Curso PHP

Do XLSX ao CMS (aula 19)

URLs

Rotas nomeadas

Rotas nomeadas

Usar rotas nomeadas nos dá garantia de um maior controle e facilita enviar parâmetros à rota de forma mais precisa e fácil.

Atualmente, se queremos substituir *placeholders*, precisamos usar um lógica que envolve array com posicionamento específico.

Além disso, **é preciso reescrever a URI da mesma forma que no roteador**, ou seja, caso atualize o arquivo *routes.php*, é preciso localizar todas as ocorrências para realizar a substituição.

```php
private function register(
    string $method,
    string $uri,
    mixed $action,
    array $middlewares = [],
    string $name = null
): void {
    // ...
}


Router::get(
    name: 'login',
    uri: '/login',
    action: [AuthController::class, 'index'],
    middlewares: [
        Visitante::class,
    ]
);
```

```php
/**
 * Menu auxiliar
 * Fica ao lado do cabeçalho
 */
'AUX' => Menu::new()
    ->if($isAuthenticated, fn (Menu $menu) => $menu
        ->submenu(fn (Menu $submenu) => $submenu
            ->wrap('details', ['class' => 'dropdown', 'dir' => 'rtl'])
            ->prepend("<summary>{$usuario->email}</summary>")
            ->link('/cadastro/editar', 'Editar perfil')
            ->link('/logout', 'Logout')
            ->link(route('editar_cadastro'), 'Editar perfil')
            ->link(route('logout'), 'Logout')
        )
    )
    ->if(! $isAuthenticated, fn (Menu $menu) => $menu
        ->link('/login', 'Login')
        ->link('/cadastro', 'Cadastro')
        ->link(route('login'), 'Login')
        ->link(route('cadastro'), 'Cadastro')
    ),
```

Aplicando o uso do helper **route**

URLs assinadas
Signed URLs

URLs assinadas

Uma *signed URL* inclui informações adicionais, por exemplo, uma data e hora de expiração, que proporcionam a você mais controle sobre o acesso a seu conteúdo.

(AWS)

```php
public static function createSignedUrl(
    string $path,
    array $params = []
): string {
    $params['expires'] ??= strtotime('+5 minutes');

    unset($params['signature'], $params['senha']);
    ksort($params);

    $params['signature'] = hash_hmac(
        'sha256',
        http_build_query($params),
        env('APP_KEY')
    );

    return static::createUrl($path, $params);
}
```

Notificações

Notificações

Notificar alguém ou o próprio sistema de algo é parte importante do processo de desenvolvimento.

No fluxo HTTP (requisição, processamento, resposta), ocorrem notificações internas a todo momento, com invocações e outros processamentos que não temos controle sobre suas ocorrências, mas que estão presentes neste fluxo.

Durante o processamento, podemos inserir outras tarefas no meio da tarefa principal realizada, que enviam uma mensagem ao usuário, como um e-mail ou SMS, isto é, da mesma forma que geramos uma auditoria de nossos *models*, podemos enviar uma mensagem ao usuário dando maiores detalhes sobre o que aconteceu naquele evento.

Notify

Helper para notificar usuário

```php
class CadastroController
{
    public function cadastrar(): Response
    {
        return response('cadastro/cadastrar');
    }

    public function salvar(Request $request): RedirectResponse
    {
        if (! $request->validate(Usuario::rules(), ['nome', 'email', 'senha'])) {
            return redirect(route('cadastro'));
        }

        $usuario = Usuario::create($request->validated);

        notify($usuario, UsuarioCadastrado::class);

        return redirect(route('login'));
    }
}
```

Método *notify*

Notify

De forma rápida, o método extrai o ID do destinatário (*recipient*) e invoca a classe *notifiable*.

É importante lembrar que usamos somente o ID, pois nosso método *resolveCallback* só consegue, até o momento, encontrar um *model* a partir deste valor.

```
function notify(
    int|usuario $recipient,
    string $notifiable
): void {
    if ($recipient instanceof Usuario) {
        $recipient = $recipient->id;
    }

    resolveCallback($notifiable, compact('recipient'));
}
```

Notifiable

Notifiable

A trait *Notifiable* agrupa a lógica para a forma de envio da notificação.

A forma de notificação, também chamada de canal, é o meio por onde determinada notificação será encaminhada.

Observe que os métodos têm visibilidade privada. Isto evita que sejam invocados de fora da classe, o que poderia gerar efeitos colaterais indesejados.

```php
trait Notifiable
{
    private function viaEmail(string $view, array $data): void
    {
        // ...
    }

    private function viaSms(string $view, array $data): void
    {
        // ...
    }
}
```

Definindo *notifiables*

Notifiables

Um *notifiable* é planejado para funcionar de forma independente, isto é, **desacoplado do fluxo HTTP**.

A lógica é que se este *notifiable* foi invocado, ele deverá fazer o que pretende ser feito, não precisa **conhecer o contexto** da invocação ou tantos outros pensamentos que poderíamos ter sobre, contudo, pode receber **dados do contexto** onde foi invocado, tal como *IP* e *user agent* em uma tentativa de login.

Notifiables

- **recipient:** o destinatário
- **subject:** o assunto tratado
- **cta:** a ação pretendida que usuário realize

(RD Station)

```php
class UsuarioCadastrado
{
    use Notifiable;

    public function __invoke(Usuario $recipient): void
    {
        $data = [
            'recipient' => $recipient,
            'subject' => "Seja bem vindo, {$recipient->nome}",
            'cta' => Link::to(route('login'), 'Fazer login'),
        ];

        $this->viaEmail('notifications/email/usuario_cadastrado', $data);
    }
}
```

Enviando emails

composer install symfony/mailer

Instala a dependência

Symfony Mailer

Os componentes *Mailer* e *Mime* do Symfony formam um sistema poderoso para criar e enviar e-mail.

[Tem] [...] suporte para mensagens de várias partes, integração Twig, inlining CSS, anexos de arquivos e muito mais. (Symfony Docs)

Usamos Symfony Mailer, pois mesmo que o PHP possua a função mail nativa, a transmissão por meio dela requer uma configuração feita no arquivo .INI.

Além disso, o nos permite aplicar failover/fallback, possibilitando usar diferentes saídas, igual temos com Guzzle, tudo isso usando o mesmo DSN.

Notify: email

A definição de um canal é o encapsulamento dos métodos usados para a transmissão do mesmo.

```php
private function viaEmail(string $view, array $data): void
{
    $transport = Transport::fromDsn(env('MAILER_DSN'));
    $mailer = new Mailer($transport);
    $message = new Email;
    $body = View::render($view, $data);

    $message->from(env('MAILER_FROM'));
    $message->to($data['recipient']->email);
    $message->subject($data['subject']);
    $message->html($body);

    $mailer->send($message);
}
```

Testando email

Testando email

Ter um servidor de testes de email é algo básico, talvez fundamental durante o desenvolvimento.

Quando se usa testes, igual temos aqui com Pest PHP, podemos realizar uma requisição falsa, permitindo simular o envio e a resposta do email. Contudo, quando se está testando de fato o ambiente, ou seja, a interface e os comportamentos que o usuário irá enxergar, é necessário receber este email em algum lugar. Neste cenário, entra em cena o serviço de *email testing* como o Mailtrap.

Mailtrap

O teste de cliente por email avalia como um e-mail renderiza e funciona em vários clientes de e-mail, dispositivos e sistemas operacionais.

O principal objetivo é garantir que o e-mail seja exibido corretamente e de forma consistente para todos os destinatários, independentemente da plataforma que estão usando.

(Mailtrap)

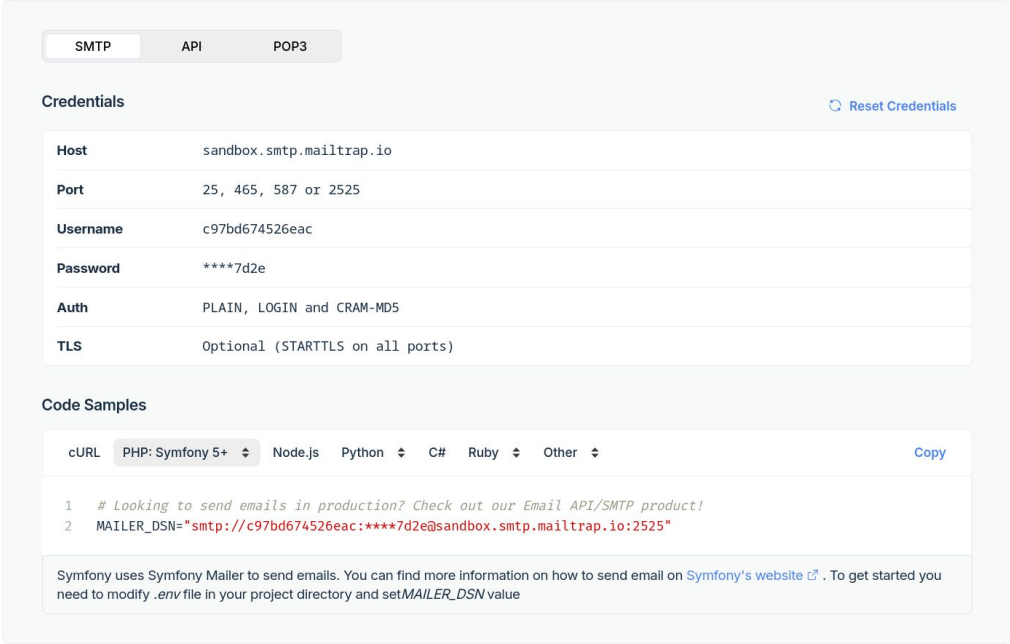
Leia mais em: mailtrap.io/blog/how-to-test-email

Mailtrap e Symfony Mailer

Uma vez que temos o componente Mailer como dependência de nosso projeto, precisamos obter um DSN.

Para tal, precisamos acessar o serviço mailtrap.io e requisitar um.

Leia mais em: mailtrap.io/blog/send-emails-in-symfony



Notificando

Quando saber quando enviar uma notificação para o usuário?

- Cadastro
- Login
- Senha
  - redefinir
  - restaurar
- Agendamento
  - cadastrado
  - alterado
- Agenda do dia

Etc...

Processamento paralelo
Jobs

Cron

O cron nada mais é do que o serviço responsável por agendar tarefas no Linux, o cron é tão estabelecido que hoje ele vem instalado na maioria das distribuições. **Com o cron é possível agendar tarefas para serem executadas periodicamente ou apenas uma vez**.

A tarefas a serem executadas pelo cron ficam dentro de uma tabela a crontab é nela que o desenvolvedor deve cadastrar as execuções desejadas.

(PHP.com.br)

Crontab: listagem e edição

Para listar tarefas, usamos o comando:

- **crontab -l**

Para editar tarefas, usamos o comando:

- **crontab -e**

Na primeira execução do comando, será perguntado o editor padrão. Escolha o de sua preferência.

Crontab: registro

```
1.  .---------------- Minuto (0-59)
2.  | .-------------- Hora (0-23)
3.  | | .------------ Dia do mês (1-31)
4.  | | | .---------- Mês (1-12)
5.  | | | | .-------- Dia da semana (0-6) (Os dias 0 e 7 se referem ao domingo)
6.  | | | | |
7.  | | | | |   .---- Comando para ser executado
8.  | | | | |   |
9.  * * * * * echo "Olá mundo" > /tmp/t.txt
```

Crontab: execução

Fila (a cada minuto):

- * * * * * /<PROJETO>/bin/execute-jobs >> /<PROJETO>/logs/crontab.log 2>&1

Agenda diária (todo dia às 6h da manhã):

- 0 6 * * * /<PROJETO>/bin/daily-schedule >> /<PROJETO>/logs/crontab.log 2>&1

Minuto a minuto

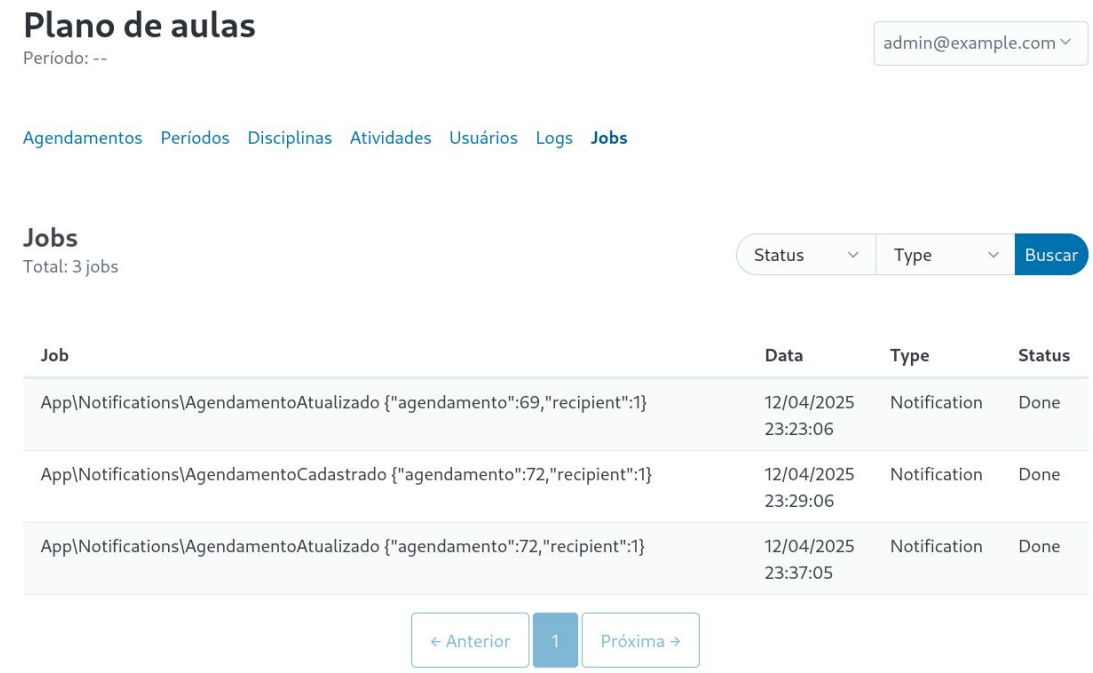
As tarefas no Crontab é executada a cada minuto, isto é, cada vez que o indicado de segundos zera (00), a tarefa é executada. Assim, para executar tarefas a cada 30 segundos, por exemplo, é uma missão mais complicada e que não se deve ser realizada com tarefas cron.

Explorando o utilitário

O utilitário Crontab é extenso e cheio de particularidades, contudo, não é nada complexo compreendê-lo. Caso queira explorar um pouco mais sobre esta ferramenta, eis um bom artigo:

- Crontab Logs: Track, Debug, and Optimize Your Cron Jobs

Listando jobs



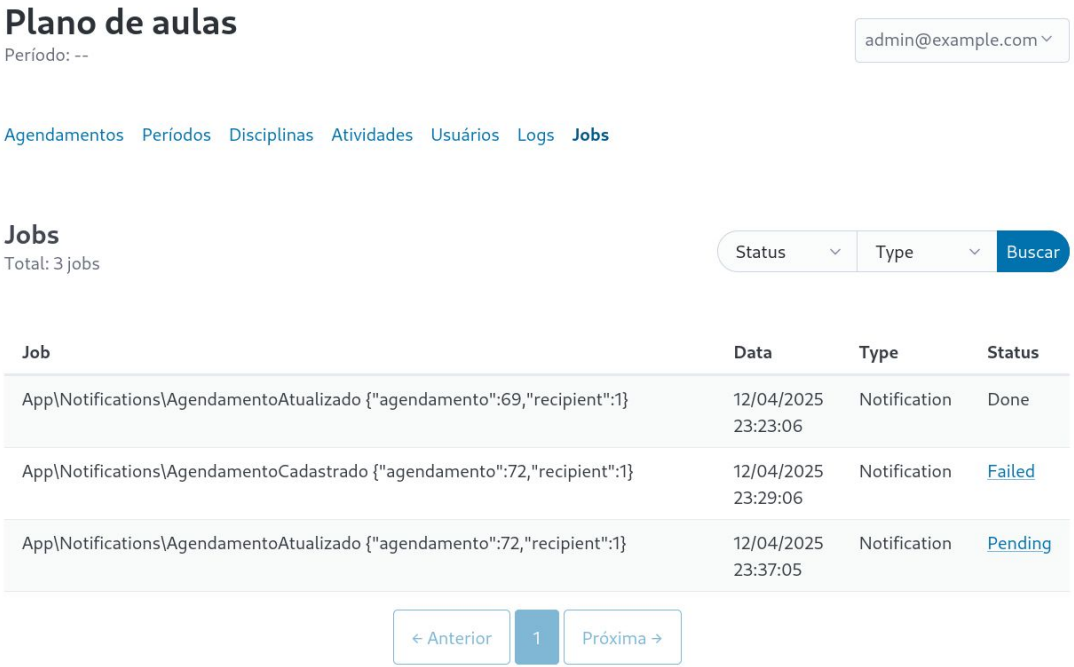
Listagem de jobs

Jobs: execução manual

Jobs: execução manual

Em alguns momentos, é necessário executar um job manualmente, seja por falha ou por necessidade imediata da execução.

Como há um controle da execução, isto é, um job executado não pode ser executado novamente, há uma garantia de não duplicar a execução do mesmo.



Jobs não concluídos permitem a execução manual

Próximos passos

Agora é hora de focar na parte de criação de conteúdo.

A próxima etapa será elaborar nosso próprio editor de texto, com tabulação e formatação e com suporte a upload de arquivos diversos, reprodutor de áudio e vídeo, galeria de imagens, interpretador de link externos...