Unidad 3
Introducción Laravel

Taller de aplicaciones web utilizando PHP

Carlos San Juan Contreras

csanjuan@ubiobio.cl

Semestre 2018-II

Instalación

¿Qué es Composer?

- Es un gestor de dependencias que se encarga automáticamente de crear proyectos, instalar, actualizar o eliminar componentes y a su vez las dependencias de éstos.

- Descargar desde Adecca el instalador de Composer
- Instalarlo como cualquier otro programa
- Abrir una terminal y escribir “Composer”. Si está bien instalado se mostrará un listado de todos los comandos disponibles.

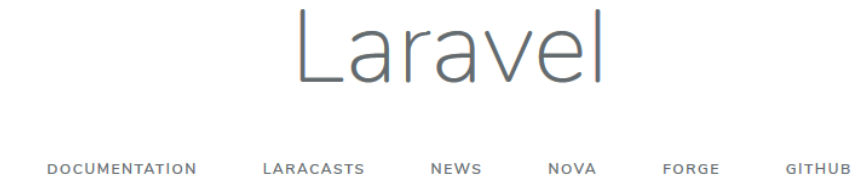
¿Cómo instalar Laravel?

- Previamente se debe tener instalado Composer.
- Abrir una terminal
- Si tenemos un servidor Xampp, dirigirse a la siguiente carpeta:
  - C:\xampp\htdocs
- Escribir en la consola:
  - composer create-project --prefer-dist laravel/laravel laravel
- Se creará una nueva carpeta llamada laravel con la última versión de laravel disponible.

Si se instaló correctamente…

Visitar la siguiente url:

http://localhost/laravel/public/



Rutas

Front Controller - Recordatorio

- El patrón de diseño Front Controller consiste en que un solo componente de la aplicación es el responsable de manejar de todas las peticiones HTTP que ésta recibe. Es decir, hay un solo punto de acceso para todas las peticiones.
- En Laravel esta función la cumple el archivo index.php que se encuentra en el directorio public/ junto con el archivo ".htaccess". Este último archivo se encarga de redirigir todas las peticiones a index.php
- El directorio public contiene además, las imágenes, archivos CSS y de Javascript que será públicos para los usuarios y visitantes de la aplicación, el resto de archivos donde se encuentra la lógica de la aplicación es privada para ellos.

Directorio Rutas

- Se ubican en el directorio " Routes" en la carpeta raíz.
- Por defecto, existen 2 archivos de rutas
  - web.php se definen las rutas para la web y
  - api.php las rutas para crear APIs para la aplicación.

Métodos Clase Router

- La clase router permite registrar rutas que responden a cualquier verbo HTTP:
  - Route::get($uri, $callback);
  - Route::post($uri, $callback);
  - Route::put($uri, $callback);
  - Route::patch($uri, $callback);
  - Route::delete($uri, $callback);
  - Route::options($uri, $callback);

Buena práctica de programación

Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Tipos de Rutas disponibles

- Ruta Básica
  - Ruta con función anónima
  - Ruta para llamar al controlador
  - Ruta con Vista
  - Ruta para redirigir
- Ruta con parámetros
  - Con parámetros requeridos
  - Con parámetros opcionales
  - Con restricción de expresión regular
  - Entre otras..

Ejemplo Básico

- Para realizar un ejemplo básico, escribir el siguiente código en web.php y luego llamarlo por Url.

```
Route::get('/prueba', function () {
    return 'Hola mundo';
});
```

- Se llama a la clase Route que ejecuta al método relacionado con el verbo HTTP (GET) que acepta dos parámetros:
- La URL que se llamará desde el navegador y
- Función anónima que devuelve lo que se quiere mostrar.

Ejemplo con Parámetros obligatorios

- Escribir el siguiente código en web.php y luego llamarlo por Url.

```
Route::get('/auto/detalle/{id}', function ($id) {
    return “Quiero ver el detalle del auto con ID: {$id}";
});
```

- Laravel se encarga de capturar el segmento de la ruta que es dinámico (está entre llaves). Por lo tanto, en la URL se pasa el id del parámetro encerrado entre llaves y en la función anónima se recibe como como argumento para que pueda ser usado dentro de la función.

Ejemplo con Parámetros opcionales

- Escribir el siguiente código en web.php y luego llamarlo por Url.

```
Route::get('/auto/{marca}/{modelo?}', function ($marca, $modelo = null) {
    if ($modelo) {
        return "Tengo un auto {$marca}, modelo {$modelo}";
    } else {
        return "Tengo un auto {$marca}";
    }
});
```

- Se usa el carácter ? después del nombre del parámetro opcional. Debe añadirse un valor por defecto al parámetro cuando se recibe en la función anónima.

Controladores

¿Cómo crear controladores?

- Existen dos formas:
- Manualmente, crear el archivo “MiNombreController” dentro de la carpeta de laravel App\Http\Controllers
- Automatizado con artisan
  - En la terminal, situarse dentro del proyecto laravel y escribir

```
php artisan make:controller UserController
```

o

```
php artisan make:controller UserController --resource
```

O

```
php artisan make:controller UserController --resource --model=Usuario
```

Vistas

Modo de uso

- Se encuentran en el directorio principal del proyecto laravel, en la carpeta /resources/views
  - Basta con crear un archivo en el directorio especificado y utilizar las variables definidas en el controlador.
  - Se usa muy similar a como lo hemos estado haciendo.

La llamada básica de la vista es así: return view('users');

Usos más complejos

- Pasando datos a la vista directamente:

```
return view('users', [
    'users' => $users
]);
```

- Pasando datos a la vista con la función “with” (users es un Array):

```
return view('users')->with(['users' => $users]);
return view('users')->with('users', $users);
return view('users', compact('users'));
```


Vistas con Blade

¿Qué es Blade?

- Es el motor de plantillas que viene incorporado. Laravel compila y guarda en caché las vistas, por lo que usar el motor de plantillas Blade no afecta el rendimiento de nuestra aplicación. Las vistas compiladas se guardan en el directorio /storage/framework/views
  - Simple y potente
  - No restringe el uso de php nativo

    @php

    …

    @endphp
  - Sistema de caché
- ¿Cómo usarlo?
  - Incorporar a todos los archivos de vista la extensión .blade.php

Mostrando variables

- Se utiliza la sintaxis de dobles llaves {{ }}
- Ejemplo:
  - {{ $nombre }}

Estructuras condicionales

- Anteponer el símbolo @ para definir la estructura a usar.
- Ejemplo:

```
@if (! empty($nombre))
    ...
@endif

@if (! empty($users))
    ...
@else
    <p>La variable esta vacía.</p>
@endif
```

También se puede utilizar la condición @elseif

Estructuras ciclicas

- Anteponer el símbolo @ para definir la estructura a usar.
- Ejemplo:

```
@foreach ($users as $user)
    <li>{{ $user }}</li>
@endforeach

@for ($i = 0; $i < 10; $i++)
    El valor actual es {{ $i }}
@endfor
```

```
@while (true)
    <p>Esto es un demonio.</p>
@endwhile
```

La variable Loop

- Está disponible dentro de los ciclos. Provee información extra que nos será de utilidad.

```
@foreach ($users as $user)
    @if ($loop->first)
        Esta es la primera iteración.
    @endif

    @if ($loop->last)
        Esta es la última iteración.
    @endif
@endforeach
```


La variable Loop

Property	Description
`$loop->index`	The index of the current loop iteration (starts at 0).
`$loop->iteration`	The current loop iteration (starts at 1).
`$loop->remaining`	The iterations remaining in the loop.
`$loop->count`	The total number of items in the array being iterated.
`$loop->first`	Whether this is the first iteration through the loop.
`$loop->last`	Whether this is the last iteration through the loop.
`$loop->depth`	The nesting level of the current loop.
`$loop->parent`	When in a nested loop, the parent's loop variable.