Laravel 5.2 ☰

Configuration

- # **Introduction**
- # **Accessing Configuration Values**
- # **Environment Configuration**
  - # Determining The Current Environment
- # **Configuration Caching**
- # **Maintenance Mode**

Introduction

All of the configuration files for the Laravel framework are stored in the `config` directory. Each option is documented, so feel free to look through the files and get familiar with the options available to you.

Accessing Configuration Values

You may easily access your configuration values using the global `config` helper function from anywhere in your application. The configuration values may be accessed using "dot" syntax, which includes the name of the file and option you wish to access. A default value may also be specified and will be returned if the configuration option does not exist:

```
$value = config('app.timezone');
```

To set configuration values at runtime, pass an array to the `config` helper:

```
config(['app.timezone' => 'America/Chicago']);
```

# Environment Configuration

It is often helpful to have different configuration values based on the environment the application is running in. For example, you may wish to use a different cache driver locally than you do on your production server. It's easy using environment based configuration.

To make this a cinch, Laravel utilizes the DotEnv PHP library by Vance Lucas. In a fresh Laravel installation, the root directory of your application will contain a `.env.example` file. If you install Laravel via Composer, this file will automatically be renamed to `.env`. Otherwise, you should rename the file manually.

All of the variables listed in this file will be loaded into the `$_ENV` PHP super-global when your application receives a request. However, you may use the `env` helper to retrieve values from these variables in your configuration files. In fact, if you review the Laravel configuration files, you will notice several of the options already using this helper:

```
'debug' => env('APP_DEBUG', false),
```

The second value passed to the `env` function is the "default value". This value will be used if no environment variable exists for the given key.

Your `.env` file should not be committed to your application's source control, since each developer / server using your application could require a different environment configuration.

If you are developing with a team, you may wish to continue including a `.env.example` file with your application. By putting place-holder values in the example configuration file, other developers on your team can clearly see which environment variables are needed to run your application.

Determining The Current Environment

The current application environment is determined via the `APP_ENV` variable from your `.env` file. You may access this value via the `environment` method on the `App` facade:

```
$environment = App::environment();
```

You may also pass arguments to the `environment` method to check if the environment matches a given value. If necessary, you may even pass multiple values to the `environment` method. If the environment matches any of the given values, the method will return `true`:

```
if (App::environment('local')) {
    // The environment is local
}

if (App::environment('local', 'staging')) {
    // The environment is either local OR staging...
}
```

An application instance may also be accessed via the `app` helper method:

```
$environment = app()->environment();
```

# Configuration Caching

To give your application a speed boost, you should cache all of your configuration files into a single file using the `config:cache` Artisan command. This will combine all of the configuration options for your application into a single file which will be loaded quickly by the framework.

You should typically run the `php artisan config:cache` command as part of your production deployment routine. The command should not be run during local development as configuration options will frequently need to be changed during the course of your application's development.

# Maintenance Mode

When your application is in maintenance mode, a custom view will be displayed for all requests into your application. This makes it easy to "disable" your application while it is updating or when you are performing maintenance. A maintenance mode check is included in the default middleware stack for your application. If the application is in maintenance mode, an `HttpException` will be thrown with a status code of 503.

To enable maintenance mode, simply execute the `down` Artisan command:

```
php artisan down
```

To disable maintenance mode, use the `up` command:

```
php artisan up
```

Maintenance Mode Response Template

The default template for maintenance mode responses is located in `resources/views/errors/503.blade.php`. You are free to modify this view as needed for your application.

Maintenance Mode & Queues

While your application is in maintenance mode, no queued jobs will be handled. The jobs will continue to be handled as normal once the application is out of maintenance mode.

Alternatives To Maintenance Mode

Since maintenance mode requires your application to have several seconds of downtime, you may consider alternatives like Envoyer to accomplish zero-downtime deployment with Laravel.