

## **SEMINARIO DE COMPUTACIÓN E INFORMATICA II**

### **LARAVEL 8.0**

**Material Elaborado por: Ing. Sandra Marcela Guerrero  
con base en Documento Laravel 5, Antonio Javier Gallego**

**UNIVERSIDAD DE NARIÑO  
FACULTAD DE INGENIERIA  
PROGRAMA DE INGENIERIA DE SISTEMAS**



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### I. CONCEPTUALIZACIÓN

**Laravel** es un framework de aplicaciones web que trabaja con PHP 5.0 en adelante.

Maneja una sintaxis elegante y fácil. Laravel tiene pocos requerimientos del sistema y es empleado por una comunidad bastante significativa. [1]

Es un framework moderno y ofrece diversas potencialidades en el desarrollo de aplicaciones ya que se centra en la calidad del código, facilidad de mantenimiento y escalabilidad con el fin de crear proyectos de cualquier tipo de tamaño [1].

#### Características [2]

1. **Blade:** El cual es un sistema de plantillas para las vistas, permite aprovechar trozos de código.
2. **Eloquent:** Integra un sistema ORM de mapeado de datos relacional llamado Eloquent. El desarrollador trabaja con objetos y Eloquent lo convierte a SQL.
3. **Artisan:** Laravel incorpora un intérprete de línea de comandos el cual facilita la interacción de los usuarios y les ayuda a realizar diferentes tareas de forma rápida.
4. **Middlewares:** Son controladores que se ejecutan antes o después de una petición para acceder a ciertos recursos.
5. **Documentación muy buena:** La cual es accesible y se complementa cada vez más gracias a la enorme comunidad que Laravel tiene.

### II. PROCESO DE INSTALACIÓN

#### 1. Descargar e Instalar XAMPP

Xampp es un servidor web que nos permitirá ejecutar proyectos PHP de forma fácil y rápida.

Para descargar XAMPP se puede hacer desde  
<https://www.apachefriends.org/es/index.html>

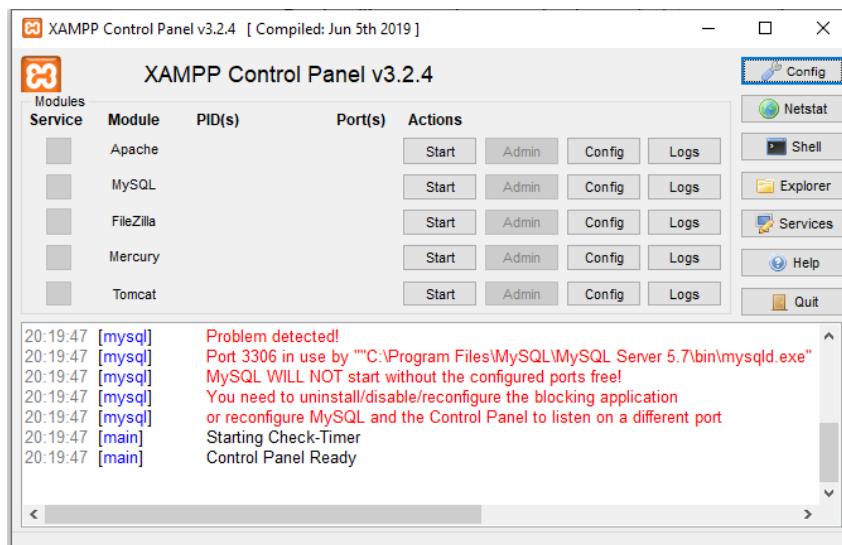
The screenshot shows the official XAMPP download page. At the top, it features the XAMPP logo and the text "XAMPP Apache + MariaDB + PHP + Perl". Below this, there's a section titled "¿Qué es XAMPP?" with a brief description. To the right, there's a video thumbnail titled "Introduction to XAMPP" showing a play button over a large white 'B' icon. At the bottom, there are four download links: "Descargar" (Windows 7.4.10 (PHP 7.4.10)), "XAMPP para Linux 7.4.10 (PHP 7.4.10)", and "XAMPP para OS X 7.4.10 (PHP 7.4.10)".



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Desde allí, se prosigue a seleccionar el sistema operativo correspondiente y continuar con el proceso de descarga. Es importante que XAMPP incluya una versión de PHP mayor o igual a 5.5.9.

El proceso de instalación es bastante sencillo y una vez finalizado deberá aparecer el panel de control de XAMPP con los módulos incorporados.



Para el caso de Windows debió crearse el directorio C:\xampp\htdocs donde se guardarán los proyectos trabajados en Laravel.

equipo > Disco local (C:) > xampp >			
Nombre	Fecha de modificación	Tipo	Tamaño
anonymous	31/07/2020 19:23	Carpeta de archivos	
apache	31/07/2020 19:23	Carpeta de archivos	
cgi-bin	31/07/2020 19:30	Carpeta de archivos	
contrib	31/07/2020 19:23	Carpeta de archivos	
FileZillaFTP	31/07/2020 19:30	Carpeta de archivos	
htdocs	22/09/2020 18:14	Carpeta de archivos	
img	31/07/2020 19:23	Carpeta de archivos	

Para comprobar que el servidor se ha instalado correctamente debemos iniciar el servicio de Apache desde el panel de control de la aplicación y luego escribir la URL: <http://localhost> en el navegador la cual nos mostrara la página que por defecto carga XAMPP.



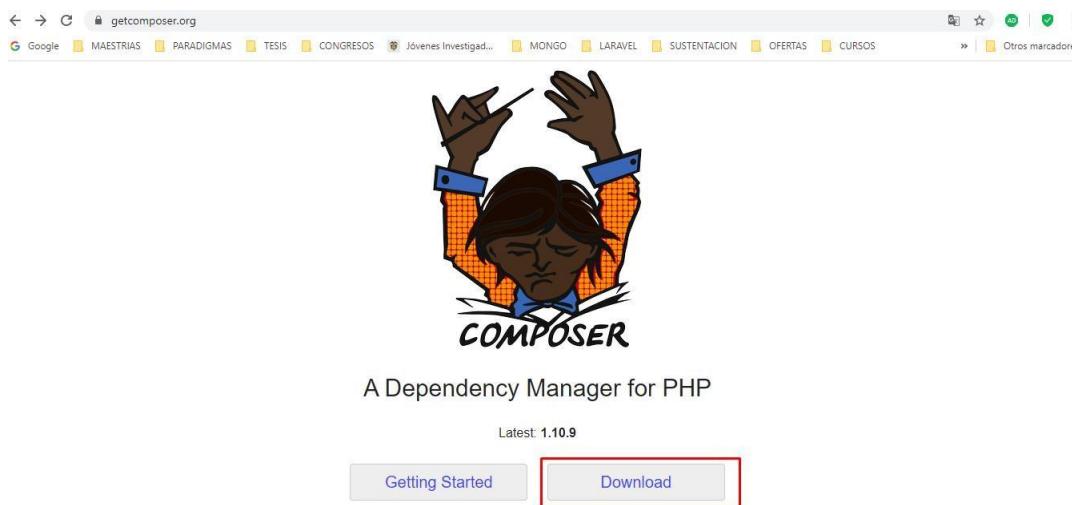


## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### 2. Descargar e Instalar Composer

Para la instalación de Laravel es necesario previamente contar con Composer el cual es un gestor de dependencias para PHP que nos permitirá descargar las librerías que sean requeridas para el proyecto con el que estemos trabajando.

La descarga de Composer puede realizarse desde <https://getcomposer.org/>, seleccionando la opción Download y el sistema operativo respectivo.



Para usuarios Windows se debe descargar el archivo ejecutable e iniciar la instalación de la misma forma que se hizo con XAMPP.

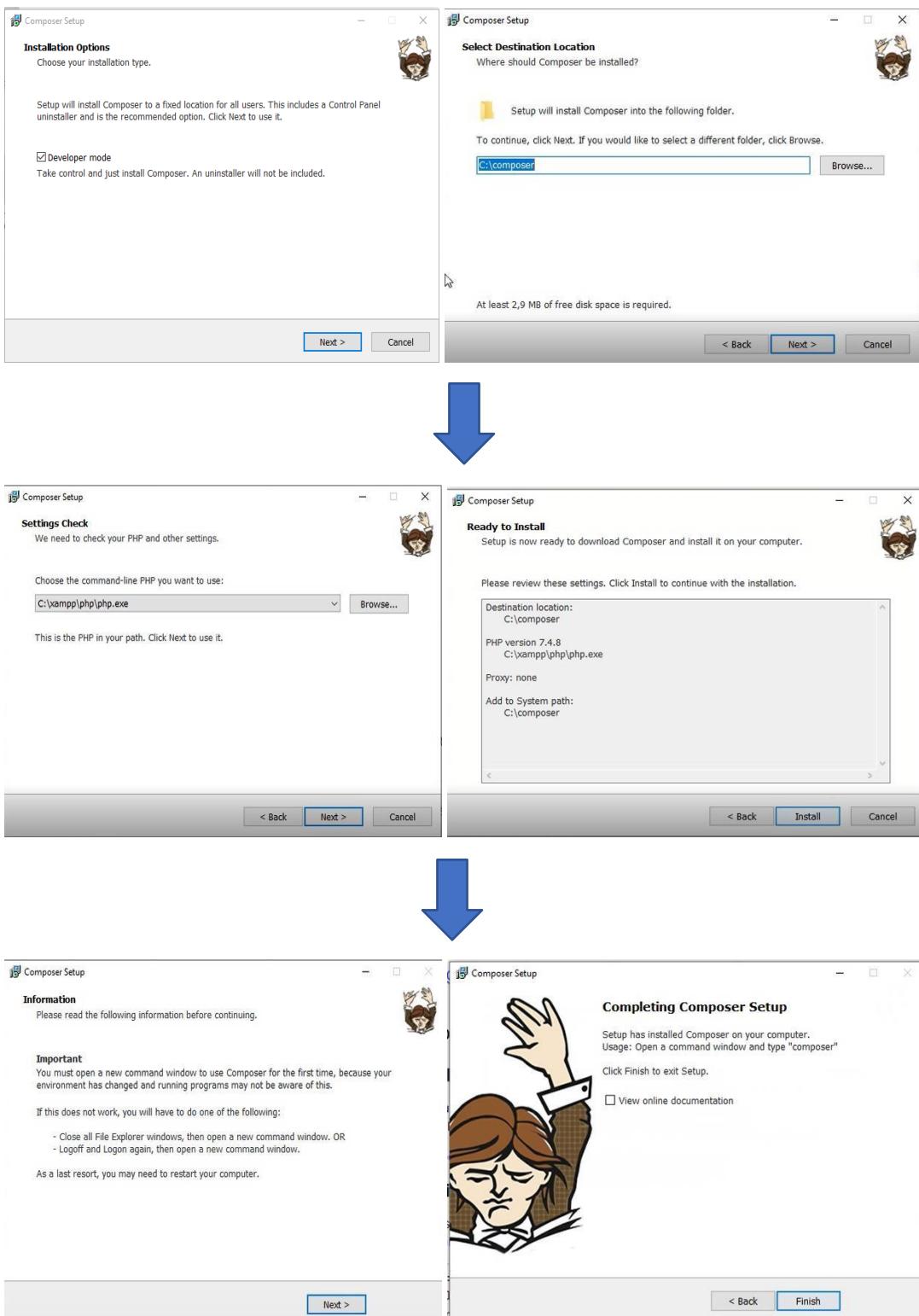
The screenshot shows the 'Download Composer' section of the Composer website. It features a 'Windows Installer' heading and instructions for setting up PATH. A callout box highlights the download link for 'Composer-Setup.exe'. Below this, a 'Command-line installation' section provides a terminal command for quick installation. A code block shows the command:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === 'e5325b19b381bfd88ce90a5ddb7823406b2a38cff6bb704b0acc')
php composer-setup.php
php -r "unlink('composer-setup.php');
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Pasos para la instalación de Composer





## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### 3. Instalar Laravel mediante Composer

Para ello nos dirigimos a la ruta **C:\xampp\htdocs** respectivamente y desde allí abrimos una consola donde ejecutaremos el siguiente comando. Esto nos permitirá corroborar que Composer fue correctamente instalado y servirá cada vez que deseemos crear un proyecto Laravel.

➤ composer create-project laravel/laravel miweb --prefer-dist

```
C:\xampp\htdocs>composer create-project laravel/laravel proyecto1 --prefer-dist
Creating a "laravel/laravel" project at "./proyecto1"
Installing laravel/laravel (v8.0.3)
- Installing laravel/laravel (v8.0.3): Downloading (100%)
Created project in C:\xampp\htdocs\proyecto1
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 106 installs, 0 updates, 0 removals
- Installing voku/portable-ascii (1.5.3): Loading from cache
- Installing symfony/polyfill-php80 (v1.18.1): Loading from cache
- Installing symfony/polyfill-mbstring (v1.18.1): Loading from cache
- Installing symfony/polyfill ctype (v1.18.1): Loading from cache
- Installing phpoftion/phpoption (1.7.5): Loading from cache
- Installing graham-campbell/result-type (v1.0.1): Downloading (100%)
- Installing vlucas/phpdotenv (v5.2.0): Downloading (100%)
- Installing symfony/css-selector (v5.1.5): Loading from cache
- Installing tijsverkoyen/css-to-inline-styles (2.2.3): Loading from cache
- Installing symfony/var-dumper (v5.1.5): Downloading (100%)
- Installing symfony/deprecation-contracts (v2.2.0): Downloading (100%)
- Installing symfony/routing (v5.1.5): Downloading (100%)
- Installing symfony/process (v5.1.5): Loading from cache
- Installing symfony/polyfill-php72 (v1.18.1): Loading from cache
- Installing paragonie/random_compat (v9.99.99): Loading from cache
- Installing symfony/polyfill-php70 (v1.18.1): Loading from cache
- Installing symfony/polyfill-intl-normalizer (v1.18.1): Loading from cache
```

Desde esa misma ruta entramos al proyecto creado y comprobamos que los ajustes quedaron listos para trabajarse. Para ello escribimos el comando

➤ php artisan

```
C:\xampp\htdocs>cd miweb
C:\xampp\htdocs\miweb>php artisan
Laravel Framework 7.22.4

Usage:
  command [options] [arguments]

Options:
  -h, --help           Display this help message
  -q, --quiet          Do not output any message
  -V, --version         Display this application version
  --ansi               Force ANSI output
  --no-ansi             Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --env[=ENV]            The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output
```

Si se desea, en ese directorio se inicializa un repositorio de Git para que se tenga un control de versiones y cambios que puedan surgir durante el proyecto.

```
C:\xampp\htdocs\miweb>git init
Initialized empty Git repository in C:/xampp/htdocs/miweb/.git/
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Desde allí listamos todos los archivos que tiene el directorio y los agregamos al repositorio.

```
C:\xampp\htdocs\curso>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: BC48-B389

Directorio de C:\xampp\htdocs\curso

06/10/2020 17:06    <DIR>      .
06/10/2020 17:06    <DIR>      ..
06/10/2020 16:58            220 .editorconfig
06/10/2020 17:03            829 .env
06/10/2020 16:58            778 .env.example
06/10/2020 16:58            111 .gitattributes
06/10/2020 16:58            163 .gitignore
06/10/2020 16:58            181 .styleci.yml
06/10/2020 16:58    <DIR>      app
06/10/2020 16:58            1.686 artisan
06/10/2020 16:58    <DIR>      bootstrap
06/10/2020 16:58            1.608 composer.json
06/10/2020 17:03            254.436 composer.lock
06/10/2020 16:58    <DIR>      config
06/10/2020 16:58    <DIR>      database
06/10/2020 16:58            974 package.json
06/10/2020 16:58            1.202 phpunit.xml
06/10/2020 16:58    <DIR>      public
06/10/2020 16:58            3.738 README.md
06/10/2020 16:58    <DIR>      resources
06/10/2020 16:58    <DIR>      routes
06/10/2020 16:58            563 server.php
06/10/2020 16:58    <DIR>      storage
06/10/2020 16:58    <DIR>      tests
06/10/2020 17:03    <DIR>      vendor
06/10/2020 16:58            559 webpack.mix.js
               14 archivos        267.048 bytes
               12 dirs   101.032.574.976 bytes libres
```

```
C:\xampp\htdocs\curso>git add -A
```

```
C:\xampp\htdocs\curso>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .editorconfig
    new file:   .env.example
    new file:   .gitattributes
    new file:   .gitignore
    new file:   .styleci.yml
    new file:   README.md
```

Realizamos el primer commit del proyecto

```
C:\xampp\htdocs\curso>git commit -m "Primer commit al proyecto Laravel"
[master (root-commit) 2d82af8] Primer commit al proyecto Laravel
 83 files changed, 10299 insertions(+)
```



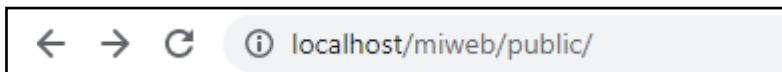
## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

4. Crear clave de nuestro proyecto la cual es una cadena de 32 caracteres que se utiliza para codificar los datos.

Con el comando php artisan key:generate logramos el objetivo.

```
C:\xampp\htdocs\miweb>php artisan key:generate
Application key set successfully.
```

5. Comprobamos que Laravel funciona correctamente, ingresando al navegador con la ruta > localhost/miweb/public



The screenshot shows a web browser window with the URL `localhost/miweb/public/` in the address bar. The page content is as follows:

Laravel

**Documentation**  
Laravel has wonderful, thorough documentation covering every aspect of the framework. Whether you are new to the framework or have previous experience with Laravel, we recommend reading all of the documentation from beginning to end.

**Laracasts**  
Laracasts offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.

**Laravel News**  
Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.

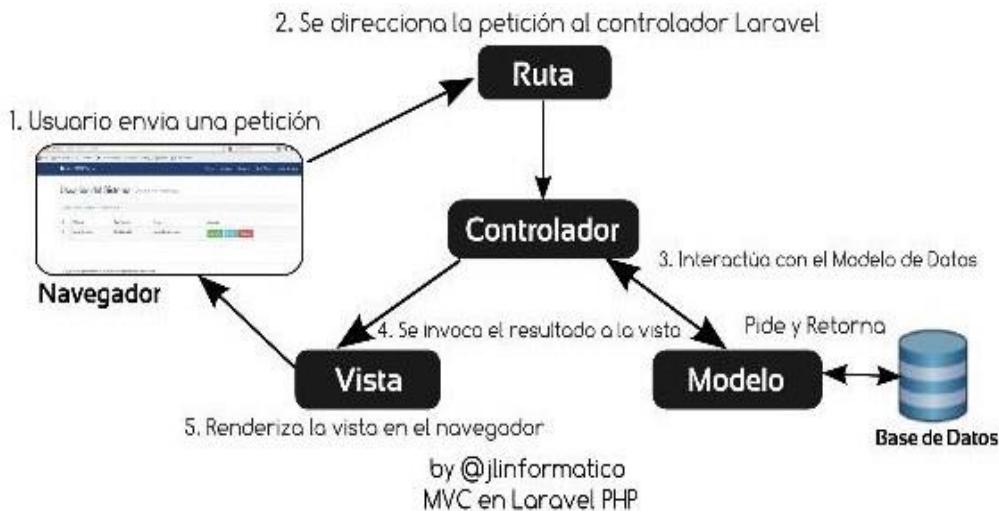
**Vibrant Ecosystem**  
Laravel's robust library of first-party tools and libraries, such as `Forge`, `Vapor`, `Nova`, and `Envoyer` help you take your projects to the next level. Pair them with powerful open source libraries like `Cashier`, `Dusk`, `Echo`, `Horizon`, `Sanctum`, `Telescope`, and more.

Shop Sponsor Build v8.6.0

### III. FUNCIONAMIENTO BÁSICO DEL FRAMEWORK

#### PATRON MVC

Laravel es un framework que trabaja la arquitectura MVC (Modelo, vista, controlador) un patrón que separa la lógica de negocio de la interfaz de usuario proporcionando flexibilidad y reutilización [3] así como también permite estructurar los componentes de un sistema software, sus responsabilidades y las relaciones existentes entre cada uno de ellos [4].



El funcionamiento básico que sigue Laravel tras una petición web a una URL de nuestro sitio es el siguiente: [5]

- Las peticiones entran a través del fichero public/index.php, el cual en primer lugar comprobará en el fichero de rutas (routes/web.php) si la URL es válida y en caso de serlo a qué controlador tiene que efectuar la petición.
- En segundo lugar, se llamará al método del controlador asignado para dicha ruta. El controlador es el punto de entrada de las peticiones del usuario, el cual, dependiendo de la petición:
  - Accederá a la base de datos (si fuese necesario) a través de los "**modelos**" para obtener datos (inserción, actualización, eliminación).
  - Tras obtener los datos necesarios los preparará para pasárselos a la vista.
- Finalmente, el controlador llamará a una vista con una serie de datos asociados, la cual se preparará para mostrarse correctamente a partir de los datos de entrada y por último se mostrará al usuario.

#### MVC en Laravel

En una aplicación web, los modelos, vistas y controladores se ubican respectivamente en los siguientes directorios facilitando así la organización del proyecto.



## Controladores

Los controladores se ubican en el directorio **app/Http/Controllers**.

Allí se crean los controladores que considere necesarios y estos estarán formados por una clase que hereda de otra llamada Controller.

Con ayuda de artisan crearemos el controlador. El comando lograr dicho objetivo es:

➤ **php artisan make:controller ClientsController**

El cual creará un archivo que tendrá el siguiente contenido

app > Http > Controllers > ClientsController.php > ...

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class ClientsController extends Controller
8  {
9
10 }
```

Para que el controlador sea funcional es necesario que este retorne arrays, objetos o vistas. Por ejemplo, utilizando el helper **view** se retornará una vista llamada main.

```
class ClientsController extends Controller
{
    public function mostrarCliente()
    {
        return view('main');
    }
}
```

## Vistas

Las vistas del proyecto estarán ubicadas en resources/views. Dentro de este directorio también se podrán incluir vistas trabajadas con el sistema de plantillas Blade. A través una ruta, el controlador podrá retornar la vista la cual es visualizada por el usuario final.

Las vistas podran tener la extension html, php o blade.php



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

```
resources > views > main.php > html > div.content
1  <!DOCTYPE html>
2  <html lang="{{ str_replace('_', '-', app()->getLocale()) }}>
3  |   <head>
4  |       <meta charset="utf-8">
5  |       <meta name="viewport" content="width=device-width, initial-scale=1">
6  |       <title>Laravel</title>
7  |       <!-- Fonts -->
8  |       <link href="https://fonts.googleapis.com/css?family=Nunito:200,600" rel="stylesheet"
9  |
10 |       <!-- Styles -->
11 |   </head>
12 |   <div class="content">
13 |       <div class="title m-b-md">
14 |           Hotel La Rivera
15 |       </div>
16 |   </div>
```

### Modelo

Los modelos se ubican en la raíz del directorio app. Allí creará un archivo el cual estará conformado por una clase que hereda de otra llamada Model.

Con ayuda de artisan crearemos el controlador. El comando lograr dicho objetivo es:

➤ **php artisan make:model Clientes**

El cual creerá un archivo que tendrá el siguiente contenido

```
app > Clientes.php > ...
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Clientes extends Model
8  {
9      //
10 }
```

Dentro del modelo se podrá manejar y acceder a la información la cual puede provenir de diversas fuentes como arreglos, bases de datos, archivos externos, etc.

### Basado en [6]

Cabe resaltar que a partir de Laravel 5, este framework incorpora más elementos que los que habitualmente cuenta un MVC, por ejemplo, Laravel cuenta con un sistema de rutas las cuales se encargan de analizar las URLs y asignarlas a un método o función, asimismo middlewares los cuales restringen ciertas áreas de la aplicación basado en el estado del usuario. [7]



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### ESTRUCTURA DE UN PROYECTO LARAVEL

La estructura de un proyecto Laravel se muestra a continuación

app	01/08/2020 15:10	Carpeta de archivos
bootstrap	01/08/2020 15:10	Carpeta de archivos
config	01/08/2020 15:10	Carpeta de archivos
database	01/08/2020 15:10	Carpeta de archivos
public	01/08/2020 15:10	Carpeta de archivos
resources	01/08/2020 15:10	Carpeta de archivos
routes	01/08/2020 15:10	Carpeta de archivos
storage	01/08/2020 15:10	Carpeta de archivos
tests	01/08/2020 15:10	Carpeta de archivos
vendor	01/08/2020 15:16	Carpeta de archivos
.editorconfig	01/08/2020 15:10	Archivo EDITORC...
.env	01/08/2020 16:18	Archivo ENV
.env.example	01/08/2020 15:10	Archivo EXAMPLE
.gitattributes	01/08/2020 15:10	Documento de te...
.gitignore	01/08/2020 15:10	Documento de te...
.styleci.yml	01/08/2020 15:10	Archivo YML
artisan	01/08/2020 15:10	Archivo
composer.json	01/08/2020 15:10	Archivo JSON
composer.lock	01/08/2020 15:15	Archivo LOCK
package.json	01/08/2020 15:10	Archivo JSON
phpunit	01/08/2020 15:10	Documento XML
README.md	01/08/2020 15:10	Archivo MD
server.php	01/08/2020 15:10	Archivo PHP
webpack.mix	01/08/2020 15:10	Archivo JavaScript

The screenshot shows the Visual Studio Code interface with the Laravel project structure. The left sidebar shows the project tree with the 'config' folder selected. The right pane shows the contents of the 'app.php' configuration file.

```
50 | // the Artisan command Line tool. You should set this to the root of
51 | // your application so that it is used when running Artisan tasks.
52 |
53 |
54 |
55 | 'url' => env('APP_URL', 'http://localhost'),
56 |
57 | 'asset_url' => env('ASSET_URL', null),
58 |
59 |
60 |
61 | // Application Timezone
62 |
63 |
64 | // Here you may specify the default timezone for your application, which
65 | // will be used by the PHP date and date-time functions. We have gone
66 | // ahead and set this to a sensible default for you out of the box.
67 |
68 |
69 | 'timezone' => 'America/Bogota',
70 |
71 |
72 | // Application Locale Configuration
73 |
74 |
75 | // The application Locale determines the default Locale that will be used
76 | // by the translation service provider. You are free to set this value
77 | // to any of the Locales which will be supported by the application.
78 |
79 |
80 |
81 |
82 |
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II

### LARAVEL 8.0

Directorio	Descripción	Subdirectorios
App	Contiene el código principal de la aplicación, como son los controladores, filtros y modelos de datos. <b>Nota:</b> En la raíz de app también podemos encontrar el fichero User.php. Este fichero es un modelo de datos que viene predefinido por Laravel para trabajar con los usuarios de la web, que incluye métodos para hacer login, registro, etc.	- <b>Console</b> - <b>Exceptions</b> - <b>Providers</b> - <b>Http:</b> Dentro de este subdirectorio se encuentra dos carpetas importantes: <ul style="list-style-type: none"><li>• <b>Controllers:</b> Contiene todos los archivos con las clases de los controladores que sirven para interactuar con los modelos, las vistas y manejar la lógica de la aplicación.</li><li>• <b>Middleware:</b> Son los filtros o clases intermedias que podemos utilizar para realizar determinadas acciones, como la validación de permisos, antes o después de la ejecución de una petición a una ruta de nuestro proyecto web.</li></ul>
Bootstrap	Se incluye el código que se carga para procesar cada una de las llamadas a nuestro proyecto. Incluye código para inicializar el framework. Se recomienda que en este archivo no se realicen cambios.	Cache
Config	Maneja archivos de configuración de la aplicación	
Database	Incluye lo relacionado a la gestión de las bases de datos del proyecto	<b>Factories</b> <b>Migrations:</b> Migraciones respectivas a la base de datos <b>Seeds:</b> Cargar datos de prueba en las tablas
Public	En este directorio pasan todas las peticiones y solicitudes. Adicionalmente se alojan los archivos CSS, JavaScript, imágenes u otros archivos que se deseen.	Contiene el archivo index.php que es con el que inicia el aplicativo.
Resources	Contiene las vistas del proyecto	<b>Views:</b> Vistas de la aplicación mediante plantillas HTML
Routes	Guarda las rutas y permite manejar el flujo de solicitudes y respuestas desde y hacia el cliente. Dichas rutas pueden ser de tipo POST, GET, PUT, DELETE o ANY	Archivos: <b>Web.php:</b> Define las rutas de la aplicación web que pueden ser ingresadas por la barra del navegador
Storage	Almacena información interna para la ejecución de la web	App Framework Logs
Test	Contiene archivos relacionadas con pruebas automatizadas	Feature Unit
Vendor	Almacena las librerías y dependencias que conforman a Laravel	

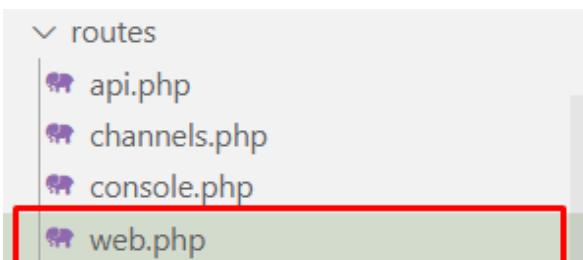
Tomado de [5]



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### IV. MANEJO DE RUTAS

Para crear una ruta en nuestra aplicación es necesario dirigirse al archivo web.php ubicado en el directorio routes.



Las rutas, en su forma más sencilla, pueden devolver directamente un valor desde el propio fichero de rutas, pero también podrán generar la llamada a una vista o a un controlador.

Si una ruta no ha sido definida saldrá el siguiente error:

404 | Not Found

Las rutas de la aplicación pueden recibir peticiones que pueden ser de tipo GET o POST.

	Tipo	Nombre de la ruta
Route::get('/', function () { // Retorna un mensaje en la ruta asignada return "Hola Mundo"; });	get	Hola Mundo

Este código se lanzaría cuando se realice una petición tipo GET a la ruta raíz de nuestra aplicación. Si se está trabajando bajo modo local la ruta <http://localhost/miweb/public> será donde se visualizará el resultado.

← → C ⓘ localhost/miweb/public/  
Apache Spark ArangoDb SGBD CONGRESOS LARAVEL  
Hola Mundo



## RUTAS QUE REFERENCIAN A UNA VISTA

Para cargar una vista se debe indicarle la ruta donde se va a cargar.

En este caso en la raíz del proyecto se cargará la vista denominada main.

```
Route::get('/', function () {
    // Carga la vista llamada main.php en la ruta principal
    return view('main');
});
```

Para el siguiente caso en la ruta hotel se cargará la vista denominada información.

```
Route::get('hotel', function () {
    // Carga la vista informacion.php en la ruta
    // http://localhost/miweb/public/hotel
    return view('informacion');
});
```

**Si las vistas están organizadas dentro de directorios**, es necesario utilizar la notación tipo "dot", en la que las barras que separan las carpetas se sustituyen por puntos.

En este primer ejemplo se tiene la vista denominada registro.php la cual está dentro de la carpeta clientes.

```
views
  clientes
    registro.php
```

La forma en cómo debería relacionar la ruta con la vista respectiva se visualiza a continuación:

```
Route::get('clientes', function () {
    // Carga la vista registro.php la cual esta
    // ubicada dentro del directorio view/clientes
    return view('clientes.registro');
});
```

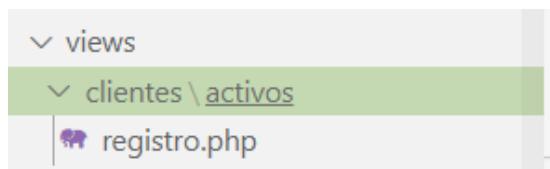
La url escrita en el navegador para acceder al contenido de dicha vista sería:

① [localhost/miweb/public/clientes](http://localhost/miweb/public/clientes)



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

El siguiente ejemplo muestra una vista ubicada en el directorio clientes/activos/registro.php



De la misma forma anterior, para enlazar la ruta con la vista respectiva seria así:

```
Route::get('clientes', function () {
    // Carga la vista registro.php la cual esta
    // ubicada dentro del directorio view/clientes/activos
    return view('clientes.activos.registro');
});
```

### RUTAS CON PARAMETROS

Si se desea añadir parámetros a una ruta simplemente se debe añadirlos entre llaves {} de la forma como se visualiza en el siguiente ejemplo:

```
Route::get('servicios/{nombre}', function ($nombre) {
    return 'Nombre: ' . $nombre;
});
```

La url escrita en el navegador seria:

① localhost/miweb/public/servicios/**restaurante** → Parámetro

#### Ejemplo # 2:

En el siguiente ejemplo se observa la ruta usuarios la cual tiene un parametro denominado **id**. Para retornar el mensaje se puede concretar de dos distintas formas:

Opción 1:

```
Route::get('/usuarios/{id}', function ($id) {
    return 'Mostrando detalle del usuario: ' . $id;
});
```

Opción 2:

```
Route::get('/usuarios/{id}', function ($id) {
    return "Mostrando detalle del usuario: {$id}";
});
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Cuyo resultado es:

← → ⌂ ⓘ localhost/curso/public/usuarios/3

Mostrando detalle del usuario: 3

Ahora bien, si dado el caso que se eligiera una ruta como la que se muestra a continuación

```
Route::get('/usuarios/nuevo', function () {
    return "Crear usuario nuevo";
});
```

Lo anterior no arroja el mensaje que se tiene estipulado en la ruta usuarios/nuevo, sino que genera lo siguiente:

← → ⌂ ⓘ localhost/curso/public/usuarios/nuevo

Mostrando detalle del usuario: nuevo

Esto se debe a que Laravel toma la primera ruta que coincide y las demás las desecha

```
Route::get('/usuarios/{id}', function ($id) {
    return "Mostrando detalle del usuario: {$id}";
});

Route::get('/usuarios/nuevo', function () {
    return "Crear usuario nuevo";
});
```

Para solucionar dicho problema se debe poner una condición a la ruta especificando las restricciones que desea considerar.

```
Route::get('/usuarios/{id}', function ($id) {
    return "Mostrando detalle del usuario: {$id}";
})->where('id', '[0-9]+');
```

Por ejemplo, para la ruta usuarios seguido de un parámetro de tipo número, se va a visualizar el mensaje “Mostrando detalle del usuario \$id; gracias a la condición **where** que indica que va ser un número del 0 al 9 y que dicho número podrá tener más de una cifra..

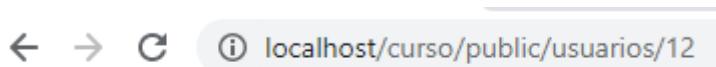
```
)->where('id', '[0-9]+');
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

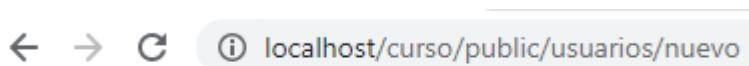
Ahora bien, el resultado en el navegador se visualiza correctamente;

Con la ruta Usuarios/12



Mostrando detalle del usuario: 12

Con la ruta Usuarios/nuevo



Crear usuario nuevo

Otra solución al respecto se podría lograr cambiando el orden de las rutas, ya que la primera coincide con la ruta nuevo, y luego la siguiente sería por él envío de un parámetro.

```
Route::get('/usuarios/nuevo', function () {
    return "Crear usuario nuevo";
});

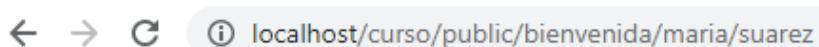
Route::get('/usuarios/{id}', function ($id) {
    return "Mostrando detalle del usuario: {$id}";
});
```

### PARAMETROS OPCIONALES

Si deseamos que el parametro sea opcional podemos agregar **?** al finalizar el nombre del parametro en la ruta respectiva. Por ejemplo en una ruta con dos parametros de los cuales uno es opcional se puede presentar algo como lo siguiente:

```
Route::get('/bienvenida/{nombre}/{apellido?}', function ($nombre, $apellido = null) {
    if ($apellido){
        return "Bienvenido {$nombre}, tu apellido es {$apellido}";
    } else{
        return "Bienvenido {$nombre}";
    }
});
```

Y el resultado en el navegador seria: Con dos parametros en la ruta bienvenida



Bienvenido maria, tu apellido es suarez



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Con un solo parametro en la ruta bienvenida.

Bienvenido maria

### RUTAS QUE REFERENCIAN A UN CONTROLADOR

Tal y como se mencionaba al inicio de esta unidad, las rutas también pueden retornar a un controlador el cual este a su vez redirecciona a la vista, siendo así la forma correcta en el que se va a trabajar y la cual está bajo el modelo MVC.

Route::get('clientes/visualizar', 'ClientsController@mostrarCliente');			
Tipo	Ruta	Nombre del controlador	Nombre del método o función dentro del controlador
Petición			dentro del controlador

Más adelante se abarcará detalladamente el uso de controladores en una aplicación Laravel.

### LISTAR LAS RUTAS EXISTENTES EN NUESTRA APLICACIÓN

Con el siguiente comando se puede visualizar el listado de las rutas que han sido creadas en nuestra aplicación: > **php artisan route:list**

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	administracion		App\Http\Controllers\Administrador\AdminController@administracion	web
	GET HEAD	api/user		Closure	api
	GET HEAD	blade		Closure	auth:api
	GET HEAD	clientes		Closure	web
	GET HEAD	clientes/visualizar		App\Http\Controllers\clientsController@mostrarCliente	web
	GET HEAD	detalle		Closure	web
	GET HEAD	hotel		Closure	web
	GET HEAD	servicios/{nombre}		Closure	web



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### V. VISTAS

Las vistas son el mecanismo donde se podrá visualizar el contenido de la aplicación en el sitio web. El usuario ve e interactúa con la aplicación a través de la vista. Los datos se preparan y se muestran en la vista mediante etiquetas HTML. Las vistas del proyecto estarán ubicadas en resources/views y se almacenan con la extensión .php



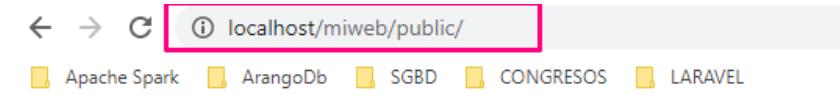
A continuación, se muestra un pequeño ejemplo de una vista denominada main.php ubicada en el directorio views.

```
EXPLORER ... main.php X servicios.php web.php AdminController.php dashboard.php
OPEN EDITORS MIWEB
resources > views > main.php
resources > views > main.php
2 <html lang="{{ str_replace('_', '-', app()->getLocale()) }}>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>La Rivera</title>
7   </head>
8   <body>
9     <div class="content">
10       <div class="title m-b-md">
11         <h1> Hotel La Rivera </h1>
12       </div>
13       <p> El hotel de tus sueños </p>
14     </div>
15   </body>
16 </html>
17
```

Desde el archivo web.php se define la ruta que hará el llamado a dicha vista

```
Route::get('/', function () {
    // Retorna un mensaje en la ruta asignada
    return view('main');
});
```

Finalmente, el resultado se observa en el navegador.



## Hotel La Rivera

El hotel de tus sueños



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### PASAR DATOS A UNA VISTA

En este caso se define la vista llamada datos.php la cual se devuelve cuando se realiza una petición de tipo GET a la ruta *visualizar* del sitio web.

Adicionalmente la vista recibe datos provenientes de un arreglo tal y como se muestra en la siguiente sección:

```
Route::get('visualizar', function () {
    return view('datos', array('nombre' => 'Javi',
                               'edad' => 23,
                               'telefono' => '7201910'));
});
```

La información en la vista se organizaría así:

```
resources > views > datos.php
1  <html>
2      <h2> Bienvenido </h2>
3
4      <h3> Tu nombre es: <?php echo $nombre; ?> </h3>
5      <h3> Tu edad es: <?php echo $edad; ?> </h3>
6      <h3> Tu telefono es: <?php echo $telefono; ?> </h3>
7
8  </html>
```

Y el resultado se muestra en el navegador:



**Bienvenido**

**Tu nombre es: Javi**

**Tu edad es: 23**

**Tu telefono es: 7201910**

Ahora bien si se tiene una ruta que recibe parametros y deseamos que estos se muestren en la vista se haria lo siguiente:

```
// Rutas con parametros y enviando a la vista
Route::get('servicios/{nombre}', function ($nombre) {
    return view('servicios', array('nombreS' => $nombre));
});
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Laravel además ofrece una alternativa que crea una notación un poco más clara. En lugar de pasar un array como segundo parámetro se puede emplear el método with para indicar una a una las variables o contenidos que queremos enviar a la vista:

```
Route::get('servicios/{nombre}', function ($nombre) {
    return view('servicios')
        ->with('nombres', $nombre);
});
```

En la vista se tendría algo como esto:

```
<html>
    <h1> Servicios </h1>
    <?php echo $nombres ?>
</html>
```

Y el resultado final sería:

The screenshot shows a browser window with the address bar containing "localhost/miweb/public/servicios/cafeteria". The main content area displays the word "Servicios" in large bold letters, followed by the word "cafeteria" in a smaller font.

## Servicios

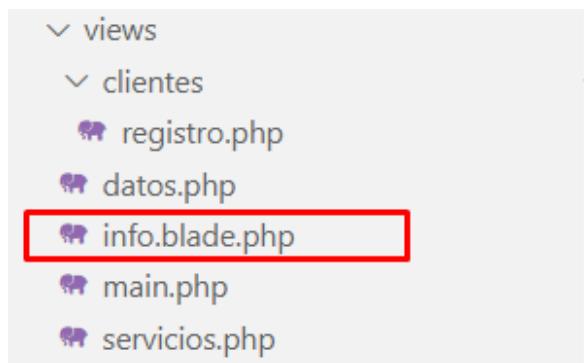
cafeteria

### USO DE BLADE

#### Crear una Vista usando una plantilla de Blade

Laravel utiliza Blade para la definición de plantillas en las vistas. Esta librería permite realizar todo tipo de operaciones con los datos, además de la sustitución de secciones de las plantillas por otro contenido, herencia entre plantillas, definición de layouts o plantillas base, etc.

Para crear una vista usando Blade se debe guardarla con el nombre de la plantilla seguida de la extensión Blade.php





## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Luego en el listado de rutas lo llamamos así:

```
Route::get('blade', function () {
    // En la url http://localhost/miweb/public/blade
    // Carga la vista info.blade.php , realizada en blade
    return view('info', array('name' => 'Josefina'));
});
```

La vista tiene el siguiente contenido:

Sin usar BLADE
<pre>Hola &lt;?php echo \$name ?&gt; &lt;br&gt; &lt;?php for (\$i = 0; \$i &lt; 10; \$i++) : ?&gt;     &lt;p&gt; &lt;?php echo 'El valor actual es: '. \$i ?&gt; &lt;/p&gt; &lt;?php endfor;?&gt; &lt;br&gt;</pre>
Usando BLADE
<pre>Hola {{ \$name }}. &lt;br&gt; @if (\$i = 0; \$i &lt; 10; \$i++)     El valor actual es {{ \$i }}     &lt;br&gt; @endif</pre>

El resultado en el navegador

The screenshot shows a browser window with the URL `localhost/miweb/public/blade`. The page content is:

```
Hola Josefina.
El valor actual es 0
El valor actual es 1
El valor actual es 2
El valor actual es 3
El valor actual es 4
El valor actual es 5
El valor actual es 6
El valor actual es 7
El valor actual es 8
El valor actual es 9
```

Below the browser window, there are three small icons: Apache Spark, ArangoDb, and SGBD.



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Manejo de Layouts

Blade también nos permite la definición de layouts para crear una estructura HTML base con secciones que serán rellenadas por otras plantillas o vistas hijas. Por ejemplo, podemos crear un layout con el contenido principal o común de nuestra web (head, body, etc.)

Para invocar a las plantillas no es necesario colocar la ruta completa, ya que Laravel por defecto buscará el archivo dentro del directorio resources/views:

Función de Blade	Descripción
@include('view')	Incluye una plantilla dentro de otra. Podemos usar múltiples directivas @include dentro de una misma plantilla de Blade.
@extends('layouts.master')	Extienda el layout que se ha creado en layouts/master.blade.php
@section('view') @endsection	Permite ir añadiendo contenido en las plantillas hijas o secciones a la página web.
@parent	Carga en la posición indicada el contenido definido por el padre para dicha sección
@yield @yield('section', 'Contenido por defecto')	Permite sustituir por el contenido que se indique o Permite establecer un contenido por defecto mediante su segundo parámetro Es usada para mostrar el contenido de una sección específica.

### CREAR UN ESTILO A LAS VISTAS CON BOOTSTRAP

Para generar un estilo base a las vistas, es necesario crear una vista de tipo Blade la cual deberá tener la extensión .blade.php y ubicarse en el directorio resources/views.

The screenshot shows the file structure of a Laravel project in VS Code. The 'resources' folder contains 'views' which in turn contains 'main.blade.php'. The code editor displays the following content:

```
<!doctype html>
<html lang="en">
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <title>Hotel la Rivera</title>
    </head>
    <body>
    </body>
</html>
```

The file 'main.blade.php' is highlighted with a red rectangle at the bottom of the list.

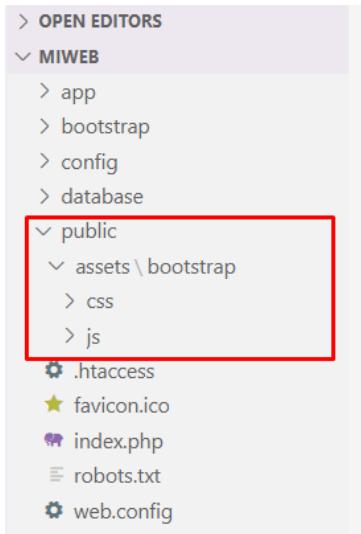
En segundo lugar, se deberá incluir la librería Bootstrap para ello es necesario acceder a la web "<http://getbootstrap.com/>" y descargarla desde ese sitio.



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

The screenshot shows the Bootstrap Documentation page for version 4.5.2. The left sidebar has a 'Download' section with a 'Download' button. The main content area is titled 'Download' and describes how to get the compiled CSS and JavaScript. It includes a bulleted list of what's included and a note about dependencies. A 'View on GitHub' button is at the top right.

Esto bajará un fichero zip comprimido con tres carpetas (*js*, *css* y *fonts*) y se extraerá en la carpeta "public/assets/bootstrap" (para ello es necesario crear las carpetas "/assets/bootstrap"). Cabe resaltar que el comprimido no incluye archivos sobre dependencias de Javascript como JQuery y Popper.js.



A continuación, se prosigue a enlazar dichos archivos al proyecto que se esté trabajando. Las rutas para la carga de los assets (css y js) se han almacenado en modo local. Para generar la ruta completa y que encuentre los recursos tendremos que escribir los siguientes comandos empleando el helper **url()**:

Para estilos:

```
<link href="{{ url('/assets/bootstrap/css/bootstrap.min.css') }}" rel="stylesheet">
```

Para archivos JavaScript:

```
<script type="text/javascript" src="{{ url('/assets/bootstrap/js/bootstrap.min.js') }}></script>
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

El resultado final en la vista sería el siguiente:

```
resources > views > main.blade.php
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
6      | <!-- Bootstrap CSS -->
7      | <!-- Invoca la ruta del archivo css en el directorio local usando Blade -->
8      | <link href="{{ url('/assets/bootstrap/css/bootstrap.min.css') }}" rel="stylesheet">
9      <title>La Rivera</title>
10 </head>
11 <body>
12 | <h1> Hotel La Rivera </h1>
13 | <!-- Optional JavaScript -->
14 | <!-- jQuery first, then Popper.js, then Bootstrap JS -->
15 | <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPHhPYtFLRl4wPQ+HxGhFJH0/tXu0Df+urqf+HnI" crossorigin="anonymous">
16 | <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js" integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81Po53l4lexwVlA5MqRxXxiVbGQ&lt;" crossorigin="anonymous">
17 | <script type="text/javascript" src="{{ url('/assets/bootstrap/js/bootstrap.min.js') }}></script>
18 </body>
19 </html>
```

Ahora bien si deseamos corroborar que Bootstrap se incorporó correctamente en el proyecto se añadira en la vista un componente del framework.

```
<h1> Hotel La Rivera </h1>
<div class="alert alert-primary" role="alert">
| | El hotel de tus sueños
</div>
```

Por otra parte, si se quiere agregar más archivos bien sea de tipo css o javascript se hara de la misma manera. Por ejemplo si deseamos agregar una fuente de Google Fonts se obtendrá algo como esto:

```
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
| <!-- Bootstrap CSS -->
| <!-- Invoca la ruta del archivo css en el directorio local usando Blade --> |
| <link href="https://fonts.googleapis.com/css2?family=Bangers&display=swap" rel="stylesheet">
| <link href="{{ url('/assets/bootstrap/css/bootstrap.min.css') }}" rel="stylesheet">
| <link href="{{ url('/assets/bootstrap/css/mystyle.css') }}" rel="stylesheet">

<title>La Rivera</title>
</head>
```

Donde el archivo mystyle.css contiene:

```
public > assets > bootstrap > css > # mystyle.css > h1
1  h1{
2      font-family: 'Bangers', cursive;
3
4 }
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

El resultado en el navegador se muestra a continuación:

← → ⌂ ⓘ localhost/miweb/public/

# HOTEL LA RIVERA

El hotel de tus sueños

## INCLUIR IMÁGENES A LAS VISTAS

```
✓ public
  ✓ assets
    > bootstrap
  ✓ imagenes
    📸 3.jpg
```

De la misma forma que los estilos para insertar una imagen en la vista, esta deberá ubicarse en el directorio assets/imagenes y luego agregarla así:

```
<img src= "{{ url('/assets/imagenes/3.jpg') }}">
```

## ACTIVIDAD # 1

Realizar el ejercicio práctico relacionado con el manejo de un video club y luego en base a los conceptos abarcados durante esta primera unidad desarrollar el primer taller individual.



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### VI. CONTROLADORES, FILTROS Y FORMULARIOS

Los controladores se almacenan en ficheros PHP en la carpeta app/Http/Controllers y normalmente se les añade el sufijo Controller.

Para crear un controlador se usa el comando

➤ **php artisan make:controller NameController**

```
PS C:\xampp\htdocs\miweb> php artisan make:controller ClientsController
```

Este comando creará el controlador ClientsController dentro de la carpeta app/Http/Controllers tal y como se muestra a continuación.

The screenshot shows a code editor with the following structure:

- EXPLORER**: Shows the project structure:
  - > OPEN EDITORS
  - < MIWEB
    - < app
    - > Console
    - > Exceptions
    - < Http
      - < Controllers
        - ClientsController.php** (highlighted with a red border)
        - Controller.php
        - > Middleware
        - Kernel.php
        - > Providers
        - User.php
- ClientsController.php**:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class ClientsController extends Controller
8 {
9     //
10    public function mostrarCliente()
11    {
12        return view('info', array('name' => 'Marcela'));
13    }
14 }
```
- Other files: main.php, servicios.php, web.php, inf.php

Una vez definido un controlador ya podemos asociarlo a una ruta. Para esto tenemos que modificar el fichero de rutas routes/web.php.

En versiones inferiores a Laravel 8.0 la forma como se llevaba a cabo era la siguiente.

```
Nombre del Controlador  Nombre del método
Route::get('clientes/visualizar', 'ClientsController@mostrarCliente');
```

En Laravel 8.0 hubo una modificación y la manera de realizar dicha acción es:

Agregando en el encabezado de routes/web.php

```
use App\Http\Controllers\ClientsController;
Nombre del Controlador  Nombre del método
Route::get('clientes/visualizar', [ClientsController::class, 'mostrarCliente']);
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

También podemos crear sub-carpetas dentro de la carpeta Controllers para una mejor organización. En este caso, la estructura de carpetas que creemos no tendrá nada que ver con la ruta asociada a la petición y, de hecho, a la hora de hacer referencia al controlador únicamente tendremos que hacerlo a través de su espacio de nombres.

Por ejemplo, para crear un controlador dentro de un subdirectorio se haría así:

```
PS C:\xampp\htdocs\miweb> php artisan make:Controller Administrador/AdminController
Controller created successfully.
```

```
app > Http > Controllers > Administrador > AdminController.php > ...
1  <?php
2
3  namespace App\Http\Controllers\Administrador;
4
5  use App\Http\Controllers\Controller;
6  use Illuminate\Http\Request;
7
8  class AdminController extends Controller
9  {
10     //
11     public function administracion(){
12         return 'Dashboard';
13     }
14 }
```

Y para enlazar la ruta al controlador se hará de la siguiente forma en el archivo routes/web.php:

```
use App\Http\Controllers\Administrador\AdminController;
Route::get('principal', [AdminController::class, 'administracion']);
```

Nombre del Controlador      Nombre del método  
↑                             ↑



## CONTROLADORES Y MANEJO DE VISTAS

Ahora bien, si queremos que el controlador en vez de retornar una cadena de texto, retorne una vista es necesario hacerlo de esta manera:

```
class AdminController extends Controller
{
    //
    public function administracion(){
        return view('bienvenida');
    }
}
```

Si deseamos ahora enviar datos a nuestra vista, el controlador se encargará de asumir dicho proceso para ello:

```
class AdminController extends Controller
{
    //
    public function administracion(){
        $categorias = [
            'musica',
            'entretenimiento',
            'deportes',
            'cine',
        ];
        return view('bienvenida', ['category' => $categorias]);
    }
}
```

Diagrama explicativo:

- Arreglo con los elementos: Se refiere al array \$categorias.
- Nombre de la vista: Se refiere a la variable `'bienvenida'`.
- Identificador: Se refiere a la variable `'category'`.
- Nombre de arreglo: Se refiere a la variable `$categorias`.

En la vista se mostrará la información de la siguiente manera:

```
resources > views >  bienvenida.php
1   <h1> Hola bienvenido </h1>
2   <ul>           Identificador
3   |       <?php foreach ($category as $c): ?>
4   |       |   <li> <?php echo $c ?> </li>
5   |       <?php endforeach ; ?>
6   </ul>
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Nota: Si dentro del arreglo existe código de javascript o de html eso puede ser riesgoso para el sitio web, por ende, para escapar código de ese tipo es posible usar el helper `e` el cual sirve para cumplir dicho objetivo. Así:

```
<ul>
    <?php foreach ($category as $c): ?>
        <li> <?php echo e($c) ?> </li>
    <?php endforeach ; ?>
</ul>
```

Si deseamos enviar desde el controlador más de un parámetro a la vista, se puede lograr de la siguiente forma.

Controlador	Opción #2 Usando la función with de Laravel
<b>Opcion #1 : Usando arreglo</b>  <pre>class AdminController extends Controller {     //     public function administracion(){         \$categorias = [             'musica',             'entretenimiento',             'deportes',             'cine',         ];          \$titulo = "Listado de categorias";         return view('bienvenida',             [                 'category' =&gt; \$categorias,                 'title' =&gt; \$titulo             ]);     } }</pre>	<b>Opción #2 Usando la función with de Laravel</b>  <pre>public function administracion(){     \$categorias = [         'musica',         'entretenimiento',         'deportes',         'cine',     ];      \$titulo = "Listado de categorias";      return view('bienvenida')         -&gt;with('category', \$categorias)         -&gt;with('title', \$titulo); }</pre>
<b>Vista</b>  <pre>&lt;h1&gt; Hola bienvenido &lt;/h1&gt; &lt;h3&gt; &lt;?= \$title ?&gt; &lt;/h3&gt; &lt;ul&gt;     &lt;?php foreach (\$category as \$c): ?&gt;         &lt;li&gt; &lt;?php echo e(\$c) ?&gt; &lt;/li&gt;     &lt;?php endforeach ; ?&gt; &lt;/ul&gt;</pre>	



## GENERAR URLs EN LARAVEL

### HELPER URL:

Con este helper se permite enlazar a rutas arbitrarias dentro de la aplicación.

Para generar la URL que apunte a una ruta que redirecciona a un controlador podemos usar el helper URL. La vista donde va a estar dicho enlace debe ser blade.php

```
resources > views > dashboard.blade.php
1
2  <h1> DASHBOARD DE ADMINISTRACIÓN </h1>
3
4  <a href="{{ url('/principal') }}"> Enlace </a>
5
      Nombre de la Ruta
```

Esto implica que la plantilla dashboard.blade tiene un enlace que apunta directamente a la ruta “principal” la cual redirecciona a la función administración del controlador Administrador\AdminController.

```
Route::get('principal', [AdminController::class, 'administracion']);
```

### Ejemplo:

Para este ejemplo vamos a iniciar con la Creación de una ruta que redirecciona a un controlador. Esta configuración se realiza desde el archivo routes/web.php

```
use App\Http\Controllers\Administracion\AdminController;
```

```
Route::get('listado', [AdminController::class, 'usuarios']);
```

Dentro del Controlador Administración se va a crear el método **usuarios** el cual contendrá un arreglo con los datos de tres usuarios, los cuales se van a pasar a la vista llamada **listado**.

```
public function usuarios(){
    $users = [
        array('id'=> '1', 'nombre' => 'Maria', 'edad' => 24),
        array('id'=> '2', 'nombre' => 'Jesus', 'edad' => 29),
        array('id'=> '3', 'nombre' => 'Sebastian', 'edad' => 21)
    ];
    return view('listado', compact('users'));
}
```



# **SEMINARIO DE COMPUTACIÓN E INFORMATICA II**

## **LARAVEL 8.0**

La vista usuarios tendrá algo como lo siguiente:

```
resources > views > 🏷 listado.blade.php

1  <ul>
2      @foreach ($users as $user)
3          <li>
4              {{ $user['nombre'] }} Visualizo El nombre del usuario
5              <a href= '{{url("/usuarios/$user[id]/$user[nombre]/$user[edad]")}}>Ver detalles</a>
6          </li>
7      @endforeach
8  </ul>
```

Envio el resto de datos del usuario a través de la nueva ruta

En la vista `usuarios.blade.php` se observa el listado de los nombres de los usuarios que fueron enviados a través del controlador. Cada nombre tiene una etiqueta de tipo enlace `<Ver detalles>` la cual redirecciona a una nueva ruta que envía como parámetro todos los datos de ese usuario seleccionado. Para lograr el objetivo se usa el helper `url`.

← → C ⓘ localhost/cursito/public/listado

- Maria [Ver detalles](#)
  - Jesus [Ver detalles](#)
  - Sebastian [Ver detalles](#)

Dentro del archivo web.php se define la nueva ruta con todos los parámetros que se envían la cual redirecciona a la función *detalle* creada en el controlador AdminController y será este método el que retornará a la vista asignada.

```
Route::get('/usuarios/{id}/{nombre}/{edad}', [AdminController::class, 'detalle']);
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

El resultado en la vista se visualiza a continuación

```
resources > views > 🌐 detalle.blade.php
1  <h2> Información detallada del usuario </h2>
2  <h3> Id: {{ $id }} </h3>
3  <h3> Nombre: {{ $nombre }} </h3>
4  <h3> Edad: {{ $edad }} </h3>
```

Y en el navegador se hará lo siguiente:

The screenshot shows a web browser window. The address bar displays "localhost/cursito/public/listado". Below the address bar, there is a navigation bar with links: "Aplicaciones", "Google", "MAESTRIAS", and "PARADIGMAS". The main content area shows a list of users with their names and a "Ver detalles" link:

- Luisa [Ver detalles](#)
- Maria [Ver detalles](#)
- Sonia [Ver detalles](#)
- Jesus [Ver detalles](#)

A large blue downward-pointing arrow is positioned below the user list, indicating a transition to the next screen. The second part of the screenshot shows the browser after clicking on the "Ver detalles" link for "Maria". The address bar now displays "localhost/cursito/public/usuarios/1/Maria/24". The navigation bar remains the same. The main content area now shows the detailed information for the selected user:

## Información detallada del usuario

**Id:** 1

**Nombre:** Maria

**Edad:** 24

## Información detallada del usuario

**Id:** 1

**Nombre:** Maria

**Edad:** 24



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Propiedades del Helper URL

Propiedad	Descripción
<code>url() -&gt;current();</code>	Obtiene la actual url sin la cadena de consulta
<code>url() -&gt;full()</code>	Obtiene la actual url incluyendo la cadena de consulta
<code>url() -&gt;previous();</code>	Obtiene la url previa de la consulta anterior.

### HELPER ACTION

A diferencia del helper URL, ACTION apunta a un controlador y una acción y Laravel nos retorna una url.

### HELPER ROUTE

El helper Route genera una URL para la ruta con un nombre asignado.

Por ejemplo, en el archivo web.php se asigna un nombre a la ruta que se desea con la propiedad name.

```
Route::get('listado', [AdminController::class, 'usuarios'])->name('usuarios.listado');
```

Luego simplemente en la vista se trabaja de la siguiente forma

```
<a href="{{ route('usuarios.listado') }}"> Regresar </a>
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### VII. USO DE FORMULARIOS EN LARAVEL

Para crear un formulario dentro de una vista en Laravel es bastante sencillo.

Lo primero que vamos a realizar es crear la ruta que redirecciona al controlador que retornara a la vista respectiva. Esto se hará en el archivo routes/web.php

```
use App\Http\Controllers\ProductosController;

Route::get('productos/registro', [ProductosController::class, 'form_registro'])
->name('formulario_registro');
```

Como se observa la ruta “*productos/registro*” es de tipo *get* la cual redirecciona a la función *form\_registro* del controlador *ProductosController*. Adicionalmente con la propiedad *name*, se le asigno un nombre a dicha ruta denominada ‘*formulario\_registro*’.

La función *form\_registro* del controlador tiene el siguiente contenido:

```
public function form_registro(){
    return view ('productos.form_registro');
```

Y la vista a la que invoca:

```
@extends('main')

@section('content')
    <h1> Registro de Productos </h1>
    Url a la que redirecciona
    <form action="{{ url('productos/registro') }}" method="POST">
        @csrf
        Nombre <input type="text" id='nombre' name='nombre'>
        <button type="submit"> Aceptar </button>
    </form>
@stop
```

Metodo de envio de datos

↑

Si nos damos cuenta la vista retorna la misma ruta con la que fue invocada pero la diferencia radica en que esta ruta es de tipo Post, redirecciona a otra función dentro del Controlador llamada ‘*registrar*’.



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

```
Route::post('productos/registro', [ProductosController::class, 'registrar'])  
->name('registro_productos');
```

Como hemos visto anteriormente, en Laravel podemos definir distintas acciones para procesar peticiones realizadas a una misma ruta, pero usando un método distinto (GET, POST, PUT, DELETE). El ejemplo anterior fue evidencia de ello, de esta forma cada ruta apuntará a un método distinto de un controlador y nos facilitará la separación del código.

Adicionalmente es importante que el formulario que vayamos a realizar incluya el token CSRF. Con blade esto es fácil, solo debes colocar `@csrf` dentro del form. Esto se hace con el fin de evitar el error 419 de Laravel que dice Pagina Expirada. [8]

### Nota:

#### Protección contra CSRF

CSRF (acrónimo de Cross-site request forgery) es un método por el cual un usuario malintencionado intenta hacer que tus usuarios, sin saberlo, envíen datos que no quieren enviar. Afortunadamente, los ataques CSRF se pueden prevenir añadiendo un token CSRF a tus formularios. [9]

Laravel proporciona una forma fácil de protegernos de este tipo de ataques. Simplemente tendremos que llamar al método `@csrf` después de abrir el formulario como lo vimos anteriormente.

```
<form action="{{ url('productos/registro') }}" method="POST">  
    @csrf  
    Nombre <input type="text" id='nombre' name='nombre'>  
    <button type="submit"> Aceptar </button>  
  
</form>
```

Token Obligatorio que debe incluir en los formularios

Finalmente, la función registrar dentro del Controlador muestra el siguiente contenido

```
public function registrar(){  
    return view ('productos.registrar');  
}
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Y la vista mostrará por ahora un simple mensaje

resources > views > productos > registrar.blade.php

```
1  @extends('main')
2
3  @section('content')
4  <h1> Producto Registrado </h1>
5
6  @stop
```

The screenshot shows a browser window with the URL `localhost/cursito/public/productos/registro`. The page has a blue header bar with navigation links: "Supermercado", "Productos", and "Usuarios". The main content area has a title "Registro de Productos" and a form field labeled "Nombre" with an empty input box. A button labeled "Aceptar" is highlighted with a blue border.

The screenshot shows a browser window with the URL `localhost/cursito/public/productos/registro`. The page has a blue header bar with navigation links: "Supermercado", "Productos", and "Usuarios". The main content area displays the message "Producto Registrado" in large, bold text.



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II

### LARAVEL 8.0

#### Elementos de un formulario

En todos los tipos de campos que se vayan a emplear para recoger datos dentro del formulario es importante añadir sus atributos **name** e **id**, ya que nos servirán después para recoger los valores rellenados por el usuario.

#### Campos de tipo Input

Tipo	Descripción	Ejemplo
Texto	Para crear un campo de texto usamos la etiqueta de HTML input, para la cual tenemos que indicar el tipo <b>text</b> y su nombre e identificador respectivo.	Aquí se visualiza un elemento de tipo texto el cual tiene un identificador y un nombre. <code>&lt;input type="text" id='nombre' name='nombre'&gt;</code> Para el siguiente ejemplo se visualiza un campo del mismo tipo, pero tiene un valor por defecto asignado (ítem value). <code>&lt;input type="text" name="apellido" id="apellido" value="Apellido"&gt;</code> Desde una vista con Blade podemos asignar el contenido de una variable (en el ejemplo \$apodo) para que aparezca el campo de texto con dicho valor. La variable apodo puede ser un parámetro enviado a través del controlador a la vista. <code>&lt;input type="text" name="apodo" id="apodo" value="{{ \$apodo }}&gt;</code>
Password	Los campos para contraseñas lo único que hacen es ocultar las letras escritas.	<code>&lt;input type="password" name="contraseña" id="contraseña"&gt;</code>
Hidden	Los campos ocultos se suelen utilizar para almacenar opciones o valores que se desean enviar junto con los datos del formulario pero que no se tienen que mostrar al usuario.	<code>&lt;input type="hidden" id='id' name='id' value= "oculto"&gt;</code>
Number	Los campos de tipo number almacenan datos de tipo numérico. Puede contar con atributos donde se maneje un rango entre máximo y mínimo.	<code>&lt;input type="number" name="edad" id="edad"&gt;</code>



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II

### LARAVEL 8.0

Email	Los campos de tipo email almacenan datos relacionadas con direcciones de correo electrónico.	<code>&lt;input type="email" name="correo" id="correo"&gt;</code>
Date	Son campos que contienen una fecha.	<code>&lt;input type="date" name="nacimiento" id="nacimiento"&gt;</code>

Si quiere profundizar sobre la lista de tipos permitidos puede ingresar a: [https://www.w3schools.com/html/html\\_form\\_input\\_types.asp](https://www.w3schools.com/html/html_form_input_types.asp)

#### Otros tipos de Campos

Tipo de Campo	Descripción	Ejemplo
TextArea	Referente a un área de texto. Esta etiqueta además permite indicar el número de filas (rows ) y columnas ( cols ) del área de texto.	<code>&lt;textarea name="texto" id="texto" rows="4" cols="50"&gt;</code> Comentarios <code>&lt;/textarea&gt;</code>
Label	Las etiquetas nos permiten poner un texto asociado a un campo de un formulario para indicar el tipo de contenido que se espera en dicho campo.	El atributo <b>for</b> se utiliza para especificar el identificador del campo relacionado con la etiqueta. <code>&lt;label for="nombre"&gt;Nombre &lt;/label&gt;</code> <code>&lt;input type="text" id='nombre' name='nombre'&gt;</code>
Radio Button	Para crear campos tipo <i>checkbox</i> o <i>tipo radio button</i> tenemos que utilizar también la etiqueta input, pero indicando el tipo checkbox o radio respectivamente.	<code>&lt;label for="genero"&gt;Elige tu género:&lt;/label&gt; &lt;br&gt;</code> <code>&lt;input type="radio" name="genero" id="fem" value="f"&gt;Femenino</code> <code>&lt;input type="radio" name="genero" id="mas" value="m"&gt;Masculino</code>
CheckBox	Adicionalmente es importante que todos tengan el mismo nombre (para la propiedad name). De esta forma los valores devueltos estarán agrupados en la variable correspondiente.	<code>&lt;label for="terminos"&gt;Aceptar términos&lt;/label&gt;</code> <code>&lt;input type="checkbox" name="terminos" id="terms" value="1"&gt;</code>



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II

### LARAVEL 8.0

Ficheros	Para generar un campo para subir ficheros utilizamos también la etiqueta <i>input</i> indicando en su tipo el valor <i>file</i>	<pre>&lt;label for="foto"&gt;Sube la foto:&lt;/label&gt; &lt;input type="file" name="foto" id="foto"&gt;</pre>
Listas Desplegables	Para crear una lista desplegable utilizamos la etiqueta HTML <i>select</i> . Las opciones las indicaremos entre la etiqueta de apertura y cierre usando elementos <i>option</i> .	<pre>&lt;select name="actividades"&gt;   &lt;option value="musica"&gt;Escuchar Musica&lt;/option&gt;   &lt;option value="deporte"&gt;Practicar Deporte&lt;/option&gt;   &lt;option value="instrumento"&gt;Tocar un instrumento&lt;/option&gt;   &lt;option value="lectura"&gt;Leer&lt;/option&gt; &lt;/select&gt;</pre>
Botones	En un formulario podremos añadir tres tipos distintos de botones: <ul style="list-style-type: none"><li>• <i>submit</i> para enviar el formulario,</li><li>• <i>reset</i> para restablecer o borrar los valores introducidos.</li><li>• <i>button</i> para crear botones normales para realizar otro tipo de acciones (como volver a la página anterior).</li></ul>	<pre>&lt;button type="submit"&gt;Enviar&lt;/button&gt;  &lt;button type="reset"&gt;Borrar&lt;/button&gt;  &lt;button type="button"&gt;Volver&lt;/button&gt;</pre>

Información extraída de [5]



## VIII. INTRODUCCIÓN A LAS BASES DE DATOS

Laravel facilita la configuración y el uso de diferentes tipos de base de datos: MySQL V5.6+, PostgreSQL 9.4+, SQLite 3.8.8+ y SQL Server 2017+. En el fichero de configuración (config/database.php) tenemos que indicar todos los parámetros de acceso a nuestras bases de datos y además especificar cuál es la conexión que se utilizará por defecto.

### Configuración del Motor de la Base de Datos

Para iniciar es necesario efectuar la configuración de acceso de la base de datos. Para ello nos dirigimos al fichero con la configuración config/database.php y editar la siguiente línea, especificando el motor que por defecto se desea emplear. En el ejemplo se toma como gestor MySQL.

```
'default' => env ('DB_CONNECTION', 'mysql')
```

Luego en el mismo archivo en la opción **connections** se encuentra la configuración respectiva respecto al gestor seleccionado. Para el caso de MySQL seria:

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'), Host de conexión
    'port' => env('DB_PORT', '3306'), → Puerto de la base de datos
    'database' => env('DB_DATABASE', 'forge'), → Nombre de la base de datos
    'username' => env('DB_USERNAME', 'forge'), → Nombre del usuario de la BD
    'password' => env('DB_PASSWORD', ''), → Contraseña de la BD
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA')
    ]) : [],
],
```

Para editar los campos que se muestran anteriormente se debe realizar desde el fichero. env de la raíz del proyecto para enlazarlos con la base de datos respectiva:

```
.env
9
10 DB_CONNECTION=mysql → Conector para el gestor de BD
11 DB_HOST=127.0.0.1 → Host
12 DB_PORT=3306 → Puerto de conexión
13 DB_DATABASE=supermercado → Nombre de la BD
14 DB_USERNAME=root → Nombre del usuario de la BD
15 DB_PASSWORD= → Contraseña de la BD
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Creación de la base de datos con phpMyAdmin

Para crear la base de datos que vamos a utilizar en MySQL podemos utilizar la herramienta PHPMyAdmin que se ha instalado con el paquete XAMPP. Para esto accedemos a la ruta <http://localhost/phpmyadmin/> y desde allí seleccionamos la opción Nueva, para crear la base de datos.

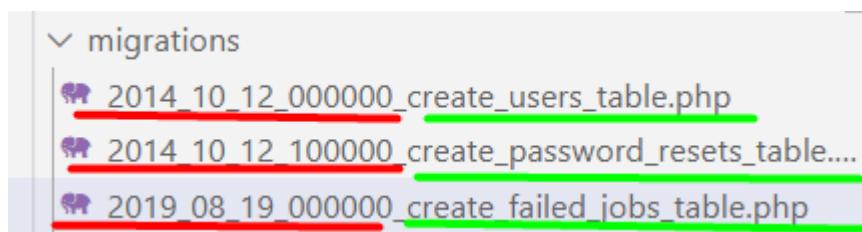
The screenshot shows the 'Bases de datos' (Databases) section of the phpMyAdmin interface. On the left, there's a sidebar with a 'Nueva' (New) button and a list of existing databases: information\_schema, mysql, performance\_schema, phpmyadmin, and test. The main area has tabs for 'Bases de datos', 'SQL', 'Estado actual', 'Cuentas de usuarios', 'Exportar', 'Importar', and 'Cor'. A sub-section titled 'Crear base de datos' (Create database) is open, showing a form with 'supermercado' in the database name field and 'utf8\_bin' in the character set dropdown. Below the form is a table listing existing databases with their character sets and options to select privileges. The table includes rows for information\_schema, mysql, performance\_schema, phpmyadmin, and test. A total count of 5 databases is shown at the bottom. At the bottom of the page, there are buttons for 'Seleccionar todo' (Select all) and 'Eliminar' (Delete).

### Tabla de migraciones

Laravel nos proporciona un mecanismo llamado Migraciones con el cual podremos diseñar la estructura de nuestra base de datos y mantener su historial de cambios a lo largo del desarrollo del proyecto [10]. Permiten que un equipo trabaje sobre una base de datos añadiendo y modificando campos, manteniendo un histórico de los cambios realizados, el estado actual de la base de datos, y compartir el esquema de la base de datos de la aplicación [5] [11].

Por defecto las migraciones se encuentran en el directorio `database/migrations`. Cada migración no es más que una clase de PHP que extiende de la clase **Migration**, que incluye en el nombre del archivo la fecha y la hora en que fue creada la migración (en formato timestamp) y el nombre de la migración. [10]

Al crear un nuevo proyecto, Laravel incluye por defecto TRES migraciones:





## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Estructura de una migración

El fichero o clase PHP generada para una migración siempre tiene una estructura similar a la siguiente:

```
database > migrations > 2020_11_09_022714_create_productos_table.php
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateProductosTable extends Migration
8  {
9      /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('productos', function (Blueprint $table) {
17             $table->id();
18             $table->timestamps();
19         });
20     }
21     /**
22     * Reverse the migrations.
23     *
24     * @return void
25     */
26     public function down()
27     {
28         Schema::dropIfExists('productos');
29     }
30 }
31
```

Elementos	Descripción
Nombre de la Clase	La cual es una clase que hereda de otra llamada Migration.
Método up	Permite especificar lo que va a efectuar la migración. Por lo general aquí se agregarán o actualizarán tablas en la base de datos, agregar columnas a una tabla ya existente o índices a la base de datos.
Método down	Nos permite revertir o «devolver» la operación realizada en el método up () .

Esto nos permitirá poder ir añadiendo y eliminando cambios sobre la base de datos y tener un control o histórico de los mismos. [5]

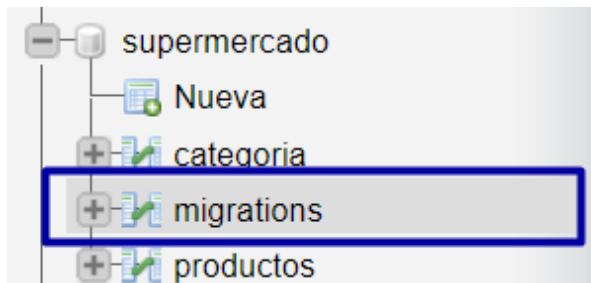


## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Dentro de Laravel es importante crear la tabla de migraciones. Para esto iniciamos ejecutando el siguiente comando de Artisan:

```
PS C:\xampp\htdocs\cursito> php artisan migrate:install
Migration table created successfully.
```

Si todo funciona correctamente nos dirigimos al navegador y accedemos a la base de datos creada con PHPMyAdmin, desde allí se deberá haber creado la tabla migrations y con esto ya tenemos configurada la base de datos y el acceso a la misma.



### Crear una migración

Para crear una migración usamos el comando de Artisan:

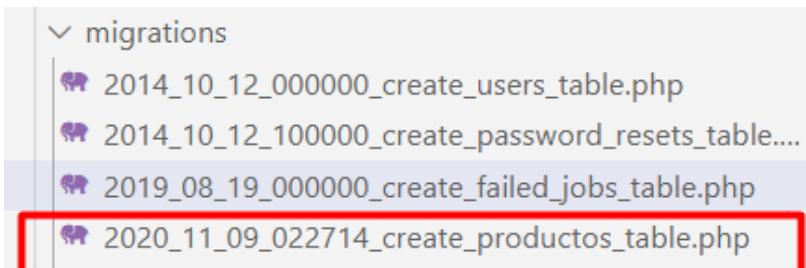
```
php artisan make:migration action_<table-name>_table.
```

La nueva migración se colocará en el directorio database/migrations. Las opciones --table y --create también se pueden usar para indicar el nombre de la tabla y si la migración creará una nueva tabla o no.

**Ejemplo:** Para la migración que involucra la creación de la tabla productos el comando será:

```
php artisan make:migration create_productos_table --create=productos
```

Esto nos creará un fichero de migración en la carpeta database/migrations con el nombre <TIMESTAMP>\_create\_productos\_table.php.



Si lo que queremos es añadir una migración que modifique los campos de una tabla existente tendremos que ejecutar el siguiente comando:

```
php artisan make:migration <action>_to_<table-name>_table --table=productos
```

**Ejemplo:**

```
php artisan make:migration update_to_productos_table --table=productos
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

En este caso se creará también un fichero en la misma carpeta, preparado para modificar los campos de dicha tabla.

**Nota: Se aconseja que el nombre de la migración lleve el nombre de la acción que se desea ejecutar.**

### Ejecutar migraciones

Comando	Descripción
<code>php artisan migrate</code>	Permite ejecutar una migración sobre la base de datos.
<code>php artisan migrate:rollback</code>	Revierte la última operación de migración
<code>php artisan migrate:rollback --step=X</code>	Permite revertir un número limitado de migraciones proporcionando la opción de paso al comando de reversión.
<code>php artisan migrate:reset</code>	Deshacer todas las migraciones efectuadas
<code>php artisan migrate:refresh</code>	Deshacer todos los cambios y volver a aplicar las migraciones
<code>php artisan migrate:fresh</code>	El comando migrate: fresh eliminará todas las tablas de la base de datos y luego ejecutará el comando migrate.
<code>php artisan migrate:status</code>	Permite comprobar el estado de las migraciones, para ver las que ya están instaladas y las que quedan pendientes.



### SCHEMA BUILDER (Constructor de esquemas)

Para especificar la tabla a crear o modificar, así como las columnas y tipos de datos de las mismas, se utiliza la clase Schema. Esta clase tiene una serie de métodos que nos permitirá especificar la estructura de las tablas independientemente del sistema de base de datos que utilicemos.

#### Crear tablas

Para crear una nueva tabla de base de datos, se empleará el método de creación en Schema Facade en el **método up** el cual acepta dos argumentos: el primero es el nombre de la tabla, mientras que el segundo es un cierre que recibe un objeto Blueprint que se puede usar para definir la nueva tabla. [11]

Ejemplo:

```
public function up()
{
    Schema::create('productos', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

En el método down tenemos el proceso inverso el cual contendrá la migración donde se podrá eliminar la tabla que hemos creado, para esto usaremos alguno de las siguientes instrucciones:

#### Opción 1:

```
public function down()
{
    Schema::dropIfExists('productos');
}
```

#### Opción 2:

```
public function down()
{
    Schema::drop('productos');
}
```

**Nota:** Cabe resaltar que al crear una migración con el comando de **Artisan make:migration** este código viene añadido por defecto, junto con la creación y eliminación de la tabla que se ha indicado y además se añaden un par de columnas por defecto (id y timestamps) [5].

#### Manejo de columnas en una tabla

El constructor **Schema::create** recibe como segundo parámetro una función que nos permite especificar las columnas que va a tener dicha tabla. En esta función podemos ir añadiendo todos los campos que queramos, indicando para cada uno de ellos su tipo y nombre, y así como también indicar una serie de modificadores como valor por defecto, índices, etc.

El constructor de esquemas contiene una variedad de tipos de columnas que se pueden especificar al crear sus tablas.

**Nota:** Para profundizar sobre los tipos de columnas existentes diríjase a la documentación de Laravel 8.0 en <https://laravel.com/docs/8.x/migrations#creating-columns>



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Se destaca los más principales a continuación:

Comando	Descripción
<code>\$table-&gt;bigIncrements('id');</code>	Tipo de dato Entero Grande de incremento automático. (equivalente a una llave primaria)
<code>\$table-&gt;bigInteger('votes');</code>	Tipo de dato entero grande.
<code>\$table-&gt;boolean('confirmed');</code>	Tipo de dato Boooleano
<code>\$table-&gt;char('name', 100);</code>	Tipo de dato Char con una longitud
<code>\$table-&gt;date('created_at');</code>	Tipo de dato Fecha
<code>\$table-&gt;decimal('amount', 8, 2);</code>	Tipo de dato Decimal con precisión (dígitos totales) y escala (dígitos decimales).
<code>\$table-&gt;double('amount', 8, 2);</code>	Tipo de dato Double con precisión (dígitos totales) y escala (dígitos decimales).
<code>\$table-&gt;float('amount', 8, 2);</code>	Tipo de dato Float con precisión (dígitos totales) y escala (dígitos decimales).
<code>\$table-&gt;increments('id');</code>	Tipo de dato Entero auto incremental (equivalente a una llave primaria)
<code>\$table-&gt;integer('votes');</code>	Tipo de dato Entero
<code>\$table-&gt;string('name');</code>	Tipo de dato Cadena
<code>\$table-&gt;string('name', 100);</code>	Tipo de dato Cadena con una longitud indicada
<code>\$table-&gt;text('description');</code>	Tipo de dato Texto
<code>\$table-&gt;unsignedInteger('votes');</code>	Tipo de dato Entero sin signo
<code>\$table-&gt;year('birth_year');</code>	Tipo de dato Año

Fuente [11]

### Modificadores de columna

Comando	Descripción
<code>-&gt;after('column')</code>	Coloque la columna "después" de otra columna (MySQL)
<code>-&gt;autoIncrement()</code>	Columna auto incremental
<code>-&gt;charset('utf8mb4')</code>	Especifique un conjunto de caracteres para la columna (MySQL)
<code>-&gt;comment('my_comment')</code>	Agregar un comentario a una columna (MySQL / PostgreSQL)
<code>-&gt;default(\$value)</code>	Especificar un valor por defecto
<code>-&gt;first()</code>	Coloca el atributo primero en la tabla (MySQL)
<code>-&gt;from(\$integer)</code>	Establecer el valor inicial de un campo de incremento automático (MySQL / PostgreSQL)
<code>-&gt;nullable(\$value = true)</code>	Permite (por defecto) insertar valores NULL en la columna.
<code>-&gt;unsigned()</code>	Establecer columnas enteras sin signo (MySQL)

Fuente [11]

### Manejo de índices

Schema soporta los siguientes tipos de índices:

Comando	Descripción
<code>\$table-&gt;primary('id');</code>	Añadir una clave primaria
<code>\$table-&gt;primary(['id', 'parent_id']);</code>	Definir una clave primaria compuesta
<code>\$table-&gt;unique('email');</code>	Definir el campo como UNIQUE
<code>\$table-&gt;index('state');</code>	Añadir un índice a una columna



### Llaves foráneas

Laravel también proporciona soporte para crear restricciones de clave externa, que se utilizan para forzar la integridad referencial en el nivel de la base de datos [11].

```
Schema::table('posts', function (Blueprint $table) {
    $table->unsignedBigInteger('user_id');

    $table->foreign('user_id')->references('id')->on('users');
});
```

En este ejemplo en primer lugar añadimos la columna " user\_id " de tipo UNSIGNED BIG\_INTEGER (siempre tendremos que crear primero la columna sobre la que se va a aplicar la clave ajena). A continuación, creamos la clave ajena entre la columna " user\_id " y la columna " id " de la tabla " users ".

**Nota:** La columna con la clave ajena tiene que ser del mismo tipo que la columna a la que apunta.

De igual forma también puede especificar la acción deseada para las propiedades "al eliminar" y "al actualizar" de la restricción:

```
$table->foreign('user_id')
->references('id')->on('users')
->onDelete('cascade');
```

Para eliminar una clave ajena, en el método **down** de la migración tenemos que utilizar el siguiente código:

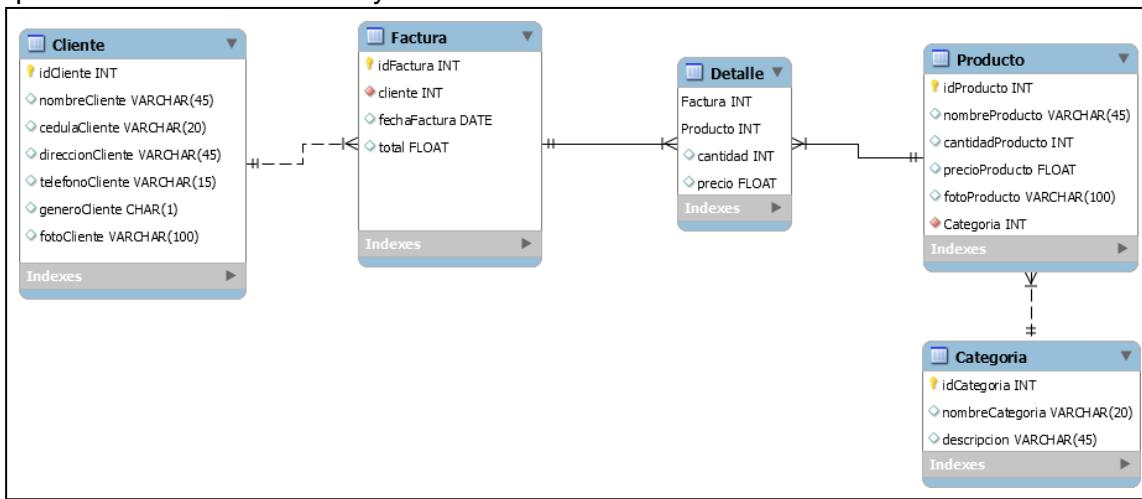
```
$table->dropForeign('posts_user_id_foreign');
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Ejemplo:

En el siguiente diagrama se muestra el esquema relacional de la base de datos Supermercado realizado en MySQL Workbench



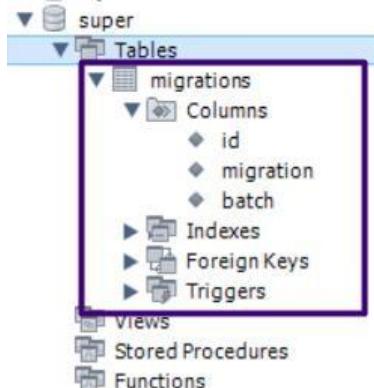
Creamos una base de datos vacía llamada Super en MySQL la cual se enlazará a Laravel de la desde el archivo de entorno .env ubicado en la raíz del proyecto.

MySQL	. env del Proyecto Laravel
<p>Query 1 super - Schema X</p> <p>Name: super</p> <p>Rename References</p> <p>Charset/Collation: Default Charset Default Collation</p>	DB_CONNECTION=mysql DB_HOST=127.0.0.1 DB_PORT=3307 DB_DATABASE=super DB_USERNAME=root DB_PASSWORD=1234

Posteriormente instalamos la tabla de migraciones con el comando respectivo:

```
PS C:\xampp\htdocs\cursito> php artisan migrate:install
Migration table created successfully.
```

Automáticamente la tabla migraciones debió ser creada en la BD Super.





## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Ahora bien, por cada tabla expuesta en el diagrama del modelo relacional se va a crear una migración empleando las instrucciones respectivas.

Veamos un ejemplo con la tabla Categoría y Producto:

### Para crear la tabla Categoría con el comando de migración

```
PS C:\xampp\htdocs\cursito> php artisan make:migration create_categoria_table --create=categoria
Created Migration: 2020_11_22_152539_create_categoria_table
```

La estructura inicialmente creada será la siguiente:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateCategoriaTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('categoria', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('categoria');
    }
}
```

Ahora bien, desde la función **up** se tiene un esquema cuya funcionalidad será crear la tabla categoria dentro de la base de datos y donde además se adicionarán los atributos respectivos.

```
public function up()
{
    Schema::create('categoria', function (Blueprint $table) {
        $table->id(); //Llave primaria, de tipo BigInteger, atributo unico, autoincremental
        $table->string('nombreCategoria',20); // Varchar 20
        $table->string('descripcion', 45); // Varchar 45
        $table->timestamps(); //create_at, update_at
    });
}
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Para crear la tabla **Productos** con el comando de migración

```
PS C:\xampp\htdocs\cursito> php artisan make:migration create_producto_table --create=producto
Created Migration: 2020_11_22_152130_create_producto_table
```

La estructura inicialmente creada será la siguiente:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateProductoTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('producto', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('producto');
    }
}
```

Ahora bien, desde la función **up** se tiene un esquema cuya funcionalidad será crear la tabla **productos** dentro de la base de datos y donde se adicionarán los atributos respectivos y la relación con la tabla **categoría**.

```
public function up()
{
    Schema::create('producto', function (Blueprint $table) {
        $table->id(); //Llave primaria, atributo unico, autoincremental
        $table->string('nombreProducto', 45); // Varchar 45
        $table->integer('cantidadProducto'); // Entero
        $table->float('precioProducto'); // Flotante
        $table->bigInteger('categoria')->unsigned();
        $table->string('fotoProducto', 100); // Varchar 45
        $table->foreign('categoria') //Creación de la llave foranea
            ->references('id')
            ->on('categoria')
            ->onDelete('cascade');
        $table->timestamps();
    });
}
```

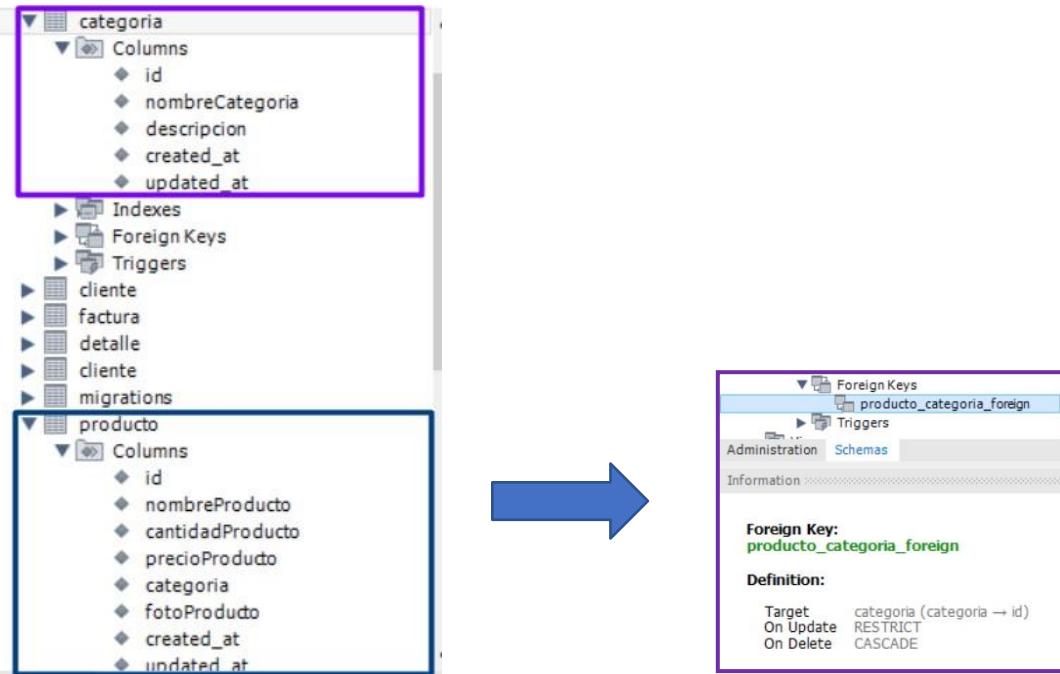


## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

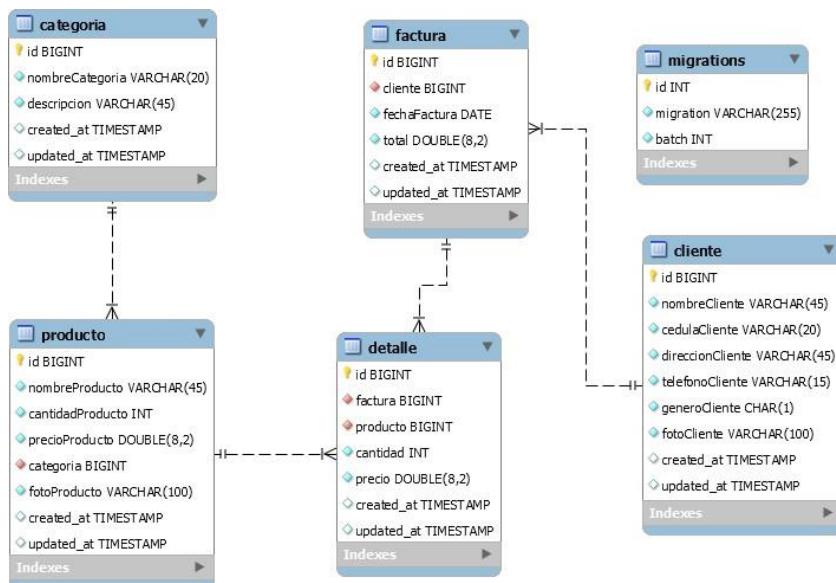
Para ejecutar las migraciones es importante respetar el orden de las mismas para evitar cualquier tipo de error, para ello se empleará los siguientes comandos:

```
PS C:\xampp\htdocs\cursito> php artisan migrate --path=database\migrations\2020_11_22_152539_create_categoria_table.php
Migrating: 2020_11_22_152539_create_categoria_table
Migrated: 2020_11_22_152539_create_categoria_table (34.41ms)
PS C:\xampp\htdocs\cursito> php artisan migrate --path=database\migrations\2020_11_22_152130_create_producto_table.php
Migrating: 2020_11_22_152130_create_producto_table
Migrated: 2020_11_22_152130_create_producto_table (88.75ms)
```

De esta forma el resultado se evidencia a continuación en el gestor respectivo:



De la misma forma se crearán las migraciones respectivas para las tablas resultantes. Al finalizar se obtendrá una estructura como la siguiente donde se incluirán los atributos de creación y actualización respectivamente por cada una:





## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### SEEDERS EN LARAVEL

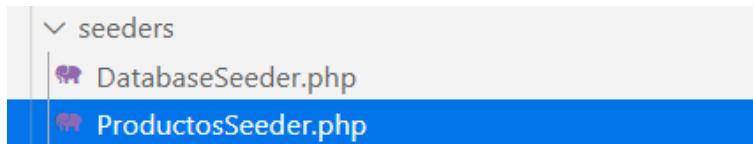
Laravel también facilita la inserción de datos iniciales. Esta opción es muy útil para tener datos de prueba cuando estamos desarrollando una web o para crear tablas que ya tienen que contener una serie de datos en producción. [5]

Para generar un seeder utilizamos el comando de Artisan **make:seeder** seguido del nombre del seeder [10] :

```
PS C:\xampp\htdocs\cursito> php artisan make:seeder ProductosSeeder
```

```
Seeder created successfully.
```

Al ejecutar este comando se generará un archivo ProductosSeeder.php dentro del directorio **database/seeders**.



Dentro del archivo debemos agregar la ruta de los Facades de Laravel que requiere para su ejecución y en el método run() escribimos el código de nuestro Seeder donde se hará la inserción de los datos:

```
<?php
namespace Database\Seeders;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;
use Illuminate\Database\Seeder;

class ProductosSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('producto')->insert([
            'id' => 7,
            'nombreProducto'=> 'Sal Refusal',
            'cantidadProducto' => 50,
            'precioProducto' => 1300,
            'categoria' => 1,
            'fotoProducto'=> '7.jpg',
        ]);
    }
}
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Si deseamos agregar más de un registro a la tabla, se podrá realizar de la siguiente manera:

```
public function run()
{
    $datos = [
        [
            'nombreProducto'=> 'Limpido Clorox',
            'cantidadProducto' => 45,
            'precioProducto' => 2000,
            'categoria' => 4,
            'fotoProducto'=> '12.jpg',
        ],
        [
            'nombreProducto'=> 'Ron Viejo de Caldas',
            'cantidadProducto' => 200,
            'precioProducto' => 60000,
            'categoria' => 2,
            'fotoProducto'=> '13.jpg',
        ]
    ];
    DB::table('producto')->insert($datos);
}
```

Para ejecutar un seeder se puede lograr de forma predeterminada, el comando db: seed en donde se ejecuta la clase DatabaseSeeder, que se puede usar para llamar a otras clases de semillas. Sin embargo, puede usar la opción --class para especificar una clase de sembradora específica para que se ejecute individualmente así:

```
php artisan db:seed --class=ProductosSeeder
```

**Nota:** Para insertar datos manualmente usted puede usar el generador de consultas o bien usar las fábricas de modelos Eloquent.

### Eliminar registros

Es posible que antes de ejecutar un seeder necesitemos eliminar el contenido existente. Para realizar esto podemos utilizar el método truncate, que se encarga de vaciar la table:

```
DB::table('categoria')->truncate();
```



## IX. CONSTRUCTOR DE CONSULTAS (QUERY BUILDER)

Laravel incluye una serie de clases que nos facilita la construcción de consultas y otro tipo de operaciones con la base de datos. Query Builder proporciona una interfaz cómoda y fluida para crear y ejecutar consultas de bases de datos.

### Consultas

Para la realización de consultas en Query Builder, la operación se podrá concretar desde el Controlador respectivo instanciando el Facade de Laravel en la parte superior:

```
use Illuminate\Support\Facades\DB;
```

### Cláusula Select

Para realizar un "**Select \* from**" que devuelva todas las filas de una tabla utilizaremos el siguiente código:

```
public function consultar(){
    $productos = DB::table('producto')->get();
    return view ('productos.listado', ['productos' => $productos]);
}
```

En la vista podrá mostrarse de la siguiente forma:

```
@extends('main')

@section('content')
    @foreach ($productos as $p)
        <p> Id: {{$p->id}} </p>
        <p> Nombre {{$p->nombreProducto}} </p>
        <p> Categoría {{$p->categoria}} </p>
        <p> Precio {{$p->precioProducto}} </p>
    @endforeach

@stop
```

Si solo deseamos visualizar algunas columnas de la tabla podemos especificarlas con la instrucción **select** de la siguiente forma:

```
$productos = DB::table('producto')
    ->select('id', 'nombreProducto')
    ->get();
return view ('productos.listado', ['productos' => $productos]);
```

### Consultar y visualizar un solo registro

Si queremos obtener un solo elemento podemos utilizar **first** en lugar de **get**, de la forma:

```
// Obtiene el primer registro de la tabla
$producto = DB::table('producto')->first();
return view ('productos.resultado', ['p' => $producto]);
```

En la vista se mostrará así:

```
<p> Id: {{$p->id}} </p>
<p> Nombre: {{$p->nombreProducto}} </p>
<p> Categoría: {{$p->categoria}} </p>
<p> Precio: {{$p->precioProducto}} </p>
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Cláusula where

Para filtrar los datos usamos la cláusula **where**, indicando el nombre de la columna y el valor a filtrar:

```
// Obtiene todos los registros de la tabla que cumplan con la condicion
$productos = DB::table('producto')
    ->where('nombreProducto', 'Azucar Morena Libra')
    ->get();
return view ('productos.listado', ['productos' => $productos]);
```

Si queremos realizar otro tipo de filtrados, como columnas que tengan un valor mayor (**>**), mayor o igual (**>=**), menor (**<**), menor o igual (**<=**), distinto del indicado (**<>**) o usar el operador like, lo podemos indicar como segundo parámetro de la forma.

#### Ejemplo con >=

```
$producto = DB::table('producto')->where('cantidadProducto', '>=', 50)->get();
return view('productos.result', ['p'=> $producto]);
```

#### Ejemplo con <>

```
$product = DB::table('producto')->where('cantidadProducto', '<>', 50)->get();
return view('productos.result', ['p'=> $product]);
```

#### Ejemplo con like

```
$producto = DB::table('producto')->where('nombreProducto', 'like', 'Arroz%')->get();
return view('productos.result', ['p'=> $producto]);
```

Si añadimos más cláusulas where a la consulta por defecto se unirán mediante el operador lógico AND . En caso de que queramos utilizar el operador lógico OR lo tendremos que realizar usando orWhere de la forma:

#### Uso de OR

```
$producto = DB::table('producto')
    ->where('nombreProducto', 'like', 'Arroz%')
    ->orwhere('nombreProducto', 'like', 'Azucar%')
    ->get();
return view('productos.result', ['p'=> $producto]);
```

#### Uso de AND

```
$producto = DB::table('producto')
    ->where('nombreProducto', 'like', 'A%')
    ->where('categoria', '=', 1)
    ->get();
return view('productos.result', ['p'=> $producto]);
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Método Value

Si ni siquiera se requiere obtener una fila completa, con Query Builder puede extraer un solo valor de un registro utilizando el método *value*. Este método devolverá el valor de la columna directamente así:

```
// Obtiene el valor del id del registro de la tabla que cumpla la condición
$producto = DB::table('producto')
    ->where('nombreProducto', 'Azucar Morena Libra')
    ->value('id');
return view ('productos.resultado', ['p' => $producto]);
```

### Método Pluck

Si desea recuperar una colección que contenga los valores de una sola columna, puede utilizar el método *pluck*.

```
// obtiene la columna que se le indique de cada registro
$names= DB::table('producto')
    ->pluck('nombreProducto');
return view ('productos.lista', ['nombres' => $names]);
```

### Cláusula OrderBy

El método *orderBy* organiza el resultado por un patrón indicado

```
$productos = DB::table('producto')
    ->orderBy('categoria', 'asc')
    ->get();
return view ('productos.listado', ['productos' => $productos]);
```

### Cláusula GroupBy – Having

Los métodos *groupBy* y *having* pueden usarse para agrupar los resultados de la consulta.

```
$productos = DB::table('producto')
    ->groupBy('id','cantidadProducto' )
    ->having('cantidadProducto', '>', 25)
    ->get();
return view ('productos.listado', ['productos' => $productos]);
```

### Cláusula Skip/Take y Offset/Limit

Para limitar la cantidad de resultados devueltos por la consulta, o para omitir una cantidad determinada de resultados en la consulta, puede usar los métodos *skip* and *take*:

```
//Skip y Take
$productos = DB::table('producto')->skip(1)->take(5)->get();
return view ('productos.listado', ['productos' => $productos]);
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

De igual manera el resultado se puede obtener con la cláusula Offset/Limit

//Offset y Limit

```
$productos = DB::table('producto')->offset(1)->limit(5)->get();  
return view ('productos.listado', ['productos' => $productos]);
```

### Cláusulas de Agregación

El generador de consultas también proporciona una variedad de métodos agregados, como recuento, máximo, mínimo, promedio y suma. Puede llamar a cualquiera de estos métodos después de construir su consulta:

```
$contador = DB::table('producto')->count(); // Cuenta  
$maximo = DB::table('producto')->max('cantidadProducto'); // Maximo  
$minimo = DB::table('producto')->min('cantidadProducto'); // Minimo  
$promedio = DB::table('producto')->avg('precioProducto'); // Promedio  
return view ('productos.lista',  
[  
    'count' => $contador,  
    'max' => $maximo,  
    'min' => $minimo,  
    'avg' => $promedio  
]);  
);
```

### JOINS

#### Clausula Inner Join

El primer argumento que se pasa al método de Join es el nombre de la tabla a la que debe unirse, mientras que los argumentos restantes especifican las restricciones de columna para la combinación.

```
$productos = DB::table('producto')  
    ->join('categoria', 'categoria', '=', 'categoria.id')  
    ->get();  
return view ('productos.listado', ['productos' => $productos]);
```

#### Clausula Left Join / Right Join

Si desea realizar una "unión izquierda" o "unión derecha" en lugar de una "unión interna", utilice los métodos leftJoin o rightJoin. Estos métodos tienen la misma firma que el método de unión.

```
$users = DB::table('users')  
    ->leftJoin('posts', 'users.id', '=', 'posts.user_id')  
    ->get();  
  
$users = DB::table('users')  
    ->rightJoin('posts', 'users.id', '=', 'posts.user_id')  
    ->get();
```

Para ampliar la información sobre Query Builder consulte:

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



## X. MODELO DE DATOS

### ¿QUÉ ES UN ORM?

El mapeado objeto-relacional (más conocido por su nombre en inglés, **Object-Relational mapping**, o por sus siglas ORM) es una técnica de programación para convertir datos entre un lenguaje de programación orientado a objetos y una base de datos relacional como motor de persistencia [5].

“Las estructuras de la base de datos relacional quedan vinculadas con las entidades lógicas o base de datos virtual definida en el ORM, de tal modo que las acciones CRUD (Create, Read, Update, Delete) a ejecutar sobre la base de datos física se realizan de forma indirecta por medio del ORM” [12].

### ¿QUE ES ELOQUENT?

Eloquent es el ORM que incluye Laravel para manejar de una forma fácil y sencilla los procesos correspondientes al manejo de bases de datos en nuestro proyecto, gracias a las funciones que provee podremos realizar complejas consultas y peticiones de base de datos sin escribir una sola línea de código SQL.

### ¿QUÉ ES UN MODELO DE DATOS?

Los modelos se guardarán como clases PHP dentro de la carpeta app/Models. Para definir un modelo que use Eloquent podemos crear una clase que herede de la clase Model y almacenarla en el directorio indicado o emplear el siguiente comando:

```
PS C:\xampp\htdocs\cursito> php artisan make:model Categoria
Model created successfully.
```

Este comando creará automáticamente un archivo con las siguientes características

```
Models > 🐘 Categoria.php
```

```
<?php
```

```
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
```

```
class Categoria extends Model
{
    use HasFactory;
}
```

**Nota:** En general el nombre de los modelos se pone en singular con la primera letra en mayúscula, mientras que el nombre de las tablas suele estar en plural.



### Especificar la tabla relacionada al modelo

Para especificar la tabla que se va a relacionar en el modelo debemos especificarlo con la propiedad \$table [5]:

```
class Categoria extends Model
{
    protected $table = 'categoria';
}
```

### Clave primaria

Laravel también asume que cada tabla tiene declarada una clave primaria con el nombre id. En el caso de que no sea así y queramos cambiarlo tendremos que sobrescribir el valor de la propiedad protegida \$primaryKey del modelo, por ejemplo: [5]

```
protected $primaryKey = 'my_id';
```

**Nota:** Es importante definir correctamente este valor ya que se utiliza en determinados métodos de Eloquent, como por ejemplo para buscar registros o para crear las relaciones entre modelos.

### Timestamps

Otra propiedad que en ocasiones tendremos que establecer son los timestamps automáticos. Por defecto Eloquent asume que todas las tablas contienen los campos updated\_at y created\_at (los cuales los podemos añadir muy fácilmente con Schema añadiendo \$table->timestamps() en la migración). Estos campos se actualizarán automáticamente cuando se cree un nuevo registro o se modifique. En el caso de que no queramos utilizarlos (y que no estén añadidos a la tabla) tendremos que indicarlo en el modelo o de otra forma nos daría un error. [5]

Para indicar que no los actualice automáticamente tendremos que modificar el valor de la propiedad pública \$timestamps a false, por ejemplo:

```
public $timestamps = false;
```

Es decir:

```
class Categoria extends Model
{
    protected $table = 'categoria';
    protected $primaryKey = 'my_id';
    public $timestamps = false;
}
```



## Uso de un modelo de datos

Una vez creado el modelo ya podemos empezar a emplearlo para recuperar datos de la base de datos, para insertar nuevos datos o para actualizarlos. El sitio correcto donde realizar estas acciones, es en el controlador, el cual se los tendrá que pasar a la vista ya preparados para su visualización [5].

Es importante que para su utilización indiquemos al inicio de la clase el espacio de nombres del modelo o modelos a utilizar. Por ejemplo, si vamos a usar el modelo Categoría se debería colocar la instrucción:

```
use App\Models\Categoria;
```

### Consultar datos

Para obtener todas las filas de la tabla asociada a un modelo usaremos el método all(). Este método nos devolverá un array de resultados, donde cada ítem del array será una instancia del modelo **Categoría**. Gracias a esto al obtener un elemento del array podemos acceder a los campos o columnas de la tabla como si fueran propiedades del objeto (`$c->id`).

```
<?php
```

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Models\Categoria;

class CategoriasController extends Controller
{

    public function index(){
        // Consultar categorias con Eloquent
        $categorias = Categoria::all();
        return view('categorias.listado',[ 'categorias' => $categorias ] );
    }
}
```

En la vista se visualiza de la misma manera con la ayuda de un foreach.

```
:sources > views > categorias > listado.blade.php
1  @extends('main')
2  @section('content')
3
4  <h1> Categorías de productos </h1>
5  <table class="table">
6      <thead>
7          <tr>
8              <th scope="col">#</th>
9              <th scope="col">Nombre Categoria</th>
10             <th scope="col">Descripción </th>
11         </tr>
12     </thead>
13     <tbody>
14         @foreach($categorias as $c)
15             <tr>
16                 <td> {{ $c->id }} </td>
17                 <td> {{ $c->nombreCategoria }} </td>
18                 <td> {{ $c->descripcion }}</td>
19             </tr>
20
21         @endforeach
22     </tbody>
23 </table>
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Eloquent también incorpora el método `find($id)` para buscar un elemento a partir del identificador único del modelo, por ejemplo:

```
$categorias = Categoria::find(1);
return view('categorias.listado',[ 'c' => $categorias] );
```

A continuación, se incluyen otros ejemplos de consultas usando Eloquent con algunos de los métodos que ya habíamos visto en la sección "Constructor de consultas":

```
// Obtener la primera categoria cuya nombre empiece por D
$categorias = Categoria::where('nombreCategoria', 'like', 'D%')->first();
return view('categorias.listado',[ 'categorias' => $categorias] );

// Obtener las categorias cuyo nombre empiece por D
$categorias = Categoria::where('nombreCategoria', 'like', 'D%')->get();
return view('categorias.listado',[ 'categorias' => $categorias] );
```

También podemos utilizar los métodos agregados para calcular el total de registros obtenidos, o el máximo, mínimo, media o suma de una determinada columna. Por ejemplo:

```
$count = Categoria::where('nombreCategoria', 'like', 'D%')->count();
$max = Producto::max('cantidadProducto');
$min = Producto::min('cantidadProducto');
$avg = Producto::avg('cantidadProducto');
$total = Producto::sum('cantidadProducto');
```

### DATOS DE ENTRADA CAPTURADOS DE UN FORMULARIO

Laravel facilita el acceso a los datos de entrada del usuario a través de solo unos pocos métodos.

Para conseguir acceso a estos métodos Laravel utiliza inyección de dependencias. Esto es simplemente añadir la clase **Request** al **constructor** o **método del controlador** en el que lo necesitemos. Laravel se encargará de inyectar dicha dependencia ya inicializada y directamente podremos usar este parámetro para obtener los datos de entrada [5].

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Models\Categoria;
use App\Models\Producto;

class CategoriasController extends Controller
{
    public function registrar(Request $request) {
    }
}
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Ejemplo:

Efectuar el registro de una nueva categoría a través de una vista que contenga un formulario que permita lograr dicho objetivo.

Definiendo la ruta categorías/registro:

```
Route::get('categorias/registro', [CategoriasController::class, 'form_registro'])  
    ->name('form_registroCategoria');
```

En el controlador Categorías creamos una nueva función:

```
public function form_registro() {  
    return view('categorias.form_registro');  
}
```

En la vista form\_registro se tiene el formulario de registro de una categoría:

```
@extends('main')  
@section('content')  
  
<div class="container">  
    <h1> Registro de Categorias </h1>  
    <form action="{{ url('categorias/registro') }}" method="POST">  
        @csrf  
  
        <label for="nombreCat">Nombre Categoria </label>  
        <input type="text" id='nombreCat' name='nombreCat' class="form-control"> <br> <br>  
  
        <label for="descripcionCat">Descripcion </label>  
        <input type="text" id='descripcionCat' name='descripcionCat' class="form-control"> <br> <br>  
  
        <button type="submit" class="btn btn-success">Enviar</button>  
  
    </form>  
@stop
```

## Registro de Categorias

Nombre Categoria

Descripcion

**ENVIAR**



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Al momento de diligenciar el formulario se envían todos los datos por método Post a la siguiente ruta:

```
Route::post('categorias/registro', [CategoriasController::class, 'registrar'])  
|     |     ->name('registrarCategoria');
```

La ruta apunta a una función dentro del controlador **Categorías** llamada **registrar** la cual recibe una petición de tipo **Request**.

**Nota:** Para añadir una entrada en la tabla de la base de datos asociada con un modelo simplemente tenemos que crear una nueva instancia de dicho modelo en la función del Controlador que se desee, asignar los valores que queramos y por último guardarlos con el método `save()`.

```
public function registrar(Request $request) {  
    $category = new Categoria();  
    $category->nombreCategoria = $request->input('nombreCat');  
    $category->descripcion = $request->input('descripcionCat');  
    $category->save();  
    return redirect()->route('listado_categorias');  
}
```

La función visualizada con anterioridad permite efectuar el registro de una nueva categoría en la base de datos con los campos diligenciados en el formulario. Al finalizar la operación, se retorna al listado de categorías para detallar el resultado.

### Categorías de productos

#	Nombre Categoría	Descripción
1	Dispensa	Diarios
2	Bebidas	Licores y bebidas
3	Fruver	Frutas y Verduras
4	Aseo Hogar	Diarios Aseo Hogar
5	Dulces	Dulceria y Galletas



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Otras Transacciones con Eloquent

#### Consultar un producto

La ruta productos/consulta redirecciona a la función form\_consulta del Controlador Productos la cual contiene la vista que tendrá el formulario de consulta un producto por algún criterio de búsqueda.

```
Route::get('productos/consulta', [ProductosController::class, 'form_consulta'])
    ->name('form_consulta');

public function form_consulta(){
    // Funcion que genera el formulario de consulta
    return view('productos.form_consulta');
}

@extends('main')
@section('content')
<br> <br>
<form action="{{ url('productos/consulta') }}" method="POST">
    @csrf
    <div class = "container">
        <div class="row">
            <div class = "col-sm-12">
                <input type="text" id="product" name="product" class="form-control" placeholder="Nombre Producto" required>
            </div>
        </div>
        <div class="row">
            <div class = "col-sm-12" align="center">
                <button type="submit" class="btn btn-primary" >Buscar</button>
            </div>
        </div>
    </div>
</form>
@stop
```

Resultado en el navegador:

A screenshot of a web browser showing a search form. The form consists of a text input field with the placeholder "Nombre Producto" and a blue rectangular button with the text "BUSCAR" in white capital letters.

La ruta productos/consulta por método POST recibe el criterio de búsqueda:

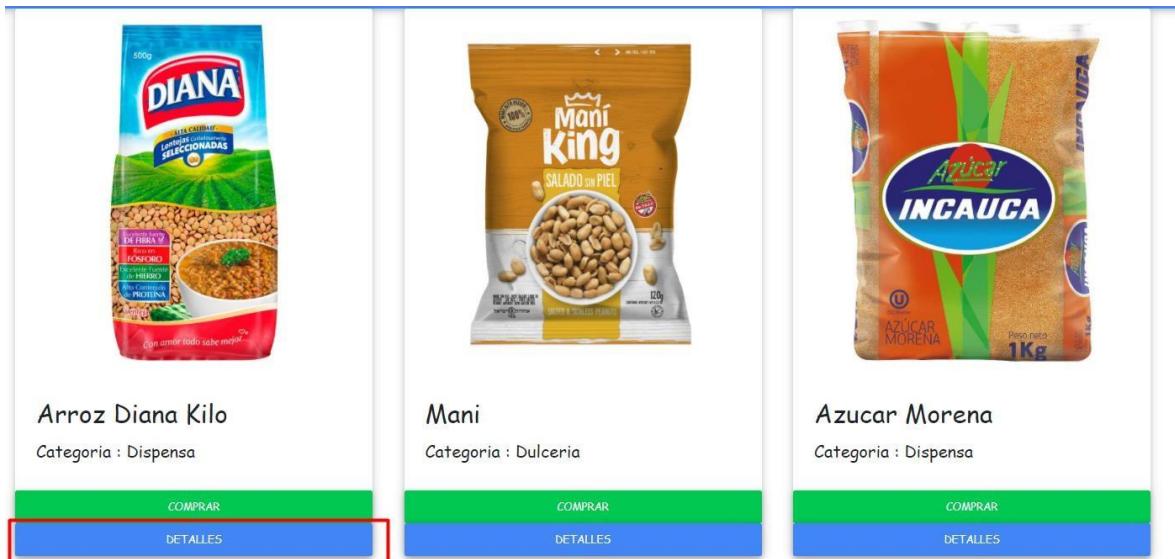
```
Route::post('productos/consulta', [ProductosController::class, 'consultar'])
    ->name('consulta_productos');
```

La función consultar en el controlador Productos tendría algo como lo siguiente:

```
public function consultar(Request $r){
    // Consulta un producto de acuerdo al criterio de búsqueda empleando Eloquent
    $name = $r->input('product');
    $producto = Producto::where('nombreProducto', 'like', $name)->first();
    if($producto)
        return view('productos.resultado', compact('producto'));
    else
        return view('productos.mensaje');
}
```

#### Visualizar detalle de un producto

En la vista listado de productos habrá una opción Detalles por cada producto la cual mostrará la información ampliada de cada uno.



En la Vista:

```

@extends('main')
@section('content')


@foreach ($productos as $p)
    <div class = "col-md-4">
        <!-- Card -->
        <div class="card" id ="tarjeta">
            <div class="view overlay">
                <img class="card-img-top" src='{{url("/img/$p->fotoProducto")}}' alt="Card image cap" height="350">
            </div>
            <div class="card-body">
                <!-- Title -->
                <h4 class="card-title"> {{ $p->nombreProducto }} </h4>
                <h6> Categoría : {{ $p->nombreCategoria }} </h6>
            </div>
            <div class="figure-action">
                <a class="btn btn-block btn-sm btn-success">
                    Comprar</a>
            </div>
            <div class="figure-action">
                <a href="{{route('visualizar_detallePro', $p->id)}}>Detalles</a>
            </div>
        </div>
        <!-- Card -->
    </div>
    @endforeach
</div>
@stop


```

Dicho enlace apuntara a la ruta:

```

Route::get('productos/visualiza/{id}', [ProductosController::class,
    'show'])
    ->name('visualizar_detallePro');

```

La cual redirecciona al método show del controlador Productos:



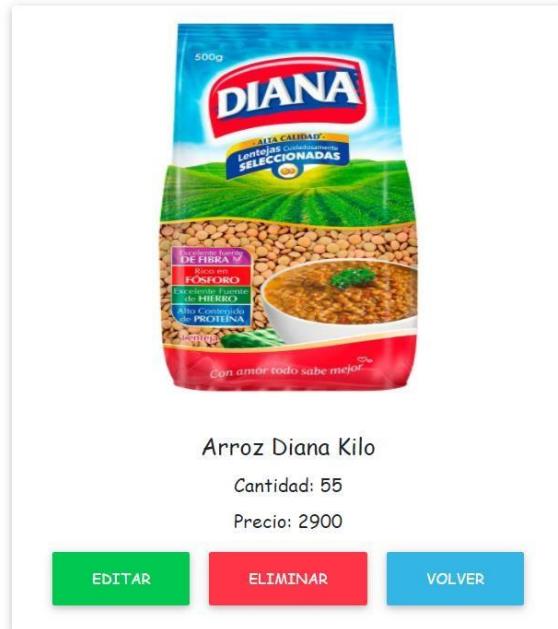
## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

```
public function show($id){  
  
    $product = Producto::findOrFail($id);  
    return view('productos.detalle', compact('product'));  
}
```

Este método realizara la búsqueda de un determinado producto y lo enviara a la vista detalle.

```
@extends('main')  
@section('content')  
    <div class="container" align="center">  
        <h3> Detalle de Producto </h3> <br> <br>  
        <div class="card" style="width: 30rem">  
            fotoProducto")}}"  
            class="card-img-top" height="350">  
            <div class="card-body">  
                <h5 class="card-title">{{$product->nombreProducto}}</h5>  
                <h6 class="card-title">Cantidad: {{$product->cantidadProducto}}</h2>  
                <h6 class="card-title">Precio: {{$product->precioProducto}}</h2>  
  
                <a href="{{route('formulario_actualizaPro',$product->id )}}" class="btn btn-success">Editar</a>  
                <a href="{{route('elimina_producto', $product->id)}}> class="btn btn-danger">Eliminar</a>  
                <a href="{{route('listado_productos')}}" class="btn btn-info"> Volver</a>  
            </div>  
        </div>  
    </div>  
@stop
```

La vista mostrará las opciones para edición, eliminación y volver al menú principal.  
Detalle de Producto



Actualizar Registro



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Al seleccionar la opción Editar esta redirige a una ruta la cual recibe como parámetro el identificador del producto que fue escogido.

```
<a href="{{route('formulario_actualizaPro',$product->id )}}" class="btn btn-success">Editar</a>

Route::get('productos/actualizar/{id}', [ProductosController::class,
    'form_actualiza'])
->name('formulario_actualizaPro');
```

La función form\_actualiza del controlador Productos pretende retornar la información del producto seleccionado y las categorías existentes para ser visualizados en la vista del formulario de edición

```
public function form_actualiza($id){
    $producto = Producto::findOrFail($id);
    $categorias = Categoria::all();
    return view ('productos.form_actualiza',
        compact('producto','categorias'));
}
```

En la vista del formulario de edición:

```
@extends('main')

@section('content')



# Actualizar Productos </h1> <form action="{{route('actualiza_producto', $producto->id) }}" method="POST"> @csrf <label for="nombrePro">Nombre Producto </label> <input type="text" id='nombrePro' name='nombrePro' class="form-control" required value='{{$producto->nombreProducto}}'> <label for="cantidadPro">Cantidad </label> <input type="number" id='cantidadPro' name='cantidadPro' class="form-control" required value='{{$producto->cantidadProduc <label for="precioPro">Precio </label> <input type="number" id='precioPro' name='precioPro' class="form-control" required value='{{$producto->precioProducto}}'> <label for="fotoPro">Foto: {{$producto->fotoProducto}}</label> <input type="file" id='fotoPro' name='fotoPro' class="form-control" value='{{$producto->fotoProducto}}'> <br> <label for="categoria"> Categoría </label> <br> <select class="form-select" aria-label="Default select example" name="categoria"> @foreach($categorias as $c) <option value="{{$c->id}}> {{ $c->nombreCategoria }} </option> @endforeach </select> <button type="submit" class="btn btn-success">Enviar</button> </form> @stop


```

Finalmente, la actualización de datos del formulario apunta la misma ruta indicada anteriormente (con método POST)

```
Route::post('productos/actualizar/{id}', [ProductosController::class, 'actualizar'])
->name('actualiza_producto');
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

La cual tiene como finalidad efectuar el proceso de actualización utilizando el ORM de Laravel así:

```
public function actualizar(Request $request, $id){  
    $product = Producto::findOrFail($id);  
    $product->nombreProducto = $request->input('nombrePro');  
    $product->cantidadProducto = $request->input('cantidadPro');  
    $product->precioProducto = $request->input('precioPro');  
    $product->fotoProducto = $request->input('fotoPro');  
    $product->categoria = $request->input('categoria');  
    $product->save();  
    return redirect()->route('listado_productos');  
}
```

### Eliminar Registro

De la misma forma la eliminación de un registro se hace de forma muy similar.

```
<a href="{{route('elimina_producto', $product->id)}}" class="btn btn-danger">Eliminar</a>  
  
Route::get('productos/eliminar/{id}', [ProductosController::class,  
    'eliminar'])  
    ->name('elimina_producto');  
  
public function eliminar($id){  
    $product = Producto::findOrFail($id);  
    $product->delete();  
    return redirect()->route('listado_productos');  
}
```



### Manejo de relaciones con Modelo de datos

Las tablas de la base de datos suelen estar relacionadas entre sí. Eloquent facilita la gestión y el trabajo con estas relaciones [5]. Utilizando el ORM el proceso es mucho más fácil, trabajando con las relaciones directamente dentro de los modelos (usando Programación Orientada a Objetos) y creando métodos personalizados que evitarán tener que construir consultas de forma manual [10].

La variedad de relaciones comunes que Eloquent tiene son [11]:

- Uno a muchos
- Muchos a muchos
- Tiene uno a través
- Tiene muchos a través
- Uno a uno (polimórfico)
- Uno a muchos (polimórfico)
- Muchos a muchos (polimórfico)

Las relaciones en Eloquent se definen como métodos dentro de la clase del Modelo. Dado que las relaciones también sirven como poderosos constructores de consultas, definir las relaciones como métodos proporciona potentes capacidades de encadenamiento y consulta.

#### Relación «Pertenece a» «Belongs\_to»

El método **belongsTo** nos permite trabajar con relaciones donde un registro pertenece a otro registro. Este método acepta como primer argumento el nombre de la clase que queremos vincular.

Eloquent determina el nombre de la llave foránea a partir del nombre del método (en este caso categoría) y agregando el sufijo `_id` a este:

**Ejemplo:** En el modelo Producto

```
class Producto extends Model {  
    protected $table = 'productos';  
  
    public function category()  
    {  
        return $this->belongsTo(Categoría::class);  
    }  
}
```

Si en la base de datos el nombre de la llave foránea no sigue esta convención es necesario pasar el nombre de la columna como segundo argumento así:

```
public function category()  
{  
    return $this->belongsTo(Categoría::class, 'categoría');  
}
```

Nombre Llave  
foranea



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Por otro lado, si el modelo padre no usa una columna id como su llave primaria o quieres relacionar el modelo a una columna diferente, puedes pasar un tercer argumento especificando el nombre de la columna que actuaría como llave del modelo padre:

```
public function category()
{
    return $this->belongsTo(Categoría::class, 'categoria', 'id');
}
```

Nombre Llave	Llave primaria
Foranea	Modelo Padre

Hecho esto, utilizando cualquiera de las formas anteriores, podemos obtener la Categoría del producto así:

```
$producto = Producto::first();
>>> $producto->category->nombrecategoría;
=> "Dispensa"
```

### Relaciones uno a muchos conhasMany

Una relación uno a muchos es utilizada cuando un modelo puede tener muchos otros modelos relacionados. Por ejemplo, una categoría puede tener un número indeterminado de productos asociados a ésta. Dentro del modelo Categoría podemos decir que una categoría tiene muchos productos:

```
class Categoría extends Model {
    protected $table = 'categorías';

    public function product() {
        return $this->hasMany(Producto::class, 'id');
    }
}
```

Ahora podemos obtener todos los productos de una Categoría:

```
>>> $categoría = Categoría::first();
=> App\Models\Categoría {#3277
    id: 1,
    nombrecategoría: "Dispensa",
    descripción: "Mercado",
    created_at: "2020-12-10 01:46:42",
    updated_at: "2020-12-10 01:46:42",
}
>>> $categoría->product;
=> Illuminate\Database\Eloquent\Collection {#3266
    all: [
        App\Models\Producto {#3258
            id: 1,
            nombreProducto: "Arroz Diana Kilo",
            cantidadProducto: 55,
            precioProducto: 2900.0,
            categoría: 1,
            fotoProducto: "1.jpg",
            created_at: "2020-12-10 01:47:52",
            updated_at: "2020-12-10 03:18:16",
        }
    ]
}
```

Si quiere profundizar revise material disponible en:  
<https://styde.net/relaciones-con-el-orm-eloquent/>



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### TINKER

Tinker es una consola de comandos con todas las clases y métodos de la aplicación. Tinker le permite interactuar con toda su aplicación Laravel, incluidos sus modelos, trabajos, eventos y más de Eloquent [13]. Para ingresar al entorno de programación interactivo Tinker, ejecute el comando Tinker Artisan y aparecerá en pantalla una línea similar a esta:

```
PS C:\xampp\htdocs\cursito> php artisan tinker
Psy Shell v0.10.4 (PHP 7.4.11 - cli) by Justin Hileman
>>> █
```

Esto indica que estamos en la consola de **tinker** y desde ahora podemos ejecutar todos los métodos de nuestra aplicación.

#### Sintaxis

En esta herramienta se puede escribir código en PHP y ejecutarlo, los llamados a las variables y funciones se hacen de la misma forma. Veamos un ejemplo:

```
>>> $pesos = 20000
=> 20000
>>> echo $pesos
20000↵
>>> $total = 2000
=> 2000
>>> $total = $total + $pesos
=> 22000
```

#### Operaciones con Modelos desde Tinker

##### Retornar todos los registros

Utilizando el método `all()` retornamos todos los registros asociados a un modelo. Para el ejemplo emplearemos el Modelo Categoría.

```
>>> $category = Categoria::all()
[!] Aliasing 'Categoria' to 'App\Models\Categoria' for this Tinker session.
=> Illuminate\Database\Eloquent\Collection {#3276
    all: [
        App\Models\Categoria {#3277
            id: 1,
            nombreCategoria: "Dispensa",
            descripcion: "Mercado",
            created_at: "2020-12-10 01:46:42",
            updated_at: "2020-12-10 01:46:42",
        },
        App\Models\Categoria {#3278
            id: 2,
            nombreCategoria: "Bebidas y licores",
            descripcion: "Jugos y Gaseosos",
            created_at: "2020-12-10 01:46:58",
            updated_at: "2020-12-10 01:46:58",
        },
        App\Models\Categoria {#3279
            id: 3,
            nombreCategoria: "Dulceria",
            descripcion: "Dulces",
            created_at: "2020-12-10 01:47:15",
            updated_at: "2020-12-10 01:47:15",
        },
    ],
}
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Los métodos all() y get() en Eloquent retornan colecciones (objetos de la clase Illuminate\Database\Eloquent\Collection) las cuales «envuelven» el array de resultados y proveen de funciones adicionales [10], por ejemplo:

Método	Descripción
<pre>&gt;&gt;&gt; \$category-&gt;first() =&gt; App\Models\Categoría {#3277     id: 1,     nombreCategoria: "Dispensa",     descripción: "Mercado",     created_at: "2020-12-10 01:46:42",     updated_at: "2020-12-10 01:46:42", }</pre>	Obtiene el primer resultado de la Colección
<pre>&gt;&gt;&gt; \$category-&gt;last() =&gt; App\Models\Categoría {#3279     id: 3,     nombreCategoria: "Dulcería",     descripción: "Dulces",     created_at: "2020-12-10 01:47:15",     updated_at: "2020-12-10 01:47:15", }</pre>	Obtiene el ultimo resultado de la Colección
<pre>&gt;&gt;&gt; \$category-&gt;random(1) =&gt; Illuminate\Database\Eloquent\Collection {#3267     all: [         App\Models\Categoría {#3277             id: 1,             nombreCategoria: "Dispensa",             descripción: "Mercado",             created_at: "2020-12-10 01:46:42",             updated_at: "2020-12-10 01:46:42",         },     ], }</pre>	Obtiene un resultado aleatorio de la Colección

Estas funciones de la clase Collection no generan nuevas consultas SQL sino que operan sobre los resultados ya encontrados.

### Seleccionar un campo con el método pluck()

Utilizando el método pluck() permite retornar una nueva colección que contenga un listado de un solo campo en vez de un listado de objetos [10]. Por ejemplo, podemos obtener solo el campo nombreCategoria de la siguiente forma:

```
>>> $category->pluck('nombreCategoria');
=> Illuminate\Support\Collection {#3262
    all: [
        "Dispensa",
        "Bebidas y licores",
        "Dulcería",
    ],
}
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

**Nota:** No todos los modelos de Eloquent retornan colecciones. En este caso particular se obtiene un objeto de la clase App/Models/Producto

```
>>> $producto = Producto::where('categoria', 1)->first()
=> App\Models\Producto {#3283} → Obtiene un Objeto del Modelo
    id: 1,
    nombreProducto: "Arroz Diana Kilo",
    cantidadProducto: 55,
    precioProducto: 2900.0,
    categoria: 1,
    fotoProducto: "1.jpg",
    created_at: "2020-12-10 01:47:52",
    updated_at: "2020-12-10 03:18:16",
}
```

Así mismo puede lograr la consulta con Query Builder, pero el resultado no un Modelo de Eloquent sino un objeto de la Clase Estándar de PHP.

```
>>> DB::table('productos')->where('categoria', 1)->first()
=> {#3299
    +"id": 1,
    +"nombreProducto": "Arroz Diana Kilo",
    +"cantidadProducto": 55,
    +"precioProducto": 2900.0,
    +"categoria": 1,
    +"fotoProducto": "1.jpg",
    +"created_at": "2020-12-10 01:47:52",
    +"updated_at": "2020-12-10 03:18:16",
}
```

### DECLARAR MÉTODOS EN EL MODELO

Con Eloquent se puede declarar métodos dentro de un modelo y utilizarlos cuando interactuemos con los objetos de estos modelos.

#### Métodos no estáticos:

Asociamos un método para ser utilizado sobre un objeto (que representa un registro de la base de datos).

```
class Producto extends Model {
    protected $table = 'productos';

    public function category()
    {
        return $this->belongsTo(Categoría::class, 'categoria', 'id');
    }

    public function hasStock(){
        return $this->cantidadProducto >= '50';
    }
}
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

En este caso hasStock() devuelve true si la cantidad del producto es igual o mayor al valor con el que está siendo comparado.

Luego para acceder al método simplemente basta con declarar un objeto del Modelo así:

### Resultado True

```
>>> $producto = Producto::where('categoria', 1)->first()
=> App\Models\Producto {#3286
    id: 1,
    nombreProducto: "Arroz Diana Kilo",
    cantidadProducto: 55,
    precioProducto: 2900.0,
    categoria: 1,
    fotoProducto: "1.jpg",
    created_at: "2020-12-10 01:47:52",
    updated_at: "2020-12-10 03:18:16",
}
>>> $producto->hasStock() → Invoca al metodo hasStock()
=> true → Resultado
```

### Resultado False

```
>>> $producto = Producto::where('categoria', 3)->first()
=> App\Models\Producto {#3280
    id: 6,
    nombreProducto: "Chocolatina Jet",
    cantidadProducto: 20,
    precioProducto: 1800.0,
    categoria: 3,
    fotoProducto: "9.jpg",
    created_at: "2020-12-14 17:06:10",
    updated_at: "2020-12-14 17:06:10",
}
>>> $producto->hasStock() → Invoca al método hasStock
=> false → Resultado
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### Métodos estáticos:

Asociamos un método a la clase del modelo como tal, la cual representa el acceso a una tabla de la base de datos. Estos métodos son usados típicamente para consultas.

Ejemplo en el Modelo Producto creamos un método estático el cual se llama findPrice() el cual recibe un valor como parámetro.

```
public static function findPrice($price){  
    return static::where('precioProducto', '=', $price)->first();  
}
```

Luego se podrá usar dicho método para buscar a un producto por el precio y obtener un objeto Producto como resultado así:

```
>>> $producto = Producto::findPrice(1800)  
=> App\Models\Producto {#3274  
    id: 6,  
    nombreProducto: "Chocolatina Jet",  
    cantidadProducto: 20,  
    precioProducto: 1800.0,  
    categoria: 3,  
    fotoProducto: "9.jpg",  
    created_at: "2020-12-14 17:06:10",  
    updated_at: "2020-12-14 17:06:10",  
}
```

Resultado

O **null** si no se encuentra ningún registro)

```
>>> $producto = Producto::findPrice(900)  
=> null → Resultado
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### XI. AUTENTICACIÓN EN LARAVEL

Laravel se esfuerza por brindar las herramientas que necesita para implementar la autenticación de manera rápida, segura y sencilla.

El archivo de configuración de autenticación de su aplicación se encuentra en config/auth.php. Este archivo contiene varias opciones bien documentadas para modificar el comportamiento de los servicios de autenticación de Laravel.

The screenshot shows the Visual Studio Code interface with the auth.php file open in the editor. The file is located in the config directory of a Laravel project named 'CURSITO'. The code in auth.php defines a 'defaults' array for the 'web' guard, which points to the 'users' provider. The code is well-commented, explaining the purpose of each section.

```
auth.php - coursito - Visual Studio Code
E Edit Selection View Go Run Terminal Help
EXPLORER ... auth.php x
config > auth.php
1 <?php
2
3 return [
4
5 /*
6 |-----
7 | Authentication Defaults
8 |-----
9 |
10 | This option controls the default authentication "guard" and password
11 | reset options for your application. You may change these defaults
12 | as required, but they're a perfect start for most applications.
13 |
14 */
15
16 'defaults' => [
17     'guard' => 'web',
18     'passwords' => 'users',
19 ],
20 /*
21 |-----
22 | Authentication Guards
23 |-----
24 |
25 |
26 | Next, you may define every authentication guard for your application.
27 | Of course, a great default configuration has been defined for you
28 | here which uses session storage and the Eloquent user provider.
29 |-----
```

#### Laravel Breeze: Paquetes de Autenticación

Laravel Breeze es un nuevo paquete para Laravel, disponible a partir de la versión 8.0 que permite añadir el sistema de autenticación al igual que hacíamos en versiones anteriores de Laravel con el paquete ui o con el comando –auth.

Laravel Breeze es una implementación mínima y simple de todas las funciones de autenticación de Laravel, que incluyen inicio de sesión, registro, restablecimiento de contraseña, verificación de correo electrónico y confirmación de contraseña. La capa de vista de Laravel Breeze está formada por plantillas Blade simples diseñadas con Tailwind CSS. Breeze proporciona un maravilloso punto de partida para comenzar una nueva aplicación de Laravel [11].

#### Instalación de Laravel Breeze

**Nota:** Para instalar Laravel Breeze se recomienda trabajar desde un nuevo proyecto.

1. Crear un proyecto laravel y ejecutar las migraciones

```
composer create-project laravel/laravel tienda --prefer-dist
cd tienda
php artisan migrate
```

2. Instala el paquete laravel/breeze usando composer

```
composer require laravel/breeze --dev
```

3. Seguidamente, ejecuta el siguiente comando para instalar Breeze:

```
php artisan breeze:install
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

4. A continuación, vamos a instalar los paquetes de Node necesarios.

Nota: Para ejecutar este comando debe tener instalado Node.js en su máquina.

```
npm install
```

5. Y finalmente vamos a compilar los archivos. scss y .js mediante el siguiente comando:

```
npm run dev
```

A continuación, puede navegar a las URL / login o / register de su aplicación en su navegador web. Todas las rutas de Breeze están definidas dentro del archivo routes/auth.php

### Resultado:

The screenshot shows two consecutive registration/login pages from a Laravel application. The first page is a registration form titled 'register' at the top. It features a logo at the top center, followed by four input fields labeled 'Name', 'Email', 'Password', and 'Confirm Password'. Below these fields are two buttons: 'Already registered?' and a dark 'REGISTER' button. The second page is a login form titled 'login' at the top. It also features a logo at the top center, followed by two input fields labeled 'Email' and 'Password'. Below these fields is a checkbox labeled 'Remember me'. At the bottom of the form are two buttons: 'Forgot your password?' and a dark 'LOGIN' button. Both pages have standard browser navigation buttons (back, forward, stop) and a URL indicator.

Las rutas que se utilizan para el login y el registro de los usuarios se encuentran en el archivo **routes/auth.php**. En cuanto a las rutas estándar que no requieren autenticación, podrás encontrarlas en el archivo **routes/web.php**.



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Observando los resultados en el archivo **routes/auth.php**

```
routes > routes/auth.php
1  <?php
2
3  use App\Http\Controllers\Auth\AuthenticatedSessionController;
4  use App\Http\Controllers\Auth\ConfirmablePasswordController;
5  use App\Http\Controllers\Auth\EmailVerificationNotificationController;
6  use App\Http\Controllers\Auth\EmailVerificationPromptController;
7  use App\Http\Controllers\Auth\NewPasswordController;
8  use App\Http\Controllers\Auth>PasswordResetLinkController;
9  use App\Http\Controllers\Auth\RegisteredUserController;
10 use App\Http\Controllers\Auth\VerifyEmailController;
11 use Illuminate\Support\Facades\Route;
12
13 Route::get('/register', [RegisteredUserController::class, 'create'])
14 |     ->middleware('guest')
15 |     ->name('register');
16
17 Route::post('/register', [RegisteredUserController::class, 'store'])
18 |     ->middleware('guest');
19
20 Route::get('/login', [AuthenticatedSessionController::class, 'create'])
21 |     ->middleware('guest')
22 |     ->name('login');
23
24 Route::post('/login', [AuthenticatedSessionController::class, 'store'])
25 |     ->middleware('guest');
26
27 Route::get('/forgot-password', [PasswordResetLinkController::class, 'create'])
28 |     ->middleware('guest')
29 |     ->name('password.request');
30
31 Route::post('/forgot-password', [PasswordResetLinkController::class, 'store'])
32 |     ->middleware('guest')
33 |     ->name('password.email');
34
35 Route::get('/reset-password/{token}', [NewPasswordController::class, 'create'])
36 |     ->middleware('guest')
37 |     ->name('password.reset');
38
39 Route::post('/reset-password', [NewPasswordController::class, 'store'])
40 |     ->middleware('guest')
41 |     ->name('password.update');
```

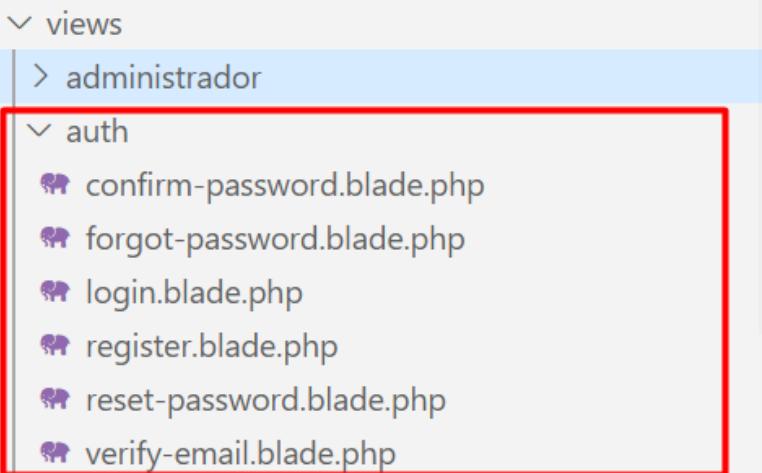
De la misma manera en **app/Http/Controllers** encuentra el listado de controladores que se crearon para el proceso de autenticación y los cuales son encargados de realizar las tareas de registro, login y gestión de contraseñas.

```
▽ app
  > Console
  > Exceptions
  ▽ Http
    ▽ Controllers
      ▽ Auth
        ▪ AuthenticatedSessionController.php
        ▪ ConfirmablePasswordController.php
        ▪ EmailVerificationNotificationController.php
        ▪ EmailVerificationPromptController.php
        ▪ NewPasswordController.php
        ▪ PasswordResetLinkController.php
        ▪ RegisteredUserController.php
        ▪ VerifyEmailController.php
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Las vistas correspondientes a la gestión del registro, **login** y de las contraseñas de los usuarios están en el directorio **resources/views/auth/** al igual que siempre:



La única diferencia en las vistas con respecto a Laravel UI es que ahora usan Tailwind en lugar de Bootstrap.

De la misma forma en el archivo **routes/web.php** inicialmente se crearon unas rutas de las cuales el desarrollador decidirá los cambios que deseé efectuar con ellas o añadir nuevas para su aplicación.

```
routes > 🗂️ web.php
    /   | WEB Routes
    8   |
    9   |
    10  | Here is where you can register web routes for your application. These
    11  | routes are loaded by the RouteServiceProvider within a group which
    12  | contains the "web" middleware group. Now create something great!
    13  |
    14  */
    15
    16 Route::get('/', function () {
    17     return view('welcome');
    18 });
    19
    20 Route::get('/dashboard', function () {
    21     return view('dashboard');
    22 })->middleware(['auth'])->name('dashboard');
    23
    24 require __DIR__.'/auth.php';
    25
```



## MIDDLEWARE O FILTROS

El middleware o filtro proporciona un mecanismo conveniente para filtrar las solicitudes HTTP que ingresan a su aplicación. Se define como una clase PHP almacenada en un fichero dentro de la carpeta app/Http/Middleware. Cada middleware se encargará de aplicar

un tipo concreto de filtro y de decidir que realizar con la petición realizada: permitir su ejecución, dar un error o redireccionar a otra página en caso de no permitirla". [5]

└ app	
└ Controllers	
└ Middleware	
└ Authenticate.php	
└ EncryptCookies.php	
└ PreventRequestsDuringMaintenance...	
└ RedirectIfAuthenticated.php	
└ TrimStrings.php	
└ TrustHosts.php	
└ TrustProxies.php	
└ VerifyCsrfToken.php	
└ Kernel.php	

"Por ejemplo, Laravel incluye un middleware que verifica que el usuario de su aplicación esté autenticado. Si el usuario no está autenticado, el middleware redirigirá al usuario a la pantalla de inicio de sesión. Sin embargo, si el usuario está autenticado, el middleware permitirá que la solicitud continúe en la aplicación." [11]

Todos estos filtros los podemos encontrar en la carpeta **app/Http/Middleware**, los cuales los podemos modificar o ampliar su funcionalidad o si se desea crear los propios.

Un middleware puede realizar tareas antes o después de pasar la solicitud más profundamente a la aplicación.

### Crear un middleware

Para crear un nuevo middleware, use el comando **make: middleware** Artisan:

```
php artisan make:middleware Prueba
```

```
app > Http > Middleware > Prueba.php
1  <?php
2
3  namespace App\Http\Middleware;
4
5  use Closure;
6  use Illuminate\Http\Request;
7
8  class Prueba
9  {
10
11     /**
12      * Handle an incoming request.
13      *
14      * @param \Illuminate\Http\Request $request
15      * @param \Closure $next
16      * @return mixed
17      */
18
19     public function handle(Request $request, Closure $next)
20     {
21         return $next($request);
22     }
23 }
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

El código generado por Artisan ya viene preparado para que podamos escribir directamente la implementación del filtro a realizar dentro de la función `handle`.

Como podemos ver, esta función solo incluye el valor de retorno con una llamada a `return $next($request);`, que lo que hace es continuar con la petición y ejecutar el método que tiene que procesarla.

Como entrada la función `handle` recibe dos parámetros:

- `$request`: En la cual nos vienen todos los parámetros de entrada de la petición.
- `$next`: El método o función que tiene que procesar la petición.

### Middleware antes o después de la petición

Para hacer que el código de un Middleware se ejecute antes o después de la petición HTTP simplemente tenemos que poner nuestro código antes o después de la llamada a `$next($request)`. Por ejemplo, el siguiente \_Middleware realizaría la acción antes de la petición:

```
public function handle(Request $request, Closure $next)
{
    // Código a ejecutar antes de la petición
    return $next($request);
}
```

Mientras que el siguiente *Middleware* ejecutaría el código **después** de la petición:

```
public function handle(Request $request, Closure $next)
{
    $response = $next($request);
    // Código a ejecutar después de la petición
    return $response;
}
```

### Ejemplo:

Crear un filtro que redirija al home si el usuario tiene menos de 18 años y en otro caso que le permita acceder a la ruta:

```
public function handle(Request $request, Closure $next)
{
    if ($request->input('age') < 18) {
        return redirect('home');
    }
    return $next($request);
}
```

Como hemos dicho antes, podemos hacer tres cosas con una petición:

Si todo es correcto permitir que la petición continúe devolviendo: `return $next($request);`

Realizar una redirección a otra ruta para no permitir el acceso con:

```
return redirect('home');
```

Lanzar una excepción o llamar al método `abort` para mostrar una página de error:

```
abort(403, 'Unauthorized action.');
```



### Uso de *Middleware*

“Laravel permite la utilización de *Middleware* de tres formas distintas: global, asociado a rutas o grupos de rutas, o asociado a un controlador o a un método de un controlador. En los tres casos será necesario registrar primero el *Middleware* en la clase `app/Http/Kernel.php`” [5].

#### Middleware global

“Para hacer que un *Middleware* se ejecute con todas las peticiones HTTP realizadas a una aplicación simplemente lo tenemos que registrar en el array `$middleware` definido en la clase `app/Http/Kernel.php`” [5]. Por ejemplo:

```
protected $middleware = [
    // \App\Http\Middleware\TrustHosts::class,
    \App\Http\Middleware\TrustProxies::class,
    \Fruitcake\Cors\HandleCors::class,
    \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
    \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
    \App\Http\Middleware\TrimStrings::class,
    \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
    \App\Http\Middleware\Prueba::class
];
```

En este ejemplo hemos registrado la clase `Prueba` al final del array. Si queremos que nuestro middleware se ejecute antes que otro filtro simplemente tendremos que colocarlo antes en la posición del array.

#### Middleware asociado a rutas

En el caso de querer que nuestro middleware se ejecute solo cuando se llame a una ruta o a un grupo de rutas también tendremos que registrarla en el fichero `app/Http/Kernel.php`, pero en el array `$routeMiddleware` [5]. Al añadirlo a este array además tendremos que asignarle un nombre o clave, que será el que después utilizaremos asociarlo con una ruta. En primer lugar, añadimos nuestro filtro al array y le asignamos el nombre `"mayor_de_edad"`.

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'mayor_de_edad'=>\App\Http\Middleware\Prueba::class
];
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Una vez registrado nuestro *middleware* ya lo podemos utilizar en el fichero de rutas `app/Http/routes.php` mediante la clave o nombre asignado, Laravel también permite asociar los filtros con las rutas usando el método `middleware()` sobre la definición de la ruta de la forma [5], por ejemplo:

```
Route::get('/mayorEdad', [Administracion::class, 'mayorEdad'])
    ->middleware(['mayor_de_edad'])
    ->name('home');
```

### Middleware dentro de controladores

También es posible indicar el middleware a utilizar desde dentro de un controlador. En este caso los filtros también tendrán que estar registrados en el array `$routeMiddleware` del fichero `app/Http/Kernel.php`. Para utilizarlos se recomienda realizar la asignación en el constructor del controlador y asignar los filtros usando su clave mediante el método `middleware` [5].

```
class UserController extends Controller
{
    /**
     * Instantiate a new UserController instance.
     *
     * @return void
     */
    public function __construct()
    {
        // Filtrar todos los métodos
        $this->middleware('auth');

        // Filtrar solo estos métodos...
        $this->middleware('log', ['only' => ['fooAction', 'barAction']]);

        // Filtrar todos los métodos excepto...
        $this->middleware('subscribed', ['except' => ['fooAction', 'barAction']]);
    }
}
```

Puede ampliar la información de Middleware en:  
<https://laravel.com/docs/8.x/middleware#introduction>



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### XII. INSTALACIÓN DE PAQUETES EN LARAVEL

Para instalar un paquete se debe utilizar el comando de **composer**:

```
composer require <nombre-del-paquete-a-instalar>
```

O también editando el fichero composer.json de la raíz de nuestro proyecto, añadir el paquete en su sección "require", y por último ejecutar el comando:

```
sudo composer update
```

La lista de todos los paquetes disponibles la podemos encontrar en <https://packagist.org>, en la cual permite realizar búsquedas, ver el número de instalaciones de un paquete, etc.

#### Ejemplo #1: Instalar y configurar el paquete que permite trabajar con archivos tipo Excel en un proyecto Laravel.

Para ello vamos a seguir los pasos que se muestran en: <https://docs.laravel-excel.com/3.1/getting-started/installation.html>

1. Descargar el paquete

```
composer require maatwebsite/excel
```

The screenshot shows the Packagist website interface. At the top, there's a search bar with the placeholder "Search packages...". Below it, the package name "maatwebsite/excel" is displayed. Underneath the package name, there's a button labeled "composer require maatwebsite/excel". Below this button, a description reads "Supercharged Excel exports and imports in Laravel". The entire screenshot is framed by a red border.

2. Registrar Maatwebsite/Excel/ExcelServiceProvider como ServiceProvider en config/app.php

```
'providers' => [
    /*
     * Package Service Providers...
     */
    Maatwebsite\Excel\ExcelServiceProvider::class,
```

Nota: Los proveedores de servicios enumerados en esa sección se cargarán automáticamente en la solicitud de su aplicación.



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

3. Agregar manualmente el Facade de Excel en config/app.php:

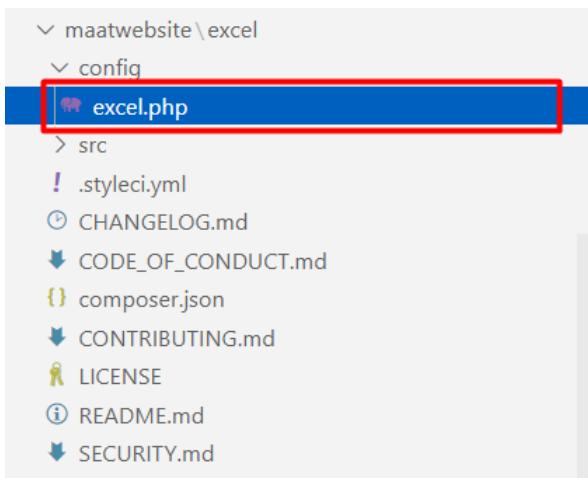
```
'aliases' => [
    'Excel' => Maatwebsite\Excel\Facades\Excel::class,
```

4. Publicar la configuración, ejecutando el comando de publicación del proveedor:

```
php artisan vendor:publish --provider="Maatwebsite\Excel\ExcelServiceProvider" --tag=config
```

```
PS C:\xampp\htdocs\cursito> php artisan vendor:publish --provider="Maatwebsite\Excel\ExcelServiceProvider" --tag=config
Copied File [\vendor\maatwebsite\excel\config\excel.php] To [\config\excel.php]
Publishing complete.
```

Esto creará un nuevo archivo de configuración llamado excel.php ubicado en /vendor/maatwebsite/Excel/config



### Configuración adicional

5. Crear una clase Export la cual va servir para encapsular el proceso de Exportación

```
php artisan make:export CategoriasExport --model=Categorias
```

En el anterior comando creamos un Export llamado CategoriasExport el cual va tomar el modelo Categorías para la exportación.

```
PS C:\xampp\htdocs\cursito> php artisan make:export CategoriasExport --model=Categoria
Export created successfully.
```

Esto creara una nueva carpeta dentro app llamada Exports.



El archivo creado contiene:



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

app > Exports > CategoriasExport.php

```
1  <?php
2
3  namespace App\Exports;
4
5  use App\Models\Categoria; → Escribir la ruta donde esta el modelo
6  use Maatwebsite\Excel\Concerns\FromCollection;
7
8  class CategoriasExport implements FromCollection Emplea la interfaz
9  {
10     /**
11      * @return \Illuminate\Support\Collection
12      */
13     public function collection()
14     {
15         return Categoria::all(); Seleccióna todos los registros del
16     } modelo Categoria
17 }
18
```

Finalmente, para utilizar la clase en un controlador seria de la siguiente forma:

app > Http > Controllers > CategoriasController.php

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\DB;
7  use App\Models\Categoria;
8
9  use App\Exports\CategoriasExport;
10 use Maatwebsite\Excel\Facades\Excel; Agregar la ubicación del archivo CategoriasExport
11 Y del Facades Excel
12
13 class CategoriasController extends Controller
14 {
15     public function exportar(){
16         return Excel::download(new CategoriasExport, 'categorias.xlsx');
17     }
18 }
```

Y la ruta que lo invoca:

```
Route::get('descarga', [CategoriasController::class, 'exportar'] )
    ->name('exportar_categorias');
```

De la misma forma el paquete también permite la importación de archivos Excel al proyecto Laravel véase en: <https://docs.laravel-excel.com/3.1/imports/>



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

**Ejemplo #2: Instalar y configurar el paquete que permite trabajar con archivos tipo PDF en un proyecto Laravel.**

Para ello vamos a seguir los pasos: <https://github.com/barryvdh/laravel-dompdf>

1. Descargar el paquete: `composer require barryvdh/laravel-dompdf`

2. Registrar Barryvdh\DomPDF como ServiceProvider en config/app.php

```
/*
 * Package Service Providers...
 */

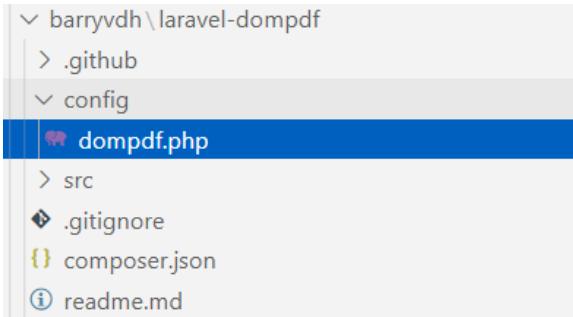
Maatwebsite\Excel\ExcelServiceProvider::class,
Barryvdh\DomPDF\ServiceProvider::class,
```

3. Agregar manualmente el Facade de Excel en config/app.php:

```
'aliases' => [
    'PDF' => Barryvdh\DomPDF\Facade::class,
```

4. Publicar la configuración, ejecutando el comando de publicación del proveedor:

```
php artisan vendor:publish --provider="Barryvdh\DomPDF\ServiceProvider"
```



A continuación, se podrá usar el paquete desde el Controlador

app > Http > Controllers > `CategoriasController.php`

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\DB;
7  use App\Models\Categoria;
8
9  use App\Exports\CategoriasExport;
10 use Maatwebsite\Excel\Facades\Excel;
11
12 use Barryvdh\DomPDF\Facade as PDF;
13
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Ahora bien, la ruta que lo invocara la función del controlador:

```
Route::get('descargarCategoriasPDF', [CategoriasController::class, 'descargarPDF'] )  
    ->name('descargar_categorias');
```

```
public function descargarPDF()  
{  
    $categorias = Categoria::all();  
    $pdf = \PDF::loadView('categorias.descargaPDF', ['categorias' => $categorias]);  
    return $pdf->download('categorias.pdf');  
}
```

---

La vista descargaPDF

```
<div align="center">  
    <h2> Listado de Categorias </h2>  
</div>  
    <table class="table" border="1" align="center">  
        <thead>  
            <tr>  
                <th scope="col">#</th>  
                <th scope="col">Nombre Categoria</th>  
                <th scope="col">Descripción </th>  
            </tr>  
        </thead>  
        <tbody>  
            @foreach($categorias as $c)  
                <tr>  
                    <td> {{ $c->id }} </td>  
                    <td> {{ $c->nombreCategoria }} </td>  
                    <td> {{ $c->descripcion }}</td>  
                </tr>  
            @endforeach  
        </tbody>  
    </table>
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

### XIII. SERVICIO API - REST CON LARAVEL

Para definir una API, mediante controladores, es recomendable emplear el fichero de rutas **routes/api.php** en lugar de **routes/web.php** el cual es un fichero similar al que se viene trabajando de forma convencional sin embargo es empleado para este tipo de acciones dentro de una aplicación [5].

#### RECIBIR DATOS DE LA API

Para crear una API se debe seguir los siguientes pasos:

1. Crear un controlador y dentro de este habrá una función la cual será destinada a concretar el procedimiento.
2. En la función se debe recibir una petición de tipo Request y recibir los datos que vendrán en formato JSON.

```
public function peticion(Request $request){  
    $nombre = $request->input('nombre');  
    $descripcion = $request->input('descripcion');  
    $resultado = "Categoria: $nombre , Descripcion: $descripcion";  
    return json_encode(array(  
        'status' => 200,  
        'response' => array(  
            'mensaje' => $resultado  
        )  
    ));  
}
```

Recibe los datos que vienen de la petición

Retorna la respuesta

3. Crear una ruta que direccione a la función previamente creada. La ruta será de tipo api Para ello desde el archivo **routes/api.php** se deberá realizar la acción:

```
routes >  api.php  
1  <?php  
2  
3  use Illuminate\Http\Request;  
4  use Illuminate\Support\Facades\Route;  
5  use App\Http\Controllers\CategoriesController;  
6  
7  /*  
8  |-----  
9  | API Routes  
10 |-----  
11 |  
12 | Here is where you can register API routes for your application. These  
13 | routes are loaded by the RouteServiceProvider within a group which  
14 | is assigned the "api" middleware group. Enjoy building your API!  
15 |  
16 */  
17  
18 Route::middleware('auth:api')->get('/user', function (Request $request) {  
19     return $request->user();  
20 });  
21  
22 Route::post('categorias/peticion', [CategoriesController::class, 'peticion']);  
23
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

Además, al ejecutar el comando `php artisan route:list` estas rutas aparecerán en el mismo listado.

```
| POST | api/categorias/peticion | recibeCategorias | App\Http\Controllers\CategoriasController@peticion | api
```

- Para verificar que el funcionamiento es el correcto podemos emplear un lector de API. En el caso particular se usará Insomnia:

```
POST http://127.0.0.1:8000/api/categorias/peticion
200 OK
{
  "status": 200,
  "response": {
    "mensaje": "Categoria: Frutas y Hortalizas , Descripcion: Frutas Frescas"
  }
}
```

Entrada:

```
POST http://127.0.0.1:8000/api/categorias/peticion
JSON
{
  "nombre": "Frutas y Hortalizas",
  "descripcion": "Frutas Frescas"
}
```

Salida:

```
{
  "status": 200,
  "response": {
    "mensaje": "Categoria: Frutas y Hortalizas , Descripcion: Frutas Frescas"
  }
}
```

Fuente: [14]



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL 8.0

De la misma forma si se desea que los datos enviados desde la API REST sean almacenados en la base de datos se puede lograr así:

```
public function peticion(Request $request){
    // Recibe el dato a insertar en categoria desde una petición API/REST -> Json

    $category = new Categoria();
    $category->nombreCategoria = $request->input('nombre');
    $category->descripcion = $request->input('descripcion');
    if($category->save()){
        $resultado = 'Insertado';
    }
    return json_encode(array(
        'status' => 200,
        'response' => array(
            'mensaje' => $resultado
        )
    ));
}
```

### ENVIAR DATOS A LA API

Asimismo, si queremos enviar una respuesta a la API con datos de nuestra aplicación se podrá lograr así:

```
public function obtenerJson(){
    $clientes = Clientes::all();
    return response()->json($clientes);
}
```

Y la ruta que invoca la función debe estar ubicada en el archivo `routes/api.php`

```
// Muestra el listado de clientes en un json
Route::get('clientes/json', [ClientesController::class, 'obtenerJson'])->name('clientes_json');
```

El resultado se puede visualizar en Insomnia:

The screenshot shows the Insomnia REST client interface. At the top, there's a header bar with 'GET ▾ http://127.0.0.1:8000/api/clientes/json'. Below it, a status bar shows '200 OK' with a green button, '445 ms', '1015 B', and '5 Minutes Ago'. The main area has tabs for 'Body ▾', 'Auth ▾', 'Query', 'Header', and 'Docs'. The 'Body' tab is selected and displays a JSON response. The response is a list of three client records, each with fields like id, nombreCliente, cedulaCliente, direccionCliente, telefonoCliente, generoCliente, fotoCliente, created\_at, and updated\_at. The JSON is as follows:

```
1 + [
2 +   {
3 +     "id": 1,
4 +     "nombreCliente": "Maria Linares Suarez",
5 +     "cedulaCliente": "1234",
6 +     "direccionCliente": "Carrera 22 A 18",
7 +     "telefonoCliente": "123",
8 +     "generoCliente": "F",
9 +     "fotoCliente": null,
10 +    "created_at": "2021-01-24T22:04:09.000000Z",
11 +    "updated_at": "2021-01-24T22:13:31.000000Z"
12 +  },
13 +  {
14 +    "id": 3,
15 +    "nombreCliente": "Juan Dominguez",
16 +    "cedulaCliente": "999",
17 +    "direccionCliente": "Carrera 22 A 18",
18 +    "telefonoCliente": "1233",
19 +    "generoCliente": "M",
20 +    "fotoCliente": null,
21 +    "created_at": "2021-01-24T22:14:26.000000Z",
22 +    "updated_at": "2021-01-24T22:14:26.000000Z"
23 +  },
24 +  {
25 +    "id": 4,
26 +    "nombreCliente": "Marcela Guerrero",
27 +    "cedulaCliente": "9885",
28 +    "direccionCliente": "Carrera 44 a",
29 +    "telefonoCliente": "7822",
30 +    "generoCliente": "F",
31 +    "fotoCliente": null,
32 +    "created_at": "2021-01-24T22:27:51.000000Z",
33 +    "updated_at": "2021-01-24T22:27:51.000000Z"
34 +  },
35 + ]
```



## SEMINARIO DE COMPUTACIÓN E INFORMATICA II

### LARAVEL 8.0

#### REFERENCIAS

- [1] desarroloweb.com, «Laravel,» [En línea]. Available: <https://desarrolloweb.com/home/laravel>. [Último acceso: 22 Septiembre 2020].
- [2] ¿QUÉ es LARAVEL y para qué sirve? - Frameworks de PHP. [Película]. 2019.
- [3] J. P. Mestras, «Estructura de las Aplicaciones Orientadas a Objetos, El patrón Modelo-Vista-Controlador (MVC),» [En línea]. Available: <https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf>. [Último acceso: 22 Septiembre 2020].
- [4] J. M. Aguilar., «¿Qué es el patrón MVC en programación y por qué es útil?,» Campusmvp, [En línea]. Available: <https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx>. [Último acceso: 22 Septiembre 2020].
- [5] A. J. G. Sanchez, «Laravel 5,» [En línea]. Available: <https://www.pdf-manual.es/programacion-web/php/149-curso-laravel-5.html>.
- [6] D. Rivera, «Patrón MVC en laravel,» Pleets Blog, 3 Julio 2019. [En línea]. Available: <https://blog.pleets.org/article/mvc-en-laravel>. [Último acceso: 5 Septiembre 2020].
- [7] D. Palacios, «Por qué Laravel NO es un framework MVC y tú deberías olvidarte de MVC,» Styde, 25 Abril 2017. [En línea]. Available: <https://styde.net/porque-laravel-no-es-mvc-y-tu-deberias-olvidarte-de-mvc/>.
- [8] Parzibyte's blog, «Solución a Error 419 en Laravel – Page Expired,» Parzibyte's blog, 4 Julio 2020. [En línea]. Available: <https://parzibyte.me/blog/2020/07/04/solucion-error-419-laravel-page-expired/>.
- [9] uniwebsidad.com, «Protección CSRF,» uniwebsidad.com, 2020. [En línea]. Available: <https://uniwebsidad.com/libros/symfony-2-4/capitulo-12/proteccion-csrf>.
- [10] Styde.net, «Curso de Laravel desde cero,» Styde.net, 9 Noviembre 2017. [En línea]. Available: <https://styde.net/introduccion-a-las-bases-de-datos-y-migraciones-con-laravel/>.
- [11] Laravel LLC, «Documentación de Laravel 8.0,» Laravel , [En línea]. Available: <https://laravel.com/docs/8.x/middleware#introduction>.
- [12] J. A. Muro, «¿Qué es un ORM?,» Deloitte, [En línea]. Available: <https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html>.
- [13] Jeff, «Tinker, la consola de comandos de Laravel,» Styde.net, 8 Mayo 2015. [En línea]. Available: <https://styde.net/tinker-consola-de-comandos-en-laravel/>.
- [14] Software Angel's, «Crear Servicio API - REST con LARAVEL - Json,» 2020.