

CREACIÓN DE UNA APLICACION WEB CON LARAVEL 8.0

En esta serie vamos a manejar lo siguiente:

1. Practica. Base de datos y Migraciones en Laravel 8.0
2. Practica. Creando rutas y controladores para el modelo
3. Practica. Insertar un nuevo registro en la base de datos
4. Practica. Listar los datos desde la base de datos
5. Practica. Validaciones con Input
6. Practica. Editar registros de la base de datos
7. Practica. Eliminar registros de la base de datos

1. Practica. Base de datos y Migraciones en Laravel 8.0

Creación de la BDD en Mysql – configuración del archivo .env

Crear una BD en MySQL <https://laravel.com/docs/8.x/migrations>



Tabla	Acción
<input type="checkbox"/> categorias	★ Examinar Estructura Buscar Insertar Vaciar Eliminar
<input type="checkbox"/> failed_jobs	★ Examinar Estructura Buscar Insertar Vaciar Eliminar
<input type="checkbox"/> migrations	★ Examinar Estructura Buscar Insertar Vaciar Eliminar
<input type="checkbox"/> password_resets	★ Examinar Estructura Buscar Insertar Vaciar Eliminar
<input type="checkbox"/> users	★ Examinar Estructura Buscar Insertar Vaciar Eliminar
5 tablas	Número de filas

Configuración del archivo .env

Digitamos el nombre de la **base de datos** que creamos

```
> tests 9 DB_CONNECTION=mysql
> vendor 10 DB_HOST=127.0.0.1
.editorconfig 11 DB_PORT=3306
.env 12 DB_DATABASE=1906984laravel
.env.example 13 DB_USERNAME=root
.gitattributes 14 DB_PASSWORD=
.gitignore
```

Configuración del archivo App/Provider/AppServiceProvider.php – Tamaño de los string

Agregamos a la clase AppServiceProvider

```
<?php
namespace App\Providers;
```

```
use Illuminate\Support\Facades\Schema;
use Illuminate\Support\ServiceProvider;
```

Configuramos el método para que reciba longitud de caracteres de 191

```
public function boot()
{
    Schema::defaultStringLength(191);
}
```

Visita la documentación oficial

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

Ejecutamos el comando en la Terminal actualiza los cambios realizados en las migraciones y se ven reflejados en PHPMysql en la BDD: `php artisan migrate`

Crea la migración de una tabla propia del proyecto

```
php artisan make:migration create_estudiante_table
```

Podemos crear la migración, el modelo, controlador y ruta con el siguiente comando

```
php artisan make:model site -mcr
```

Nota: Se debe colocar los nombres de los modelos en inglés, ya que le antepone a la migración la letra s

Deshace la última migración realizada

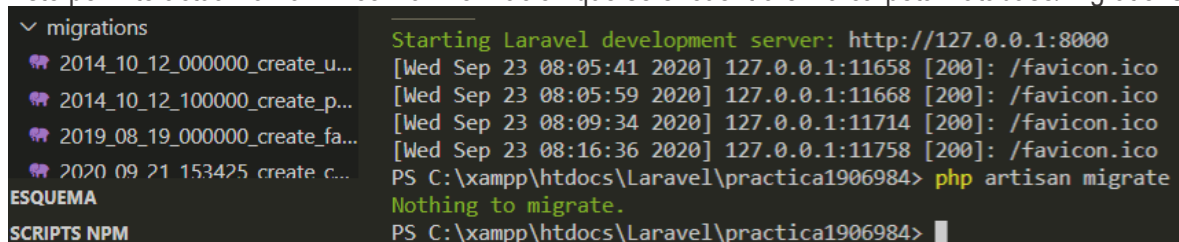
```
php artisan migrate:rollback
```

Actualiza los cambios realizados en una migración

```
php artisan migrate:refresh
```

```
php artisan migrate:fresh
```

Esto permite actualizar la BD con la información que se encuentra en la carpeta Database/Migrations



```

migrations
  2014_10_12_000000_create_u...
  2014_10_12_100000_create_p...
  2019_08_19_000000_create_fa...
  2020_09_21_153425_create c...
ESQUEMA
SCRIPTS NPM
Starting Laravel development server: http://127.0.0.1:8000
[Wed Sep 23 08:05:41 2020] 127.0.0.1:11658 [200]: /favicon.ico
[Wed Sep 23 08:05:59 2020] 127.0.0.1:11668 [200]: /favicon.ico
[Wed Sep 23 08:09:34 2020] 127.0.0.1:11714 [200]: /favicon.ico
[Wed Sep 23 08:16:36 2020] 127.0.0.1:11758 [200]: /favicon.ico
PS C:\xampp\htdocs\Laravel\practica1906984> php artisan migrate
Nothing to migrate.
PS C:\xampp\htdocs\Laravel\practica1906984>

```

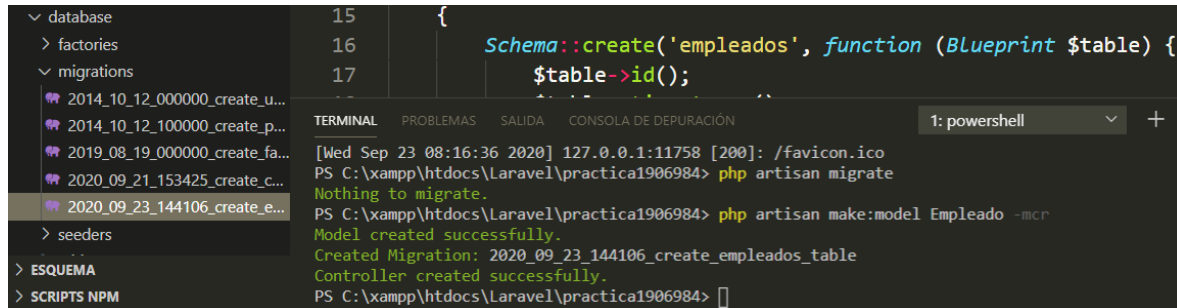
Si vamos hasta la BD creada observamos que se han creado nuevas tablas



Creación de un nuevo modelo

Creamos el modelo adecuado Empleado para realizar CRUD con -mcr le indico que cree además el **modelo, controlador y recursos** para el modelo **Empleado**

Utilizamos la instrucción `php artisan make:model Empleado -mcr`



Te crea el controlador en la ruta `App/Http/Controllers/EmpleadoController.php`

Archivo que contiene la clase

```
class EmpleadoController extends Controller
{
```

Los métodos index, créate, store, Show, Edit, Update, Destroy para interactuar con la base de datos

```
public function index()
```

También crea el modelo empleado en la ruta **Models**

El cual dispone de la siguiente clase y a Eloquent un (ORM)

```
<?php

namespace App\Models;

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

class Empleado extends Model
```

```
{
    use HasFactory;
}
```

También si vamos hasta la carpeta Database se ha creado una **migración** de empleado con el nombre empleadostable

```
<?php

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

class CreateEmpleadosTable extends Migration
{
```

Aquí es donde nos ubicamos para hacer nuestra tabla de la BD
Automaticamente contiene un id, y timestamps()

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

Creación del Modelo Empleado

```
public function up()
{
    Schema::create('empleados', function (Blueprint $table) {
        $table->integer('id')->autoIncrement();
        $table->string('nombre', 30);
        $table->string('email', 40);
        $table->date('fecha_alta');
        $table->string('foto');
        $table->timestamps();
    });
}
```

}

Para crear este modelo ejecutamos el comando `php artisan migrate`

```

> factories      16      Schema::create('empleados', function
migrations      17      $table->id();
2014_10_12_00000... 18      $table->string('nombre',30);
2014_10_12_10000... 19      $table->integer('precio');
2019_08_19_00000... 20      $table->date('fecha_alta');
2020_09_21_15342... 21
2020_09_23_14410... 22

> seeders
.gitignore
public
.htaccess
favicon.ico
index.php
ESQUEMA
SCRIPTS NPM

```

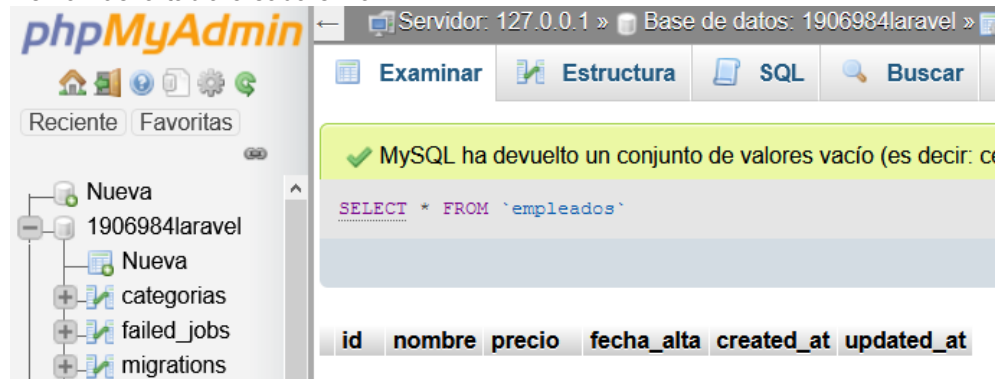
TERMINAL PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN

```

Illuminate\Database\Schema\Blueprint::__call("autoIncrement")
2 C:\xampp\htdocs\Laravel\practica1906984\vendor\laravel\framework\src\Illuminate\Database\Schema\Blueprint.php:100
CreateEmpleadosTable::{closure}(Object(Illuminate\Database\SqlConnection): Object(Illuminate\Database\Connection))
PS C:\xampp\htdocs\Laravel\practica1906984> php artisan migrate
Migrating: 2020_09_23_144106_create_empleados_table
Migrated: 2020_09_23_144106_create_empleados_table (399.71ms)
PS C:\xampp\htdocs\Laravel\practica1906984>

```

Ver la nueva tabla creada en la BD



En la documentación de Laravel encontramos otros tipos de datos admitidos por Laravel

<code>\$table->timestamp('added_on', 0);</code>	TIMESTAMP equivalent column with precision (total digits).
<code>\$table->timestampTz('added_on', 0);</code>	TIMESTAMP (with timezone) equivalent column with precision (total digits).
<code>\$table->timestamps(0);</code>	Adds nullable <code>created_at</code> and <code>updated_at</code> TIMESTAMP equivalent columns with precision (total digits).
<code>\$table->timestampsTz(0);</code>	Adds nullable <code>created_at</code> and <code>updated_at</code> TIMESTAMP (with timezone) equivalent columns with precision (total digits).
<code>\$table->tinyIncrements('id');</code>	Auto-incrementing UNSIGNED TINYINT (primary key) equivalent column.
<code>\$table->tinyInteger('votes');</code>	TINYINT equivalent column.
<code>\$table->unsignedBigInteger('votes');</code>	UNSIGNED BIGINT equivalent column.
<code>\$table->unsignedDecimal('amount', 8, 2);</code>	UNSIGNED DECIMAL equivalent column with a precision (total digits) and scale (decimal digits).
<code>\$table->unsignedInteger('votes');</code>	UNSIGNED INTEGER equivalent column.

Si necesitamos actualizar un campo de la tabla por alguna circunstancia se nos olvidó incluirlo o deseamos eliminar un campo lo hacemos con el comando
`php artisan migrate:fresh`

```

TERMINAL  PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  1

Migrated: 2014_10_12_100000_create_password_resets_table (27.00ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (33.71ms)
Migrating: 2020_09_21_153425_create_categorias_table
Migrated: 2020_09_21_153425_create_categorias_table (8.17ms)
Migrating: 2020_09_23_144106_create_empleados_table
Migrated: 2020_09_23_144106_create_empleados_table (10.03ms)
Migrating: 2020_09_25_145627_create_personas_table
Migrated: 2020_09_25_145627_create_personas_table (13.80ms)
PS C:\xampp\htdocs\Laravel\practica1906984> php artisan migrate:fresh

```

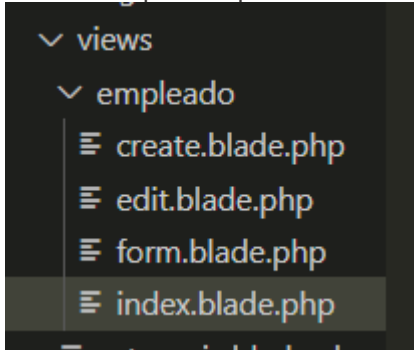
Nota: solo podemos ejecutar `migrate:rollback` para eliminar migraciones de la BD también podemos pasar el numero de la migración `step=1`

2. Practica. Creando rutas y controladores para el modelo

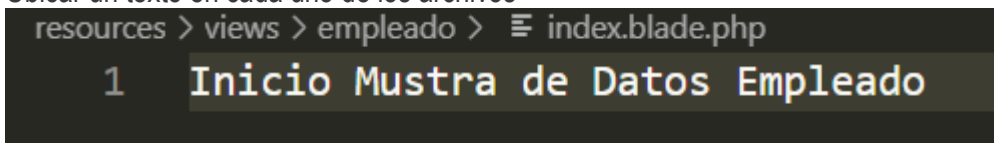
Comando que permite crear un controlador de tipo resource para un modelo existente

```
php artisan make:controller EmpleadoController --resource
```

Crear la carpeta empleado con el siguiente contenido



Ubicar un texto en cada uno de los archivos



Controlador y métodos para poder trabajar en empleado

Podemos crear nuevos métodos para los demás archivos de la carpeta empleado. Como tener acceso a estos métodos que están dentro del controlador ingresamos a la ruta **App/Http/Controllers/EmpleadosControllers.php**

Aquí encontramos todos los métodos definidos automáticamente (**index**, **create**, **store**, **show**, **edit**, **update**, **destroy**) vamos a acceder a ellos para que el controlador tome la vista y muestre su contenido.

En el método **index** (creamos la vista de los datos Aquí va la tabla mostrando los registros)

```
public function index()
{
    return view('empleado.index');
}
```

Crear una ruta en carpeta Routes/web.php utilizamos el controlador estándar con **resource**

```
Route::resource('/empleados', 'App\Http\Controllers\EmpleadoController');
```

Creamos un segundo controlador en el método **create** (Aquí va el formulario de registro)

```
public function create()
```



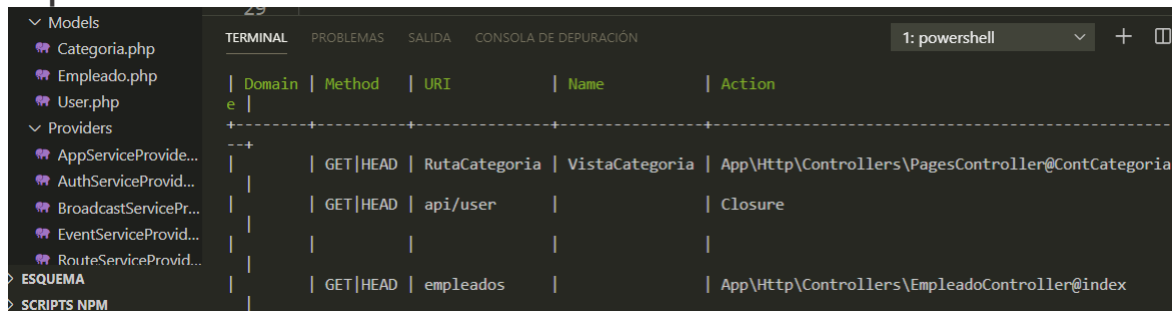
```
{
    return view('empleado.create');
}
```

Podemos hacer uso de la misma ruta con el método **resource** uniendo todas las rutas en una sola a diferencia de trabajar por separado los métodos GET, POST como lo veníamos trabajando.

```
Route::resource('/empleados', 'App\Http\Controllers\EmpleadoController');
```

Para mirar todas las rutas del proyecto de forma global

Php artisan route:list



Domain	Method	URI	Name	Action
	GET HEAD	RutaCategoria	VistaCategoria	App\Http\Controllers\PagesController@ContCategoria
	GET HEAD	api/user		Closure
	GET HEAD	empleados		App\Http\Controllers\EmpleadoController@index

3. Practica. Blade Templates (Vista View – Pagina Web con Bootstrap)

Utilizando el sistema de plantillas Blade. Sistema incorporado por Laravel para trabajar sus vistas (View). Para trabajar con Blade debemos ubicarlo al crear el archivo en el nombre ej. **empleado.blade.php**

Podemos verificar que continúa trabajando con el script PHP, pero ahora utilizaremos la sintaxis de Blade

```
<h3>Presentación de usuarios {{$numero}} </h3>
```

Ahora trabajamos con la plantilla de **Bootstrap** – Traemos el código de su plantilla y lo pegamos en nuestro archivo **plantilla.blade.php**

Starter template

Be sure to have your pages set up with the latest design and development standards. That means using an HTML5 doctype and including a viewport meta tag for proper responsive behaviors. Put it all together and your pages should look like this:

```
<!doctype html>
<html lang="en">
<head>
```

Copy to clipboard

Copy

Para no repetir el código presente en la plantilla de Bootstrap trabajamos de la siguiente manera: Creamos un archivo nuevo **plantillaweb.blade.php**; que hereda a las paginas **cliente.blade.php** y **producto.blade.php**

En la plantilla definimos una sección dinámica llamada **secciondinamica** dentro de **body**

Programa Análisis y Desarrollo de Sistemas de Información – ADSI

Página 8


```
<body>
  <div class="container">
    @yield('secciondinamica')
  </div>
```

La cual llamamos en los archivos

```
@extends('Plantillaweb')

@section('secciondinamica')
<h1>Entrada para Entrada de| Clientes</h1>
@endsection
```

4. Practica Insertar un nuevo registro en la BDD

Creamos un formulario en la ruta Views/Producto/**create.blade.php** Debemos tener en cuenta:

Utilizar el atributo **name** en cada uno de tus **input**, este debe ser el mismo campo de su base de datos. El botón debe ser tipo **submit** para que ejecute la acción. Por otro lado configuramos el método **POST** y la acción debe llamar a una nueva ruta.

Nota: Creamos una carpeta en **Views** para el modelo empleado con las vistas (Index.blade, create.blade, form.blade, edit.blade)

Protección **CSRF**: Laravel facilita la protección de su aplicación de los ataques de falsificación de solicitudes entre sitios (CSRF). Las falsificaciones de solicitudes entre sitios son un tipo de explotación maliciosa en la que se realizan comandos no autorizados en nombre de un usuario autenticado.

```
{{csrf_field()}}
```

Laravel genera automáticamente un "token" CSRF para cada sesión de usuario activa gestionada por la aplicación. Este token se usa para verificar que el usuario autenticado es el que realiza las solicitudes a la aplicación.

Por lo tanto simplemente tenemos que agregar **@csrf** directiva Blade para generar el campo de token. En action de formulario podemos colocar la ruta. Lo consultamos con el comando

Php artisan route (despliega el listado de rutas)

```
<form action="{{route('empleado.store')}}" method="POST">
```

Creación de Formulario para Productos

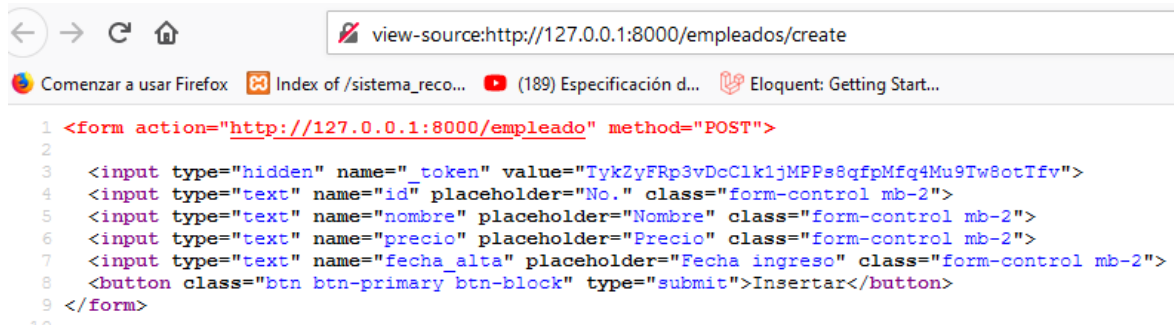
```
@extends('plantillaweb')
@section('secciondinamica')
```

```
<h3>Formulario para registro de productos</h3>

<form action="{{route('empleados.store')}}" method="POST">
    @csrf
<label for="">Código producto</label>
<input type="text" name="id" placeholder="No." class="form-control mb-2">
    <input type="text" name="nombre" placeholder="Nombre" class="form-control mb-2">
    <input type="text" name="precio" placeholder="Precio" class="form-control mb-2">
    <input type="date" name="fecha_alta" placeholder="Fecha ingreso" class="form-control mb-2">
    <input type="text" name="idCategoria" placeholder="Categoría No." class="form-control mb-2">
    <input type="text" name="existencia" placeholder="Existencia" class="form-control mb-2">
    <button class="btn btn-primary btn-block" type="submit">Insertar</button>
</form>
```

A manera de práctica observemos como queda el código fuente de la pagina

0



Ahora vamos a configurar el controlador en la carpeta App/Http/Controllers/EmpleadosController.php

Trabajamos ahora en el método store

<https://laravel.com/docs/8.x/requests#storing-uploaded-files>

Como práctica podemos realizar la siguiente instrucción

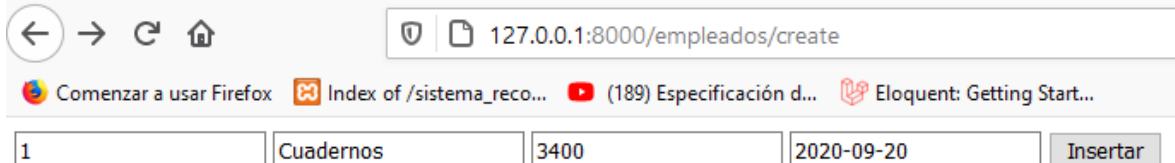
```
public function store(Request $request)
```

Programa Análisis y Desarrollo de Sistemas de Información – ADSI

Página 10

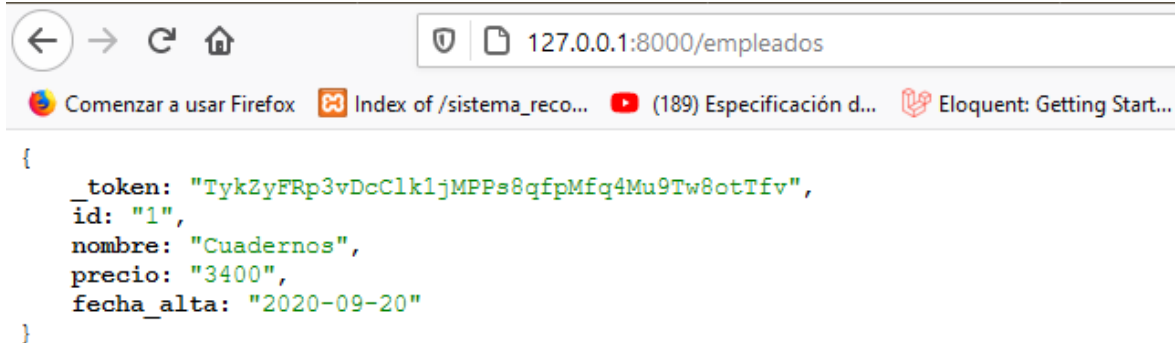
```
{
    $request = request()->all();
    return response()->json($request);
    //ahora retornamos un response para mirar lo que
    esta enviando
}
```

Vamos hasta formulario y hacemos envío de datos



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/empleados/create'. The browser's address bar also shows '127.0.0.1:8000/empleados/create'. Below the address bar, there are several tabs: 'Comenzar a usar Firefox', 'Index of /sistema_reco...', '(189) Especificación d...', and 'Eloquent: Getting Start...'. The main content area of the browser shows a form with four input fields: '1', 'Cuadernos', '3400', and '2020-09-20'. To the right of these fields is a button labeled 'Insertar'.

Verificamos el envío



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/empleados'. The browser's address bar also shows '127.0.0.1:8000/empleados'. Below the address bar, there are several tabs: 'Comenzar a usar Firefox', 'Index of /sistema_reco...', '(189) Especificación d...', and 'Eloquent: Getting Start...'. The main content area of the browser shows a JSON response:

```
{
  _token: "TykZyFRp3vDcClk1jMPPs8qfpMfq4Mu9Tw8otTfv",
  id: "1",
  nombre: "Cuadernos",
  precio: "3400",
  fecha_alta: "2020-09-20"
}
```

Insertar los datos a BD App/Http/Controllers/EmpleadosController.php

<https://laravel.com/docs/8.x/helpers#method-back>

Otro método para realizar **return back()**

<https://laravel.com/docs/8.x/eloquent#retrieving-single-models>

```
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\Flight;
use Illuminate\Http\Request;

class FlightController extends Controller
{
    /**
     * Create a new flight instance.
     *
     * @param Request $request
     * @return Response
     */
    public function store(Request $request)
    {
        // Validate the request...

        $flight = new Flight;

        $flight->name = $request->name;

        $flight->save();
    }
}
```

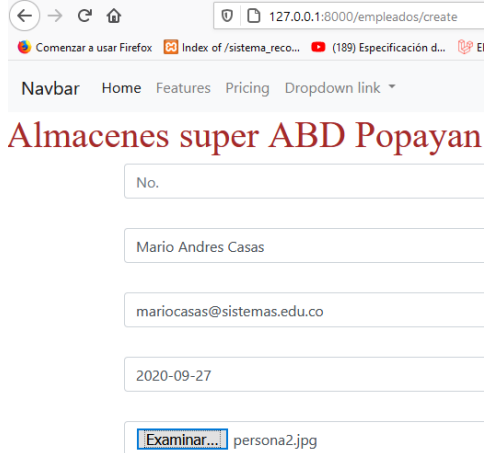
En el método **store** creamos la instancia (objeto) del modelo trabajado realizando las operaciones para almacenar los registros en la base de datos.

```
public function store(Request $request)
{
    $Empleado = new Empleado;

    $Empleado->nombre = $request->nombre;
    $Empleado->email = $request->email;
    $Empleado->fecha_alta = $request->fecha_alta;
    $Empleado->foto = $request->foto;

    $Empleado->save();
    return redirect()->route('empleados.create');
}
```

Ejecutamos el formulario identificando la captura de datos en MySQL



Verificamos la información guardada en la BDD

	id	nombre	email	fecha_alta	foto	created_at
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	1	Camila	camilita@gmail.com	2020-09-25	C:\xampp\tmp\phpC24E.tmp	2020-09-26 18:35:31
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	2	Julio Prado	julio@sena.edu.co	2020-09-21	C:\xampp\tmp\php21AC.tmp	2020-09-26 18:38:06
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	3	Juan Garcia	Juan@sena.edu.co	2020-09-21	C:\xampp\tmp\phpA924.tmp	2020-09-26 18:44:08
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	4	Carlos Julian Jimenez	carlosjulian@sistemas.edu.co	2020-09-27	C:\xampp\tmp\php3AFB.tmp	2020-09-28 01:18:41
<input type="checkbox"/> Editar <input type="checkbox"/> Copiar <input type="checkbox"/> Borrar	5	Mario Andres Casas	mariocasas@sistemas.edu.co	2020-09-27	C:\xampp\tmp\phpF3DE.tmp	2020-09-28 01:22:45

5. Practica Listar los datos desde la BDD

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

En el **Controlador ProductoController.php** paginamos la muestra de datos que acabamos de introducir; modificamos el método **index** indicando que la clase trabajada se llama el método **All** para mostrar todos los registros de la base de datos

```
$datos = Producto:: All();
```

```
public function index()
{
    $datos = Producto:: simplePaginate(1);
    return view('producto.index', compact('datos'));
}
```

Creamos una tabla con Bootstrap en Views/Producto/index.blade.php

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

```
<table class="table table-sm table-dark">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Mark</td>
```

Copy

Podemos verificar si se encuentra funcionando el ciclo si enviamos estos datos que representa la tabla

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    @foreach($datos as $dato)
      <tr>
        <th scope="row">1</th>
        <td>Mark</td>
        <td>Otto</td>
        <td>@mdo</td>
        <td>@mdo</td>
      </tr>
    @endforeach

  </tbody>
</table>
```

Creando los demás campos de la tabla

```
@foreach($datos as $dato)
    <tr>
    <th scope="row">{{$dato->id}}</th>
    <td>{{$dato->nombre}}</td>
    <td>{{$dato->email}}</td>
    <td>{{$dato->fecha_alta}}</td>
    <td>{{$dato->foto}}</td>

    </tr>
@endforeach
{{ $servicios->links() }}
```

6. Validaciones con Input

Validaciones Input

Laravel proporciona varios enfoques diferentes para validar los datos entrantes de su aplicación. En el controlador volveremos a utilizar la variable `$request` donde pasamos `validate`, el cual recibe el nombre de nuestro campo con su respectiva validación.

<https://laravel.com/docs/5.8/validation#introduction>

Agregamos validación en carpeta `Controllers/Pagescontroller.php` dentro de la función **crear** hacemos la misma validación para los demás campos

```
public function store(Request $request){
    $request->validate([
        'nombre' => 'required',
        'celular' => 'required',
    ]);
}
```

Dentro del formulario podemos validar para que el error tenga un mensaje dentro de una ventana

```
<form action="{{route('empleados.store')}}" method="POST">
    @csrf
    @error('nombre')
```



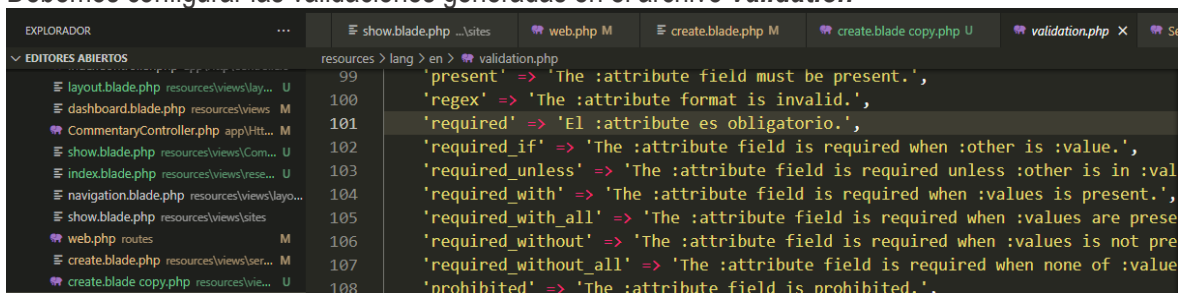
```
<div class="alert alert-danger alert-
dismissible fade show" role="alert">
    El nombre es requerido
    <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
        <span aria-hidden="true">&times;</span>
    </button>
</div>
@enderror
```

```
<small class="text-danger">{{$errors-
>first('nombre')}}</small>
```

Podemos dejar los valores ingresados a formulario si agregamos en los input
<https://laravel.com/docs/8.x/requests#old-input>

```
<input type="text" name="nombre" placeholder="Nombre" class
="form-control mb-2" value="{{(' $dato->nombre)}}">
```

Debemos configurar las validaciones generadas en el archivo **validation**



7. Editar registros de la base de datos

Agregamos un **botón**: Dentro de nuestra tabla (index.blade.php) para cada registro de la tabla producto, enviando al nombre de la ruta **usuarios.edit**, **\$usuario** con el parámetro **\$usuario**, así viajará el id de nuestra nota específica.

```
<td>
    <a href="{{ route('usuarios.edit', $usuario) }}">
<button type="button">
```

```
class="btn btn-warning btn-sm">Editar</button></a>
```

Configuramos el controlador de edición

Controlador editar: El controlador editar retornará la vista con un formulario para editar la nota correspondiente. Estamos haciendo la misma petición que en detalle, utilizando el \$id, para buscar la nota en nuestra base de datos.

```
public function edit(Producto $producto)
{
    return view('producto.edit', compact('producto'));
}
```

Formulario para editar registros Views/Productos/edit.blade.php

```
@extends('plantillaweb')

@section('secciondinamica')
<h1>Editar Producto {{$producto->id}} </h1>
<form action="{{route('productos.update', $producto)}}" method="POST" enctype="multipart/form-data">
    @method('PUT')
    @csrf

    <label>Código del producto</label>
    <input type="text" name="codigo" placeholder="Codigo"
class="form-control mb-2" value="{{$producto->codigo}}">

    <label>Nombre del producto</label>
    <input type="text" name="nombre" placeholder="Nombre"
class="form-control mb-2" value="{{$producto->nombre}}">

    <label>Descripción</label>
    <input type="text" name="descripcion" placeholder="Descripción" class="form-control mb-2" value="{{$producto->descripcion}}">
```

```
<button type="submit" class="btn btn-primary btn-
block">Insertar</button>
</form>

@endsection
```

```
public function update(Request $request, Producto $producto)
{
    $producto ->nombre = $request->nombre;
    $producto ->email = $request->email;
    $producto ->fecha_alta = $request->fecha_alta;
    $producto ->foto = $request->foto;
    $producto ->save();

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

8. Eliminar registros de la base de datos

Para eliminar un empleado de nuestra base de datos, solo debemos crear un formulario con nuestro botón de eliminar, configurar nuestra ruta y el controlador que ejecutará la acción en el archivo `index.blade.php`

```
<form action="{{ route('productos.destroy', $dato) }}" class
="d-inline" method="POST">
    @method('DELETE')
    @csrf
    <button type="submit" class="btn btn-danger btn-
sm">Eliminar</button>
</form>
```

Finalmente creamos el método Destroy en el controlador de Producto

```
public function destroy(Producto $producto)
{
    $Producto->delete();
    return redirect()->route('productos.index');
}
```

Bibliografía

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

<https://www.youtube.com/watch?v=KKpXpWCTlbo&list=PLPI81lqbj-4KHPEGngoy5PSjjxcwnpCdb>

<https://aprendible.com/series/laravel-desde-cero>

<https://imacreste.com/bootstrap-colores-fuentes-y-tipografias/#Small>

<https://bluuweb.github.io/tutorial-laravel/bases-datos/>

http://www.oscarabadfolgueira.com/crear-una-base-datos-mysql-desde-consola/#Acceder_a_mysql_desde_la_consola

BEATI Hernán (2015). PHP - Creación de páginas Web dinámicas 2a edición. Editorial Alfaomega. 2015

W3C The World Wide Web Consortium (2016). HTML 5.1 W3C Recommendation.

Recuperado de: <https://www.w3.org/TR/html>

W3C The World Wide Web Consortium (2017). CSS Snapshot 2017. Recuperado de:

<https://www.w3.org/TR/CSS>

<https://www.cursosdesarrolloweb.es/cursos-gratuitos/>

CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor(es)	Franco Reina	Instructor	CTPI	Junio 2021