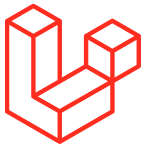
Introduction to Laravel

DWES

1.- Introduction



Laravel

Laravel is an open-source framework for developing web applications in PHP 5 or higher, created in 2011 by Taylor Otwell, inspired by Ruby on Rails and Symfony, from which it has adopted their main advantages.

Laravel makes development easier by simplifying work with common tasks such as authentication, routing, session management, caching, etc. Some of the main features and advantages of Laravel are:

- It is designed to develop under the MVC pattern
- Integrates an ORM (Object-Relational Mapping) system called Eloquent
- Uses a view template system called Blade
- Incorporates a command line interpreter called Artisan

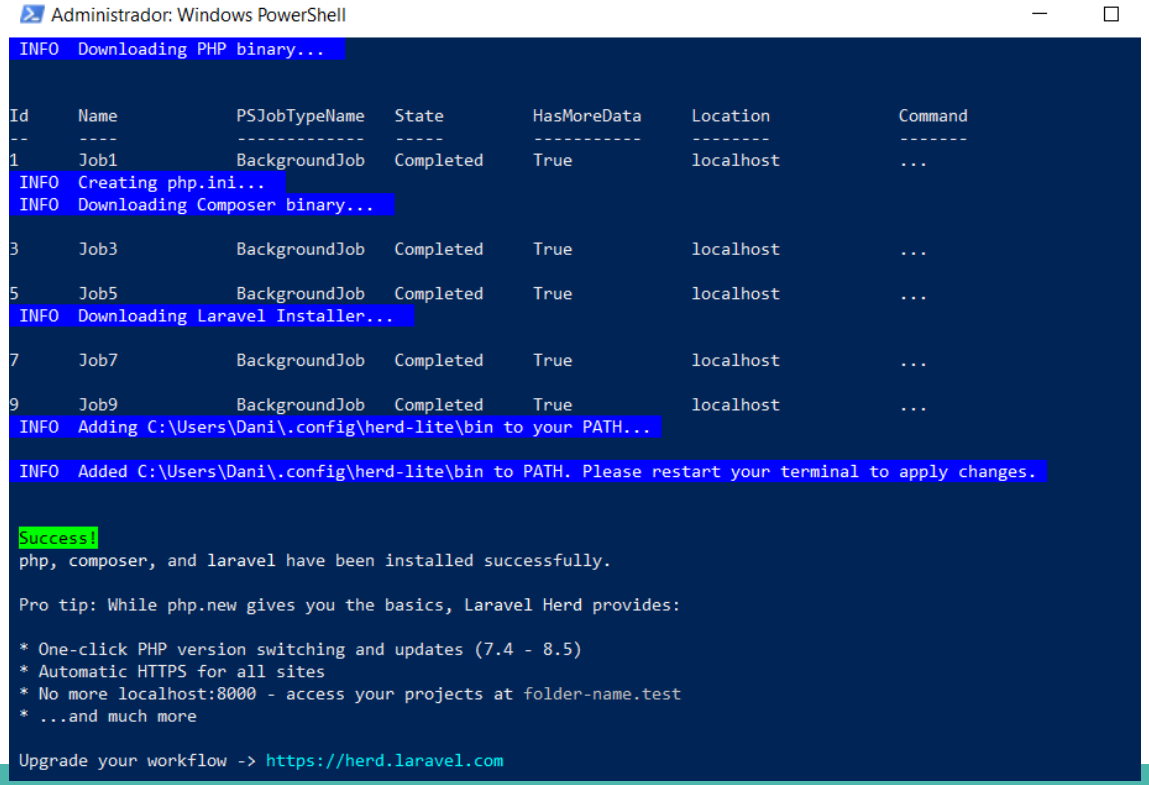
1.- Introduction

More interesting information:

- The most recent version is 12.x
- The documentation is available at https://laravel.com/docs/12.x
- Laravel 12.x requires a minimum PHP version of 8.2
- Laravel requires Composer to be installed
- To run npm, install Node.js https://nodejs.org/es/download
- A free Laravel learning platform is https://laracasts.com

2.- Installation and environment preparation

Install everything as indicated in https://laravel.com/docs/12.x#installing-php



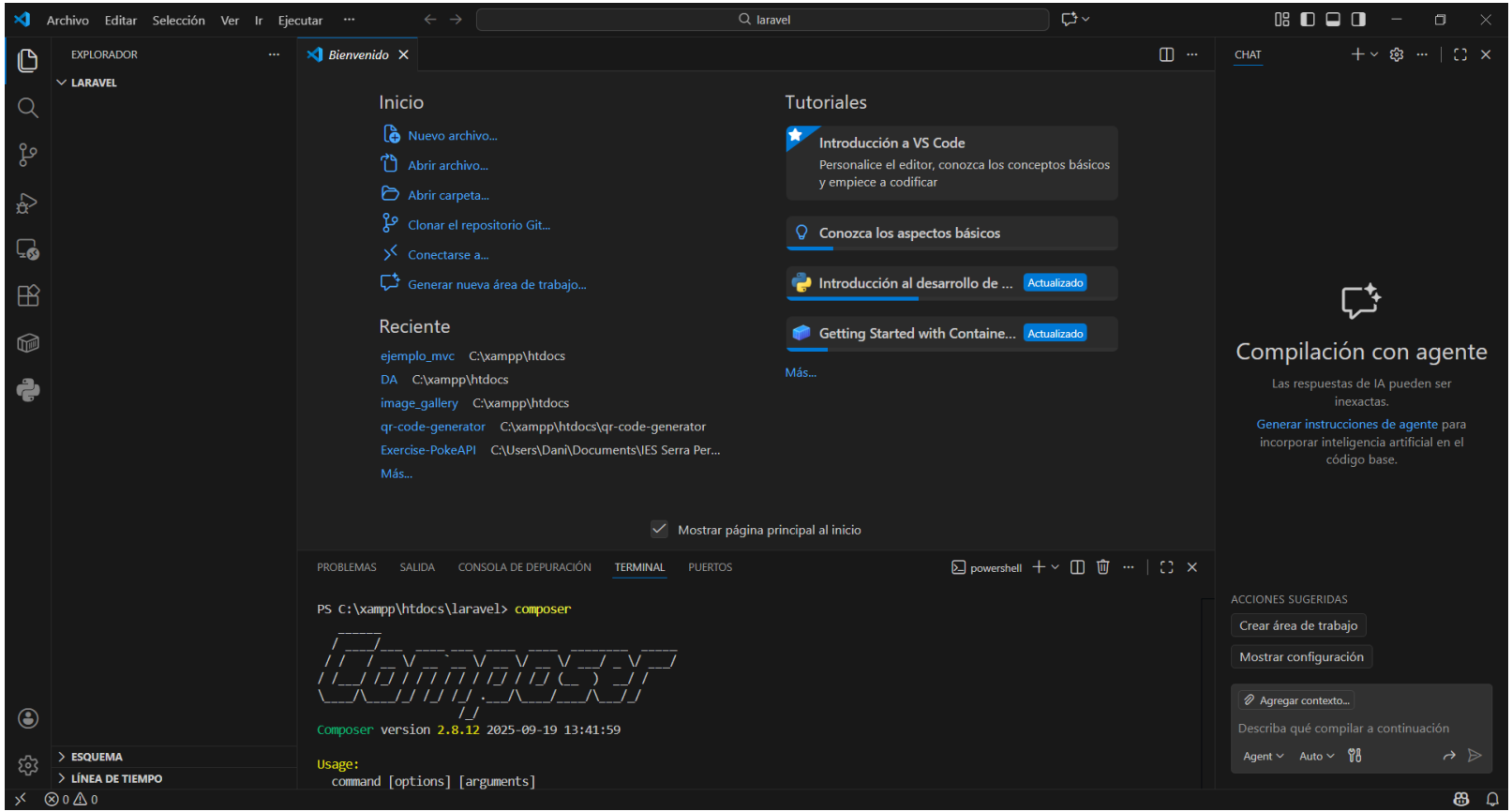
2.- Installation and environment preparation

Check that the installation has been correctly done:

```
PS C:\Windows\system32> php -v
PHP 8.4.0 (cli) (built: Nov 21 2024 08:18:43) (NTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.4.0, Copyright (c) Zend Technologies
    with Zend OPcache v8.4.0, Copyright (c), by Zend Technologies
PS C:\Windows\system32>
```

```
PS C:\Windows\system32> composer
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 2.8.12 2025-09-19 13:41:59
```

2.- Installation and environment preparation



2.- Installation and environment preparation

We can create a project executing:

laravel new example-app

```
PS C:\xampp\htdocs\laravel> laravel new example-app

  _                               _
 | |                             | |
 | |     __ _ _ __ __ ___   _____| |
 | |    / _` | '__/ _` \ \ / / _ \ |
 | |___| (_| | | | (_| |\ v /  __/ |
 |______\__,_|_|  \__,_| \_/ \___|_|

 Which starter kit would you like to install? [None]:
  [none     ] None
  [react    ] React
  [vue      ] Vue
  [livewire] Livewire
 >


 Which testing framework do you prefer? [Pest]:
  [0] Pest
  [1] PHPUnit
 >


 Do you want to install Laravel Boost to improve AI assisted coding? (yes/no) [yes]:
 >
```

```
 Which database will your application use? [SQLite]:
  [sqlite ] SQLite
  [mysql  ] MySQL
  [mariadb] MariaDB
  [pgsql  ] PostgreSQL (Missing PDO extension)
  [sqlsrv ] SQL Server (Missing PDO extension)
 >
```

2.- Installation and environment preparation

If there is an error executing “npm install”

```
PS C:\xampp\htdocs\laravel\example-app> npm install
npm : No se puede cargar el archivo C:\Program Files\nodejs\npm.ps1 porque la ejecución de scripts está deshabilitada en
este sistema. Para obtener más información, consulta el tema about_Execution_Policies en
https:/go.microsoft.com/fwlink/?LinkID=135170.
En línea: 1 Carácter: 1
+ npm install
+ ~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

Launch ‘Powershell’ as administrator and execute:

Set-ExecutionPolicy RemoteSigned

```
PS C:\Windows\system32>
>> Set-ExecutionPolicy RemoteSigned

Cambio de directiva de ejecución
La directiva de ejecución te ayuda a protegerte de scripts en los que no confías. Si cambias dicha directiva, podrías
exponerte a los riesgos de seguridad descritos en el tema de la Ayuda about_Execution_Policies en
https:/go.microsoft.com/fwlink/?LinkID=135170. ¿Quieres cambiar la directiva de ejecución?
[S] Sí  [O] Sí a todo  [N] No  [T] No a todo  [U] Suspender  [?] Ayuda (el valor predeterminado es "N"): s
PS C:\Windows\system32>
```

2.- Installation and environment preparation

Now “npm install” should work:

```
PS C:\xampp\htdocs\laravel\example-app> npm install

added 83 packages, and audited 84 packages in 12s

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 11.6.2 -> 11.7.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.7.0
npm notice To update run: npm install -g npm@11.7.0
npm notice
```

```
PS C:\xampp\htdocs\laravel\example-app> npm run build

> build
> vite build

vite v7.3.0 building client environment for production...
✓ 53 modules transformed.
public/build/manifest.json              0.33 kB │ gzip:  0.17 kB
public/build/assets/app-1YpEKTRp.css   33.83 kB │ gzip:  8.48 kB
public/build/assets/app-CAiCLEjY.js    36.35 kB │ gzip: 14.71 kB
✓ built in 800ms
```

2.- Installation and environment preparation

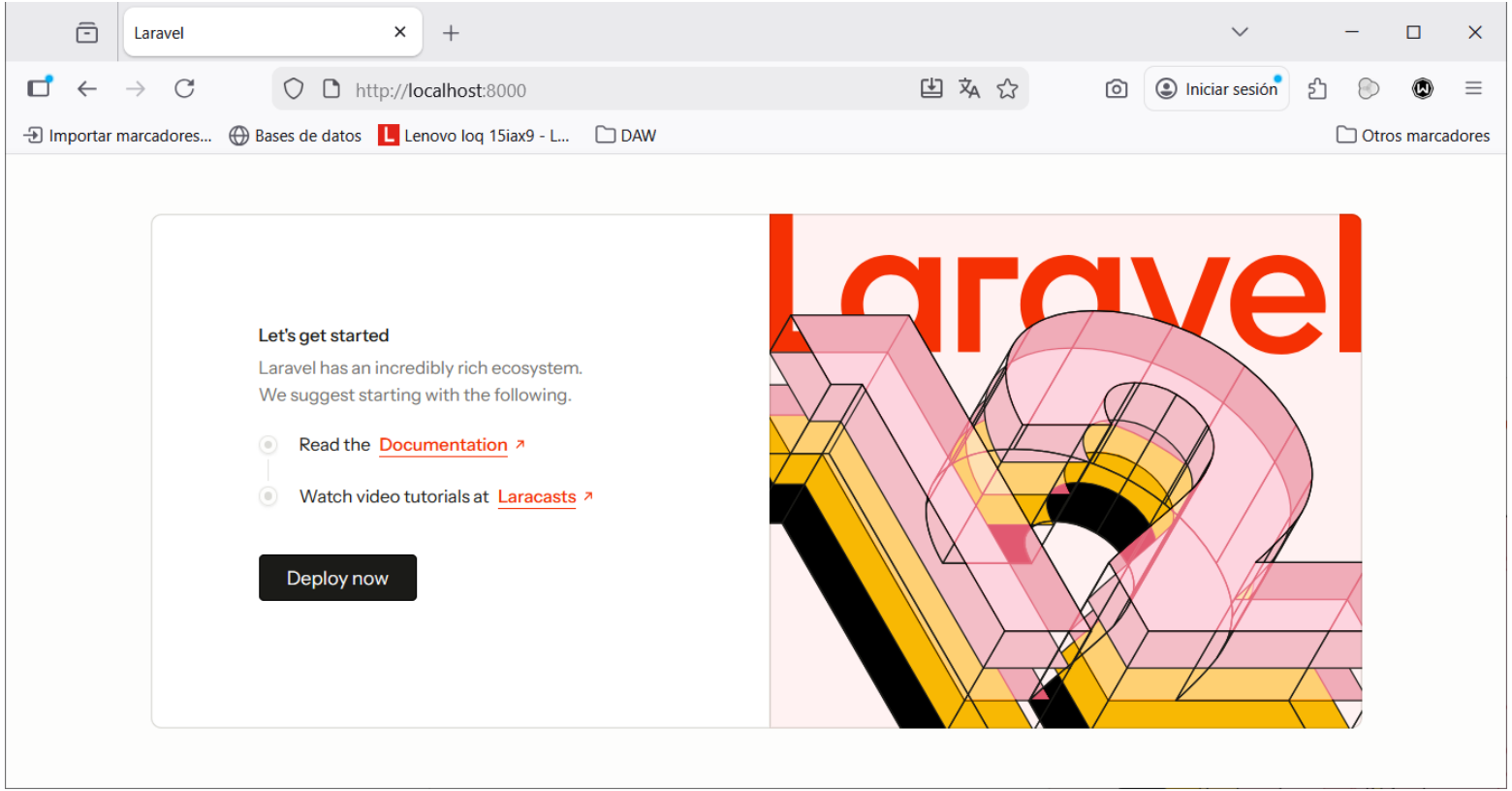
Run the application executing “composer run dev”:

```
PS C:\xampp\htdocs\laravel\example-app> composer run dev
> Composer\Config::disableProcessTimeout
> npx concurrently -c "#93c5fd,#c4b5fd,#fdba74" "php artisan serve" "php artisan queue:listen --tries=1" "npm run dev" --name
s='server,queue,vite'
[vite]
[vite] > dev
[vite] > vite
[vite]
[vite]
[vite]   VITE v7.3.0  ready in 355 ms
[vite]
[vite]   ➜  Local:   http://localhost:5173/
[vite]   ➜  Network: use --host to expose
[vite]
[vite]   LARAVEL v12.44.0  plugin v2.0.1
[vite]
[vite]   ➜  APP_URL: http://localhost:8000
```

Or “php artisan serve”:

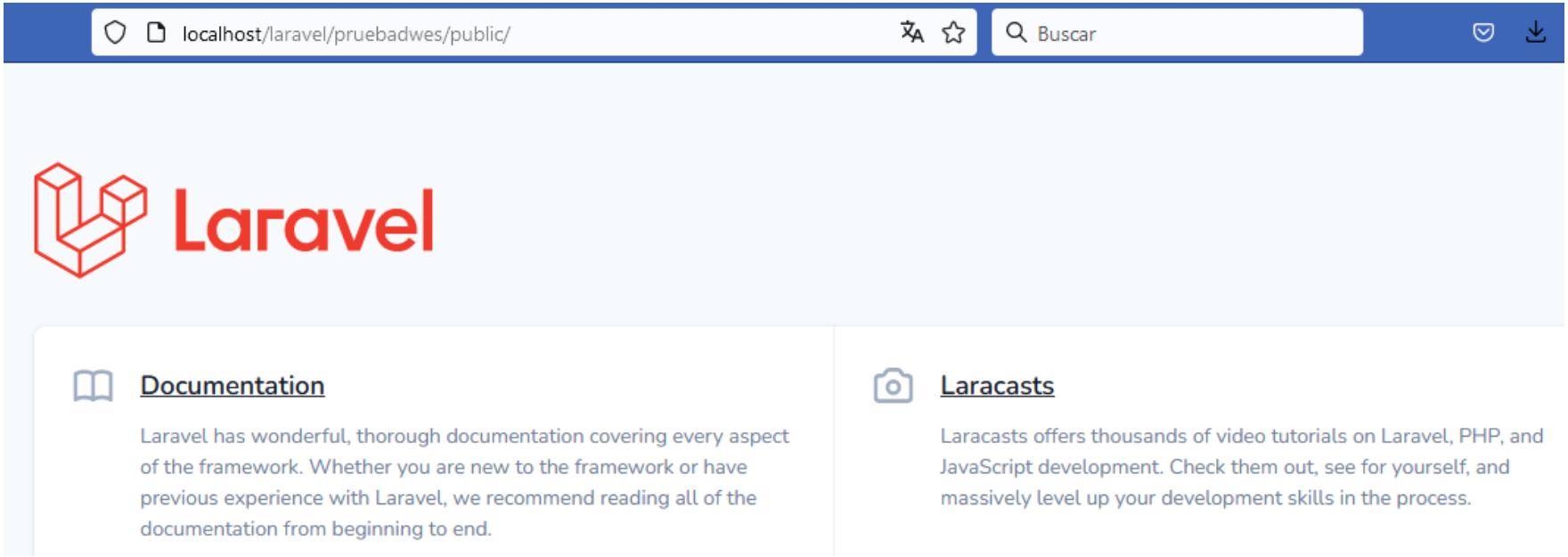
```
PS C:\xampp\htdocs\laravel\example-app> php artisan serve

  INFO  Server running on [http://127.0.0.1:8000].

  Press Ctrl+C to stop the server
```

2.- Installation and environment preparation



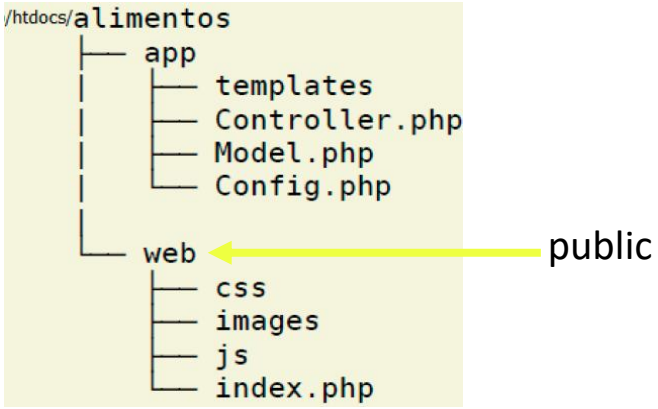
2.- Installation and environment preparation

If we use XAMPP we can have PHP version errors. When solved, we need to browse to public folder:



2.- Installation and environment preparation

Remember the previous unit and the separation between public and private elements of the web application:



For convenience, we will work by creating the project in the root directory of the web server, so it will be necessary to include public in the URL, but it is not recommended. To eliminate this part of the URL, different techniques can be used https://fuubar.wordpress.com/2013/06/04/remover-segmento-public-de-la-url-en-laravel/

3.- Configuration

All of the configuration files for Laravel are stored in the config directory.

Laravel needs almost no additional configuration out of the box. You are free to get started developing! However, you may review the config/app.php file.

The configuration options can be changed from each of the files, but thanks to the DotEnv system, the options can be included in the .env file in the root directory of the Laravel project.

You can see that the following function appears in the configuration files when loading the values: 'name' => env('VariableName', 'DefaultValue').

This function looks for the variable in the .env file, if it exists it takes its value and otherwise it takes the value passed by default in the function.

4.- Directory structure

The default Laravel application structure is intended to provide a great starting point for both large and small applications. But you are free to organize your application however you like. Laravel imposes almost no restrictions on where any given class is located - as long as Composer can autoload the class.

We are going to comment on some of the most important points of the Laravel directory structure.

You can consult the structure in more detail at:
https://laravel.com/docs/12.x/structure

4.- Directory structure

- **App**: Contains the main code of the application. This folder is in turn divided into many subfolders that we will analyze later

- **Bootstrap**: This folder includes the code that processes each of the calls to our project. Normally we will not have to modify anything in this folder

- **Config**: Here are all the application configuration files

- **Database**: This folder includes everything related to the definition of our project's database. Inside it we can find three folders: factories, migrations and seeders

4.- Directory structure

- **Public**: It is the only public folder, the only one that should be visible on the web server. All requests go through this folder, since index.php is located there, this file is the one that starts the entire framework execution process. This directory also contains CSS, Javascript, images and other files that you want to make public.

- **Resources**: This folder contains three folders
  - Js: contains Javascript files
  - Css: contains css files
  - Views: contains the views of our application. In general, they will be HTML templates that the controllers use to display the information.

- **Routes**: This folder contains all the application routing files

4.- Directory structure

- **Storage**: In this folder Laravel stores all the internal information necessary for the execution of the website, such as the session files, the cache, the compilation of the views, meta information and system logs. Normally we will not have to touch anything inside this folder, it is only usually accessed to consult the logs

- **Tests**: This folder is used for files with automated tests. Laravel includes a system that facilitates the entire testing process with PHPUnit

- **Vendor**: All the libraries and dependencies that make up the Laravel framework are stored in this folder. We will not have to modify this folder either, since all the code it contains are libraries that are installed and updated using the Composer tool

4.- Directory structure

- **.env**: This file is used to store configuration values that are specific to the current machine or installation. Which allows us to easily change the configuration depending on the machine on which it is installed and have different options for production, for different developers, etc. Important, this file should not be included in version control, it should be in the .gitignore when using a git version control tool

- **Composer.json**: This file is the one used by Composer to install Laravel. In an initial installation, only the installation of one package will be specified, the Laravel framework itself, but we can specify the installation of other libraries or external packages that add functionality to Laravel

4.- Directory structure – App folder

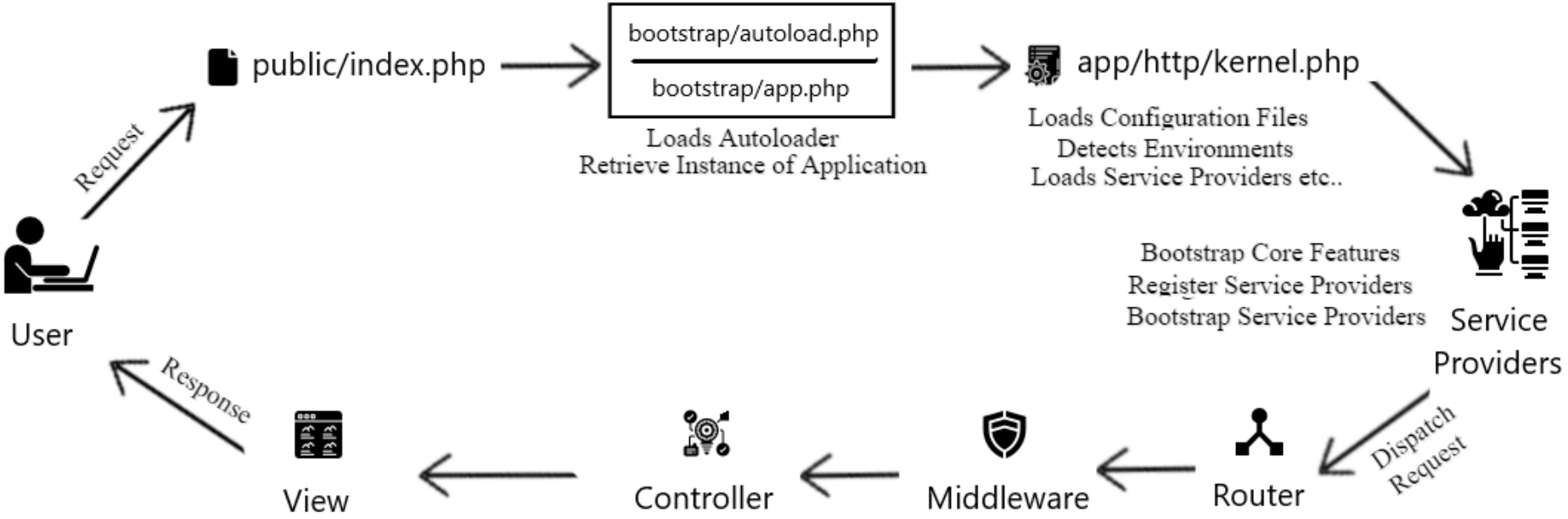
The app folder contains the main code of the project, such as routes, controllers, filters, and data models. Inside there are other directories, the most important one is Http:

- **Http/Controllers:** Contains all the files with the controller classes that are used to interact with the models, views and manage the application logic

- **Http/Middleware:** These are the filters or intermediate classes that we can use to perform certain actions, such as validating permissions, before or after executing a request to a route in our web project

- **Models:** This directory contains all your Eloquent model classes. The Eloquent ORM included with Laravel helps work with your database

5.- Request lifecycle



6.- Routing

As mentioned above, the documentation is available here

The routes are defined including the following:

- The URL of the request
- The method used (mainly GET, POST, PUT and DELETE)
- The function, view or controller to execute

The route:list Artisan command can easily provide an overview of all of the routes that are defined by your application:

php artisan route:list

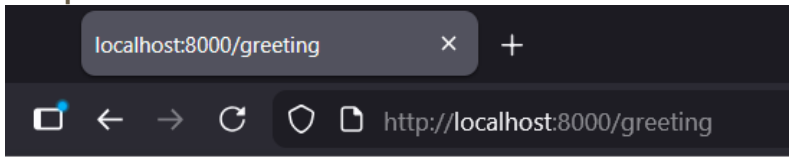
```
PS C:\xampp\htdocs\laravel\example-app> php artisan route:list

  GET|HEAD   / ..........................................................................
  POST       _boost/browser-logs ..... boost.browser-logs
  GET|HEAD   storage/{path} ..... storage.local
  GET|HEAD   up ..........................................................................

                                                                    Showing [4] routes
```

6.- Routing

The following route:

```php
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

Will generate this response:

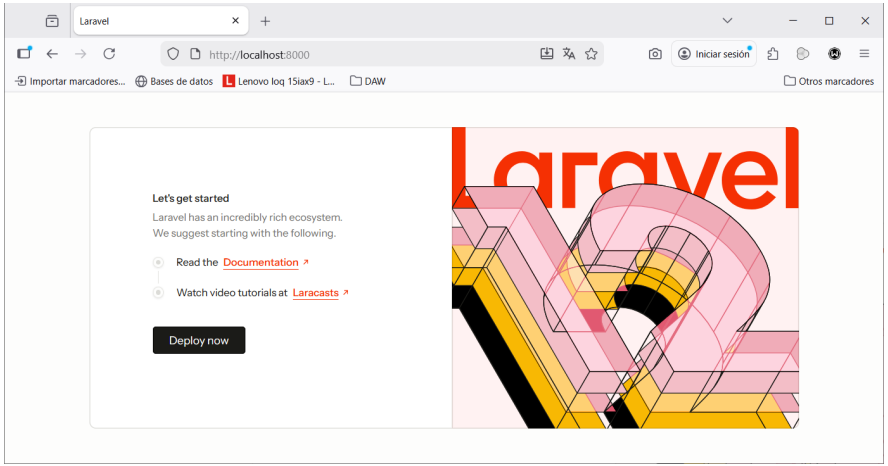


6.- Routing

The following route:

```php
Route::get('/', function () {
    return view('welcome');
});
```

Will show the “welcome” view (resources/views/welcome.blade.php):



6.- Routing

Sometimes you may need to register a route that responds to multiple HTTP verbs. You may do so using the match method. Or, you may even register a route that responds to all HTTP verbs using the any method:

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

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

6.- Routing

Sometimes you will need to capture segments of the URI within your route. For example, you may need to capture a user's ID from the URL. You may do so by defining route parameters:

```php
Route::get('/user/{id}', function (string $id) {
    return 'User '.$id;
});
```

You may define as many route parameters as required by your route:

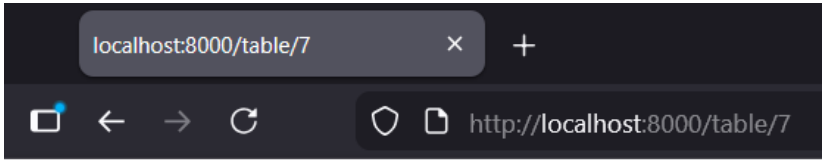
```php
Route::get('/posts/{post}/comments/{comment}', function (string $postId, string $commentId) {
    // ...
});
```

6.- Routing

The following example:

```php
Route::get('/table/{number}', function ($number) {
    for($i =1; $i <= 10 ; $i++){
        echo "$i * $number = ". $i* $number ."<br>";
    }
});
```

Will generate the response:



1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
10 * 7 = 70

6.- Routing

Occasionally you may need to specify a route parameter that may not always be present in the URI. You may do so by placing a ? mark after the parameter name. Make sure to give the route's corresponding variable a default value:

```php
Route::get('/user/{name?}', function (?string $name = null) {
    return $name;
});

Route::get('/user/{name?}', function (?string $name = 'John') {
    return $name;
});
```

6.- Routing

You may constrain the format of your route parameters using the where method on a route instance. The where method accepts the name of the parameter and a regular expression defining how the parameter should be constrained:

```php
Route::get('/user/{name}', function (string $name) {
    // ...
})->where('name', '[A-Za-z]+');

Route::get('/user/{id}', function (string $id) {
    // ...
})->where('id', '[0-9]+');

Route::get('/user/{id}/{name}', function (string $id, string $name) {
    // ...
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

6.- Routing

For convenience, some commonly used regular expression patterns have helper methods that allow you to quickly add pattern constraints to your routes:

```php
Route::get('/user/{id}/{name}', function (string $id, string $name) {
    // ...
})->whereNumber('id')->whereAlpha('name');

Route::get('/user/{name}', function (string $name) {
    // ...
})->whereAlphaNumeric('name');

Route::get('/user/{id}', function (string $id) {
    // ...
})->whereUuid('id');

Route::get('/user/{id}', function (string $id) {
    //
})->whereUlid('id');

Route::get('/category/{category}', function (string $category) {
    // ...
})->whereIn('category', ['movie', 'song', 'painting']);
```

7.- Views

Documentation on views is available here https://laravel.com/docs/12.x/views

- In the MVC pattern, views allow data to be presented visually to the user so that the user can interact and make another request.
- Thanks to these, the presentation part of the logic (controllers) and data persistence (model) can be separated.
- Thus, the views do not make queries to the database or process them, they simply receive the data and prepare it to display it with HTML code.

7.- Views

Defining views

Views should be saved in the resources/views directory as php files. They contain HTML code and PHP code (even template blade code). Below is the code for a view (resources/views/home.php):

```
<html>
    <head>
        <title>Mi Web</title>
    </head>
    <body>
        <h1>¡Hola <?php echo $nombre; ?>!</h1>
    </body>
</html>
```

7.- Views

Referencing views

With the view already created, it can be referenced from the routes file**:**

```
Route::get('/', function() {
    return view('home', array('nombre' => 'Dani'));
});
```

This path will return the content of the 'home' view that corresponds to the home.php file located within the resources/views/ directory. Additionally, it is provided with an array of data that the view will use.

7.- Views

Defining views

This in an example of a blade view, stored in resources/views/greeting.blade.php:

```
<html>
    <body>
        <h1>Hello, {{ $name }}</h1>
    </body>
</html>
```

7.- Views

Referencing views

With the view already created, it can be referenced from the routes file**:**

```
Route::get('/', function () {
    return view('greeting', ['name' => 'Dani']);
});
```

7.- Views

Referencing views

To maintain an organization in the views directory, these can be grouped using subdirectories, for example, you can have a resources/views/user directory and within this directory three views login.php, register.php and profile.php.

Thus, to call the corresponding view, dot notation will be used, for example, user.login will refer to the view resources/views/user/login.php.

```
Route::get('register', function()  {
    return view('user.register’);
});
```

7.- Views

Passing data to a view

The second parameter of the view function is used to pass parameters:

```
Route::get('user/profile/{id}', function($id) {
    $user = // Cargar los datos del usuario a partir de $id;
    return view('user.profile', array('user' => $user));
});
```

7.- Views

Passing data to a view

Laravel offers a bit clearer notation:

```
$view = view('home')->with('nombre', 'Javi');

$view = view('user.profile’)
                        ->with('user', $user)
                        ->with('editable', false);

Route::get('home/{user}', function($user) {
        $view = view('home')->with('nombre', $user);
        return $view;
});
```

7.- Views

Blade templates

Using Blade you can define templates in the views. This library allows you to perform all types of operations with the data, in addition to replacing template sections with other content, inheritance between templates, defining layouts or base templates, etc.

View files that use the Blade templating system must have the .blade.php extension. This extension also does not have to be included when referencing a view from the routes file or from a controller. That is, view('home') will be used whether the file is called home.php or home.blade.php

7.- Views

Blade templates

In general, the code that includes Blade in a view will begin with the symbols @ or {{, which will later be processed and prepared to be displayed on the screen.

Blade does not add processing overhead, since all views are preprocessed and cached; on the contrary, it provides us with utilities that will help us in the design and modularization of the views.

7.- Views

Blade templates

To display data, double braces {{ }} are used, which in addition to displaying the data, applies the htmlentities function to avoid injection attacks. Inside the braces, either a variable or a function that returns the content to be displayed will be written:

```
Hola {{ $name }}.
La hora actual es {{ time() }}.
```

Show a data only if it exists:

```
{{ isset($name) ? $name : 'Valor por defecto' }}
{{ $name or 'Valor por defecto' }}
```

7.- Views

Blade templates

Comments:

{{-- Este comentario no se mostrará en HTML --}}

Control structures:

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

7.- Views

Blade templates

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

7.- Views

Blade templates

Including a template within another template

To include a template within another in Blade, the @include instruction is used to which you can also pass an array with data. In this way the views can be reused and their content can even be separated as was traditionally done in PHP with include/require.

```
@include('view_name’)
@include('view_name', array('some'=>'data'))
```

8.- Layouts

The documentation: https://laravel.com/docs/12.x/blade#building-layouts

Blade also allows you to define layouts to create a base HTML structure in which there will be sections that will be filled with other templates or views.

The most common thing is to create a layout with the main or common content: head, body, nav... and define sections that will be filled by other templates to complete the code.

The layout can be used on all pages of the web application so that code reuse is optimized.

8.- Layouts

Below is a layout template (resources/views/layouts/master.blade.php):

```
<html>
    <head>
        <title>Mi Web</title>
    </head>
    <body>
        @section('menu')
            Contenido del menu
        @show
        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

8.- Layouts

The instructions @section and @yield have been used in the layout:

@section('NombreDeSección'): allows you to add content in that section.

@yield('NombreDeSección’): this directive directive is used to display the contents of a given section. It replaces the content of that section of the layout with that indicated in the template that extends the layout. You can define default content : @yield('NombreDeSección', 'Contenido por defecto’)

Let’s see an example.

8.- Layouts

```
<!-- resources/views/layouts/app.blade.php -->

<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            This is the master sidebar.
        @show

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

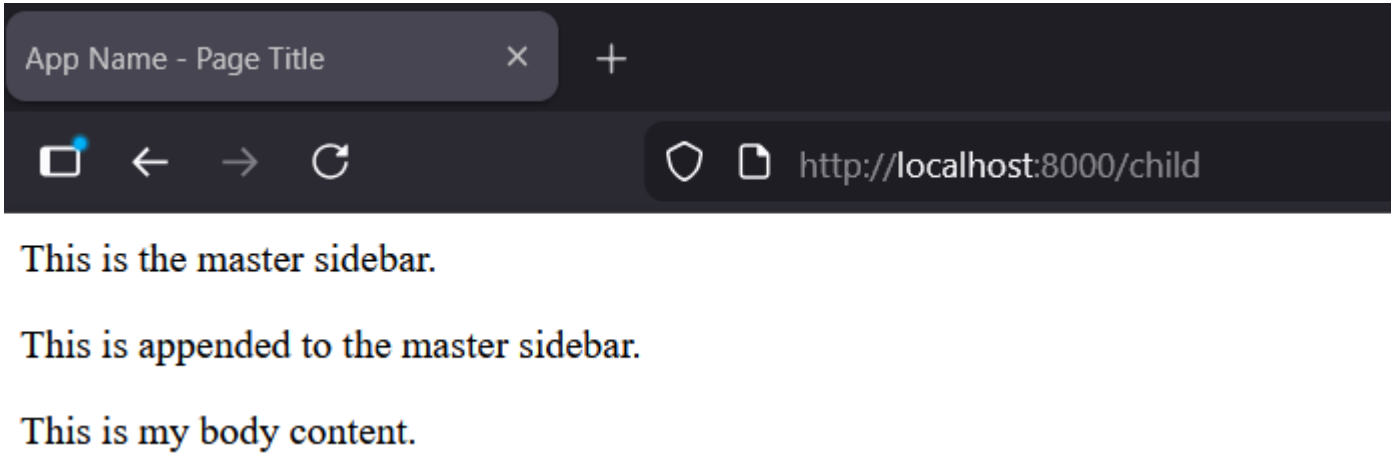
8.- Layouts

```
<!-- resources/views/child.blade.php -->

@extends('layouts.app')

@section('title', 'Page Title')

@section('sidebar')
    @parent

    <p>This is appended to the master sidebar.</p>
@endsection

@section('content')
    <p>This is my body content.</p>
@endsection
```

8.- Layouts

The last step is to configure the route and access it:

```php
Route::get('/child', function()
{
    return view('child');
});
```



8.- Layouts

In this example, the sidebar section is utilizing the @parent directive to append (rather than overwriting) content to the layout's sidebar.

The @parent directive will be replaced by the content of the layout when the view is rendered.

@extends('RutaDelLayout'): indicates which layout should be used.

@section('NombreDeSección'): allows you to add content in that section.

9.- Quickstart

We are going to do a basic Laravel tutorial named “Basic Task List”.

The tutorial uses Laravel version 5.2, we have to create the project with that version with the following command:

```
composer create-project laravel/laravel quickstart 5.2.*
```

Now follow the tutorial:

https://laravel.com/docs/5.2/quickstart

10.- Exercise

Follow the steps to develop an application for the management of a video store.

- Create a project named “videoclub”
- Define the routes (they will just will return a string):

Route	Type	String to return
/	GET	Main page
Login	GET	User login
Logout	GET	User logout
Catalog	GET	Movie list
Catalog/show/{id}	GET	Detail view of the movie {id}
Catalog/create	GET	Add movie
Catalog/edit/{id}	GET	Edit movie {id}

10.- Exercise

Create the routes so that they simply return a string, this way it will be easily verified that they are correctly created.

To verify that the routes have been created correctly you can use the Artisan command to display the list of routes.

To check the correct behavior, you must access the URLs in the web browser.

10.- Exercise

- Creating the main layout

Copy the navbar.blade.php file to the directory “resources/views/partials”

Create the file resources/views/layouts/master.blade.php and add the content of the base template indicated by Bootstrap in its documentation:
https://getbootstrap.com/docs/4.1/getting-started/introduction/#starter-template

In “master.Blade.php” modify the following sections:

- Remove the <h1> tag and include the navbar file instead: @include('partials.navbar’)
- Add the following code below:

```
<div class="container">
    @yield('content’)
</div>
```

10.- Exercise

- Creating the remaining views

At this point, all the views that will extend from the main layout must be created, showing in the created section the text that had been previously defined within the routes.

As an example, the content of the home page is shown:

```
@extends('layouts.master’)

@section('content’)
    Pantalla principal
@stop
```

10.- Exercise

To maintain an organization, views will be grouped according to their function and will be created for all actions except logout:

Route	Folder	View
/	resources/views/	home.blade.php
login	resources/views/auth/	login.blade.php
catalog	resources/views/catalog/	index.blade.php
catalog/show/{id}	resources/views/catalog/	show.blade.php
catalog/create	resources/views/catalog/	create.blade.php
catalog/edit/{id}	resources/views/catalog/	edit.blade.php

10.- Exercise

At this point, the views are already created, so the routes will need to be updated, so that instead of returning the text they return the created views.

Remember that dot notation is used to indicate the path of the views:

```
return view('home’);
return view('catalog.index’);
return view('catalog.show', array('id'=>$id));
```

Once the routes have been updated, you can check their operation by accessing the web application from the browser.