Manual de programador

ÍNDICE

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 obtiene 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 mediante el método *decrement()*. 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 *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. Se utiliza en la vista *listar* y redirige a *mostrar-producto*.

```
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. La fecha mínima de reserva es día siguiente al día actual para evitar que reserven citas en el mismo día o en días anteriores. La variable *$citas* 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 mediante Javascript en la vista. 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 e 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 a la que apuntan. Por lo tanto, en este documento, solo se presentará un ejemplo de uno de ellos.

```php
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.

```php
public function boot() {
        Blade::directive("priceformat", function ($valor) {
            return "<?php echo number_format(floatval($valor),2,',','.').' €'; ?>";
        });
    }
```

Carpeta database

En la carpeta, 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*.

```php
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.

```php
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.

```php
        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 adquiridos por el usuario. La lógica de programación detrás de ella se limita a calcular el costo total de los productos, por lo que no se considera relevante incluirla en este manual de programación.

Archivo empresa.blade.php

Vista que muestra información de la empresa junto con el equipo de trabajadoras. Se trata de una página HTML que no posee lógica programada, por lo que no resulta relevante incluirla en la presente documentación.

Archivo error.blade.php

La vista en cuestión se utiliza para informar al usuario sobre la ocurrencia de un error. Los mensajes se transmiten a través del mecanismo *'with'* de la función y se visualizan en pantalla. No posee una lógica de programación significativa.

Archivo home.blade.php

Vista que se muestra al usuario cuando se ha realizado con éxito el proceso de inicio de sesión. No incluye una lógica de programación relevante.

Archivo inicio.blade.php

La vista corresponde a la **página principal** de la barbería, y cuenta con un botón que redirige a la página de reservas y un formulario de inicio de sesión. Además, permite la opción de registro para nuevos usuarios y la recuperación de contraseña, aunque debido a la falta de un servicio de correo electrónico, esta característica no se encuentra en funcionamiento.
Esta vista incluye lógica de programación, y se encarga de mostrar el formulario de inicio de sesión si el usuario no ha iniciado sesión previamente. En caso contrario, se despliega un botón para cerrar la sesión.

```
@if (Auth::check())
        {{-- LogOut --}}
@else
        {{-- LogIn --}}
@endif
```

Archivo servicios.blade.php

Esta vista presenta los diferentes servicios que la barbería ofrece. Dado que no incluye lógica de programación, no resulta relevante para los fines de la presente documentación

Carpeta auth

Esta carpeta alberga los diversos archivos necesarios para el proceso de inicio de sesión, registro, verificación y otros aspectos relevantes. Se han realizado ajustes en el estilo mediante el uso de Bootstrap, con el objetivo de asegurar una apariencia consistente con el resto de la página web. Además, se han incluido botones y se han modificado las rutas de redirección hacia otras secciones de la web, aunque sin incurrir en cambios significativos en la lógica de programación ni en su código.

Carpeta components

Archivo footer.blade.php

Este archivo almacena el pie de página que se incluirá al final de la estructura de diseño. Incluye el logo de la barbería, la información del derecho de autor y los distintos iconos con los enlaces a las redes sociales de la barbería.

Archivo header.blade.php

Este archivo almacena el encabezado de la página web. Incluye el logo de la empresa y el menú de navegación. Según si el usuario está autenticado o no, se mostrarán diferentes opciones. Cuando el usuario no está autenticado, se presentan las opciones: Inicio, Reservar cita, Empresa, Servicios y Contacto. Si el usuario está autenticado, se muestran las opciones: Tienda y Carrito. La identificación del usuario se puede verificar mediante el *helper* de Laravel "*Auth::check()*", y con una estructura condicional *IF* se puede controlar su visualización.

```
@if (Auth::check())
    <li class="nav-item">
        <a href="{{ route('productos.index') }}" class="nav-link menu-item {{ Route::is('productos.index') || Route::is('productos.show') ? 'active' : 'link-info' }}">
            Tienda
        </a>
    </li>
    <li class="nav-item">
        <a href="{{ route('carrito') }}" class="nav-link menu-item {{ Route::is('carrito') || Route::is('agregarProductoCarrito') ? 'active' : 'link-info' }}">
            Carrito
        </a>
    </li>
@endif
```

Carpeta contacto

Archivo contacto.blade.php

Esta vista presenta información de contacto de la barbería, incluyendo su número telefónico, dirección electrónica y física, así como un formulario de contacto. El formulario se dirige a la ruta *contactos.store* y la función almacenará los datos en la tabla *contactos* si los datos son correctos. En caso de que se produzca un error en la introducción de los campos por parte del usuario, se redirigirá a esta vista y se proporcionará la información relevante para que el usuario pueda realizar los cambios necesarios.

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li><strong>{{ $error }}</strong></li>
            @endforeach
        </ul>
    </div>
    <hr>
@endif
<form action="{{ route('contactos.store') }}" method="post">
    @csrf
    {{-- Formulario de contacto --}}
</form>
```

Archivo enviado.blade.php

El archivo cuenta con una estructura similar a la vista previa, pero se ha reemplazado el formulario de contacto con un mensaje que confirma el correcto envío del formulario y un botón que redirige al inicio.

Carpeta layouts

app.blade.php

Vista requerida para el proceso de inicio de sesión en Laravel. Se ha modificado para que sea visualizada a través del diseño principal de la página web. En lugar de realizar cambios en cada llamada a esta vista, se ha optado por reescribirla para mantener una mayor simplicidad. Una mejora futura sería realizar un seguimiento de los archivos que hacen uso de esta vista con el fin de modificarlos y utilizar la vista adecuada en su lugar.

```
@extends ('layouts.mainLayout')
@section('content-title', 'Barberia Laravel')
@section('content-area')
    @yield('content')
@endsection
```

mainLayout.blade.php

El archivo principal de la página web contiene la estructura completa del documento HTML. Incluye los enlaces y scripts necesarios para ***Bootstrap***, ***Jquery*** y ***DataTables***, así como también un enlace al archivo CSS con los estilos personalizados. La estructura se divide en tres partes: una llamada al archivo de **header** que incluye el logo y el menú de navegación, un **área de contenido** donde se visualizarán las diferentes vistas y una llamada al archivo de **footer** que incluye el logo, el derecho de autor y los enlaces a las redes sociales. La directiva *@include* se utiliza para incluir el header y el footer en la página, mientras que la directiva *@yield* permite que cada vista se integre entre ambos.

```
@include('components.header')
@yield('content-area')
@include('components.footer')
```

Este archivo contiene un script encargado de aplicar formato a la tabla de productos utilizando *DataTables*. Además, se ha definido un objeto para traducir la tabla al idioma español.

```
<script>
    $(document).ready(function() {
        $('#tablaProductos').DataTable({
            language: {
                search: "Buscar en la tabla:  ",
                lengthMenu: "Mostrar _MENU_ productos",
                info: "Mostrando productos _START_ a _END_ de un total de _TOTAL_",
                paginate: {
                    previous: "Anterior",
                    next: "Siguiente"
                }
            }
        });
    });
</script>
```

Carpeta productos

Archivo listar.blade.php

La vista **listar** muestra información sobre todos los productos en una tabla organizada. La tabla incluye la imagen, nombre, categoría, precio y un enlace para ver más detalles del producto.
Se utiliza *DataTables* para permitir la visualización de diferentes cantidades de líneas de productos en la tabla, la navegación a través de diferentes páginas y la posibilidad de filtrar productos mediante el campo de búsqueda en la tabla.
La función *index* del controlador *ProductoController.php* se utiliza para obtener los productos que tienen al menos un producto en stock y los parámetros de la vista se utilizan para mostrarlos en esta vista mediante un bucle *foreach*.

Archivo mostrar-producto.blade.php

Es la vista que se presenta cuando un usuario selecciona el enlace correspondiente en la tabla anterior. En esta vista, se muestra información detallada sobre el producto, incluyendo su descripción. Además, se proporciona un selector de cantidad de producto para que el usuario pueda elegir la cantidad deseada de productos a comprar. La cantidad seleccionada debe estar comprendida entre un mínimo de 1 y un máximo igual a la cantidad de producto en stock registrada en la tabla, con el fin de garantizar que el usuario no adquiera una cantidad mayor a la disponible.
Una vez seleccionada la cantidad de productos deseada, el usuario debe hacer clic en el botón **Añadir al carrito**, lo que activará la función *actualizarCarrito*. El formulario transmitirá la cantidad seleccionada, así como otros detalles del producto, incluyendo su identificación, nombre, precio, imagen y stock, como entradas ocultas del formulario.

```
<form action="{{ route('actualizarCarrito') }}" method="post">
    @csrf
    <div class="row featurette pb-4 pt-4">
        {{-- Código --}}
        <div class="col-md-7 order-md-2 pb-4">
            <div class="row row-cols-1 row-cols-sm-2 g-4">
                {{-- Código --}}
                <div class="col d-flex flex-column gap-2">
                    <h4 class="fw-semibold mb-0 h3">Stock</h4>
                    <select name="cantidad" id="cantidad" class="form-select">
                        @for ($i = 1; $i <= $producto->stock; $i++)
                            <option>{{ $i }}</option>
                        @endfor
                    </select>
                </div>
            </div>
        </div>
        {{-- Código --}}
    </div>
    <div class="pb-4 pt-4">
        <input type="hidden" name="id" value="{{ $producto->id }}">
        <input type="hidden" name="nombre" value="{{ $producto->nombre }}">
        <input type="hidden" name="precio" value="{{ $producto->precio }}">
        <input type="hidden" name="imagen" value="{{ $producto->imagen }}">
        <input type="hidden" name="stock" value="{{ $producto->stock }}">
        <input type="submit" class="btn btn-lg btn-info fw-bold border-white" value="Añadir al carrito">
    </div>
</form>
```

Carpeta reservas

Archivo guardado.blade.php

La vista de **resumen de reserva** se muestra después de una reserva exitosa para proporcionar al usuario una visión general de su reserva. Se utiliza un algoritmo en PHP para separar la fecha, almacenada como una cadena, en año, mes y día, lo que permite mostrar la fecha de una manera más atractiva, presentando, por ejemplo, el nombre del mes en lugar de solo el número.
También se muestra un mensaje que indica la barbera seleccionada por el usuario o un mensaje genérico si no se ha seleccionado una barbera en particular.

```
@php
    $meses = ['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio', 'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre'];
    $dia = intval(substr($fecha, 8));
    $mes = intval(substr($fecha, 5, 2));
    $year = substr($fecha, 0, 4);
@endphp
<h1 class="link-info h1">{{ $servicio }}</h1>
<hr>
<br>
<div class='container-fluid'>
    <h2>{{ $nombre }} has hecho una reserva para las {{ $hora }} h</h2>
    <h2>del {{ $dia }} de {{ $meses[$mes] }} de {{ $year }}</h2>
    @if ($trabajadora != 'Cualquiera')
        <h2>{{ $trabajadora }} te atendera encantada</h2>
    @else
        <h2>Nuestras barberas te atenderan encantadas</h2>
    @endif
    <br />
    <p class="lead">
        <a href="{{ route('inicio') }}" class="btn btn-lg btn-info fw-bold border-white">
            Volver
        </a>
    </p>
</div>
```

Archivo reservas.blade.php

En esta página, los usuarios pueden reservar una cita para cualquiera de los servicios ofrecidos por nuestra barbería, de lunes hasta viernes, en las franjas horarias establecidas y con cualquiera de nuestras barberas. Para efectuar la reserva, es necesario que los usuarios proporcionen obligatoriamente sus datos de contacto. A continuación, pueden elegir el día deseado para realizar la reserva siendo como mínimo el día siguiente al que acceden a la página y, a partir de una lista desplegable, elegir la franja horaria disponible. También deben elegir el servicio y, si tienen preferencia, la barbera. En caso de haber algún error al tratar de realizar la reserva, como campos vacíos, se informará al usuario mediante alertas de color rojo que indicarán los cambios necesarios para completar correctamente la reserva.

En la parte inferior de la vista se ha incorporado un script escrito en lenguaje JavaScript con el objetivo de controlar la correcta realización de la reserva. De esta forma, si el usuario selecciona un día fin de semana, un horario ya ocupado o una barbera que no esté disponible para el turno elegido, el botón de reserva se encontrará deshabilitado y no se permitirá la formalización de la misma.
He optado por utilizar JavaScript en lugar de PHP debido a que creo que la comprobación es mejor realizada en el lado cliente, evitando así llamadas continuas al servidor que puedan resultar incómodas para el usuario. Como mejora futura hay que refactorizar este código para mejorarlo de forma lógica y a la hora de leerlo.