

PROGRAMACIÓN WEB

Manual LARAVEL



Autor:

Alejandro Guerrero Medina

Grado en Ingeniería Informática

Curso 2022 - 2023

Índice

1. Prolegómeno	2
2. Instalación de Laravel	2
2.1. Requisitos del sistema	2
2.2. Instalación mediante Composer	2
3. Creación de Base de Datos	5
4. Instalación de Bootstrap	6
5. Lanzamiento de la aplicación	7
6. Creación 1º Prototipo de la web de los artículos	8
6.1. Creación Controlador de Artículos	8
6.2. Creación layout general de la Aplicación	8
6.3. Creación vista <code>artículos.blade.php</code>	9
6.4. Registro de las rutas creadas	9
7. Creación de formulario para introducir artículos	10
7.1. Creación del request para validar el formulario	11
7.2. Acceso a la base de datos	12
7.3. Insertar artículos en la base de datos desde el formulario	16
7.4. Paginación de los artículos	17
8. Login y Registro Usuario	18
8.1. Integración de los formularios en la web	19
9. Conclusiones	24

1. Prolegómeno

Sea este documento una guía sobre el *framework* basado en PHP, LARAVEL, con objeto de especificar cada paso realizado para desplegar una aplicación web en correcto funcionamiento siguiendo el manual proporcionado por el campus virtual, así como ayudarse de la documentación oficial en caso de error.

2. Instalación de Laravel

2.1. Requisitos del sistema

Para la correcta funcionalidad de LARAVEL JETSTREAM se necesita:

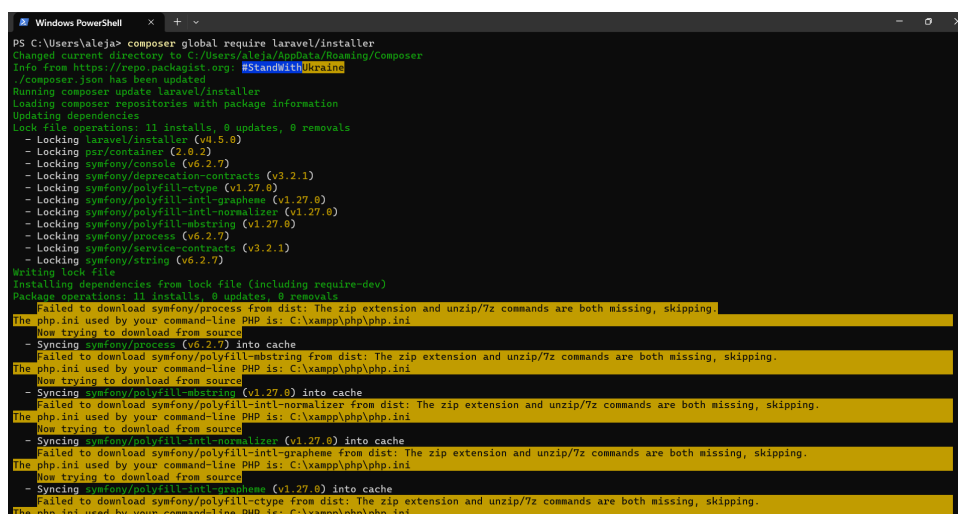
- Versión de PHP igual o superior a la 7.3.0
- Extensión de PHP OpenSSL
- Extensión de PHP PDO
- Extensión de PHP Mbstring
- Extensión de PHP Tokenizer
- Extensión de PHP XML

2.2. Instalación mediante Composer

Al igual que CODEIGNITER, LARAVEL puede instalarse mediante el manejador de dependencias de PHP, COMPOSER. Simplemente, iniciamos una nueva terminal y ejecutamos los siguientes comandos:

```
$ composer global require laravel/installer
```

Comenzará a descargar el *framework* desde el repositorio en el que se encuentra.



```
PS C:\Users\aleja> composer global require laravel/installer
Changed current directory to C:\Users\aleja\AppData\Roaming\Composer
Info from https://repo.packagist.org: #StandWithUkraine
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Updating dependencies
Lock file operations: 11 installs, 0 updates, 0 removals
  - Locking laravel/installer (v4.5.0)
  - Locking psr/container (2.0.2)
  - Locking symfony/console (v6.2.7)
  - Locking symfony/deprecation-contracts (v3.2.1)
  - Locking symfony/polyfill-ctype (v1.27.0)
  - Locking symfony/polyfill-intl-grapheme (v1.27.0)
  - Locking symfony/polyfill-intl-normalizer (v1.27.0)
  - Locking symfony/polyfill-mbstring (v1.27.0)
  - Locking symfony/process (v6.2.7)
  - Locking symfony/service-contracts (v3.2.1)
  - Locking symfony/string (v6.2.7)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 11 installs, 0 updates, 0 removals
  - Failed to download symfony/process from dist: The zip extension and unzip/7z commands are both missing, skipping.
    The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
    Use the --prefer-dist option to force installation from source.
  - Syncing symfony/process (v6.2.7) into cache
  - Failed to download symfony/polyfill-mbstring from dist: The zip extension and unzip/7z commands are both missing, skipping.
    The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
    Use the --prefer-dist option to force installation from source.
  - Syncing symfony/polyfill-mbstring (v1.27.0) into cache
  - Failed to download symfony/polyfill-intl-normalizer from dist: The zip extension and unzip/7z commands are both missing, skipping.
    The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
    Use the --prefer-dist option to force installation from source.
  - Syncing symfony/polyfill-intl-normalizer (v1.27.0) into cache
  - Failed to download symfony/polyfill-intl-grapheme from dist: The zip extension and unzip/7z commands are both missing, skipping.
    The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
    Use the --prefer-dist option to force installation from source.
  - Syncing symfony/polyfill-intl-grapheme (v1.27.0) into cache
  - Failed to download symfony/polyfill-ctype from dist: The zip extension and unzip/7z commands are both missing, skipping.
    The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
    Use the --prefer-dist option to force installation from source.
```

Figura 1: Instalación de LARAVEL.

Luego, de nuevo con una terminal, tendremos que dirigirnos a nuestro directorio raíz donde tengamos instalado XAMPP, concretamente a la carpeta `htdocs`. Nuestra ruta es: “C:\xampp\htdocs”, y ejecutamos el siguiente comando

```
$ laravel new nombreProyecto --jet
```

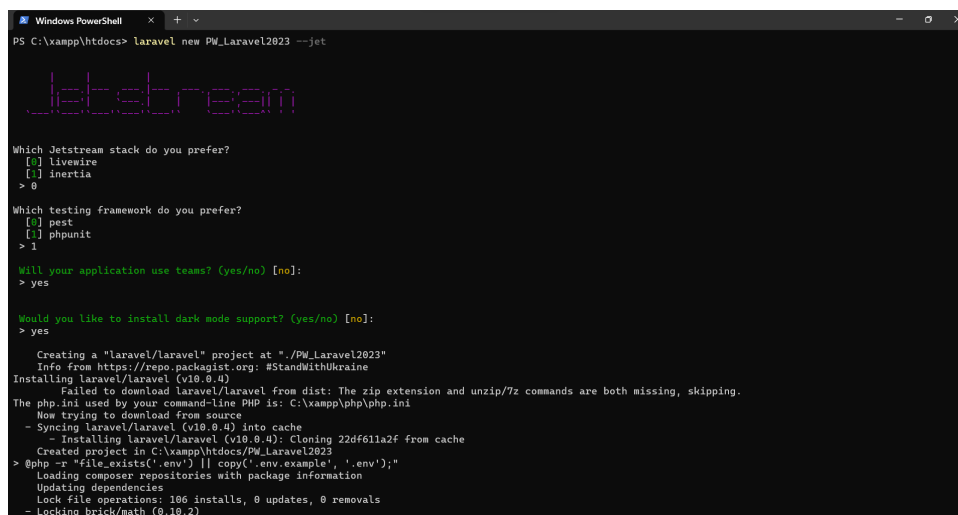
Comenzará a crear nuestro proyecto, cuyo nombre será *PW_Laravel2023*. Debemos indicar las siguientes etiquetas –a continuación se muestra como se vería en la terminal reflejado–:

```
Which Jetstream stack do you prefer?
  [0] livewire
  [1] inertia
> 0

Which testing framework do you prefer?
  [0] pest
  [1] phpunit
> 1

Will your application use teams? (yes/no) [no]:
> yes

Would you like to install dark mode support? (yes/no) [no]:
> yes
```



```
Windows PowerShell
PS C:\xampp\htdocs> laravel new PW_Laravel2023 --jet

Jetstream

Which Jetstream stack do you prefer?
  [0] livewire
  [1] inertia
> 0

Which testing framework do you prefer?
  [0] pest
  [1] phpunit
> 1

Will your application use teams? (yes/no) [no]:
> yes

Would you like to install dark mode support? (yes/no) [no]:
> yes

Creating a "laravel/laravel" project at "../PW_Laravel2023"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v10.0.0)
Failed to download laravel/laravel from dist: The zip extension and unzip/7z commands are both missing, skipping.
The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
Now trying to download from source
- Syncing laravel/laravel (v10.0.0) into cache
- Installing laravel/laravel (v10.0.0): Cloning 22df611a2f from cache
Created project in C:\xampp\htdocs\PW_Laravel2023
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 106 installs, 0 updates, 0 removals
- Locking brick/math (0.10.2)
```

Figura 2: Creación del proyecto *PW_Laravel2023*.

Seguidamente, ejecutamos los comandos:

```
$ cd PW_Laravel2023
$ npm install
$ npm run dev
```

Nos aparecerá una pequeña interfaz en la terminal, como la siguiente:

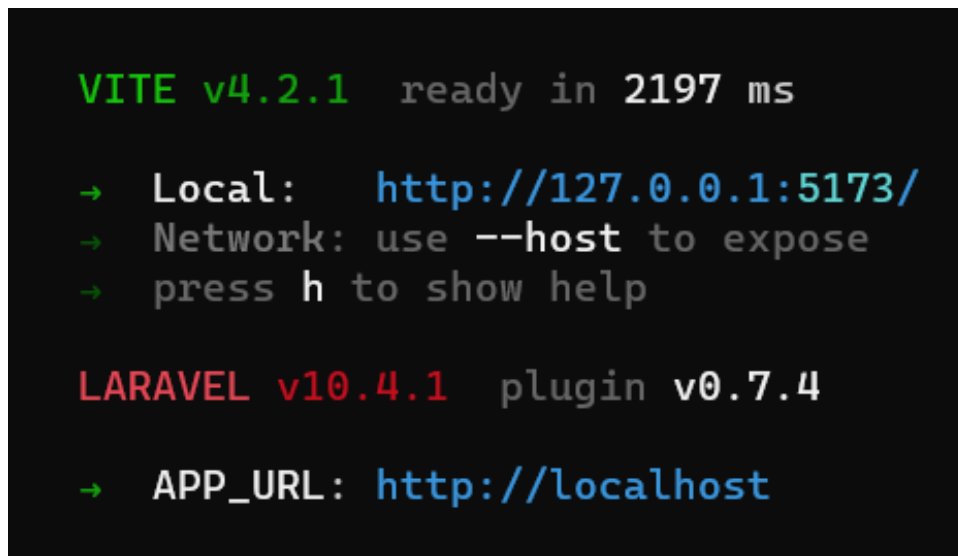


Figura 3: Interfaz de npm run dev.

A continuación, en nuestro *Visual Studio Code*, abrimos el directorio del proyecto creado, buscamos el fichero `jetstream.php`, nos dirigimos a la sección “features” y descomentamos las características permitidas:

```
1 // ...
2 'features' => [
3     Features::termsAndPrivacyPolicy(),
4     Features::profilePhotos(),
5     Features::api(),
6     Features::teams(['invitations' => true]),
7     Features::accountDeletion(),
8 ],
9 // ...
```

```
/*
|-----
| Features
|-----
|
| Some of Jetstream's features are optional. You may disable the features
| by removing them from this array. You're free to only remove some of
| these features or you can even remove all of these if you need to.
|
*/

'features' => [
    Features::termsAndPrivacyPolicy(),
    Features::profilePhotos(),
    Features::api(),
    Features::teams(['invitations' => true]),
    Features::accountDeletion(),
],
```

Figura 4: jetstream.php modificado

Ya tendríamos LARAVEL perfectamente configurado para trabajar con él.

3. Creación de Base de Datos

Antes de trabajar con la terminal de VISUAL STUDIO, es recomendable establecer en las variables de entorno del sistema los parámetros para que se detecte `mysql` como instrucción en la terminal. Deberíamos de tener algo como lo siguiente:

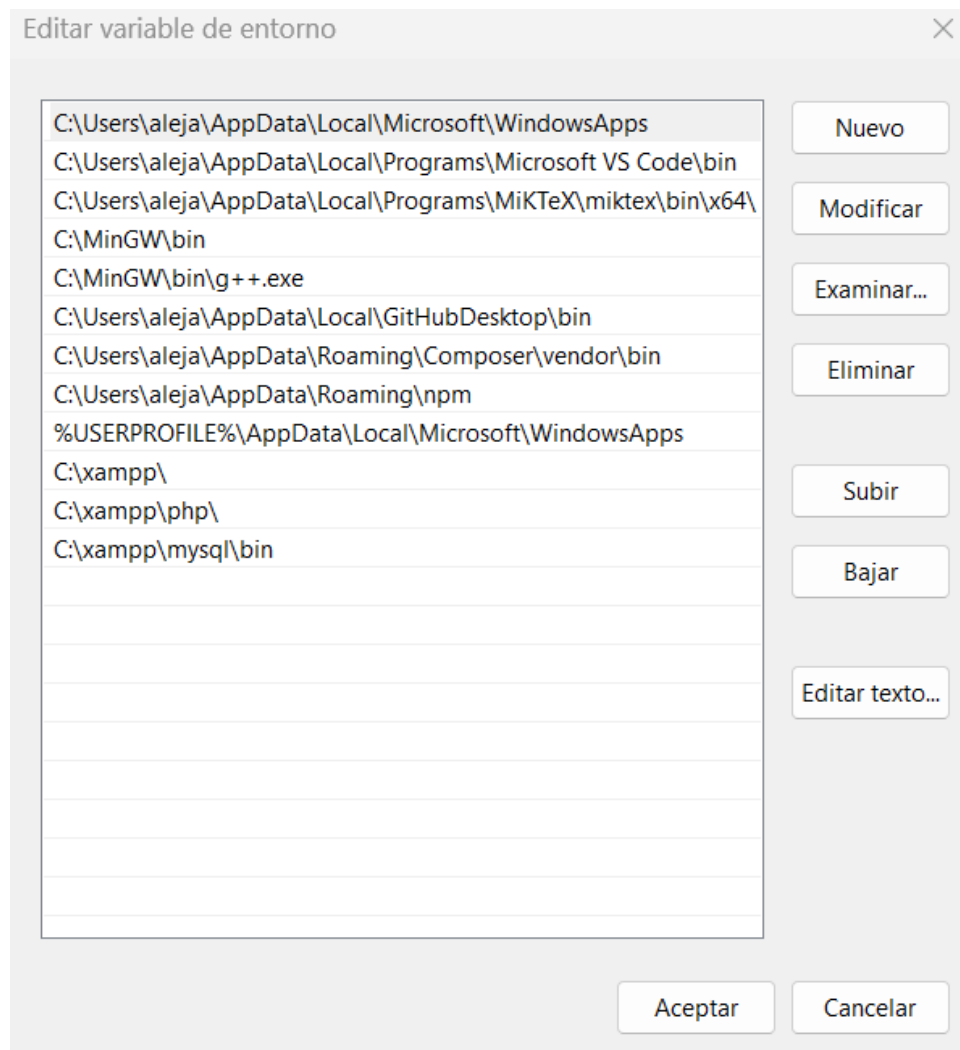
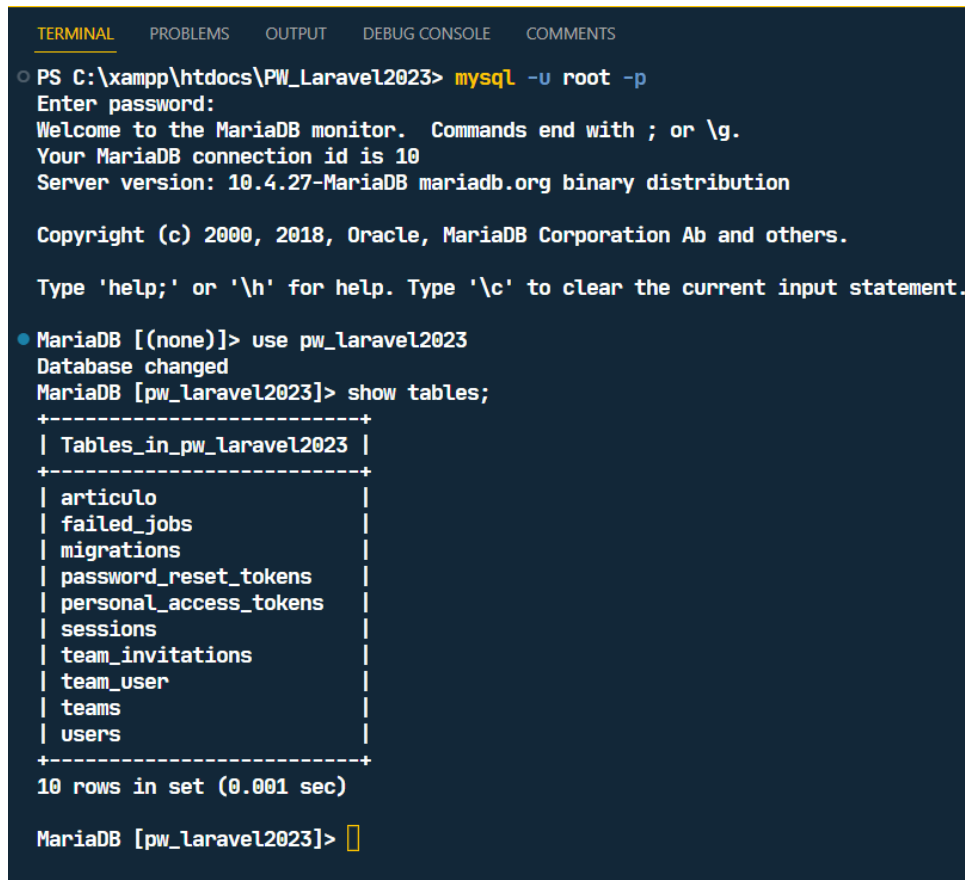


Figura 5: Editar variables de entorno

Seguidamente, en la terminal introducir los siguientes comandos:

```
$ mysql -u root -p
$ create database pw_laravel2023;
$ use pw_laravel2023;
$ show tables;
$ exit
```

La contraseña, en caso de que no funcione introducir una, dejar en blanco.



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  COMMENTS
PS C:\xampp\htdocs\PW_Laravel2023> mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 10
Server version: 10.4.27-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use pw_laravel2023
Database changed
MariaDB [pw_laravel2023]> show tables;
+-----+
| Tables_in_pw_laravel2023 |
+-----+
| articulo                 |
| failed_jobs              |
| migrations               |
| password_reset_tokens    |
| personal_access_tokens   |
| sessions                 |
| team_invitations         |
| team_user                |
| teams                    |
| users                    |
+-----+
10 rows in set (0.001 sec)

MariaDB [pw_laravel2023]> 
```

Figura 6: Creación de la base de datos

Con esto creamos una base de datos llamada `pw.laravel2023`. Posteriormente, en el fichero `.env` revisamos los resultados de su actualización, donde podremos introducir la contraseña de la base de datos –en caso de que nos funcione con contraseña; durante el desarrollo de los pasos no funcionaba, por lo que se ha optado por dejarlo en blanco–.

A continuación, introducimos el comando:

```
$ php artisan migrate
```

en la terminal de comandos, con lo que introduciremos las tablas por defecto de usuario y sesión a nuestra base de datos para poder usar `Laravel` como un proyecto normal e iniciar nuestra página.

4. Instalación de Bootstrap

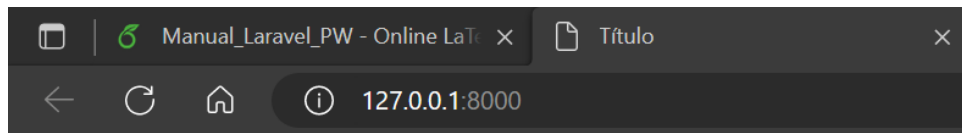
En la página principal de `BOOTSTRAP`, descargar la carpeta con su última versión y descomprimir las carpetas `CSS` y `JS` en la carpeta `public` del proyecto.

5. Lanzamiento de la aplicación

Con el comando:

```
$ php artisan serve
```

lanzamos el servidor:



Titulo con layouts Laravel 2023

Programación Web

Bienvenido Alejandro

Figura 7: Servidor en funcionamiento

En primeras instancias, debería de aparecer la página principal de LARAVEL, pero debido a que se han seguido los pasos del manual hasta el 7, las vistas se han modificado y revertir los pasos era tedioso, de ahí que se muestre esa instantánea.

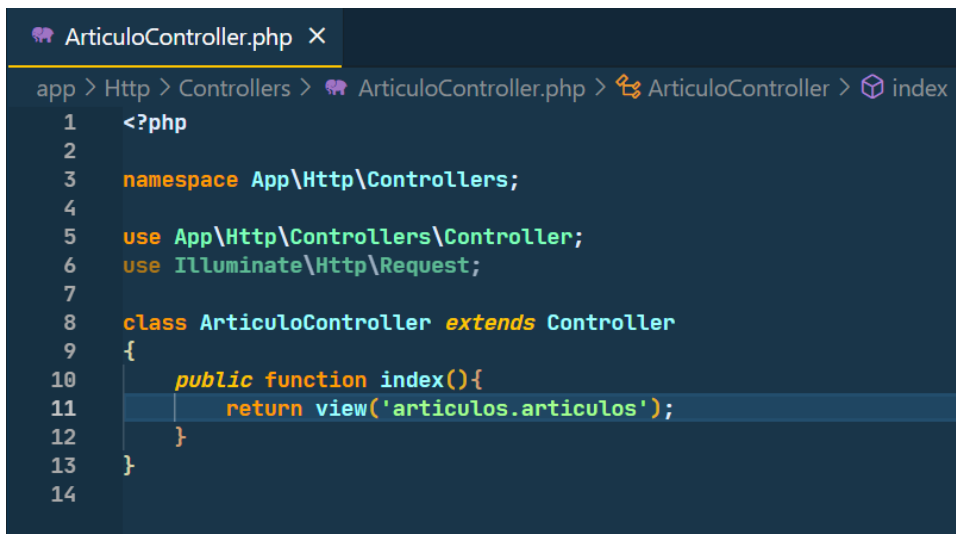
6. Creación 1º Prototipo de la web de los artículos

6.1. Creación Controlador de Artículos

Introducimos el comando:

```
$ php artisan make:controller ArticuloController
```

en la terminal, lo que nos creará un controlador vacío llamado `ArticuloController` en la ruta predefinida por LARAVEL. Creamos un método `index()` que devolverá la vista donde se van a mostrar todos los artículos. Esta vista será llamada `articulos.blade.php`, situada en su ruta por defecto.

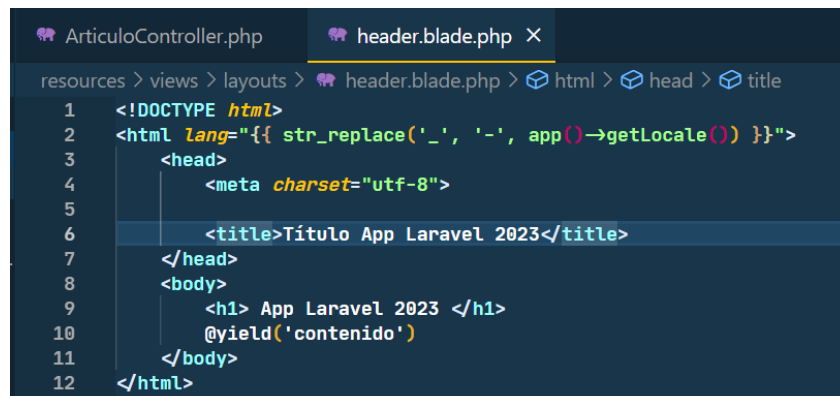


```
ArticuloController.php X
app > Http > Controllers > ArticuloController.php > ArticuloController > index
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Http\Controllers\Controller;
6  use Illuminate\Http\Request;
7
8  class ArticuloController extends Controller
9  {
10     public function index(){
11         return view('articulos.articulos');
12     }
13 }
14
```

Figura 8: Servidor en funcionamiento

6.2. Creación layout general de la Aplicación

Creamos un *layout* desde donde llamaremos a muchas de las vistas de nuestra aplicación llamado `header.blade.php` en el directorio de *layouts*, con el siguiente código incrustado:

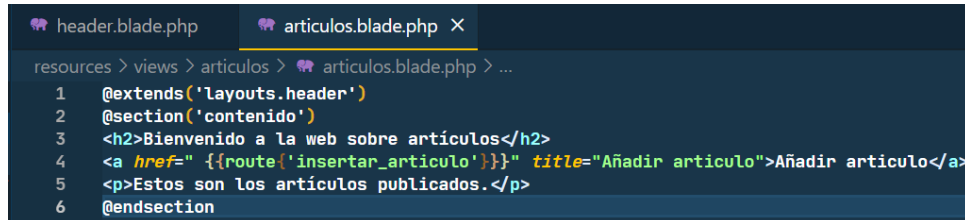


```
ArticuloController.php X header.blade.php X
resources > views > layouts > header.blade.php > html > head > title
1  <!DOCTYPE html>
2  <html lang="{{ str_replace('_', '-', app()>getLocale()) }}">
3  <head>
4      <meta charset="utf-8">
5
6      <title>Título App Laravel 2023</title>
7  </head>
8  <body>
9      <h1> App Laravel 2023 </h1>
10     @yield('contenido')
11 </body>
12 </html>
```

Figura 9: Layout `header.blade.php`

6.3. Creación vista `articulos.blade.php`

Creamos una vista ahora para los artículos, ídem como la vista anterior:



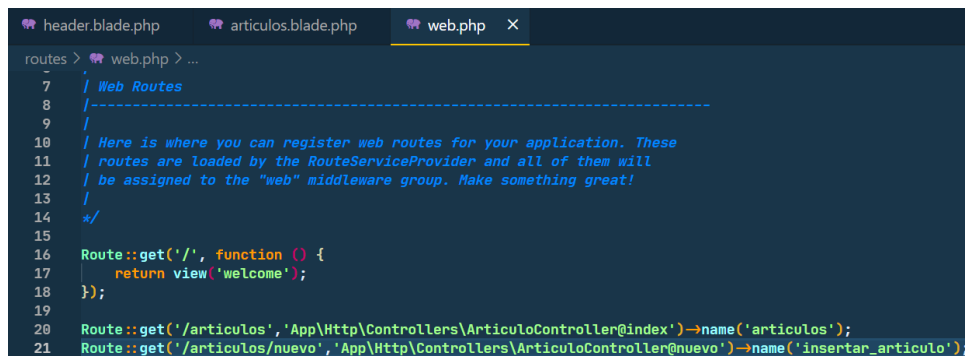
```
resources > views > articulos > articulos.blade.php > ...
1  @extends('layouts.header')
2  @section('contenido')
3  <h2>Bienvenido a la web sobre artículos</h2>
4  <a href="{{route('insertar_articulo')}}" title="Añadir artículo">Añadir artículo</a>
5  <p>Estos son los artículos publicados.</p>
6  @endsection
```

Figura 10: Layout `articulos.blade.php`

A diferencia del anterior *layout*, en este hemos introducido una llamada a la ruta cuyo nombre es `insertar_artículo` que no está creada todavía. Ahora registramos la ruta actual, es decir, la del controlador `articulos` con el método `index`.

6.4. Registro de las rutas creadas

Modificamos el fichero `routes/web.php`:



```
routes > web.php > ...
7  / Web Routes
8  /-----
9  /
10 / Here is where you can register web routes for your application. These
11 / routes are loaded by the RouteServiceProvider and all of them will
12 / be assigned to the "web" middleware group. Make something great!
13 /
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
20 Route::get('/articulos', 'App\Http\Controllers\ArticuloController@index')->name('articulos');
21 Route::get('/articulos/nuevo', 'App\Http\Controllers\ArticuloController@nuevo')->name('insertar_articulo');
```

Figura 11: Rutas en `web.php` modificadas

Si probamos la ruta `/articulos`, se nos mostrará la siguiente vista:

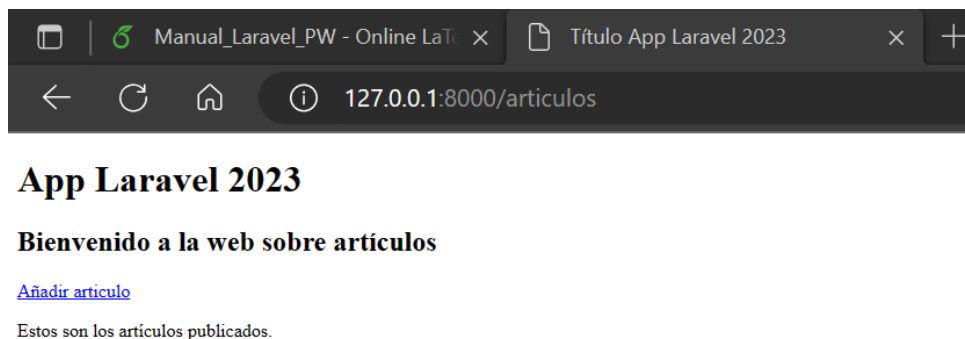


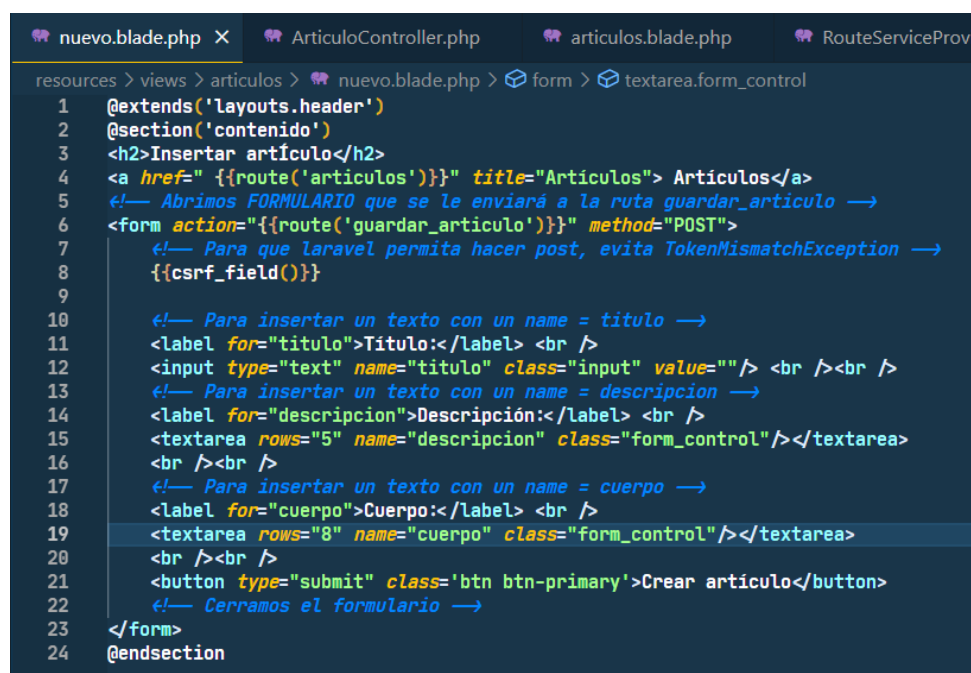
Figura 12: Vista `articulos`

7. Creación de formulario para introducir artículos

Creamos un método nuevo en el controlador `ArticuloController.php` que llamará a la vista `nuevo.blade.php`, que crearemos a continuación:

```
1 // ...
2 public function nuevo(){
3     return view('articulos.nuevo');
4 }
5 //...
```

Vista `nuevo.blade.php`:



```
nuevo.blade.php X  ArtículoController.php  articulos.blade.php  RouteServiceProvider

resources > views > articulos > nuevo.blade.php > form > textarea.form_control
1 @extends('layouts.header')
2 @section('contenido')
3 <h2>Insertar artículo</h2>
4 <a href="{{route('articulos')}}" title="Artículos"> Artículos</a>
5 <!-- Abrimos FORMULARIO que se le enviará a la ruta guardar_articulo -->
6 <form action="{{route('guardar_articulo')}}" method="POST">
7     <!-- Para que laravel permita hacer post, evita TokenMismatchException -->
8     {{csrf_field()}}
9
10    <!-- Para insertar un texto con un name = titulo -->
11    <label for="titulo">Titulo:</label> <br />
12    <input type="text" name="titulo" class="input" value="" /> <br /><br />
13    <!-- Para insertar un texto con un name = descripcion -->
14    <label for="descripcion">Descripción:</label> <br />
15    <textarea rows="5" name="descripcion" class="form_control" /></textarea>
16    <br /><br />
17    <!-- Para insertar un texto con un name = cuerpo -->
18    <label for="cuerpo">Cuerpo:</label> <br />
19    <textarea rows="8" name="cuerpo" class="form_control" /></textarea>
20    <br /><br />
21    <button type="submit" class="btn btn-primary">Crear artículo</button>
22    <!-- Cerramos el formulario -->
23 </form>
24 @endsection
```

Figura 13: `nuevo.blade.php`

También debemos de guardar la ruta de `guardar_articulo` puesto que no está reflejada. Esto es añadir la siguiente línea PHP en el fichero `web.php`:

```
1 Route::get('/articulos', 'ArticuloController@guardar')->name('guardar_articulo');
```

Ahora, podremos acceder a nuestro controlador del formulario por medio de la URL: `/articulos/nuevo`



App Laravel 2023

Insertar artículo

[Artículos](#)

Título:

Descripción:

Cuerpo:

Crear artículo

Figura 14: Vista nuevo artículo

Lo siguiente será crear el método guardar en el **controlador** de artículo. Sin embargo, primero crearemos un **request**, `NuevoArticuloRequest.php`, para validar el formulario.

7.1. Creación del request para validar el formulario

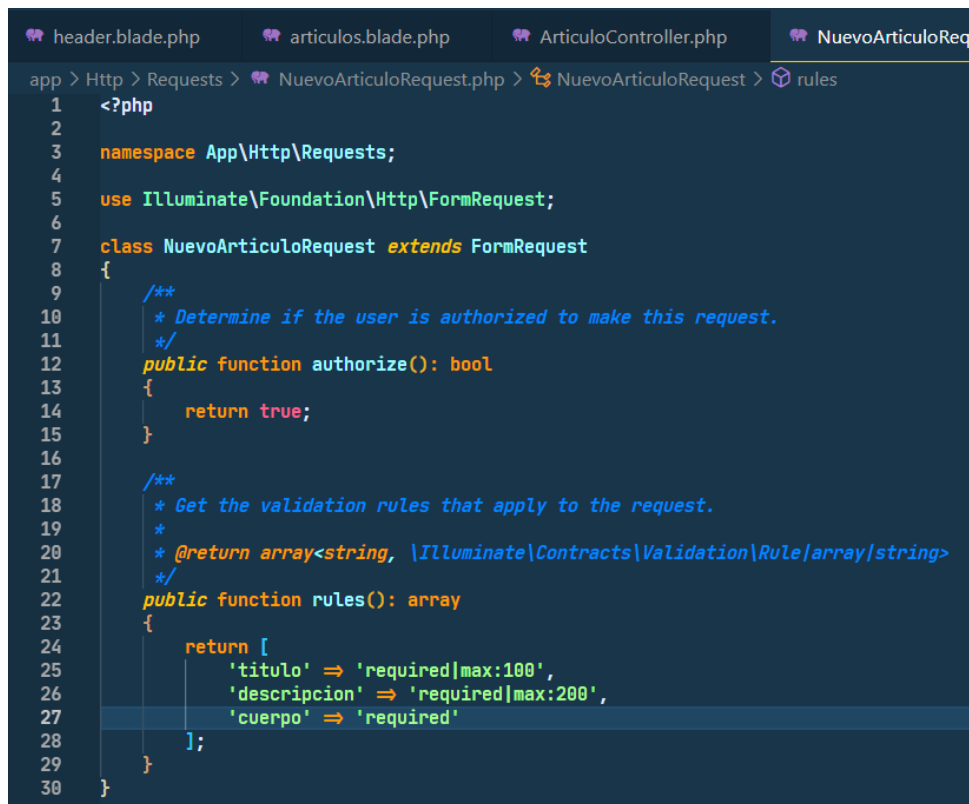
Para validar, nos crearemos un `form_request` de LARAVEL, que es una clase que extiende de un `request`, permitiéndose ponerle la lógica de la validación de formulario en una clase externa del controlador propio.

Para crearnos dicho `form_request`, escribimos en la terminal:

```
php artisan make:request NuevoArticuloRequest
```

Si accedemos a él, veremos que dispone de dos métodos:

- **authorize:** asegura que el usuario que hace el post esté autorizado. Lo dejaremos con valor `true`.
- **rules:** especifica las reglas de validación del formulario.



```
1  <?php
2
3  namespace App\Http\Requests;
4
5  use Illuminate\Foundation\Http\FormRequest;
6
7  class NuevoArticuloRequest extends FormRequest
8  {
9      /**
10       * Determine if the user is authorized to make this request.
11       */
12       public function authorize(): bool
13       {
14           return true;
15       }
16
17       /**
18       * Get the validation rules that apply to the request.
19       *
20       * @return array<string, \Illuminate\Contracts\Validation\Rule|array|string>
21       */
22       public function rules(): array
23       {
24           return [
25               'titulo' => 'required|max:100',
26               'descripcion' => 'required|max:200',
27               'cuerpo' => 'required'
28           ];
29       }
30  }
```

Figura 15: Request NuevoArticuloRequest.php

Añadimos en `ArticuloController.php` el uso de `NuevoArticuloRequest.php` para que tenga en cuenta las reglas de validación asignadas:

```
1 use App\Http\Requests\NuevoArticuloRequest;
```

Ahora, comenzaremos con la creación del **Modelo** puesto que no tenemos conectada la base de datos, así como ninguna referencia a la misma, pese a que tenemos la estructura principal de la aplicación desplegada.

7.2. Acceso a la base de datos

Creemos una tabla `articulos` en la base de datos. Tendrá los siguientes campos:

- **id** (int, auto incremental y clave primaria)
- **titulo** (varchar 100)
- **descripcion** (varchar 200)
- **cuerpo** (text)
- **created_at** (timestamp)
- **updated_at** (timestamp)

```
1 CREATE TABLE `articulos` (  
2   `id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,  
3   `titulo` VARCHAR( 100 ) NOT NULL ,  
4   `descripcion` VARCHAR( 200 ) NOT NULL ,  
5   `cuerpo` TEXT NOT NULL,  
6   `created_at` TIMESTAMP NOT NULL,  
7   `updated_at` TIMESTAMP NULL  
8 )
```

Nota: a la hora de crear esta tabla, el manual indicaba que el valor por defecto de `updated_at` había de ser `NOT NULL`, lo cual daba fallo. Se ha colocado dicho campo como `NULL`.

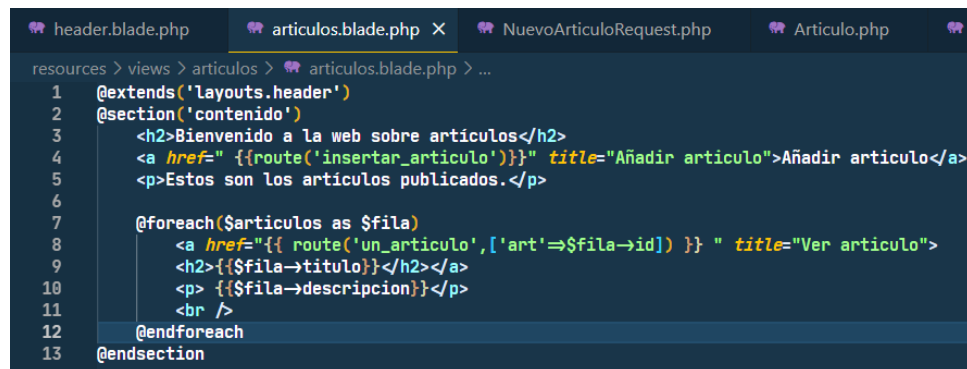
Modificamos un previo modelo `Articulo.php` añadiendo la siguiente línea y modificando el valor de la tabla previa, que era `articulo`. Esto será:

```
1 // ...  
2  
3 protected $table = 'articulos';  
4  
5 //Agregar datos  
6 protected $fillable = ['titulo','descripcion','cuerpo'];  
7 }
```

En el **controlador** `ArticuloController.php` modificamos el método `index()` para que muestre todos los artículos. No podemos olvidarnos de añadir la ruta del modelo.

```
1 use App\Models\Articulo;  
2  
3 // ...  
4  
5 class ArticuloController extends Controller  
6 {  
7     public function index(){  
8         $articulos = Articulo::all();  
9         return view('articulos.articulos')->with(['articulos' => $articulos]);  
10    }  
11  
12    // ...  
13  
14 }
```

En la **vista** `articulos.blade.php`, modificamos el mismo para mostrar el título y la descripción de los artículos que recibamos, y a través del título de un artículo, dirigirnos a una página web que muestre todos los datos de dicho artículo:



```
resources > views > articulos > articulos.blade.php > ...
1 @extends('layouts.header')
2 @section('contenido')
3     <h2>Bienvenido a la web sobre artículos</h2>
4     <a href="{{ route('insertar_articulo') }}" title="Añadir artículo">Añadir artículo</a>
5     <p>Estos son los artículos publicados.</p>
6
7     @foreach($articulos as $fila)
8         <a href="{{ route('un_articulo', ['art' => $fila->id]) }}" title="Ver artículo">
9         <h2>{{ $fila->titulo }}</h2></a>
10        <p> {{ $fila->descripcion }}</p>
11        <br />
12    @endforeach
13 @endsection
```

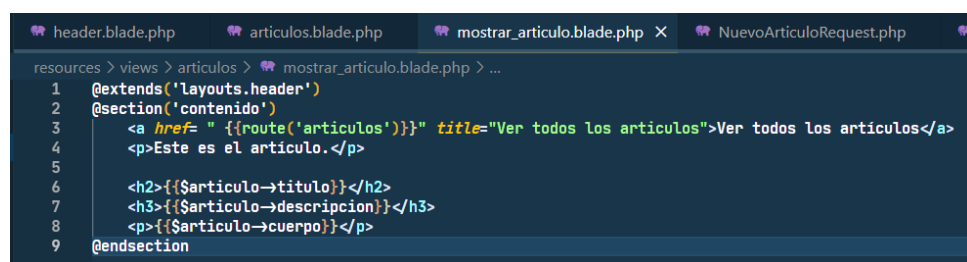
Figura 16: Modificación articulos.blade.php

Y creamos la ruta para un_articulo en routes/web.php:

```
Route::get('/articulos/{art}', 'ArticuloController@mostrar_articulo')->name('un_articulo');
```

En ArticuloController.php añadimos el método mostrar_articulo que devolverá la vista donde se va a mostrar un artículo, donde llamaremos una vista pendiente de creación, mostrar_articulo.blade.php

Vista mostrar_articulo.blade.php:



```
resources > views > articulos > mostrar_articulo.blade.php > ...
1 @extends('layouts.header')
2 @section('contenido')
3     <a href="{{ route('articulos') }}" title="Ver todos los artículos">Ver todos los artículos</a>
4     <p>Este es el artículo.</p>
5
6     <h2>{{ $articulo->titulo }}</h2>
7     <h3>{{ $articulo->descripcion }}</h3>
8     <p>{{ $articulo->cuerpo }}</p>
9 @endsection
```

Figura 17: Vista mostrar_articulos.blade.php

Ahora podremos acceder a un solo artículo por medio de una URL como la siguiente:

`http://localhost:8000/articulos/2`

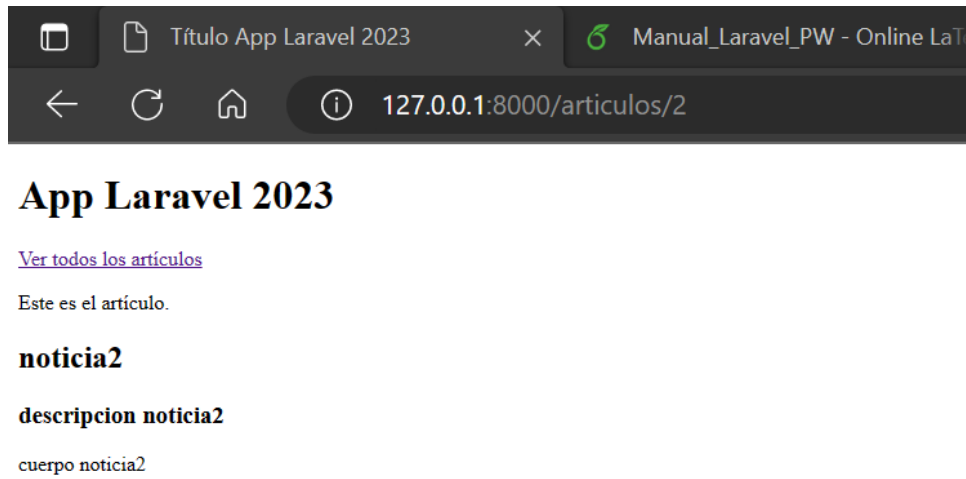


Figura 18: Vista articulo 2

Y si queremos ver un listado de todos los artículos podremos acceder mediante la URL de `/articulos`



Figura 19: Vista todos los articulos

7.3. Insertar artículos en la base de datos desde el formulario

Ahora que tenemos el formulario de los artículos, vamos a crear un artículo con los datos que se introducen en el formulario. Para ello vamos a modificar el método `guardar` del **controlador** `ArticuloController.php`

```
1 // ...
2
3 public function guardar(NuevoArticuloRequest $request){
4     $art = Articulo::create($request->only('titulo','descripcion','cuerpo'));
5     return redirect()->route('un_articulo',['art'=>$art->id]);
6 }
7
8 // ...
```

Simplemente añadimos el artículo llamando al método `create`, y después hacemos un redireccionamiento a la ruta llamada `un_articulo` a la que le pasamos como parámetro `art`.

7.4. Paginación de los artículos

A continuación, realizaremos una paginación de los artículos de nuestra aplicación.

Modificamos el controlador `ArticuloController.php`, concretamente el método `index()` para que muestre los artículos en orden descendiente por su publicación y salgan paginados, con un máximo de dos artículos en cada página.

```
1 // ...
2
3 public function index(){
4     $articulos = Articulo::orderBy('id', 'desc')->paginate(2);
5     return view('articulos.articulos')->with(['articulos' => $articulos]);
6 }
7
8 // ...
```

Y en `articulos.blade.php` añadimos el método `render` para ver en qué páginas nos ubicamos

```
1 // ...
2
3 {{ $articulos->render()$ }}
4
5 @endsection
```



Figura 20: Artículos paginados descendientemente

8. Login y Registro Usuario

La herramienta JETSTREAM nos proporciona tablas, vistas, modelos, etc. por defecto. Por ello, al inicio del proyecto migramos estas tablas. El *middleware* proporcionado contiene una ruta en `routes/web.php`.

Veamos el login y registro por defectos:

/login:

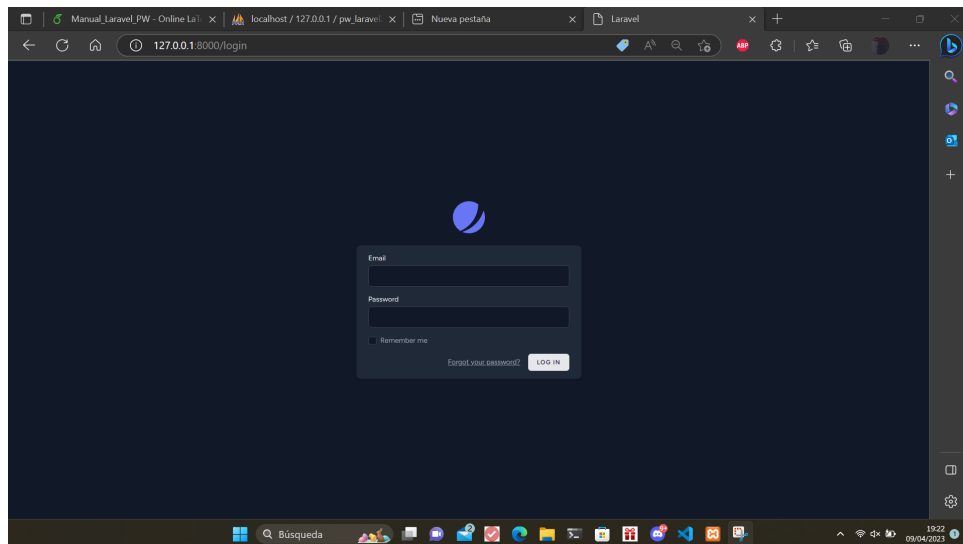


Figura 21: Vista web login

/register:

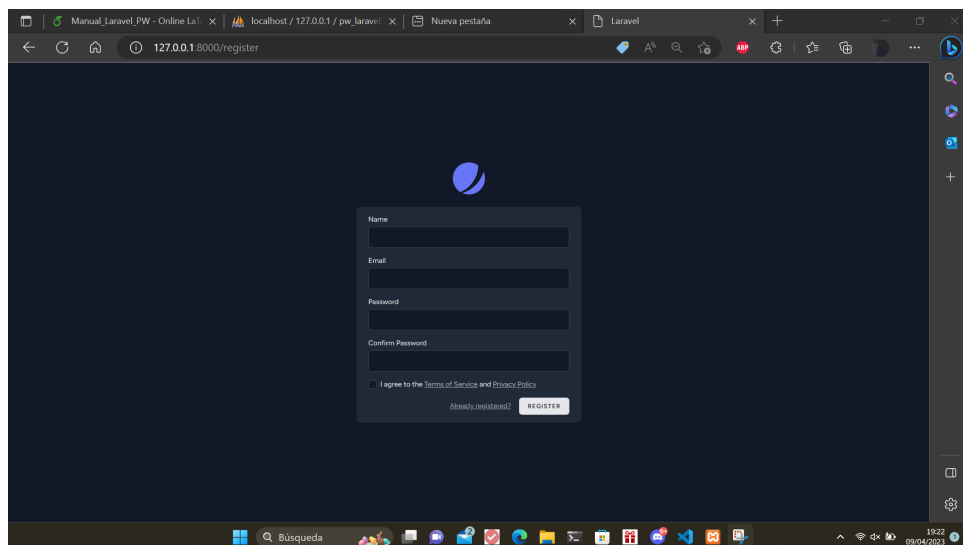


Figura 22: Vista web register

Si creamos un usuario y entramos, por defecto nos saldrá la vista `dashboard` que viene un ejemplo de todo lo que podemos ver y hacer con esta herramienta.

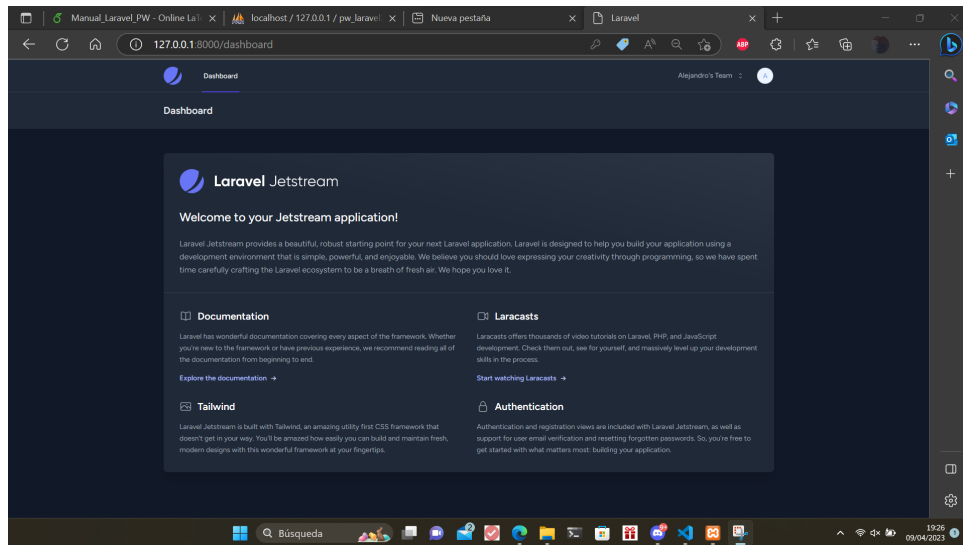


Figura 23: Vista web dashboard

8.1. Integración de los formularios en la web

Para la integración de los formularios de registro y *login* en la web, previamente hacemos una copia de seguridad de los archivos `resources/views/auth/login.php` y `resources/views/auth/register.php`.

A continuación, modificaremos estos archivos aplicando los siguientes cambios:

1. Al comienzo, heredar de nuestro *layout*.
2. Justo debajo, crear la sección de contenido
3. Posteriormente, eliminar la etiqueta de apertura y cierre `<x-guest-layout>`
4. Finalmente, eliminar la etiqueta `<x-jet-authentication-card-logo/>`
5. Hacer lo mismo con la vista *login*.
6. Añadir al final de la vista *login*, un botón de registrarse que enlace con la página *register* –usar la etiqueta `<a>` y href `"{{route('register')}}"`–.
7. En el archivo `RouteServiceProvider` contenido en la carpeta `app/Providers` cambiamos la variable `Home` que será la que una vez logueada o registrada nos llevará a tal ruta.

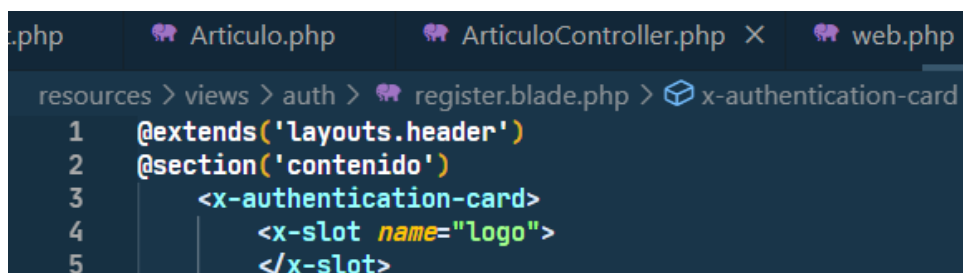
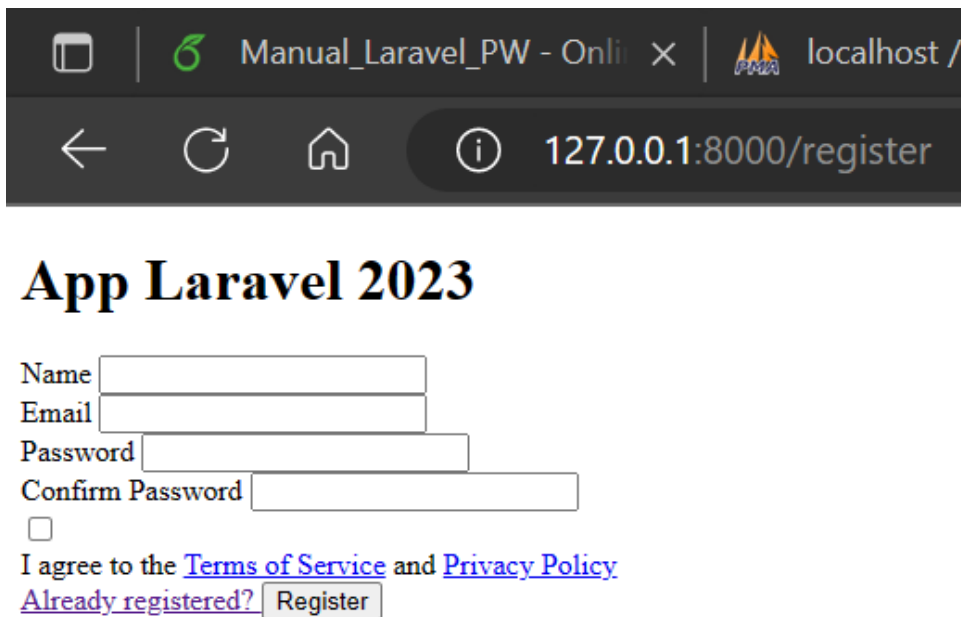


Figura 24: register modificado

```
        </x-button>
      </div>
    </form>
  </x-authentication-card>
@endsection
```

Figura 25: register modificado

De tal forma que el la vista register se vería así:



The screenshot shows a web browser window with the title "Manual_Laravel_PW - Onli" and the address bar displaying "localhost / 127.0.0.1:8000/register". The page content includes the heading "App Laravel 2023" and a registration form with the following elements:

- Name:
- Email:
- Password:
- Confirm Password:
- ☐ checkbox
- I agree to the [Terms of Service](#) and [Privacy Policy](#)
- [Already registered?](#)
-

Figura 26: Vista register modificada

ídem para login:

```

t.php | Artículo.php | ArtículoController.php | web.php
resources > views > auth > login.blade.php > x-authentication-card
1  @extends('layouts.header')
2  @section('contenido')
3      <x-authentication-card>
4          <x-slot name="logo">
5              </x-slot>
6
7          <x-validation-errors class="mb-4" />

```

Figura 27: login modificado

```

<x-button class="ml-4">
    {{ __('Log in') }}
</x-button>
<a href="{{ route('register') }}" class="btn btn-primary">Regístrate</a>
</div>
</form>
</x-authentication-card>
@endsection

```

Figura 28: login modificado

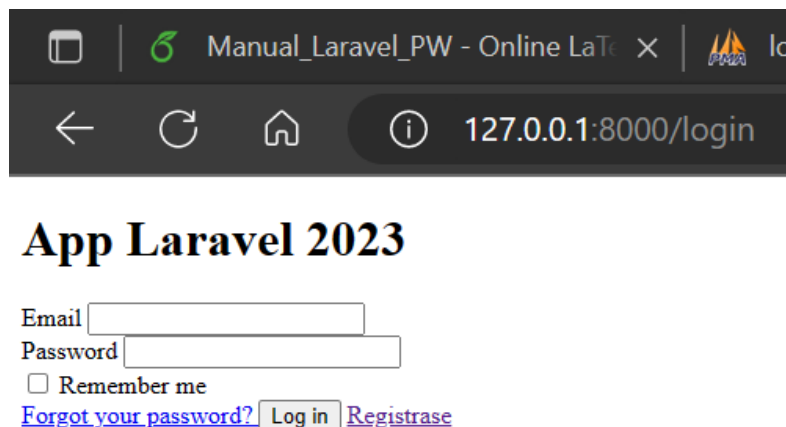


Figura 29: Vista login modificada

```
1 public const HOME = '/articulos';
```

Y en `articulos.blade.php` deberemos añadir el siguiente condicional para que nos muestre el usuario que se ha logueado:

```
1 // ...
2
```

```

3      @if (Auth::check())
4          <button type="button" class="btn btn-primary">{{Auth::user()->name}}</
            button>
5          <a href="/user" title="Usuario">Usuario</a>
6      @endif
7
8      // ...

```

Si nos logueamos desde login, nos redireccionará a `articulos` y veremos nuestro nombre de usuario en el botón agregado.



Figura 30: Vista `articulos` con botón usuario

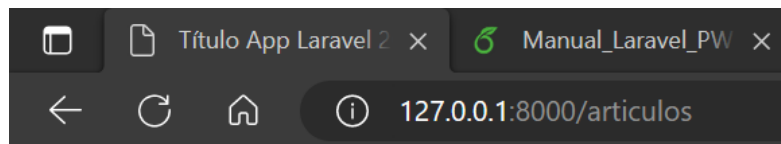
Por último, crearemos un botón para cerrar sesión añadiendo el siguiente código a nuestro `articulos.blade.php`:

```

1      @if (Auth::check())
2          <button type="button" class="btn btn-primary">{{Auth::user()->name}}</
            button>
3          <form method="POST" action="{{route('logout') }}">
4              @csrf
5              <a href="{{ route('logout') }}"
6                  onclick="event.preventDefault();

```

```
7         this.closest('form').submit();">
8         
9     </a>
10 </form>
11 @endif
```



App Laravel 2023

Bienvenido a la web sobre artículos

[Añadir artículo](#)

Estos son los artículos publicados.



noticia3

descripcion noticia3

noticia2

descripcion noticia2

« Previous [Next](#) »

Showing 1 to 2 of 3 results

Figura 31: Vista final artículos con imagen

9. Conclusiones

Hay varios puntos que destacar del manual de LARAVEL disponible en el campus virtual de la asignatura:

- Pese a que el manual indica que los pasos 1 al 7 son necesarios de hacer para completar el tutorial, bien es cierto que durante el desarrollo del mismo se da a entender por las imágenes y por la misma naturaleza de qué va desarrollándose en cada apartado que **obvia** los pasos realizados del 1 al 7, como ocurre con el controlador `ArticuloController.php` el cual es alterado completamente, lo cual es extraño por sí solo.
- Hay varias imágenes que además de estar colocadas en un orden dudable o confuso, están **desactualizadas** o contienen información falsa y que no refleja exactamente lo que se pide. Son esos fallos minúsculos que han dificultado mucho el desarrollo de este manual, como en una sección donde se hace un `.blade` que extiende a un `layout 'layouts.app'`, lo cual es un grave error pues en ningún momento usamos ese `layout` provisto por LARAVEL.

No obstante, y a diferencia que ocurrió durante el seguimiento de la documentación de CODEIGNITER, los errores encontrados son triviales y no se desencadenan en problemas mayores de dependencias, problemas con la base de datos, etcétera. En cambio, los fallos descubiertos son fáciles de arreglar, pero difíciles de encontrar dado que se basan en etiquetas incorrectas, un paso dado por hecho en el tutorial cuando en ningún momento se hace referencia al mismo...