

Laravel

SEARCH

Documentation

Laracasts

N

Prologue

Release Notes

Upgrade Guide

Contribution Guide

API Documentation

Getting Started

Installation

Configuration

Directory Structure

Request Lifecycle

Dev Environments

Homestead

Valet

Core Concepts

Service Container

Service Providers

Facades

Contracts

The HTTP Layer

Routing

Middleware

CSRF Protection

Controllers

Requests

Responses

Views

Session

Validation

Frontend

Blade Templates

Localization

Frontend Scaffolding

Compiling Assets

Security

Authentication

API Authentication

Authorization

Encryption

Hashing

Password Reset

Installation

Installation

Server Requirements

Installing Laravel

Configuration

Web Server Configuration

Pretty URLs

Installation

Server Requirements

The Laravel framework has a few system requirements that must be satisfied by the [Laravel Homestead](#) virtual machine or Homestead as your local Laravel development environment.

However, if you are not using Homestead, you must satisfy the following requirements:

- PHP >= 5.6.4
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Installing Laravel

Laravel utilizes [Composer](#) to manage its dependencies. You must have Composer installed on your machine.

Via Laravel Installer

First, download the Laravel installer using Composer:

```
composer global require "laravel/installer"
```

INTRODUCCIÓN A LARAVEL

DISEÑO DE SISTEMAS SOFTWARE

Contenido

1. Introducción
2. Instalación
3. Estructura de un proyecto
4. Artisan
5. Logging
6. Pruebas automatizadas

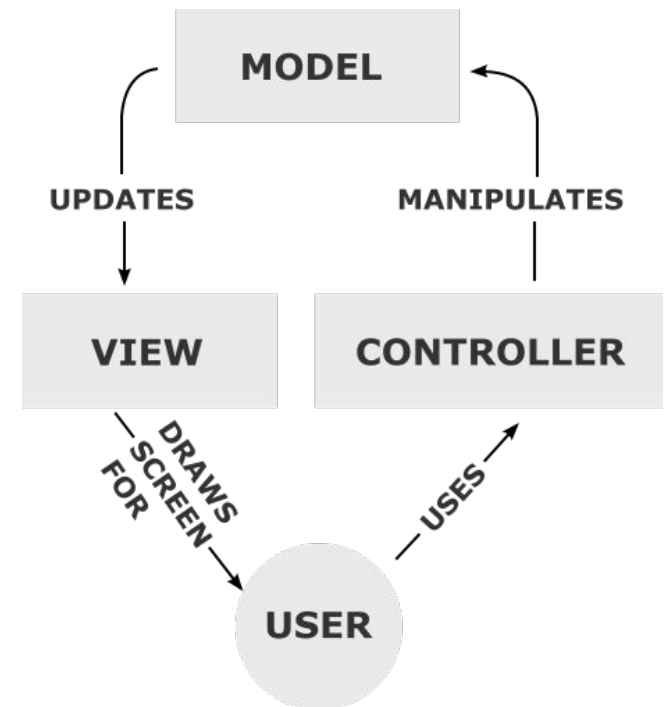
Introducción

- Laravel es un framework de código abierto para el desarrollo de aplicaciones web en PHP ≥ 7.3 que posee una sintaxis simple y elegante
- Características principales:
 - Creado en 2011 por Taylor Otwell
 - Inspirado en *Ruby on rails* y *Symfony*, de quien posee dependencias
 - Está diseñado para utilizar el patrón MVC
 - Integra un sistema de mapeado de datos objeto-relacional llamado *Eloquent ORM*
 - Utiliza un sistema de plantillas llamado *Blade*, el cual hace uso de la cache para darle mayor velocidad

Modelo - Vista - Controlador

MVC es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario

- **Modelo:** Es la representación de la información, gestiona todos los accesos a la información
- **Controlador:** Recibe las peticiones del usuario, realiza peticiones al modelo cuando es necesario y decide qué vistas se deben mostrar en cada caso
- **Vista:** Se encarga de representar los datos de forma visual



Uso del framework

- Laravel proporciona una estructura de proyecto preconfigurada con un comportamiento predefinido
- Para que nuestro código funcione tiene que estar ubicado en las carpetas adecuadas y respetar las convenciones definidas por el framework
- La etiqueta de cierre `?>` DEBE omitirse en los archivos que contengan solamente código PHP, siguiendo el estándar PSR-2 (<http://www.php-fig.org/psr/psr-2/>)
- Laravel proporciona también una serie de utilidades para tareas como *logging* o la automatización de pruebas, iremos descubriéndolas a medida que sea necesario

Introducción a Laravel

INSTALACIÓN

Instalación de Laravel

- Requisitos e instrucciones de instalación:
<https://laravel.com/docs/8.x>
- Todos los requisitos están instalados en los laboratorios, para instalarlo en ordenadores personales ver el documento **Software necesario para las prácticas en Moodle**

Creación de un proyecto

- La forma más sencilla de crear un proyecto Laravel es usando Composer

```
$ composer create-project laravel/laravel=8.6.* miproyecto --prefer-dist
```

- Esto creará la carpeta `miproyecto` con todo el contenido del *framework* Laravel preparado

Ejecución de un proyecto

- Para comprobar que todo funciona correctamente entra en la carpeta `miproyecto` y ejecuta:

```
$ php artisan serve
```

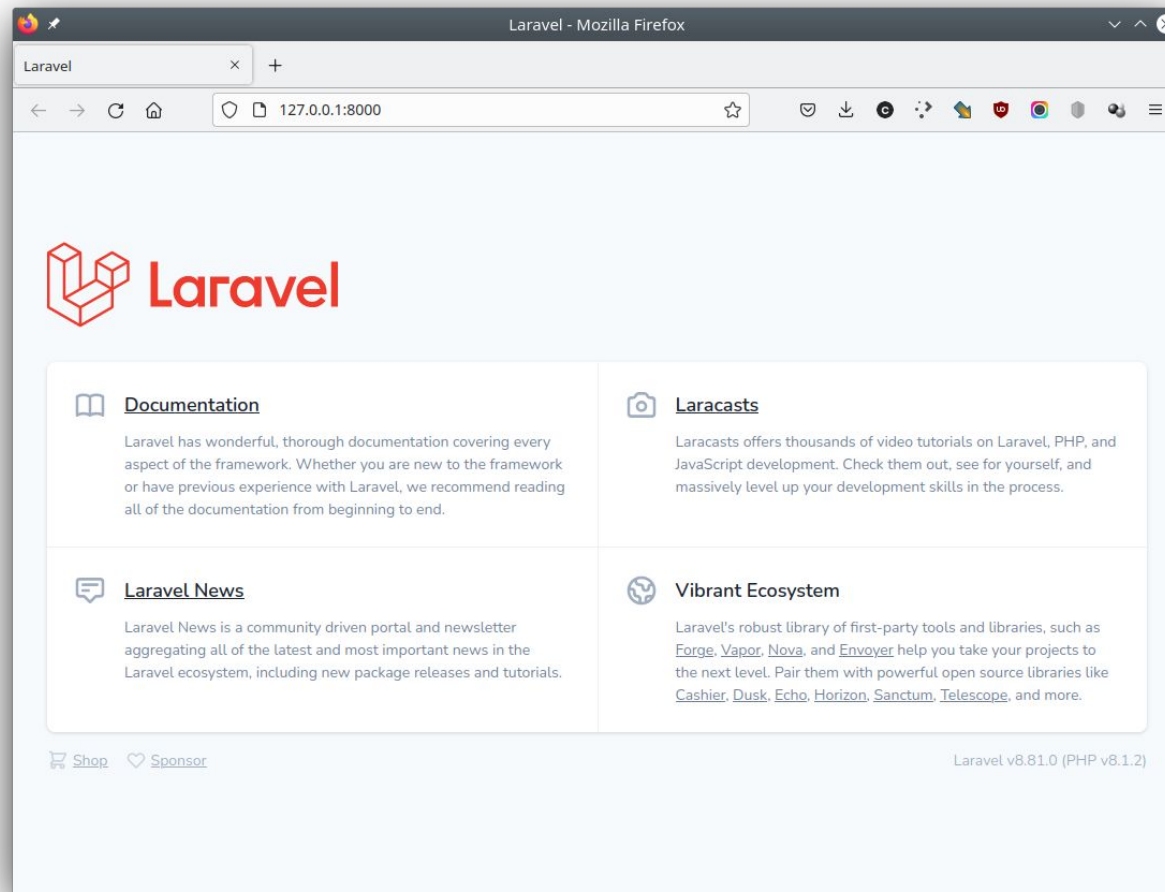
Si todo ha ido bien obtendrás la siguiente salida:

```
Laravel development server started: <http://127.0.0.1:8000>
```

- El servidor web está en marcha esperando peticiones en el puerto 8000

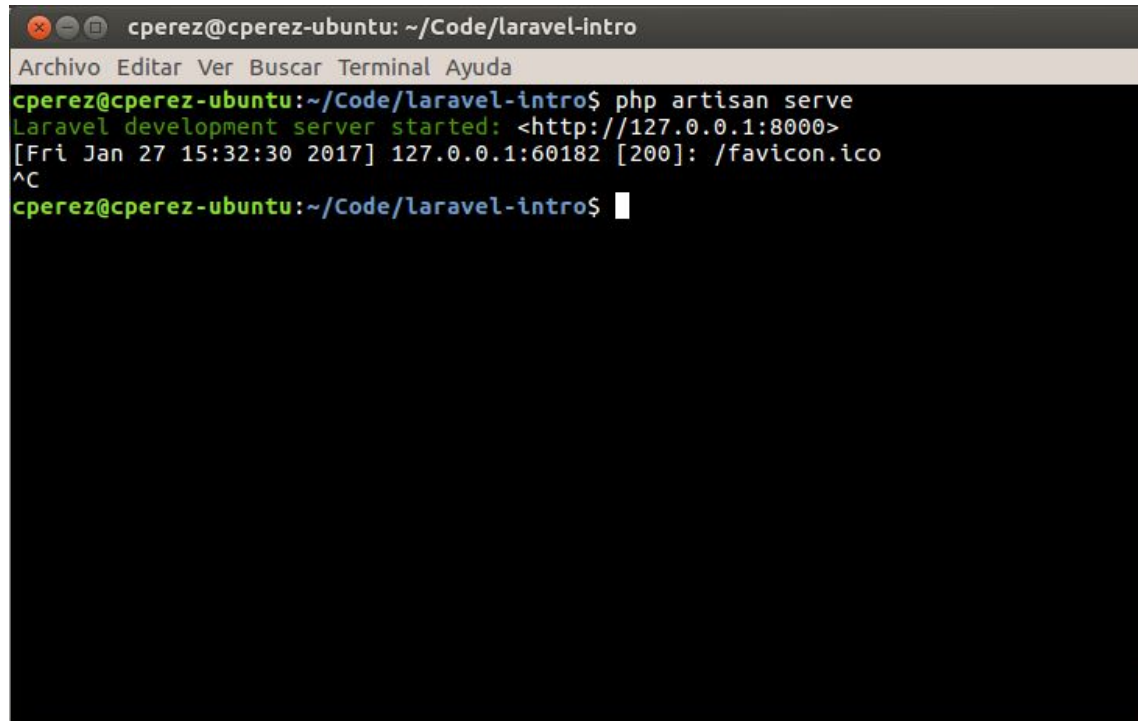
Prueba del proyecto

- Abre un navegador y escribe la dirección **http://localhost:8000**



Detener el proyecto

- Para detener la ejecución del proyecto hay que terminar el proceso que ejecuta el servidor web
- Ve al terminal donde se está ejecutando y presiona `Ctrl+C`



```
cperez@cperez-ubuntu: ~/Code/laravel-intro
Archivo Editar Ver Buscar Terminal Ayuda
cperez@cperez-ubuntu:~/Code/laravel-intro$ php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
[Fri Jan 27 15:32:30 2017] 127.0.0.1:60182 [200]: /favicon.ico
^C
cperez@cperez-ubuntu:~/Code/laravel-intro$
```

Introducción a Laravel

ESTRUCTURA DE UN PROYECTO

Estructura de un proyecto

app/	→ Código de la aplicación
config/	→ Configuración de toda la aplicación
database/	→ Configuración / inicialización de la BBDD
public/	→ Carpeta pública del sitio (con los assets)
resources/	→ Recursos de la aplicación
lang/	→ Traducciones
views/	→ Vistas
routes/	→ Rutas de la aplicación
storage/	→ Caché y temporales
tests/	→ Pruebas automatizadas
vendor/	→ Librerías y dependencias del framework
.env	→ Fichero con la configuración de entorno
artisan	→ CLI de Laravel
composer.json	→ Configuración de Composer

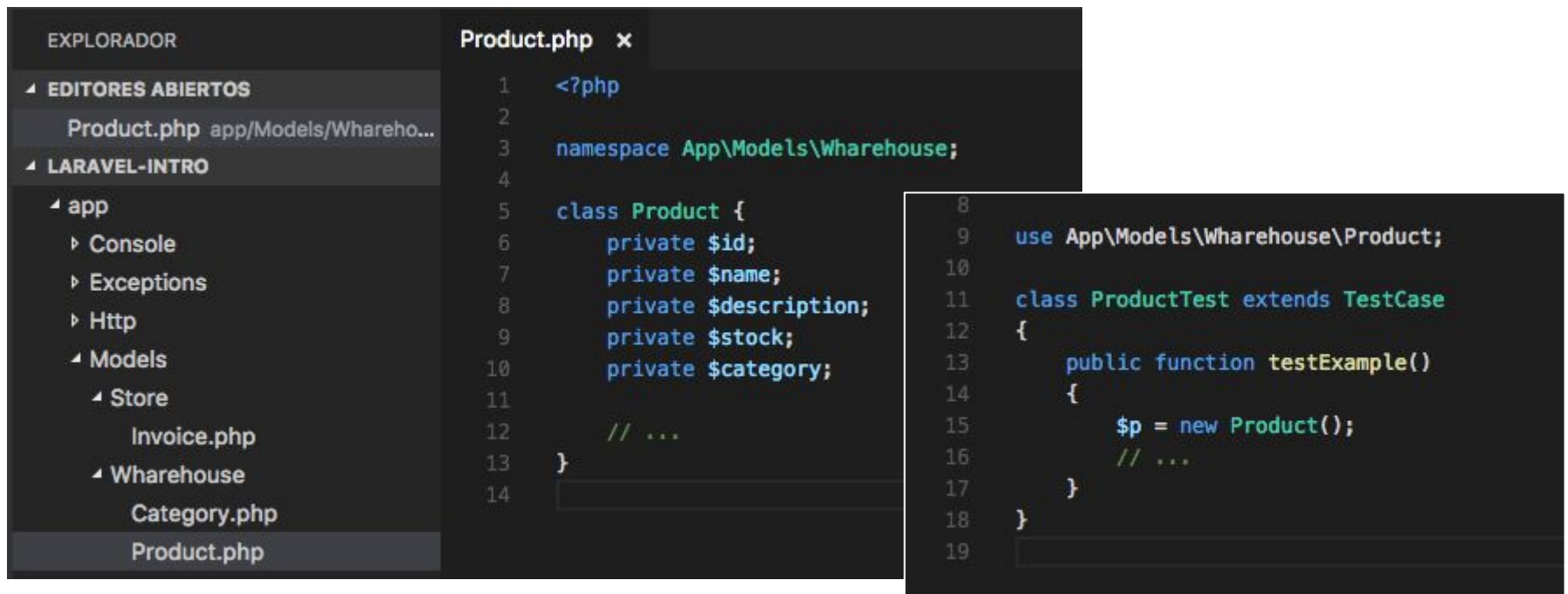
<https://laravel.com/docs/8.x/structure>

Estructura de un proyecto

- La carpeta `app/` contiene el código principal del proyecto: controladores, filtros y modelos de datos
- Los modelos de datos se guardan en la carpeta `app/Models/`, aunque se pueden organizar en subcarpetas
- Por defecto se incluye el modelo `User.php`

Estructura de un proyecto

- Las clases se cargan automáticamente si el nombre de la clase coincide con el del archivo `.php` y el *namespace* coincide con la estructura de carpetas, no es necesario usar `require()`

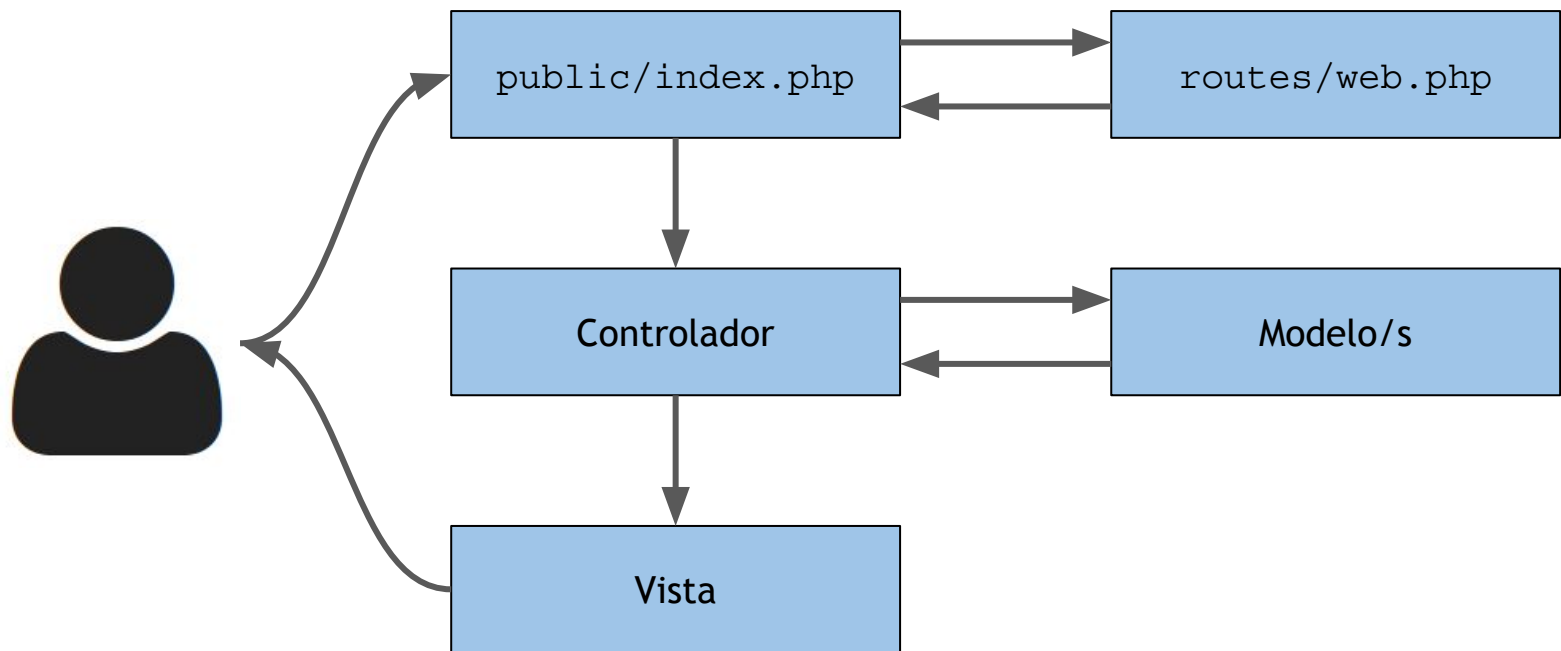


```
EXPLORADOR
└─ EDITORES ABIERTOS
    └─ Product.php app/Models/Whareho...
└─ LARAVEL-INTRO
    └─ app
        ├── Console
        ├── Exceptions
        ├── Http
        └─ Models
            ├── Store
            │   ├── Invoice.php
            └─ Warehouse
                ├── Category.php
                └─ Product.php

Product.php x
1  <?php
2
3  namespace App\Models\Wharehouse;
4
5  class Product {
6      private $id;
7      private $name;
8      private $description;
9      private $stock;
10     private $category;
11
12     // ...
13 }
14

8
9  use App\Models\Wharehouse\Product;
10
11  class ProductTest extends TestCase
12  {
13      public function testExample()
14      {
15          $p = new Product();
16          // ...
17      }
18  }
19
```

Funcionamiento básico



Introducción a Laravel

ARTISAN

Artisan

- Es el interfaz de línea de comandos (CLI) de Laravel
- Permite realizar múltiples tareas necesarias durante el proceso de desarrollo o despliegue de una aplicación
- Para ver una lista de todas las opciones de Artisan ejecuta el siguiente comando:

```
$ php artisan  
# O también:  
$ php artisan list
```

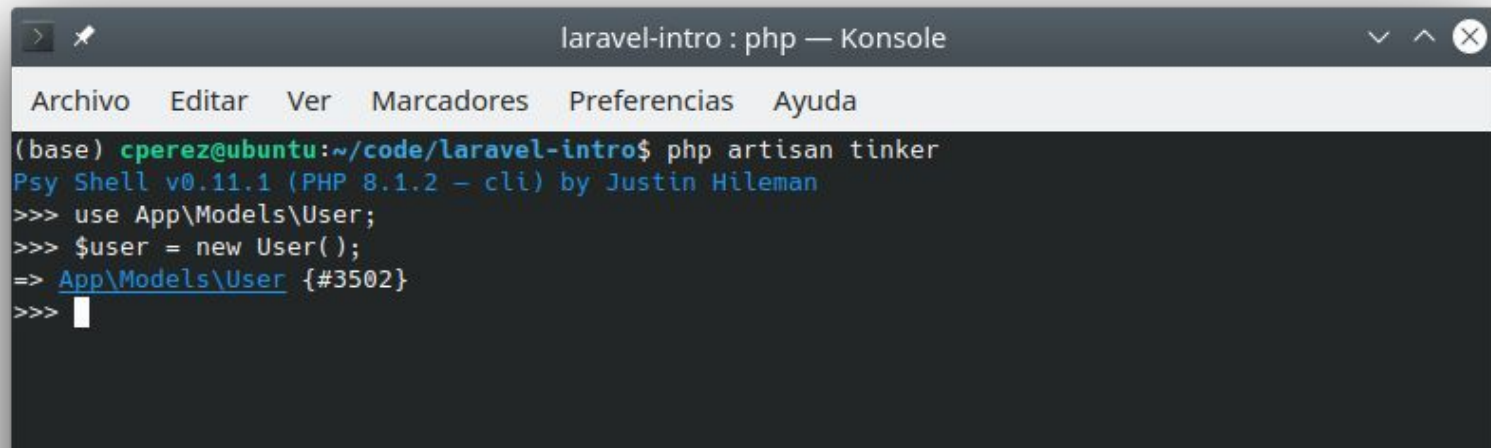
- Para ver más ayuda de una opción ejecuta:

```
$ php artisan help <opción>
```

Probando código con Tinker

- Para probar código sin tener que ejecutar el servidor web puedes usar Tinker, una herramienta para interactuar con el código desde la línea de comandos:

```
$ php artisan tinker
```



The screenshot shows a terminal window titled "laravel-intro : php — Konsole". The terminal displays the command `$ php artisan tinker` being executed. The output shows the Tinker shell environment: `(base) cperez@ubuntu:~/code/laravel-intro$ php artisan tinker`, followed by `Psy Shell v0.11.1 (PHP 8.1.2 - cli) by Justin Hileman`. The prompt `>>>` is shown, followed by the command `use App\Models\User;`, then `$user = new User();`, and the output `=> App\Models\User {#3502}`. The prompt `>>>` is shown again, followed by a cursor.

Probando código con Tinker

- Por el momento no es posible volver a cargar el código modificado en Tinker
- Si modificas el código de tus archivos tendrás que salir de Tinker y volver a entrar para que admita las modificaciones

Introducción a Laravel

PRUEBAS AUTOMATIZADAS

Pruebas automatizadas

- Laravel permite ejecutar pruebas automatizadas con PHPUnit
<https://laravel.com/docs/8.x/testing>
- Los scripts con las pruebas se organizan en dos carpetas dentro de la carpeta `tests`
 - Unit: pruebas unitarias (métodos)
 - Feature: interacciones entre objetos
- Se pueden generar scripts para pruebas con artisan

```
# Crea un test en la carpeta Feature
$ php artisan make:test UserTest
# Crea un test en la carpeta Unit
$ php artisan make:test UserTest --unit
```

Pruebas automatizadas

- Las pruebas se basan en el uso de aserciones
<https://phpunit.readthedocs.io/en/9.5/assertions.html>

```
class MathTest extends TestCase
{
    public function testExample()
    {
        $this->assertEquals(Math::sum(2, 2), 4);
    }
}
```

Convenciones

- Las pruebas para la clase `MiClase` van en la clase `MiClaseTest`
- La clase `MiClaseTest` va en el archivo `MiClaseTest.php`
- Las clases de pruebas extienden la clase `Tests\TestCase`
- Los métodos para las pruebas son públicos y se llaman `testDescripcionDeLaPrueba()`

Ejecución de pruebas

- PHPUnit viene incluido en las dependencias de Laravel, puedes ejecutarlo con el siguiente comando dentro de la carpeta del proyecto

```
$ vendor/bin/phpunit
```

Pruebas automatizadas

Math.php

```
1 <?php
2
3 namespace App;
4
5 class Math {
6     public static function sum($a, $b) {
7         return $a + $b;
8     }
9 }
10
```

MathTest.php

```
1 <?php
2
3 namespace Tests\Unit;
4
5 use Tests\TestCase;
6 use Illuminate\Foundation\Testing\DatabaseMigrations;
7 use Illuminate\Foundation\Testing\DatabaseTransactions;
8
9 use App\Math;
10
11 class MathTest extends TestCase
12 {
13
14     public function testExample()
15     {
16         $this->assertEquals(Math::sum(2, 2), 4);
17     }
18 }
19
```

```
cperez@cperez-ubuntu: ~/Code/laravel-intro
Archivo  Editor  Ver  Buscar  Terminal  Ayuda

cperez@cperez-ubuntu:~/Code/laravel-intro$ ./phpunit
PHPUnit 5.7.8 by Sebastian Bergmann and contributors.

...

Time: 1.28 seconds, Memory: 18.00MB

OK (3 tests, 3 assertions)
cperez@cperez-ubuntu:~/Code/laravel-intro$
```

Introducción a Laravel

LOGGING

Logging

- Laravel lleva integrada la librería Monolog para la gestión de archivos de registro (logs)
<https://laravel.com/docs/8.x/logging>
- Usando Monolog puedes imprimir mensajes desde el código que se almacenarán en el archivo de registro, por defecto en `storage/logs/laravel.log`

Logging: niveles de importancia

- Los mensajes pueden tener distintos niveles de importancia, de menor a mayor: `debug`, `info`, `notice`, `warning`, `error`, `critical`, `alert`, `emergency`
- Sólo se guardarán en el log los que sean de importancia mayor o igual a la establecida en el fichero `config/logging.php`

Logging: uso de Monolog

- Para poder usar Monolog hay que incluir la clase Log

```
use Illuminate\Support\Facades\Log;
```

- Esta clase ofrece métodos estáticos que se corresponden con los niveles de importancia de los mensajes

```
Log::debug("Debug message");
```

```
Log::warning("This is a warning");
```

Logging: ejemplo

```
Product.php x
1  <?php
2
3  namespace App\Models\Warehouse;
4
5  use Illuminate\Support\Facades\Log;
6
7  class Product {
8      function __construct() {
9          Log::debug("Creating new product");
10     }
11 }
12
```

```
cperez@cperez-ubuntu: ~/Code/laravel-intro
Archivo Editar Ver Buscar Terminal Ayuda
cperez@cperez-ubuntu:~/Code/laravel-intro$ tail -f storage/logs/laravel.log
[2017-01-29 15:38:00] local.DEBUG: Creating new product
```

```
cperez@cperez-ubuntu: ~/Code/laravel-intro
Archivo Editar Ver Buscar Terminal Ayuda
cperez@cperez-ubuntu:~/Code/laravel-intro$ php artisan tinker
Psy Shell v0.8.1 (PHP 7.0.13-0ubuntu0.16.04.1 - cli) by Justin Hileman
>>> use App\Models\Warehouse\Product;
=> null
>>> $p = new Product();
=> App\Models\Warehouse\Product {#644}
>>>
```

Introducción a Laravel

EJERCICIO

Ejercicio

- Crea un proyecto de Laravel usando Composer
- Mueve el código de los ejercicios de la sesión anterior a clases dentro de la carpeta `app/` del proyecto
- Prueba que el código se puede cargar y funciona usando Tinker
- Crea pruebas de código unitarias para las funciones y métodos y ejecútalas con PHPUnit

NOTA: Ahora para poder usar clases del espacio de nombres por defecto como `DateTime` o `DateInterval` hay que importarlas con `use`:

```
use DateTime;  
use DateInterval;
```