



# Curso PHP

Do XLSX ao CMS (aula 13)

---

# Menu dinâmico



# Menu dinâmico

Aproveitando os recursos de macro do Twig, podemos abstrair a lógica de detecção do link ativo com base na URI atual.

```
View::addGlobals([
    'APP_LOCALE' => str_replace('_', '-', env('APP_LOCALE')),
    'CURRENT_URI' => $request->getPathInfo(),
    'MAIN_MENU' => [
        'Agendamentos' => '/agendamentos',
        'Períodos' => '/periodos',
        'Disciplinas' => '/disciplinas',
        'Atividades' => '/atividades',
    ],
]);

{% macro main_menu(links) %}
<ul>
    {% for label, href in links %}
    <li>
        {% if CURRENT_URI starts with href %}
        <a href="{{ href }}" aria-current="true"><strong>{{ label }}</strong></a>
        {% else %}
        <a href="{{ href }}" aria-current="false">{{ label }}</a>
        {% endif %}
    </li>
    {% endfor %}
</ul>
{% endmacro %}
```

---

# Fluxo HTTP

# Requisição

---



# Requisição

Utilizamos o **component HttpFoundation** do Symfony para gestão do processo de requisição. Contudo, precisamos interceptar alguns comportamentos específicos.

Neste cenário, definimos uma instância do componente e encaminhamos a ele os comportamentos esperados.

Esta abordagem é necessária, pois iremos usar métodos que não existe no componente. Assim, realizamos um encapsulamento e não uma extensão.

```
<?php

namespace App\Http;

use App\Traits\IsSingleton;
use Symfony\Component\HttpFoundation\Request as BaseRequest;

class Request
{
    use IsSingleton;

    private BaseRequest $request;

    public static function boot(): static
    {
        static::getInstance()->request = BaseRequest::createFromGlobals();

        return static::getInstance();
    }

    public function __get($name): mixed
    {
        return $this->request->{$name};
    }

    public function __call($method, $args): mixed
    {
        return call_user_func_array([$this->request, $method], $args);
    }
}
```

# Sessão

---



# Symfony Sessions

O componente *Symfony HttpFoundation* possui um subsistema de sessão muito poderoso e flexível, projetado para fornecer gerenciamento de sessão que você pode usar para armazenar informações sobre o usuário entre solicitações por meio de uma interface orientada a objetos clara usando uma variedade de drivers de armazenamento de sessão.

*Symfony sessions* são projetadas para substituir o uso das funções PHP super globais e nativas `$_SESSION` relacionadas à manipulação da sessão como `session_start()`, `session_regenerate_id()`, `session_id()`, `session_name()` e `session_destroy()`.

([Symfony Docs](#))





# Sessão

Sessões são registro do estado da aplicação, isto é, um identificador (cookie ou token) permite conectar o cliente (navegador do usuário) ao estado isolado da aplicação.

Assim, tudo que o usuário fizer, ficará registrado e poderá ser recuperar ao navegar entre páginas, pois este estado será recuperado na próxima requisição.

```
// captura requisição
$request = Request::boot();

// habilita reescrita de método (PUT, PATCH, DELETE)
$request->enableHttpMethodParameterOverride();

// define sessão
$request->setSession(new Session(
    new NativeSessionStorage(handler: new NativeFileSessionHandler)
));

// adiciona variáveis globais ao contexto da view
View::addGlobals([
    'APP_LOCALE' => str_replace('_', '-', env('APP_LOCALE')),
    'CURRENT_URI' => $request->getPathInfo(),
    'ERRORS' => $request->getErrors(),
    'MAIN_MENU' => [
        'Agendamentos' => '/agendamentos',
        'Períodos' => '/periodos',
        'Disciplinas' => '/disciplinas',
        'Atividades' => '/atividades',
    ],
]);
```

# Validação

---



# Respect Validation

A **Respect Validation** oferece uma abordagem simplificada para a validação de dados, enfatizando a simplicidade e a flexibilidade na integração com outras estruturas de validação como *Symfony* e *Zend*.

O pacote permite a criação de regras de validação complexas através da encadeamento e combinação de validadores, fornecendo ferramentas robustas para atender a requisitos específicos de negócios.

Respect Validation suporta negação de regras e regras condicionais, permitindo que os desenvolvedores definam com precisão o que os dados não devem fazer, aumentando assim a segurança e a integridade do aplicativo.

([Younes Rafie](#))

# composer require respect/validation

Instala a dependência

---



# Validação: regras

Para validar os dados enviados pelo usuário, precisamos criar regras.

Estas regras devem residir em um local de fácil acesso.

Usaremos o próprio model, pois deixa a regra mais explícita.

```
public static function rules(): array
{
    return [
        'atividade_id' => Validator::intVal()->callback(Atividade::exists(...)),
        'disciplina_id' => Validator::intVal()->callback(Disciplina::exists(...)),
        'conteudo' => Validator::notEmpty()->max(512),
        'data' => Validator::date('Y-m-d'),
    ];
}
```



## Validação: uso

Agora que temos nossas regras, podemos tornar o controller mais coeso.

Ao tentar validar, caso não sejam atendidas as regras, podemos redirecionar o usuário de volta ao formulário.

- `$request->validate([ ... ])`: para validar
- `$request->validated`: para itens válidos

```
public function salvar(Request $request): Response
{
    if (! $request->validate(Agendamento::rules())) {
        return redirect('/agendamentos/cadastrar');
    }

    $agendamento = Agendamento::create($request->validated);

    return redirect('/agendamentos');
}
```

## Cadastrar agendamento

[← Voltar para listagem](#)

Atividade

must be an integer number

Disciplina

must be an integer number

Data

Conteúdo

Conteúdo

The value must not be empty

 Cadastrar

Mensagens de erro da validação serão traduzidas quando o assunto for **i18n**.

# Resposta

---





# Resposta

Agora que temos uma novo processo para gerenciar a requisição, um controle de estado da aplicação (sessão) e um coletor de erros, podemos simplificar o envio da resposta ao usuário.

No futuro, podemos ter outro tipos de respostas, como JsonResponse ou StreamedResponse.

```
function redirect(  
    string $url,  
    int $status = Response::HTTP_FOUND  
) : RedirectResponse {  
    return new RedirectResponse($url, $status);  
}  
  
function response(  
    string $viewName,  
    array $data = [],  
    int $status = Response::HTTP_OK  
) : Response {  
    return new Response(View::render($viewName, $data), $status);  
}
```

# Testes

---



# Guzzle Http

O **Guzzle** é um cliente HTTP PHP que facilita o envio de solicitações HTTP e trivializar para integrar com serviços da Web.

- Interface simples para construir strings de consulta, solicitações POST, streaming de uploads grandes, streaming de grandes downloads, uso de cookies HTTP, upload de dados JSON, etc.
- Pode enviar solicitações síncronas e assíncronas usando a mesma interface.
- Utiliza interfaces PSR-7 para solicitações, respostas e fluxos. Isso permite que você utilize outras bibliotecas compatíveis com PSR-7 com o Guzzle.
- Abstramente o transporte HTTP subjacente, permitindo que você escreva ambiente e transporte de código agnóstico; ou seja, sem dependência rígida de **cURL**, **fluxos PHP**, **sockets** ou **loops de eventos** não bloqueadores.
- O sistema de middleware permite aumentar e compor o comportamento do cliente.

([Guzzle Documentation](#))

# composer require guzzlehttp/guzzle

Instala dependência

---



# Suíte de testes

É necessário é executar os testes na ordem correta, ou seja, primeiro os teste unitários e depois os teste de funcionalidades.

Assim, precisamos atualizar nosso **phpunit.xml**.

```
<testsuites>
  <testsuite name="Unit">
    <directory suffix="Test.php">tests/Unit</directory>
  </testsuite>
  <testsuite name="Feature">
    <directory suffix="Test.php">tests/Feature</directory>
  </testsuite>
</testsuites>
```



## Set-up

Para testar nosso sistema, precisamos criar uma instância independente, isto é, sempre que for executado os testes, o servidor embutido do PHP será executado e finalizado automaticamente ao final do teste.

```
beforeAll(function () {  
    refreshDatabase();  
    runServer();  
});
```

```
afterAll(function () {  
    stopServer();  
});
```



# Testando

Nossos testes se resumem a obter o HTML da página e identificar se nela existem partes que buscamos, algo que veríamos caso o acesso fosse realizado manualmente.

Faremos isso como Regex, mas tenha em mente que este não é a “forma correta”, pois algumas aplicações podem ser *client-side rendering (CSR)*, isto é, um HTML mínimo é enviado ao navegador e por meio de uso do JavaScript, a página é “montada” na tela do navegador. Contudo, para fins didáticos, aplicamos esta técnica.

```
test('não há períodos cadastrados', function () {  
  $response = httpclient()->get('/periodos');  
  $responseContent = $response->getBody()->getContents();  
  
  expect($responseContent)->toMatch('/total: 0 períodos/i');  
  expect($responseContent)->toMatch('/não há períodos cadastrados/i');  
});
```