UNIVERSIDADE FEDERAL DO PARÁ
FACULDADE DE ENGENHARIA DE COMPUTAÇÃO

GABRIEL CAMPELO GOMES

DESENVOLVIMENTO DE UM SISTEMA DE GESTÃO DE VENDAS PARA ENTREGA EM DOMICÍLIO

TUCURUÍ
2025

GABRIEL CAMPELO GOMES

DESENVOLVIMENTO DE UM SISTEMA DE GESTÃO DE VENDAS PARA ENTREGA EM DOMICÍLIO

Trabalho de Conclusão de Curso, apresentado como requisito parcial para obtenção de grau de bacharel em Engenharia de Computação, pela universidade Federal do Pará.
Orientador: Prof. Dr. Iago Lins de Medeiros

TUCURUÍ
2025

GABRIEL CAMPELO GOMES

DESENVOLVIMENTO DE UM SISTEMA DE GESTÃO DE VENDAS PARA ENTREGA EM DOMICÍLIO

Trabalho de Conclusão de Curso, apresentado como requisito parcial para obtenção de grau de bacharel em Engenharia de Computação, pela universidade Federal do Pará.
Orientador: Prof. Dr. Iago Lins de Medeiros

Data de aprovação: ____/____/_____

Banca Examinadora

Orientador
Prof. Dr. Iago Lins de Medeiros

Examinador
Prof. Me. Thiego Maciel Nunes

Examinador
Prof. Eng. Erick Mamede

Aos meus pais que ao longo de todos os anos da minha vida dedicaram-se incansavelmente para que hoje eu chegasse até aqui.

AGRADECIMENTOS

Primeiramente a Deus que me concede a graça de viver e de ter experiencias excepcionais a cada dia. Aos meus pais Sr. Alonso C. Gomes e Sr.ª Ernestina C. Gomes, que são meu principal exemplos de vida e a quem desejo honrar, inclusive por meio deste trabalho. Os senhores me deram tudo aquilo que tenho de mais precioso: meus princípios, e valores, minha família e bons exemplos que guardarei com amor no meu coração até o fim dos meus dias.

A minha amada esposa, Amanda D. Gomes, que, principalmente nessa reta final me impulsionou a não desistir e a persistir mesmo frente à exaustão da rotina diária. Ela que me incentivou quando ninguém mais acreditava. Obrigado, meu bem, por ser aquilo que me falta e muito mais. Obrigado por persistir e resistir sempre ao meu lado.

Às minhas filhas, que mesmo ainda tão pequenas são fonte de motivação diária, meu combustível, meu revigorar e meu animo. Aos meus irmãos que são pessoas que eu admiro e amo, pessoas completamente diferentes de mim, mas pelas quais eu torço e rezo pelos seus êxitos, para que alcancem seus objetivos, que sejam bem-sucedidos, mas que principalmente sejam felizes onde quer que estejam.

Aos meus amigos que me ajudaram muitas vezes a suavizar o fardo carregado, com a alegria das conversas e com o compartilhamento também dos problemas. Ao Prof. Iago L Medeiros que me orientou neste trabalho e me permitiu assim finalizar mais este ciclo que será tão importante pra minha vida pessoal e profissional.

Obrigado a todos!

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo de sistema de gestão de vendas (PDV) para delivery, em arquitetura em três camadas, com stack 100% Python: FastAPI (back-end/REST), Kivy (front-end/GUI) e MongoDB (persistência NoSQL). O objetivo é oferecer uma solução acessível, escalável e personalizável para pequenos e médios negócios, enfrentando limitações recorrentes de custo, flexibilidade e integração. A pesquisa é aplicada, de caráter exploratório-descritivo, conduzida de forma incremental com testes e ajustes contínuos. O protótipo implementa autenticação JWT, CRUD de produtos e pedidos, atualização de status em tempo real e relatórios básicos. A avaliação qualitativa, baseada em entrevista com usuária de referência, indicou ganhos operacionais (maior agilidade no atendimento e no fechamento de caixa, redução de erros e preservação de informações). O estudo contribui nos âmbitos acadêmico (aplicação de práticas de Engenharia de Software e desenvolvimento full-stack), tecnológico (integração API–GUI–NoSQL) e prático (base para evolução comercial). São discutidas vantagens e limitações frente a soluções de mercado e delineados trabalhos futuros (dashboards analíticos, integração com meios de pagamento, testes automatizados, melhorias de UX móvel e sincronização offline).

Palavras-chave: PDV; delivery; FastAPI; Kivy; MongoDB; arquitetura cliente–servidor; full-stack Python.

ABSTRACT

This study presents the development of a point-of-sale (POS) prototype for delivery, following a three-tier architecture and a fully Python-based stack: FastAPI (back end/REST), Kivy (front end/GUI), and MongoDB (NoSQL persistence). The goal is to provide an affordable, scalable, and customizable solution for small and medium-sized businesses, addressing common constraints related to cost, flexibility, and integration. The research is applied, exploratory-descriptive, and implemented incrementally with continuous testing and refinements. The prototype includes JWT authentication, CRUD for products and orders, real-time status updates, and basic reporting. A qualitative assessment with a representative end user indicated operational improvements (faster service and cash closing, fewer errors, and better information retention). The work offers academic contributions (software-engineering practices and full-stack development), technological contributions (API–GUI–NoSQL integration), and practical value (a foundation for commercial evolution). We discuss strengths and limitations compared to market solutions and outline future work (analytical dashboards, payment-gateway integration, automated testing, mobile UX enhancements, and offline synchronization).

Keywords: POS; delivery; FastAPI; Kivy; MongoDB; client–server architecture; Python full-stack.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

O setor de entregas em domicílio (delivery) vive um momento de expansão acelerada no Brasil. Segundo a Associação Brasileira de Bares e Restaurantes (ABRASEL), cerca de 71% dos estabelecimentos já oferecem algum tipo de serviço de entrega, e 76% dos consumidores utilizam o delivery com frequência, sinalizando como esse modelo está cada vez mais presente no cotidiano brasileiro (ABRASEL, 2023, 2025).

A evolução tecnológica no setor, com destaque para a integração de sistemas e a automação de processos, tem fortalecido a eficiência e a competitividade das micro e pequenas empresas. Esse fortalecimento ocorre especialmente quando soluções de PDV (Ponto de Venda) e de catálogo/delivery se conectam a marketplaces e plataformas de e-commerce, possibilitando gestão integrada de pedidos e estoque, além de agilizar o atendimento (BAESSO; ROMANI-DIAS, 2023; BRENER, 2024).

No contexto do desenvolvimento full stack com Python, a combinação da linguagem com ferramentas modernas e robustas vem ganhando destaque, pois permite operações assíncronas rápidas e escaláveis, além de garantir segurança com módulos nativos da biblioteca padrão (como *secrets* e *hmac*) (MONGODB DEVELOPER, 2024; PYTHON SOFTWARE FOUNDATION, s.d.).

Diante desse cenário, este trabalho tem como objetivo projetar e implementar um protótipo de sistema PDV para delivery, com foco em viabilidade de uma solução acessível e escalável. A proposta se apoia em integrações tecnológicas e busca também reduzir a fragmentação do conhecimento, contribuindo tanto para o desenvolvimento profissional quanto para a melhoria das práticas de entrega tecnológica (BRENER, 2024).

A versão aqui apresentada corresponde ao aprimoramento de uma primeira implementação, que atendia parcialmente às demandas, mas não estava tecnicamente bem estruturada por utilizar códigos não padronizados e soluções pouco escaláveis.

1.1. Problema de pesquisa

Embora o setor de delivery tenha crescido rapidamente e os sistemas de gestão de ponto de venda (PDV) estejam cada vez mais presentes no mercado brasileiro, persistem limitações relevantes nas soluções disponíveis. Entre os principais problemas estão o custo elevado de adoção e manutenção, a baixa flexibilidade para personalizações e a integração insuficiente com sistemas legados. Essas limitações também aparecem em soluções Software as a Service (SaaS), especialmente no tocante à customização por

cliente e à dificuldade de integração com aplicações instaladas localmente (on-premises) (ARAUJO; VAZQUEZ, 2013).

Esses fatores dificultam o acesso de pequenos e médios empreendedores a ferramentas tecnológicas adequadas, comprometendo a eficiência operacional e a capacidade de competir em um mercado cada vez mais digital e dinâmico (ABRASEL, 2023). Além disso, o avanço das tecnologias e a diversificação de canais de venda exigem sistemas capazes de atender múltiplas plataformas, com interfaces adaptáveis e bancos de dados escaláveis (SISCHEF, 2023; WEF, 2025).

Diante desse cenário, formula-se a seguinte questão de pesquisa: Como desenvolver um sistema de PDV multiplataforma, de baixo custo, escalável e baseado em tecnologias modernas como FastAPI, Kivy e MongoDB, capaz de atender às necessidades de pequenos e médios negócios de delivery, superando as limitações observadas nas soluções atuais?

1.2. Justificativa

O forte crescimento do delivery e a digitalização dos processos comerciais exigem soluções tecnológicas acessíveis, flexíveis e integradas — características ainda pouco presentes em grande parte das soluções disponíveis no mercado (OECD, 2021).

No Brasil, pequenos e médios empreendedores enfrentam obstáculos como custos elevados, falta de personalização e baixa integração entre plataformas, fatores que dificultam a adoção de sistemas de PDV adaptáveis às suas realidades (PRESTES, 2025).

Além do impacto econômico, existe também uma dimensão educacional e profissional. Muitos desenvolvedores em formação ainda lidam com a fragmentação do conhecimento, comum nos cursos de graduação, o que limita a capacidade de criar soluções funcionais alinhadas às demandas reais do mercado (ROMÃO et al., 2024). Nesse sentido, projetos que integram teoria e prática, utilizando tecnologias modernas como FastAPI, Kivy e MongoDB, reduzem essa lacuna e permitem uma atuação mais efetiva como desenvolvedor full stack (SISCHEF, 2023; WEF, 2025).

Portanto, a relevância deste trabalho não está apenas em oferecer ao mercado um protótipo de sistema PDV multiplataforma para delivery, mas também em contribuir para a formação de competências técnicas, para o domínio de arquiteturas full stack e para a integração tecnológica em um cenário profissional cada vez mais competitivo e dinâmico.

1.3. Objetivo geral

Desenvolver um protótipo funcional de sistema de gestão de vendas (PDV) multiplataforma para delivery, utilizando FastAPI (back-end/API), Kivy (front-end/interface) e MongoDB (persistência de dados). O objetivo é demonstrar a viabilidade técnica e os benefícios de uma arquitetura full stack baseada em Python, ao mesmo tempo em que promove a construção de competências técnicas relevantes para o desenvolvimento profissional

1.4. Objetivos específicos

No front-end, pretende-se desenvolver telas independentes (Screens) e seus respectivos componentes, como botões, campos de texto e entradas de dados. A navegação será gerenciada por meio de um ScreenManager, que garantirá a comunicação entre as diferentes telas e a consistência visual da aplicação.

No back-end, o objetivo é implementar autenticação de usuários utilizando JSON Web Token (JWT), assegurando a validação de cada requisição. Além disso, o sistema deverá processar chamadas da interface e retornar informações persistidas no banco de dados MongoDB, garantindo segurança e integridade no fluxo de dados.

Quanto ao banco de dados, busca-se modelar coleções específicas para usuários, pedidos e itens, estruturando-as de forma flexível. Essa modelagem permitirá que cada pedido seja registrado de acordo com suas particularidades, explorando o potencial de adaptação oferecido pelo MongoDB.

No que se refere à integração, pretende-se assegurar a comunicação eficiente entre a interface do usuário (front-end), o back-end (API) e o banco de dados. Para isso, serão implementados mecanismos de tratamento e conversão de dados, reduzindo erros e garantindo consistência no funcionamento da aplicação.

Por fim, no âmbito de desempenho e qualidade, objetiva-se otimizar o tempo de atendimento, o envio de pedidos para a produção e o despacho para entrega, além de disponibilizar relatórios básicos, como vendas diárias e pedidos segmentados por forma de pagamento.

1.5. Organização do Trabalho

Esta monografia está organizada da seguinte forma: **Capítulo 1** aborda os aspectos gerais do trabalho, nela apresenta-se um cenário atual do comercio brasileiro. Também se introduz a realidade de entregas em domicílio, delivery. Isso no leva a demanda crescente por adoção de sistemas de automação comercial que auxiliam no funcionamento de diversos estabelecimentos dando-lhe potencial de crescimento e competitividade.

Capítulo 2 reúne a Fundamentação Teórica, que é essencial para entender todo o contexto do trabalho. Nesse Capitulo são abordados conceitos como: desenvolvimento *full stack,* que se dá quando um desenvolvedor trabalha em todas as camadas do projeto; arquitetura API REST que possibilita escalabilidade da aplicação de forma que vários dispositivos se comuniquem; a utilização de linguagem python para o desenvolvimento de sistemas, que integrado a seus frame works abrem porta para muitas possibilidades de aplicação e, por fim discorre sobre MongoDB, sua aplicabilidade e potencialidades no contexto de PDVs.

Capítulo 3 apresenta trabalhos relacionados que desenvolveram aplicações que corroboram com este trabalho, reforçando assim a sua relevância e impacto real. Esses trabalhos utilizam conceitos como arquitetura em camadas, utilização do MongoDB e banco de dados não relacionais como utilizados no projeto deste trabalho

Capítulo 4 fala sobre o desenvolvimento pratico da aplicação que motiva este tralho, apresentando desde a demanda inicial pela automação, objetivos alcançados na sua primeira versão e problemas que demandaram o aperfeiçoamento do sistema na sua segunda versão. Apresenta também como foram implementadas as etapas do projeto e como se relacionam as suas camadas.

Capítulo 5 são apresentadas algumas perguntas feitas a gerente do estabelecimento que foi a pessoa designada a apresentar a demanda inicial e com quem foi feito o levantamento de requisitos. Essas perguntas e respostas abordam tanto mento antes da implantação da automação quanto depois, que são os resultados que serão analisados qualitativamente quanto ao impacto e relevância.

Capítulo 6 apresenta as contribuições deste trabalho no desenvolvimento profissional, na melhoria e aprimoramento empresarial e como alternativa para pequenos e médios negócios que precisam de melhoria na sua gestão de vendas

2. FUNDAMENTAÇÃO TEÓRICA

2.1. Engenharia de Software e Desenvolvimento Full Stack

A fundamentação teórica tem como objetivo apresentar os conceitos, tecnologias e abordagens que servem de base para o desenvolvimento do sistema proposto. Nesse contexto, são discutidos aspectos de engenharia de software, arquitetura de sistemas, linguagens e frameworks de desenvolvimento e bancos de dados, relacionando-os às necessidades de um sistema de PDV voltado para delivery.

Segundo Pressman e Maxim (2020), a engenharia de software consiste em aplicar princípios, métodos e ferramentas de forma sistemática para produzir software de qualidade, confiável e economicamente viável. A adoção de boas práticas nesse campo é essencial para reduzir custos, evitar falhas e garantir a manutenção contínua das soluções ao longo do tempo. Sommerville (2019) complementa que a engenharia de software deve combinar processos formais e adaptação prática, a fim de atender às demandas específicas de cada projeto.

No contexto deste trabalho, a fundamentação se estrutura em três eixos principais: a evolução dos sistemas de ponto de venda (PDV), os paradigmas de arquitetura de software relevantes, e a análise das tecnologias escolhidas — Python, FastAPI, Kivy e MongoDB —, ressaltando seu papel na implementação do protótipo.

2.2. Aplicações de PDV e automação para *delivery*

A adoção de sistemas de PDV no setor de alimentação e delivery deixou de ser apenas um apoio administrativo para se tornar um diferencial competitivo. Esses sistemas, ao automatizarem processos antes feitos manualmente, permitem maior agilidade no atendimento, redução de erros e eficiência no controle financeiro e operacional (ABRAHÃO, 2024; ABRASEL, 2023).

Entre as principais aplicações estão o gerenciamento de pedidos em tempo real, a integração com aplicativos de entrega e o cálculo automático de taxas e comissões, funcionalidades que melhoram a comunicação entre cozinha, atendimento e entregadores. A automação também garante a preservação das informações, evitando perdas que frequentemente ocorrem em processos manuais, como extravio de comandas (SYSMIDDLE, 2024; GILBERT, 2019).

Pesquisas recentes destacam que a automação pode aumentar significativamente a produtividade e a rentabilidade de pequenos e médios estabelecimentos, sobretudo aqueles que enfrentam limitações de mão de obra e recursos financeiros. Isso ocorre porque os sistemas integrados reduzem o retrabalho, otimizam a gestão de insumos e possibilitam a emissão de relatórios que auxiliam na tomada de decisão baseada em dados (IMARC GROUP, 2025; WEF, 2025).

Nesse cenário, a automação aplicada ao delivery vai além da simples substituição de tarefas manuais: trata-se de um elemento estratégico que conecta operações de venda, logística e análise de dados, criando condições para maior previsibilidade, escalabilidade e competitividade, principalmente em mercados locais.

2.3. Arquitetura cliente-servidor e APIs REST

A arquitetura cliente-servidor é um modelo clássico de organização de sistemas computacionais que separa as responsabilidades entre cliente e servidor. O cliente é responsável por interagir com o usuário e enviar solicitações, enquanto o servidor processa essas solicitações, aplica regras de negócio e retorna respostas adequadas (TANENBAUM; WETHERALL, 2011). Essa abordagem promove escalabilidade, modularidade e facilidade de manutenção, permitindo que cada parte evolua de forma independente.

No contexto de aplicações modernas, o cliente pode ser representado por aplicativos desktop, sistemas móveis ou navegadores web, enquanto o servidor concentra a lógica da aplicação e se comunica com bancos de dados e serviços externos (FOWLER, 2012). Essa separação é especialmente importante em ambientes que exigem alta disponibilidade, desempenho e segurança, como é o caso de sistemas de PDV integrados a plataformas de delivery.

Um padrão amplamente adotado para a comunicação entre cliente e servidor é o REST (Representational State Transfer), proposto por Fielding (2000). Esse estilo arquitetural define princípios que possibilitam a criação de APIs leves, escaláveis e de fácil integração, utilizando os métodos HTTP — como GET, POST, PUT e DELETE — para representar operações sobre recursos. A simplicidade do REST, aliada ao suporte nativo de protocolos web, consolidou-o como a abordagem preferencial para integração entre sistemas heterogêneos.

No caso deste trabalho, a escolha por uma arquitetura cliente-servidor baseada em APIs REST permite integrar o front-end desenvolvido em Kivy, o back-end construído

com FastAPI e o banco de dados MongoDB, de forma modular e escalável. Essa opção técnica garante maior flexibilidade na evolução do protótipo e maior aderência a padrões amplamente utilizados em sistemas corporativos.

2.4. Linguagem Python no desenvolvimento de sistemas

A linguagem Python consolidou-se como uma das mais utilizadas no desenvolvimento de sistemas modernos, destacando-se pela simplicidade sintática, ampla comunidade e diversidade de bibliotecas disponíveis. Rankings recentes, como o TIOBE Index (2025) e o PYPL (2025), posicionam Python entre as linguagens mais populares, especialmente em áreas como ciência de dados, inteligência artificial, desenvolvimento web e automação.

Sua natureza interpretada, multiplataforma e de tipagem dinâmica torna a linguagem acessível e adaptável, permitindo a criação de protótipos rápidos sem abrir mão de escalabilidade. A combinação de simplicidade e robustez garante que desenvolvedores em formação possam aprender rapidamente, ao mesmo tempo em que profissionais avançados podem construir aplicações complexas com alta performance (NORTHWOOD, 2018; WAZLAWICK, 2021).

No contexto do desenvolvimento full stack, Python apresenta vantagens por oferecer suporte tanto a bibliotecas para back-end (como FastAPI, Flask e Django) quanto a ferramentas de front-end multiplataforma, como o framework Kivy. Essa característica permite implementar sistemas completos com uma base tecnológica única, reduzindo custos de integração e curva de aprendizado.

A escolha do Python neste trabalho justifica-se pela experiência prévia do autor com a linguagem, por sua ampla documentação e pelo suporte a operações assíncronas, essenciais para aplicações que lidam com múltiplos pedidos simultâneos em sistemas de delivery.

2.5. Framework Kivy e desenvolvimento de interfaces gráficas

O Kivy é um framework de código aberto para o desenvolvimento de interfaces gráficas multiplataforma, baseado em Python. Sua proposta é permitir que uma mesma aplicação seja executada em diferentes sistemas operacionais (Windows, Linux, Android e iOS) com mínima necessidade de ajustes, favorecendo a portabilidade (ABUBAKAR et al., 2021; KIVY, 2025).

Entre suas principais características estão o uso da linguagem de marcação KV Language para definição de layouts, o suporte a multitouch e a capacidade de integração com bibliotecas externas em Python. Essas funcionalidades tornam o Kivy especialmente útil em aplicações que exigem interfaces customizáveis e com foco em interação do usuário.

No contexto de sistemas de PDV, o Kivy oferece recursos adequados para a construção de telas dinâmicas, formulários, botões, menus e relatórios, elementos essenciais para a operação diária de estabelecimentos comerciais. Além disso, sua arquitetura modular favorece a manutenção e evolução da aplicação.

Apesar das vantagens, o Kivy apresenta algumas limitações, como a menor popularidade em relação a frameworks mais difundidos (Flutter, React Native) e a necessidade de otimizações para garantir desempenho em dispositivos móveis de baixo custo. Ainda assim, sua escolha neste trabalho se justifica pela integração nativa com Python e pela flexibilidade no desenvolvimento multiplataforma.

2.6. Banco de dados MongoDB: modelo NoSQL, vantagens e desvantagens

O MongoDB é um sistema de gerenciamento de banco de dados baseado no modelo NoSQL orientado a documentos. Ao contrário dos bancos relacionais, que utilizam tabelas e colunas, o MongoDB armazena dados em documentos JSON/BSON, permitindo maior flexibilidade na estruturação das informações (GYŐRÖDI et al., 2022).

Essa característica é particularmente útil em aplicações de delivery e PDV, nas quais diferentes pedidos podem conter estruturas variáveis de itens, formas de pagamento múltiplas e atributos específicos de cada transação. Dessa forma, o modelo NoSQL reduz a rigidez e facilita a evolução do sistema sem necessidade de migrações complexas (BOŽIĆ, 2022).

Entre as principais vantagens do MongoDB estão:

- **Escalabilidade horizontal**, possibilitando distribuir dados em múltiplos servidores;
- **Flexibilidade de esquema**, permitindo adicionar ou remover campos sem reestruturação;
- **Alto desempenho em operações de leitura e escrita**, essencial para sistemas em tempo real;

- **Integração nativa com APIs REST**, facilitando a comunicação com frameworks como FastAPI.

Por outro lado, o MongoDB apresenta desvantagens que precisam ser consideradas. A ausência de transações complexas como as dos bancos relacionais pode dificultar aplicações que exigem consistência total. Além disso, a gestão de grandes volumes de dados requer configurações de replicação e sharding mais avançadas, aumentando a complexidade de administração (HOSSAIN et al., 2020).

A escolha pelo MongoDB neste trabalho se justifica pelo equilíbrio entre flexibilidade, escalabilidade e custo-benefício, características adequadas a pequenos e médios empreendedores que buscam soluções tecnológicas acessíveis sem abrir mão de desempenho.

3. TRABALHOS RELACIONADOS

Tabela 1 - Trabalhos relacionados 1/2

Trabalho / Fonte	**Multiplataforma**	**Custo para o usuário**	**Flexibilidade / Personalização**
Este trabalho (GOMES, 2025) – Protótipo PDV full stack	**Sim** (desktop; potencial mobile com Kivy)	**Baixo** (sem mensalidade; código do autor)	**Alta** (código aberto do protótipo; NoSQL flexível)
Pediline (BRENER et al., 2024)	Parcial (web + app externo)	Assinatura mensal (SaaS)	Baixa (personalização restrita ao provedor)
Cooked with Care (CHAUHAN et al., 2022)	Limitado (web browser)	Baixo (projeto acadêmico)	Restrita (schema relacional rígido)
Aplicações com Kivy (ABUBAKAR et al., 2021)	**Sim** (desktop + mobile)	Baixo (open source)	Alta (alto grau de customização)
Sischef (2023)	**Sim** (web/mobile)	Médio/Alto (licença/mensalidade)	Média (configurações; pouca customização profunda)

Fonte: Desenvolvida pelo autor deste trabalho

Tabela 2 - Trabalhos relacionados 2/2

Trabalho / Fonte	**Base tecnológica**	**Público-alvo principal**	**Observações relevantes**
Este trabalho (GOMES, 2025) – Protótipo PDV full stack	**Python (FastAPI, Kivy)** + **MongoDB**	Pequenos e médios negócios de delivery	Validado em contexto real; pendências: integração de pagamentos e dashboards (planejados)
Pediline (BRENER et al., 2024)	SaaS em nuvem	Pequenos negócios de alimentação	Integra com apps de entrega; dependência do serviço
Cooked with Care (CHAUHAN et al., 2022)	Arquitetura web + **SQL**	Protótipo acadêmico / estudo de caso	Escalabilidade limitada para cenários dinâmicos
Aplicações com Kivy (ABUBAKAR et al., 2021)	**Kivy + Python**	Projetos acadêmicos e startups	Demonstra viabilidade multiplataforma real
Sischef (2023)	SaaS em nuvem	Pequenos e médios negócios	Adoção ampla; custo recorrente e menor flexibilidade

Fonte: Desenvolvida pelo autor deste trabalho

As Tabelas 1 e 2 concentram apenas aplicações e protótipos concretos, comparáveis por critérios objetivos (multiplataforma, custo, flexibilidade e base tecnológica). Por isso, estudos conceituais e revisões técnicas foram mantidos somente no texto, pois não oferecem implementação direta que permita cotejar custo para o usuário, suporte a plataformas ou nível de personalização de modo uniforme. Exemplos: Baesso & Romani-Dias (2023) discutem oportunidades de automação e integração com marketplaces, mas sem um sistema implementado; Hossain et al. (2020) tratam de arquitetura em camadas (referencial metodológico); OECD (2021) e WEF (2025) trazem tendências e dados setoriais — todos fundamentais para contextualizar o problema, porém não comparáveis em tabela.

Entre os trabalhos com implementação, Pediline (Brener et al., 2024) se destaca pela proposta SaaS voltada ao catálogo e ao delivery, com integração a apps de entrega; entretanto, o custo recorrente de assinatura e a personalização restrita representam barreiras para pequenos negócios que exigem ajustes finos nos fluxos de venda e de produção. Já o protótipo acadêmico Cooked with Care (Chauhan et al., 2022) demonstra um website de pedidos sobre arquitetura cliente–servidor; embora útil didaticamente, enfrenta limitações de escalabilidade e menor flexibilidade frente a cenários de delivery com dados altamente variáveis.

No eixo de interfaces multiplataforma, aplicações com Kivy mostram a viabilidade prática de um único código para desktop e mobile, reduzindo esforço de manutenção — atributo alinhado às necessidades de estabelecimentos com recursos limitados (Abubakar et al., 2021). Entre as soluções de mercado, o Sischef ilustra a oferta web/mobile pronta para uso, porém com custo de licença/mensalidade e customização moderada, o que reforça o espaço para alternativas de baixo custo e alto grau de personalização.

No plano de fundamentação técnica, estudos sobre NoSQL — como Győrödi et al. (2022) e Božić (2022) — sustentam a adoção do MongoDB ao evidenciar desempenho estável com grandes volumes e trade-offs típicos (flexibilidade de esquema versus governança/consistência), úteis para modelar pedidos heterogêneos e formas de pagamento sem migrações onerosas. Esses trabalhos permanecem fora da Tabela por não serem PDVs ou soluções de delivery, mas embasam as decisões de design deste TCC.

Em síntese, os estudos comparados apontam lacunas persistentes: combinar multiplataforma, baixo custo e personalização profunda com uma arquitetura integrada (API–GUI–NoSQL). A proposta deste TCC ocupa essa interseção ao apresentar um protótipo full stack em Python (FastAPI + Kivy + MongoDB), validado em contexto real e com trajetória de evolução definida (integração de pagamentos, dashboards e testes automatizados),

diferenciando-se de SaaS comerciais com custo recorrente e de protótipos acadêmicos menos flexíveis.

4. DESENVOLVIMENTO DE UM SISTEMA DE GESTÃO DE VENDAS PARA ENTREGA EM DOMICÍLIO

A metodologia representa o conjunto de procedimentos técnicos e científicos adotados para alcançar os objetivos deste trabalho. Neste projeto, optou-se por um método de pesquisa aplicada, orientada à solução de um problema prático, com abordagem exploratória e descritiva, para entender melhor o problema e o contexto, desenvolvida sob o paradigma de Engenharia de Software e baseada no modelo full stack utilizando FastAPI, Kivy e MongoDB (LAKATOS; MARCONI, 2017).

4.1. Processo de desenvolvimento:

Este projeto iniciou a partir da demanda de automação de gestão do restaurante de comida Japonesa Hayatoo Sushi, situado na cidade de Tucuruí-PA. No momomento em que o proprietário, Douglas M, Reis fez o primeiro contado, apresentou a necessidade de implementar um sistema em seu estabelecimento para auxiliar na administração de pedidos, valores e entregas.

Primeiramente foi feito o levantamento de requisitos com entrevistas com a gerente do estabelecimento, Elenilda C. Gomes. Ela esclareceu quais informações eram mais relevantes e o de que maneira eles gostariam que o a ferramenta funcionasse. a seguir duas das perguntas feitas à gerente seguidas de suas respectivas respostas.

Pergunta1 - "Quais as principais dificuldades que levaram a busca pela automação do processo de vendas?

Resposta 1: "A anotação manual dos pedidos causa muitos erros e atrasos. buscamos a automação das comandas para amenizar esses problemas e agilizar o atendimento dos clientes e para fins de controle, pois por vezes as comandas eram perdidas pelos entregadores [...]"

Pergunta 2: Como acontece fechamento de caixa?

Reposta 2: "O fechamento manual é feito através da seleção de comandas por forma de pagamento (dinheiro/pix/cartão) e posteriormente somava-se os valores um a um para chegar ao valor final de rendimento"

Outras perguntas foram feitas buscando levantar os dados exatos desejados a serem processado. Os primeiros dados trabalhados foram os seguintes.

Cliente: Nome, endereço e telefone.

Pedidos: Formas de pagamento, dia e hora da realização do pedido, estado (aberto ou fechado), código e quantidade de cada item para cálculo.

Itens: código, categoria, nome, valor e descrição.

Posteriormente foi escolhida a tecnologia, que seria apenas linguagem Python, por já ser a linguagem de programação com a qual estava mais familiarizado. Essa oportunidade também seria importante para o aprimoramento do conhecimento da linguagem em um cenário real.

Na sua primeira versão a ferramenta executava as seguintes tarefas:

- Criar, ler e editar pedidos. Em um arquivo de pedido .CSV.
    - Incluir, editar e excluir itens, adicionais de itens e observação.
    - Calcular valor final de pagamento
    - Segmentar o valor final entre as formas de pagamento disponíveis
- Encerrar o pedido para edições (mediante pagamento)
- Informar relatório de pedidos por forma de pagamento e seus montantes
- Imprimir termicamente cupons não fiscais
- Imprimir vias de produção para a preparação
- Fechar caixa e armazenar registro do dia
- Carregar o cardápio com seus itens de arquivo local também .CSV

Esse projeto estava funcional, porém não estava adequado a boas práticas de desenvolvimento. Apresentava alguns problemas de lentidão à medida que o estabelecimento passou a ter um fluxo cada vez maior de pedidos por dia. Outra questão que levou a necessidade de aprimoramento foi o fato de que alguns erros comprometiam o funcionamento geral da aplicação, causando prejuízo ao andamento dos trabalhos.

A partir dessas situações veio a intensão de uma arquitetura modular e em camadas, que pudesse estar mais adequada as demandas de mercado e que pudessem me levar a um novo nível de conhecimento quanto ao desenvolvimento de software.

Para a versão trabalhada neste trabalho foram escolhidas as seguintes tecnologias: linguagem Python, Kivy (framework Python para o desenvolvimento de interfaces python),

FastAPI (framework Python para o desenvolvimento ágil de APIs) e MongoDB Sistema Gerenciador de Banco de dados não relacionais. A seguir essas tecnologias serão apresentadas

4.2. Arquitetura

A modularidade foi aplicada por meio da separação entre camadas e módulos (rotas, modelos e serviços). Essa organização reduz acoplamento, facilita testes unitários e acelera correções/evoluções, além de permitir escalabilidade horizontal no back-end sem impactar a interface.

A aplicação tem primeiramente a camada de apresentação (front-end) que foi construída com Kivy, ela está em comunicação com a segunda camada que é a camada de rotas (back-end/API) por meio de funções python que realizam as requisições a partir dos dados de entrada. A segunda camada desenvolvida com FastAPI realiza a autenticação e a validação das requisições, assim como permite ou bloqueia o acesso a telas, seja por falta de autenticação ou pelo token ter expirado. Por fim a terceira camada, a camada de banco de dados não relacionais NoSQL que podemos chamar de camada de persistência, mantém os dados íntegros e acessíveis.

4.3. Ambiente

Para o desenvolvimento dessa ferramenta foi utilizado VScode onde criou-se um ambiente virtual com Pyenv, isso possibilitou um ambiente de desenvolvimento isolado do ambiente global da máquina. Essa prática tem muitos benefícios, dentre eles ter ali instalados aapenas o que é essencial a sua aplicação. Por meio do arquivo requeriments é possível listar todas a bibliotecas instaladas, essa lista ´s útil para reconstituir o ambiente virtual em outra máquina com as bibliotecas devidas.

Python 3 é a base principal deste projeto, que pode ser chamado de um fullstack python devido ao fato de que frameworks da própria linguagem fundamentam as suas diferentes camadas. Kivy é um framework da linguagem Python voltado ao desenvolvimento de interfaces gráficas, a versão utilizada foi a 2.3.1 dentro de seus arquivos .KV são organizados os componentes da interface gráfica como telas, botões e caixas de entrada. Esses componentes vinculados ao arquivo main.py realizam chamadas e exibem respostas. Esse frame work foi escolhido por conta de sua praticidade na construção de telas que podem ser multiplataformas e responsivas, fator importante para possíveis ampliações futuras

Para o desenvolvimento da API foi utilizado FastAPI 0.116.1 que é um framework Python para o desenvolvimento ágil e com documentação automática, isso porque essa

tecnologia cria automaticamente uma acessos explicativos e funcionais as requisições criadas, permitindo teste e a leitura de estrutaras de respostas e de entradas requeridas.

O banco de dados foi feito com MongoDB Compass 1.46.8 que é classificado como não relacional, o que permite, por exemplo, que seus registros de uma mesma classe tenham formatos e tamanho diferentes entre si. Isso faz com que MongoDB seja ideal para essa aplicação

4.4. Estratégia de desenvolvimento

Adotou-se a metodologia incremental, permitindo a entrega de partes funcionais do sistema em ciclos. Essa abordagem possibilitou testes e ajustes constantes, aumentando a qualidade do produto (PRESSMAN; MAXIM, 2021). Cada incremento contemplou o levantamento e análise de requisitos seguido da implementação de funcionalidades com teste destas, à medida que eram construídas. Posteriormente foram feitos avaliações e ajustes/melhorias. Dessa forma ciclos aprimoram a ferramenta a cada vez

4.5. Desenvolvimento do Sistema

Este capítulo descreve as etapas, decisões e resultados obtidos durante a construção do protótipo de sistema PDV para *delivery*, desenvolvido com **FastAPI**, **Kivy** e **MongoDB**. A implementação seguiu abordagem incremental, com entregas parciais e testes contínuos.

4.6. Levantamento de requisitos funcionais e não funcionais

Requisitos funcionais

- Cadastrar, ler, editar e remover produtos.
- Cadastrar, ler, editar e remover pedidos.
- Atualizar status de pedidos (em preparo, em entrega, finalizado).
- Calcular automaticamente valores de pedidos.
- Calcular e exibir relatórios de vendas.
- Autenticar usuários com credenciais seguras.

Requisitos não funcionais

- Adaptabilidade à logística do estabelecimento (flexibilidade para edições, inclusive pós-encerramento de pedidos).

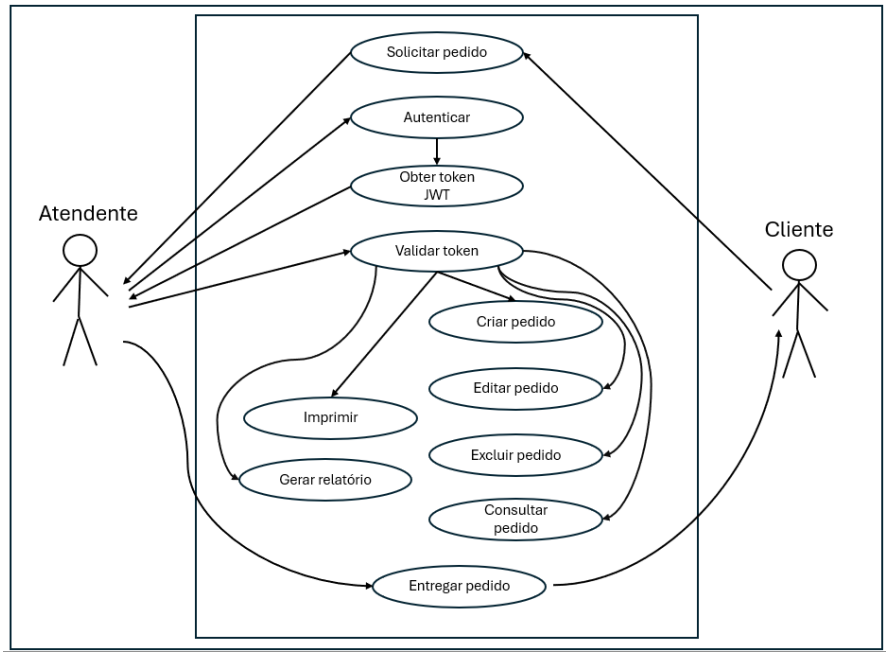
- Manutenibilidade: código modular e documentação mínima.
- Disponibilidade/robustez: operação estável em uso contínuo.
- Desempenho: respostas adequadas sob múltiplas requisições simultâneas.
- Segurança: autenticação/autorização, proteção de dados e segredos.
- Usabilidade: interface clara e responsiva, com baixa curva de aprendizado.

Diagrama de casos de uso

O diagrama de casos de uso contempla os seguintes atores:

- **Atendente**: registra pedidos, atualiza status e consulta dados.
- **Cliente:** solicita pedido, efetua pagamento, recebe o pedido.

Figura 1 *- Diagram de caso de uso*



Fonte: Desenvolvida pelo autor deste trabalho

Na Figura 1 podemos ver o ciclo completo do atendimento que se inicia quando o cliente solicita um pedido. Após a solicitação o atendente, se não estiver logado, deve fazer login para garantir acesso as operações do sistema. Depois de logado a cada requisição feita o token daquele usuário é enviado no header, em português cabeçalho. Posteriormente o atendente vai criar um novo pedido, inserir itens. Esses itens terão seus valores calculados e retornados. O cliente então informa as formas de pagamento, o atendente as registra, recebe o valor, entrega o pedido ou envia o pedido e então pedido é encerrado e bloqueado para edição.

Implementação da API com FastAPI

A API foi desenvolvida em Python 3.10 utilizando a FastAPI, estruturada em módulos Rotas que foram definidas utilizando (@app.get, @app.post, etc.); Modelos de classes criados com Pydantic para validação de dados; e Conexão gerenciada com a biblioteca motor para acesso assíncrono ao MongoDB.implementada com OAuth2 e JWT para autenticação.

Figura 2 *- Diretórios front end -*

Fonte: Desenvolvida pelo autor deste trabalho

Na Figura 2 estão os nomes dos arquivos do diretório de frontend/API. O arquivo main.py faz importações dos demais para que configurações, do arquivo config.py, e dados de conexão com o banco de dados do arquivo db.py sejam carregados, por exemplo. Essa estrutura segmentada proporciona organização e torna mais fácil a busca por erros em código mais sucintos e específicos.

Figura 3 *- Código do arquivo main.py*

```
1   from fastapi import FastAPI
2   from .db import init_db
3   from .routers import auth, pedidos
4
5   app = FastAPI(title="PDV Delivery API")
6
7
8   app.include_router(auth.router, prefix="/auth")
9   app.include_router(pedidos.router)  # sem prefixo
```

Fonte: Desenvolvida pelo autor deste trabalho

Pode-se ver na Figura 3 as importações de rotas que foram criadas. Cada uma possui seu arquivo específico, como visto na Figura 2. Em seus respetivos arquivos estão as requisições HTTP próprias de cada uma. Essas requisições são o que permite o trafego de informação entre banco de dados e interface do usuário.

4.7. Implementação da interface com Kivy

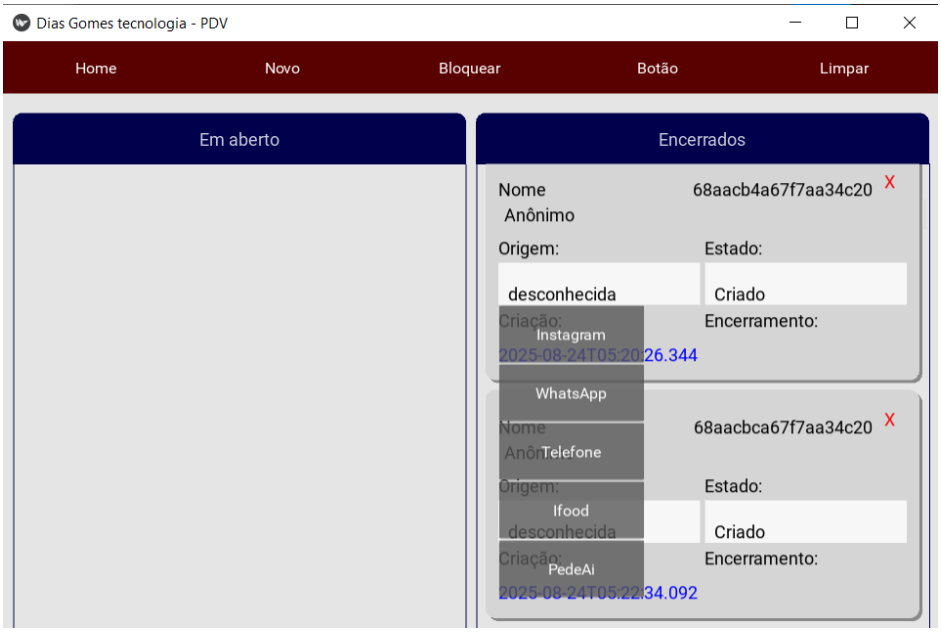
A interface foi construída utilizando a biblioteca ScreenManager, que gerencia as telas (Screen) que foram criadas: login, pedidos e relatórios. Nessa etapa os arquivos principais (main.py e main.kv) estão na pasta raiz do front-end, cada tela segue essa mesma organização: um arquivo .py e outro .kv. Cada screen fica dentro de uma pasta com mesmo nome e de lá são importadas para a tela principal.

Figura 4 *- Caixa de diálogo para login*

Fonte: Desenvolvida pelo autor deste trabalho

Na Figura 5 podemos notar o nível de personalização no painel de login. Neste painel foi configurado para que os elementos reajam a sobreposição do cursor do mouse, o botão atem no estado normal e sob o mouse tem outra aparência ao ser clicado. O campo de senha pode ter o conteúdo visualizado enquanto a figura de visão bloqueada e clicado. Por fim existe um exibidor de texto abaixo do botão de entrada para que mensagens como “Usuário ou senha incorreto” sejam exibidas dependendo da situação.

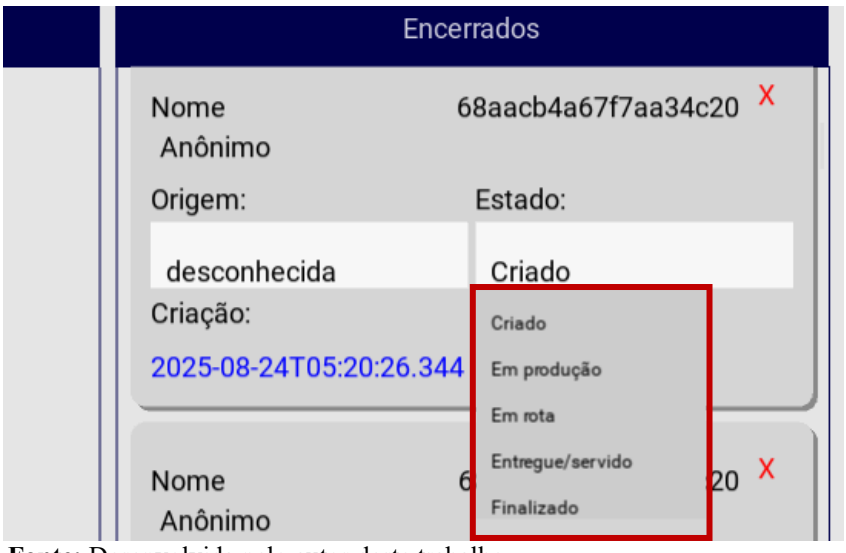
Figura 5 *- Tela de gestão de pedidos*



Fonte: Desenvolvida pelo autor deste trabalho

Nessa imagem, vista na Figura 5 vemos a tela de gestão de pedidos, os pedidos estão listados na coluna da direita. No “x” vermelho de cada pedido acontece a sua respectiva exclusão. Ele apresenta informações como origem, que é por onde o pedido foi feito; Estado que pode ser “criado”, “em produção”, “em rota” ou “encerrado”. Os botões do painel superior são o mesmo componente do botão de login. A personalização de um componente que pode ser chamado diversas vezes permite que, se necessário, sejam feitas mudanças em todos esses botões de uma só vez caso seja feita a mudança de estilo no componente pai.

Figura 6 *- Abertura das opções de estado de pedido*



Fonte: Desenvolvida pelo autor deste trabalho

4.8. Integração com MongoDB

O banco de dados MongoDB foi configurado localmente, com coleções: produtos e pedidos. Cada documomento segue um formato JSON/BSON. A conexão assíncrona foi feita com motor.motor_asyncio, permitindo operações simultâneas de leitura e escrita.

4.9. Funcionalidades desenvolvidas

- Cadastro e gerenciamento e produtos.
- Registro de pedidos e cálculo automático de valores.
- Atualização de status em tempo real.
- Relatórios resumidos de vendas.
- Autenticação de usuários.

4.10. Desafios encontrados e soluções aplicadas

Este trabalho por si só é resultado de soluções encontradas para problemas da primeira versão, que apesar de funcional estava distante do ideal de uma aplicação comercializável. Aquela versão se resumia a dois arquivos principais um main.py e um main.kv que realizavam todas as tarefas apresentadas até aqui.

Naquela versão os arquivos de pedidos eram .csv, da mesma forma os arquivos de menu e inclusive de usuário, o que era uma grande brecha de segurança. Essa arquitetura simples deixava o código muito longo e todas as classes, funções e componentes estavam ali. Um dos desafios enfrentados foi a curva de aprendizado em FastAPI, e Kivy. Compreender um cenário mais elaborado de arquivos interdependentes exigiu mais atenção e foco. FastAPI possui os arquivos de rotas, especificidades quanto as requisições autenticadas e validade de token que demandaram paciência.

Quanto ao Kivy, segmentar os arquivos de telas e componentes de forma que eles se comunicassem corretamente dentro da hierarquia de páginas gerou um tempo maior par o desenvolvimento dessa etapa, principalmente quando um componente realizava uma chamada para que outro recebesse a resposta. Felizmente, por fim, percebi que a organização das estruturas e da maneira de estudar são de extrema importância para elevar o nível do desenvolvimento a um patamar mais profissional e efetivo. Tanto FastAPI quanto Kivy são ferramentas que potencializam a produtividade, uma vez que haja dedicação no seu aprendizado.

4.11.Segurança e boas práticas

Adotou-se autenticação baseada em JWT com expiração e escopo por rota. Recomenda-se HTTPS em produção e proteção de segredos por variáveis de ambiente. Operações sensíveis devem seguir o princípio do menor privilégio e manter logs mínimos. Validações de entrada no back-end são realizadas com Pydantic.

5. RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos durante o desenvolvimento e execução do sistema PDV para delivery, bem como a análise crítica em relação aos objetivos propostos, desempenho técnico, benefícios e limitações, além da comparação com soluções similares no mercado.

5.1. Apresentação dos resultados

Para que seja enfatizada a experiencia do usuário e o impacto da implementação desta automação no ambiente de trabalho do estabelecimento foram feitas algumas perguntas para a então gerente do local. Que é quem demandou a ferramenta e a sua principal utilizadora. As perguntas e respostas estão a seguir:

Pergunta: Depois do sistema instalado houve mudanças na logística ou em alguma rotina do estabelecimento?

Resposta: “Com a implantação do sistema conseguimos agilizar o andamento dos pedidos, a soma de valores, o controle de saída de pedidos e extravio de comandas, além de agilizar o fechamento de caixa através da soma automática de valores, até mesmo pra consultar comandas antigas salvas no sistema. Pois a comandas manual perdia a qualidade e a visibilidade das informações depois de um longo tempo guardada”.

Pergunta: de 0 a 10 o aplicativo atendeu as demandas que fizeram ele necessário? Justifique

Resposta: “nota 8

De início precisou de alguns ajustes, mas com o tempo foi se aprimorando e conseguiu atender as expectativas”

Pergunta: de 0 a 10 quão facial era utilizá-lo? Justifique

Resposta: “Nota 8. De início foi necessário parar e estudar o funcionamento do aplicativo para utilizá-lo, mas logo ficava mais fácil.”

Ao analisarmos as respostas podemos perceber que a implantação do sistema PDV foi bem-sucedida e dentro do esperado para um cenário de adoção de software de automação. Impactos positivos foram gerados como: ganho operacional imediato, qualidade e preservação de informação, aderência as necessidades do negócio e mudanças positivas de rotina e logística.

5.2. Comparativo com soluções existentes no mercado

Quando comparado a sistemas comerciais de PDV e *delivery*, como iFood PDV, Linx Degust e Consumer, o protótipo apresenta vantagens e desvantagens:

Vantagens

- Custo zero de licenciamento e flexibilidade para customizações.
- Possibilidade de adaptação para diferentes segmentos de negócio.
- Arquitetura simples e modular, permitindo evolução incremental.

Desvantagens

- Menor conjunto de funcionalidades prontas (ex.: ausência de integração direta com marketplaces).
- Falta de suporte técnico profissional imediato.
- Recursos de análise e relatórios mais limitados que concorrentes consolidados.

De modo geral, o sistema cumpre o propósito de demonstrar a viabilidade técnica e a integração das tecnologias propostas, oferecendo uma base sólida para evoluir em direção a uma solução comercial competitiva.

5.3. Comparativo com soluções de mercado

Tabela 3 - Comparativo com soluções de mercado 1/2

Solução	Integrações	Pagamentos	Offline
Protótipo (este trabalho)	API REST (FastAPI); sem integração com marketplaces (planejada)	Não integrado (planejado)	Não (planejado com sincronização)
Integração iFood (APIs/Parceiros)	APIs oficiais (Produtos, Pedidos, Pedidos Hub)	Via parceiros/PDVs integrados	N/A (plataforma/serviço)
Linx Degust One	ERP+PDV; módulos para food service; PDV mobile	TEF/fiscal (conforme módulo)	Não informado publicamente
Consumer PDV	Integração com iFood; site de delivery; bot WhatsApp	Integração TEF (maquininha)	Desktop Offline/Online (edições)

Fonte: Desenvolvida pelo autor deste trabalho

Tabela 4 - Comparativo com soluções de mercado 2/2

Solução	Customização	Observações
Protótipo (este trabalho)	Alta (código-aberto/stack Python)	Relatórios básicos; GUI Kivy multiplataforma
Integração iFood (APIs/Parceiros)	Por integradores/parceiros	APIs permitem integrar PDV/ERP ao iFood
Linx Degust One	Configuração por módulos	Solução comercial consolidada no setor
Consumer PDV	Alta (planos e add-ons)	Versão grátis até 200 pedidos/mês

Fonte: Desenvolvida pelo autor deste trabalho

O comparativo das Tabelas 3 e 4 evidenciam que o protótipo desenvolvido neste trabalho, baseado em FastAPI, Kivy e MongoDB, se destaca pela alta capacidade de customização, por ser construído em código aberto e com tecnologias multiplataforma. No entanto, ainda apresenta limitações relevantes, como a ausência de integração direta com sistemas de pagamento e marketplaces, além de não dispor de funcionamento offline consolidado, aspectos que foram apenas planejados para fases futuras. Por outro lado, a proposta já demonstra vantagens acadêmicas e práticas, como relatórios básicos e uma interface gráfica adaptável, servindo como prova de conceito para sistemas mais acessíveis e flexíveis.

Em contraste, as soluções de mercado analisadas — como integrações oficiais do iFood, sistemas consolidados como Linx Degust One e plataformas voltadas para pequenos empreendedores, como Consumer PDV — apresentam maior maturidade na integração com APIs, gateways de pagamento e suporte offline. Além disso, oferecem módulos configuráveis e versões comerciais robustas, embora muitas vezes com custos elevados ou planos limitados. Assim, enquanto o protótipo se posiciona como uma alternativa em evolução, com ênfase em autonomia tecnológica e escalabilidade, as soluções de mercado refletem consolidação e integração imediata, mas com menor flexibilidade de personalização para o pequeno empreendedor.

6. CONCLUSÕES E TRABALHOS FUTUROS

O desenvolvimento deste protótipo de sistema PDV para delivery, utilizando FastAPI para o back-end, Kivy para a interface e MongoDB como banco de dados, demonstrou a viabilidade técnica de integrar tecnologias modernas e de código aberto para atender às demandas do setor de alimentação e entregas.
Os resultados obtidos comprovam que é possível criar uma solução flexível, escalável e de baixo custo, mantendo a performance necessária para operações de pequeno e médio porte.

6.1. Contribuições do trabalho

Este trabalho contribuiu em três frentes principais. Academicamente serviu como estudo de caso para aplicação de conceitos de Engenharia de Software e desenvolvimento full stack em Python. Tecnologicamente demonstrou a integração entre API, interface e banco de dados NoSQL em um sistema real, com arquitetura modular e escalável. De maneira prática ofereceu um protótipo funcional que pode ser adaptado e expandido para uso comercial, incentivando pequenos empreendedores a adotar soluções próprias.

6.2. Sugestões para melhorias e expansões

Para evoluir a solução proposta, sugerem-se as seguintes melhorias:
Implementar dashboards analíticos com gráficos e métricas de desempenho, integrar com gateways de pagamento reais, ampliando as opções de transação, adotar testes automatizados para aumentar a robustez e reduzir erros em produção. Otimizar a interface para melhor experiência em dispositivos móveis. Criar mecanismos para operação em locais com internet instável.

6.3. Considerações finais

O sistema desenvolvido cumpre seu papel como prova de conceito validada em ambiente real, evidenciando ganhos operacionais observados na aplicação piloto: maior agilidade no atendimento e no fechamento de caixa, redução de erros típicos de transcrição e perda de comandas, além de preservação e rastreabilidade das informações. Esses resultados derivam diretamente das funcionalidades implementadas — atualização de status em tempo real, CRUD de produtos e pedidos, e relatórios básicos — e demonstram que a solução atende às rotinas de balcão, produção e entrega com fluxos mais padronizados e previsíveis.

Do ponto de vista tecnológico, a arquitetura em três camadas mostrou-se adequada: o FastAPI estruturou as APIs REST com validação de dados e documentação automática, sustentando desempenho e clareza de contratos para evolução; o Kivy viabilizou uma interface responsiva e customizável, simplificando o fluxo do operador em desktop e abrindo caminho para cenários móveis; o MongoDB forneceu flexibilidade de esquema para pedidos heterogêneos (itens, adicionais, formas de pagamento), mantendo boa resposta em consultas frequentes do PDV. A autenticação JWT e a separação de responsabilidades contribuíram para segurança, manutenibilidade e testabilidade do conjunto.

Ao associar tecnologias de código aberto, boas práticas de desenvolvimento e um modelo escalável, a solução posiciona-se como alternativa viável para negócios que buscam autonomia tecnológica e redução de custos operacionais (sem mensalidades e com alta capacidade de personalização). Trata-se de um ponto de partida sólido: academicamente, permite avançar em pesquisas de eficiência, usabilidade e confiabilidade; comercialmente, oferece base para incorporar integração com meios de pagamento, dashboards analíticos, testes automatizados, melhorias de UX móvel e sincronização offline — evoluções já previstas no backlog e facilitadas pelas escolhas arquiteturais adotadas, sem demandar reestruturações profundas.

7. REFERÊNCIAS

ABRASEL. *Evolução do delivery no Brasil é tema de painel no 37º Congresso Abrasel; saiba como participar.* Brasília: Abrasel, 15 jul. 2025. Disponível em: https://abrasel.com.br/noticias/noticias/evolucao-delivery-brasil-painel-37-congresso-abrasel-saiba-como-participar/. Acesso em: 20 ago. 2025.

ABRASEL. *Movimentos no delivery empolgam mercado, diz Abrasel.* Brasília: Abrasel, 17 abr. 2025. Disponível em: https://abrasel.com.br/noticias/noticias/movimentos-no-delivery-empolgam-mercado-diz-abrasel/. Acesso em: 20 ago. 2025.

ABRASEL. *Delivery: tendências para este mercado em crescimento no Brasil.* Recife: Abrasel-PE, 31 jul. 2023. Disponível em: https://pe.abrasel.com.br/noticias/noticias/delivery-tendencias-para-este-mercado-em-crescimento-no-brasil/. Acesso em: 20 ago. 2025.

ABRAHÃO, A. *Arquitetura de sistemas distribuídos: desafios e tendências.* Revista Brasileira de Computação Aplicada, v. 12, n. 2, 2024.

ABUBAKAR, A. et al. *Development of cross-platform applications using Kivy: opportunities and challenges.* International Journal of Computer Applications, v. 183, n. 12, p. 23–29, 2021.

ARAUJO, V. M.; VAZQUEZ, J. A. Business and technical requirements of Software-as-a-Service: implications in Portuguese enterprise business context. *International Journal in Foundations of Computer Science & Technology*, v. 3, n. 6, 2013. Disponível em: https://arxiv.org/abs/1312.2243. Acesso em: 20 ago. 2025.

BAESSO, L. G. A. S.; ROMANI-DIAS, M. Desafios e oportunidades da adesão aos marketplaces de delivery: contribuições sob a luz da teoria de capacidades dinâmicas. *Revista FSA*, Teresina, v. 20, n. 12, p. 3–29, 2023. Disponível em: https://www4.unifsa.com.br/revista/index.php/fsa/article/view/2816/491494135. Acesso em: 20 ago. 2025.

BOŽIĆ, J. *Systematic review of NoSQL databases: performance and scalability aspects.* Journal of Big Data, v. 9, n. 44, 2022.

BRENER, F.; GUILHERME, Y.; SOARES, L. S. Pediline: solução SaaS para gerenciamento de catálogo e delivery. In: *Anais do XV Workshop de Sistemas de Informação (WSIS).* Porto Alegre: SBC, 2024. p. 131–135. Disponível em: https://sol.sbc.org.br/index.php/wsis/article/view/33686. Acesso em: 20 ago. 2025.

CHAUHAN, A. et al. Food Ordering Website “Cooked with Care”: An Application of MERN Stack. *International Journal of Advanced Computer Science and Applications*, v. 13, n. 10, p. 45–52, 2022.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures.* 2000. Tese (Doutorado em Ciência da Computação) – University of California, Irvine. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Acesso em: 20 ago. 2025.

FOWLER, M. *Patterns of enterprise application architecture.* Boston: Addison-Wesley, 2002.

GYŐRÖDI, C. A. et al. A comparative study of MongoDB and document-based MySQL for big data application data management. *Big Data and Cognitive Computing*, v. 6, n. 2, p. 49, 2022. DOI: 10.3390/bdcc6020049. Disponível em: https://www.mdpi.com/2504-2289/6/2/49. Acesso em: 20 ago. 2025.

HOSSAIN, M. et al. Layered architecture for scalable software systems. *IEEE Access*, v. 8, p. 112345–112359, 2020.

IMARC GROUP. *Point of Sale (POS) System Market: Global Industry Trends, Share, Size, Growth, Opportunity and Forecast 2025–2030.* IMARC Reports, 2025.

KIVY. *Kivy Documentation.* 2025. Disponível em: https://kivy.org/doc/stable/. Acesso em: 5 set. 2025.

LUTZ, M. *Learning Python.* 5. ed. Sebastopol: O'Reilly Media, 2013.

NORTHWOOD, C. *The full stack developer.* New York: Apress, 2018.

OECD. *The digital transformation of SMEs.* Paris: OECD Publishing, 2021. DOI: 10.1787/bdb9256a-en.

PYPL. *PYPL Popularity of Programming Language Index.* 2025. Disponível em: https://pypl.github.io/PYPL.html. Acesso em: 20 ago. 2025.

PYTHON SOFTWARE FOUNDATION. *Python documentation.* [s. l.]: PSF, [s. d.]. Disponível em: https://docs.python.org/3/. Acesso em: 5 set. 2025.

RAMÍREZ, S. *FastAPI Documentation.* 2018– . Disponível em: https://fastapi.tiangolo.com/. Acesso em: 5 set. 2025.

RICHARDSON, L.; RUBY, S. *RESTful web services.* Sebastopol: O'Reilly, 2007.

ROMÃO, E. et al. *Análise comparativa de sistemas de ponto de venda no contexto brasileiro.* Revista de Engenharia e Pesquisa Aplicada, v. 9, n. 1, 2024.

SANTOS, J.; LIMA, P. Comparative study of Python frameworks for web development. *Journal of Web Engineering*, v. 21, n. 3, p. 233–245, 2022.

SMITH, M. 8 best practices for building FastAPI and MongoDB applications. *MongoDB Developer Center*, 23 abr. 2024. Disponível em: https://www.mongodb.com/developer/languages/python/. Acesso em: 5 set. 2025.

SOMMERVILLE, I. *Software engineering.* 10. ed. Harlow: Pearson, 2019.

SYSMIDDLE. *Omnichannel trends in POS systems: integration challenges and opportunities.* Sysmiddle White Paper, 2024.

TANENBAUM, A. S.; WETHERALL, D. J. *Computer networks.* 5. ed. Upper Saddle River: Prentice Hall, 2011.

TIOBE. *TIOBE Index.* 2025. Disponível em: https://www.tiobe.com/tiobe-index/. Acesso em: 20 ago. 2025.

VAN ROSSUM, G.; DRAKE, F. L. *The Python language reference manual.* Python Software Foundation, 2022. Disponível em: https://docs.python.org/3/reference/. Acesso em: 5 set. 2025.

WAZLAWICK, R. S. *Engenharia de software: conceitos e práticas.* 3. ed. Rio de Janeiro: Elsevier, 2021.

WEF – WORLD ECONOMIC FORUM. *What more can be done to help small businesses thrive in the digital economy?* 20 jan. 2025. Disponível em: https://www.weforum.org/stories/2025/01/digital-economy-small-businesses/. Acesso em: 3 set. 2025.

ANEXOS

Anexo A – Estrutura de Diretórios e Arquivos do Projeto