Texto

O conteúdo gerado por IA pode estar incorreto.  
**Data de Entrega:** 10/06/2025  
**Nome do Projeto:** *Quick Lauch*  
**Integrantes do Grupo:**

* William Bitencourt de Castro - 12723115263
* Marcos Leones Leite Souza – 1272312642
* Adailson Souza da Cunha - 1272319876

**1. Descrição Geral**

Este projeto tem como objetivo desenvolver um MVP (Minimum Viable Product) de um aplicativo de delivery de comida, com foco em usabilidade, simplicidade de navegação e responsividade para usuários em dispositivos móveis e desktop. O app "Quick Launch" busca atender usuários que desejam encontrar e comprar refeições de maneira rápida, segura e prática, desde o login até a finalização e avaliação do pedido.

**2. Wireframes**

As imagens dos wireframes das principais telas estão representadas em anexo no material completo. As telas planejadas incluem:

* Tela de Login
* Tela de Cadastro
* Página Inicial (listagem de restaurantes)
* Página de Restaurante (detalhes e produtos e avaliação)
* Carrinho de Compras
* Confirmação de Pedido  
    
  Diagrama

  O conteúdo gerado por IA pode estar incorreto.

**3. Protótipos de Alta Fidelidade**

As telas de alta fidelidade foram desenvolvidas no Figma, respeitando os princípios de design e usabilidade com foco em:

* Cores suaves e agradáveis
* Ícones intuitivos
* Layout limpo e bem-organizado
* Componentes reutilizáveis
* Navegação simples e responsiva  
    
    
  Interface gráfica do usuário

  O conteúdo gerado por IA pode estar incorreto.

**4. Protótipo HTML, CSS e JavaScript**

Um protótipo funcional inicial foi desenvolvido com:

* **HTML:** Estrutura semântica das páginas principais
* **CSS:** Estilização responsiva (mobile-first)
* **JavaScript:** Simulação básica de interações, como exibição do carrinho, login simulado, e confirmação de pedido

Os arquivos estão organizados na pasta /src do repositório com a tag EntregaP1A3.

**5. Arquitetura da Solução e Tecnologias**

Para a construção do projeto, foi utilizada a stack MERN (com SQLite no lugar de MongoDB), seguindo as melhores práticas de desenvolvimento para aplicações web modernas.

Frontend: Desenvolvido em React.js, uma biblioteca JavaScript para a construção de interfaces de usuário dinâmicas e reativas. Utilizamos o React Router para gerenciar a navegação (SPA - Single Page Application) e o Axios para a comunicação com o backend. O estado da aplicação é gerenciado através de Hooks (useState, useEffect) e Context API (para o controle de visualização Desktop/Mobile).

Backend: Construído com Node.js e o framework Express.js, criando uma API RESTful robusta e organizada. A arquitetura é modular, com separação de responsabilidades entre rotas e acesso ao banco de dados.

Banco de Dados: SQLite, um banco de dados relacional leve e baseado em arquivo, ideal para prototipagem e MVPs. Ele garante a integridade dos dados através de chaves primárias e estrangeiras.

**6. Descrição Detalhada do Backend**O servidor backend é o cérebro da aplicação, responsável por toda a lógica de negócio e persistência de dados.

**6.1. Estrutura de Pastas**

**A estrutura foi organizada da seguinte forma:**

/db: Contém a configuração e inicialização do banco de dados SQLite (database.js).

/routes: Cada arquivo corresponde a um recurso da aplicação (ex: auth.js, pedidos.js), definindo os endpoints da API.

app.js: Arquivo principal que configura o servidor Express e anexa as rotas.

seed.js: Script para popular o banco de dados com dados iniciais de teste.

**6.2. Principais Endpoints da API REST**

**A comunicação entre frontend e backend ocorre através dos seguintes endpoints:**

POST /auth/login: Autêntica o usuário.

POST /auth/register: Cadastra um novo usuário.

POST /auth/reset-password: Atualiza a senha.

GET /restaurantes: Retorna a lista de restaurantes.

GET /produtos?restaurante\_id={id}: Retorna os produtos de um restaurante.

GET /carrinho/:usuario\_id: Retorna os itens no carrinho.

POST /carrinho: Adiciona um item ao carrinho.

POST /pedidos: Cria um novo pedido.

GET /pedidos/usuario/:id: Retorna o histórico de pedidos de um usuário.

POST /avaliacoes: Registra a avaliação de um restaurante.

GET /perfil/:usuario\_id: Busca os dados do perfil de um usuário.

POST /perfil/endereco: Adiciona ou atualiza o endereço.

**7. Integração Frontend-Backend**

A integração é o coração da funcionalidade do projeto. O frontend em React consome dinamicamente todos os dados da API REST do backend.

* **Busca de Dados (GET):** Telas de restaurantes, cardápios e histórico de pedidos usam o useEffect para disparar uma requisição axios.get() assim que o componente é montado.
* **Envio de Dados (POST):** Ações como login, cadastro, adicionar ao carrinho e finalizar pedido disparam requisições axios.post() para o backend.
* **Gerenciamento de Estado do Usuário:** Após o login, os dados do usuário são salvos no localStorage do navegador para permitir requisições personalizadas.
* **Feedback ao Usuário:** Todas as chamadas de API são tratadas com blocos then/catch, exibindo alertas de sucesso ou erro.

**8. Heurísticas de Usabilidade Aplicadas**

**O projeto foi desenvolvido com base nas 10 heurísticas de usabilidade de Jakob Nielsen.**

1. Visibilidade do status do sistema: O sistema fornece feedback constante (alertas, mensagens de carregando).
2. Correspondência entre o sistema e o mundo real: A aplicação usa termos e ícones familiares (Carrinho, Perfil, Estrelas de avaliação).
3. Controle e liberdade do usuário: O usuário pode navegar livremente, voltar e sair da sua conta.
4. Consistência e padrões: O design é consistente em toda a aplicação (cabeçalho, cores, botões).
5. Prevenção de erros: O sistema impede ações inválidas (ex: finalizar pedido sem endereço).
6. Reconhecimento em vez de memorização: O carrinho e o histórico de pedidos evitam que o usuário precise memorizar informações.
7. Flexibilidade e eficiência de uso: A aplicação é responsiva e permite alternar a visualização entre desktop e mobile.
8. Estética e design minimalista: A interface é limpa e focada na tarefa principal.
9. Ajudar os usuários a reconhecerem, diagnosticar e recuperar de erros: Mensagens de erro são claras e informativas.
10. Ajuda e documentação: A interface intuitiva dispensa a necessidade de um manual complexo.