	<p align="center"> SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD </p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

NOMBRE COMPLETO DEL APRENDIZ/GRUPO: NÚMERO DOCUMENTO:	Brayan Estiven Godoy Homez
	1028862384

CIUDAD Y FECHA:	Soacha	DÍA: 8	MES: septiembre	AÑO: 2024
PROGRAMA DE FORMACIÓN:	Análisis y Desarrollo de Software			
NÚMERO DE FICHA:	2672364			
NOMBRE INSTRUCTOR:				
FASE DEL PROYECTO:	Evaluación			
RESULTADO DE APRENDIZAJE	02-Verificar la calidad del software de acuerdo con las prácticas asociadas en los procesos de desarrollo.			
NOMBRE DE LA EVIDENCIA	220501098 – Controlar la calidad del servicio de software de acuerdo con los estándares técnicos. Uso de Frameworks - Modelos y base da datos			

TALLER PARA EVALUAR:


☐ DESEMPEÑO
 ☐ PRODUCTO
 ☐ CONOCIMIENTO

MODELOS Y BD RELACIONAL

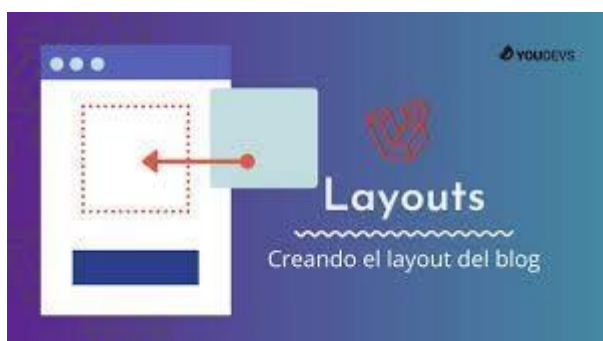
ENTREGABLE: Evidencia digital

FORMA: Individual en plataforma.

En esta guía haremos inserción y consulta de datos en tablas SQL y haremos vistas que heredan datos entre si

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

VISTAS LAYOUT EN LARAVEL



En Laravel se acostumbra a usar técnicas para procesar datos en documentos HTML a través de directivas Laravel que permiten hacer ciclos, condicionales dentro de documentos HTML para imprimir los datos de modo estratégico en cada archivo de vistas, es común que haya ciclos para recorrer arreglos de datos para imprimir en etiquetas HTML y también que haya vistas que prestan etiquetas a otras para evitar la repetición. Tal es el caso de los layouts, que son vistas tipo plantillas que permiten definir partes de vistas comunes a toda una aplicación como la barra de navegación, la cabecera y el pie de página.

Los layout de Laravel son archivos HTML comunes que pueden tener estilos CSS y programación Javascript, barra de navegación `<nav>`, cabeceras `<header>` y pies de página `<footer>`, también tienen zonas donde compartirán datos con otras vistas Blade. El layout se puede extender a otras vistas para evitar su repetición y llenar en esa segunda vista solo lo que deba cambiar en ella.

Es común que Laravel tenga un layout base en `resources/views/layouts/app.blade.php`

En el siguiente código HTML la directiva `@yield` de Laravel dice que el código dentro de ella se puede compartir a otras vistas, en este caso el título `<title>` y un contenido dentro del body.

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	<p>Versión:</p>
		<p>Fecha:</p>
		<p>Instrumento: No. 220501098-7</p>

Veamos este contenido de una vista layout `app.blade.php`

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@yield('title') - Pet Shop App</title>
  <link href="{{ asset('css/app.css') }}" rel="stylesheet">
</head>
<body>
  <!-- Cabecera -->
  <header>
    <h1>Mi Aplicación de Mascotas</h1>
    <nav>
      <ul>
        <li><a href="/">Inicio</a></li>
        <li><a href="/productos">Productos</a></li>
        <li><a href="/contacto">Contacto</a></li>
      </ul>
    </nav>
  </header>

  <!-- Contenido dinámico -->
  <div class="container">
    @yield('content')
  </div>

  <!-- Pie de página -->
  <footer>
    <p>&copy; 2024 Pet Shop. Todos los derechos reservados.</p>
  </footer>

  <!-- Scripts -->
  <script src="{{ asset('js/app.js') }}"></script>
</body>
</html>
```

	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7

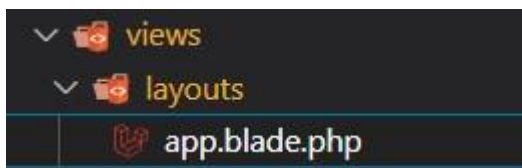
Luego, otras vistas que se extiendan de esta podrán usar todo el esqueleto del **layout** e inyectar algunas partes dinámicas o variables. Por ejemplo si esta vista se llama **index.blade.producto** y extiende contenidos del layout queda así. El título se personalizaría con la palabra Productos, se insertaría dentro del contenido unos títulos y listas HTML. Al final se cierra la sección con **@endsection** y todo lo demás del layout será compartido o heredado.

```
@extends('layouts.app')

@section('title', 'Productos')

@section('content')
    <h2>Lista de productos</h2>
    <ul>
        <li>Producto 1</li>
        <li>Producto 2</li>
        <!-- Aquí va la lista de productos -->
    </ul>
@endsection
```

Tenemos este **layout** en nuestra app llamado app.blade.php dentro de su carpeta layouts:



	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7


Veremos a continuación donde se usa la directiva Blade **@yield**: vemos que se usa en el título y dentro del body:

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

```

1  <!DOCTYPE html>
2  <html lang="es">
3  <!-- Archivo app.blade.php -->
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>@yield('title', 'Pet Shop App Service')</title>
8
9      <!-- Tu hoja de estilos personalizada -->
10     <link href="{{ asset('css/app.css') }}" rel="stylesheet">
11 </head>
12 <body>
13     <!-- Barra de navegación -->
14     <header>
15         <h3>CIDE Pet Shop Boys</h3>
16         <nav>
17             <ul>
18                 <li><a href="{{ url('/') }}">Inicio</a></li>
19                 <li><a href="{{ route('productos.create') }}">Nuevo Producto</a></li>
20                 <li><a href="{{ route('productos.index') }}">Ver Productos</a></li>
21             </ul>
22         </nav>
23     </header>
24
25     <!-- Sección donde las vistas hijas inyectarán contenido -->
26     <main>
27         @yield('content')
28     </main>
29
30     <!-- Pie de página -->
31     <footer>
32         <p>&copy; 2024 Servicio de Mascotas. Todos los derechos reservados.</p>
33     </footer>
34
35     <!-- Scripts; código de JavaScript -->
36     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.js"></script>
37 </body>
38 </html>

```

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

Ahora veamos cómo se extiende el layout **app** a la otra vista **inicio.blade.php**

```

1  @extends('layouts.app')
2
3  @section('title', 'Bienvenido a Pet Shop')
4
5  @section('content')
6  <section class="hero-large">
7      <div class="hero-content">
8          <h2>Tus mascotas son tus verdaderos amigos, cuídalos!</h2>
9          <p>Bienvenido a Pet Shop. Ofrecemos excelentes productos y servicios pa
10         <a href="{{ route('productos.index') }}" class="btn btn-primary">Ver Pr
11      </div>
12  </section>
13


```

Y se cierra después de terminar el código dentro del body:

```

29      <!-- Agregar mas productos segun sea necesario -->
30  </div>
31  </section>
32
33  @endsection
34

```

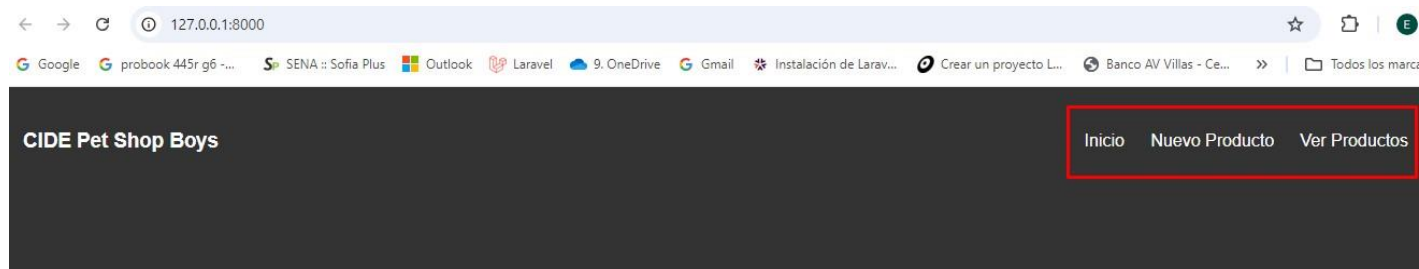
	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL	Versión:
		Fecha:

Instrumento: No. 220501098-7

ACTIVIDAD


La intención que tenemos es compartir el **layout** a las demás vistas que lo necesiten con su barra de navegación, cabecera, footer:

La vista inicio se cargará en la raíz del proyecto y compartirá contenidos de **layout** incluida su barra de navegación y **footer**:



LOS MODELOS

Los modelos en MVC hacen gestión con la base de datos y la lógica de negocio del sistema. Define cómo se hace la interacción con la BD.

	<p style="text-align: center;">SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

Ya hemos trabajado sobre las migraciones como un mecanismo para gestionar la estructura de la base de datos, en este caso con MySQL. Hemos creado varias tablas, ejemplo una tabla usuarios (users):

- En Laravel las tablas se sugieren en inglés, pero hacerlas en castellano también es posible para comenzar, ejemplo: customers (clientes), products (productos), countries (países), etc, sales(ventas).
- Las clases son en singular: Customer, Product, Country
- Es altamente recomendable recordar esta notación para evitar confusiones
- Al crear un proyecto nuevo en Laravel, se cuenta con unas migraciones predeterminadas que se recomienda encarecidamente no borrar, pues son útiles para la gestión de autenticación de usuarios y manejo de sesiones
- Observe la notación usada para tablas en inglés: para nombres en plural con una o más palabras con notación snake_case para las tablas usuarios, migraciones, trabajos fallidos, tokens de acceso personal: users, migrations, failed_jobs, personal_access_tokens
- Las tablas roles y customers que vemos, no vienen por defecto en Laravel, las hemos creado con migraciones como un ejercicio adicional, combinando incluso algunos datos en castellano para cercanía de nosotros.
- La tabla users si viene por defecto para hacer trabajo de autenticación, inicio de sesión, manejo de recordatorios, por tanto, es imperativo que no la elimine, más bien complementa en ella los cambios necesarios para guardar allí los usuarios de su sistema

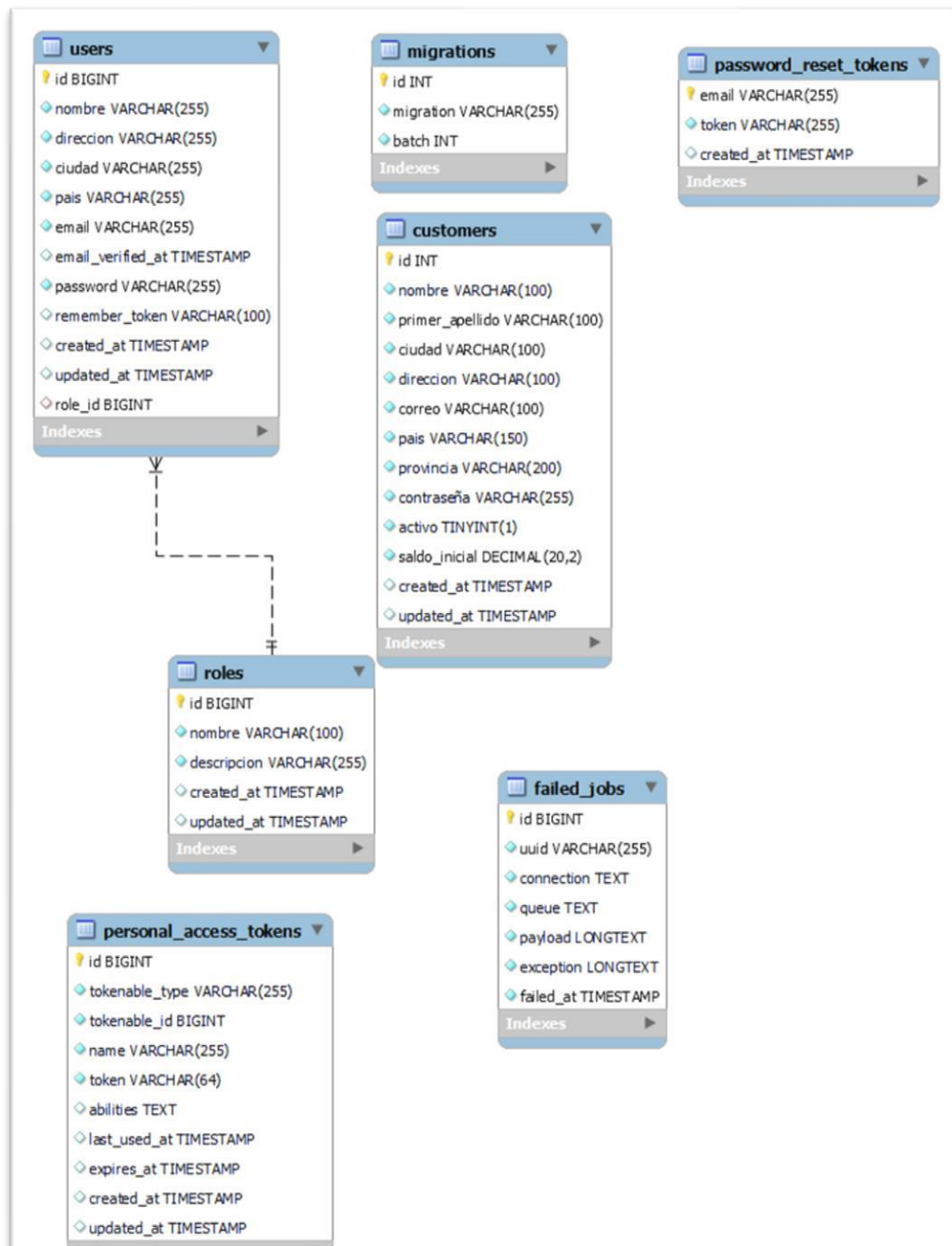


SERVICIO NACIONAL DE APRENDIZAJE SENA
REGIONAL CUNDINAMARCA
CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD

Versión:

Fecha:

Instrumento: No. 220501098-7




	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7

Figura 1. Diagrama relacional Laravel

MODELO Laravel

- Los modelos Laravel se usan para configurar las tablas de la BD del sistema. - Un modelo Laravel es una clase que se extiende de la clase **Model**, propia del
- Ejemplo de una tabla **usuarios** implementada con un modelo.
- Como puede verse, la tabla usuarios va a ser representada con una clase Usuario que implementará mediante métodos el acceso a sus datos.
- Dentro del modelo se podrá implementar más adelante si esta tabla se relaciona con otras
- Puede verse que la tabla agrega un atributo protegido **protected \$table = 'usuarios'** que recibe el nombre de la tabla a la cual apuntará
- También hay un atributo protegido **\$fillable** que guarda dentro de un arreglo [] la lista de columnas de tabla: **['nombre', 'email', 'password']**
- Con el presente modelo, se puede crear, borrar, actualizar y consultar usuarios
- Todas las columnas de tabla que se deseen gestionar, se deben relacionar en el arreglo **\$fillable = ['columna1', 'columna2', 'columna3', ..., columna_n]**
- El siguiente modelo código le dice a Laravel que hay 'apunte' a una tabla usuarios con estas columnas:

nombre	Email	password

	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Usuario extends Model
{
    // Definir la tabla asociada
    protected $table = 'usuarios';

    // Definir los campos que pueden asignarse masivamente
    protected $fillable = ['nombre', 'email', 'password'];

    // Definir relaciones, validaciones o cualquier otra lógica del negocio
}

```

Crear Modelo en Laravel

El modelo se debe crear usando con la Terminal de comandos Laravel, escribiendo en notación Camel Case.

Php make:model NombreModelo

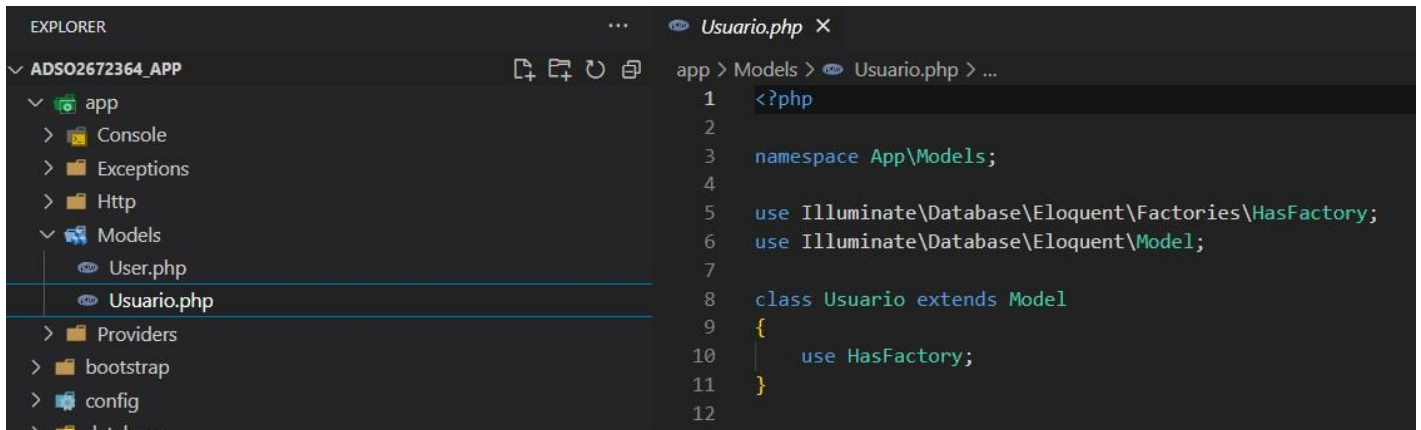
```
$ php artisan make:model Usuario
```

	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7

```
CIDE-1957@DESKTOP-90NVCGM MINGW64 /d/adso2672364_app
$ php artisan make:model Usuario

INFO Model [D:\adso2672364_app\app\Models\Usuario.php] created successfully.
```

Y se verá en la carpeta Models del sistema:



```

1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Usuario extends Model
9  {
10     use HasFactory;
11 }
12

```

Si la tabla se llama proveedores, el Modelo será Proveedor

```
CIDE-1957@DESKTOP-90NVCGM MINGW64 /d/adso2672364_app
$ php artisan make:model Proveedor

INFO Model [D:\adso2672364_app\app\Models\Proveedor.php] created successfully.
```

Si la tabla se llama **users** hay que ver que Laravel ya tiene una tabla **users** y un modelo **User** por defecto, lo sugerido es actualizar el Modelo **User** y la migración de usuarios de acuerdo a los campos que necesitemos manejar según el diseño relacional de la aplicación.

Ejercicio1

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	<p>Versión:</p>
		<p>Fecha:</p>
		<p>Instrumento: No. 220501098-7</p>

Crear migración y modelo para gestionar los productos de una entidad comercializadora de productos tecnológicos de hardware

Crear una tabla productos así:

id	nombre	descripcion	país_origen	presentacion	precio	stock

```
$ php artisan make:migration create_productos_table
```

```
CIDE-1957@DESKTOP-90NVCGM MINGW64 /d/adso2672364_app
$ php artisan make:migration create_productos_table

INFO Migration [D:\adso2672364_app\database\migrations\2024_09_09_011101_create_productos_table.php] created successfully.
```

Crearemos esta nueva tabla productos a través de una migración:

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	<p>Versión:</p>
		<p>Fecha:</p>
		<p>Instrumento: No. 220501098-7</p>

```

12     public function up(): void
13     {
14         Schema::create('productos', function (Blueprint $table) {
15             $table->id();
16             $table->string('nombre');
17             $table->string('descripcion');
18             $table->string('pais_origen');
19             $table->string('presentacion');
20             $table->decimal('precio', 20,2);
21             $table->unsignedInteger('stock');
22             $table->timestamps();
23         });
24     }

```

productos

- id BIGINT
- nombre VARCHAR(255)
- descripcion VARCHAR(255)
- pais_origen VARCHAR(255)
- presentacion VARCHAR(255)
- precio DECIMAL(20,2)
- stock INT
- created_at TIMESTAMP
- updated_at TIMESTAMP

Indexes

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
id	bigint unsigned	NO	PRI	NULL	auto_increment
nombre	varchar(255)	NO		NULL	
descripcion	varchar(255)	NO		NULL	
pais_origen	varchar(255)	NO		NULL	
presentacion	varchar(255)	NO		NULL	
precio	decimal(20,2)	NO		NULL	
stock	int unsigned	NO		NULL	
created_at	timestamp	YES		NULL	
updated_at	timestamp	YES		NULL	

Ahora creamos un Modelo ajustado a los campos nombre, descripción, pais_origen, precio y stock:

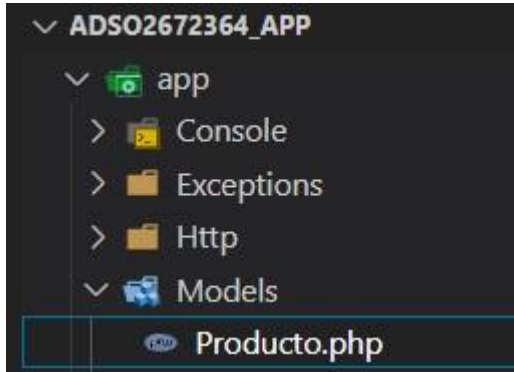
	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7


```

CIDE-1957@DESKTOP-90NVCGM MINGW64 /d/adso2672364_app
$ php artisan make:model Producto

INFO Model [D:\adso2672364_app\app\Models\Producto.php] created successfully.

```



	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	<p>Versión:</p>
		<p>Fecha:</p>
		<p>Instrumento: No. 220501098-7</p>

```

app > Models > Producto.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\EloquentModel;
7
8  class Producto extends Model
9  {
10     use HasFactory;
11 }
12


```

Modificamos la clase con los atributos \$table = 'productos' y \$fillable = ['nombre', 'descripcion', 'pais_origen', 'presentacion', 'precio', 'stock']

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

```

1  <?php
2  |
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Producto extends Model
9  {
10     use HasFactory;
11
12     // Indicar la tabla a la que apunta este modelo:
13     protected $table = 'productos';
14
15     // Describir los campos que deben ser tenidos en cuenta
16     // No es necesario incluir campos autoincrementables
17     protected $fillable = [
18         'nombre',
19         'descripcion',
20         'pais_origen',
21         'presentacion',
22         'precio',
23         'stock'
24     ];
25 }
```

	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7

Para verificar la funcionalidad del Modelo y la tabla productos intentamos hacer un trabajo de CRUD para insertar productos

- Crear una vista **create.blade.php** con un formulario que permita editar los datos del producto. Crear una carpeta producto para almacenar la nueva vista. create.blade.php es un nombre sugerido por Laravel para la vista de creación de productos
- Un controlador ProductoController que reciba los datos desde el navegador cliente
- Una ruta web **get** para mostrar el formulario
- Una ruta **post** para enviar los datos desde el formulario a ProductoController que permita guardar los datos en la tabla productos

Trabajo sobre la vista. Debemos usar el método POST, la directiva Laravel **@csrf** y casillas input tipo texto y de tipo número del producto:

La ruta que queremos definir para enviar los datos al Controlador la llamaremos **producto.store**.

Store es una palabra sugerida por Laravel para el proceso de almacenar los datos en el sistema. Emplearemos esa palabra también en el controlador

```

8      <h1>Formulario productos</h1>
9      <form action="{{ route('productos.store') }}" method="POST">
10         @csrf

```


	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	<p>Versión:</p>
		<p>Fecha:</p>
		<p>Instrumento: No. 220501098-7</p>

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Productos</title>
5      <link rel="stylesheet" href="{{ asset('css/style.css') }}">
6  </head>
7  <body>
8      <h1>Formulario productos</h1>
9      <form action="{{ route('productos.store') }}" method="POST">
10         @csrf
11         <label for="nombre">Nombre:</label>
12         <input type="text" id="nombre" name="nombre" required><br><br>
13
14         <label for="descripcion">Descripción:</label>
15         <input type="text" id="descripcion" name="descripcion" required><br><br>
16
17         <label for="pais">País de origen:</label>
18         <input type="text" id="pais" name="pais_origen" required><br><br>
19
20         <label for="presentacion">Presentación del producto:</label>
21         <input type="text" id="presentacion" name="presentacion" required><br><br>
22
23         <label for="precio">Precio:</label>
24         <input type="number" id="precio" name="precio" required><br><br>
25
26         <label for="stock">Stock:</label>

```

No olvidar los **name** de cada elemento del formulario, ellos se deben citar también en el controlador porque serán enviados allí.

	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7

```

23     <label for="precio">Precio:</label>
24     <input type="number" id="precio" name="precio" required><br><br>
25
26     <label for="stock">Stock:</label>
27     <input type="number" id="stock" name="stock" required><br><br>
28
29     <button type="submit">Enviar</button>
30 </form>
31 </body>
32 </html>

```

Controlador **ProductoController**

Debemos crear un controlador **ProductoController** para que haga la operación de validar y enviar los datos a la tabla productos o seleccionarlos de allí:

php artisan make:controller ProductoController.

```

CIDE-1957@DESKTOP-90NVCGM MINGW64 /d/adso2672364_app (main)
$ php artisan make:controller ProductoController

```

¿Qué queremos que haga el controlador?


- El controlador llamará a la vista que muestra el formulario para que reciba los datos.
- El controlador debe recibir los datos del formulario de producto a través de un objeto \$request

	<p style="text-align: center;">SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

- El objeto \$request traerá todos los datos que el usuario haya escrito y enviado en el formulario
- Hará verificación y validación de datos
- Una vez validados usará métodos para insertar datos a la tabla productos creando un nuevo objeto de Producto
- Si la validación fracasa no deberá permitir el ingreso de datos a la tabla
- Seguido de esto, se deberá retornar a la vista del formulario
- En un primer ejercicio solo nos ocuparemos de recibir los datos, validarlos, sin mostrar mensajes de error, luego enviarlos a la tabla.

¿Cómo validar datos?

El controlador implementará unas **reglas o filtros de validación** usando el método `validate()` de Laravel:

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7


```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Producto;
6  use Illuminate\Http\Request;
7
8  class ProductoController extends Controller
9  {
10     // Método para insertar datos en la tabla productos
11     public function store(Request $request)
12     {
13         // Crear reglas de validación de datos una vez se reciben del formulario
14         $validados = $request->validate([
15             'nombre'=>'required|string|min:3|max:30',
16             'descripcion'=>'required|string|min:5|max:255',
17             'pais_origen'=>'required|string|min:5|max:30',
18             'presentacion'=>'required|string|min:5|max:30',
19             'precio'=>'required|numeric',
20             'stock'=>'required|integer'
21         ]);

```

Las reglas de validación las hemos dispuesto de esta manera:

1. El nombre del producto debe ser obligatorio, es una cadena de texto, con una longitud mínima de 3 y máxima de 30, esto se resume en **'required|string|min:3|max:30'**
2. La descripción es también obligatoria, cadena de texto, con longitud entre 5 y 255 caracteres: **'required|string|min:5|max:255'**
3. El país y la presentación son también obligatorias, cadenas de texto entre 5 y 30: **'required|string|min:5|max:30'**
4. El precio es obligatorio y numérico: **'required|numeric'**
5. El stock es obligatorio y entero: **'required|integer'**

	SERVICIO NACIONAL DE APRENDIZAJE SENA	Versión:

REGIONAL CUNDINAMARCA

Fecha:


CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL

ACTIVIDAD

Instrumento: No. 220501098-7

6. En este código `$validado` es un arreglo asociativo que guarda los datos validados según las condiciones planteadas por el método `validate()`

```
// Crear reglas de validación de datos una vez se reciben del formulario
$validados = $request->validate([
    'nombre'=>'required|string|min:3|max:30',
    'descripcion'=>'required|string|min:5|max:255',
    'pais_origen'=>'required|string|min:5|max:30',
    'presentacion'=>'required|string|min:5|max:30',
    'precio'=>'required|numeric',
    'stock'=>'required|integer'
]);
```


	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

LISTAR PRODUCTOS

Un método recomendado para listar todos los recursos es el `index()` en Laravel.


Para el efecto se necesita configurar dentro del controlador un método que seleccione todos los registros de una tabla y los retorne como una colección de datos a una vista `index.blade.php` y una ruta que llame al método:

Método `index()` de controlador:

```

16     public function index()
17     {
18         // Seleccionar todos los productos de la BD
19         $productos = Producto::all();
20         return view('producto.index', ['productos'=> $productos]);
21     }

```

	SERVICIO NACIONAL DE APRENDIZAJE SENA	Versión:

Este método llama a la clase `Producto`, que en sí apunta a la tabla `productos` MySQL, de allí el método `all()` se encarga de consultar todos los datos de la tabla, emulando el comando SQL `SELECT * FROM productos`.

Luego los datos encontrados en la tabla `productos`, son enviados a la variable `$productos` que guarda la colección de ellos.

Una vez la colección está lista es retonada a la vista vista `index` y un arreglo asociativo con los productos `['productos'=>$productos]`

Por supuesto debemos crear la vista `index` y que ella reciba el arreglo de productos encontrados, los recorra y los imprima en pantalla:

REGIONAL CUNDINAMARCA
CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL
ACTIVIDAD

Fecha:

Instrumento: No. 220501098-7

VISTA INDEX

La vista `index.blade.php` debe recibir los datos enviados por el controlador, recorrer el arreglo `producto` por producto y mostrar en pantalla.

Para recorrer el arreglo usaremos la directiva de Laravel `@forelse` que ejecuta un ciclo `for` dentro de la vista HTML y para mostrar en pantalla usaremos etiquetas HTML `<table>` para crear una tabla, `<th>` para dibujar la cabecera de tabla, `<tr>` para mostrar las filas y `<td>` para mostrar los datos:



SERVICIO **NACIONAL DE APRENDIZAJE SENA**
REGIONAL CUNDINAMARCA
CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD


Versión:

Fecha:

Instrumento: No. 220501098-7



```
1
2 @extends('layouts.app')
3
4 @section('content')
5
6     <div class="container">
7         <h2>Productos</h2>
8         <table class="table">
9             <thead>
10                 <tr>
11                     <th>Id Producto</th>
12                     <th>Nombre</th>
13                     <th>Descripción</th>
14                     <th>País de origen</th>
15                     <th>Presentación</th>
16                     <th>Precio</th>
17                     <th>Stock</th>
18                 </tr>
19             </thead>
20             <tbody>
21                 @foreach($productos as $producto)
22                     <tr>
23                         <td>{{ $producto->id }}</td>
24                         <td>{{ $producto->nombre }}</td>
25                         <td>{{ $producto->descripcion }}</td>
26                         <td>{{ $producto->pais_origen }}</td>
```

	SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD	Versión:
		Fecha:
		Instrumento: No. 220501098-7

ACTIVIDAD

Instrumento: No. 220501098-7

```

26         <td>{{ $producto->pais_origen }}</td>
27         <td>{{ $producto->presentacion }}</td>
28         <td>{{ $producto->precio }}</td>
29         <td>{{ $producto->stock }}</td>
30         <td>
31             <a href="">Actualizar</a>
32         </td>
33         <td>
34             <a href="">Eliminar</a>
35         </td>
36     </tr>
37     @endforeach
38 </tbody>
39 </table>
40 </div>
41 @endsection

```

Comportamiento esperado de la aplicación:

- Se abre en la raíz del proyecto

- La raíz del proyecto debe abrir la vista inicio.blade.php
- La vista inicio.blade.php hereda la estructura de la vista layout app.blade.php
- Al heredar del layout la vista inicio debe mostrar la barra de navegación
- La barra de navegación muestra un menú Inicio, Nuevo Producto, Ver Productos
- Cada menú lo tendremos vinculado con rutas para agregar producto, ver los productos e ir a la vista de inicio.



- Al ir al menú Nuevo Producto veremos que se abre el formulario de crear producto:



Nuevo producto

Nombre:
PURINA PARA POLLOS DE

Descripción:
PURINA MARCA MEGAKIT

País de origen:
COLOMBIA

Presentación del producto:
UNIDAD X 50 KG

Precio:
145000

Stock:
1

Enviar

Al enviar el producto revisamos la BD:

Result Grid									
Filter Rows:									
Edit: Export/Import: Wrap Cell Content:									
	id	nombre	descripcion	pais_origen	presentacion	precio	stock	created_at	updated_at
▶	1	CROQUETAS HUMEDAS, ALIMENTO CANINI	ALIMENTOS CANINO	COLOMBIA	UNIDAD	235000.00	123	2024-09-09 21:23:11	2024-09-09 21:23:11
	2	REVITALIZANTE POST BAÑO	MULTIFUNCIONAL	COLOMBIA	UNIDAD	35000.00	2231	2024-09-09 22:16:10	2024-09-09 22:16:10
	3	CROQUETAS	CROQUETAS SABORIZADAS	COLOMBIA	UNIDAD EN BOLSA X 1 KM	85500.00	2	2024-09-15 14:55:25	2024-09-15 14:55:25
	4	COMIDA HUMEDA	COMIDA HUMEDA PARA GATOS	COLOMBIA	UNIDAD X 100 GR	2500.00	1	2024-09-15 16:37:45	2024-09-15 16:37:45
	5	PURINA PARA POLLOS DE LEVANTE	PURINA MARCA MEGAKITCHEN	COLOMBIA	UNIDAD X 50 KG	145000.00	1	2024-09-16 11:37:25	2024-09-16 11:37:25
*		NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

	<p align="center"> SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD </p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

Revisando el menú de producto esperamos que aparezcan todos los ítems:

← → ↺

127.0.0.1:8000/productos/index

☆

📁

🌐

Google

probook 445r g6 -...

SENA :: Sofia Plus

Outlook

Laravel

9. OneDrive

Gmail

Instalación de Larav...

Crear un proyecto L...

Banco AV Villas - Ce...

»

Todos los marca

CIDE Pet Shop Boys

[Inicio](#)
[Nuevo Producto](#)
[Ver Productos](#)

Productos


Id Producto	Nombre	Descripción	País de origen	Presentación	Precio	Stock	
1	CROQUETAS HUMEDAS, ALIMENTO CANINI	ALIMENTOS CANINO	COLOMBIA	UNIDAD	235000.00	123	Actualizar Eliminar
2	REVITALIZANTE POST BAÑO	MULTIFUNCIONAL	COLOMBIA	UNIDAD	35000.00	2231	Actualizar Eliminar
3	CROQUETAS	CROQUETAS SABORIZADAS	COLOMBIA	UNIDAD EN BOLSA X 1 KM	85500.00	2	Actualizar Eliminar
4	COMIDA HUMEDA	COMIDA HUMEDA PARA GATOS	COLOMBIA	UNIDAD X 100 GR	2500.00	1	Actualizar Eliminar
5	PURINA PARA POLLOS DE LEVANTE	PURINA MARCA MEGAKITCHEN	COLOMBIA	UNIDAD X 50 KG	145000.00	1	Actualizar Eliminar

© 2024 Servicio de Mascotas. Todos los derechos reservados.

Con lo cual logramos dos de las funciones CRUD: Crear Productos (Insertar) y leer productos (Select) y además creamos un layout que se hereda a la vista inicio.blade.php

EJERCICIOS

Conteste:

	<p style="text-align: center;">SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL</p>	Versión:
		Fecha:

1. El método de controlador store se recomienda para:
 - a. Seleccionar datos de la BD
 - b. Almacenar datos en la BD**
 - c. Mostrar un formulario

2. La vista create.blade.php se usa para:
 - a. Crear datos en la BD
 - b. Abrir la vista para insertar datos la BD**
 - c. Ninguna de las anteriores

3. El método index de ProductoController se usa para
 - a. Mostrar el formulario de inserción de productos
 - b. Consultar todos los productos y enviarlos al formulario**
 - c. Mostrar el formulario de actualización de productos

4. Php artisan make:controller InicioController
 - a. Crea una migración productod
 - b. Crea una tabla productos
 - c. Crea un controlador ProductoController**

5. Php artisan make:model Product
 - a. Crea una migración product
 - b. Crea una tabla productos
 - c. Crea un modelo Product**

6. Php artisan serve
 - a. Abre el proyecto Laravel actual**
 - b. Vuelve a crear la carpeta vendor
 - c. Crea de nuevo todas las migraciones

7. Php artisan make:migration create_cities_table
 - a. Abre el proyecto Laravel actual
 - b. Crea la migración para gestionar la tabla cities**

	<p style="text-align: center;">SERVICIO NACIONAL DE APRENDIZAJE SENA REGIONAL CUNDINAMARCA CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	Versión:
		Fecha:
		Instrumento: No. 220501098-7

c. Crea de nuevo todas las migraciones

8. `$request->validate(['id_cliente'=>'required|integer', 'name'=>'required'])` permite que:

a. En el método `store()` de un controlador se valide el `id_cliente` y nombre

b. En el método `create()` de un controlador se valide el `id_cliente` y nombre

c. En el método `index()` de un controlador se valide el `id_cliente` y nombre

9. `Return redirect()->back()` permite:

a. Retornar el resultado del método y redirigir el control desde el controlador a la vista anterior

b. Retornar el resultado del método y redirigir el control desde el layout la vista anterior

c. Retornar el resultado del método a la vista `create.blade.php`

10. Revise esta ruta:

```
Route::get('/productos/create', [ProductoController::class, 'create'])->name('productos.create');
```

a. Llama al método `'producto'` del controlador `ProductoController`

b. Llama al método `'class'` del controlador `ProductoController`

c. Llama al método `'create'` del controlador `ProductoController`

11. La ruta Laravel:

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	<p>Versión:</p>
		<p>Fecha:</p>
		<p>Instrumento: No. 220501098-7</p>

```
Route::get('/productos/create', [ProductoController::class, 'create'])->name('productos.create');
```

- Invoca al método create de ProductoController para insertar un producto
- Invoca al método create de ProductoController que llama la vista del formulario de productos
- Invoca al a la vista store.blade.php

12. Esta parte de código en la vista Laravel:

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

- Envía el formulario actual a la ruta 'productos.store' para ver los productos
- Envía el formulario actual a la ruta 'productos.store' para insertar los productos
- La línea @csrf permite a Laravel enviar un código de seguridad al servidor junto a los datos del formulario para insertar un producto

13. Escoja cuatro de sus tablas SQL de proyecto de software y codifique los controladores, vistas, rutas y modelos necesarios para las funcionalidades de insertar datos y seleccionar todos los datos.

14. Si una de sus tablas se llama proveedor, debe:

- Crear una carpeta 'proveedor' en views
- Crear una carpeta layouts en views

	<p>SERVICIO NACIONAL DE APRENDIZAJE SENA</p> <p>REGIONAL CUNDINAMARCA</p> <p>CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD</p>	<p>Versión:</p>
		<p>Fecha:</p>
		<p>Instrumento: No. 220501098-7</p>

- c. Dentro de la carpeta layouts debe crear una vista app.blade.php que cargue una barra de navegación,
- d. Su migración se debe llamar create_proveedores_table
- e. Su modelo debe llamarse Proveedor
- f. Su controlador debe ser ProveedorController
- g. La vista para insertar los datos debe llamarse create.blade.php dentro de una carpeta views/proveedor
- h. La vista para mostrar los datos de todos los proveedores se llamará index.blade.php
- i. Los métodos de controlador serán create(), store() e index()
- j. Las URI de las rutas deben usar palabras sustantivas, ejemplo '/proveedores', '/proveedores/create', '/proveedores/index'



SERVICIO **NACIONAL DE APRENDIZAJE SENA**
REGIONAL CUNDINAMARCA
CENTRO INDUSTRIAL Y DE DESARROLLO EMPRESARIAL ACTIVIDAD

Versión:

Fecha:

Instrumento: No. 220501098-7