Guia de Atualização Arquitetural: Integração Laravel na Stack PET/TALL para cPanel

Este documento estabelece as diretrizes para a transição de aplicações PHP estruturadas para o framework Laravel, utilizando a arquitetura PET/TALL (PHP, Eloquent, Tailwind, Livewire/Alpine.js). O foco é a otimização para ambientes de hospedagem compartilhada (cPanel), priorizando estabilidade, segurança e produtividade no desenvolvimento assistido por I.A.

1. Fundamentos da Transição Arquitetural

A escolha do Laravel dentro da filosofia PET/TALL no cPanel não é apenas uma preferência tecnológica, mas uma decisão estratégica de infraestrutura. Diferente de arquiteturas Node.js, que exigem processos contínuos via Phusion Passenger e consomem RAM de forma ininterrupta, o PHP é o "rei nativo" do cPanel. Ele opera no modelo de ciclo curto: o script executa sob demanda via Apache/LiteSpeed, entrega o HTML e libera a memória imediatamente.

A adoção do **Livewire** (o "L" da stack TALL) em vez de HTMX puro é recomendada para projetos que escalam em regras de negócio complexas, pois o Livewire oferece uma abstração de estado mais robusta dentro do ecossistema Laravel, mantendo a reatividade sem a necessidade de APIs JSON manuais.

Tabela Comparativa de Infraestrutura (cPanel)

Dimensão	Node.js + React (SPA)	PHP Laravel + PET/TALL
Motor Nativo	Phusion Passenger / Proxy Reverso	Apache / LiteSpeed (Nativo)
Consumo de RAM	Persistente (Consumo constante)	Sob Demanda (Libera após execução)
Gestão de Processos	Requer PM2 ou Restart via cPanel	Gerenciado nativamente pelo Servidor Web

Complexidade de Deploy	Alta (Builds, NPM, Restart de App)	Baixa (Sincronização Git direta)
Segurança de Dados	Exposição via APIs REST/JWT	Encapsulada via Socket Local/Eloquent

2. Matriz de Responsabilidades Tecnológicas (PET/TALL)

Para garantir a coesão do sistema, cada componente deve atuar estritamente em seu escopo:

- **PHP/Laravel:** Motor de negócio, roteamento, lógica de autenticação e controle de acesso.
- **Eloquent ORM:** Abstração de banco de dados, garantindo integridade sem queries SQL manuais.
- **Livewire:** Reatividade assíncrona baseada em servidor (Server-Driven HTML).
- **Alpine.js:** Comportamento visual e estados locais no cliente (ex: modais, dropdowns).
- **Tailwind CSS:** Design system utilitário. **Nota:** O build deve ser feito localmente e o manifesto incluído no Git para evitar dependência de NPM no servidor.

3. Configuração e Segurança do Ambiente cPanel

O isolamento de diretórios é a primeira linha de defesa. No cPanel, nunca coloque a raiz do Laravel dentro da `public_html`.

Estrutura de Diretórios e Permissões

O diretório do Laravel deve residir na raiz da conta (`/home/usuario/`), protegendo o arquivo `.env` e a lógica do app.

```
/home/usuario/
│
├── meu-app-laravel/          # Raiz do framework (fora da área pública)
│   ├── app/
│   ├── bootstrap/cache/      # Permissão 775
│   ├── storage/              # Permissão 775
│   ├── .env                  # Credenciais protegidas
│   └── public/               # Assets públicos
│
└── public_html/              # Link simbólico para meu-app-laravel/public
```

Comando de Arquiteto: Para realizar o vínculo correto, apague a pasta `public_html` vazia e execute via terminal (ou cron job se o terminal estiver desabilitado):

ln -s /home/usuario/meu-app-laravel/public /home/usuario/public_html

4. Persistência de Dados com Eloquent ORM

O Eloquent simplifica a comunicação com o MySQL do cPanel. É imperativo seguir a convenção de nomenclatura do cPanel, que exige o prefixo da conta em bancos e usuários.

Configuração de Conexão (Socket Unix)

No cPanel, prefira `localhost` em vez de `127.0.0.1`. O uso de `localhost` força a conexão via Unix Socket, que é mais rápida e estável em ambientes compartilhados.

- **DB_HOST:** `localhost`
- **DB_DATABASE:** `prefixo_nome_banco`
- **DB_USERNAME:** `prefixo_usuario_banco`

Exemplo Prático: Eloquent vs. SQL Manual

// Eloquent: Seguro contra SQL Injection por padrão e legível
$membros = Membro::where('status', 'ativo')->orderBy('nome')->get();

As *Migrations* são essenciais para versionar o banco. Se o terminal estiver bloqueado pela hospedagem, você pode disparar as migrações via rota temporária: `Route::get('/migrate', fn() => Artisan::call('migrate'));`.

5. Interface Reativa com Laravel Livewire

O Livewire substitui a necessidade de frameworks pesados como React para criar interatividade, eliminando a complexidade de rotas duplicadas e autenticação via Token.

Componente de Busca em Tempo Real

Classe do Componente (PHP):

```
class BuscaMembros extends Component {
    public $busca = '';

    public function render() {
        return view('livewire.busca-membros', [
            'membros' => Membro::where('nome', 'like', "%{$this->busca}%")->get(),
```

```
        ]);
    }
}
```

View (Blade):

```
<div>
    <input type="text" wire:model.live.debounce.400ms="busca" class="form-input">
    <ul class="mt-4">
        @foreach($membros as $membro)
            <li>{{ $membro->nome }}</li>
        @endforeach
    </ul>
</div>
```

6. Gerenciamento de Comportamento Local com Alpine.js

Use Alpine.js para interações que não dependem do servidor. Isso reduz a latência e economiza requisições.

Exemplo: Dropdown Declarativo

```
<div x-data="{ aberto: false }" class="relative">
    <button @click="aberto = !aberto">Menu</button>
    <div x-show="aberto" @click.away="aberto = false" class="absolute shadow-lg">
        Conteúdo do Menu
    </div>
</div>
```

7. Fluxo de Desenvolvimento e Deploy via Git

O terminal do cPanel é frequentemente limitado. Para garantir um deploy fluido, utilize as táticas abaixo:

1. **Desenvolvimento Local:** Configure o Tailwind e execute o build. O arquivo `public/build/manifest.json` deve ser commitado.
2. **Workaround de SSH Bloqueado:** Se o servidor recusar a chave SSH, crie um arquivo `.gitconfig` na raiz da conta cPanel:

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

1. **Troubleshooting de Clone:** Em casos extremos de permissão, torne o repositório **Público** temporariamente, realize o clone no cPanel via "Git Version Control" e retorne-o para **Privado** imediatamente.
2. **Pós-Deploy:** Sempre execute a limpeza de cache para evitar que configurações antigas do `.env` fiquem presas em memória:
    - `php artisan config:cache`
    - `php artisan view:clear`

8. Manual de Padronização para I.A. (System Prompts)

Instrua sua I.A. (Cursor, ChatGPT) com os prompts abaixo para manter a integridade da stack.

Prompt de Kickoff

Estou iniciando um projeto Laravel no cPanel utilizando a stack PET/TALL.
O back-end deve usar Eloquent para persistência (sempre localhost para sockets).
Reatividade via Livewire (Server-Driven HTML).
Alpine.js apenas para estados de UI locais (modais, abas).
Estilização exclusiva com Tailwind CSS.
Não sugira Node.js, APIs JSON ou bundles JS complexos.
Assuma que a public_html é um link simbólico para a pasta /public.
Como estruturar o modelo e componente Livewire para [FUNCIONALIDADE]?

Prompt de Refatoração

Refatore este código para seguir a filosofia PET/TALL.
Substitua chamadas fetch() e lógica de manipulação de DOM por componentes Livewire.
Garanta que as classes do Tailwind sejam utilizadas em vez de CSS personalizado.
Remova lógicas de estado global no frontend, movendo a verdade para o Banco de Dados.

9. Conclusão e Checklist de Validação

A stack PET/TALL transforma o cPanel em uma plataforma de alta performance com baixo custo operacional.

Checklist de Validação Final

- [ ] **Segurança:** Arquivo `.env` e pastas de lógica fora da `public_html`.
- [ ] **Permissões:** Pastas `storage` e `bootstrap/cache` com permissão 775.
- [ ] **Link Simbólico:** `public_html` apontando corretamente para `laravel/public`.
- [ ] **Banco de Dados:** Conexão via `localhost` (Unix Socket) com prefixo da conta.

- [ ] **Build:** Assets do Tailwind compilados localmente e manifestos sincronizados via Git.
- [ ] **Deploy:** Cache limpo via `php artisan config:clear` após a atualização.