

 UNIVERSIDAD DON BOSCO	<p style="text-align: center;"><b>UNIVERSIDAD DON BOSCO</b>  <b>FACULTAD DE ESTUDIOS TECNOLÓGICOS</b>  <b>COORDINACIÓN DE COMPUTACIÓN Y MÓVILES</b></p>
<p style="text-align: center;"><b>CICLO 1</b></p>	<p style="text-align: center;"><b>Desarrollo de Aplicaciones Web</b>  <b>con Software Interpretado en el Servidor</b>  <b>Guía de práctica No. 11</b>  <b>Trabajar con Frameworks PHP</b></p>

## I. OBJETIVOS

Que el estudiante:

- Tenga una noción firme de lo que es un framework.
- Esté en capacidad de trabajar aplicaciones web haciendo uso de un framework.
- Adquiera habilidad obtener, instalar y configurar un framework PHP como Laravel.
- Realice una pequeña aplicación haciendo uso del framework Laravel.

## II. INTRODUCCION TEORICA

### Descripción

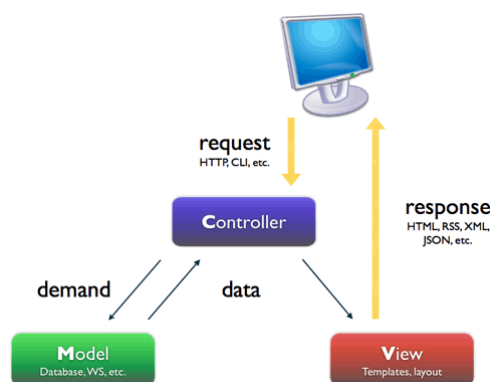
En la actualidad, el desarrollo de aplicaciones para la web y, en general, de las aplicaciones informáticas, demanda mucho tiempo de desarrollo y personal para su realización. Es por esto que se han desarrollado numerosos *frameworks* que permiten implementar las aplicaciones partiendo de una amplia base de librerías y herramientas que aportan todo un marco de trabajo para hacer conexiones a bases de datos, manejo de errores, control de archivos, etc. Con estas herramientas los desarrolladores solo tienen que preocuparse al cien por ciento por la lógica del negocio y no por desarrollar librerías complementarias que ya están disponibles en el *framework*. La mayor parte de estos *frameworks* están contruidos bajo el modelo MVC (**Modelo Vista Controlador**).

### ¿Qué es un *framework*?

Conceptualmente, *framework* es un **conjunto estandarizado de conceptos, prácticas y criterios** utilizados para **normalizar un tipo de problemática** particular que sirve para **enfrentar y resolver nuevos problemas** de índole similar.

Ahora bien, en el **desarrollo de software** o de una aplicación informática un *framework* constituye un **esquema o un patrón conceptual y tecnológico** definido con módulos de software concretos, con base en el cual puede organizarse otro proyecto de software más complejo.

El propósito del *framework* es contribuir al rápido desarrollo de las aplicaciones, permitiendo tiempos de desarrollo significativamente más cortos, en comparación al tiempo que requeriría desarrollarlas sin un framework.



Una de las razones que ha llevado al desarrollo de *frameworks* para distintos lenguajes, ha sido brindar a los programadores y diseñadores una **mejor organización y estructura** a los proyectos de software que contribuya a desarrollar aplicaciones informáticas con mayor rapidez y que facilite el mantenimiento con base en la organización de la aplicación.

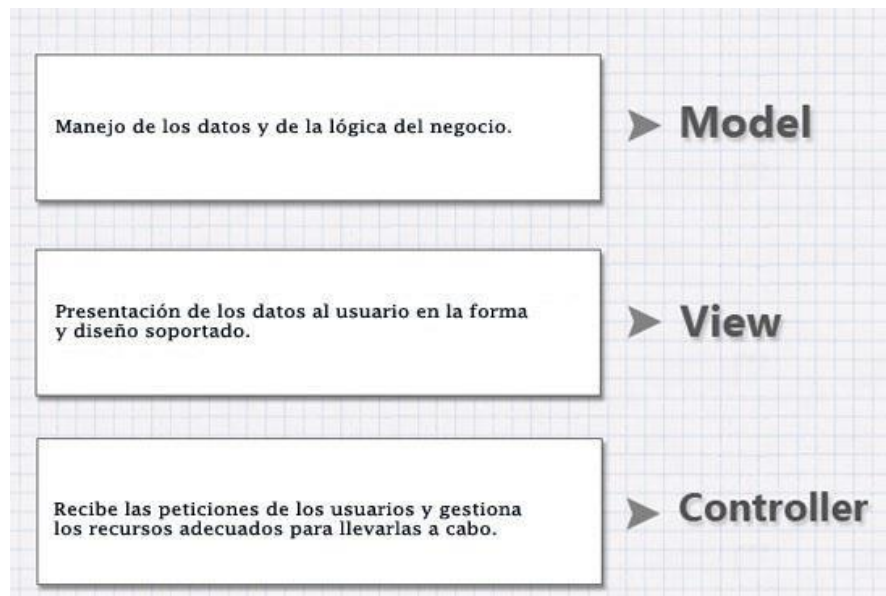
Por lo general, los *frameworks* utilizan modelos y paradigmas de programación bastante probados:

- Modelo Vista Controlador.
- Programación Orientada a Objetos.

### Modelo Vista-Controlador

El Modelo Vista Controlador es un patrón de programación que se fundamenta en la separación de la lógica del negocio del aspecto visual. Toda aplicación desarrollada bajo este esquema estará estructurada en tres capas: la capa de datos, la capa de interfaz y la capa lógica.

- **Modelo**: que procesa u obtiene datos, permitiendo gestionar la entrada y salida de la información almacenada en una base de datos.
- **Vista**: invocada desde el controlador y que representa la forma en que los datos son presentados en pantalla. En las aplicaciones para la web, es la encargada de mostrar las páginas html.
- **Controlador**: controla qué sucede en la aplicación, gestionando las peticiones, obteniendo los datos solicitados de un modelo, los procesa y se los envía a una vista para que puedan ser mostrados de forma adecuada.



### Razones para utilizar el patrón MVC

La razón de mayor peso para utilizar este patrón de programación es la **facilidad para el mantenimiento** del código en el futuro, ya que se encuentran separadas las secuencias de comando en modelo, vista y controlador, de modo que se hace fácil la localización de alguna funcionalidad, a la hora de desarrollar.


Además de esto, se puede decir que la **velocidad para el desarrollo** de las aplicaciones es otra de las razones de peso para utilizar un *framework* MVC.

### Frameworks de PHP

Existen numerosos *frameworks* de PHP, entre los que se pueden mencionar Zend Framework, Cake, Symfony, CodeIgniter, Akelos, Prado, ZooP, etc. Cada uno proporciona características diversas. Es difícil, sugerir cuál puede ser el mejor de todos, habría que conocer cada una. A la hora de decidir cuál utilizar, debe influir el conocimiento de las características que proporciona cada una. La siguiente imagen muestra un cuadro comparativo entre algunos de los Frameworks más conocidos y populares:

PHP Framework	PHP4	PHP5	MVC	Multiple DB's	ORM	DB Objects	Templates	Caching	Validation	Ajax	Auth Module	Modules
<a href="#">Akelos</a> 🇺🇸🇧🇷	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">ash.MVC</a> 🇺🇸	-	✓	✓	-	-	✓	✓	-	✓	-	✓	✓
<a href="#">CakePHP</a> 🇺🇸🇧🇷	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
<a href="#">CodeIgniter</a> 🇺🇸🇧🇷	✓	✓	✓	✓	-	✓	✓	✓	✓	-	-	-
<a href="#">DIY</a> 🇺🇸	-	✓	✓	-	✓	✓	✓	✓	-	✓	-	-
<a href="#">eZ Components</a> 🇺🇸	-	✓	-	✓	-	✓	✓	✓	✓	-	-	-
<a href="#">Fusebox</a> 🇺🇸	✓	✓	✓	✓	-	-	-	✓	-	✓	-	✓
<a href="#">PHP on TRAX</a> 🇺🇸	-	✓	✓	✓	✓	✓	-	-	✓	✓	-	✓
<a href="#">PHPDevShell</a> 🇺🇸	-	✓	✓	-	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">PhpOpenbiz</a> 🇺🇸	-	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	-
<a href="#">Prado</a> 🇺🇸🇧🇷	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">QPHP</a> 🇺🇸	✓	✓	✓	✓	-	✓	✓	-	✓	✓	✓	✓
<a href="#">Seagull</a> 🇺🇸	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">Symfony</a> 🇺🇸🇧🇷	-	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
<a href="#">WACT</a> 🇺🇸🇧🇷	✓	✓	✓	✓	-	✓	✓	-	✓	-	-	✓
<a href="#">WASP</a> 🇺🇸	-	✓	✓	-	-	✓	✓	-	✓	✓	✓	✓
<a href="#">Yii</a> 🇺🇸🇧🇷	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">Zend</a> 🇺🇸🇧🇷	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">ZooP</a> 🇺🇸	✓	✓	✓	✓	-	✓	✓	✓	✓	✓	✓	-

- **MVC:** Indicates whether the framework comes with inbuilt support for a Model-View-Controller setup.
- **Multiple DB's:** Indicates whether the framework supports multiple databases without having to change anything.
- **ORM:** Indicates whether the framework supports an object-record mapper, usually an implementation of ActiveRecord.
- **DB Objects:** Indicates whether the framework includes other database objects, like a TableGateWay.
- **Templates:** Indicates whether the framework has an inbuilt template engine.
- **Caching:** Indicates whether the framework includes a caching object or some way other way of caching.
- **Validation:** Indicates whether the framework has an inbuilt validation or filtering component.
- **Ajax:** Indicates whether the framework comes with inbuilt support for Ajax.
- **Auth Module:** Indicates whether the framework has an inbuilt module for handling user authentication.
- **Modules:** Indicates whether the framework has other modules, like an RSS feed parser, PDF module or anything else (useful).
- **EDP:** Event Driven Programming.

Framework	Creadores	Descripción
	<ul style="list-style-type: none"> <li>● Rasmus Lerdorf, creador de PHP.</li> <li>● Mantenido por la empresa EllisLab.</li> </ul>	<p>Entorno de desarrollo de código abierto que permite crear aplicaciones web dinámicas con PHP. Su principal objetivo es ayudar a que los desarrolladores puedan realizar aplicaciones informáticas orientadas a la web mucho más rápido que creando toda la estructura desde cero.</p>

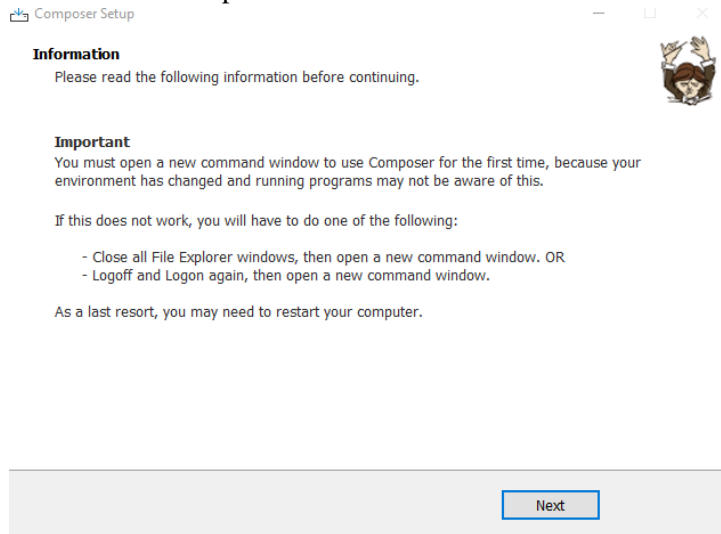
	<p>Michał Tatarynowicz desarrolla una versión compacta de un framework basado en los conceptos de Ruby on Rails. Es mantenido por una comunidad de desarrolladores en cakephp.org.</p>	<p>Es un <i>framework</i> para el desarrollo de aplicaciones web, escrito en PHP, creado sobre los conceptos de Ruby on Rails.</p>
	<ul style="list-style-type: none"> <li>• Zeev Suraski y Andi Gutmans, desarrolladores de PHP.</li> <li>• Mantenido por Zend Technologies, creadores y desarrolladores de PHP.</li> </ul>	<p><i>Framework</i> diseñado para desarrollar aplicaciones PHP basado en el modelo vista-controlador. Abreviado como ZF.</p>
	<ul style="list-style-type: none"> <li>• Fabien Potencier y actual director ejecutivo de Sensio Labs.</li> <li>• Patrocinado por Sensio Labs, compañía francesa que provee consultoría, servicios y formación en tecnología Open Source.</li> </ul>	<p>Completo <i>framework</i> PHP desarrollado en PHP 5.3 con el propósito de optimizar el desarrollo de aplicaciones web. Symfony es compatible con la mayor parte de gestores de bases de datos, como: MySQL, PostgreSQL, Oracle y Microsoft SQL Server.</p>
	<p>Framework de código abierto creado para desarrollar aplicaciones y servicios web con PHP 5. La filosofía de este framework es desarrollar aplicaciones de forma elegante y simple.</p>	<p>Fue creado en el 2011, con una gran influencia de otros Frameworks como Ruby On Rails, Sinatra e incluso Symfony y ASP.NET. Fue desarrollado por Taylor Otwell.</p>
	<p>Framework de código abierto, orientado a objetos, cuyas siglas Yii (Yes It Is!) significan ¡Si lo es!. Surge como alternativa a PRADO, superando muchos de los problemas presentados por este Framework. Yii es mucho más fácil y eficiente que PRADO.</p>	<p>Fue lanzado oficialmente el 3 de diciembre del 2008. Actualmente, se dispone de la versión 2.0 en beta. La última versión de la rama 1, está en la versión 1.1.14, lanzada el 11 de agosto del 2013. Este Framework es desarrollado por la compañía Yii Software LLC.</p>

## El framework Laravel.

## Instalación de Composer

Descargamos Composer, que permite administrador de paquetes de desarrollo para Laravel de la página oficial <https://getcomposer.org/download/>

Específicamente el instalador para Windows



## Instalación de Laravel

Al finalizar la instalación de Composer abrimos la línea de comandos para instalar Laravel de forma global y colocamos el comando *composer global require laravel/installer*

```
C:\> Select C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 10.0.19044.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jonas>composer global require laravel/installer
```

Al finalizar la instalación deberá salir un mensaje similar al siguiente

```
C:\WINDOWS\system32\cmd.exe

- Downloading symfony/polyfill-mbstring (v1.25.0)
- Downloading symfony/polyfill-intl-normalizer (v1.25.0)
- Downloading symfony/polyfill-intl-grapheme (v1.25.0)
- Downloading symfony/polyfill-ctype (v1.25.0)
- Downloading symfony/string (v5.4.3)
- Downloading symfony/deprecation-contracts (v2.5.1)
- Downloading psr/container (1.1.2)
- Downloading symfony/service-contracts (v2.5.1)
- Downloading symfony/polyfill-php73 (v1.25.0)
- Downloading symfony/console (v5.4.7)
- Downloading laravel/installer (v4.2.10)
- Installing symfony/polyfill-php80 (v1.25.0): Extracting archive
- Installing symfony/process (v5.4.7): Extracting archive
- Installing symfony/polyfill-mbstring (v1.25.0): Extracting archive
- Installing symfony/polyfill-intl-normalizer (v1.25.0): Extracting archive
- Installing symfony/polyfill-intl-grapheme (v1.25.0): Extracting archive
- Installing symfony/polyfill-ctype (v1.25.0): Extracting archive
- Installing symfony/string (v5.4.3): Extracting archive
- Installing symfony/deprecation-contracts (v2.5.1): Extracting archive
- Installing psr/container (1.1.2): Extracting archive
- Installing symfony/service-contracts (v2.5.1): Extracting archive
- Installing symfony/polyfill-php73 (v1.25.0): Extracting archive
- Installing symfony/console (v5.4.7): Extracting archive
- Installing laravel/installer (v4.2.10): Extracting archive
4 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Generating autoload files
11 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!

C:\Users\Jonas>
```

Navegamos a la carpeta donde crearemos el proyecto por ejemplo, en este ejemplo me muevo a C:/User/NombreUsuario/Desktop/EjemploLaravel y en esa carpeta ejecutamos el comando **laravel new nombreProyecto**

Iniciara la instalación de las dependencias de Laravel

```
C:\WINDOWS\system32\cmd.exe - laravel new EjemploLaravel

C:\Users\Jonas\Desktop\EjemploLaravel>laravel new EjemploLaravel

Laravel

Creating a "laravel/laravel" project at "C:/Users/Jonas/Desktop/EjemploLaravel"
Installing laravel/laravel (v8.6.11)
- Downloading laravel/laravel (v8.6.11): Extracting archive
Created project in C:/Users/Jonas/Desktop/EjemploLaravel/EjemploLaravel
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 110 installs, 0 updates, 0 removals
- Locking asm89/stack-cors (v2.1.1)
- Locking brick/math (0.9.3)
- Locking dflydev/dot-access-data (v3.0.1)
- Locking doctrine/inferno (2.0.4)
- Locking doctrine/instantiator (1.4.1)
- Locking doctrine/lexer (1.2.3)
- Locking dragonmantank/cron-expression (v3.3.1)
- Locking egulias/email-validator (2.1.25)
- Locking facade/flare-client-php (1.9.1)
- Locking facade/ignition (2.17.5)
- Locking facade/ignition-contracts (1.0.2)
```

Al finalizar aparecerá el siguiente mensaje

```
C:\WINDOWS\system32\cmd.exe

23/110 =====> 20%
33/110 =====> 30%
40/110 =====> 36%
44/110 =====> 40%
53/110 =====> 48%
55/110 =====> 50%
61/110 =====> 55%
65/110 =====> 59%
68/110 =====> 61%
71/110 =====> 64%
77/110 =====> 70%
85/110 =====> 77%
90/110 =====> 81%
99/110 =====> 90%
106/110 =====> 96%
109/110 =====> 99%
110/110 =====> 100%

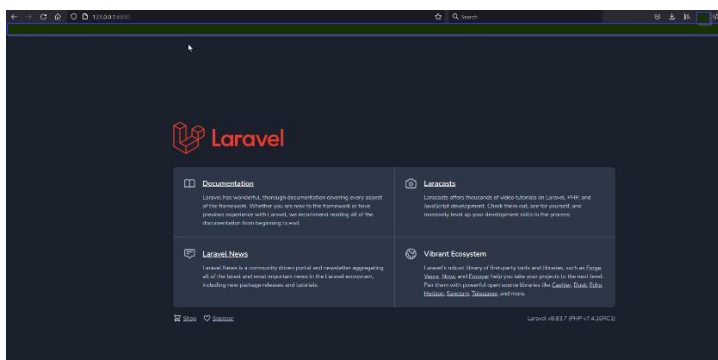
75 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Package swiftmailer/swiftmailer is abandoned, you should avoid using it. Use symfony/mailer instead.
Generating optimized autoload files
> @Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
77 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
> @php artisan key:generate --ansi

Application ready! Build something amazing.

C:\Users\Jonas\Desktop\EjemploLaravel>
```

Para ver si el proyecto se instaló correctamente utilizamos el comando **php artisan serve** dentro de la carpeta del proyecto y se accede a la dirección detallada en la consola de comandos (no cerrar la consola)

```
C:\Users\Jonas\Desktop\EjemploLaravel\EjemploLaravel>php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Tue Apr 5 17:46:21 2022] PHP 7.4.16RC1 Development Server (http://127.0.0.1:8000) started
```




### III. MATERIALES Y EQUIPO

Para la realización de la guía de práctica se requerirá lo siguiente:

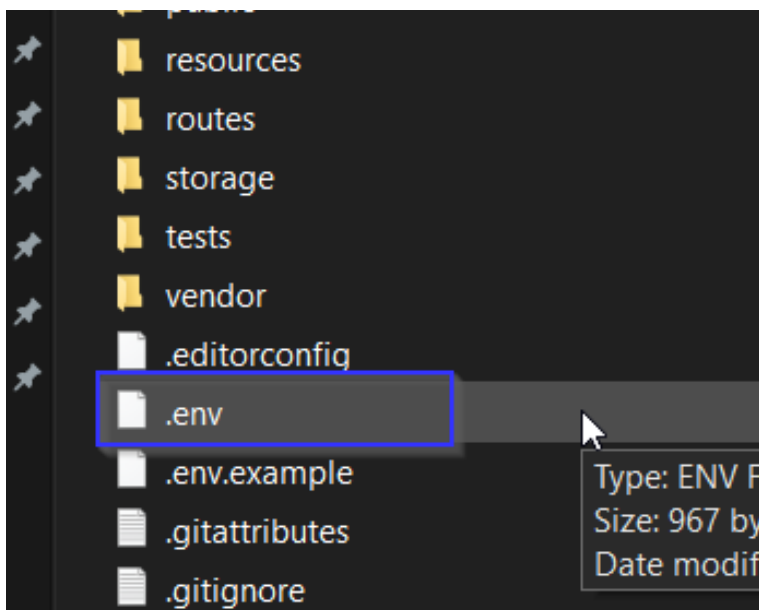
No.	Requerimiento	Cantidad
1	Guía de práctica #12: Trabajar con <i>frameworks</i> PHP	1
2	Computadora con Wamp Server y Sublime Text 3 instalado	1
3	Memoria USB o disco flexible	1

### IV. PROCEDIMIENTO

1. Creamos la base de datos llamada *EjemploLaravel* sin ninguna tabla dentro

►  `ejemplolaravel`

2. Abrimos el proyecto y buscamos dentro del proyecto el archivo `.env`



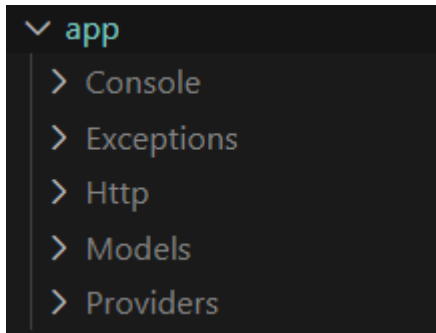
3. Editamos el archivo desde la línea 11 a 16 con los valores de la base de datos

```
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=ejemplolaravel
15 DB_USERNAME=root
16 DB_PASSWORD=
```



## Entendiendo la estructura de una aplicación con Laravel

1. Laravel es un Framework complejo que cuenta con muchos archivos de configuración y ordena la estructura en formato MVC
2. Carpeta App contiene toda la lógica de la aplicación, en esta se encuentran los modelos, controladores, entre otros



No se revisará a detalle cada carpeta pero se detallarán las que utilizaremos en este ejemplo.

**La carpeta Models** contiene la lógica del negocio, que reflejara la base de datos 1 a 1 generalmente

**La carpeta Http/Controllers** contiene los controladores que manejan las peticiones realizadas a la aplicación y que vista deberá entregar.

Un controlador tiene 7 métodos principales que permiten las operaciones de CRUD

- index() – Obtiene todos los registros de la base de datos
- show() – Obtiene un solo registro de la bd
- create() – redirecciona a la vista para mostrar el formulario (no funciona con API)
- store() – almacena en la base de datos la información
- index() – devuelve un formulario para editar un valor
- update() – realiza un update de un registro en la bd
- destroy() – elimina un registro en la bd

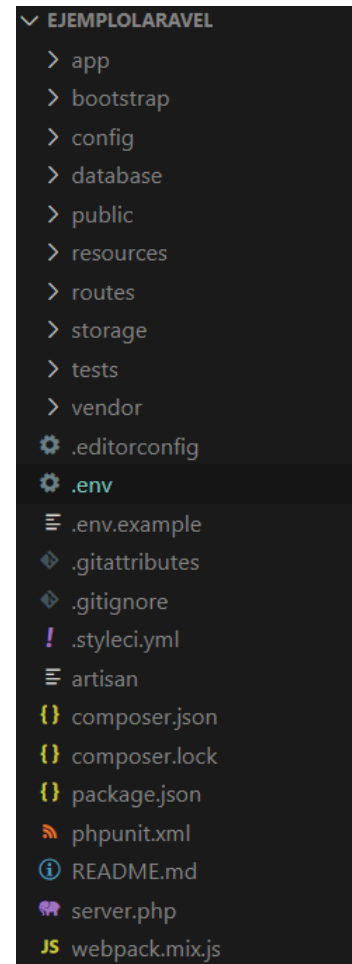
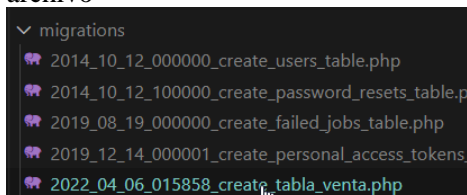
## Creemos un modelo

1. Creamos una migración con el comando

***php artisan make:migration create\_venta\_tabla***

```
C:\Users\Jonas\Desktop\EjemploLaravel\EjemploLaravel>php artisan make:migration create_venta_tabla --create=Venta
Created Migration: 2022_04_06_024341_create_venta_tabla
```

2. Buscamos dentro de la carpeta database/migrations la migración que se creó y veremos el siguiente archivo





**Nota: Borrar las migraciones que vienen de ejemplo para evitar que la base de datos cree esas tablas**

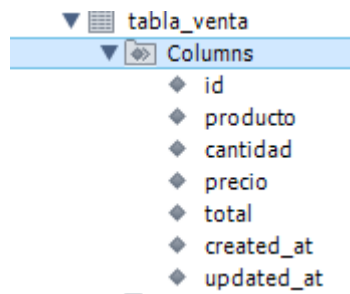
En el modificaremos la función up() con los siguientes valores

```
public function up()
{
    Schema::create('Ventas', function (Blueprint $table) {
        $table->id();
        $table->string('producto');
        $table->integer('cantidad');
        $table->decimal('precio');
        $table->decimal('total');
        $table->timestamps();
    });
}
```

En la línea de comandos escribimos el comando **php artisan migrate**

```
C:\Users\Jonas\Desktop\EjemploLaravel\EjemploLaravel>php artisan migrate
Migrating: 2022_04_06_015858_create_tabla_venta
Migrated: 2022_04_06_015858_create_tabla_venta (15.00ms)
```

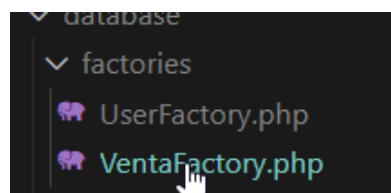
Esto creara dentro de la base de datos la tabla Venta con sus respectivas columnas



En la consola de Windows escribir el siguiente comando **php artisan make:model --factory Venta**

```
C:\Users\Jonas\Desktop\EjemploLaravel\EjemploLaravel>php artisan make:model --factory Venta
Model created successfully.
Factory created successfully.
```

Crearé un archivo Venta.php dentro del directorio App/Models y un factory asociado a el



Cambiamos los valores dentro del return

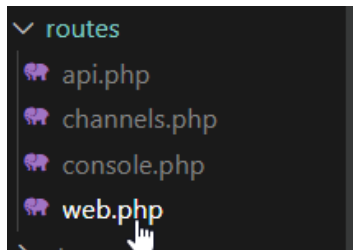
```
14     public function definition()  
15     {  
16         return [  
17             'producto' => $this->faker->producto(),  
18             'cantidad' => $this->faker->cantidad(),  
19             'precio' => $this->faker->precio(),  
20             'total' => $this->faker->total(),  
21         ];  
22     }
```

Realizamos una pequeña inserción por Query en la base de datos

```
insert into ventas(producto, cantidad, precio, total) values  
( 'Martillo mecanico',1,123.00,123.00),  
( 'Tornillos 12 pza',3,0.90,2.70)
```

## 5. Revisamos las rutas y las vistas

Vamos a la carpeta routes/web.php



En ese archivo modificaremos la línea 16 en adelante

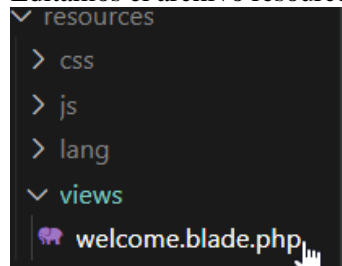
Original

```
16 Route::get('/', function () {  
17     return view('welcome');  
18 });
```

Cambios que se deben realizar, definimos una consulta de todas las ventas y las mandemos como parámetro a la página de inicio

```
16 Route::get('/', function () {  
17  
18     $ventas = \App\Venta::all();  
19  
20     return view('welcome',['ventas' => $ventas]);  
21 });
```

Editamos el archivo resources/views/welcome.blade.php



Borramos el body del documento y colocamos el siguiente código

```
<body>
<div class="flex-center position-ref full-height">
  @if (Route::has('login'))
    <div class="top-right links">
      @auth
        <a href="{{ url('/home') }}">Home</a>
      @else
        <a href="{{ route('login') }}">Login</a>
        <a href="{{ route('register') }}">Register</a>
      @endauth
    </div>
  @endif
  <div class="content">
    <div class="title m-b-md">
      Ejemplo Laravel
    </div>
    <div class="container">
      <table class="table">
        <thead>
          <tr>
            <th scope="col">Id</th>
            <th scope="col">Producto</th>
            <th scope="col">Cantidad</th>
            <th scope="col">Precio</th>
            <th scope="col">Total</th>
          </tr>
          <tbody>
            @foreach ($ventas as $venta)
              <tr>
                <th scope="row">{{ $venta->id }}
                <td>{{ $venta->producto }}</td>
                <td>{{ $venta->cantidad }}</td>
                <td>{{ $venta->precio }}</td>
                <td>{{ $venta->total }}</td>
              </tr>
            @endforeach
          </tbody>
        </table>
      </div>
    </div>
  </div>
</body>
```

Utilizamos el comando ***php artisan serve*** para ejecutar el servidor y nos mostrara los datos que se encuentran en la base de datos

```
C:\Users\Jonas\Desktop\EjemploLaravel\EjemploLaravel>php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Tue Apr  5 20:55:38 2022] PHP 7.4.16RC1 Development Server (http://127.0.0.1:8000) started
```

Recordar entrar al enlace que el comando indique desde un navegador

Laravel

Id	Producto	Cantidad	Precio	Total
1	Martillo mecanico	1	123.00	123.00
2	Tornillos 12 pza	3	0.90	2.70

A continuación crearemos un controlador para procesar las peticiones, para ello ejecutamos el siguiente comando:

**php artisan make:controller VentasController**

```
C:\xampp\htdocs\DSS\EjemploLaravel>php artisan make:controller VentasController
Controller created successfully.
```

Con el controlador ya creado, primero procederemos a modificar nuestro archivo de rutas, **router/web.php** :

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\VentasController;

/*
|-----
| Web Routes
|-----
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

/*Route::get('/', function () {
    $ventas = \App\Models\Venta::all();

    return view('welcome', ['ventas'=>$ventas]);
});*/

Route::get('/venta',[VentasController::class,'index']->name('venta.index'));

Route::get('/venta/create', [VentasController::class,'create']->name('venta.create'));

Route::post('/venta/create', [VentasController::class,'store']->name('venta.store'));
```

El siguiente paso será agregar las siguientes funciones al propio controlador

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Venta;

class VentasController extends Controller
{
    public function index(){
        $listaVentas = Venta::all();

        return view('venta.index',compact('listaVentas'));
    }

    public function create(){
        return view ('venta.create');
    }

    public function store(Request $request){
        $nuevaVenta = new Venta();

        $nuevaVenta->producto = $request->producto;
        $nuevaVenta->cantidad = $request->cantidad;
        $nuevaVenta->precio = $request->precio;
        $nuevaVenta->total = $request->total;

        $nuevaVenta->save();

        return redirect()->route('venta.index');
    }
}
```

A este punto ya tenemos el enrutamiento dentro de nuestro proyecto y un controlador que responda a dichas peticiones, solamente falta agregar las vistas, en primer lugar dentro de resources\views crear 2 carpetas: *layouts* y *venta*



Dentro de layouts crearemos nuestra primera plantilla de Blade, Blade es un motor de plantillas simple y a la vez poderoso proporcionado por Laravel. A diferencia de otros motores de plantillas populares de PHP, Blade no te impide utilizar código PHP plano en tus vistas. Crear el archivo **miPlantilla.blade.php** y agregar el siguiente contenido:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>@yield('titulo')</title>
    <style>
        .tabla{
            border-collapse: collapse;
```

```

        border: 1px solid;
    }

    .tabla td{
        border-collapse: collapse;
        border: 1px solid;
    }

    .tabla th{
        border-collapse: collapse;
        border: 1px solid;
    }
}
</style>
</head>
<body>
    @yield('contenido')
</body>
</html>

```

A continuación, crear dentro de la carpeta **venta** el archivo **index.blade.php** y agregar el siguiente código:

```

@extends('layouts.miPlantilla')
@section('titulo','Inicio')
@section('contenido')
    <a href="{{route('venta.create')}}">Nueva materia</a>
    <table border='1' style='border-collapse:collapse'>
    <thead>
        <tr>
            <th scope="col">Id</th>
            <th scope="col">Producto</th>
            <th scope="col">Cantidad</th>
            <th scope="col">Precio</th>
            <th scope="col">Total</th>
        </tr>
    </thead>
    <tbody>
        @foreach($listaVentas as $venta)
            <tr>
                <th scope="row">{{ $venta->id }}</th>
                <td>{{ $venta->producto }}</td>
                <td>{{ $venta->cantidad }}</td>
                <td>{{ $venta->precio }}</td>
                <td>{{ $venta->total }}</td>
                <td>
                    <a href="#" class="btn">Modificar</a>
                    <a href="#" class="btn">Eliminar</a>
                </td>
            </tr>
        @endforeach
    </tbody>
    </table>
</section>
</body>
</html>

```

```

        </tr>
    @endforeach
</tbody>
</table>
@endsection

```

Finalmente, dentro de la carpeta **venta** crear el archivo **create.blade.php** y agregar el siguiente contenido:

```

@extends('layouts.miPlantilla')
@section('titulo','Crear')
@section('contenido')
    <a href="{{route('venta.index')}}">Volver</a>
    <form method="post" action="{{route('venta.store')}}">
        @csrf <!--generacion automatica de token para peticiones de tipo POST -->
        Producto: <br>
        <input type="text" name="producto" /></br>
        Cantidad: <br>
        <input type="text" name="cantidad" /></br>
        Precio: <br>
        <input type="text" name="precio" /></br>
        Total: <br>
        <input type="text" name="total" /></br>
        <input type="submit" value="Enviar"/>
    </form>
@endsection

```

Ejecutaremos una vez mas el comando **php artisan serve**, pero esta vez accederemos a la url **localhost:8000/venta**

**NOTA:** tener en cuenta que el puerto puede variar, el puerto que utilizará en su equipo se indicará en consola una vez ejecutado el comando.



Id	Producto	Cantidad	Precio	Total	
1	Martillo mecanico	1	123.00	123.00	<a href="#">Modificar</a> <a href="#">Eliminar</a>
2	Tornillos 12 pza	3	0.90	2.70	<a href="#">Modificar</a> <a href="#">Eliminar</a>



← → ↻ ⓘ localhost:8000/venta/create

[Volver](#)

Producto:

Taladro Electrico

Cantidad:

2

Precio:

112

Total:

224

Enviar

← → ↻ ⓘ localhost:8000/venta

[Nueva materia](#)

Id	Producto	Cantidad	Precio	Total	
1	Martillo mecanico	1	123.00	123.00	<a href="#">Modificar</a> <a href="#">Eliminar</a>
2	Tornillos 12 pza	3	0.90	2.70	<a href="#">Modificar</a> <a href="#">Eliminar</a>
4	Taladro Electrico	2	112.00	224.00	<a href="#">Modificar</a> <a href="#">Eliminar</a>

Resultado personal:

[Nueva materia](#)

Id	Producto	Cantidad	Precio	Total	
1	Martillo mecanico	1	123.00	123.00	<a href="#">Modificar</a> <a href="#">Eliminar</a>
2	Tornillos 12 pza	3	0.90	2.70	<a href="#">Modificar</a> <a href="#">Eliminar</a>
3	pOLLO	22	1.45	20.00	<a href="#">Modificar</a> <a href="#">Eliminar</a>

**José Adrián López Medina**

*Técnico en Ingeniería en Computación*

Ciclo I - 2025

[Volver](#)

Producto:

Cantidad:

Precio:

Total:

Enviar

**José Adrián López Medina**

*Técnico en Ingeniería en Computación*

Ciclo I - 2025

## V. DISCUSIÓN DE RESULTADOS

1. Investigar como implementar Bootstrap a un proyecto de laravel, implementar las clases y componentes de Bootstrap dentro de las vistas para mejorar la presentación.
2. Basándose en el ejemplo realizado, implementar las funcionalidades para actualizar y eliminar, teniendo en cuenta que al actualizar la información previa debe cargarse en un formulario y al eliminar se debe mostrar algún tipo de ventana o mensaje de confirmación.

### Resultado personal:

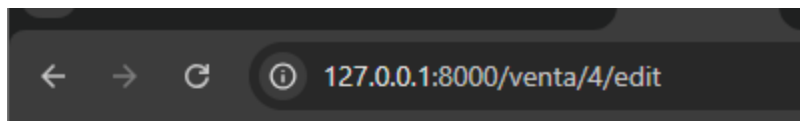


← → ↻ ⓘ 127.0.0.1:8000/venta ☆ ⬇ Incógnito ⋮

[Nueva materia](#)

Id	Producto	Cantidad	Precio	Total	
1	Martillo mecanico	1	123.00	123.00	<a href="#">Editar</a> <button>Eliminar</button>
2	Tornillos 12 pza	3	0.90	2.70	<a href="#">Editar</a> <button>Eliminar</button>
4	test	12	12.00	20.00	<a href="#">Editar</a> <button>Eliminar</button>

**José Adrián López Medina**  
*Técnico en Ingeniería en Computación*  
Ciclo I - 2025



## Editar Venta

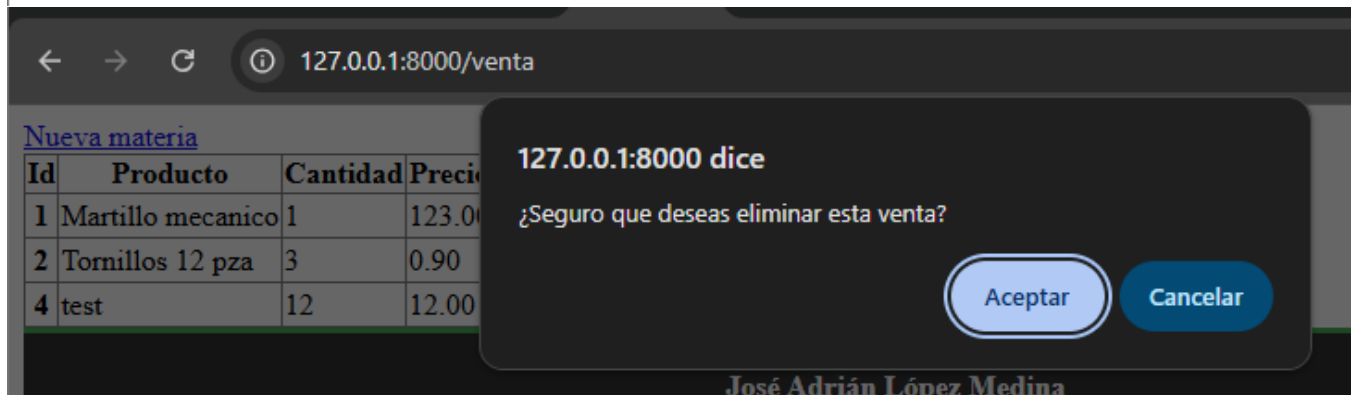
Producto

Cantidad

Precio

Total

[Cancelar](#)



## VI. BIBLIOGRAFIA

- Documentación oficial de Laravel <https://laravel.com/docs/7.x/installation>