Curso PHP

Do XLSX ao CMS (aula 18)

Coletor de erros

Coletor de erros

Na aula 8, quando abordamos o tópico de depuração, fizemos uma introdução ao pacote **filp/whoops**.

Para relembrar: **filp/whoops** é uma biblioteca PHP que fornece um manipulador de erros mais amigável e informativo do que o padrão do PHP. Ele é especialmente útil durante o desenvolvimento, pois exibe detalhes detalhados sobre erros e exceções, facilitando a depuração.

Contudo, no nosso ambiente de produção, não podemos exibir os detalhes técnicos do sistema ao usuário comum, pois este pode está mal-intencionado e se aproveitar do erro para manipular o sistema indevidamente. Para isto, ao invés de mostrar os detalhes do erro, precisamos induzir o sistema a mostrar uma página de erro genérica, sem muitos detalhes do que aconteceu, mas que forneça um status HTTP correto, isto é, “erro 404”, “erro 500” e afins.

```php
tap(new Run, function (Run $runner) {
    $handler = env('APP_DEBUG') ? new PrettyPageHandler : function (Throwable $exception) {
        // define a resposta apropriada
        $response = match (true) {
            $exception instanceof ModelNotFoundException => responseError(Response::HTTP_NOT_FOUND),
            default => responseError(Response::HTTP_INTERNAL_SERVER_ERROR)
        };

        // enviar resposta
        $response->send();
    };

    $runner->prependHandler($handler);
    $runner->register();
});
```

Preparando Whoops para mostrar uma página de erro ao encontrar um *exception*

Página de erros genérica

Auditoria

Auditoria

A auditoria de modelos é uma prática essencial para rastrear e registrar as alterações feitas nos dados do seu aplicativo. Ela permite que você mantenha um **histórico completo de quem fez, o que fez e quando fez**, fornecendo insights valiosos para fins de segurança, conformidade e análise.

Auditable

A *trait Auditable* é uma ferramenta de programação que encapsula a lógica de auditoria para modelos Eloquent em um projeto PHP.

O objetivo principal dessa trait é simplificar o processo de rastreamento e registro das alterações feitas nos dados de um modelo, promovendo a reutilização de código e a padronização das práticas de auditoria.

```php
trait Auditable
{
    public function audits(): MorphMany
    {
        return $this->morphMany(Audit::class, 'auditable');
    }

    public static function bootAuditable()
    {
        static::created(function (Model $model) {
            Audit::create([ /* ... */ ]);
        });

        static::updated(function (Model $model) {
            Audit::create([ /* ... */ ]);
        });

        static::deleted(function (Model $model) {
            Audit::create([ /* ... */ ]);
        });
    }
}
```

Implementação

Polimorfismo

Relacionamento polimórfico

Relacionamentos polimórficos no Eloquent são uma ferramenta poderosa para criar associações flexíveis entre modelos. Eles permitem que um modelo pertença a vários outros modelos em uma única associação, sem a necessidade de criar tabelas de junção separadas para cada relacionamento.

Em um relacionamento polimórfico, um modelo "pai" pode se relacionar com vários modelos "filhos" diferentes. Isso é alcançado usando duas colunas na tabela do modelo "filho":

- **`auditable_id`**: armazena o ID do modelo "pai" relacionado.
- **`auditable_type`**: armazena o nome da classe do modelo "pai" relacionado.

```php
DB::schema()->create('audits', function (Blueprint $table) {
    $table->id();
    $table->morphs('auditable');
    $table->json('old_values')->nullable();
    $table->json('new_values')->nullable();
    $table->string('action');
    $table->timestamps();
});
```

Tabela *audits*

Observando eventos

composer require illuminate/events

Adiciona dependência

Illuminate Events

Illuminate/events é um componente que fornece um sistema de despacho e escuta de eventos.

Ele permite que você execute código em resposta a eventos que ocorrem em sua aplicação, promovendo a separação de preocupações e a extensibilidade.

- Gerar logs de auditoria quando um modelo é atualizado.
- Enviar emails de notificação quando um usuário é registrado.
- Invalidar caches quando dados são alterados.
- Processar pagamentos em segundo plano quando um pedido é feito.

Logs

Logs

Logs são registros de eventos que ocorrem em um sistema, aplicação ou dispositivo.

Eles fornecem um histórico detalhado das atividades, erros e outros eventos relevantes, permitindo que desenvolvedores, administradores de sistemas e outros profissionais:

- Monitorem
- Façam diagnósticos de problemas
- Analisem o desempenho

composer require monolog/monolog

Adiciona dependência

Monolog

O Monolog é uma biblioteca de registro (logging) para PHP que oferece uma maneira flexível e poderosa de registrar eventos em sua aplicação. Ele permite que você envie seus logs para diversos destinos, como arquivos, bancos de dados, e-mails, serviços web e muito mais.

O Monolog segue a especificação PSR-3, garantindo a compatibilidade com outras bibliotecas e frameworks.

(Monolog docs)

```php
function logAppend(Throwable $exception, Level $level = Level::Error): void
{
    try {
        $config = require PROJECT_ROOT.'/config/log.php';

        // ...

        $logger->log($level, $exception->getMessage(), [
            'key' => sha1(random_bytes(16)),
            'file' => $exception->getFile(),
            'line' => $exception->getLine(),
            'trace' => $exception->getTraceAsString(),
            'request' => [
                'method' => Request::getMethod(),
                'uri' => Request::getPathInfo(),
            ],
        ]);
    } catch (Exception $e) {
        error_log("Falha ao registrar log: ".$e->getMessage());
    }
}
```

logAppend: para adicionar nova linha ao log

```php
function logReader($limit, ?string $level_name = null)
{
    $config = require PROJECT_ROOT.'/config/log.php';
    $filename = $config['filename'];

    // ...

    for ($i = $seekLine; $i < $totalLines; $i++) {
        // ...

        if ($level_name === null || $level_name === $decoded['level_name']) {
            $lines[$i] = $decoded;
        }
    }

    return array_reverse($lines);
}
```

logReader: para ler log

```php
function logRemoveLine(string $key): bool
{
    $config = require PROJECT_ROOT.'/config/log.php';

    // ...

    while (! $inputFile->eof()) {
        // ...

        if (str_contains($line, "\"key\":\"{$key}\"")) {
            $removed = true;

            continue;
        }

        $outputFile->fwrite($line.PHP_EOL);
    }

    // ...

    return true;
}
```

logRemoveLine: para remover linha específica do log

Próximos passos

Enviando e-mail com PHP