FACULDADE DE TECNOLOGIA DE PRAIA GRANDE (FATEC)

DESENVOLVIMENTO DE SOFTWARE MULTIPLATAFORMA

Artur Souza Revollo

Bruno dos Santos de Araujo

Gabriel Dutra Amarante Carvalho

Luccas Lohan Parrilla

GELADINHO SANTISTA

Praia Grande

2025

Sumário

LISTA DE ILUSTRAÇÕES

DIAGRAMAS

FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
API RESTful	Representational State Transfer API
App	Aplicativo
CD	Continuous Delivery / Continuous Deployment
CI	Continuous Integration
HTTP	HyperText Transfer Protocol
ID	Identificador
JWT	Json Web Token
ML	Machine Learning
NoSQL	Not Only SQL
OAuth	Open Authorization
REST	Representational State Transfer API
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais

SEO	Search Engine Optimization
SSR	Server-Side Rendering
UML	Unified Modeling Language
VS Code	Visual Studio Code

1 INTRODUÇÃO

Este capítulo apresenta a introdução do projeto Geladinho Santista.

1.1 Objetivo do documento

Este documento descreve o projeto a ser desenvolvido, abordando os seguintes pontos:

- **Participantes do Grupo:** Identificação dos membros envolvidos no desenvolvimento do projeto.
- **Empresa Representada:** Apresentação da empresa que será atendida pelo projeto.
- **Descrição da Problemática:** Explicação dos desafios enfrentados pela empresa que o projeto visa resolver.
- **Solução Proposta:** Detalhamento da solução planejada, que envolve o desenvolvimento de um aplicativo mobile com back-end no formato API RESTful (Representational State Transfer API), integrado a um algoritmo de machine learning.

1.2 Visão Geral do Sistema

Criação de um aplicativo mobile que recomenda sabores personalizados de geladinhos e doces para os clientes da loja Geladinho Santista, utilizando algoritmos de aprendizado de máquina para classificar preferências e padrões de compra.

1.3 Público-Alvo

O público-alvo deste projeto são **clientes da Geladinho Santista**, que buscam uma experiência mais prática e personalizada ao realizar seus pedidos de geladinhos gourmet e doces.

Este público inclui:

- **Consumidores que desejam praticidade e agilidade**:

    - Clientes que preferem realizar pedidos de forma rápida e eficiente, sem precisar de interações manuais, como o envio de mensagens via WhatsApp.
    - O uso do cardápio digital acessível via QR Code visa facilitar o processo de escolha e pedido, tornando-o mais acessível e conveniente.
- **Clientes com preferências específicas**:
    - Pessoas que desejam um serviço mais personalizado, com sugestões de produtos baseadas em seu histórico de compras e preferências alimentares.
    - O algoritmo de aprendizado de máquina ajudará a identificar e classificar perfis de clientes, oferecendo recomendações personalizadas, como sabores de geladinhos ou combinações especiais.
- **Clientes que buscam uma experiência moderna e digitalizada**:
    - Consumidores que já estão familiarizados com tecnologias digitais e desejam utilizar aplicativos para facilitar o processo de compra e personalização.
    - O sistema integrado ao WhatsApp e o aplicativo mobile com funcionalidades de recomendação e personalização serão atraentes para esse público.
- **Proprietários e gestores da Geladinho Santista**:
    - Além dos consumidores finais, o público-alvo inclui os gestores e donos da loja, que serão beneficiados com o painel administrativo que permitirá a gestão de pedidos, controle de estoque e análise detalhada de vendas e padrões de consumo.
    - A solução permitirá que os proprietários compreendam melhor a demanda dos produtos, otimizem a produção e melhorem a gestão do negócio.
- **Público que valoriza a experiência de compra personalizada**:
    - O projeto também visa atrair clientes que buscam uma experiência de compra mais personalizada e com maior interação digital, utilizando

dados para oferecer um atendimento que se adapta aos gostos e hábitos de consumo de cada cliente.

1.4 Escopo do Projeto

1.4.1 Objetivo Geral

Desenvolver um sistema digital integrado para a gestão de pedidos, otimização do atendimento e análise das preferências dos clientes na empresa Geladinho Santista. O sistema incluirá um cardápio digital acessível via QR Code, um painel administrativo para controle de pedidos e estoque, além de recursos de aprendizado de máquina para personalizar as recomendações de produtos.

O sistema incluirá um cardápio digital acessível via QR Code, um painel administrativo para controle de pedidos e estoque, além de recursos de aprendizado de máquina para personalizar as recomendações de produtos.

1.4.2 Objetivos Específicos

- **Desenvolver um cardápio digital** acessível via QR Code para facilitar a realização de pedidos pelos clientes.
- **Criar um painel administrativo** que permita aos administradores da loja gerenciar pedidos, controlar estoque e acessar dados gerenciais sobre vendas e demanda.
- **Implementar relatórios gerenciais** que ofereçam análises detalhadas sobre o comportamento de vendas, estoque e sazonalidade dos produtos.
- **Incluir funcionalidades de aprendizado de máquina** para gerar recomendações personalizadas de sabores e produtos com base nas preferências e padrões de compra dos clientes.
- **Integrar o sistema ao WhatsApp**, facilitando a comunicação entre o cliente e a loja, além de permitir a automação dos pedidos e a coleta de dados sobre preferências.

Delimitação do Problema

O sistema visa resolver as dificuldades enfrentadas pela Geladinho Santista ao gerenciar pedidos manualmente via WhatsApp, que inclui:

- **Pedidos conflitantes** e problemas com a falta de estoque devido à falta de atualização em tempo real.
- **Falta de automação** no processo de gestão de pedidos, que ainda depende de trabalho manual.
- **Dificuldade de rastrear preferências** dos clientes de forma estruturada, o que impede uma personalização eficiente do atendimento.
- **Ausência de um cardápio digital fixo**, o que dificulta a atualização e o acesso às informações pelos clientes.

Soluções Propostas

- **Aplicativo Mobile** para que os clientes possam fazer pedidos diretamente, com recomendações de produtos baseadas em seus históricos de compras.
- **Algoritmos de aprendizado de máquina** para analisar os padrões de compra e sugerir sabores personalizados, ajudando a loja a otimizar o atendimento e a produção.
- **Integração com WhatsApp** para facilitar a comunicação e envio dos pedidos.
- **Dashboard para administração** com visualização de métricas de vendas, estoque e sugestões baseadas em dados, melhorando a tomada de decisões da loja.

Tecnologias Utilizadas

- **Front-end**: NextJS para o desenvolvimento web. React Native para o desenvolvimento do aplicativo mobile.
- **Back-end**: Node.js para a criação de uma API RESTful.
- **Banco de Dados**: Firebase para armazenamento e gerenciamento de dados.
- **Aprendizado de Máquina**: Algoritmo treinado para análise de dados de vendas e preferências dos clientes.
- **Hospedagem na Nuvem**: Vercel e Render para garantir escalabilidade e disponibilidade do sistema.

1.4.3 Justificativa

A solução proposta visa melhorar a eficiência operacional, reduzir erros e otimizar a experiência do cliente, proporcionando:

- **Automação** dos processos de pedidos e gestão de estoque.
- **Personalização do atendimento**, com recomendações inteligentes baseadas nos dados dos clientes.
- **Melhoria na experiência do cliente**, permitindo pedidos mais rápidos e precisos.

1.4.4 Limitações do Projeto

- O escopo não inclui integração com outros canais de vendas ou sistemas de gestão de terceiros além do WhatsApp e da plataforma de e-commerce existente.
- O foco inicial será nas funcionalidades de pedidos, gestão de estoque e recomendações personalizadas, com funcionalidades adicionais sendo consideradas em fases futuras.

2 REQUISITOS DO SISTEMA

Este capítulo descreve de forma geral os requisitos sistema, sendo: Requisitos Funcionais, Requisitos Não Funcionais, Regras de Negócio e Casos de Uso.

2.1 Requisitos Funcionais

Os requisitos funcionais (RF) descrevem as funcionalidades específicas que o sistema deve oferecer para atender às necessidades do projeto. Os principais requisitos funcionais incluem:

[RF 01] Visualizar Cardápio

- O sistema deve oferecer um cardápio digital acessível via **QR Code**, permitindo que os clientes visualizem os produtos disponíveis e façam pedidos de maneira simples e rápida.
- O cardápio digital deve ser atualizado em tempo real, garantindo que os produtos em estoque estejam disponíveis e visíveis para os clientes.

[RF 02 Visualizar Produto

- O sistema deve permitir que os clientes visualizem detalhes completos dos produtos, incluindo imagens, descrições, preços e opções de personalização (quando aplicável), de forma clara e intuitiva.
- Além disso, deve ser possível visualizar a disponibilidade em estoque e as informações relacionadas à preparação e ingredientes dos produtos, garantindo uma experiência completa para o cliente.

[RF 03] Adicionar produto ao carrinho

- O sistema deve permitir que os clientes adicionem produtos ao carrinho de compras com um único clique, de forma rápida e sem interrupções.
- O carrinho deve ser atualizado automaticamente, exibindo a quantidade de itens, o preço total e permitindo ajustes como a alteração da quantidade ou remoção de produtos. O cliente deve ser notificado visualmente sobre a

adição bem-sucedida ao carrinho para facilitar a navegação durante o processo de compra.

[RF 04] Enviar pedido

- O sistema deve permitir que o cliente envie seu pedido de forma clara e eficiente, após revisar o carrinho de compras. Ao finalizar a seleção dos produtos, o cliente deve ser apresentado a uma tela de confirmação, onde será possível revisar os itens, quantidade, preço total e detalhes do pedido.
- O processo de envio do pedido deve ser realizado com um único clique, garantindo que todos os dados necessários sejam transmitidos corretamente para o sistema de processamento. Após o envio, o cliente deve receber uma confirmação no WhatsApp com o número do pedido e o status estimado de entrega ou preparo.

[RF 05] Gerenciar Cardápio Digital

- O sistema deve permitir que os administradores gerenciem o cardápio digital de forma simples e intuitiva. Isso inclui a capacidade de adicionar, editar e remover produtos, bem como organizar os itens em categorias para facilitar a navegação dos clientes. Além disso, deve ser possível atualizar informações como preços, descrições, imagens e opções de personalização.

- Todas as alterações feitas no cardápio devem ser refletidas em tempo real para os clientes, garantindo que as informações apresentadas sejam sempre precisas e atualizadas.

[RF 06] Cadastrar Produtos

- O sistema deve permitir que os administradores cadastrem novos produtos de maneira simples e intuitiva. O processo de cadastro deve incluir campos para nome, descrição, preço, categoria, imagens, opções de personalização (se houver) e quantidade em estoque. O cadastro de produtos deve ser realizado de forma ágil e eficiente, permitindo a atualização rápida do cardápio digital.

[RF 07] Gerenciar Pedidos

- O sistema deve permitir aos administradores gerenciar e visualizar os pedidos em tempo real, proporcionando um painel administrativo para o acompanhamento detalhado do status de cada pedido.
- Isso inclui a capacidade de visualizar informações como itens do pedido, quantidade, tempo estimado de preparo e status de entrega.
- Além disso, deve ser possível filtrar e buscar pedidos de acordo com diversos critérios, como data, status ou cliente, facilitando o controle e a organização do fluxo de trabalho.

[RF 08] Gerenciar Estoque

- O sistema deve integrar o painel administrativo com o controle de estoque, permitindo que os administradores visualizem e gerenciem a disponibilidade dos produtos em tempo real. O sistema deve alertar os gestores sobre a falta de estoque ou a necessidade de reposição de produtos, com base nas vendas realizadas. Além disso, o sistema deve fornecer recomendações para otimizar o estoque com base nas preferências dos clientes e nas demandas históricas, ajudando a melhorar a eficiência e reduzir desperdícios.

[RF 09] Recomendações Personalizadas

- O sistema deve utilizar **algoritmos de aprendizado de máquina** para gerar recomendações personalizadas de sabores e combinações de produtos, com base no histórico de compras e preferências dos clientes.
- O algoritmo deve classificar os clientes em grupos (clusters) com preferências similares e sugerir novos produtos ou variações de sabores.

[RF 10] Gerar Relatórios Gerenciais

- O sistema deve permitir aos administradores gerenciar e visualizar os pedidos em tempo real, integrando com o **painel administrativo** para controle de estoque e status de pedidos.
- O sistema deve alertar os gestores sobre a disponibilidade dos produtos e ajudar a otimizar o estoque com base nas preferências e demandas.

2.2 Requisitos Não Funcionais

Os requisitos não funcionais (RNF) descrevem as características de qualidade que o sistema deve possuir. Baseado nos textos, os principais requisitos não funcionais incluem:

[RNF 01] Usabilidade:

- O sistema deve ser intuitivo e fácil de usar tanto para os clientes quanto para os administradores. O design do cardápio digital e do painel administrativo deve ser simples e acessível.

[RNF 02] Desempenho:

- O sistema deve ser capaz de processar pedidos e recomendações de forma rápida, com tempos de resposta mínimos, garantindo que os clientes tenham uma experiência ágil ao fazer pedidos.

[RNF 03] Escalabilidade:

- A solução deve ser escalável, ou seja, deve ser capaz de suportar o crescimento da demanda, seja com mais clientes, mais produtos ou com a expansão para mais unidades da empresa.

[RNF 04] Segurança:

- O sistema deve garantir a segurança dos dados dos clientes, especialmente informações sensíveis, como preferências alimentares, alergias e histórico de compras. A autenticação e a proteção dos dados devem ser tratadas de acordo com as melhores práticas de segurança.

[RNF 05] Acessibilidade:

- O sistema deve ser acessível em múltiplos dispositivos móveis, garantindo que os clientes possam realizar pedidos de qualquer lugar e a qualquer momento.

[RNF 06] Disponibilidade e Confiabilidade:

- O sistema deve ser altamente disponível, com um tempo de inatividade mínimo, assegurando que os pedidos possam ser feitos a qualquer momento. A hospedagem na nuvem (AWS ou Vercel) deve garantir alta disponibilidade e confiabilidade.

2.3 Regras de negócio

As regras de negócio descrevem os processos e as condições que o sistema deve seguir durante sua operação. As principais regras de negócio para o projeto incluem:

Gestão de Estoque:

- Quando um produto atingir o limite mínimo de estoque, o sistema deve gerar um alerta para os administradores, solicitando reposição.
- Os produtos devem ser automaticamente removidos do cardápio digital caso estejam fora de estoque ou não disponíveis.

Processamento de Pedidos:

- Todos os pedidos feitos pelos clientes devem ser registrados no sistema em tempo real, com informações como tipo de produto, quantidade e dados do cliente.
- O sistema deve gerar um número de identificação único para cada pedido e permitir que o administrador acompanhe seu status (em preparação, pronto para entrega, etc.).

Recomendações de Produtos:

- As recomendações de sabores devem ser feitas com base no histórico de pedidos do cliente e nas preferências coletadas. O sistema deve sugerir produtos de acordo com os padrões de consumo do cliente e a sazonalidade dos produtos.

Relatórios Gerenciais:

- O sistema deve gerar relatórios diários, sobre as vendas, estoque e padrões de compra, ajudando os administradores a tomarem decisões sobre quais produtos devem ser promovidos ou descontinuados.

Privacidade e Consentimento:

- O sistema deve garantir que as informações dos clientes sejam coletadas e armazenadas de acordo com as regulamentações de privacidade e proteção de dados. O cliente deve consentir com a coleta de seus dados antes de usá-los para personalização de produtos.

2.4 Casos de Uso

1. Acessar QR CODE
2. Visualizar Cardápio Digital
3. Escolher Produto(s)
4. Adicionar ao Carrinho
5. Enviar pedido
6. Recebe confirmação pelo WhatsApp
7. Aguarda entrega ou retirada

3 ARQUITETURA DO SOFTWARE

Este capítulo descreve a organização estrutural do sistema desenvolvido, incluindo os principais componentes, suas responsabilidades e as tecnologias utilizadas. A arquitetura foi planejada para garantir escalabilidade, manutenibilidade e desempenho, cobrindo desde o backend até a interface mobile e a integração com componentes de machine learning.

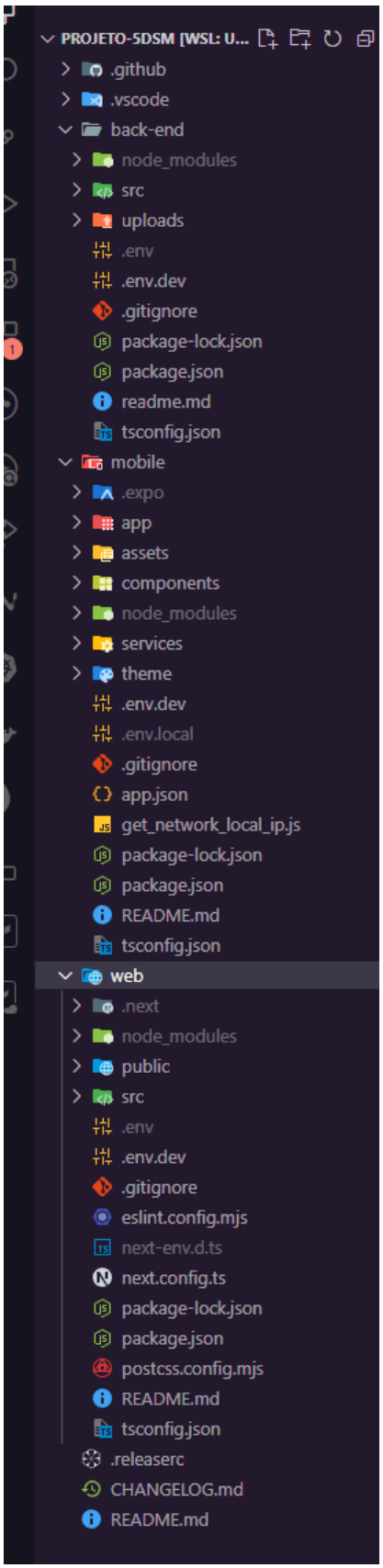
3.1 Visão Geral da Arquitetura

A arquitetura do sistema foi projetada com base em princípios de modularidade, organização em camadas e componentização, com o objetivo de garantir escalabilidade, manutenibilidade e clareza na separação de responsabilidades. A aplicação é composta por três grandes partes: o front-end web, o front-end mobile e o back-end, além de uma camada adicional voltada para o processamento de dados com uso de Machine Learning.

Todo o desenvolvimento foi realizado utilizando a linguagem TypeScript, escolhida por oferecer tipagem estática e uma melhor estruturação do código, reduzindo a incidência de erros e tornando a aplicação mais robusta. No caso do modelo de Machine Learning, a linguagem utilizada foi Python, amplamente reconhecida por sua expressividade e suporte a bibliotecas de ciência de dados.

A estrutura do back-end adota uma arquitetura em camadas, onde cada parte do sistema tem uma função bem definida. Quando uma rota da API (Application Programming Interface) é acionada, ela encaminha a requisição para um use case, responsável por executar a lógica de negócio. O use case, por sua vez, interage com o repositório, que é o ponto de acesso aos dados da aplicação. Este repositório comunica-se com o banco de dados por meio das APIs do Firebase, utilizando tanto o Firestore, para armazenamento de dados estruturados, quanto o Firebase Storage, para arquivos. Essa organização proporciona um código mais limpo, testável e fácil de manter. A Figura (1) a seguir demonstra visualmente essa estrutura no ambiente de desenvolvimento, representada por meio da organização de pastas e arquivos no VS Code (Visual Studio Code).

Figura 1 - Estrutura das pastas do Projeto em Geral



No desenvolvimento da aplicação web, foi utilizado o framework Next.js, baseado em React, que permite tanto a renderização do lado do servidor (Server-Side Rendering – SSR) quanto a geração de páginas estáticas, o que resulta em melhor desempenho e otimização para mecanismos de busca (SEO). A arquitetura do front-end web segue os princípios de componentização, onde cada funcionalidade é representada por um componente reutilizável, promovendo maior coesão e facilidade de manutenção.

A aplicação mobile foi desenvolvida com React Native, utilizando a plataforma Expo para facilitar o processo de construção, testes e deploy. A arquitetura também é baseada em componentes reutilizáveis, mantendo consistência com o front-end web. A escolha por React Native e Expo permite o desenvolvimento multiplataforma, utilizando um único código-base para Android e iOS, o que agiliza o processo e reduz custos de manutenção.

O back-end foi construído sobre a plataforma Node.js, utilizando o framework Fastify, escolhido por seu desempenho superior em comparação com alternativas mais tradicionais, como o Express. O uso do Firebase como serviço de autenticação e banco de dados proporciona escalabilidade, segurança e sincronização em tempo real, dispensando a necessidade de servidores próprios para operações básicas.

Por fim, a aplicação conta com uma camada de Machine Learning desenvolvida em Python, que realiza processamentos e análises específicas de dados. Essa camada é desacoplada do restante da arquitetura, garantindo independência de execução e facilitando futuras atualizações no modelo.

Essa arquitetura distribuída e bem segmentada foi pensada para garantir desempenho, segurança e flexibilidade, permitindo a evolução contínua do sistema com base em novas demandas e integrações futuras.

3.2 Arquitetura do Backend (API)

O back-end foi implementado utilizando Node.js com o framework Fastify, visando alto desempenho e baixa latência. Ele expõe uma API RESTful responsável por receber requisições dos aplicativos frontend (web e mobile), processar os dados, aplicar regras de negócio e retornar as respostas apropriadas.

3.3 Arquitetura do Mobile (Android/iOS)

A aplicação mobile foi desenvolvida utilizando React Native com o framework Expo, o que permitiu o desenvolvimento para plataformas Android e iOS com um único código-base. A interface foi construída com componentes reutilizáveis, utilizando Typescript para tipagem estática e maior segurança no código.

O aplicativo se comunica com a API do back-end por meio de requisições HTTP (HyperText Transfer Protocol), seguindo o padrão REST (Representational State Transfer), consumindo os dados fornecidos e exibindo as informações ao usuário de forma responsiva e eficiente.

3.4 Integração com o Algoritmo de Machine Learning

A integração com o algoritmo de Machine Learning foi feita via API. O modelo, desenvolvido em Python, foi hospedado em um servidor separado ou ambiente cloud (ex: Google Cloud Functions, AWS Lambda, etc.), e é acionado pelo backend com os dados necessários para realizar inferências ou classificações.

A resposta do modelo é então processada e retornada ao frontend, possibilitando decisões inteligentes ou sugestões baseadas em aprendizado de máquina.

3.5 Banco de Dados e Estrutura de Dados

O banco de dados utilizado foi o Firebase Firestore, que é um banco NoSQL em nuvem e altamente escalável. Ele armazena dados em forma de documentos e

coleções, o que facilita o gerenciamento de informações estruturadas e semiestruturadas.

A estrutura dos dados foi organizada para refletir as principais entidades do sistema, como clientes, pedidos, endereços, produtos, administradores, pedidos e categorias. Cada documento é identificado por um ID único, e as relações entre dados foram modeladas de forma a otimizar as consultas.

3.6 Tecnologias Utilizadas

Front-end Web:

- **Next.js** – Framework para React com renderização do lado do servidor (SSR) e geração estática.
- **React.js** – Biblioteca para construção da interface do usuário.
- **Typescript** – Superset de JavaScript que adiciona tipagem estática ao código.

Front-end Mobile:

- **Expo** – Plataforma que simplifica o desenvolvimento com React Native.
- **React Native** – Framework para desenvolvimento de aplicativos mobile nativos.
- **Typescript** – Tipagem estática para melhor organização e segurança do código.

Backend:

- **Node.js** – Ambiente de execução para JavaScript no lado do servidor.
- **Fastify** – Framework web leve e rápido para Node.js.
- **Firebase (Firestore + Storage)** – Serviços de autenticação e banco de dados em tempo real na nuvem.
- **Typescript** – Utilizado no backend para garantir consistência no desenvolvimento.

Machine Learning:

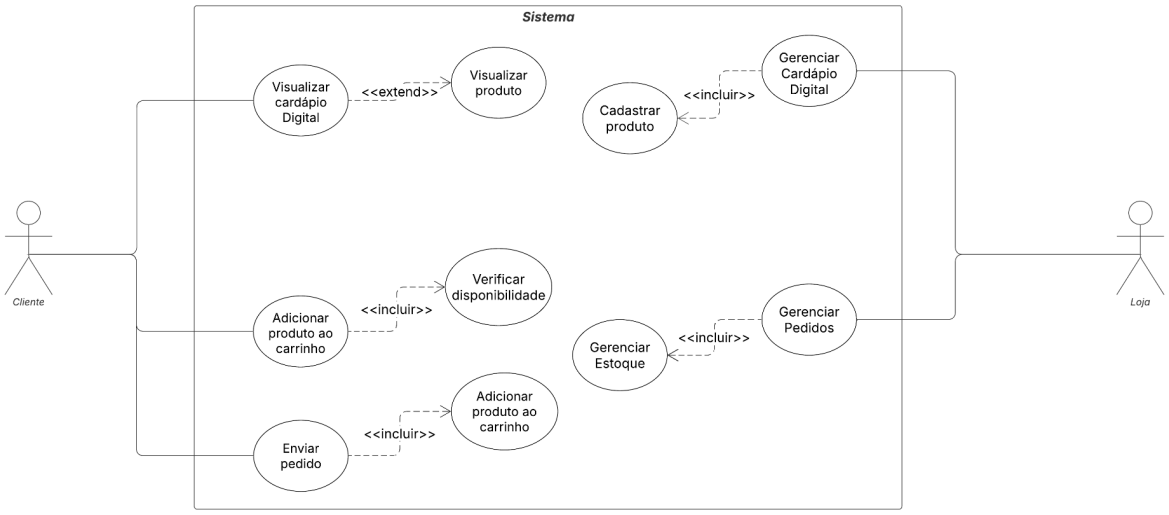
- **Python** – Linguagem utilizada para desenvolvimento do modelo de machine learning.

4 MODELAGEM DO SISTEMA

Este capítulo apresenta a modelagem do sistema, incluindo os diagramas UML (Unified Modeling Language) utilizados para representar graficamente os processos, entidades e interações do sistema.

4.1 Diagramas UML (Casos de Uso, Classe, Sequência, Atividade)

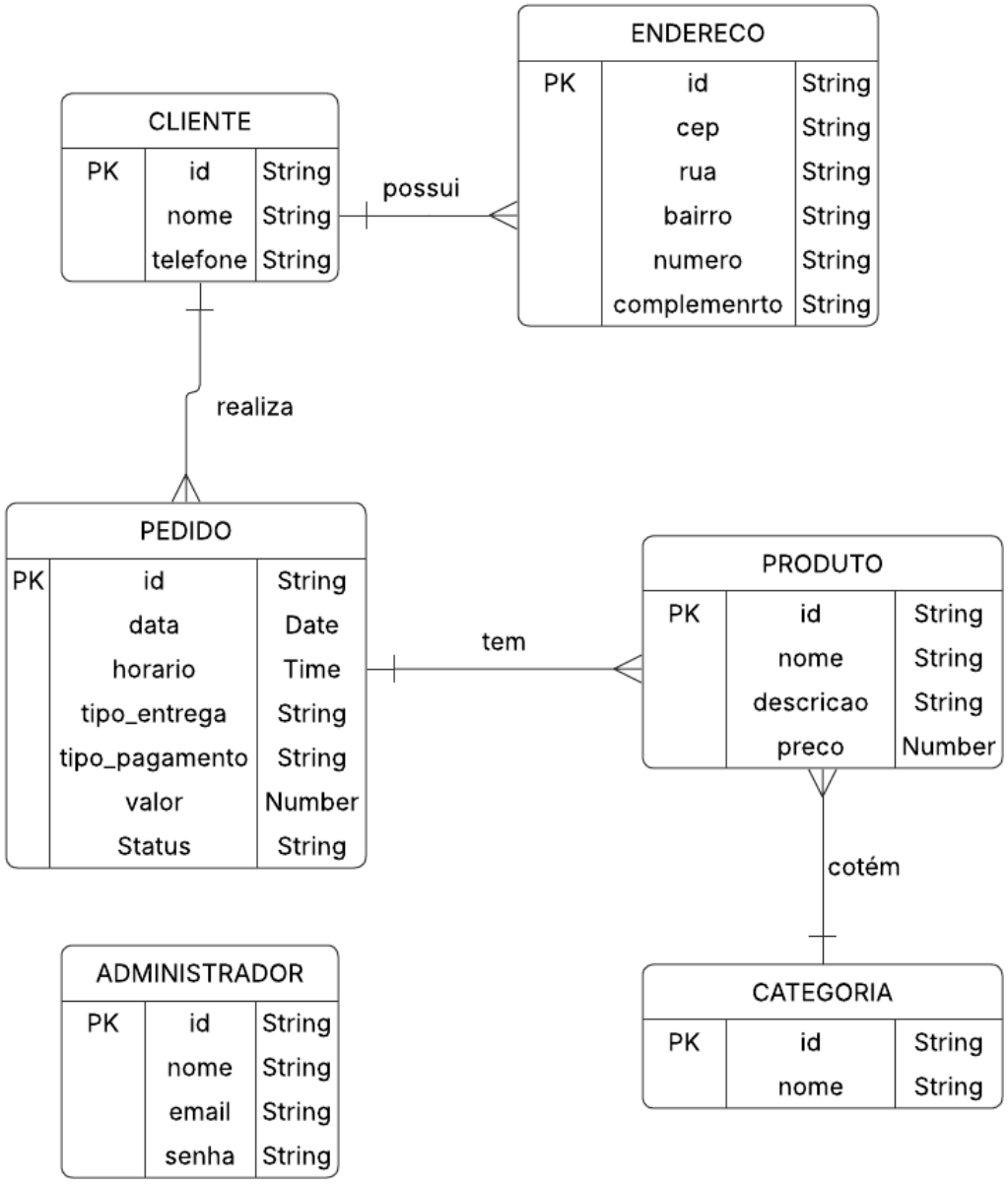
Diagrama 1 - Diagrama de caso uso principal da aplicação



4.2 Estrutura do Banco de Dados (DER e Diagrama Relacional)

O banco de dados utilizado neste projeto é não relacional, do tipo orientado a documentos, o que proporciona maior flexibilidade na estrutura dos dados, sendo ideal para aplicações que exigem rápida escalabilidade e evolução do modelo ao longo do tempo. Embora a persistência seja feita em formato de documentos, a representação visual pelo diagrama (2) abaixo, segue o modelo relacional, facilitando a compreensão das entidades, relacionamentos e campos envolvidos.

Diagrama 2 - Diagrama do Banco de Dados



Como observado no diagrama (2), o Cliente pode possuir um ou vários endereços (relação 1:N) e realizar um ou vários pedidos (relação 1:N). A seguir, detalha-se cada entidade com seus respectivos campos e a importância de cada um para o sistema:

CLIENTE

- id (String): Identificador único do cliente.
- nome (String): Nome completo do cliente.
- telefone (String): Telefone de contato do cliente.

Essas informações são essenciais para identificar e se comunicar com o cliente, além de associar seus pedidos e endereços.

ENDERECO

- id (String): Identificador único do endereço.
- cep (String): Código postal do endereço.
- rua (String): Nome da rua.
- bairro (String): Bairro onde se localiza o endereço.
- numero (String): Número da residência ou estabelecimento.
- complemento (String): Informações adicionais sobre o endereço (ex: bloco, apartamento).

Essa estrutura permite o armazenamento completo dos dados de entrega e localização dos clientes.

PEDIDO

- id (String): Identificador único do pedido.
- data (Date): Data em que o pedido foi realizado.
- horario (Time): Horário da realização do pedido.
- tipo_entrega (String): Tipo de entrega selecionado (delivery, retirada).
- tipo_pagamento (String): Forma de pagamento utilizada (cartão, dinheiro, Pix).
- valor (Number): Valor total do pedido.
- Status (String): Status atual que o pedido se encontra ()

Os dados do pedido são essenciais para o controle e histórico das transações realizadas.

PRODUTO

- id (String): Identificador único do produto.
- nome (String): Nome do produto.
- descricao (String): Descrição detalhada do produto.
- preco (Number): Valor do produto.

Cada produto é vinculado a uma ou mais categorias e pode ser associado a múltiplos pedidos. Esses campos são importantes tanto para exibição ao cliente quanto para controle interno de estoque e vendas.

CATEGORIA

- id (String): Identificador da categoria.
- nome (String): Nome da categoria (ex: bebidas, lanches, sobremesas).

As categorias permitem a organização dos produtos, facilitando a busca e a navegação para o cliente.

ADMINISTRADOR

- id (String): Identificador do administrador.
- nome (String): Nome do administrador.
- email (String): E-mail de acesso.
- senha (String): Senha criptografada para autenticação.

Essa entidade representa os usuários com permissões administrativas, sendo responsáveis por ações como o cadastro de produtos, categorias, visualização de pedidos e gerenciamento do sistema.

Apesar de o banco ser não relacional, o uso deste diagrama (2) auxilia na organização e clareza do modelo de dados, facilitando tanto o desenvolvimento quanto a manutenção do sistema. As relações entre entidades são representadas de maneira lógica e serão implementadas conforme as melhores práticas de bancos orientados a documentos, como os documentos incorporados.

4.3 Modelagem do Algoritmo de Machine Learning

1. Objetivo do Algoritmo

O sistema de recomendação tem como objetivo sugerir combos de geladinhos baseados em:

- Histórico de compras do cliente
- Similaridade entre produtos (sabores frequentemente comprados juntos)
- Contexto (clima, perfil do usuário, categoria dos produtos)

2. Técnica Utilizada: Filtragem Híbrida

Combina duas abordagens principais:

- **Abordagem de Filtragem Colaborativa**

  Como Funciona: Analisa padrões de compra de usuários similares.

  Exemplo: ("Clientes como você também compraram...")

- **Abordagem de Filtragem Baseada em Itens**

  Como funciona:Recomenda produtos similares aos que o usuário já comprou.

  Exemplo: ("Quem comprou X também gostou de Y")

3. Algoritmo Escolhido: LightFM

- **Biblioteca: lightfm (Python)**
- **Por quê?:**

- Suporte nativo a dados híbridos (colaborativos + features de itens/usuários)
- Escalável para grandes volumes de dados
- Bom desempenho em cenários com poucos dados (cold start)

4. Pré-processamento dos Dados

Entradas do Modelo:

- Interações: (cliente_id, sabor, quantidade_comprada)
- Features dos Itens: Categoria do sabor (ex: "Gourmetizados", "Zero Lactose")
- Features dos Usuários (opcional): Cidade, frequência de compras

5. Treinamento do Modelo

Passos no Google Colab:

1. Extrai dados do Firestore → Transforma em matrizes esparsas
2. Treina o modelo com 20 épocas (iterações)
3. Avalia com métricas de precisão (@k)
4. Exporta os pesos para arquivos .pkl

6. Geração de Recomendações

Para um cliente específico:

1. O modelo calcula scores para todos os sabores
2. Filtra itens já comprados
3. Ordena pelos maiores scores
4. Aplica regras de negócio (ex: evitar chocolates em dias quentes)

7. Métricas de Avaliação

- Precisão@5: % de acertos nas top 5 recomendações
- Coverage: % de itens do catálogo recomendados
- A/B Testing: Comparação de taxas de conversão entre versões do modelo

8. Diagrama do Fluxo de Dados

Diagrama 3 - Diagrama do Fluxo de Dados Machine Learning

5 DESENVOLVIMENTO DO BACKEND (API)

Este capítulo tem como objetivo descrever o Desenvolvimento do Backend (API).

5.1 Estrutura do Projeto

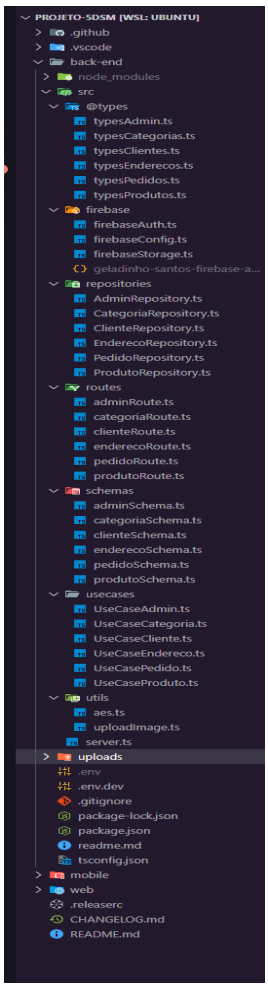
A arquitetura do projeto segue o padrão baseado em camadas, o que favorece a organização, a manutenibilidade e a escalabilidade da aplicação. O fluxo inicia-se com o acesso a uma rota, que aciona um use case — responsável por encapsular e executar a lógica de negócio da aplicação.

O use case interage com o repositório, cuja função é abstrair o acesso aos dados. Esse repositório realiza operações com o banco de dados por meio das APIs do Firebase, utilizando tanto o Firestore para armazenamento estruturado quanto o Firebase Storage para arquivos, conforme as necessidades de cada funcionalidade.

Essa separação de responsabilidades contribui para um código mais limpo, modular e testável, facilitando futuras expansões ou alterações.

A Figura (2), apresentada a seguir, demonstra essa organização por meio da estrutura de pastas e arquivos exibida no ambiente de desenvolvimento VS Code, evidenciando a divisão clara entre camadas como routes, useCases, repositories e services.

Figura 2 - Estrutura das pastas do Back-End



5.2 Endpoints e Métodos HTTP

A API do sistema Geladinho Santista segue o estilo arquitetural REST, utilizando os métodos HTTP padrão para realizar operações sobre os recursos. Abaixo estão listados os principais endpoints organizados por entidade, bem como os respectivos métodos HTTP disponíveis para cada um:

Administrador

- POST `/admins/` — Cria um novo administrador.
- POST /admins/login - Realiza o login do administrador.
- GET `/admins/` — Retorna a lista de administradores cadastrados.

- `GET /admins/{id}` — Retorna os dados de um administrador específico.

Cliente

- `POST /clientes/` — Cria um novo cliente.
- `GET /clientes/` — Retorna todos os clientes.
- `GET /clientes/{id}` — Retorna os dados de um cliente específico.

Produto

- `POST /produtos/` — Cria um novo produto.
- `GET /produtos/` — Retorna todos os produtos cadastrados.
- `GET /produtos/{id}` — Retorna os dados de um produto específico.

Endereço

- `POST /enderecos/` — Cadastra um novo endereço.
- `GET /enderecos/` — Lista todos os endereços disponíveis.
- `GET /enderecos/{id}` — Retorna um endereço específico por ID.

Categoria

- `POST /categoria/` — Cria uma nova categoria de produto.
- `GET /categoria/` — Lista todas as categorias cadastradas.
- `GET /categoria/{id}` — Retorna os detalhes de uma categoria específica.

Pedido

- `POST /pedido/` — Registra um novo pedido.
- `GET /pedido/` — Lista todos os pedidos realizados.
- `GET /pedido/{id}` — Retorna os detalhes de um pedido específico.
- PUT /pedido/{id} - Atualiza o status de um pedido pelo ID

5.3 Autenticação e Segurança (JWT, OAuth, etc.)

O sistema Geladinho Santista utiliza o Firebase Authentication como mecanismo de autenticação de usuários. A autenticação é realizada exclusivamente por meio de e-mail e senha, onde o usuário informa suas credenciais para acessar o sistema. Após o login, o próprio Firebase gerencia o estado de autenticação da sessão, sem a necessidade de tokens JWT (JSON Web Token) ou protocolos como OAuth (Open Authorization). O controle de acesso às funcionalidades é feito com base no status autenticado do usuário, garantindo que apenas usuários devidamente autenticados possam realizar ações protegidas, como cadastro de dados ou acesso a informações sensíveis. Dessa forma, a segurança do sistema é assegurada de forma simples e eficaz, utilizando os recursos nativos do Firebase.

5.4 Tratamento de Erros e Logs

O sistema utiliza tratamento de erros centralizado no back-end para garantir respostas apropriadas e consistentes ao front-end. Os erros são capturados com try/catch nos handlers do Fastify e, quando necessário, tratados com mensagens personalizadas.

Tratamentos HTTP Implementados

A API responde com os seguintes códigos HTTP padronizados:

- 200 OK – Requisição executada com sucesso.
- 201 Created – Recurso criado com sucesso.
- 400 Bad Request – Dados ausentes ou inválidos no corpo da requisição.
- 404 Not Found – Recurso não localizado (ex: pedido ou cliente inexistente).
- 500 Internal Server Error – Erro inesperado durante o processamento no servidor.

6 DESENVOLVIMENTO DO APP MOBILE

Este capítulo tem como objetivo documentar como as funcionalidades do sistema foram implementadas utilizando HTML, CSS, JavaScript e Node.js, detalhando a estrutura dos componentes, as interações entre eles e a aplicação de boas práticas de desenvolvimento, com foco na modularidade, reutilização de código e manutenibilidade.

6.1 Estrutura do Projeto Mobile

A estrutura do projeto mobile foi desenvolvida utilizando tecnologias modernas voltadas para desempenho, produtividade e manutenção simplificada. O aplicativo foi construído com React Native, um framework que permite o desenvolvimento de aplicativos nativos para Android e iOS a partir de um único código-base. Essa abordagem garante economia de tempo e esforço, mantendo a experiência do usuário próxima à de um app (Aplicativo) nativo.

Para facilitar o processo de desenvolvimento, testes e deploy, foi utilizada a plataforma Expo, que oferece um conjunto de ferramentas e serviços que simplificam a configuração inicial, o gerenciamento de dependências e a publicação do aplicativo. O uso do Expo também facilita a integração com recursos nativos dos dispositivos, sem a necessidade de configurações mais complexas típicas de projetos React Native puros.

A aplicação é estruturada com base na componentização, ou seja, cada funcionalidade ou tela é dividida em componentes reutilizáveis, promovendo melhor organização e manutenibilidade do código. Toda a lógica de exibição, navegação e consumo da API é separada por responsabilidades, seguindo boas práticas de desenvolvimento.

A linguagem utilizada é o TypeScript, um superset do JavaScript que adiciona tipagem estática ao código. Isso proporciona maior robustez durante o desenvolvimento, melhora a legibilidade, facilita a detecção precoce de erros e contribui para a escalabilidade da aplicação à medida que ela evolui.

Essa combinação de tecnologias — React Native, Expo e TypeScript — possibilita um ambiente de desenvolvimento moderno, produtivo e alinhado com as exigências atuais de qualidade e desempenho em aplicações mobile.

6.2 Fluxo de Navegação

6.2.1 Fluxo App Cardápio Digital (Cliente Final)

O Aplicativo Cardápio Digital foi desenvolvido para proporcionar uma experiência de navegação intuitiva e eficiente ao cliente final, permitindo que o mesmo possa visualizar os produtos disponíveis, realizar o pedido e acompanhar o status da sua solicitação.

A estrutura de navegação segue uma sequência linear, com opções de retorno entre as telas, visando garantir maior controle durante o processo de compra.

O fluxo é composto pelas seguintes etapas:

1. **Tela de Boas-Vindas**
   Ao abrir o aplicativo, o cliente é recepcionado com a tela inicial, contendo o botão de acesso ao cardápio e as informações da loja, como horário de funcionamento, endereço e contato.

2. **Tela de Cardápio**
   Nesta tela, o cliente visualiza a lista de produtos disponíveis, organizados por categorias, como:

- Geladinhos;
- Bolos inteiros;
- Bolos em pedaços;
- Bebidas.

É possível visualizar o nome, descrição e preço de cada item.
O cliente pode clicar em um produto para visualizar mais detalhes.

3. **Tela de Detalhes do Produto**
   Ao selecionar um produto, o cliente acessa os detalhes do item, podendo:

- Verificar informações adicionais;
- Alterar a quantidade desejada;
- Adicionar o produto ao carrinho.

4. **Tela de Carrinho de Compras**
   Após selecionar os produtos, o cliente visualiza o resumo do pedido com os seguintes dados:

- Lista de itens;
- Quantidade;
- Preço total;
- Métodos de pagamento disponíveis (exemplo: dinheiro, cartão, pix);
- Opções de entrega (delivery ou retirada).

O cliente pode editar os itens antes de prosseguir.

5. **Tela de Dados para Entrega**
   Nesta etapa, o cliente preenche o formulário com os seguintes dados:

- Nome completo;
- Telefone/WhatsApp;
- Endereço de entrega.

6. Tela de Pedido Confirmado
   Após o envio do pedido, o cliente visualiza a tela de confirmação, contendo:

- Status do pedido (exemplo: em produção);

- Tempo estimado de entrega;
- Detalhes do pedido;
- Informações da entrega;
- Total pago.

Existe também a opção de acompanhamento pelo WhatsApp da loja.

7. **Fluxo de Retorno**

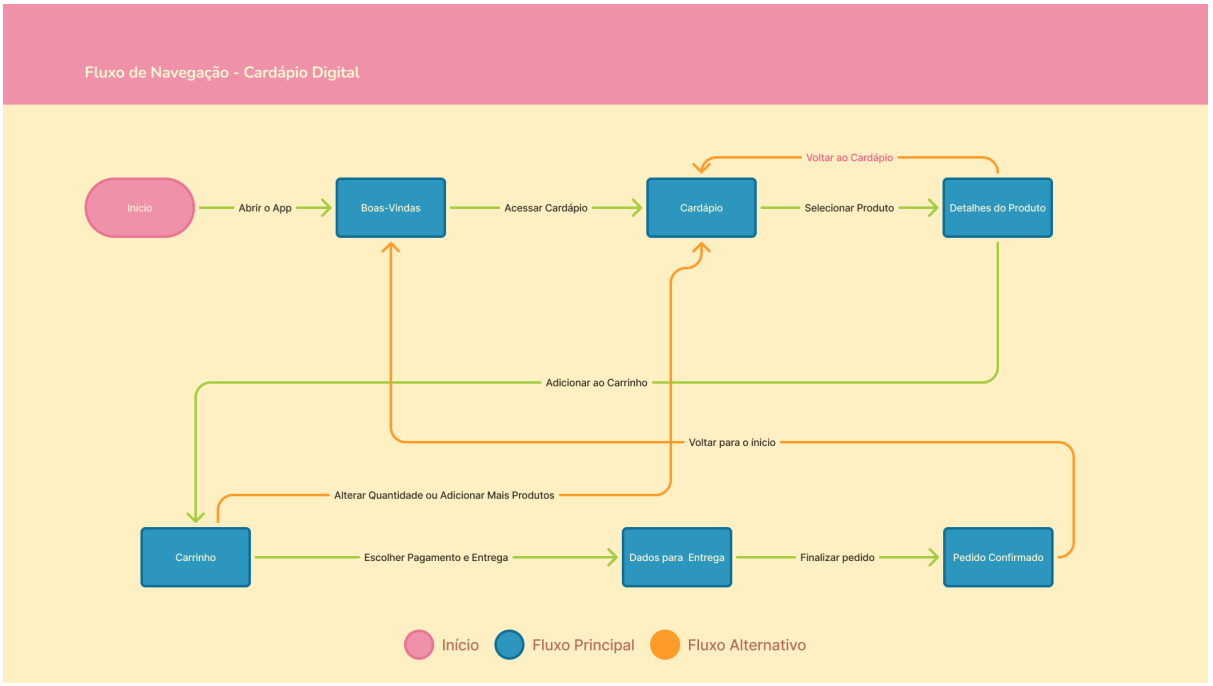
   Durante todo o processo, o cliente pode retornar a etapas anteriores, como voltar ao cardápio ou ao carrinho, caso deseje alterar o pedido.

6.2.2 Resumo da Estrutura de Navegação:

- Início → Tela de Boas-Vindas → Cardápio → Detalhes do Produto → Carrinho → Dados para Entrega → Pedido Confirmado

- **Fluxos Alternativos:**
  - Voltar para o cardápio;
  - Editar a quantidade de produtos;
  - Alterar itens do carrinho;
  - Retornar ao início após a confirmação do pedido.

Diagrama 4 - Fluxo de navegação do Cardápio Digital



6.2.3 Fluxo de Navegação – App de Gerenciamento de Pedido

O Aplicativo de Gerenciamento de Pedido (Painel Administrativo) foi projetado para atender às necessidades do administrador da loja, proporcionando uma navegação fluida e eficiente no acompanhamento e gerenciamento dos pedidos.

A estrutura de navegação é composta pelas seguintes etapas:

1. **Tela de Login**
   O acesso ao sistema é realizado por meio da autenticação do administrador com seu e-mail e senha. Após a validação, o usuário é direcionado para o Dashboard principal.
2. **Dashboard (Tela Inicial)**
   Nesta tela, o administrador visualiza um resumo geral da situação atual dos pedidos, segmentados por status: Pendentes, Em Produção, Prontos e Finalizados. Também é possível visualizar os pedidos recentes e o total de

vendas do dia.

A partir do Dashboard, o usuário pode navegar para as seguintes seções principais:

- Pedidos;
- Produtos;
- Histórico.

3. **Tela de Pedidos**

   Nesta seção, o administrador consegue filtrar os pedidos conforme o status desejado, utilizando as abas de navegação entre: Pendentes, Em Produção, Prontos e Finalizados.

   Cada pedido listado apresenta um resumo com as principais informações, como: código do pedido, nome do cliente, forma de pagamento, horário do pedido e valor total.

   Ao selecionar um pedido, o administrador é direcionado para os detalhes completos do pedido, onde poderá alterar o status conforme o andamento da produção.

4. **Tela de Produtos**

   A tela de produtos apresenta uma listagem de todos os itens do cardápio cadastrados.

   O administrador pode adicionar um novo produto clicando no botão de adição, que o leva até o formulário de cadastro de produto.

5. **Tela de Cadastro de Produto**

   Nesta tela, o administrador preenche as informações necessárias para criar um novo produto, incluindo:

- Nome do produto;
- Descrição;
- Categoria;
- Preço;
- Imagem do produto.

Após o cadastro, o novo item é adicionado à lista de produtos disponíveis.

6. **Tela de Histórico de Pedidos**

   O histórico apresenta todos os pedidos que foram finalizados.

Há a possibilidade de aplicar filtros por mês e ano, facilitando a consulta de pedidos passados.

Ao selecionar um pedido, o administrador poderá visualizar os detalhes completos da venda.

7. **Navegação Geral (Menu Inferior - TabBar)**

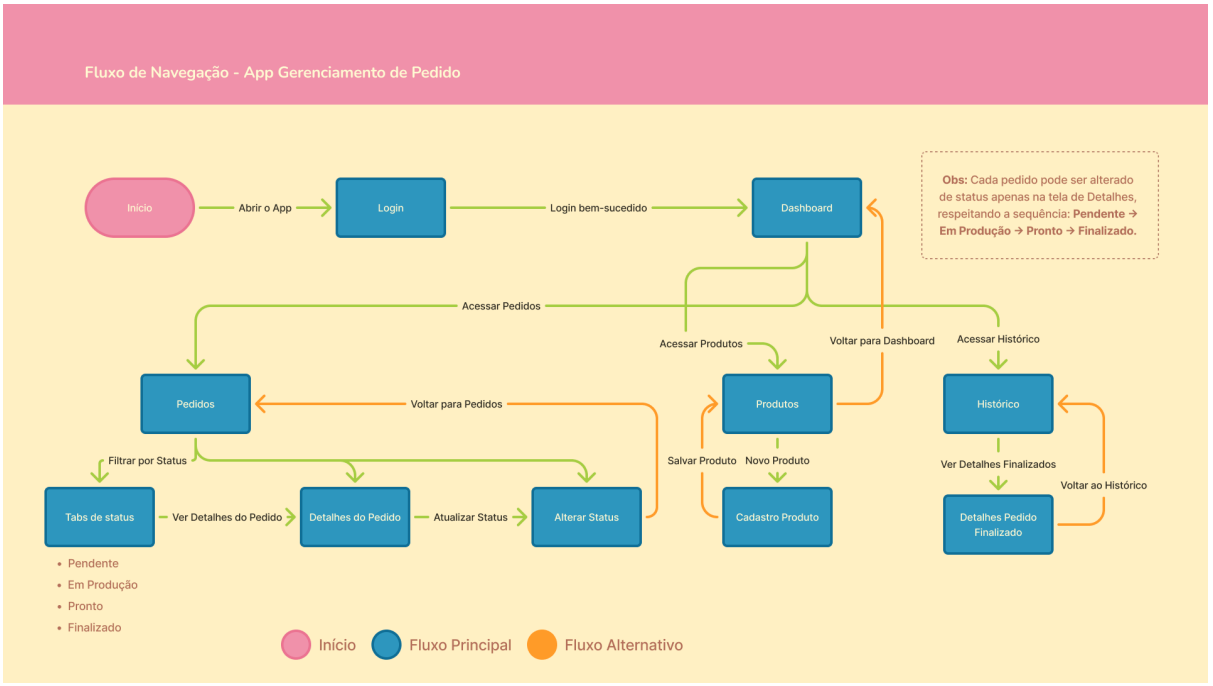
Após o login, todas as telas possuem um menu de navegação inferior (TabBar), que permite o acesso rápido às seções principais:

- Dashboard;
- Pedidos;
- Produtos;
- Histórico.

6.2.4 Resumo da Estrutura de Navegação:

- Início → Login → Dashboard
- Dashboard → Pedidos → Detalhes de Pedido → Alterar Status → Voltar para Pedidos
- Dashboard → Produtos → Cadastro de Produto → Voltar para Produtos ou Dashboard
- Dashboard → Histórico → Detalhes do Pedido Finalizado → Voltar para Histórico

Diagrama 5 - Fluxo de navegação do Aplicativo Mobile



6.3 Comunicação com a API

A comunicação entre o front-end e o back-end da aplicação será realizada por meio de requisições HTTP para uma API RESTful desenvolvida utilizando o framework Fastify em Node.js. Essa API será responsável por expor os recursos e operações do sistema, como cadastro de usuários, autenticação, registro de pedidos e gerenciamento de produtos.

6.4 Gerenciamento de Estados

Tabela 1 - Controle dos estados da aplicação

Estado	Dados do Estado	O que significa?
Lista de Pedidos	Quando o app busca ou atualiza a lista de pedidos	Quando o dono abre a aba "Pedidos", o app consulta o back-end e carrega todos os pedidos do dia

Status de um Pedido	Armazena e atualiza o status atual de cada pedido (Pendente, Em produção, Pronto, Concluído)	Quando o dono clica em "Finalizar Pedido", o app atualiza o estado desse pedido para "Finalizar"
Dados de Login	Controla se o usuário está autenticado	Se o usuário fez login com sucesso, o app salva o estado de usuário logado até ele sair
Tab selecionada (Pedidos)	Controla qual aba está aberta (Pendentes / Produção / Pronto / Finalizado)	Se o dono clicar em "Pronto", o app altera o estado da tab atual para a aba "Pronto"
Pedidos sendo carregados (Loading State)	Estado de carregamento enquanto os dados vêm da API	Exibir um spinner ou loading enquanto busca os pedidos do servidor
Lista de Produtos	Quando o webapp busca os pedidos no back-end	Quando o cliente abre a página de cardápio, o webapp consulta o back-end e carrega todos os produtos

6.5 Interface do Usuário e Design System

6.5.1 Interface do Usuário e Design System (App Cardápio Digital)

O desenvolvimento da interface do usuário do App Cardápio Digital – Geladinho Santista seguiu princípios de design centrado no usuário, priorizando uma experiência simples, acessível e alinhada com a identidade da marca.

6.5.2 Diretrizes de Interface:

- **Foco na usabilidade:**
  Todas as telas foram desenhadas para facilitar o processo de pedido, com navegação linear, poucos cliques e informações objetivas.
- **Layout Mobile First:**
  O design foi pensado exclusivamente para dispositivos móveis, com

resolução base de 375x812px, visando atender os clientes que realizam pedidos via smartphone.

- **Hierarquia Visual:**
  Uso de títulos, subtítulos, textos de apoio e botões com pesos visuais bem definidos para guiar a jornada do usuário.
- **Acessibilidade:**
  As cores foram escolhidas com base em contrastes adequados para leitura confortável, e os botões possuem tamanho e espaçamento suficientes para toque em tela.

6.5.3 Design System Utilizado:

- **Paleta de Cores:**
  As cores foram selecionadas para refletir a identidade visual da marca Geladinho Santista e ao mesmo tempo proporcionar clareza na interação.

Tabela 2 - Paleta de cores da Aplicação Web

Cor	**Código HEX**	**Aplicação**
Rosa Principal	#F493AE	Botões, títulos, cabeçalhos e áreas de destaque
Amarelo BG	#FFFBD6	Fundo das telas
Verde Atlantis	#A8CF45	Status de confirmação e botões de sucesso
Laranja Sunshade	#FF9B2B	Status intermediário (Pedido em Produção)
Laranja Card	#FACB96	Fundo dos cards
Chocolate	#A45E4D	Textos de destaque
Azul Button	#3198C2	Botões de edição

Tipografia:

Fonte utilizada: Nunito, com os seguintes pesos e tamanhos:

Tabela 3 - Tipografia da Aplicação Web

Tipo de texto	**Peso**	**Tamanho (px)**
Títulos	Bold	18px a 24px
Subtítulos	Medium	14px a 18px
Corpo de texto	Regular	12px a 14px
Botões	SemiBold	14px a 16px

Componentes de Interface (UI Elements):

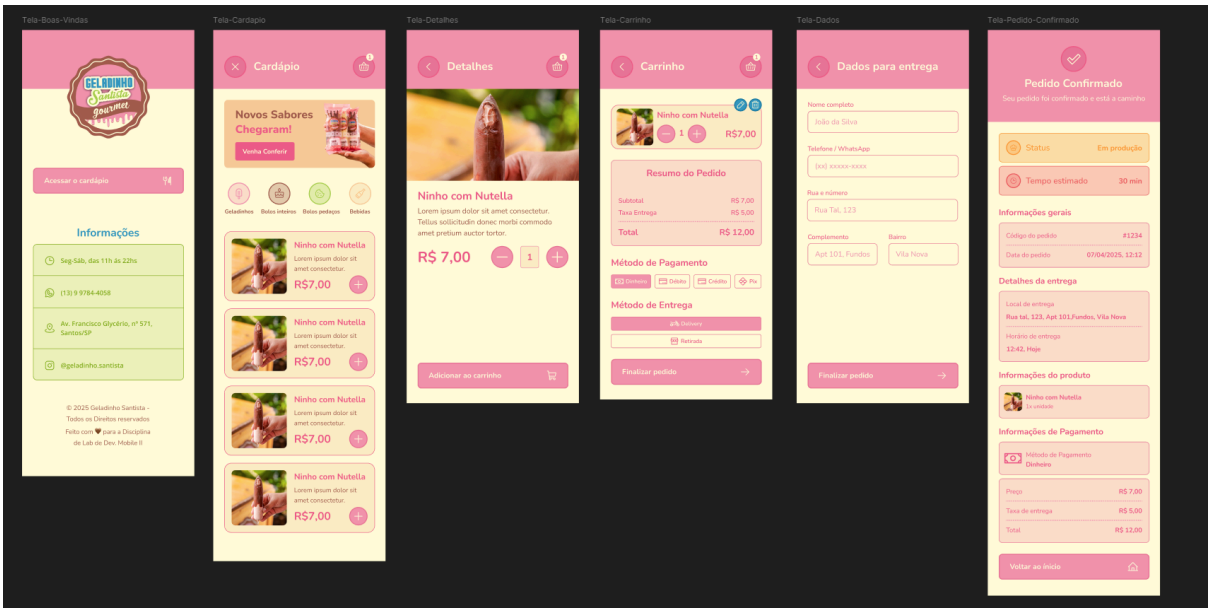
- Botões de ação (Adicionar ao Carrinho, Finalizar Pedido)
- Cards de produtos
- Barra de etapas no pedido
- Formulários com campos de entrada (dados de entrega)
- Mensagens de status (Pedido Confirmado)
- Ícones para status, categorias e navegação
- **Iconografia:**

  Utilização da biblioteca Phosphor Icons com ícones que reforçam a leitura visual:

  Exemplos: Carrinho, Check, Plus, Minus, entre outros.
- **Estilo de Imagem dos Produtos:**

  As imagens dos produtos possuem fundo limpo e boa resolução, mantendo o foco no item ofertado.

Figura 3 - Layout do cardápio digital



6.5.2 Interface do Usuário e Design System – App de Gerenciamento de Pedido

Tabela 4 - Paleta de Cores Utilizada no Mobile

Cor	Código HEX	Aplicação
Rosa Principal	#F493AE	Barra inferior (TabBar), botões principais e destaques visuais
Amarelo BG (Fundo)	#FFFBD6	Fundo geral das telas
Verde Atlantis	#DDEEA8	Cards de pedidos finalizados, feedbacks de sucesso
Laranja Sunshade	#FFDEA8	Status intermediário: “Em produção”
Azul Boston Blue	#B5DEEC	Status: “Pronto”
Amarelo YellowBG	#FFF388	Status: “Pendente”
Marrom Chocolate	#A45E4D	Textos importantes e detalhes de pedido

Tabela 5 - Tipografia Aplicada no mobile

Tipo de texto	Peso	Tamanho (px)
Títulos (ex.: Dashboard, Pedidos)	Bold	18px a 24px
Subtítulos (ex.: "Resumo Geral", "Pedidos Recentes")	Medium	14px a 18px
Corpo de texto (detalhes dos pedidos, endereços, nomes de clientes)	Regular	12px a 14px
Botões (ex.: "Iniciar Produção", "Finalizar Pedido")	SemiBold	14px a 16px

6.5.3 Componentes do Design System

Tabela 6 - Ícones Utilizados (Phosphor Icons) no mobile

Componente	Descrição / Uso
Cards de Pedido	Exibição resumida de cada pedido, incluindo status, cliente, forma de pagamento e valor
TabBar Inferior	Navegação entre as seções principais: Início, Pedidos, Histórico, Produtos
Botões de Ação	Ações como: Iniciar Produção, Marcar como Pronto, Finalizar Pedido, Adicionar Produto
Pedidos (TabBar)	Abas superiores na tela Pedidos: Pendente, Em Produção, Pronto, Concluído
Filtros de Status	Uso de cores nos cards para identificação visual rápida (Pendente – Amarelo, Em Produção – Laranja, Pronto – Azul, Finalizado – Verde)

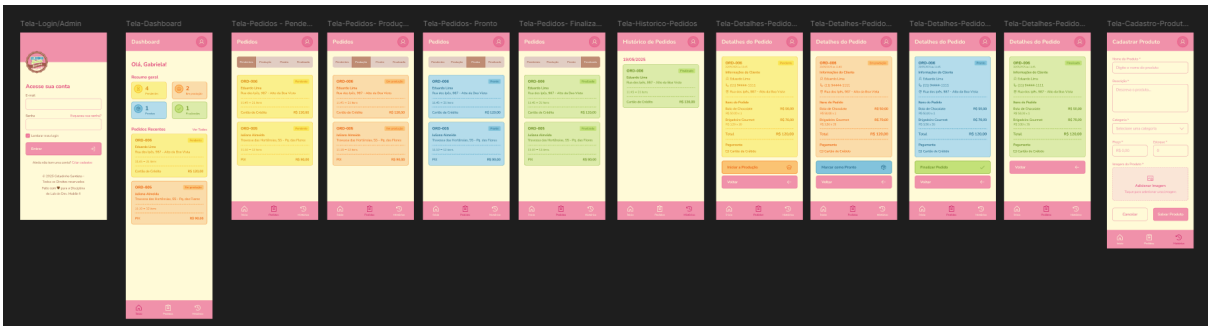
Formulários	Para login, cadastro de produto e alteração de status
Feedback de Status	Uso de cores nos cards para identificação visual rápida (Pendente – Amarelo, Em Produção – Laranja, Pronto – Azul, Finalizado – Verde)

6.5.4 Observações Gerais

- O design seguiu o mesmo estilo visual e a identidade do **App Cardápio Digital**, garantindo unidade de marca entre os dois aplicativos.

- Todas as cores e componentes visuais foram desenhados no **Figma** e posteriormente adaptados para o desenvolvimento em **React Native + Expo Router**, com uso do **Phosphor Icons**.

Figura 4 - Layout do Aplicativo Mobile



7 IMPLEMENTAÇÃO DO ALGORITMO DE MACHINE LEARNING

Este capítulo documenta a implementação do sistema de recomendação híbrido para a Geladinho Santista, desde a coleta de dados até a integração com o aplicativo mobile.

7.1 Objetivo e Funcionalidade do Modelo

Objetivo Principal: Recomendar combos de geladinhos personalizados para cada cliente, aumentando o ticket médio e a satisfação do usuário.

Funcionalidades Chave:

- Recomendações Contextuais: Sugere produtos baseados em clima (ex: frutas em dias quentes)
- Personalização: Adapta-se ao perfil histórico do cliente (ex: preferência por chocolates)
- Cold Start: Usa categorias de produtos para recomendações iniciais (novos clientes)

Técnica Utilizada:

Filtragem Híbrida combinando:

- Filtragem Colaborativa (LightFM)
- Similaridade por Co-ocorrência (Matriz de sabores comprados juntos)

7.2 Coleta e Pré-processamento de Dados

Fontes de Dados:

Fonte: Firestore

Variáveis Coletadas: cliente_id, sabor, data_hora, categoria, valor_total

Exemplo: {cliente_id: "123", sabor: "Nutella", ...}

Fonte: App Mobile

Variáveis Coletadas: clima_local, canal_compra (opcional)

Exemplo: clima: "Quente"

Dados de Treino/Teste:

- Divisão 70/30 com base temporal (pedidos mais recentes para teste)

7.3 Treinamento e Avaliação do Modelo

Arquitetura do LightFM:

Figura 5 - Arquitetura do LightFM

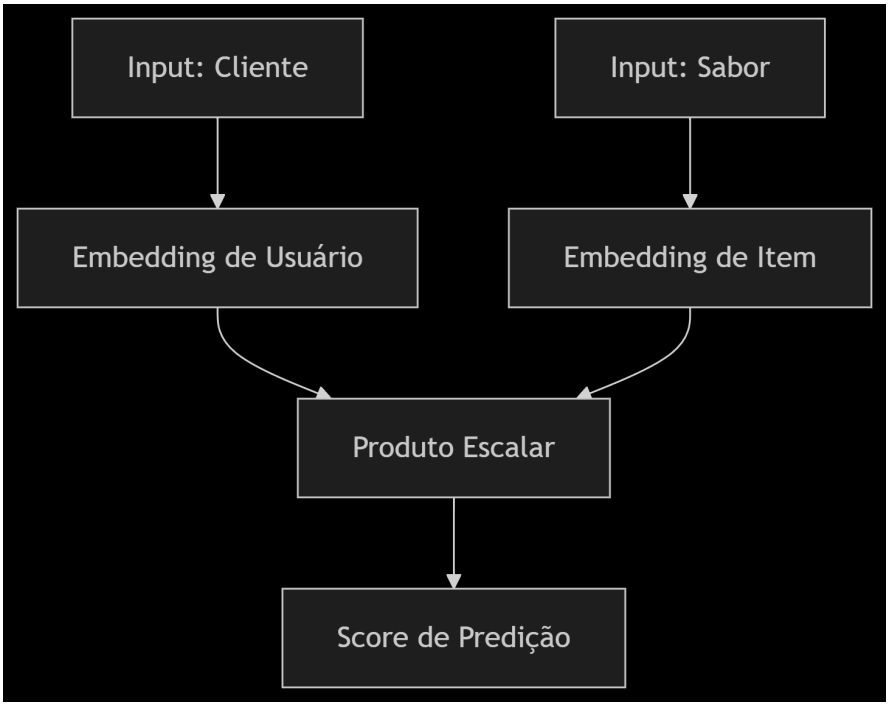


Figura 6 - Código dos Hiperparâmetros

```python
model = LightFM(
    loss='warp-kos',        # Otimizado para ranking
    no_components=30,       # Dimensão dos embeddings
    learning_rate=0.05,
    item_alpha=0.001        # Regularização L2
)
```

Figura 7 - Métricas de Avaliação

Métrica	Valor Obtido	Descrição
Precisão@5	0.63	Acerto nas top 5 recomendações
AUC	0.81	Capacidade de discriminar itens relevantes
Coverage	92%	% de itens do catálogo recomendados

Figura 8 - Código de Avaliação

```python
from lightfm.evaluation import precision_at_k, auc_score
print(f"Precisão@5: {precision_at_k(model, test_data, k=5).mean():.2f}")
print(f"AUC: {auc_score(model, test_data).mean():.2f}")
```

7.4 Deployment e Integração com a API

Fluxo de Deployment:

1. **Exportação do Modelo:**

Figura 9 - Código que exporta o modelo

```python
import joblib
joblib.dump(model, 'modelo_geladinho.pkl')
```

2. **API FastAPI (Colab/Ngrok):**

Figura 10 - Código de recomendação para o cliente tomar um sabor

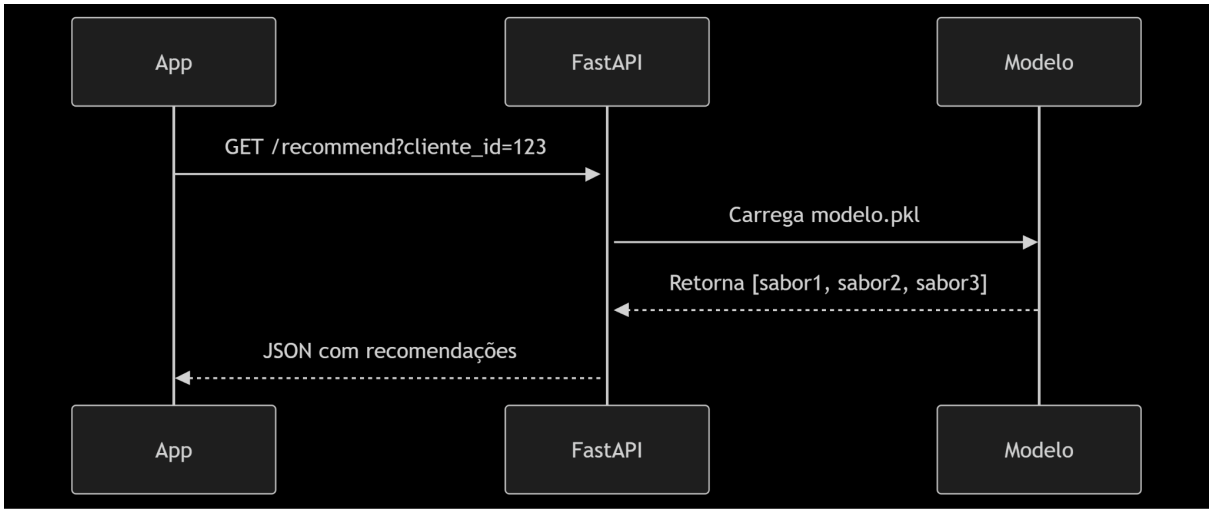
```python
@app.get("/recommend")
async def recommend(cliente_id: str, clima: str = None):
    scores = model.predict(cliente_id, itens_disponiveis)
    if clima == "Quente":
        scores = ajustar_por_clima(scores)  # Reduz score de chocolates
    return {"recommendations": top_3_itens(scores)}
```

3. **Integração Mobile:**

Figura 11 - Código do uso no front-End

```typescript
// React Native
const response = await axios.get(`${API_URL}/recommend`, {
  params: { cliente_id: '123', clima: 'Quente' }
});
```

Diagrama 6 - Diagrama de Sequência do ML



7.5 Monitoramento e Atualização do Modelo

Estratégia de Monitoramento:

- Firebase Analytics: Rastreia cliques nas recomendações
- Métricas Chave:
  - Taxa de Conversão: % de recomendações que viram pedidos
  - Novelty: % de itens novos recomendados

- Pipeline de Retreinamento:

1. Acionador Mensal:

Figura 12 - Código do acionador mensal

```python
# Agendado via GitHub Actions
!python retrain_model.py --source firestore --output modelo_v2.pkl
```

1. Validação Canary:
   - 10% dos usuários recebem recomendações do novo modelo
   - Compara métricas com versão estável

Rollback Automático:

Se precision@5 cair >15%, volta para versão anterior automaticamente.

8 IMPLANTAÇÃO E MANUTENÇÃO

Este capítulo descreve como o sistema é implantado, mantido e monitorado. São detalhados os processos de integração contínua, infraestrutura utilizada para hospedagem, além das práticas de monitoramento, atualizações e suporte ao sistema.

8.1 Estratégia de Deploy (CI/CD)

A estratégia de deploy adotada no projeto segue o modelo de Integração Contínua e Entrega Contínua (CI/CD). O processo é automatizado a partir da branch main do repositório no GitHub. Sempre que há um novo commit na main, os serviços são automaticamente construídos e implantados nas respectivas plataformas de hospedagem.

- **Frontend Web**: Implantado automaticamente na Vercel, que detecta alterações na pasta do web e realiza o build e o deploy da aplicação.
- **Backend (API)**: Implantado na Render, que também realiza o build automaticamente com base no diretório configurado.
- **Serviço de Machine Learning**: O algoritmo em Python é implantado como serviço na Render, com execução contínua ou sob demanda, dependendo da arquitetura configurada.

Essa abordagem reduz falhas humanas, garante entregas frequentes e possibilita testes automatizados antes da publicação.

8.2 Infraestrutura e Hospedagem (Cloud, Servidores, Banco de Dados)

A infraestrutura do sistema está inteiramente baseada em serviços em nuvem, garantindo escalabilidade, disponibilidade e facilidade de manutenção.

- **Front-end Web**: Hospedado na Vercel, plataforma de hospedagem especializada em aplicações front-end modernas. Oferece build automatizado, CDN global e suporte nativo a SSR (Server-Side Rendering).

- **Back-end (API)**: Hospedado na Render, plataforma que permite deploys contínuos de servidores Node.js. O back-end lida com autenticação, regras de negócio e integração com o banco de dados e serviços externos.
- **Machine Learning**: Também hospedado na Render, como uma aplicação Python independente, acessada via API HTTP pelo backend.
- **Banco de Dados**: O sistema utiliza o Firebase Firestore, banco de dados NoSQL em nuvem, escalável, gerenciado pelo Google. O acesso ao banco é feito diretamente pelas APIs do backend.

8.3 Monitoramento e Logs

A aplicação conta com mecanismos básicos de monitoramento e geração de logs oferecidos pelas próprias plataformas de hospedagem:

- **Vercel**: Exibe logs de build e execução do frontend, permitindo fácil diagnóstico de erros.
- **Render**: Apresenta logs em tempo real do backend e dos serviços de ML, além de notificar falhas de deploy ou falhas de execução.
- **Firebase**: Disponibiliza monitoramento de acesso, autenticação e leitura/escrita no banco de dados, com ferramentas integradas ao console para análise de uso.

8.4 Atualizações e Suporte

As atualizações do sistema seguem o mesmo processo de CI/CD: qualquer alteração no código publicada na branch main do GitHub aciona automaticamente o processo de build e deploy nas plataformas configuradas.

A estrutura modular do sistema permite que atualizações sejam feitas de forma isolada em cada parte (frontend, backend ou ML), sem necessidade de reimplantar todo o sistema.

O suporte é realizado pela própria equipe de desenvolvimento, que acompanha os feedbacks dos usuários e realiza correções ou melhorias contínuas, baseadas em testes, logs e métricas de uso do sistema.

9 DOCUMENTAÇÃO TÉCNICA E DE USUÁRIO

Este capítulo tem como objetivo explicar sobre a documentação técnica e de usuário.

9.1 Guia do Desenvolvedor

Este projeto está dividido em quatro módulos principais: Web, Back-end, Mobile e Machine Learning. Cada parte foi desenvolvida com tecnologias modernas, visando escalabilidade e manutenção simples. Abaixo, detalhamos como clonar o repositório e uma visão geral de cada módulo.

Clonando o Repositório

Para começar, clone o repositório oficial do projeto:

git clone https://github.com/odutra-dev/projeto-5dsm.git

Após o clone, você verá a seguinte estrutura de diretórios:

projeto-5dsm/

├── web/

├── back-end/

├── mobile/

└── machine learning/

9.1.1 Parte Web

- **Tecnologia**: Next.js com TypeScript

- **Descrição**: Aplicação web responsável pelo cardápio digital e a criação dos pedidos pelos clientes.

Comandos para iniciar:

cd web

npm install

npm run dev

9.1.2 Parte Back-end

- **Tecnologia**: Node.js com fastify TypeScript

- **Descrição**: API responsável pelas regras de negócio, autenticação, comunicação com o banco de dados e integração com os outros módulos.

  Comandos para iniciar:

  cd back-end

  npm install

  npm run dev

9.1.3 Parte Mobile

- **Tecnologia**: Expo com React Native e TypeScript

- **Descrição**: Aplicativo mobile desenvolvido para rodar em Android e iOS utilizando Expo. Responsável pela parte dos funcionários da empresa e se comunica com a API do back-end.

  Comandos para iniciar:

  cd mobile

  npm install

  npx expo start

9.1.4 Parte Machine Learning

- **Tecnologia**: Python (Google Colab)

- **Descrição**: Contém um notebook do Google Colab com o pipeline de machine learning utilizado no projeto, incluindo pré-processamento, treinamento e inferência de modelos.

  Como acessar:

O arquivo .ipynb pode ser aberto diretamente no Google Colab para testes e modificações.

9.2 Manual do Usuário

9.2.1 Acesso ao Sistema

O sistema desenvolvido é composto por dois aplicativos distintos:

- App Cardápio Digital (Cliente Final)
- App de Gerenciamento de Pedidos (Painel Administrativo)

O acesso a cada aplicativo ocorre da seguinte forma:

- Clientes finais: acessam o Cardápio Digital por meio de um link fornecido pela loja ou através de um QR Code disponível no ponto de venda.
- Administradores: acessam o Painel Administrativo mediante login com credenciais individuais, fornecidas pela equipe da loja.

9.2.2 Navegação no App Cardápio Digital (Cliente Final)

Fluxo de Uso:

- **Tela de Boas-Vindas:**
  O cliente visualiza informações da loja, como endereço, horário de funcionamento e redes sociais, além de um botão para acessar o cardápio.

- **Tela de Cardápio:**
  O usuário visualiza os produtos disponíveis organizados por categorias (ex: Geladinhos, Doces, Bebidas).
- **Tela de Detalhes do Produto:**
  Exibição ampliada com foto, descrição, preço e botão para adicionar ao carrinho.
- **Tela do Carrinho:**
  O cliente revisa os itens selecionados, ajusta quantidades, escolhe o método de pagamento e o tipo de entrega.
- **Tela de Dados para Entrega:**
  Em caso de delivery, o cliente deve informar nome, endereço, telefone e forma de pagamento.
- **Tela de Confirmação de Pedido:**
  Apresenta o resumo final do pedido com o status atual e orientações para aguardar a entrega ou retirada.

9.2.3 Navegação no App de Gerenciamento de Pedidos (Painel Administrativo)

Fluxo de Uso:

- **Tela de Login:**
  Permite que o administrador insira suas credenciais para acesso ao sistema.
- **Dashboard:**
  Visão geral dos pedidos, incluindo gráficos, cards informativos sobre pedidos em andamento, finalizados e produtos vendidos no dia.
- **Tela de Pedidos:**
  Lista de pedidos categorizada por status: Pendente, Em Produção, Pronto e Finalizado.
- **Detalhes do Pedido:**
  Exibição detalhada com informações do cliente, itens do pedido, valor total, forma de pagamento e horário. Possibilidade de atualização do status (ex.: iniciar produção, marcar como pronto, finalizar).

- **Histórico:**
  Consulta de pedidos finalizados com filtros por data e possibilidade de visualização detalhada.
- **Produtos:**
  Lista de produtos cadastrados na loja.
- **Cadastro de Produto:**
  Formulário para inclusão de novos produtos com campos como: nome, descrição, preço e imagem.

9.2.4 Requisitos Técnicos para Uso

Tabela 7 – Requisitos Técnicos para Uso

Aplicativo	**Dispositivo**	**Conexão Necessária**
Cardápio Digital (Cliente)	Smartphone Android / iOS	Internet ou Wi-Fi
Painel Administrativo (Admin)	Smartphone ou tablet Android / iOS	Internet estável

9.2.5 Suporte ao Usuário

Em caso de dúvidas, falhas ou dificuldades de acesso, o usuário pode entrar em contato com a equipe de suporte da loja pelos seguintes canais:

- WhatsApp da loja
- Instagram oficial da marca
- Atendimento presencial

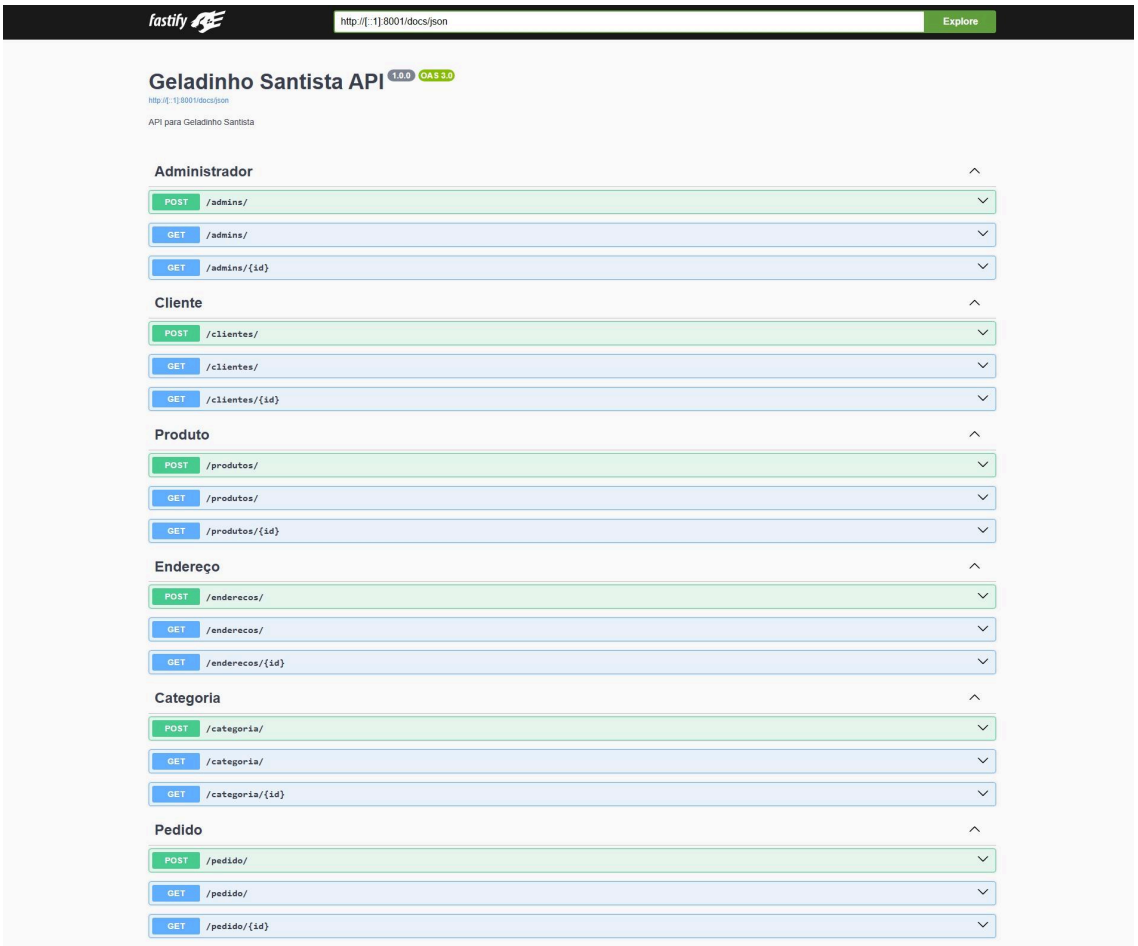
9.3 API Documentation (Swagger ou Postman)

Para facilitar a compreensão e a utilização dos recursos disponibilizados pela API, foi utilizada a ferramenta Swagger para a documentação automática e interativa dos endpoints. Essa abordagem permite que desenvolvedores explorem os métodos disponíveis, verifiquem os parâmetros de entrada e saída, e realizem

testes diretamente na interface, promovendo maior agilidade no desenvolvimento e integração com outros sistemas.

A Figura (9), apresentada a seguir, demonstra a interface gerada pelo Swagger, exibindo de forma clara os endpoints, os métodos HTTP utilizados, os tipos de dados esperados e as respostas possíveis. Essa documentação é essencial para garantir a transparência e a padronização na comunicação entre os sistemas.

Figura 13 - Documentação gerada pelo swagger



9.4 Perguntas Frequentes (FAQ)

9.4.1 O aplicativo estará disponível na Google Play Store?

Não. Inicialmente, o aplicativo será disponibilizado apenas para testes internos, sendo compartilhado diretamente com os clientes selecionados via link de instalação ou QR Code, como parte da fase beta do projeto acadêmico.

9.4.2 O cliente final precisará fazer login?

Não. O aplicativo do Cardápio Digital foi projetado para ser de acesso livre, sem necessidade de cadastro prévio. O objetivo é oferecer uma experiência rápida e prática para o cliente realizar seus pedidos.

9.4.3 O pagamento será feito dentro do aplicativo?

Não. O pagamento continuará sendo realizado pelos métodos tradicionais já usados pela loja, como Pix, cartão de crédito/débito ou dinheiro na entrega. O aplicativo apenas coleta o método escolhido e encaminha a informação para a loja.

9.4.4 O aplicativo será integrado com o sistema atual da loja (Nex)?

Nesta primeira versão, não. O sistema será independente e paralelo ao sistema atual da loja. Caso haja necessidade, futuras integrações podem ser avaliadas em próximas fases de desenvolvimento.

9.4.5 O sistema permite alterar o status dos pedidos?

Sim. No Painel Administrativo (App de Gerenciamento), o administrador pode atualizar o status dos pedidos de forma manual, passando pelas etapas: Pendente → Em Produção → Pronto → Finalizado.

9.4.6 O aplicativo vai armazenar os dados dos clientes?

Apenas os dados necessários para o pedido, como nome, telefone, endereço e forma de pagamento. Os dados não serão utilizados para outros fins e serão tratados apenas para a gestão dos pedidos da loja.

9.4.7 Qual o papel da Inteligência Artificial (Machine Learning) no projeto?

Nesta primeira etapa, a IA será utilizada para análise de histórico de vendas e sugestões internas para o lojista, como identificação de produtos mais vendidos ou sugestões de combos. As recomendações para o cliente final podem ser implementadas em versões futuras.

9.4.8 É possível personalizar o cardápio no futuro?

Sim. Futuramente, novas funcionalidades podem ser adicionadas ao Painel Administrativo, permitindo que a própria loja adicione, edite ou remova produtos do cardápio.

9.4.9 Quem dá manutenção no aplicativo após o projeto?

Como este é um projeto acadêmico, a equipe de alunos é responsável apenas pelo desenvolvimento e entrega da versão final para a disciplina. Caso a loja deseje evoluções futuras, poderá buscar suporte com a faculdade ou com empresas especializadas.

10 GERENCIAMENTO DO PROJETO

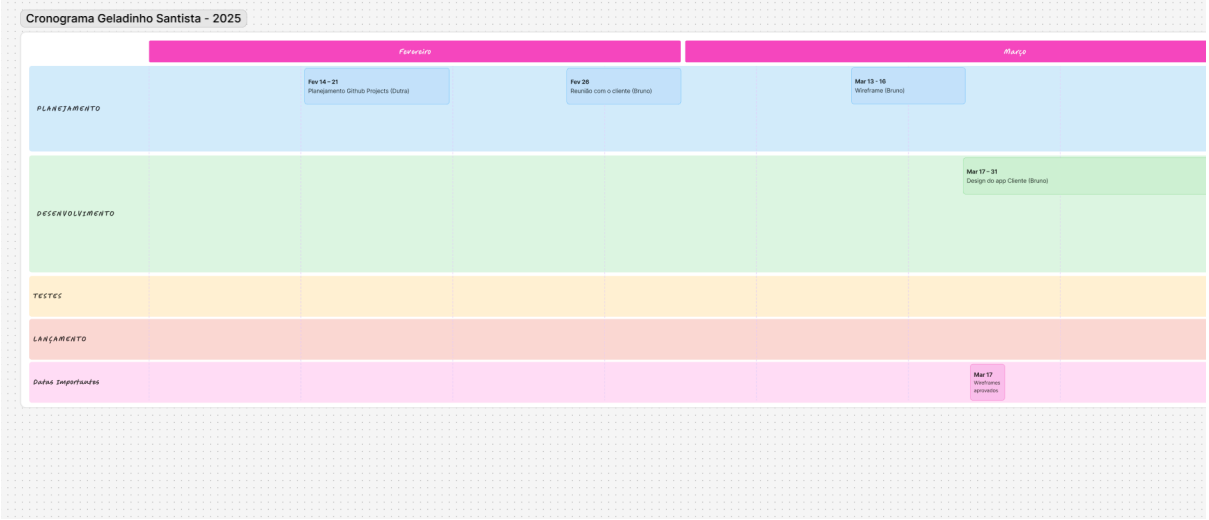
Este capítulo tem como objetivo documentar o gerenciamento do projeto.

10.1 Metodologia Utilizada (Scrum, Kanban, etc.)

A metodologia adotada para o gerenciamento do projeto foi o Kanban, uma abordagem visual para o controle do fluxo de trabalho. Utilizamos quadros com cartões que representam as tarefas, distribuídas entre colunas que indicam o estágio de execução (como “A Fazer”, “Em Andamento” e “Concluído”). Essa metodologia permitiu uma visualização clara das atividades em progresso e facilitou o acompanhamento do andamento do projeto em tempo real. O Kanban foi escolhido por sua simplicidade, flexibilidade e eficácia no gerenciamento de tarefas em equipes dinâmicas.

10.2 Cronograma e Marcos do Projeto

Figura 14 - Cronograma do projeto



Figma. Disponível em: <https://www.figma.com/board/gVRYYTaNYXNhfXLZ0euC01/Gr%C3%A1fico-de-Gantt---Geladinho-Santista?node-id=0-1&t=9EFAbxIHuwPYucFN-1>. Acesso em: 16 jun. 2025.

10.3 Ferramentas de Gerenciamento de Projeto

Durante a execução do projeto, diversas ferramentas foram utilizadas para comunicação e organização das tarefas, entre elas:

- **GitHub Projects**: Usado como ferramenta principal de gerenciamento das atividades, com a criação de quadros Kanban para organizar e acompanhar o progresso das tarefas de desenvolvimento. Os cartões incluíam descrições e responsáveis, facilitando o controle das entregas.
- **WhatsApp**: Utilizado como canal de comunicação rápida entre os membros da equipe, permitindo trocas ágeis de informações, dúvidas e alinhamentos informais ao longo do projeto.
- **Microsoft Teams**: Funcionou como um repositório central de informações, onde foram armazenados documentos, arquivos de referência e registros importantes do projeto. Também foi utilizado para compartilhamento de materiais com fácil acesso por todos os membros da equipe.
- **Discord**: Utilizado como principal plataforma para a realização de reuniões online, discussões em grupo e alinhamentos semanais. Seu uso proporcionou uma comunicação por voz eficiente, além da criação de canais específicos para diferentes tópicos do projeto.

10.4 Gestão de Riscos

A gestão de riscos é uma etapa essencial no desenvolvimento de software, especialmente no que diz respeito à segurança da informação e à integridade do sistema. Durante o ciclo de vida do projeto, foram adotadas práticas e medidas para identificar, avaliar e mitigar possíveis riscos técnicos, operacionais e de segurança.

No aspecto da segurança da informação, foram tomados cuidados específicos com o armazenamento, transmissão e acesso aos dados. A criptografia foi aplicada em dados sensíveis, tanto em trânsito quanto em repouso, garantindo confidencialidade e proteção contra acessos não autorizados. Além disso, políticas

de autenticação e controle de acesso foram implementadas para restringir o uso de funcionalidades críticas apenas a usuários autorizados.

Quanto à qualidade do software, foram realizados testes sistemáticos para garantir o correto funcionamento dos métodos implementados. A validação dos métodos foi feita por meio da comparação entre a lógica esperada e os resultados obtidos, garantindo a aderência às regras de negócio e a confiabilidade das respostas da aplicação.

A aplicação também conta com mecanismos de redundância, como o uso de branches no controle de versões (por exemplo, no Git), e backups periódicos para evitar perda de dados em caso de falhas ou imprevistos. Esses mecanismos contribuem para a continuidade do serviço e para a rápida recuperação do sistema em situações críticas.

Durante o desenvolvimento, foram avaliados os riscos associados a alterações no sistema, considerando o impacto potencial de cada modificação. As mudanças foram submetidas a revisões de código e testes antes da integração definitiva, minimizando erros e garantindo a estabilidade do sistema.

Essas práticas demonstram o compromisso com a segurança, estabilidade e confiabilidade do software, promovendo uma base sólida para sua evolução e manutenção futura.