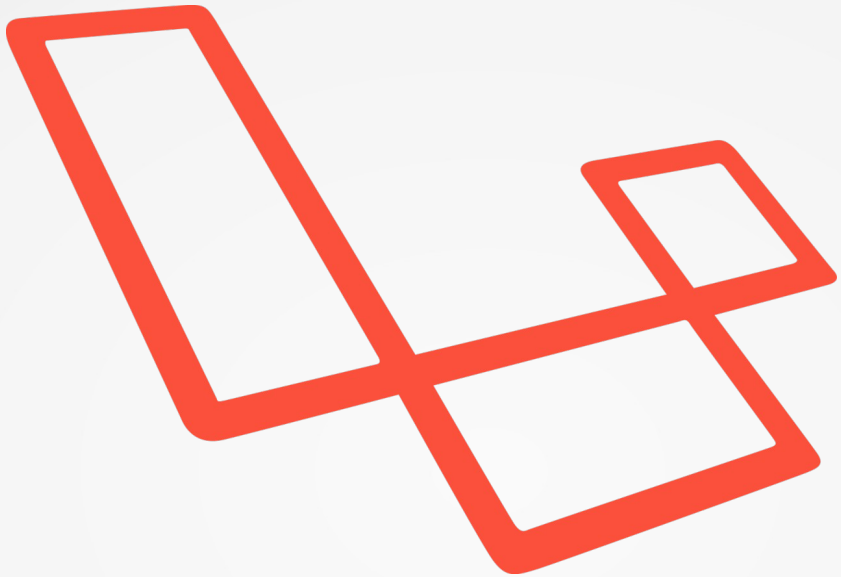
Framework de PHP para Artesanos Web



Introducción a Laravel

Diego Rodero / diego@rodero.es
Ingeniero en Informática por la UGR
Coordinador TIC en Escuela Andaluza de Salud Pública
Socio fundador y CTO en Eralapps S.L.

Índice

Introducción II

- La curva de aprendizaje es muy buena
  - Enseguida aprendes lo básico y estás haciendo webs como churros
- Es muy elegante
  - La sintaxis es muy bonita, sencilla y entendible
  - Favorece enormemente el mantenimiento del código
- Está optimizando para crear APIs de manera muy rápida y sencilla

Introducción III

- Se basa en el patrón de diseño
  - Modelo-Vista-Controlador (MVC)
- En la versión 4 de Laravel era más estricto, en la versión 5 es mucho más flexible (tiene muchas más cosas que modelos, vistas y controladores, como “traits”, “fachades”, etc)
- Tiene mecanismos de inyección de código, principalmente de tipo “middleware”
- Se basa en el lenguaje PHP ofreciendo un “lenguaje” por encima para hacerlo más sencillo y elegante.
  - Blade para las plantillas
  - Eloquent para el trabajo con bases de datos

Requisitos para el taller

- PHP 7.1
- Composer
- Laravel (como paquete de composer)
- MariaDB
- Optativo
  - Editor de código: Visual Studio Code / Sublime Text
  - Cliente de SQL: Dbeaver / MySQL Workbench

Instalación / MariaDB

- Instalación
  - sudo apt-get install mariadb-server
- Ponemos password en usuario de base de datos root
  - sudo mysql_secure_installation
- MariaDB / acceder sin ser root de ubuntu
  - Entramos como sudo:
    - sudo mysql -u root -p
  - Ejecutamos este código SQL:
    - use mysql;
    - update user set plugin='' where User='root';
    - flush privileges;

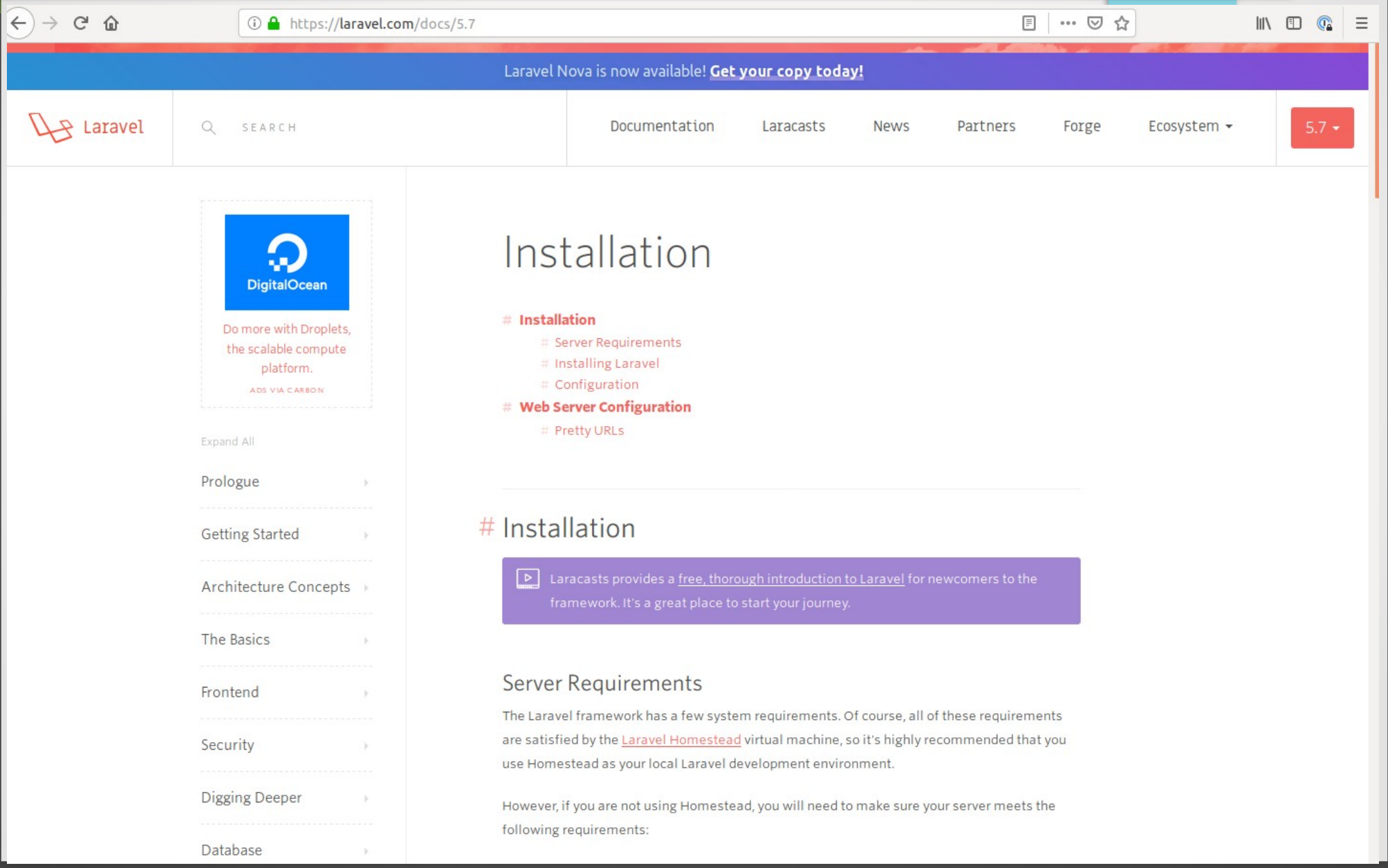
Instalación / PHP y Composer

- Instalamos PHP 7.2 y algunas librerías
  - sudo apt install php7.2-cli
  - sudo apt install php7.2-zip php7.2-mbstring php7.2-xml php7.2-mysql
- Instalamos composer (https://getcomposer.org/download/)
  - php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
  - php composer-setup.php
  - php -r "unlink('composer-setup.php');"
- Ponemos composer en un lugar del path
  - sudo mv composer.phar /usr/local/bin/composer

Instalación / Laravel

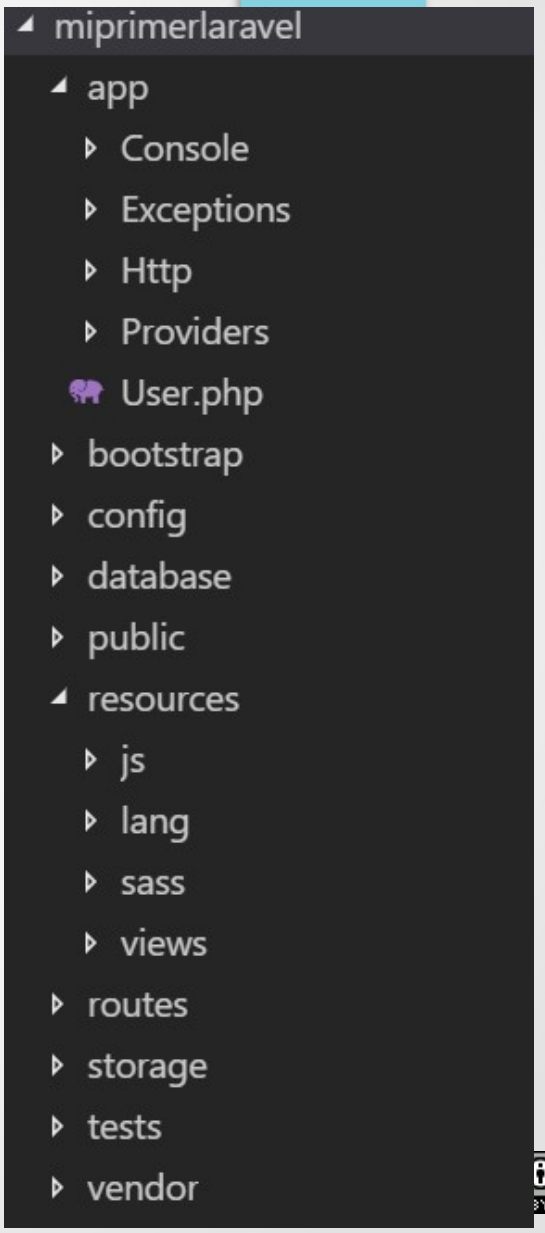
- Se lanza el comando de composer para instalar de forma global Laravel:
  - composer global require laravel/installer
- Hacer que el comando laravel esté en el path:
  - $HOME/.config/composer/vendor/bin

Instalación / Laravel



Creación de un proyecto

- laravel new miprimerlaravel
- php artisan serve
- Navegamos a:
  - http://127.0.0.1:8000
- Laravel tiene un servidor de páginas web integrado



Código del taller

http://bit.ly/tallerlaravel

Enrutado básico I

/routes/web.php

```
Route::get('/', function() {
    return view('wellcome');
});
```

/resources/views/welcome.blade.php

```
<html>
    <head>
        <title>Mi primera web con Laravel</title>
    </head>
    <body>
        <h1>Taller de Laravel UGR</h1>
    </body>
</html>
```

Enrutado básico II

/routes/web.php

```
Route::get('/contacto', function() {
    return view('contacto');
});

Route::get('/sobrenosotros', function() {
    return view('sobrenosotros');
});
```

/resources/views/contacto.blade.php

/resources/views/sobrenosotros.blade.php

Enrutado básico III

/resources/views/wellcome.blade.php
/resources/views/about.blade.php
/resources/views/contact.blade.php

```
<div>
        <ul>
            <li><a href="/">Inicio</a></li>
            <li><a href="/contact">Contacto</a></li>
            <li><a href="/about">Sobre nosotros</a></li>
        </ul>
</div>
```

Ficheros de plantillas BLADE I

- Es el motor de plantillas de Laravel
- Utiliza una sintaxis propia para las vistas
- Las vistas se “compilan” en PHP plano y se “cachean”.
- No penalizan el rendimiento de la aplicación

Ficheros de plantillas BLADE II

- Los ficheros de “layout” (diseño), crean secciones donde irá el código de otras páginas:
  - @yield('contenido')
- Los ficheros de código, “extiendien” una plantilla, y configuran las secciones donde va el contenido:
  - @extends('layout')
  - @section('contenido')
  - <h1>Mi primera sitio web</h1>
  - @endsection

En vivo y en directo

- Creamos una plantilla
  - /resources/views/layout.blade.php
- En las páginas de Wellcome, Sobre Nosotros y Contacto, usamos la plantilla

Pasando datos a las vistas I

- Cambiamos en el fichero de rutas la llamada a la página principal:

```php
Route::get('/', function () {

    $tareas = [
        'Ir a la tienda',
        'Hacer los deberes de inglés',
        'Dar de comer a los peces'
    ];

    $titulo = 'Tareas Principales';

    return view('welcome')->withTareas($tareas)->withTitulo('titulo');
});
```

Pasando datos a las vistas II

- Se pueden utilizar dos formas de pasar parámetros:
  - return view('welcome')→withTareas($tareas)→ withTitulo('titulo');
  - return view('welcome', compact('tareas','titulo'));

Pasando datos a las vistas III

- En el fichero wellcome, mostramos las variables de título y lista de tareas:

```
@extends('layout')

@section('contenido')
<h1>Página principal - {{ $titulo }}</h1>

<ul>
    @foreach ($tareas as $tarea)
        <li>{{ $tarea }}</li>
    @endforeach
</ul>

@endsection
```

Pasando datos a las vistas IV

- Blade proporciona “alias” sobre cierto código PHP para hacerlo mas corto y elegante:

```
<ul>
    @foreach ($tareas as $tarea)
        <li>{{ $tarea }}</li>
    @endforeach
</ul>
```

- La sintaxis de doble llave lo que hace es hacer un “echo” de la variable pero “escapándola” para evitar ataques de inyección de código.

Controladores I

- En el fichero de rutas, tendremos las rutas con las llamadas a cada controlador.
- En el controlador, tendremos la lógica de negocio de la aplicación

```
> /routes/web.php

    Route::get('/', 'PaginasController@portada');
```

- Podemos generar un controlador con el comando:
  - php artisan make:controller PaginasController

Controladores II

- Se ha creado el controlador en:
  - /app/Http/Controllers/PaginasController.php
- Trasladamos el código que antes teníamos en el fichero de rutas:

```php
class PaginasController extends Controller
{
    public function portada() {
        $tareas = [
            'Ir a la tienda',
            'Hacer los deberes de inglés',
            'Dar de comer a los peces'
        ];

        $titulo = 'Tareas Principales';

        return view('welcome', compact('tareas','titulo'));
    }
}
```

Controladores III

- Hacemos lo mismo para las otras dos rutas:

```
Route::get('/contacto', 'PaginasController@contacto');
Route::get('/sobrenosotros', 'PaginasController@sobrenosotros');
```

```
public function contacto() {
    return view('contact');
}

public function sobrenosotros() {
    return view('about');
}
```

Controladores IV

- Ahora mismo nuestra web tiene:
  - Un fichero donde definimos las rutas
  - Un controlador con el código que se va a ejecutar para cada ruta
  - Una vista con el código HTML que se cargará para cada ruta.

Migraciones y Bases de Datos I

- La conexión a la base de datos se configura en el fichero:
  - .env
- Este fichero está detro del **.gitignore**, por lo que no se subirá al repositorio de código.
  - Tendremos un fichero .env por cada desarrollador y otro .env en producción
- En el fichero
  - /config/database.php
- Tenemos la configuración más detallada

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=miprimerlaravel
DB_USERNAME=laravel
DB_PASSWORD=laravel
```

Migraciones y Bases de Datos II

- Para crear nuestra base de datos, lanzamos estos comandos:

```
mariadb -u root -ptallerosl

CREATE DATABASE miprimerlaravel;

CREATE USER 'laravel'@'%' IDENTIFIED BY 'laravel';

GRANT ALL PRIVILEGES ON miprimerlaravel.* TO 'laravel'@'%' WITH GRANT OPTION;
```

Migraciones y Bases de Datos III

- Las migraciones son como un sistema de control de versiones sobre la estructura de la base de datos.
  - Guardamos creación y alteración de estructuras de tablas
  - Hace independiente a la web frente al motor de base de datos utilizado
- Permite recrear la estructura de la base de datos en cualquier momento

Migraciones y Bases de Datos IV

- Lanzar las migraciones pendientes
  - php artisan migrate
- Podemos echar para atrás migraciones. Para deshacer la anterior:
  - php artisan migrate:roolback
- Para deshacer todos los cambios y lanzar las migraciones desde cero:
  - php artisan migrate:fresh
- **¡¡CUIDADO!!** Esto borra todos los datos

Migraciones y Bases de Datos V

- Analizamos el fichero de migración de la creación de la tabla de usuarios:
  - /database/migrations/ 2014_10_12_000000_create_users_table.php
- Lanzamos las migraciones actuales (las que trae por defecto) con:
  - php artisan migrate
- Creamos una nueva migración:
  - php artisan make:migration create_proyectos_table

Migraciones y Bases de Datos VI

- Vemos el fichero recién creado y lo modificamos:

```
public function up()
{
    Schema::create('proyectos', function (Blueprint $table)
        $table->increments('id');
        $table->string('titulo');
        $table->string('descripcion');
        $table->timestamps();
    });
}
```

- php artisan migrate

Eloquent y Namespacing I

- Convención: El nombre del modelo va en singular, la tabla en plural
- Creamos el modelo:
  - php artisan make:model Proyecto
- Se guarda en:
  - */app/Proyecto.php*
- En principio, la clase está vacía.

Eloquent y Namespacing II

- Vamos a usar **tinker** para probar el modelo e insertar datos en la base de datos:
  - php artisan tinker

```
>>> \App\Proyecto::all();

>>> \App\Proyecto::first();

>>> $proyecto = new \App\Proyecto;
>>> $proyecto->titulo = 'Mi primer proyecto';
>>> $proyecto->descripcion = 'Lorem impsum';

>>> $proyecto;

>>> $proyecto->save();
```

Eloquent y Namespacing III

- El objeto lo crea como una **colección**. Las colecciones son Arrays de PHP con *esteroides*.
- Ya tenemos la **Vista**, el **Modelo** y el **Controlador**.
- Vamos a crear una nueva ruta para mostrar los proyectos, un nuevo controlador y una vista para mostrarlo (el modelo ya lo tenemos).
  - Route::get('/proyectos', 'ProyectosController@index');

Eloquent y Namespacing IV

- php artisan make:controller ProyectosController

- /app/Http/Controllers/ProyectosController.php
- public function index() {

```
    return view('proyectos.index');
}
```

Eloquent y Namespacing V

- /resources/views/proyectos/index.blade.php

- <html>
  <head>
    <title></title>
  </head>
  <body>
    <h1>Proyectos</h1>
  </body>
  </html>

Eloquent y Namespacing VI

- Modificamos el controlador para leer los proyectos de la BD y pasárselos a la vista.

```php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use \App\Proyecto;

class ProyectosController extends Controller
{
    public function index() {
        $proyectos = Proyecto::all();

        return view('proyectos.index', compact('proyectos'));
    }
}
```

Eloquent y Namespacing VII

- En la vista, mostramos los proyectos:

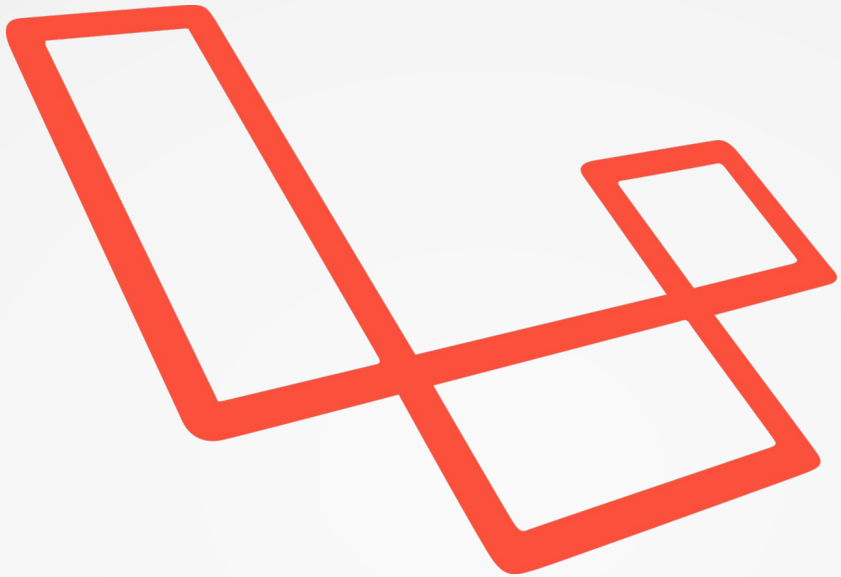
```
<html>
<head>
    <title></title>
</head>
<body>
    <h1>Proyectos</h1>

    <ul>
        @foreach ($proyectos as $proyecto)
            <li>{{ $proyecto->titulo }}</li>
        @endforeach
    </ul>
</body>
</html>
```

Proyectos Reales

- Registro de Actividades de Tratamiento del RGPD en la EASP
- Control de Jonradas:
  - Gestión de inscripciones y asistentes al congreso de APISA 2018
- Panel de Control de Apps de Eralapps
- API para la conexión entre la app de registro de inventario y la aplicación web GLPI de la EASP

Framework de PHP para Artesanos Web



Introducción a Laravel

Diego Rodero / diego@rodero.es
Ingeniero en Informática por la UGR
Coordinador TIC en Escuela Andaluza de Salud Pública
Socio fundador y CTO en Eralapps S.L.