Manual Prático de Prompts: Desenvolvimento Laravel e Livewire no cPanel

Este documento estabelece o padrão arquitetural e operacional para o desenvolvimento de aplicações modernas utilizando a stack **PET/TALL** (PHP, Echo-JS/Alpine, Tailwind, Laravel/Livewire) em ambientes de hospedagem compartilhada. Como arquiteto, seu objetivo é garantir que a I.A. produza código alinhado à filosofia **Hypermedia-Driven** , otimizando o consumo de recursos do cPanel e eliminando a complexidade desnecessária de SPAs tradicionais.

1. Fundamentos da Stack e Contextualização para I.A.

1.1 Diretriz Arquitetural PET/TALL (Server-Driven HTML)

Diferente de arquiteturas baseadas em Node.js que exigem processos persistentes e APIs JSON complexas, a stack PET/TALL foca na eficiência do ciclo de vida do PHP.**Tabela Comparativa de Eficiência em Ambiente cPanel:**| Dimensão | SPA (Node.js + React) | PET/TALL (Laravel + Livewire) | Vantagem Arquitetural || ------ | ------ | ------ | ------ || **Recursos (RAM)** | Persistente (Consumo constante). | Sob demanda (Executa e libera RAM). | Estabilidade em planos básicos. || **SEO / Indexação** | Depende de pré-renderização. | **Nativo/Instantâneo** . | Indexação imediata pelo Google. || **Segurança** | Expõe endpoints JSON e JWT. | Conexão Local (Socket MySQL). | Credenciais protegidas fora da web. || **Deploy** | Build local + NPM + Restart. | Sincronização Git direta. | Zero downtime e facilidade de manutenção. |

1.2 Configuração de "System Prompt" (As Leis de Diretriz)

Forneça estas instruções à I.A. antes de iniciar o projeto. Este prompt proíbe explicitamente o desvio para APIs JSON ou Fetch API manual.

```
# SYSTEM PROMPT: LEIS DE DIRETRIZ ARQUITETURAL
Assuma o papel de um Engenheiro de Software Sênior focado em
Laravel e cPanel.
1. O Back-End é estritamente Laravel (PHP de ciclo curto).
2. É proibido o uso de APIs JSON pesadas ou chamadas manuais de
Fetch API/Axios.
3. A comunicação deve ser Hypermedia-Driven via Livewire ou HTMX
(Server-Driven HTML).
4. O servidor deve devolver fragmentos puros de HTML, nunca
objetos JSON brutos.
5. Para lógica de interface local (modais/menus), use
exclusivamente Alpine.js.
6. Segurança: Use Eloquent ORM ou PDO para sanitizar todas as
queries (Prevenção de SQL Injection).
7. Estilização: Utilize classes utilitárias do Tailwind CSS ou
Tokens do Design System Emerald Archive.
```

2. Transição de PHP Puro para Laravel

2.1 Modernização de Código e Segurança Eloquent

Instrua a I.A. a refatorar scripts antigos focando na segurança dos dados (Red Alert: SQL Injection).

- **Prompt de Refatoração Segura:** "Refatore este script PHP estruturado colar código para um Controller e Model do Laravel. **Alerta Vermelho:** Não utilize queries SQL dinâmicas com variáveis soltas. Converta tudo para a sintaxe **Eloquent ORM** (ex: Membro::where('status', 'ativo')->get()) para garantir a sanitização automática via PDO."

2.2 Componentização com Blade

Utilize o motor de templates Blade para evitar repetição de código e criar layouts mestre.

- **Prompt de Estruturação Blade:** "Converta este HTML misto em um sistema de templates Blade. Crie um layout.blade.php com diretivas @stack para scripts e @yield para conteúdo. Utilize componentes aninhados para elementos repetitivos como cards e alertas, seguindo a semântica do Blade."

3. Desenvolvimento de Componentes Livewire e Reatividade

3.1 A Lógica do Livewire (PHP Reativo)

Solicite componentes que gerenciem o estado no servidor, devolvendo apenas o HTML necessário para o DOM.

- **Prompt de Busca com Debounce:** "Crie um componente Livewire para 'Busca de Membros'. O input de busca deve utilizar a diretiva wire:model.live.debounce.400ms para evitar sobrecarga no servidor cPanel. O PHP deve filtrar os dados no banco e devolver apenas as linhas da tabela em fragmentos HTML."

3.2 Matriz de Responsabilidades (Livewire vs. Alpine.js)

Defina o escopo para evitar que a I.A. sobrecarregue o servidor com efeitos visuais.

1. **Livewire (Server-Side):** Persistência no banco de dados, validação de regras de negócio, cálculos complexos e autenticação.
2. **Alpine.js (Client-Side):** Abertura de modais, estados de menus dropdown, transições visuais instantâneas e feedbacks de carregamento locais.

4. Gerenciamento de Banco de Dados no cPanel (MySQL)

4.1 Configuração Técnica de Tabelas

Utilize os padrões técnicos extraídos da fonte para garantir performance e integridade dos índices.**Especificações para Tabelas de Usuários/Sistemas:**| Coluna | Tipo (MySQL) | Tamanho | Atributos / Motivo || ------ | ------ | ------ | ------ || **id** | INT | UNSIGNED | Performance e Auto Incremento. || **email** | VARCHAR | 191 | Limite ideal para **Índices Únicos** no MySQL/InnoDB. || **senha** | VARCHAR | 255 | Espaço para hashes de criptografia (bcrypt/argon2). || **status** | VARCHAR | 50 | Uso de strings semânticas para estados. |

- **Configuração de Engine:** Sempre use Engine=InnoDB e Collation=utf8mb4_unicode_ci para suporte a acentos e emojis sem corromper dados.

4.2 Segurança de Conexão e Arquivo .env

O Laravel no cPanel deve ser configurado para rodar em localhost.

- **Prompt de Segurança .env:** "Gere um arquivo .env para Laravel no cPanel. Configure o DB_HOST como 127.0.0.1 ou localhost. Instrua-me sobre como manter este arquivo e a pasta do aplicativo um nível acima da public_html, expondo apenas a pasta public via link simbólico."

5. Deploy Automatizado e Seguro via Git

5.1 Correção do Erro 128 (Repositórios Privados)

Para clonar repositórios privados sem terminal e evitar falhas de autenticação HTTPS, utilize o Personal Access Token (PAT) via .gitconfig.**Checklist de Configuração Automática:**

1. Crie o arquivo .gitconfig na raiz da sua conta cPanel (acima da public_html).
2. Insira o seguinte bloco de código (substituindo seus dados):

```
[url "https://seu_usuario:seu_token_ghp@github.com/"]
    insteadOf = https://github.com/
```

5.2 Estrutura de Pastas de Produção

O código fonte **nunca** deve residir dentro da public_html. Organize como:

- /home/usuario/app-laravel (Código fonte, .env, logs).
- /home/usuario/public_html (Apenas o conteúdo da pasta public via symlink).

6. Solução de Problemas e Design System

6.1 Prompt de Correção de Rumo (O "Puxão de Orelha")

Utilize este comando caso a I.A. sugira soluções baseadas em Node.js ou JSON:**"PARE! Você está violando a diretriz PET/TALL.** Estamos operando em um ambiente cPanel limitado. É estritamente proibido o uso de Fetch API manual ou exposição de JSON para lógica de negócio. O 'Red Alert' de segurança proíbe APIs desprotegidas. Refaça a lógica utilizando componentes Livewire e devolva fragmentos de HTML puros gerados pelo PHP."

6.2 UX e Design System Emerald Archive

O design deve ser agnóstico de builds pesados. Utilize o sistema de **Custom Properties** para garantir flexibilidade sem compilação.

- **Prompt de Estilização:** "Estilize este componente utilizando os tokens do design system **Emerald Archive** . Use propriedades CSS como var(--primary), var(--accent) e var(--surface) em vez de cores hexadecimais fixas. Isso permitirá a troca de tema sem necessidade de rodar npm run build no servidor."

6.3 Feedback de Carregamento

Implemente respostas visuais imediatas para compensar a latência da hospedagem compartilhada.

- **Prompt de Feedback:** "Implemente um botão de submissão que, ao ser clicado, utilize Alpine.js para disparar a classe .is-busy e exibir um spinner. O botão deve ser desabilitado momentaneamente para evitar submissões duplicadas (double-click) enquanto o Livewire processa os dados no servidor."