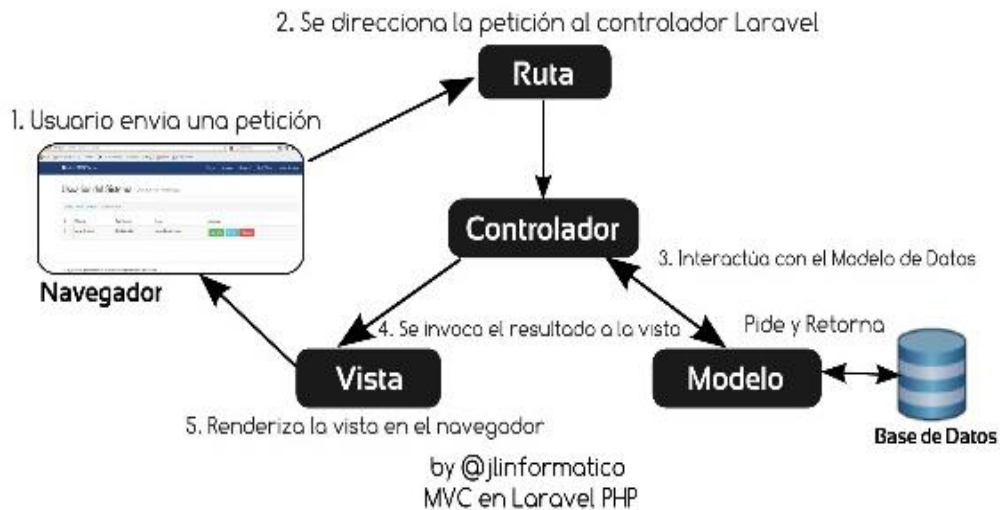


III. FUNCIONAMIENTO BÁSICO DEL FRAMEWORK

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 que 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

```
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
```

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 creará 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]

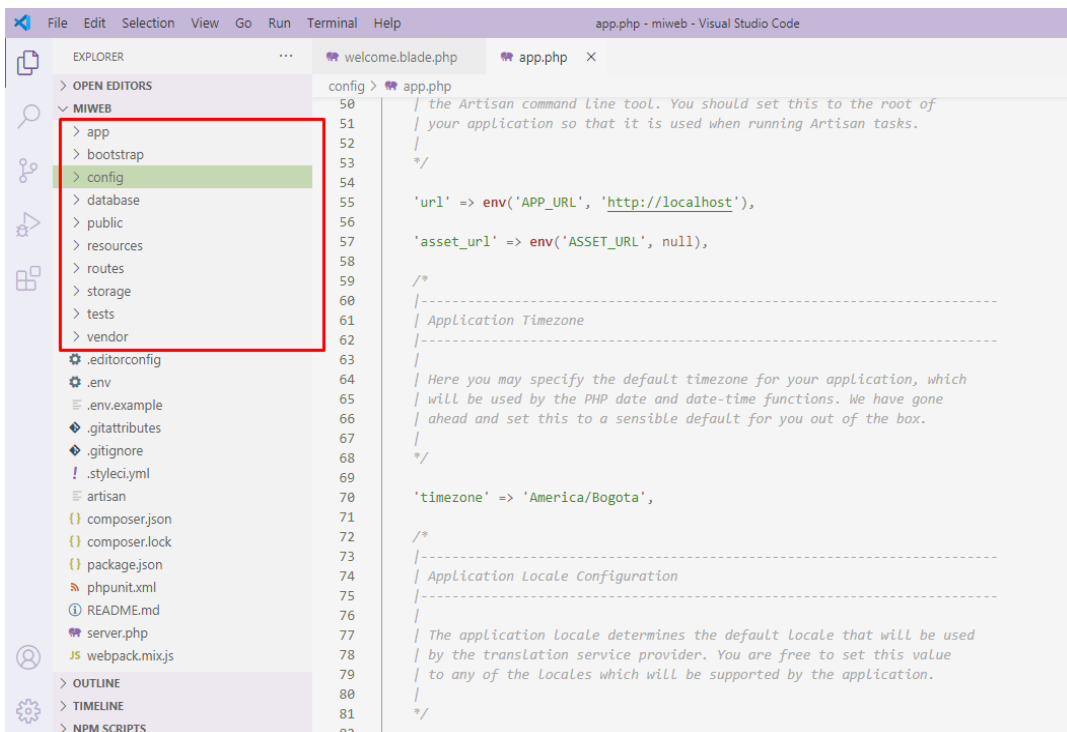


SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL

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...	1 KB
.env	01/08/2020 16:18	Archivo ENV	1 KB
.env.example	01/08/2020 15:10	Archivo EXAMPLE	1 KB
.gitattributes	01/08/2020 15:10	Documento de te...	1 KB
.gitignore	01/08/2020 15:10	Documento de te...	1 KB
.styleci.yml	01/08/2020 15:10	Archivo YML	1 KB
artisan	01/08/2020 15:10	Archivo	2 KB
composer.json	01/08/2020 15:10	Archivo JSON	2 KB
composer.lock	01/08/2020 15:15	Archivo LOCK	228 KB
package.json	01/08/2020 15:10	Archivo JSON	2 KB
phpunit	01/08/2020 15:10	Documento XML	2 KB
README.md	01/08/2020 15:10	Archivo MD	5 KB
server.php	01/08/2020 15:10	Archivo PHP	1 KB
webpack.mix	01/08/2020 15:10	Archivo JavaScript	1 KB

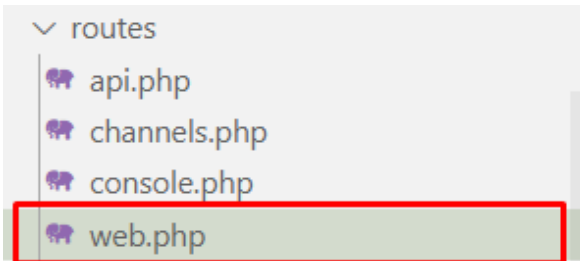


Directorio	Descripción	Subdirectorios
App	<p>Contiene el código principal de la aplicación, como son los controladores, filtros y modelos de datos.</p> <p>Nota: En la raíz de app también podemos encontrar el fichero User.php Este fichero es un modelo de datos que viene predefinido por Laravel para trabajar con los usuarios de la web, que incluye métodos para hacer login, registro, etc.</p>	<ul style="list-style-type: none"> - Console - Exceptions - Providers - Http: Dentro de este subdirectorio se encuentra dos carpetas importantes: <ul style="list-style-type: none"> • <u>Controllers:</u> 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. • <u>Middleware:</u> 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.
Bootstrap	<p>Se incluye el código que se carga para procesar cada una de las llamadas a nuestro proyecto. Incluye código para inicializar el framework.</p> <p>Se recomienda que en este archivo no se realicen cambios.</p>	Cache
Config	Maneja archivos de configuración de la aplicación	
Database	Incluye lo relacionado a la gestión de las bases de datos del proyecto	Factories Migrations: Migraciones respectivas a la base de datos Seeds: Cargar datos de prueba en las tablas
Public	<p>En este directorio pasan todas las peticiones y solicitudes. Adicionalmente se alojan los archivos CSS, JavaScript, imágenes u otros archivos que se deseen.</p>	Contiene el archivo index.php que es con el que inicia el aplicativo.
Resources	Contiene las vistas del proyecto	Views: Vistas de la aplicación mediante plantillas HTML
Routes	<p>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</p>	Archivos: Web.php: 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]

IV. MANEJO DE RUTAS

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

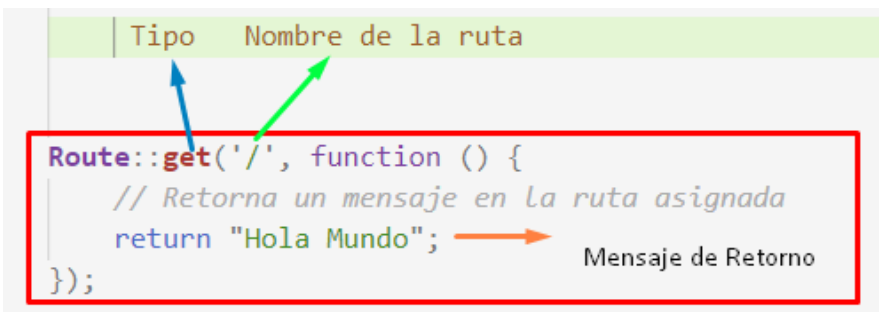


Las rutas, en su forma más sencilla, pueden devolver directamente un valor desde el propio fichero de rutas, pero también podrán generar la llamada a una vista o a un controlador.

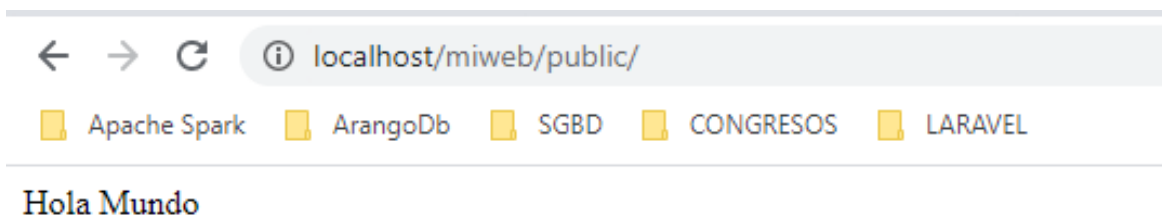
Si una ruta no ha sido definida saldrá el siguiente error:

404 | Not Found

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



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.



RUTAS QUE REFERENCIAN A UNA VISTA

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

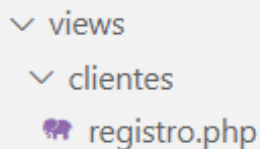
En este caso en la raíz del proyecto se cargará la vista denominada main.

```
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.



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');  
});
```

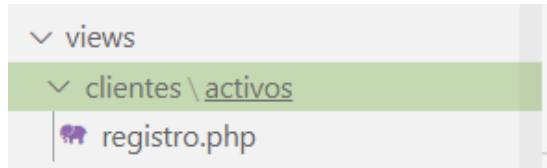
La url escrita en el navegador para acceder al contenido de dicha vista sería:





SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL

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:

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:

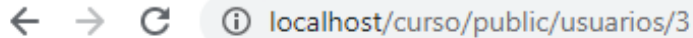
Opción 1:

```
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}";  
});
```


Cuyo resultado es:



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:



Mostrando detalle del usuario: nuevo

Esto se debe a que Laravel toma la primera ruta que coincida 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..

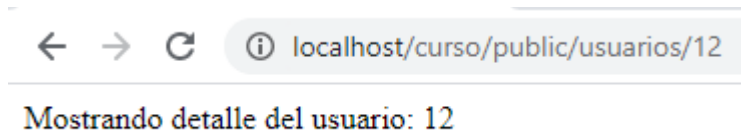
```
})->where('id', '[0-9]+');
```



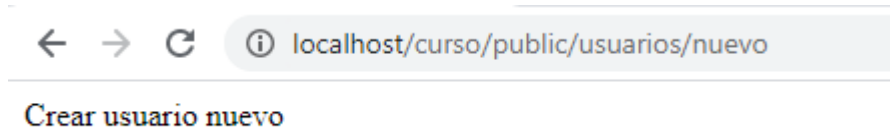
SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL

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

Con la ruta Usuarios/12



Con la ruta Usuarios/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 seria por él envío de un parámetro.

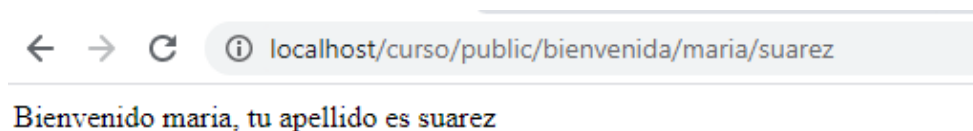
```
Route::get('/usuarios/nuevo', function () {  
    return "Crear usuario nuevo";  
});  
  
Route::get('/usuarios/{id}', function ($id) {  
    return "Mostrando detalle del usuario: {$id}";  
});
```

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





SEMINARIO DE COMPUTACIÓN E INFORMATICA II LARAVEL

Con un solo parametro en la ruta bienvenida.

← → ↻ ⓘ localhost/curso/public/bienvenida/maria

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 metodo o función dentro del controlador

Petición

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**

```
C:\xampp\htdocs\miweb>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