Project name: HalliApp
Version/Level: 0.0.1 - Documentation
Updated data: 27/11/25
Icon: NULL

Sumário

1. Introduction

1.1. Description:

1.2. Problem:

1.3. Target audience:

1.4. Objective:

1.5. Technologies:

1.5.1. **Front:** React + Tailwind

1.5.2. **Back:** NodeJs

1.5.3. **Database: Firebase**

1.5.4. **Design & Prototype: Figma**

2. Scope

2.1. What does this technology?

O **HalliApp** é uma solução de delivery *SaaS*. Suas capacidades tecnológicas incluem:

- **Gestão Omni-channel:** Controle centralizado de cardápio e finanças para administradores.
- **Interface de Consumo:** Experiência otimizada com filtragem dinâmica por categorias e sacola reativa para o cliente final.
- **Logística Inteligente:** Roteirização para entregadores baseada em proximidade geográfica.
- **Sincronização Real-time:** Status de pedidos atualizados instantaneamente via Firebase.

2.2. What isn't this technology?

- **Não é um Marketplace Geral:** É uma solução *white-label* exclusiva para a marca do estabelecimento, não um agregador como o iFood.
- **Não é um ERP Contábil:** Não realiza gestão fiscal pesada ou folha de pagamento.
- **Não é Gestão de Insumos:** Foca no produto final vendido, não na gramatura técnica de ingredientes crus.

3. Requirements

3.1. Functional requirements:

ID	Requisito	Descrição Técnica
RF01	**Autenticação**	Login via e-mail/senha e provedores sociais usando Firebase Auth.
RF02	**Painel Admin**	Dashboard de KPIs e navegação lateral para gestão do restaurante.
RF03	**Gestão de Menu**	CRUD completo de produtos com controle de estoque e categorias.
RF04	**Motor de Busca**	Filtro *client-side* por texto e categoria (visto no componente `FilterBar.tsx`).
RF05	**Sacola Reativa**	Gestão de estado global (Zustand) para o carrinho (visto no `page.tsx`).
RF06	**Checkout**	Fluxo de fechamento com cálculo de subtotal, taxas e descontos.
RF07	**Pagamentos**	Integração para processamento de PIX e cartões.
RF08	**Logística**	Algoritmo de priorização de entregas por distância geográfica.

RF09	**Relatórios**	Geração de resumos financeiros diários, semanais e mensais.
RF10	**Unidades**	Configuração de geofencing (raio de entrega) e horários de funcionamento.

3.2. Non-functional requirements:

ID	Requisito	Especificação
RNF01	**Performance**	Tempo de carregamento inicial (LCP) inferior a 2 segundos.
RNF02	**Escalabilidade**	Suporte a acessos simultâneos massivos via arquitetura Serverless.
RNF03	**Responsividade**	Design Mobile-First adaptável de 360px até 4K (Tailwind CSS).
RNF04	**Segurança**	Implementação de *Firestore Rules* para isolamento de dados de usuários.
RNF05	**Disponibilidade**	Garantia de 99.9% de tempo de atividade (SLA).

4. Architecture

4.1. Overview

O **HalliApp** utiliza uma arquitetura **BaaS (Backend as a Service)** híbrida. A interface é um WebApp de alta performance (Next.js) que consome o Firebase para persistência reativa (Real-time) e uma camada de serviços Node.js para processamento de lógica complexa, garantindo uma experiência de usuário fluida e dados consistentes para o restaurante.

4.2. Layers

4.3. Front-end (React + NodeJs)

- **Framework:** Next.js 15+ com App Router.
- **Estilização:** Tailwind CSS utilizando Design Tokens para consistência visual.
- **Gerenciamento de Estado:** Zustand para estados globais (Sacola/Filtros) e Hooks customizados para lógica de interface.
- **Comunicação:** SDK do Firebase para leitura direta de dados reativos e Fetch API para endpoints de lógica de negócio.

4.4. Back-end

Runtime: Node.js.

Responsabilidades:

- Cálculo complexo de taxas de entrega.
- Geração de relatórios financeiros agregados (Insights).
- Webhooks para confirmação de pagamentos (PIX/Cartão).
- Disparo de notificações via WhatsApp/E-mail.

4.5. Database / Storage

- **NoSQL Database:** Firebase Firestore para persistência de pedidos, menu e métricas em tempo real.
- **Storage:** Firebase Cloud Storage para armazenamento otimizado de imagens dos produtos (assets).

4.6. API

4.7. Dataflow

- **Entrada:** Usuário interage com o WebApp e gera um evento (ex: Adicionar ao carrinho).
- **Sincronização:** O Firebase Auth valida a sessão; os dados são gravados no Firestore.
- **Processamento:** O Node.js escuta novas ordens, valida o estoque e processa o pagamento via API externa.

- **Feedback:** O status do pedido é atualizado via WebSocket (Real-time) e refletido instantaneamente na tela do cliente e do admin.

4.8. Security & compliance

- **Firestore Rules:** Bloqueio de leitura/escrita por nível de privilégio (Admin vs Cliente).
- **Sanitização:** Validação de inputs no lado do servidor para evitar Injeções.
- **LGPD:** Dados sensíveis de clientes criptografados e política de retenção de dados configurada.

4.9. Deploy & Infra

- **Ambiente de Produção:** Vercel (Edge Functions e Hosting).
- **CI/CD:** Pipeline automatizado via GitHub Actions integrando com Vercel Pro.
- **Escalabilidade:** Auto-scaling nativo da infraestrutura Serverless, suportando picos de tráfego em horários de pico de pedidos.

4.10. Observability / Operação

- **Logs:** Vercel Logs e Google Cloud Logging.
- **Performance:** Monitoramento de Core Web Vitals e Sentry para rastreamento de erros em tempo real.

4.11. Estrutura de dados

- `users`**:** Perfis, endereços e histórico.
- `products`**:** Itens do menu, preços e categorias.
- `orders`**:** Pedidos ativos, histórico de status e log de entrega.
- `analytics`**:** Registros brutos de vendas para geração de dashboards.

5. UML modeling

5.1. Use Case table

Ator	Caso de Uso	Descrição
Cliente	Efetuar Pedido	Seleção de produtos, configuração da sacola e pagamento.
Admin	Gerenciar Menu	Inclusão e alteração de itens do cardápio em tempo real.
Admin	Auditar Vendas	Análise de relatórios e métricas de desempenho do negócio.
Entregador	Roteirizar	Visualização de entregas otimizada por menor distância.
Sistema	Pagamento	Validação de transação e atualização automática do status do pedido.

5.2. Use Case diagram
5.3. Class diagram
5.4. Sequence diagram
5.5. Activities diagram
5.6. States diagram
5.7. Components diagram

6. ER modeling

Para o Firebase Firestore, a estrutura de dados é orientada a documentos:

6.1. Tables

6.2. Coleção `users`:

- `uid`: String (Primary Key - Firebase Auth)
- `name`: String
- `email`: String
- `addresses`: Array of Objects { street, number, coordinates }

6.3. Coleção `products`:

- `id`: String
- `name`: String
- `price`: Number
- `category`: String (ex: proteinas, saladas)
- `imageUrl`: String

6.4. Coleção `orders`:

- `orderId`: String
- `userId`: String (Foreign Key)
- `items`: Array of `CartItem` { id, name, price, quantity }
- `total`: Number
- `status`: String (pending, preparing, out_for_delivery, delivered)

6.5. Attributes

6.6. Datatypes

6.7. Relationship

6.8. Integrity rules

7. System flow

7.1. User Flow

1. **Acesso:** Entrada pela Landing Page (Início).
2. **Identificação:** Login obrigatório para promoções e histórico de pedidos.
3. **Seleção:** Navegação por categorias e busca de itens.
4. **Revisão:** Conferência da sacola (Drawer no mobile / Sidebar no desktop).
5. **Checkout:** Definição de endereço e método de pagamento.
6. **Status:** Acompanhamento do pedido em tempo real.

7.2. Data Flow

1. **Input:** Interação do usuário via componentes React.
2. **Store:** Estado persistido temporariamente via **Zustand** no *client*.
3. **Request:** Envio do pedido para o **Firebase**.
4. **Trigger:** Notificação automática no painel administrativo do restaurante.
5. **Persistence:** Gravação definitiva para relatórios financeiros.

7.3. Nav-Flow (Arquitetura de Navegação)

- **Pública:** `/` (Menu), `/login`, `/cadastro`.
- **Privada (Cliente):** `/pedidos`, `/promocoes`, `/checkout`.
- **Privada (Admin):** `/admin/dashboard`, `/admin/menu`, `/admin/financeiro`.

7.4. Wireframes

8. Business Rules (Regras de Negócio)

Esta seção define as condições lógicas para o funcionamento do software:

- **RN01 - Horário de Funcionamento**: O sistema deve impedir a finalização de pedidos fora do intervalo definido no `restaurantInfo`.
- **RN02 - Cálculo de Frete**: O valor da entrega é calculado com base na distância entre o estabelecimento e o endereço do cliente (geofencing).
- **RN03 - Promoções**: Descontos só são aplicados se o utilizador estiver autenticado e o item pertencer a uma categoria em promoção no Firebase.
- **RN04 - Gestão de Estoque**: Se a quantidade de um produto chegar a zero no Firestore, o botão "Adicionar" deve ser desativado na interface.

9. Tests

9.1. Test strategies

9.2. Test scenarios

9.3. Coverage

9.4. Unit test

9.5. Integration test

9.6. Manual test

10. Deployment

10.1. Step to step to run locally

10.2. Step to step to deploy

10.3. Venv

10.4. Required script

11. Maintenance & Evolution

11.1. Logs

11.2. Backup

11.3. Future updates

11.4. Planned improvements

12. Conclusion