**¿Qué es un framework*?***

Una definición sencilla para *framework*, sería un conjunto estandarizado de conceptos y criterios para apuntar a un tipo de problema particular que sirve como referencia para resolverlo. En pocas palabras se trata de un entorno de trabajo.

En el ámbito que nos ocupa, es decir, en el software, un *framework* se define como una estructura conceptual asistida con determinados módulos concretos que pueden servir de base para la organización y desarrollo de una aplicación. Es un compendio de reglas a tener en cuenta para poder desarrollar un proyecto en base a esas reglas y tenerlo todo encapsulado siguiendo, en nuestro caso el modelo vista controlador.

**Arquitectura**

El framework utilizado se basa en el patrón MVC[[1]](#footnote-1) y en base a este patrón podemos decir, de forma breve:

**Modelo**: Maneja las operaciones lógicas, y de manejo de información (previamente enviada por su ancestro), para resultar de una forma explicable. Cada miembro debe ser llamado, con su correcto nombre y en principio, con su verdadera naturaleza: el manejo de información, su complementación directa.

**Vista**: Expresa la última forma de los datos: la interfaz gráfica que interactúa con el usuario final del programa (GUI). Después de todo, a este miembro le toca evidenciar la información obtenida hasta hacerla llegar al controlador.

**Controlador**: Se controla el acceso a nuestra aplicación, y esto puede incluir: archivos, scripts, o programas, cualquier tipo de información que permita la interfaz. Así, podremos diversificar nuestro contenido de forma dinámica, y estática (a la vez) pues, solo debemos controlar ciertos aspectos.

A lo que respecta a la parte lógica de un *framework*, esta se basa en el MVC[[2]](#footnote-2) y además necesita de un sistema encargado de rutear, es decir, dividir las peticiones sin tantos condicionantes.

A continuación nos centramos, en las paginas sucesivas, a detallar el *framework* utilizado y el porqué de esta elección.

El framework escogido para este proyecto es Laravel, Laravel 5.4 para ser más exactos. El motivo es porque durante este curso hemos tratado con él y por tanto se ha decidido arriesgar a implementar un proyecto más grande con este *framework* como base.

**Laravel 5.4**. **Un poco de historia**.

Filosofía principal: Desarrollar código php de forma simple y elegante, evitando así el código espagueti[[3]](#footnote-3).

Creado en 2011 y desarrollado por Taylor Otwell[[4]](#footnote-4). Laravel, intenta aprovechar los mejor de otros frameworks además de las características de las últimas versiones de PHP. La mayor parte de Laravel está formada por dependencias de Simfony, esto conlleva a la hora de desarrollar Laravel un desarrollo, también de sus dependencias.

**Laravel 5.4. Características**.

**

***Fig. 1.1.*** *Logo de Laravel*

* Sistema de ruteo, también RESTful[[5]](#footnote-5)
* Blade, Motor de plantillas
* Peticiones Fluent

**

***Fig. 1.2.*** *Logo de Composer*

* Eloquent ORM[[6]](#footnote-6)
* Basado en Composer[[7]](#footnote-7)
* Soporte para el caché
* Soporte para MVC
* Usa componentes de Symfony
* Adopta las especificaciones PSR-2 y PSR-4

Si bien ya se ha mencionado antes, vamos a detallar el funcionamiento de Laravel basado en el patron MVC, para tratar de ver las particularidades del framework en cuestión.

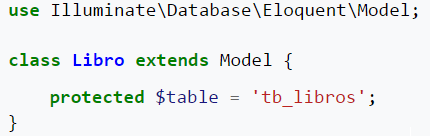
Hay que tener en cuenta, que en Laravel no se desarrolla siguiendo un estricto patrón MVC, se propone para ello usar *Routes with Closures*[[8]](#footnote-8) con el objetivo de hacer el código más claro. No obstante, se puede usar el MVC tradicional.

**Modelo en Laravel**.

Como se mencionó con anterioridad, Laravel dispone de un sistema de mapeo de datos relacional llamado Eloquent ORM que facilita la creación de modelos. Se funda en el patrón active record[[9]](#footnote-9). Para crear modelos es tan sencillo como desde la consola del sistema, dentro de la carpeta del proyecto:

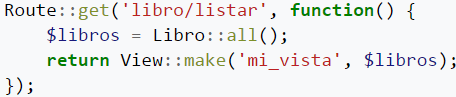
php artisan make:model Libro

Donde User es el nombre del modelo a crear. Esto genera un archivo php dentro de la carpeta app que es donde se almacenan todos los modelos.

**

***Fig.1.3.*** *Modelo creado desde comando, recien creado no incluye variables, el aquí mostrado es solo de ejemplo.*

Realmente, el código anterior puede llegar a ser más simple aún. En el caso que el nombre de la tabla coincida con el nombre de la clase. Ya que Laravel, usa el paradigma de programación donde se favorece "la convención sobre la configuración". Y si ahora necesitamos disponer un listado, en una determinada ruta, por ejemplo en: http://mi-aplicacion.com/libro/listar, entonces, sólo haría falta crear la Ruta e interactuar con el modelo 'Libro' anteriormente creado, tal que así:

**

***Fig.1.4.*** *Detalle de ruta para mostrar todos los libro de la base de datos. Se devuelve la vista con todos los libros*

**Vista en Laravel**

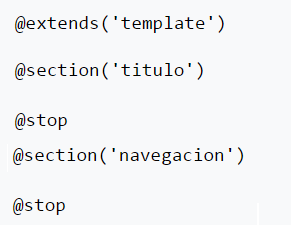
Laravel se basa en un sistema de procesamiento de plantillas llamado Blade. Con este sistema, se consigue un código limpio dentro de las vistas. El sistema Blade permite una sintaxis reducida en su escritura. Además gestiona la memoria caché muy bien lo que hace vistas rápidas. Lo que realmente dota a Blade de gran ventaja es su sistema de manejar plantillas.

Ejemplo de vista, estas constan de dos archivos, por un lado la plantilla en sí:

**

***Fig. 1.5.*** *Plantilla web. El código @yield() identifica al método donde como parámetro se indica el nombre de la zona desplegar.*

Y luego, el código de la vista, donde se define la plantilla a usar y la información de las distintas zonas a desplegar:

**

***Fig. 1.6****. En primer lugar se identifica la plantilla a utilizar y luego se definen las zonas*

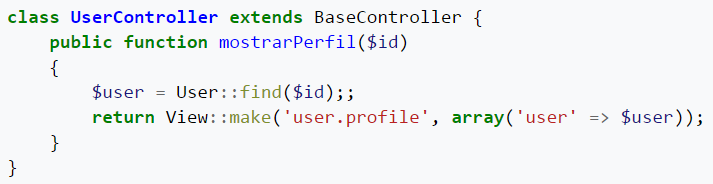
**Controlador en Laravel**

Permiten organizar el código en clases sin tener que escribirlo todo en las rutas. Cabe notar, que todos los controladores se extienden del BaseController (controlador principal). Para crear controladores, igual que con los modelos, se puede usar un comando para ello:

php artisan make:controller NameController --resource

Donde NameController es el nombre del controlador y con el parámetro --resource, conseguimos que nos cree dentro del archivo la estructura de funciones tales como índice, store, etc.

Como ejemplo de controlador:

**

***Fig. 1.7.*** *Detalle de controlador. Dentro de este se pueden añadir tantas funciones como sean necesario. En este caso se devuelve a la vista el perfil solicitado del usuario en función del id suministrado.*

Los controladores se pueden llamar desde las rutas de la siguiente forma:

**

***Fig.1.8.*** *Detalle de llamada al controlador a través de una ruta. Mediante petición get, se llamada la vista con un id definido y se llama al controlador necesario y al método en cuestión*

**Migraciones en Laravel**

Por último, para no hacer muy extenso este capítulo y mencionar todo lo relacionado con laravel, sí que haremos especial mención a las migraciones, útiles para confeccionar la base de datos del proyecto que nos ocupa.

Las migraciones son una excelente opción para controlar la base de datos, permitiéndonos una fácil modificación de la misma. Las migraciones están relacionadas con el schema builder de Laravel para una fácil construcción del esquema de la base de datos.

Para llevar a cabo una migración, usamos el siguiente comando desde la terminal abierta en el proyecto:

php artisan[[10]](#footnote-10) make:migration create\_aspectos\_table

Lo que se consigue con esto es crear el archivo con el nombre especificado después de migration. No es más que una clase heredera de Migration en la que se define el esquema de la tabla a crear.

La nueva migración aparecerá en el directorio database/migrations. El fichero de migración contiene en el nombre la fecha la cual permite a Laravel determinar el orden de las migraciones.

Si nos centramos ahora en la estructura del fichero obtenido tras hacer el make:migration, observamos que la clase contiene dos métodos. El método up(), sirve para añadir nuevas tablas, columnas o índices a la base de datos, mientras que el método down(), sirve para llevar a cabo una especie de rollback sobre las operaciones ejecutadas por up ().

Ejemplo de función up():

**

***Fig.1.9.*** *Detalle de función up para crear la tabla flight. Se añaden todas las columnas y el tipo de datos*

1. **MVC**: Modelo, vista controlador. Controlador→Modelo→Vista. Es importante tener este patrón claro a la hora de trabajar con un *framework*, sobre todo si este se basa en este patrón como base de trabajo. Hay que tener en cuenta que este modelo difiere un poco según quien lo defina y existen controversias sobre donde debería ir, a veces cada funcionalidad. [↑](#footnote-ref-1)
2. **Controlador**: Ser capaz de manejar rutas, archivos, clases, métodos y funciones.

   **Modelo**: es como un script habitual en el servidor, solo que agrupado bajo un modelo reutilizable.

   **Vista**: como incluyendo cualquier archivo en nuestra ejecución, muy simple. [↑](#footnote-ref-2)
3. **Código espagueti**: Clasificado como un anti-patrón de programación. Se define así a los programas que tienen una estructura compleja e incomprensible. El motivo de su denominación, es que este código se asemeja a un plato de espaguetis, un montón de hilos enmarañados y anulados. [↑](#footnote-ref-3)
4. Creador de Laravel, Lumen, Forge, Envoyer, y Spark. Es desarrollador web. [↑](#footnote-ref-4)
5. Se dice que una aplicación es RESTful, cuando sigue un conjunto de principios de arquitectura tales como: Un protocolo cliente/servidor sin estado, un conjunto de operaciones bien definidas que se aplican a todos los recursos de información, una sintaxis universal para identificar los recursos y el uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. [↑](#footnote-ref-5)
6. El elocuente ORM ofrece una implementación ActiveRecord simple para trabajar con su base de datos. Cada tabla de la base de datos tiene un "modelo" correspondiente que se utiliza para interactuar con esa tabla. [↑](#footnote-ref-6)
7. Se trata de un sistema de gestión de paquetes para la programación en PHP. Nos provee de una forma estándar el manejo de dependencia de php y la librerías necesarias. Desarrollado por Nils Adermann i Jordi Boggiano. La primera versión salió en 2012. [↑](#footnote-ref-7)
8. Closure o clausura. Se trata de funciones evaluadas en un entorno que contiene varias variables dependientes de otro entorno. Cuando se llama al Closure, este puede acceder a estas variables. El uso explicito de Closures se asocia con la programación funcional. También se puede dar el caso de que una Closure aparezca cuando una función está definida dentro de otra función. [↑](#footnote-ref-8)
9. **Patrón active record**: Es un patrón de arquitectura que se halla en aplicaciones que almacenan sus datos en Bases de datos relacionales. La interfaz de un cierto objeto debe incluir funciones como por ejemplo insertar (INSERT), actualizar (UPDATE) o eliminar (DELETE) y propiedades que correspondan de cierta manera a las columnas de la base de datos asociada. [↑](#footnote-ref-9)
10. Tanto en esta como en las instrucciones anteriores se usan comandos de artisan que es la línea de comandos propia de Laravel. Hay una gran variedad de instrucciones y estas se pueden consultar a través de: php artisan list. [↑](#footnote-ref-10)