SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL
“ALMIRANTE TAMANDARÉ”

CURSO TÉCNICO DESENVOLVIMENTO DE SISTEMAS

GUILHERME FANTATO RODRIGUES

JOÃO MATEUS NEPOMUCENO

LEONARDO LOPES MORAES

VINÍCIUS OLIVEIRA DOMINGOS DE JESUS

YANN BENAVIDE ZORNEK

OTIMIZAÇÃO DA EXPERIÊNCIA DO CLIENTE: UM ESTUDO DE CASO COM O APLICATIVO ATENDE+

SÃO BERNARDO DO CAMPO

2025

GUILHERME FANTATO RODRIGUES

JOÃO MATEUS NEPOMUCENO

LEONARDO LOPES MORAES

VINÍCIUS DOMINGOS OLIVEIRA DE JESUS

YANN BENAVIDE ZORNEK

OTIMIZAÇÃO DA EXPERIÊNCIA DO CLIENTE: UM ESTUDO DE CASO COM O APLICATIVO ATENDE+

Trabalho de conclusão de curso para obtenção do título técnico em Desenvolvimento de Sistemas apresentado à Instituição de ensino Serviço Nacional de Aprendizagem Industrial “Almirante Tamandaré”.

Orientador(a): Prof. Pedro Castilho Cardoso

SÃO BERNARDO DO CAMPO

2025

GUILHERME FANTATO RODRIGUES

JOÃO MATEUS NEPOMUCENO

LEONARDO LOPES MORAES

VINÍCIUS DOMINGOS OLIVEIRA DE JESUS

YANN BENAVIDE ZORNEK

OTIMIZAÇÃO DA EXPERIÊNCIA DO CLIENTE: UM ESTUDO DE CASO COM O APLICATIVO ATENDE+

Trabalho de conclusão de curso para obtenção do título técnico em Desenvolvimento de Sistemas apresentado à Instituição de ensino Serviço Nacional de Aprendizagem Industrial “Almirante Tamandaré”. Aprovado em:

_____

Prof. Edgard Coutinho Silva

_____

Prof. Pedro Castilho Cardoso

Dedicamos este trabalho aos nossos professores por toda dedicação e empenho durante os 4 semestres de nossa formação.

AGRADECIMENTOS

Agradecemos inicialmente a Deus em todas as suas formas de manifestação por nos conceder força e inspiração ao longo de nossa jornada.

A família pelo amor incondicional e incentivo expressado, demonstrando força e confiança em nós inclusive nos momentos de estresse.

Aos professores pelo compromisso de ensino, pela orientação e dedicação em nosso aprendizado, acreditando em nosso potencial até quando nós mesmos duvidamos.

Por fim, agradecemos a todos que, de alguma forma, contribuíram para o nosso estudo e aprimoramento individual e coletivo com conselhos, ensinamentos ou qualquer contribuição presente em nossas vidas.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema voltado para o gerenciamento de pedidos, com o objetivo de otimizar o processo de atendimento ao cliente e permitir o autoatendimento do mesmo em sistemas como o de uma pizzaria. Diante do crescimento do setor alimentício e da necessidade de serviços mais ágeis e precisos, identificou-se que muitos estabelecimentos ainda enfrentam dificuldades relacionadas a falhas na comunicação dos pedidos, demora no atendimento e falta de integração entre as etapas de produção e entrega. As metodologias aplicadas foram fundamentadas em **Scrum e Kanban**, permitindo compreender as necessidades da aplicação e elaborar um produto com potencial funcional e eficiente.

Palavras-chave: aplicativo *mobile*; pedidos; pizzaria; tecnologia; *back-end*.

ABSTRACT

This work presents the development of a system designed for order management, aiming to optimize the customer service process and enable customer self-service in a pizzaria. Given the growth of the food service sector and the increasing demand for faster and more accurate services, it was identified that many establishments still face challenges related to communication failures in orders, delays in customer service, and a lack of integration between production and *delivery* stages. The methodologies applied was based on **Scrum and Kanban**, allowing for a deeper understanding of the application needs and the development of a functional and efficient product. As a result, the project met the proposed expectations, demonstrating practical applicability and potential for future expansions.

Keywords: mobile application; orders; pizzeria; technology.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API – Interfaces de Programação de Aplicações

API REST – *Representational State Transfer*

BA – *Business Analytics*

BE – *Back-End*

CRUD – *Create, Read, Update, Delete*

CSS – *Cascading Style Sheets*

DB – *Database*

DBA – Administrador de Banco de Dados

DER – Diagrama de Entidade-Relacionamento

FE – *Front-End*

IDE – *Integrated Development Environment*

JWT – JSON *Web Token*

MDN – *Mozilla Developer Network*

MER – Modelo de Entidade-Relacionamento

ORM – *Object-Relational Mapping*

PO – *Product Owner* (Dono do Produto)

QA – *Quality Assurance*

QR – *Quick Response* (*Code*)

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

RN – Regras de Negócio

SQL – *Structured Query Language*

TDD – *Test-Driven Development*

UX – *User Experience*

UI – *User Interface*

VS Code – Visual Studio Code

1 INTRODUÇÃO

O avanço das tecnologias digitais tem promovido transformações expressivas em diversos setores, incluindo o ramo alimentício, que passou a demandar soluções cada vez mais ágeis, precisas e integradas. Nesse cenário, a adoção de sistemas informatizados tornou-se indispensável para otimizar processos, reduzir falhas operacionais e aprimorar o atendimento ao cliente. A crescente busca por rapidez, conveniência e autonomia no consumo evidencia a necessidade de ferramentas tecnológicas que atendam às novas expectativas dos usuários. (LAUDON; LAUDON, 2020)

Entretanto, muitos estabelecimentos ainda enfrentam problemas como falhas na comunicação de pedidos, atrasos no atendimento e falta de integração entre as etapas de produção, entrega e controle interno. Tais dificuldades comprometem a eficiência operacional, geram retrabalho e impactam diretamente a experiência do cliente. Diante desse contexto, surge a necessidade de soluções que automatizem o e ofereçam maior autonomia ao consumidor, especialmente em ambientes dinâmicos como pizzarias. (FITZSIMMONS; FITZSIMMONS, 2014)

A escolha do presente objetivo justifica-se pela importância de modernizar e tornar mais eficiente o processo de controle de pedidos, etapa essencial para o bom funcionamento de pizzarias e estabelecimentos similares. O uso de tecnologias adequadas contribui para agilizar a comunicação entre atendentes, cozinha e o próprio consumidor, minimizar erros recorrentes e proporcionar uma experiência mais satisfatória ao cliente. Assim, compreender e propor soluções digitais para essa demanda torna-se relevante tanto do ponto de vista técnico quanto gerencial.

Nesse sentido, o objetivo principal deste trabalho é desenvolver um aplicativo *mobile* funcional e intuitivo, capaz de otimizar o processo de atendimento, permitindo que o próprio cliente realize de forma autônoma todo o fluxo do pedido, desde a escolha dos itens até o acompanhamento dele. Busca-se, com isso, reduzir falhas, agilizar o serviço e demonstrar como soluções digitais podem impactar positivamente o desempenho de pequenos e médios negócios do setor alimentício.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolver um software para gerenciar o autoatendimento de novos clientes nas pizzarias.

1.1.2 Objetivos Específicos

- Economizar tempo do cliente
- Facilitar e diversificar a forma de fazer um pedido
- Modernizar o atendimento de uma pizzaria
- Permitir o autoatendimento de um cliente

2 REVISÃO DE LITERATURA

O avanço das tecnologias digitais e o crescimento do consumo móvel transformaram profundamente a forma como os estabelecimentos do setor alimentício realizam o atendimento ao cliente. Atualmente, soluções de autoatendimento, aplicativos de pedidos e sistemas integrados de gestão representam o estado mais avançado das práticas adotadas para otimizar processos e melhorar a experiência do usuário. Segundo James A. Fitzsimmons e Mona J. Fitzsimmons (2014), a digitalização dos serviços tem ampliado a eficiência operacional, reduzido tempos de espera e permitido maior autonomia ao consumidor, alinhando-se às novas demandas do mercado.

De forma geral, o setor alimentício tem incorporado sistemas *mobile* que oferecem funcionalidades como montagem de pedidos, acompanhamento de preparo, pagamento online e recomendação personalizada. Aplicativos amplamente utilizados no Brasil, como iFood e Rappi, consolidaram o uso de plataformas digitais como meio principal para pedidos de restaurantes, demonstrando a maturidade dessa tecnologia e a aceitação do público. Contudo, conforme destacam Turban, Pollard e Wood (2018), grande parte desses serviços ainda é voltada para o ambiente externo (*delivery*), enquanto muitos estabelecimentos carecem de soluções internas eficientes para o atendimento presencial, principalmente no modelo de autoatendimento.

Outra tendência consolidada no estado da arte é a adoção de interfaces intuitivas, fundamentadas em princípios modernos de UX/UI, que visam reduzir a carga cognitiva do usuário e facilitar a navegação em telas reduzidas. Nielsen (2014) destaca que sistemas projetados com foco na usabilidade apresentam maior taxa de aceitação e menor índice de erros, características essenciais em ambientes de alta rotatividade, como pizzarias e restaurantes. Além disso, o uso de QR Codes como ponto de acesso ao sistema tornou-se uma prática amplamente aceita, permitindo que o cliente vincule seu dispositivo à mesa de forma rápida, sem necessidade de interação com funcionários.

Do ponto de vista tecnológico, *frameworks* híbridos como React Native têm sido amplamente adotados no desenvolvimento *mobile* por oferecerem portabilidade, boa performance e redução de custos, características essenciais para projetos que demandam agilidade e compatibilidade multiplataforma. Satyanarayanan (2015) ressalta que aplicações móveis modernas precisam equilibrar desempenho,

conectividade e experiência do usuário, o que torna essas tecnologias adequadas para sistemas de pedidos em tempo real. Além disso, soluções baseadas em arquitetura cliente-servidor e APIs REST consolidaram-se como padrão para integração entre *front-end* e *back-end*, permitindo comunicação eficiente, escalabilidade e segurança operacional (FIELDING, 2000).

Adicionalmente, serviços em nuvem, como Supabase, Firebase e outros provedores de *back-end*, representam uma evolução significativa no estado da arte. Eles facilitam o gerenciamento de dados, autenticação e armazenamento de imagens, além de oferecerem alta disponibilidade e sincronização em tempo real, fatores cruciais para aplicativos de autoatendimento. De acordo com Pautasso, Zimmermann e Leymann (2008), tais abordagens permitem abstrair complexidades da infraestrutura, concentrando esforços na entrega de valor ao usuário.

Apesar dos avanços, ainda existem limitações nos sistemas amplamente utilizados. Muitos aplicativos comerciais não oferecem personalização profunda de pedidos, integração direta com mesas físicas, ou controle interno completo para garçons e administradores. Além disso, a dependência de plataformas externas para *delivery* reduz a autonomia dos estabelecimentos, aumentando custos operacionais e dificultando a fidelização do cliente. Nesse contexto, soluções próprias e customizáveis tornam-se essenciais para atender necessidades específicas de cada restaurante.

Assim, o sistema insere-se no estado da arte ao combinar tendências modernas já consolidadas — como autoatendimento *mobile*, QR Code, APIs REST, banco de dados relacional, interface intuitiva e integrações em nuvem, com requisitos específicos do ambiente interno da pizzaria, oferecendo uma alternativa completa, autônoma e alinhada às práticas tecnológicas mais recentes. Dessa forma, o projeto se posiciona como uma evolução natural das soluções existentes, atendendo lacunas ainda presentes no mercado e fornecendo uma base sólida para futuras expansões.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Arquitetura Cliente-Servidor

A arquitetura cliente-servidor constitui um dos modelos fundamentais para o desenvolvimento de sistemas distribuídos, sendo amplamente adotada em aplicações web e *mobile*. Segundo Tanenbaum e Van Steen (2017), esse modelo se baseia na divisão de responsabilidades entre dois elementos principais: o cliente, que solicita serviços e representa a camada de interação com o usuário, e o servidor, responsável por fornecer os recursos, processar dados e gerenciar a lógica de negócio. Essa distribuição permite um fluxo organizado de comunicação e facilita a escalabilidade do sistema.

Em uma visão conceitual, o cliente atua como a interface que permite ao usuário enviar requisições e receber respostas. Ele é tipicamente caracterizado por sua natureza leve, pois delega grande parte do processamento ao servidor. Já o servidor concentra operações críticas, como validação de dados, persistência das informações, controle de acesso e execução das regras de negócio. Essa separação é essencial para manter a consistência dos dados e garantir a segurança da plataforma, principalmente em sistemas de múltiplos usuários (TANENBAUM; VAN STEEN, 2017).

A comunicação entre cliente e servidor ocorre, tradicionalmente, por meio de protocolos padronizados, entre os quais o protocolo HTTP se destaca. Conforme Fielding (2000), o HTTP permite que sistemas distribuídos se comuniquem de forma independente da linguagem ou plataforma utilizada, utilizando requisições baseadas em verbos como *GET*, *POST*, *PUT* e *DELETE*. Esse aspecto de independência é fundamental para a interoperabilidade de sistemas, especialmente em ambientes heterogêneos onde aplicativos *mobile*, navegadores web e serviços externos precisam se comunicar com o mesmo servidor.

Em arquiteturas modernas, além da divisão funcional, é comum a adoção de estruturas multicamadas. Bass, Clements e Kazman (2013) destacam que arquiteturas em camadas permitem organizar o software em blocos lógicos como apresentação, aplicação e dados, favorecendo a modularidade e a manutenção. Essa abordagem reduz a dependência entre componentes, facilita o isolamento de falhas e possibilita atualizações independentes, sem comprometer o sistema como um todo.

Outro aspecto essencial da arquitetura cliente-servidor é o conceito de estado. Sistemas podem operar de forma *stateful*, mantendo informações de sessão no servidor, ou *stateless*, onde cada requisição é tratada de forma independente. Fielding (2000) argumenta que a abordagem *stateless* gera maior escalabilidade e melhor distribuição de carga, motivo pelo qual se tornou predominante em APIs *REST* modernas. Já sistemas que exigem personalização contínua ou persistência de contexto, como plataformas de e-commerce, utilizam mecanismos como *tokens*, cookies ou sessões armazenadas de maneira distribuída.

No contexto de aplicativos *mobile* e sistemas de pedidos digitais, a arquitetura cliente-servidor é essencial para garantir sincronização entre múltiplos dispositivos e centralização das informações. Esse modelo permite que diferentes clientes, como smartphones, tablets e pontos de atendimento, operem sobre a mesma base de dados em tempo real. Além disso, a separação entre camadas de interface e lógica de negócio possibilita maior flexibilidade para futuras expansões, como integração com sistemas externos, *dashboards* administrativos ou módulos complementares.

Por fim, é importante ressaltar que a robustez da arquitetura cliente-servidor também depende de práticas complementares, como tratamento de erros distribuídos, balanceamento de carga, autenticação, mecanismos de cache e políticas de segurança. Tais elementos são indispensáveis para garantir disponibilidade, desempenho e integridade em sistemas que lidam com alto volume de requisições e informações sensíveis. Dessa forma, a arquitetura cliente-servidor permanece como um dos pilares fundamentais para o desenvolvimento de soluções escaláveis e confiáveis em ambientes computacionais contemporâneos.

3.2 APIs e Serviços Web

As APIs (Application Programming Interfaces) desempenham papel central no desenvolvimento de sistemas distribuídos, pois permitem a comunicação estruturada entre aplicações, serviços e dispositivos. De acordo com Paul e Prakash (2018), uma API define um conjunto de regras, protocolos e padrões que estabelecem como diferentes componentes de software devem interagir, garantindo interoperabilidade, modularidade e reutilização de funcionalidades. Em ambientes contemporâneos, onde aplicações precisam se comunicar continuamente com servidores, bancos de dados

e serviços externos, as APIs tornam-se fundamentais para a integração entre sistemas heterogêneos.

Um dos modelos mais amplamente utilizados para a implementação de APIs é o estilo arquitetural REST (Representational State Transfer). Proposto por Fielding (2000), o REST estabelece princípios como comunicação sem estado (*stateless*), uso de recursos identificáveis por URIs, e operações padronizadas baseadas no protocolo HTTP. Essa abordagem oferece simplicidade, escalabilidade e independência entre cliente e servidor, tornando-se ideal para aplicações web e *mobile*. A utilização dos métodos HTTP, como GET, POST, PUT e DELETE, facilita a estruturação das operações e promove uma padronização que reduz ambiguidades na comunicação entre componentes.

A segurança em APIs é outro aspecto crítico. De acordo com OWASP (2023), APIs são frequentemente alvos de ataques devido à sua exposição constante e à troca direta de dados sensíveis. Por isso, mecanismos como autenticação, autorização, criptografia e validação de entrada tornam-se essenciais para garantir a integridade e a confidencialidade das informações. Modelos de autenticação baseados em *tokens*, como JSON Web *Tokens* (JWT), são amplamente empregados devido à sua capacidade de funcionar de maneira distribuída, sem a necessidade de gerenciamento de sessão no servidor.

Em contextos de sistemas distribuídos, a capacidade de versionamento de APIs também é fundamental. Segundo Microsoft (2022), a evolução contínua dos serviços exige mecanismos que permitam mudanças sem quebrar compatibilidade com clientes já existentes. Estratégias como versionamento por URI, cabeçalhos ou parâmetros de requisição possibilitam a coexistência de múltiplas versões de uma mesma API, permitindo adaptações graduais e minimizando impactos para os usuários.

Por fim, o papel das APIs na integração de sistemas é fundamental em ecossistemas digitais contemporâneos. Elas permitem que serviços externos, como gateways de pagamento, provedores de autenticação, sistemas de gestão comercial e plataformas de notificação, sejam incorporados a aplicações de forma modular e segura. Essa característica reduz o acoplamento, aumenta a flexibilidade e possibilita a ampliação contínua das funcionalidades de uma plataforma sem a necessidade de reestruturar todo o software.

Assim, APIs e Serviços Web constituem a base da comunicação entre sistemas modernos, permitindo interoperabilidade, escalabilidade e integração distribuída. No contexto de aplicações *mobile* conectadas a servidores remotos, sua função é essencial para garantir que fluxos de dados ocorram de maneira eficiente, segura e padronizada.

3.3 Sistemas Mobile e Apps Híbridos

A computação móvel consolidou-se como um dos pilares da tecnologia contemporânea, impulsionada pela crescente adoção de smartphones e pela necessidade de acesso constante a serviços digitais. Segundo Satyanarayanan (2015), sistemas *mobile* representam uma evolução natural dos sistemas distribuídos, caracterizados pela mobilidade do usuário, limitações de hardware, variações de conectividade e necessidade de interfaces adaptativas. Esses fatores moldam tanto a arquitetura quanto as decisões de design e implementação das aplicações modernas.

O desenvolvimento de aplicativos móveis pode seguir diferentes abordagens, entre elas o desenvolvimento nativo, o desenvolvimento híbrido e as soluções baseadas em web. O desenvolvimento nativo utiliza linguagens e *frameworks* específicos de cada plataforma, como Swift e Objective-C no iOS, ou Java e Kotlin no Android, garantindo alto desempenho e acesso completo aos recursos do dispositivo (DARWIN; BUCH; FERREIRA, 2018). No entanto, essa abordagem gera maior custo de desenvolvimento, pois exige duas bases de código independentes.

Como alternativa, surgem os aplicativos híbridos, que combinam tecnologias web (HTML, CSS, JavaScript) com contêineres nativos capazes de executar o código em diferentes plataformas. De acordo com Grönli et al. (2014), soluções híbridas reduzem custos ao permitir a criação de um único código-fonte para múltiplos sistemas operacionais, mantendo desempenho satisfatório para a maioria dos cenários de uso. Essa abordagem, portanto, favorece projetos educacionais, aplicações comerciais e sistemas de atendimento que demandam agilidade e compatibilidade com diversos dispositivos.

Um dos principais fatores que influenciam a adoção de *frameworks* híbridos é o equilíbrio entre desempenho, custo, manutenção e tempo de entrega. Mahmood e Riaz (2018) destacam que arquiteturas híbridas aproveitam APIs nativas expostas por *bridges*, que servem como camadas de comunicação entre o código JavaScript e os

componentes do sistema operacional. Dessa forma, sensores, câmeras, GPS e armazenamento interno podem ser utilizados de maneira transparente, ampliando a utilidade das aplicações híbridas em cenários práticos.

Adicionalmente, os sistemas *mobile* demandam atenção especial à experiência do usuário, uma vez que são executados em dispositivos com telas menores, gestos variados e contextos de uso mais dinâmicos. Nielsen e Budiu (2013) afirmam que interfaces móveis devem priorizar acessibilidade, simplicidade e resposta imediata às ações do usuário, garantindo que a navegação ocorra de maneira fluida mesmo em condições de conectividade reduzida. *Frameworks* híbridos modernos incorporam esses princípios, permitindo interfaces responsivas que se adaptam a diferentes tamanhos de tela, densidades de pixels e orientações.

No contexto de aplicativos conectados a serviços remotos, a comunicação por meio de APIs se torna indispensável. Satyanarayanan (2015) ressalta que sistemas *mobile* precisam operar com tolerância a latência, interrupções e mudança de redes, o que exige mecanismos como cache local, sincronização parcial e estratégias *offline-first*. Aplicações híbridas facilitam essas práticas, pois utilizam tecnologias originalmente desenvolvidas para a web, onde tolerância à variabilidade de rede já é uma preocupação consolidada.

Portanto, os sistemas *mobile* e os aplicativos híbridos representam soluções eficazes para o desenvolvimento de aplicações multiplataforma, combinando portabilidade, boa performance, escalabilidade e redução de custos. Para projetos que exigem integração com APIs, atualização frequente, interfaces intuitivas e disponibilidade em múltiplos dispositivos, a arquitetura híbrida apresenta-se como uma escolha tecnicamente consistente e alinhada às tendências contemporâneas do desenvolvimento de software.

3.4 Banco de Dados Relacionais

Os bancos de dados relacionais constituem o paradigma dominante para armazenamento estruturado de informações em sistemas de pequeno, médio e grande porte. Conforme enfatizam Elmasri e Navathe (2011), o modelo relacional organiza dados em tabelas bidimensionais chamadas relações, compostas por tuplas e atributos, garantindo consistência, integridade e facilidade de acesso. Esse modelo, amplamente difundido no Brasil, tornou-se base para aplicações corporativas,

governamentais, educacionais e comerciais devido à sua robustez e simplicidade conceitual.

A teoria relacional foi construída para oferecer independência lógica e física dos dados, permitindo que mudanças na estrutura interna não afetem a forma como o usuário manipula as informações. Korth, Silberschatz e Sudarshan (2016) explicam que essa característica é essencial para sistemas distribuídos e aplicações que exigem escalabilidade, pois possibilita adaptações sem comprometer as operações já existentes. A separação entre esquema, armazenamento e acesso proporciona maior controle sobre a evolução do sistema.

Um aspecto central dos bancos relacionais é a linguagem SQL **(***Structured Query Language***)**, responsável por manipular dados e estruturas de maneira padronizada. De acordo com Silva (2018), o uso do SQL permite operações complexas, como junções, consultas condicionais e agregações, mantendo legibilidade e consistência na execução. Isso torna o SQL um recurso indispensável em sistemas que dependem de consultas rápidas, confiáveis e otimizadas, como plataformas de atendimento e sistemas de autoatendimento.

Os Sistemas Gerenciadores de Banco de Dados (SGBDs) relacionais amplamente utilizados no Brasil, como PostgreSQL, MySQL, Oracle e SQL Server, incorporam mecanismos de segurança, controle de acesso, replicação e otimização de consultas. De acordo com Silva (2018), esses SGBDs oferecem alta confiabilidade e desempenho, tornando-se adequados para aplicações que exigem alta disponibilidade, como sistemas de pedidos digitais e plataformas integradas com dispositivos móveis.

Portanto, o modelo relacional permanece como uma solução madura, sólida e eficaz para representar dados estruturados em ambientes computacionais diversos. Em sistemas que demandam consistência, integração com APIs e múltiplos acessos simultâneos, o banco de dados relacional oferece fundamentos essenciais para manter a integridade e o funcionamento contínuo da aplicação.

3.5 UX/UI e Autoatendimento

A Experiência do Usuário (UX) e o Design de Interface (UI) desempenham um papel fundamental no desenvolvimento de sistemas digitais, especialmente em aplicações móveis e plataformas de autoatendimento. Segundo Barbosa e Silva

(2011), UX refere-se ao conjunto de percepções, emoções e respostas que o usuário experimenta ao interagir com um sistema, abrangendo aspectos como usabilidade, acessibilidade, aprendizado, satisfação e eficiência. Assim, o design não se resume à estética, mas à construção de interações que facilitem o uso e potencializem o valor da tecnologia.

De acordo com Nielsen (2014), a usabilidade é composta por atributos como facilidade de aprendizado, eficiência, memorização, redução de erros e satisfação. Esses princípios orientam a criação de interfaces funcionais e intuitivas, permitindo que o usuário alcance seus objetivos com o mínimo de atrito. Em ambientes móveis, esses atributos tornam-se ainda mais relevantes devido a fatores como telas menores, interação por toque e variações de contexto de uso, como deslocamento, restrições de iluminação e limitações momentâneas de atenção.

No contexto brasileiro, pesquisadores como Cybis, Betiol e Faust (2015) destacam que o design centrado no usuário deve envolver processos iterativos de prototipagem, testes e refinamento. A participação ativa dos usuários nas etapas iniciais permite identificar dificuldades, expectativas e padrões cognitivos, aumentando a aderência do sistema ao seu público-alvo. Esse enfoque humanizado é particularmente importante em interfaces de autoatendimento, onde a autonomia do usuário precisa ser maximizada.

O autoatendimento é uma tendência crescente em setores comerciais, educacionais e de serviços. De acordo com Pereira (2019), sistemas de autoatendimento reduzem filas, agilizam processos e ampliam a autonomia do usuário, desde que sejam projetados com clareza visual e fluxo de navegação coerente. Para plataformas de pedidos, como em restaurantes e lanchonetes, o autoatendimento exige que informações como preços, categorias, imagens, adicionais e etapas de compra estejam organizadas de forma clara e acessível, minimizando a necessidade de suporte humano.

A psicologia cognitiva também contribui para o entendimento do comportamento do usuário. Conforme explica Lidwell, Holden e Butler (2010), princípios como carga cognitiva reduzida, hierarquia visual, proximidade e alinhamento ajudam a orientar a atenção e facilitar a tomada de decisão. Em interfaces de autoatendimento, isso se traduz em elementos como botões destacados,

categorização clara e feedback imediato ao toque, diminuindo erros e evitando frustrações.

Além disso, a velocidade de resposta é um fator decisivo para a percepção de qualidade. Estudos de Rocha e Baranauskas (2003) apontam que, em contextos digitais, atrasos superiores a poucos segundos afetam negativamente a experiência, comprometendo a confiança do usuário no sistema. Aplicações móveis de autoatendimento, portanto, devem equilibrar comunicação com o servidor, otimização de imagens, mecanismos de cache e atualizações em tempo real para garantir fluidez no uso.

Por fim, interfaces bem projetadas contribuem diretamente para o sucesso do negócio. Conforme Lourenço (2020), sistemas de autoatendimento com boa experiência do usuário aumentam taxas de conversão, melhoram a percepção de qualidade do estabelecimento e reduzem erros operacionais. Assim, a integração entre UX, UI e princípios de autoatendimento é essencial para sistemas que envolvem pedidos, pagamentos ou registro de informações.

3.6 Metodologias Ágeis

As metodologias ágeis surgiram como resposta às limitações dos modelos tradicionais de desenvolvimento de software, especialmente o modelo em cascata, que apresenta baixa flexibilidade diante de mudanças de requisitos. De acordo com Schwaber e Sutherland (2020), o movimento ágil prioriza a entrega contínua de valor, a comunicação eficiente entre os membros da equipe e a capacidade de adaptação rápida às necessidades do projeto. No Brasil, autores como Moura (2014) destacam que as metodologias ágeis se tornaram amplamente utilizadas em ambientes corporativos e educacionais por promoverem dinamismo, colaboração e melhoria contínua.

O Manifesto Ágil, publicado em 2001, estabelece quatro valores centrais:

1. *Indivíduos e interações mais que processos e ferramentas;*
2. *Software em funcionamento mais que documentação extensa;*
3. *Colaboração com o cliente mais que negociação de contratos;*
4. *Responder a mudanças mais que seguir um plano.*

Esses valores representam uma mudança de mentalidade, na qual o foco é o produto entregue e a satisfação do usuário, e não apenas o cumprimento estrito de

etapas rígidas. Pressman e Maxim (2016) observam que essa abordagem favorece ambientes onde requisitos são voláteis, como sistemas de atendimento, plataformas web e aplicativos móveis.

Entre os *frameworks* ágeis mais adotados, destaca-se o Scrum, amplamente aplicado no Brasil em equipes de desenvolvimento de software. Segundo Schwaber e Sutherland (2020), o Scrum organiza o processo em ciclos curtos chamados *sprints*, nos quais a equipe planeja, desenvolve e revisa incrementos do produto. Esse modelo favorece a transparência, a inspeção contínua e a adaptação, pilares essenciais para manter a qualidade e o foco nas necessidades reais do usuário.

Outro método relevante é o Kanban, que se baseia na visualização do fluxo de trabalho por meio de quadros organizados em colunas representando etapas do processo. Para Anderson (2010), o Kanban busca limitar o trabalho em progresso (*Work in Progress – WIP*), identificar gargalos e aumentar a eficiência do fluxo produtivo. No Brasil, Kanban é frequentemente utilizado em conjunto com o Scrum, criando abordagens híbridas que equilibram previsibilidade e flexibilidade.

As metodologias ágeis também possuem impacto significativo na qualidade do software. Conforme destaca Pacheco e Maldonado (2012), o ciclo iterativo permite testes constantes, integração contínua e ajustes rápidos de funcionalidades, o que contribui diretamente para a robustez e evolução do produto final. Em sistemas de autoatendimento e aplicativos móveis, que lidam com mudanças frequentes e dependem de resposta rápida às demandas do consumidor, esse modelo se mostra particularmente eficiente.

Assim, as metodologias ágeis estruturam o processo de desenvolvimento de software de forma dinâmica, colaborativa e orientada a resultados. Sua aplicação favorece projetos em que a comunicação, a rápida adaptação e a entrega incremental de valor são essenciais, garantindo maior qualidade, eficiência e proximidade com as necessidades reais do usuário final.

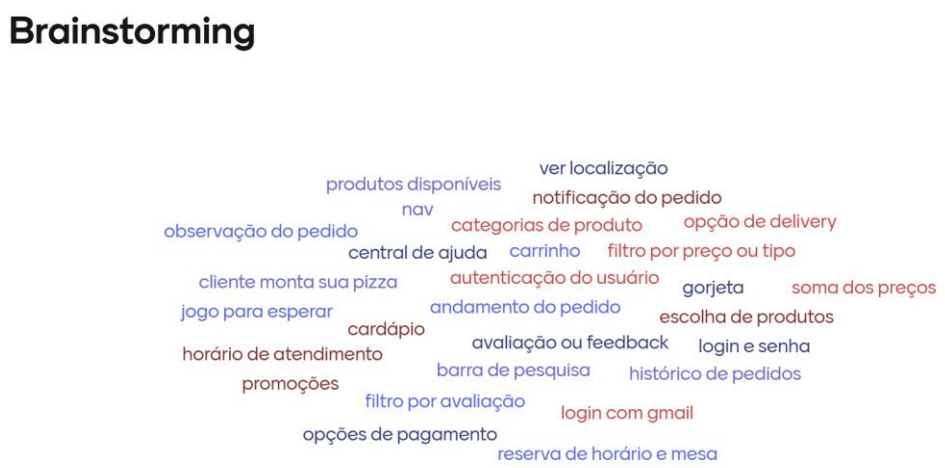
4 DESENVOLVIMENTO

4.1 Brainstorming (Chuva de Ideias)

Para iniciar o projeto, o grupo realizou uma sessão de *brainstorming*, técnica amplamente utilizada em processos criativos para estimular a geração livre de ideias e promover a participação coletiva dos integrantes. Segundo Vergara (2016), o *brainstorming* favorece o pensamento divergente e contribui para a identificação de diferentes perspectivas sobre um problema, permitindo que alternativas sejam discutidas, refinadas e organizadas de maneira sistemática. Dessa forma, essa metodologia foi essencial para que a equipe levantasse possibilidades, definisse funcionalidades relevantes para o sistema e estruturasse o Backlog do Produto de forma coerente com as necessidades identificadas

A Figura 1 apresenta o *brainstorming* realizado pelo grupo.

Figura 1 - Brainstorming visual no aplicativo Mentimeter.



Fonte: Autoral.

4.2 Backlog Produto

O *Product Backlog*, ou *backlog* do produto, consiste em uma lista evolutiva, priorizada e constantemente atualizada que reúne todas as funcionalidades, requisitos, melhorias e correções necessárias para a construção do software. Segundo Carvalho (2019), o *backlog* funciona como o artefato central de planejamento em metodologias ágeis, orientando a equipe sobre o que deve ser desenvolvido e em qual ordem. Após a realização da etapa de *brainstorming*, os integrantes do grupo registraram e organizaram todas as ideias coletadas, transformando-as em itens do

backlog. Em seguida, o *Product Owner* (PO) realizou a priorização desses elementos, conforme preconizado por Rosa (2015), com o objetivo de estruturar o fluxo de trabalho e garantir que as tarefas mais relevantes para o valor do produto fossem executadas primeiro durante o desenvolvimento do aplicativo.

- Backlog Do Produto
  - Andamento do pedido – Prioridade Alta
  - Avaliação/Feedback - Prioridade Média
  - Central de Ajuda - Prioridade Média
  - Recomendação de Pedidos/Histórico - Prioridade Leve
  - Barra de Pesquisa - Prioridade Média
  - Filtro - Prioridade Média
  - Tela de Login - Prioridade Importante
  - Carrinho - Prioridade Importante
  - Gorjeta - Prioridade Leve
  - Pagamentos - Prioridade Alta
  - Cardápio - Prioridade Alta
  - Escolha de Produtos - Prioridade Alta
  - Observações do pedido - Prioridade Média
  - Menu de Navegação - Prioridade Alta
  - Opção *Delivery* - Prioridade Alta
  - Localização - Prioridade Leve

4.3 Histórias de Usuários

As histórias de usuário constituem uma técnica amplamente utilizada nas metodologias ágeis para descrever requisitos de forma simples, objetiva e centrada nas necessidades do usuário. De acordo com Carvalho (2019), as histórias de usuário permitem que o time compreenda o propósito de cada funcionalidade, destacando o valor entregue ao cliente e facilitando o alinhamento entre desenvolvimento e expectativa do usuário. Trata-se de uma forma narrativa curta que expressa, sob a perspectiva do usuário final, o que ele deseja realizar no sistema e por qual motivo. Seguindo a abordagem apresentada por Rosa (2015), essas histórias geralmente adotam o formato: “Como [persona], eu quero [intenção], para que [benefício]”, o que torna a documentação clara, compreensível e facilita a priorização ao longo do processo de desenvolvimento.

- BACKLOG com história de usuários
    - Andamento do pedido - Prioridade Alta

Como usuário, eu quero saber como está o andamento do meu pedido para que eu possa me situar enquanto ao tempo de demora para ele estar pronto.

    - Avaliação/Feedback - Prioridade Média

Como usuário, eu quero poder avaliar o serviço do local que eu estou comendo para que eu possa ajudar a melhorar o serviço e a qualidade para a próxima vez que eu for pedir.

    - Central de Ajuda - Prioridade Média

Como usuário, eu quero poder ser atendido caso haja algum erro no meu pedido ou no aplicativo para que a pizzaria corrija e melhore meu atendimento.

    - Recomendação de Pedidos/Histórico - Prioridade Leve

Como usuário, eu desejo lembrar os meus pedidos antigos para que eu possa escolher mais rápido os sabores parecidos que correspondem aos meus gostos.

- Barra de Pesquisa - Prioridade Média

Como cliente, eu gostaria de pesquisar os produtos disponíveis, pois assim seria mais fácil de buscar os meus produtos preferidos ou algo que eu esteja buscando.

- Filtro - Prioridade Média

Como usuário, eu quero filtrar os produtos disponíveis em categorias e classificações para que eu possa ver todos os itens listados de acordo com minhas especificações.

- Tela de Login - Prioridade importante

Como usuário, eu gostaria de ter uma boa tela de login, que seja intuitiva e fácil, para que assim eu possa fazer o login mais rápido e poder ver minhas informações ou personalizar o meu perfil.

- Carrinho - Prioridade importante

Como cliente, eu quero adicionar produtos ao meu carrinho para que eu possa revisar e finalizar minha compra com facilidade.

- Gorjeta - Prioridade leve

Como cliente, eu quero poder adicionar uma gorjeta no momento do pagamento para que eu possa entregar um feedback pessoal e uma recompensa pelo bom atendimento.

- Pagamentos - Prioridade Alta

Como cliente necessito que tenha pagamentos convencionais como cartões e pix. Além de mostrar visivelmente a área desse mesmo pagamento, para que eu veja qual a melhor opção.

- Cardápio - Prioridade Alta

Como cliente desejo ter a visualização dos produtos no cardápio, tendo imagem dos produtos, informações nutricionais e destaque de promoções e produtos mais pedidos para que possa ter base e certeza do que desejo.

- Escolha de Produtos - Prioridade Alta

Como cliente quero escolher sabores, tamanhos e opção de pedir bebidas em uma tela intuitiva para que o meu agrado seja satisfeito em escolher diferentes preferências sem me confundir com a interface.

- Observações do pedido - Prioridade Média

Como cliente, posso adicionar observações no pedido, que são comentários que necessitam de atenção seja no momento da produção ou da entrega da comida, como "pepperoni extra", "sem cogumelo" ou "favor tocar campainha".

- Menu de Navegação - Prioridade Alta

O site necessita de um menu com as principais opções no topo da tela de forma organizada, como "cardápio", "contato", "restaurantes", "carrinho". Dessa forma, posso acessar as principais funcionalidades sem precisar olhar em lugares diferentes.

- Opção *Delivery* - Prioridade Alta

Como cliente, posso ter a opção de *delivery* para comer pizza onde eu quiser, desde que eu a peça por ligação ou online.

- Localização - Prioridade Leve

Como cliente, posso encontrar a localização exata de todos os restaurantes dentro do site de forma fácil, caso queira visitá-los.

4.4 Organograma

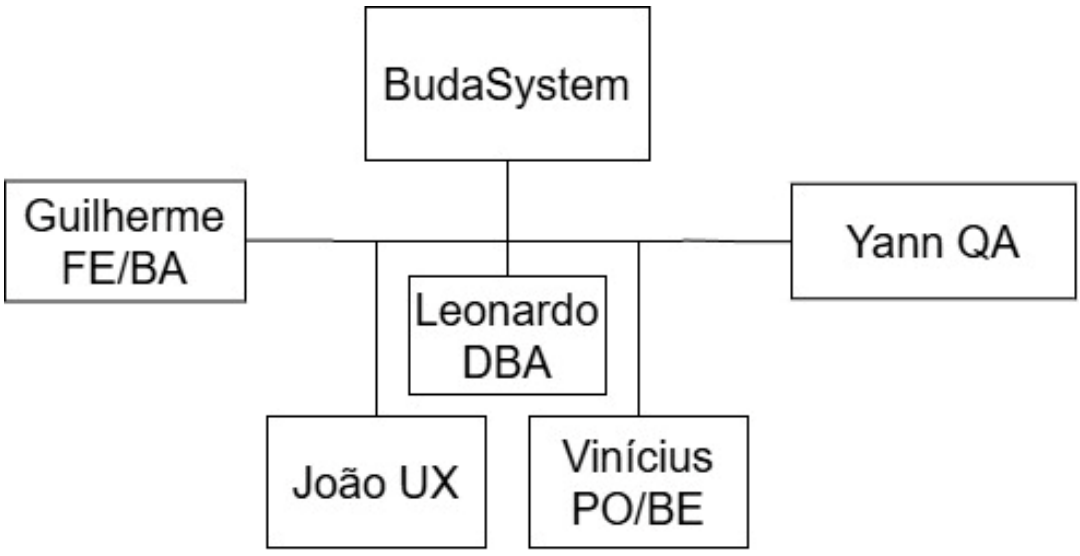
O organograma é uma representação gráfica que demonstra a estrutura hierárquica e funcional de uma organização, evidenciando os vínculos de autoridade, responsabilidade e comunicação existentes entre seus setores, equipes e cargos. Segundo Chiavenato (2014), o organograma funciona como um espelho da estrutura organizacional, permitindo visualizar como o trabalho é distribuído e como se estabelecem as relações formais dentro da instituição. Essa ferramenta é essencial para a clara compreensão dos níveis de subordinação e da divisão de tarefas, contribuindo para uma gestão mais eficiente e coerente.

Além de facilitar o entendimento da estrutura interna, o organograma também desempenha papel estratégico no alinhamento das atividades organizacionais. Para Maximiano (2020), a representação gráfica favorece a definição de papéis, evita sobreposição de funções e auxilia na coordenação das equipes, promovendo maior transparência na comunicação e garantindo que todos saibam suas responsabilidades. Vergara (2016) reforça que instrumentos visuais desse tipo são fundamentais em processos de coordenação de grupos, pois ampliam a clareza organizacional e fortalecem o trabalho colaborativo.

No contexto do projeto, o organograma foi elaborado com o propósito de definir funções e responsabilidades entre os integrantes, estabelecendo uma estrutura clara para o desenvolvimento do sistema onde todos estavam no mesmo nível hierárquico. A utilização dessa ferramenta permitiu distribuir atividades de acordo com as competências individuais, otimizar o fluxo de trabalho e tornar o processo de desenvolvimento mais organizado, conforme sugerem as práticas de gestão apresentadas por Oliveira (2018). Dessa forma, o organograma não apenas representou a hierarquia do grupo, mas também funcionou como um instrumento para aumentar a eficiência e a coesão durante a execução do projeto.

A Figura 2 ilustra o organograma do grupo com as funções de cada integrante.

Figura 2 - Organograma do grupo BudaSystems



Fonte: Autoral.

4.5 Levantamento De Requisitos

O levantamento de requisitos é uma das etapas mais importantes do processo de desenvolvimento de software, pois busca identificar, organizar e documentar as necessidades do usuário e os objetivos que o sistema deve atender. De acordo com Pressman e Maxim (2016), essa etapa é fundamental para reduzir ambiguidades, evitar retrabalho e garantir que o produto final reflita corretamente as expectativas do cliente e do negócio. Para Sommerville (2019), requisitos bem definidos estabelecem uma base sólida para o projeto, permitindo que as fases seguintes, modelagem, implementação e testes, sejam conduzidas com maior precisão e eficiência.

No presente projeto, o levantamento de requisitos foi realizado por meio de técnicas como *brainstorming*, análise de necessidades e discussões em grupo, permitindo a consolidação dos pontos essenciais para o funcionamento do sistema. Conforme destacam Bezerra (2018) e Oliveira (2018), a participação ativa da equipe durante essa fase contribui para maior entendimento das demandas e melhora a qualidade das decisões de projeto. A partir dessa análise, foram definidas as Regras de Negócio (RN), os Requisitos Funcionais (RF) e os Requisitos Não Funcionais (RNF), que orientaram todo o planejamento da aplicação.

4.5.1 Regras de Negócio (RN)

As regras de negócio consistem em diretrizes que representam políticas, procedimentos, restrições e condições específicas do domínio da aplicação. Chiavenato (2014) ressalta que regras bem definidas auxiliam na padronização dos processos organizacionais, garantindo consistência na execução das atividades. No contexto do desenvolvimento de software, Pressman e Maxim (2016) afirmam que as regras de negócio funcionam como elementos orientadores que regulam o comportamento do sistema e delimitam suas possibilidades de operação.

No projeto Atende+, as regras de negócio foram elaboradas com o objetivo de refletir a dinâmica real de funcionamento de uma pizzaria, assegurando que o sistema atendesse integralmente os processos de solicitação, montagem dos pedidos, controle de mesas, pagamentos e gestão interna. A formalização dessas regras permitiu estabelecer uma estrutura lógica para o *backend* e facilitou a criação dos requisitos funcionais correspondentes.

A Tabela 1 apresenta as Regras de Negócio definidas para o sistema, estruturando as condições e políticas que orientam seu funcionamento.

Tabela - 1 Tabela de Regras do Negócio

RN	Nome	Descrição	Relacionamentos RN	RF Associados	RNF Associados
RN01	Número de mesa	O cliente deve estar vinculado a uma mesa.	RN07, RN14	—	RNF07
RN02	Pedido concluído	Certificar-se de que o pedido foi concluído.	RN01	RF13	—
RN03	Cadastro CPF	O usuário deve conseguir cadastrar seu CPF no pagamento	RN13	RF09, RF17	—
RN04	Modificação de produtos e categorias	Apenas o administrador pode modificar produtos e categorias.	—	RF04	—
RN05	Informação dos preços	O preço dos produtos deve ser informado ao cliente.	—	RF02	—
RN06	Requisitos mínimos do pedido	O cliente deve pedir ao menos um produto para abrir um pedido.	—	RF11	—
RN07	Relação de pedido	Todo pedido deve estar relacionado com uma mesa.	RN01	RF11, RF12, RF13, RF14, RF15	—
RN08	Requisitos do produto	O produto deve possuir nome, preço, categoria e descrição.	—	RF01, RF02, RF08, RF10	—

RN09	Atualização de Status do pedido	O status deve ser atualizado pelo administrador e exibido ao cliente.	RN11	RF12	—
RN10	Cadastros de Usuário	Cada usuário deve ter um cadastro.	RN14	RF06, RF07, RF16	—
RN11	Permissão de Visualização da Lista de pedidos	Lista de pedidos em aberto só no desktop.	RN09	RF14	—
RN12	Permissão de Manipulação de Pedidos	Somente garçom e administrador manipulam pedidos.	RN09	RF11, RF12, RF13, RF14, RF15	—
RN13	Forma de pagamento	Cliente pode escolher método de pagamento.	—	RF17	—
RN14	Login Usuário	Usuário tem que estar logado para acessar o app	RN 01, RN10	RF06, RF12, RF16	RF01, RF02
RN15	QRCode número da mesa	Cliente deve ler o QRCode para vincular a mesa ao pedido.	RN01	RF11, RF12	—

Fonte: Autoral.

4.5.2 Requisitos Funcionais (RF)

Os requisitos funcionais descrevem as funcionalidades específicas que o sistema deve realizar a fim de atender às necessidades dos usuários e garantir o cumprimento das regras de negócio. Segundo Sommerville (2019), requisitos funcionais definem comportamentos observáveis do software e configuram a base para seu desenvolvimento técnico. Bezerra (2018) complementa afirmando que uma boa especificação funcional deve ser clara, objetiva e orientada ao valor entregue ao usuário.

No presente projeto, os requisitos funcionais foram derivados diretamente das regras de negócio e organizados em categorias relacionadas à autenticação, controle de pedidos, gerenciamento de produtos, operação das mesas, integração com QR Codes e fluxo de pagamento. Essa estruturação permitiu que a equipe compreendesse as funcionalidades prioritárias e orientou as etapas subsequentes de modelagem do banco de dados e construção da API.

A Tabela 2 reúne os Requisitos Funcionais do sistema, descrevendo as funcionalidades que devem ser implementadas para atender às necessidades do usuário e às regras de negócio estabelecidas.

Tabela - 2 Tabela de Requisitos Funcionais.

RF	**Nome**	**Descrição**	**Relaciona-mentos RN**	**RF Associados**	**RNF Associados**
RF01	Cadastrar Produto	O sistema deve permitir que o usuário (Funcionário) cadastre um novo produto informando nome, preço, descrição, categoria e imagem.	RN05	—	—
RF02	Catálogo de produtos	Exibição dos produtos com nome, descrição, ingredientes, preço e foto.	RN05	RF01	RNF05
RF03	Listar Produtos por Categoria	Consultar produtos filtrando por categoria.	RN02	RF04	RNF05
RF04	Cadastrar Categoria	Cadastro de uma nova categoria.	RN10	—	RNF05
RF05	Listar Produtos por Categoria	Consultar produtos por categoria.	RN08	RF04	RNF05
RF06	Cadastrar Usuário	Cadastro com nome, e-mail e senha.	RN10, RN14	RF05	RNF01, RNF02, RNF05

RF07	Consultar Detalhes do Usuário	Consultar dados do usuário autenticado.	RN10	RF06	RNF05
RF08	Remover Item do Pedido	Remover item específico de pedido.	RN08	—	RNF05
RF09	Cadastrar CPF	Permite o usuário cadastrar seu CPF no pagamento	RN03	—	RNF05, RNF06
RF10	Adicionar item ao pedido	Adicionar item a pedido existente.	RN08	—	RNF05, RNF06
RF11	Criar Pedido	Criar uma pedido.	RN01, RN07, RN12, RN15	RF10, RF09	RNF07
RF12	Detalhar Pedido	Consultar pedido com produtos e status.	RN07, RN09, RN12, RN14, RN15	—	—
RF13	Finalizar Pedido	Mudar status do pedido para true.	RN02, RN06, RN07, RN12	—	—
RF14	Listar pedidos	Listar pedidos true e false.	RN05, RN07, RN11, RN12	—	—
RF15	Enviar pedido	Enviar pedido ao cliente.	RN07, RN12	—	RNF07
RF16	Login	Realizar login no sistema.	RN10, RN14	—	RNF01, RNF02
RF17	Formas de pagamento disponíveis	Permite pagamento por débito, crédito ou dinheiro.	RN13	—	—

Fonte: Autoral.

4.5.3 Requisitos Não Funcionais (RNF)

Os requisitos não funcionais representam características qualitativas que o sistema deve possuir, como desempenho, segurança, usabilidade, confiabilidade e manutenibilidade. Para Pressman e Maxim (2016), esses requisitos são essenciais para determinar como o sistema deve se comportar, influenciando diretamente a experiência do usuário e a qualidade geral do produto. Já Sommerville (2019) destaca que requisitos não funcionais são tão críticos quanto funções explícitas, pois falhas nesses atributos podem comprometer toda a aplicação mesmo que suas funcionalidades básicas estejam corretas.

No projeto, os requisitos não funcionais incluíram aspectos como tempo de resposta, acessibilidade, armazenamento seguro de dados, criptografia de informações sensíveis, consistência na interface e compatibilidade com diferentes dispositivos. Esses elementos garantiram maior robustez ao sistema, especialmente por se tratar de um aplicativo utilizado em ambientes com fluxo contínuo de clientes.

A Tabela 3 apresenta os Requisitos Não Funcionais, que especificam as características de qualidade esperadas do sistema, como desempenho, segurança, usabilidade e confiabilidade.

Tabela - 3 Tabela de Requisitos Não Funcionais.

RNF	Nome	Descrição	Relacionamentos RN	RF Associados	RNF Associados
RNF01	Autenticação de login	Autenticar usuário para realizar login.	RN14	RF06	RNF02
RNF02	Criptografia da senha	Criptografia da senha para aumento de segurança.	RN14	RF06	RNF01
RNF03	Compatibilidade com navegadores	Compatível com diversos navegadores.	—	—	—

RNF04	Vinculação de imagem do produto	Vincular imagens em nuvem ao produto.	—	—	—
RNF05	Acesso à internet	O sistema deve possuir acesso à internet.	—	RF02, RF03, RF04, RF05, RF06, RF07, RF08, RF09, RF10	—
RNF06	Padronização do aplicativo	Aplicativo com ícones e identidade padronizada.	—	RF09, RF10	—
RNF07	Atualização automática	Pedidos atualizam automaticamente.	RN01	RF08, RF10, RF11, RF15	—
RNF08	Rotas	Ligações entre rotas e páginas.	—	—	—

Fonte: Autoral.

4.6 Banco de Dados

A estrutura de banco de dados da aplicação foi projetada para garantir confiabilidade, integridade dos dados e desempenho adequado às operações essenciais do sistema, como cadastro de produtos, gestão de pedidos, controle de usuários e registro de pagamentos. Para atender a esses requisitos, foi adotado o Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL, aliado ao uso do Prisma ORM como ferramenta de integração entre a camada de aplicação e a camada de persistência.

4.6.1 PostgreSQL

O PostgreSQL foi selecionado como SGBD por sua robustez, aderência a padrões SQL e confiabilidade em operações transacionais. Trata-se de uma solução amplamente utilizada em sistemas corporativos que exigem integridade referencial, suporte a grandes volumes de dados e estabilidade em ambientes distribuídos. Por

meio deste, é permitido que o sistema de autoatendimento realize operações críticas, como criação de pedidos, atualização de status e consultas de produtos, de forma precisa e segura.

4.6.2 Modelagem do Banco de Dados

A modelagem do banco de dados foi desenvolvida a partir do levantamento de requisitos, das regras de negócio e da análise dos fluxos operacionais identificados durante as etapas iniciais do projeto. Para assegurar integridade, coerência e escalabilidade, adotou-se um processo estruturado envolvendo modelagem conceitual (MER), modelagem lógica (DER) e definição física das tabelas, esta última representada no Dicionário de Dados apresentado nas subseções seguintes.

De acordo com Elmasri e Navathe (2011), o Diagrama Entidade-Relacionamento (DER) permite representar os dados de forma abstrata, destacando entidades, atributos e relacionamentos essenciais ao domínio do problema. Posteriormente, esse modelo foi convertido para um Modelo Entidade-Relacionamento (MER), que formaliza chaves primárias, chaves estrangeiras e cardinalidades, servindo como base para geração das estruturas físicas no PostgreSQL. A utilização dessa abordagem segue práticas recomendadas por Date (2012) e Pressman e Maxim (2016), garantindo aderência entre lógica de negócio e estrutura de armazenamento.

Coerência da Modelagem e Integridade Relacional

O modelo do banco de dados foi estruturado de maneira relacional e normalizada, aplicando:

- **Chaves primárias** para identificação única das entidades;
- **Chaves estrangeiras** para garantir integridade referencial (ex.: produto → categoria, order → mesa, itens → produto);
- **Constraints** que asseguram consistência, como atributos obrigatórios, tipos adequados e valores compatíveis;
- **Relacionamentos fortes (1:N) e fracos (N:N)** representados por tabelas associativas (como ingredientes_produtos e itens_adicionais), conforme descrito por Korth e Silberschatz (2016).

Esses componentes asseguram que o fluxo de dados, desde a escolha do produto até o pagamento, ocorra de maneira segura, coerente e automatizada.

4.6.3 Diagrama Entidade-Relacionamento (DER)

O Diagrama Entidade-Relacionamento (DER) representa a estrutura lógica do banco de dados, destacando entidades, atributos e relacionamentos antes da implementação física. Essa etapa é essencial para analisar a estrutura conceitual das informações e orientar a construção do modelo físico.

No contexto do sistema, o DER descreve as entidades principais apresentadas na subseção anterior. O DER facilita a visualização dos relacionamentos entre essas entidades, como:

- Um *usuário* pode realizar vários *pedidos*;
- Um *pedido* pode conter múltiplos *itens*;
- Cada *produto* pertence a uma *categoria*;
- Cada *item do pedido* está associado a um *produto* específico.

O diagrama completo do DER encontra-se no **Apêndice A**.

4.6.4 Modelo Entidade-Relacionamento (MER)

O Modelo Entidade-Relacionamento (MER) corresponde ao modelo lógico/físico do banco de dados, representando tabelas, tipos de dados, chaves primárias, chaves estrangeiras e cardinalidades.

Ele é resultado direto da transformação do DER para a estrutura implementável no PostgreSQL.

No MER do sistema são identificados:

- Tabelas com seus respectivos atributos;
- Chaves primárias, responsáveis pela identificação única dos registros;
- Chaves estrangeiras, estabelecendo integridade referencial entre tabelas;
- Cardinalidades e regras de negócio estruturadas;
- *Constraints* de unicidade e obrigatoriedade;
- Estrutura final que será consumida via Prisma ORM.

Esse diagrama permite compreender como o banco foi fisicamente implementado e garante a coerência entre API e persistência.

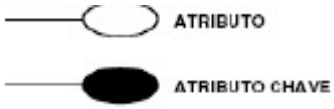
O MER completo encontra-se no **Apêndice B**.

4.6.5 Legenda e Convenções Utilizadas

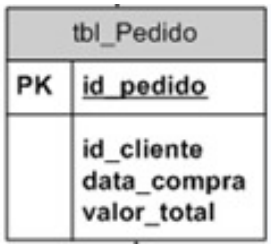
Para manter consistência visual, os diagramas DER e MER seguem as convenções comuns de modelagem de dados:

- **Retângulos:** entidades/tabelas.

- **Linhas com círculos:** atributos das entidades (no DER).



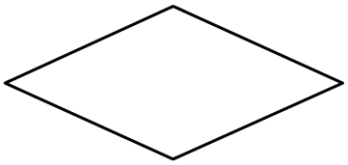
- **Retângulos com divisões internas:** tabelas contendo atributos (no MER).



- **Chave (PK):** identificação única da entidade/tabela.

- **Chave (FK):** referência a uma chave primária de outra tabela.

- **Losangos:** representam relacionamentos (no DER).



- **Setas ou linhas com cardinalidade:** indicam relacionamentos 1:1, 1:N ou 0:1.

1:N ——————————— **N:1**

Esses padrões seguem as recomendações de Santos (2018) para modelagem conceitual e lógica.

4.6.6 Dicionário de Dados (Físico)

O Dicionário de Dados (Físico) é um artefato essencial na documentação e modelagem de sistemas, pois descreve detalhadamente os elementos que compõem o banco de dados, como tabelas, atributos, tipos de dados, chaves e restrições estruturais. Sua principal finalidade é garantir padronização, consistência e integridade no tratamento das informações ao longo do ciclo de desenvolvimento. De acordo com Date (2012), a definição clara dos componentes de um banco de dados reduz ambiguidades e fortalece a coerência entre regras de negócio e estruturas de armazenamento. Pressman e Maxim (2016) destacam que a documentação precisa dos dados facilita a comunicação entre analistas, desenvolvedores e demais envolvidos, assegurando que todos compartilhem o mesmo entendimento da arquitetura do sistema.

No contexto do projeto, o Dicionário de Dados foi elaborado para representar de maneira estruturada entidades como clientes, pedidos, mesas, produtos e pagamentos, que compõem o núcleo funcional do sistema de autoatendimento. A organização desses elementos permitiu mapear relacionamentos críticos, como a vinculação entre mesas e pedidos ou entre produtos e ingredientes, garantindo que as operações do aplicativo fossem implementadas com segurança, coerência e aderência às regras de negócio definidas. Dessa forma, o Dicionário de Dados atua como um guia técnico indispensável, sustentando a consistência do banco de dados e contribuindo para o desempenho confiável da aplicação.

A Tabela 4 apresenta a estrutura da entidade *Cliente*, responsável por armazenar informações essenciais para identificação e interação do usuário com o sistema. Estes dados são fundamentais para operações como autenticação, registro de pedidos, personalização da experiência e vinculação às mesas. A definição clara dos atributos garante integridade e consistência nos processos de autoatendimento e gerenciamento de dados.

Tabela 4 - Dicionário de Dados da entidade Cliente.

Dicionário de Dados					
Cliente					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de identificação do cliente	3
nome	varchar	255 bytes		nome do cliente	Martins
senha	varchar	255 bytes		senha do cliente	1234
email	varchar	255 bytes		email do cliente	Martins@teste.com
cpf	Int	32 bytes		identificação do cliente para colocar na nota fiscal do pagamento	43981515838
data_de_nascimento	date	8 bytes		Data de nascimento do cliente para venda de produtos +18	2013-03-26
created_at	timeStamp(3)	8 bytes		Dia e hora da criação dos itens	2025-06-23 11:41:19
updated_at	timeStamp(3)	8 bytes		Dia e hora da última atualização do item	2025-06-24 12:33:20

Fonte: Autoral.

A Tabela 5 descreve a entidade Funcionários, que contempla os dados dos colaboradores autorizados a operar o sistema administrativo da pizzaria, incluindo garçons e administradores. Sua modelagem possibilita controle de acesso, registro de atividades internas e responsabilização sobre ações operacionais. Essa entidade é essencial para garantir segurança e rastreabilidade dentro do fluxo de pedidos.

Tabela 5 - Dicionário de Dados da entidade Funcionários.

Dicionário de Dados					
Funcionários					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Identificação única do usuário	1
cargo_id	Int	32 bytes	FK	Número de identificação do cargo em que o funcionário está inserido	3
nome	varChar	255 bytes		Nome pessoal do usuário	José
email	varChar	255 bytes		Endereço de email do usuário	jose@gmail.com
senha	varChar	255 bytes		Senha para login do usuário	123123
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do usuário	2025-06-23 11:41:19
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-06-24 12:33:20

Fonte: Autoral.

A Tabela 6 detalha a entidade Cargos, cuja finalidade é registrar os diferentes papéis desempenhados pelos funcionários no ambiente organizacional. Define a hierarquia do sistema, aplicando regras específicas de permissão e autenticação conforme o cargo, fortalecendo a divisão de responsabilidades dentro do sistema.

Tabela 6 - Dicionário de Dados da entidade Cargos.

Dicionário de Dados					
Cargos					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de Identificação dos cargos	1
nome	varChar	255 bytes		Nome para os cargos dos funcionários	Garçom

created_at	timeStamp(3)	8 bytes		Hora e dia da criação do cargo	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45

Fonte: Autoral.

A Tabela 7 apresenta a entidade Categorias, responsável por agrupar os produtos de forma organizada, facilitando a navegação do cliente no cardápio digital. Essa estruturação contribui para uma experiência mais intuitiva no aplicativo, além de apoiar o gerenciamento administrativo dos itens ofertados pela pizzaria.

Tabela 7 - Dicionário de Dados da entidade Categorias.

Dicionário de Dados					
Categorias					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Identificador das categorias	1
nome	varChar	255 bytes		Nome dos grupos de produtos	Pizzas
created_at	timeStamp(3)	8 bytes		Hora e dia da criação da categoria	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45

Fonte: Autoral.

A Tabela 8 descreve os Adicionais, elementos complementares que podem ser incorporados aos produtos principais. Sua definição permite adicionar ingredientes extras, personalizar opções e calcular preços adicionais, ampliando a flexibilidade do pedido e garantindo maior autonomia ao cliente no processo de escolha.

Tabela 8 - Dicionário de Dados da entidade Adicionais.

Dicionário de Dados					
Adicionais					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Identificador do adicional	1

nome	varChar	255 bytes		Nome do adicional	Catupiry
preço	float	4 bytes		preço do adicional	R$ 7,70
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do adicional	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45

Fonte: Autoral.

A Tabela 9 detalha a relação entre Categorias e Adicionais, permitindo associar grupos específicos de complementos a determinados tipos de produtos. Isso garante coerência no cardápio e evita combinações inconsistentes, assegurando que apenas adicionais pertinentes sejam exibidos ao cliente.

Tabela 9 - Dicionário de Dados da entidade Categorias_adicionais.

Dicionário de Dados					
Categorias_adicionais					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de indentificação da categoria_adicional	1
adicional_id	Int	32 bytes	FK	Número de Identificação do adicional	20
categoria_id	Int	32 bytes	FK	Número de Identificação da categoria	10
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do adicional	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45

Fonte: Autoral.

A Tabela 10 apresenta a entidade *Produtos*, uma das mais importantes do sistema. Ela contempla informações sobre os itens comercializados, como pizzas,

bebidas e promoções. A estrutura dessa entidade sustenta tanto a exibição do cardápio quanto o cálculo automático de preços e validações de disponibilidade.

Tabela 10 - Dicionário de Dados da entidade Produtos.

Dicionário de Dados					
Produtos					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Identificador único do produto	12
nome	varChar	255 bytes		Nome do produto adquirido	Coca-Cola
preço	float	4 bytes		Valor atribuído para o produto	R$ 7,00
imagem	varChar	255 bytes		Imagem do produto	http://res.cloudinary.com/exemplolink/image/upload/exemplolink
descrição	varChar	255 bytes		Descrição que explica qual é o produto	Lata de Coca-Cola 250ml
created_at	timeStamp(3)	8 bytes		Dia e hora da criação do produto	2025-06-23 11:41:19
updated_at	timeStamp(3)	8 bytes		Dia e hora da última atualização do produto	2025-06-24 12:33:20
categoria_id	Int	32 bytes	FK	Número identificador da categoria do produto	3

Fonte: Autoral.

A Tabela 11 descreve a entidade *Ingredientes*, que reúne os componentes utilizados na composição dos produtos. A modelagem permite maior precisão no controle interno de estoque, personalização de pedidos e padronização do preparo dos itens.

Tabela 11 - Dicionário de Dados da entidade Ingredientes.

Dicionário de Dados					
Ingredientes					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor

id	Int	32 bytes	PK	Número de Identificação dos ingredientes	1
nome	varChar	255 bytes		Nome atribuído para os ingredientes dos produtos	Queijo
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do ingrediente	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45

Fonte: Autoral.

A Tabela 12 apresenta a relação entre *Ingredientes* e *Produtos*, definindo quais itens compõem cada produto do cardápio. Essa entidade garante consistência na montagem das pizzas e possibilita ajustes dinâmicos, como remoção ou substituição de ingredientes a pedido do cliente.

Tabela 12 - Dicionário de Dados da entidade Ingredientes_produtos.

Dicionário de Dados					
Ingredientes_produtos					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de Identificação dos ingredientes_produtos	1
created_at	timeStamp(3)	8 bytes		Hora e dia da criação dos ingredientes_produtos	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45
produto_id	Int	32 bytes	FK	Número de identficação do produto	2
Ingrediente_id	Int	32 bytes	FK	Número de identificação do ingrediente	5

Fonte: Autoral.

A Tabela 13 aborda a entidade *Itens*, que representa cada produto individual dentro de um pedido. Ela é essencial para o fluxo de compra, permitindo registrar quantidade, preço unitário, valor final e observações específicas. Sua estrutura sustenta cálculos e operações do carrinho de compras.

Tabela 13 - Dicionário de Dados da entidade Itens.

Dicionário de Dados					
Itens					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de identificação dos itens	30
quantidade	smallInt	2 bytes		Número de itens pedidos	2
observação	varChar	255 bytes		um apontamento para o item	pizza bem recheada
status	varChar	256 bytes		mostra em que estágio de preparo está o item	Em Preparo
itens_adicionais_id	Int	32 bytes	FK	ID do itens_adicionais	12
itens_ingredientes_id	Int	32 bytes	FK	ID do itens_ingredientes	22
pedido_id	Int	32 bytes	FK	ID do pedido	553
produto_id	Int	32 bytes	FK	ID do produto	16
created_at	timeStamp(3)	8 bytes		Dia e hora da criação dos itens	2025-06-23 11:41:19
updated_at	timeStamp(3)	8 bytes		Dia e hora da última atualização do item	2025-06-24 12:33:20

Fonte: Autoral.

A Tabela 14 detalha a relação entre *Itens* e *Adicionais*, registrando todos os complementos selecionados pelo cliente para cada produto. Essa entidade assegura precisão no preço final e transparência na personalização do pedido.

Tabela 14 - Dicionário de Dados da entidade Itens_adicionais.

Dicionário de Dados					
Itens_adicionais					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de Identificação dos Itens_adicionais	1
adicionado	boolean	1 byte		Deseja adicionar esse adicional?	0
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do itens_adicionais	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45
categorias_adicionais_id	Int	32 bytes	FK	Número de identificação das categorias dos adicionais	23

Fonte: Autoral.

A Tabela 15 descreve a relação entre *Itens* e *Ingredientes*, permitindo identificar ingredientes removidos, substituídos ou ajustados pelo cliente. Essa tabela é especialmente útil para pedidos personalizados e garante que a cozinha receba instruções claras para montagem.

Tabela 15 - Dicionário de Dados da entidade Itens_ingredientes.

Dicionário de Dados					
Itens_ingredientes					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de Identificação dos Itens_ingredientes	1
adicionado	boolean	1 byte		Deseja adicionar esse ingrediente?	0
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do itens_ingredientes	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45
ingredientes_produto_id	Int	32 bytes	FK	Número de identificação dos ingredientes do produto	15

Fonte: Autoral.

A Tabela 16 documenta a entidade Pedido (*Order)*, que representa o pedido completo do cliente. Contempla informações como status, horário, valor total e vínculo com a mesa e o pagamento. É o núcleo do sistema, conectando praticamente todas as demais entidades e garantindo o fluxo operacional da pizzaria.

Tabela 16 - Dicionário de Dados da entidade Pedido *(Order).*

Dicionário de Dados					
Pedido					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de identificação do pedido	1

mesa_id	Int	32 bytes	FK	Id da mesa	14
status	boolean	1 byte		O pedido está pronto?	1
entrega	boolean	1 byte		O pedido já foi enviado?	0
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do pedido	2025-02-11 06:21:19
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização do pedido	2025-02-13 08:11:20
cliente_id	Int	32 bytes	FK	Id do usuário	1

Fonte: Autoral.

A Tabela 17 apresenta a entidade *Mesas*, que controla os espaços físicos da pizzaria e sua vinculação aos pedidos. Essa estrutura é fundamental para o funcionamento do QR Code, permitindo que o cliente seja associado automaticamente a uma mesa ao iniciar um pedido.

Tabela 17 - Dicionário de Dados da entidade Mesas.

Dicionário de Dados					
Mesa					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de Identificação das mesas	1
numero	Int	32 bytes		Número atribuido para as mesas	13
created_at	timeStamp(3)	8 bytes		Hora e dia da criação da mesa	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45

Fonte: Autoral.

A Tabela 18 descreve a entidade *Pagamento*, responsável pelo registro das formas e status dos pagamentos realizados pelos clientes. Sua modelagem assegura organização contábil, segurança nas transações e acompanhamento do fluxo financeiro da pizzaria.

Tabela 18 - Dicionário de Dados da entidade Pagamento.

Dicionário de Dados					
Pagamento					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de identificação do pagamento	7
pago	boolean	1 byte		Status do pagamento, verifica se já foi pago ou não	1
mtdo_pagto_id	Int	32 bytes	FK	Número de identficação do método de pagamento	2
pedido_id	Int	32 bytes	FK	Número do pedido para fazer o pagamento	5
created_at	timeStamp(3)	8 bytes		Dia e hora da criação dos itens	2025-08-19 11:41:19
updated_at	timeStamp(3)	8 bytes		Dia e hora da última atualização do item	2025-08-19 21:23:54

Fonte: Autoral.

A Tabela 19 apresenta os *Métodos de Pagamento* disponíveis no sistema, como cartão, PIX ou dinheiro. Ela define quais opções podem ser selecionadas pelo cliente e sustenta a lógica de checkout dentro do aplicativo.

Tabela 19 - Dicionário de Dados da entidade Metodos_pagamentos.

Dicionário de Dados					
Metodos_pagamentos					
Nome	Tipo	Tamanho	Pk/Fk	Descrição	Valor
id	Int	32 bytes	PK	Número de Identificação do método de pagamento	1
nome	varChar	255 bytes		Nome do método de pagamento	Crédito
created_at	timeStamp(3)	8 bytes		Hora e dia da criação do método de pagamento	2025-04-11 17:54:23
updated_at	timeStamp(3)	8 bytes		Hora e dia da última atualização	2025-04-11 18:32:45

Fonte: Autoral.

4.6.7 Prisma ORM

O Prisma ORM foi adotado como ferramenta de acesso aos dados devido ao seu modelo de tipagem estática, sua integração nativa com Node.js e suas ferramentas de migração automática. Ele traduz a estrutura do banco para modelos tipados, eliminando erros comuns de digitação ou inconsistências entre camadas. O uso do Prisma aumentou a produtividade no desenvolvimento do *backend*, além de garantir maior segurança e consistência nas operações com o banco.

4.6.8 Hospedagem via Supabase

A camada de banco de dados foi hospedada no Supabase, plataforma que fornece infraestrutura gerenciada para PostgreSQL, com recursos como backups automáticos e painéis de monitoramento. No contexto da aplicação, a adoção do Supabase eliminou a necessidade de configurações manuais de servidores e permitiu a visualização da entrada e saída de dados pelos integrantes do grupo simultaneamente, aumentando a segurança e facilitando o gerenciamento do ambiente de produção.

4.6.9 Justificativa Técnica da Solução Adotada

A combinação entre PostgreSQL, Prisma ORM e a infraestrutura do Supabase fortalece o sistema ao unir confiabilidade em cenários de alta concorrência, produtividade no desenvolvimento com redução de erros, escalabilidade conforme o crescimento da demanda, manutenibilidade por meio de uma arquitetura organizada e segurança avançada com controle de acesso e isolamento de ambientes. Juntos, esses elementos garantem que a aplicação atenda tanto aos requisitos funcionais quanto aos não funcionais, mantendo qualidade e permitindo expansão futura.

4.7 Cronograma Realizado

O cronograma realizado do projeto reflete a execução prática das tarefas distribuídas entre as semanas de desenvolvimento, evidenciando o ritmo de trabalho da equipe e a evolução progressiva das entregas. Ao longo de onze semanas, os integrantes concluíram atividades relacionadas à configuração inicial do ambiente, implementação de funcionalidades essenciais, desenvolvimento do aplicativo *mobile*, criação de telas, integração com o banco de dados e finalização da documentação.

A equipe organizou-se em sprints que concentraram esforços em etapas específicas, como construção das principais telas do aplicativo (Login, Cardápio, Perfil, Configurações, Carrinho), implementação de mecanismos de busca, gerenciamento de produtos, desenvolvimento do fluxo de pedidos, funcionalidades de pagamento, histórico, avaliação e acompanhamento do pedido. Também foram realizados períodos destinados exclusivamente ao refinamento da interface (Figma), implementação de adicionais e ingredientes, e à elaboração dos documentos técnicos.

A Tabela 20 ilustra o cronograma realizado das atividades do grupo conforme o mês correspondente e as tarefas das sprints.

Tabela 20 – Cronograma das tarefas realizadas

	Setembro	Outubro	Novembro
Configuração de ambiente	X		
Configurar banco de dados	X		
Tela de Login	X		
Botão de Navegação	X		
Término Tela de Login	X		
Inserir produtos no database	X		
Cardápio	X		
Escolha de produtos	X		
Observações do pedido	X		
Categoria no DB	X		
Tela de Configurações	X		
Perfil do Usuário	X		
Barra de Pesquisa		X	
Carrinho		X	
Andamento do pedido		X	
Pagamentos		X	
Avaliação do pedido		X	
Histórico de pedidos		X	
Documentação			X
Figma			X
Adicionais			X
Ingredientes			X

Fonte: Autoral.

4.8 Metodologia Scrum

Durante o desenvolvimento do projeto, adotou-se a metodologia ágil Scrum como base para organização das tarefas, acompanhamento da equipe e definição das entregas. O uso dessa abordagem permitiu estruturar o trabalho de forma iterativa e incremental, possibilitando adaptações contínuas conforme as necessidades surgiam ao longo do processo. Segundo Schwaber e Sutherland (2020), o Scrum oferece um

framework leve, porém eficaz, para lidar com ambientes complexos, promovendo transparência, inspeção e adaptação em ciclos curtos de desenvolvimento.

A equipe estruturou suas atividades em *sprints* semanais, cada uma com objetivos claros e metas específicas, definidas a partir do *backlog* priorizado pelo *Product Owner*. No início de cada ciclo, eram planejadas as tarefas a serem desenvolvidas, distribuídas entre os membros conforme suas habilidades e responsabilidades. Ao final de cada *sprint*, realizava-se uma revisão rápida das entregas, avaliando o que foi concluído, o que precisava ser ajustado e quais elementos deveriam ser priorizados para o próximo período.

Além disso, foram realizadas reuniões frequentes de acompanhamento, que funcionaram como momentos de alinhamento sobre dificuldades, progresso individual e soluções colaborativas para obstáculos técnicos ou organizacionais. Essa dinâmica fomentou comunicação constante, facilitou o entendimento coletivos dos requisitos e garantiu que as funcionalidades fossem desenvolvidas de forma integrada e coerente com a visão geral do projeto.

Por meio da aplicação do Scrum, o desenvolvimento da aplicação ocorreu de maneira organizada, flexível e orientada à entrega contínua de valor, permitindo aprimorar o produto ao longo do tempo e assegurar que o resultado final atendesse aos requisitos levantados e às expectativas dos usuários.

4.9 Metodologia Kanban

Além do uso do Scrum, o projeto também incorporou práticas do método Kanban para aprimorar o fluxo de trabalho e manter a equipe alinhada quanto ao andamento das atividades. O Kanban, segundo Anderson (2010), é uma abordagem visual de gestão que permite controlar processos por meio de quadros organizados em etapas, facilitando a identificação de gargalos, o monitoramento do progresso e a priorização contínua das tarefas.

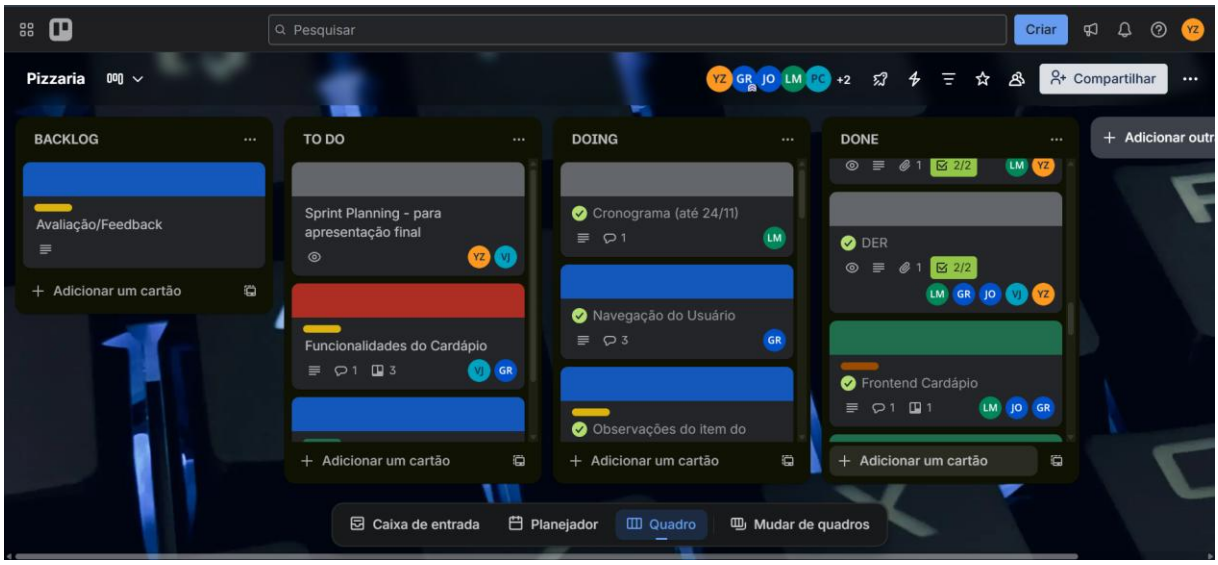
Durante o desenvolvimento, a equipe utilizou quadros digitais, por meio do Trello, divididos em colunas representando os estágios do fluxo de produção, como *To Do*, *Doing* e *Done*. Em junção com a metodologia Scrum, foi possível adicionar o quadro *Backlog.* Cada tarefa era representada por um cartão que se movia ao longo das colunas à medida que avançava. Esse formato proporcionou uma visão clara e

imediata do estado atual do desenvolvimento, permitindo que todos os envolvidos acompanhassem simultaneamente o andamento do projeto.

A aplicação do Kanban favoreceu a autonomia dos integrantes, permitiu ajustes rápidos no planejamento diário e facilitou a identificação de prioridades emergentes. Além disso, a equipe pôde visualizar facilmente tarefas acumuladas, evitando sobrecarga de trabalho e garantindo que as entregas fossem distribuídas de maneira equilibrada. Essa prática tornou o fluxo de trabalho mais estável, previsível e eficiente, complementando os ciclos iterativos definidos pelo Scrum e contribuindo diretamente para a organização e conclusão das funcionalidades do *Atende+*.

A Figura 03 ilustra a metodologia Kanban por meio do aplicativo trello.

Figura 3 – Metodologia Kanban com Trello



Fonte: Autoral.

4.10 Tecnologias Utilizadas

A implementação do sistema exigiu a seleção de tecnologias modernas, estáveis e coerentes com os requisitos funcionais e não funcionais do projeto. As escolhas consideraram desempenho, escalabilidade, segurança, facilidade de manutenção, produtividade da equipe e compatibilidade entre as camadas de software. Esta seção apresenta detalhadamente as tecnologias empregadas, suas funções dentro da solução e as justificativas técnicas para sua utilização.

4.10.1 Ambiente de Desenvolvimento Móvel (*frontend*)

O aplicativo móvel foi construído utilizando React Native integrado ao ecossistema Expo. Essa combinação permitiu o desenvolvimento multiplataforma por meio de uma única base de código, resultando em maior eficiência e menor custo de manutenção. O Expo foi empregado para facilitar o acesso a funcionalidades nativas, agilizando processos de build, testes e execução em ambiente real.

Foram utilizados módulos do Expo tais como *expo-camera* e *expo-barcode-scanner*, responsáveis pela captura de imagens e leitura de QR Codes; *expo-font*, utilizado para carregamento de fontes personalizadas; além de outras extensões que simplificaram a integração de recursos nativos.

A navegação do aplicativo foi estruturada com o uso da biblioteca React Navigation, responsável pela organização das rotas e transições entre telas. Para armazenamento local de dados leves, como *tokens* e preferências do usuário, utilizou-se o AsyncStorage. Bibliotecas como *react-native-svg*, *react-native-qrcode-svg* e *react-native-safe-area-context* complementaram a camada visual, oferecendo renderização vetorial, geração de QR Codes e adequação da interface às áreas seguras dos dispositivos.

4.10.2 Backend e Lógica de Aplicação (API)

A API responsável pela lógica do sistema foi implementada utilizando Node.js em conjunto com o *framework* Express. Essa estrutura permitiu organizar rotas REST, middlewares, validações e tratamentos de exceções, promovendo flexibilidade e alto desempenho operacional.

Para autenticação, foi adotado o padrão JWT (*jsonwebtoken*), que possibilita controle de acesso baseado em *tokens*. O armazenamento seguro de senhas foi realizado com *bcryptjs*, garantindo criptografia robusta. Variáveis sensíveis do ambiente do servidor foram gerenciadas com *dotenv*, assegurando boa organização e proteção dos dados de configuração.

O sistema também empregou middlewares como Multer e *express-fileupload* para tratamento de arquivos enviados ao servidor, especialmente imagens associadas a produtos. O pacote *express-async-errors* foi utilizado para aprimorar o tratamento de exceções assíncronas. Para comunicação entre o aplicativo e a API, o Axios foi

empregado como cliente HTTP, permitindo padronização dos cabeçalhos e interceptação de requisições.

4.10.3 Persistência e Serviços de Dados

A camada de persistência utiliza o SGBD PostgreSQL, hospedado na plataforma Supabase. Essa solução garante integridade transacional, suporte a tipos de dados avançados e ferramentas administrativas que facilitam a manutenção do banco durante o desenvolvimento.

O acesso ao banco é mediado pelo Prisma ORM, ferramenta que fornece modelos tipados, migrações versionadas, consultas seguras e abstração de SQL. O uso do Prisma contribuiu para a padronização do código, evitando inconsistências comuns em operações manuais.

O armazenamento de imagens de produtos foi realizado utilizando o serviço Cloudinary, que disponibiliza entrega via CDN e otimizações automáticas de carregamento, reduzindo a carga sobre o servidor e melhorando o desempenho das interfaces.

4.10.4 Ferramentas de Apoio

Ferramentas de teste de API, como Insomnia, foram utilizadas no processo de validação dos endpoints, permitindo simular requisições e verificar a coerência das respostas com a lógica implementada.

No ambiente administrativo, desenvolvido parcialmente com Next.js, foi utilizada a biblioteca *cookies-next* para manipulação de cookies, assegurando controle adequado de sessão e autenticação nesse ambiente.

4.10.5 Justificativa das Escolhas Tecnológicas

As tecnologias selecionadas foram baseadas em critérios de desempenho, segurança e adequação ao escopo do sistema. O uso de React Native e Expo proporcionou desenvolvimento eficiente e experiência nativa ao usuário. O Node.js com Express ofereceu leveza e escalabilidade à API. O PostgreSQL, mediado pelo Prisma ORM, garantiu integridade dos dados e facilidade de manutenção. Tecnologias como Cloudinary, Axios e Git reforçaram práticas modernas de desenvolvimento, promovendo organização e confiabilidade.

A uniformidade da stack baseada em JavaScript contribuiu para maior produtividade, uma vez que permitiu que o mesmo conjunto de conhecimentos fosse aplicado em diferentes partes do sistema. O uso de serviços externos e soluções em nuvem possibilitou escalabilidade e redução de dependências locais.

4.11 Linguagens de Programação

As linguagens de programação utilizadas no desenvolvimento do aplicativo foram selecionadas considerando critérios de desempenho, compatibilidade entre camadas, facilidade de manutenção e adequação ao modelo arquitetural adotado. A escolha dessas linguagens permitiu a criação de uma solução coesa, distribuída e escalável, além de favorecer a produtividade da equipe durante o processo de desenvolvimento. Esta seção apresenta a descrição das linguagens utilizadas, bem como suas aplicações específicas dentro do sistema.

4.11.1 JavaScript

A principal linguagem empregada no projeto é o JavaScript, utilizada tanto no desenvolvimento do aplicativo móvel quanto na implementação da API. Essa escolha proporcionou unificação da stack tecnológica, permitindo o compartilhamento de conceitos entre *front-end* e *back-end*, além de reduzir o esforço cognitivo da equipe.

No *front-end,* o JavaScript atua na construção das interfaces, manipulação de estados, navegação entre telas e integração com APIs. No *backend*, é responsável pela estruturação das rotas, controle da lógica de negócio, gerenciamento de middlewares, integração com o banco de dados por meio do Prisma e tratamento das requisições recebidas pelo servidor.

A ampla adoção do JavaScript no ecossistema de desenvolvimento moderno, somada à extensa disponibilidade de bibliotecas e suporte comunitário, contribui para a manutenção contínua do sistema e facilita a evolução futura da aplicação.

4.11.2 TypeScript

Em módulos específicos, especialmente em partes administrativas, componentes mais complexos e na manipulação de dados estruturados, foi utilizado TypeScript, um superconjunto do JavaScript que introduz tipagem estática. Por meio do TypeScript, tornou-se o código mais seguro, robusto e fácil de manter,

principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

Essas características tornam o código mais seguro, robusto e fácil de manter, principalmente em componentes que lidam com dados sensíveis, autenticação ou integrações externas.

4.11.3 SQL

A camada de persistência utiliza SQL por meio do SGBD PostgreSQL. Mesmo com a adoção do Prisma ORM, que abstrai consultas e operações, o SQL permanece como base fundamental para definição e organização das estruturas do banco. A adoção do SQL garantiu precisão, confiabilidade e padronização no armazenamento e resgate das informações essenciais ao funcionamento do sistema, especialmente no que diz respeito a pedidos, produtos, categorias e usuários.

4.11.4 Justificativa das Escolhas

A combinação entre JavaScript, TypeScript e SQL permitiu o desenvolvimento de uma solução coerente e tecnicamente eficiente. A unificação da *stack* em torno do JavaScript reduziu a complexidade do projeto e aumentou a produtividade da equipe. O uso de TypeScript assegurou maior qualidade e previsibilidade do código nas partes críticas do sistema. Já o SQL forneceu uma base sólida para garantir integridade e confiabilidade dos dados.

Dessa forma, a seleção das linguagens de programação adotadas atende plenamente aos requisitos arquiteturais, funcionais e operacionais, oferecendo suporte tanto para o cenário atual quanto para possíveis expansões futuras.

4.12 Frameworks

Os *frameworks* utilizados no desenvolvimento do app desempenham papel fundamental na organização da arquitetura, na produtividade da equipe e na robustez da aplicação. A escolha dessas ferramentas buscou garantir padronização, modularidade, reutilização de componentes e simplificação de operações críticas

tanto no *front-end* quanto no *back-end*. Nesta seção são descritos os principais *frameworks* empregados, suas aplicações e a justificativa técnica de sua adoção.

4.12.1 React Native

O React Native foi adotado como *framework* principal para o desenvolvimento do aplicativo móvel. Baseado em componentes reutilizáveis, permite construir interfaces nativas utilizando JavaScript e JSX, mantendo alto desempenho e estética compatível com plataformas Android e iOS. Sua estrutura declarativa favorece a legibilidade, a manutenção e a escalabilidade da interface, características essenciais para o aplicativo de autoatendimento.

4.12.2 Expo

O Expo foi utilizado como camada intermediária para gestão de recursos nativos dentro do React Native, simplificando o processo de desenvolvimento e implantação. A utilização do Expo acelerou significativamente o ciclo de desenvolvimento, eliminando configurações manuais complexas e proporcionando testes contínuos em dispositivos reais.

4.12.3 Express.js

No *backend*, o *framework* Express.js foi empregado para estruturar a API REST que conecta o aplicativo ao banco de dados. Reconhecido por sua leveza e flexibilidade, o Express possibilita a criação de rotas organizadas, controle de middlewares e tratamento centralizado de requisições. O uso do Express permitiu à equipe manter uma API modular, escalável e de fácil manutenção, atendendo às necessidades do sistema de pedidos, autenticação e gerenciamento de produtos.

4.12.4 Prisma ORM

O Prisma foi o *framework* adotado para manipulação do banco de dados PostgreSQL. Ele fornece mapeamento objeto-relacional moderno, tipado e seguro. Sua utilização reduziu erros comuns na manipulação manual de SQL e tornou a integração entre a API e a camada de dados mais consistente e produtiva.

4.12.5 Next.js (Painel Administrativo)

Para o desenvolvimento de interfaces administrativas, o *framework* Next.js foi utilizado devido à sua capacidade de gerenciar rotas, lidar com renderização híbrida e oferecer boa performance em *dashboards* com carregamento dinâmico. Sua estrutura baseada em React facilita a reutilização de componentes e a padronização visual do sistema.

4.12.6 Justificativa das Escolhas

Os frameworks selecionados oferecem uma solução moderna, modular e de fácil manutenção, em que cada tecnologia cumpre um papel claro dentro da arquitetura: React Native + Expo garantem uma experiência fluida e multiplataforma, Express.js organiza a lógica de negócio com segurança, Prisma ORM assegura integridade e simplicidade no acesso aos dados e Next.js estrutura o ambiente administrativo com eficiência e escalabilidade. Essa combinação facilita integrações futuras, reduz custos de manutenção e reforça boas práticas de engenharia de software, assegurando a continuidade do projeto.

4.13 Arquitetura do Sistema

A arquitetura do sistema foi projetada com base em princípios de modularidade, escalabilidade, segurança e manutenibilidade. A solução adotada segue o modelo de arquitetura em camadas, distribuída entre aplicação móvel (*front-end*), API (*back-end*), serviços externos e banco de dados relacional. Essa abordagem facilita a separação de responsabilidades, reduz o acoplamento entre os componentes e permite evolução incremental do projeto de forma organizada.

A arquitetura pode ser descrita como **Cliente–Servidor distribuída**, reforçada por módulos especializados para autenticação, armazenamento e comunicação entre serviços. A seguir, são detalhados os elementos que compõem a arquitetura, bem como os fluxos de comunicação, tecnologias integradas e justificativas técnicas que fundamentam seu desenho estrutural.

4.13.1 Visão Geral da Arquitetura

A aplicação é composta por três camadas principais:

- **Camada de Apresentação (*Front-End* / *Mobile*)**
    a. Desenvolvida em **React Native** com o ecossistema **Expo**.
    b. Responsável pela interação direta com o usuário.
    c. Consome APIs REST do *backend*.
    d. Gerencia navegação, validações e armazenamento local via *Async Storage.*
- **Camada Lógica (*Back-End* / API REST)**
    e. Construída utilizando **Node.js** com **Express.js**.
    f. Implementa regras de negócio, validações, tratamento de erros e autenticação.
    g. Utiliza **Prisma ORM** para acesso ao banco de dados.
    h. Segue uma estrutura modular baseada em rotas, *controllers*, middlewares e serviços.
- **Camada de Dados (Banco de Dados e Armazenamento)**
    i. Utiliza **PostgreSQL** hospedado no **Supabase**.
    j. Armazena entidades principais: *costumers*, *products*, *orders*, *payments*, *tables*, etc.
    k. Imagens são armazenadas no **Cloudinary**, reduzindo carga do *backend*.

4.13.2 Fluxo Geral de Comunicação

1. O usuário interage com o aplicativo *mobile*.
2. O aplicativo envia requisições HTTP (GET, POST, PUT, DELETE) para a API.
3. O *back-end* processa a requisição, aplica regras de negócio e acessa o banco.
4. O *back-end* retorna dados em formato JSON para o cliente.
5. O aplicativo apresenta as informações ao usuário com base no retorno.

Este ciclo ocorre de forma rápida, garantindo uma experiência fluida.

4.13.3 Arquitetura da API

A API segue o padrão ***RESTful***, garantindo padronização e fácil integração. Estrutura modular:

- **Routes** — definem *endpoints*.
- **Controllers** — processam requisições e respostas.
- **Services** — implementam regras de negócio específicas.
- ***Middlewares*** — filtros e verificações, como:
    - Autenticação (JWT)
    - Autorização
    - Validação de dados
    - Upload de arquivos (Multer/express-fileupload)
    - CORS
    - Tratamento de erros (express-async-errors)

4.13.4 Segurança e Autenticação

Para garantir a proteção dos dados e identificar usuários, o sistema utiliza:

- **JWT (JSON Web Token)** para rotas privadas.
- **bcryptjs** para criptografar senhas.
- **Variáveis de ambiente (.env)** para proteger informações sensíveis.
- **CORS** para liberar somente origens autorizadas.

4.13.5 Banco de Dados e Persistência

O banco segue modelo relacional, com:

- Tabelas normalizadas
- Relacionamentos 1:N, 1:1 e 1:0
- Mapeamento através do **Prisma ORM**, com:
    - Migrações automáticas
    - Tipagem de consultas
    - Integração direta com o Supabase

Imagens são armazenadas no **Cloudinary**, retornando apenas URLs ao banco.

4.13.6 Armazenamento Local e Sessões

No aplicativo *mobile*, o **Async Storage** é utilizado para:

- Guardar o token JWT
- Manter sessão ativa
- Armazenar preferências e dados temporários

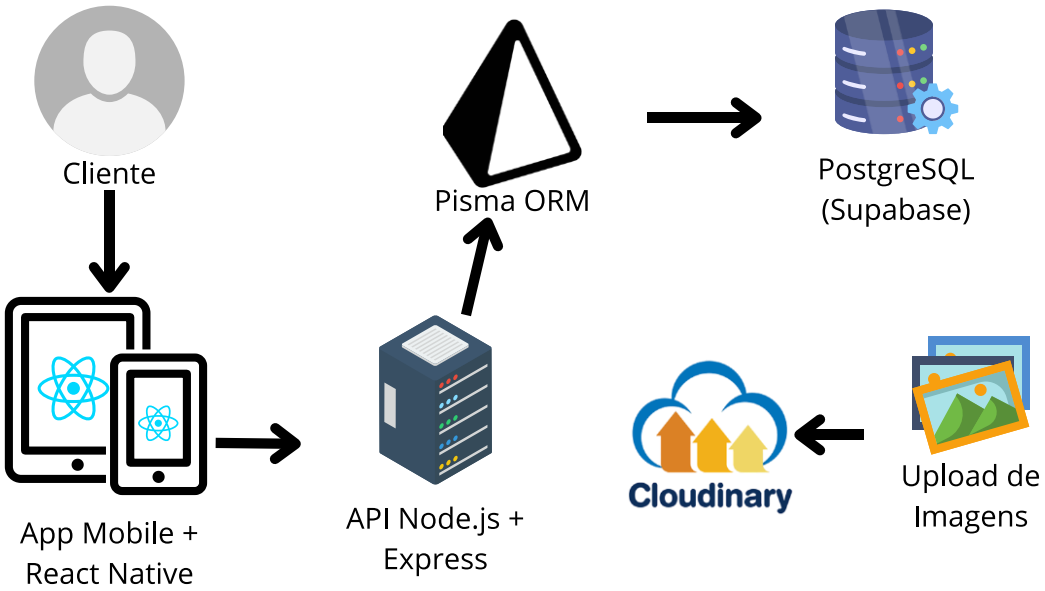
4.13.7 Arquitetura de Deploy e Hospedagem

- ***Back-end*** pode ser hospedado em plataformas como Render, Vercel (via *serverless*) ou Railway.
- **Banco de Dados** hospedado no Supabase.
- **Aplicativo *Mobile*** distribuído via Expo (*builds* nativas para Android).
- **Imagens** hospedadas no Cloudinary.

4.13.8 Diagrama Simplificado da Arquitetura

A Figura 4 ilustra o diagrama simplificado da arquitetura do sistema.

Figura 4 – Diagrama simplificado da arquitetura.



Fonte: Autoral.

4.13.9 Benefícios da Arquitetura Adotada

A adoção da arquitetura proposta neste projeto proporciona um conjunto significativo de vantagens que impactam diretamente a eficiência, a organização e o desempenho do sistema. Ao estruturar a aplicação em camadas independentes e bem definidas, torna-se possível otimizar o fluxo de desenvolvimento, melhorar a manutenção do código e garantir maior confiabilidade na comunicação entre os componentes. Além disso, essa abordagem reforça boas práticas de desenvolvimento de software ao promover segurança, padronização e capacidade de expansão, aspectos essenciais para a evolução contínua do sistema e para sua adequação em ambientes reais de produção.

4.14 Controle de Versionamento

O controle de versionamento foi um componente fundamental no desenvolvimento do projeto, atuando como ferramenta de gestão das alterações realizadas no código e garantindo organização, rastreabilidade e segurança durante todo o ciclo de construção do sistema. Conforme destacam Pressman e Maxim (2016), sistemas de versionamento permitem registrar cada modificação, facilitando a colaboração entre os desenvolvedores, a identificação de erros e a recuperação de versões anteriores sempre que necessário. Para Sommerville (2019), esse mecanismo é indispensável em ambientes de desenvolvimento modernos, pois reduz riscos, evita perdas de trabalho e assegura que múltiplos integrantes possam contribuir simultaneamente sem comprometer a integridade do projeto.

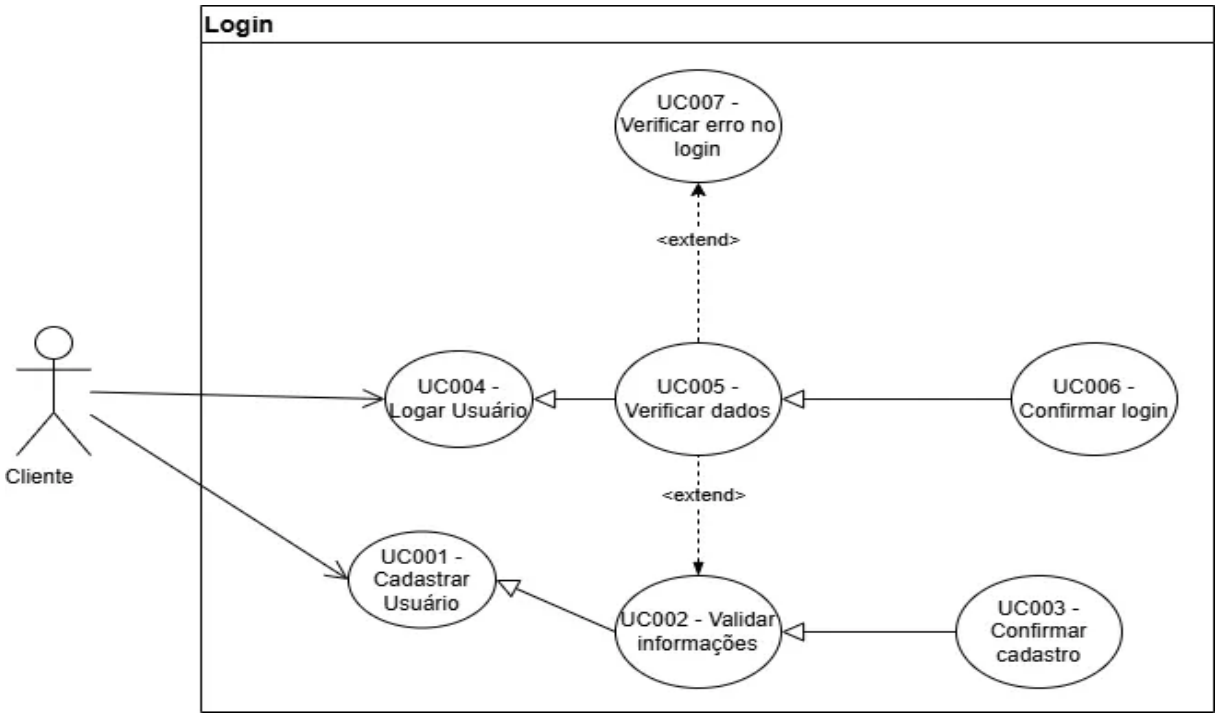
Nesse contexto, o GitHub foi utilizado como plataforma principal para controle de versionamento, permitindo gerenciar *commits*, *branches*, *merges* e revisões de forma estruturada. Essa prática possibilitou acompanhar a evolução do código de maneira transparente, facilitando a integração contínua e promovendo maior qualidade nas entregas. Assim, o controle de versionamento consolidou-se como um elemento essencial para a organização do desenvolvimento e para a manutenção da consistência técnica do projeto.

4.15 Fluxograma e Diagramas de Casos de Uso

Para complementar a compreensão do funcionamento interno do sistema e evidenciar de forma visual os processos que estruturam o fluxo de atendimento do App, este tópico apresenta os diagramas desenvolvidos ao longo do projeto. Os anexos aqui reunidos representam, de maneira clara e organizada, etapas essenciais como login, atendimento, seleção de produtos, pagamento e interação do cliente com o sistema por meio do autoatendimento. Essas representações visuais reforçam a coerência entre os requisitos levantados, a modelagem do banco de dados e a implementação final, permitindo uma análise mais precisa da lógica operacional adotada.

A Figura 5 ilustra os Casos de Uso no contexto de Login do usuário.

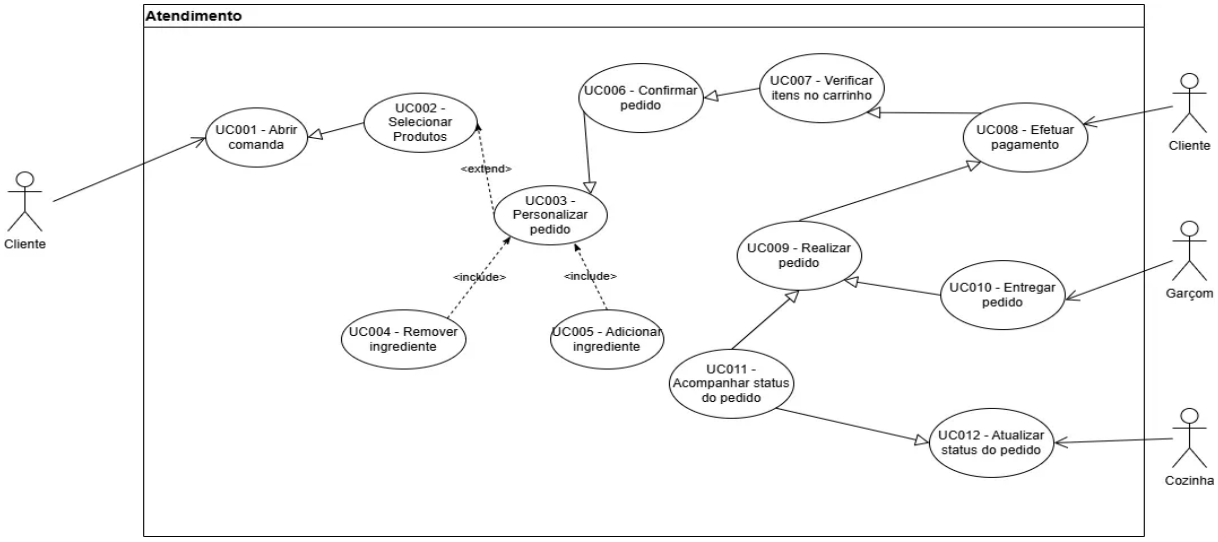
Figura 5 – Diagrama de Caso de Uso Login



Fonte: Autoral.

A Figura 6 ilustra os Casos de Uso no contexto de Atendimento do usuário.

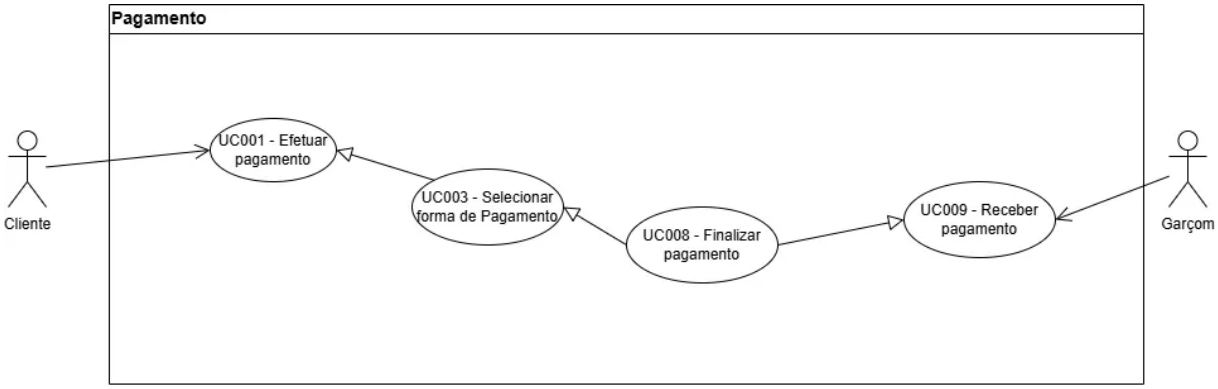
Figura 6 – Diagrama de Caso de Uso Atendimento



Fonte: Autoral.

A Figura 7 ilustra os Casos de Uso no contexto de Pagamento do usuário.

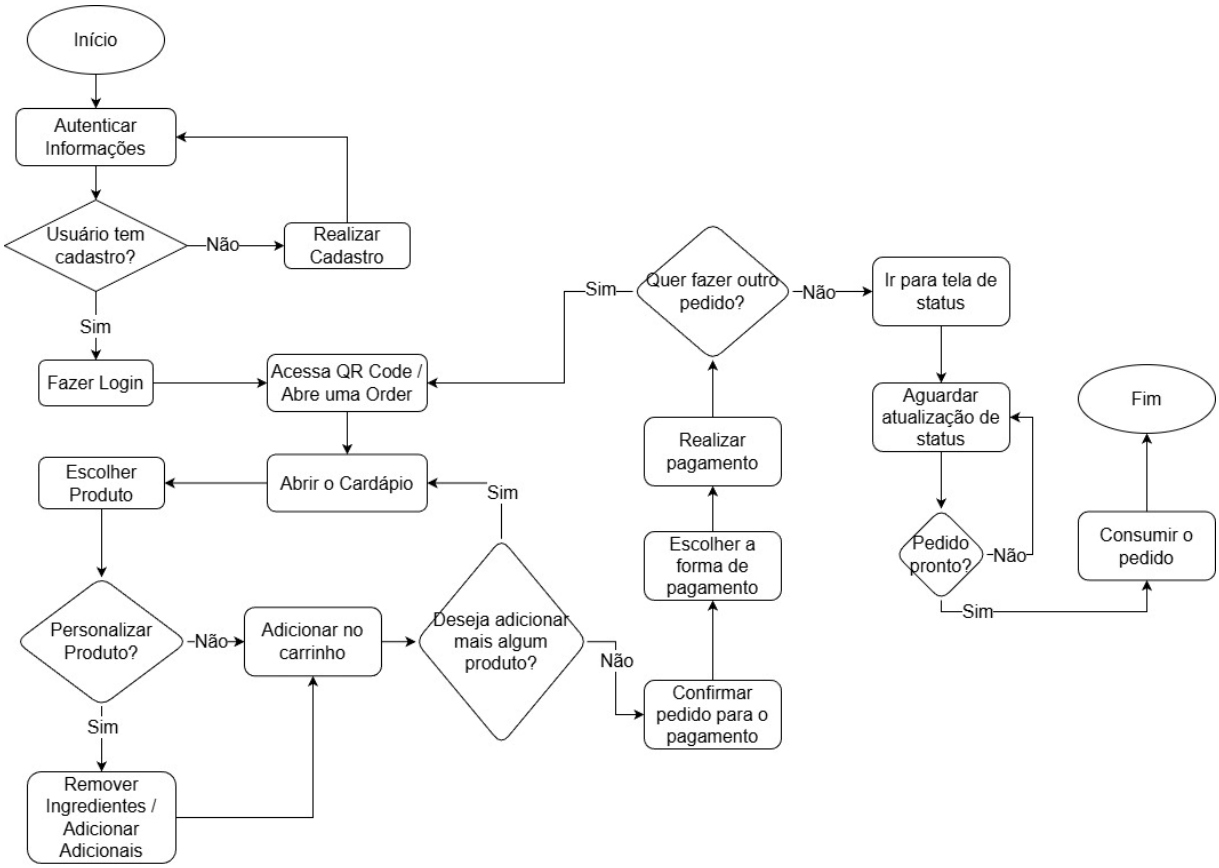
Figura 7 – Diagrama de Caso de Uso Pagamento



Fonte: Autoral.

A Figura 08 ilustra o fluxo geral do cliente durante toda a aplicação.

Figura 8 – Fluxograma do cliente no app



Fonte: Autoral.

4.16 Wireframe das Telas

Nesta seção, são apresentados *prints* do fluxo do projeto incluindo as principais funcionalidades com a finalidade de demonstrar de forma prática as tecnologias do projeto.

As figuras 9 e 10 apresentam a criação de um novo cadastro e a tela principal do login.

Figura 9 – Tela de Cadastro

Fonte: Autoral.

Figura 10 – Tela de Login

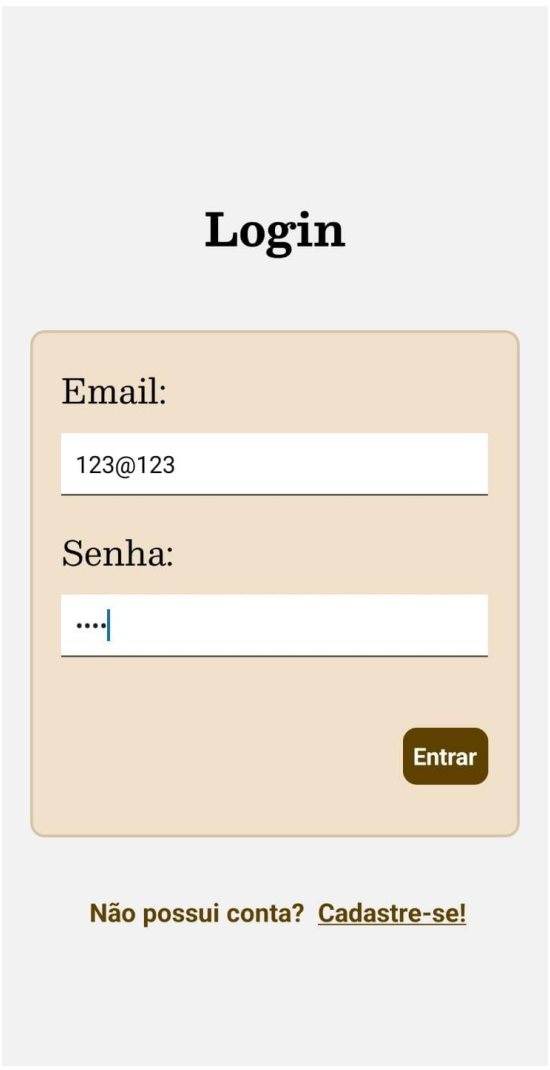
Fonte: Autoral.

As figuras 11 e 12 apresentam uma mensagem de erro caso o usuário tenta logar com uma conta não existente e uma representação fictícia do preenchimento de dados do usuário.

Figura 11 – Tela de alert de erro ao logar

Fonte: Autoral.

Figura 12 – Tela de Login funcional



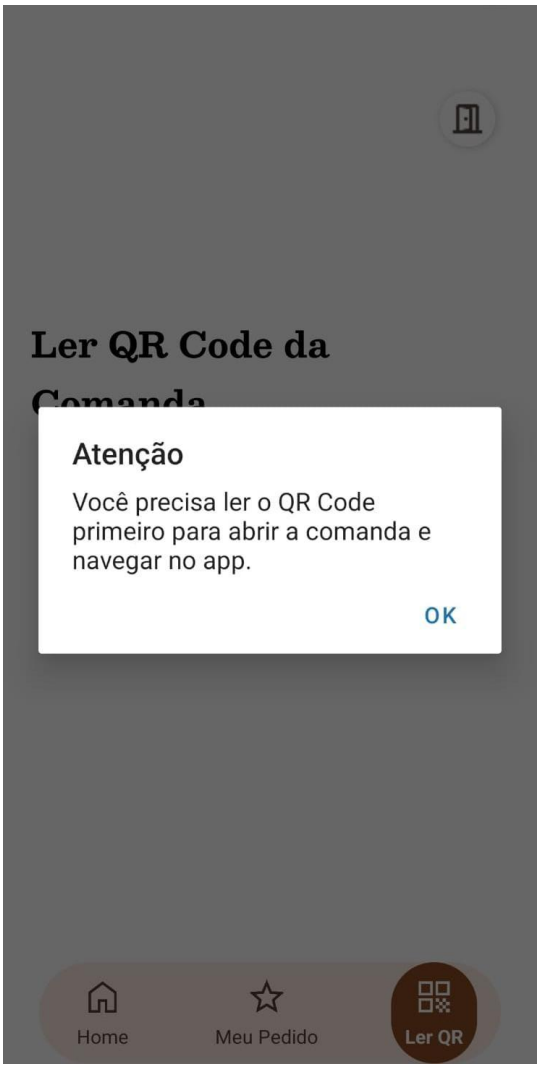
Fonte: Autoral.

A figura 13 apresenta a tela inicial do projeto, onde o cliente pode e deve fazer a leitura do QR Code da mesa onde está sentado. A figura 14 apresenta um *alert* caso o cliente tente navegar para a tela do cardápio sem estar com uma comanda vinculada à uma mesa.

Figura 13 – Tela para leitura do QR Code

Fonte: Autoral.

Figura 14 – Tela com alert de erro por não ter comanda aberta



Fonte: Autoral.

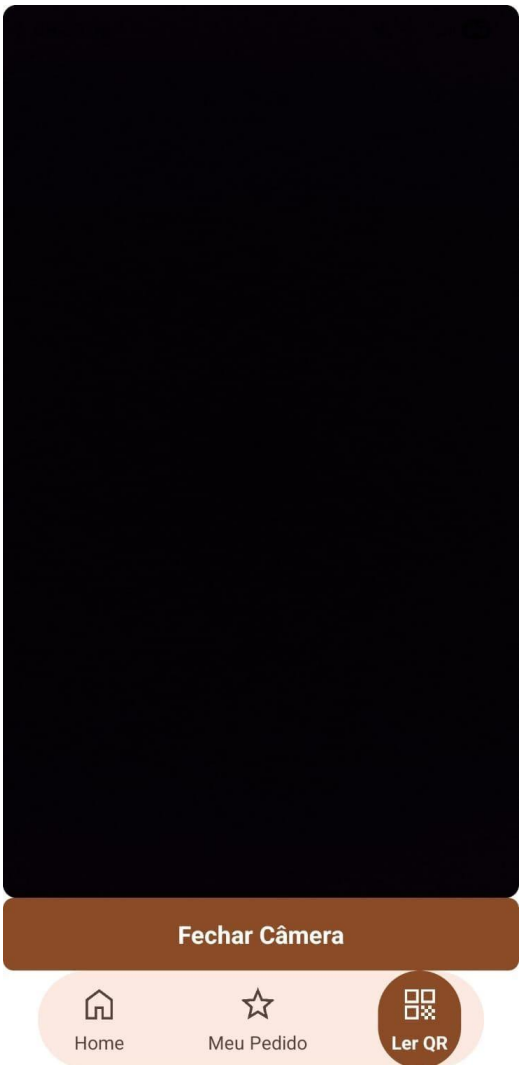
As figuras 15 e 16 apresentam a funcionalidade de leitura de QR code através da câmera.

Figura 15 – Tela para leitura do QR Code

,

Fonte: Autoral.

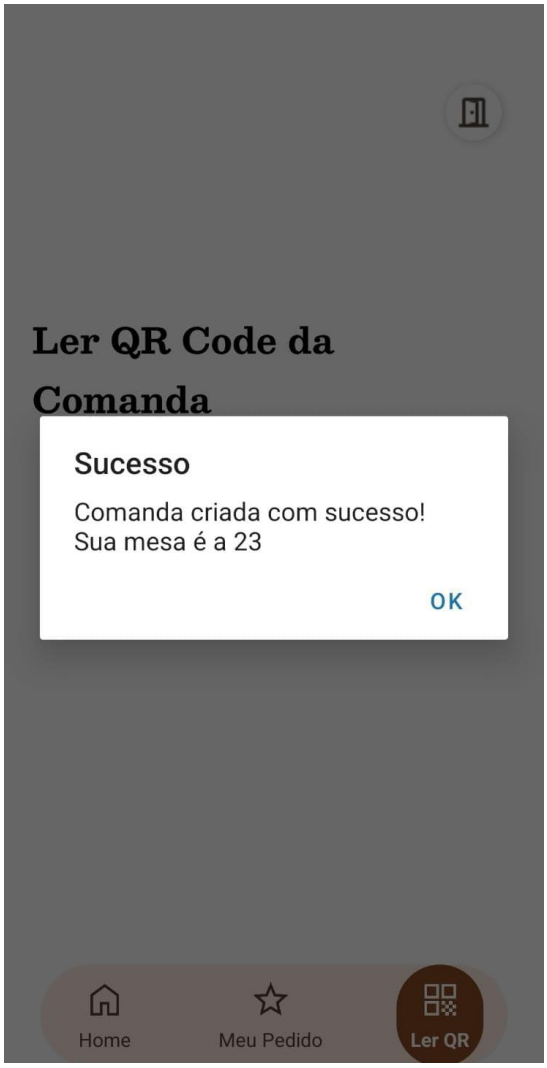
Figura 16 – Tela da câmera para leitura do QR Code



Fonte: Autoral.

A figura 17 apresenta uma mensagem após ler um QR Code para notificar ao cliente que ele obteve sucesso ao entrar em uma mesa. A figura 18 apresenta um *hover* – ação ao clicar em um botão – para navegar para a aba de “meu pedido”.

Figura 17 – Tela com alert de sucesso ao criar comanda



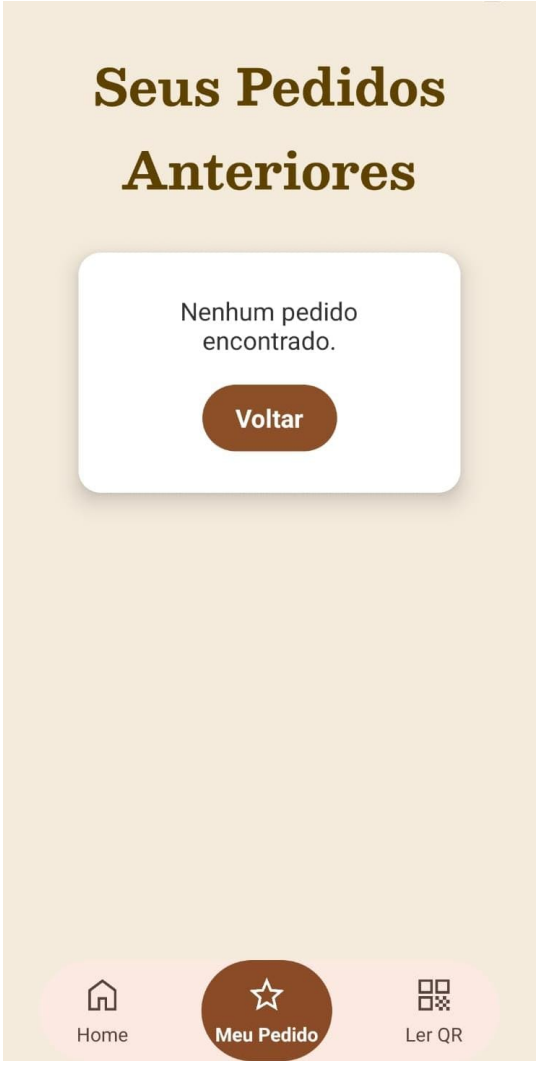
Fonte: Autoral.

Figura 18 – Tela de navegação para “meu pedido”

Fonte: Autoral.

A figura 19 apresenta a tela que apresentará os pedidos feitos e seu status. A figura 20 apresenta o cardápio e tela principal do app.

Figura 19 – Tela Meu Pedido



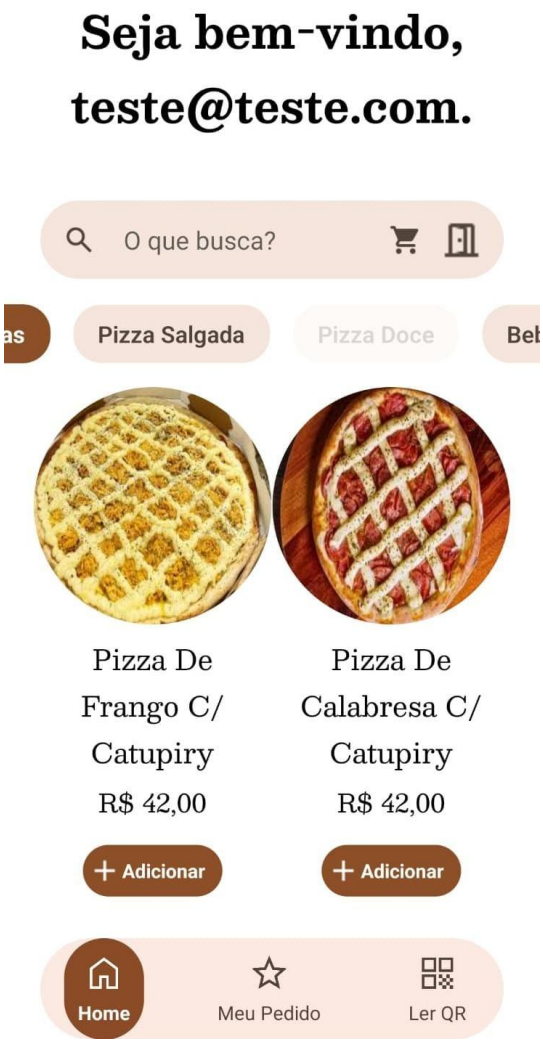
Fonte: Autoral.

Figura 20 - Tela do cardápio

Fonte: Autoral.

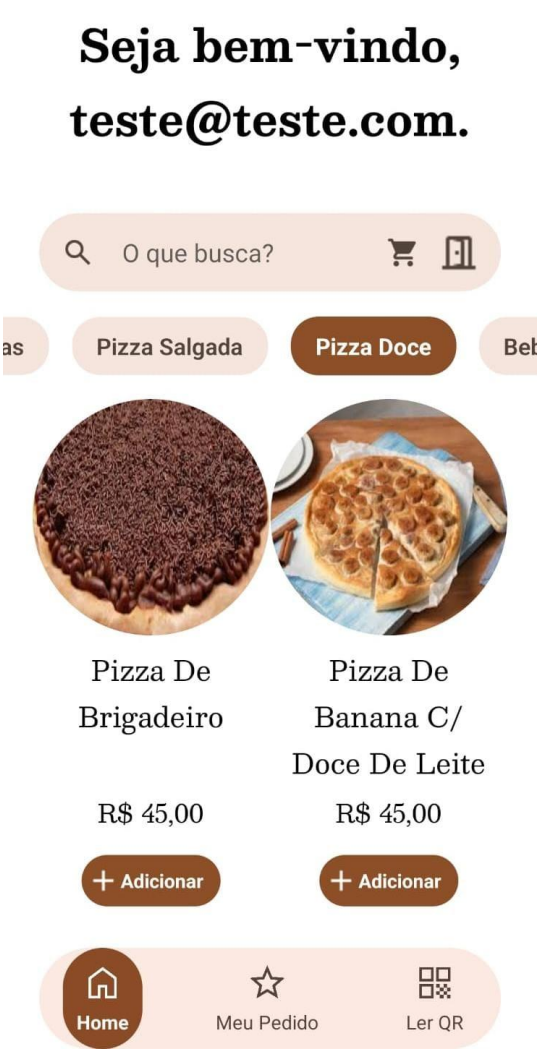
As figuras 21 e 22 exemplificam a funcionalidade de pesquisa por filtro baseado na categoria dos produtos.

Figura 21 – Mudando Categoria de produto no filtro



Fonte: Autoral.

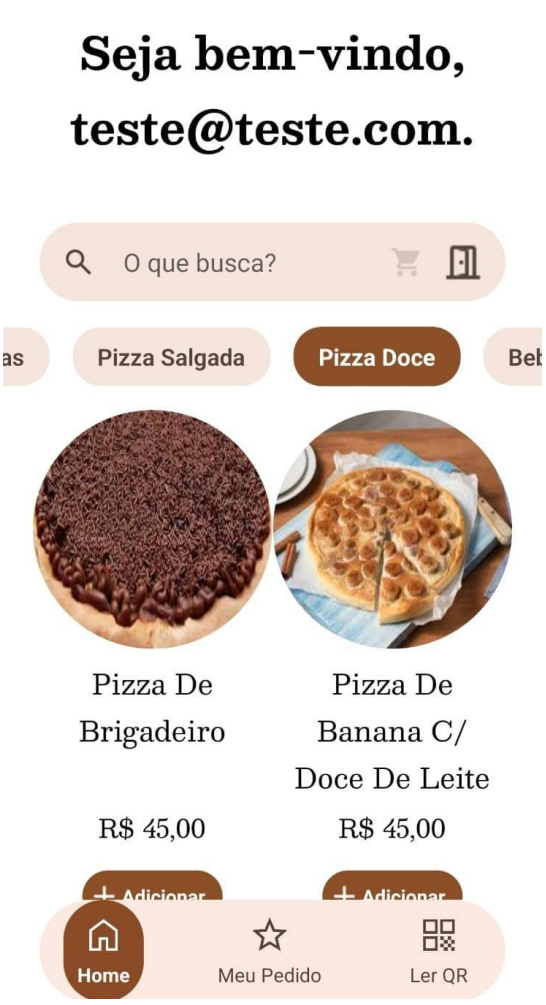
Figura 22 - Tela com nova categoria



Fonte: Autoral.

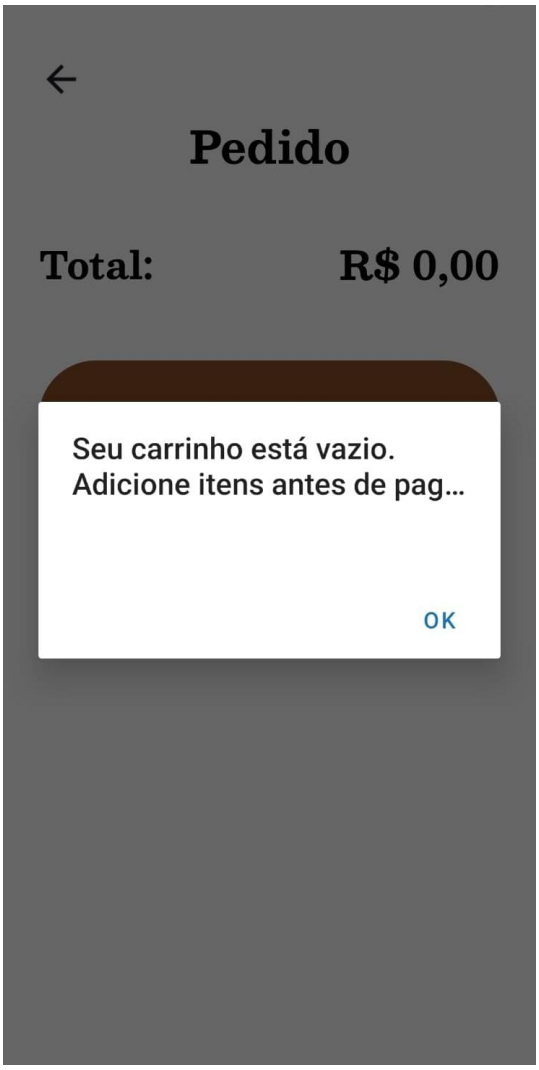
A figura 23 demonstra a navegação para o carrinho e a figura 24 apresenta um erro caso o usuário tente pagar sem ter um produto.

Figura 23 – Tela abrindo Carrinho



Fonte: Autoral.

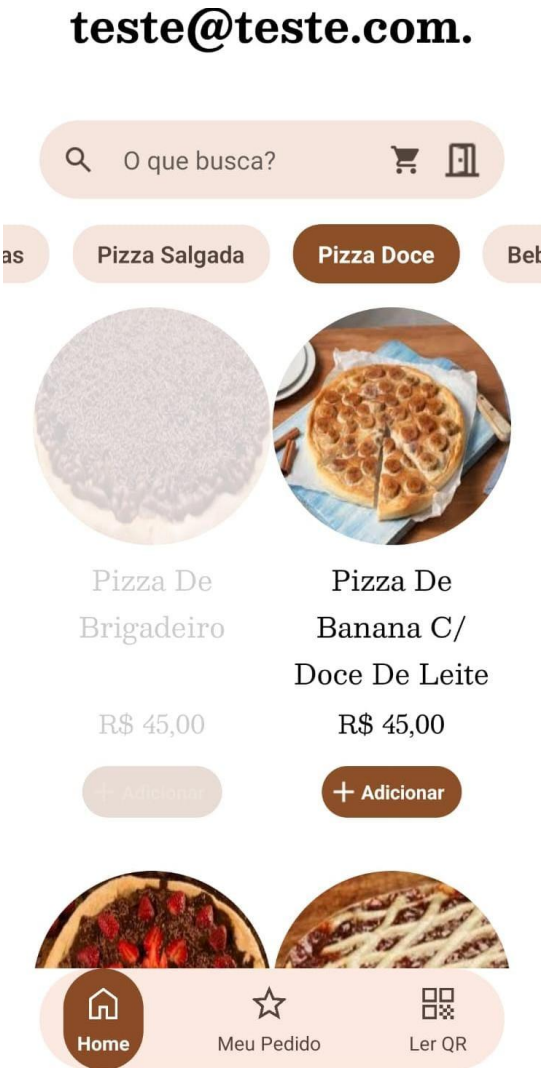
Figura 24 – Mensagem de erro ao pagar sem produto no carrinho



Fonte: Autoral.

A figura 25 e a 26 apresentam a seleção de um pedido e sua personalização de ingredientes a partir do conteúdo selecionado

Figura 25 - Selecionando Produto



Fonte: Autoral.

Figura 26 – Tela de customização do produto

Fonte: Autoral.

As figuras 27 e 28 apresentam imagens que exemplificam a funcionalidade do carrinho e sua exibição.

Figura 27 – Tela com a customização do produto

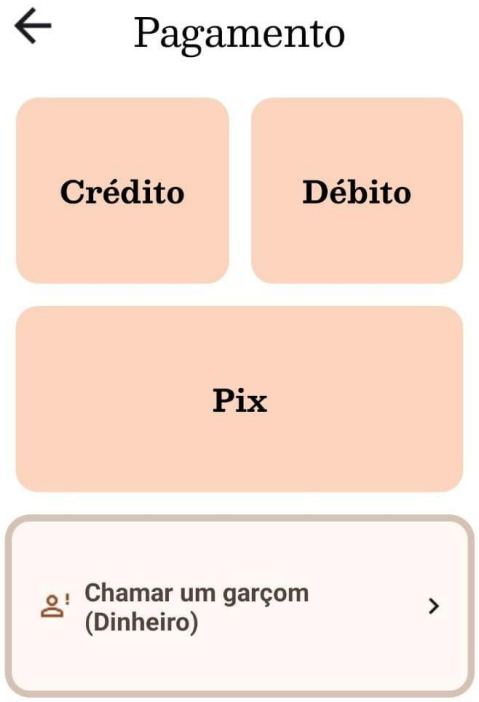
Fonte: Autoral.

Figura 28 – Tela do carrinho pronto

Fonte: Autoral.

A figura 29 apresenta os tipos de pagamentos presentes no sistema, incluindo crédito, débito, pix e dinheiro. A figura 30 apresenta a tela após a seleção do tipo de pagamento para adicionar um CPF no pedido.

Figura 29 – Tela dos tipos de pagamento



Fonte: Autoral.

Figura 30 – Tela para inserir CPF

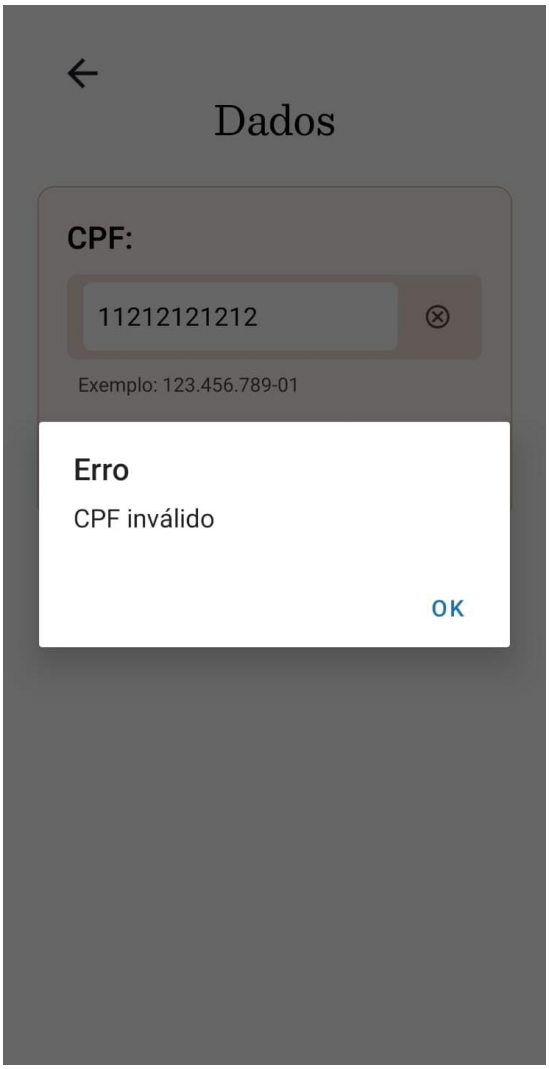
Fonte: Autoral.

A figura 31 demonstra como o botão de pagar fica ao ser selecionado e a figura 32 demonstra a funcionalidade da verificação de um CPF. Caso o CPF inserido não exista ele não é validado para a próxima etapa.

Figura 31 – Tela com um CPF falso inserido

Fonte: Autoral.

Figura 32 – Mensagem de erro com o CPF



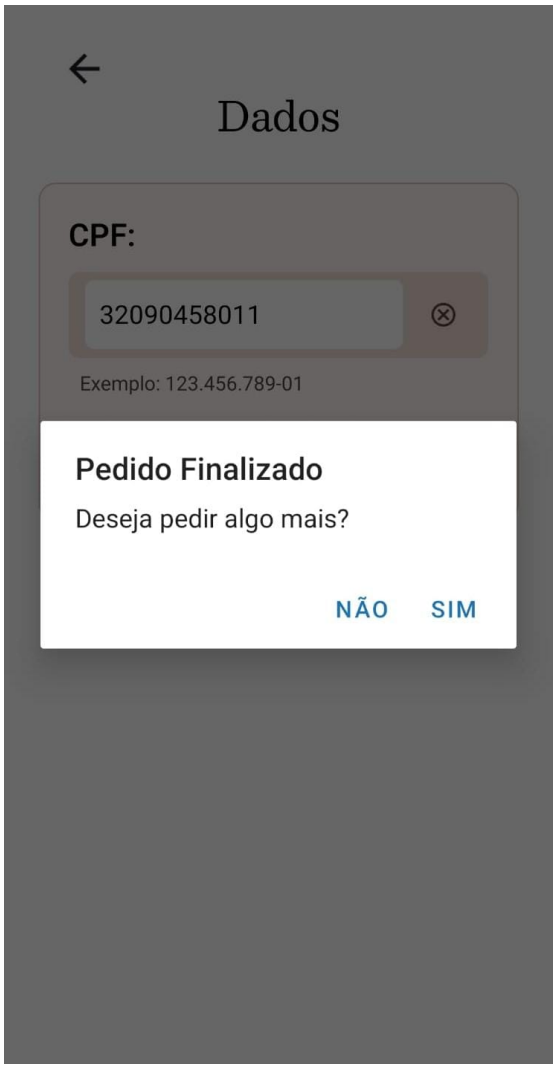
Fonte: Autoral.

A figura 33 apresenta um CPF fictício para validação da funcionalidade e a figura 34 demonstra a tela exibida para o usuário em que ele pode fazer um novo pedido ou finalizar.

Figura 33 – Tela com um CPF real

Fonte: Autoral.

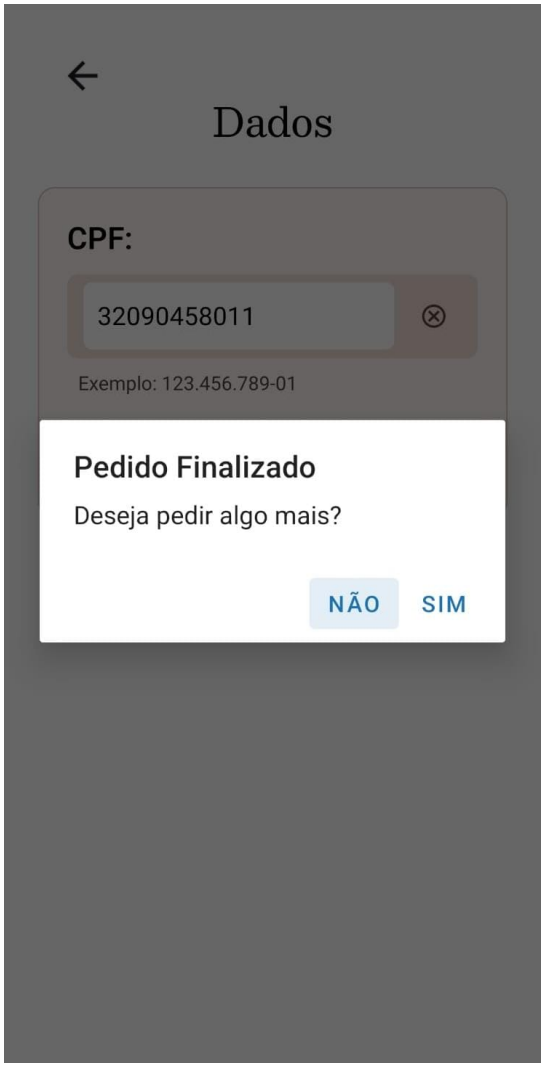
Figura 34 – Tela para finalizar pedido



Fonte: Autoral.

As figuras 35 e 36 apresentam respectivamente a tela de seleção em que o usuário não deseja fazer um novo e após finalizado o pedido ele direciona para a visualização do status em que o pedido se encontra.

Figura 35 – Seleção de finalização de pedido



Fonte: Autoral.

Figura 36 – Tela do Status dos Pedidos Finalizados

Fonte: Autoral.

5 CONCLUSÃO

Diante do cenário de transformação digital e da crescente necessidade de eficiência no setor alimentício, o desenvolvimento do aplicativo *mobile* proposto revelou-se uma solução eficaz para modernizar o gerenciamento de pedidos em uma pizzaria. O sistema atendeu às expectativas do objetivo central do projeto ao disponibilizar uma aplicação funcional, intuitiva e orientada ao autoatendimento, permitindo que o cliente controle todo o fluxo de compra de maneira prática e autônoma.

Durante seu desenvolvimento, verificou-se que a adoção de tecnologias digitais promoveu ganhos expressivos na organização interna, reduzindo erros operacionais, acelerando o atendimento e aprimorando a comunicação entre equipe, cozinha e clientes. Esses resultados reforçam a relevância da integração entre desenvolvimento tecnológico e gestão de processos, sobretudo para pequenos e médios estabelecimentos que buscam competitividade, padronização e qualidade no serviço prestado.

Assim, percebe-se que o aplicativo desenvolvido não apenas alcança as metas inicialmente propostas, como também demonstra o potencial das soluções digitais para transformar rotinas, otimizar fluxos operacionais e elevar o nível de satisfação do consumidor. A ferramenta apresenta-se como uma alternativa viável, escalável e promissora para negócios que desejam inovar, ampliar sua eficiência e fortalecer sua presença no mercado por meio da digitalização. Por fim, evidencia-se a possibilidade de expansão da aplicação para contextos além de pizzarias.

REFERÊNCIAS

ABNT. NBR 17060:2022 — Diretrizes de Acessibilidade em Tecnologias da Informação e Comunicação. Rio de Janeiro: ABNT, 2022.

ANDERSON, David J. Kanban: Evolução Contínua para seu Negócio de Tecnologia. São Paulo: Casa do Código, 2010.

ANDERSON, David J. Kanban: Successful Evolutionary Change for Your Technology Business. Sequim: Blue Hole Press, 2010.

ATLASSIAN. Histórias de usuários com exemplos e um template. Disponível em: https://www.atlassian.com/br/agile/project-management/user-stories. Acesso em: 10 nov. 2025.

BARBOSA, Simone Diniz Junqueira; SILVA, Bruno Santana da. Interação Humano-Computador. Rio de Janeiro: Elsevier, 2011.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. Software Architecture in Practice. 3. Ed. Boston: Addison-Wesley, 2013.

BEZERRA, Eduardo. Engenharia de Software: Teoria e Prática. Rio de Janeiro: LTC, 2018.

BROWN, Tim. Design Thinking: Uma Metodologia Poderosa para Decretar o Fim das Velhas Ideias. Rio de Janeiro: Elsevier, 2010.

CARVALHO, R. D. Scrum e Métodos Ágeis: Guia completo para gestão de projetos modernos. São Paulo: Brasport, 2019.

CHIAVENATO, Idalberto. Introdução à Teoria Geral da Administração. 9. Ed. Rio de Janeiro: Elsevier, 2014.

CYBIS, Walter; BETIOL, Adriana Holtz; FAUST, Robson. Ergonomia e Usabilidade: Conhecimentos, Métodos e Aplicações. 3. Ed. São Paulo: Novatec, 2015.

DAM, Rikke Friis; SIANG, Teo Yu. Design Thinking: Process and Mindset. Interaction Design Foundation, 2021. Disponível em: https://www.interaction-design.org/. Acesso em: 30 nov. 2024.

DARWIN, Brian; BUCH, Narendra; FERREIRA, Carlos. Mobile Application Development: Native and Hybrid Approaches. Journal of Software Engineering, v. 13, n. 2, p. 45–60, 2018.

DATE, C. J. Na Introduction to Database Systems. 2012.

DEBOIS, Michael. The Origin of Agile. 2011. Disponível em: https://goo.gl/znkU3n. Acesso em: 30 nov. 2024.

Design Responsivo – Aprendendo desenvolvimento web | MDN. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Core/CSS_layout/Responsive_Design. Acesso em: 10 nov. 2025.

DONDIS, D. A.; CAMARGO, Jefferson Luiz. Sintaxe da linguagem visual. São Paulo: Martins Fontes, 2007.

ELMASRI, Ramez; NAVATHE, Shamkant B. Sistemas de Banco de Dados. 6. Ed. São Paulo: Pearson Addison Wesley, 2011.

FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. 2000. Tese (Doutorado) – University of California, Irvine, 2000.

FITZSIMMONS, James A.; FITZSIMMONS, Mona J. Administração de Serviços: Operações, Estratégia e Tecnologia da Informação. 8. Ed. Porto Alegre: AMGH, 2014.

GARTNER. Market Share: Devices Worldwide. 2022. Disponível em: https://www.gartner.com/. Acesso em: 30 nov. 2024.

GIT. Git – Reference. Disponível em: https://git-scm.com/docs. Acesso em: 10 nov. 2025.

GOOGLE. Google Fonts. Disponível em: https://fonts.google.com/. Acesso em: 10 nov. 2025.

GOOGLE CAREER CERTIFICATES. Intro to UX (User Experience). Disponível em: https://www.youtube.com/watch?v=2QQQtiFwXjU. Acesso em: 10 nov. 2025.

GONÇALVES, César. Modelagem de Dados: Fundamentos, Técnicas e Aplicações. Rio de Janeiro: Brasport, 2014.

GRÖNLI, Tor-Michael et al. Hybrid versus Native Mobile Applications: A Comparison of Performance and Usability. Procedia Computer Science, v. 27, p. 825–833, 2014.

GRÖNLI, Tor-Michael et al. Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs HTML5. Procedia Computer Science, v. 35, p. 189–196, 2014.

Herança em JavaScript | Alura. Disponível em: https://www.alura.com.br/artigos/heranca-em-javascript. Acesso em: 10 nov. 2025.

Herança em JavaScript — MDN. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Extensions/Advanced_JavaScript_objects/Classes_in_JavaScript. Acesso em: 10 nov. 2025.

IURICODE. Padrões de commits. Disponível em: https://github.com/iuricode/padroes-de-commits. Acesso em: 10 nov. 2025.

ISSO. ISSO/IEC 25010: Systems and Software Engineering — SQuaRE. Geneva: ISSO, 2011.

JOÃO GOMES FILHO. Gestalt do objeto: Sistema de leitura visual da forma. São Paulo: Escrituras, 2008.

KORTH, Henry F.; SILBERSCHATZ, Abraham; SUDARSHAN, S. Sistemas de Banco de Dados. 6. Ed. São Paulo: AMGH, 2016.

LAUDON, K. C.; LAUDON, J. P. Sistemas de Informação Gerenciais. 14. Ed. São Paulo: Pearson, 2020.

LIDWELL, William; HOLDEN, Kritina; BUTLER, Jill. Princípios Universais do Design. Porto Alegre: Bookman, 2010.

LOURENÇO, Paulo. Experiência do Usuário em Sistemas de Autoatendimento. São Paulo: Atlas, 2020.

MAHMOOD, Kashif; RIAZ, Faiza. Performance Evaluation of Hybrid Mobile Applications Using Bridging Techniques. International Journal of Computer Applications, v. 181, n. 25, p. 12–19, 2018.

MAXIMIANO, Antônio César Amaru. Teoria Geral da Administração: Da Revolução Urbana à Revolução Digital. 8. Ed. São Paulo: Atlas, 2020.

MICROSOFT. API Versioning Guidance. Disponível em: https://learn.microsoft.com/. Acesso em: 30 nov. 2024.

MOURA, Henrique de Sousa. Métodos Ágeis para Desenvolvimento de Software. São Paulo: Novatec, 2014.

NIELSEN, Jakob. Usabilidade na Web: Projetando com Usabilidade. Rio de Janeiro: Elsevier, 2014.

NIELSEN, Jakob; BUDIU, Raluca. Mobile Usability. Berkeley: New Riders, 2013.

OLIVEIRA, Djalma de Pinho Rebouças de. Planejamento Estratégico. 32. Ed. São Paulo: Atlas, 2018.

OWASP. API Security Top 10. Disponível em: https://owasp.org/. Acesso em: 30 nov. 2024.

PACHECO, Roberto Carlos dos Santos; MALDONADO, José Carlos. Engenharia de Software: Fundamentos, Métodos e Padrões. 2. Ed. Rio de Janeiro: LTC, 2012.

PAUL, Dipankar; PRAKASH, Anupam. API Management. New York: Apress, 2018.

PAUTASSO, Cesare; ZIMMERMANN, Olaf; LEYMANN, Frank. RESTful Web Services vs. Big Web Services. In: WWW Conference, 2008.

PEREIRA, Rodrigo F. Autoatendimento Digital: Tendências e Aplicações em Serviços. Belo Horizonte: PUC Minas, 2019.

PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de Software: Uma Abordagem Profissional. 8. Ed. Porto Alegre: AMGH, 2016.

ROCHA, F. G. O Que é TDD? Disponível em: https://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151. Acesso em: 10 nov. 2025.

ROCHA, Helena M. M.; BARANAUSKAS, Maria Cecília C. Design e Avaliação de Interfaces Humano-Computador. Campinas: UNICAMP, 2003.

ROSA, L. A. Kanban para Iniciantes: Gestão Visual e Produtividade. Rio de Janeiro: Alta Books, 2015.

SANTOS, J. C. Modelagem de Dados: Conceitos, Diagramas e Práticas. São Paulo: Érica, 2018.

SATYANARAYANAN, Mahadev. Fundamental Challenges in Mobile Computing. In: ACM Symposium on Distributed Computing, 2015.

SATYANARAYANAN, Mahadev. The Emergence of Edge Computing. Computer, v. 48, n. 1, p. 30–39, 2015.

SCHWABER, Ken; SUTHERLAND, Jeff. The Scrum Guide. Disponível em: https://scrumguides.org/. Acesso em: 30 nov. 2024.

SILVA, Marco Túlio da. Banco de Dados: Teoria, Prática e Implementações. São Paulo: Novatec, 2018.

SOMMERVILLE, Ian. Engenharia de Software. 10. Ed. São Paulo: Pearson, 2019.

SOMMERVILLE, Ian. Software Engineering. 9. Ed. Boston: Addison-Wesley, 2011.

STAGE & SNAPSHOT. Git Cheat Sheet. Disponível em: https://education.github.com/git-cheat-sheet-education.pdf. Acesso em: 10 nov. 2025.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. Distributed Systems: Principles and Paradigms. 3. Ed. Hoboken: Pearson, 2017.

TURBAN, Efraim; POLLARD, Carol; WOOD, Robert. Administração de Tecnologia da Informação. 8. Ed. Rio de Janeiro: LTC, 2018.

TURBAN, Efraim; POLLARD, Carol; WOOD, G. Information Technology for Management. 11. Ed. Wiley, 2018.

VIANNA, Maurício et al. Design Thinking: Inovação em Negócios. Rio de Janeiro: MJV Press, 2012.

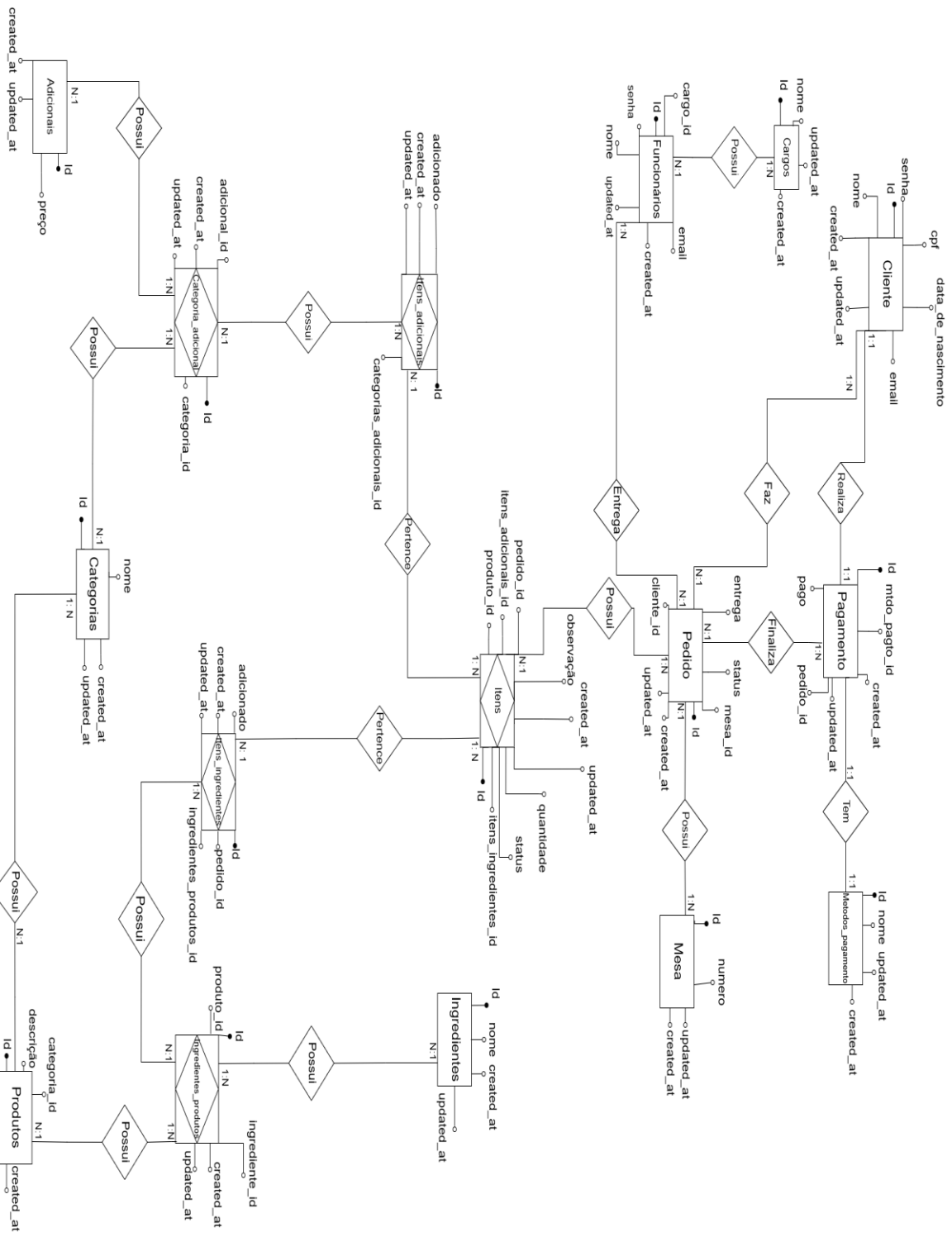
VERGARA, Sylvia Constant. Gestão de Pessoas. 17. Ed. São Paulo: Atlas, 2016.

VERGARA, Sylvia Constant. Projetos e Relatórios de Pesquisa em Administração. 16. Ed. São Paulo: Atlas, 2016.

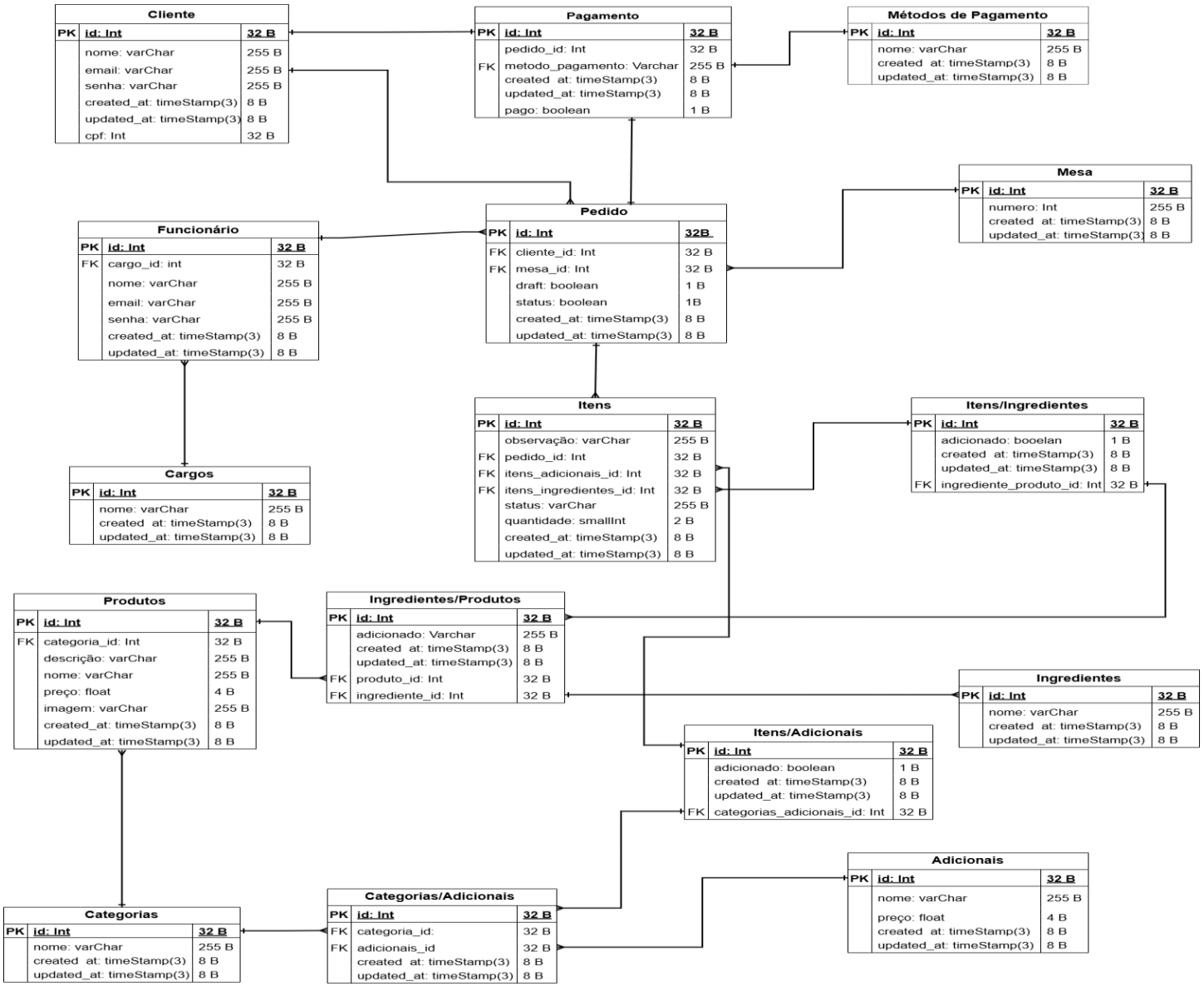
W3SCHOOLS. SQL Tutorial. Disponível em: https://www.w3schools.com/sql/default.asp. Acesso em: 10 nov. 2025.

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

APÊNDICE A – DIAGRAMA DE ENTIDADE DE RELACIONAMENTO



APÊNDICE B – MODELO DE ENTIDADE DE RELACIONAMENTO



GLOSSÁRIO

Aplicativo *Mobile* – Software desenvolvido para ser executado em dispositivos móveis como smartphones e tablets.

Arquitetura Cliente-Servidor – Modelo no qual o cliente faz solicitações e o servidor processa e responde.

Back-End – Parte do software que lida com regras de negócio, banco de dados e APIs.

Banco de Dados Relacional – Sistema de armazenamento estruturado que usa tabelas relacionáveis.

Brainstorming – Técnica de geração de ideias em grupo para solucionar problemas.

Design Thinking – Metodologia centrada no usuário para criação de soluções inovadoras.

Dicionário de Dados – Documento que descreve campos, tipos e regras das tabelas do banco de dados.

Entidade – Objeto do mundo real representado no banco de dados.

Front-End – Parte visual da aplicação com a qual o usuário interage.

História de Usuário – Descrição simples de uma necessidade do usuário em metodologias ágeis.

Interface – Meio de interação entre usuário e sistema.

Prototipagem – Criação de modelos iniciais de telas ou funções de um sistema.

Modelo Entidade-Relacionamento (MER) – Representação conceitual das entidades

Diagrama Entidade-Relacionamento (DER) – Representação gráfica do MER.

Regras de Negócio (RN) – Condições que orientam o funcionamento do sistema baseado no negócio.

Requisitos Funcionais (RF) – Funcionalidades obrigatórias que o sistema deve executar.

Requisitos Não Funcionais (RNF) – Características de qualidade (segurança, performance, usabilidade).

Versionamento – Controle histórico das alterações realizadas no código-fonte.

API – Conjunto de rotinas que permitem a comunicação entre sistemas.

Token JWT – Método de autenticação baseado em *tokens*.

ORM – Ferramenta que mapeia dados do banco para objetos no código.

SaaS / BaaS (Ex.: Supabase) – Plataforma que fornece serviços prontos como banco de dados e autenticação.

QR Code – Código visual bidimensional que pode ser escaneado para acessar informações.

Qrcode do Github:

Qrcode do Trello:

OBS: Para entrar no aplicativo Trello, é necessário ter uma conta.

Qrcode do google drive:

ÍNDICE REMISSIVO