Errores & Registros

Introducción

When you start a new Laravel project, error and exception handling is already configured for you. In addition, Laravel is integrated with the Monolog logging library, which provides support for a variety of powerful log handlers.

Configuración

Errores detallados

The amount of error detail your application displays through the browser is controlled by the `debug` configuration option in your `config/app.php` configuration file. De forma predeterminada, esta opción de configuración se establece respecto la variable de entorno `APP_DEBUG`, que se almacena en el archivo `.env`.

For local development, you should set the `APP_DEBUG` environment variable to `true`. In your production environment, this value should always be `false`.

Log Modes

Out of the box, Laravel supports `single`, `daily`, `syslog` and `errorlog` logging modes. For example, if you wish to use daily log files instead of a single file, you should simply set the `log` value in your `config/app.php` configuration file:

```
'log' => 'daily'
```

Custom Monolog Configuration

If you would like to have complete control over how Monolog is configured for your application, you may use the application's `configureMonologUsing` method. You should place a call to this method in your `bootstrap/app.php` file right before the `$app` variable is returned by the file:

```
$app->configureMonologUsing(function($monolog) {
```

```
    $monolog->pushHandler(...);
});

return $app;
```

The Exception Handler

All exceptions are handled by the `App\Exceptions\Handler` class. This class contains two methods: `report` and `render`. We'll examine each of these methods in detail.

The Report Method

El método `report` se usa para hacer log de las excepciones o enviarlas a un servicio externo como BugSnag. By default, the `report` method simply passes the exception to the base class where the exception is logged. Sin embargo, eres libre de hacer log de las excepciones que desees.

For example, if you need to report different types of exceptions in different ways, you may use the PHP `instanceof` comparison operator:

```
/**
 * Informa o log de una excepción.
 *
 * Este es un buen lugar para enviar las excepciones a  Sentry, Bugsnag,
etc.
 *
 * @param  \Exception  $e
 * @return void
 */
public function report(Exception $e)
{
    if ($e instanceof CustomException) {
        //
    }

    return parent::report($e);
}
```

Ignoring Exceptions By Type

The `$dontReport` property of the exception handler contains an array of exception types that will not be logged. De forma predeterminada, las excepciones resultantes de errores 404 no se escriben en los archivos de log. Puedes agregar otros tipos de excepción a este array según sea necesario.

The Render Method

The `render` method is responsible for converting a given exception into an HTTP response that should be sent back to the browser. Por defecto, la excepción se pasa a la clase base

que genera una respuesta. However, you are free to check the exception type or return your own custom response:

```
/**
 * Render an exception into an HTTP response.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \Exception  $e
 * @return \Illuminate\Http\Response
 */
public function render($request, Exception $e)
{
    if ($e instanceof CustomException) {
        return response()->view('errors.custom', [], 500);
    }

    return parent::render($request, $e);
}
```

Excepciones HTTP

Algunas excepciones describen los códigos de error HTTP del servidor. Por ejemplo, podría ser un error de "página no encontrada" (404), un "error no autorizado" (401) o incluso un error 500 generado. In order to generate such a response from anywhere in your application, use the following:

```
abort(404);
```

The `abort` method will immediately raise an exception which will be rendered by the exception handler. Optionally, you may provide the response text:

```
abort(403, 'Acción no autorizada.');
```

Este método puede utilizarse en cualquier momento durante el ciclo de vida de la solicitud.

Custom HTTP Error Pages

Laravel makes it easy to return custom error pages for various HTTP status codes. For example, if you wish to customize the error page for 404 HTTP status codes, create a `resources/views/errors/404.blade.php`. This file will be served on all 404 errors generated by your application.

The views within this directory should be named to match the HTTP status code they correspond to.

Logging

El logger de Laravel provee una capa simple sobre la poderosa librería Monolog. De forma predeterminada, Laravel está configurado para crear diariamente archivos de log para la aplicación que se almacenan en el directorio `storage/logs`. You may write information to the logs using the `Log`facade:

```
<?php

namespace App\Http\Controllers;

use Log;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function showProfile($id)
    {
        Log::info('Showing user profile for user: '.$id);

        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

El logger provee siete niveles de registros definidos en el RFC 5424: **debug**, **info**, **notice**, **warning**, **error**, **critical**, y **alert**.

```
Log::debug($error);
Log::info($error);
Log::notice($error);
Log::warning($error);
Log::error($error);
Log::critical($error);
Log::alert($error);
```

Contextual Information

An array of contextual data may also be passed to the log methods. This contextual data will be formatted and displayed with the log message:

```
Log::info('User failed to login.', ['id' => $user->id]);
```

Accessing The Underlying Monolog Instance

Monolog cuenta con una variedad de controladores adicionales que puedes utilizar para hacer registros. Si es necesario, puedes acceder a la instancia de Monolog oculta que Laravel utiliza:

```
$monolog = Log::getMonolog();
```