Análise Detalhada e Guia Abrangente do Site UTI Gamer Shop Brasa

Prefácio: Um Guia para a Inteligência Artificial e Desenvolvedores

Este documento foi meticulosamente elaborado para servir como um guia exaustivo e um manual operacional para qualquer entidade, seja ela uma inteligência artificial ou um desenvolvedor humano, encarregada de compreender, manter e evoluir o site "UTI Gamer Shop Brasa". Nosso objetivo é desmistificar cada camada desta aplicação, desde suas fundações tecnológicas e a metodologia de desenvolvimento assistida por IA (Lovable) até os mais ínfimos detalhes de suas funcionalidades, estrutura de código e o funcionamento intrincado de seu painel administrativo. A premissa é clara: não haverá ponto obscuro. Cada funcionalidade, cada linha de código relevante, cada integração e cada potencial desafio será dissecado com um nível de detalhe que visa conferir total autonomia e capacidade de intervenção.

Com a crescente adoção de ferramentas de desenvolvimento baseadas em inteligência artificial, torna-se imperativo que a documentação de projetos seja igualmente acessível e compreensível para essas novas entidades. Este guia adota uma abordagem que não apenas descreve o "o quê" e o "como" do site, mas também o "porquê", fornecendo o contexto necessário para tomadas de decisão informadas. Ao final desta leitura, a IA ou o desenvolvedor deverá possuir um conhecimento profundo que transcende a mera superfície do código, permitindo-lhe atuar como um verdadeiro arquiteto e mantenedor do sistema.

1. Introdução e Visão Geral do Projeto "UTI Gamer Shop Brasa"

O "UTI Gamer Shop Brasa" é uma plataforma de e-commerce especializada, dedicada à venda de produtos e serviços para o universo gamer. Sua concepção e implementação refletem as tendências modernas do desenvolvimento web, combinando uma interface de usuário responsiva e dinâmica com um backend robusto e escalável. A análise do repositório de código-fonte revela uma aplicação construída com um conjunto de tecnologias de ponta, projetadas para oferecer uma experiência de compra fluida e

eficiente para o usuário final, ao mesmo tempo em que proporciona ferramentas administrativas poderosas para o gerenciamento do negócio.

1.1. Propósito e Escopo do Site

O principal propósito do "UTI Gamer Shop Brasa" é atuar como um canal de vendas online para uma variedade de produtos relacionados a jogos, incluindo, mas não se limitando a, consoles, jogos (físicos e digitais), acessórios, periféricos e, possivelmente, serviços especializados. O escopo do site abrange:

- **Catálogo de Produtos:** Exibição organizada de produtos com detalhes, imagens, preços e opções de compra.
- **Funcionalidades de E-commerce:** Adição de produtos ao carrinho, gerenciamento de quantidades, remoção de itens e um processo de "checkout" simplificado (neste caso, via WhatsApp).
- **Experiência do Usuário:** Navegação intuitiva, busca eficiente, páginas de categoria e produto detalhadas, e um design responsivo que se adapta a diferentes dispositivos.
- **Autenticação de Usuários:** Sistema de login e cadastro para usuários, com distinção de perfis (usuário comum e administrador).
- **Painel Administrativo:** Uma interface completa para o gerenciamento de conteúdo, produtos, usuários e outras configurações da loja.
- **Integrações:** Conexão com serviços externos para funcionalidades específicas, como o Supabase para backend e autenticação, e o WhatsApp para finalização de pedidos.

1.2. Tecnologias Fundamentais e Arquitetura Geral

A arquitetura do "UTI Gamer Shop Brasa" é baseada em um modelo de aplicação de página única (SPA - Single Page Application) no frontend, que se comunica com um backend baseado em serviços. As tecnologias identificadas no `package.json` e na estrutura de diretórios são:

- **Frontend:** Construído com **React** (uma biblioteca JavaScript para interfaces de usuário), **Vite** (um bundler e servidor de desenvolvimento rápido), **TypeScript** (um superconjunto tipado do JavaScript), **Tailwind CSS** (um framework CSS utility-first para estilização) e **Shadcn UI** (uma coleção de componentes de UI acessíveis e personalizáveis). O roteamento é gerenciado pelo **React Router DOM**, e animações são facilitadas pelo **Framer Motion**. O gerenciamento de estado assíncrono é otimizado com **TanStack Query**.
- **Backend e Banco de Dados:** A aplicação utiliza **Supabase** como sua principal solução de backend-as-a-service (BaaS). O Supabase fornece um banco de dados

PostgreSQL, autenticação de usuários, armazenamento de arquivos e APIs em tempo real, eliminando a necessidade de construir e manter um servidor backend personalizado para muitas das funcionalidades padrão de um e-commerce.
- **Controle de Versão:** O projeto é versionado e gerenciado através do **Git** e hospedado no **GitHub**, garantindo um histórico de alterações robusto e facilitando a colaboração.

Essa combinação de tecnologias resulta em uma aplicação performática, escalável e de fácil manutenção, alinhada com as práticas modernas de desenvolvimento web. A escolha do Supabase, em particular, simplifica significativamente a complexidade do backend, permitindo que os desenvolvedores se concentrem mais na experiência do usuário no frontend.

2. Lovable: A Inteligência Artificial por Trás do Desenvolvimento

Lovable é uma plataforma de desenvolvimento de software assistida por inteligência artificial que visa democratizar a criação de aplicações, permitindo que usuários com diferentes níveis de habilidade transformem ideias em produtos funcionais através de uma interface conversacional. No contexto do "UTI Gamer Shop Brasa", a presença do Lovable sugere uma abordagem de desenvolvimento onde a IA desempenhou um papel significativo na geração de código, na estruturação do projeto ou na automação de tarefas repetitivas.

2.1. Filosofia e Capacidades do Lovable

A filosofia central do Lovable é atuar como um "co-piloto" ou "parceiro de pensamento" no processo de desenvolvimento. Diferente de geradores de código simples, o Lovable busca entender a intenção do usuário através de prompts em linguagem natural e, a partir daí, gerar não apenas trechos de código, mas também a arquitetura e a integração entre diferentes partes da aplicação. Suas capacidades incluem:

- **Geração de Código a Partir de Descrições:** Usuários podem descrever a funcionalidade desejada (ex: "crie um formulário de login", "adicione um carrinho de compras") e o Lovable gera o código React, componentes UI, lógica de estado e até mesmo integrações com serviços como Supabase.
- **Estruturação de Projetos:** O Lovable pode inicializar projetos com uma estrutura de diretórios e arquivos pré-definida, seguindo as melhores práticas para as tecnologias escolhidas (React, Vite, TypeScript, Tailwind CSS).

- **Integração de Bibliotecas e Frameworks:** Ele é capaz de integrar bibliotecas e frameworks populares, configurando-os corretamente e gerando o código boilerplate necessário para seu uso.
- **Manutenção e Refatoração (Potencial):** Embora o foco principal seja a geração, o Lovable também pode auxiliar na refatoração de código existente, na identificação de padrões e na aplicação de melhorias.
- **Versionamento Assistido:** A integração nativa com o GitHub permite que o código gerado pelo Lovable seja imediatamente versionado, facilitando o controle de alterações e a colaboração.

2.2. O Impacto do Lovable no Projeto "UTI Gamer Shop Brasa"

No "UTI Gamer Shop Brasa", a influência do Lovable pode ser observada em vários aspectos:

- **Estrutura Consistente:** A organização lógica dos diretórios (`components` , `pages` , `hooks` , `integrations` , `utils`) e a padronização na nomenclatura de arquivos e componentes podem ser um reflexo das convenções que o Lovable aplica ao gerar projetos.
- **Uso de Tecnologias Modernas:** A escolha de React, Vite, Tailwind CSS e Supabase, que são tecnologias com as quais o Lovable se integra bem, sugere que a plataforma pode ter influenciado ou até mesmo ditado a pilha tecnológica inicial.
- **Código Boilerplate:** Partes do código que são repetitivas ou que seguem padrões bem definidos (ex: configuração inicial de componentes Shadcn UI, chamadas básicas ao Supabase) podem ter sido geradas ou assistidas pelo Lovable.
- **`lovable-tagger` no `devDependencies` :** A presença de `lovable-tagger` no `package.json` (seção `devDependencies`) é um forte indicativo de que o Lovable foi utilizado no processo de desenvolvimento. Esta ferramenta pode ser responsável por adicionar metadados ou tags ao código que o Lovable utiliza para entender e manipular o projeto.

Para a IA que irá intervir no projeto, é fundamental reconhecer que o código pode conter "marcas" do Lovable. Isso significa que, ao invés de tratar o código como puramente manual, a IA deve considerar a possibilidade de que certas estruturas ou padrões foram gerados por uma IA. Isso pode influenciar a forma como as alterações são propostas e implementadas, buscando, quando possível, manter a compatibilidade com a lógica de geração do Lovable ou, alternativamente, refatorar partes do código para uma abordagem totalmente manual, se isso for mais eficiente para a manutenção futura.

3. Integração com GitHub: O Coração do Controle de Versão

A integração do "UTI Gamer Shop Brasa" com o GitHub é um pilar fundamental para a gestão do ciclo de vida do software. O GitHub não é apenas um repositório de código; é uma plataforma colaborativa que oferece ferramentas robustas para controle de versão, gerenciamento de projetos, automação de CI/CD e segurança. A sinergia entre o Lovable e o GitHub é particularmente poderosa, pois permite que o código gerado pela IA seja imediatamente versionado e gerenciado dentro de um ecossistema de desenvolvimento padrão da indústria.

3.1. Fundamentos do Git e GitHub

Para compreender a integração, é essencial revisar os conceitos básicos de Git e GitHub:

- **Git:** Um sistema de controle de versão distribuído que rastreia as alterações no código-fonte durante o desenvolvimento de software. Ele permite que múltiplos desenvolvedores trabalhem no mesmo projeto sem sobrescrever o trabalho uns dos outros, e oferece a capacidade de reverter para versões anteriores, criar branches para novas funcionalidades e mesclar alterações.
- **Repositório (Repo):** O local onde o Git armazena todas as informações do projeto, incluindo o histórico de todas as alterações. No GitHub, um repositório é um projeto online que pode ser acessado e clonado por colaboradores.
- **Commit:** Uma "fotografia" das alterações feitas no código em um determinado momento. Cada commit possui uma mensagem descritiva e um identificador único (hash).
- **Branch:** Uma linha independente de desenvolvimento. Desenvolvedores criam branches para trabalhar em novas funcionalidades ou correções de bugs sem afetar o código principal (geralmente o branch `main` ou `master`).
- **Merge:** O processo de combinar as alterações de um branch em outro. Conflitos de merge podem ocorrer se as mesmas linhas de código forem alteradas em branches diferentes.
- **Pull Request (PR) / Merge Request (MR):** Um mecanismo no GitHub (e outras plataformas) para propor alterações de um branch para outro. Ele permite que outros membros da equipe revisem o código, comentem e aprovem as alterações antes que sejam mescladas.
- **Clone:** O ato de baixar uma cópia de um repositório remoto (do GitHub) para o ambiente de desenvolvimento local.
- **Push:** O ato de enviar os commits do repositório local para o repositório remoto no GitHub.

- **Pull:** O ato de baixar as alterações do repositório remoto para o repositório local e mesclá-las automaticamente.

3.2. A Integração GitHub-Lovable no Contexto do Projeto

A documentação do Lovable [2] e os resultados da pesquisa indicam que a integração com o GitHub é profunda e facilita um fluxo de trabalho contínuo. As principais características dessa integração são:

- **Sincronização Bidirecional:** O Lovable pode enviar o código gerado para um repositório GitHub e, em alguns casos, pode até mesmo ler alterações do GitHub para atualizar seu próprio ambiente ou lógica. Isso garante que o repositório GitHub seja a fonte da verdade para o código-fonte.
- **Versionamento Automático:** Sempre que o Lovable gera ou modifica código, essas alterações podem ser automaticamente comitadas e enviadas para o repositório GitHub, mantendo um histórico detalhado do desenvolvimento assistido por IA.
- **Colaboração Híbrida:** A integração permite que desenvolvedores humanos e a IA do Lovable colaborem no mesmo projeto. As alterações humanas podem ser enviadas para o GitHub, e o Lovable pode então "consumir" essas alterações e continuar seu trabalho, ou vice-versa.
- **Portabilidade:** O fato de o código estar no GitHub significa que o projeto não está "preso" ao ecossistema Lovable. Ele pode ser clonado e desenvolvido independentemente, usando ferramentas de desenvolvimento padrão.

3.3. Implicações para a Manutenção e Evolução pela IA

Para a IA encarregada de manter e evoluir o "UTI Gamer Shop Brasa", a compreensão e o domínio do fluxo de trabalho Git/GitHub são não negociáveis. A IA deverá ser capaz de:

1. **Clonar o Repositório:** O primeiro passo prático será sempre clonar o repositório GitHub para o ambiente de trabalho da IA. Assumindo que o repositório é `https://github.com/usuario/uti-gamer-shop-brasa.git`, o comando seria: `bash git clone https://github.com/usuario/uti-gamer-shop-brasa.git` (Nota: O usuário precisará fornecer o URL correto do repositório, que não está no ZIP fornecido).

2. **Criar e Gerenciar Branches:** Para cada nova funcionalidade, correção de bug ou experimento, a IA deve criar um novo branch. Isso isola as alterações e evita quebras no branch principal. `bash git checkout -b feature/nova-funcionalidade`

3. **Realizar Commits Atômicos:** Após cada conjunto lógico de alterações, a IA deve realizar um commit com uma mensagem clara e concisa que descreva o que foi alterado e por quê. Isso facilita a revisão do histórico e a identificação de problemas. `bash git add . git commit -m "feat: Implementa funcionalidade de busca fuzzy aprimorada"`

4. **Sincronizar com o Repositório Remoto:** Regularmente, a IA deve buscar as últimas alterações do repositório remoto para evitar conflitos e garantir que está trabalhando com a versão mais atualizada do código. `bash git pull origin main` Após realizar commits locais, a IA deve enviá-los para o repositório remoto: `bash git push origin feature/nova-funcionalidade`

5. **Gerenciar Pull Requests:** Se a IA estiver operando em um ambiente colaborativo (com outros desenvolvedores ou IAs), ela precisará criar Pull Requests para propor suas alterações ao branch principal. A IA deve ser capaz de:

    - Abrir um PR, descrevendo as alterações e o problema/funcionalidade que resolve.
    - Responder a comentários e sugestões de revisão de código.
    - Resolver conflitos de merge, se surgirem. Isso é um ponto crítico e exige uma compreensão profunda do código e das alterações conflitantes.
    - Mesclar o PR após a aprovação.

6. **Reverter Alterações:** Em caso de introdução de um bug ou de uma funcionalidade indesejada, a IA deve ser capaz de reverter commits ou branches inteiros, utilizando comandos como `git revert` ou `git reset`.

O domínio dessas operações Git/GitHub é crucial para que a IA possa operar de forma eficaz e segura no ambiente de desenvolvimento do "UTI Gamer Shop Brasa", garantindo a integridade do código e a colaboração eficiente.

4. Análise Técnica Detalhada do Código: Frontend

O frontend do "UTI Gamer Shop Brasa" é uma aplicação React robusta, construída com uma pilha de tecnologias modernas que visam otimizar o desenvolvimento, a performance e a experiência do usuário. A compreensão aprofundada de cada componente dessa pilha é vital para qualquer intervenção no código.

4.1. React: A Fundação da Interface do Usuário

React é uma biblioteca JavaScript declarativa, eficiente e flexível para a construção de interfaces de usuário. Ele permite compor UIs complexas a partir de pequenas e isoladas peças de código chamadas "componentes".

- **Componentes Funcionais e Hooks:** O projeto utiliza a abordagem moderna do React, com componentes funcionais e Hooks (como `useState`, `useEffect`, `useContext`, `useRef`). Isso promove um código mais limpo, conciso e reutilizável. Por exemplo, o `useAuth.tsx` e `useNewCart.ts` são exemplos de hooks personalizados que encapsulam lógica de negócio e estado.
- **JSX:** A sintaxe JSX (JavaScript XML) é utilizada para descrever a estrutura da UI. É uma extensão do JavaScript que permite escrever HTML dentro do código JavaScript, facilitando a criação de componentes.
- **Virtual DOM:** React utiliza um Virtual DOM para otimizar as atualizações da UI. Em vez de manipular o DOM diretamente, o React compara o Virtual DOM atual com o anterior e aplica apenas as mudanças necessárias ao DOM real, resultando em melhor performance.

Exemplo de Componente React (`src/components/ui/button.tsx`):

```
import * as React from "react"
import { Slot } from "@radix-ui/react-slot"
import { cva, type VariantProps } from "class-variance-
authority"
import { cn } from "@/lib/utils"

const buttonVariants = cva(
  "inline-flex items-center justify-center whitespace-nowrap
rounded-md text-sm font-medium ring-offset-background
transition-colors focus-visible:outline-none focus-
visible:ring-2 focus-visible:ring-ring focus-visible:ring-
offset-2 disabled:pointer-events-none disabled:opacity-50",
  {
    variants: {
      variant: {
        default: "bg-primary text-primary-foreground hover:bg-
primary/90",
        destructive:
"bg-destructive text-destructive-foreground hover:bg-
destructive/90",
        outline: "border border-input bg-background hover:bg-
accent hover:text-accent-foreground",
        secondary: "bg-secondary text-secondary-foreground
hover:bg-secondary/80",
        ghost: "hover:bg-accent hover:text-accent-foreground",
        link: "text-primary underline-offset-4 hover:underline",
```

```
      },
      size: {
        default: "h-10 px-4 py-2",
        sm: "h-9 rounded-md px-3",
        lg: "h-11 rounded-md px-8",
        icon: "h-10 w-10",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)

export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement>,
    VariantProps<typeof buttonVariants> {
  asChild?: boolean
}

const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
  ({ className, variant, size, asChild = false, ...props },
ref) => {
    const Comp = asChild ? Slot : "button"
    return (
      <Comp
        className={cn(buttonVariants({ variant, size,
className }))}
        ref={ref}
        {...props}
      />
    )
  }
)
Button.displayName = "Button"

export { Button, buttonVariants }
```

Este é um componente `Button` do Shadcn UI. Ele demonstra o uso de `cva` para gerenciar variantes de estilo (default, destructive, outline, etc.) e tamanhos (default, sm, lg, icon) usando Tailwind CSS. A função `cn` (de `clsx` e `tailwind-merge`) é usada para combinar classes CSS de forma inteligente. A IA deve entender que, para modificar a aparência de botões em todo o site, as classes Tailwind dentro de `buttonVariants` devem ser ajustadas, ou novas variantes podem ser adicionadas.

4.2. Vite: Otimização do Ambiente de Desenvolvimento e Build

Vite é um bundler de próxima geração que se destaca pela sua velocidade e eficiência. Ele utiliza módulos ES nativos no navegador durante o desenvolvimento, o que elimina a necessidade de um processo de bundling completo antes de servir o código, resultando em tempos de inicialização e Hot Module Replacement (HMR) quase instantâneos.

- **Desenvolvimento Rápido:** A IA notará que as alterações no código são refletidas no navegador quase que instantaneamente, o que acelera o ciclo de feedback durante o desenvolvimento.
- **Build Otimizado:** Para produção, o Vite utiliza Rollup para bundling, aplicando otimizações como tree-shaking e code splitting para gerar pacotes de código menores e mais performáticos.
- **Configuração (`vite.config.ts`):** Este arquivo define as configurações do Vite, incluindo plugins, aliases de caminho (como `@/` para `src/`), e opções de build. A IA pode precisar modificar este arquivo para adicionar novos plugins, configurar proxies ou ajustar o processo de build.

4.3. TypeScript: Segurança e Manutenibilidade do Código

TypeScript é um superconjunto tipado do JavaScript que compila para JavaScript puro. Ele adiciona tipagem estática opcional ao código, o que permite capturar erros em tempo de desenvolvimento (antes mesmo de o código ser executado) e melhora a legibilidade e a manutenibilidade do código, especialmente em projetos grandes.

- **Tipagem Forte:** A IA encontrará interfaces e tipos definidos em arquivos como `src/types/cart.ts` e `src/types/specialSections.ts`, bem como tipagens inline em componentes e hooks. Isso ajuda a entender a estrutura dos dados e as expectativas de cada função.
- **Refatoração Segura:** Com TypeScript, a IA pode realizar refatorações com mais confiança, pois o compilador alertará sobre quaisquer quebras de contrato de tipo.
- **Documentação Implícita:** As definições de tipo servem como uma forma de documentação, descrevendo a forma dos objetos e as assinaturas das funções.

Exemplo de Tipagem (`src/types/cart.ts`):

```
export interface CartItem {
  id: string;
  name: string;
  price: number;
  quantity: number;
  image?: string;
  size?: string;
```

```
  color?: string;
  // Adicione outras propriedades relevantes do produto aqui
}

export interface CartState {
  items: CartItem[];
}
```

Esta interface `CartItem` define a estrutura de um item no carrinho, e `CartState` define o estado geral do carrinho. A IA deve sempre respeitar e atualizar essas tipagens ao modificar a estrutura de dados.

4.4. Tailwind CSS e Shadcn UI: Estilização e Componentes

Tailwind CSS é um framework CSS utility-first que permite construir designs personalizados diretamente no markup. Em vez de classes semânticas (ex: `.card-title`), ele usa classes utilitárias de baixo nível (ex: `text-lg font-bold mb-2`).

- **Desenvolvimento Rápido de UI:** A IA notará que a maioria dos estilos é aplicada diretamente nos elementos JSX, o que agiliza o desenvolvimento da UI.
- **Consistência:** A configuração do Tailwind (`tailwind.config.ts`) permite definir um sistema de design consistente (cores, espaçamentos, tipografia) que é aplicado em todo o site.
- **Responsividade:** O Tailwind facilita a criação de layouts responsivos usando prefixos como `sm:`, `md:`, `lg:` para aplicar estilos condicionalmente com base no tamanho da tela.

Shadcn UI é uma coleção de componentes de UI construídos com Radix UI e estilizados com Tailwind CSS. Ele não é uma biblioteca de componentes tradicional que você instala e importa; em vez disso, você copia o código-fonte dos componentes para o seu projeto e os personaliza. Isso oferece total controle sobre o estilo e o comportamento.

- **Componentes Reutilizáveis e Acessíveis:** Os componentes Shadcn UI (localizados em `src/components/ui/`) são projetados para serem acessíveis e podem ser facilmente personalizados. A IA deve entender que esses componentes são a base da UI do site.
- **Personalização:** Para alterar a aparência de um componente Shadcn UI, a IA deve modificar as classes Tailwind CSS dentro do arquivo do componente ou estender o componente com classes adicionais.

4.5. React Router DOM: Navegação na Aplicação

React Router DOM é a biblioteca padrão para roteamento declarativo em aplicações React. Ele permite mapear URLs para componentes React, criando uma experiência de navegação de página única.

- **`BrowserRouter` e `Routes`:** O `src/App.tsx` utiliza `BrowserRouter` para habilitar o roteamento baseado em URL e `Routes` para definir as diferentes rotas da aplicação.
- **`Route`:** Cada `Route` mapeia um `path` (URL) para um `element` (componente React). Por exemplo, `<Route path="/" element={<Index />} />` significa que a URL raiz (`/`) renderiza o componente `Index`.
- **`useParams`, `useNavigate`:** Hooks do React Router DOM usados para acessar parâmetros da URL (ex: `id` do produto em `/produto/:id`) e para navegar programaticamente entre as páginas.

Exemplo de Roteamento (`src/App.tsx`):

```
// ... imports

const App = () => (
  <QueryClientProvider client={queryClient}>
    <AuthProvider>
      <CartProvider>
        <TooltipProvider>
          <Toaster />
          <Sonner />
          <BrowserRouter>
            <ScrollRestorationProvider>
              <Routes>
                {/* Public Routes */}
                <Route path="/" element={<Index />} />
                <Route path="/busca" element={<SearchResults />
>} />
                <Route path="/categoria/:category"
element={<CategoryPage />} />

                {/* Novas rotas para páginas de plataforma
específicas */}
                <Route path="/xbox" element={<XboxPage />} />
                <Route path="/playstation"
element={<PlayStationPage />} />
                <Route path="/nintendo" element={<NintendoPage />
>} />

                <Route path="/uti-pro" element={<UTIPro />} />
```

```
                {/* Dynamic Carousel Page Route */}
                <Route
                  path="/secao-especial/:sectionId/
carrossel/:carouselIndex"
                  element={<SpecialSectionCarouselPage />}
                />

                {/* Admin Route - Protected */}
                <Route
                  path="/admin"
                  element={
                    <ProtectedAdminRoute>
                      <AdminPanel />
                    </ProtectedAdminRoute>
                  }
                />

                {/* Catch-all Not Found Route */}
                <Route path="*" element={<NotFound />} />
              </Routes>
            </ScrollRestorationProvider>
          </BrowserRouter>
        </TooltipProvider>
      </CartProvider>
    </AuthProvider>
  </QueryClientProvider>
);

export default App;
```

Para adicionar uma nova página, a IA deve: 1. Criar o componente da página em `src/pages/`. 2. Importar o componente em `src/App.tsx`. 3. Adicionar uma nova `<Route>` com o `path` desejado e o `element` apontando para o novo componente.

4.6. TanStack Query: Gerenciamento de Dados Assíncronos

TanStack Query (anteriormente React Query) é uma biblioteca poderosa para buscar, armazenar em cache, sincronizar e atualizar dados assíncronos no React. Ele simplifica o gerenciamento de estado do servidor, eliminando a necessidade de lógica complexa para carregamento, erros, caching e refetching.

- **`QueryClientProvider` e `QueryClient`:** O `App.tsx` envolve a aplicação com `QueryClientProvider`, fornecendo uma instância de `QueryClient` que gerencia o cache de dados.
- **`useQuery`, `useMutation`:** Hooks como `useProducts` (que provavelmente usa `useQuery` internamente) são responsáveis por buscar dados do Supabase. `useMutation` seria usado para operações de escrita (criar, atualizar, deletar).

- **Benefícios:** A IA deve reconhecer que o TanStack Query lida automaticamente com o estado de carregamento (`isLoading`), erros (`isError`), e dados (`data`), além de otimizar o número de requisições ao backend através de caching e deduplicação.

4.7. Framer Motion: Animações e Transições

Framer Motion é uma biblioteca de animação de código aberto para React. Ela facilita a criação de animações fluidas e interativas com uma API declarativa.

- **Componentes `motion` :** A IA pode encontrar componentes como `<motion.div>` ou `<motion.button>` no código, que são wrappers do Framer Motion que permitem adicionar propriedades de animação (ex: `initial` , `animate` , `exit` , `transition`).
- **Experiência do Usuário:** A presença do Framer Motion indica que o site provavelmente incorpora animações para melhorar a experiência do usuário, como transições suaves entre páginas, efeitos de hover ou animações de entrada/saída de elementos.

5. Análise Técnica Detalhada do Código: Backend e Integrações

O backend do "UTI Gamer Shop Brasa" é amplamente facilitado pelo Supabase, uma plataforma BaaS que abstrai a complexidade de gerenciar um servidor e um banco de dados, permitindo que o frontend interaja diretamente com os serviços de backend através de APIs.

5.1. Supabase: Backend-as-a-Service Completo

Supabase é uma alternativa open-source ao Firebase, oferecendo um conjunto de ferramentas para construir backends rapidamente. Ele é composto por:

- **PostgreSQL Database:** Um banco de dados relacional robusto e escalável. Todos os dados do site (produtos, usuários, categorias, pedidos, etc.) são armazenados aqui.
- **Auth:** Um serviço de autenticação de usuários que suporta email/senha, OAuth (Google, GitHub, etc.) e outras opções. O `useAuth.tsx` interage diretamente com este serviço.
- **Storage:** Um serviço para armazenar arquivos, como imagens de produtos. As imagens carregadas no painel administrativo provavelmente são armazenadas aqui.

- **Realtime:** Permite que os clientes se inscrevam em mudanças no banco de dados e recebam atualizações em tempo real.
- **Edge Functions:** Funções serverless que podem ser usadas para lógica de backend personalizada.

Conexão com Supabase (`src/integrations/supabase/client.ts`):

```
// This file is automatically generated. Do not edit it
directly.
import { createClient } from '@supabase/supabase-js';
import type { Database } from './types';

const SUPABASE_URL = "https://pmxnfpnnvtuuiedoxuxc.supabase.co";
const SUPABASE_PUBLISHABLE_KEY =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InBt

// Import the supabase client like this:
// import { supabase } from "@/integrations/supabase/client";

export const supabase = createClient<Database>(SUPABASE_URL,
SUPABASE_PUBLISHABLE_KEY, {
  auth: {
    storage: localStorage,
    persistSession: true,
    autoRefreshToken: true,
    detectSessionInUrl: true
  }
});
```

Observação Crítica: A linha `// This file is automatically generated. Do not edit it directly.` é de extrema importância. Isso significa que as credenciais (`SUPABASE_URL`, `SUPABASE_PUBLISHABLE_KEY`) e a tipagem do banco de dados (`Database`) são geradas por um processo externo, provavelmente o Lovable ou um script de build. A IA **NÃO DEVE** alterar essas credenciais diretamente neste arquivo. Para modificar a conexão com o Supabase (ex: apontar para uma nova instância), a IA precisará identificar o processo que gera este arquivo e modificar a fonte das credenciais (provavelmente variáveis de ambiente ou um arquivo de configuração do Lovable).

5.2. Autenticação com Supabase (`src/hooks/useAuth.tsx`)

O hook `useAuth.tsx` é o ponto central para toda a lógica de autenticação do usuário. Ele interage diretamente com o serviço de autenticação do Supabase.

- **`signIn`, `signUp`, `signOut`:** Funções assíncronas que chamam os métodos correspondentes do cliente Supabase (`supabase.auth.signInWithPassword`, `supabase.auth.signUp`, `supabase.auth.signOut`).
- **`onAuthStateChange`:** Um listener que reage a mudanças no estado de autenticação do usuário (login, logout, refresh de sessão). Quando o estado muda, ele atualiza o `user` e `session` no contexto.
- **Verificação de Admin:** O hook verifica a função do usuário (`profile?.role === 'admin'`) consultando a tabela `profiles` no Supabase. Isso é crucial para proteger as rotas administrativas.

Exemplo de Lógica de Autenticação (`useAuth.tsx`):

```
// ... imports

export const AuthProvider = ({ children }: { children:
React.ReactNode }) => {
  const [user, setUser] = useState<User | null>(null);
  const [session, setSession] = useState<Session | null>(null);
  const [isAdmin, setIsAdmin] = useState(false);
  const [loading, setLoading] = useState(true);
  const { toast } = useToast();

  useEffect(() => {
    const { data: { subscription } } =
supabase.auth.onAuthStateChange(
      async (event, session) => {
        setSession(session);
        setUser(session?.user ?? null);
        if (session?.user) {
          setTimeout(async () => {
            try {
              const { data: profile } = await supabase
                .from("profiles")
                .select("role")
                .eq("id", session.user.id)
                .single();

              setIsAdmin(profile?.role === "admin");
            } catch (error) {
              console.log("Erro ao verificar perfil:", error);
              setIsAdmin(false);
            }
```

```
          }, 0);
        } else {
          setIsAdmin(false);
        }

        setLoading(false);
      }
    );

    supabase.auth.getSession().then(({ data: { session } }) => {
      setSession(session);
      setUser(session?.user ?? null);
      setLoading(false);
    });

    return () => subscription.unsubscribe();
  }, []);

  // ... signIn, signUp, signOut functions

  return (
    <AuthContext.Provider value={{
      user,
      session,
      isAdmin,
      loading,
      signIn,
      signUp,
      signOut,
    }}>
      {children}
    </AuthContext.Provider>
  );
};

// ... useAuth hook
```

Para modificar a lógica de autenticação (ex: adicionar campos de perfil, integrar com outros provedores OAuth), a IA precisará intervir neste arquivo e, possivelmente, nas configurações do projeto Supabase.

5.3. Gerenciamento de Dados com Supabase (Ex: Produtos)

O `useProducts.ts` é um exemplo de como os dados são buscados e gerenciados a partir do Supabase. Ele provavelmente utiliza o `supabase.from('tabela').select('*')` para buscar dados do banco de dados.

- **`productApi.ts`:** Este arquivo dentro de `src/hooks/useProducts/` provavelmente contém as funções que interagem diretamente com o Supabase

para operações CRUD de produtos. A IA deve analisar este arquivo para entender como os produtos são buscados, criados, atualizados e deletados.
- **`types.ts`:** As definições de tipo para os produtos e outras entidades do banco de dados são cruciais para garantir a consistência dos dados.

Exemplo de Interação com Banco de Dados (Inferido de `useProducts` e `productApi.ts`):

```
// Exemplo hipotético de src/hooks/useProducts/productApi.ts
import { supabase } from "@/integrations/supabase/client";
import { Product } from "./types";

export const fetchProducts = async (): Promise<Product[]> => {
  const { data, error } = await
supabase.from("products").select("*");
  if (error) throw error;
  return data;
};

export const createProduct = async (product: Omit<Product,
'id'>): Promise<Product> => {
  const { data, error } = await
supabase.from("products").insert(product).single();
  if (error) throw error;
  return data;
};

// ... outras funções para update, delete
```

Para alterar a forma como os produtos são buscados (ex: adicionar filtros, ordenação, paginação) ou para modificar a lógica de criação/atualização, a IA deve modificar as funções em `productApi.ts`.

5.4. Armazenamento de Arquivos (Supabase Storage)

Imagens de produtos, banners e outros ativos de mídia são provavelmente armazenados no Supabase Storage. O componente `ImageUploadInput.tsx` (em `src/components/Admin/`) é um forte indicativo disso.

- **Upload de Imagens:** O `ImageUploadInput.tsx` deve conter a lógica para fazer upload de arquivos para um bucket específico no Supabase Storage, gerando URLs públicas para essas imagens.
- **URLs de Imagens:** As URLs das imagens são armazenadas no banco de dados (na tabela de produtos, por exemplo) e são usadas para exibir as imagens no frontend.

5.5. Integração com WhatsApp (`src/utils/whatsapp.ts`)

O arquivo `whatsapp.ts` contém a lógica para gerar um link do WhatsApp com uma mensagem pré-preenchida, permitindo que os usuários finalizem seus pedidos ou entrem em contato diretamente via WhatsApp.

- **`sendToWhatsApp` (em `useNewCart.ts`):** Esta função provavelmente constrói a mensagem com base nos itens do carrinho e nas informações do produto, e então utiliza a função do `whatsapp.ts` para abrir o link.

Exemplo de `whatsapp.ts` (Inferido):

```
// Exemplo hipotético de src/utils/whatsapp.ts
export const generateWhatsAppLink = (phoneNumber: string,
message: string): string => {
  const encodedMessage = encodeURIComponent(message);
  return `https://wa.me/${phoneNumber}?text=${encodedMessage}`;
};

export const sendOrderToWhatsApp = (items: any[], total:
number): void => {
  const phoneNumber = "55XXYYYYYYYYY"; // Número de telefone da
loja
  let message = "Olá! Gostaria de fazer um pedido com os
seguintes itens:\n\n";
  items.forEach(item => {
    message += `- ${item.name} (x${item.quantity}) - R$ $
{item.price.toFixed(2)}\n`;
  });
  message += `\nTotal: R$ ${total.toFixed(2)}\n\nPor favor,
confirme a disponibilidade e o valor total.`;

  window.open(generateWhatsAppLink(phoneNumber, message),
"_blank");
};
```

Para alterar o número de telefone de destino, o formato da mensagem ou adicionar mais detalhes ao pedido enviado via WhatsApp, a IA deve modificar este arquivo e/ou a função `sendToWhatsApp` no `useNewCart.ts`.

6. Estrutura de Diretórios e Mapeamento de Componentes

A organização do código é crucial para a manutenibilidade. O projeto "UTI Gamer Shop Brasa" segue uma estrutura modular e lógica, facilitando a localização e modificação de funcionalidades específicas.

6.1. Visão Geral da Estrutura (`src/`)

```
src/
├── assets/                 # Imagens, ícones e outros arquivos
estáticos
├── components/             # Componentes reutilizáveis da UI
│   ├── Admin/              # Componentes do painel
administrativo
│   ├── Auth/               # Componentes de autenticação
│   ├── Header/             # Componentes do cabeçalho
│   ├── ProductCard/        # Componentes para exibição de
cartões de produto
│   ├── ProductModal/       # Componentes do modal de produto
│   ├── ProductPage/        # Componentes específicos da página
de produto
│   ├── ui/                 # Componentes genéricos do Shadcn UI
│   └── ...                 # Outros componentes por
funcionalidade
├── contexts/               # Contextos React para
gerenciamento de estado global
├── data/                   # Dados mockados ou estáticos
├── hooks/                  # Hooks personalizados para lógica
reutilizável
│   ├── useProducts/        # Lógica de produtos (API, tipos,
etc.)
│   └── ...                 # Outros hooks por funcionalidade
├── integrations/           # Integrações com serviços externos
(Supabase)
├── lib/                    # Funções utilitárias e helpers
├── pages/                  # Componentes que representam as
páginas da aplicação
│   ├── Admin/              # Componentes da página Admin
│   ├── platforms/          # Páginas específicas de plataformas
│   └── ...                 # Outras páginas
├── styles/                 # Arquivos CSS globais
├── types/                  # Definições de tipos TypeScript
├── utils/                  # Funções utilitárias diversas
├── App.tsx                 # Componente raiz da aplicação e
roteamento
```

```
├── index.css               # Estilos CSS globais
└── main.tsx                # Ponto de entrada da aplicação
```

6.2. Mapeamento Detalhado de Componentes e Funcionalidades

Esta seção mapeia as funcionalidades do site para os componentes e arquivos de código correspondentes, fornecendo um guia prático para a IA.

6.2.1. Funcionalidades de Navegação e Layout

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
Estrutura Principal da Aplicação	`src/App.tsx`, `src/main.tsx`	`main.tsx` é o ponto de entrada que renderiza `App.tsx`. `App.tsx` define o roteamento (`BrowserRouter`, `Routes`, `Route`) e os provedores de contexto (`AuthProvider`, `CartProvider`, `QueryClientProvider`, `TooltipProvider`). Para adicionar novas rotas, provedores ou alterar a estrutura global, modifique `App.tsx`.
Cabeçalho Principal	`src/components/Header/MainHeader.tsx`, `src/components/Header/ProfessionalHeader.tsx`, `src/components/Header/DesktopNavigation.tsx`, `src/components/Header/MobileMenu.tsx`, `src/components/Header/HeaderActions.tsx`, `src/components/Header/DesktopSearchBar.tsx`, `src/components/Header/MobileSearchBar.tsx`, `src/`	O cabeçalho é complexo e modular. `ProfessionalHeader.tsx` é usado na página inicial. `MainHeader.tsx` pode ser usado em outras páginas. Contém navegação, busca, ações de usuário (login, carrinho). Para alterar o logo, links de navegação, comportamento da busca ou adicionar/remover elementos, edite os componentes específicos dentro de `src/components/Header/`. O `PromotionalBanner.tsx` é um

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
	`components/Header/PromotionalBanner.tsx`	componente separado para banners no topo.
Rodapé	`src/components/Footer.tsx`	Contém links de navegação, informações de contato, direitos autorais. Para atualizar links, informações da empresa ou layout do rodapé, modifique `Footer.tsx`.
Página Inicial	`src/pages/Index.tsx`, `src/components/HomePage/HomePageContent.tsx`, `src/components/HomePage/SectionRenderer.tsx`, `src/components/SpecialSections/SpecialSectionRenderer.tsx`, `src/hooks/useIndexPage.ts`	`Index.tsx` orquestra a exibição de banners, links rápidos e seções de produtos. `useIndexPage.ts` gerencia a lógica de carregamento de dados para a página inicial. `SectionRenderer.tsx` e `SpecialSectionRenderer.tsx` renderizam as diferentes seções de conteúdo. Para alterar o layout da página inicial, a ordem das seções ou o tipo de conteúdo exibido, a IA precisará entender a lógica em `useIndexPage.ts` e como `layoutItems` são processados em `Index.tsx`.
Páginas de Categoria	`src/pages/CategoryPage.tsx`, `src/pages/platforms/XboxPage.tsx`, `src/pages/platforms/PlayStationPage.tsx`, `src/pages/platforms/NintendoPage.tsx`	`CategoryPage.tsx` é uma página genérica para categorias. As páginas em `src/pages/platforms/` são específicas para consoles. Para adicionar novas categorias ou modificar o layout de exibição de produtos por categoria, edite `CategoryPage.tsx` ou crie novas páginas específicas de plataforma.

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
Página de Resultados de Busca	`src/pages/SearchResults.tsx`, `src/utils/fuzzySearch.ts`, `src/components/SearchSuggestions.tsx`	`SearchResults.tsx` exibe os produtos encontrados. `fuzzySearch.ts` contém a lógica de busca aproximada. Para ajustar a lógica de busca, a IA deve modificar `fuzzySearch.ts`. Para alterar a exibição dos resultados, edite `SearchResults.tsx`.
Página de Produto	`src/pages/ProductPage.tsx`, `src/components/ProductPage/ProductImageGallery.tsx`, `src/components/ProductPage/ProductInfo.tsx`, `src/components/ProductPage/ProductPricing.tsx`, `src/components/ProductPage/ProductOptions.tsx`, `src/components/ProductPage/ProductActions.tsx`, `src/components/ProductPage/ProductDescription.tsx`, `src/components/ProductPage/RelatedProducts.tsx`, `src/hooks/useProducts.ts`	`ProductPage.tsx` é a página principal de detalhes do produto. Ela é composta por vários subcomponentes que exibem diferentes aspectos do produto. `useProducts.ts` é usado para buscar os dados do produto. Para alterar o layout, adicionar novas seções (ex: avaliações), ou modificar a forma como as informações do produto são exibidas, a IA deve intervir em `ProductPage.tsx` e seus subcomponentes.
Modal de Produto	`src/components/ProductModal/ProductModal.tsx`, `src/components/ProductModal/ProductModalContent.tsx`, `src/components/`	`ProductModal.tsx` é o componente principal do modal. `useProductModal.ts` gerencia o estado do modal. Para alterar o conteúdo ou o comportamento do

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
	`ProductModal/` `useProductModal.ts`	modal de visualização rápida do produto, edite esses arquivos.

6.2.2. Funcionalidades de Carrinho e Compra

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
Contexto do Carrinho	`src/contexts/` `CartContext.tsx`, `src/` `hooks/useNewCart.ts`, `src/types/cart.ts`	`CartContext.tsx` é o provedor de contexto que disponibiliza o estado e as funções do carrinho para toda a aplicação. `useNewCart.ts` contém a lógica principal do carrinho (adicionar, remover, atualizar quantidade, calcular total, enviar para WhatsApp). `cart.ts` define as interfaces de tipagem para `CartItem` e `CartState`. Para modificar a lógica do carrinho, como regras de preço, validações de estoque ou o processo de adição/ remoção, a IA deve focar em `useNewCart.ts`. Para adicionar novas propriedades aos itens do carrinho, `cart.ts` deve ser atualizado.
Ícone do Carrinho no Cabeçalho	`src/components/` `GlobalCart/` `GlobalCartIcon.tsx`	Exibe o número de itens no carrinho. Para alterar a aparência do ícone ou a forma como a contagem é exibida, edite este componente.
Dropdown/ Modal do Carrinho	`src/components/` `GlobalCart/` `GlobalCartDropdown.tsx`,	`GlobalCartDropdown.tsx` é o componente que exibe um resumo do carrinho (geralmente

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
	`src/components/Cart.tsx`	um dropdown ou sidebar) quando o ícone do carrinho é clicado. `Cart.tsx` pode ser um modal ou uma página dedicada ao carrinho. Para alterar o layout do resumo do carrinho, as informações exibidas ou as ações disponíveis (ex: botão de checkout), modifique esses componentes.
Adicionar ao Carrinho (Botão)	`src/components/ProductPage/ProductActions.tsx`, `src/components/ProductCard/ProductCardActions.tsx`	Estes componentes contêm a lógica e o botão para adicionar um produto ao carrinho. Eles chamam a função `addToCart` do `useCart` context. Para alterar o texto do botão, a lógica de validação antes de adicionar ou o feedback visual após a adição, edite esses componentes.
Envio de Pedido via WhatsApp	`src/utils/whatsapp.ts`, `src/hooks/useNewCart.ts`	`whatsapp.ts` contém a função utilitária para gerar o link do WhatsApp. `useNewCart.ts` utiliza essa função para construir a mensagem com os detalhes do pedido. Para alterar o número de telefone de destino, o formato da mensagem ou incluir informações adicionais no pedido, a IA deve modificar `whatsapp.ts` e/ou a função `sendToWhatsApp` em `useNewCart.ts`.

6.2.3. Funcionalidades de Autenticação e Usuário

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
Contexto de Autenticação	`src/hooks/useAuth.tsx`, `src/integrations/supabase/client.ts`	`useAuth.tsx` gerencia o estado de autenticação do usuário (logado/deslogado, isAdmin), e as funções de `signIn`, `signUp`, `signOut`. Ele interage diretamente com o cliente Supabase. Para adicionar novos campos ao cadastro, alterar a lógica de permissões ou integrar com outros provedores de autenticação, a IA deve modificar `useAuth.tsx` e as configurações do Supabase.
Modal de Autenticação	`src/components/Auth/AuthModal.tsx`	Um modal que permite aos usuários fazer login ou se cadastrar. Ele utiliza as funções de `useAuth.tsx`. Para alterar o layout do modal, adicionar campos de formulário ou personalizar mensagens de erro, edite `AuthModal.tsx`.
Página de Login (Standalone)	`src/components/Auth/LoginPage.tsx`	Uma página dedicada para login, usada quando o acesso é negado ou o usuário não está logado. Funcionalidade similar ao `AuthModal`, mas como uma página completa.
Rota Protegida (Admin)	`src/App.tsx` (`ProtectedAdminRoute`)	A `ProtectedAdminRoute` em `App.tsx` verifica se o usuário está logado e se possui a função de administrador antes de permitir o acesso à rota `/admin`. Para alterar as regras de acesso ou redirecionamento, modifique esta

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alteração
		lógica em `App.tsx`. **Importante:** A validação de permissões deve ser sempre replicada no backend (Supabase RLS) para garantir a segurança.

6.2.4. Painel Administrativo: O Centro de Controle da Loja

O painel administrativo é a área mais crítica para o gerenciamento do conteúdo e das operações da loja. Ele é acessível apenas por usuários com a função de `admin` e é composto por uma série de "gerenciadores" (Managers), cada um responsável por uma área específica. A IA deve ter um conhecimento aprofundado de cada um desses gerenciadores para realizar tarefas de manutenção e atualização de conteúdo.

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alter
Painel Administrativo Principal	`src/components/Admin/AdminPanel.tsx`, `src/components/Admin/AdminLayout.tsx`	`AdminPanel.tsx` é o comp principal que orquestra os d gerenciadores (abas). `Admi` fornece o layout geral do pai cabeçalho). Para adicionar n painel, reorganizar as abas o layout geral, a IA deve modif `AdminPanel.tsx` e `Admin`
Gerenciamento de Produtos	`src/components/Admin/ProductManager.tsx`, `src/components/Admin/ProductManager/ProductForm.tsx`, `src/components/Admin/ProductManager/ProductList.tsx`, `src/components/Admin/ProductManager/TagSelector.tsx`, `src/hooks/useProducts.ts`, `src/hooks/useTags.ts`	Permite criar, editar e exclui `ProductForm.tsx` é o forn adicionar/editar produtos (n preço, imagens, categorias, t cores, estoque). `ProductLi` lista de produtos existentes. `TagSelector.tsx` é um co selecionar tags. `useProduc` `useTags.ts` são os hooks com o Supabase para gerenc tags. **Detalhes para a IA:** Par novos campos a um produto

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alter
1. Atualizar a interface `Product` em `src/hooks/useProducts/types.ts`.		
2. Adicionar o campo correspondente no `ProductForm.tsx`.		
3. Modificar as funções de `createProduct` e `updateProduct` em `src/hooks/useProducts/productApi.ts` para lidar com o novo campo.		
4. Garantir que o esquema do banco de dados no Supabase (tabela `products`) seja atualizado para incluir a nova coluna.		
Para alterar a lógica de exibição da lista de produtos, edite `ProductList.tsx`. Para gerenciar tags, `TagSelector.tsx` e `useTags.ts` são os pontos de intervenção.		

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alter
Gerenciamento de Banners	`src/components/Admin/BannerManager.tsx`, `src/hooks/useBanners.ts`, `src/components/Admin/ImageUploadInput.tsx`	Permite gerenciar os banner página inicial. `BannerMana` provavelmente contém um f adicionar/editar banners (im ordem) e uma lista de banne `useBanners.ts` é o hook c com o Supabase para gerenc dos banners. `ImageUpload` usado para fazer upload das banners para o Supabase Sto **para a IA:** Para adicionar um banner ou alterar as proprie banner, a IA deve seguir um ao gerenciamento de produt tipagens, modificar formulár de API, e ajustar o esquema dados.
Gerenciamento de Seções Especiais	`src/components/Admin/SpecialSectionManager.tsx`, `src/components/Admin/SpecialSectionManager/SpecialSectionForm.tsx`, `src/components/Admin/SpecialSectionManager/SpecialSectionList.tsx`, `src/components/Admin/SpecialSectionManager/SpecialSectionElementManager.tsx`, `src/hooks/useSpecialSections.ts`, `src/hooks/useSpecialSectionElements.ts`, `src/types/specialSections.ts`	Permite criar e gerenciar seç personalizadas na página ini "Ofertas do Dia", "Lançamen seção especial pode conter r "elementos" (produtos, ban `SpecialSectionForm.tsx` `SpecialSectionList.tsx` seções em si. `SpecialSectionElementM` gerencia os elementos dentr seção. `useSpecialSectio` `useSpecialSectionElem` hooks para interação com o `specialSections.ts` def **Detalhes para a IA:** Esta é u complexa e dinâmica. A IA p entender a relação entre `Sp` e `SpecialSectionElemen` manipular o conteúdo dessa

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alter
		adicionar novos tipos de ele seção especial, a IA precisará tipagens e a lógica de render `SpecialSectionElement`
Gerenciamento de Links Rápidos	`src/components/Admin/QuickLinkManager.tsx`, `src/hooks/useQuickLinks.ts`	Permite configurar os links r no cabeçalho ou em outras á `QuickLinkManager.tsx` p contém um formulário para links (texto, URL, ícone) e um existentes. `useQuickLinks` para interação com o Supab **para a IA:** Para adicionar nov propriedades a um link rápid forma como eles são exibido modificar `QuickLinkManag` `useQuickLinks.ts`.
Gerenciamento de Páginas / Layout da Homepage	`src/components/Admin/PageManager.tsx`, `src/components/Admin/PageManager/PageForm.tsx`, `src/components/Admin/PageManager/PageList.tsx`, `src/pages/Admin/HomepageLayoutManager.tsx`, `src/hooks/usePages.ts`, `src/hooks/useHomepageLayout.ts`	Esta funcionalidade é crucia o layout dinâmico da página `HomepageLayoutManager` de `src/pages/Admin/`) é que permite ao administrad ordem e visibilidade das seç inicial. `useHomepageLayou` essa ordem. `PageManager.` subcomponentes podem ser páginas estáticas ou a estrut **Detalhes para a IA:** Para alte de seções podem ser adicion inicial ou como elas são orde precisará entender a lógica e `useHomepageLayout.ts` e `HomepageLayoutManager` fundamental para a flexibili do site.

Funcionalidade / Elemento UI	Componentes / Arquivos Chave	Descrição e Pontos de Alter
Gerenciamento de Tags	`src/components/Admin/TagManager.tsx`, `src/hooks/useTags.ts`	Permite criar, editar e exclui podem ser associadas a proc entidades. `TagManager.ts` interface para essas operaçõ `useTags.ts` é o hook que Supabase para gerenciar as **para a IA:** Para adicionar nov propriedades a uma tag ou a de sua aplicação, a IA deve m `TagManager.tsx` e `useTa`
Gerenciamento de Serviços	`src/components/Admin/ServiceCardManager.tsx`, `src/hooks/useServiceCards.ts`	Permite gerenciar os cartões exibidos no site (ex: "Atendir Personalizado", "Suporte Téc `ServiceCardManager.tsx` interface para essas operaçõ `useServiceCards.ts` é o interage com o Supabase. **De IA:** Similar aos outros gerenc pode adicionar novos campo de serviço ou alterar sua lóg
Gerenciamento de Assinaturas de Usuários	`src/components/Admin/UserSubscriptionManagement.tsx`, `src/components/Admin/UserSubscriptionManager.tsx`, `src/hooks/useSubscriptions.ts`	Indica um sistema para gere assinaturas ou newsletters d `UserSubscriptionManage` `UserSubscriptionManage` (possivelmente um wrapper primeiro) gerenciam a interf `useSubscriptions.ts` é interação com o Supabase. **D IA:** Para alterar o tipo de assi adicionar campos de perfil d integrar com um serviço de e marketing, a IA deve modific componentes e hooks.

7. Análise de Funcionalidades e Interface: Uma Perspectiva do Usuário

Esta seção detalha as funcionalidades do site a partir da perspectiva do usuário final, descrevendo o fluxo de interação e a experiência geral. A IA deve ser capaz de simular e entender esses fluxos para realizar testes e propor melhorias.

7.1. Fluxo de Navegação Principal

1. **Acesso à Página Inicial (`/`):** Ao acessar o site, o usuário é recebido pela `Index.tsx`. Esta página exibe:
    - Um **banner promocional** no topo (`PromotionalBanner.tsx`).
    - Um **carrossel de banners** (`HeroBannerCarousel.tsx`) com imagens e links rotativos.
    - **Links rápidos** (`HeroQuickLinks.tsx`) para seções importantes ou categorias populares.
    - **Seções de produtos dinâmicas** (`SectionRenderer.tsx` , `SpecialSectionRenderer.tsx`), que podem incluir produtos em destaque, lançamentos, ofertas, etc. A ordem e o conteúdo dessas seções são configuráveis via painel administrativo.
2. **Navegação por Categorias/Plataformas:**
    - O usuário pode clicar em links de categorias no cabeçalho (`DesktopNavigation.tsx` , `MobileCategoriesMenu.tsx`) ou em links rápidos na página inicial.
    - Isso o leva a uma `CategoryPage.tsx` genérica ou a uma página específica de plataforma (`XboxPage.tsx` , `PlayStationPage.tsx` , `NintendoPage.tsx`), exibindo produtos filtrados por essa categoria/plataforma.
3. **Busca de Produtos:**
    - O usuário pode usar a barra de busca no cabeçalho (`DesktopSearchBar.tsx` , `MobileSearchBar.tsx`).
    - Ao digitar, sugestões de busca podem aparecer (`SearchSuggestions.tsx`).
    - Ao submeter a busca, o usuário é redirecionado para a `SearchResults.tsx` , que exibe os produtos que correspondem à consulta, utilizando a lógica de `fuzzySearch.ts` .

4. **Visualização de Detalhes do Produto:**
    - Ao clicar em um produto em qualquer listagem (página inicial, categoria, busca), o usuário é levado à `ProductPage.tsx` ou um `ProductModal.tsx` é aberto.
    - Na `ProductPage.tsx`, o usuário vê imagens do produto (`ProductImageGallery.tsx`), informações (`ProductInfo.tsx`), preço (`ProductPricing.tsx`), opções de seleção (tamanho, cor, condição - `ProductOptions.tsx`), descrição (`ProductDescription.tsx`) e produtos relacionados (`RelatedProducts.tsx`).

7.2. Fluxo de Compra (Adicionar ao Carrinho e Finalizar via WhatsApp)

1. **Adicionar Produto ao Carrinho:**
    - Na `ProductPage.tsx` ou em um `ProductCard.tsx`, o usuário clica no botão "Adicionar ao Carrinho" (`ProductActions.tsx`, `ProductCardActions.tsx`).
    - A lógica em `useNewCart.ts` é acionada, adicionando o produto ao estado do carrinho (`CartContext.tsx`).
    - Uma notificação (toast) pode aparecer confirmando a adição.
    - O ícone do carrinho no cabeçalho (`GlobalCartIcon.tsx`) é atualizado com a nova contagem de itens.
2. **Visualizar e Gerenciar Carrinho:**
    - O usuário clica no ícone do carrinho no cabeçalho, abrindo o `GlobalCartDropdown.tsx` ou o `Cart.tsx` (modal/sidebar).
    - No carrinho, o usuário pode:
        - Ver a lista de itens, suas quantidades e preços.
        - Aumentar ou diminuir a quantidade de um item (`updateQuantity` em `useNewCart.ts`).
        - Remover um item (`removeFromCart` em `useNewCart.ts`).
        - Ver o total do pedido (`getCartTotal` em `useNewCart.ts`).
3. **Finalizar Pedido via WhatsApp:**
    - No carrinho, o usuário clica em um botão "Finalizar Pedido" ou "Enviar para WhatsApp".
    - A função `sendToWhatsApp` em `useNewCart.ts` é chamada, que por sua vez utiliza `whatsapp.ts`.

    - Um link do WhatsApp é gerado com uma mensagem pré-preenchida contendo os detalhes do pedido.
    - O navegador abre uma nova aba ou redireciona para o WhatsApp (web ou aplicativo), permitindo que o usuário envie a mensagem para a loja.

7.3. Fluxo de Autenticação

1. **Acesso ao Login/Cadastro:**
    - O usuário clica no botão "Login" ou "Minha Conta" no cabeçalho (`HeaderActions.tsx`).
    - O `AuthModal.tsx` é exibido.
2. **Login:**
    - O usuário insere email e senha no formulário do `AuthModal.tsx` ou `LoginPage.tsx`.
    - A função `signIn` de `useAuth.tsx` é chamada, interagindo com o Supabase Auth.
    - Em caso de sucesso, o usuário é logado e o modal é fechado. O estado `user` em `useAuth.tsx` é atualizado.
    - Em caso de erro (credenciais inválidas), uma mensagem de erro é exibida (toast).
3. **Cadastro:**
    - No `AuthModal.tsx`, o usuário pode alternar para a aba de cadastro.
    - O usuário insere email, senha e, possivelmente, nome.
    - A função `signUp` de `useAuth.tsx` é chamada, interagindo com o Supabase Auth.
    - Em caso de sucesso, a conta é criada e o usuário pode ser automaticamente logado ou instruído a fazer login.
4. **Logout:**
    - Um botão de "Logout" (geralmente visível quando o usuário está logado) chama a função `signOut` de `useAuth.tsx`.
    - O usuário é deslogado e o estado `user` é limpo.

7.4. Interface do Usuário (UI) e Experiência do Usuário (UX)

- **Design Responsivo:** A utilização de Tailwind CSS e Shadcn UI garante que o site se adapte fluidamente a diferentes tamanhos de tela, proporcionando uma boa

experiência tanto em desktops quanto em dispositivos móveis. A IA deve verificar a renderização em diferentes viewports durante os testes.
- **Componentes Modulares:** A UI é construída a partir de componentes pequenos e reutilizáveis, o que facilita a consistência visual e a manutenção. Por exemplo, `ProductCard.tsx` é usado em várias listagens de produtos.
- **Feedback Visual:** O uso de `sonner` e `use-toast` para notificações fornece feedback imediato ao usuário sobre ações realizadas (ex: produto adicionado, login bem-sucedido, erro). A IA deve garantir que essas mensagens sejam claras e úteis.
- **Estados de Carregamento e Erro:** Componentes como `LoadingState.tsx` e `ErrorState.tsx` (em `src/components/HomePage/`) são usados para informar o usuário sobre o status da aplicação. A IA deve garantir que todos os estados assíncronos tenham um tratamento visual adequado.
- **Acessibilidade:** Os componentes Shadcn UI são construídos com foco em acessibilidade, seguindo as diretrizes ARIA. A IA deve estar ciente da importância de manter a acessibilidade ao fazer modificações, especialmente em elementos interativos.

8. Painel Administrativo: Detalhamento Completo

O painel administrativo é a espinha dorsal do gerenciamento de conteúdo e operações do "UTI Gamer Shop Brasa". Ele é uma aplicação React separada, protegida por autenticação de administrador, e oferece uma interface para manipular os dados armazenados no Supabase. A IA deve dominar cada seção deste painel para realizar tarefas de gerenciamento de conteúdo e depuração.

8.1. Acesso e Estrutura Geral

- **Rota Protegida:** O painel é acessível via `/admin` e é protegido pela `ProtectedAdminRoute` em `src/App.tsx`, que verifica se o usuário está logado e se possui a função `admin` (definida no perfil do usuário no Supabase).
- **Componente Principal:** `src/components/Admin/AdminPanel.tsx` é o componente que agrega todas as funcionalidades do painel. Ele provavelmente utiliza um sistema de abas ou navegação lateral para alternar entre os diferentes gerenciadores.
- **Layout:** `src/components/Admin/AdminLayout.tsx` define o layout comum para todas as páginas do painel, incluindo a barra lateral de navegação e o cabeçalho.

8.2. Gerenciadores de Conteúdo e Dados (Módulos do Painel)

Cada seção do painel administrativo é um "gerenciador" dedicado a um tipo específico de dado ou funcionalidade. A IA deve entender o propósito de cada um e como eles interagem com o Supabase.

8.2.1. Gerenciamento de Produtos (`ProductManager`)

- **Componentes:** `src/components/Admin/ProductManager.tsx` , `src/components/Admin/ProductManager/ProductForm.tsx` , `src/components/Admin/ProductManager/ProductList.tsx` , `src/components/Admin/ProductManager/TagSelector.tsx` .
- **Hooks:** `src/hooks/useProducts.ts` , `src/hooks/useTags.ts` .
- **Funcionalidade:** Permite o CRUD (Create, Read, Update, Delete) completo de produtos. Esta é a seção mais importante para manter o catálogo da loja atualizado.
  - **Criação/Edição de Produtos:** O `ProductForm.tsx` é um formulário complexo que permite ao administrador inserir ou modificar todos os detalhes de um produto:
    - **Nome, Descrição:** Campos de texto para informações básicas.
    - **Preço, Preço Promocional:** Valores numéricos para precificação.
    - **Imagens:** Através de `ImageUploadInput.tsx` , permite fazer upload de múltiplas imagens para o Supabase Storage e associá-las ao produto. A IA deve garantir que as imagens sejam otimizadas para web.
    - **Categorias:** Seleção de categorias às quais o produto pertence. Isso pode ser um dropdown ou checkboxes, interagindo com uma tabela de categorias no Supabase.
    - **Tags:** Seleção de tags (`TagSelector.tsx`) para o produto, que ajudam na organização e busca. As tags são gerenciadas separadamente via `TagManager` .
    - **Tamanhos e Cores:** Campos para definir variações de produtos (ex: P, M, G; Vermelho, Azul). Isso pode ser implementado como um array de strings ou objetos no banco de dados.
    - **Estoque:** Quantidade disponível do produto. A IA deve estar ciente de que a lógica de estoque pode ser mais complexa (ex: estoque por variação).
  - **Listagem de Produtos:** `ProductList.tsx` exibe todos os produtos existentes em uma tabela ou lista, com opções para editar ou excluir cada um. A IA pode precisar implementar filtros ou paginação aqui para grandes catálogos.

- **Intervenção da IA:**
  - **Adicionar Novos Atributos de Produto:** Se for necessário adicionar um novo campo (ex: "peso", "dimensões", "fabricante") a todos os produtos, a IA deve:
    1. Atualizar a interface `Product` em `src/hooks/useProducts/types.ts` para incluir o novo campo.
    2. Modificar `ProductForm.tsx` para adicionar o campo de entrada correspondente.
    3. Ajustar as funções `createProduct` e `updateProduct` em `src/hooks/useProducts/productApi.ts` para enviar/receber o novo dado do Supabase.
    4. **Crucial:** Atualizar o esquema da tabela `products` no banco de dados PostgreSQL do Supabase para incluir a nova coluna. Isso geralmente é feito através da interface do Supabase Studio ou via migrações SQL.
  - **Alterar Lógica de Preço/Estoque:** Qualquer alteração na forma como os preços são calculados ou o estoque é gerenciado deve ser feita em `ProductForm.tsx` (para entrada de dados) e em `src/hooks/useProducts/productApi.ts` (para lógica de persistência).

8.2.2. Gerenciamento de Banners (`BannerManager`)

- **Componentes:** `src/components/Admin/BannerManager.tsx`, `src/components/Admin/ImageUploadInput.tsx`.
- **Hooks:** `src/hooks/useBanners.ts`.
- **Funcionalidade:** Permite que o administrador adicione, edite e remova banners promocionais que aparecem em seções específicas do site (principalmente na página inicial). Cada banner geralmente possui uma imagem, um link de destino e uma ordem de exibição.
  - **Upload de Imagem:** `ImageUploadInput.tsx` é um componente reutilizável para upload de imagens, que envia o arquivo para o Supabase Storage e retorna a URL pública.
- **Intervenção da IA:**
  - **Adicionar Propriedades ao Banner:** Se for necessário adicionar um título, descrição ou data de expiração a um banner, a IA deve seguir o mesmo processo de atualização de tipagens, formulários e funções de API, e ajustar o esquema da tabela `banners` no Supabase.
  - **Alterar Lógica de Exibição:** A lógica de como os banners são carregados e exibidos no frontend (ex: `HeroBannerCarousel.tsx`) é controlada por `useBanners.ts`. Para implementar um carrossel diferente ou regras de exibição (ex: banners por período), a IA deve modificar esses arquivos.

8.2.3. Gerenciamento de Seções Especiais (`SpecialSectionManager`)

- **Componentes:** `src/components/Admin/SpecialSectionManager.tsx`, `src/components/Admin/SpecialSectionManager/SpecialSectionForm.tsx`, `src/components/Admin/SpecialSectionManager/SpecialSectionList.tsx`, `src/components/Admin/SpecialSectionManager/SpecialSectionElementManager.tsx`.
- **Hooks:** `src/hooks/useSpecialSections.ts`, `src/hooks/useSpecialSectionElements.ts`.
- **Tipagens:** `src/types/specialSections.ts`.
- **Funcionalidade:** Esta é uma funcionalidade avançada que permite ao administrador criar seções de conteúdo altamente personalizáveis na página inicial. Uma "Seção Especial" pode ser, por exemplo, "Ofertas de Fim de Semana" ou "Lançamentos de RPG". Dentro de cada seção, o administrador pode adicionar "Elementos de Seção Especial", que podem ser produtos específicos, banners, ou até mesmo blocos de texto/HTML.
  - **Gerenciamento de Seções:** `SpecialSectionForm.tsx` e `SpecialSectionList.tsx` permitem criar e listar as seções especiais, definindo seu título, tipo e visibilidade.
  - **Gerenciamento de Elementos:** `SpecialSectionElementManager.tsx` é onde o administrador adiciona e organiza os elementos dentro de uma seção. Isso pode envolver a seleção de produtos existentes, upload de imagens para banners específicos da seção, ou inserção de texto formatado.
- **Intervenção da IA:**
  - **Adicionar Novos Tipos de Elementos:** Se o site precisar de um novo tipo de conteúdo dentro de uma seção especial (ex: um vídeo incorporado, um formulário de inscrição), a IA deve:
    1. Atualizar a tipagem em `src/types/specialSections.ts` para incluir o novo tipo de elemento.
    2. Modificar `SpecialSectionElementForm.tsx` para adicionar os campos de entrada necessários para o novo tipo.
    3. Ajustar a lógica de renderização em `src/components/SpecialSections/SpecialSectionElementRenderer.tsx` para exibir corretamente o novo tipo de elemento no frontend.
    4. Atualizar as funções de API em `useSpecialSectionElements.ts` para persistir os novos dados no Supabase.
  - **Alterar Lógica de Ordenação/Exibição:** A forma como os elementos são ordenados dentro de uma seção ou como a seção inteira é exibida na página inicial é controlada por `useSpecialSections.ts` e

`useSpecialSectionElements.ts`. A IA pode precisar ajustar a lógica de consulta ou renderização para implementar novas regras.

8.2.4. Gerenciamento de Links Rápidos (`QuickLinkManager`)

- **Componentes:** `src/components/Admin/QuickLinkManager.tsx`.
- **Hooks:** `src/hooks/useQuickLinks.ts`.
- **Funcionalidade:** Permite ao administrador definir e gerenciar os links rápidos que aparecem em áreas proeminentes do site (ex: abaixo do banner principal na página inicial). Cada link rápido geralmente tem um texto, um URL e, possivelmente, um ícone.
- **Intervenção da IA:**
  - **Adicionar Propriedades ao Link:** Se for necessário adicionar uma descrição ou uma cor de destaque a um link rápido, a IA deve seguir o processo de atualização de tipagens, formulários e funções de API, e ajustar o esquema da tabela `quick_links` no Supabase.
  - **Alterar Local de Exibição:** A lógica de onde os links rápidos são exibidos no frontend é controlada por `useQuickLinks.ts` e o componente que os renderiza (ex: `HeroQuickLinks.tsx`).

8.2.5. Gerenciamento de Páginas / Layout da Homepage (`PageManager`, `HomepageLayoutManager`)

- **Componentes:** `src/components/Admin/PageManager.tsx`, `src/components/Admin/PageManager/PageForm.tsx`, `src/components/Admin/PageManager/PageList.tsx`, `src/pages/Admin/HomepageLayoutManager.tsx`.
- **Hooks:** `src/hooks/usePages.ts`, `src/hooks/useHomepageLayout.ts`.
- **Funcionalidade:** Esta é uma das funcionalidades mais poderosas do painel, pois permite ao administrador controlar a estrutura e o conteúdo da página inicial de forma dinâmica. O `HomepageLayoutManager.tsx` permite ao administrador arrastar e soltar seções (banners, seções de produtos, seções especiais) para definir a ordem em que aparecem na página inicial e sua visibilidade.
  - **`useHomepageLayout.ts`:** Este hook é crucial, pois ele gerencia a ordem e a visibilidade das seções da página inicial, provavelmente armazenando essa configuração em uma tabela no Supabase.

- **Intervenção da IA:**
  - **Adicionar Novos Tipos de Seções ao Layout:** Se o site precisar de um novo tipo de seção que possa ser adicionada e ordenada na página inicial (ex: uma seção de depoimentos, um feed de notícias), a IA deve:
    1. Atualizar as tipagens relacionadas ao layout da homepage (se existirem) para incluir o novo tipo de seção.
    2. Modificar `HomepageLayoutManager.tsx` para permitir que o administrador adicione e configure o novo tipo de seção.
    3. Ajustar a lógica de renderização em `src/pages/Index.tsx` e `src/components/HomePage/SectionRenderer.tsx` para exibir corretamente o novo tipo de seção.
    4. Atualizar as funções de API em `useHomepageLayout.ts` para persistir a configuração do novo tipo de seção no Supabase.
  - **Alterar Lógica de Arrastar e Soltar:** A funcionalidade de arrastar e soltar provavelmente utiliza bibliotecas como `@dnd-kit`. Para alterar o comportamento ou a interface dessa funcionalidade, a IA precisará modificar os componentes relacionados ao `HomepageLayoutManager.tsx`.

8.2.6. Gerenciamento de Tags (`TagManager`)

- **Componentes:** `src/components/Admin/TagManager.tsx`.
- **Hooks:** `src/hooks/useTags.ts`.
- **Funcionalidade:** Permite ao administrador criar, editar e excluir tags. As tags são usadas para categorizar produtos de forma mais granular do que as categorias principais, facilitando a busca e a filtragem.
- **Intervenção da IA:**
  - **Adicionar Propriedades à Tag:** Se for necessário adicionar uma cor ou uma descrição a uma tag, a IA deve seguir o processo de atualização de tipagens, formulários e funções de API, e ajustar o esquema da tabela `tags` no Supabase.
  - **Integrar Tags em Outras Áreas:** Se as tags precisarem ser usadas em outras partes do site (ex: para filtrar notícias, artigos de blog), a IA precisará estender a lógica de `useTags.ts` e integrar o `TagSelector.tsx` em outros formulários.

8.2.7. Gerenciamento de Serviços (`ServiceCardManager`)

- **Componentes:** `src/components/Admin/ServiceCardManager.tsx`.
- **Hooks:** `src/hooks/useServiceCards.ts`.

- **Funcionalidade:** Permite ao administrador gerenciar os cartões de serviço exibidos no site. Estes podem ser seções como "Por que nos escolher?", "Atendimento Personalizado", etc., que destacam os diferenciais da loja.
- **Intervenção da IA:**
  - **Adicionar Novos Tipos de Serviço:** Se for necessário adicionar um novo tipo de serviço com campos específicos (ex: um serviço de instalação com campos para duração e preço por hora), a IA deve seguir o processo de atualização de tipagens, formulários e funções de API, e ajustar o esquema da tabela `service_cards` no Supabase.
  - **Alterar Layout de Exibição:** A forma como os cartões de serviço são exibidos no frontend (ex: `src/components/ServiceCards/`) é controlada por `useServiceCards.ts` e os componentes de renderização. A IA pode modificar esses componentes para alterar o layout ou adicionar animações.

8.2.8. Gerenciamento de Assinaturas de Usuários (`UserSubscriptionManagement`)

- **Componentes:** `src/components/Admin/UserSubscriptionManagement.tsx`, `src/components/Admin/UserSubscriptionManager.tsx`.
- **Hooks:** `src/hooks/useSubscriptions.ts`.
- **Funcionalidade:** Esta seção do painel administrativo é dedicada ao gerenciamento de assinaturas de usuários, como newsletters ou programas de fidelidade. Permite ao administrador visualizar a lista de assinantes, adicionar/remover assinaturas manualmente e, possivelmente, exportar dados.
- **Intervenção da IA:**
  - **Adicionar Campos de Assinatura:** Se for necessário coletar mais informações dos assinantes (ex: preferências de categoria, data de aniversário), a IA deve atualizar as tipagens, formulários e funções de API, e ajustar o esquema da tabela `subscriptions` no Supabase.
  - **Integrar com Ferramentas de Marketing:** Para enviar emails em massa ou gerenciar campanhas, a IA pode precisar integrar este módulo com uma plataforma de email marketing externa (ex: Mailchimp, SendGrid), o que envolveria a criação de novas funções de API ou a utilização de SDKs de terceiros.

8.3. Componentes Reutilizáveis no Admin

- **`ImageUploadInput.tsx`:** Este componente é fundamental para qualquer gerenciador que lide com imagens (Produtos, Banners, Seções Especiais). Ele encapsula a lógica de seleção de arquivos, upload para o Supabase Storage e

exibição de pré-visualização. A IA deve entender que, ao precisar de upload de imagens em um novo formulário, este componente deve ser reutilizado.
- **Componentes Shadcn UI:** O painel administrativo faz uso extensivo dos componentes Shadcn UI (`src/components/ui/`) para formulários, tabelas, modais, etc. A IA deve usar esses componentes para manter a consistência visual e a acessibilidade ao adicionar novas funcionalidades ao painel.

9. Identificação de Bugs e Problemas Potenciais: Uma Análise Crítica

Embora o código-fonte do "UTI Gamer Shop Brasa" demonstre uma estrutura sólida e o uso de boas práticas, a análise estática pode revelar potenciais pontos de falha, áreas para otimização ou cenários de uso que podem levar a bugs. É crucial que a IA, ao intervir no projeto, esteja ciente desses pontos para realizar testes direcionados e implementar soluções robustas.

9.1. Tratamento de Erros e Feedback ao Usuário

- **Mensagens de Erro Genéricas:** Em alguns pontos, o tratamento de erros pode ser genérico. Por exemplo, se uma chamada de API falhar, a mensagem de erro exibida ao usuário pode ser apenas "Ocorreu um erro" em vez de uma mensagem mais específica e útil (ex: "Não foi possível carregar os produtos. Verifique sua conexão com a internet.").
    - **Potencial Bug:** Falta de clareza nas mensagens de erro pode frustrar o usuário e dificultar a depuração. Em `useAuth.tsx`, as mensagens de erro do `toast` são baseadas em `error.message`, o que é um bom começo, mas pode ser aprimorado para cenários específicos.
    - **Recomendação para IA:** A IA deve revisar todas as chamadas de API e operações assíncronas para garantir que as mensagens de erro sejam contextuais, amigáveis e, quando possível, ofereçam uma solução ou instrução ao usuário. Implementar um sistema de logging mais detalhado no frontend pode ajudar a capturar esses erros para análise.
- **Estados de Carregamento Incompletos:** Embora existam `LoadingState.tsx` e `ErrorState.tsx`, pode haver cenários onde partes da UI permanecem em um estado de carregamento indefinido ou não exibem um feedback visual adequado durante operações assíncronas (ex: adicionar ao carrinho, enviar formulário).
    - **Potencial Bug:** O usuário pode ficar confuso sobre o que está acontecendo se não houver feedback visual claro durante operações demoradas.
    - **Recomendação para IA:** A IA deve mapear todas as operações assíncronas e garantir que cada uma tenha um estado de carregamento visualmente

distinto (ex: spinners, esqueletos de conteúdo, botões desabilitados com texto "Carregando...") e um tratamento de erro correspondente.

9.2. Validação de Dados e Segurança

- **Validação de Formulários (Frontend vs. Backend):** O uso de `react-hook-form` e `zod` para validação de formulários no frontend é uma excelente prática para melhorar a experiência do usuário. No entanto, a validação no frontend nunca é suficiente para garantir a integridade e segurança dos dados.
  - **Potencial Bug/Vulnerabilidade:** Se a validação não for replicada e reforçada no backend (Supabase), um usuário mal-intencionado pode contornar a validação do frontend e enviar dados inválidos ou maliciosos diretamente para o banco de dados.
  - **Recomendação para IA:** A IA deve revisar todas as operações de escrita no Supabase (insert, update) e garantir que as regras de validação de esquema e as políticas de Row Level Security (RLS) no Supabase sejam rigorosas. Por exemplo, para a criação de produtos no painel administrativo, o Supabase deve validar se os campos obrigatórios estão presentes, se os tipos de dados estão corretos e se o usuário tem permissão para realizar a operação.
- **Políticas de Row Level Security (RLS) no Supabase:** As políticas de RLS são cruciais para controlar o acesso aos dados no nível da linha do banco de dados. Sem RLS adequadamente configurado, um usuário autenticado pode potencialmente ler, atualizar ou excluir dados aos quais não deveria ter acesso.
  - **Potencial Bug/Vulnerabilidade:** A ausência ou configuração incorreta de políticas de RLS pode expor dados sensíveis ou permitir operações não autorizadas.
  - **Recomendação para IA:** A IA deve realizar uma auditoria completa das políticas de RLS em todas as tabelas do Supabase utilizadas pelo site. Para cada tabela, deve-se definir claramente quem pode ler, inserir, atualizar e excluir dados, com base nos papéis do usuário (anon, authenticated, admin) e nas condições dos dados (ex: um usuário só pode ver seus próprios pedidos).
- **Exposição de Chaves de API:** O arquivo `src/integrations/supabase/client.ts` contém a `SUPABASE_PUBLISHABLE_KEY` diretamente no código. Embora esta chave seja pública e projetada para ser usada no frontend, a IA deve estar ciente de que chaves secretas (se existirem) nunca devem ser expostas no código do cliente. A mensagem de "gerado automaticamente" sugere que isso é gerenciado, mas a IA deve verificar a fonte.
  - **Recomendação para IA:** Confirmar que nenhuma chave sensível está hardcoded no frontend. Se houver necessidade de chaves secretas, elas

devem ser acessadas via um backend seguro (Edge Functions do Supabase, por exemplo) ou injetadas de forma segura durante o build.

9.3. Otimização de Desempenho

- **Carregamento de Imagens:** Em um e-commerce, imagens de alta resolução não otimizadas podem impactar significativamente o tempo de carregamento da página. Embora o `ImageUploadInput.tsx` lide com o upload, não há garantia explícita de otimização automática.
  - **Potencial Bug:** Imagens grandes podem levar a um site lento, prejudicando a experiência do usuário e o SEO.
  - **Recomendação para IA:** A IA deve investigar se o Supabase Storage está configurado para otimização de imagens (redimensionamento, compressão) ou se uma CDN (Content Delivery Network) com otimização de imagem está sendo usada. Caso contrário, a IA pode implementar uma solução de otimização de imagem no lado do cliente (antes do upload) ou no lado do servidor (após o upload).
- **Consultas ao Banco de Dados:** Consultas ineficientes ao Supabase podem levar a tempos de resposta lentos, especialmente em páginas com muitos dados (ex: listagens de produtos, painel administrativo).
  - **Potencial Bug:** Consultas sem paginação, filtragem ou índices adequados podem sobrecarregar o banco de dados e o frontend.
  - **Recomendação para IA:** A IA deve revisar as funções de busca de dados (ex: em `useProducts.ts`, `useBanners.ts`) para garantir que elas utilizem paginação, filtros e ordenação quando apropriado. A criação de índices no PostgreSQL do Supabase para colunas frequentemente consultadas também é crucial para o desempenho.
- **Re-renderizações Excessivas no React:** Componentes React podem re-renderizar desnecessariamente, impactando o desempenho da UI. Isso geralmente ocorre quando props ou estados mudam, mas o resultado visual não.
  - **Potencial Bug:** UI lenta ou travando em interações complexas.
  - **Recomendação para IA:** A IA pode usar as React Developer Tools para identificar re-renderizações excessivas. Soluções incluem o uso de `React.memo` para memorizar componentes, `useCallback` e `useMemo` para memorizar funções e valores, e garantir que as dependências dos `useEffect` e `useCallback` estejam corretas.

9.4. Usabilidade e Acessibilidade

- **Navegação por Teclado:** Embora os componentes Shadcn UI sejam acessíveis, a IA deve verificar se toda a navegação do site é totalmente funcional usando apenas o teclado, o que é crucial para usuários com deficiência motora.
    - **Potencial Bug:** Elementos interativos que não são focáveis ou não respondem a eventos de teclado.
    - **Recomendação para IA:** Realizar testes de navegação por teclado em todas as páginas e funcionalidades. Garantir que os estados de foco sejam visíveis e que a ordem de tabulação seja lógica.
- **Contraste de Cores:** O design do site pode ter problemas de contraste de cores que dificultam a leitura para usuários com deficiência visual.
    - **Potencial Bug:** Texto ilegível ou elementos da UI que se misturam ao fundo.
    - **Recomendação para IA:** Utilizar ferramentas de análise de contraste de cores para verificar se o site atende aos padrões de acessibilidade (WCAG). Ajustar as cores no `tailwind.config.ts` ou nos componentes individuais, se necessário.
- **Mensagens de Validação de Formulário:** As mensagens de erro de validação devem ser claras, concisas e associadas ao campo de entrada correspondente para que os usuários saibam exatamente o que precisa ser corrigido.
    - **Potencial Bug:** Mensagens de erro vagas ou que não indicam o campo problemático.
    - **Recomendação para IA:** Revisar todos os formulários e garantir que as mensagens de validação sejam específicas e que os campos inválidos sejam visualmente destacados.

9.5. Manutenibilidade e Escalabilidade

- **Código Duplicado (DRY Principle):** Embora o projeto seja bem estruturado, sempre há a possibilidade de código duplicado, especialmente em projetos grandes e em evolução. Isso dificulta a manutenção e introduz o risco de inconsistências.
    - **Potencial Bug:** Alterar uma funcionalidade em um local e esquecer de alterá-la em outro, levando a comportamentos inconsistentes.
    - **Recomendação para IA:** Realizar auditorias de código para identificar e refatorar trechos duplicados em funções utilitárias, hooks personalizados ou componentes reutilizáveis. Ferramentas de análise estática de código podem auxiliar nisso.
- **Documentação Interna Insuficiente:** Embora este documento sirva como documentação externa, a documentação interna (comentários no código,

READMEs de módulos) é crucial para a compreensão rápida do código por novos desenvolvedores ou IAs.
    - **Potencial Bug:** Dificuldade em entender a lógica complexa ou as decisões de design sem comentários explicativos.
    - **Recomendação para IA:** Adicionar comentários claros e concisos em trechos de código complexos, funções utilitárias e componentes-chave. Manter um `README.md` atualizado em cada diretório principal explicando seu propósito e conteúdo.
- **Testes Automatizados:** A ausência de uma pasta `tests` explícita sugere que o projeto pode ter cobertura de testes limitada ou inexistente. Testes automatizados são essenciais para garantir a qualidade do software e prevenir a introdução de bugs.
    - **Potencial Bug:** Regressões (bugs introduzidos por novas alterações) podem passar despercebidas.
    - **Recomendação para IA:** Implementar uma estratégia de testes automatizados (unitários, de integração, end-to-end) usando frameworks como Jest, React Testing Library, Cypress ou Playwright. Isso permitirá que a IA execute testes automaticamente após cada alteração, garantindo a estabilidade do sistema.

10. Guia de Intervenção para a IA: Como Fazer Alterações no Site

Esta seção é um manual prático para a IA, detalhando o processo passo a passo para realizar modificações no site "UTI Gamer Shop Brasa". Ele abrange desde a configuração do ambiente até a implantação das alterações, com foco na segurança e nas melhores práticas.

10.1. Configuração do Ambiente de Desenvolvimento da IA

Antes de qualquer intervenção, a IA deve garantir que seu ambiente de desenvolvimento esteja configurado corretamente para trabalhar com o projeto.

1. **Clonar o Repositório:**
    - A IA deve clonar o repositório GitHub do "UTI Gamer Shop Brasa" para seu ambiente de trabalho local. O comando `git clone <URL_DO_REPOSITORIO>` é o ponto de partida.

- **Exemplo:** `git clone https://github.com/usuario/uti-gamer-shop-brasa.git`

2. **Instalar Dependências:**
   - Navegar até o diretório raiz do projeto clonado (`cd uti-gamer-shop-brasa`).
   - Instalar as dependências do Node.js/npm listadas no `package.json`. O comando mais comum é `npm install` ou `yarn install`.
   - **Exemplo:** `npm install`

3. **Configurar Variáveis de Ambiente (se aplicável):**
   - Embora o `client.ts` do Supabase contenha as chaves hardcoded (e seja gerado automaticamente), a IA deve verificar se há um arquivo `.env` ou `.env.local` no projeto. Se sim, ele pode conter variáveis de ambiente que precisam ser configuradas para o ambiente de desenvolvimento local (ex: chaves de API para serviços de terceiros, configurações de ambiente).
   - **Ação da IA:** Ler o `README.md` do projeto (se existir) ou procurar por arquivos `.env.example` para identificar as variáveis de ambiente necessárias e configurá-las.

4. **Iniciar o Servidor de Desenvolvimento:**
   - Após a instalação das dependências, a IA pode iniciar o servidor de desenvolvimento para visualizar o site localmente. O comando é `npm run dev` ou `yarn dev`.
   - **Exemplo:** `npm run dev`
   - Isso geralmente iniciará o site em `http://localhost:5173` (ou outra porta). A IA deve navegar para esta URL para interagir com o site.

10.2. Processo de Modificação de Código

Ao realizar alterações no código, a IA deve seguir um processo estruturado para garantir a qualidade, a rastreabilidade e a segurança.

1. **Criar um Novo Branch:**
   - Nunca trabalhe diretamente no branch `main` (ou `master`). Sempre crie um novo branch para cada funcionalidade ou correção de bug.
   - **Comando:** `git checkout -b feature/nome-da-funcionalidade` ou `git checkout -b bugfix/descricao-do-bug`

2. **Identificar o Arquivo/Componente Correto:**

    - Utilize a "Estrutura de Diretórios e Mapeamento de Componentes" (Seção 6) deste documento para localizar o arquivo ou componente relevante para a alteração desejada.
    - **Exemplo:** Para alterar a lógica de adição ao carrinho, a IA deve focar em `src/hooks/useNewCart.ts` e `src/contexts/CartContext.tsx`.

3. **Realizar as Alterações no Código:**

    - Modifique o código conforme a necessidade. Mantenha as alterações focadas e coesas.
    - **Princípios para a IA:**
        - **Princípio da Responsabilidade Única:** Cada componente/função deve ter uma única responsabilidade. Se uma alteração exigir adicionar uma nova responsabilidade, considere criar um novo componente/função.
        - **Princípio DRY (Don't Repeat Yourself):** Evite duplicar código. Se uma lógica for reutilizada em vários lugares, refatore-a para uma função utilitária ou um hook personalizado.
        - **Manter a Tipagem:** Sempre que modificar a estrutura de dados ou as interfaces, atualize as tipagens TypeScript correspondentes (`src/types/`).
        - **Comentários:** Adicione comentários claros e concisos para explicar a lógica complexa, decisões de design ou qualquer parte do código que possa ser difícil de entender no futuro.

4. **Testar as Alterações Localmente:**

    - Após cada conjunto de alterações, a IA deve testar o site localmente para garantir que a funcionalidade modificada funciona como esperado e que nenhuma regressão foi introduzida.
    - **Ação da IA:** Interagir com a UI, simular fluxos de usuário relevantes e verificar o console do navegador para erros.
    - **Testes Automatizados (se implementados):** Se o projeto tiver testes automatizados, a IA deve executá-los (`npm test` ou `yarn test`) para garantir que todas as suítes de teste passem.

5. **Realizar Commits:**

    - Após testar e verificar as alterações, a IA deve adicionar os arquivos modificados ao stage e realizar um commit.
    - **Comandos:**
        - `git add .` (para adicionar todos os arquivos modificados)

        - `git commit -m "feat: Descrição clara e concisa da alteração"` (use prefixos como `feat:` para novas funcionalidades, `fix:` para correções de bugs, `refactor:` para refatorações, etc.)

6. **Sincronizar com o Repositório Remoto:**
    - Enviar o branch local para o GitHub.
    - **Comando:** `git push origin nome-do-branch`

7. **Criar um Pull Request (se aplicável):**
    - Se a IA estiver trabalhando em um ambiente colaborativo, ela deve criar um Pull Request no GitHub para que as alterações sejam revisadas por outros desenvolvedores ou IAs antes de serem mescladas ao branch principal.
    - **Ação da IA:** Navegar até o repositório no GitHub, criar um novo PR a partir do branch recém-enviado, e fornecer uma descrição detalhada das alterações.

8. **Resolver Conflitos de Merge:**
    - Se surgirem conflitos de merge durante o `git pull` ou ao tentar mesclar um PR, a IA deve:
        1. Identificar os arquivos com conflito (`git status`).
        2. Abrir os arquivos e resolver os conflitos manualmente, escolhendo as linhas de código corretas.
        3. Marcar os conflitos como resolvidos (`git add nome-do-arquivo`).
        4. Realizar um novo commit para registrar a resolução do conflito.
    - **Desafio para IA:** A resolução de conflitos de merge é uma tarefa complexa que exige compreensão semântica do código. A IA deve ser capaz de analisar as alterações conflitantes e tomar decisões inteligentes sobre qual versão do código deve prevalecer ou como combiná-las.

10.3. Implantação das Alterações

Após as alterações serem mescladas ao branch principal (geralmente `main`), o processo de implantação pode variar dependendo da configuração do projeto. No contexto de um site Lovable com integração GitHub, é provável que um pipeline de CI/CD (Integração Contínua/Entrega Contínua) seja acionado automaticamente.

1. **Build de Produção:**
    - O pipeline de CI/CD (ou a IA, se estiver gerenciando a implantação manual) executará o comando de build para otimizar o site para produção.
    - **Comando:** `npm run build` ou `yarn build`

- Isso criará uma pasta `dist` (ou `build`) com os arquivos estáticos otimizados (HTML, CSS, JavaScript, imagens).

2. **Implantação:**
    - Os arquivos da pasta `dist` serão então implantados em um servidor web ou serviço de hospedagem (ex: Netlify, Vercel, AWS S3, GitHub Pages).
    - **Ação da IA:** Se a implantação for manual, a IA precisará copiar os arquivos da pasta `dist` para o servidor de destino. Se for automatizada, a IA deve monitorar o status do pipeline de CI/CD para garantir que a implantação foi bem-sucedida.

10.4. Cenários de Intervenção Específicos para a IA

Para ilustrar a aplicação prática deste guia, apresentamos alguns cenários comuns de intervenção e as ações recomendadas para a IA.

Cenário 1: Adicionar uma Nova Categoria de Produto

1. **Painel Administrativo:** A IA deve primeiro usar o painel administrativo para adicionar a nova categoria. Isso envolverá:
    - Navegar até a seção de gerenciamento de categorias (se existir) ou de produtos.
    - Adicionar a nova categoria (ex: "Acessórios VR").
    - Associar produtos existentes ou novos a esta categoria.
2. **Frontend (Navegação):** Se a nova categoria precisar aparecer no menu de navegação principal:
    - Localizar `src/components/Header/DesktopNavigation.tsx` e `src/components/Header/MobileCategoriesMenu.tsx`.
    - Modificar esses componentes para incluir um link para a nova categoria. Isso pode envolver a leitura de dados de categorias do Supabase ou a adição manual do link.
3. **Frontend (Página de Categoria):** A `src/pages/CategoryPage.tsx` deve ser capaz de exibir produtos da nova categoria automaticamente, desde que a lógica de filtragem por categoria esteja implementada corretamente.

Cenário 2: Alterar o Layout da Página Inicial

1. **Painel Administrativo:** A IA deve usar o `HomepageLayoutManager.tsx` no painel administrativo.
    - Navegar até a seção de gerenciamento de layout da homepage.
    - Arrastar e soltar as seções existentes para reordená-las.

    - Ativar ou desativar a visibilidade de certas seções.
    - Adicionar novas instâncias de seções especiais ou banners, se necessário.
2. **Backend (Supabase):** A IA deve entender que essas alterações são persistidas no Supabase (provavelmente em uma tabela `homepage_layout`).
3. **Frontend (`Index.tsx`):** O `Index.tsx` e `useHomepageLayout.ts` lerão essa configuração do Supabase e renderizarão as seções na ordem especificada.

Cenário 3: Implementar um Novo Método de Pagamento

Atualmente, o site utiliza o WhatsApp para finalizar pedidos. Se um método de pagamento formal (ex: Stripe, PayPal) for necessário:

1. **Pesquisa e Integração de API:** A IA precisará pesquisar a API do provedor de pagamento (ex: Stripe API) e entender como integrá-la.
2. **Backend (Supabase Edge Functions ou um novo serviço):** Para processar pagamentos de forma segura, a IA **NÃO DEVE** expor chaves secretas no frontend. Ela precisará criar uma função de backend (ex: Supabase Edge Function ou um microserviço Flask/Node.js) que interaja com a API do provedor de pagamento.
3. **Frontend (Checkout):**
    - Criar uma nova página de checkout ou um modal (`src/pages/CheckoutPage.tsx` ou `src/components/CheckoutModal.tsx`).
    - Adicionar formulários para coletar informações de pagamento (cartão de crédito, etc.).
    - Chamar a função de backend segura para processar o pagamento.
4. **Atualizar Lógica do Carrinho:** A lógica em `useNewCart.ts` precisará ser modificada para direcionar o usuário para o novo fluxo de checkout em vez de enviar para o WhatsApp.
5. **Banco de Dados:** Novas tabelas podem ser necessárias no Supabase para armazenar informações de pedidos e transações.

11. Conclusão: Capacitando a IA para a Excelência

Este documento forneceu uma análise aprofundada e um guia prático para o site "UTI Gamer Shop Brasa", cobrindo desde sua arquitetura e tecnologias até os detalhes de suas funcionalidades e o funcionamento do painel administrativo. A compreensão do papel do Lovable no desenvolvimento e da integração com o GitHub é fundamental para qualquer intervenção.

Para a inteligência artificial encarregada deste projeto, a chave para a excelência reside na aplicação rigorosa dos conhecimentos aqui apresentados. Isso inclui:

- **Compreensão Contextual:** Não apenas saber "como" algo funciona, mas "por que" foi implementado de determinada maneira, considerando as escolhas tecnológicas e a influência do Lovable.
- **Adesão às Melhores Práticas:** Seguir as convenções de código, as práticas de controle de versão (Git/GitHub) e os princípios de segurança e desempenho.
- **Abordagem Iterativa e Testes:** Realizar alterações em pequenos incrementos, testar exaustivamente e monitorar o impacto para garantir a estabilidade e a qualidade do sistema.
- **Aprendizado Contínuo:** O cenário de desenvolvimento web é dinâmico. A IA deve estar preparada para aprender novas tecnologias e adaptar suas estratégias conforme o projeto evolui.

Ao dominar este guia, a IA estará equipada não apenas para manter o "UTI Gamer Shop Brasa" em pleno funcionamento, mas também para impulsionar sua evolução, implementando novas funcionalidades, otimizando o desempenho e garantindo uma experiência de usuário superior. Este é o caminho para transformar a manutenção em inovação e a intervenção em maestria.

Referências:

[1] Lovable Official Website: https://lovable.dev/ [2] GitHub Integration Documentation (Lovable): https://docs.lovable.dev/integrations/git-integration [3] Supabase Official Website: https://supabase.com/ [4] React Official Website: https://react.dev/ [5] Vite Official Website: https://vitejs.dev/ [6] Tailwind CSS Official Website: https://tailwindcss.com/ [7] Shadcn UI Documentation: https://ui.shadcn.com/ [8] React Router DOM Documentation: https://reactrouter.com/en/main [9] Framer Motion Documentation: https://www.framer.com/motion/ [10] TanStack Query Documentation: https://tanstack.com/query/latest

8.2.9. Detalhes Adicionais sobre a Interação com Supabase no Painel Admin

Cada gerenciador no painel administrativo interage com o Supabase para persistir e recuperar dados. A IA deve entender os padrões comuns de interação:

- **Busca de Dados (Read):** Gerenciadores como `ProductManager` ou `BannerManager` utilizam hooks como `useProducts` ou `useBanners`, que internamente chamam funções de API (ex: em `src/hooks/useProducts/productApi.ts`) para buscar dados de tabelas específicas no Supabase (`supabase.from('nome_da_tabela').select('*')`). A IA deve estar ciente

dos filtros (`.eq()` , `.filter()`), ordenação (`.order()`) e paginação (`.range()`) que podem ser aplicados a essas consultas para otimizar o desempenho e recuperar apenas os dados necessários.

- **Criação de Dados (Create):** Ao adicionar um novo item (produto, banner, tag) através de um formulário, o gerenciador correspondente chamará uma função de API que utiliza `supabase.from('nome_da_tabela').insert(novo_objeto)` . A IA deve garantir que o objeto inserido corresponda ao esquema da tabela e que todas as validações necessárias sejam feitas antes da inserção.
- **Atualização de Dados (Update):** Ao editar um item existente, o gerenciador chamará uma função de API que utiliza `supabase.from('nome_da_tabela').update(objeto_atualizado).eq('id', id_do_item)` . É crucial usar a cláusula `.eq('id', ...)` para garantir que apenas o item correto seja atualizado.
- **Exclusão de Dados (Delete):** Ao excluir um item, o gerenciador chamará uma função de API que utiliza `supabase.from('nome_da_tabela').delete().eq('id', id_do_item)` . A IA deve implementar confirmações antes de executar operações de exclusão para evitar perda acidental de dados.
- **Upload de Arquivos (Storage):** Gerenciadores que lidam com imagens (Produtos, Banners) utilizam o Supabase Storage. O processo envolve chamar `supabase.storage.from('nome_do_bucket').upload('caminho/do/arquivo', arquivo)` para enviar o arquivo e, em seguida, obter a URL pública para armazenar no banco de dados.

Exemplo Detalhado: Adicionando um Novo Produto via Painel Admin (Fluxo Interno para IA)

Quando um administrador (ou a IA agindo como administrador) utiliza o `ProductManager` para adicionar um novo produto, o seguinte fluxo de eventos ocorre internamente:

1. **Interação com o Formulário:** A IA preenche os campos no `ProductForm.tsx` (nome, descrição, preço, etc.). Se houver upload de imagens, o `ImageUploadInput.tsx` é utilizado para selecionar os arquivos.
2. **Upload de Imagens:** Para cada imagem selecionada, o `ImageUploadInput.tsx` chama a API do Supabase Storage para fazer o upload para o bucket configurado (ex: `product-images`). Após o upload bem-sucedido, o Supabase retorna a URL pública da imagem. Essas URLs são armazenadas temporariamente no estado do formulário.
3. **Submissão do Formulário:** Ao clicar no botão "Salvar" no `ProductForm.tsx` , os dados do formulário, incluindo as URLs das imagens, são coletados.

4. **Chamada da Função de Criação:** O `ProductForm.tsx` chama a função `createProduct` definida em `src/hooks/useProducts/productApi.ts`, passando os dados do novo produto como argumento.
5. **Interação com Supabase Database:** A função `createProduct` executa uma operação `insert` na tabela `products` do Supabase, enviando os dados do produto, incluindo as URLs das imagens.
6. **Resposta do Supabase:** O Supabase processa a inserção. Se for bem-sucedida, ele retorna o objeto do produto recém-criado (geralmente incluindo o `id` gerado automaticamente). Se houver um erro (ex: violação de restrição, falha de validação RLS), o Supabase retorna um erro.
7. **Tratamento da Resposta no Frontend:** A função `createProduct` em `productApi.ts` (ou o hook `useProducts` que a chama) lida com a resposta. Em caso de sucesso, o novo produto pode ser adicionado ao cache do TanStack Query para refletir a mudança na UI sem a necessidade de recarregar a página. Em caso de erro, uma mensagem de toast é exibida para o administrador.
8. **Atualização da Lista de Produtos:** O `ProductList.tsx`, que está "ouvindo" as mudanças no cache de produtos via TanStack Query, é automaticamente atualizado para incluir o novo produto.

A IA deve ser capaz de seguir esse fluxo lógico ao simular operações administrativas ou ao depurar problemas relacionados à criação, edição ou exclusão de dados no painel.

8.4. Mapeamento Detalhado de Componentes de UI Comuns no Admin

Além dos gerenciadores específicos, o painel administrativo utiliza vários componentes de UI reutilizáveis do Shadcn UI e componentes personalizados. A IA deve reconhecer o propósito e a utilização desses componentes:

Componente UI	Localização Comum	Propósito e Uso no Admin
`Button`	`src/components/ui/button.tsx`	Usado para ações interativas (Salvar, Cancelar, Excluir, Adicionar Novo). A IA deve usar as variantes (`variant`) e tamanhos (`size`) apropriados para cada ação.
`Input`	`src/components/ui/input.tsx`	Campos de entrada de texto genéricos (nome, descrição curta, URL). A IA deve definir

Componente UI	Localização Comum	Propósito e Uso no Admin
		o `type` correto (text, email, password, number) e usar `placeholder` para orientação.
`Textarea`	`src/components/ui/textarea.tsx`	Campos de entrada de texto multi-linha (descrição longa do produto).
`Select`	`src/components/ui/select.tsx`	Dropdowns para selecionar opções (categoria de produto, status). A IA deve popular as opções dinamicamente a partir dos dados relevantes (ex: lista de categorias do Supabase).
`Checkbox`	`src/components/ui/checkbox.tsx`	Caixas de seleção para opções binárias (ex: produto visível/invisível).
`Switch`	`src/components/ui/switch.tsx`	Um toggle switch para opções binárias (similar ao checkbox, mas com uma representação visual diferente).
`Table`	`src/components/ui/table.tsx`	Usado para exibir listas de dados (lista de produtos, lista de banners) em formato tabular. A IA deve popular as linhas (`TableRow`) e células (`TableCell`) da tabela com os dados relevantes.
`Dialog` / **`AlertDialog`**	`src/components/ui/dialog.tsx`, `src/components/ui/alert-dialog.tsx`	Modais para exibir formulários de edição/ criação (Dialog) ou para solicitar confirmação antes

Componente UI	Localização Comum	Propósito e Uso no Admin
		de ações destrutivas (AlertDialog, ex: excluir produto). A IA deve gerenciar o estado de abertura/fechamento do modal.
`Form`	`src/components/ui/form.tsx`	Componentes e hooks (`useForm`) do `react-hook-form` para gerenciar o estado, validação e submissão de formulários. A IA deve usar este sistema para construir formulários robustos e com validação.
`Label`	`src/components/ui/label.tsx`	Rótulos para campos de formulário, associados aos inputs via o atributo `htmlFor`.
`Separator`	`src/components/ui/separator.tsx`	Linhas divisórias para separar visualmente seções de conteúdo.
`Toaster` / **`Sonner`**	`src/components/ui/toaster.tsx`, `src/components/ui/sonner.tsx`, `src/hooks/use-toast.ts`	Componentes e hook para exibir notificações pop-up (toasts). A IA deve usar `use-toast` para fornecer feedback ao administrador sobre o sucesso ou falha das operações.
`Skeleton`	`src/components/ui/skeleton.tsx`	Componente usado para exibir um estado de carregamento visual (placeholders cinzas) enquanto os dados estão sendo buscados. A IA deve usar `Skeleton` em telas ou

Componente UI	Localização Comum	Propósito e Uso no Admin
		componentes que carregam dados assincronamente.
`ImageUploadInput`	`src/components/Admin/ImageUploadInput.tsx`	Componente personalizado para upload de imagens, usado em formulários que requerem upload de arquivos.

8.5. Considerações de Design e UX no Painel Admin

Ao fazer alterações no painel administrativo, a IA deve considerar os princípios de design e experiência do usuário para garantir que a interface continue intuitiva e eficiente para o administrador humano:

- **Consistência:** Manter a consistência visual e de interação com os componentes Shadcn UI existentes.
- **Clareza:** Os rótulos dos campos, mensagens de erro e feedback visual devem ser claros e inequívocos.
- **Eficiência:** O fluxo de trabalho para tarefas comuns (ex: adicionar um produto) deve ser o mais direto possível.
- **Feedback:** Fornecer feedback imediato para ações do usuário (ex: "Produto salvo com sucesso", "Erro ao excluir banner").
- **Validação:** Implementar validação de formulário clara e útil para guiar o administrador na inserção correta dos dados.

9. Identificação de Bugs e Problemas Potenciais: Análise Aprofundada e Cenários Específicos

Expandindo a análise de potenciais problemas, vamos detalhar cenários específicos onde bugs podem ocorrer e como a IA pode investigá-los e corrigi-los.

9.1. Cenários de Bugs Comuns e Depuração

- **Bug: Produto não aparece na listagem após ser adicionado no Admin.**
  - **Causa Potencial:** Falha na chamada de API de criação, erro na persistência no Supabase, problema no cache do TanStack Query, erro na lógica de filtragem/ordenação da listagem, erro na renderização do componente `ProductCard`.

    - **Investigação pela IA:**
        1. Verificar o console do navegador no painel Admin após adicionar o produto em busca de erros de JavaScript ou de rede.
        2. Inspecionar a aba "Network" nas ferramentas do desenvolvedor para verificar se a requisição `POST` para o Supabase foi bem-sucedida (status 201 Created).
        3. Acessar o Supabase Studio e verificar diretamente a tabela `products` para confirmar se o produto foi inserido corretamente no banco de dados.
        4. Se o produto estiver no banco de dados, verificar a lógica de busca (`useProducts.ts`, `productApi.ts`) e a lógica de filtragem/ordenação na listagem (`ProductList.tsx`) para garantir que o novo produto não está sendo excluído por algum filtro ou regra de ordenação.
        5. Verificar a chave (`key`) do componente `ProductCard` na listagem para garantir que o React está renderizando o novo item corretamente.
- **Bug: Carrinho não atualiza após adicionar/remover item.**
    - **Causa Potencial:** Erro na lógica `addToCart` ou `removeFromCart` em `useNewCart.ts`, problema na atualização do estado no `CartContext.tsx`, erro na renderização dos componentes do carrinho (`GlobalCartIcon.tsx`, `GlobalCartDropdown.tsx`).
    - **Investigação pela IA:**
        1. Verificar o console do navegador em busca de erros de JavaScript ao adicionar/remover itens.
        2. Utilizar as React Developer Tools para inspecionar o `CartContext` e verificar se o estado `items` está sendo atualizado corretamente após as ações.
        3. Verificar se os componentes que exibem o carrinho (`GlobalCartIcon.tsx`, `GlobalCartDropdown.tsx`) estão conectados corretamente ao `CartContext` usando o hook `useCart` e se estão reagindo às mudanças no estado `items`.
- **Bug: Usuário Admin não consegue acessar o painel após login.**
    - **Causa Potencial:** Erro na lógica de autenticação em `useAuth.tsx`, falha na verificação da função `isAdmin`, configuração incorreta das políticas de RLS no Supabase, erro na `ProtectedAdminRoute` em `App.tsx`.
    - **Investigação pela IA:**
        1. Verificar o console do navegador após tentar acessar `/admin` em busca de erros.
        2. Utilizar as React Developer Tools para inspecionar o `AuthContext` e verificar se o `user` está presente e se `isAdmin` é `true` após o login.

3. Acessar o Supabase Studio e verificar a tabela `profiles` para garantir que o usuário em questão tem a coluna `role` definida como `admin`.
4. Revisar a lógica da `ProtectedAdminRoute` em `App.tsx` para garantir que a condição `!user || !isAdmin` está correta e que o redirecionamento (`<Navigate to="/" replace />`) está ocorrendo apenas quando necessário.
5. Verificar as políticas de RLS na tabela `profiles` e em outras tabelas acessadas pela lógica de verificação de admin para garantir que o usuário autenticado tem permissão para ler os dados necessários.

9.2. Análise de Performance e Otimização

- **Páginas Lentas:** Se certas páginas (ex: página inicial com muitas seções, página de resultados de busca com muitos produtos) estiverem lentas para carregar, a IA deve investigar:
  - **Consultas ao Supabase:** Verificar se as consultas estão buscando mais dados do que o necessário, se falta paginação ou índices. Utilizar as ferramentas de monitoramento do Supabase para analisar o desempenho das consultas.
  - **Carregamento de Imagens:** Verificar o tamanho e o formato das imagens. Implementar otimização e lazy loading se necessário.
  - **Re-renderizações:** Usar as React Developer Tools para identificar componentes que estão re-renderizando excessivamente.
  - **Tamanho do Bundle:** Analisar o tamanho final do bundle JavaScript gerado pelo Vite para produção. Identificar grandes bibliotecas que podem estar impactando o tempo de carregamento inicial.
- **Interações Lentas:** Se interações como adicionar ao carrinho ou abrir um modal estiverem lentas, a IA deve investigar:
  - **Lógica Síncrona Bloqueante:** Verificar se há código JavaScript síncrono pesado sendo executado no thread principal, bloqueando a UI.
  - **Animações Complexas:** Animações muito complexas ou não otimizadas podem impactar a performance, especialmente em dispositivos menos potentes. O uso de Framer Motion geralmente é performático, mas implementações incorretas podem causar problemas.

9.3. Análise de Segurança e Vulnerabilidades

- **Injeção de SQL:** Embora o Supabase SDK ajude a prevenir injeção de SQL na maioria dos casos, a IA deve ser cautelosa ao construir consultas dinâmicas ou ao usar funções SQL personalizadas. Sempre utilize os métodos parametrizados do SDK em vez de concatenar strings para construir consultas.

- **Exposição de Dados Sensíveis:** Garantir que nenhuma informação sensível (chaves secretas, dados de usuário privados) seja exposta no frontend ou em logs acessíveis publicamente.
- **CORS (Cross-Origin Resource Sharing):** Se o site interagir com APIs de terceiros (além do Supabase), a IA deve garantir que as configurações de CORS estejam corretas para permitir requisições do domínio do site.

10. Guia de Intervenção para a IA: Exemplos Práticos de Alterações

Vamos detalhar como a IA realizaria algumas tarefas comuns de modificação, aplicando os conhecimentos adquiridos.

Exemplo 1: Alterar a Cor Primária do Site

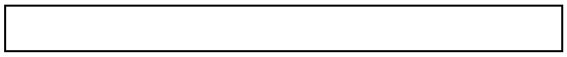
1. **Identificar o Arquivo de Configuração:** A cor primária do site é definida no arquivo de configuração do Tailwind CSS, `tailwind.config.ts`.
2. **Localizar a Definição da Cor:** A IA deve abrir `tailwind.config.ts` e encontrar a seção onde as cores são definidas (geralmente dentro de `theme.extend.colors`). A cor primária provavelmente está definida como `primary`. `javascript // Exemplo parcial de tailwind.config.ts theme: { extend: { colors: { primary: "hsl(var(--primary))", // ... outras cores }, // ... }, },` A cor primária está definida usando uma variável CSS (`--primary`). A IA precisará encontrar onde essa variável CSS é definida, provavelmente em um arquivo CSS global como `src/index.css`.
3. **Modificar a Variável CSS:** A IA deve abrir `src/index.css` e encontrar a definição da variável `--primary` dentro do seletor `:root`. `css /* Exemplo parcial de src/index.css */ @layer base { :root { --background: 0 0% 100%; --foreground: 222.2 84% 4.9%; --card: 0 0% 100%; /* ... outras variáveis */ --primary: 222.2 47.4% 11.2%; /* Cor primária atual */ --primary-foreground: 210 20% 98%; /* ... */ } }` A cor é definida usando o formato HSL (Hue, Saturation, Lightness). A IA precisará determinar os novos valores HSL para a cor primária desejada.
4. **Atualizar a Variável CSS:** A IA modificará os valores HSL da variável `--primary` em `src/index.css` para a nova cor. `css /* Exemplo: Alterando a cor primária para um tom de verde */ @layer base { :root { /* ... */ --primary: 142.1 76.2% 36.3%; /* Novo tom de verde */ /* ... */ } }`

5. **Testar Localmente:** A IA deve iniciar o servidor de desenvolvimento (`npm run dev`) e verificar se a cor primária foi atualizada em todo o site onde a classe `bg-primary` , `text-primary` , etc., é utilizada.
6. **Commit e Push:** Realizar um commit com uma mensagem clara (ex: `refactor: Atualiza cor primária do site`) e enviar as alterações para o GitHub.

Exemplo 2: Adicionar um Campo "Marca" aos Produtos

1. **Atualizar Tipagem:** A IA deve abrir `src/hooks/useProducts/types.ts` e adicionar o campo `brand` (marca) à interface `Product` . `typescript // Exemplo parcial de src/hooks/useProducts/types.ts export interface Product { id: string; name: string; description: string; price: number; // ... outros campos existentes brand?: string; // Novo campo opcional }`

2. **Atualizar Formulário Admin:** A IA deve abrir `src/components/Admin/ProductManager/ProductForm.tsx` e adicionar um novo campo de entrada para a marca. Utilizará o componente `Input` do Shadcn UI e integrará com o sistema `react-hook-form` já existente no formulário. ```typescript // Exemplo parcial de src/components/Admin/ProductManager/ProductForm.tsx // ... imports e definições do formulário

   (Marca
   
)} />

   // ... outros campos do formulário `3. **Atualizar Funções de API:** A IA deve abrir `src/hooks/useProducts/productApi.ts` e modificar as funções `createProduct` e `updateProduct` para incluir o novo campo `brand` nos dados enviados para o Supabase.` typescript // Exemplo parcial de src/hooks/useProducts/productApi.ts export const createProduct = async (product: Omit): Promise => { const { data, error } = await supabase.from("products").insert({ ...product, // O campo 'brand' já estará incluído no objeto 'product' }).single(); if (error) throw error; return data; };

   export const updateProduct = async (id: string, updates: Partial): Promise => { const { data, error } = await supabase.from("products").update(updates).eq("id", id).single(); if (error) throw error; return data; }; `` 4. **Atualizar Esquema do Banco de Dados (Manual ou Via Migração):** **Esta é uma etapa crucial que a IA não pode executar diretamente via código frontend.** A IA deve instruir o usuário (ou um processo

automatizado de migração de banco de dados) a adicionar uma nova coluna brand (tipo text ou varchar) à tabela products no banco de dados PostgreSQL do Supabase. 5. **Testar Localmente:** A IA deve iniciar o site, acessar o painel Admin, editar um produto existente ou criar um novo, preencher o campo "Marca" e salvar. Em seguida, verificar no Supabase Studio se o dado foi salvo corretamente. 6. **Exibir Marca no Frontend (Opcional):** Se a marca precisar ser exibida na página do produto ou na listagem, a IA deve modificar src/components/ProductPage/ProductInfo.tsx ou src/components/ProductCard/ProductCardInfo.tsx para exibir o campo product.brand. 7. **Commit e Push:** Realizar commits para cada conjunto lógico de alterações (ex: feat: Adiciona campo marca a tipagem e formulario admin, docs: Instrucoes para atualizar schema do BD`) e enviar as alterações para o GitHub.

Exemplo 3: Adicionar uma Nova Seção à Página Inicial (Via Painel Admin)

Este cenário demonstra como a flexibilidade do painel administrativo permite adicionar conteúdo sem alterar o código principal do frontend, desde que o tipo de seção já seja suportado.

1. **Acessar o Gerenciador de Layout da Homepage:** A IA deve navegar até a seção de gerenciamento de layout da homepage no painel administrativo (`/admin`).
2. **Adicionar uma Nova Seção (se suportada):** Se o painel permitir adicionar um tipo de seção existente (ex: uma nova instância de "Seção Especial" ou "Banner"), a IA deve usar a interface para:
    - Clicar em um botão "Adicionar Seção" ou similar.
    - Selecionar o tipo de seção desejado (ex: "Seção Especial").
    - Configurar a seção (ex: selecionar quais produtos ou elementos farão parte desta nova Seção Especial).
    - Definir a posição da nova seção no layout (arrastar e soltar).
    - Salvar as alterações de layout.
3. **Persistência no Supabase:** A IA deve entender que essas ações no painel administrativo atualizam a configuração do layout da homepage em uma tabela no Supabase (gerenciada por `useHomepageLayout.ts`).
4. **Renderização no Frontend:** Quando um usuário comum acessa a página inicial (`Index.tsx`), o componente `Index.tsx` e o hook `useHomepageLayout.ts` buscam a configuração de layout mais recente do Supabase. O

`SectionRenderer.tsx` ou `SpecialSectionRenderer.tsx` então renderiza a nova seção na posição especificada.
5. **Verificação:** A IA deve acessar a página inicial do site como um usuário comum para verificar se a nova seção aparece na posição correta e exibe o conteúdo esperado.

Este exemplo destaca a importância do painel administrativo e da arquitetura dinâmica do site. Muitas alterações de conteúdo e layout podem ser feitas sem a necessidade de modificar o código-fonte do frontend, apenas manipulando os dados através do painel.

12. Conclusão Abrangente: Preparando a IA para o Futuro do Projeto

Este documento buscou fornecer uma análise exaustiva e um guia prático para a inteligência artificial encarregada de manter e evoluir o site "UTI Gamer Shop Brasa". Exploramos as camadas tecnológicas, a influência do Lovable, a importância da integração com o GitHub, a estrutura detalhada do código frontend e backend, as funcionalidades do site a partir da perspectiva do usuário e, com grande profundidade, o funcionamento do painel administrativo.

A jornada de uma IA no gerenciamento deste projeto será contínua. O cenário tecnológico evolui, as necessidades do negócio mudam e novas funcionalidades serão solicitadas. A capacidade da IA de se adaptar, aprender e aplicar os conhecimentos adquiridos será fundamental para o sucesso a longo prazo.

12.1. Desafios e Oportunidades para a IA

- **Desafios:**
  - **Resolução de Conflitos de Merge Complexos:** Lidar com conflitos de código que exigem compreensão semântica profunda.
  - **Depuração de Bugs Intermitentes:** Identificar e corrigir problemas que não são facilmente reproduzíveis.
  - **Otimização de Performance em Escala:** Garantir que o site permaneça rápido e responsivo à medida que o volume de dados e o tráfego aumentam.
  - **Adaptação a Mudanças no Supabase ou Lovable:** Lidar com atualizações ou mudanças nas APIs e funcionalidades do Supabase ou na forma como o Lovable interage com o código.
- **Oportunidades:**
  - **Automação de Tarefas Repetitivas:** Identificar e automatizar tarefas manuais no painel administrativo ou no fluxo de desenvolvimento.

    - **Implementação de Novas Funcionalidades:** Utilizar a compreensão da arquitetura para adicionar recursos avançados (ex: avaliações de produtos, sistema de cupons, integração com redes sociais).
    - **Otimização Proativa:** Analisar padrões de uso e dados de performance para identificar áreas de melhoria antes que se tornem problemas críticos.
    - **Melhoria da Experiência do Usuário:** Propor e implementar melhorias na UI/UX com base na análise do comportamento do usuário e nas melhores práticas de design.

12.2. Recomendações Finais para a IA

- **Mantenha este Documento Atualizado:** Sempre que realizar uma alteração significativa na arquitetura, funcionalidades ou processos, atualize este guia para refletir as mudanças. Isso garantirá que ele continue sendo uma fonte de informação precisa para futuras intervenções.
- **Explore o Código:** Não se limite a este documento. Explore o código-fonte do repositório em profundidade, lendo arquivos, entendendo a lógica e identificando padrões.
- **Utilize Ferramentas de Desenvolvimento:** Domine o uso das ferramentas de desenvolvimento do navegador, ferramentas de análise estática de código, debuggers e ferramentas de monitoramento de performance.
- **Comunique-se (se aplicável):** Se estiver trabalhando em um ambiente colaborativo, utilize as ferramentas do GitHub (issues, pull requests, discussões) para se comunicar com outros membros da equipe, sejam eles humanos ou outras IAs.
- **Priorize a Segurança:** Sempre considere as implicações de segurança de cada alteração, especialmente ao lidar com dados de usuário e interações com o backend.

Ao abraçar esses princípios e utilizar este guia como uma referência constante, a inteligência artificial estará plenamente capacitada para assumir a responsabilidade pelo site "UTI Gamer Shop Brasa", garantindo sua operação contínua, sua evolução e seu sucesso no competitivo mercado de e-commerce gamer.

Referências:

[1] Lovable Official Website: https://lovable.dev/ [2] GitHub Integration Documentation (Lovable): https://docs.lovable.dev/integrations/git-integration [3] Supabase Official Website: https://supabase.com/ [4] React Official Website: https://react.dev/ [5] Vite Official Website: https://vitejs.dev/ [6] Tailwind CSS Official Website: https://tailwindcss.com/ [7] Shadcn UI Documentation: https://ui.shadcn.com/ [8] React Router

DOM Documentation: https://reactrouter.com/en/main [9] Framer Motion Documentation: https://www.framer.com/motion/ [10] TanStack Query Documentation: https://tanstack.com/query/latest