Tabla de contenido

[Introducción 2](#_Toc1144)

[Datos de la base de datos 2](#_Toc6998)

[Carpeta Routes 3](#_Toc1001)

[Archivo web.php 3](#_Toc26711)

[Carpeta Controllers 4](#_Toc12168)

[Archivo MainController.php 4](#_Toc20403)

[Archivo ContactoController.php 5](#_Toc21606)

[Archivo ProductoController.php 6](#_Toc868)

[Archivo ReservaController.php 7](#_Toc14616)

[Carpeta Models 8](#_Toc8737)

[Carpeta Provider 8](#_Toc4620)

[Carpeta database 9](#_Toc9452)

[Archivo ProductoFactory.php 9](#_Toc22874)

[Archivo 2023\_01\_31\_112707\_create\_productos\_table.php 9](#_Toc30227)

[Archivo DatabaseSeeder.php 9](#_Toc11852)

[Carpeta views 10](#_Toc25895)

[Archivo carrito.blade.php 10](#_Toc16880)

[Archivo compraRealizada.blade.php 11](#_Toc1974)

# Introducción

Bienvenido al manual de programador para la creación de una página web de barbería utilizando Laravel. En este manual, se explicará detalladamente el código utilizado para la creación de una página web atractiva y funcional para una barbería. Laravel es un marco de desarrollo de aplicaciones web de código abierto que ofrece una solución eficiente y escalable para la creación de aplicaciones web.

Es importante destacar que en este manual no se incluirán variables propias que requiere Laravel, como la clase Auth, ni código que no aporta información relevante, como puede ser código HTML esquemático o CSS. Nuestro enfoque se centrará en las partes de código que son cruciales para el correcto funcionamiento de la página web de barbería.

# Datos de la base de datos

Es necesario crear manualmente la base de datos, que deberá llamarse "barberiaLaravel" y asignarle un nombre de usuario y contraseña también "barberiaLaravel". También es posible utilizar otros datos al modificar el archivo .env para garantizar la correcta conexión.

La página web ha sido diseñada para trabajar con los datos presentes en los archivos reservas.sql y productos.sql, los cuales contienen información real de productos y reservas.

Los datos en reservas.sql están limitados al mes de marzo para facilitar la visualización de citas disponibles y el comportamiento de la página web en dicha situación. No fue posible configurar el seeder para restringir los datos a un solo mes.

El archivo productos.sql contiene información real de productos y una imagen asociada a cada uno para proporcionar una sensación de página web real. Se puede utilizar el seeder para generar mas datos pero no aparecerán las imágenes.

Al introducir los datos en la base de datos, es importante asegurarse de que se están introduciendo en la tabla correcta, ya que de lo contrario podría generarse un error.

# Carpeta Routes

## Archivo web.php

Las rutas *get* son adecuadas para aquellas páginas web que no requieren recibir parámetros adicionales o que pueden obtener la información requerida a partir de las variables de sesión.

Route::get('/', [MainController::class, 'inicio'])->name('inicio');

Route::get('/empresa', [MainController::class, 'empresa'])->name('empresa');

Route::get('/servicios', [MainController::class, 'servicios'])->name('servicios');

Route::get('/carrito', [MainController::class, 'carrito'])->name('carrito');

Route::get('/compraRealizada', [MainController::class, 'compraRealizada'])->name('compraRealizada');

Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');

Las rutas *post* son adecuadas para aquellas funciones que requieren recibir parámetros adicionales para su ejecución, ya sea para realizar algún tipo de modificación en los datos o para guardarlos en el servidor.

Route::post('/actualizarCantidadCarrito', [MainController::class, 'actualizarCantidadCarrito'])->name('actualizarCantidadCarrito');

Route::post('/actualizarCarrito', [MainController::class, 'actualizarCarrito'])->name('actualizarCarrito');

Las rutas *resource* son necesarias para poder trabajar con las tablas de la base de datos.

Route::resource('productos', 'App\Http\Controllers\ProductoController');

Route::resource('reservas', 'App\Http\Controllers\ReservaController');

Route::resource('contactos', 'App\Http\Controllers\ContactoController');

# Carpeta Controllers

## Archivo MainController.php

Las funciones "inicio", "empresa", "servicios" y "carrito" redirigen a sus correspondientes vistas respectivamente. No incluyen ninguna lógica compleja, por lo que no es necesario detallarlas en este documento, excepto la función "carrito", que inicia la sesión para recopilar los productos que el usuario ha agregado en caso de que los haya. Se ha decidido realizar esta acción en este punto para mantener el código HTML lo más sencillo posible.

*public* function carrito() {

        session\_start();

*return* view('carrito');

    }

La función "actualizarCarrito" se encarga de almacenar un producto específico recibido como parámetro en la sesión del servidor PHP. Para llevar a cabo esta tarea, se inicia la sesión y, en caso de ser el primer producto, se crea una variable correspondiente. Se guardan los datos necesarios en un array. Posteriormente, se verifica si el producto ya se encuentra guardado en la sesión o si se trata de un nuevo producto agregado. En función de esto, se determina el identificador para sustituir la cantidad o para agregarlo como un producto completo. Finalmente, se redirige al usuario a la tienda.

*public* function actualizarCarrito(Request $producto) {

        session\_start();

        if (!isset($\_SESSION['productosComprados'])) {

            $\_SESSION['productosComprados'] = [];

        }

        $compra = [

            "id" => $producto->id,

            "nombre" => $producto->nombre,

            "precio" => $producto->precio,

            "cantidad" => $producto->cantidad,

            "stock" => $producto->stock,

            "imagen" => $producto->imagen,

        ];

        $estabaEnCarrito = false;

        $idEstabaEnCarrito = 0;

        foreach ($\_SESSION['productosComprados'] as $key => $producto) {

            if ($producto['id'] == $compra['id']) {

                $estabaEnCarrito = true;

                $idEstabaEnCarrito = $key;

        }}

        if ($estabaEnCarrito) {

            $\_SESSION['productosComprados'][$idEstabaEnCarrito]['cantidad'] = $compra['cantidad'];

        } else {

            $\_SESSION['productosComprados'][] = $compra;

        }

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

    }

La función "actualizarCantidadCarrito" tiene como objetivo modificar la cantidad de un artículo que desea comprar el usuario cuando este altera el valor en la página "Carrito". Para ello, se obtienen los datos de los productos de la sesión y se modifica el valor de la cantidad correspondiente.

*public* function actualizarCantidadCarrito(Request $request) {

        session\_start();

        foreach ($\_SESSION['productosComprados'] as $key => $producto) {

            if ($request->productos[$producto['id']] == 0) {

                unset($\_SESSION['productosComprados'][$key]);

            } else {

                $\_SESSION['productosComprados'][$key]['cantidad'] = $request->productos[$producto['id']];

            }

        }

*return* view('carrito');

    }

La función "compraRealizada" se encarga del proceso de compra de los productos del usuario. En primer lugar, se obtienen los productos de la sesión, se almacenan en una variable y se destruye la sesión para evitar que los productos aparezcan nuevamente en el carrito. Luego, se resta la cantidad correspondiente de cada producto comprado del stock disponible en la base de datos. Finalmente, se redirige a una vista que muestra un resumen de los productos comprados.

*public* function compraRealizada() {

        session\_start();

        $productosComprados = $\_SESSION['productosComprados'];

        session\_destroy();

        foreach ($productosComprados as $producto) {

            $stock = Producto::where('id', $producto['id']);

            $stock->decrement(

                'stock',

                $producto['cantidad']

            );

        }

*return* view('compraRealizada')->with(['productosComprados' => $productosComprados]);

    }

## Archivo ContactoController.php

La función "index" tiene como objetivo redirigir a los usuarios a la vista de contacto para que puedan completar el formulario de contacto. Sin embargo, para verificar si el servicio de la base de datos está disponible, se utiliza una estructura de control de excepciones "try-catch" y se intenta acceder a la tabla "Contacto". En caso de que el servicio no esté disponible, se redirige a la vista de error y se muestra un mensaje informando al usuario.

*public* function index() {

        try {

            $contacto = Contacto::all();

*return* view('contacto.contacto');

        } catch (\Throwable $e) {

*return* view('error')->with([

                "mensajeError1" => "Lo sentimos, en este momento no podemos atender su solicitud a través del formulario de nuestra página web",

                "mensajeError2" => "Por favor, vuelva a intentarlo más tarde o contáctenos a través de otras vías disponibles"

            ]);

    }}

La función "store" tiene como objetivo almacenar los datos del usuario y su mensaje en la base de datos, de manera que posteriormente, a través de otra aplicación diferente, un responsable pueda leer los mensajes de los clientes y ponerse en contacto con ellos si fuera necesario. Finalmente, se redirige a una vista que informa al usuario sobre el éxito en el envío del mensaje de contacto.

*public* function store(Request $request) {

        $reglas = [

            'nombre' => 'required',

            'email' => 'required',

            'asunto' => 'required',

            'mensaje' => 'required',

        ];

        $request->validate($reglas);

        Contacto::create([

            'nombre' => $request->nombre,

            'telefono' => $request->telefono,

            'email' => $request->email,

            'asunto' => $request->asunto,

            'mensaje' => $request->mensaje

        ]);

*return* view('contacto.enviado');

    }

## Archivo ProductoController.php

La función "index" se encarga de obtener todos los productos de la base de datos cuyo stock sea mayor a cero y crear una variable de sesión para los productos comprados si aún no existe. Finalmente, se redirige a una vista que muestra los productos en una tabla. En caso de que la base de datos no esté disponible, se redirige a una vista de error que informa al usuario de que la tienda no está disponible en ese momento.

*public* function index() {

        try {

            $productos = Producto::where('stock', '>', 0)->get();

            if (!isset($\_SESSION['productosComprados'])) {

                $\_SESSION['productosComprados'] = [];

            }

*return* view('productos.listar')->with(['productos' => $productos]);

        } catch (\Throwable $e) {

*return* view('error')->with(["mensajeError1" => "Lo sentimos, no se ha podido acceder a la base de datos"]);

        }

    }

La función "show" se encarga de redirigir a la página de detalle de producto cuando un usuario hace clic para obtener más información sobre un producto. La función transmite la información del objeto seleccionado para su visualización.

*public* function show(Producto $producto)

    {

*return* view('productos.mostrar-producto')->with(['producto' => $producto]);

    }

## Archivo ReservaController.php

La función "index" se encarga de obtener y gestionar todas las citas existentes en la base de datos para una determinada fecha. Esta fecha será el día siguiente al día actual y se utilizará para limitar la disponibilidad de citas en los tramos horarios que ya han sido ocupados por trabajadoras o que se encuentran en fines de semana. Además, se implementa un mecanismo de control de errores mediante un "try-catch", para redirigir al usuario a una página de error en caso de que la base de datos no esté disponible y informarle de este hecho.

*public* function index() {

        try {

            $tomorrow = Carbon::tomorrow()->toDateString('Y-m-d');

            $citas = Reserva::all(['fecha', 'hora', 'trabajadora']);

*return* view('reservas.reservas')->with(["tomorrow" => $tomorrow, 'citas' => $citas]);

        } catch (\Throwable $e) {

*return* view('error')->with(["mensajeError1" => "Lo sentimos, en este momento no aceptamos reservas de citas previas"]);

        }

    }

La función "store" se encargará de almacenar la información relacionada con la reserva de la cita en la base de datos. Primero, se realizará una validación de los datos introducidos, para garantizar que no haya campos vacíos. Una vez realizada la validación, se registrará la reserva en la base de datos, y finalmente se redirigirá al usuario a una vista que mostrará un resumen de la cita realizada.

*public* function store(Request $request) {

        $reglas = [

            'nombre' => 'required|max:50',

            'telefono' => 'required|max:9',

            'email' => 'required',

            'fecha' => 'required',

            'hora' => 'required',

            'servicio' => 'required',

            'trabajadora' => 'required',

        ];

        $request->validate($reglas);

        Reserva::create([

            'nombre' => $request->nombre,

            'telefono' => $request->telefono,

            'email' => $request->email,

            'fecha' => $request->fecha,

            'hora' => $request->hora,

            'servicio' => $request->servicio,

            'trabajadora' => $request->trabajadora,

        ]);

*return* view('reservas.guardado')->with([

            'nombre' => $request->nombre,

            'fecha' => $request->fecha,

            'hora' => $request->hora,

            'servicio' => $request->servicio,

            'trabajadora' => $request->trabajadora

        ]);

    }

# Carpeta Models

En la carpeta de Models se encuentran los modelos correspondientes a las tablas de la base de datos que serán utilizadas en la página web. Cada uno de ellos incluye la variable "$table" que verifica la tabla correspondiente al modelo, "$fillable" que es un array que permite indicar cuáles son las columnas de la tabla que pueden ser llenadas en masa (por medio de seeders), y "$hidden" que sirve para ocultar información en la tabla de datos, como contraseñas.

Se encuentran archivos llamados Contacto.php, Producto.php y Reserva.php en esta carpeta, los cuales son idénticos, salvo por la tabla que apuntan. Por lo tanto, en este documento, solo se presentará un ejemplo.

class Producto *extends* Model {

    use HasFactory;

*protected* $table = 'productos';

*protected* $fillable = [

        'nombre',

        'fabricante',

        'categoria',

        'precio',

        'stock',

        'descripcion',

        'imagen'

    ];

*protected* $hidden = [];

}

# Carpeta Provider

En el archivo AppServiceProvider.php se ha implementado una nueva directiva con el objetivo de darle formato a los precios de los productos de manera adecuada.

*public* function boot() {

        Blade::directive("priceformat", function ($valor) {

*return* "<?php echo number\_format(floatval($valor),2,',','.').' €'; ?>";

        });

    }

# Carpeta database

En la carpeta correspondiente, se almacenarán los archivos que se utilizarán para crear las diferentes tablas de la base de datos, para definir el formato de las columnas y para crear semillas de datos. Debido a que los archivos, con excepción de la tabla a la que hacen referencia, son similares entre sí, en este documento se presentará como ejemplo el archivo correspondiente a la tabla de productos.

## Archivo ProductoFactory.php

Nos indicará los formatos de los datos que serán creados de forma aleatoria mediante el seeder.

*public* function definition() {

*return* [

            'nombre' => $this->faker->name(),

            'fabricante' => $this->faker->company(),

            'categoria' => $this->faker->randomElement(["Barba", "Cabello", "Complementos", "Fijacion", "Afeitado"]),

            'precio' => $this->faker->randomFloat(2, 0.01, 100),

            'stock' => $this->faker->numberBetween(0, 100),

            'descripcion' => $this->faker->paragraph(1),

            'imagen' => $this->faker->imageUrl(100, 100),

        ];

    }

## Archivo 2023\_01\_31\_112707\_create\_productos\_table.php

Es el archivo encargado de crear la tabla en la base de datos. En él, se especifican los formatos de las columnas y cualquier característica adicional, como restricciones en la entrada de datos, tales como números positivos únicos, una lista fija de opciones, o la necesidad de un valor único.

*public* function up() {

        Schema::create('productos', function (Blueprint $table) {

            $table->id();

            $table->string('nombre');

            $table->string('fabricante');

            $table->enum('categoria', ["Barba", "Cabello", "Complementos", "Fijacion", "Afeitado"]);

            $table->float('precio', 7, 2)->unsigned();

            $table->integer('stock')->unsigned();

            $table->text('descripcion');

            $table->string('imagen');

            $table->timestamps();

        });

    }

## Archivo DatabaseSeeder.php

Archivo encargado de la creacion de datos falsos para las pruebas.

Producto::factory(100)->create();

        Reserva::factory(100)->create();

        User::factory()->create([

            'name' => 'user',

            'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', *// password*

            'email' => 'user@email.com',

        ]);

        User::factory(10)->create();

# Carpeta views

## Archivo carrito.blade.php

En esta vista, se presenta la página del carrito con los productos agregados por el usuario, si los hubiere. Se inicia recopilando los datos de los productos de la sesión, si los hubiere, y se calcula el precio total.

if (isset($\_SESSION['productosComprados'])) {

        $productosComprados = $\_SESSION['productosComprados'];

    } else {

        $\_SESSION['productosComprados'] = [];

        $productosComprados = $\_SESSION['productosComprados'];

    }

    $total = 0;

    foreach ($productosComprados as $producto) {

        $precioTotalProducto = $producto['precio'] \* $producto['cantidad'];

        $total += $precioTotalProducto;

    }

Luego, los productos se muestran en una tabla, con un elemento *select* que permite al usuario elegir la cantidad deseada. En el caso de que la cantidad cambie, el formulario se actualizará automáticamente para reflejar el cambio.

<select name="productos[{{ $producto['id'] }}]" id="productos[{{ $producto['id'] }}]"

    class="form-select" onchange="this.form.submit()">

    @for ($i = 0; $i <= intval($producto['stock']); $i++)

        <option {{ intval($producto['cantidad']) == $i ? 'selected' : '' }}>

            {{ $i }}

        </option>

    @endfor

</select>

Por último, se emplea un pequeño script de JavaScript para deshabilitar el botón de compra en el caso de que el usuario no haya agregado productos al carrito.

*const* tablaCarrito = document.getElementById("tablaCarrito");

*const* botonComprar = document.getElementById("botonComprar");

if (tablaCarrito.children[1].firstElementChild.id == 'carritoVacio') {

    botonActualizarCarrito.disabled = true;

    botonComprar.disabled = true;

}

## Archivo compraRealizada.blade.php

Esta vista muestra un resumen de los productos comprados por el usuario. No tiene logica de programacion mas alla de obtener el precio total de los productos por lo que carece de importancia mostrarlo en este documento.

## Archivo empresa.blade.php

Vista que muestra informacion de la empresa junto con el equipo de trabajadoras. Es una pagina HTML sin logica, por lo que no es necesaria mostrarla en este documento.

## Archivo error.blade.php

Vista utilizada para mostrarle al usuario que ha ocurrido un error. Recibe los mensajes mediante el with de la funcion y lo muestra por pantalla. No tiene logica de programacion.

## Archivo home.blade.php

Vista que se muestra al usuario cuando hace el login correctamente. No tiene logica de programacion.

## Archivo inicio.blade.php

Vista de la pagina principal de la barberia. Tiene un boton que redirige a la pagina de reservas y un formulario de login. Tambien permite registrarse a un usuario nuevo y la recuperacion de contraseña, pero al no tener un servicio de envio de email esta caracteristica no funciona.

Esta vista si tiene logica de programacion, y consiste en mostrar el formulario de ingreso de sesion si el usuario no esta logueado o, en el caso que el usuario este logueado, un boton para desloguearse.

@if (Auth::check())

*{{-- LogOut --}}*

@else

*{{-- LogIn --}}*

@endif

## Archivo servicios.blade.php

Es una vista con los distintos servicios que ofrece la barberia. No tiene logica de programacion por lo que no tiene interes en este documento.