

# Linktree

28/09/2024

Matheus Patricio da S e Silva  
projeto - CRUD em Laravel

## Sumário:

Sumário:.....	2
<b>1. Visão Geral.....</b>	<b>4</b>
<b>2. Objetivos.....</b>	<b>4</b>
<b>3. Especificações.....</b>	<b>4</b>
<b>4. Documentação das Rotas.....</b>	<b>5</b>
<b>4.1 Rotas Públicas.....</b>	<b>5</b>
Rota Principal.....	5
<b>4.2 Rotas de Administração.....</b>	<b>5</b>
Grupo de Rotas de Administração.....	5
<b>4.2.1 Autenticação.....</b>	<b>5</b>
<b>4.2.2 Rotas Protegidas.....</b>	<b>6</b>
Gerenciamento de Páginas.....	6
Gerenciamento de Links.....	7
<b>4.3 Páginas Dinâmicas.....</b>	<b>8</b>
<b>5. Documentação do Login.....</b>	<b>8</b>
Objetivo.....	8
Especificações.....	9
<b>5.1 loginAction.....</b>	<b>9</b>
Funcionalidade.....	9
Importância.....	10
Implementação.....	10
<b>5.2 Medidas de Segurança na Autenticação.....</b>	<b>11</b>
Tratamento de Credenciais Incorretas.....	11
<b>6. Registro de Usuários.....</b>	<b>13</b>
<b>6.1 Método register.....</b>	<b>13</b>
Descrição:.....	14
Explicação do Processo de Registro.....	14
<b>6.2 Método registerAction.....</b>	<b>16</b>
• Explicação do registerAction.....	16
Código:.....	16
<b>7. Estrutura da Página e Funcionalidades Principais:.....</b>	<b>18</b>
<b>7.1 Organização Visual e Usabilidade.....</b>	<b>18</b>
<b>7.2 Documentação das Funcionalidades.....</b>	<b>20</b>

7.3 Script de Ordenação com Sortable.js.....	21
<b>8. Métodos de Edição de Links no AdminController.....</b>	<b>23</b>
8.1 Métodos de Edição de Links.....	23
8.2 editLink(\$slug, \$linkid).....	23
8.3 editLinkAction(\$slug, \$linkid, Request \$request).....	23
8.4. Exemplos de Configuração de Cores.....	24
8.5. Campos Disponíveis para Edição.....	24
<b>9. Método delLink - Funcionalidade Excluir Link.....</b>	<b>28</b>
Funcionamento.....	29
<b>10. Controladores.....</b>	<b>30</b>
10.1 AdminController.....	30
10.2 PageController.....	39
Método index.....	40
Método createPage.....	40
<b>11. Modelos Blade e Estrutura Visual.....</b>	<b>43</b>
11.1 Index.blade.php.....	43
11.2 Visão Geral.....	44
Observações Importantes.....	44
Dependências.....	44
11.3 page_editlink.blade.php.....	45
page_editlink.blade.php.....	45
Visão Geral.....	45
11.4 page_links.blade.php.....	47
page_links.blade.php.....	47
Visão Geral.....	47
Estrutura.....	47
11.5 page.blade.php.....	49
Visão Geral.....	49
11.6 Menu de Navegação Lateral.....	49
11.7 template.blade.php.....	50
Visão Geral.....	50
11.8 notfound.blade.php.....	51
Visão Geral.....	51
<b>12. Considerações Finais.....</b>	<b>52</b>
<b>13. Contato e Suporte.....</b>	<b>53</b>

## 1. Visão Geral

Este projeto é um sistema CRUD (Create, Read, Update, Delete) desenvolvido em Laravel, permitindo que usuários gerenciem links de maneira simples e intuitiva, semelhante ao Linktree.

## 2. Objetivos

- O objetivo deste projeto é desenvolver uma aplicação web que funcione como um serviço de gerenciamento de links, semelhante ao Linktree. A aplicação permitirá que os usuários criem, visualizem, editem e excluam links, proporcionando uma interface intuitiva e responsiva. O foco é facilitar a organização e o compartilhamento de links em um único local, melhorando a acessibilidade e a experiência do usuário.

## 3. Especificações

**Plataforma:** Aplicação web acessível via navegadores.

**Tecnologia:**

- Backend: Laravel (PHP)
- Frontend: Blade (motor de templates do Laravel), HTML, CSS (Bootstrap)
- Banco de Dados: MySQL

**Funcionalidades:**

- Adição de links com título e URL.
- Exibição de todos os links em uma lista.
- Edição de links existentes.
- Exclusão de links.

**Requisitos:**

- PHP 8.0 ou superior.
- Composer instalado.

- Servidor web (por exemplo, Apache ou Nginx) para execução.

**Interface:** Responsiva, utilizando Bootstrap para uma melhor experiência em dispositivos móveis e desktops.

## 4. Documentação das Rotas

Esta documentação descreve as rotas da aplicação Laravel, incluindo rotas principais e rotas administrativas. As rotas são organizadas em dois grupos: rotas públicas e rotas de administração.

### 4.1 Rotas Públicas

#### Rota Principal

- **Método:** GET
- **URL:** /
- **Controlador:** HomeController@index
- **Descrição:** Renderiza a página inicial da aplicação.

### 4.2 Rotas de Administração

#### Grupo de Rotas de Administração

- **Prefixo:** /admin

##### 4.2.1 Autenticação

- **Rota de Login**
  - **Método:** GET
  - **URL:** /admin/login
  - **Controlador:** AdminController@login
  - **Nome da Rota:** login
  - **Descrição:** Exibe o formulário de login para administradores.
- **Rota de Ação de Login**
  - **Método:** POST

- **URL:** `/admin/login`
  - **Controlador:** `AdminController@loginAction`
  - **Descrição:** Processa o envio do formulário de login.
- **Rota de Registro**
  - **Método:** `GET`
  - **URL:** `/admin/register`
  - **Controlador:** `AdminController@register`
  - **Nome da Rota:** `register`
  - **Descrição:** Exibe o formulário de registro para novos administradores.
- **Rota de Ação de Registro**
  - **Método:** `POST`
  - **URL:** `/admin/register`
  - **Controlador:** `AdminController@registerAction`
  - **Descrição:** Processa o envio do formulário de registro.
- **Rota de Logout**
  - **Método:** `GET`
  - **URL:** `/admin/logout`
  - **Controlador:** `AdminController@logout`
  - **Descrição:** Realiza o logout do administrador.

### 4.2.2 Rotas Protegidas

- **Rota do Painel Administrativo**
  - **Método:** `GET`
  - **URL:** `/admin/`
  - **Controlador:** `AdminController@index`
  - **Middleware:** `auth`
  - **Descrição:** Exibe o painel administrativo. Esta rota requer autenticação.

### Gerenciamento de Páginas

- **Links da Página**
  - **Método:** `GET`
  - **URL:** `/admin/{slug}/links`
  - **Controlador:** `AdminController@pageLinks`
  - **Descrição:** Exibe os links da página especificada pelo slug.
- **Design da Página**

- **Método:** GET
  - **URL:** /admin/{slug}/design
  - **Controlador:** AdminController@pageDesign
  - **Descrição:** Exibe a página de design da página especificada pelo slug.
- **Estatísticas da Página**
  - **Método:** GET
  - **URL:** /admin/{slug}/stats
  - **Controlador:** AdminController@pageStats
  - **Descrição:** Exibe as estatísticas da página especificada pelo slug.

## Gerenciamento de Links

- **Atualização da Ordem dos Links**
  - **Método:** GET
  - **URL:** /admin/linkorder/{linkid}/{pos}
  - **Controlador:** AdminController@linkOrderUpdate
  - **Descrição:** Atualiza a ordem dos links.
- **Criar Novo Link**
  - **Método:** GET
  - **URL:** /admin/{slug}/newlink
  - **Controlador:** AdminController@newLink
  - **Descrição:** Exibe o formulário para criar um novo link.
- **Ação de Criar Novo Link**
  - **Método:** POST
  - **URL:** /admin/{slug}/newlink
  - **Controlador:** AdminController@newLinkAction
  - **Descrição:** Processa o envio do formulário para criar um novo link.
- **Editar Link**
  - **Método:** GET
  - **URL:** /admin/{slug}/editlink/{linkid}
  - **Controlador:** AdminController@editLink
  - **Descrição:** Exibe o formulário para editar o link especificado.
- **Ação de Editar Link**
  - **Método:** POST
  - **URL:** /admin/{slug}/editlink/{linkid}
  - **Controlador:** AdminController@editLinkAction

- **Descrição:** Processa o envio do formulário para editar o link.
- **Deletar Link**
  - **Método:** DELETE
  - **URL:** /admin/{slug}/dellink/{linkid}
  - **Controlador:** AdminController@delLink
  - **Descrição:** Deleta o link especificado.

### 4.3 Páginas Dinâmicas

- **Método:** GET
- **URL:** /{slug}
- **Controlador:** PageController@index
- **Descrição:** Renderiza uma página dinâmica com base no slug. Garante que o slug não comece com `admin`.

```

8 // Rota principal
9 Route::get(uri: '/', action: [HomeController::class, 'index']);
10
11 // Prefixo para rotas de admin
12 Route::prefix(prefix: '/admin')->group(callback: function () { void {
13     // Rota de exibição do formulário de login
14     Route::get(uri: '/login', action: [AdminController::class, 'login'])->name(name: 'login');
15     // Rota de envio do formulário de login
16     Route::post(uri: '/login', action: [AdminController::class, 'loginAction']);
17
18     // Rota para exibir o formulário de registro (com nome para a rota)
19     Route::get(uri: '/register', action: [AdminController::class, 'register'])->name(name: 'register');
20     // Rota para enviar o formulário de registro
21     Route::post(uri: '/register', action: [AdminController::class, 'registerAction']);
22
23     Route::get(uri: '/logout', action: [AdminController::class, 'logout']);
24
25     // Rota protegida pelo middleware de autenticação
26     Route::get(uri: '/', action: [AdminController::class, 'index'])->middleware(middleware: 'auth');
27
28     Route::get(uri: '{slug}/links', action: [AdminController::class, 'pageLinks']);
29     Route::get(uri: '{slug}/design', action: [AdminController::class, 'pageDesign']);
30     Route::get(uri: '{slug}/stats', action: [AdminController::class, 'pageStats']);
31
32     Route::get(uri: '/linkorder/{linkid}/{pos}', action: [AdminController::class, 'linkOrderUpdate']);
33
34     Route::get(uri: '{slug}/newlink', action: [AdminController::class, 'newLink']);
35     Route::post(uri: '{slug}/newlink', action: [AdminController::class, 'newLinkAction']);
36
37     Route::get(uri: '{slug}/editlink/{linkid}', action: [AdminController::class, 'editLink']);
38     Route::post(uri: '{slug}/editlink/{linkid}', action: [AdminController::class, 'editLinkAction']);
39
40     // Altere aqui para permitir o método DELETE
41     Route::delete(uri: '{slug}/dellink/{linkid}', action: [AdminController::class, 'delLink']);
42 });
43
44 // Rota para páginas dinâmicas
45 Route::get(uri: '{slug}', action: [PageController::class, 'index'])->where(name: 'slug', expression: '^(?!admin).*'); // Garante que 'admin' não será um slug

```

## 5. Documentação do Login

### Objetivo



O objetivo desta seção é descrever o funcionamento do sistema de login da aplicação, detalhando a lógica implementada no código e a URL de acesso para os usuários.

## Especificações

- **URL de Acesso:** O endpoint para acessar a página de login é `/admin/login`.
- **Métodos:** O sistema suporta dois métodos HTTP para login: `GET` e `POST`. O método `GET` é utilizado para exibir o formulário de login, enquanto o `POST` é utilizado para processar as credenciais fornecidas pelo usuário.

```
1 reference | 0 overrides
public function login(Request $request): Factory|Redirector|RedirectResponse|View
{
    // Método GET: exibe a página de login
    if ($request->isMethod(method: 'get')) {
        return view(view: 'admin/login');
    }

    // Regras de validação
    $rules = [
        'email' => 'required|email',
        'password' => 'required|min:7'
    ];

    // Mensagens de erro personalizadas
    $messages = [
        'email.required' => 'O campo e-mail é obrigatório.',
        'email.email' => 'Informe um e-mail válido.',
        'password.required' => 'O campo senha é obrigatório.',
        'password.min' => 'A senha deve ter pelo menos 7 caracteres.'
    ];

    // Validação das credenciais
    $validator = Validator::make(data: $request->all(), rules: $rules, messages: $messages);

    if ($validator->fails()) {
        return redirect()->back()->withErrors(provider: $validator)->withInput();
    }

    // Tentativa de autenticação
    if (Auth::attempt(credentials: ['email' => $request->email, 'password' => $request->password])) {
        return redirect(to: '/admin');
    }

    return redirect()->back()->with(key: 'error', value: 'E-mail e/ou senha não conferem.');
```

## 5.1 loginAction

### Funcionalidade

O método `loginAction` é responsável por gerenciar a autenticação de usuários no sistema. Ele processa as credenciais (e-mail e senha) fornecidas pelo usuário e tenta autenticar o acesso ao painel administrativo.

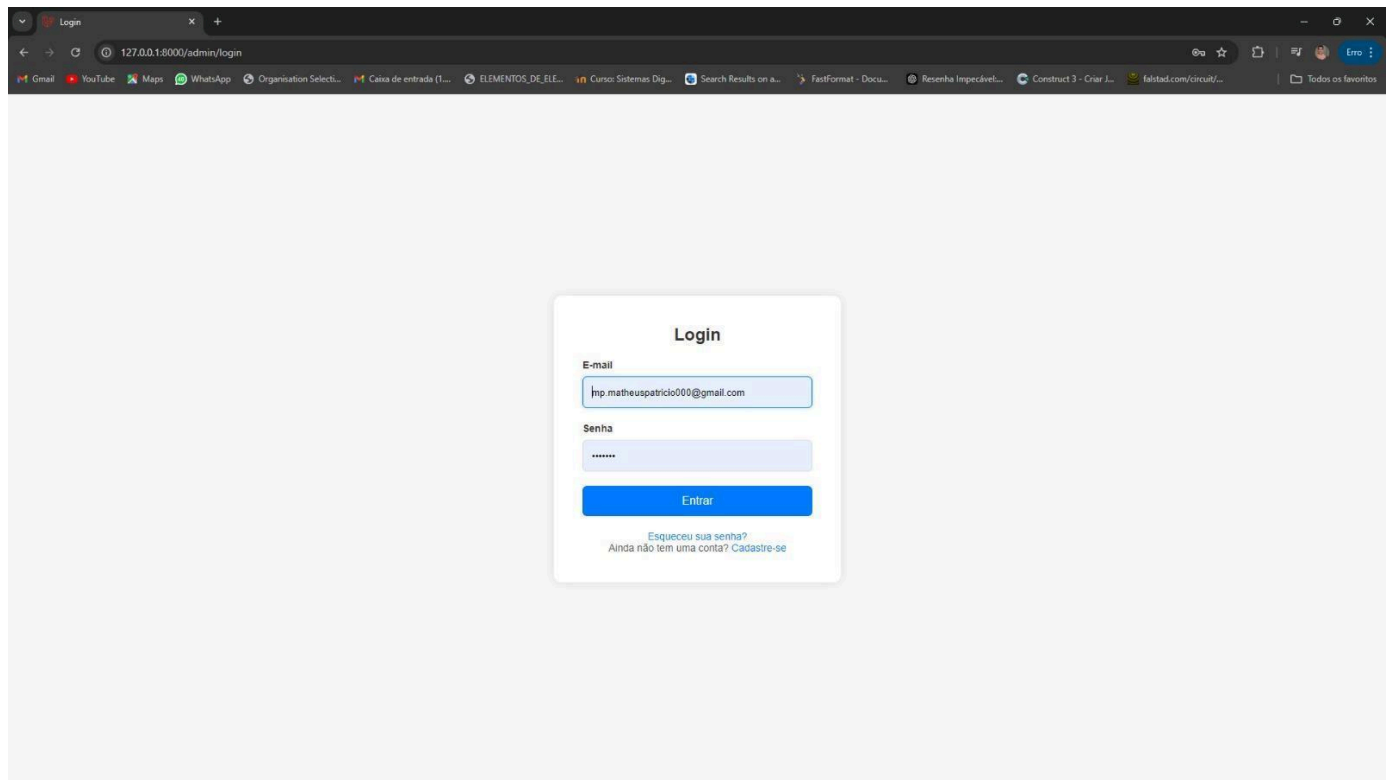
## Importância

- **Segurança:** Garante que apenas usuários autenticados tenham acesso à área administrativa, protegendo informações sensíveis.
- **Validação de Credenciais:** Valida as credenciais do usuário e fornece feedback adequado em caso de falhas, como e-mail ou senha incorretos.
- **Redirecionamento:** Após a autenticação bem-sucedida, redireciona o usuário para o painel administrativo, melhorando a experiência do usuário.

## Implementação

1. **Coleta de Credenciais:** Recebe as credenciais do usuário a partir da requisição.
2. **Autenticação:** Utiliza o método `Auth::attempt` para validar as credenciais.
3. **Redirecionamento:**
  - Se a autenticação for bem-sucedida, o usuário é redirecionado para `/admin`.
  - Se falhar, o usuário é redirecionado de volta para a página de login com uma mensagem de erro.

```
1 reference | 0 overrides
58 public function loginAction(Request $request): mixed|Redirector|RedirectResponse
59 {
60     $creds = $request->only(keys: 'email', 'password');
61
62     if (Auth::attempt(credentials: $creds)) {
63         return redirect(to: '/admin');
64     } else {
65         return redirect(to: '/admin/login')->with(key: 'error', value: 'E-mail e/ou senha não conferem.');
```



## 5.2 Medidas de Segurança na Autenticação

### Tratamento de Credenciais Incorretas

- **Mensagens de Erro:**
  - Mensagens de erro são exibidas ao usuário quando o login falha.

```

26      <!-- Exibir mensagem de erro do login -->
27      @if (session('error'))
28          <div class="error">
29              <p>{{ session('error') }}</p>
30          </div>
31      @endif

```

- **Validação de Entrada:**
  - O sistema verifica se os campos obrigatórios (e-mail e senha) estão preenchidos, evitando dados vazios.

```
15      <!-- Exibir erros de validação -->
16      @if ($errors->any())
17          <div class="error">
18              <ul>
19                  @foreach ($errors->all() as $error)
20                      <li>{{ $error }}</li>
21                  @endforeach
22              </ul>
23          </div>
24      @endif
```

- **Proteção contra Força Bruta:**

- Implementação de bloqueio temporário após várias tentativas de login incorretas.

- **Token CSRF:**

- Uso de token CSRF (@csrf) no formulário para proteger contra ataques de Cross-Site Request Forgery.

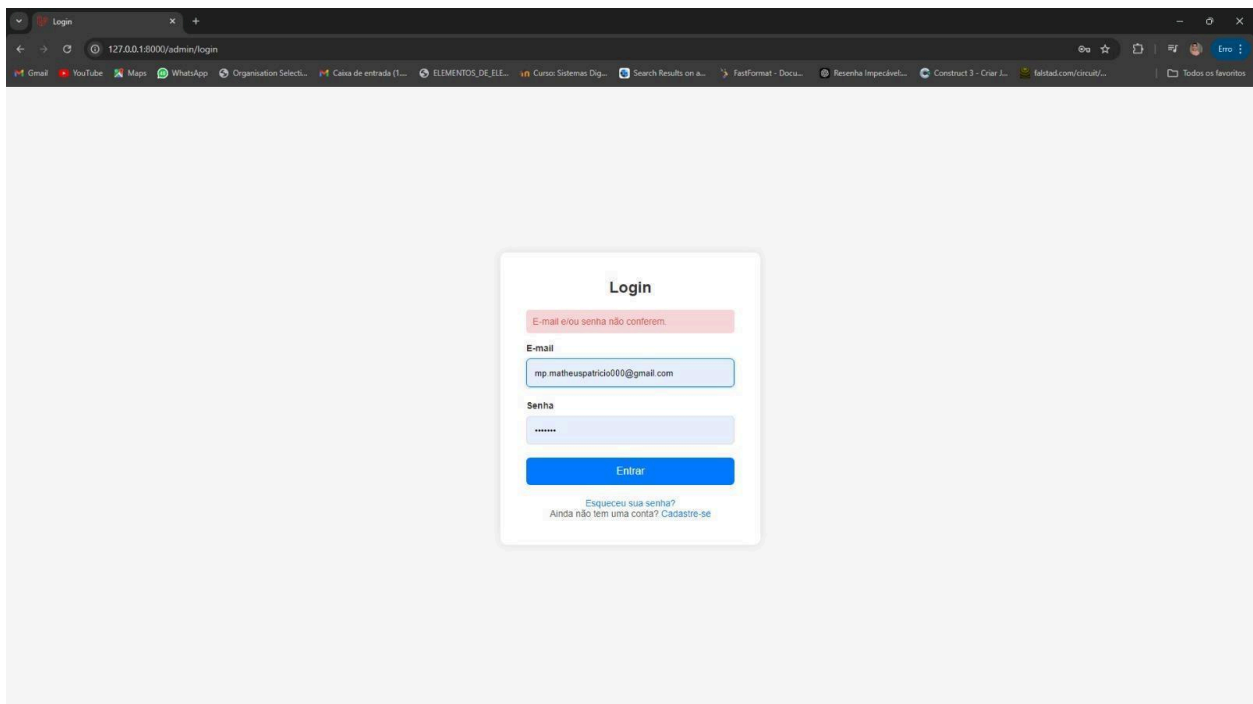
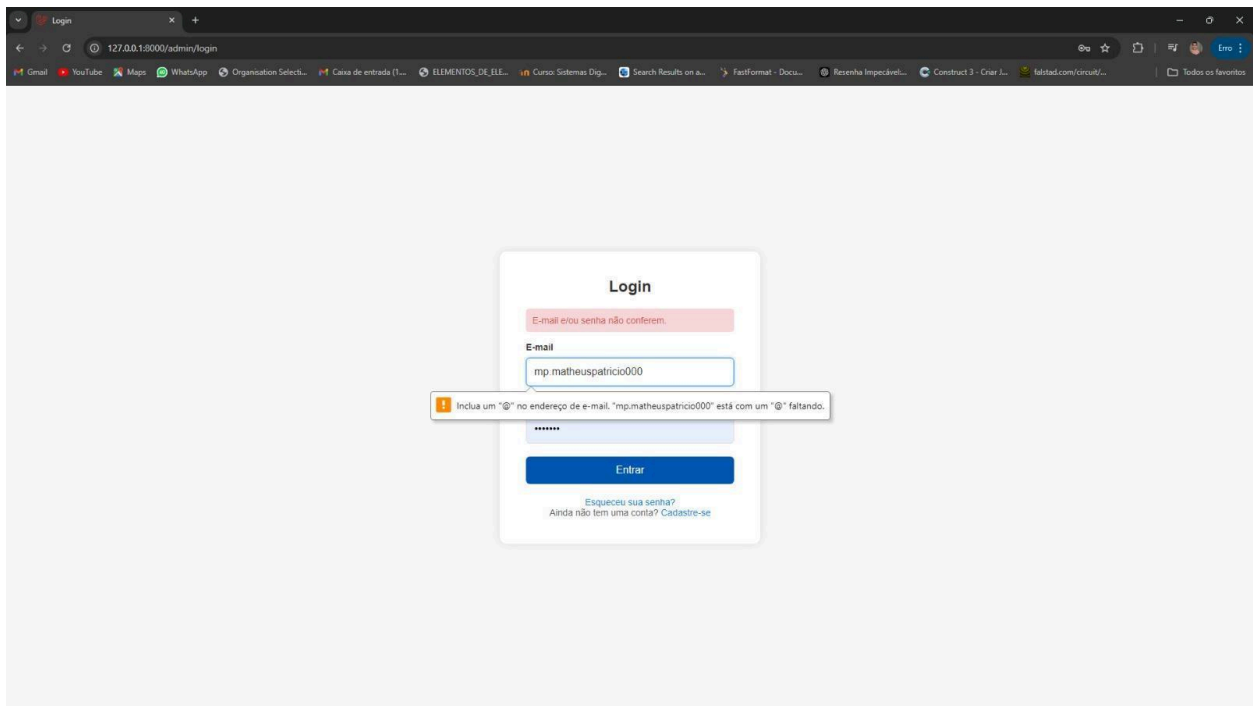
```
33      <!-- Formulário de Login -->
34      <form action="{{ route('login') }}" method="POST">
35          @csrf <!-- validando que é um forms -->
36
```

- **Armazenamento Seguro de Senhas:**

- Senhas são armazenadas usando técnicas de hash, como o **bcrypt**.

- **Redirecionamento Seguro:**

- Após o login bem-sucedido, o usuário é redirecionado para uma página segura.



## 6. Registro de Usuários

### 6.1 Método register

Este método exibe a página de registro para requisições do tipo GET e processa o registro do usuário para requisições do tipo POST.

### Descrição:

- **GET:** Exibe a view `admin/register` com o formulário de registro.
- **POST:** Realiza a validação dos dados e, em caso de sucesso, cria um novo usuário no banco de dados, logando-o automaticamente após o registro.

## Explicação do Processo de Registro

- **Exibição do Formulário de Registro:**
  - Caso a requisição seja do tipo **GET**, o método retorna a view `admin/register`, onde o formulário de registro é exibido para o usuário.

- **Validação de Dados:**

O método define regras e mensagens de erro personalizadas para validar os campos de entrada:

- **name:** Obrigatório, tipo string e com um tamanho máximo de 255 caracteres.
- **email:** Obrigatório, em formato de e-mail e único na tabela `users`.
- **password:** Obrigatório, com um tamanho mínimo de 7 caracteres, e que deve ser confirmado.

Em caso de erro de validação, o usuário é redirecionado de volta ao formulário com os erros exibidos.

### Criação do Usuário:

- Se os dados são válidos, cria-se uma instância do modelo `User`:
- **name**, **email** e **password** são atribuídos aos atributos do usuário.
- A senha é criptografada utilizando `Hash::make` antes de ser salva.

Após a criação do usuário, o sistema faz o login automático e redireciona para a página `/admin`.

### Código:

```

19  class AdminController extends Controller
69      public function register(Request $request): Factory|Redirector|RedirectResponse|View
70  {
71      // Exibe o formulário de registro
72      if ($request->isMethod(method: 'get')) {
73          return view(view: 'admin/register');
74      }
75
76      // Regras de validação
77      $rules = [
78          'name' => 'required|string|max:255',
79          'email' => 'required|email|unique:users,email',
80          'password' => 'required|min:7|confirmed'
81      ];
82
83      // Mensagens de erro personalizadas
84      $messages = [
85          'name.required' => 'O campo nome é obrigatório.',
86          'email.required' => 'O campo e-mail é obrigatório.',
87          'email.email' => 'Informe um e-mail válido.',
88          'email.unique' => 'Este e-mail já está em uso.',
89          'password.required' => 'O campo senha é obrigatório.',
90          'password.min' => 'A senha deve ter pelo menos 7 caracteres.',
91          'password.confirmed' => 'As senhas não conferem.'
92      ];
93
94      // Validação dos dados
95      $validator = Validator::make(data: $request->all(), rules: $rules, messages: $messages);
96
97      if ($validator->fails()) {
98          return redirect()->back()->withErrors(provider: $validator)->withInput();
99      }
100
101      // Criação do novo usuário
102      $user = new User();
103      $user->name = $request->name;
104      $user->email = $request->email;
105      $user->password = Hash::make(value: $request->password);
106      $user->save();
107
108      // Login automático após o registro
109      Auth::login(user: $user);
110
111      return redirect(to: '/admin');
112  }

```

## 6.2 Método registerAction

Este método alternativo de registro verifica se o e-mail já existe antes de cadastrar um novo usuário. Além disso, ele cria uma página associada ao novo usuário.

- **Explicação do registerAction**
- **Verificação de Existência de E-mail:**
  - O método `registerAction` primeiro verifica se o e-mail fornecido já existe na tabela `users`.
  - Caso o e-mail já exista, o usuário é redirecionado para a página de registro com uma mensagem de erro.
- **Criação do Usuário e Página Associada:**
  - Se o e-mail é único, o método cria um novo usuário e uma **página personalizada** para ele.
  - Esta página associada recebe:
    - `slug`: Gerado automaticamente com o ID do usuário.
    - Configurações padrão, como `op_title`, `op_font_color`, `op_bg_type`, `op_bg_value`, `op_profile_image` e `op_description`.
  - Após salvar a página, o usuário é logado automaticamente.
- **Redirecionamento Final:**
  - Após o cadastro e login, o sistema redireciona o usuário para `/admin` com uma mensagem de sucesso.

**Código:**



```

114 1 reference | 0 overrides
115 public function registerAction(Request $request): mixed|RedirectResponse
116 {
117     $creds = $request->only(keys: 'email', 'password', 'name');
118
119     // Verifica se o email já existe
120     $hasEmail = User::where(column: 'email', operator: $creds['email'])->count();
121
122     if ($hasEmail === 0) {
123         // Criação do novo usuário
124         $newUser = new User();
125         $newUser->name = $creds['name'];
126         $newUser->email = $creds['email'];
127         $newUser->password = Hash::make(value: $creds['password']);
128         $newUser->save();
129
130         // Criar uma nova página associada ao usuário
131         $page = new Page();
132         $page->id_user = $newUser->id; // ID do novo usuário
133         $page->slug = 'pagina-inicial-' . $newUser->id; // Slug para a nova página
134         $page->op_title = 'Minha Página Inicial'; // Título padrão
135         $page->op_font_color = '#000000'; // Cor da fonte padrão
136         $page->op_bg_type = 'solid'; // Tipo de fundo padrão
137         $page->op_bg_value = '#FFFFFF'; // Cor de fundo padrão
138         $page->op_profile_image = 'default_profile_image.jpg'; // Imagem de perfil padrão
139         $page->op_description = 'Descrição padrão da página.'; // Descrição padrão
140         $page->save(); // Salvar a nova página
141
142         Auth::login(user: $newUser); // Logar o novo usuário
143
144         return redirect(to: '/admin')->with(key: 'success', value: 'Usuário cadastrado com sucesso!');
145     } else {
146         return redirect(to: '/admin/register')->with(key: 'error', value: 'Já existe um usuário com este e-mail.');
```

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/admin/register'. The page contains a registration form titled 'Cadastro' with the following fields and elements:

- Nome:** A text input field containing 'EduTec2024'.
- E-mail:** A text input field containing 'edutec@gmail.com'.
- Senha:** A password input field with masked characters '\*\*\*\*\*'.
- Confirme a Senha:** A confirmation password input field with masked characters '\*\*\*\*\*'.
- Cadastrar:** A blue button with white text.
- Footer:** A small text link that says 'Esqueceu sua senha? Já tem cadastro? Faça login'.

## 7. Estrutura da Página e Funcionalidades Principais:

- **Página Principal do Usuário**

- Ao fazer login, o usuário é redirecionado para o painel principal (`/admin`).
- Aqui, ele pode visualizar informações principais, incluindo suas páginas associadas, com destaque para o layout organizado e limpo.

- **Gerenciamento de Links**

- A página inclui a opção para o usuário visualizar, adicionar, editar e excluir links associados a uma página específica, acessada através do `slug` da página.
- Funções como `pageLinks` listam todos os links para a página correspondente do usuário, enquanto `newLink` e `editLink` permitem que ele adicione ou modifique links específicos.

- **Botões de Navegação e Ações**

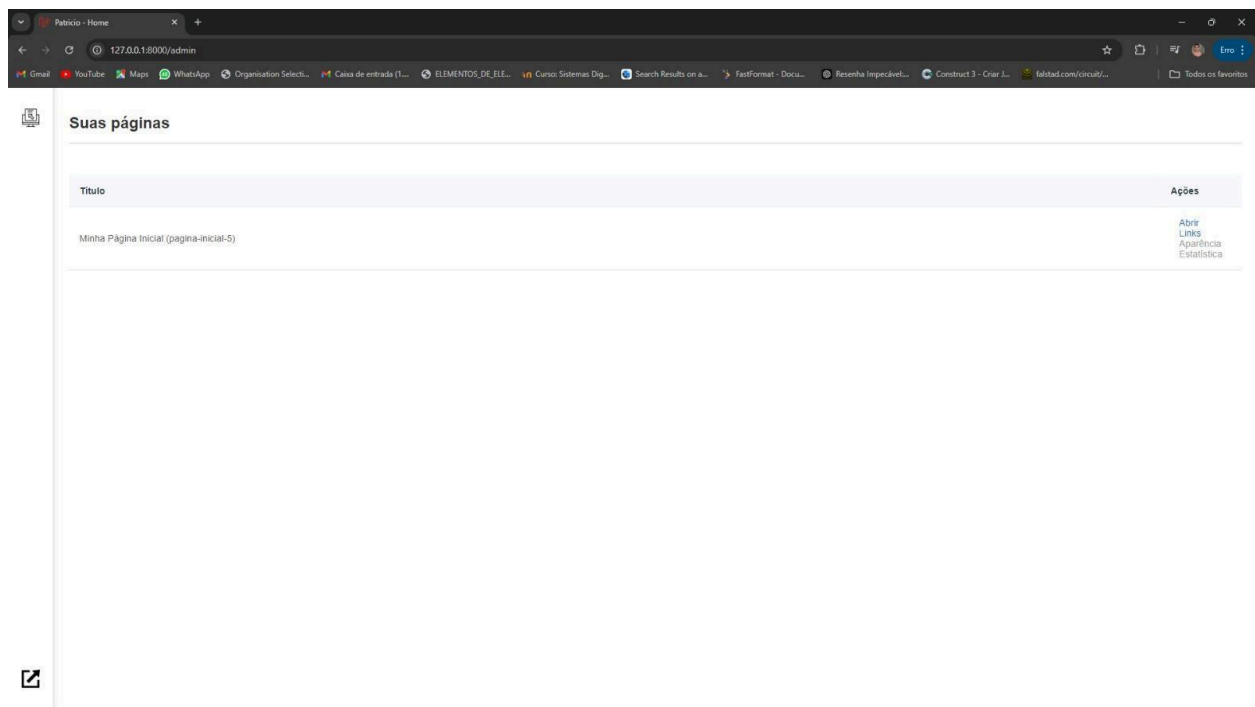
- **Logout:** Um botão de logout permite que o usuário encerre a sessão e seja redirecionado para a tela de login.
- **Botões Aparência e Estatísticas:** Embora desabilitados, esses botões são exibidos em tom cinza claro para indicar futuras funcionalidades. Eles já têm a estrutura de lógica implementada, mas as interfaces associadas a eles ainda não foram totalmente desenvolvidas.
- **Aparência:** Planejado para permitir que o usuário customize o design de suas páginas (cores, estilos de fundo, entre outros).
- **Estatísticas:** Dará ao usuário a capacidade de monitorar e analisar dados de visualização de suas páginas, possibilitando uma gestão mais estratégica.

### 7.1 Organização Visual e Usabilidade

- A página foi desenhada para oferecer uma interface organizada e funcional, seguindo o padrão visual de um painel administrativo. O layout também apresenta

uma boa hierarquia visual, com botões intuitivos e uma área de navegação que guia o usuário nas principais ações de forma simples e eficaz.

- A escolha por exibir botões desabilitados sugere ao usuário as atualizações que estarão disponíveis em breve, indicando que o sistema é expansível e moderno.





## 7.2 Documentação das Funcionalidades

- **Botão "Novo Link":**
  - Localizado no topo da lista, este botão permite criar um novo link para a página do usuário. Ao ser clicado, direciona para a página de criação de um link novo.
- **Lista de Links:**
  - Cada link é representado com as seguintes informações:
    - **Título e URL** do link.
    - Ícone para arrastar e reordenar.
    - Botões de ação:
      - **Editar:** Abre o formulário para edição completa do link selecionado.
      - **Excluir:** Remove o link com confirmação de exclusão.
- **Reordenamento de Links:**
  - A funcionalidade de arrastar e soltar (*drag-and-drop*) é utilizada para alterar a ordem dos links diretamente na interface.

- **Persistência da Ordem:** Embora a biblioteca atualize visualmente a ordem, a aplicação garante que a nova ordem seja refletida no banco de dados através de uma requisição assíncrona (AJAX). A rota `/admin/linkorder` recebe o novo índice de cada link e ajusta as posições no banco.
- **Validação e Segurança:**
  - Cada alteração de ordem é protegida com um *token* CSRF para garantir a segurança da requisição.
  - Ao término do reordenamento, a página é recarregada para refletir a nova ordem visual.

### 7.3 Script de Ordenação com *Sortable.js*

O script *Sortable* fornece um efeito de animação ao arrastar os itens, facilitando a interação. A função *onEnd* garante que:

- Cada alteração de ordem envia uma solicitação para atualizar a posição do link no servidor.
- Em caso de sucesso, a interface é recarregada para confirmar visualmente a alteração.

Esse sistema foi desenvolvido para garantir que a ordem dos links seja mantida conforme estabelecida pelo usuário, evitando discrepâncias entre o que é exibido e o que está salvo no banco de dados.

Patricio - Minha Página Inicial - x +





127.0.0.1:8000/admin/pagina-inicial-1/links

Gmail YouTube Maps WhatsApp Organisation Selecti... Caixa de entrada (1... ELEMENTOS\_DE\_ELE... >> Todos os favoritos

Página: Minha Página Inicial

Links Aparência Estatísticas

Novo Link

	02 https://mail.google....	Editar	Excluir
	01 https://www.youtub...	Editar	Excluir
	04 https://www.youtub...	Editar	Excluir
	03 https://www.youtub...	Editar	Excluir

800 x 200

120 x 120

**Minha Página Inicial**

Descrição padrão da página.

**Meus Links**

02

04

Novo Link

	02 https://mail.google.com/mail/u/0/?tab=rm&ogbl#inbox	Editar	Excluir
	01 https://www.youtube.com/	Editar	Excluir
	04 https://www.youtube.com/	Editar	Excluir
	03 https://www.youtube.com/	Editar	Excluir



## 8. Métodos de Edição de Links no AdminController

A parte de edição de links no `AdminController` permite ao usuário alterar as configurações dos links associados a uma página específica, incluindo cor de fundo, cor do texto, tipo de borda, entre outros. Essa funcionalidade utiliza métodos HTTP `GET` e `POST` para exibir e processar o formulário de edição dos links.

### 8.1 Métodos de Edição de Links

#### 8.2 `editLink($slug, $linkid)`

Exibe o formulário de edição para um link específico associado a uma página.

- **Parâmetros:**
  - `$slug`: Identificador único da página.
  - `$linkid`: ID do link a ser editado.
- **Lógica:**
  - Verifica se a página e o link associados existem para o usuário autenticado.
  - Exibe a view `admin/page_editlink`, passando os dados da página e do link.

#### 8.3 `editLinkAction($slug, $linkid, Request $request)`

Processa o formulário de edição de um link.

- **Parâmetros:**
  - `$slug`: Slug da página.
  - `$linkid`: ID do link a ser editado.
  - `$request`: Contém os dados submetidos do formulário.
- **Validações:**
  - `status`: Booleano que indica se o link está ativo ou inativo.
  - `title`: String, título do link com no mínimo 2 caracteres.
  - `href`: URL válida para o link.
  - `op_bg_color`: Cor de fundo no formato hexadecimal (#RRGGBB ou #RGB).
  - `op_text_color`: Cor do texto no formato hexadecimal.
  - `op_border_type`: Tipo de borda, aceitando `square` (quadrada) ou `rounded` (arredondada).
  - `label`: Campo opcional para uma etiqueta de texto.
- **Lógica:**
  - Busca o link para edição e aplica os campos validados.
  - Salva as alterações no link.
  - Redireciona o usuário para a página de links.

## 8.4. Exemplos de Configuração de Cores

Para alterar a cor de fundo e a cor do texto de um link, utilize valores hexadecimais, como mostrado nos exemplos abaixo:

- **Cor de Fundo (`op_bg_color`):** #FFFFFF (branco) ou #000000 (preto)
- **Cor do Texto (`op_text_color`):** #FF5733 (vermelho alaranjado) ou #333333 (cinza escuro)

**Nota:** A validação hex é aplicada para assegurar que somente valores válidos de cor hexadecimal sejam aceitos.

## 8.5. Campos Disponíveis para Edição

- **Título (`title`):** Nome exibido para o link.
- **URL (`href`):** Endereço para o qual o link redireciona.
- **Cor de Fundo (`op_bg_color`):** Configuração visual do link.
- **Cor do Texto (`op_text_color`):** Ajuste da cor da fonte do link.





- **Tipo de Borda (`op_border_type`):** Formato da borda; aceita valores `square` ou `rounded`.

## Novo Link

Status:

Ativado

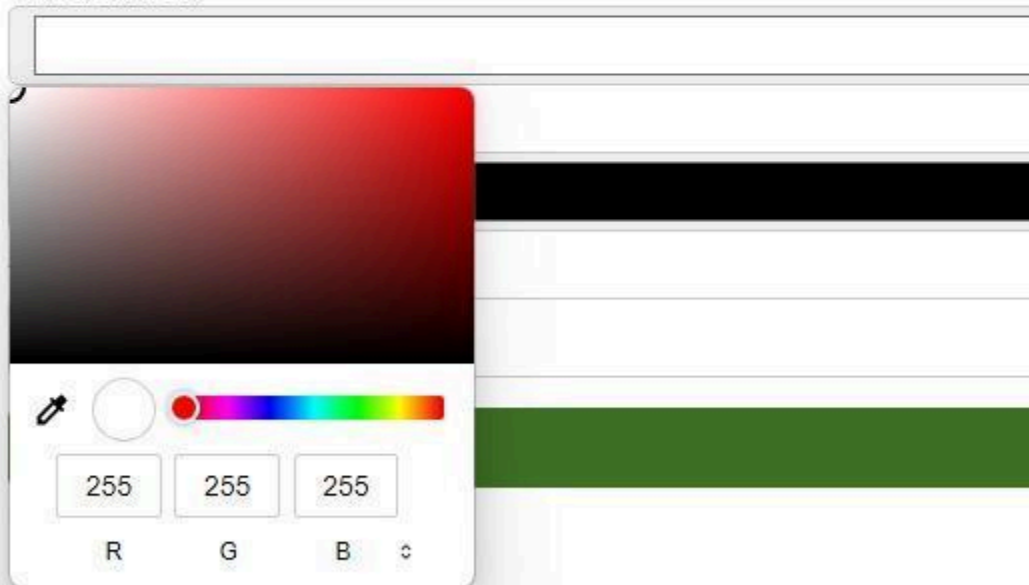
Título do link:

Google

Url do link:

<https://mail.google.com/mail/u/0/?tab=rm&ogbl#inbox>

Cor do fundo:



## Editar Link

- The title field must be at least 2 characters.

Status:

Ativado

Título do link:

oioioioio

Url do link:

<https://www.youtube.com/>

Cor do fundo:



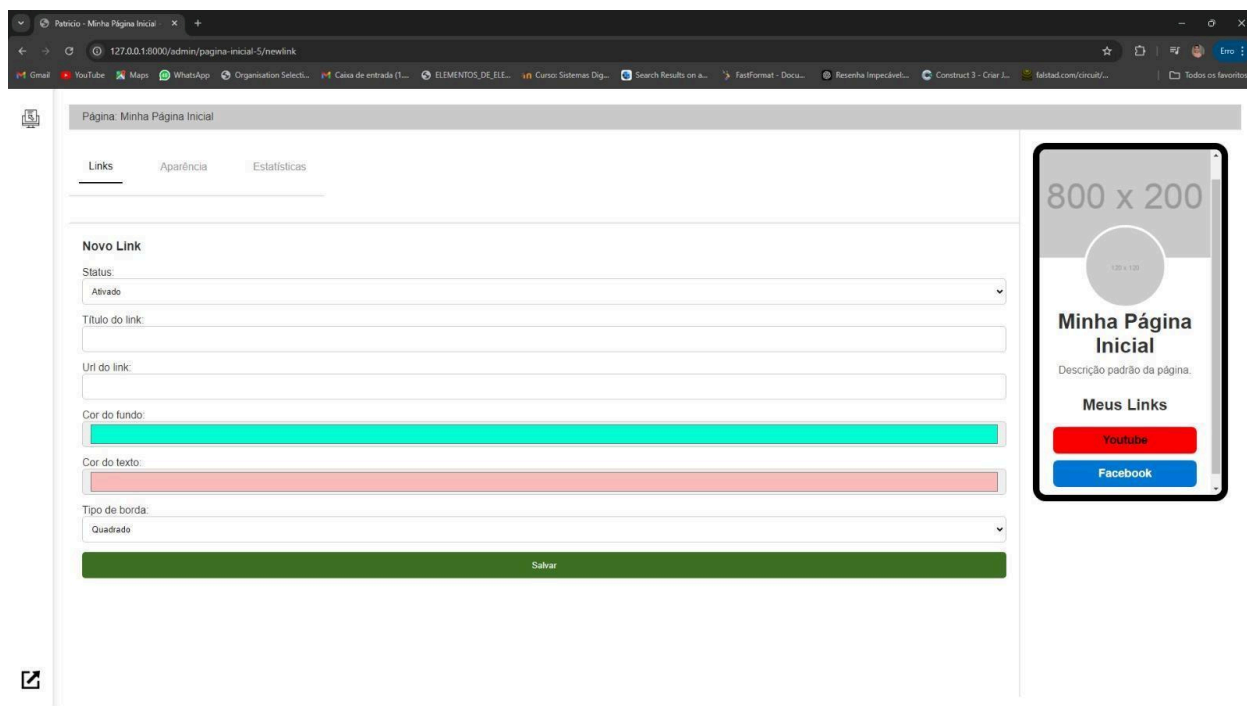
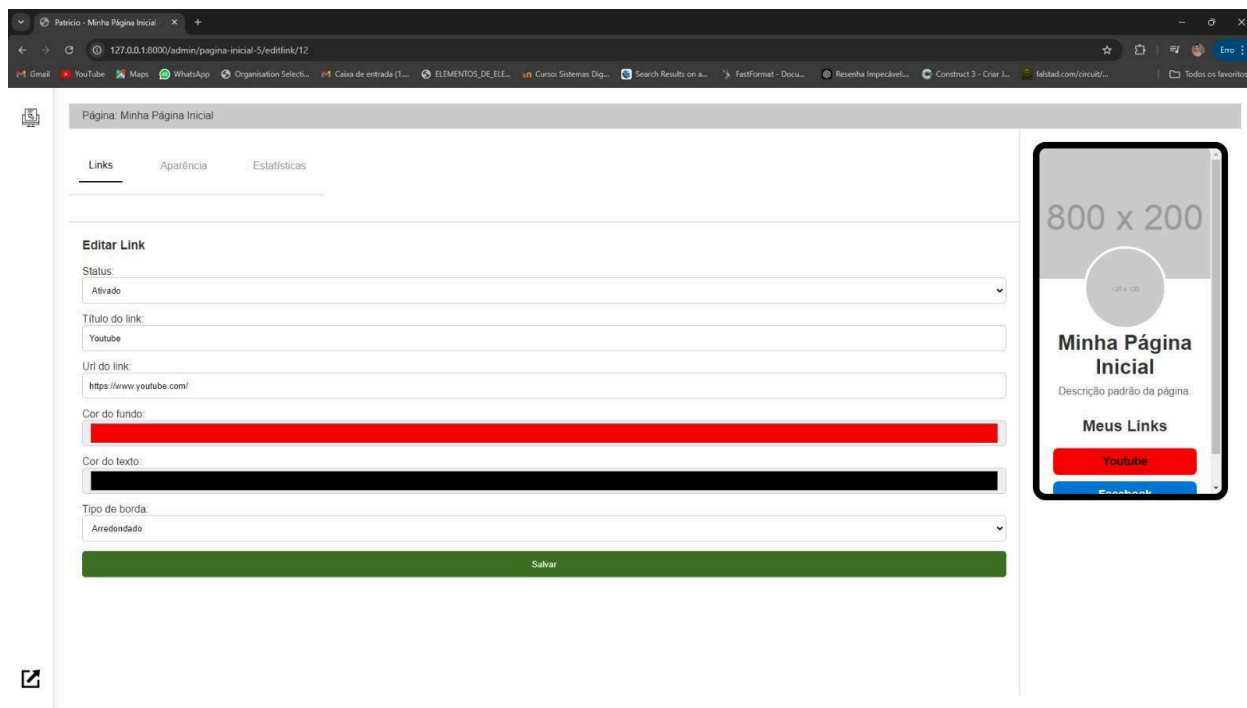
Cor do texto:



Tipo de borda:

Arredondado





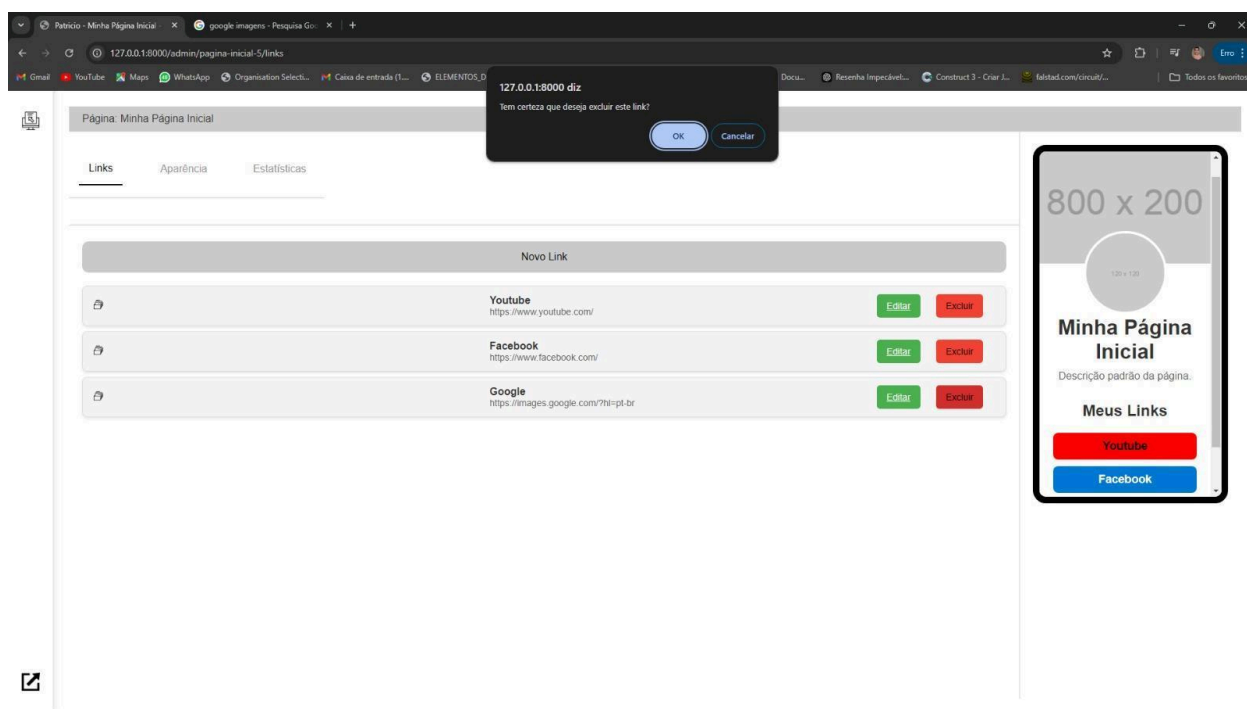
## 9. Método de Link - Funcionalidade Excluir Link

O método `delLink` do `AdminController` é responsável por excluir um link específico associado a uma página do usuário autenticado. Essa exclusão é feita diretamente no

banco de dados, removendo permanentemente o link. Além de apagar o link, o método reordena os links restantes para manter uma sequência de exibição correta.

## Funcionamento

- **Busca da Página:** Através do **slug**, o método localiza a página associada ao usuário autenticado.
- **Verificação do Link:** Verifica a existência do link específico (**linkid**) relacionado a essa página.
- **Exclusão do Link:** Se o link existir, ele é deletado do banco de dados.
- **Reordenação dos Links Restantes:** Após a exclusão, o método ajusta a ordem dos links restantes, garantindo que a sequência continue consistente. Ele reatribui a posição (**order**) de cada link na página para evitar buracos na sequência de exibição.



```

1 reference | 0 overrides
377 public function dellink($slug, $linkid): Redirector\RedirectResponse
378 {
379     $user = Auth::user();
380     $page = Page::where(column: 'id_user', operator: $user->id)
381         ->where(column: 'slug', operator: $slug)
382         ->first();
383
384     // Verifica se a página existe
385     if (!$page) {
386         return redirect(to: '/admin');
387     }
388
389     // Agora busca o link associado à página
390     $link = Link::where(column: 'id_page', operator: $page->id)
391         ->where(column: 'id', operator: $linkid)
392         ->first();
393
394     // Verifica se o link existe
395     if ($link) {
396         $link->delete();
397         // Corrigindo as posições
398         $allLinks = Link::where(column: 'id_page', operator: $page->id)
399             ->orderBy(column: 'order', direction: 'ASC')
400             ->get();
401         foreach ($allLinks as $linkKey => $linkItem) {
402             $linkItem->order = $linkKey;
403             $linkItem->save();
404         }
405         return redirect(to: './admin/' . $page->slug . '/links');
406     }
407
408     // Se o link não for encontrado, redireciona para a área inicial
409     return redirect(to: '/admin');
410 }
411
412
413

```

## 10. Controladores

### 10.1 AdminController

O **AdminController** é o controlador responsável por gerenciar as principais funcionalidades administrativas da aplicação. Ele engloba operações relacionadas à autenticação de usuários, manipulação de links personalizados e gestão de páginas associadas aos usuários registrados. Suas funções incluem:

- **Autenticação de Usuários:** realiza o login e logout dos administradores, além de exibir a interface de login e registro.
- **Gerenciamento de Usuários:** valida e cadastra novos usuários, incluindo a criação de uma página inicial padrão para cada usuário recém-registrado.

- **Gestão de Páginas e Links:** exibe páginas e links associados aos usuários, permite adicionar, editar, reordenar e excluir links personalizados, utilizando uma biblioteca de drag-and-drop para reordenar visualmente e salvar no banco de dados.
- **Configurações de Design e Estatísticas:** inclui as interfaces para personalização de design da página e visualização de estatísticas (botões atualmente desativados e indicados como funcionalidades futuras).

O **AdminController** assim organiza e facilita a administração de conteúdos e interações, oferecendo um ponto central de controle e navegação para usuários autenticados.

```

19 class AdminController extends Controller
20 {
21     /**
22      * Exibe a tela de login e autentica o usuário.
23      * - Se o método for GET, renderiza a página de login.
24      * - Se for POST, valida o formulário e realiza a autenticação.
25      */
26     public function login(Request $request): Factory|Redirector|RedirectResponse|View
27     {
28         // Verifica o método HTTP e exibe a view de login
29         if ($request->isMethod(method: 'get')) {
30             return view(view: 'admin/login');
31         }
32
33         // Define regras e mensagens de validação para login
34         $rules = [
35             'email' => 'required|email',
36             'password' => 'required|min:7'
37         ];
38
39         $messages = [
40             'email.required' => 'O campo e-mail é obrigatório.',
41             'email.email' => 'Informe um e-mail válido.',
42             'password.required' => 'O campo senha é obrigatório.',
43             'password.min' => 'A senha deve ter pelo menos 7 caracteres.'
44         ];
45
46         // Redireciona de volta caso haja erros de validação
47         $validator = Validator::make(data: $request->all(), rules: $rules, messages: $messages);
48
49         if ($validator->fails()) {
50             return redirect()->back()->withErrors(provider: $validator)->withInput();
51         }
52
53         // Tenta autenticar o usuário
54         if (Auth::attempt(credentials: ['email' => $request->email, 'password' => $request->password])) {
55             return redirect(to: '/admin');
56         }
57
58         // Redireciona com mensagem de erro caso a autenticação falhe
59         return redirect()->back()->with(key: 'error', value: 'E-mail e/ou senha não conferem.');
```

```

74  /**
75   * Exibe a tela de registro e cadastra um novo usuário.
76   * - Se o método for GET, renderiza a página de registro.
77   * - Se for POST, valida o formulário e registra o novo usuário.
78   */
79  1 reference | 0 overrides
80  public function register(Request $request): Factory|Redirector|RedirectResponse|View
81  {
82      if ($request->isMethod(method: 'get')) {
83          return view(view: 'admin/register');
84      }
85      // Define regras e mensagens de validação para registro
86      $rules = [
87          'name' => 'required|string|max:255',
88          'email' => 'required|email|unique:users,email',
89          'password' => 'required|min:7|confirmed'
90      ];
91
92      $messages = [
93          'name.required' => 'O campo nome é obrigatório.',
94          'email.required' => 'O campo e-mail é obrigatório.',
95          'email.email' => 'Informe um e-mail válido.',
96          'email.unique' => 'Este e-mail já está em uso.',
97          'password.required' => 'O campo senha é obrigatório.',
98          'password.min' => 'A senha deve ter pelo menos 7 caracteres.',
99          'password.confirmed' => 'As senhas não conferem.'
100      ];
101
102      $validator = Validator::make(data: $request->all(), rules: $rules, messages: $messages);
103
104      if ($validator->fails()) {
105          return redirect()->back()->withErrors(provider: $validator)->withInput();
106      }
107      // Cria e salva o novo usuário
108      $user = new User();
109      $user->name = $request->name;
110      $user->email = $request->email;
111      $user->password = Hash::make(value: $request->password);
112      $user->save();
113
114      Auth::login(user: $user);
115
116      return redirect(to: '/admin');
117  }

```



```

118 1 reference | 0 overrides
119 public function registerAction(Request $request): mixed|RedirectResponse
120 {
121     $creds = $request->only(keys: 'email', 'password', 'name');
122
123     // Verifica se o email já existe
124     $hasEmail = User::where(column: 'email', operator: $creds['email'])->count();
125
126     if ($hasEmail === 0) {
127         // Criação do novo usuário
128         $newUser = new User();
129         $newUser->name = $creds['name'];
130         $newUser->email = $creds['email'];
131         $newUser->password = Hash::make(value: $creds['password']);
132         $newUser->save();
133
134         // Criar uma nova página associada ao usuário
135         $page = new Page();
136         $page->id_user = $newUser->id; // ID do novo usuário
137         $page->slug = 'pagina-inicial-' . $newUser->id; // Slug para a nova página
138         $page->op_title = 'Minha Página Inicial'; // Título padrão
139         $page->op_font_color = '#000000'; // Cor da fonte padrão
140         $page->op_bg_type = 'solid'; // Tipo de fundo padrão
141         $page->op_bg_value = '#FFFFFF'; // Cor de fundo padrão
142         $page->op_profile_image = 'default_profile_image.jpg'; // Imagem de perfil padrão
143         $page->op_description = 'Descrição padrão da página.'; // Descrição padrão
144         $page->save(); // Salvar a nova página
145
146         Auth::login(user: $newUser); // Logar o novo usuário
147
148         return redirect(to: '/admin')->with(key: 'success', value: 'Usuário cadastrado com sucesso!');
149     } else {
150         return redirect(to: '/admin/register')->with(key: 'error', value: 'Já existe um usuário com este e-mail.');
```

```

152 /**
153  * Exibe a tela inicial do administrador com as páginas associadas ao usuário autenticado.
154  */
155 1 reference | 0 overrides
156 public function index(): Factory|View
157 {
158     $user = Auth::user();
159
160     $pages = Page::where(column: 'id_user', operator: $user->id)->get();
161
162     return view(view: 'admin.index', data: [
163         'pages' => $pages
164     ]);
165 }
166 /**
167  * Realiza o logout do usuário e redireciona para a página de login.
168  */
169 1 reference | 0 overrides
170 public function logout(): Redirector|RedirectResponse
171 {
172     Auth::logout();
173     return redirect(to: '/admin');
```

```

174 public function pageLinks($slug): Factory|Redirector|RedirectResponse|View
175 {
176     $user = Auth::user();
177     $page = Page::where(column: 'slug', operator: $slug)
178         ->where(column: 'id_user', operator: $user->id)
179         ->first();
180
181     if ($page) {
182         // Obtendo os links associados à página
183         $links = DB::table(table: 'links')->where(column: 'id_page', operator: $page->id)->get();
184
185         return view(view: 'admin/page_links', data: [
186             'menu' => 'links',
187             'page' => $page,
188             'links' => $links, // Adicionando a variável $links à view
189         ]);
190     } else {
191         return redirect(to: '/admin');
192     }
193 }
194
195 1 reference | 0 overrides
196 public function linkOrderUpdate($linkid, $pos): array
197 {
198     $user = Auth::user();
199     $link = Link::find(id: $linkid);
200
201     $myPages = Page::where(column: 'id_user', operator: $user->id)->pluck(column: 'id')->toArray();
202
203     if (in_array(needle: $link->id_page, haystack: $myPages)) {
204         if ($link->order > $pos) {
205             // Subiu o item - jogando os próximos para baixo
206             $afterLinks = Link::where(column: 'id_page', operator: $link->id_page)
207                 ->where(column: 'order', operator: '>=', value: $pos)
208                 ->get();
209             foreach ($afterLinks as $afterLink) {
210                 $afterLink->order++;
211                 $afterLink->save();
212             }
213         } elseif ($link->order < $pos) {
214             // Desceu o item - jogando os anteriores para cima
215             $beforeLinks = Link::where(column: 'id_page', operator: $link->id_page)
216                 ->where(column: 'order', operator: '<=', value: $pos)
217                 ->get();
218             foreach ($beforeLinks as $beforeLink) {
219                 $beforeLink->order--; // Corrigido para beforeLink
220                 $beforeLink->save();
221             }
222         }
223     }

```

```
223         // Posicionando o item
224         $link->order = $pos;
225         $link->save();
226
227         // Corrigindo as posições
228         $alllinks = Link::where(column: 'id_page', operator: $link->id_page)
229             ->orderBy(column: 'order', direction: 'ASC')
230             ->get();
231         foreach ($alllinks as $linkKey => $linkItem) {
232             $linkItem->order = $linkKey;
233             $linkItem->save();
234         }
235     }
236
237     return [];
238 }
239
```

```

257 public function newLinkAction($slug, Request $request): Redirector|RedirectResponse
258 {
259     $user = Auth::user();
260     $page = Page::where(column: 'id_user', operator: $user->id)
261         ->where(column: 'slug', operator: $slug)
262         ->first();
263
264     if ($page) {
265         $fields = $request->validate(rules: [
266             'status' => ['required', 'boolean'],
267             'title' => ['required', 'min:2'],
268             'href' => ['required', 'url'],
269             'op_bg_color' => ['required', 'regex:/^[#][0-9A-F]{3,6}$/i'],
270             'op_text_color' => ['required', 'regex:/^[#][0-9A-F]{3,6}$/i'],
271             'op_border_type' => ['required', Rule::in(values: ['square', 'rounded'])],
272             'label' => ['nullable', 'string']
273         ]);
274
275         $newLink = new Link();
276         $newLink->id_page = $page->id;
277         $newLink->status = $fields['status'];
278         $newLink->border = 1;
279         $newLink->title = $fields['title'];
280         $newLink->href = $fields['href'];
281         $newLink->op_bg_color = $fields['op_bg_color'];
282         $newLink->op_text_color = $fields['op_text_color'];
283         $newLink->op_border_type = $fields['op_border_type'];
284         $newLink->id_user = $user->id;
285         $newLink->url = $fields['href'];
286         $newLink->label = $fields['label'] ?? 'Default Label';
287
288         // Define o campo `order` com base no próximo valor disponível
289         $newLink->order = Link::where(column: 'id_page', operator: $page->id)->max(column: 'order') + 1;
290
291         $newLink->save();
292
293         return redirect(to: './admin/' . $page->slug . '/links');
294     } else {
295         return redirect(to: '/admin');
296     }
297 }
298

```

```

1 reference | 0 overrides
299 public function editLink($slug, $linkid): Factory|Redirector|RedirectResponse|View
300 {
301     $user = Auth::user();
302     $page = Page::where(column: 'id_user', operator: $user->id)
303         ->where(column: 'slug', operator: $slug)
304         ->first();
305
306     // Verifica se a página existe
307     if (!$page) {
308         return redirect(to: '/admin');
309     }
310
311     // Agora busca o link associado à página
312     $link = Link::where(column: 'id_page', operator: $page->id)
313         ->where(column: 'id', operator: $linkid)
314         ->first();
315
316     // Verifica se o link existe
317     if ($link) {
318         return view(view: 'admin/page_editlink', data: [
319             'menu' => 'links', // Corrigido para 'links'
320             'page' => $page,
321             'link' => $link
322         ]);
323     }
324
325     // Se o link não for encontrado, redireciona para a área inicial
326     return redirect(to: '/admin');
327 }
328

```

```

329 public function editLinkAction($slug, $linkid, Request $request): Redirector|RedirectResponse
330 {
331
332     $user = Auth::user();
333     $page = Page::where(column: 'id_user', operator: $user->id)
334         ->where(column: 'slug', operator: $slug)
335         ->first();
336
337     // Verifica se a página existe
338     if (!$page) {
339         return redirect(to: '/admin');
340     }
341
342     // Agora busca o link associado à página
343     $link = Link::where(column: 'id_page', operator: $page->id)
344         ->where(column: 'id', operator: $linkid)
345         ->first();
346
347     // Verifica se o link existe
348     if ($link) {
349
350         $fields = $request->validate(rules: [
351             'status' => ['required', 'boolean'],
352             'title' => ['required', 'min:2'],
353             'href' => ['required', 'url'],
354             'op_bg_color' => ['required', 'regex:/^[#][0-9A-F]{3,6}$/i'],
355             'op_text_color' => ['required', 'regex:/^[#][0-9A-F]{3,6}$/i'],
356             'op_border_type' => ['required', 'Rule::in(values: ['square', 'rounded'])],
357             'label' => ['nullable', 'string']
358         ]);
359
360         $link->status = $fields['status'];
361         $link->title = $fields['title'];
362         $link->href = $fields['href'];
363         $link->op_bg_color = $fields['op_bg_color'];
364         $link->op_text_color = $fields['op_text_color'];
365         $link->op_border_type = $fields['op_border_type'];
366         $link->save();
367
368         return redirect(to: './admin/' . $page->slug . '/links');
369     }
370
371
372
373     // Se o link não for encontrado, redireciona para a área inicial
374     return redirect(to: '/admin');
375 }
376

```

```

377 public function delLink($slug, $linkid): Redirector\RedirectResponse
378 {
379     $user = Auth::user();
380     $page = Page::where(column: 'id_user', operator: $user->id)
381         ->where(column: 'slug', operator: $slug)
382         ->first();
383
384     // Verifica se a página existe
385     if (!$page) {
386         return redirect(to: '/admin');
387     }
388
389     // Agora busca o link associado à página
390     $link = Link::where(column: 'id_page', operator: $page->id)
391         ->where(column: 'id', operator: $linkid)
392         ->first();
393
394     // Verifica se o link existe
395     if ($link) {
396         $link->delete();
397         // Corrigindo as posições
398         $allLinks = Link::where(column: 'id_page', operator: $page->id)
399             ->orderBy(column: 'order', direction: 'ASC')
400             ->get();
401         foreach ($allLinks as $linkKey => $linkItem) {
402             $linkItem->order = $linkKey;
403             $linkItem->save();
404         }
405         return redirect(to: '../admin/' . $page->slug . '/links');
406     }
407
408     // Se o link não for encontrado, redireciona para a área inicial
409     return redirect(to: '/admin');
410 }
411
412
413 1 reference | 0 overrides
414 public function pageDesign($slug): Factory|View
415 {
416     return view(view: 'admin/page_design', data: [
417         'menu' => 'design'
418     ]);
419 }
420

```

```

421 1 reference | 0 overrides
422 public function pageStats($slug): Factory|View
423 {
424     return view(view: 'admin/page_stats', data: [
425         'menu' => 'stats'
426     ]);
427 }
428
429

```

## 10.2 PageController

O **PageController** é responsável por gerenciar a exibição e criação de páginas de perfil no sistema, permitindo que cada usuário visualize e configure suas páginas personalizadas. Abaixo estão os métodos e funcionalidades principais desse controlador.

---

## Método **index**

- **Descrição:** Exibe uma página existente com base no **slug** fornecido na URL.
  - **Funcionamento:**
    - Primeiro, busca uma página no banco de dados com o **slug** especificado.
    - Se a página for encontrada, define o **background** da página com base nas configurações do usuário:
      - **Imagem:** utiliza uma URL para exibir a imagem de fundo.
      - **Cor:** define uma cor ou gradiente linear.
    - Recupera e exibe os links ativos associados a essa página, ordenados pela posição configurada.
    - Registra uma visualização no banco de dados para a página exibida, armazenando a data e incrementando o contador.
  - **Retorno:** Renderiza a página de visualização com as informações configuradas ou exibe uma página "não encontrada" se o **slug** não existir.
  - **Variáveis Disponíveis na View:**
    - **font\_color, profile\_image, title, description, fb\_pixel, bg, e links.**
- 

## Método **createPage**

- **Descrição:** Cria uma nova página personalizada associada ao usuário autenticado.
- **Funcionamento:**
  - Obtém o usuário atualmente autenticado no sistema.
  - Cria um novo registro na tabela **Page** e associa o **id\_user** ao ID do usuário logado.
  - Preenche os campos da página com valores enviados pelo usuário e aplica padrões onde necessário (como o tipo de fundo ou cor).
  - Salva a nova página no banco de dados.



- **Retorno:** Redireciona para a área administrativa com uma mensagem de sucesso.

---

Esses métodos juntos proporcionam ao usuário a funcionalidade de criar e visualizar páginas personalizadas, com controle de design e links personalizados para sua exibição.

```

app > Http > Controllers > PageController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\Auth; // Importa Auth para pegar o usuário autenticado
7  use App\Models\Page;
8  use App\Models\Link;
9  use App\Models\View;
10
11  2 references | 0 implementations
12  class PageController extends Controller
13  {
14      // Método de exibição de página existente
15      1 reference | 0 overrides
16      public function index($slug): Factory|View
17      {
18          $page = Page::where(column: 'slug', operator: $slug)->first();
19
20          if ($page) {
21              // Background
22              $bg = '#ffffff';
23              switch ($page->op_bg_type) {
24                  case 'image':
25                      $bg = "url(' . url(path: '/media/uploads') . '/' . $page->op_bg_value . "')";
26                      break;
27                  case 'color':
28                      $colors = explode(separator: ',', string: $page->op_bg_value);
29                      $bg = 'linear-gradient(90deg, ' . $colors[0] . ',' . $colors[1] . ')';
30                      $bg .= !empty($colors[1]) ? $colors[1] : $colors[0];
31                      $bg .= ')';
32                      break;
33              }
34
35              // Links
36              $links = Link::where(column: 'id_page', operator: $page->id)
37                  ->where(column: 'status', operator: 1)
38                  ->orderBy(column: 'order')
39                  ->get();
40
41              // Registrar a visualização
42              $view = View::firstOrCreate(
43                  attributes: ['id_page' => $page->id, 'view_date' => date(format: 'Y-m-d')]
44              );
45              $view->total = ($view->exists ? $view->total : 0) + 1;
46              $view->save();
47
48              return view(view: 'page', data: [
49                  'font_color' => $page->op_font_color,

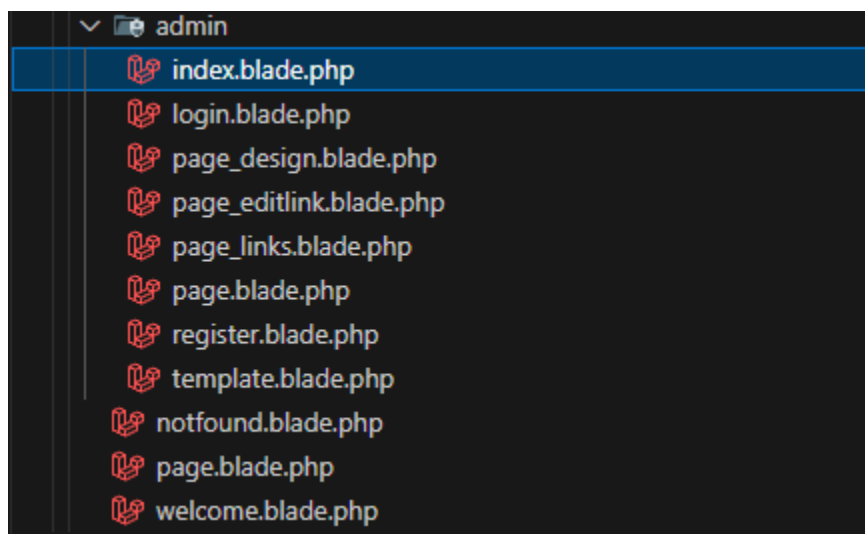
```

```

47         return view(view: 'page', data: [
48             'font_color' => $page->op_font_color,
49             'profile_image' => $page->op_profile_image ? url(path: '/media/uploads') . '/' . $page->op_profile_image : null,
50             'title' => $page->op_title,
51             'description' => $page->op_description ?? 'Sem descrição disponível',
52             'fb_pixel' => $page->op_fb_pixel ?? null,
53             'bg' => $bg,
54             'links' => $links,
55         ]);
56     } else {
57         return view(view: 'notfound');
58     }
59 }
60
61 // Método para criar uma nova página associada ao usuário logado
62 0 references | 0 overrides
63 public function createPage(Request $request): mixed|RedirectResponse
64 {
65     $user = Auth::user(); // Obtém o usuário autenticado
66
67     $page = new Page();
68     $page->id_user = $user->id; // Define o id_user como o ID do usuário logado
69     $page->op_title = $request->input(key: 'op_title');
70     $page->slug = $request->input(key: 'slug');
71     $page->op_bg_type = $request->input(key: 'op_bg_type', default: 'color');
72     $page->op_bg_value = $request->input(key: 'op_bg_value', default: '#ffffff');
73     $page->op_font_color = $request->input(key: 'op_font_color', default: '#000000');
74     $page->op_description = $request->input(key: 'op_description', default: 'Descrição padrão');
75
76     $page->save(); // Salva a nova página no banco
77
78     return redirect(to: '/admin')->with(key: 'success', value: 'Página criada com sucesso!');
79 }
80
81

```

## 11. Modelos Blade e Estrutura Visual



### 11.1 Index.blade.php

## 11.2 Visão Geral

A página `index.blade.php` é uma página de listagem das páginas gerenciáveis pelo usuário no sistema de administração. Ela utiliza o template `admin.template`.

Dentro da seção `content`, a página apresenta um cabeçalho com o título "Suas páginas" e uma tabela listando as páginas do usuário. Para cada página, exibe o título e uma série de links para acessar e gerenciar a página.

- **Cabeçalho da Tabela**
  - Coluna **Título**: Exibe o título da página.
  - Coluna **Ações**: Fornece uma largura fixa de 20px para exibir as ações disponíveis para cada página.
- **Corpo da Tabela**
  - Cada linha da tabela representa uma página e contém:
    - **Título da Página**: Nome da página com o título `$page->op_title`, seguido pelo `slug`.
    - **Ações**:
      - **Abrir**: Link para abrir a página em uma nova aba.
      - **Links**: Link para gerenciar os links associados à página.
      - **Aparência**: Link desativado (classe "disabled") para acessar o design da página.
      - **Estatística**: Link desativado para acessar as estatísticas da página.

### Observações Importantes

- **Links Desativados**: Os links "Aparência" e "Estatística" têm a classe `disabled`, indicando que essas funcionalidades podem estar indisponíveis.
- **Iteração de Dados**: A exibição das páginas é feita usando o loop `@foreach`, que percorre a coleção `$pages`.
- **URLs Dinâmicas**: Os URLs são construídos dinamicamente com o uso de `{{ url() }}`, com base no `slug` de cada página para permitir acesso e gerenciamento específicos.

### Dependências

Para o correto funcionamento dessa página, é necessário:

- Um layout `admin.template`.
- A coleção `$pages`, que deve ser passada do controlador para a view, contendo o título (`op_title`) e `slug` de cada página.

```

1  @extends(view: 'admin.template')
2
3  @section(section: 'title', content: 'Patricio - Home')
4
5  @section (section: 'content')
6
7  <header>
8  |   <h2>Suas páginas</h2>
9  </header>
10
11 <table>
12 |   <thead>
13 |     <tr>
14 |       <th>Título</th>
15 |       <th width="20px">Ações</th>
16 |     </tr>
17 |   </thead>
18 |   <tbody>
19 |     @foreach ($pages as $page)
20 |       <tr>
21 |         <td>{{ $page->op_title }} ({{ $page->slug }})</td>
22 |         <td>
23 |           <a href="{{ url(path: '/' . $page->slug) }}" target="_blank">Abrir</a>
24 |           <a href="{{ url(path: '/admin/' . $page->slug . '/links') }}">Links</a>
25 |           <a href="{{ url(path: '/admin/' . $page->slug . '/design') }}" class="disabled">Aparência</a>
26 |           <a href="{{ url(path: '/admin/' . $page->slug . '/stats') }}" class="disabled">Estatística</a>
27 |         </td>
28 |       </tr>
29 |     @endforeach
30 |   </tbody>
31 </table>
32
33 @endsection
34

```

## 11.3 page\_editlink.blade.php

### page\_editlink.blade.php

#### Visão Geral

Página para criação e edição de links no painel administrativo, utilizando o template `admin.page`. Exibe um formulário para configurar atributos do link, como status, título, URL, cores e tipo de borda.

- **Cabeçalho:** Mostra "Editar Link" se `$link` estiver definido; caso contrário, "Novo Link".
- **Exibição de Erros:** Exibe uma lista de erros de validação, se houver.
- **Formulário:**
  - **Status:** Define se o link está ativado ou desativado.
  - **Título:** Título do link.
  - **URL:** Endereço do link.
  - **Cor do Fundo/Text:** Escolha de cores.
  - **Tipo de Borda:** Opções de borda quadrada ou arredondada.

Formulário protegido por CSRF (`@csrf`) para segurança.

---

Esta página centraliza a criação e edição dos links com campos básicos de configuração.

```

14
15 <form method="POST">
16     @csrf
17     <label>
18         Status:<br />
19         <select name="status">
20             <option {{isset($link) ? ($link->status == '1' ? 'selected' : '') : ''}} value="1">Ativado</option>
21             <option {{isset($link) ? ($link->status == '0' ? 'selected' : '') : ''}} value="0">Desativado</option>
22         </select>
23     </label>
24
25     <label>
26         Título do link: <br />
27         <input type="text" name="title" value="{{isset($link->title) ? $link->title : ''}}"/>
28     </label>
29
30     <label>
31         Url do link: <br />
32         <input type="text" name="href" value="{{isset($link->href) ? $link->href : ''}}"/>
33     </label>
34
35     <label>
36         Cor do fundo: <br />
37         <input type="color" name="op_bg_color" value="{{isset($link->op_bg_color) ? $link->op_bg_color : '#FFFFFF'}}"/>
38     </label>
39
40     <label>
41         Cor do texto: <br />
42         <input type="color" name="op_text_color" value="{{isset($link->op_text_color) ? $link->op_text_color : '#000000'}}"/>
43     </label>
44
45     <label>
46         Tipo de borda:<br />
47         <select name="op_border_type">
48             <option {{isset($link) ? ($link->op_border_type == 'square' ? 'selected' : '') : ''}} value="square">Quadrado
49             </option>
50             <option {{isset($link) ? ($link->op_border_type == 'rounded' ? 'selected' : '') : ''}} value="rounded">
51                 Arredondado</option>
52         </select>
53     </label>
54
55     <label>
56         <input type="submit" value="Salvar" />
57     </label>
58 </form>
59 @endsection

```

## 11.4 page\_links.blade.php

### page\_links.blade.php

#### Visão Geral

Página administrativa para listar, criar, editar, excluir e reordenar links associados a uma página específica.

#### Estrutura

#### Template

php



```
@extends('admin.page')
```

- Usa o layout `admin.page` para consistência visual no painel.
- **Botão "Novo Link"** Botão para adicionar um novo link, direcionando para a página de criação.
- **Listagem de Links** Cada link é listado em um item (`li`) com:
  - Ícone para arrastar e reordenar.
  - Informações do link (título e URL).
  - Botões de ação:
    - **Editar:** Abre a página de edição do link.
    - **Excluir:** Remove o link com confirmação.
- **Reordenação Dinâmica** Utiliza **Sortable.js** para permitir reordenação por "arrastar e soltar", enviando a nova ordem ao servidor via JavaScript e AJAX.

---

Essa página oferece uma interface completa para gerenciamento e ordenação de links.



```

1  @extends(view: 'admin.page')
2
3  @section(section: 'body')
4  <a class="bigbutton" href="{{ url(path: './admin/' . $page->slug . '/newlink') }}">Novo Link</a>
5
6  <ul id="links">
7      @foreach ($links as $link)
8          <li class="link--item" data-id="{{ $link->id }}">
9              <div class="link--item-order">
10                 
11             </div>
12             <div class="link--item-info">
13                 <div class="link--item-title">{{ $link->title }}</div>
14                 <div class="link--item-href">{{ $link->href }}</div>
15             </div>
16             <div class="link--item-buttons">
17                 <a href="{{ url(path: './admin/' . $page->slug . '/editlink/' . $link->id) }}" class="button-edit">Editar</a>
18                 <form action="{{ url(path: './admin/' . $page->slug . '/dellink/' . $link->id) }}" method="POST" style="display:inline;">
19                     @csrf
20                     @method('DELETE')
21
22                     <button type="submit" class="button-delete" onclick="return confirm('Tem certeza que deseja excluir este link?')">Excluir</button>
23                 </form>
24             </div>
25         </li>
26     @endforeach
27 </ul>
28
29 <script src="https://cdn.jsdelivr.net/npm/sortablejs@latest/Sortable.min.js"></script>
30 <script>
31     document.addEventListener("DOMContentLoaded", function() {
32         new Sortable(document.querySelector("#links"), {
33             animation: 150,
34             onEnd: async (e) => {
35                 let id = e.item.getAttribute('data-id');
36                 let link = "{{ url(path: './admin/linkorder') }}/${id}/${e.newIndex}";
37
38                 try {
39                     let response = await fetch(link, { method: 'POST', headers: { 'X-CSRF-TOKEN': '{{ csrf_token() }}' } });
40                     if (response.ok) {
41                         location.reload();
42                     } else {
43                         console.error('Erro ao atualizar a ordem do link');
44                     }
45                 } catch (error) {
46                     console.error('Erro ao se conectar ao servidor:', error);
47                 }
48             }
49         });
50     });
51 </script>

```

## 11.5 page.blade.php

### Visão Geral

O arquivo `page_links.blade.php` é uma interface de administração no sistema, projetada para gerenciar links específicos de uma página. O layout da página oferece um painel com navegação lateral para acessar diferentes seções (Links, Aparência e Estatísticas) e uma área de visualização em tempo real da página selecionada.

## 11.6 Menu de Navegação Lateral

O menu de navegação lateral permite ao usuário alternar entre as seções de Links, Aparência e Estatísticas da página. Cada item de menu usa uma lógica de estado ativo

(**active**) e desativado (**disabled**) para mostrar visualmente ao usuário a seção que está acessando e desabilitar as seções que não estão ativas.

```
resources > views > admin > page.blade.php
1  @extends(view: 'admin.template')
2
3  @section(section: 'title', content: 'Patricio - ' . $page->op_title . ' - Links')
4
5  @section(section: 'content')
6
7  <div class="preheader">
8      Página: {{$page->op_title}}
9  </div>
10
11 <div class="area">
12     <div class="leftside">
13         <header>
14             <header>
15                 <ul>
16                     <li @if ($menu == 'links') class="active" @endif><a
17                         href="{{url(path: '../admin/' . $page->slug . '/links')}}">Links</a></li>
18                     <li @if ($menu == 'design') class="active disabled" @endif><a
19                         href="{{url(path: '../admin/' . $page->slug . '/design')}}>Aparência</a></li>
20                     <li @if ($menu == 'stats') class="active disabled" @endif><a
21                         href="{{url(path: '../admin/' . $page->slug . '/stats')}}>Estatísticas</a></li>
22                 </ul>
23             </header>
24         </header>
25         @yield(section: 'body')
26     </div>
27     <div class="rightside">
28         <iframe frameborder="0" src="{{url(path: '../' . $page->slug)}}"></iframe>
29     </div>
30 </div>
31
32
33 @endsection
```

## 11.7 template.blade.php

### Visão Geral

Este arquivo (**template.blade.php**) é o layout principal para as páginas do painel administrativo. Ele define a estrutura HTML básica e inclui uma navegação superior com links para a página inicial do painel e para logout. Além disso, permite a inclusão dinâmica de conteúdo e títulos de página, mantendo o layout consistente em todas as páginas do painel.

```
resources > views > admin >  template.blade.php
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link rel="stylesheet" href="{{ url(path: 'assets/css/admin.template.css') }}">
7      <title>@yield(section: 'title')</title>
8  </head>
9  <body>
10     <nav>
11         <div class="va--top">
12             <a href="{{ url(path: '/admin') }}">
13                 
14             </a>
15         </div>
16
17         <div class="nav--button">
18             <a href="{{ url(path: '/admin/logout') }}">
19                 
20             </a>
21         </div>
22     </nav>
23     <section class="container">
24         @yield(section: 'content')
25     </section>
26
27 </body>
28 </html>
29
30
```

## 11.8 notfound.blade.php

### Visão Geral

O arquivo `notfound.blade.php` é uma página de erro personalizada que informa ao usuário que a página solicitada não foi encontrada (404). Esta página é projetada para ser simples e amigável, proporcionando ao usuário uma experiência clara e um link de retorno à página inicial.


```

resources > views > notfound.blade.php
1  <!-- resources/views/notfound.blade.php -->
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Página Não Encontrada</title>
8      <style>
9          body {
10             font-family: Arial, sans-serif;
11             background-color: #f8f9fa;
12             text-align: center;
13             padding: 50px;
14         }
15         h1 {
16             font-size: 2.5rem;
17             color: #333;
18         }
19         p {
20             font-size: 1.2rem;
21             color: #666;
22         }
23         a {
24             display: inline-block;
25             margin-top: 20px;
26             text-decoration: none;
27             color: #007bff;
28             border: 1px solid #007bff;
29             padding: 10px 20px;
30             border-radius: 5px;
31             transition: background-color 0.3s, color 0.3s;
32         }
33         a:hover {
34             background-color: #007bff;
35             color: #fff;
36         }
37     </style>
38 </head>
39 <body>
40     <h2>Página não encontrada</h2>
41     <p>Desculpe, a página que você está tentando acessar não existe ou foi removida.</p>
42     <a href="{{ url(path: '/') }}">Voltar para a Página Inicial</a>
43 </body>
44 </html>
45

```

## 12. Considerações Finais

Este projeto CRUD em Laravel foi desenvolvido com o intuito de oferecer uma plataforma simples e eficiente para o gerenciamento de links. A arquitetura em Laravel, juntamente com as boas práticas de segurança e a interface amigável,



tornam esta aplicação robusta e acessível. O projeto também está preparado para futuras melhorias, como estatísticas e customizações de design.

## 13. Contato e Suporte

Para dúvidas, sugestões ou suporte técnico, entre em contato:

**Matheus Patricio**

Email: [matheuspatricio.aspx@gmail.com](mailto:matheuspatricio.aspx@gmail.com)

LinkedIn: <https://www.linkedin.com/in/matheus-patricio-ab77031b6/>