

Roteamento Dinâmico PHP



Crie um Router e classes adicionais para lidar com um arquivo web.php com rotas dinâmicas como esse:

Use OOP, boas práticas, namespace, etc composer .env etc

```
Route::get('/', [HomeController::class, 'index']);
Route::get('/category/list/{nameCategory}', [HomeController::class, 'list']);
Route::get('/blog/', [HomeController::class, 'listBlogAllPosts']);
Route::get('/blog/show/{title}/{id}', [HomeController::class, 'show']);
Route::get('/admin/users', [UserController::class, 'index'])->middleware('admin') -
>name('admin.users');
Route::get('/profile/edit', [ProfileController::class, 'edit'])
->middleware(['auth', 'verified'])
->name('profile.edit');
Route::post('/profile/update', [ProfileController::class, 'update'])
->middleware(['auth', 'verified'])
->name('profile.update');
```



Aqui está um exemplo de implementação de um sistema de roteamento dinâmico usando PHP, princípios de OOP, namespaces, boas práticas, e ferramentas como `composer` e `.env`. Este projeto suporta rotas configuradas em um arquivo `web.php` e oferece suporte a middlewares, controladores e métodos associados.

Estrutura de Arquivos

plaintext

```
project/
├── app/
│   ├── Controllers/
│   │   ├── HomeController.php
│   │   ├── UserController.php
│   │   └── ProfileController.php
│   ├── Middlewares/
│   │   ├── AdminMiddleware.php
│   │   ├── AuthMiddleware.php
│   │   └── VerifiedMiddleware.php
│   └── Router/
│       ├── Router.php
│       ├── Route.php
│       └── MiddlewareHandler.php
├── public/
│   └── index.php
├── routes/
│   └── web.php
├── vendor/
└── .env
```

```
└─ composer.json
└─ bootstrap.php
```

Arquivo `composer.json`

```
json

{
    "autoload": {
        "psr-4": {
            "App\\": "app/"
        }
    },
    "require": {
        "vlucas/phpdotenv": "^5.5"
    }
}
```

Arquivo `.env`

```
plaintext

APP_ENV=local
APP_DEBUG=true
APP_URL=http://localhost
```

Arquivo `bootstrap.php`

```
php

<?php
require_once __DIR__ . '/vendor/autoload.php';

use Dotenv\Dotenv;

$dotenv = Dotenv::createImmutable(__DIR__);
$dotenv->load();
```

Arquivo `routes/web.php`

```
php

<?php

use App\Controllers\HomeController;
use App\Controllers\UserController;
use App\Controllers\ProfileController;

return [
    ['method' => 'GET', 'uri' => '/', 'action' => [HomeController::class, 'index']],
```

```
[ 'method' => 'GET', 'uri' => '/category/list/{nameCategory}', 'action' => [HomeController::class, 'list']],
[ 'method' => 'GET', 'uri' => '/blog/', 'action' => [HomeController::class, 'listBlogAllPosts']],
[ 'method' => 'GET', 'uri' => '/blog/show/{title}/{id}', 'action' => [HomeController::class, 'show']],
[ 'method' => 'GET', 'uri' => '/admin/users', 'action' => [UserController::class, 'index'], 'middleware' => ['admin']],
[ 'method' => 'GET', 'uri' => '/profile/edit', 'action' => [ProfileController::class, 'edit'], 'middleware' => ['auth', 'verified']],
[ 'method' => 'POST', 'uri' => '/profile/update', 'action' => [ProfileController::class, 'update'], 'middleware' => ['auth', 'verified']],
];
```

Arquivo `app/Router/Route.php`

php

```
<?php

namespace App\Router;

class Route
{
    public string $method;
    public string $uri;
    public $action;
    public array $middleware;

    public function __construct(string $method, string $uri, $action, array $middleware = [])
    {
        $this->method = $method;
        $this->uri = $uri;
        $this->action = $action;
        $this->middleware = $middleware;
    }

    public function matches(string $requestUri, string $requestMethod): bool
    {
        if ($this->method !== $requestMethod) {
            return false;
        }

        $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
        return preg_match("#^{$pattern}$#", $requestUri);
    }

    public function extractParameters(string $requestUri): array
    {
        $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
        preg_match("#^{$pattern}$#", $requestUri, $matches);
        array_shift($matches);
        return $matches;
    }
}
```

Arquivo `app/Router/Router.php`

php

```
<?php
```

```
namespace App\Router;

use App\Router\MiddlewareHandler;

class Router
{
    protected array $routes = [];

    public function addRoute(Route $route)
    {
        $this->routes[] = $route;
    }

    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach ($this->routes as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                MiddlewareHandler::handle($route->middleware);

                [$controller, $method] = $route->action;
                $controllerInstance = new $controller();
                return call_user_func_array([$controllerInstance, $method], $params);
            }
        }

        http_response_code(404);
        echo "404 Not Found";
    }
}
```

Arquivo `app/Router/MiddlewareHandler.php`

php

```
<?php

namespace App\Router;

use App\Middlewares\AdminMiddleware;
use App\Middlewares\AuthMiddleware;
use App\Middlewares\VerifiedMiddleware;

class MiddlewareHandler
{
    protected static array $middlewareMap = [
        'admin' => AdminMiddleware::class,
        'auth' => AuthMiddleware::class,
        'verified' => VerifiedMiddleware::class,
    ];

    public static function handle(array $middlewares)
    {
        foreach ($middlewares as $middleware) {
            $middlewareClass = self::$middlewareMap[$middleware] ?? null;

            if ($middlewareClass) {
                $instance = new $middlewareClass();
                $instance->handle();
            }
        }
    }
}
```

Exemplo de Middleware: `app/Middlewares/AdminMiddleware.php`

```
php

<?php

namespace App\Middlewares;

class AdminMiddleware
{
    public function handle()
    {
        // Lógica para verificar se o usuário é administrador
        if (!isset($_SESSION['user']) || $_SESSION['user']['role'] !== 'admin') {
            http_response_code(403);
            die("403 Forbidden");
        }
    }
}
```

Arquivo `public/index.php`

```
php

<?php

require_once __DIR__ . '/../bootstrap.php';

use App\Router\Router;
use App\Router\Route;

$routes = require __DIR__ . '/../routes/web.php';
$router = new Router();

foreach ($routes as $route) {
    $router->addRoute(
        new Route($route['method'], $route['uri'], $route['action'], $route['middleware'] ??
        [])
    );
}

$router->dispatch($_SERVER['REQUEST_URI'], $_SERVER['REQUEST_METHOD']);
```

Com essa estrutura, você pode facilmente adicionar rotas, middlewares e controladores, enquanto mantém uma arquitetura bem organizada e extensível.



como instalar o dumper do symfony em modo dev



O **Symfony VarDumper** é uma ferramenta útil para depuração em projetos PHP, e você pode instalá-lo em modo de desenvolvimento com o Composer. Siga os passos abaixo para instalar e configurar o VarDumper no seu projeto:

1. Instalar o VarDumper como uma dependência de desenvolvimento

Execute o seguinte comando no terminal:

```
bash

composer require symfony/var-dumper --dev
```

Este comando adicionará o VarDumper ao seu projeto apenas no ambiente de desenvolvimento.

2. Usar o `dump()` em seu código

Após a instalação, você pode usar a função `dump()` para exibir variáveis e objetos com formatação legível. Exemplo:

```
php

<?php

require __DIR__ . '/vendor/autoload.php';

$data = ['name' => 'John', 'email' => 'john@example.com'];
dump($data);
```

O `dump()` exibe as informações no terminal ou navegador de maneira bem formatada e fácil de interpretar.

3. Habilitar o dump para CLI

Se você estiver executando scripts PHP no terminal (CLI), o VarDumper é automaticamente configurado para exibir a saída.

4. Habilitar o dump no navegador (opcional)

Se você quiser integrar o dump no navegador durante o desenvolvimento, pode habilitar o **Dumper Server**:

1. Inicie o servidor dumper no terminal:

```
bash

php bin/console server:dump
```

2. Nos seus scripts PHP, o `dump()` enviará os dados para o console em vez do navegador.
-

5. Desabilitar em produção

Como a dependência está instalada em modo `--dev`, ela não será carregada em ambientes de produção. Certifique-se de não deixar chamadas `dump()` ou `dd()` em código que será implantado.

6. Alternativa: Usar `dd()` para depuração rápida

O `dd()` (dump and die) encerra a execução do script após exibir a variável. Exemplo:

```
php

dd($data);
```

Isso é útil para depuração rápida, mas evite usar em produção.

Agora você tem o **Symfony VarDumper** configurado e pronto para depurar variáveis de forma eficiente no seu projeto PHP!



```
Mas para eu usar o arquivo exatamente assim, como ficaria? Route::get('/', [HomeController::class,
'index']);
Route::get('/category/list/{nameCategory}', [HomeController::class, 'list']);
Route::get('/blog/', [HomeController::class, 'listBlogAllPosts']);
Route::get('/blog/show/{title}/{id}', [HomeController::class, 'show']);
Route::get('/admin/users', [UserController::class, 'index'])->middleware('admin') -
>name('admin.users');
Route::get('/profile/edit', [ProfileController::class, 'edit'])
->middleware(['auth', 'verified'])
->name('profile.edit');
Route::post('/profile/update', [ProfileController::class, 'update'])
->middleware(['auth', 'verified'])
->name('profile.update');
```



Para usar um arquivo de rotas no formato exatamente como você descreveu, podemos criar uma classe `Route` que suporte métodos estáticos para definir as rotas de forma declarativa, como no exemplo. Isso tornará o sistema de rotas mais próximo do estilo de frameworks como Laravel.

Aqui está como implementar:

Estrutura de Arquivos Atualizada

plaintext

```
project/
├── app/
│   ├── Controllers/
│   │   ├── HomeController.php
│   │   ├── UserController.php
│   │   └── ProfileController.php
```

```

├── Middlewares/
│   ├── AdminMiddleware.php
│   ├── AuthMiddleware.php
│   └── VerifiedMiddleware.php
├── Router/
│   ├── Router.php
│   ├── Route.php
│   └── MiddlewareHandler.php
├── public/
│   └── index.php
├── routes/
│   └── web.php
├── vendor/
├── .env
├── composer.json
└── bootstrap.php

```

Arquivo `routes/web.php`

```

php

<?php

use App\Router\Route;
use App\Controllers\HomeController;
use App\Controllers\UserController;
use App\Controllers\ProfileController;

Route::get('/', [HomeController::class, 'index']);
Route::get('/category/list/{nameCategory}', [HomeController::class, 'list']);
Route::get('/blog/', [HomeController::class, 'listBlogAllPosts']);
Route::get('/blog/show/{title}/{id}', [HomeController::class, 'show']);
Route::get('/admin/users', [UserController::class, 'index'])->middleware('admin')->name('admin.users');
Route::get('/profile/edit', [ProfileController::class, 'edit'])->middleware(['auth', 'verified'])->name('profile.edit');
Route::post('/profile/update', [ProfileController::class, 'update'])->middleware(['auth', 'verified'])->name('profile.update');

```

Arquivo `app/Router/Route.php`

Esta classe permitirá a definição das rotas de maneira fluida e declarativa.

```

php

<?php

namespace App\Router;

class Route
{
    private static array $routes = [];
    private string $method;
    private string $uri;
    private $action;
    private array $middleware = [];
    private ?string $name = null;

    public function __construct(string $method, string $uri, $action)
    {
        $this->method = $method;
        $this->uri = $uri;
    }
}

```



```

        $this->action = $action;
    }

    public static function get(string $uri, $action): self
    {
        return self::addRoute('GET', $uri, $action);
    }

    public static function post(string $uri, $action): self
    {
        return self::addRoute('POST', $uri, $action);
    }

    private static function addRoute(string $method, string $uri, $action): self
    {
        $route = new self($method, $uri, $action);
        self::$routes[] = $route;
        return $route;
    }

    public function middleware($middleware): self
    {
        $this->middleware = is_array($middleware) ? $middleware : [$middleware];
        return $this;
    }

    public function name(string $name): self
    {
        $this->name = $name;
        return $this;
    }

    public static function getRoutes(): array
    {
        return self::$routes;
    }

    public function matches(string $requestUri, string $requestMethod): bool
    {
        if ($this->method !== $requestMethod) {
            return false;
        }

        $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
        return preg_match("#^{$pattern}$#", $requestUri);
    }

    public function extractParameters(string $requestUri): array
    {
        $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
        preg_match("#^{$pattern}$#", $requestUri, $matches);
        array_shift($matches);
        return $matches;
    }

    public function getAction()
    {
        return $this->action;
    }

    public function getMiddleware(): array
    {
        return $this->middleware;
    }
}

```

Arquivo `app/Router/Router.php`

Este arquivo processa as rotas definidas no arquivo `web.php`.

php

```

<?php

namespace App\Router;

use App\Router\MiddlewareHandler;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();
                if (is_array($action)) {
                    [$controller, $method] = $action;
                    $controllerInstance = new $controller();
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        http_response_code(404);
        echo "404 Not Found";
    }
}

```

Arquivo `public/index.php`

php

```

<?php

require_once __DIR__ . '/../bootstrap.php';

use App\Router\Router;

// Carrega as rotas definidas
require __DIR__ . '/../routes/web.php';

// Inicia o roteador
$routeur = new Router();
$routeur->dispatch($_SERVER['REQUEST_URI'], $_SERVER['REQUEST_METHOD']);

```

Adicionando Middleware Suporte

Já cobrimos o middleware no código anterior com a classe `MiddlewareHandler`. Isso permite que middlewares sejam aplicados dinamicamente às rotas.

Com isso, seu arquivo `web.php` funcionará exatamente como o exemplo fornecido, utilizando uma abordagem declarativa e fluida para definir as rotas!



Cade o arquivo `MiddlewareHandler`?



O arquivo `MiddlewareHandler` gerencia a execução dos middlewares associados às rotas. Aqui está a implementação:

Arquivo `app/Router/MiddlewareHandler.php`

```
php

<?php

namespace App\Router;

class MiddlewareHandler
{
    /**
     * Mapeia middlewares por nomes para suas respectivas classes.
     */
    protected static array $middlewareMap = [
        'admin' => \App\Middlewares\AdminMiddleware::class,
        'auth' => \App\Middlewares\AuthMiddleware::class,
        'verified' => \App\Middlewares\VerifiedMiddleware::class,
    ];

    /**
     * Executa os middlewares associados a uma rota.
     *
     * @param array $middlewares Lista de middlewares associados à rota.
     */
    public static function handle(array $middlewares)
    {
        foreach ($middlewares as $middleware) {
            $middlewareClass = self::$middlewareMap[$middleware] ?? null;

            if (!$middlewareClass || !class_exists($middlewareClass)) {
                throw new \Exception("Middleware '{$middleware}' não encontrado.");
            }

            $instance = new $middlewareClass();
            $instance->handle(); // Chamamos o método `handle` do middleware.
        }
    }
}
```

Como funciona?

- Mapeamento de middlewares:** O método `MiddlewareHandler` contém um array `$middlewareMap` onde o nome do middleware (como `'auth'` ou `'admin'`) é associado à classe que o implementa.
- Execução:** Quando uma rota possui middlewares (como definido no arquivo `web.php`), o `MiddlewareHandler` localiza a classe associada e executa seu método `handle()`.
- Exemplo de Middleware:** Cada middleware é uma classe que implementa a lógica necessária, como autenticação, verificação de permissões, etc.

Exemplo de Middlewares

`app/Middlewares/AuthMiddleware.php`

```
php

<?php

namespace App\Middlewares;

class AuthMiddleware
{
    public function handle()
    {
        session_start();
        if (empty($_SESSION['user'])) {
            http_response_code(401);
            die('Acesso negado: usuário não autenticado.');
```

`app/Middlewares/AdminMiddleware.php`

```
php

<?php

namespace App\Middlewares;

class AdminMiddleware
{
    public function handle()
    {
        session_start();
        if (empty($_SESSION['user']) || $_SESSION['user']['role'] !== 'admin') {
            http_response_code(403);
            die('Acesso negado: somente administradores.');
```

`app/Middlewares/VerifiedMiddleware.php`

```
php

<?php

namespace App\Middlewares;

class VerifiedMiddleware
{
    public function handle()
    {
        session_start();
        if (empty($_SESSION['user']) || !$_SESSION['user']['verified']) {
            http_response_code(403);
            die('Acesso negado: conta não verificada.');
```

```
}
}
```

Exemplo de Sessão de Usuário para Teste

Para testar os middlewares, adicione este exemplo ao início do arquivo `public/index.php`:

```
php

session_start();

// Exemplo de usuário autenticado
$_SESSION['user'] = [
    'id' => 1,
    'name' => 'John Doe',
    'role' => 'admin', // Ou 'user'
    'verified' => true,
];
```

Com o `MiddlewareHandler`, você pode gerenciar middlewares de forma dinâmica e organizada, vinculando-os facilmente às rotas.



Quero que isso `http_response_code(404);`
`echo "404 Not Found";` chame o controller `NotFoundController`



Para configurar o comportamento onde um controlador específico é chamado ao invés de usar diretamente `http_response_code(404)` e `echo "404 Not Found"`, você pode implementar um fallback que acione um controlador como `NotFoundController`. Isso mantém o sistema mais organizado e extensível.

Atualização no `Router.php`

Modifique o arquivo `Router.php` para chamar o `NotFoundController` quando nenhuma rota corresponder.

```
php

<?php

namespace App\Router;

use App\Controllers\NotFoundController;
use Exception;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
```

```

        if ($route->matches($requestUri, $requestMethod)) {
            $params = $route->extractParameters($requestUri);

            MiddlewareHandler::handle($route->getMiddleware());

            $action = $route->getAction();
            if (is_array($action)) {
                [$controller, $method] = $action;
                $controllerInstance = new $controller();
                return call_user_func_array([$controllerInstance, $method], $params);
            }

            if (is_callable($action)) {
                return call_user_func_array($action, $params);
            }
        }
    }

    // Chamar o controlador para páginas 404
    $this->handleNotFound();
}

/**
 * Gerencia requisições não encontradas chamando o NotFoundController.
 */
private function handleNotFound()
{
    http_response_code(404);

    $controller = new NotFoundController();
    if (method_exists($controller, 'index')) {
        return $controller->index();
    }

    throw new Exception("NotFoundController ou método 'index' não implementado.");
}
}

```

Criando o `NotFoundController`

O controlador será responsável por renderizar uma página 404 personalizada.

Arquivo `app/Controllers/NotFoundController.php`

```

php

<?php

namespace App\Controllers;

class NotFoundController
{
    public function index()
    {
        echo '<h1>404 - Página não encontrada</h1>';
        echo '<p>A página que você está procurando não existe ou foi removida.</p>';
        echo '<a href="/">Voltar para a página inicial</a>';
    }
}

```

Como funciona?

1. **Verificação de correspondência de rotas:** O `Router` percorre todas as rotas registradas. Se nenhuma delas corresponder ao método e URI da requisição, o método `handleNotFound` será chamado.
2. **Chamando o `NotFoundController`:** No método `handleNotFound`, o `NotFoundController` é instanciado e seu método `index` é chamado. Você pode personalizar esse comportamento ou adicionar métodos adicionais ao controlador.
3. **HTTP Status Code:** O `http_response_code(404)` garante que o navegador e ferramentas como crawlers recebam o código de erro correto.

Exemplo de Teste

Caso uma rota inválida seja acessada, como:

```
plaintext

GET /rota/invalida
```

O navegador exibirá a página gerada pelo `NotFoundController`:

```
html

<h1>404 - Página não encontrada</h1>
<p>A página que você está procurando não existe ou foi removida.</p>
<a href="/">Voltar para a página inicial</a>
```

Com essa abordagem, você tem um sistema centralizado para gerenciar erros 404, seguindo o padrão MVC. Isso permite estilizar a página de erro e adicionar lógica adicional conforme necessário.



Crie um controller para testar a rota `Route::get('/blog/show/{title}/{id}', [HomeController::class, 'show'])`;



Aqui está um exemplo de como implementar o método `show` no `HomeController` para testar a rota `Route::get('/blog/show/{title}/{id}', [HomeController::class, 'show'])`.

Arquivo `app/Controllers/HomeController.php`

```
php

<?php

namespace App\Controllers;

class HomeController
```

```
{
    /**
     * Exibe os detalhes de um post do blog.
     *
     * @param string $title Título do post (slug ou nome amigável).
     * @param int $id ID do post no banco de dados.
     */
    public function show(string $title, int $id)
    {
        // Simulação de busca no banco de dados
        $posts = [
            1 => ['title' => 'Primeiro Post', 'content' => 'Conteúdo do primeiro post.'],
            2 => ['title' => 'Segundo Post', 'content' => 'Conteúdo do segundo post.'],
        ];

        if (!isset($posts[$id])) {
            http_response_code(404);
            echo '<h1>404 - Post não encontrado</h1>';
            return;
        }

        $post = $posts[$id];

        // Renderiza o post
        echo '<h1>' . htmlspecialchars($post['title']) . '</h1>';
        echo '<p>' . htmlspecialchars($post['content']) . '</p>';
        echo '<a href="/blog/">Voltar para o Blog</a>';
    }
}
```

Como funciona o método `show`

1. Parâmetros da URL:

- O parâmetro `{title}` é recebido como uma string representando o título do post (normalmente no formato slug).
- O parâmetro `{id}` é um inteiro que representa o ID do post.

2. Simulação de banco de dados:

- Um array `\$posts` simula os dados de um banco de dados, mapeando IDs para os detalhes do post.
- Caso o `id` fornecido na URL não exista no array, uma mensagem 404 personalizada é exibida.

3. Exibição do post:

- Se o post for encontrado, o título e o conteúdo são exibidos de forma segura, utilizando `htmlspecialchars` para evitar problemas de segurança.

4. Link de retorno:

- Um link para voltar à página principal do blog é incluído.

Testando a Rota

Exemplo de URL válida

plaintext

```
GET /blog/show/primeiro-post/1
```


Saída no navegador:

```
html

<h1>Primeiro Post</h1>
<p>Conteúdo do primeiro post.</p>
<a href="/blog/">Voltar para o Blog</a>
```

Exemplo de URL inválida

```
plaintext

GET /blog/show/nao-existe/999
```

Saída no navegador:

```
html

<h1>404 - Post não encontrado</h1>
```

Expansão futura

Para um sistema mais realista, você pode integrar com um banco de dados usando o PDO ou um ORM como Eloquent. Isso permitirá que os posts sejam buscados dinamicamente e que o conteúdo do blog seja gerenciado de forma mais robusta.



```
agora para testar esse middleware Route::get('/profile/edit', [DashboardController::class, 'edit'])-
>middleware(['auth']);
```



Para testar a rota `Route::get('/profile/edit', [DashboardController::class, 'edit'])->middleware(['auth']);` com o middleware `auth`, precisamos:

1. **Criar o Middleware `AuthMiddleware`**: Garante que o usuário esteja autenticado.
2. **Implementar o `DashboardController`**: Define o método `edit`.
3. **Simular Sessão para Usuário Autenticado**: Testar diferentes cenários.

1. Middleware `AuthMiddleware`

Arquivo: `app/Middlewares/AuthMiddleware.php`

php

```
<?php

namespace App\Middlewares;

class AuthMiddleware
{
    public function handle()
    {
        session_start();

        // Verifica se o usuário está autenticado
        if (empty($_SESSION['user'])) {
            http_response_code(401);
            die('<h1>401 - Não autorizado</h1><p>Você precisa estar logado para acessar esta
página.</p>');
        }
    }
}
```

2. `DashboardController`

Arquivo: `app/Controllers/DashboardController.php`

php

```
<?php

namespace App\Controllers;

class DashboardController
{
    /**
     * Mostra a página de edição de perfil.
     */
    public function edit()
    {
        session_start();

        $user = $_SESSION['user'] ?? null;

        if (!$user) {
            http_response_code(401);
            die('<h1>401 - Não autorizado</h1>');
        }

        echo '<h1>Editar Perfil</h1>';
        echo '<p>Bem-vindo, ' . htmlspecialchars($user['name']) . '</p>';
        echo '<form method="POST" action="/profile/update">
            <label>Nome:</label><br>
            <input type="text" name="name" value="' . htmlspecialchars($user['name']) .
            '</form>';
    }
}
```

3. Simulando Sessão

No arquivo `public/index.php`, simule o usuário autenticado:

Arquivo: `public/index.php`

```
php

session_start();

// Simule um usuário logado
$_SESSION['user'] = [
    'id' => 1,
    'name' => 'John Doe',
];

// Comente a linha abaixo para simular um usuário não autenticado
// unset($_SESSION['user']);
```

4. Configurando a Rota

No arquivo `routes/web.php`:

```
php

<?php

use App\Router\Route;
use App\Controllers\DashboardController;

Route::get('/profile/edit', [DashboardController::class, 'edit'])->middleware(['auth']);
```

Testando a Rota

Cenário 1: Usuário autenticado

Com `\$_SESSION['user']` definido no `public/index.php`, acesse:

```
plaintext

GET /profile/edit
```

Saída esperada no navegador:

```
html

<h1>Editar Perfil</h1>
<p>Bem-vindo, John Doe</p>
<form method="POST" action="/profile/update">
  <label>Nome:</label><br>
  <input type="text" name="name" value="John Doe"><br>
  <button type="submit">Salvar</button>
</form>
```

Cenário 2: Usuário não autenticado

Comente ou remova a linha que define `\$_SESSION['user']` no `public/index.php`:

```
plaintext
```

```
GET /profile/edit
```

Saída esperada no navegador:

```
html
```

```
<h1>401 - Não autorizado</h1>
<p>Você precisa estar logado para acessar esta página.</p>
```

Expansão Futura

- **Redirects:** Adicionar um redirecionamento para a página de login se o usuário não estiver autenticado.
- **Banco de Dados:** Carregar o usuário autenticado a partir de um banco de dados em vez de usar uma sessão simulada.
- **Gerenciamento de Sessão Segura:** Integrar tokens de autenticação ou frameworks para maior segurança.

Com isso, o middleware `auth` está configurado e funcional!



Crie um helper para setar e pegar Sessions e cookies



Aqui está um helper para gerenciar **sessions** e **cookies**, permitindo que você defina, recupere e remova valores de ambos de forma centralizada.

Arquivo: `app/Helpers/SessionHelper.php`

```
php
```

```
<?php
namespace App\Helpers;

class SessionHelper
{
    /**
     * Inicializa a sessão, caso ainda não esteja ativa.
     */
    public static function startSession(): void
    {
        if (session_status() === PHP_SESSION_NONE) {
            session_start();
        }
    }
}
```

```

}

/**
 * Define um valor na sessão.
 *
 * @param string $key
 * @param mixed $value
 */
public static function setSession(string $key, mixed $value): void
{
    self::startSession();
    $_SESSION[$key] = $value;
}

/**
 * Obtém um valor da sessão.
 *
 * @param string $key
 * @param mixed $default Valor padrão caso a chave não exista.
 * @return mixed
 */
public static function getSession(string $key, mixed $default = null): mixed
{
    self::startSession();
    return $_SESSION[$key] ?? $default;
}

/**
 * Remove um valor da sessão.
 *
 * @param string $key
 */
public static function unsetSession(string $key): void
{
    self::startSession();
    unset($_SESSION[$key]);
}

/**
 * Destroi a sessão completamente.
 */
public static function destroySession(): void
{
    self::startSession();
    session_destroy();
}
}

```

Arquivo: `app/Helpers/CookieHelper.php`

php

```

<?php

namespace App\Helpers;

class CookieHelper
{
    /**
     * Define um cookie.
     *
     * @param string $key
     * @param string $value
     * @param int $expire Tempo de expiração em segundos (padrão: 1 hora).
     * @param string $path Caminho válido para o cookie (padrão: '/').
     */
    public static function setCookie(string $key, string $value, int $expire = 3600, string
$path = '/'): void
    {

```

```

        setcookie($key, $value, time() + $expire, $path);
    }

    /**
     * Obtém um valor de um cookie.
     *
     * @param string $key
     * @param mixed $default Valor padrão caso o cookie não exista.
     * @return mixed
     */
    public static function getCookie(string $key, mixed $default = null): mixed
    {
        return $_COOKIE[$key] ?? $default;
    }

    /**
     * Remove um cookie.
     *
     * @param string $key
     * @param string $path Caminho válido para o cookie (padrão: '/').
     */
    public static function unsetCookie(string $key, string $path = '/'): void
    {
        setcookie($key, '', time() - 3600, $path);
    }
}

```

Como usar os helpers

Sessões

1. Definir um valor na sessão:

php

```

use App\Helpers\SessionHelper;

SessionHelper::setSession('user', ['id' => 1, 'name' => 'John Doe']);

```

2. Recuperar um valor da sessão:

php

```

$user = SessionHelper::getSession('user');

```

3. Remover um valor da sessão:

php

```

SessionHelper::unsetSession('user');

```

4. Destruir toda a sessão:

```
php
```

```
SessionHelper::destroySession();
```

Cookies

1. Definir um cookie:

```
php
```

```
use App\Helpers\CookieHelper;

CookieHelper::setCookie('token', 'abc123', 3600); // Expira em 1 hora
```

2. Recuperar um cookie:

```
php
```

```
$token = CookieHelper::getCookie('token');
```

3. Remover um cookie:

```
php
```

```
CookieHelper::unsetCookie('token');
```

Exemplo integrado em um controlador

```
php
```

```
<?php

namespace App\Controllers;

use App\Helpers\SessionHelper;
use App\Helpers\CookieHelper;

class AuthController
{
    public function login()
    {
        // Exemplo de login
        SessionHelper::setSession('user', ['id' => 1, 'name' => 'John Doe']);
        CookieHelper::setCookie('auth_token', 'secure-token', 3600);

        echo 'Usuário logado com sucesso!';
    }

    public function logout()
    {
        SessionHelper::destroySession();
    }
}
```

```

        CookieHelper::unsetCookie('auth_token');
    }
    echo 'Usuário deslogado com sucesso!';
}

```

Com esses helpers, você pode gerenciar **sessions** e **cookies** de forma eficiente, limpa e reutilizável.



Agora usando esses helpers, refaça o controller para usar a rota agora para testar esse middleware
 Route::get('/profile/edit', [DashboardController::class, 'edit']->middleware(['auth']));



Aqui está o `DashboardController` atualizado para testar a rota `Route::get('/profile/edit', [DashboardController::class, 'edit']->middleware(['auth']));` utilizando os helpers para sessões:

Arquivo: `app/Controllers/DashboardController.php`

```

php

<?php

namespace App\Controllers;

use App\Helpers\SessionHelper;

class DashboardController
{
    /**
     * Mostra a página de edição de perfil.
     * Só é acessível para usuários autenticados devido ao middleware `auth`.
     */
    public function edit()
    {
        // Obtém o usuário autenticado da sessão
        $user = SessionHelper::getSession('user');

        if (!$user) {
            http_response_code(401);
            echo '<h1>401 - Não autorizado</h1>';
            echo '<p>Você precisa estar logado para acessar esta página.</p>';
            echo '<a href="/login">Ir para a página de login</a>';
            return;
        }

        // Renderiza a página de edição de perfil
        echo '<h1>Editar Perfil</h1>';
        echo '<p>Bem-vindo, ' . htmlspecialchars($user['name']) . '</p>';
        echo '<form method="POST" action="/profile/update">
            <label>Nome:</label><br>
            <input type="text" name="name" value="' . htmlspecialchars($user['name']) .
        ' "><br>
            <button type="submit">Salvar</button>
        </form>';
    }
}

```


Middleware `AuthMiddleware` Atualizado com Helpers

Caso ainda não tenha o middleware configurado, aqui está a implementação:

Arquivo: `app/Middlewares/AuthMiddleware.php`

```
php

<?php
namespace App\Middlewares;
use App\Helpers\SessionHelper;
class AuthMiddleware
{
    public function handle()
    {
        $user = SessionHelper::getSession('user');

        if (!$user) {
            http_response_code(401);
            die('<h1>401 - Não autorizado</h1><p>Você precisa estar logado para acessar esta
página.</p>');
        }
    }
}
```

Atualização no Arquivo `routes/web.php`

Certifique-se de que a rota esteja configurada corretamente:

Arquivo: `routes/web.php`

```
php

<?php
use App\Router\Route;
use App\Controllers\DashboardController;

Route::get('/profile/edit', [DashboardController::class, 'edit'])->middleware(['auth']);
```

Simulando um Usuário para Testes

Defina um usuário logado no início do seu projeto, por exemplo, no arquivo `public/index.php`:

Arquivo: `public/index.php`

```
php

use App\Helpers\SessionHelper;

// Simule um usuário autenticado
```

```
SessionHelper::setSession('user', [
    'id' => 1,
    'name' => 'John Doe',
]);

// Descomente para simular um usuário não autenticado
// SessionHelper::unsetSession('user');
```

Testando a Rota

Cenário 1: Usuário autenticado

1. Certifique-se de que a sessão contém um usuário.
2. Acesse a URL:

```
plaintext
```

```
GET /profile/edit
```

Saída esperada:

```
html
```

```
<h1>Editar Perfil</h1>
<p>Bem-vindo, John Doe</p>
<form method="POST" action="/profile/update">
  <label>Nome:</label><br>
  <input type="text" name="name" value="John Doe"><br>
  <button type="submit">Salvar</button>
</form>
```

Cenário 2: Usuário não autenticado

1. Remova ou comente o usuário da sessão em `public/index.php`.
2. Acesse a URL:

```
plaintext
```

```
GET /profile/edit
```

Saída esperada:

```
html
```

```
<h1>401 - Não autorizado</h1>
<p>Você precisa estar logado para acessar esta página.</p>
<a href="/login">Ir para a página de login</a>
```

Expansão Futura

- **Redirecionamento Dinâmico:** Adicione lógica para redirecionar usuários não autenticados automaticamente para a página de login.
- **Sistema de Autenticação Completo:** Utilize cookies para tokens de sessão em vez de depender apenas de variáveis de sessão, para maior segurança.

Essa implementação aproveita os helpers para tornar o código mais limpo, reutilizável e centralizado para gerenciar autenticação.



crie uma view com twig e bootstrap com menu superior e alguns cards



Aqui está um exemplo de uma **view** utilizando o **Twig** para renderizar o conteúdo, **Bootstrap** para o estilo e um **menu superior** com **cards**.

Estrutura do Projeto

Vamos criar uma estrutura simples:

```
bash

/app
  /Controllers
  /Views
    /templates
      layout.twig
      home.twig
  /Helpers
  /Middlewares
/public
  /assets
    /css
      bootstrap.min.css
  index.php
```

1. Instalando o Twig e Bootstrap

Certifique-se de que o **Twig** e o **Bootstrap** estejam instalados.

- Para **Twig**, se ainda não estiver instalado, use o Composer:

```
bash

composer require "twig/twig:^3.0"
```

- **Bootstrap** pode ser integrado via CDN diretamente no template ou baixado. Aqui vamos usar o **CDN**.

2. Configurando o Twig no Projeto

No seu arquivo de configuração do projeto (por exemplo, `public/index.php` ou um arquivo de configuração específico), configure o Twig:

Arquivo: `public/index.php`

```
php

<?php

require '../vendor/autoload.php';

// Configuração do Twig
$loader = new \Twig\Loader\FilesystemLoader('../app/Views/templates');
$twig = new \Twig\Environment($loader, [
    'cache' => false, // Em produção, use um diretório de cache para melhorar o desempenho
]);

// Exemplo de renderização
echo $twig->render('home.twig', [
    'title' => 'Página Inicial',
    'user' => 'John Doe',
]);
```

3. Layout Principal (Template Base)

Arquivo: `app/Views/templates/layout.twig`

Este será o layout base com o menu superior e a inclusão do Bootstrap:

```
html

<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ title }}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-KyZXEJ3QJbuj4v+Lz6d8H9pdX+xFsxLFf3Cq9lRzU5xU0Zkt+dDeJH0zBevJ5+44" crossorigin="anonymous">
</head>
<body>

    <!-- Menu Superior -->
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand" href="#">Minha Aplicação</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Alternar navegação">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page" href="/">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/profile/edit">Perfil</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
```

```

        <li class="nav-item">
          <a class="nav-link" href="/blog">Blog</a>
        </li>
      </ul>
    </div>
  </div>
</nav>

<div class="container mt-4">
  {% block content %} {% endblock %}
</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-pzjw8f+ua7Kw1TIq0bb2dM1sbe0qL5lhpvcfgz5lhvXw9EoPuwRGtgLs5M9gqzd" crossorigin="anonymous"></script>
</body>
</html>

```

4. Home Page (Com Cards)

Arquivo: `app/Views/templates/home.twig`

Aqui, vamos renderizar a página inicial com alguns **cards** e o conteúdo passando via o controller:

```

html

{% extends 'layout.twig' %}

{% block content %}
  <div class="row">
    <div class="col-md-12">
      <h1>Bem-vindo, {{ user }}</h1>
      <p>Esta é a página inicial da nossa aplicação.</p>
    </div>
  </div>

  <div class="row">
    <!-- Card 1 -->
    <div class="col-md-4">
      <div class="card">
        
        <div class="card-body">
          <h5 class="card-title">Card 1</h5>
          <p class="card-text">Descrição do card 1. Um conteúdo interessante sobre o tema.</p>
          <a href="#" class="btn btn-primary">Ir para Card 1</a>
        </div>
      </div>
    </div>

    <!-- Card 2 -->
    <div class="col-md-4">
      <div class="card">
        
        <div class="card-body">
          <h5 class="card-title">Card 2</h5>
          <p class="card-text">Descrição do card 2. Detalhes sobre um segundo tema.</p>
          <a href="#" class="btn btn-primary">Ir para Card 2</a>
        </div>
      </div>
    </div>

    <!-- Card 3 -->
    <div class="col-md-4">

```

```

        <div class="card">
            
            <div class="card-body">
                <h5 class="card-title">Card 3</h5>
                <p class="card-text">Descrição do card 3. Mais conteúdo interessante par
a o usuário.</p>
                <a href="#" class="btn btn-primary">Ir para Card 3</a>
            </div>
        </div>
    </div>
</div>
{% endblock %}

```

5. Controller para Renderizar a View

Arquivo: `app/Controllers/HomeController.php`

Aqui está um exemplo de como renderizar a view com o Twig no controller.

```

php

<?php

namespace App\Controllers;

use App\Helpers\SessionHelper;

class HomeController
{
    /**
     * Exibe a página inicial com os cards.
     */
    public function index()
    {
        // Você pode passar dados do controlador para a view
        echo $twig->render('home.twig', [
            'title' => 'Página Inicial',
            'user' => SessionHelper::getSession('user', 'Visitante'),
        ]);
    }
}

```

6. Testando a Rota

No arquivo de rotas, defina a rota para a página inicial que utiliza o controller:

Arquivo: `routes/web.php`

```

php

<?php

use App\Router\Route;
use App\Controllers\HomeController;

Route::get('/', [HomeController::class, 'index']);

```

Conclusão

Agora, quando você acessar a URL `/`, o Twig irá renderizar o layout principal com o menu superior, incluindo três cards com informações dinâmicas. A estrutura com **Twig** e **Bootstrap** proporciona uma forma simples e limpa de criar views interativas e responsivas.



```
use assim <?php

class View{
    public static function View(string $view, array $data = []){

        $path = dirname(__FILE__) . '/../resources/views/';

        $loader = Twig\Loader\FilesystemLoader($path);
        $twig = new Twig\Environment($loader);

        echo $twig->render($view, $data);

    }
} criado em library e as views em /src/views
```



Entendido! Vamos ajustar a estrutura conforme o seu pedido. Utilizaremos a classe `View` que você mencionou e mudaremos o caminho das views para `/src/views`. Vamos também corrigir a sintaxe para usar corretamente o Twig.

Aqui está o passo a passo para configurar o projeto com a classe `View`:

1. Criando a Classe View

Arquivo: `library/View.php`

```
php

<?php

namespace Library;

use Twig\Loader\FilesystemLoader;
use Twig\Environment;

class View
{
    /**
     * Renderiza uma view com os dados fornecidos.
     *
     * @param string $view Nome da view a ser renderizada.
     * @param array $data Dados a serem passados para a view.
     */
    public static function render(string $view, array $data = []): void
    {
```

```
// Caminho das views
$path = dirname(__FILE__) . '/../src/views/';

// Carrega o Twig
$loader = new FilesystemLoader($path);
$twig = new Environment($loader, [
    'cache' => false, // Em produção, use um diretório de cache para melhorar o desempenho
]);

// Renderiza e exibe a view com os dados
echo $twig->render($view, $data);
}
}
```

2. Estrutura de Diretórios do Projeto

Certifique-se de que a estrutura de diretórios esteja organizada da seguinte maneira:

```
bash

/project
  /library
    View.php
  /src
    /views
      layout.twig
      home.twig
  /public
    index.php
  /vendor
    (Twig e outras dependências)
```

3. Exemplo de Layout Principal (layout.twig)

Este será o template base, onde a estrutura de HTML comum (como o menu superior) será incluída, e as views específicas poderão ser renderizadas.

Arquivo: `src/views/layout.twig`

```
html

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{ title }}</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-KyZXEJ3QJbuj4v+Lz6d8H9pdX+xFsxLFf3Cq9lRzU5xU0Zkt+dDeJH0zBevJ5+44" crossorigin="anonymous">
</head>
<body>

  <!-- Menu Superior -->
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
      <a class="navbar-brand" href="#">Minha Aplicação</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Alternar nave
```



```
gação">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link active" aria-current="page" href="/">Home</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/profile/edit">Perfil</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/blog">Blog</a>
      </li>
    </ul>
  </div>
</div>
</nav>

<div class="container mt-4">
  {% block content %} {% endblock %}
</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-pzjw8f+ua7Kw1TIq0bb2dM1sbe0qL5lhpvcfgz5lhvXw9EoPuwRGTgLS5M9gqzd" s" crossorigin="anonymous"></script>
</body>
</html>
```

4. Home Page (home.twig)

Aqui está um exemplo de como a página inicial será renderizada, usando a classe `View` para incluir o layout e os dados:

Arquivo: `src/views/home.twig`

```
html

{% extends 'layout.twig' %}

{% block content %}
  <div class="row">
    <div class="col-md-12">
      <h1>Bem-vindo, {{ user }}</h1>
      <p>Esta é a página inicial da nossa aplicação.</p>
    </div>
  </div>

  <div class="row">
    <!-- Card 1 -->
    <div class="col-md-4">
      <div class="card">
        
        <div class="card-body">
          <h5 class="card-title">Card 1</h5>
          <p class="card-text">Descrição do card 1. Um conteúdo interessante sobre o tema.</p>
          <a href="#" class="btn btn-primary">Ir para Card 1</a>
        </div>
      </div>
    </div>

    <!-- Card 2 -->
    <div class="col-md-4">
      <div class="card">
```

```

do Card 2">
    
        <h5 class="card-title">Card 2</h5>
        <p class="card-text">Descrição do card 2. Detalhes sobre um segundo tem
a.</p>
        <a href="#" class="btn btn-primary">Ir para Card 2</a>
    </div>
</div>
</div>

<!-- Card 3 -->
<div class="col-md-4">
    <div class="card">
do Card 3">
    <div class="card-body">
        <h5 class="card-title">Card 3</h5>
        <p class="card-text">Descrição do card 3. Mais conteúdo interessante par
a o usuário.</p>
        <a href="#" class="btn btn-primary">Ir para Card 3</a>
    </div>
</div>
</div>
</div>
{% endblock %}

```

5. Controller para Renderizar a View

Arquivo: `src/Controllers/HomeController.php`

Aqui está um exemplo de como renderizar a página com os dados necessários (como o nome do usuário):

```

php

<?php

namespace Src\Controllers;

use Library\View;
use Src\Helpers\SessionHelper;

class HomeController
{
    public function index()
    {
        // Passando dados para a view
        $user = SessionHelper::getSession('user', 'Visitante');

        View::render('home.twig', [
            'title' => 'Página Inicial',
            'user' => $user,
        ]);
    }
}

```

6. Configuração da Rota

Certifique-se de que as rotas estejam configuradas corretamente para usar o controlador `HomeController`:

Arquivo: `src/routes/web.php`

```
php

<?php

use Src\Controllers\HomeController;

Route::get('/', [HomeController::class, 'index']);
```

7. Teste no Navegador

Quando você acessar o endpoint `/`, o sistema irá:

1. Renderizar o layout com o **menu superior**.
2. Exibir o **nome do usuário** (se estiver armazenado na sessão) e os **cards**.

Essa estrutura permite que você tenha uma separação clara entre lógica de renderização e layout. Usando o **Twig** e a classe `View`, seu código fica mais limpo e fácil de gerenciar.



se usar o conceito de template master numa pasta a parte em views, tipo src/views/template e src/views/partials como que ficaria



Se você deseja usar o conceito de **template master** (layout principal) e **partials** (partes reutilizáveis como cabeçalhos, rodapés, menus) organizados em pastas separadas, isso ajudará a manter o código mais modular e reutilizável.

Aqui está como a estrutura de diretórios e os arquivos ficariam organizados para seguir esse conceito:

1. Estrutura de Diretórios

A estrutura de diretórios ficaria assim:

```
bash

/project
  /library
    View.php
  /src
    /views
      /partials
        header.twig
        footer.twig
      /templates
        layout.twig
        home.twig
  /public
    index.php
```

```
/vendor
(dependências do Twig e outras)
```

- ``/src/views/partials``: Armazenará os arquivos de **partes reutilizáveis** do layout, como cabeçalhos, rodapés, menus.
- ``/src/views/templates``: Contém o template **master**, ou seja, o layout principal que será utilizado por outras views.
- ``/src/views/home.twig``: A view específica que renderiza o conteúdo principal, estendendo o layout e utilizando as partes parciais.

2. Classe View (sem mudanças)

A classe ``View`` que você criou permanece a mesma, pois ela já gerencia a renderização das views.

```
php

namespace Library;

use Twig\Loader\FilesystemLoader;
use Twig\Environment;

class View
{
    /**
     * Renderiza uma view com os dados fornecidos.
     *
     * @param string $view Nome da view a ser renderizada.
     * @param array $data Dados a serem passados para a view.
     */
    public static function render(string $view, array $data = []): void
    {
        $path = dirname(__FILE__) . '/../src/views/';

        $loader = new FilesystemLoader($path);
        $twig = new Environment($loader, [
            'cache' => false,
        ]);

        echo $twig->render($view, $data);
    }
}
```

3. Template Master (layout.twig)

Este é o layout principal onde o conteúdo será injetado dentro de um template padrão. O layout também inclui os **partials** (como o cabeçalho e o rodapé).

Arquivo: ``src/views/templates/layout.twig``

```
html

<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ title }}</title>
```

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-KyZXEJ3QJbuj4v+Lz6d8H9pdX+xFsxLFf3Cq9lRzU5xU0Zkt+dDeJH0zBevJ5+44" crossorigin="anonymous">
</head>
<body>

  <!-- Incluir Cabeçalho -->
  {% include 'partials/header.twig' %}

  <div class="container mt-4">
    {% block content %} {% endblock %}
  </div>

  <!-- Incluir Rodapé -->
  {% include 'partials/footer.twig' %}

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-pzjw8f+ua7Kw1TIq0bb2dM1sbe0qL5lhpvcfgz5lhvXw9EoPuwRGtGls5M9gqzd" crossorigin="anonymous"></script>
</body>
</html>
```

- O comando `{% include 'partials/header.twig' %}` inclui o **cabeçalho** em todas as páginas que usarem este layout.
- O comando `{% include 'partials/footer.twig' %}` inclui o **rodapé**.
- O bloco `{% block content %}` é onde o conteúdo da página específica será renderizado.

4. Cabeçalho (header.twig)

Aqui fica o **menu superior** e outras informações gerais de cabeçalho.

Arquivo: `src/views/partials/header.twig`

html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Minha Aplicação</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Alternar navegação">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="/">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/profile/edit">Perfil</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/blog">Blog</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

- Este arquivo contém o código HTML para o **menu superior**.

5. Rodapé (footer.twig)

Aqui você coloca o conteúdo do **rodapé**.

Arquivo: `src/views/partials/footer.twig`

```
html

<footer class="bg-light text-center text-lg-start">
  <div class="text-center p-3">
    &copy; 2024 Minha Aplicação
  </div>
</footer>
```

- Este arquivo contém o código HTML para o **rodapé** da página.

6. Home Page (home.twig)

Esta é a view específica, que usa o layout e os **partials**. O conteúdo específico da página será inserido no bloco `content`.

Arquivo: `src/views/home.twig`

```
html

{% extends 'templates/layout.twig' %}

{% block content %}
  <div class="row">
    <div class="col-md-12">
      <h1>Bem-vindo, {{ user }}</h1>
      <p>Esta é a página inicial da nossa aplicação.</p>
    </div>
  </div>

  <div class="row">
    <!-- Card 1 -->
    <div class="col-md-4">
      <div class="card">
        
        <div class="card-body">
          <h5 class="card-title">Card 1</h5>
          <p class="card-text">Descrição do card 1. Um conteúdo interessante sobre o tema.</p>
          <a href="#" class="btn btn-primary">Ir para Card 1</a>
        </div>
      </div>
    </div>

    <!-- Card 2 -->
    <div class="col-md-4">
      <div class="card">
        
        <div class="card-body">
          <h5 class="card-title">Card 2</h5>
          <p class="card-text">Descrição do card 2. Detalhes sobre um segundo tema.</p>
          <a href="#" class="btn btn-primary">Ir para Card 2</a>
        </div>
      </div>
    </div>
  </div>
</div>
```

```

        </div>
    </div>

    <!-- Card 3 -->
    <div class="col-md-4">
        <div class="card">
            
            <div class="card-body">
                <h5 class="card-title">Card 3</h5>
                <p class="card-text">Descrição do card 3. Mais conteúdo interessante par
a o usuário.</p>
                <a href="#" class="btn btn-primary">Ir para Card 3</a>
            </div>
        </div>
    </div>
</div>
{% endblock %}

```

- O arquivo `home.twig` estende o `layout.twig`, e o conteúdo específico da página é colocado dentro do bloco `content`.

7. Controller para Renderizar a View

Arquivo: `src/Controllers/HomeController.php`

```

php

<?php

namespace Src\Controllers;

use Library\View;
use Src\Helpers\SessionHelper;

class HomeController
{
    public function index()
    {
        // Passando dados para a view
        $user = SessionHelper::getSession('user', 'Visitante');

        View::render('home.twig', [
            'title' => 'Página Inicial',
            'user' => $user,
        ]);
    }
}

```

- O controlador **HomeController** chama a classe **View** e renderiza o template `home.twig` com os dados necessários.

8. Configuração da Rota

Arquivo: `src/routes/web.php`

```
php

<?php

use Src\Controllers\HomeController;

Route::get('/', [HomeController::class, 'index']);
```

9. Resultado no Navegador

Quando você acessar a URL ` /`:

1.



quero carregar o navbar.twig que está em partials dentro do header.twig que está em partials também. E o footer.twig de partials carregar na home.twig



Entendido! Para carregar o arquivo `navbar.twig` dentro do `header.twig`, e o `footer.twig` dentro do `home.twig`, podemos usar o conceito de **inclusão de templates** do Twig. Aqui está como podemos organizar isso.

1. Estrutura de Diretórios

A estrutura de diretórios continua a mesma:

```
bash

/project
  /library
    View.php
  /src
    /views
      /partials
        navbar.twig
        header.twig
        footer.twig
      /templates
        layout.twig
      home.twig
  /public
    index.php
  /vendor
    (dependências do Twig e outras)
```

2. Arquivo `header.twig`

Aqui, vamos incluir o `navbar.twig` dentro do `header.twig`:

Arquivo: `src/views/partials/header.twig`


```
html
```

```
<header>
  <!-- Incluir o Navbar -->
  {% include 'partials/navbar.twig' %}
</header>
```

- A linha `{% include 'partials/navbar.twig' %}` irá carregar o conteúdo do `navbar.twig` dentro do `header.twig`.

3. Arquivo `navbar.twig`

Este é o conteúdo do **menu superior** (navbar) que será incluído no cabeçalho:

Arquivo: `src/views/partials/navbar.twig`

```
html
```

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Minha Aplicação</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Alternar navegação">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="/">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/profile/edit">Perfil</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/blog">Blog</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

4. Arquivo `footer.twig`

Aqui, o rodapé da página será configurado:

Arquivo: `src/views/partials/footer.twig`

```
html
```

```
<footer class="bg-light text-center text-lg-start">
  <div class="text-center p-3">
    &copy; 2024 Minha Aplicação
  </div>
</footer>
```

5. Arquivo `layout.twig` (Template Master)

O layout principal (`layout.twig`) vai incluir o `header.twig` e o `footer.twig`, e também define o bloco de conteúdo para a página específica. Este layout pode ser utilizado por outras páginas.

Arquivo: `src/views/templates/layout.twig`

```
html

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{ title }}</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-KyZXEJ3QJbuj4v+Lz6d8H9pdX+xFsxLFf3Cq9lRzU5xU0Zkt+dDeJH0zBevJ5+44" crossorigin="anonymous">
</head>
<body>

  <!-- Incluir Cabeçalho com o Navbar -->
  {% include 'partials/header.twig' %}

  <div class="container mt-4">
    {% block content %} {% endblock %}
  </div>

  <!-- Incluir Rodapé -->
  {% include 'partials/footer.twig' %}

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-pzjw8f+ua7Kw1TIq0bb2dM1sbe0qL5lhpvcfgz5lhvXw9EoPuwRGtgLs5M9gqzd" crossorigin="anonymous"></script>
</body>
</html>
```

- A linha `{% include 'partials/header.twig' %}` irá incluir o arquivo `header.twig` (que por sua vez já inclui o `navbar.twig`).
- A linha `{% include 'partials/footer.twig' %}` incluirá o arquivo `footer.twig` para o rodapé.

6. Arquivo `home.twig` (Conteúdo Específico da Página)

A `home.twig` vai estender o `layout.twig` e definir o conteúdo específico da página no bloco `content`.

Arquivo: `src/views/home.twig`

```
html

{% extends 'templates/layout.twig' %}

{% block content %}
  <div class="row">
    <div class="col-md-12">
      <h1>Bem-vindo, {{ user }}</h1>
      <p>Esta é a página inicial da nossa aplicação.</p>
    </div>
  </div>

  <div class="row">
    <!-- Card 1 -->
```

```

<div class="col-md-4">
  <div class="card">
    
    <div class="card-body">
      <h5 class="card-title">Card 1</h5>
      <p class="card-text">Descrição do card 1. Um conteúdo interessante sobre
o tema.</p>
      <a href="#" class="btn btn-primary">Ir para Card 1</a>
    </div>
  </div>

  <!-- Card 2 -->
  <div class="col-md-4">
    <div class="card">
      
      <div class="card-body">
        <h5 class="card-title">Card 2</h5>
        <p class="card-text">Descrição do card 2. Detalhes sobre um segundo tem
a.</p>
        <a href="#" class="btn btn-primary">Ir para Card 2</a>
      </div>
    </div>

    <!-- Card 3 -->
    <div class="col-md-4">
      <div class="card">
        
        <div class="card-body">
          <h5 class="card-title">Card 3</h5>
          <p class="card-text">Descrição do card 3. Mais conteúdo interessante par
a o usuário.</p>
          <a href="#" class="btn btn-primary">Ir para Card 3</a>
        </div>
      </div>
    </div>
  </div>
{% endblock %}

```

- O arquivo `home.twig` estende o template `layout.twig` e utiliza o bloco `content` para definir o conteúdo específico da página.

7. Controller para Renderizar a View

Arquivo: `src/Controllers/HomeController.php`

```

php

<?php

namespace Src\Controllers;

use Library\View;
use Src\Helpers\SessionHelper;

class HomeController
{
    public function index()
    {
        // Passando dados para a view
        $user = SessionHelper::getSession('user', 'Visitante');

        View::render('home.twig', [
            'title' => 'Página Inicial',

```

```
'user' => $user,
]);
    }
}
```

- O controlador **HomeController** renderiza a página `home.twig`, passando os dados necessários para a view, como o nome do usuário.

8. Configuração da Rota

Arquivo: `src/routes/web.php`

```
php

<?php
use Src\Controllers\HomeController;
Route::get('/', [HomeController::class, 'index']);
```

Resultado no Navegador

Quando você acessar a URL `/`, o sistema irá:

1. **Carregar o layout** definido no arquivo `layout.twig`.
2. **Incluir o cabeçalho** e o **navbar** através de `header.twig` e `navbar.twig`.
3. **Incluir o rodapé** através de `footer.twig`.
4. Exibir o conteúdo específico da página dentro do bloco `content` da `home.twig`.

Essa estrutura proporciona modularidade e reutilização de código, com a separação clara entre o layout, os componentes parciais e o conteúdo específico de cada página.



quero por o header e footer na home, about etc E NÃO NO TEMPLATE MASTER. tem como?



Sim, é possível carregar o **header** e **footer** diretamente nas views específicas, como `home.twig`, `about.twig`, etc., em vez de colocá-los no template master (`layout.twig`). Dessa forma, o layout principal (`layout.twig`) seria mais genérico e as views específicas (como `home`, `about`) é que incluiriam os parciais do **header** e **footer**.

Como fazer isso?

Aqui está como você pode organizar:

1. Estrutura de Diretórios

A estrutura de diretórios permanece a mesma, mas o `layout.twig` será usado como um template básico para a estrutura HTML, e o **header** e **footer** serão incluídos nas views específicas.

```
bash

/project
  /library
    View.php
  /src
    /views
      /partials
        navbar.twig
        header.twig
        footer.twig
      /templates
        layout.twig
        home.twig
        about.twig
  /public
    index.php
  /vendor
    (dependências do Twig e outras)
```

2. Arquivo `header.twig`

O **header** ainda será o mesmo, contendo a estrutura de navegação (navbar).

Arquivo: `src/views/partials/header.twig`

```
html

<header>
  <!-- Incluir o Navbar -->
  {% include 'partials/navbar.twig' %}
</header>
```

3. Arquivo `navbar.twig`

Este é o conteúdo do **menu superior** (navbar) que será incluído dentro do **header**.

Arquivo: `src/views/partials/navbar.twig`

```
html

<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Minha Aplicação</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Alternar navegação">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="/">Home</a>
        </li>
        <li class="nav-item">
```

```

        <a class="nav-link" href="/about">Sobre</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/profile/edit">Perfil</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/blog">Blog</a>
      </li>
    </ul>
  </div>
</div>
</nav>

```

4. Arquivo `footer.twig`

Este arquivo permanece o mesmo, contendo o rodapé da página.

Arquivo: `src/views/partials/footer.twig`

```

html

<footer class="bg-light text-center text-lg-start">
  <div class="text-center p-3">
    &copy; 2024 Minha Aplicação
  </div>
</footer>

```

5. Arquivo `layout.twig` (Template Master)

O layout agora será um template mais simples, sem incluir o **header** e **footer**, permitindo que as views específicas incluam esses componentes. O `layout.twig` servirá apenas como o esqueleto básico do HTML.

Arquivo: `src/views/templates/layout.twig`

```

html

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{ title }}</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
    rel="stylesheet" integrity="sha384-KyZXEJ3QJbuj4v+Lz6d8H9pdX+xFsxLFf3Cq9lRzU5xU0Zkt+dDeJ
    H0zBevJ5+44" crossorigin="anonymous">
</head>
<body>

  <!-- Bloco de conteúdo será injetado aqui -->
  {% block content %} {% endblock %}

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-pzjw8f+ua7Kw1TIq0bb2dM1sbe0qL5lhpcvfgz5lhvXw9EoPuwRGtgLs5M9gqzd
    s" crossorigin="anonymous"></script>
</body>
</html>

```

O `layout.twig` agora é um arquivo "genérico", sem o **header** e **footer**.

6. Arquivo `home.twig` (Home Page Específica)

Agora, vamos adicionar o **header** e o **footer** diretamente na `home.twig`. A view `home.twig` irá usar o layout, mas incluirá o **header** e **footer**.

Arquivo: `src/views/home.twig`

```
html

{% extends 'templates/layout.twig' %}

{% block content %}
    <!-- Incluir Cabeçalho -->
    {% include 'partials/header.twig' %}

    <div class="container mt-4">
        <h1>Bem-vindo, {{ user }}</h1>
        <p>Esta é a página inicial da nossa aplicação.</p>
    </div>

    <div class="row">
        <!-- Card 1 -->
        <div class="col-md-4">
            <div class="card">
                
                <div class="card-body">
                    <h5 class="card-title">Card 1</h5>
                    <p class="card-text">Descrição do card 1. Um conteúdo interessante sobre o tema.</p>
                    <a href="#" class="btn btn-primary">Ir para Card 1</a>
                </div>
            </div>
        </div>

        <!-- Card 2 -->
        <div class="col-md-4">
            <div class="card">
                
                <div class="card-body">
                    <h5 class="card-title">Card 2</h5>
                    <p class="card-text">Descrição do card 2. Detalhes sobre um segundo tema.</p>
                    <a href="#" class="btn btn-primary">Ir para Card 2</a>
                </div>
            </div>
        </div>

        <!-- Card 3 -->
        <div class="col-md-4">
            <div class="card">
                
                <div class="card-body">
                    <h5 class="card-title">Card 3</h5>
                    <p class="card-text">Descrição do card 3. Mais conteúdo interessante para o usuário.</p>
                    <a href="#" class="btn btn-primary">Ir para Card 3</a>
                </div>
            </div>
        </div>
    </div>

    <!-- Incluir Rodapé -->
```

```
{% include 'partials/footer.twig' %}
{% endblock %}
```

- O arquivo `home.twig` estende o `layout.twig` e usa o bloco `content` para o conteúdo específico.
- O **header** e o **footer** são incluídos diretamente dentro da view.

7. Arquivo `about.twig` (Outra Página)

Da mesma forma, a página `about.twig` inclui o **header** e o **footer** diretamente, sem depender do template master para isso.

Arquivo: `src/views/about.twig`

```
html

{% extends 'templates/layout.twig' %}

{% block content %}
    <!-- Incluir Cabeçalho -->
    {% include 'partials/header.twig' %}

    <div class="container mt-4">
        <h1>Sobre Nós</h1>
        <p>Esta é a página sobre a nossa aplicação. Aqui você encontra mais informações sobre o que fazemos.</p>
    </div>

    <!-- Incluir Rodapé -->
    {% include 'partials/footer.twig' %}
{% endblock %}
```

8. Controller para Renderizar as Views

Arquivo: `src/Controllers/HomeController.php`

```
php

<?php

namespace Src\Controllers;

use Library\View;
use Src\Helpers\SessionHelper;

class HomeController
{
    public function index()
    {
        // Passando dados para a view
        $user = SessionHelper::getSession('user', 'Visitante');

        View::render('home.twig', [
            'title' => 'Página Inicial',
            'user' => $user,
        ]);
    }
}
```


Arquivo: `src/Controllers/AboutController.php`

```
php

<?php

namespace Src\Controllers;

use Library\View;

class AboutController
{
    public function index()
    {
        // Renderizar página sobre
        View::render('about.twig', [
            'title' => 'Sobre Nós',
        ]);
    }
}
```

9. Configuração da Rota

Arquivo: `src/routes/web.php`

```
php

<?php

use Src\Controllers\HomeController;
use Src\Controllers\AboutController;

Route::get('/', [HomeController::class
```



como carregar o css que vai estar em public/assets/css e o index em public



Para carregar corretamente os arquivos de **CSS** e **JS** que estão em `public/assets/css` e outros recursos dentro da pasta `public`, você precisa garantir que o caminho dos arquivos está configurado corretamente e acessível a partir do seu navegador.

1. Estrutura de Diretórios

Aqui está como deve ser a estrutura do seu projeto para garantir o acesso correto aos arquivos estáticos:

```
bash

/project
/public
/assets
```

```

        /css
            style.css
    index.php
/src
    /views
        /partials
            navbar.twig
            header.twig
            footer.twig
        /templates
            layout.twig
            home.twig
            about.twig
/vendor
    (dependências do Twig e outras)

```

2. Configuração do `index.php` para Roteamento

Dentro do arquivo `index.php`, você provavelmente está carregando as rotas. Vamos garantir que ele serve os arquivos estáticos corretamente, usando o diretório `public`.

Arquivo: `public/index.php`

```

php

<?php
require_once __DIR__ . '/../vendor/autoload.php';

// Configuração de roteamento ou qualquer outra lógica do seu framework

// Exemplo de como você pode carregar a rota para a página inicial
// Esta parte pode variar conforme a implementação do seu roteador
$uri = $_SERVER['REQUEST_URI'];

if ($uri === '/') {
    // Chamar controlador para a home page
    $controller = new \Src\Controllers\HomeController();
    $controller->index();
}

```

3. Carregar o CSS e os Arquivos Estáticos nas Views

Agora, para carregar o **CSS** que está em `public/assets/css/style.css` e garantir que o navegador acesse os arquivos estáticos corretamente, basta usar URLs relativas à pasta `public`.

Exemplo: Arquivo `src/views/templates/layout.twig`

```

html

<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ title }}</title>

    <!-- Carregar o arquivo CSS -->
    <link rel="stylesheet" href="/assets/css/style.css">
</head>

```

```
<body>

  <!-- Bloco de conteúdo será injetado aqui -->
  {% block content %} {% endblock %}

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-pzjw8f+ua7Kw1TIq0bb2dM1sbe0qL5lhpvcfgz5lhvXw9EoPuwRGTgLS5M9gqzd" crossorigin="anonymous"></script>
</body>
</html>
```

4. Acessando os Arquivos Estáticos no Navegador

- O caminho `/assets/css/style.css` refere-se ao diretório `public/assets/css/style.css`. O `public` é o diretório raiz acessível diretamente no navegador, então qualquer coisa dentro dele pode ser acessada diretamente via URL.
- Quando você usa `/assets/css/style.css`, o navegador procurará esse arquivo em `http://localhost/assets/css/style.css`, ou qualquer que seja o domínio configurado para o seu projeto.

5. Carregando o CSS e o JavaScript no HTML de Cada Página

Caso você precise carregar o CSS e outros arquivos JS apenas nas páginas específicas, você pode organizar o carregamento dessas dependências dentro do bloco `content` ou, ainda, como parte do template principal. Vamos dar um exemplo usando **Twig**.

Arquivo: `src/views/home.twig`

```
html

{% extends 'templates/layout.twig' %}

{% block content %}
  <!-- Incluir Cabeçalho -->
  {% include 'partials/header.twig' %}

  <div class="container mt-4">
    <h1>Bem-vindo, {{ user }}</h1>
    <p>Esta é a página inicial da nossa aplicação.</p>
  </div>

  <div class="row">
    <!-- Card 1 -->
    <div class="col-md-4">
      <div class="card">
do Card 1">
        
        <div class="card-body">
          <h5 class="card-title">Card 1</h5>
          <p class="card-text">Descrição do card 1. Um conteúdo interessante sobre o tema.</p>
          <a href="#" class="btn btn-primary">Ir para Card 1</a>
        </div>
      </div>
    </div>

    <!-- Card 2 -->
    <div class="col-md-4">
      <div class="card">
do Card 2">
        
        <div class="card-body">
          <h5 class="card-title">Card 2</h5>
```

```

a.</p>
    <p class="card-text">Descrição do card 2. Detalhes sobre um segundo tem
    <a href="#" class="btn btn-primary">Ir para Card 2</a>
  </div>
</div>
</div>

<!-- Card 3 -->
<div class="col-md-4">
  <div class="card">
    
    <div class="card-body">
      <h5 class="card-title">Card 3</h5>
      <p class="card-text">Descrição do card 3. Mais conteúdo interessante par
a o usuário.</p>
      <a href="#" class="btn btn-primary">Ir para Card 3</a>
    </div>
  </div>
</div>
</div>

<!-- Incluir Rodapé -->
{% include 'partials/footer.twig' %}
{% endblock %}

```

Conclusão

- O arquivo `style.css` localizado em `public/assets/css/style.css` pode ser carregado no HTML usando o caminho `/assets/css/style.css`.
- O diretório `public` é a raiz do seu servidor web, portanto, qualquer recurso dentro dele pode ser acessado diretamente via URL relativa à pasta `public`.
- Você pode incluir o **CSS** e **JS** no template `layout.twig`, ou de maneira específica nas views, conforme necessário.



namespace HefestoMVC, crie uma conexão PDO com banco de dados com esse namespace e usando `.env`



Para criar uma conexão PDO com banco de dados dentro do namespace `HefestoMVC`, usando um arquivo `.env` para configurar as variáveis de ambiente, siga o passo a passo abaixo.

1. Estrutura de Diretórios

Aqui está uma estrutura sugerida para o seu projeto:

```

bash

/project
  /public
    index.php
  /src
    /Config
      Database.php
    /Controllers
      HomeController.php

```

```

    /Models
      User.php
    /Views
      home.twig
  /vendor
  .env
  composer.json

```

2. Instalação do Composer e Bibliotecas

Primeiro, instale as dependências necessárias via Composer, incluindo o pacote `vlucas/phpdotenv` para ler variáveis de ambiente do arquivo `.env`:

```

bash

composer require vlucas/phpdotenv

```

3. Criando o arquivo `.env`

O arquivo `.env` vai armazenar as configurações de banco de dados (e outras variáveis de ambiente) de forma segura.

Arquivo: `.env`

```

env

DB_HOST=localhost
DB_NAME=seu_banco
DB_USER=seu_usuario
DB_PASS=sua_senha

```

4. Carregar as variáveis do `.env`

No `index.php` ou em outro arquivo de configuração, carregue as variáveis do `.env` com a ajuda da biblioteca `vlucas/phpdotenv`.

Arquivo: `public/index.php`

```

php

<?php

// Autoload do Composer
require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

// Iniciar a aplicação, exemplo:
use HefestoMVC\Config\Database;

$db = new Database();

```

```
$conn = $db->connect();

// Resto do código...
```

5. Configuração de Conexão com Banco de Dados (Classe Database)

Agora, vamos criar a classe `Database` dentro do namespace `HefestoMVC\Config` para gerenciar a conexão PDO.

Arquivo: `src/Config/Database.php`

```
php

<?php

namespace HefestoMVC\Config;

use PDO;
use PDOException;

class Database
{
    private $host;
    private $dbName;
    private $username;
    private $password;
    private $conn;

    public function __construct()
    {
        // Carregar as variáveis de ambiente
        $this->host = $_ENV['DB_HOST'];
        $this->dbName = $_ENV['DB_NAME'];
        $this->username = $_ENV['DB_USER'];
        $this->password = $_ENV['DB_PASS'];
    }

    public function connect()
    {
        // Verificar se a conexão já existe
        if ($this->conn === null) {
            try {
                // Criar a conexão PDO
                $dsn = "mysql:host={$this->host};dbname={$this->dbName};charset=utf8";
                $this->conn = new PDO($dsn, $this->username, $this->password);
                $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
                $this->conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
            } catch (PDOException $e) {
                // Caso ocorra um erro na conexão
                echo "Erro ao conectar ao banco de dados: " . $e->getMessage();
            }
        }

        // Retornar a conexão PDO
        return $this->conn;
    }
}
```

6. Usando a Conexão no Controlador

Agora, você pode usar a classe `Database` em seu controlador para acessar o banco de dados.

Arquivo: `src/Controllers/HomeController.php`

php

```
<?php

namespace HefestoMVC\Controllers;

use HefestoMVC\Config\Database;

class HomeController
{
    private $db;

    public function __construct()
    {
        $this->db = new Database();
    }

    public function index()
    {
        // Obter a conexão PDO
        $conn = $this->db->connect();

        // Exemplo de consulta ao banco de dados
        $stmt = $conn->query("SELECT * FROM users");
        $users = $stmt->fetchAll();

        // Passar dados para a view, por exemplo
        // Aqui você poderia usar um motor de template como Twig
        // Exemplo:
        echo "<pre>";
        print_r($users);
        echo "</pre>";
    }
}
```

7. Acessando a Conexão no `index.php`

No `index.php`, você pode acessar o controlador para testar a conexão e executar a lógica.

Arquivo: `public/index.php`

php

```
<?php

require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

use HefestoMVC\Controllers\HomeController;

// Criar instância do controlador
$controller = new HomeController();

// Chamar a ação desejada (exemplo: index)
$controller->index();
```

8. Conexão PDO e Execução de Consultas

No exemplo acima, dentro do controlador `HomeController`, a conexão PDO é criada através do método `connect()`. A consulta ao banco de dados é realizada usando o método `query()` e os resultados são obtidos usando `fetchAll()`.

9. Considerações Finais

- **Segurança:** As variáveis de ambiente no `.env` ajudam a manter informações sensíveis, como credenciais do banco de dados, seguras.
- **Flexibilidade:** A classe `Database` pode ser facilmente reutilizada em outros lugares do seu projeto.
- **PDO:** Usamos PDO para uma conexão segura e eficiente com o banco de dados.

Agora, você tem um sistema de conexão PDO com banco de dados dentro do namespace `HefestoMVC`, utilizando o arquivo `.env` para as configurações do banco de dados.



```
corrija <?php

class Connection {

    private $conn;

    public function connect(){
        if($this->conn === null){
            try{
                $dsn = "mysql:host={ $_ENV['DB_HOST']};dbname={$_ENV['DB_NAME']}";
                $username = $_ENV['DB_USER'];
                $password = $_ENV['DB_PASSWORD'];
                $this->conn = new PDO($dsn, $username, $password);
                $this->conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);
            }
            catch(PDOException $e){
                echo "Connection failed: " . $e->getMessage();
            }
        }
        return $this->conn;
    }
}
```



O código que você forneceu tem um pequeno erro de sintaxe na forma como você está concatenando as variáveis dentro da string do `dsn`. As variáveis dentro da string precisam estar corretamente interpoladas. Para corrigir isso, é necessário remover os espaços extra e garantir que as variáveis sejam interpoladas corretamente dentro das chaves `{ }`.

Aqui está a versão corrigida:

php

```
<?php

class Connection {

    private $conn;

    public function connect(){
        if ($this->conn === null) {
            try {
                // Corrigir a interpolação de variáveis
                $dsn = "mysql:host={$_ENV['DB_HOST']};dbname={$_ENV['DB_NAME']}";
                $username = $_ENV['DB_USER'];
                $password = $_ENV['DB_PASSWORD'];

                // Criar a conexão PDO
                $this->conn = new PDO($dsn, $username, $password);

                // Configurar o modo de recuperação de dados
                $this->conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);
            } catch (PDOException $e) {
                // Exibir erro de conexão
                echo "Connection failed: " . $e->getMessage();
            }
        }

        return $this->conn;
    }
}
```

Correções:

1. **Interpolação de variáveis:** O código anterior tinha espaços dentro das chaves da interpolação ``{\$_ENV['DB_HOST']}``, o que fazia com que as variáveis não fossem corretamente interpoladas. A forma correta de fazer isso é ``{\$_ENV['DB_HOST']}`` sem espaços.
2. **Melhoria na formatação:** Removi espaços extras para melhorar a legibilidade do código.

Agora, o código deve funcionar corretamente para estabelecer uma conexão PDO com o banco de dados, utilizando as configurações armazenadas no arquivo ``.env``.

Como usar:

Para que o código funcione, você precisa garantir que as variáveis de ambiente estejam carregadas corretamente a partir do arquivo ``.env``, por exemplo, usando a biblioteca ``vlucas/phpdotenv``.

Aqui está um exemplo de como carregar o ``.env`` no início da execução do seu código:

php

```
// No início do seu arquivo
require_once __DIR__ . '/vendor/autoload.php';

$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();

// Agora você pode usar o Connection
$connection = new Connection();
$conn = $connection->connect();
```

Isso garante que as variáveis de ambiente sejam carregadas e estejam disponíveis para uso dentro do seu código.



deixe ela estatica



Para tornar a conexão **estática** (ou seja, para que você possa acessar o método `connect()` sem instanciar a classe), você pode modificar o método `connect()` e a propriedade `$conn` para serem estáticos.

Aqui está a versão corrigida e modificada para ser estática:

```
php

<?php

class Connection {

    private static $conn;

    // Tornar o método e a propriedade estáticos
    public static function connect() {
        if (self::$conn === null) {
            try {
                // Corrigir a interpolação de variáveis
                $dsn = "mysql:host={$_ENV['DB_HOST']};dbname={$_ENV['DB_NAME']}";
                $username = $_ENV['DB_USER'];
                $password = $_ENV['DB_PASSWORD'];

                // Criar a conexão PDO
                self::$conn = new PDO($dsn, $username, $password);

                // Configurar o modo de recuperação de dados
                self::$conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);
            } catch (PDOException $e) {
                // Exibir erro de conexão
                echo "Connection failed: " . $e->getMessage();
            }
        }

        return self::$conn;
    }
}
```

Alterações Feitas:

1. **Propriedade `$conn`**: Agora é `private static $conn`, o que significa que ela será compartilhada por todas as instâncias da classe, ou seja, uma única conexão será reutilizada.
2. **Método `connect()`**: Tornado estático, com o uso de `self::$conn` em vez de `$this->conn`, porque não estamos mais instanciando a classe.
3. **Uso de `self::`**: Para acessar a variável estática `$conn` dentro da própria classe, utilizamos `self::$conn` em vez de `$this->conn`.

Como Usar:

Agora você pode usar o método `connect()` sem precisar criar uma instância da classe. Basta chamar o método de forma estática, assim:

```
php

// Carregar as variáveis de ambiente do .env
require_once __DIR__ . '/vendor/autoload.php';
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();

// Usar a conexão estática
$conn = Connection::connect();

// Agora você pode usar a variável $conn como sua conexão PDO
```

Benefícios:

- **Conexão reutilizável:** A conexão com o banco de dados é criada uma vez e reutilizada em todo o código, o que economiza recursos.
- **Acesso fácil:** Como o método `connect()` é estático, você não precisa instanciar a classe para acessar a conexão, basta chamar `Connection::connect()` diretamente.



agora vamos fazer um crud. Comece como find, findAll e findby devendo ser usada para qualquer tipo de dados



Para criar um CRUD básico em PHP usando PDO e o conceito de métodos `find()`, `findAll()` e `findBy()`, vamos criar uma classe genérica que pode ser usada para qualquer tipo de dado. Vou criar uma classe base que se conecta ao banco de dados e fornece esses métodos de forma reutilizável para qualquer tabela do banco.

1. Classe `Database` para Conexão com o Banco

Primeiro, vamos garantir que temos uma classe de conexão PDO estática (como fizemos anteriormente).

```
php

<?php

class Connection {

    private static $conn;

    // Tornar o método e a propriedade estáticos
    public static function connect() {
        if (self::$conn === null) {
            try {
                // Corrigir a interpolação de variáveis
                $dsn = "mysql:host={$_ENV['DB_HOST']};dbname={$_ENV['DB_NAME']}";
                $username = $_ENV['DB_USER'];
                $password = $_ENV['DB_PASSWORD'];
```

```

        // Criar a conexão PDO
        self::$conn = new PDO($dsn, $username, $password);

        // Configurar o modo de recuperação de dados
        self::$conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_OBJ);
    } catch (PDOException $e) {
        // Exibir erro de conexão
        echo "Connection failed: " . $e->getMessage();
    }
}

return self::$conn;
}
}

```

2. Classe Genérica para CRUD

Agora vamos criar uma classe genérica que possa ser usada para realizar operações de `find`, `findAll` e `findBy`. Essa classe irá permitir que você pesquise qualquer tabela, fornecendo apenas o nome da tabela e os parâmetros necessários para a consulta.

Arquivo: `src/Models/Model.php`

php

```

<?php

namespace HefestoMVC\Models;

use HefestoMVC\Config\Database;

class Model {

    protected $table;
    protected $conn;

    public function __construct($table) {
        $this->table = $table;
        $this->conn = Database::connect(); // Conectar ao banco de dados
    }

    // Buscar um registro pela chave primária (ID)
    public function find($id) {
        $query = "SELECT * FROM {$this->table} WHERE id = :id LIMIT 1";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(); // Retorna o registro como objeto
    }

    // Buscar todos os registros da tabela
    public function findAll() {
        $query = "SELECT * FROM {$this->table}";
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros como array de objetos
    }

    // Buscar registros com base em uma condição
    public function findBy($field, $value) {
        $query = "SELECT * FROM {$this->table} WHERE {$field} = :value";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':value', $value, \PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros encontrados
    }
}

```

```
}
}
```

Explicação:

1. **Construtor:** O construtor recebe o nome da tabela que será manipulada, o que torna essa classe reutilizável para qualquer tabela. Ele também inicializa a conexão com o banco de dados.
2. `find()`: Método para buscar um registro com base no ID (assumindo que o ID é a chave primária da tabela).
3. `findAll()`: Método para buscar todos os registros de uma tabela.
4. `findBy()`: Método para buscar registros com base em uma condição (`campo`, `valor`).

3. Usando a Classe Genérica em um Controller

Agora, podemos criar um controlador para testar os métodos de `find`, `findAll` e `findBy` com uma tabela de exemplo (digamos `users`).

Arquivo: `src/Controllers/UserController.php`

```
php

<?php

namespace HefestoMVC\Controllers;

use HefestoMVC\Models\Model;

class UserController {

    private $userModel;

    public function __construct() {
        // Criar instância do modelo User com a tabela 'users'
        $this->userModel = new Model('users');
    }

    // Método para mostrar todos os usuários
    public function index() {
        $users = $this->userModel->findAll();
        echo "<pre>";
        print_r($users);
        echo "</pre>";
    }

    // Método para mostrar um único usuário com base no ID
    public function show($id) {
        $user = $this->userModel->find($id);
        echo "<pre>";
        print_r($user);
        echo "</pre>";
    }

    // Método para mostrar usuários com base em um campo (ex: email)
    public function findByEmail($email) {
        $users = $this->userModel->findBy('email', $email);
        echo "<pre>";
        print_r($users);
        echo "</pre>";
    }
}
```

4. Testando no `index.php`

Agora, no arquivo `public/index.php`, você pode testar os métodos do controlador.

Arquivo: `public/index.php`

```
php

<?php

require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

use HefestoMVC\Controllers\UserController;

$controller = new UserController();

// Mostrar todos os usuários
$controller->index();

// Mostrar um usuário com ID 1
$controller->show(1);

// Buscar usuários por email
$controller->findByEmail('example@example.com');
```

5. Considerações Finais

Com essa estrutura, você tem uma classe genérica `Model` que pode ser usada para fazer consultas (`find`, `findAll`, `findBy`) em qualquer tabela do banco de dados. Basta passar o nome da tabela ao criar a instância do modelo, e você pode facilmente modificar os métodos para incluir mais operações de CRUD conforme necessário.

Expansões:

- Para um CRUD completo, você pode adicionar métodos como `create()`, `update()`, e `delete()` na classe `Model` ou em uma classe derivada específica.
- A classe `Model` pode ser estendida para adicionar métodos personalizados para tabelas específicas.

Dessa forma, o código é flexível e reutilizável, permitindo que você trabalhe com qualquer tabela do banco de dados.



E a camada models (src/database/models) não será usada?



Sim, você pode estruturar sua aplicação de forma que a camada de **Models** seja separada e organizada, garantindo que o código fique mais modular e fácil de manter. A camada **Models** deve representar as

tabelas do banco de dados, com métodos para realizar operações CRUD específicas para cada tipo de dado. E você pode usar a classe **Model** genérica que criamos anteriormente dentro dessa camada para fazer as consultas de forma reutilizável.

Vamos reorganizar o código de acordo com essa arquitetura de pastas:

1. **Diretório** ``src/Database/Models``: Onde você colocará os modelos específicos para cada tabela do banco de dados, como ``User``, ``Post``, etc. Esses modelos irão estender a classe genérica ``Model``.
2. **Classe genérica** ``Model``: A classe base ``Model`` continua a ser genérica, com métodos como ``find()``, ``findAll()``, e ``findBy()``, que você pode usar em qualquer tabela.

Estrutura do Projeto:

```
bash

/src
  /Database
    /Models
      - User.php
      - Post.php
  /Controllers
    - UserController.php
  /Config
    - Database.php
  /Views
  /public
  .env
  /vendor
```

1. Classe Genérica ``Model`` (`src/Database/Models/Model.php`)

Esta classe ainda será genérica, mas agora ela está em ``src/Database/Models``.

```
php

<?php

namespace HefestoMVC\Database\Models;

use HefestoMVC\Config\Database;

class Model {

    protected $table;
    protected $conn;

    public function __construct($table) {
        $this->table = $table;
        $this->conn = Database::connect(); // Conectar ao banco de dados
    }

    // Buscar um registro pela chave primária (ID)
    public function find($id) {
        $query = "SELECT * FROM {$this->table} WHERE id = :id LIMIT 1";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(); // Retorna o registro como objeto
    }

    // Buscar todos os registros da tabela
```

```

public function findAll() {
    $query = "SELECT * FROM {$this->table}";
    $stmt = $this->conn->prepare($query);
    $stmt->execute();
    return $stmt->fetchAll(); // Retorna todos os registros como array de objetos
}

// Buscar registros com base em uma condição
public function findBy($field, $value) {
    $query = "SELECT * FROM {$this->table} WHERE {$field} = :value";
    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':value', $value, \PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetchAll(); // Retorna todos os registros encontrados
}
}

```

2. Model Específico: `User.php` (src/Database/Models/User.php)

Aqui, vamos criar um modelo específico para a tabela `users`. Esse modelo irá estender a classe `Model` para herdar todos os métodos genéricos.

```

php

<?php

namespace HefestoMVC\Database\Models;

class User extends Model {

    public function __construct() {
        // Definir o nome da tabela como 'users'
        parent::__construct('users');
    }

    // Métodos personalizados para a tabela 'users' podem ser adicionados aqui
    public function findByEmail($email) {
        return $this->findBy('email', $email);
    }
}

```

3. Controlador `UserController.php` (src/Controllers/UserController.php)

Agora, vamos usar a classe `User` dentro do controlador para realizar as operações de busca. Esse controlador usará o modelo `User` para buscar dados da tabela `users`.

```

php

<?php

namespace HefestoMVC\Controllers;

use HefestoMVC\Database\Models\User;

class UserController {

    private $userModel;

    public function __construct() {
        // Criar uma instância do modelo User
        $this->userModel = new User();
    }
}

```



```
// Mostrar todos os usuários
public function index() {
    $users = $this->userModel->findAll();
    echo "<pre>";
    print_r($users);
    echo "</pre>";
}

// Mostrar um único usuário com base no ID
public function show($id) {
    $user = $this->userModel->find($id);
    echo "<pre>";
    print_r($user);
    echo "</pre>";
}

// Buscar usuários por email
public function findByEmail($email) {
    $users = $this->userModel->findByEmail($email);
    echo "<pre>";
    print_r($users);
    echo "</pre>";
}
}
```

4. Testando no `index.php` (public/index.php)

No `index.php`, agora você pode chamar os métodos de busca definidos no controlador para testar a funcionalidade.

```
php

<?php

require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

use HefestoMVC\Controllers\UserController;

$controller = new UserController();

// Mostrar todos os usuários
$controller->index();

// Mostrar um usuário com ID 1
$controller->show(1);

// Buscar usuários por email
$controller->findByEmail('example@example.com');
```

5. Expansão para Outras Tabelas

A classe `Model` pode ser reutilizada para qualquer tabela. Por exemplo, se você quiser criar um modelo para a tabela `posts`, basta criar um arquivo `Post.php` dentro de `src/Database/Models` e estender a classe `Model`.

php

```
<?php

namespace HefestoMVC\Database\Models;

class Post extends Model {

    public function __construct() {
        parent::__construct('posts'); // Definir o nome da tabela
    }

    // Métodos personalizados para a tabela 'posts'
}
```

6. Vantagens dessa Estrutura

- **Reutilização:** Com a classe `Model`, você pode criar modelos específicos para qualquer tabela, reutilizando os métodos de busca genéricos.
- **Organização:** A estrutura do projeto agora está mais organizada, com modelos separados para cada tabela e operações CRUD específicas.
- **Facilidade de manutenção:** Como o código é modular, você pode facilmente expandir e adicionar novos métodos ou tabelas sem mexer nas partes que já estão funcionando.

Essa abordagem permite que você crie e organize facilmente os modelos de dados no seu projeto, mantendo o código limpo e organizado.



melhore os controllers. Use Container para não usar algo como `private $userModel;`

```
public function __construct() {
    // Criar uma instância do modelo User
    $this->userModel = new User();
} e adapte para chamar os dados da tabela e armazenar e exibir na view com twig
```



Para melhorar os **Controllers** e utilizar um **Container** de dependências (como o **Pimple** ou **PHP-DI**), podemos seguir uma abordagem mais flexível, onde os modelos são injetados automaticamente, sem a necessidade de inicializá-los manualmente em cada controlador. Além disso, vamos adaptar para usar **Twig** para renderizar as views e exibir os dados.

Abaixo, mostro como implementar isso, começando com a criação de um **Container** de dependências e depois utilizando o **Twig** para renderizar as views.

1. Container de Dependências

Primeiro, criamos o **Container** para gerenciar a injeção de dependências, garantindo que o controlador não precise instanciar diretamente os modelos.

Você pode usar o **PHP-DI** ou o **Pimple**. Aqui, vamos usar o **PHP-DI**.

Instalação do PHP-DI (via Composer):

```
bash

composer require php-di/php-di
```

Agora, vamos configurar o **Container** para gerenciar dependências.

Arquivo: `src/Config/Container.php`

```
php

<?php

namespace HefestoMVC\Config;

use DI\ContainerBuilder;
use HefestoMVC\Database\Models\User;
use HefestoMVC\Database\Models\Post;

class Container {

    public static function getContainer() {
        $containerBuilder = new ContainerBuilder();
        $containerBuilder->addDefinitions([
            // Definindo a injeção de dependência dos modelos
            User::class => \DI\autowire(User::class),
            Post::class => \DI\autowire(Post::class),
            // Definindo o Twig como dependência
            \Twig\Environment::class => function() {
                $loader = new \Twig\Loader\FilesystemLoader(__DIR__ . '/../views');
                return new \Twig\Environment($loader);
            },
        ]);
        return $containerBuilder->build();
    }
}
```

2. Controlador Usando o Container

Agora, vamos refatorar o controlador para usar o **Container** para injeção de dependência. O controlador vai buscar os dados usando os modelos e passar para a view Twig para renderização.

Arquivo: `src/Controllers/UserController.php`

```
php

<?php

namespace HefestoMVC\Controllers;

use HefestoMVC\Database\Models\User;
use HefestoMVC\Config\Container;
use Twig\Environment;

class UserController {

    private $userModel;
    private $twig;
```

```
public function __construct(User $userModel, Environment $twig) {
    // O container injeta automaticamente as dependências
    $this->userModel = $userModel;
    $this->twig = $twig;
}

// Mostrar todos os usuários
public function index() {
    $users = $this->userModel->findAll();

    // Renderizar a view com os dados
    echo $this->twig->render('user/index.twig', ['users' => $users]);
}

// Mostrar um único usuário com base no ID
public function show($id) {
    $user = $this->userModel->find($id);

    // Renderizar a view com o usuário
    echo $this->twig->render('user/show.twig', ['user' => $user]);
}

// Buscar usuários por email
public function findByEmail($email) {
    $users = $this->userModel->findByEmail($email);

    // Renderizar a view com os resultados
    echo $this->twig->render('user/index.twig', ['users' => $users]);
}
}
```

3. View com Twig

Agora, vamos criar as views para exibir os dados. Vamos criar a view `user/index.twig` e `user/show.twig` para listar e exibir os detalhes de cada usuário.

Arquivo: `src/views/user/index.twig`

```
twig

<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Usuários</title>
    <link href="/assets/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">HefestoMVC</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active"><a class="nav-link" href="/">Home</a></li>
            <li class="nav-item"><a class="nav-link" href="/users">Usuários</a></li>
        </ul>
    </nav>

    <div class="container mt-4">
        <h1>Lista de Usuários</h1>
        <div class="row">
            {% for user in users %}
            <div class="col-md-4">
                <div class="card">
                    <div class="card-body">
                        <h5 class="card-title">{{ user.name }}</h5>
                        <p class="card-text">{{ user.email }}</p>
                    </div>
                </div>
            </div>
            </div>
        </div>
    </div>
</body>
</html>
```

```

        <a href="/user/show/{{ user.id }}" class="btn btn-primary">Ver Detal
hes</a>
    </div>
</div>
</div>
{% endfor %}
</div>
</div>
</body>
</html>

```

Arquivo: `src/views/user/show.twig`

```

twig

<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ user.name }} - Detalhes</title>
    <link href="/assets/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">HefestoMVC</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active"><a class="nav-link" href="/">Home</a></li>
            <li class="nav-item"><a class="nav-link" href="/users">Usuários</a></li>
        </ul>
    </nav>

    <div class="container mt-4">
        <h1>Detalhes de {{ user.name }}</h1>
        <p><strong>Email:</strong> {{ user.email }}</p>
        <p><strong>Data de Criação:</strong> {{ user.created_at }}</p>
    </div>
</body>
</html>

```

4. Atualizando o `index.php` para Usar o Container

No arquivo `public/index.php`, vamos inicializar o **Container** e invocar o controlador com as dependências injetadas.

Arquivo: `public/index.php`

```

php

<?php

require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

use HefestoMVC\Config\Container;
use HefestoMVC\Controllers\UserController;

// Criar o container de dependências
$container = Container::getContainer();

```

```
// Rota para listar usuários
if ($_SERVER['REQUEST_URI'] == '/users') {
    $controller = $container->get(UserController::class);
    $controller->index();
}

// Rota para mostrar um usuário específico
elseif (preg_match('/\/user\/show\/(\d+)/', $_SERVER['REQUEST_URI'], $matches)) {
    $controller = $container->get(UserController::class);
    $controller->show($matches[1]);
}

// Buscar por email (exemplo de rota)
elseif (preg_match('/\/user\/email\/(.+)/', $_SERVER['REQUEST_URI'], $matches)) {
    $controller = $container->get(UserController::class);
    $controller->findByEmail($matches[1]);
}
```

5. Considerações Finais

- **Injeção de Dependência:** Agora, o controlador `UserController` não precisa instanciar manualmente o modelo. O **Container** faz isso automaticamente.
- **Twig para Views:** Usamos o **Twig** para renderizar as views, passando os dados diretamente do modelo para a view.
- **Flexibilidade:** Você pode criar outros controladores (para outros modelos como `Post`, `Comment`, etc.) e seguir a mesma abordagem, usando o container para injeção de dependências.

Essa estrutura deixa o código mais limpo, modular e fácil de manter. Além disso, a utilização do **Container** facilita a gestão de dependências e torna a aplicação mais flexível, permitindo que você gerencie as dependências de forma centralizada.



Minha tabela é articles e já tenho a classe View em /library. Container.php criei em library também

```
<?php
namespace HefestoMVCLibrary;
```

```
use Twig\Environment;
use Twig\Loader\FilesystemLoader;
```

```
class View
{
    public static function render(string $view, array $data = [])
    {
        $path = dirname(__FILE__) . '/../src/views/';

        $loader = new FilesystemLoader($path);
        $twig = new Environment($loader);

        echo $twig->render($view, $data);
    }
}
```

```
}
}
```



Com a classe ``View`` que você já criou em ``/Library``, podemos refatorar o código para usá-la para renderizar as views com o **Twig**, em vez de criar uma nova instância do **Twig** diretamente nos controladores. Além disso, podemos integrar a tabela ``articles`` e a classe ``Model`` para realizar as operações de CRUD para os artigos.

Vamos organizar isso com a abordagem sugerida. A estrutura do seu código será mais limpa e reutilizável, usando a classe ``View`` já existente para renderizar as views.

1. Modelo ``Article`` (`src/Database/Models/Article.php`)

Vamos criar o modelo para a tabela ``articles``. Esse modelo irá estender a classe genérica ``Model`` e usará os métodos para buscar dados da tabela ``articles``.

```
php

<?php

namespace HefestoMVC\Database\Models;

use HefestoMVCLibrary\Connection;

class Article extends Model {

    public function __construct() {
        parent::__construct('articles'); // Definindo a tabela 'articles'
    }

    // Métodos personalizados para a tabela 'articles' podem ser adicionados aqui
    public function findByTitle($title) {
        return $this->findBy('title', $title);
    }
}
```

2. Controlador ``ArticleController.php`` (`src/Controllers/ArticleController.php`)

Agora, vamos criar o controlador que irá lidar com as rotas de exibição e manipulação dos artigos. O controlador usará o **Container** para injetar as dependências, como o modelo ``Article``, e usará a classe ``View`` para renderizar as views.

Arquivo: ``src/Controllers/ArticleController.php``

```
php

<?php

namespace HefestoMVC\Controllers;

use HefestoMVC\Database\Models\Article;
use HefestoMVCLibrary\View;

class ArticleController {
```

```
private $articleModel;

public function __construct(Article $articleModel) {
    $this->articleModel = $articleModel;
}

// Mostrar todos os artigos
public function index() {
    $articles = $this->articleModel->findAll();

    // Passar os dados para a view usando a classe View
    View::render('article/index.twig', ['articles' => $articles]);
}

// Mostrar um artigo específico com base no ID
public function show($id) {
    $article = $this->articleModel->find($id);

    // Passar os dados do artigo para a view
    View::render('article/show.twig', ['article' => $article]);
}

// Buscar artigos por título
public function findByTitle($title) {
    $articles = $this->articleModel->findByTitle($title);

    // Passar os artigos encontrados para a view
    View::render('article/index.twig', ['articles' => $articles]);
}
}
```

3. View `index.twig` (src/views/article/index.twig)

Agora, vamos criar a view para listar os artigos. Essa view será renderizada pelo controlador com os dados dos artigos.

Arquivo: `src/views/article/index.twig`

```
twig

<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Artigos</title>
    <link href="/assets/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">HefestoMVC</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active"><a class="nav-link" href="/">Home</a></li>
            <li class="nav-item"><a class="nav-link" href="/articles">Artigos</a></li>
        </ul>
    </nav>

    <div class="container mt-4">
        <h1>Lista de Artigos</h1>
        <div class="row">
            {% for article in articles %}
            <div class="col-md-4">
                <div class="card">
                    <div class="card-body">
                        <h5 class="card-title">{{ article.title }}</h5>
                        <p class="card-text">{{ article.excerpt }}</p>
                    </div>
                </div>
            </div>
            </div>
        </div>
    </div>
</body>
</html>
```



```

        <a href="/article/show/{{ article.id }}" class="btn btn-primary">Ver
    Detalhes</a>
    </div>
</div>
</div>
{% endfor %}
</div>
</div>
</body>
</html>

```

4. View `show.twig` (src/views/article/show.twig)

Agora, vamos criar a view para exibir os detalhes de um artigo específico.

Arquivo: `src/views/article/show.twig`

```

twig

<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ article.title }} - Detalhes</title>
    <link href="/assets/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">HefestoMVC</a>
        <ul class="navbar-nav mr-auto">
            <li class="nav-item active"><a class="nav-link" href="/">Home</a></li>
            <li class="nav-item"><a class="nav-link" href="/articles">Artigos</a></li>
        </ul>
    </nav>

    <div class="container mt-4">
        <h1>{{ article.title }}</h1>
        <p><strong>Publicado em:</strong> {{ article.created_at }}</p>
        <div>
            <p>{{ article.content }}</p>
        </div>
    </div>
</body>
</html>

```

5. Atualizando o `index.php` para Usar o Container e as Rotas

Agora, vamos configurar as rotas no arquivo `public/index.php` para que o controlador correto seja chamado e os dados sejam passados para as views.

Arquivo: `public/index.php`

```

php

<?php
require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente

```

```
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

use HefestoMVC\Config\Container;
use HefestoMVC\Controllers\ArticleController;

// Criar o container de dependências
$container = Container::getContainer();

// Rota para listar artigos
if ($_SERVER['REQUEST_URI'] == '/articles') {
    $controller = $container->get(ArticleController::class);
    $controller->index();
}

// Rota para mostrar um artigo específico
elseif (preg_match('/\article/show\/(\d+)/', $_SERVER['REQUEST_URI'], $matches)) {
    $controller = $container->get(ArticleController::class);
    $controller->show($matches[1]);
}

// Buscar por título (exemplo de rota)
elseif (preg_match('/\article\/title\/(.+)/', $_SERVER['REQUEST_URI'], $matches)) {
    $controller = $container->get(ArticleController::class);
    $controller->findByTitle($matches[1]);
}
```

6. Considerações Finais

1. **Injeção de Dependência:** O controlador agora usa o **Container** para injetar o modelo `Article`, evitando a necessidade de instanciá-lo diretamente.
2. **Renderização com Twig:** A classe `View` é usada para renderizar as views, passando os dados necessários para as templates do Twig.
3. **Rotas e Controladores:** As rotas foram configuradas para chamar os métodos do controlador `ArticleController`, e as views foram organizadas para exibir os dados dos artigos de forma estruturada.
4. **Flexibilidade:** A abordagem com o **Container** permite que você facilmente adicione novos controladores e modelos no futuro, mantendo o código organizado e fácil de manter.

Essa estrutura modulariza a aplicação, separando as responsabilidades de forma mais eficiente e permitindo um desenvolvimento mais escalável.



E o container? Precisa ser refatorado



Sim, o `Container` precisa ser ajustado para garantir que ele forneça corretamente as dependências e funcione de forma eficiente. O objetivo é garantir que ele possa injetar as dependências necessárias (como os modelos) nos controladores e que seja fácil de escalar.

Vamos refatorar o `Container` para usar um padrão simples de **injeção de dependência**, o que pode ser feito utilizando a técnica de **singleton** ou **factory methods**.

1. Refatorando o Container

A classe `Container` irá manter um registro das instâncias e ser responsável por gerenciar a criação dos objetos. Ela vai verificar se a classe já foi instanciada antes de criar uma nova instância.

Arquivo: `library/Container.php`

```
php

<?php

namespace HefestoMVCLibrary;

use ReflectionClass;
use Exception;

class Container
{
    // Array para armazenar as instâncias dos objetos
    private static $instances = [];

    /**
     * Método para obter a instância de uma classe
     *
     * @param string $class
     * @return object
     * @throws Exception
     */
    public static function get(string $class)
    {
        // Verifica se a instância já foi criada
        if (!isset(self::$instances[$class])) {
            self::$instances[$class] = self::create($class);
        }

        return self::$instances[$class];
    }

    /**
     * Criação de instância de uma classe
     *
     * @param string $class
     * @return object
     * @throws Exception
     */
    private static function create(string $class)
    {
        try {
            // Usando Reflection para instanciar a classe com as dependências
            $reflection = new ReflectionClass($class);

            // Verificando se a classe tem um construtor
            if ($constructor = $reflection->getConstructor()) {
                // Obtém as dependências do construtor
                $dependencies = self::getDependencies($constructor);

                // Cria uma nova instância com as dependências
                return $reflection->newInstanceArgs($dependencies);
            }

            // Se a classe não tiver um construtor, instanciamos diretamente
            return new $class;
        } catch (Exception $e) {
            throw new Exception("Não foi possível criar a instância de $class: " . $e->getMessage());
        }
    }

    /**
     * Obtém as dependências de uma classe através de seu construtor
     *
     * @param \ReflectionMethod $constructor
     */
}
```

```

    * @return array
    */
    private static function getDependencies(\ReflectionMethod $constructor)
    {
        $dependencies = [];
        foreach ($constructor->getParameters() as $param) {
            // Verifica se a dependência é uma classe
            if ($class = $param->getClass()) {
                // Se for, obtemos a instância do Container
                $dependencies[] = self::get($class->getName());
            } else {
                // Se não for uma classe, adicionamos null (pode ser tratado com valores pad
                $dependencies[] = null;
            }
        }
        return $dependencies;
    }
}

```

2. Explicação da Refatoração do Container

- **Armazenamento de Instâncias:** O `Container` agora mantém um array estático `$instances` que armazena instâncias já criadas para evitar múltiplas instâncias da mesma classe.
- **Método `get`:** Esse método verifica se a instância da classe já foi criada. Se não, ele chama o método `create` para instanciá-la. Caso a classe tenha dependências (como o modelo `Article`), elas serão resolvidas automaticamente.
- **Método `create`:** Usamos o **Reflection** para inspecionar o construtor da classe e identificar as dependências. Essas dependências são passadas para a instância da classe.
- **Método `getDependencies`:** Esse método pega as dependências do construtor da classe e resolve as instâncias necessárias, chamando o método `get` recursivamente para cada classe.

3. Ajuste no `ArticleController`

Agora, ao invés de instanciar o modelo `Article` diretamente no controlador, o **Container** cuidará disso para você. O controlador não precisa mais conhecer a lógica de criação das dependências.

Arquivo: `src/Controllers/ArticleController.php`

```

php

<?php

namespace HefestoMVC\Controllers;

use HefestoMVC\Database\Models\Article;
use HefestoMVCLibrary\View;

class ArticleController {

    private $articleModel;

    // O modelo será injetado automaticamente pelo Container
    public function __construct(Article $articleModel) {
        $this->articleModel = $articleModel;
    }

    // Mostrar todos os artigos

```

```

public function index() {
    $articles = $this->articleModel->findAll();

    // Passar os dados para a view usando a classe View
    View::render('article/index.twig', ['articles' => $articles]);
}

// Mostrar um artigo específico com base no ID
public function show($id) {
    $article = $this->articleModel->find($id);

    // Passar os dados do artigo para a view
    View::render('article/show.twig', ['article' => $article]);
}

// Buscar artigos por título
public function findByTitle($title) {
    $articles = $this->articleModel->findByTitle($title);

    // Passar os artigos encontrados para a view
    View::render('article/index.twig', ['articles' => $articles]);
}
}

```

4. Uso do Container no `index.php`

No arquivo `public/index.php`, o Container será usado para injetar as dependências automaticamente nos controladores.

Arquivo: `public/index.php`

php

```

<?php

require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

use HefestoMVC\Config\Container;
use HefestoMVC\Controllers\ArticleController;

// Criar o container de dependências
$container = Container::getContainer();

// Rota para listar artigos
if ($_SERVER['REQUEST_URI'] == '/articles') {
    $controller = $container->get(ArticleController::class);
    $controller->index();
}

// Rota para mostrar um artigo específico
elseif (preg_match('/\/article\/show\/(\d+)/', $_SERVER['REQUEST_URI'], $matches)) {
    $controller = $container->get(ArticleController::class);
    $controller->show($matches[1]);
}

// Buscar por título (exemplo de rota)
elseif (preg_match('/\/article\/title\/(.+)/', $_SERVER['REQUEST_URI'], $matches)) {
    $controller = $container->get(ArticleController::class);
    $controller->findByTitle($matches[1]);
}
}

```

5. Conclusão

Com a refatoração do **Container**, agora as dependências (como o modelo ``Article``) são injetadas automaticamente nos controladores através do construtor. O **Container** cuida da criação e resolução das dependências, mantendo o código mais limpo, escalável e fácil de manter.

Você pode continuar a adicionar mais controladores, modelos e outras dependências conforme necessário, sem a necessidade de alterar a lógica de instância de objetos manualmente em cada controlador.



Ele usa o php/DI ou não?



Sim, o ``Container`` que criamos pode ser considerado uma **implementação simples** de um contêiner de **injeção de dependência (DI)**, mas não é uma implementação completa de um framework de DI como o **PHP-DI** ou **Symfony DependencyInjection**. Ele segue o **princípio de injeção de dependência** ao resolver e fornecer instâncias de classes e suas dependências, sem que você precise criar manualmente as instâncias em cada controlador.

Comparação com DI mais robustos (como PHP-DI):

- **PHP-DI e Symfony DI** fornecem recursos avançados, como:
 - **Escopo de vida das instâncias** (singleton, instâncias por requisição, etc.)
 - **Autowiring automático** para classes e parâmetros
 - **Configuração de parâmetros complexos** (ex.: valores do `` .env ``)
 - **Facilidade de configuração** através de arquivos YAML ou XML

Enquanto isso, a solução que criamos com o ``Container`` é uma implementação bem simples e manual, em que a injeção de dependência ocorre de maneira automática por meio de **Reflection** para verificar e resolver as dependências das classes.

O que nosso ``Container`` faz:

- **Injeção automática de dependências:** Ao usar o construtor de cada classe, o ``Container`` analisa as dependências da classe e as resolve. Ou seja, se o controlador ``ArticleController`` espera uma instância do modelo ``Article``, o ``Container`` cuida de passá-la sem que você precise instanciá-la manualmente.
- **Singleton:** O ``Container`` armazena instâncias de objetos já criados para evitar múltiplas instâncias da mesma classe durante o ciclo de vida da aplicação.

Diferenças principais com DI frameworks avançados:

1. **Escopo:** O DI framework pode controlar o escopo das instâncias (singleton, transiente, etc.), enquanto nosso ``Container`` só cria uma instância por classe e a reutiliza.
2. **Configuração:** Frameworks como PHP-DI e Symfony DI oferecem mais opções de configuração, como autowiring, uso de arquivos de configuração (YAML, XML), e podem ser configurados para

funcionar com containers de serviços mais complexos.

3. **Autowiring:** No nosso caso, a **Reflection** permite injeção de dependência sem a necessidade de declarar explicitamente os parâmetros. No entanto, frameworks avançados oferecem mais flexibilidade e capacidade de personalização nesse sentido.

Conclusão:

O **Container** que criamos não é uma implementação "completa" de um framework de DI, mas **implementa os conceitos básicos de DI**, como injeção de dependência no construtor e resolução automática das dependências através de reflection. Ele pode ser considerado uma forma simplificada de DI, adequada para aplicações menores e mais simples. Para aplicações maiores e mais complexas, frameworks dedicados de **injeção de dependência** (como PHP-DI ou Symfony DI) seriam mais adequados.



if (\$class = \$param->getClass()) is deprecated



Você está correto! O método `getClass()` da classe `ReflectionParameter` foi descontinuado (deprecated) no PHP 8.0.0. A recomendação é usar o método `getType()` para obter o tipo do parâmetro, que funciona para qualquer tipo de dados, incluindo classes.

Vamos atualizar o código para usar o `getType()` de forma correta.

Alteração no método `getDependencies` do `Container`

O método `getType()` retorna um objeto `ReflectionNamedType`, e você pode verificar se o tipo é uma classe através de `isBuiltin()` e `getName()`.

Aqui está a correção no código:

Código Atualizado

```
php

<?php

namespace HefestoMVCLibrary;

use ReflectionClass;
use ReflectionNamedType;
use Exception;

class Container
{
    private static $instances = [];

    public static function get(string $class)
    {
        // Verifica se a instância já foi criada
        if (!isset(self::$instances[$class])) {
            self::$instances[$class] = self::create($class);
        }
    }
}
```

```

        return self::$instances[$class];
    }

    private static function create(string $class)
    {
        try {
            // Usando Reflection para instanciar a classe com as dependências
            $reflection = new ReflectionClass($class);

            if ($constructor = $reflection->getConstructor()) {
                $dependencies = self::getDependencies($constructor);
                return $reflection->newInstanceArgs($dependencies);
            }

            return new $class;
        } catch (Exception $e) {
            throw new Exception("Não foi possível criar a instância de $class: " . $e->getMessage());
        }
    }

    private static function getDependencies(\ReflectionMethod $constructor)
    {
        $dependencies = [];

        foreach ($constructor->getParameters() as $param) {
            $type = $param->getType();

            // Verifica se o tipo do parâmetro é uma classe (ou interface)
            if ($type instanceof ReflectionNamedType && !$type->isBuiltin()) {
                $dependencies[] = self::get($type->getName()); // Resolve a dependência
            } else {
                // Se o parâmetro não for uma classe, podemos adicionar null ou outro valor padrão
                $dependencies[] = null;
            }
        }

        return $dependencies;
    }
}

```

Explicação das Alterações:

- **Uso de `getType()`**: O método `getType()` retorna o tipo do parâmetro como um `ReflectionType`. Quando o tipo é uma classe ou interface, ele retornará um objeto `ReflectionNamedType`. Usamos isso para verificar se o parâmetro é uma classe e, em seguida, obtemos o nome da classe com `\$type->getName()`.
- **Verificação de `isBuiltin()`**: Para garantir que o parâmetro seja uma classe e não um tipo básico (como `int`, `string`, etc.), usamos `\$type->isBuiltin()` para verificar se o tipo é nativo do PHP.

Agora, o código está atualizado e funciona corretamente para versões mais recentes do PHP (8.0 e superiores), sem depender de métodos descontinuados.



e o index como q fica?



O arquivo `index.php` vai servir como o ponto de entrada para a aplicação, onde você vai registrar as rotas e, usando o **Container**, vai instanciar os controladores e injetar as dependências necessárias.

Com a refatoração no `Container` e a injeção automática de dependências, o arquivo `index.php` vai ficar mais simples. Ele será responsável por capturar a requisição, determinar qual controlador e método chamar, e passar os dados para a view.

Aqui está um exemplo de como o `index.php` pode ser ajustado:

1. Arquivo `public/index.php`

```
php

<?php

// Carregar o autoload do Composer
require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

// Iniciar o Container para resolução de dependências
use HefestoMVCLibrary\Container;
use HefestoMVC\Controllers\ArticleController;
use HefestoMVC\Controllers\HomeController;

// Verificar a URL da requisição e determinar qual ação do controlador deve ser chamada
$requestUri = $_SERVER['REQUEST_URI'];

// Definir o Container (serviços)
$container = Container::getContainer();

// Rota para a página inicial
if ($requestUri === '/') {
    $controller = $container->get(HomeController::class);
    $controller->index();
}

// Rota para listar artigos
elseif ($requestUri === '/articles') {
    $controller = $container->get(ArticleController::class);
    $controller->index();
}

// Rota para mostrar um artigo específico
elseif (preg_match('/^\/article\/show\/(\d+)\$', $requestUri, $matches)) {
    $controller = $container->get(ArticleController::class);
    $controller->show($matches[1]);
}

// Se nenhuma rota for encontrada, retornar 404
else {
    header("HTTP/1.1 404 Not Found");
    echo "404 - Página não encontrada.";
}
```

2. Explicação do Fluxo no `index.php`:

- **Autoload e Variáveis de Ambiente:** O arquivo começa com o `require_once` para o autoload do Composer e carrega as variáveis do `.env`. Isso garante que o projeto use todas as dependências e configurações corretamente.

- **Container de Dependências:** O `Container::getContainer()` é responsável por fornecer as instâncias das classes (controladores, modelos, etc.), utilizando a lógica de injeção de dependências que criamos no `Container`.
- **Verificação de URL:** O código verifica a URL da requisição usando `$_SERVER['REQUEST_URI']` e define qual controlador e ação será chamada com base nisso. Se a URL corresponder a um padrão, o controlador apropriado será chamado.
- **Rotas:**
 - Se a URL for `/`, o controlador `HomeController` será executado.
 - Se a URL for `/articles`, o controlador `ArticleController` será executado.
 - Se a URL corresponder ao padrão `/article/show/{id}`, o método `show` do `ArticleController` será executado, passando o `id` como parâmetro.
- **404:** Se nenhuma das rotas definidas for correspondente, uma resposta HTTP 404 é enviada e uma mensagem de "Página não encontrada" é exibida.

3. Considerações sobre o `index.php`:

Esse modelo é bem simples e funciona para aplicativos pequenos. No entanto, conforme a aplicação cresce, você pode querer usar um **router** dedicado (como o [AltoRouter](#) ou um sistema de rotas personalizado) para gerenciar as rotas de forma mais flexível, suportando métodos HTTP diferentes (GET, POST, PUT, DELETE, etc.), parâmetros dinâmicos e muito mais.

4. Exemplo de fluxo de dados para a view:

Agora, para cada método no controlador (como `index` ou `show`), você pode passar os dados para a view através da classe `View::render`.

Por exemplo, no `ArticleController`:

```
php

public function index()
{
    $articles = $this->articleModel->findAll(); // Pega todos os artigos
    View::render('article/index.twig', ['articles' => $articles]);
}

public function show($id)
{
    $article = $this->articleModel->find($id); // Pega o artigo com o id fornecido
    View::render('article/show.twig', ['article' => $article]);
}
```

A `View::render` vai buscar a view correspondente (ex.: `article/index.twig` ou `article/show.twig`) e renderizar com os dados fornecidos.

Com esse modelo, o fluxo de dados e as dependências são bem controlados, e o código está bem organizado para escalabilidade.

5. Conclusão

Com essa estrutura, você tem:

- Injeção de dependência usando o `Container``.
- Roteamento básico com controle de requisições e execução de ações de controladores.
- Separação clara entre a lógica do controlador, modelos e visualizações (views).

Essa abordagem funciona bem para aplicações simples, e você pode expandi-la conforme a necessidade, implementando recursos como **middleware**, **autenticação**, **validação de dados**, etc.



mas teu container não tem isso `getContainer`



Você está correto! No exemplo que forneci, mencionei o método `getContainer``, mas ele não existe na implementação atual do nosso `Container``. Em vez disso, estamos utilizando a abordagem de **instância estática** do `Container``, que resolve e armazena as dependências diretamente.

Vamos corrigir isso e ajustar o exemplo para refletir a implementação correta, mantendo o fluxo de dependência dentro do container.

Corrigindo o Container para Adicionar o Método `getContainer``

Para que o container possa ser utilizado de forma mais flexível e com uma abordagem mais próxima do que mencionei, podemos criar um método `getContainer()`` que vai centralizar o gerenciamento das instâncias. No entanto, isso não é estritamente necessário, visto que estamos usando a abordagem estática. O que podemos fazer é centralizar a lógica da resolução de dependências diretamente em uma classe `App`` ou algo semelhante.

1. Atualizando o Container

Aqui está a versão simplificada e corrigida do `Container``, sem a necessidade de um método `getContainer``, já que o acesso ao container já é feito de forma estática:

```
php

<?php

namespace HefestoMVCLibrary;

use ReflectionClass;
use ReflectionNamedType;
use Exception;

class Container
{
    private static $instances = [];

    // Método para obter instâncias resolvidas ou criadas
    public static function get(string $class)
    {
        // Verifica se a instância já foi criada
        if (!isset(self::$instances[$class])) {
            self::$instances[$class] = self::create($class);
        }

        return self::$instances[$class];
    }
}
```

```
// Método para criar instâncias, resolvendo as dependências
private static function create(string $class)
{
    try {
        $reflection = new ReflectionClass($class);

        if ($constructor = $reflection->getConstructor()) {
            $dependencies = self::getDependencies($constructor);
            return $reflection->newInstanceArgs($dependencies);
        }

        return new $class;
    } catch (Exception $e) {
        throw new Exception("Não foi possível criar a instância de $class: " . $e->getMessage());
    }
}

// Método para resolver as dependências de um construtor
private static function getDependencies(ReflectionMethod $constructor)
{
    $dependencies = [];

    foreach ($constructor->getParameters() as $param) {
        $type = $param->getType();

        // Verifica se o tipo do parâmetro é uma classe (ou interface)
        if ($type instanceof ReflectionNamedType && !$type->isBuiltin()) {
            $dependencies[] = self::get($type->getName()); // Resolve a dependência
        } else {
            // Se o parâmetro não for uma classe, podemos adicionar null ou outro valor padrão
            $dependencies[] = null;
        }
    }

    return $dependencies;
}
}
```

2. Usando o Container no `index.php`

Agora, para o `index.php`, como estamos utilizando o método estático `Container::get()`, não precisamos de um método `getContainer()`. Em vez disso, podemos simplesmente usar `Container::get()` para resolver as dependências diretamente.

Aqui está a forma corrigida de usar o `Container` no `index.php`:

Exemplo de `index.php` (agora corrigido)

```
php

<?php

// Carregar o autoload do Composer
require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

// Verificar a URL da requisição e determinar qual ação do controlador deve ser chamada
$requestUri = $_SERVER['REQUEST_URI'];

// Roteamento básico e resolução de dependências com o Container
if ($requestUri === '/') {
```

```

        $controller = HefestoMVCLibrary\Container::get(\HefestoMVC\Controllers\HomeController::class);
        $controller->index();
    }

    elseif ($requestUri === '/articles') {
        $controller = HefestoMVCLibrary\Container::get(\HefestoMVC\Controllers\ArticleController::class);
        $controller->index();
    }

    elseif (preg_match('/^\/article\/show\/(\d+)\$/', $requestUri, $matches)) {
        $controller = HefestoMVCLibrary\Container::get(\HefestoMVC\Controllers\ArticleController::class);
        $controller->show($matches[1]);
    }

    // Se nenhuma rota for encontrada, retornar 404
    else {
        header("HTTP/1.1 404 Not Found");
        echo "404 - Página não encontrada.";
    }
}

```

3. Explicação

- **Usando o `Container::get()`:** Agora, no `index.php`, usamos `HefestoMVCLibrary\Container::get()` para resolver automaticamente as dependências dos controladores. Por exemplo, quando a URL é `/articles`, o `Container::get(\HefestoMVC\Controllers\ArticleController::class)` vai criar uma instância do controlador `ArticleController`, injetando automaticamente o modelo `ArticleModel` ou outras dependências.
- **Rotas:** O roteamento ainda é feito de forma simples, mas como a resolução de dependências é feita pelo container, podemos usar os controladores sem nos preocupar com a criação manual de instâncias e dependências.
- **Redução de código:** Agora o código no `index.php` está mais conciso e limpo. A lógica de construção do controlador e suas dependências foi delegada ao `Container`, o que melhora a organização do código.

Conclusão

Agora, o container está configurado corretamente para resolver dependências de controladores e injetar os dados automaticamente. Isso facilita a manutenção e expansão do projeto, pois você não precisa mais criar instâncias manualmente em cada controlador. Esse padrão de injeção de dependência ajuda a organizar melhor as dependências da aplicação.



mas eu uso arquivo de rotas: <?php

```

use HefestoMVCLibrary\Route;
use HefestoMVC\controllers\HomeController;

```

```

Route::get('/', [HomeController::class,'index']);
Route::get('/articles', [HomeController::class,'articles']);
Route::get('/show/{id}', [HomeController::class, 'show']); e route <?php

```

```

namespace HefestoMVCLibrary;

class Route
{
    private static array $routes = [];
    private array $middleware = [];

    public function __construct(private string $method, private string $uri, private $action) {}

    public static function get(string $uri, $action)
    {
        return self::addRoute('GET', $uri, $action);
    }

    public static function post(string $uri, $action)
    {
        return self::addRoute('POST', $uri, $action);
    }

    private static function addRoute(string $method, string $uri, $action)
    {
        $route = new self($method, $uri, $action);
        self::$routes[] = $route;
        return $route;
    }

    public function middleware($middleware)
    {
        $this->middleware = is_array($middleware) ? $middleware : [$middleware];
    }

    public static function getRoutes()
    {
        return self::$routes;
    }

    public function matches(string $requestUri, string $requestMethod)
    {
        if ($this->method !== $requestMethod) {
            return false;
        }

        $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
        return preg_match("#^{$pattern}$#", $requestUri);
    }

    public function extractParameters(string $requestUri)
    {
        $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
        preg_match("#^{$pattern}$#", $requestUri, $matches);
    }

```

```

        array_shift($matches);
        return $matches;
    }

    public function getAction()
    {
        return $this->action;
    }

    public function getMiddleware()
    {
        return $this->middleware;
    }
} e router <?php

namespace HefestoMVCLibrary;

use Exception;
use HefestoMVCLibrary\Route;
use HefestoMVCLibrary\MiddlewareHandler;
use HefestoMVC\controllers\NotFoundController;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();
                if (is_array($action)) {
                    [$controller, $method] = $action;
                    $controllerInstance = new $controller();
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        $this->handleNotFound();
    }
    private function handleNotFound()
    {
        http_response_code(404);
    }
}

```

```

$controller = new NotFoundController();
if (method_exists($controller, 'index')) {
    return $controller->index();
}

throw new Exception("NotFoundController ou método 'index' não implementado.");
}
}

```



Com a estrutura que você tem com o arquivo de rotas, o ``Router`` e o ``Route``, vamos ajustar para garantir que você consiga rodar corretamente as rotas e realizar a resolução de dependências no controlador, assim como usar o **Container** para injeção automática.

Passo 1: Alterações no Container

Vamos manter o container com a injeção automática de dependências, mas será necessário garantir que ele seja utilizado no roteamento.

Passo 2: Ajuste no Router

O ``Router`` vai continuar sendo responsável por comparar a URL da requisição e encontrar a rota correspondente. Ao encontrar a rota, ele deve resolver as dependências do controlador usando o **Container**.

A alteração importante é que, ao invocar o controlador, em vez de criar a instância diretamente, usaremos o ``Container::get()`` para obter a instância e injetar as dependências automaticamente.

Ajuste do Router

```

php

namespace HefestoMVCLibrary;

use HefestoMVCLibrary\Container;
use HefestoMVC\controllers\NotFoundController;
use Exception;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                // Aqui você pode adicionar a lógica para middleware, se necessário
                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();
                if (is_array($action)) {
                    [$controller, $method] = $action;

                    // Usando o Container para resolver a instância do controlador
                    $controllerInstance = Container::get($controller);
                    return call_user_func_array([$controllerInstance, $method], $params);
                }
            }
        }
    }
}

```



```

        if (is_callable($action)) {
            return call_user_func_array($action, $params);
        }
    }
}

$this->handleNotFound();
}

private function handleNotFound()
{
    http_response_code(404);
    $controller = new NotFoundController();
    if (method_exists($controller, 'index')) {
        return $controller->index();
    }

    throw new Exception("NotFoundController ou método 'index' não implementado.");
}
}

```

Explicação das alterações:

1. Resolução de Dependências no Controlador:

- Ao invés de instanciar o controlador diretamente, usamos o `Container::get($controller)` para que o container resolva todas as dependências do controlador e o crie de forma automática.

2. Lógica de Roteamento:

- O `Router::dispatch` verifica as rotas definidas e, ao encontrar uma correspondente, chama o controlador e o método associado.
- O método `call_user_func_array` é utilizado para passar os parâmetros extraídos da URL para o método do controlador.

Passo 3: Exemplo de Controle de Rota no index.php

Aqui está um exemplo simples de como o arquivo `index.php` usaria o `Router` para disparar a requisição:

```

php

<?php

// Carregar o autoload do Composer
require_once __DIR__ . '/../vendor/autoload.php';

// Carregar variáveis de ambiente
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__ . '/../');
$dotenv->load();

// Criar uma instância do Router
use HefestoMVCLibrary\Router;

$route = new Router();

// Capturar a URL e o método da requisição
$requestUri = $_SERVER['REQUEST_URI'];
$requestMethod = $_SERVER['REQUEST_METHOD'];

// Chamar o método dispatch do Router
$route->dispatch($requestUri, $requestMethod);

```

Passo 4: Exemplo de Controlador com Dependências

Agora, no seu controlador, por exemplo, o `HomeController`, vamos garantir que as dependências sejam injetadas corretamente via **Container**.

Aqui está um exemplo simples:

```
php

namespace HefestoMVC\Controllers;

use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\View;

class HomeController
{
    private $articleModel;

    // O Container irá injetar a dependência automaticamente
    public function __construct(ArticleModel $articleModel)
    {
        $this->articleModel = $articleModel;
    }

    public function index()
    {
        // Usar o modelo para pegar os artigos
        $articles = $this->articleModel->findAll();

        // Passar os dados para a view
        View::render('home.twig', ['articles' => $articles]);
    }

    public function articles()
    {
        // Lógica para listar os artigos
        $articles = $this->articleModel->findAll();

        View::render('articles/index.twig', ['articles' => $articles]);
    }

    public function show($id)
    {
        // Lógica para exibir um artigo específico
        $article = $this->articleModel->find($id);

        View::render('articles/show.twig', ['article' => $article]);
    }
}
```

Explicação do Controlador:

- O controlador `HomeController` agora recebe o modelo `ArticleModel` como dependência através do construtor.
- O **Container** resolve a dependência automaticamente, então você não precisa instanciá-la manualmente no controlador.
- Os métodos `index`, `articles` e `show` agora usam o modelo para buscar os dados e passá-los para a view.

Considerações Finais

Agora a arquitetura está bem configurada com **injeção de dependências** via **Container** e **rota** através da classe ``Router``. Com isso, você pode facilmente adicionar novas rotas, controladores e modelos sem se preocupar com a criação manual de instâncias. Além disso, a estrutura está escalável, permitindo adicionar novos recursos como autenticação, middleware e tratamento de exceções conforme necessário.



crie a view `home.twig` para listar cards com os artigos com `id`, `title` e `content` sendo resumido em 100 caracteres e paginando



Aqui está a implementação de uma **view** ``home.twig`` para listar artigos em cards com **ID**, **Título** e **Conteúdo** resumido (limite de 100 caracteres), além de implementar a **paginação**:

Estrutura da View (``home.twig``)

```
twig

<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Artigos</title>
  <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
</head>
<body>
  <!-- Navbar (pode ser incluída em um arquivo separado, se necessário) -->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container">
      <a class="navbar-brand" href="/">HefestoMVC</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Alternar navegação">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav ms-auto">
          <li class="nav-item">
            <a class="nav-link active" href="/">Home</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="/articles">Artigos</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>

  <!-- Corpo da Página -->
  <div class="container mt-4">
    <h1 class="mb-4">Lista de Artigos</h1>

    <div class="row">
      {% for article in articles %}
        <div class="col-md-4 mb-4">
          <div class="card">
            <div class="card-header">
              <h5 class="card-title">{{ article.title }}</h5>
            </div>
            <div class="card-body">
```

```

</p>
        <p class="card-text">{{ article.content|slice(0, 100) ~ '...' }}
        <a href="/article/show/{{ article.id }}" class="btn btn-primar
y">Leia mais</a>
    </div>
    <div class="card-footer text-muted">
        ID: {{ article.id }}
    </div>
</div>
</div>
{% else %}
    <p class="col-12">Nenhum artigo encontrado.</p>
{% endfor %}
</div>

<!-- Paginação -->
<div class="d-flex justify-content-center mt-4">
    <nav aria-label="Paginação">
        <ul class="pagination">
            {% if current_page > 1 %}
                <li class="page-item">
                    <a class="page-link" href="?page={{ current_page - 1 }}" aria-la
bel="Previous">
                        <span aria-hidden="true">&laquo;</span>
                    </a>
                </li>
            {% else %}
                <li class="page-item disabled">
                    <span class="page-link">&laquo;</span>
                </li>
            {% endif %}

            {% for i in 1..total_pages %}
                <li class="page-item {% if i == current_page %}active{% endif %}">
                    <a class="page-link" href="?page={{ i }}">{{ i }}</a>
                </li>
            {% endfor %}

            {% if current_page < total_pages %}
                <li class="page-item">
                    <a class="page-link" href="?page={{ current_page + 1 }}" aria-la
bel="Next">
                        <span aria-hidden="true">&raquo;</span>
                    </a>
                </li>
            {% else %}
                <li class="page-item disabled">
                    <span class="page-link">&raquo;</span>
                </li>
            {% endif %}
        </ul>
    </nav>
</div>

<script src="/assets/js/bootstrap.bundle.min.js"></script>
</body>
</html>

```

Explicação da View:

1. Listagem de Artigos:

- Utilizamos um loop `for` do Twig para iterar sobre a variável `articles` que deve conter a lista de artigos.
- Para cada artigo, exibimos o título (`{{ article.title }}`), o conteúdo resumido (`{{ article.content|slice(0, 100) ~ '...' }}`) limitando a 100 caracteres, e o ID do artigo (`{{ article.id }}`).
- A rota para ler mais detalhes de um artigo é definida como `/article/show/{{ article.id }}`.

2. Paginação:

- A página atual e o total de páginas são passados para a view com as variáveis `current_page` e `total_pages`.
- A lógica de paginação usa a estrutura de paginação do Bootstrap, mostrando links para navegar entre as páginas.
- O Twig verifica se há páginas anteriores ou seguintes e ativa/desativa os links de navegação conforme necessário.

3. Estilos com Bootstrap:

- A página é estilizada com o framework **Bootstrap** para garantir uma aparência responsiva e organizada.
- Você pode ajustar o caminho do CSS do Bootstrap caso esteja usando um arquivo local ou CDN.

Passo 2: Atualizando o Controlador para Passar Dados

No controlador `HomeController`, quando você renderiza a view `home.twig`, você precisa garantir que está passando os dados de artigos, paginação e outros parâmetros.

Aqui está um exemplo de como fazer isso:

```
php

namespace HefestoMVC\Controllers;

use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\View;

class HomeController
{
    private $articleModel;

    public function __construct(ArticleModel $articleModel)
    {
        $this->articleModel = $articleModel;
    }

    public function index()
    {
        // Definir a página atual
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        $articlesPerPage = 6; // Ajuste o número de artigos por página
        $totalArticles = $this->articleModel->countAll(); // Conta total de artigos
        $totalPages = ceil($totalArticles / $articlesPerPage);

        // Pegar os artigos da página atual
        $offset = ($currentPage - 1) * $articlesPerPage;
        $articles = $this->articleModel->findAll($offset, $articlesPerPage);

        // Renderizar a view passando os dados
        View::render('home.twig', [
            'articles' => $articles,
            'current_page' => $currentPage,
            'total_pages' => $totalPages
        ]);
    }
}
```

Explicação do Controlador:

1. Paginação:

- Calculamos a página atual (`\$currentPage`), número de artigos por página (`\$articlesPerPage`), e o total de artigos.
- Usamos o método `countAll()` para contar o total de artigos e, a partir daí, calculamos o número total de páginas.
- Calculamos o **offset** para saber a partir de qual artigo pegar na consulta.

2. Exibindo os Artigos:

- Usamos o método `findAll(\$offset, \$limit)` do modelo `ArticleModel` para obter os artigos com base na página atual.

3. Passando Dados para a View:

- Passamos para a view as variáveis `articles`, `current_page` e `total_pages`, que serão usadas para exibir os artigos e controlar a paginação na view.

Passo 3: Atualização no Modelo

Para garantir que o modelo `ArticleModel` saiba como lidar com a paginação, você pode atualizar os métodos `findAll` e `countAll` da seguinte maneira:

```
php

namespace HefestoMVC\Models;
use HefestoMVCLibrary\Database;

class ArticleModel
{
    private $db;

    public function __construct(Database $db)
    {
        $this->db = $db;
    }

    public function findAll($offset = 0, $limit = 10)
    {
        $stmt = $this->db->connect()->prepare("SELECT * FROM articles LIMIT :limit OFFSET :offset");
        $stmt->bindParam(':limit', $limit, \PDO::PARAM_INT);
        $stmt->bindParam(':offset', $offset, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetchAll();
    }

    public function countAll()
    {
        $stmt = $this->db->connect()->query("SELECT COUNT(*) FROM articles");
        return $stmt->fetchColumn();
    }
}
```

Explicação:

1. Método `findAll`:

- Este método agora aceita parâmetros `\$offset` e `\$limit` para buscar artigos específicos com base na página solicitada.

2. Método `countAll`:

- Conta o número total de artigos na tabela `articles` para calcular a quantidade total de páginas.

Agora você tem uma view completa com os artigos listados em cards, com conteúdo resumido, e paginados corretamente.



```
adapte esse countall a isso <?php

namespace HefestoMVC\database;

class Model {

    protected $table;
    protected $conn;

    public function __construct($table) {
        $this->table = $table;
        $this->conn = Connection::connect(); // Conectar ao banco de dados
    }

    // Buscar um registro pela chave primária (ID)
    public function find($id) {
        $query = "SELECT * FROM {$this->table} WHERE id = :id LIMIT 1";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(); // Retorna o registro como objeto
    }

    // Buscar todos os registros da tabela
    public function findAll() {
        $query = "SELECT * FROM {$this->table}";
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros como array de objetos
    }

    // Buscar registros com base em uma condição
    public function findBy($field, $value) {
        $query = "SELECT * FROM {$this->table} WHERE {$field} = :value";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':value', $value, \PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros encontrados
    }
}
```



Para adaptar o método `countAll` à estrutura que você forneceu, mantendo o modelo de classe `Model` que utiliza o PDO para conexão com o banco de dados, você pode adicionar um novo método à classe.

Esse método irá contar o número de registros de uma tabela específica, e pode ser facilmente integrado à lógica de paginação.

Aqui está a adaptação:

Atualização do modelo `Model`

```
php

namespace HefestoMVC\database;

class Model {

    protected $table;
    protected $conn;

    public function __construct($table) {
        $this->table = $table;
        $this->conn = Connection::connect(); // Conectar ao banco de dados
    }

    // Buscar um registro pela chave primária (ID)
    public function find($id) {
        $query = "SELECT * FROM {$this->table} WHERE id = :id LIMIT 1";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(); // Retorna o registro como objeto
    }

    // Buscar todos os registros da tabela
    public function findAll($offset = 0, $limit = 10) {
        $query = "SELECT * FROM {$this->table} LIMIT :limit OFFSET :offset";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':limit', $limit, \PDO::PARAM_INT);
        $stmt->bindParam(':offset', $offset, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros como array de objetos
    }

    // Buscar registros com base em uma condição
    public function findBy($field, $value) {
        $query = "SELECT * FROM {$this->table} WHERE {$field} = :value";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':value', $value, \PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros encontrados
    }

    // Contar o número total de registros na tabela
    public function countAll() {
        $query = "SELECT COUNT(*) FROM {$this->table}";
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        return $stmt->fetchColumn(); // Retorna o número de registros encontrados
    }
}
```

Explicação do método `countAll`:

1. Método `countAll`:

- O método `countAll` executa uma consulta SQL com a função `COUNT(*)` para contar todos os registros da tabela atual (`\$this->table`).
- A função `fetchColumn` é usada para retornar o valor da primeira coluna da primeira linha do resultado da consulta, ou seja, o número total de registros na tabela.

Isso permite que você obtenha o número total de registros para poder calcular a quantidade de páginas na paginação, como fizemos anteriormente.

Como usar:

Agora, você pode usar o método `countAll` da seguinte maneira no controlador:

```
php

namespace HefestoMVC\Controllers;

use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\View;

class HomeController
{
    private $articleModel;

    public function __construct(ArticleModel $articleModel)
    {
        $this->articleModel = $articleModel;
    }

    public function index()
    {
        // Definir a página atual
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        $articlesPerPage = 6; // Ajuste o número de artigos por página
        $totalArticles = $this->articleModel->countAll(); // Conta total de artigos
        $totalPages = ceil($totalArticles / $articlesPerPage);

        // Pegar os artigos da página atual
        $offset = ($currentPage - 1) * $articlesPerPage;
        $articles = $this->articleModel->findAll($offset, $articlesPerPage);

        // Renderizar a view passando os dados
        View::render('home.twig', [
            'articles' => $articles,
            'current_page' => $currentPage,
            'total_pages' => $totalPages
        ]);
    }
}
```

Explicação do controlador `HomeController`:

1. Contagem de Artigos:

- O método `countAll` da classe `ArticleModel` é chamado para obter o número total de artigos na tabela `articles`.

2. Cálculo de Páginas:

- O número total de artigos (`$totalArticles`) é usado para calcular o número total de páginas com base no número de artigos por página (`$articlesPerPage`).

3. Busca de Artigos com Paginação:

- O método `findAll($offset, $limit)` é usado para buscar os artigos da página atual, respeitando o número de artigos por página (`$articlesPerPage`).

Com isso, você tem a contagem de registros, a listagem paginada de artigos e a renderização da view com a paginação.



Crie uma classe de paginação como helper a parte e mostre como usar no controller e na views. Use a ideia de offset do mysql



Vamos criar uma classe de paginação (Pagination) para gerenciar a lógica de paginação em seu sistema. Essa classe vai receber o número total de registros, o número de registros por página e a página atual, calculando o offset corretamente para a consulta SQL e também gerando os links de navegação para as páginas.

Passo 1: Criando a Classe de Paginação

A classe `Pagination` ficará em um arquivo separado na pasta `/library/` ou em um diretório apropriado. A classe calculará o `offset` para as consultas e fornecerá os links de navegação.

Arquivo: `Pagination.php`

```
php

namespace HefestoMVCLibrary;

class Pagination {

    private $totalRecords;
    private $recordsPerPage;
    private $currentPage;
    private $totalPages;

    public function __construct(int $totalRecords, int $recordsPerPage, int $currentPage = 1) {
        $this->totalRecords = $totalRecords;
        $this->recordsPerPage = $recordsPerPage;
        $this->currentPage = $currentPage;
        $this->totalPages = ceil($this->totalRecords / $this->recordsPerPage);
    }

    // Calcula o offset para a consulta SQL
    public function getOffset(): int {
        return ($this->currentPage - 1) * $this->recordsPerPage;
    }

    // Retorna o número total de páginas
    public function getTotalPages(): int {
        return $this->totalPages;
    }

    // Retorna o número da página atual
    public function getCurrentPage(): int {
        return $this->currentPage;
    }

    // Gera os links de navegação de página
    public function getPaginationLinks(string $baseUrl): string {
        $links = '';

        // Link para a primeira página
        if ($this->currentPage > 1) {
            $links .= '<a href="' . $baseUrl . '?page=1">Primeira</a>';
        }

        // Link para a página anterior
        if ($this->currentPage > 1) {
            $links .= ' <a href="' . $baseUrl . '?page=' . ($this->currentPage - 1) . '">Ant
```

```

erior</a>';
    }

    // Link para a próxima página
    if ($this->currentPage < $this->totalPages) {
        $links .= ' <a href="' . $baseUrl . '?page=' . ($this->currentPage + 1) . '">Pró
xima</a>';
    }

    // Link para a última página
    if ($this->currentPage < $this->totalPages) {
        $links .= ' <a href="' . $baseUrl . '?page=' . $this->totalPages . '">Última</a
>';
    }

    return $links;
}
}

```

Explicação da Classe `Pagination`

1. Construtor:

- O construtor recebe o número total de registros (`\$totalRecords`), o número de registros por página (`\$recordsPerPage`) e a página atual (`\$currentPage`). Ele calcula o número total de páginas usando `ceil(\$totalRecords / \$recordsPerPage)`.

2. Métodos:

- `getOffset()`: Calcula o valor do `offset` com base na página atual.
- `getTotalPages()`: Retorna o número total de páginas.
- `getCurrentPage()`: Retorna o número da página atual.
- `getPaginationLinks()`: Gera os links de navegação com base na página atual e no número total de páginas.

Passo 2: Usando a Classe de Paginação no Controlador

Agora que temos a classe de paginação, vamos integrá-la no controlador para fornecer a lógica de paginação para os artigos. Usaremos o método `getOffset()` da classe `Pagination` para obter o valor correto do `offset` para as consultas SQL.

Atualizando o `HomeController`

php

```

namespace HefestoMVC\Controllers;

use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\Pagination;
use HefestoMVCLibrary\View;

class HomeController
{
    private $articleModel;

    public function __construct(ArticleModel $articleModel)
    {
        $this->articleModel = $articleModel;
    }

    public function index()
    {

```

```
// Definir a página atual
$currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
$articlesPerPage = 6; // Ajuste o número de artigos por página

// Contar o total de artigos
$totalArticles = $this->articleModel->countAll(); // Conta total de artigos
$pagination = new Pagination($totalArticles, $articlesPerPage, $currentPage);

// Pegar os artigos da página atual
$offset = $pagination->getOffset();
$articles = $this->articleModel->findAll($offset, $articlesPerPage);

// Gerar os links de paginação
$paginationLinks = $pagination->getPaginationLinks('/articles'); // Base URL para os links

// Renderizar a view passando os dados
View::render('home.twig', [
    'articles' => $articles,
    'paginationLinks' => $paginationLinks,
    'currentPage' => $pagination->getCurrentPage(),
    'totalPages' => $pagination->getTotalPages()
]);
}
```

Passo 3: Atualizando a View `home.twig`

Agora, vamos modificar a view `home.twig` para exibir os artigos e a navegação de paginação gerada pela classe `Pagination`.

Arquivo: `home.twig`

```
twig

{% extends 'template/master.twig' %}

{% block content %}
    <div class="container">
        <div class="row">
            {% for article in articles %}
                <div class="col-md-4 mb-4">
                    <div class="card">
                        <div class="card-body">
                            <h5 class="card-title">{{ article.title }}</h5>
                            <p class="card-text">{{ article.content[:100] }}...</p>
                            <a href="/show/{{ article.id }}" class="btn btn-primary">Leia mais</a>
                        </div>
                    </div>
                </div>
            {% endfor %}
        </div>

        <div class="pagination">
            <nav>
                <ul class="pagination">
                    <li class="page-item {% if currentPage == 1 %}disabled{% endif %}">
                        <a class="page-link" href="/articles?page=1">Primeira</a>
                    </li>
                    <li class="page-item {% if currentPage == 1 %}disabled{% endif %}">
                        <a class="page-link" href="/articles?page={{ currentPage - 1 }}">Anterior</a>
                    </li>

                    <li class="page-item {% if currentPage == totalPages %}disabled{% endif %}">
                        <a class="page-link" href="/articles?page={{ currentPage + 1 }}">Próxima</a>
                    </li>
                </ul>
            </nav>
        </div>
    </div>
{% endblock %}
```

```

        <a class="page-link" href="/articles?page={{ currentPage + 1 }}">Pró
xima</a>
    </li>

    <li class="page-item {% if currentPage == totalPages %}disabled{% endif
%}">
        <a class="page-link" href="/articles?page={{ totalPages }}">Última</
a>
    </li>
</ul>
</nav>
</div>
</div>
{% endblock %}

```

Explicação da View `home.twig`

1. Exibição dos Artigos:

- o O loop `{% for article in articles %}` percorre os artigos e exibe o título e o resumo do conteúdo (primeiros 100 caracteres).

2. Paginação:

- o A navegação de páginas é gerada usando a classe `Pagination`. Usamos as variáveis `currentPage` e `totalPages` para gerar os links para as páginas anterior e próxima.
- o O botão "Leia mais" redireciona para a página de visualização do artigo.

Resultado

Agora você tem uma implementação completa de paginação usando a classe `Pagination`, que calcula o `offset` para consultas SQL e gera os links de navegação. Essa solução pode ser reutilizada em outros controladores e páginas de seu sistema.



Mas Pagination não poderia ser resolvido com o Container no controller?



Sim, você pode usar o `Container` para injetar a classe de `Pagination` diretamente no controlador, ao invés de instanciá-la manualmente dentro do controlador. Isso segue o princípio de **Injeção de Dependência** (DI), que facilita a manutenção, testes e reutilização das classes, além de deixar o código mais limpo e desacoplado.

Vamos refatorar o código para utilizar o `Container` para gerenciar a dependência da classe `Pagination`. Isso pode ser feito adicionando a `Pagination` ao `Container` e injetando-a diretamente no controlador, de maneira que o controlador não precise mais se preocupar com a criação da instância de `Pagination`.

Passo 1: Atualizando o `Container` para Registrar a Classe de Paginação

No seu `Container`, adicione a classe `Pagination` para ser injetada. O `Container` pode ser configurado para criar instâncias da classe `Pagination` quando necessário, de modo que ela seja resolvida automaticamente.

Atualizando o `Container.php`

```
php

namespace HefestoMVCLibrary;

use HefestoMVC\Models\ArticleModel;
use HefestoMVC\Controllers\HomeController;
use HefestoMVCLibrary\Pagination;

class Container
{
    private static $instances = [];

    public static function set(string $key, $value)
    {
        self::$instances[$key] = $value;
    }

    public static function get(string $key)
    {
        if (isset(self::$instances[$key])) {
            return self::$instances[$key];
        }

        return null;
    }

    public static function register()
    {
        // Registra a classe de Pagination no Container
        self::set('pagination', function () {
            return new Pagination();
        });

        // Outros registros do Container
        self::set('articleModel', function () {
            return new ArticleModel();
        });
    }
}
```

Aqui, registramos a `Pagination` no `Container`, de modo que sempre que precisarmos dela, podemos chamá-la com `\$container->get('pagination')`.

Passo 2: Atualizando o Controlador para Usar a `Pagination` do Container

No seu controlador, ao invés de instanciar a classe `Pagination` diretamente, vamos injetá-la usando o `Container`. Isso vai tornar o controlador mais desacoplado, uma vez que ele não precisa saber como a `Pagination` é criada.

Atualizando o `HomeController.php`

```
php

namespace HefestoMVC\Controllers;

use HefestoMVCLibrary\View;
use HefestoMVCLibrary\Container;
use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\Pagination;

class HomeController
```

```
{
    private $articleModel;
    private $pagination;

    public function __construct(ArticleModel $articleModel, Pagination $pagination)
    {
        $this->articleModel = $articleModel;
        $this->pagination = $pagination;
    }

    public function index()
    {
        // Definir a página atual
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        $articlesPerPage = 6; // Ajuste o número de artigos por página

        // Contar o total de artigos
        $totalArticles = $this->articleModel->countAll(); // Conta total de artigos
        $this->pagination = new Pagination($totalArticles, $articlesPerPage, $currentPage);

        // Pegar os artigos da página atual
        $offset = $this->pagination->getOffset();
        $articles = $this->articleModel->findAll($offset, $articlesPerPage);

        // Gerar os links de paginação
        $paginationLinks = $this->pagination->getPaginationLinks('/articles'); // Base URL p
        ara os links

        // Renderizar a view passando os dados
        View::render('home.twig', [
            'articles' => $articles,
            'paginationLinks' => $paginationLinks,
            'currentPage' => $this->pagination->getCurrentPage(),
            'totalPages' => $this->pagination->getTotalPages()
        ]);
    }
}
```

Passo 3: Ajustando a Rota para Usar o Container

Agora, no momento de fazer a chamada para o controlador, precisamos garantir que as dependências (como `Pagination`) sejam passadas corretamente.

Atualizando o `index.php` ou onde o Container é Inicializado

```
php

use HefestoMVCLibrary\Container;
use HefestoMVC\Controllers\HomeController;
use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\Pagination;

// Registra todas as dependências no Container
Container::register();

// Obtém as dependências do Container
$articleModel = Container::get('articleModel');
$pagination = Container::get('pagination');

// Cria o controlador com as dependências injetadas
$homeController = new HomeController($articleModel, $pagination);

// Chama o método do controlador
$homeController->index();
```

Explicação

1. Container:

- O `Container` agora é responsável por resolver todas as dependências e injetá-las nos controladores. No caso do `HomeController`, o `ArticleModel` e a `Pagination` são injetados automaticamente.

2. Injeção de Dependências no Controlador:

- A `Pagination` é passada para o controlador através do construtor, usando o `Container`. O controlador agora não precisa mais saber como instanciar a `Pagination`, o que o torna mais flexível e desacoplado.

3. Controlador:

- O controlador recebe a `Pagination` e o `ArticleModel` como dependências e utiliza esses objetos para manipular os dados e passar para a view.

Vantagens de Usar o Container

- **Desacoplamento:** O controlador não precisa se preocupar em como criar instâncias de objetos como `Pagination` ou `ArticleModel`. Ele simplesmente os recebe como dependências, o que facilita a manutenção e os testes.
- **Flexibilidade:** Se no futuro você precisar alterar a criação de objetos (por exemplo, adicionar uma configuração especial ao criar a `Pagination`), pode fazer isso centralizadamente no `Container` sem precisar alterar o controlador.
- **Facilidade de Testes:** Injeção de dependência facilita os testes unitários, pois você pode facilmente mockar as dependências ao testar o controlador.

Passo 4: Continuar Usando o Container nas Views

Finalmente, o `Container` pode ser útil para injetar outras dependências nas views ou outros componentes que você desejar, mantendo a lógica centralizada e modular. Isso garante que todas as dependências sejam resolvidas de forma coerente ao longo da aplicação.

Dessa forma, com o uso do `Container`, você tem uma arquitetura mais limpa e escalável, e o código fica mais organizado e fácil de manter.



```
mas já temos o Container, lembra? <?php
namespace HefestoMVCLibrary;
```

```
use Exception;
use ReflectionClass;
use ReflectionNamedType;
```

```
class Container
{
    private static $instances = [];

    public static function get(string $class)
    {
        // Verifica se a instância já foi criada
```



```

    if (!isset(self::$instances[$class])) {
        self::$instances[$class] = self::create($class);
    }

    return self::$instances[$class];
}

private static function create(string $class)
{
    try {
        // Usando Reflection para instanciar a classe com as dependências
        $reflection = new ReflectionClass($class);

        if ($constructor = $reflection->getConstructor()) {
            $dependencies = self::getDependencies($constructor);
            return $reflection->newInstanceArgs($dependencies);
        }

        return new $class;
    } catch (Exception $e) {
        throw new Exception("Não foi possível criar a instância de $class: " . $e->getMessage());
    }
}

private static function getDependencies(\ReflectionMethod $constructor)
{
    $dependencies = [];

    foreach ($constructor->getParameters() as $param) {
        $type = $param->getType();

        // Verifica se o tipo do parâmetro é uma classe (ou interface)
        if ($type instanceof ReflectionNamedType && !$type->isBuiltin()) {
            $dependencies[] = self::get($type->getName()); // Resolve a dependência
        } else {
            // Se o parâmetro não for uma classe, podemos adicionar null ou outro valor padrão
            $dependencies[] = null;
        }
    }

    return $dependencies;
}
}

```



Sim, você já tem um `Container` que usa **Reflection** para injetar dependências automaticamente. O que você pode fazer é simplificar ainda mais a injeção de dependências e garantir que o `Container` seja capaz de gerenciar corretamente a criação de instâncias dos controladores e suas dependências.

Vamos revisar como fazer isso funcionar com o **Container** atual, onde o controlador pode ser automaticamente resolvido sem a necessidade de inicialização manual no `index.php`.

Como Funciona o Container Atual

O seu `Container` já usa Reflection para instanciar as dependências automaticamente com base no construtor da classe. Isso significa que, para qualquer classe que precise de dependências (como `ArticleModel` e `Pagination`), o `Container` vai automaticamente resolver as dependências quando você solicitar a instância da classe.

Passo 1: Utilizando o Container para Resolver as Dependências

No exemplo anterior, vamos usar o `Container` para resolver as dependências automaticamente quando instanciamos o `HomeController`. O `Container` vai cuidar de injetar o `ArticleModel` e o `Pagination` no controlador quando for necessário.

Exemplo de Como Usar o Container para Resolver as Dependências

Em vez de instanciar manualmente o `HomeController` no `index.php`, você vai usar o `Container` para pegar a instância do controlador e resolver as dependências automaticamente:

```
php

use HefestoMVCLibrary\Container;
use HefestoMVC\Controllers\HomeController;
use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\Pagination;

// Registra todas as dependências no Container (se necessário)
Container::set('articleModel', ArticleModel::class);
Container::set('pagination', Pagination::class);

// Pega a instância do controlador, já resolvendo as dependências automaticamente
$homeController = Container::get(HomeController::class);

// Chama o método do controlador
$homeController->index();
```

Passo 2: Atualizando o Controlador para Usar as Dependências

Agora, no seu `HomeController`, não será necessário instanciar o `ArticleModel` ou `Pagination` manualmente. Eles serão automaticamente injetados pelo `Container`.

Atualizando o `HomeController.php`

```
php

namespace HefestoMVC\Controllers;

use HefestoMVCLibrary\View;
use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\Pagination;

class HomeController
{
```

```
private $articleModel;
private $pagination;

// O Container automaticamente injeta as dependências no controlador
public function __construct(ArticleModel $articleModel, Pagination $pagination)
{
    $this->articleModel = $articleModel;
    $this->pagination = $pagination;
}

public function index()
{
    // Definir a página atual
    $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
    $articlesPerPage = 6; // Ajuste o número de artigos por página

    // Contar o total de artigos
    $totalArticles = $this->articleModel->countAll(); // Conta total de artigos
    $this->pagination = new Pagination($totalArticles, $articlesPerPage, $currentPage);

    // Pegar os artigos da página atual
    $offset = $this->pagination->getOffset();
    $articles = $this->articleModel->findAll($offset, $articlesPerPage);

    // Gerar os links de paginação
    $paginationLinks = $this->pagination->getPaginationLinks('/articles'); // Base URL para os links

    // Renderizar a view passando os dados
    View::render('home.twig', [
        'articles' => $articles,
        'paginationLinks' => $paginationLinks,
        'currentPage' => $this->pagination->getCurrentPage(),
        'totalPages' => $this->pagination->getTotalPages()
    ]);
}
```

Passo 3: Testando a Funcionalidade

Agora, o `HomeController` será instanciado automaticamente pelo `Container`, e as dependências como `ArticleModel` e `Pagination` serão resolvidas e passadas para o controlador sem você precisar instanciá-las manualmente.

1. **Injeção Automática:** O `Container` vai verificar o construtor do `HomeController`, ver quais dependências ele precisa (como `ArticleModel` e `Pagination`), e automaticamente injetar essas dependências.
2. **Flexibilidade:** Caso você precise adicionar outras dependências, basta adicioná-las ao construtor do controlador, e o `Container` vai gerenciar isso.

Passo 4: Resolver no `Router`

No seu `Router`, ao invés de instanciar os controladores manualmente, você pode simplesmente passar o nome da classe para o `Container`, e ele vai cuidar de resolver as dependências:

Exemplo do `Router`

```
php

namespace HefestoMVCLibrary;
```

```

use HefestoMVC\Controllers\HomeController;
use HefestoMVC\Controllers\NotFoundController;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();
                if (is_array($action)) {
                    [$controller, $method] = $action;
                    $controllerInstance = Container::get($controller); // Resolvendo a depen
dência automaticamente
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        $this->handleNotFound();
    }

    private function handleNotFound()
    {
        http_response_code(404);
        $controller = new NotFoundController();
        if (method_exists($controller, 'index')) {
            return $controller->index();
        }

        throw new Exception("NotFoundController ou método 'index' não implementado.");
    }
}

```

Resumo

- Usamos **Reflection** no `Container` para resolver e injetar automaticamente as dependências, o que simplifica o código e facilita a manutenção.
- O `Container` cuida de resolver as dependências e instanciar os objetos necessários para o controlador.
- No controlador, as dependências são automaticamente injetadas, e você não precisa mais se preocupar em instanciá-las manualmente.
- O **Router** também foi ajustado para usar o `Container` e resolver as dependências do controlador automaticamente.

Essa abordagem permite que você tenha um código mais limpo e desacoplado, facilitando a manutenção e testes.



Essa paginação tá dando erro pois carrega antes dados nulos do que o controller



O problema ocorre porque a classe de paginação é instanciada antes dos dados necessários estarem disponíveis no controlador. Para corrigir isso, podemos ajustar a lógica no controlador para que a instância de `Pagination` seja criada **após** os dados do banco de dados serem recuperados, garantindo que a contagem total de artigos esteja disponível.

Aqui está a correção:

Passo 1: Ajustar o `HomeController`

No controlador, inicialize a classe de paginação **somente após obter o total de registros do banco de dados**:

```
php

namespace HefestoMVC\Controllers;

use HefestoMVCLibrary\View;
use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\Pagination;

class HomeController
{
    private $articleModel;
    private $pagination;

    public function __construct(ArticleModel $articleModel)
    {
        $this->articleModel = $articleModel;
    }

    public function index()
    {
        // Obter a página atual da query string ou padrão para 1
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        $articlesPerPage = 6; // Número de artigos por página

        // Obter o total de artigos do banco
        $totalArticles = $this->articleModel->countAll();

        // Inicializar a paginação com base no total de artigos
        $this->pagination = new Pagination($totalArticles, $articlesPerPage, $currentPage);

        // Obter o offset e carregar os artigos da página atual
        $offset = $this->pagination->getOffset();
        $articles = $this->articleModel->findAll($offset, $articlesPerPage);

        // Gerar links de paginação
        $paginationLinks = $this->pagination->getPaginationLinks('/articles');

        // Renderizar a view com os dados
        View::render('home.twig', [
            'articles' => $articles,
            'paginationLinks' => $paginationLinks,
            'currentPage' => $this->pagination->getCurrentPage(),
            'totalPages' => $this->pagination->getTotalPages()
        ]);
    }
}
```

Passo 2: Ajustar o `Pagination`

Caso o problema persista, garanta que o construtor da classe `Pagination` verifica corretamente os valores recebidos:

```
php

namespace HefestoMVCLibrary;

class Pagination
{
    private int $totalItems;
    private int $itemsPerPage;
    private int $currentPage;

    public function __construct(int $totalItems, int $itemsPerPage, int $currentPage)
    {
        // Garantir valores válidos para paginação
        $this->totalItems = max(0, $totalItems);
        $this->itemsPerPage = max(1, $itemsPerPage);
        $this->currentPage = max(1, $currentPage);

        // Corrigir página atual se estiver fora do limite
        $totalPages = $this->getTotalPages();
        if ($this->currentPage > $totalPages) {
            $this->currentPage = $totalPages;
        }
    }

    public function getTotalPages(): int
    {
        return (int)ceil($this->totalItems / $this->itemsPerPage);
    }

    public function getOffset(): int
    {
        return ($this->currentPage - 1) * $this->itemsPerPage;
    }

    public function getPaginationLinks(string $baseUrl): array
    {
        $totalPages = $this->getTotalPages();
        $links = [];

        for ($i = 1; $i <= $totalPages; $i++) {
            $links[] = [
                'url' => "{$baseUrl}?page={$i}",
                'label' => $i,
                'active' => $i === $this->currentPage
            ];
        }

        return $links;
    }

    public function getCurrentPage(): int
    {
        return $this->currentPage;
    }
}
```

Passo 3: Ajustar o Modelo para Suporte à Paginação

No modelo, ajuste o método `findAll` para aceitar parâmetros de `offset` e `limit`:

php

```
namespace HefestoMVC\Models;

use HefestoMVC\Database\Model;

class ArticleModel extends Model
{
    public function __construct()
    {
        parent::__construct('articles');
    }

    public function findAll(int $offset = 0, int $limit = 10)
    {
        $query = "SELECT * FROM {$this->table} LIMIT :offset, :limit";
        $stmt = $this->conn->prepare($query);
        $stmt->bindValue(':offset', $offset, \PDO::PARAM_INT);
        $stmt->bindValue(':limit', $limit, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetchAll();
    }

    public function countAll(): int
    {
        $query = "SELECT COUNT(*) as total FROM {$this->table}";
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        return (int)$stmt->fetch()->total;
    }
}
```

Passo 4: Garantir a Ordem de Execução no `index.php`

No `index.php`, garanta que o roteador está executando o controlador corretamente **após** os dados do banco serem carregados.

Com essas correções, a classe de paginação será inicializada somente após os dados necessários estarem disponíveis no controlador. Isso evita problemas de inicialização com valores `null`.



Mas aí ferra a ideia do container



Você está certo: uma solução ideal deve aproveitar o **Container** para resolver dependências, incluindo a inicialização da classe de paginação. A ideia é permitir que o controlador não se preocupe em instanciar diretamente classes como `Pagination` ou mesmo `ArticleModel`, delegando essa responsabilidade ao Container.

Aqui está a solução aprimorada para continuar utilizando o **Container**:

Passo 1: Atualizar a Classe `Pagination`

Modifique a classe `Pagination` para permitir que seja resolvida pelo Container. Adicione um construtor que aceite as dependências necessárias:

```
php

namespace HefestoMVCLibrary;

class Pagination
{
    private int $totalItems;
    private int $itemsPerPage;
    private int $currentPage;

    public function __construct(int $totalItems, int $itemsPerPage, int $currentPage)
    {
        $this->totalItems = max(0, $totalItems);
        $this->itemsPerPage = max(1, $itemsPerPage);
        $this->currentPage = max(1, $currentPage);

        // Corrigir página atual se estiver fora do limite
        $totalPages = $this->getTotalPages();
        if ($this->currentPage > $totalPages) {
            $this->currentPage = $totalPages;
        }
    }

    public function getTotalPages(): int
    {
        return (int)ceil($this->totalItems / $this->itemsPerPage);
    }

    public function getOffset(): int
    {
        return ($this->currentPage - 1) * $this->itemsPerPage;
    }

    public function getPaginationLinks(string $baseUrl): array
    {
        $totalPages = $this->getTotalPages();
        $links = [];

        for ($i = 1; $i <= $totalPages; $i++) {
            $links[] = [
                'url' => "{$baseUrl}?page={$i}",
                'label' => $i,
                'active' => $i === $this->currentPage
            ];
        }

        return $links;
    }

    public function getCurrentPage(): int
    {
        return $this->currentPage;
    }
}
```

Passo 2: Ajustar o Controlador para o Container

Modifique o controlador para usar o **Container** para resolver as dependências, incluindo o `Pagination`:

php

```
namespace HefestoMVC\Controllers;

use HefestoMVCLibrary\View;
use HefestoMVC\Models\ArticleModel;
use HefestoMVCLibrary\Pagination;

class HomeController
{
    private ArticleModel $articleModel;
    private Pagination $pagination;

    public function __construct(ArticleModel $articleModel, Pagination $pagination)
    {
        $this->articleModel = $articleModel;
        $this->pagination = $pagination;
    }

    public function index()
    {
        // Obter a página atual da query string
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        $articlesPerPage = 6;

        // Obter total de artigos
        $totalArticles = $this->articleModel->countAll();

        // Atualizar a instância de paginação
        $pagination = new Pagination($totalArticles, $articlesPerPage, $currentPage);

        // Obter o offset e carregar os artigos da página atual
        $offset = $pagination->getOffset();
        $articles = $this->articleModel->findAll($offset, $articlesPerPage);

        // Gerar links de paginação
        $paginationLinks = $pagination->getPaginationLinks('/articles');

        // Renderizar a view
        View::render('home.twig', [
            'articles' => $articles,
            'paginationLinks' => $paginationLinks,
            'currentPage' => $pagination->getCurrentPage(),
            'totalPages' => $pagination->getTotalPages()
        ]);
    }
}
```

Passo 3: Resolver `Pagination` no Container

Atualize o Container para que ele consiga resolver corretamente a classe `Pagination`. Como a classe `Pagination` depende de valores dinâmicos (como `\$totalItems`, `\$itemsPerPage` e `\$currentPage`), precisamos usar um **factory** no Container:

php

```
namespace HefestoMVCLibrary;

class Container
{
    private static $instances = [];

    public static function get(string $class, array $parameters = [])
    {
        // Se a instância já existir, retorna
```

```

        if (!isset(self::$instances[$class])) {
            self::$instances[$class] = self::create($class, $parameters);
        }

        return self::$instances[$class];
    }

    private static function create(string $class, array $parameters = [])
    {
        try {
            $reflection = new \ReflectionClass($class);

            // Obter o construtor e seus parâmetros
            $constructor = $reflection->getConstructor();
            if ($constructor) {
                $dependencies = self::resolveDependencies($constructor, $parameters);
                return $reflection->newInstanceArgs($dependencies);
            }

            return new $class();
        } catch (\Exception $e) {
            throw new \Exception("Erro ao resolver classe {$class}: " . $e->getMessage());
        }
    }

    private static function resolveDependencies(\ReflectionMethod $constructor, array $parameters)
    {
        $dependencies = [];

        foreach ($constructor->getParameters() as $param) {
            $type = $param->getType();

            if ($type instanceof \ReflectionNamedType && !$type->isBuiltin()) {
                $dependencies[] = self::get($type->getName());
            } elseif (isset($parameters[$param->getName()])) {
                $dependencies[] = $parameters[$param->getName()];
            } else {
                $dependencies[] = null; // Ou algum valor padrão
            }
        }

        return $dependencies;
    }
}

```

Passo 4: Atualizar o `index.php` para Resolver `Pagination`

No arquivo principal, resolva a rota e use o Container para injetar as dependências:

php

```

use HefestoMVCLibrary\Container;
use HefestoMVCLibrary\Routing;
use HefestoMVCLibrary\Route;

$requestUri = $_SERVER['REQUEST_URI'];
$requestMethod = $_SERVER['REQUEST_METHOD'];

// Configurar rotas
require_once __DIR__ . '/../src/routes/web.php';

// Resolver o controlador e executar a ação correspondente
$route = new Route();
$response = $route->dispatch($requestUri, $requestMethod);

if ($response instanceof Closure) {

```

```
    echo $response();
}
```

Agora, o **Pagination** é resolvido automaticamente pelo Container sem a necessidade de instanciá-lo manualmente no controlador.



Expanda com mais opções como limit, offset, order by, like, joint e insert, update e delete sendo abstrato, para qualquer dados e tamanho de tabelas <?php

```
namespace HefestoMVC\Database\Models;

use HefestoMVC\Config\Database;

class Model {

    protected $table;
    protected $conn;

    public function __construct($table) {
        $this->table = $table;
        $this->conn = Database::connect(); // Conectar ao banco de dados
    }

    // Buscar um registro pela chave primária (ID)
    public function find($id) {
        $query = "SELECT * FROM {$this->table} WHERE id = :id LIMIT 1";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id, \PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(); // Retorna o registro como objeto
    }

    // Buscar todos os registros da tabela
    public function findAll() {
        $query = "SELECT * FROM {$this->table}";
        $stmt = $this->conn->prepare($query);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros como array de objetos
    }

    // Buscar registros com base em uma condição
    public function findBy($field, $value) {
        $query = "SELECT * FROM {$this->table} WHERE {$field} = :value";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':value', $value, \PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetchAll(); // Retorna todos os registros encontrados
    }
}
```

```
}
}
```



Aqui está uma versão expandida e mais completa da classe `Model`, com suporte a operações mais flexíveis, como `limit`, `offset`, `order by`, `like`, `join`, e métodos para `insert`, `update` e `delete`. A classe mantém uma abordagem abstrata para trabalhar com qualquer tabela.

php

```
namespace HefestoMVC\Database\Models;

use HefestoMVC\Config\Database;
use PDO;

class Model
{
    protected $table;
    protected $conn;

    public function __construct($table)
    {
        $this->table = $table;
        $this->conn = Database::connect(); // Conectar ao banco de dados
    }

    // Buscar um registro pela chave primária (ID)
    public function find($id)
    {
        $query = "SELECT * FROM {$this->table} WHERE id = :id LIMIT 1";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id, PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->fetch(PDO::FETCH_OBJ);
    }

    // Buscar todos os registros da tabela
    public function findAll($limit = null, $offset = null, $orderBy = null)
    {
        $query = "SELECT * FROM {$this->table}";
        if ($orderBy) {
            $query .= " ORDER BY {$orderBy}";
        }
        if ($limit !== null) {
            $query .= " LIMIT :limit";
            if ($offset !== null) {
                $query .= " OFFSET :offset";
            }
        }

        $stmt = $this->conn->prepare($query);

        if ($limit !== null) {
            $stmt->bindParam(':limit', $limit, PDO::PARAM_INT);
            if ($offset !== null) {
                $stmt->bindParam(':offset', $offset, PDO::PARAM_INT);
            }
        }

        $stmt->execute();
        return $stmt->fetchAll(PDO::FETCH_OBJ);
    }

    // Buscar registros com base em uma condição
    public function findBy($conditions = [], $limit = null, $offset = null, $orderBy = null)
    {
        $query = "SELECT * FROM {$this->table}";
    }
}
```

```

        if (!empty($conditions)) {
            $query .= " WHERE " . implode(' AND ', array_map(fn($field) => "{$field} = :{$fi
eld}", array_keys($conditions)));
        }
        if ($orderBy) {
            $query .= " ORDER BY {$orderBy}";
        }
        if ($limit !== null) {
            $query .= " LIMIT :limit";
            if ($offset !== null) {
                $query .= " OFFSET :offset";
            }
        }

        $stmt = $this->conn->prepare($query);

        foreach ($conditions as $field => $value) {
            $stmt->bindValue(":{field}", $value, is_int($value) ? PDO::PARAM_INT : PDO::PAR
AM_STR);
        }

        if ($limit !== null) {
            $stmt->bindParam(':limit', $limit, PDO::PARAM_INT);
            if ($offset !== null) {
                $stmt->bindParam(':offset', $offset, PDO::PARAM_INT);
            }
        }

        $stmt->execute();
        return $stmt->fetchAll(PDO::FETCH_OBJ);
    }

    // Inserir um registro
    public function insert(array $data)
    {
        $fields = implode(', ', array_keys($data));
        $placeholders = implode(', ', array_map(fn($field) => ":{field}", array_keys($dat
a)));

        $query = "INSERT INTO {$this->table} ({$fields}) VALUES ({$placeholders})";
        $stmt = $this->conn->prepare($query);

        foreach ($data as $field => $value) {
            $stmt->bindValue(":{field}", $value, is_int($value) ? PDO::PARAM_INT : PDO::PAR
AM_STR);
        }

        return $stmt->execute();
    }

    // Atualizar um registro
    public function update(array $data, $conditions = [])
    {
        $setClause = implode(', ', array_map(fn($field) => "{$field} = :{$field}", array_key
s($data)));
        $whereClause = !empty($conditions)
            ? " WHERE " . implode(' AND ', array_map(fn($field) => "{$field} = :where_{$fiel
d}", array_keys($conditions)))
            : '';

        $query = "UPDATE {$this->table} SET {$setClause}{$whereClause}";
        $stmt = $this->conn->prepare($query);

        foreach ($data as $field => $value) {
            $stmt->bindValue(":{field}", $value, is_int($value) ? PDO::PARAM_INT : PDO::PAR
AM_STR);
        }

        foreach ($conditions as $field => $value) {
            $stmt->bindValue(":where_{$field}", $value, is_int($value) ? PDO::PARAM_INT : PD
O::PARAM_STR);
        }

        return $stmt->execute();
    }
}

```

```
// Excluir registros
public function delete($conditions = [])
{
    $whereClause = !empty($conditions)
        ? " WHERE " . implode(' AND ', array_map(fn($field) => "{$field} = :{$field}", array_keys($conditions)))
        : '';

    $query = "DELETE FROM {$this->table}{$whereClause}";
    $stmt = $this->conn->prepare($query);

    foreach ($conditions as $field => $value) {
        $stmt->bindValue(":" . $field, $value, is_int($value) ? PDO::PARAM_INT : PDO::PARAM_STR);
    }

    return $stmt->execute();
}

// Pesquisar com LIKE
public function search($field, $searchTerm, $limit = null, $offset = null, $orderBy = null)
{
    $query = "SELECT * FROM {$this->table} WHERE {$field} LIKE :searchTerm";
    if ($orderBy) {
        $query .= " ORDER BY {$orderBy}";
    }
    if ($limit !== null) {
        $query .= " LIMIT :limit";
        if ($offset !== null) {
            $query .= " OFFSET :offset";
        }
    }

    $stmt = $this->conn->prepare($query);
    $stmt->bindValue(':searchTerm', "%{$searchTerm}%", PDO::PARAM_STR);

    if ($limit !== null) {
        $stmt->bindParam(':limit', $limit, PDO::PARAM_INT);
        if ($offset !== null) {
            $stmt->bindParam(':offset', $offset, PDO::PARAM_INT);
        }
    }

    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_OBJ);
}

// Executar JOIN
public function join($table, $foreignKey, $primaryKey, $columns = '*', $conditions = [])
{
    $query = "SELECT {$columns} FROM {$this->table}
        INNER JOIN {$table} ON {$this->table}.{$foreignKey} = {$table}.{$primaryKey}";

    if (!empty($conditions)) {
        $query .= " WHERE " . implode(' AND ', array_map(fn($field) => "{$field} = :{$field}", array_keys($conditions)));
    }

    $stmt = $this->conn->prepare($query);

    foreach ($conditions as $field => $value) {
        $stmt->bindValue(":" . $field, $value, is_int($value) ? PDO::PARAM_INT : PDO::PARAM_STR);
    }

    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_OBJ);
}
```

Como Usar

Exemplo de uso no controlador:

```
php

use HefestoMVC\Database\Models\Model;

// Inicializar modelo para a tabela 'articles'
$articleModel = new Model('articles');

// Buscar um artigo
$article = $articleModel->find(1);

// Inserir um novo artigo
$articleModel->insert([
    'title' => 'Novo Artigo',
    'content' => 'Conteúdo do artigo',
]);

// Atualizar um artigo
$articleModel->update(['title' => 'Título Atualizado'], ['id' => 1]);

// Deletar um artigo
$articleModel->delete(['id' => 1]);

// Buscar com condições e ordenação
$articles = $articleModel->findBy(['status' => 'published'], 10, 0, 'created_at DESC');
```

Essa implementação é genérica e cobre a maioria dos casos comuns de manipulação de dados em bancos de dados usando PDO.



mostre um exemplo de join



Aqui está um exemplo de como usar o método `join` da classe `Model`:

Cenário:

Vamos supor que temos duas tabelas no banco de dados:

1. `articles`: contém informações sobre os artigos.
 - **id** (primary key)
 - **title**
 - **content**
 - **author_id**
2. `authors`: contém informações sobre os autores.
 - **id** (primary key)
 - **name**
 - **email**

Queremos obter uma lista de artigos com informações sobre o autor, como o nome e o e-mail, usando um `INNER JOIN`.

Exemplo no Código:

```
php

use HefestoMVC\Database\Models\Model;

// Inicializar o modelo para a tabela 'articles'
$articleModel = new Model('articles');

// Fazer o join com a tabela 'authors'
$articlesWithAuthors = $articleModel->join(
    'authors',          // Tabela secundária
    'author_id',        // Chave estrangeira na tabela 'articles'
    'id',               // Chave primária na tabela 'authors'
    'articles.title, articles.content, authors.name AS author_name, authors.email AS author_email',
    ['articles.status' => 'published'] // Condições adicionais (opcional)
);

// Iterar sobre os resultados
foreach ($articlesWithAuthors as $article) {
    echo "Título: {$article->title}" . PHP_EOL;
    echo "Conteúdo: {$article->content}" . PHP_EOL;
    echo "Autor: {$article->author_name} ({$article->author_email})" . PHP_EOL;
    echo str_repeat('-', 50) . PHP_EOL;
}
```

Query Gerada:

A query SQL equivalente seria algo como:

```
sql

SELECT
    articles.title,
    articles.content,
    authors.name AS author_name,
    authors.email AS author_email
FROM
    articles
INNER JOIN
    authors
ON
    articles.author_id = authors.id
WHERE
    articles.status = 'published';
```

Explicação:

1. Parâmetros do `join`:

- Primeiro argumento: a tabela a ser unida (`authors`).
- Segundo argumento: a chave estrangeira na tabela principal (`author_id` em `articles`).
- Terceiro argumento: a chave primária na tabela secundária (`id` em `authors`).

- Quarto argumento: colunas específicas a serem selecionadas. Aqui usamos `AS` para apelidar colunas.
 - Quinto argumento: condições adicionais para o `WHERE` (opcional).
2. **Iteração dos resultados:** O método `fetchAll(PDO::FETCH_OBJ)` garante que os resultados sejam retornados como objetos, facilitando o acesso às propriedades.

Resultado Esperado:

Se houver artigos publicados no banco, a saída será algo como:

```
less

Título: Aprendendo PHP
Conteúdo: Um guia completo para iniciantes...
Autor: João Silva (joao@email.com)
-----
Título: Frameworks PHP Modernos
Conteúdo: Uma comparação entre Laravel, Symfony...
Autor: Maria Oliveira (maria@email.com)
-----
```

Essa abordagem genérica torna o método `join` útil para qualquer combinação de tabelas com relações no banco de dados.



tem muita repetição, crie metodos para serem chamados. E se puder usar assim \$article = \$articleModel->findBy()->where(id =2)->limit 5 -> join;



Podemos melhorar a classe `Model` para permitir uma abordagem mais fluida (chaining) e reutilizar métodos, evitando repetições. Aqui está a versão aprimorada:

Classe Model Melhorada:

```
php

namespace HefestoMVC\Database\Models;

use PDO;

class Model
{
    protected $table;
    protected $conn;
    protected $query;
    protected $bindings = [];

    public function __construct($table)
    {
        $this->table = $table;
        $this->conn = Connection::connect(); // Conectar ao banco de dados
        $this->resetQuery();
    }
}
```

```

private function resetQuery()
{
    $this->query = "SELECT * FROM {$this->table}";
    $this->bindings = [];
}

public function find($id)
{
    return $this->where('id', '=', $id)->limit(1)->get()[0] ?? null;
}

public function where($field, $operator, $value)
{
    $this->query .= empty($this->bindings) ? " WHERE" : " AND";
    $this->query .= " {$field} {$operator} :{$field}";
    $this->bindings[$field] = $value;

    return $this;
}

public function limit($limit)
{
    $this->query .= " LIMIT {$limit}";
    return $this;
}

public function offset($offset)
{
    $this->query .= " OFFSET {$offset}";
    return $this;
}

public function orderBy($field, $direction = 'ASC')
{
    $this->query .= " ORDER BY {$field} {$direction}";
    return $this;
}

public function join($table, $foreignKey, $primaryKey, $columns = '*')
{
    $this->query = str_replace('SELECT *', "SELECT {$columns}", $this->query);
    $this->query .= " INNER JOIN {$table} ON {$this->table}.{$foreignKey} = {$table}.{$p
rimaryKey}";
    return $this;
}

public function like($field, $value)
{
    $this->query .= empty($this->bindings) ? " WHERE" : " AND";
    $this->query .= " {$field} LIKE :{$field}";
    $this->bindings[$field] = "%{$value}%";

    return $this;
}

public function get()
{
    $stmt = $this->conn->prepare($this->query);
    foreach ($this->bindings as $param => $value) {
        $stmt->bindValue(":" . $param, $value);
    }
    $stmt->execute();
    $this->resetQuery();

    return $stmt->fetchAll(PDO::FETCH_OBJ);
}

public function insert(array $data)
{
    $columns = implode(', ', array_keys($data));
    $placeholders = ':' . implode(', ', array_keys($data));
    $query = "INSERT INTO {$this->table} ({$columns}) VALUES ({$placeholders})";

    $stmt = $this->conn->prepare($query);

```

```

        foreach ($data as $key => $value) {
            $stmt->bindValue(":{key}", $value);
        }
        return $stmt->execute();
    }

    public function update(array $data, $id)
    {
        $setClause = implode(' , ', array_map(fn($key) => "{$key} = :{$key}", array_keys($data)));
        $query = "UPDATE {$this->table} SET {$setClause} WHERE id = :id";

        $stmt = $this->conn->prepare($query);
        foreach ($data as $key => $value) {
            $stmt->bindValue(":{key}", $value);
        }
        $stmt->bindValue(":id", $id);
        return $stmt->execute();
    }

    public function delete($id)
    {
        $query = "DELETE FROM {$this->table} WHERE id = :id";
        $stmt = $this->conn->prepare($query);
        $stmt->bindValue(":id", $id);
        return $stmt->execute();
    }
}

```

Exemplo de Uso:

1. Buscar Artigos com Join e Paginação

php

```

$articleModel = new Model('articles');

// Buscar artigos publicados, com autores, ordenados por data de criação
$articles = $articleModel
    ->where('status', '=', 'published')
    ->join('authors', 'author_id', 'id', 'articles.*, authors.name AS author_name')
    ->orderBy('created_at', 'DESC')
    ->limit(10)
    ->offset(0)
    ->get();

// Iterar nos resultados
foreach ($articles as $article) {
    echo "Título: {$article->title}" . PHP_EOL;
    echo "Autor: {$article->author_name}" . PHP_EOL;
    echo "Conteúdo: " . substr($article->content, 0, 100) . "..." . PHP_EOL;
}

```

2. Inserir um Novo Registro

php

```

$articleModel = new Model('articles');
$newArticle = [
    'title' => 'Novo Artigo',
    'content' => 'Este é o conteúdo do novo artigo.',
];

```

```
'author_id' => 1,
'status' => 'draft',
];
$articleModel->insert($newArticle);
```

3. Atualizar um Registro

php

```
$articleModel = new Model('articles');
$updateData = [
    'title' => 'Título Atualizado',
    'content' => 'Conteúdo atualizado do artigo.',
];
$articleModel->update($updateData, 5);
```

4. Deletar um Registro

php

```
$articleModel = new Model('articles');
$articleModel->delete(10); // Excluir artigo com ID = 10
```

Benefícios:

1. **Interface fluida:** Os métodos podem ser encadeados para consultas mais legíveis.
2. **Abstração genérica:** A classe é reutilizável para qualquer tabela.
3. **Flexibilidade:** Suporte para consultas complexas (joins, filtros, ordenação, paginação).



Agora com offset e essa classe e se possível usar Container que criamos, faça uma paginação de resultados, uma classe



A implementação da paginação pode ser feita com uma classe separada para lidar especificamente com o cálculo e configuração dos parâmetros de paginação. Vamos integrá-la com a classe `Model` e o `Container`.

Classe de Paginação

php

```
namespace HefestoMVC\Helpers;

class Pagination
```

```
{
    private $totalRecords;
    private $perPage;
    private $currentPage;
    private $offset;

    public function __construct($totalRecords, $perPage, $currentPage = 1)
    {
        $this->totalRecords = $totalRecords;
        $this->perPage = $perPage;
        $this->currentPage = max(1, $currentPage); // Garante que a página atual seja pelo m
    }

    public function getOffset()
    {
        return $this->offset;
    }

    public function getLimit()
    {
        return $this->perPage;
    }

    public function getTotalPages()
    {
        return ceil($this->totalRecords / $this->perPage);
    }

    public function getCurrentPage()
    {
        return $this->currentPage;
    }

    public function getNextPage()
    {
        return $this->currentPage < $this->getTotalPages() ? $this->currentPage + 1 : null;
    }

    public function getPrevPage()
    {
        return $this->currentPage > 1 ? $this->currentPage - 1 : null;
    }
}
```

Uso no Controller (com o Container)

php

```
namespace HefestoMVC\Controllers;

use HefestoMVC\Database\Models\Model;
use HefestoMVC\Helpers\Pagination;
use HefestoMVCLibrary\Container;
use HefestoMVCLibrary\View;

class ArticleController
{
    private $articleModel;

    public function __construct()
    {
        // Usando o Container para resolver a dependência
        $this->articleModel = Container::get(Model::class);
        $this->articleModel->setTable('articles'); // Define a tabela dinamicamente
    }

    public function index()
```

```
{
    $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
    $perPage = 5;

    // Total de registros
    $totalRecords = $this->articleModel->countAll();

    // Configuração da paginação
    $pagination = new Pagination($totalRecords, $perPage, $currentPage);

    // Buscar registros com offset e limit
    $articles = $this->articleModel
        ->orderBy('created_at', 'DESC')
        ->limit($pagination->getLimit())
        ->offset($pagination->getOffset())
        ->get();

    // Renderizar a view
    View::render('articles/index.twig', [
        'articles' => $articles,
        'pagination' => [
            'currentPage' => $pagination->getCurrentPage(),
            'totalPages' => $pagination->getTotalPages(),
            'nextPage' => $pagination->getNextPage(),
            'prevPage' => $pagination->getPrevPage(),
        ],
    ]);
}
```

Exemplo da View (`articles/index.twig`)

```
twig

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Articles</title>
    <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h1>Articles</h1>
        <div class="row">
            {% for article in articles %}
                <div class="col-md-4 mb-3">
                    <div class="card">
                        <div class="card-body">
                            <h5 class="card-title">{{ article.title }}</h5>
                            <p class="card-text">{{ article.content[:100] }}...</p>
                            <a href="/show/{{ article.id }}" class="btn btn-primary">Read More</a>
                        </div>
                    </div>
                </div>
            {% endfor %}
        </div>

        <nav>
            <ul class="pagination justify-content-center">
                {% if pagination.prevPage %}
                    <li class="page-item">
                        <a class="page-link" href="?page={{ pagination.prevPage }}">Previous
                    </li>
                {% endif %}
            </ul>
        </nav>
    </div>
</body>
</html>
```

```

        {% for page in 1..pagination.totalPages %}
            <li class="page-item {% if page == pagination.currentPage %}active{% end
if %}">
                <a class="page-link" href="?page={{ page }}">{{ page }}</a>
            </li>
        {% endfor %}

        {% if pagination.nextPage %}
            <li class="page-item">
                <a class="page-link" href="?page={{ pagination.nextPage }}">Next</a>
            </li>
        {% endif %}
    </ul>
</nav>
</div>

<script src="/assets/js/bootstrap.bundle.min.js"></script>
</body>
</html>

```

Benefícios da Abordagem:

1. **Integração com o Container:** Todas as dependências, incluindo o modelo, são resolvidas automaticamente.
2. **Flexibilidade:** A classe de paginação é genérica e pode ser usada para qualquer tabela.
3. **Desacoplamento:** A lógica de paginação está separada do modelo, tornando o código mais modular.
4. **View Dinâmica:** Os links de paginação se adaptam ao número de páginas disponíveis.



como eu listaria todos os dados com o Model que criamos?



Para listar todos os dados de uma tabela usando o `Model` que criamos, basta usar o método `get` no `Model` para executar uma consulta genérica. Aqui está um exemplo:

Exemplo: Listar todos os dados com o `Model`

Código do Controller:

```

php

namespace HefestoMVC\Controllers;

use HefestoMVC\Database\Models\Model;
use HefestoMVCLibrary\Container;
use HefestoMVCLibrary\View;

class ArticleController
{
    private $articleModel;

    public function __construct()
    {
        // Usando o Container para resolver a dependência do Model
        $this->articleModel = Container::get(Model::class);
    }
}

```

```

        $this->articleModel->setTable('articles'); // Define a tabela dinamicamente
    }

    public function listAll()
    {
        // Busca todos os registros da tabela 'articles'
        $articles = $this->articleModel->get();

        // Renderiza a view e envia os dados para a camada de apresentação
        View::render('articles/list.twig', ['articles' => $articles]);
    }
}

```

Exemplo de View (`articles/list.twig`):

```

twig

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>All Articles</title>
    <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
</head>
<body>
    <div class="container mt-5">
        <h1>All Articles</h1>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Title</th>
                    <th>Content</th>
                    <th>Created At</th>
                </tr>
            </thead>
            <tbody>
                {% for article in articles %}
                <tr>
                    <td>{{ article.id }}</td>
                    <td>{{ article.title }}</td>
                    <td>{{ article.content[:100] }}...</td>
                    <td>{{ article.created_at }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>

    <script src="/assets/js/bootstrap.bundle.min.js"></script>
</body>
</html>

```

Métodos do `Model` Envolvidos:

1. `setTable(\$table)` : Define a tabela com a qual o modelo trabalhará.
2. `get()` : Recupera todos os registros da tabela, sem condições adicionais.

Resultado:

Quando você acessa o método `listAll` no controlador, todos os registros da tabela `articles` serão exibidos na tabela HTML renderizada pela view `articles/list.twig`.

Se precisar de paginação ou ordenação, você pode adicionar métodos fluentes como `limit`, `orderBy`, ou `offset` ao modelo antes de chamar o método `get`.



Melhore a view `home.twig` com a paginação anterior que fique tipo assim: Início 5 6 7 8 9 Fim sendo o 7 o meio. Se fosse 12 seria Início 10 11 12 13 14 Fim se for a ultima pagina tipo 51 fica algo como Início 49 50 51 Fim e se for no começo, ficaria Início 1 2 3 Fim



Para implementar uma paginação que exiba os botões de navegação no formato sugerido (com início, fim, e páginas dinâmicas ao redor da atual), precisamos ajustar a lógica de renderização na view. Aqui está uma versão melhorada da `home.twig`:

Lógica para Paginação na View (`home.twig`)

```
twig

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Home</title>
  <link rel="stylesheet" href="/assets/css/bootstrap.min.css">
</head>
<body>
  <div class="container mt-5">
    <h1>Articles</h1>

    <div class="row">
      {% for article in articles %}
        <div class="col-md-4 mb-4">
          <div class="card">
            <div class="card-body">
              <h5 class="card-title">{{ article.title }}</h5>
              <p class="card-text">{{ article.content[:100] }}...</p>
              <a href="/show/{{ article.id }}" class="btn btn-primary">Read Mo
re</a>
            </div>
          </div>
        </div>
      {% endfor %}
    </div>

    {# Paginação #}
    <nav>
      <ul class="pagination justify-content-center">
        {# Botão "Início" #}
        <li class="page-item {% if currentPage == 1 %}disabled{% endif %}">
          <a class="page-link" href="?page=1">Início</a>
        </li>

        {# Botões de páginas #}
        {% for page in pagination.pages %}
          <li class="page-item {% if page == currentPage %}active{% endif %}">
            <a class="page-link" href="?page={{ page }}">{{ page }}</a>
          </li>
        </for>
      </ul>
    </nav>
  </div>
</body>
</html>
```

```

        {% endfor %}

        {# Botão "Fim" #}
        <li class="page-item {% if currentPage == pagination.lastPage %}disabled{% e
ndif %}">
            <a class="page-link" href="?page={{ pagination.lastPage }}">Fim</a>
        </li>
    </ul>
</nav>
</div>

<script src="/assets/js/bootstrap.bundle.min.js"></script>
</body>
</html>

```

Ajustando o Controller para a Lógica de Paginação:

Aqui está um exemplo de como configurar a paginação com a classe `Pagination`:

Controller:

php

```

namespace HefestoMVC\Controllers;

use HefestoMVC\Database\Models\Model;
use HefestoMVCLibrary\Container;
use HefestoMVCLibrary\View;
use HefestoMVCLibrary\Pagination;

class HomeController
{
    private $articleModel;

    public function __construct()
    {
        $this->articleModel = Container::get(Model::class);
        $this->articleModel->setTable('articles');
    }

    public function index()
    {
        // Obtendo a página atual
        $currentPage = $_GET['page'] ?? 1;

        // Configuração da paginação
        $pagination = Container::get(Pagination::class);
        $pagination->setPagination($this->articleModel, $currentPage, 9); // 9 artigos por p
ágina

        // Dados para a view
        $data = [
            'articles' => $pagination->getData(),
            'currentPage' => $pagination->getCurrentPage(),
            'pagination' => [
                'pages' => $pagination->getPageRange(5), // Exibe um range dinâmico de 5 pág
inas
                'lastPage' => $pagination->getLastPage(),
            ],
        ];

        View::render('home.twig', $data);
    }
}

```

Classe `Pagination` Ajustada:

Adicione o método `getPageRange` para criar o intervalo dinâmico:

php

```
public function getPageRange($range = 5)
{
    $start = max(1, $this->currentPage - floor($range / 2));
    $end = min($this->lastPage, $start + $range - 1);

    // Ajusta o início se o fim atingir o limite máximo
    $start = max(1, $end - $range + 1);

    return range($start, $end);
}
```

Fluxo Completo:

1. O **Controller** usa o `Pagination` para calcular os artigos a serem exibidos e a estrutura de páginas.
2. A **View** renderiza os artigos e os botões de paginação com base na lógica.
3. A **navegação** é dinâmica, ajustando-se ao total de páginas e à página atual.

Resultado Final:

Com a lógica acima, a paginação se adapta automaticamente:

- **Primeiras páginas:** `Início 1 2 3 Fim`.
- **Meio do range:** `Início 5 6 7 8 9 Fim`.
- **Últimas páginas:** `Início 49 50 51 Fim`.



Meu controller tá assim <?php
namespace HefestoMVC\controllers;

```
use HefestoMVCLibrary\View;
use HefestoMVC\helpers\Pagination;
use HefestoMVC\database\models\Article;
```

```
class HomeController
{
```

```
    private $articleModel;
    private $pagination;
```

```
    public function __construct(Article $articleModel, Pagination $pagination)
    {
        $this->articleModel = $articleModel;
        $this->pagination = $pagination;
    }
}
```

```

}

public function index()
{
    $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
    $perPage = 5;

    // Total de registros
    $totalRecords = $this->articleModel->countAll();

    // Configuração da paginação
    $pagination = new Pagination($totalRecords, $perPage, $currentPage);

    // Buscar registros com offset e limit
    $articles = $this->articleModel
        ->orderBy('created', 'DESC')
        ->limit($pagination->getLimit())
        ->offset($pagination->getOffset())
        ->get();

    // Renderizar a view
    View::render('home.twig', [
        'articles' => $articles,
        'pagination' => [
            'currentPage' => $pagination->getCurrentPage(),
            'totalPages' => $pagination->getTotalPages(),
            'nextPage' => $pagination->getNextPage(),
            'prevPage' => $pagination->getPrevPage(),
        ],
    ]);
}

public function show(string $title, int $id)
{
    $article = $this->articleModel->find($id);

    // Passar os dados do artigo para a view
    View::render('article/show.twig', ['article' => $article]);
}
} e pagination assim <?php

```

```
namespace HefestoMVC\helpers;
```

```

class Pagination
{
    private $totalRecords;
    private $perPage;
    private $currentPage;
    private $offset;
    private $lastPage;
}

```

```

public function __construct($totalRecords, $perPage, $currentPage = 1)
{
    $this->totalRecords = $totalRecords;
    $this->perPage = $perPage;
    $this->currentPage = max(1, $currentPage); // Garante que a página atual seja pelo menos 1
    $this->offset = ($this->currentPage - 1) * $this->perPage;
}

public function getOffset()
{
    return $this->offset;
}

public function getLimit()
{
    return $this->perPage;
}

public function getTotalPages()
{
    return ceil($this->totalRecords / $this->perPage);
}

public function getCurrentPage()
{
    return $this->currentPage;
}

public function getNextPage()
{
    return $this->currentPage < $this->getTotalPages() ? $this->currentPage + 1 : null;
}

public function getPrevPage()
{
    return $this->currentPage > 1 ? $this->currentPage - 1 : null;
}

public function getPageRange($range = 5)
{
    $start = max(1, $this->currentPage - floor($range / 2));
    $end = min($this->lastPage, $start + $range - 1);

    // Ajusta o início se o fim atingir o limite máximo
    $start = max(1, $end - $range + 1);

    return range($start, $end);
}
}

```



Sua implementação do controller e da classe de paginação está no caminho certo! Vou ajudar ajustando os pontos pendentes para garantir que a paginação dinâmica funcione corretamente e esteja alinhada com a estrutura sugerida.

Ajustando a Classe de Paginação

Na classe `Pagination`, o cálculo da última página (`lastPage`) precisa ser ajustado para garantir que seja consistente com o número total de páginas. Além disso, o método `getPageRange` precisa ser corrigido para usar `getTotalPages`.

Aqui está a classe revisada:

```
php

namespace HefestoMVC\helpers;

class Pagination
{
    private $totalRecords;
    private $perPage;
    private $currentPage;
    private $offset;

    public function __construct($totalRecords, $perPage, $currentPage = 1)
    {
        $this->totalRecords = $totalRecords;
        $this->perPage = $perPage;
        $this->currentPage = max(1, $currentPage); // Garante que a página atual seja pelo m
        $this->offset = ($this->currentPage - 1) * $this->perPage;
    }

    public function getOffset()
    {
        return $this->offset;
    }

    public function getLimit()
    {
        return $this->perPage;
    }

    public function getTotalPages()
    {
        return (int) ceil($this->totalRecords / $this->perPage);
    }

    public function getCurrentPage()
    {
        return $this->currentPage;
    }

    public function getNextPage()
    {
        $next = $this->currentPage + 1;
        return $next <= $this->getTotalPages() ? $next : null;
    }

    public function getPrevPage()
    {
        $prev = $this->currentPage - 1;
        return $prev >= 1 ? $prev : null;
    }

    public function getPageRange($range = 5)
```

```
{
    $totalPages = $this->getTotalPages();

    $start = max(1, $this->currentPage - floor($range / 2));
    $end = min($totalPages, $start + $range - 1);

    // Ajusta o início se o fim atingir o limite máximo
    $start = max(1, $end - $range + 1);

    return range($start, $end);
}
```

Ajustando o Controller

Aqui está a versão ajustada do controller para usar a paginação dinâmica corretamente:

php

```
namespace HefestoMVC\controllers;

use HefestoMVCLibrary\View;
use HefestoMVC\helpers\Pagination;
use HefestoMVC\database\models\Article;

class HomeController
{
    private $articleModel;
    private $pagination;

    public function __construct(Article $articleModel, Pagination $pagination)
    {
        $this->articleModel = $articleModel;
        $this->pagination = $pagination;
    }

    public function index()
    {
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        $perPage = 5;

        // Total de registros
        $totalRecords = $this->articleModel->countAll();

        // Configuração da paginação
        $pagination = new Pagination($totalRecords, $perPage, $currentPage);

        // Buscar registros com offset e limit
        $articles = $this->articleModel
            ->orderBy('created', 'DESC')
            ->limit($pagination->getLimit())
            ->offset($pagination->getOffset())
            ->get();

        // Dados da paginação
        $paginationData = [
            'currentPage' => $pagination->getCurrentPage(),
            'totalPages' => $pagination->getTotalPages(),
            'pages' => $pagination->getPageRange(5), // Range dinâmico de 5 páginas
            'nextPage' => $pagination->getNextPage(),
            'prevPage' => $pagination->getPrevPage(),
        ];

        // Renderizar a view
        View::render('home.twig', [
            'articles' => $articles,
            'pagination' => $paginationData,
        ]);
    }
}
```

```

}

public function show(string $title, int $id)
{
    $article = $this->articleModel->find($id);

    // Passar os dados do artigo para a view
    View::render('article/show.twig', ['article' => $article]);
}
}

```

Ajustando a View `home.twig`

Aqui está a view com a paginação dinâmica implementada no formato sugerido:

```

twig

<nav>
<ul class="pagination justify-content-center">
    {# Botão "Início" #}
    <li class="page-item {% if pagination.currentPage == 1 %}disabled{% endif %}">
        <a class="page-link" href="?page=1">Início</a>
    </li>

    {# Botão "Anterior" #}
    {% if pagination.prevPage %}
        <li class="page-item">
            <a class="page-link" href="?page={{ pagination.prevPage }}">&laquo;</a>
        </li>
    {% endif %}

    {# Botões das páginas #}
    {% for page in pagination.pages %}
        <li class="page-item {% if page == pagination.currentPage %}active{% endif %}">
            <a class="page-link" href="?page={{ page }}">{{ page }}</a>
        </li>
    {% endfor %}

    {# Botão "Próximo" #}
    {% if pagination.nextPage %}
        <li class="page-item">
            <a class="page-link" href="?page={{ pagination.nextPage }}">&raquo;</a>
        </li>
    {% endif %}

    {# Botão "Fim" #}
    <li class="page-item {% if pagination.currentPage == pagination.totalPages %}disabled{% endif %}">
        <a class="page-link" href="?page={{ pagination.totalPages }}">Fim</a>
    </li>
</ul>
</nav>

```

Como Funciona

1. Paginação Dinâmica:

- O método `getPageRange` define o intervalo dinâmico de páginas baseado na posição da página atual.
- Os botões "Início" e "Fim" garantem navegação direta para as extremidades.

2. Controller:

- Carrega os dados paginados e a estrutura de navegação.
- Passa os dados necessários para a view.

3. View:

- Renderiza os artigos e a navegação dinâmica.

Com essas mudanças, sua paginação está configurada para ser eficiente e atender ao formato desejado.



tem que arrumar isso `http://127.0.0.1:8000/?page=2` para que o Route e Router entendam e carreguem a paginação correta <?php

```
namespace HefestoMVCLibrary;
```

```
class Route
```

```
{
    private static array $routes = [];
    private array $middleware = [];

    public function __construct(private string $method, private string $uri, private $action) {}

    public static function get(string $uri, $action) <?php
```

```
namespace HefestoMVCLibrary;
```

```
use Exception;
use HefestoMVCLibrary\Route;
use HefestoMVCLibrary\MiddlewareHandler;
use HefestoMVC\controllers\NotFoundController;
```

```
class Router
```

```
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                // Aqui você pode adicionar a lógica para middleware, se necessário
                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();
                if (is_array($action)) {
                    [$controller, $method] = $action;

                    // Usando o Container para resolver a instância do controlador
                    $controllerInstance = Container::get($controller);
                    return call_user_func_array([$controllerInstance, $method], $params);
                }
            }
        }
    }
}
```

```

        if (is_callable($action)) {
            return call_user_func_array($action, $params);
        }
    }
}

$this->handleNotFound();
}

private function handleNotFound()
{
    http_response_code(404);
    $controller = new NotFoundController();
    if (method_exists($controller, 'index')) {
        return $controller->index();
    }

    throw new Exception("NotFoundController ou método 'index' não implementado.");
}
{
    {
        return self::addRoute('GET', $uri, $action);
    }

    public static function post(string $uri, $action)
    {
        return self::addRoute('POST', $uri, $action);
    }

    private static function addRoute(string $method, string $uri, $action)
    {
        $route = new self($method, $uri, $action);
        self::$routes[] = $route;
        return $route;
    }

    public function middleware($middleware)
    {
        $this->middleware = is_array($middleware) ? $middleware : [$middleware];
    }

    public static function getRoutes()
    {
        return self::$routes;
    }

    public function matches(string $requestUri, string $requestMethod)
    {
        if ($this->method !== $requestMethod) {
            return false;
        }
    }
}

```

```

    }

    $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
    return preg_match("#^{$pattern}$#", $requestUri);
}

public function extractParameters(string $requestUri)
{
    $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
    preg_match("#^{$pattern}$#", $requestUri, $matches);
    array_shift($matches);
    return $matches;
}

public function getAction()
{
    return $this->action;
}

public function getMiddleware()
{
    return $this->middleware;
}
}

```



Para ajustar o roteador para entender e lidar corretamente com parâmetros de consulta (como `?page=2`), precisamos estender a funcionalidade de correspondência (`matches`) e extração de parâmetros (`extractParameters`) para incluir query strings. Além disso, garantimos que o parâmetro `page` seja acessível dentro do controlador.

Aqui está uma solução refinada:

Ajustando a Classe `Route`

A classe `Route` precisa ser ajustada para lidar com query strings:

```

php

namespace HefestoMVCLibrary;

class Route
{
    private static array $routes = [];
    private array $middleware = [];

    public function __construct(private string $method, private string $uri, private $action) {}

    public static function get(string $uri, $action)
    {
        return self::addRoute('GET', $uri, $action);
    }
}

```

```

public static function post(string $uri, $action)
{
    return self::addRoute('POST', $uri, $action);
}

private static function addRoute(string $method, string $uri, $action)
{
    $route = new self($method, $uri, $action);
    self::$routes[] = $route;
    return $route;
}

public function middleware($middleware)
{
    $this->middleware = is_array($middleware) ? $middleware : [$middleware];
}

public static function getRoutes()
{
    return self::$routes;
}

public function matches(string $requestUri, string $requestMethod): bool
{
    if ($this->method !== $requestMethod) {
        return false;
    }

    // Remove query string para comparação da rota base
    $baseUri = strtok($requestUri, '?');
    $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);

    return preg_match("#^{$pattern}$#", $baseUri);
}

public function extractParameters(string $requestUri): array
{
    $baseUri = strtok($requestUri, '?');
    $pattern = preg_replace('/\{[^}]+\}/', '([^/]+)', $this->uri);
    preg_match("#^{$pattern}$#", $baseUri, $matches);
    array_shift($matches);

    // Inclui parâmetros da query string
    $queryString = [];
    parse_str(parse_url($requestUri, PHP_URL_QUERY), $queryString);

    return array_merge($matches, $queryString);
}

public function getAction()
{
    return $this->action;
}

public function getMiddleware()
{
    return $this->middleware;
}
}

```

Ajustando o Router

Agora o `Router` precisa despachar corretamente, incluindo parâmetros da query string:

```

php

namespace HefestoMVCLibrary;

```

```

use Exception;
use HefestoMVC\controllers\NotFoundController;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                // Middleware handling (se necessário)
                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();

                if (is_array($action)) {
                    [$controller, $method] = $action;

                    // Resolver o controlador usando o Container
                    $controllerInstance = Container::get($controller);
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        $this->handleNotFound();
    }

    private function handleNotFound()
    {
        http_response_code(404);
        $controller = new NotFoundController();
        if (method_exists($controller, 'index')) {
            return $controller->index();
        }

        throw new Exception("NotFoundController ou método 'index' não implementado.");
    }
}

```

Usando no Controlador

No controlador, os parâmetros agora podem ser acessados diretamente:

php

```

public function index($page = 1)
{
    $currentPage = max(1, (int) $page);
    $perPage = 5;

    // Total de registros
    $totalRecords = $this->articleModel->countAll();

    // Configuração da paginação
    $pagination = new Pagination($totalRecords, $perPage, $currentPage);

    // Buscar registros com offset e limit
    $articles = $this->articleModel
        ->orderBy('created', 'DESC')
        ->limit($pagination->getLimit())
        ->offset($pagination->getOffset())
        ->get();
}

```

```
// Dados da paginação
$paginationData = [
    'currentPage' => $pagination->getCurrentPage(),
    'totalPages' => $pagination->getTotalPages(),
    'pages' => $pagination->getPageRange(5), // Range dinâmico de 5 páginas
    'nextPage' => $pagination->getNextPage(),
    'prevPage' => $pagination->getPrevPage(),
];

// Renderizar a view
View::render('home.twig', [
    'articles' => $articles,
    'pagination' => $paginationData,
]);
```

Configurando a Rota

A rota para a página inicial com paginação seria configurada assim:

```
php

use HefestoMVCLibrary\Route;

Route::get('/', [HomeController::class, 'index']);
```

Se precisar de rotas para páginas específicas:

```
php

Route::get('/page/{page}', [HomeController::class, 'index']);
```

Dessa forma, sua aplicação suportará tanto `/` quanto `/page/{page}` e os parâmetros de consulta (`?page=2`) serão devidamente tratados e despachados pelo roteador.



Não foi mexido no Router e deu erros



Entendido! O erro ocorre porque o roteador não está processando corretamente os parâmetros da query string, como o `?page=2`, que são essenciais para a paginação. Vamos corrigir isso sem alterar a estrutura existente do seu `Router` e `Route`. Vou adaptar a forma como o Router lida com os parâmetros da URL e com a query string.

Alterando o `Router` para Suportar Query Strings

O problema é que o `Router` não está considerando os parâmetros da query string ao corresponder a uma rota. Para resolver isso, precisamos ajustar a forma como ele lida com a URL e extrai os parâmetros da query string.

Aqui está a solução:

Ajustando a Classe `Route`

Na classe `Route`, precisamos apenas garantir que a query string seja extraída e manipulada corretamente. O objetivo aqui é ajustar a extração de parâmetros para não apenas considerar os parâmetros da rota, mas também a query string.

Atualização do método `matches` e `extractParameters`:

```
php

namespace HefestoMVCLibrary;

class Route
{
    private static array $routes = [];
    private array $middleware = [];

    public function __construct(private string $method, private string $uri, private $action) {}

    public static function get(string $uri, $action)
    {
        return self::addRoute('GET', $uri, $action);
    }

    public static function post(string $uri, $action)
    {
        return self::addRoute('POST', $uri, $action);
    }

    private static function addRoute(string $method, string $uri, $action)
    {
        $route = new self($method, $uri, $action);
        self::$routes[] = $route;
        return $route;
    }

    public function middleware($middleware)
    {
        $this->middleware = is_array($middleware) ? $middleware : [$middleware];
    }

    public static function getRoutes()
    {
        return self::$routes;
    }

    public function matches(string $requestUri, string $requestMethod): bool
    {
        if ($this->method !== $requestMethod) {
            return false;
        }

        // Remove query string para comparação da rota base
        $baseUri = strtok($requestUri, '?');
        $pattern = preg_replace('/\{\{[^}\}]+\}/', '([^\s]+)', $this->uri);

        return preg_match("#^{$pattern}$#", $baseUri);
    }

    public function extractParameters(string $requestUri): array
    {
        // Remove a query string da URL para fazer o match
        $baseUri = strtok($requestUri, '?');
        $pattern = preg_replace('/\{\{[^}\}]+\}/', '([^\s]+)', $this->uri);
    }
}
```

```

preg_match("#^{${pattern}}$", $baseUri, $matches);
array_shift($matches);

// Agora, extraímos os parâmetros da query string
$queryString = [];
parse_str(parse_url($requestUri, PHP_URL_QUERY), $queryString);

// Retorna tanto os parâmetros da rota quanto os da query string
return array_merge($matches, $queryString);
}

public function getAction()
{
    return $this->action;
}

public function getMiddleware()
{
    return $this->middleware;
}
}

```

Atualizando o Método `dispatch` do `Router`

Agora que a `Route` está preparada para lidar com parâmetros da query string, vamos garantir que o `Router` os passe corretamente para o controlador.

Atualização do método `dispatch`:

```

php

namespace HefestoMVCLibrary;

use Exception;
use HefestoMVC\controllers\NotFoundController;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                $params = $route->extractParameters($requestUri);

                // Middleware handling (se necessário)
                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();

                if (is_array($action)) {
                    [$controller, $method] = $action;

                    // Resolver o controlador usando o Container
                    $controllerInstance = Container::get($controller);
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        $this->handleNotFound();
    }
}

```



```
private function handleNotFound()
{
    http_response_code(404);
    $controller = new NotFoundController();
    if (method_exists($controller, 'index')) {
        return $controller->index();
    }

    throw new Exception("NotFoundController ou método 'index' não implementado.");
}
}
```

Controlador `HomeController`

Agora que os parâmetros da query string estão sendo corretamente passados para o controlador, podemos usar o valor de `page` diretamente no controlador. No caso de `?page=2`, o valor de `page` será acessado e utilizado para gerar a paginação correta.

php

```
namespace HefestoMVC\controllers;

use HefestoMVCLibrary\View;
use HefestoMVC\helpers\Pagination;
use HefestoMVC\database\models\Article;

class HomeController
{
    private $articleModel;
    private $pagination;

    public function __construct(Article $articleModel, Pagination $pagination)
    {
        $this->articleModel = $articleModel;
        $this->pagination = $pagination;
    }

    public function index($page = 1)
    {
        $currentPage = max(1, (int) $page);
        $perPage = 5;

        // Total de registros
        $totalRecords = $this->articleModel->countAll();

        // Configuração da paginação
        $pagination = new Pagination($totalRecords, $perPage, $currentPage);

        // Buscar registros com offset e limit
        $articles = $this->articleModel
            ->orderBy('created', 'DESC')
            ->limit($pagination->getLimit())
            ->offset($pagination->getOffset())
            ->get();

        // Dados da paginação
        $paginationData = [
            'currentPage' => $pagination->getcurrentPage(),
            'totalPages' => $pagination->getTotalPages(),
            'pages' => $pagination->getPageRange(5), // Range dinâmico de 5 páginas
            'nextPage' => $pagination->getNextPage(),
            'prevPage' => $pagination->getPrevPage(),
        ];

        // Renderizar a view
        View::render('home.twig', [
            'articles' => $articles,
        ];
```

```

        'pagination' => $paginationData,
    });
}

public function show(string $title, int $id)
{
    $article = $this->articleModel->find($id);

    // Passar os dados do artigo para a view
    View::render('article/show.twig', ['article' => $article]);
}
}

```

Rota para Paginação

Agora, você pode definir as rotas para permitir que o parâmetro `page` seja passado diretamente na URL ou como parte da query string.

Por exemplo:

```

php

use HefestoMVCLibrary\Route;

// Rota padrão para a home
Route::get('/', [HomeController::class, 'index']);

// Rota com suporte para páginação via query string
Route::get('/page/{page}', [HomeController::class, 'index']);

```

Ou diretamente pela query string, como você já está fazendo (`?page=2`), sem necessidade de mudar as rotas.

Agora, ao acessar `/page/2` ou `/index?page=2`, a paginação será corretamente processada e você poderá navegar entre as páginas.



Acho que ficou muito ruim isso. tente melhorar sem incluir uma nova rota



Entendi! Vamos melhorar a solução para garantir que a lógica de paginação funcione de forma simples e eficaz, sem incluir uma nova rota e mantendo a estrutura que você já possui.

A ideia é usar o parâmetro `page` diretamente na query string, ou seja, ao acessar a URL com `?page=2`, por exemplo, a página será processada corretamente sem precisar mudar a forma como a rota é configurada.

Melhorias:

1. **Ajuste do `Router` e `Route`:** Podemos melhorar a extração da query string e a passagem dos parâmetros para o controlador. Não precisaremos de uma rota extra para a paginação, apenas

vamos ajustar o comportamento do `Router` para lidar com a `page` diretamente na query string.

2. **Método de Paginação no `HomeController`**: No `HomeController`, vamos pegar o valor da query string (`?page=2`) diretamente, sem precisar de uma nova rota, mas mantendo a lógica de paginação intacta.

Passo 1: Ajuste no `Router` para Suporte à Query String

Na classe `Router`, precisamos garantir que o método `dispatch` leve em conta a query string, de forma que o parâmetro `page` seja extraído automaticamente. Além disso, vamos modificar a extração de parâmetros para que a URL funcione com a query string.

Atualizando o `dispatch` no `Router`

```
php

namespace HefestoMVCLibrary;

use Exception;
use HefestoMVC\controllers\NotFoundController;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                // Extrair os parâmetros da URL
                $params = $route->extractParameters($requestUri);

                // Extrair parâmetros da query string
                $queryParams = $this->extractQueryParams($requestUri);

                // Combinar os parâmetros da rota com os da query string
                $params = array_merge($params, $queryParams);

                // Middleware handling (se necessário)
                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();

                if (is_array($action)) {
                    [$controller, $method] = $action;

                    // Resolver o controlador usando o Container
                    $controllerInstance = Container::get($controller);
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        $this->handleNotFound();
    }

    private function extractQueryParams(string $uri)
    {
        // Extrair parâmetros da query string
        parse_str(parse_url($uri, PHP_URL_QUERY), $queryParams);
        return $queryParams;
    }

    private function handleNotFound()
    {

```

```

        http_response_code(404);
        $controller = new NotFoundController();
        if (method_exists($controller, 'index')) {
            return $controller->index();
        }

        throw new Exception("NotFoundController ou método 'index' não implementado.");
    }
}

```

Passo 2: Ajuste no `HomeController` para Usar Query String para Paginação

Agora que o `Router` está extrair corretamente os parâmetros da URL, podemos pegar o valor da página diretamente da query string (se presente) no `HomeController`.

Atualizando o método `index` no `HomeController`:

php

```

namespace HefestoMVC\controllers;

use HefestoMVCLibrary\View;
use HefestoMVC\helpers\Pagination;
use HefestoMVC\database\models\Article;

class HomeController
{
    private $articleModel;
    private $pagination;

    public function __construct(Article $articleModel, Pagination $pagination)
    {
        $this->articleModel = $articleModel;
        $this->pagination = $pagination;
    }

    public function index()
    {
        // Pegando a página da query string, com valor padrão de 1 se não existir
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        $perPage = 5;

        // Total de registros
        $totalRecords = $this->articleModel->countAll();

        // Configuração da paginação
        $pagination = new Pagination($totalRecords, $perPage, $currentPage);

        // Buscar registros com offset e limit
        $articles = $this->articleModel
            ->orderBy('created', 'DESC')
            ->limit($pagination->getLimit())
            ->offset($pagination->getOffset())
            ->get();

        // Dados da paginação
        $paginationData = [
            'currentPage' => $pagination->getCurrentPage(),
            'totalPages' => $pagination->getTotalPages(),
            'pages' => $pagination->getPageRange(5), // Range dinâmico de 5 páginas
            'nextPage' => $pagination->getNextPage(),
            'prevPage' => $pagination->getPrevPage(),
        ];

        // Renderizar a view
        View::render('home.twig', [
            'articles' => $articles,

```

```

        'pagination' => $paginationData,
    });
}

public function show(string $title, int $id)
{
    $article = $this->articleModel->find($id);

    // Passar os dados do artigo para a view
    View::render('article/show.twig', ['article' => $article]);
}
}

```

Passo 3: Mantendo a Rota Atual

Neste caso, **não precisamos modificar a rota**, pois estamos usando o parâmetro ``page`` diretamente da query string (como ``?page=2``) e o ``Router`` já está preparado para lidar com isso sem a necessidade de uma nova rota.

Rota Configurada

A rota para o controlador ``HomeController`` permanece a mesma:

```

php

use HefestoMVCLibrary\Route;

// Rota padrão para a home (que pode usar a query string ?page=1)
Route::get('/', [HomeController::class, 'index']);

```

Com isso, ao acessar a URL ``http://127.0.0.1:8000/?page=2``, o ``Router`` será capaz de processar o parâmetro ``page=2`` e passar esse valor para o controlador, que então irá gerar a paginação corretamente.

Resumo

- ``Router``: Ajustamos para garantir que os parâmetros da query string, como ``page``, sejam extraídos corretamente e passados para o controlador.
- ``HomeController``: Mantivemos a lógica de paginação intacta, mas agora ela usa o valor de ``page`` da query string.
- **Rota**: A rota não precisa de alterações, já que estamos apenas extraindo ``page`` da URL.

Com isso, a paginação funciona corretamente sem a necessidade de criar novas rotas ou alterar a estrutura do seu sistema.



Deprecated: parse_str(): Passing null to parameter #1 (\$string) of type string is deprecated in /mnt/d/myProjects/_projects/hefesto-mvc-php/library/Route.php on line 58



O erro que você está recebendo ocorre porque o `parse_str()` está recebendo um valor `null` em vez de uma string válida. Isso normalmente acontece quando não há query string na URL, o que faz com que o `parse_url()` não consiga extrair nada.

Vamos corrigir isso verificando se a query string está presente antes de tentar usá-la. Aqui está como podemos ajustar o código para evitar o erro:

Ajuste no `Router` para Evitar o Erro

Alteraremos o método `extractQueryParams` para garantir que a query string seja extraída de forma segura:

Código Atualizado para o `Router`

```
php

namespace HefestoMVCLibrary;

use Exception;
use HefestoMVC\controllers\NotFoundController;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                // Extrair os parâmetros da URL
                $params = $route->extractParameters($requestUri);

                // Extrair parâmetros da query string de forma segura
                $queryParams = $this->extractQueryParams($requestUri);

                // Combinar os parâmetros da rota com os da query string
                $params = array_merge($params, $queryParams);

                // Middleware handling (se necessário)
                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();

                if (is_array($action)) {
                    [$controller, $method] = $action;

                    // Resolver o controlador usando o Container
                    $controllerInstance = Container::get($controller);
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        $this->handleNotFound();
    }

    private function extractQueryParams(string $uri)
    {
        // Extrair parâmetros da query string de forma segura
        $queryParams = [];
        $parsedUrl = parse_url($uri);

        if (isset($parsedUrl['query'])) {
            parse_str($parsedUrl['query'], $queryParams);
        }
    }
}
```

```

        return $queryParams;
    }

    private function handleNotFound()
    {
        http_response_code(404);
        $controller = new NotFoundController();
        if (method_exists($controller, 'index')) {
            return $controller->index();
        }

        throw new Exception("NotFoundController ou método 'index' não implementado.");
    }
}

```

Explicação do Ajuste:

1. `extractQueryParams`: A principal mudança foi verificar se a URL tem uma query string antes de tentar usar o `parse_str()`. O método `parse_url()` agora está sendo usado para garantir que a URL seja parseada corretamente, e a query string é extraída apenas se existir (`isset($parsedUrl['query'])`).
2. **Evitar `null` no `parse_str()`**: O `parse_str()` só é chamado se a query string for encontrada, evitando o erro de passagem de `null`.

Com essas alterações, você não deverá mais ver o aviso de depreciação, pois agora garantimos que o `parse_str()` será chamado apenas quando houver uma query string válida.



Fatal error: Uncaught Error: Unknown named parameter \$page in
/mnt/d/myProjects/_projects/hefesto-mvc-php/library/Router.php on line 35



O erro `Unknown named parameter $page` ocorre porque a forma como você está passando parâmetros para o controlador não corresponde ao que o PHP espera, principalmente quando você está tentando passar um parâmetro diretamente pela URL e o método do controlador está tentando obter esse parâmetro.

Esse problema ocorre principalmente quando você utiliza rotas que contêm parâmetros na URL, como `/articles/{id}`, e o parâmetro extraído da URL não é tratado corretamente quando você chama o método do controlador.

Vamos corrigir isso de forma mais robusta, considerando que sua rota pode ter parâmetros, e que esses parâmetros precisam ser passados corretamente para o método do controlador.

O que precisa ser ajustado:

1. **Parâmetros da rota e query string**: Você está misturando parâmetros de rota e parâmetros da query string. Precisamos garantir que os parâmetros sejam passados de forma adequada para o método do controlador.

2. **Uso correto do `call_user_func_array`**: O PHP espera que os parâmetros passados para o método de um controlador correspondam à sua assinatura (o nome dos parâmetros no método).

Solução proposta

1. **Melhorar o tratamento dos parâmetros**: Garantir que os parâmetros de URL e query string sejam corretamente passados para o método do controlador.
2. **Refatorar a extração dos parâmetros para garantir compatibilidade**.

Código Atualizado para o `Router`

Aqui está a versão corrigida do código:

```
php

namespace HefestoMVCLibrary;

use Exception;
use HefestoMVC\controllers\NotFoundController;

class Router
{
    public function dispatch(string $requestUri, string $requestMethod)
    {
        foreach (Route::getRoutes() as $route) {
            if ($route->matches($requestUri, $requestMethod)) {
                // Extrair os parâmetros da URL (parâmetros de rota)
                $params = $route->extractParameters($requestUri);

                // Extrair parâmetros da query string de forma segura
                $queryParams = $this->extractQueryParams($requestUri);

                // Combinar os parâmetros da rota com os da query string
                $params = array_merge($params, $queryParams);

                // Middleware handling (se necessário)
                MiddlewareHandler::handle($route->getMiddleware());

                $action = $route->getAction();

                if (is_array($action)) {
                    [$controller, $method] = $action;

                    // Resolver o controlador usando o Container
                    $controllerInstance = Container::get($controller);

                    // Usar call_user_func_array para passar os parâmetros para o método
                    return call_user_func_array([$controllerInstance, $method], $params);
                }

                if (is_callable($action)) {
                    return call_user_func_array($action, $params);
                }
            }
        }

        $this->handleNotFound();
    }

    private function extractQueryParams(string $uri)
    {
        // Extrair parâmetros da query string de forma segura
        $queryParams = [];
        $parsedUrl = parse_url($uri);

        if (isset($parsedUrl['query'])) {
            parse_str($parsedUrl['query'], $queryParams);
        }
    }
}
```



```

    }

    return $queryParams;
}

private function handleNotFound()
{
    http_response_code(404);
    $controller = new NotFoundController();
    if (method_exists($controller, 'index')) {
        return $controller->index();
    }

    throw new Exception("NotFoundController ou método 'index' não implementado.");
}
}

```

Explicação das Mudanças:

1. **Combinação de parâmetros:** Quando a URL da rota contém parâmetros (por exemplo, `/articles/{id}`), esses parâmetros são extraídos corretamente pela função `extractParameters()`. Após isso, os parâmetros da query string (se houver) são extraídos pela função `extractQueryParams()` e combinados com os parâmetros da rota. Isso garante que todos os parâmetros necessários sejam passados para o método do controlador.
2. `call_user_func_array()`: Ao usar `call_user_func_array()`, os parâmetros extraídos são passados corretamente para o método do controlador, evitando o erro de "Unknown named parameter".

Exemplo de Rota:

Considerando que você tenha uma rota como:

```

php

Route::get('/articles', [ArticleController::class, 'index']);
Route::get('/articles/{id}', [ArticleController::class, 'show']);

```

Com isso, se a URL for `http://127.0.0.1:8000/articles?page=2`, a query string `page=2` será extraída e passada para o método do controlador.

Teste:

Após essas mudanças, você deve conseguir acessar as rotas e passar os parâmetros de forma correta, incluindo os parâmetros da query string como `page`, sem o erro de "Unknown named parameter".