
            return new StockCheckResult(false, message);
        }
    }

    return new StockCheckResult(true);
}
```

Fonte: O autor

Figura 7 - StockService: Atualização de Estoque

```
public async Task DecreaseStockAsync(IEnumerable<StockItemRequest> items, CancellationToken cancellationToken = default)
{
    var groupedItems = items
        ?.Where(i => i.Quantity > 0)
        .GroupBy(i => i.ProductId)
        .Select(g => new StockItemRequest(g.Key, g.Sum(x => x.Quantity)))
        .ToList() ?? [];

    if (groupedItems.Count == 0)
    {
        return;
    }

    var productIds = groupedItems.Select(g => g.ProductId).ToList();

    var stocks = await _context.Stocks
        .Where(s => productIds.Contains(s.produtoid))
        .ToDictionaryAsync(s => s.produtoid, cancellationToken);

    foreach (var item in groupedItems)
    {
        if (!stocks.TryGetValue(item.ProductId, out var stock))
        {
            continue;
        }

        stock.quantidade = Math.Max(0, stock.quantidade - item.Quantity);
        stock.atualizadoem = DateTime.UtcNow;
        stock.disponibilidadeid = await ResolveAvailabilityIdAsync(stock.quantidade, stock.minimoalerta, cancellationToken);
    }

    await _context.SaveChangesAsync(cancellationToken);
}
```

Fonte: O autor

5.3 Segurança e Autenticação de Usuários

A segurança dos dados foi assegurada com autenticação baseada em JWT (*JSON Web Token*) para gerar um token de acesso seguro após o *login*, permitindo que o usuário navegue pela plataforma sem precisar se autenticar a cada página.

Essa implementação garante que apenas usuários autorizados (atendentes e administradores) acessem funcionalidades específicas, preservando a integridade do sistema.

5.4 Relatórios e *Dashboard*

O módulo de Relatórios e *Dashboard* é o produto final da coleta e processamento de dados. Este módulo visa consolidar informações de vendas, consumo de produtos e fluxo financeiro. Ao integrar esses dados, o sistema apoia decisões estratégicas, como o controle de custos e a precificação dos itens do menu, elementos diretamente ligados ao sucesso financeiro do estabelecimento (RIBEIRO, 2024). Os dados são exibidos em *dashboards* interativos, permitindo ao gestor acompanhar o desempenho do estabelecimento e identificar tendências de consumo. Na Figura 8 está apresentado o *DashboardController*, que consolida informações de vendas, consumo de produtos e fluxo financeiro, disponibilizando relatórios em *dashboards* interativos para apoiar a gestão.

5.5 Prototipação

A prototipação foi uma etapa crucial para validar a usabilidade e o fluxo de uso da plataforma, garantindo que as interfaces fossem simples, diretas e intuitivas. As telas a seguir apresentam o resultado da prototipação, demonstrando as principais funcionalidades e o modo de interação do atendente e do administrador com o sistema.

5.5.1 Login

Na Figura 9 está mostrada a tela de *Login* do sistema, projetada de forma simples e direta para garantir um acesso rápido e seguro aos diferentes perfis de usuário.

Figura 8 - *DashboardController*

```
[HttpGet("sales/last-day")]
public async Task<ActionResult<LastDaySalesResponse>> GetLastDaySalesAsync()
{
    var now = DateTime.UtcNow;
    var start = now.AddDays(-1);
    var previousStart = start.AddDays(-1);

    var lastDayQuery = _context.Comanda
        .Where(c => ClosedStatuses.Contains(c.status) && c.fechadoem != null)
        .Where(c => c.fechadoem >= start && c.fechadoem < now);

    var lastDaySales = await lastDayQuery.CountAsync();

    var lastDayPayments = await _context.Pagamentos
        .Where(p => p.pagoem >= start && p.pagoem < now)
        .Where(p => lastDayQuery.Select(c => c.id).Contains(p.comandaid))
        .SumAsync(p => (decimal?)p.valorpago) ?? 0m;

    var previousQuery = _context.Comanda
        .Where(c => ClosedStatuses.Contains(c.status) && c.fechadoem != null)
        .Where(c => c.fechadoem >= previousStart && c.fechadoem < start);

    var previousSales = await previousQuery.CountAsync();

    var previousPayments = await _context.Pagamentos
        .Where(p => p.pagoem >= previousStart && p.pagoem < start)
        .Where(p => previousQuery.Select(c => c.id).Contains(p.comandaid))
        .SumAsync(p => (decimal?)p.valorpago) ?? 0m;

    var salesChange = CalculatePercentageChange(lastDaySales, previousSales);
    var valueChange = CalculatePercentageChange(lastDayPayments, previousPayments);

    return new LastDaySalesResponse(lastDaySales, lastDayPayments, salesChange, valueChange);
}
```

Fonte: O autor

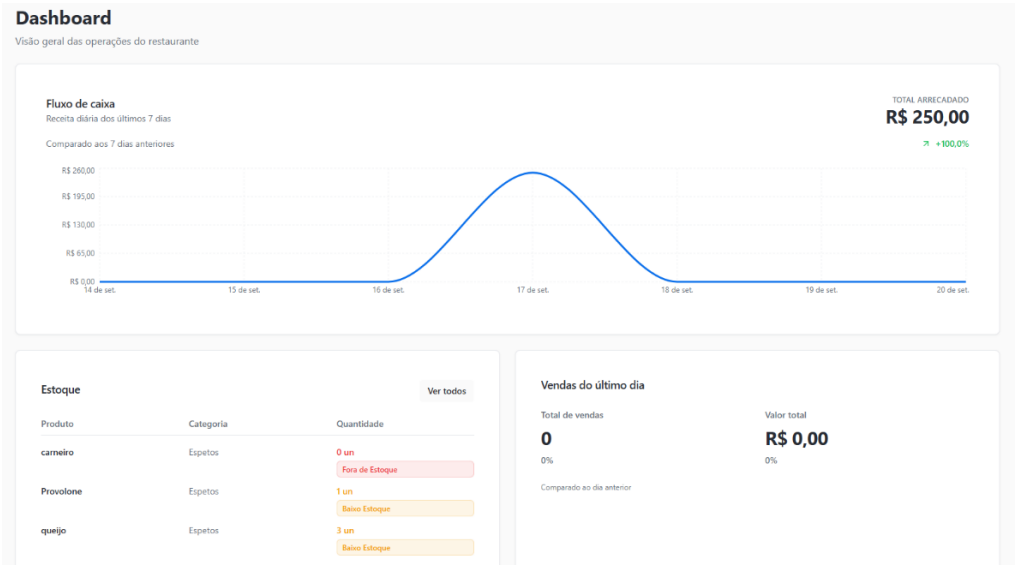
Figura 9 – *Login*

Fonte: O autor.

5.5.2 Dashboard

Na Figura 10 está apresentado o *Dashboard* principal, que centraliza indicadores-chave de desempenho e oferece uma visão em tempo real das operações do estabelecimento.

Figura 10 – *Dashboard*

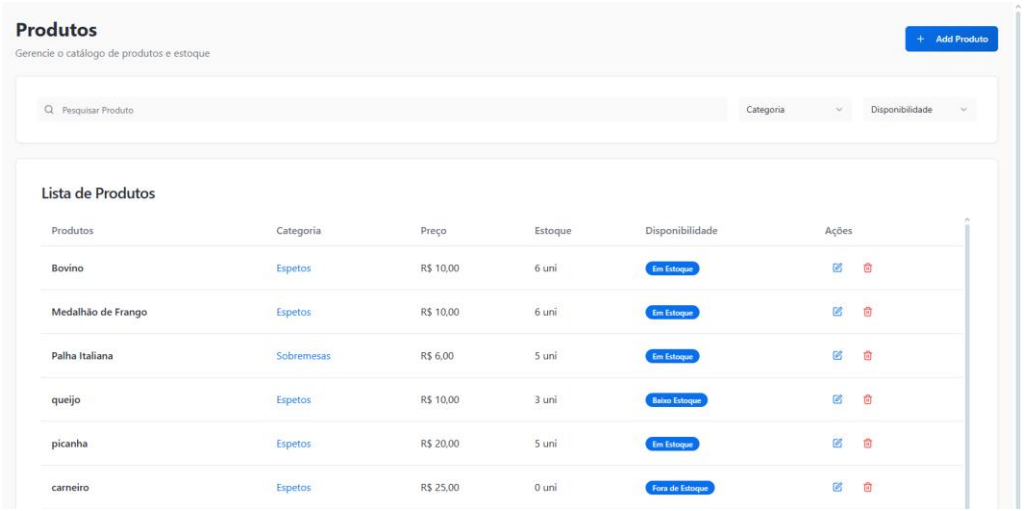


Fonte: O autor.

5.5.3 Gerenciamento de Produtos

Na Figura 11 está exibido a tela de gerenciamento de produtos, permitindo cadastrar, editar e monitorar itens do cardápio, com destaque para o controle de estoque em tempo real.

Figura 11 – Produtos



Fonte: O autor.

5.5.4 Comandas

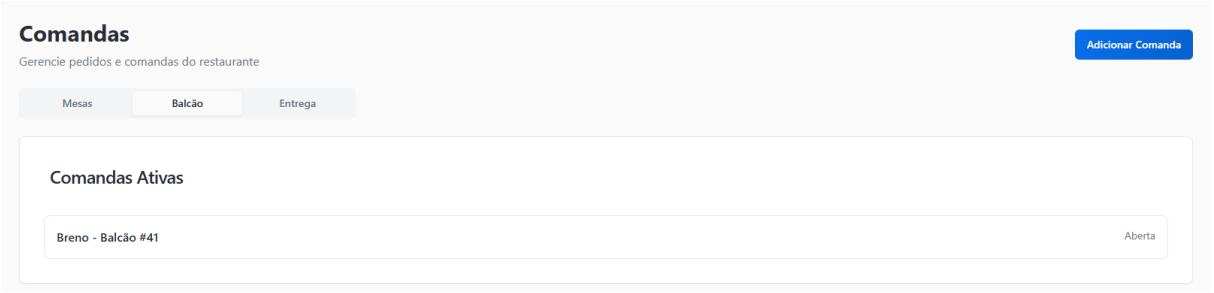
Nas Figuras 12 e 13 está apresentadas as telas de comandas, que permitem gerenciar pedidos realizados em mesas e no balcão. Ambas oferecem visão

detalhada do consumo em tempo real, facilitando o controle do atendimento e a finalização dos pedidos.

Figura 12 – Mesas

Fonte: O autor.

Figura 13 – Balcão

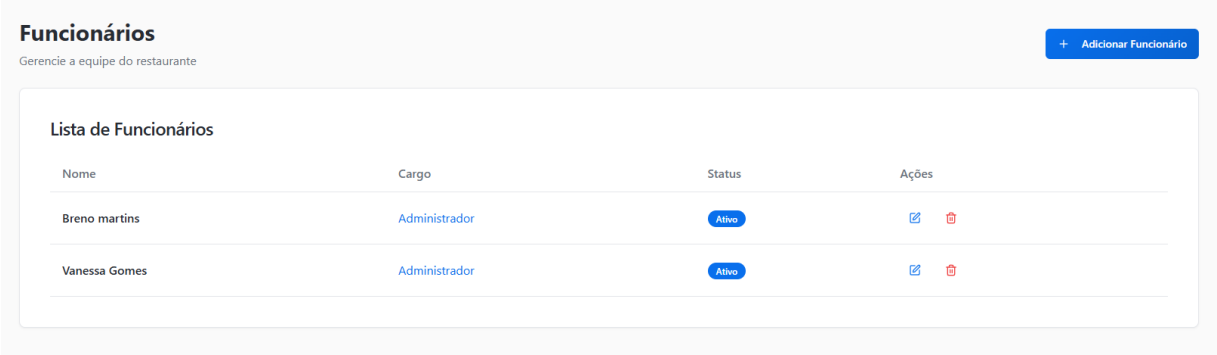


Fonte: O autor.

5.5.5 Usuários

Na Figura 14 está exibido a tela de gerenciamento de usuários, permitindo ao administrador controlar o fluxo de atendimento e os acessos de cada perfil do sistema.

Figura 14 – Usuários



Fonte: O autor.

6 Conclusão e Trabalhos Futuros

O desenvolvimento do sistema NEXXOS alcançou seu objetivo principal ao otimizar processos operacionais e oferecer uma solução eficiente para o controle de comandas, pedidos, estoque e pagamentos. A robustez e a eficácia da aplicação foram verificadas por meio da coleta de feedback da família do autor, que administra quatro empresas no setor alimentício. Para essa validação, o sistema foi submetido a um *deploy* e testado em cenários de uso real durante o atendimento. As sugestões coletadas, baseadas na experiência prática com outros sistemas, direcionaram as melhorias finais, garantindo que o NEXXOS atendesse às demandas reais do setor. A aplicação resultante mostrou-se robusta e funcional, garantindo agilidade no atendimento, redução de erros manuais e maior precisão no acompanhamento de insumos.

A solução NEXXOS se destaca na parte de integração em tempo real, a integração automática do registro de pedidos com a baixa no estoque em tempo real previne inconsistências e possibilita uma gestão antecipada de insumos. Usabilidade superior, a interface foi desenvolvida com um design altamente intuitivo, visando a uma curva de aprendizado mínima para os garçons. Esse foco na usabilidade visa melhorar os processos comparados a softwares legados, principalmente na agilidade do registro de pedidos. Arquitetura robusta, a arquitetura modular baseada em .NET (C#) e *ReactJS/TypeScript* assegura escalabilidade e desempenho, com uma API *RESTful* que reforça a flexibilidade e viabiliza integrações futuras.

A implementação de módulos específicos — como comandas digitais, integração automática com o estoque e registro de pagamentos — contribuiu para um fluxo de trabalho mais organizado e transparente. Além disso, o uso de uma arquitetura modular favorece a escalabilidade do sistema, permitindo que novos recursos sejam incorporados sem comprometer a estabilidade já existente.

Ressalta-se que as escolhas tecnológicas desempenharam papel fundamental para o êxito do projeto, não apenas como ferramentas de implementação, mas como elementos estruturais que garantiram o alinhamento entre requisitos funcionais, desempenho e usabilidade. O .NET (C#) no *back-end* ofereceu robustez e segurança para o processamento de pedidos e autenticação de usuários, além de assegurar escalabilidade para lidar com o aumento de transações simultâneas em ambientes reais. O *ReactJS* aliado ao *TypeScript* no *front-end*

possibilitou a criação de interfaces dinâmicas e confiáveis, em que a tipagem estática reduziu falhas de integração e proporcionou maior consistência na comunicação com a API, resultando em uma experiência de uso fluida e intuitiva para garçons e gestores. Já o *PostgreSQL*, utilizado como banco de dados relacional, assegurou integridade transacional e desempenho em consultas complexas, permitindo o controle em tempo real de estoque, comandas e fluxo financeiro sem comprometer a estabilidade do sistema. Complementarmente, o uso do *Trello*, baseado em metodologia ágil, garantiu o acompanhamento estruturado das entregas, favorecendo a organização do desenvolvimento e a priorização de funcionalidades críticas. Dessa forma, a combinação dessas tecnologias não apenas viabilizou a implementação, mas também reforçou a confiabilidade, a eficiência e a escalabilidade da solução proposta, contribuindo diretamente para o alcance dos objetivos do trabalho.

Como trabalhos futuros, destacam-se a expansão das funcionalidades de relatórios e *dashboards*, com análises mais avançadas para apoiar a tomada de decisão gerencial, e a integração com sistemas de fidelização de clientes. Outro ponto de evolução é o desenvolvimento do aplicativo mobile nativo que complemente o *PWA* já existente, oferecendo uma experiência ainda mais integrada aos dispositivos móveis. A versão nativa possibilitará explorar recursos específicos dos *smartphones*, como notificações *push*, acesso *offline* e maior personalização, ampliando a mobilidade e a eficiência no atendimento. Esse avanço tem o potencial de tornar o sistema ainda mais completo e acessível, beneficiando tanto os garçons no registro de pedidos quanto os gestores no acompanhamento em tempo real das operações. Para agregar valor contábil à gestão, também se prevê a integração com sistemas de emissão de nota fiscal eletrônica, facilitando a conformidade fiscal e reduzindo o retrabalho administrativo

Referências

EVANGELISTA, K. P. Gestão de estoques por meio da curva ABC: estudo de caso em um restaurante universitário de Patos de Minas. *Revista Perquirere*, Patos de Minas, v. 17, n. 1, p. 117-135, 2020.

OLIVEIRA, Breno Martins. Experiência prática em gestão de bares e restaurantes, com base em quatro empresas familiares do setor alimentício. Informação pessoal, Franca-SP, 2024.

RIBEIRO, R. E. M. Gestão de precificação e controle de estoque: Estudo de caso em restaurante no Piauí. *Research, Society and Development*, v. 13, n. 3, p. e48530138122, 2024.

VASCONCELOS, Y. L. Gestão de estoque em restaurantes: um estudo de caso. *Revista Gestão da Produção em Sistemas Agroindustriais – GEPROS*, Bauru, v. 8, n. 3, p. 45-57, 2013.

OLIVEIRA, J. et al. Desenvolvimento de um Sistema para a Comissão Própria de Avaliação: Uma Abordagem *FullStack* Utilizando C# e *React.js*. In: ANAIS DO CONGRESSO DE ENGENHARIA DE SOFTWARE DO CENTRO UNIVERSITÁRIO ACADEMIA (CONES-UNIACADEMIA), Juiz de Fora, v. 6, n. 1, p. 77-88, 2023.

SANTANA, E. J. Usabilidade em Sistemas de Informação: uma avaliação do módulo ensino do Sig@UFPE. 2020. 79 f. Trabalho de Conclusão de Curso (Especialização em Gestão da Informação e do Conhecimento) - Universidade Federal de Pernambuco, Recife, 2020.

SANTOS, L. O. C.; GUIMARÃES JUNIOR, D. S. Tecnologia da informação na gestão de restaurantes: uma revisão sistemática. Revista Hospitalidade, São Paulo, v. 19, p. 345–375, 2023.