Apuntes Laravel

¿Qué es? Laravel es un framework de desarrollo web con PHP, que simplifica el trabajo facilitando con herramientas el desarrollo de tareas comunes como enrutamiento web, la autentificación , el acceso a base de datos, la programación de la lógica de negocios, etc.

Algunas de las herramientas de las que se compone Laravel:

Blade : el motor de plantillas de Laravel para crear vistas dinámicas de forma sencilla.

Eloquent: el ORM de Laravel que permite interactuar con la base de datos mediante modelos sin usar SQL.

Artisan: interfaz de línea de comandos que simplifica ciertas tareas en Laravel como la creación y ejecución de migraciones, modelos y controladores.

MVC(modelo-vista-controlador)

El patrón MVC separa la gestión de los datos de la lógica de negocio y de la interfaz de usuario.

🔷 Modelo

- Gestiona el acceso a la base de datos
- No se accede directamente desde la vista
- Se accede **a través del controlador**

🔷 Controlador

- Recibe la petición del usuario
- Pide datos al modelo
- Envía/Solicita datos a la base de datos
- E**nvía los datos a la vista**

🔷 Vista

- Muestra los datos al usuario **preparados por el controlador**
- El usuario interactúa con ella realizar **nuevas peticiones al controlador**

📍 El patrón MVC separa los datos, la lógica de negocio y la interfaz de usuario.

Estructura de un proyecto Laravel

Las carpetas clave son:

- `app/Http/Controllers` → controladores
- `app/Models` → modelos
- `database/migrations` → migraciones
- `routes/web.php` y `routes/api.php` → rutas (estos son los “routes dispatcher”)
- `resources/views` → vistas
- `public/index.php` → raíz del proyecto que invocará al route dispatcher

📌 Si te preguntan:

> ¿Qué archivo gestiona los routes dispatcher?
>
> 👉 **web.php / api.php**

Rutas en Laravel

Las rutas Laravel permitirán asociar las diferentes URLs de nuestro proyecto con el controlador que la gestiona, son generadas por el “route dispatcher”, el archivo web.php o api.php de la carpeta routes. Las rutas podrán implementar los diferentes verbos HTTP mediante llamadas a métodos estáticos de la clase Route:

```
Route::get(), Route::post(), Route::put(), etc.
```

Ejemplo:

```
Route::get('/', [ControladorRaiz::class, 'funcion_raiz']);
```

→ la petición get a la URL base de la aplicación será gestionada por ControladorRaiz, concretamente por la función “funcion_raiz” de dicho controlador.

A una ruta se le puede dar un nombre o alias:

```
Route::get('/', [ControladorRaiz::clase, 'funcion_raiz'])->name('Bienvenida);
```

Esto será de utilidad cuando queramos hacer redirecciones a las diferentes URLs de nuestra aplicación desde nuestro código.

🔷 Rutas

- Las rutas asocian **URL → controlador**
- Se definen en `web.php` o `api.php`
- Usan métodos HTTP:
  - `Route::get()`
  - `Route::post()`
  - `Route::put()`
  - etc.

🔷 Alias (name)

- Sirven para redirecciones
- Evitan escribir URLs a mano

🔷 Parámetros

- Se pasan por la URL
- Ejemplo: `/producto/{id}`
- Opcionales: `{id?}`

👉 Si te preguntan:

> ¿Para qué sirven los parámetros en rutas?
>
> Respuesta: **para enviar datos (como un id) al controlador.**

Modelos y migraciones en Laravel

Por medio de los modelos de datos y las migraciones Laravel nos permite realizar todas las tareas asociadas a la base de datos sin necesidad de escribir una sola sentencia SQL. Las **migraciones** nos permitirán definir la estructura de la base de datos (tablas), mientras que el **modelo** nos permitirá interactuar con la base de datos relacional sin tener que usar código SQL.

Para realizar la configuración de Laravel para que se conecte a la base de datos accederemos al archivo **.env** . Esta configuración debemos realizarla antes de ejecutar cualquier migración.

Cada vez que creamos un nuevo proyecto Laravel se crean automáticamente una serie de migraciones que construyen una serie de tablas en nuestra base de datos. Estos archivos se encuentran en la carpeta **database/migrations.**

Una vez ejecutada una migración se pueden deshacer, ya que Laravel creará la tabla migrations que nos permitirá auditar las migraciones.

Comando de **Artisan**, ejemplo de migración para crear la tabla “clientes”:

```
php artisan make:migration create_clientes_table
```

Con el comando: `php artisan migrate` se ejecutarán las migraciones que no han sido ejecutadas. En caso de error en el proceso de diseño de la tabla podemos deshacer las migraciones con el comando: `php artisan migrate:rollback`. Con este comando se deshace la ultima migración lanzada. Si queremos deshacer una concreta ejecutamos `php artisan migrate:rollback –batch=X`, donde X será el numero de lote.

🔷 Migraciones

- Definen la estructura de la base de datos
- No usan SQL
- Se ejecutan con `php artisan migrate`
- Se pueden deshacer (rollback)

> Las migraciones permiten crear y modificar tablas de la base de datos de forma controlada.

Los **modelos** nos permitirán interactuar con la base de datos para realizar consultas, modificaciones o actualizaciones en las tablas sin tener que usar sentencias SQL.

Para crear un modelo usaremos el siguiente comando:

```
php artisan make:model nombre_modelo.
```

Cuando creamos un modelo debemos definir características de la tabla asociada al modelo como `$fillable`, que indicará qué campos pueden ser editados o `$hidden` que indica que campos de la tabla no se podrán mostrar al usuario. Un modelo llevará asociada una migración, por lo que si quisiéramos crear el modelo y la migración al mismo tiempo podríamos ejecutar el comando:
`php artisan make:model –migration.`

🔷 Modelos

- Representan una tabla
- Permiten consultar, insertar y modificar datos
- Usan Eloquent
- Propiedades importantes:
  - `$fillable` - `$hidden`

> Modelo ≠ migración
>
> La migración crea la tabla, el modelo trabaja con los datos.

Los controladores

El controlador responderá a la petición hecha por el cliente. La asociación ruta-controlador es realizada en el “route dispatcher”, archivos web.php o api.php.

Para crear un controlador ejecutamos el comando:

```
php artisan make:controller NombreControlador.
```

Los controladores contienen funciones públicas que implementan la lógica de negocio y gestionan las vistas. Todo controlador controlará varias vistas, aunque si necesita mostrar datos deberá acceder al modelo previamente para obtener los datos.

- Responden a las peticiones del cliente
- Contienen la lógica de negocio
- Se asocian a rutas
- Llaman a modelos y vistas

👉 Regla típica:

> Normalmente hay un controlador por modelo (aunque depende del proyecto).

Vistas: plantillas blade.

Dentro del patrón MVC la vista se encarga de interactuar con el usuario. Para la creación de las vistas en Laravel contamos con el motor de plantillas Blade. Las vistas contendrán código HTML y directivas Blade que nos permitirán iterar sobre los datos, mostrar errores generados en el proceso de validación de los datos enviados a través de un formulario, etc. Las vistas podrán ser llamadas directamente desde el “route dispatcher” si se trata de vistas con contenido estático, pero lo más común es que sean llamadas desde un controlador que a su vez habrá interactuado previamente con un modelo para obtener los datos que serán pasados a la vista.

- Interfaz de usuario
- HTML + Blade
- Muestran datos
- Normalmente se llaman desde el controlador

👉 Si preguntan:

> ¿Puede una vista ser llamada directamente?
>
> ✔️ Sí, si es estática
>
> ✔️ Lo normal: desde un controlador

Requisitos e instalación de Laravel

- PHP
- PostgreSQL
- Composer
- Laravel installer

📌 Lo importante no son los comandos, sino saber:

> Composer es el gestor de dependencias de PHP.

Para instalar Composer lo primero que haremos será descargar Composer desde su sitio web:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
```

Podemos instalar Composer de forma local dentro de un directorio de un proyecto o globalmente en el sistema. Esta segunda forma de instalación nos permitirá que el gestor de dependencias se ejecute en cualquier parte del sistema.

Para realizar una instalación global ejecutaremos el siguiente comando:

```
php composer-setup.php --install-dir=/usr/local/bin –filename=composer
```

Una vez realizada la instalación, si queremos eliminar el instalador ejecutamos el siguiente comando:

```
php -r "unlink('composer-setup.php');"
```

⚠️ Una vez instalado Composer necesitamos renombrar el archivo `/usr/local/bin/composer.phar` como `/usr/local/bin/composer`. De no hacerlo, nos podría dar error a la hora de ejecutar el Laravel/Installer.

Para comprobar si Composer se ha instalado correctamente escribiremos en la terminal el siguiente comando que mostrará la versión instalada: `composer --version`

Para **instalar Laravel** haremos uso de composer y ejecutaremos el siguiente comando:

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

Para verificar que todo ha ido bien ejecutaremos el comando que nos muestra la versión de Laravel/installer. `Laravel --version`

Una vez instalado el instalador de Laravel ya podremos crear nuestros proyectos en Laravel.