**PÓS-GRADUAÇÃO EM DESENVOLVIMENTO FULL STACK**

**DESENVOLVIMENTO DE UMA APLICAÇÃO WEB COM IOT PARA MONETIZAÇÃO DO MARKETPLACE INFORMAL DE CAFÉ E OUTROS CONSUMÍVEIS EM UM AMBIENTE EMPRESARIAL**

LUIZ GUSTAVO PFITSCHER E FELDMANN

Orientador: Eduardo Campos Pellanda

2025



**SUMÁRIO**

[1. Contextualização da proposta 3](#_Toc189414993)

[2. Objetivos da construção da solução 5](#_Toc189414994)

[3. Elaboração da jornada do usuário 7](#_Toc189414995)

[4. Apelo mercadológico da solução 9](#_Toc189414996)

[5. Ciclo de desenvolvimento da solução 11](#_Toc189414997)

[5.1. Análise de Viabilidade e MVP 11](#_Toc189414998)

[5.2. Elaboração da Arquitetura de Software 12](#_Toc189414999)

[5.3. Preparação dos Ambientes de Teste e Produção e Pipelines de CI/CD 12](#_Toc189415000)

[5.4. Desenvolvimento Incremental 12](#_Toc189415001)

[6. Mockup da proposta de solução 13](#_Toc189415002)

[7. Arquitetura de Software 16](#_Toc189415003)

[7.1. Front-End 17](#_Toc189415004)

[7.2. Back-end 18](#_Toc189415005)

[7.3. Banco de Dados 19](#_Toc189415006)

[7.4. Segurança e Autenticação 20](#_Toc189415007)

[7.5. APIs 21](#_Toc189415008)

[7.6. IoT 21](#_Toc189415009)

[8. Validação da solução 23](#_Toc189415010)

[8.1. Uso do *TypeScript* 23](#_Toc189415011)

[8.2. Uso do *Next.js* 23](#_Toc189415012)

[8.3. Testes Automatizados 23](#_Toc189415013)

[8.4. Ambiente de Desenvolvimento 24](#_Toc189415014)

[8.5. Ambiente de Testes e Pipeline de CI/CD 24](#_Toc189415015)

[8.6. Ambiente de Produção 24](#_Toc189415016)

[9. Registros das evidências do projeto 25](#_Toc189415017)

[10. Considerações finais e expectativas 26](#_Toc189415018)

[10.1. Trabalhos Futuros 26](#_Toc189415019)

[Referências 28](#_Toc189415020)

# Contextualização da proposta

O desenvolvimento da solução foi motivado pela percepção de um problema recorrente no ambiente de trabalho: a gestão heterogênea dos clubes de café dentro da empresa. Devido à diversidade da gostos, hábitos e propensão para arcar com custos de cada indivíduo, surgem de forma orgânica pequenos clubes de café gerenciados por diferentes equipes. Cada um conta com recursos, grau de investimentos e sistemas de gestão próprios. Essas associações enfrentam uma série de desafios como:

1. **Gestão manual de pagamentos e membros:**

A responsabilidade de administrar os pagamentos, registrar o consumo, projetar a reposição de estoque e até mesmo encomendar os insumos recai de maneira desproporcional sobre um membro do grupo, o qual também deve fiscalizar o uso privado e restrito aos participantes. O ônus dessas atribuições muitas vezes justifica uma compensação ao responsável, o que pode gerar desacordos entre os demais.

1. **Falta de transparência e praticidade no uso dos recursos:**

Devido à natureza informal e descentralizada da organização, a prestação de contas e a tomada de decisões sobre alocação de recursos captados podem ser insatisfatórias, o que contribui para ineficiência operacional e baixa adesão dos consumidores.

1. **Dificuldade de acesso ao equipamento de outros times:**

Quando os colaboradores se deslocam para outras áreas da empresa, a natureza privada dos equipamentos e insumos utilizados por cada time impede o uso compartilhado por visitantes. A realização de acordos informais e bilaterais entre grupos, além de ser pouco prática, é difícil de fiscalizar, o que evidencia a necessidade de um sistema padronizado para gerenciar e controlar o acesso aos recursos de forma justa.

A proposta consiste na criação de um sistema integrado que une hardware *IoT* e software, proporcionando uma experiência mais justa, eficiente e simplificada tanto para consumidores quanto para gestores dos clubes de café ou outras iniciativas informais que ofereçam produtos dentro do ambiente corporativo. Cada equipamento — seja cafeteira, moedor de grãos ou outros utensílios — será equipado com dispositivos *IoT*, permitindo o controle remoto e integração com um aplicativo web. O *app* atuará como um *marketplace*, oferecendo as seguintes funcionalidades:

1. **Pagamento Digital Simplificado:**

Os usuários poderão pagar pelos produtos utilizando métodos de pagamento amplamente acessíveis, como PIX ou cartão de crédito/débito.

1. **Conta com Saldo Comum:**

A plataforma permitirá que os usuários tenham um saldo único, que poderá ser utilizado para consumo de produtos e serviços distribuídos por diversos setores da empresa, de maneira unificada.

1. **Democratização e Otimização do Acesso aos Recursos:**

A proposta busca aprimorar o aproveitamento dos equipamentos e recursos, permitindo um compartilhamento mais eficiente entre os times. Isso resulta em um maior retorno sobre o investimento, com a maximização do uso dos recursos disponíveis. Como consequência, o tempo de *payback* é reduzido, o que incentiva a realização de investimentos adicionais para melhorar a infraestrutura e expandir a solução.

1. **Simplificação da Gestão para os Administradores:**

A plataforma tem o potencial de gerar relatórios de consumo, projeções de estoque, e permitir o gerenciamento centralizado dos custos, simplificando as tarefas de administração e oferecendo uma visão clara e organizada para os gestores de cada clubes de café.

Este projeto é um piloto, com um *MVP* em uma organização específica, que busca promover inovação e transformação digital, com potencial para ser escalado e aplicado em diversas empresas.

# Objetivos da construção da solução

Este projeto prático tem como metas aprofundar e consolidar os conhecimentos adquiridos ao longo do curso, abaixo relacionadas em objetivos estratégicos e seus respectivos objetivos específicos:

1. **Reforçar o aspecto empreendedor na construção de soluções digitais inovadoras**:
   1. Utilizar o *Business Model Canvas* como ferramenta para mapear e estruturar o apelo mercadológico da solução.
2. **Mitigar problemas decorrentes da codificação web em *JavaScript*:**
   1. Aperfeiçoar a fluência em *TypeScript* como alternativa para implementações mais robustas e seguras.
3. **Familiarizar-se com ferramentas de prototipagem de interfaces gráficas:**
   1. Desenvolver habilidades básicas no uso do *Figma*.
4. **Dominar o uso de um *framework* web orientado a componentes para desenvolvimento de interfaces de usuário:**
   1. Especializar-se no uso de *React* para construção de interfaces dinâmicas e escaláveis.
5. **Explorar o paradigma de renderização no servidor:**
   1. Aprender a utilizar o *NextJS*.
   2. Identificar os casos de uso de *server-side components* e *client-side components,* compreendendo suas respectivas vantagens e desafios.
6. **Aprimorar-se no desenvolvimento e uso de *API*s:**
   1. Implementar autenticação para comunicação segura no consumo de dados do servidor por meio de *API*s *RESTful*.
7. **Adquirir experiência em ferramentas de teste automatizado de software:**
   1. Utilizar o *Playwright* para controle de qualidade da solução desenvolvida.
8. **Aplicar práticas *DevOps*:**
   1. Construção de uma pipeline CI/CD completa.
   2. Conteinerização da aplicação e seu ambiente utilizando *Docker*.
   3. Automatização de testes e implantação por meio de *GitHub Actions*.
9. **Implantar a aplicação em uma plataforma de nuvem:**
   1. Hospedar o projeto desenvolvido na *Vercel* como evidência de entrega.

Com o cumprimento desses objetivos, busca-se aplicar os conhecimentos do curso a aspectos centrais da realidade do mercado, fortalecendo a base técnica para o crescimento profissional.

# Elaboração da jornada do usuário

Ao acessar a página inicial do website, um novo usuário deverá primeiramente realizar o cadastro de uma nova conta, pelo preenchimento de um formulário com campos de nome, sobrenome, endereço eletrônico e criação de uma senha. Como recurso de proteção contra a inscrição de contas falsas, é obrigatório ao novo usuário acessar um link enviado para o e-mail de cadastro para verificar sua identidade antes de poder autenticar-se.

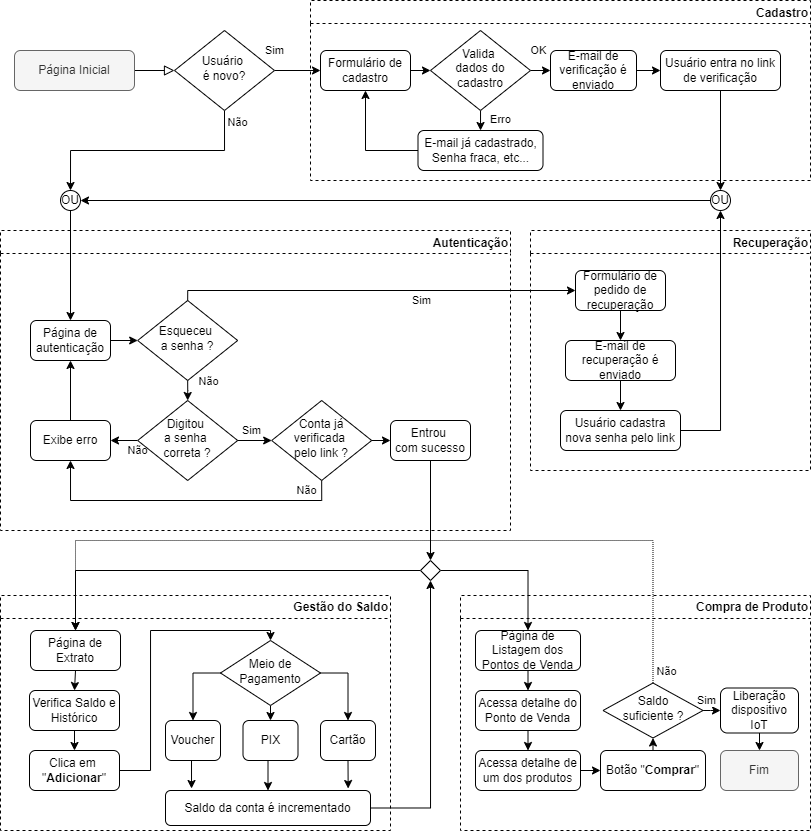
Um usuário possuidor de uma conta já verificada pode navegar da página inicial até a página de autenticação, onde deverá informar o endereço de e-mail associado à sua conta e a senha correspondente. Em caso de esquecimento da senha, poderá seguir o link apropriado para redefinição dela, o qual redireciona para um formulário solicitando informar o endereço de e-mail para o qual será enviado um link com instruções. Esse link levará o usuário a uma página onde uma nova senha poderá ser cadastrada.

Uma vez autenticado o usuário, terá acesso a um menu principal oferecendo as ações disponíveis. Uma jornada de uso típica pode começar pela consulta ao seu saldo financeiro pré-pago, e ao histórico de movimentações, como depósitos e compras realizadas. Tais informações estão disponíveis em uma página dedicada, onde também é oferecido um link que leva à área de adição de saldo. Nessa área, o usuário encontra alguns métodos de pagamentos possíveis para efetuar o depósito de valores que poderão ser utilizados na compra de produtos.

A partir do menu lateral também é possível acessar uma listagem de pontos de venda, cada qual descrito por título, endereço, fotos e média de notas das avaliações deixadas por outros usuários. Cada item da lista de pontos de venda redireciona a uma página do seu respectivo detalhamento onde, além dos dados básicos supracitados, também constam horários de funcionamento, e lista de produtos oferecidos no local. Ao usuário é facultado deixar sua própria avaliação nesta página, clicando no botão apropriado, que o leva a uma caixa de diálogo que recebe uma nota em estrelas e um comentário em texto.

Cada item da lista de produtos leva a uma página de detalhamento, onde constam título, descrição, fotos, preço e o botão “Comprar”. A jornada do usuário culmina na compra de um produto por meio deste botão. Após a confirmação explícita dessa intenção em uma caixa de diálogo, o preço do item é debitado da conta e um comando é enviado para o dispositivo *IoT* correspondente no espaço físico do ponto de venda, liberando o produto para o consumo do usuário.

**Figura 3.1** – Fluxograma detalhando a jornada do usuário.



# Apelo mercadológico da solução

A crescente necessidade de otimização de processos e melhora da experiência dos colaboradores nas empresas abre espaço para soluções inovadoras que atendem a demandas específicas. No caso dos clubes de café, essa é uma necessidade que frequentemente é negligenciada, mas que pode ter impacto direto na produtividade e no bem-estar dos colaboradores (Smith, 2005). A proposta visa suprir essa lacuna, criando uma solução prática e escalável que facilita tanto o controle dos recursos como a experiência de consumo, alinhando-se com as tendências de transformação digital nas organizações.

A proposta permite que os times se organizem de forma autônoma para expor os seus recursos, como cafeteiras e moedores, sem centralizar a responsabilidade na gestão corporativa, nem onerar um determinado colaborador com responsabilidades de gerir pagamentos ou fiscalizar uso.

Cada equipe pode aderir conforme sua conveniência, enquanto a plataforma oferece uma interface homogênea e transparente para todos os usuários, garantindo uma experiência consistente e simplificada. Isso promove a liberdade e flexibilidade, mantendo a disponibilidade no uso dos recursos, e cria um ecossistema colaborativo onde a gestão dos equipamentos é mais eficiente e acessível, sem a necessidade de uma estrutura formal.

Para concretizar esse modelo de negócios, as principais despesas incorridas são os serviços de nuvem e hospedagem, fundamentais para o funcionamento da plataforma. Também será necessário arcar com as comissões dos operadores de pagamento digital que facilitam as transações entre os consumidores dos produtos e os gestores dos estabelecimentos. Em contraponto, a principal fonte de receitas vem da taxa cobrada sobre cada transação realizada por meio do aplicativo. Além disso, a venda de dados da base de usuários pode gerar uma segunda linha de receita, desde que respeitando as normas de privacidade e anonimização das estatísticas.

Estes e outros aspectos da oferta de valor e do modelo de negócios estão mais bem explicados nos *Canvas* da Figura 4.1 e Figura 4.2, respectivamente.

|  |
| --- |
| **Figura 4.1** - *Value Proposition Canvas*. |
| Gerir pagamentos  Manter estoque  Fiscalizar uso  Melhor retorno sobre investimentos em equipamentos  Cultura de colaboração e compartilhamento  Experiência transparente no consumo  Automação na gestão de pagamentos  Gestão simplificada e autônoma  Acesso facilitado aos produtos de outros times  Equipamentos de outros times inacessíveis  Gestor sobrecarregado com tarefas  Transparência nas transações e uso dos recursos  Intermédio de pagamentos  Aplicativo plataforma de marketplace  Hardware IoT  Compartilhamento p/ monetizar recursos privados |

|  |
| --- |
| **Figura 4.2** - *Business Model Canvas*. |
| Vide **Figura 4.1.**  Suporte  Informe de estatísticas  Operadores de pagamento digital  Representantes dos clubes de café em cada time  E-mail e notificações  Aplicativo Web  Colaboradores de empresas que participam de clubes do café  Venda de dados de usuários  Taxa sobre transação  Divulgação  Comissão dos operadores de pagamento  Serviços de nuvem e hospedagem  Equipamentos instalados  Base de usuários  Pagamento  Divulgação  Instalação de IOT  Desenvolvimento do app |

# Ciclo de desenvolvimento da solução

Antes do início do desenvolvimento da solução foi definido um plano de projeto, que foi fielmente executado conforme relatado a seguir:

## Análise de Viabilidade e MVP

A partir da ideia inicial da aplicação foi realizado um *survey* na empresa, identificando os possíveis pontos de venda e seus respectivos produtos, máquinas e dispositivos utilizados no preparo, e a sua possibilidade de acionamento por dispositivos *IoT*. Adicionalmente foram conduzidas entrevistas com potenciais usuários da solução proposta para mapear o valor percebido e suas expectativas.

Essa análise culminou na seleção de um local mais adequado para a implantação do “piloto” do projeto. Foi escolhida uma cozinha com grande circulação de pessoas e frequentada pelos times mais exigentes em relação a qualidade do café que consomem. Esse público, por consequência, despende maiores investimentos em insumos e equipamentos. Decidiu-se por monetizar o preparo do café *espresso* de uma cafeteira italiana, que é de fácil conexão a um interruptor inteligente conectado à rede sem fio.

Também foi realizada a delimitação de escopo do produto mínimo viável. Para não incorrer em custos de operadora de meios de pagamento, foi decidido por inicialmente implementar apenas o pagamento por *voucher*, gerado por intermédio do Autor, atuando como intermediário dos pagamentos.

Visando acelerar o *time to market*, foi inicialmente postergado o desenvolvimento de um painel de administrador do sistema. Durante esse período as informações devem ser inseridas e monitoradas diretamente pelo banco de dados. Esses e outros aspectos complementares ficam pendentes como trabalhos futuros, conforme descrito no Capítulo 10.

## Elaboração da Arquitetura de Software

Uma vez definidos os requisitos do projeto, foi realizada uma pesquisa sobre as tecnologias apresentadas no Curso, selecionando as mais adequadas para a implementação da solução. O projeto de arquitetura buscou equilibrar o menor tempo de desenvolvimento e menores custos de hospedagem dos serviços (nesse primeiro momento), mas sem comprometer o potencial de escalabilidade futura. Estes aspectos estão mais bem descritos no Capítulo 7.

## Preparação dos Ambientes de Teste e Produção e Pipelines de CI/CD

A preocupação com a qualidade esteve presente desde o início do ciclo de desenvolvimento, o que levou à adoção de práticas de *DevOps*. A configuração inicial dos ambientes de teste e produção e da *pipeline* de *CI/CD* foi realizada antes do início da codificação das funcionalidades. Esses preparativos asseguraram que o desenvolvimento subsequente ocorresse em um fluxo controlado e confiável, como descrito no Capítulo 8.

## Desenvolvimento Incremental

O desenvolvimento da solução seguiu uma abordagem incremental, com a implementação das funcionalidades de forma sequencial, ordenada de maneira lógica por domínios, e priorizando os componentes essenciais para o MVP. O processo começou com a conexão ao banco de dados, seguida pela implementação de cadastro e autenticação de usuários, listagem de pontos de venda e seus respectivos produtos, fluxos de pagamento e extratos financeiros, e culminando com a funcionalidade de compra.

A configuração prévia dos ambientes de testes e *pipelines* de *CI/CD* possibilitou a execução de testes automatizados constantes, garantindo a integração contínua de novas funcionalidades. Além disso, foi possível realizar demonstrações frequentes para os usuários e *stakeholders* por meio do ambiente hospedado, o que facilitou a coleta de *feedback* em tempo real e a validação constante das funcionalidades implementadas.

# Mockup da proposta de solução

A ferramenta [*Figma*](https://www.figma.com/design/bAJduV9UPCEe9Aokj37ooP/TCC-Fullstack-PURCS) foi utilizada para a prototipação da interface gráfica da solução desenvolvida, cobrindo os fluxos de navegação tratados no Capítulo 3. Procurou-se oferecer uma interface minimalista e objetiva, respeitando aos padrões tradicionais de navegação e usabilidade familiares à maioria dos usuários.

Foi adotada a *MUI* como *design system* devido à sua rica oferta de componentes maduros e de fácil de uso, com vasta documentação e exemplos, integração no *Figma*, e por já ser familiar aos usuários, em vista de sua ampla difusão.

Optou-se por priorizar o design da versão *mobile*, uma vez que se espera que a solução seja utilizada por usuários em trânsito pelos pontos de venda e suas vizinhanças, sugerindo o menor uso da versão para *desktop*. Ainda assim, o design responsivo permite a utilização em dispositivos dos mais variados fatores de forma.

|  |  |  |
| --- | --- | --- |
| **Figura 6.1** - Sequência de páginas no fluxo de cadastro:  (a) formulário de dados cadastrais; (b) aviso sobre envio do e-mail de confirmação; (c) página acessada pelo link enviado no e-mail de confirmação. | | |
|  |  |  |
| **(a)** | **(b)** | **(c)** |
| **Figura 6.2** - Sequência de páginas do fluxo de entrada: (a) página inicial do *site*; (b) formulário de credenciais; (c) menu para navegação na área autenticada. | | |
|  |  |  |
| **(a)** | **(b)** | **(c)** |

|  |  |  |
| --- | --- | --- |
| **Figura 6.3** - Fluxo alternativo para recuperação de senha: (a) solicitação da recuperação; (b) confirmação do envio do link; (c) formulário para nova senha. | | |
|  |  |  |
| **(a)** | **(b)** | **(c)** |

|  |  |
| --- | --- |
| **Figura 6.4** -Páginas de finanças do usuário: (a) listagem de transações realizadas; (b) métodos de pagamento para adição de saldo. | |
|  |  |
| **(a)** | **(b)** |

|  |  |  |
| --- | --- | --- |
| **Figura 6.5.** Jornada de compra: (a) listagem dos pontos de venda; (b) detalhamento do ponto de venda; (c) detalhamento do produto. | | |
|  |  |  |
| **(a)** | **(b)** | **(c)** |

# Arquitetura de Software

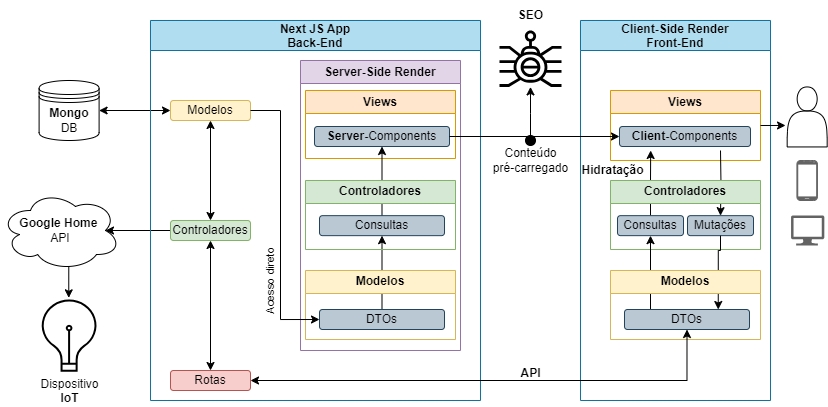
A arquitetura da solução foi projetada para equilibrar simplicidade, manutenibilidade, organização do código, modularidade e eficiência. A escolha do *Next.js* como framework central permite que o *front-end* e o *back-end* coexistam no mesmo container, resultando em um mono-repositório ou serviço monolítico. No entanto, apesar dessa estrutura consolidada, a aplicação é organizada internamente de maneira modular, dividida em camadas como modelos de entidades, controladores e rotas (Martin, 2017), propiciando um código bem estruturado e de fácil manutenção e extensão.

Tanto a renderização no servidor quanto a renderização no cliente aderem ao padrão *Model-Controller-View*, o que permite uma clara separação de responsabilidades e melhor organização do código. Os *DTO*s são reutilizados em ambos os lados, promovendo consistência na comunicação entre as camadas.

As *API*s desempenham um papel propositalmente reduzido dentro dessa arquitetura: elas são responsáveis apenas pela hidratação de dados (como consulta ao saldo do usuário) e mutações (como a realização de compras), mantendo o foco na parte dinâmica da aplicação. Por outro lado, as listagens de páginas de lojas e produtos são geradas puramente no lado do servidor, por meio de acesso direto aos modelos de *ORM*, valendo-se de funcionalidades próprias do *Next JS,* como as *server-functions*.

Essa abordagem permite que as páginas sejam renderizadas e entregues com o conteúdo estático (aquele independente de usuário) já disponível, o que melhora a visibilidade e indexação nos motores de busca, além de oferecer menor tempo de carregamento da página para o usuário final (Krause, 2024).

**Figura 7.1.** Visão geral da arquitetura da solução.



## Front-End

Para aprimorar a experiência do usuário e manter a consistência visual em toda a aplicação, foi adotado o *Material UI* (MUI) como o sistema de design. O *MUI* oferece uma vasta biblioteca de componentes prontos e altamente customizáveis, como botões, formulários e tabelas, que já contam com as melhores práticas de usabilidade e acessibilidade, garantindo uma interface moderna, responsiva e familiar. Esses componentes podem ser combinados e associados (como um “*Lego*”), para criar visualizações complexas e customizáveis a partir de elementos atômicos e minimalistas. Isso permite uma criação ágil das interfaces, mantendo um aspecto consistente e maximizando a produtividade em um curto tempo, ideal para um *MVP*.

Para gerenciar e otimizar o consumo de dados dinâmicos na aplicação, foi utilizado o *Tanstack React Query*. Essa é uma poderosa biblioteca para o gerenciamento de dados assíncronos, como chamadas *API* para hidratação de componentes. Ela facilita o consumo de dados ao fornecer uma abstração sobre requisições HTTP, oferecendo diversas funcionalidades (Afonso, 2023):

Pelo uso de cache, a biblioteca armazena automaticamente as respostas das requisições, evitando chamadas repetidas e reduzindo a sobrecarga de rede. Além disso, a expiração periódica atualiza os dados automaticamente de maneira transparente a cada intervalo de tempo, garantindo a revalidação dos dados antigos para que apenas os dados mais atualizados sejam exibidos. Similarmente, a biblioteca gerencia a invalidação e atualização dos dados de requisição quando ocorrem mutações sobre as fontes desses dados, melhorando a consistência global da interface. Todos esses recursos são implementados por meio de *hooks* inteligentes que apenas escalonam a renderização dos componentes caso haja de fato alguma alteração nos dados finais, proporcionando uma experiência de usuário fluida e em tempo real.

## Back-end

O *back-end* da aplicação foi desenvolvido utilizando os recursos oferecidos pelo *Next.js*, com foco no uso de *server-side components* sempre que possível. Essa abordagem garante que tais componentes sejam renderizados no lado do servidor, proporcionando páginas com conteúdo pré-carregado. Isso melhora a performance percebida pelo usuário, refletida no tempo de carregamento, além de permitir a otimização para motores de busca (SEO).

Ao se trabalhar com esse tipo de componentes, adota-se um paradigma de implementação diferente, devido a proibição no uso de *hooks* que ocasionem comportamentos dinâmicos, como *useState* ou *useEffect*. Em vez disso, os dados devem ser estáticos do ponto de vista dos componentes, sendo passados diretamente como propriedades. Para as consultas de dados, são utilizadas as chamadas *server functions*, próprias do Next.js. Essas funções, que são explicitamente marcadas com a diretiva *"use server"*, permitem que o servidor execute a lógica de negócios diretamente, sem a necessidade de chamadas *API*s. Assim, componentes *server-side* têm acesso direto aos controladores que utilizam a *ORM* (Tazetdinov, 2023). Essa simplificação evita a sobrecarga de uma chamada utilizando a rede, permitindo o acesso imediato aos dados armazenados no banco de dados, melhorando ainda mais a performance e a experiência do usuário.

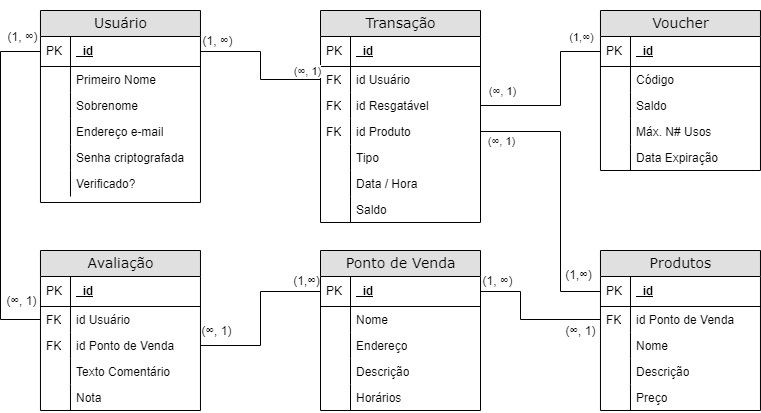
## Banco de Dados

Para camada de persistência de dados, a aplicação utiliza o *MongoDB*, um banco de dados *NoSQL* amplamente utilizado. Ele armazena dados em documentos no formato *JSON*, permitindo que as informações seja mais complexas do que uma simples tabela, e evitando um esquema rígido como ocorre nos bancos de dados relacionais. Isso facilita a adaptação a mudanças no modelo de dados ao longo do tempo, ideal para uma solução em fase de *MVP* (Holmes, 2013).

Para interagir com o *MongoDB* de forma mais eficiente, foi utilizada a biblioteca *Mongoose* como *ORM* (*Object-Relational Mapping*). Essa biblioteca facilita a comunicação com o banco de dados ao abstrair as operações de consulta, atualização e validação, fornecendo uma camada que simplifica essas interações. Ele permite a definição de modelos de dados diretamente no código *TypeScript*, assegurando que as classes e interfaces sejam consistentes com os dados armazenados no banco.

Isso traz vantagens significativas, como aumento da produtividade e manutenibilidade do código. O uso de tipagem e validações ajuda a prevenir erros e inconsistências, além de permitir a definição de esquemas que estruturam os dados de forma mais organizada. Assim, conseguimos combinar a flexibilidade do *MongoDB* com recursos de organização e controle, mitigando as limitações típicas do modelo *NoSQL* e proporcionando uma solução mais robusta e confiável (Holmes, 2013).

**Figura 7.3.1.** Diagrama Entidade-Relacionamento.



## Segurança e Autenticação

A segurança das informações dos usuários é uma prioridade crescente nas arquiteturas contemporâneas, principalmente desde o advento da LGPD. A preocupação está não apenas na proteção do próprio sistema, mas também em garantir que informações eventualmente vazadas não possam ser reutilizadas.

Por isso, as senhas dos usuários recebem um sal antes da sua criptografia com *bcrypt* e subsequente armazenamento. O sal é um valor único e aleatório adicionado à senha antes do seu processamento, o que garante sua unicidade apesar de o usuário potencialmente utilizar a mesma senha em várias plataformas. Em caso de vazamento de dados do banco, as senhas originais em texto plano não podem ser descriptografadas, e mesmo as senhas na sua forma criptografada serão diferentes das utilizadas em outros locais, evitando ataques de repetição (Sullivan, 2011).

Para gerenciar a autenticação do usuário durante a sessão, foi utilizado o *JWT* (*JSON Web Token*). Esse é um padrão aberto para autenticação e autorização, no qual um *token* é gerado, assinado no servidor e então enviado ao cliente. Ele contém informações codificadas, como o ID do usuário e data de expiração da sessão, que podem ser rapidamente verificadas pelo servidor a cada requisição subsequente. Ele é seguro pois não pode ser alterado sem invalidar a assinatura eletrônica gerada pelo servidor (IETF, 2015). A principal vantagem desse método é que ele elimina a necessidade de registrar sessões no banco de dados, pois todas as informações necessárias para autenticar o usuário já estão contidas no próprio *token*. Isso torna a solução mais escalável, pois reduz a sobrecarga associada ao armazenamento e gerenciamento de sessões.

Por fim, variáveis de ambiente são utilizadas para parametrizar os segredos necessários para o funcionamento da aplicação, como as credenciais do banco de dados, as chaves de assinatura do *JWT* e de criptografia das senhas de usuário. Em produção, esses valores são mantidos na solução de segredos da plataforma de nuvem (*Vercel*), que é própria para o armazenamento de credenciais e configurações sensíveis, garantindo que esses dados não sejam expostos no código fonte da aplicação e potencialmente publicados por engano (Wiggins, 2025). Dessa forma, a segurança do sistema é fortalecida, protegendo tanto os dados dos usuários quanto a própria infraestrutura.

## APIs

As *API*s *REST* (*Representational State Transfer*) são amplamente utilizadas na comunicação entre sistemas distribuídos, como aplicações web. Elas fazem uso do protocolo *HTTP* e seus métodos (ou verbos) para executar operações sobre recursos, identificados por *URLs* únicas. Esses recursos mapeiam as entidades manipuladas pela *API*, como usuários, produtos, compras, entre outros. O corpo da requisição ou resposta geralmente contém dados estruturados em *JSON*, permitindo a troca eficiente de informações detalhadas sobre o recurso em questão (Massé, 2011).

Nesse contexto, para garantir a integridade e a consistência dos dados nas requisições e respostas *REST*, foi utilizado o *Zod*, uma biblioteca de validação e interpretação de dados. Esta é responsável por definir os esquemas de todos os *DTO*s (*Data Transfer Objects*), permitindo validar e processar os dados antes de serem enviados e ao serem recebidos. Isso evita erros comuns relacionados a dados inconsistentes ou malformados e melhora a robustez da aplicação.

Além disso, a esquematização, tipagem e validação dos objetos trocados gera mensagens de erro claras e direcionadas, facilitando o diagnóstico e a resolução de problemas, tanto no lado do servidor quanto no lado do cliente. Ao usar essa biblioteca, a aplicação consegue capturar e tratar erros precocemente, e de forma mais eficiente, proporcionando uma experiência de desenvolvedor mais ágil e um fluxo de trabalho mais seguro.

## IoT

A aplicação realiza a liberação do produto comprado por meio do acionamento de um dispositivo *IoT*, permitindo a automação do processo de venda. Para o *MVP*, foi utilizada a tomada inteligente *TP-Link Tapo P110*, escolhida por seu baixo custo e compatibilidade com diversas plataformas de gerenciamento de dispositivos *IoT*. Essa escolha possibilita futuras expansões para outros modelos e dispositivos, mantendo a flexibilidade da solução.

A integração com esse dispositivo foi realizada por meio da *API* do *Google Home Graph*, que permite monitorar e controlar dispositivos conectados a uma conta Google. Através dessa *API*, a aplicação pode enviar comandos para ativar ou desativar a tomada conforme necessário, garantindo que o equipamento só seja acionado após a confirmação de uma compra válida.

# Validação da solução

Para garantir a qualidade e confiabilidade da solução, foi adotada uma série de boas práticas e tecnologias que proporcionaram um desenvolvimento robusto e consistente:

## Uso do *TypeScript*

O *TypeScript* foi utilizado em substituição ao *JavaScript*, garantindo maior confiabilidade no código, com tipagem estática que facilita a identificação de erros durante o desenvolvimento. Além disso, o *ESLint* foi configurado como *linter* para garantir a consistência do código, fazendo constantemente verificações de análise estática e dinâmica (Krause, 2024).

## Uso do *Next.js*

A escolha estratégica do *Next.js*, por utilizar o *Server-Side Rendering*, reduz o volume de dados que devem ser enviados ao cliente, o que minimiza o tamanho da *API*. Além de poupar linhas de código dedicadas aos objetos *DTO* e aos controladores, isso simplifica o gerenciamento de estados de *query* no cliente. Tudo isso colabora para reduzir potenciais pontos de falha.

## Testes Automatizados

Para a validação de funcionalidades ponta-a-ponta, foram implementados testes automatizados utilizando o *Playwright*. Essa ferramenta simula as interações do usuário através da emulação dos principais navegadores do mercado, e a cada etapa verifica o atendimento das expectativas e pós-condições para validar o comportamento esperado do sistema (Pathak, 2025). Além de poupar tempo do desenvolvedor por meio da automação, ajuda a trazer maior robustez contra problemas de regressão.

## Ambiente de Desenvolvimento

Foi criado um projeto *Docker Compose* que levanta um ambiente local, incluindo um banco de dados configurado com dados pré-*seeded*, permitindo realizar os testes das funcionalidades em desenvolvimento com dados reprodutíveis e em um ambiente isolado. Essa configuração replica as condições de produção e subsidia o funcionamento do *Playwright* supracitado.

## Ambiente de Testes e Pipeline de CI/CD

A *pipeline* de *CI/CD* foi configurada usando *GitHub Actions*, proporcionando uma automação eficiente no processo de *build* e teste, e que executa a cada *push* ou *pull request*. A aprovação da *pipeline* é uma condição obrigatória para o *merge* de código na *branch* principal, garantindo que apenas código validado seja integrado (Laster, 2023).

O primeiro passo na *pipeline* é gerar um container para o *build* da aplicação *Next.js*. Em seguida, um segundo container hospeda o website enquanto o *Playwright* executa os testes automatizados. Esse processo utiliza o mesmo *Docker Compose* do ambiente de desenvolvimento, garantindo que o ambiente de testes esteja sempre alinhado com ele (Wiggins, 2025). Além disso, a natureza efêmera desses contêineres também assegura que os testes sejam independentes entre si e consistentes.

## Ambiente de Produção

Para o ambiente de produção, foi configurado um banco de dados na plataforma *MongoDB Atlas* e a hospedagem da aplicação *Next.js* na nuvem *Vercel*. A infraestrutura foi configurada para realizar *deploy* automático da *branch master* do repositório *GitHub* após cada *merge*, garantindo uma entrega contínua e sem interrupções apenas do código validado.

# Registros das evidências do projeto

Este trabalho de conclusão acompanha um portifólio de evidências do projeto desenvolvido, acessível pelos *links* abaixo relacionados:

**Quadro 9.1.** Listagem do material que acompanha este documento.

|  |  |
| --- | --- |
| **Artefatos** | **Link** |
| Website hospedado | <https://tcc-fullstack-pucrs-luizfeldmann.vercel.app> |
| Repositório do código-fonte | <https://github.com/luizfeldmann/tcc-fullstack-pucrs> |
| Design de UI em *Figma* | <https://tinyurl.com/luizfeldmann-figma-fullstack> |

# Considerações finais e expectativas

Foi desenvolvido um *MVP* funcional de um aplicativo web em *NextJS*, utilizando o *design system* Material UI conforme protótipo elaborado no *Figma*. A aplicação implementa funcionalidades essenciais, como cadastro de usuários, adição de saldo, consulta de extratos e a listagem de pontos de venda dentro da empresa onde o projeto foi concebido. Além disso, o *MVP* integra a solução ao hardware *IoT*, permitindo que a compra de um produto listado acione a liberação de uma cafeteira, conforme idealizado. Todo o processo de desenvolvimento seguiu práticas de *DevOps*, e o sistema foi hospedado em nuvem para acesso público e testagem prática.

O trabalho atendeu com êxito aos objetivos definidos no capítulo (2), validando o modelo de negócios proposto e servindo como prova de conceito para uma solução escalável. O projeto demonstrou viabilidade técnica e aplicabilidade prática no ambiente corporativo. Futuramente, há grande potencial para expansão e refinamento, caso sejam implementados os itens listados como próximos passos.

## Trabalhos Futuros

Devido às limitações de tempo e recursos, o escopo do MVP foi restrito. No entanto, o projeto possui margens claras para evolução e completude por meio dos seguintes desenvolvimentos futuros:

1. **Adição de mais meios de pagamento:**

Ampliar a integração com operadores de pagamento digital para suportar transações via PIX e cartões de crédito/débito, proporcionando maior flexibilidade e conveniência aos usuários.

1. **Implementação da interface de gestão:**

Criar uma interface dedicada aos gestores dos equipamentos, possibilitando visualização de relatórios detalhados, monitoramento das receitas arrecadadas e ferramentas para projeção de estoque e gerenciamento de insumos. Essa funcionalidade é essencial para atender às necessidades dos cedentes de equipamentos e fortalecer o modelo de negócios.

Com a implementação desses itens, o projeto poderá evoluir para um produto completo, preparado para atender a empresas de diferentes tamanhos e com potencial para escalar em ambientes corporativos diversos.

# Referências

AFONSO, Daniel. **State Management with React Query**: Improve developer and user experience by mastering server state in React. 1. ed. Birmingham: Packt Publishing, 2023. ISBN 978-1-80324-483-9.

HOLMES, Simon. **Mongoose for Application Development**. 1. ed. Birmingham: Packt Publishing, 2013. ISBN 978-1-78216-819-5.

IETF - Internet Engineering Task Force. **RFC 7519**: JSON Web Token (JWT). 2015. Disponível em: <<https://tools.ietf.org/html/rfc7519>>. Acesso em: 2 fev. 2025.

KRAUSE, Martin. **The Complete Developer**: Master the Full Stack with TypeScript, React, Next.js, MongoDB, and Docker. 1. ed. San Francisco: No Starch Press, 2024. ISBN 978-1-7185-0328-1.

LASTER, Brent. **Learning GitHub Actions**: Automation and Integration of CI/CD with GitHub. 1. ed. Sebastopol: O'Reilly Media, 2023. ISBN 978-1-098-13107-4.

MARTIN, Robert C. **Clean Architecture:** A Craftsman's Guide to Software Structure and Design. 1. ed. Boston: Prentice Hall, 2017. ISBN 978-0-13-449416-6. DOI: 10.5555/3175742.

MASSÉ, Mark. **REST API design rulebook:** designing consistent RESTful web service interfaces. 1. ed. Sebastopol: O'Reilly Media, 2011. ISBN 978-1-4493-1050-9.

PATHAK, Kailash. **Playwright vs Selenium vs Cypress:** A Detailed Comparison. LambdaTest, 2025. Disponível em: <<https://www.lambdatest.com/blog/playwright-vs-selenium-vs-cypress>>. Acesso em: 2 fev. 2025.

SMITH, Andrew P. **Caffeine at work**. Human Psychopharmacology: Clinical and Experimental. [S. l.]: Wiley, ago. 2005. DOI 10.1002/hup.705. Disponível em: <<http://dx.doi.org/10.1002/hup.705>>.

SULLIVAN, Bryan; LIU, Vincent. **Web Application Security:** A Beginner's Guide. 1. ed. New York: McGraw-Hill Education, 2011. ISBN 0071776168.

TAZETDINOV, Andrei. **Next.js Cookbook**: Learn how to build scalable and high-performance apps from scratch. 1. ed. Mumbai: BPB Publications, 2023. ISBN 978-93-55518-45-3.

WIGGINS, Adam. **The twelve-factor app**. [S.l.: s.n.], 2011. Disponível em: <<https://12factor.net/>>. Acesso em: 2 fev. 2025.