

DESARROLLO DE APLICACIONES WEB CON LARAVEL 11

UNIDAD 4

1 LARAVEL 11

- Framework de PHP
- Código abierto
- Patrón de diseño MVC (Modelo-Vista-Controlador)
- Artisan: interfaz por comandos que facilitan el desarrollo.
- Motor de plantillas Blade
- Eloquent ORM
- Documentación: <https://laravel.com/docs/11.x>



Forge Vapor Ecosystem ▾ News Partners

DOCUMENTATION



The PHP Framework for Web Artisans

Laravel is a web application framework with expressive, elegant syntax. We've already laid the foundation — freeing you to create without sweating the small things.



GET STARTED

WATCH LARACASTS



2 ENTORNO DE TRABAJO

► SERVIDOR WEB

- XAMPP
- Servidor de desarrollo de Laravel: *php artisan serve*

► NAVEGADOR (**Cuidado con las mayúsculas y minúsculas!!**)

- <http://localhost/nombreProyecto/public/>
- <http://localhost:8000>

► COMPOSER

- Windows (hay que desactivar el debuggin en php.ini durante la instalación): <https://getcomposer.org/Composer-Setup.exe>
- LinuxMint: `sudo apt install composer`

► Extensiones VISUAL STUDIO CODE

- Laravel Snippets
- Laravel blade Snippets
- PHP intelephense
- PHP Debug (Ya configurado en apache, en VSC, reiniciar VSC después de tener instalada la extensión, ir a run and debug y seleccionar 'Listen for Xdebug'.)

► Configuraciones:

- php.ini: Permitir extensión=zip

► Linux:

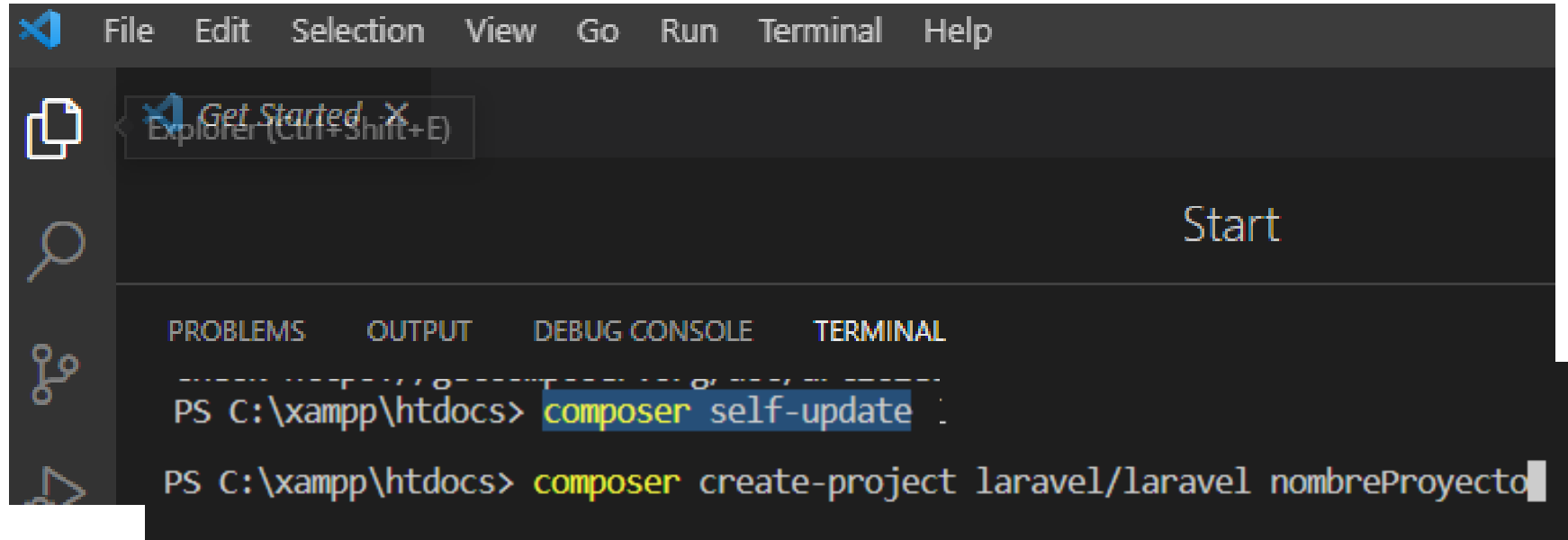
- Instalar extensiones: xml y sqlite3
 - `sudo apt install php-xml`
 - `sudo apt install php-sqlite3`
- Habilitar el módulo mod_rewrite para reescribir URL: `sudo a2enmod rewrite`
- Configurar apache para permitir reescribir rutas en var/www
 - `sudo xed`
 - Abrir `/etc/apache2/apache2.conf`

```
;extension=soap
;extension=sockets
;extension=sodium
;extension=sqlite3
;extension=tidy
;extension=xsl
extension=zip
```

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

- Reiniciar apache (**Escritorio/restart.sh**)

3 CREACIÓN DE PROYECTO



The screenshot shows the Visual Studio Code editor with a dark theme. The menu bar at the top includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar contains icons for Explorer, Search, Source Control, and Run and Debug. The main editor area displays a 'Start' button. Below the editor, the 'TERMINAL' tab is active, showing a command prompt window. The prompt is 'PS C:\xampp\htdocs>' and the command 'composer self-update' is entered and highlighted. Below that, the command 'composer create-project laravel/laravel nombreProyecto' is entered.

```
PS C:\xampp\htdocs> composer self-update :
PS C:\xampp\htdocs> composer create-project laravel/laravel nombreProyecto
```

- ▶ Si descargas el proyecto de un repositorio debes ejecutar el comando: **composer update** para que se descarguen las dependencias (crea la carpeta **vendor**).
- ▶ El fichero **.env** contiene parámetros de configuración y no se sube a los repositorios. Puedes generarlo copiando el fichero **.env.example**. Si necesitas generar una clave de aplicación ejecuta el comando: **php artisan key:generate**
- ▶ En Linux, debemos establecer los permisos correctos sobre la carpeta del proyecto si está ubicado en /var/www/html:
 - ▶ **sudo chown -R \$USER:www-data .**
 - ▶ **find . -type d -exec chmod 775 {} \;**
 - ▶ **find . -type f -exec chmod 664 {} \;**
 - ▶ **sudo chmod -R g+s .**
- ▶ Comando para limpiar cache cuando las cosas están bien y las rutas dan problemas: **php artisan route:clear**

4 ESTRUCTURA DE PROYECTO

Public

Esta es la carpeta más importante ya que es donde se ponen todos los archivos que el cliente va a mostrar al usuario cuando introduzcamos la URL de nuestro sitio web. Normalmente se carga el archivo `index.php` por defecto.

Routes

Otra de las carpetas que más vamos a usar a lo largo de este curso de Laravel. En ella se albergan todas las rutas (redirecciones web) de nuestro proyecto, pero más concretamente en el archivo `web.php`

```
1 | Dada una ruta → se cargará una vista
```

Resources

Esta es nuestra carpeta de recursos donde guardaremos los siguientes archivos, que también, están separados por sus carpetas... como cada nombre indica:

- `css` Archivos CSS
- `js` Archivos JS (JavaScript)
- `lang` Archivos relacionados con el idioma del sitio (variables & strings)
- `views` Archivos de nuestras vistas, lo que las rutas cargan

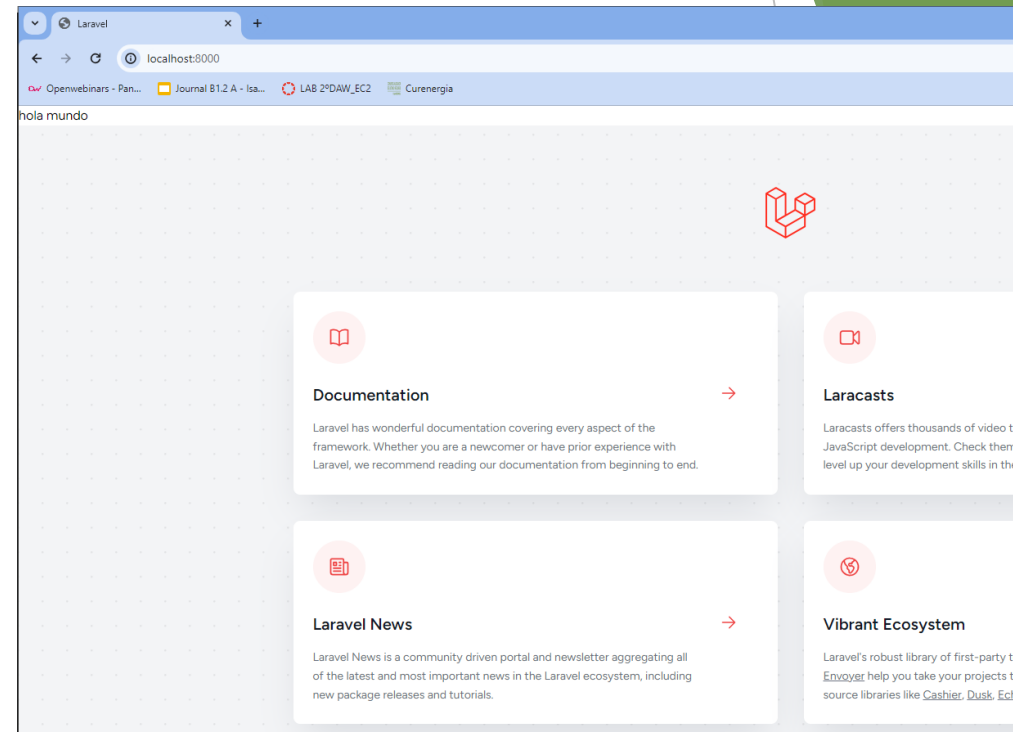
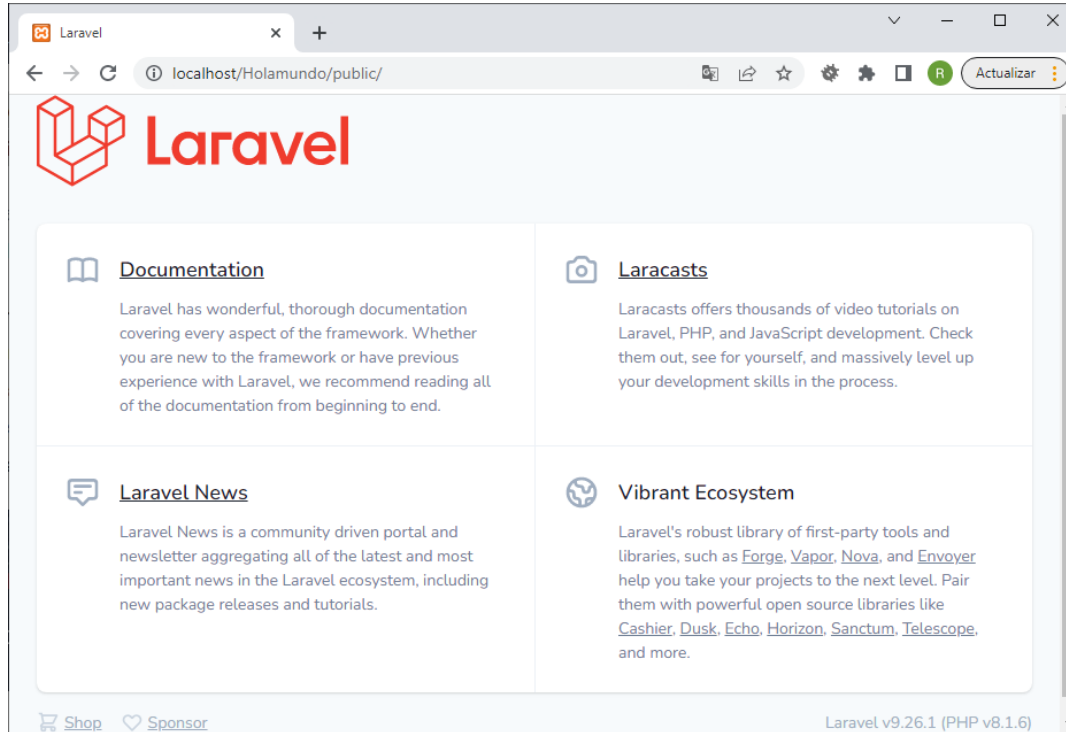
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

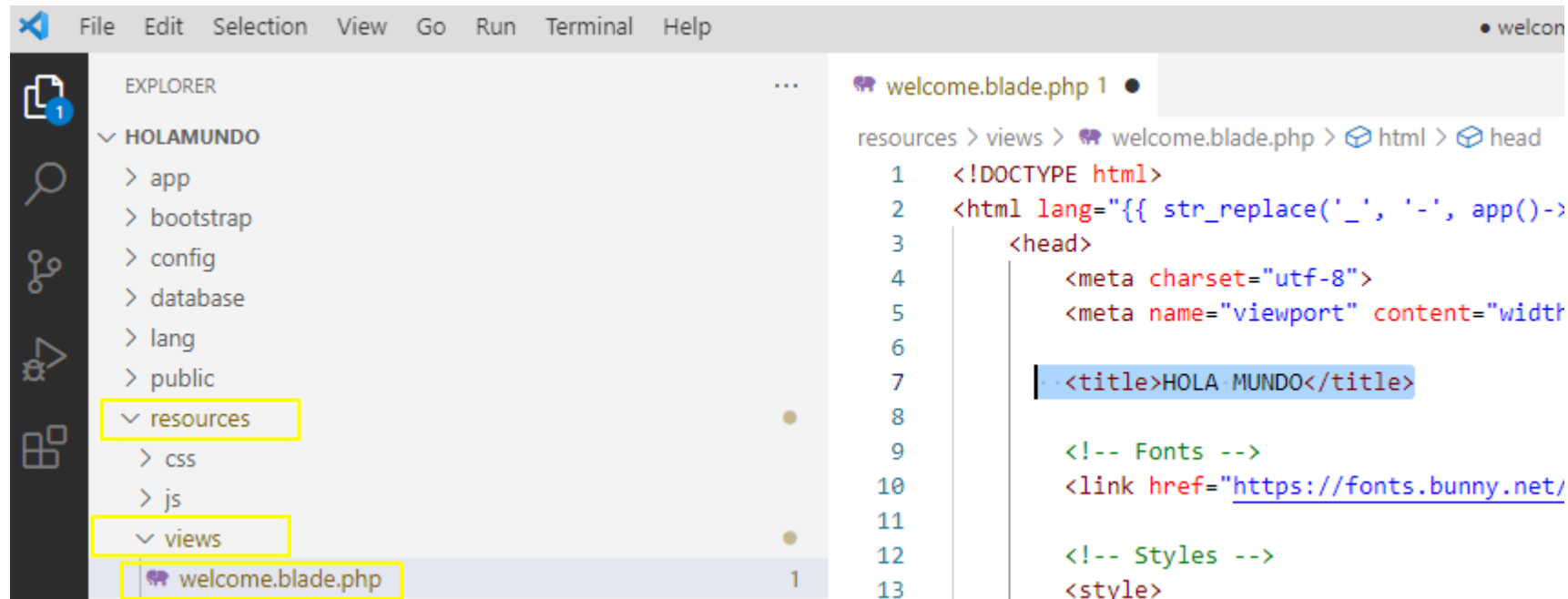
Prueba la nueva tecnología PowerShell multiplataforma <https://aka.ms/pscore6>

PS C:\xampp\htdocs\HolaMundo> █

5 ACCESO AL SITIO WEB



6 MODIFICACIÓN PÁGINA INICIO



File Edit Selection View Go Run Terminal Help

EXPLORER

- ✓ HOLAMUNDO
 - > app
 - > bootstrap
 - > config
 - > database
 - > lang
 - > public
 - ✓ resources
 - > css
 - > js
 - ✓ views
 - welcome.blade.php

resources > views > welcome.blade.php > html > head

```
1 <!DOCTYPE html>
2 <html lang="{{ str_replace('_', '-', app())->
3     <head>
4         <meta charset="utf-8">
5         <meta name="viewport" content="width=
6
7         <title>HOLA MUNDO</title>
8
9         <!-- Fonts -->
10        <link href="https://fonts.bunny.net/
11
12        <!-- Styles -->
13        <style>
```

7 RUTAS

- ▶ Evalúan la url y retornan la vista que se carga en el navegador. Único punto de entrada: [miSitio/public/index.php](#)
- ▶ Configuración de rutas: [route/web.php](#)
- ▶ Por defecto se carga la vista *welcome* ([resources/view/welcome.blade.php](#))
- ▶ Orden en las rutas: Se selecciona la primera regla que coincide.
 - ▶ Ruta para crear un coche ([coches/crear](#)) hay que ponerla antes de la ruta de mostrar los datos de un coche ya que interpreta crear como un parámetro, en este caso una matrícula)
- ▶ Tipos
 - ▶ [/](#): Punto de entrada al sitio web
 - ▶ [/coches](#): Ruta básica, para ver todos los coches
 - ▶ [/ coches/1111AAA](#) : Ruta con un parámetro, para ver el coche cuya matrícula es la especificada en la ruta.
 - ▶ [/ coches/1111AAA /1](#): Ruta con dos parámetros, para ver el propietario cuyo id se indica en la ruta y que corresponde a la matrícula que también se indica en la ruta.
 - ▶ [/ coches/1111AAA /1](#) : Ruta con parámetros, el último opcional. Si no se pasa el id del propietario, sería equivalente a la ruta [/ coches/1111AAA](#) .

7.1 DEFINICIÓN DE RUTAS

```
// PUNTO DE ENTRADA: http://localhost/tallerLaravel/public/
```

```
Route::get('/', function () {  
    //return view('welcome');  
    return "<h1>Página en construcción</h1>";  
});
```

```
// RUTA BÁSICA: http://localhost/tallerLaravel/public/coches
```

```
Route::get('/coches', function () {  
    return "<h1>Página para gestionar coches, se mostrarán todos los coches</h1>";  
});
```

```
//RUTA CON UN PARÁMETRO
```

```
Route::get('/coches/{matricula}', function ($matricula) {  
    return "<h1>Página para ver el coche con matrícula $matricula</h1>";  
});
```

```
//RUTA CON MÁS DE UN PARÁMETRO
```

```
Route::get('/coches/{matricula}/{propietario}', function ($matricula,$propietario) {  
    return "<h1>Página para ver el coche con matrícula $matricula del propietario $propietario</h1>";  
});
```

```
//RUTA CON PARÁMETROS OPCIONALES (AGRUPAR DOS ANTERIORES EN UNA)
```

```
Route::get('/coches/{matricula}/{propietario?}', function ($matricula,$propietario=null) {  
    if($propietario!=null){  
        return "<h1>Página para ver el coche con matrícula $matricula del propietario $propietario</h1>";  
    }  
    else{  
        return "<h1>Página para ver el coche con matrícula $matricula</h1>";  
    }  
});
```

```
// RUTA BÁSICA: http://localhost/tallerLaravel/public/coches/crear
```

```
Route::get('/coches/crear', function () {  
    return "<h1>Página para crear un coche</h1>";  
});
```

/

/coches

/ coches/1111AAA

/ coches/1111AAA /1

/ coches/1111AAA /1

7.2 MÉTODOS DE ENRUTAMIENTO

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

```
Route::match(['get', 'post'], '/', function () {
    //
});

Route::any('/', function () {
    //
});
```

las rutas GET, POST, PUT, y DELETE representan los métodos HTTP que se utilizan para interactuar con los recursos de una aplicación web. Cada uno tiene un propósito específico basado en el diseño de APIs:

- **GET**: Recuperación de recursos
- **POST**: Inserción de recursos
- **PUT**: Modificación de recursos
- **DELETE**: Borrado de recursos

7.3 ALIAS

- ▶ Permite asignar un nombre a una ruta.
- ▶ Este nombre se puede usar internamente en Laravel para hacer redirecciones a rutas en atributos *href* de enlaces o *action* de formularios.

```
// RUTA BÁSICA: http://localhost/tallerLaravel/public/coches
Route::get('/coches', function () {
    return "<h1>Página para gestionar coches, se mostrarán todos los coches</h1>";
})-> name("coches");
```

```
// RUTA BÁSICA: http://localhost/tallerLaravel/public/coches/crear
Route::get('/coches/crear', function () {
    return "<h1>Página para crear un coche</h1>";
})-> name("crearCoche");
```

```
//RUTA CON PARÁMETROS OPCIONALES (AGRUPAR DOS ANTERIORES EN UNA)
Route::get('/coches/{matricula}/{propietario?}', function ($matricula,$propietario=null) {
    if($propietario!=null){
        return "<h1>Página para ver el coche con matrícula $matricula del propietario $propietario</h1>";
    }
    else{
        return "<h1>Página para ver el coche con matrícula $matricula</h1>";
    }
})-> name("verCoche");
```

```
<nav>
    <a href={{ route('coches') }}>Inicio</a> |
    <a href={{ route('crearCoche') }}>Crear Coche</a>
    <hr/>
</nav>
```

```
<form action="{{route('borrarCoche',$coche->id)}}" method="post">
```

8 CONTROLADORES

- ▶ Definen la lógica de negocio de al app.
- ▶ Intermediario entre la vista y el servidor web.
- ▶ Cada ruta definida en **web.php** es gestionada por un controlador.
- ▶ Se ubican en **app/Http/Controllers**
- ▶ Creación: **php artisan make:controller cocheController**

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class cocheController extends Controller
{
    //
}
```

8.1 ASIGNACIÓN RUTA-CONTROLADOR

- ▶ Controlador de múltiples rutas
 - ▶ Método **inicio**: Para mostrar todos los vehículos
 - ▶ Método **crear**: Para crear un vehículo
 - ▶ Método **mostrar**: Para mostrar los datos de un vehículo
- ▶ Controlador de una única ruta:
 - ▶ Se usan cuando una acción es muy compleja.
 - ▶ Solamente tienen un método `__invoke`

```
namespace App\Http\Controllers;

class ProvisionServer extends Controller
{
    /**
     * Provision a new web server.
     */
    public function __invoke()
    {
        // ...
    }
}
```

```
use App\Http\Controllers\ProvisionServer;

Route::post('/server', ProvisionServer::class);
```

```
php artisan make:controller ProvisionServer --invokable
```

```
cocheController.php X
app > Http > Controllers > cocheController.php > cocheController > mostrar
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class cocheController extends Controller
8  {
9      function inicio(){
10         return "<h1>Página para gestionar coches, se mostrarán todos los coches</h1>";
11     }
12     function crear(){
13         return "<h1>Página para crear un coche</h1>";
14     }
15     function mostrar($matricula,$propietario=null){
16         if($propietario!=null){
17             return "<h1>Página para ver el coche con matrícula $matricula del propietario $propietario</h1>";
18         }
19         else{
20             return "<h1>Página para ver el coche con matrícula $matricula</h1>";
21         }
22     }
23 }
```

```
web.php X
routes > web.php > ...
14 |
15 */
16 Route::get('/', function () {
17     //return view('welcome');
18     return "<h1>Página en construcción</h1>";
19 });
20 Route::get('coches', [cocheController::class,"inicio"]-> name("coches");
21 Route::get('coches', [cocheController::class,"crear"]->name("crearCoche");
22 Route::get('coches/{matricula}/{propietario?}', [cocheController::class,"mostrar"]-> name("verCoche");
--
```

8.2 GRUPOS DE RUTAS

- ▶ Agrupar rutas del mismo controlador
- ▶ Se define un grupo de rutas para el controlador
- ▶ En cada ruta se indican dos parámetros
 - ▶ Ruta
 - ▶ Método del controlador que gestiona la ruta

```
web.php x cocheController.php index.blade.php
routes > web.php > ...
8 | Web Routes
9 | -----
10 |
11 | Here is where you can register web routes for your application. These
12 | routes are loaded by the RouteServiceProvider within a group which
13 | contains the "web" middleware group. Now create something great!
14 |
15 | */
16 Route::get('/', function () {
17     // La vista por defecto del sitio será mostrar vehiculos
18     return view('coches/index');
19 });
20 Route::controller(cocheController::class)->group(
21     function(){
22         Route::get('coches', "inicio") -> name("coches");
23         Route::get('coches/crear', "crear")-> name("crearCoche");
24         Route::get('coches/{matricula}/{propietario}', "mostrar")-> name("verCoche");
25     }
26 );
```

```
cocheController.php x
app > Http > Controllers > cocheController.php > cocheController > mostrar
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class cocheController extends Controller
8 {
9     function inicio(){
10         return "<h1>Página para gestionar coches, se mostrarán todos los coches</h1>";
11     }
12     function crear(){
13         return "<h1>Página para crear un coche</h1>";
14     }
15     function mostrar($matricula,$propietario=null){
16         if($propietario!=null){
17             return "<h1>Página para ver el coche con matrícula $matricula del propietario $propietario</h1>";
18         }
19         else{
20             return "<h1>Página para ver el coche con matrícula $matricula</h1>";
21         }
22     }
23 }
```

9 VISTAS

- ▶ Se ubican **resources/views**
- ▶ Tienen extensión **.blade.php**
- ▶ Se cargan desde el Controlador:
 - ▶ Sin parámetros:
 - ▶ `return view("coches/index")`
 - ▶ Con parámetros
 - ▶ `return view("coches/propietario", ["matricula"=>$matricula,"propietario"=>$propietario])`
 - ▶ `return view("coches/propietario", compact("matricula","propietario"))`
- ▶ Visualización de parámetros **{{ \$matricula }}**: Para visualizar resultados php en una vista o acceder a parámetros.

```
verPropietario.blade.php X
resources > views > verPropietario.blade.php > ...
1 <!DOCTYPE html>
2 <html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
3     <head>
4         <meta charset="utf-8">
5         <meta name="viewport" content="width=device-width, initial-scale=1">
6
7         <title>Laravel</title>
8         <!-- CDN Bootstrap -->
9         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet">
10        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js" rel="script">
11    </head>
12    <body class="antialiased">
13        <h1 class="text-primary">Fecha:{{ date('d/m/Y') }}</h1>
14        <h1 class="text-primary">Matricula:{{ $matricula }}</h1>
15        <h1 class="text-primary">Proietario:{{ $propietario }}</h1>
16    </body>
17 </html>
```

9.1 BOOTSTRAP

- ▶ Via CDN
- ▶ Instalar en el proyecto
 - ▶ Instalar Node.js
 - ▶ Instalar Bootstrap
 - ▶ `composer require laravel/ui`
 - ▶ `php artisan ui bootstrap`
 - ▶ Instalar mix
 - ▶ `npm i laravel-mix`
 - ▶ Copiar ficheros `package.json` y `webpack.mix.js`
 - ▶ Compilar Bootstrap
 - ▶ `npm install`
 - ▶ `npm run dev` (servidor de desarrollo)
 - ▶ `npm run production` (servidor de producción)

```
<!doctype html>
<html>
<head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-kenU1KFdBle4zVF0s0G1M5b4hcpyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0l4"
crossorigin="anonymous"></script>

</head>
<body>
    <p class="text-white bg-dark">This is example from LaravelTuts.com</p>
</body>
</html>
```

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

    <!-- Styles -->
    <link href="{{ asset('css/app.css') }}" rel="stylesheet">
</head>
<body>
    <p class="text-white bg-dark">This is example from LaravelTuts.com</p>
</body>
</html>
```


10 PLANTILLAS (I)

- ▶ Contienen la parte común a todas las páginas del sitio
- ▶ Se crean en **resources/views**
- ▶ Extensión **.blade.php** para reconocer las directivas de blade
- ▶ Motor de plantilla blade define unas **directivas**, pequeñas funciones ya escritas que aceptan parámetros y que cada una de ellas hace una función diferente dentro de Laravel: @yield, @extends, @section, ...
 - ▶ **@yield('nombre')**: Partes variables de la plantilla
 - ▶ **@extends('layout.plantilla')**: Importar plantilla en vista
 - ▶ **@section('nombre','valor')** : Modificar variable de plantilla en vista.
 - ▶ **@section('nombre') datos @ensection**: Modificar variable de plantilla en vista.

10 PLANTILLAS (II)

plantilla.blade.php X

resources > views > plantilla.blade.php > ...

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Taller</title>
6   </head>
7   <body>
8     <header>
9       <!-- TÍTULO DE LA PÁGINA - DINÁMICO -->
10      <h1>@yield('apartado')</h1>
11      <hr/>
12    </header>
13    <!-- MENÚ -->
14    <nav>
15      <a href={{ route('coches') }}>Inicio</a> |
16      <a href={{ route('crearCoche') }}>Crear Coche</a>
17    </nav>
18    <!-- CONTENIDO DINÁMICO EN FUNCIÓN DE LA SECCIÓN QUE SE VISITA -->
19    @yield('contenido')
20  </body>
21  <footer>
22    <!-- MENSAJE -->
23    @yield('mensaje')
24  </footer>
25 </body>
26 </html>
```

index.blade.php X

propietario.blade.php

v

resources > views > coches > index.blade.php > ...

```
1 @extends('plantilla')
2 @section('apartado',"MOSTRAR VEHÍCULOS")
3 @section('contenido')
4   <table border="1">
5     <tr>
6       <th>Código</th>
7       <th>Propietario</th>
8       <th>Matrícula</th>
9       <th>Color</th>
10      <th>Teléfono</th>
11      <th>Email</th>
12    </tr>
13  </table>
14 @endsection
15 @section('mensaje')
```

propietario.blade.php X

resources > views > coches > propietario.blade.php

```
1 @extends('plantilla')
2 @section('apartado',"PROPIETARIO: $propietario Vehículo:$matricula")
3 @section('contenido')
4 @endsection
5 @section('mensaje')
```

11 REDIRECCIONES

Redirección	Descripción
<code>return 'texto'</code>	Devuelve un texto
<code>return view('vista')</code>	Devuelve una vista
<code>return view('vista',compact('dato1','dato2'))</code>	Devuelve una vista con dos parámetros. Es posible usar @isset(\$dato1) en blade para comprobar si el dato existe.
<code>return redirect()->route('ruta',param)</code>	Devuelve una ruta definida con un parámetro.
<code>return redirect()->route('ruta',[param1,param2])</code>	Devuelve una ruta definida con dos parámetros.
<code>return redirect()->route('ruta')->with('nombreVar',\$valor)</code>	Devuelve una ruta y añade a la sesión una variable. Para recuperar la variable en la vista : @if(session(nombreVar')) <h5>session(nombreVar')</h5> @endif
<code>return back()->with('nombreVar',\$valor)</code>	Vuelve a la página anterior y añade a la sesión una variable.

12 BASES DE DATOS (MIGRACIONES Y ELOQUENT)

- ▶ Migraciones
 - ▶ Permiten gestionar la BD del sitio web.
 - ▶ Clases a partir de la cuales se crean las tablas de la BD
 - ▶ database/migrations
- ▶ Eloquent
 - ▶ ORM (Object Relational Mapping)
 - ▶ Transforma tablas en clases y registros en objetos
 - ▶ App/models

12.1 CONFIGURACIÓN

- ▶ Soporta múltiples SGBD
- ▶ Archivo **.env**: Se definen variables de configuración. En caso de subir el proyecto a un repositorio git, este archivo nunca se sube, por seguridad, para que no queden expuestos datos de usuarios y contraseñas.
- ▶ Configuración: **config/database**. Si no existen las variables en el archivo .env, se toman los valores por defecto de **config/database**: 'default' => env('DB_CONNECTION', 'mysql'),

```
.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:RBmERmh20vmidc3
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=laravel
15 DB_USERNAME=root
16 DB_PASSWORD=
```

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=tallerLaravel
DB_USERNAME=root
DB_PASSWORD=
```

- MariaDB 10.3+ (Version Policy)
- MySQL 5.7+ (Version Policy)
- PostgreSQL 10.0+ (Version Policy)
- SQLite 3.8.8+
- SQL Server 2017+ (Version Policy)

database.php

config > database.php

```
1 <?php
2
3 use Illuminate\Support\Str;
4
5 return [
6
7     /*
8      |-----
9      | Default Database Connection Name
10     |-----
11     |
12     | Here you may specify which of the database connections below you
13     | to use as your default connection for all database work. Of course
14     | you may use many connections at once using the Database library.
15     |
16     */
17
18     'default' => env('DB_CONNECTION', 'mysql'),
19 ]
```

12.2 MIGRACIONES

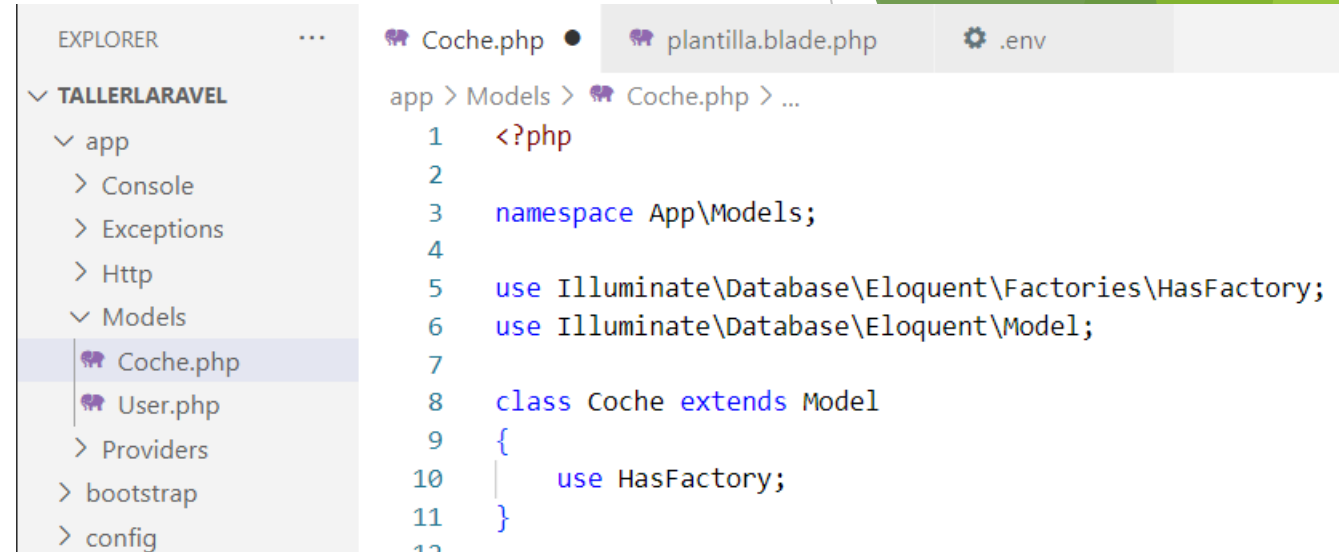
- ▶ Permiten gestionar la base de datos de nuestro sitio web; tanto crear nuevas BBDD como editarlas desde Laravel.
- ▶ Se alojan en la ruta database/migrations y tienen extensión .php. Hay 4 de Laravel que serán de utilidad para gestionar usuarios del sitio.
- ▶ Son clases que heredan de Migration. Dos métodos
 - ▶ up: Se ejecuta cuando se crea la BD
 - ▶ down: Se ejecuta cuando hay un cambio de versión y se usa para deshacer el esquema
- ▶ Las migraciones se crean las tablas de la BD y se ejecutan en el orden en el que aparecen en el explorador. Hay que tener cuidado al definir claves externas ya que la tabla donde está definida la clave primaria debe estar creada antes.
- ▶ Comandos:
 - ▶ **php artisan migrate:** Realiza una migración. Si la base de datos no existe, se crea. Ejecuta el método up de todas las migraciones que no se hayan ejecutado aún.
 - ▶ **php artisan migrate:rollback:** Deshace la última migración
 - ▶ **php artisan migrate:reset:** Borrar todas las tablas
 - ▶ **php artisan migrate:fresh:** Borra todas las tablas y luego el up. Se borran los datos!!
 - ▶ **php artisan migrate:refresh:** Ejecuta el down de todas las tablas y luego el up. Se borran los datos!!
 - ▶ **php artisan make:migration create_nombreTabla_table:** Genera una migración de una tabla, pero vamos a generar las migraciones a la vez que los modelos Eloquent.
- ▶ Visualizar el contenido de la tabla migrations

```
2014_10_12_000000_create_users_table.php X
database > migrations > 2014_10_12_000000_create_users_table.php > ...
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('users', function (Blueprint $table) {
17             $table->id();
18             $table->string('name');
19             $table->string('email')->unique();
20             $table->timestamp('email_verified_at')->nullable();
21             $table->string('password');
22             $table->rememberToken();
23             $table->timestamps();
24         });
25     }
26
27     /**
28      * Reverse the migrations.
29      *
30      * @return void
31      */
32     public function down()
33     {
34         Schema::dropIfExists('users');
35     }
36 };
37
```

	id	migration	batch
▶	1	2014_10_12_000000_create_users_table	1
	2	2014_10_12_100000_create_password_resets...	1
	3	2019_08_19_000000_create_failed_jobs_table	1
	4	2019_12_14_000001_create_personal_access...	1
*	NULL	NULL	NULL

12.3 CREAR MODELO ELOQUENT

- ▶ Son clases que representan el modelo para trabajar con las tablas de la base de datos como si fueran clases de POO y con los registros como si fueran objetos de esas clases.
- ▶ Por convención el **modelo en singular y mayúsculas la primera letra** y la **migración debe estar en plural y minúscula** (el plural se añade automáticamente, con una 's', cuidado con los plurales que deben que acaban en 'es' por ejemplo para la tabla reparación).
- ▶ **php artisan make:model Coche --migration**: Crea modelo y migración.
 - ▶ app/Models/Coche.php
 - ▶ database/migrations/~_create_coches_table.php
- ▶ Si el nombre de la tabla no coinciden con la notación usada, se puede indicar en el atributo `$table` del modelo.
- ▶ **Tinker**: Herramienta para usar Eloquent desde la línea de comandos cuando aún no están definidos controladores ni vistas.
 - ▶ php artisan tinker
 - ▶ exit



```
EXPLORER
TALLERLARAVEL
├── app
│   ├── Console
│   ├── Exceptions
│   ├── Http
│   └── Models
│       ├── Coche.php
│       ├── User.php
│       ├── Providers
│       ├── bootstrap
│       └── config
├── ...
└── ...

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

```
namespace App\Models;

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

class Reparacion extends Model
{
    use HasFactory;
    protected $table = 'reparaciones';
}
```

12.4 DEFINIR MIGRACIÓN

- ▶ Permite definir los campos que tiene la tabla
- ▶ Tipos de columnas de tablas:
<https://laravel.com/docs/11.x/migrations#available-column-types>
- ▶ Opciones de columnas:
<https://laravel.com/docs/11.x/migrations#column-modifiers>

```
public function up()
{
    Schema::create('personal_access_tokens', function (Blueprint $table) {
        $table->id();
        $table->morphs('tokenable');
        $table->string('name');
        $table->string('token', 64)->unique();
        $table->text('abilities')->nullable();
        $table->timestamp('last_used_at')->nullable();
        $table->timestamp('expires_at')->nullable();
        $table->timestamps();
    });
}
```

```
public function up()
{
    Schema::create('piezas', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
        $table->enum("clase",["Refrigeración","Filtro","Motor","Otros"]);
        $table->string("descripcion");
        $table->float("precio");
        $table->integer("stock");
    });
}
```

```
public function up()
{
    Schema::create('reparaciones', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
        $table->foreignId("coche_id")->constrained()->onDelete("restrict")->onUpdate("cascade");
        $table->dateTime("fecha");
        $table->integer("tiempo");
        $table->boolean("pagado")->default(false);
    });
}
```


12.5 AGREGAR/MODIFICAR/BORRRAR COLUMNA

► Agregar

- `php artisan make:migration add_color_to_coches_table`: Crea una migración para añadir el campo

► Modificar

- Instalar librería: `composer require doctrine/dbal`
- `php artisan make:migration modificarColumnas_to_vehiculo_table`

```
7 class ModificarColumnasToCochesTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::table('coches', function (Blueprint $table) {
17             $table->string("matricula",7)->change(); //Cambiar Longitud
18             $table->string("color")->nullable(false)->change(); //Cambiar admite nulos
19         });
20     }
```

```
2022_12_26_115612_add_color_to_coches_table.php X
database > migrations > 2022_12_26_115612_add_color_to_coches_table.php > AddColorToCochesTable
7 class AddColorToCochesTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::table('coches', function (Blueprint $table) {
17             //Añadir al final
18             $table->string("color")->nullable(true);
19             //Añadir al detrás de una columna con creat
20             //$table->string("color")->nullable(true)->after("matricula");
21         });
22     }
```

```
public function up(): void
{
    //
    Schema::table('clientes',function(Blueprint $table){
        $table->dropColumn('email');
        $table->dropColumn('nombre');
    });
}
```

12.6 CREAR CLAVES EXTERNAS

- ▶ <https://laravel.com/docs/11.x/migrations#foreign-key-constraints>
- ▶ Si se sigue la convención de nombres de laravel, el nombre del campo sobre el que se define la clave externa debe llamarse **nombreTablaSingular_id**. Por ejemplo, una clave externa a un id de la tabla piezas debería llamarse **pieza_id**.
- ▶ Si la clave externa apunta a una clave primaria cuyo nombre sigue la convención de nombres de Laravel (La clave primaria está en la tabla **pieza** en el campo **id**)

```
$table->foreignId("pieza_id")->constrained()->onDelete('restrict')->onUpdate('cascade');
```

- ▶ En caso de que no se siga la convención de nombres de Laravel

```
$table->string('paciente')->nullable(false);  
$table->foreign('paciente')  
    ->references('dni')  
    ->on('pacientes')  
    ->onDelete('restrict')  
    ->onUpdate('cascade');  
$table->integer('dentista')->nullable(false);  
$table->foreign('dentista')->references('numCol')  
    ->on('dentistas')  
    ->onDelete('restrict')  
    ->onUpdate('cascade')  
    ->nullable(false);
```

12.7 ATRIBUTOS DEFINIDOS EN EL MODELO

- ▶ Si no se siguen la convención de nombres de Laravel es posible indicar en el modelo ciertas propiedades.
 - ▶ Nombre de la clase y nombre de la tabla no siguen la convención laravel
 - ▶ `protected $table = 'nombreTabla';`
 - ▶ Clave primaria no se llama id
 - ▶ `protected $primaryKey = 'campold';`
- ▶ Eloquent no soporta claves primarias compuestas

```
class Pieza_reparacion extends Model
{
    use HasFactory;
    protected $table = "pieza_reparaciones";

    function pieza(){
        //Recupera la pieza de esta pieza_reparación
        return $this->belongsTo(Pieza::class);
    }

    function reparacion(){
        //Recupera la reparacion de esta pieza_reparación
        return $this->belongsTo(Reparacion::class);
    }
}
```

12.8 RELACIONES 1:N

- ▶ Eloquent vamos a poder gestionar las relaciones entre nuestras tablas de la base de datos de una manera muy sencilla y sin sentencias SQL.
- ▶ Se crean métodos en el modelo que recuperan los registros relacionados.

- ▶ Lado 1: 1 vehículo pertenece a 1 propietario o 1 Reparación es de un coche
 - ▶ Se define un método que devuelve `this->belongsTo()`. Se devuelve el objeto relacionado. Si se respeta la convención de nombres solamente hay que indicar la clase relacionada; si no se sigue la convención de nombres hay que indicar también el campo sobre el que se define la clave externa (`foreign_key`), y el campo que define la clave primaria (`owner_key`).

```
return $this->belongsTo(Post::class, 'foreign_key', 'owner_key');
```

- ▶ Este método se trata como un atributo más de la clase. Aunque es un método, no es necesario poner paréntesis cuando se le llama.

```
<td>{{${c->propietario->nombre}}}</td>
```

- ▶ Lado N: 1 reparación tiene muchas piezas
 - ▶ Se define un método que devuelve `this->hasMany()->get()`. Se devuelve un array con todos los objetos relacionados. Si se respeta la convención de nombres solamente hay que indicar la clase relacionada; si no se sigue la convención de nombres hay que indicar también el campo sobre el que se define la clave externa (`foreign_key`), y el campo que define la clave primaria (`local_key`).

```
return $this->hasMany(Comment::class, 'foreign_key', 'local_key');
```

- ▶ Este método se trata como un atributo más de la clase, en este caso sí es necesario poner paréntesis cuando se le llama.

```
{{-- Mostrar las pieza de la reparación --}}
```

```
@foreach ($r->piezasReparacion() as $pr)
```

```
class Coche extends Model
{
    use HasFactory;

    //Creamos la relación muchos a 1
    // 1 coche es de un propietario
    function propietario(){
        return $this->belongsTo(Propietario::class);
    }
}
```

```
class Reparacion extends Model
{
    use HasFactory;
    //Este modelo espera que la tabla se llame reparacions,
    //pero nuestra tabla se llama reparaciones
    protected $table = "reparaciones";

    function coche(){
        return $this->belongsTo(Coche::class);
    }

    function piezasReparacion(){
        return $this->hasMany(Pieza_reparacion::class)->get();
    }
}
```

12.9 SEEDERS

- ▶ Permiten inicializar las tablas de la base de datos con datos.
- ▶ Comando: **php artisan make:seeder UsersS**
- ▶ Se almacenan en database/seeders
- ▶ Ejecutar seeders
 - ▶ Los indicados en DatabaseSeeder.php: **php artisan db:seed**
 - ▶ Uno concreto: **php artisan db:seed --class=UserSeeder**

```
UsersS.php X
database > seeders > UsersS.php > ...
9  use Illuminate\Support\Facades\Schema;
10
11  class UsersS extends Seeder
12  {
13      /**
14       * Run the database seeds.
15       */
16      public function run(): void
17      {
18          //
19          DB::table('users')->insert([
20              'name' => 'admin',
21              'email' => 'admin@admin.com',
22              'password' => Hash::make('admin'),
23          ]);
24      }
25  }
```

```
DatabaseSeeder.php X
database > seeders > DatabaseSeeder.php > ...
8  class DatabaseSeeder extends Seeder
9  {
10      /**
11       * Seed the application's database.
12       */
13      public function run(): void
14      {
15          // \App\Models\User::factory(10)->create();
16
17          // \App\Models\User::factory()->create([
18              //     'name' => 'Test User',
19              //     'email' => 'test@example.com',
20              // ]);
21          $this->call([
22              UsersS::class
23          ]);
24      }
25  }
```

13 CRUD

13.1 CONSULTAS

Operación	Consulta
Select * from vehiculos	Vehiculo::all()
Select * from vehículos order by matricula desc, color	Vehiculo::orderBy('matricula',DESC)->orderBy('color')->get()
Select * from vehiculos where color = rojo	\$vehiculos = Vehiculo::where('color','rojo')->get()
Select * from vehiculos where color = rojo order by codigo desc	\$vehiculos = Vehiculo::where('color','rojo')->orderBy('codigo','desc')->get()
Select * from vehiculos where color = rojo order by codigo desc limit 1	\$vehiculos = Vehiculo::where('color','rojo')->orderBy('codigo','desc')->limit(1)->get()
Select * from vehiculos where codigo = 1	\$vehiculo = Vehiculo::find(1)
Select * from vehiculos where color = rojo order by codigo asc limit 1	\$vehiculo = Vehiculo::where('color','rojo')->orderBy('codigo','desc')->first()
Operadores: =, <, >, <=, >=, <>, !=, LIKE, NOT LIKE, BETWEEN, ILIKE(case-sensitive)	\$vehiculos = Vehiculo::where('codigo','<','10')->get()
AND	\$vehiculos = Vehiculo::where('codigo','<','10')->Where('color','azul')->get()
OR	\$vehiculos = Vehiculo::where('codigo','<','10')->orWhere('color','azul')->get()

13.2 SELECT

- El controlador recupera la información y se la pasa a la vista en el parámetro compact.
- Hay que tener en cuenta si se recupera un sólo objeto o se recupera un array

```
cocheController.php ×
app > Http > Controllers > cocheController.php > cocheController > mos
8 class cocheController extends Controller
9 {
10     function inicio(){
11         //RECUPERAR TODOS LOS COCHES
12         $coches = Coche::all();
13         //PASAR LOS COCHES A LA VISTA
14         return view('coches/index',compact('coches'));
15     }
16 }
```

index.blade.php ×

resources > views > coches > index.blade.php > ...

```
1 @extends('plantilla')
2 @section('apartado',"MOSTRAR VEHÍCULOS")
3 @section('contenido')
4 <table border="1">
5     <tr>
6         <th>Código</th>
7         <th>Propietario</th>
8         <th>Matrícula</th>
9         <th>Color</th>
10    </tr>
11    {{-- MOSTRAR COCHES --}}
12    @foreach ($coches as $c)
13        <tr>
14            <td>{{$c->id}}</td>
15            <td>{{$c->propietario}}</td>
16            <td>{{$c->matricula}}</td>
17            <td>{{$c->color}}</td>
18        </tr>
19    @endforeach
20 </table>
21 @endsection
22 @section('mensaje')
```

13.3 PAGINACIÓN

- Paginación de Eloquent
 - Compatible con la paginación de Bootstrap y TailWindCSS
 - Incluir la Librería tailwindcss para que se vean correctamente las flechas y los números
 - `<script src="https://cdn.tailwindcss.com"></script>`

```
plantilla.blade.php × cocheController.php index.blade.php
resources > views > plantilla.blade.php > ...
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>Taller</title>
6 <script src="https://cdn.tailwindcss.com"></script>
7 </head>
8 <body>
9 <header>
```

```
cocheController.php × index.blade.php
app > Http > Controllers > cocheController.php > cocheController > ini
2
3 namespace App\Http\Controllers;
4
5 use App\Models\Coche;
6 use App\Models\Propietario;
7 use Illuminate\Http\Request;
8
9 class cocheController extends Controller
10 {
11     function inicio(){
12         //RECUPERAR TODOS LOS COCHES
13         //$coches = Coche::all();
14         $coches = Coche::paginate(1);
15         //PASAR LOS COCHES A LA VISTA
16         return view('coches/index',compact('coches'));
17     }
}
```

```
index.blade.php ×
resources > views > coches > index.blade.php >
24 <a href="{{route('verCoche', $coche)}}">Ver Coche</a>
25 <a href="{{route('verCoche', $coche)}}">Ver Coche</a>
26 <a href="{{route('verCoche', $coche)}}">Ver Coche</a>
27 </a>
28 </td>
29 </tr>
30 @endforeach
31 </table>
32 {{ $coches->links() }}
33 @endsection
34 @section('mensaje')
```


13.4 CREATE

- ▶ Lo normal es recuperar la información del objeto que se va a crear de un formulario por el método **post**. Para ello, el método del controlador que trata ese formulario debe tener como parámetro un objeto de la clase **Request**.
- ▶ Se debe crear un objeto de la clase deseada y rellenar sus atributos con los campos del formulario. A ellos se accede a través del objeto de la clase Request y su nombre es el name que se les ha asignado en el formulario.
- ▶ Para hacer el insert en la tabla se ejecuta el método **save()**.
- ▶ Pueden hacerse inserciones por consulta:

```
use App\Models\Flight;

$flight = Flight::create([
    'name' => 'London to Paris',
]);
```

```
20 Route::controller(cocheController::class)->group(
21     function(){
22         Route::get('coches', "inicio") -> name("coches");
23         Route::get('coches/crear', "crear")-> name("crearCoche");
24         Route::post('coches/crear', "insertar")-> name("insertarCoche");
```

crear.blade.php

resources > views > coches > crear.blade.php > ...

```
3 @section('contenido')
4 <form action="{{route('insertarCoche')}}" method="post">
5     {{-- @csrf:Cláusula de seguridad para evitar ataques desde otros sitios--}}
6     @csrf {{-- Cláusula para obtener un token de formulario al enviarlo --}}
7     <p>
8         <label>Propietario</label><br/>
9         <select name="propietario">
10             @foreach ($propietarios as $p)
11                 <option value="{{{$p->id}}}">{{{$p->nombre}}}</option>
12             @endforeach
13         </select>
14     </p>
```

cocheController.php

crear.blade.php

app > Http > Controllers > cocheController.php > cocheController > insertar

```
17 function crear(){
18     //Propietarios para el select
19     $propietarios = Propietario::all();
20     return view('coches/crear',compact('propietarios'));
21 }
22 function insertar(Request $request){
23     //Validaciones
24     $request->validate(["propietario"=>"required",
25         "matricula"=>"required", "color"=>"required"]);
26     $coche = new Coche();
27     $coche->matricula = $request->matricula;
28     $coche->propietario = $request->propietario;
29     $coche->color = $request->color;
30     //Guardar el coche en la BD
31     if($coche->save()){
32         $mensaje="Vehículo creado correctamente";
33     }
34     else{
35         $mensaje="Error al crear el vehículo";
36     }
37     return back()->with("mensaje",$mensaje);
38 }
```

25
26
--

```
Route::get('coches/editar/{id}', "editar")-> name("editarCoche");//Carga formulario
Route::put('coches/editar/{id}', "modificar")-> name("modificarCoche");//Guardar en la BD
```

13.5 UPDATE

- Lo normal es crear rutas **put** y recuperar la información del objeto que se va a modificar de un formulario por el método **put**. Para ello, el método del controlador que trata ese formulario debe tener como parámetro un objeto de la clase **Request**.
- Se debe recuperar el objeto a modificar de la BD y sobre él hacer las modificaciones que vienen del formulario.
- Para hacer el update en la tabla se ejecuta el método **save()** del objeto recuperado con la modificaciones de los campos realizadas.
- Se pueden realizar actualizaciones masivas por consulta

```
Flight::where('active', 1)
->where('destination', 'San Diego')
->update(['delayed' => 1]);
```

```
editar.blade.php X
resources > views > coches > editar.blade.php > form > p > input
4 <form action="{{route('modificarCoche',$coche->id)}}" method="post">
5 @method('PUT') {{-- Cambiar método de post a put --}}
6 {{-- @csrf:Cláusula de seguridad para evitar ataques desde otros sitios--}}
7 @csrf {{-- Cláusula para obtener un token de formulario al enviarlo --}}
8 <p>
9 <label>Propietario</label><br/>
10 <select name="propietario">
```

```
cocheController.php X
app > Http > Controllers > cocheController.php > ...
50 }
51 function editar($id){
52     //Propietarios para el select
53     $propietarios = Propietario::all();
54     $coche=Coche::find($id);
55     return view('coches/editar',compact('coche','propietarios'));
56 }
57 function modificar(Request $request, $id){
58     //Validaciones
59     $request->validate(["propietario"=>"required",
60 "matricula"=>"required","color"=>"required"]);
61     $coche = Coche::find($id);
62     $coche->matricula = $request->matricula;
63     $coche->propietario = $request->propietario;
64     $coche->color = $request->color;
65     if($coche->save()){
66         return back() -> with('mensaje', 'Vehículo modificado');
67     }
68     else{
69         return back() -> with('mensaje', 'Error al modificar el vehículo');
70     }
71 }
```

13.6 DELETE

- ▶ Se pueden definir rutas para el método delete. En el formulario hay que modificar el método con @method para poder usar estas rutas.
- ▶ Lo normal es recuperar la información del objeto que se va a borrar de un formulario por el método **delete**. Para ello, el método del controlador que trata ese formulario debe tener como parámetro un objeto de la clase **Request**.
- ▶ Se debe recuperar el objeto a borrar de la BD.
- ▶ Para hacer el delete en la tabla se ejecuta el método **delete()** del objeto recuperado.
- ▶ Controlador: Crear función para confirmar y para borrar
- ▶ Es posible borrar varios registros con el método destroy del modelo concreto y pasando las claves primarias de los registros a borrar.
- ▶ Es posible realizar borrados masivos usando una consulta:

```
Flight::destroy([1, 2, 3]);
```

```
$deleted = Flight::where('active', 0)->delete();
```

```
28 Route::delete('coches/borrar/{id}', "borrar")-> name("borrarCoche");//Borrar de la BD
29 Route::get('coches/{matricula}/{propietario?}', "mostrar")-> name("verCoche");
30 }
31 );
```

preBorrar.blade.php X

resources > views > coches > preBorrar.blade.php > form > table > tr > td

```
4
5 <form action="{{route('borrarCoche',$coche->id)}}" method="post">
6     @method('DELETE') {{-- Cambiar método de post a delete --}}
7     {{-- @csrf:Cláusula de seguridad para evitar ataques desde otros sitios--}}
8     @csrf {{-- Cláusula para obtener un token de formulario al enviarlo --}}
```

cocheController.php X

app > Http > Controllers > cocheController.php > ...

```
70 }
71 }
72 function preBorrar($id){
73     $coche=Coche::find($id);
74     return view('coches/preBorrar',compact('coche'));
75 }
76 function borrar(Request $request, $id){
77
78     $coche = Coche::find($id);
79     $ok = $coche->delete();
80     $coches = Coche::all();
81     if($ok){
82         return redirect()->route('coches') -> with('mensaje', 'Vehículo borrado');
83     }
84     else{
85         return redirect()->route('coches') -> with('mensaje', 'Error al borrar el vehículo');
86     }
87 }
88 }
```

13.7 TRANSACCIONES

► De forma automática

- No hay que preocuparse de hacer commit o rollback.
- `DB::transaction(function() use ($p1, $p2){acciones}.`

```
try {
    DB::transaction(function() use ($c){
        if($c->save()){
            if(!$c->usuario->save()){
                return back()->with('mensaje','Error, no se ha modificado el cliente');
            }
        }
        else{
            return back()->with('mensaje','Error, no se ha modificado el cliente');
        }
    });
} catch (Exception $e) {
    $error=true;
    return back()->with('mensaje','Error, no se ha modificado el cliente');
}
finally{
    if(!$error){
        return redirect()->route('clientes')->with('mensaje','Cliente modificado correctamente');
    }
}
```

► Manual

```
DB::beginTransaction();

try {
    DB::insert(...);
    DB::insert(...);
    DB::insert(...);

    DB::commit();
    // all good
} catch (\Exception $e) {
    DB::rollback();
    // something went wrong
}
```

14 VALIDACIONES

Definición de la validación

```
function insertarCoche(Request $request){
    //request contienen los datos del formulario que ha hecho el post
    //Validaciones
    $request->validate(['propietario'=>'required',
        'matricula'=>'required|min:7|max:10',
        'color'=>'required']);
}

function insertarPaciente(Request $request){
    //Validaciones de campos
    $request->validate([
        'dni'=> 'required|min:9|unique:App\Models\Paciente,dni',
        'nombre'=>'required',
        'telefono'=>'required',
        'email'=>'required|email:rfc,dns',
        'fechaN'=>'required'
    ]);
}
```

Reglas validate: <https://laravel.com/docs/10.x/validation#available-validation-rules>

Configuración de Idioma: config/app.php 'locale' => 'es',

Texto Validaciones: resources/lang/es/validation.php

Mensaje Personalizado

```
crear.blade.php X
resources > views > coches > crear.blade.php > form > p
6      @csrf {{-- Clausula para obtener un token de formulario al enviarlo --}}
7      <p>
8          <label>Propietario</label><br/>
9          <select name="propietario">
10             @foreach ($propietarios as $p)
11                 <option value="{{ $p->id }}">{{ $p->nombre }}</option>
12             @endforeach
13         </select>
14         @error('propietario')
15             <div class="alert alert-danger">
16                 Selecciona Propietario
17             </div>
18         @enderror
19     </p>
```

Mensaje Configurado en validation.php

```
@error('matricula')
    <div class="alert alert-danger">
        {{ $message }}
    </div>
@enderror
```

Obtener todos los mensajes

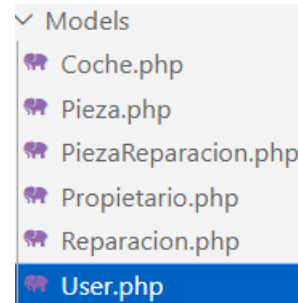
```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

15 RECORDAR DATOS

```
crear.blade.php X
resources > views > coches > crear.blade.php > form > p > select > option
7      </p>
8      <label>Propietario</label><br/>
9      <select name="propietario">
10         @foreach ($propietarios as $p)
11             @if (old('propietario')== $p->id)
12                 <option value="{{ $p->id }}" selected="selected">{{ $p->nombre }}</option>
13             @else
14                 <option value="{{ $p->id }}">{{ $p->nombre }}</option>
15             @endif
16         @endforeach
17     </select>
18     @error('propietario')
19         <div class="alert alert-danger">
20             Selecciona Propietario
21         </div>
22     @enderror
23 </p>
24 <p>
25     <label>Matrícula</label><br/>
26     <input type="text" name="matricula" placeholder="1234ABC" maxlength="7"
27         value="{{ old('matricula') }}">
28     @error('matricula')
29         <div class="alert alert-danger">
30             Introduce la Matrícula
31         </div>
32     @enderror
33 </p>
34 <p>
35     <label>Color</label><br/>
36     <input type="text" name="color" placeholder="color" value="{{ old('color') }}">
37     @error('color')
38         <div class="alert alert-danger">
```

16 AUTENTICACIÓN DE USUARIOS

- ▶ **Laravel Breeze:** Sistema de autenticación de Laravel. Incluye todas las características: login, registros, cambiar contraseña, verificación de email y confirmación de contraseña. Más rígido.
- ▶ **Sistema Personalizado:** Usando las clases que proporciona Laravel. Adaptado a nuestras necesidades.
 - ▶ Crear rutas:
 - ▶ get: login, registro, salir
 - ▶ post: loguear, registrar
 - ▶ Crear vistas: login y registro
 - ▶ Crear controlador Login



```
2014_10_12_000000_create_users_table.php X
database > migrations > 2014_10_12_000000_create_users_table.php > CreateUsersTable

7  class CreateUsersTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('users', function (Blueprint $table) {
17             $table->id();
18             $table->string('name');
19             $table->string('email')->unique();
20             $table->timestamp('email_verified_at')->nullable();
21             $table->string('password');
22             $table->rememberToken();
23             $table->timestamps();
24         });
25     }
}
```


17.1 RUTAS

```
web.php x LoginController.php Authenticate.php plantilla.blade.php login.blade.php registro.blade.php
routes > web.php > #Function#57f5c0d3
21
22 Route::controller(cocheController::class)->group(
23     function(){
24         Route::get('coches', "inicio") -> name("coches")->middleware("auth");
25         Route::get('coches/crear', "crear")-> name("crearCoche")->middleware("auth");
26         Route::post('coches/crear', "insertar")-> name("insertarCoche")->middleware("auth");
27         Route::get('coches/editar/{id}', "editar")-> name("editarCoche")->middleware("auth");//Carga formulario
28         Route::put('coches/editar/{id}', "modificar")-> name("modificarCoche")->middleware("auth");//Guardar en la BD
29         Route::get('coches/preBorrar/{id}', "preBorrar")-> name("preBorrarCoche")->middleware("auth");//Carga formula
30         Route::delete('coches/borrar/{id}', "borrar")-> name("borrarCoche")->middleware("auth");//Borrar de la BD
31         Route::get('coches/{matricula}/{propietario?}', "mostrar")-> name("verCoche")->middleware("auth");
32     }
33 );
34 Route::controller(LoginController::class)->group(
35     function(){
36         Route::get('login', "login")->name("login");
37         Route::get('registro', "registro")->name("registro");
38         Route::post('loguear', "loguear")->name("loguear");
39         Route::post('registrar', "registrar")->name("registrar");
40         Route::get('salir', "salir")->name("salir");
41     }
42 );
```

Cuidado con el import!!, debe ser de nuestro controller!!

Hay que indicar que solamente podemos acceder a la ruta si estamos logueados y en caso de no estarlo se redirige a lo que esté configurado en **Authenticate.php**. Se puede hacer de forma individual en cada ruta que lo necesite o en el constructor del controlador

```
Authenticate.php x LoginController.php plantilla.blade.php login.blade.php registro
app > Http > Middleware > Authenticate.php > ...
1 <?php
2
3 namespace App\Http\Middleware;
4
5 use Illuminate\Auth\Middleware\Authenticate as Middleware;
6
7 class Authenticate extends Middleware
8 {
9     /**
10      * Get the path the user should be redirected to when they are not authenticated.
11      *
12      * @param \Illuminate\Http\Request $request
13      * @return string|null
14      */
15     protected function redirectTo($request)
16     {
17         if (! $request->expectsJson()) {
18             return route('login');
19         }
20     }
21 }
```

```
class CocheController extends Controller
{
    function __construct()
    {
        //Siempre verifica que se este logueado
        $this->middleware('auth');
    }
}
```


17.2 LoginController

LoginController.php × plantilla.blade.php login.blade.php registro.blade.php

app > Http > Controllers > LoginController.php > LoginController > registrar

```
9
10 class LoginController extends Controller
11 {
12     function login(){
13         return view('login/login');
14     }
15
16     function registro(){
17         return view('login/registro');
18     }
19
20     function loguear(Request $request){
21         //Validaciones
22         $request->validate(["email"=>"required|email:rfc,dns",
23             "ps"=>"required"]);
24         //Credenciales de acceso
25         $credenciales=["email"=>$request->email,"password"=>$request->ps];
26         //Recordar credenciales
27         $recordar = $request->has("recordar");
28         //Autenticar
29         if(Auth::attempt($credenciales,$recordar)){
30             $request->session()->regenerate();
31             return redirect()->intended("coches");
32         }
33         else{
34             return back()->with("mensaje","Email o contraseña no válidos");
35         }
36     }
37 }
```

```
function registrar(Request $request){
    //Validaciones
    $request->validate(["email"=>"required|unique:App\Models\User,email|email:rfc,dns",
        "nombre"=>"required","ps"=>"required"]);
    //Crear modelo User
    $user = new User();
    $user->name=$request->nombre;
    $user->email=$request->email;
    $user->password= Hash::make($request->ps);
    //Guardar el usuario
    if($user->save()){
        //Indicar que el usuario está autenticado
        Auth::login($user);
        return redirect(route("coches"));
    }
    else{
        return back()->with("mensaje","Se ha producido un error al registrarse");
    }
}

function salir(Request $request){
    Auth::logout();
    $request->session()->invalidate();
    $request->session()->regenerateToken();
    return redirect(route("login"));
}
```

17.3 INFO USUARIO LOGUEADO

plantilla.blade.php ×

resources > views > plantilla.blade.php > ...

```
23 <a class="btn btn-light" href="{{ route('crearCoche') }}">Crear Coche</a>
24 <a class="btn btn-light" href="{{ route('crearCoche') }}">Reparaciones</a>
25 <a class="btn btn-light" href="{{ route('salir') }}">Salir</a>
26 @auth
27 {{Auth::user()->email}} {{Auth::user()->name}}
28 @endauth
29 <hr/>
30 </nav>
```

18 IMÁGENES

- ▶ Se guardan en la carpeta **public**

- ▶ Para mostrarlas ``

- ▶ Subir imagen desde formulario

- ▶ Controlador

```
//Subir imagen al servidor
//Sube un fichero procedente de un formulario (campo imagen) a la carpeta img
// de storage/app/public.
$path = $r->file('imagen')->store('img','public');
//Recuperar el path de la imagen y guardarlo en variable
$p->img=$path;
```

- ▶ Crear enlace simbólico a carpeta storage en la carpeta public **php artisan storage:link**

- ▶ Mostrar imagen ``

- ▶ Borrar imagen almacenada en Storage

```
//Borrar la imagen
Storage::delete('public/'.$p->img);
```

19 VARIABLES DE SESIÓN

- ▶ Crear variable de sesión: `session(["producto"=>$p]);`
- ▶ Destruir variable de sesión: `session()->forget('producto');`
- ▶ Acceder a variable de sesión: `session('producto')`
- ▶ Mostar en blade una variable de sesión:

```
@if(session('producto')!=null)
    <h1>
        Producto{{session('producto')}}
    </h1>
@endif
```