



# Construye con Laravel

PHP



Qualentum Lab

# Construye con Laravel

Un **framework** es un conjunto de herramientas y componentes predefinidos que permiten a los desarrolladores construir aplicaciones web de manera más eficiente al proporcionar una estructura organizada y reutilizable para el desarrollo.

En este fastbook, vamos a poner el foco en uno de los principales marcos de trabajo para el desarrollo en PHP, **en Laravel: un framework de desarrollo web** de código abierto.

Autor: Jesús Donoso

☰ Introducción

☰ Controladores Laravel

☰ Creación de un proyecto Laravel

☰ Vistas en Laravel

☰ Estructura de un proyecto Laravel

☰ Operaciones en BBDD en Laravel

☰ Enrutamiento

☰ Formulario y Validación

☰ Modelos en Laravel

☰ Recursos de interés

# Introducción



Creado por Taylor Otwell y lanzado en 2011, Laravel se caracteriza por ser una plataforma con una sintaxis clara, por su sencillez de uso, y contar con una amplia gama de funcionalidades que facilitan la creación de aplicaciones web de alta calidad.

En cuanto a su importancia en el desarrollo web, Laravel desempeña un papel fundamental en la simplificación y optimización del proceso de desarrollo. A continuación, se detallan algunas de sus principales **características**.

- Sintaxis elegante:** Laravel destaca por su sintaxis limpia y expresiva, lo que contribuye a la legibilidad y mantenimiento del código, aumentando la productividad de los desarrolladores.
- ORM eloquent:** el uso de un *Object-Relational Mapping* (*ORM*) como *Eloquent* facilita la interacción con bases de datos, permitiendo a los desarrolladores trabajar con datos de manera orientada a objetos en lugar de escribir consultas SQL directas.
- Enrutamiento flexible:** el sistema de enrutamiento de Laravel es altamente adaptable y permite la definición de rutas y controladores de forma clara y eficiente, simplificando la gestión de URLs y la creación de API.

- Motor de plantillas blade:** Laravel incluye este motor de plantillas que simplifica la creación de vistas HTML, fomentando la reutilización de componentes y la creación de diseños flexibles.
- Seguridad incorporada:** este marco de trabajo se preocupa por la seguridad, proporcionando protección contra amenazas comunes como *CSRF* (*Cross-Site Request Forgery*) y *XSS* (*Cross-Site Scripting*) de manera predeterminada.
- Autenticación y autorización completas:** Laravel ofrece una funcionalidad completa para la autenticación y autorización, lo que facilita la implementación de sistemas de registro, inicio de sesión y gestión de roles.
- Paquetes y complementos:** la comunidad de Laravel ha desarrollado una amplia variedad de paquetes y complementos que extienden las capacidades del marco de trabajo, permitiendo a los desarrolladores ahorrar tiempo y esfuerzo en el desarrollo de aplicaciones.
- Migraciones y semillas:** herramientas con las que cuenta Laravel para simplificar la gestión de bases de datos y la introducción de datos de prueba.
- Testing integrado:** facilita la realización de pruebas unitarias y funcionales, mejorando la fiabilidad de las aplicaciones.
- Documentación y comunidad activa:** el marco de trabajo se beneficia de una documentación oficial extensa y una comunidad de desarrolladores activa que comparte recursos, tutoriales y soluciones a problemas comunes.

Como decimos, Laravel se ha convertido en un framework muy importante en el desarrollo web debido a su capacidad para **simplificar** tareas complejas, **acelerar** el proceso de desarrollo y **mejorar la seguridad** en las aplicaciones. Estas características han contribuido a su popularidad entre los desarrolladores que buscan crear aplicaciones web modernas y escalables.

# Creación de un proyecto Laravel



Antes de iniciarte en el estudio y la ejecución de los comandos para que puedas crear un proyecto Laravel es necesario que cuentes con estos **requisitos**:

- Tener [PHP](#) previamente instalado en el sistema con una versión igual o superior a 7.4.
- Asegurarse de tener Composer instalado. Puede descargarse e instalarse desde la web oficial de [Composer](#).
- Es necesario disponer de un servidor web, como Apache o Nginx.
- Se debe contar con una base de datos.

El primer paso para crear un proyecto Laravel es abrir un terminal o línea de comandos y ejecutar el siguiente comando. Se debe reemplazar *nombre\_del\_proyecto* con el nombre deseado para el proyecto:

```
composer create-project --prefer-dist laravel/laravel nombre_del_proyecto
```

Este comando descarga e instala Laravel y todas sus dependencias en una carpeta llamada con el mismo nombre del proyecto. A continuación, ya se puede acceder a la carpeta del proyecto recién creado:

```
cd nombre_del_proyecto
```

El siguiente paso necesario es copiar el archivo `.env.example` y crear un nuevo archivo llamado `.env`. Después, se debe generar una clave de cifrado de la aplicación. Los comandos para la copia y la generación de la clave de cifrado son:

```
cp .env.example .env  
php artisan key:generate
```

A continuación, se debe editar el archivo `.env` y configurar las credenciales de la base de datos. También estableceremos los valores para `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME` y `DB_PASSWORD` de acuerdo con el entorno de desarrollo.

Hay que recordar que Laravel utiliza migraciones para gestionar la estructura de la base de datos. Las migraciones son las consultas que se realizan a la base de datos para la creación de tablas. ¿Cómo? En primer lugar, se deben ejecutar mediante este comando:

```
php artisan migrate
```

Y tras su ejecución, se crean las tablas predeterminadas en la base de datos, como las tablas de usuarios.

A la hora de ejecutar la aplicación de Laravel localmente, se puede utilizar el servidor de desarrollo de Laravel. Para iniciar el servidor es necesario hacer uso de este comando:

```
php artisan serve
```

El servidor se inicia en la dirección en **<http://localhost:8000>**; y puedes acceder a la aplicación acudiendo a dicha dirección desde un navegador web.

Pues bien, ya estaría creado y configurado un proyecto Laravel. Tras estos pasos, también estaría listos para continuar el desarrollo. Es importante recordar que este es solo el comienzo de la creación de un proyecto y que puedes personalizarlo según las necesidades de la aplicación.

 Por último, te recomiendo que consultes la [documentación Laravel](#), ya que encontrarás información exhaustiva y una comunidad activa que te puede ayudar en el proceso de desarrollo.

# Estructura de un proyecto Laravel



Como hemos adelantado, una característica muy relevante de un proyecto Laravel es que su organización y estructura facilitan el desarrollo de aplicaciones web de manera eficiente y ordenada. En el directorio raíz del proyecto Laravel se encuentran varios **archivos y carpetas fundamentales**. Ahora vamos a definir algunos de estos archivos y directorios:

- **app:** en este directorio se encuentra la lógica central de la aplicación. Contiene modelos, controladores, proveedores de servicios y otros archivos relacionados con la funcionalidad de la aplicación.
- **bootstrap:** contiene archivos de inicio, como el archivo *app.php*, que inicializan la aplicación.
- **config:** directorio que almacena archivos de configuración que permiten ajustar la aplicación a las necesidades específicas.
- **database:** directorio que contiene las migraciones que definen la estructura de la base de datos y semillas que se utilizan para poblar la base de datos con datos iniciales.
- **public:** ubicación de los recursos accesibles públicamente, como imágenes, hojas de estilo y scripts JavaScript.
- **resources:** en este directorio se encuentran las vistas Blade, que se utilizan para renderizar las páginas HTML de la aplicación. Así como los archivos de traducción y recursos de front-end, como archivos SASS y JavaScript.

- **routes:** directorio en el que se definen las rutas de la aplicación, especificando qué controlador y método debe manejar cada URL.
- **storage:** directorio que contiene los archivos generados por la aplicación, como los registros, los archivos de sesiones y la caché.
- **tests:** directorio que contiene las pruebas unitarias y las funcionales para asegurar la calidad del código.
- **vendor:** ubicación en la que Composer almacena las dependencias de terceros que se utilizan en el proyecto.
- **.env:** archivo de entorno donde se configuran variables específicas de la aplicación, como la configuración de la base de datos.
- **.gitignore:** archivo que especifica qué archivos y directorios no deben incluirse en el control de versiones.
- **composer.json:** archivo en el que se definen las dependencias del proyecto y otras configuraciones específicas de Composer.
- **README.md:** fichero que contiene la documentación básica del proyecto que puede ser útil para otros desarrolladores.

---

**Esta estructura organizativa de un proyecto Laravel es fundamental para mantener el código limpio y ordenado, facilitando la colaboración entre equipos de desarrollo y el mantenimiento a largo plazo de la aplicación.**

---

# Enrutamiento



En este apartado, vamos a estudiar las **claves del enrutamiento de Laravel**: cómo se gestionan las solicitudes HTTP y cómo se responde a ellas. Como característica reseñable, quédate con que **su sistema de enrutamiento es muy flexible y potente a la hora de definir las rutas y los controladores** para manejar las diversas URL de la aplicación.

## Definición de rutas

En Laravel, las rutas se definen en los archivos `routes/web.php` para rutas web y en `routes/api.php` para rutas de una API. Se pueden definir rutas utilizando el **método Route**: seguido del verbo HTTP correspondiente (por ejemplo, `get`, `post`, `put`, `delete`) y el URL al que responderá la ruta.

A continuación, te comarto un ejemplo de definición de una ruta:

```
Route::get('/inicio', function () {
    return '¡Bienvenido a la página de inicio!';
});
```

Como vemos en este ejemplo, se define la ruta para que cuando el usuario acceda a la URL `"/inicio"` aparezca el texto `"¡Bienvenido a la página de inicio!"` en el navegador.

También es posible definir rutas con parámetros que capturan valores dinámicos de la URL. Los parámetros se indican encerrándose entre llaves {} en la definición de la ruta. Por ejemplo, una ruta con parámetro se vería así:

```
Route::get('/usuario/{id}', function ($id) {
    return 'Mostrando información del usuario con ID: ' . $id;
});
```

Es una práctica común dar nombres a las rutas, lo que facilita la generación de URL y la referencia a ellas en la aplicación. Una ruta con nombre se define de esta manera:

```
Route::get('/contacto', function () {
    return 'Página de contacto';
})->name('contacto');
```

1

## Rutas con controladores

En lugar de definir la lógica de manejo directamente en la definición de una ruta, es una buena práctica utilizar controladores. Un controlador es una clase que se encarga de manejar la lógica de una ruta en particular. En el ejemplo a continuación, la ruta `usuarios` llama al método `index` del controlador `UserController` cuando se accede a la ruta.

```
Route::get('/usuarios', 'UserController@index');
```

2

## Rutas con middleware

El middleware permite ejecutar acciones antes de que una ruta se procese. Pueden aplicarse middleware a rutas individuales o a un grupo de rutas. En este caso, el middleware "auth" asegura que el usuario esté autenticado antes de acceder a la página de administración.

```
Route::get('/admin', 'AdminController@index')->middleware('auth');
```

3

## Rutas con parámetros opcionales

Laravel permite definir parámetrosopcionales en las rutas utilizando el signo de interrogación "?". Por ejemplo, una ruta con parámetrosopcionales se vería así:

```
Route::get('/producto/{nombre}/{marca?}', function ($nombre, $marca = null)
{
    // $marca es un parámetro opcional
});
```

En el ejemplo anterior si el usuario accede a la URL "/producto/nombre" se ejecuta la función de la misma forma que si el usuario accede a la URL "/producto/nombre/marca". Dentro de los argumentos de la función se ha marcado que marca pueda ser nulo.

Todos los conceptos detallados en este apartado son fundamentales para comprender cómo funciona el enrutamiento en Laravel, ya que, con este sistema de enrutamiento, es posible definir cómo se manejan las solicitudes HTTP y conectarlas a las acciones y controladores correspondientes en una aplicación web.

# Modelos en Laravel



Los modelos en Laravel son clases de PHP que representan las tablas de la base de datos de la aplicación.

Siguiendo el patrón de diseño **Active Record**, un modelo no solo define la estructura de una tabla, sino que también incluye métodos para realizar operaciones de lectura y escritura en esa tabla.

## Creación de modelos

Para crear un modelo en Laravel, se utiliza el comando *artisan* que se puede observar a continuación:

```
php artisan make:model nombre_modelo
```

La ejecución del comando anterior genera un archivo de modelo en el directorio `app/Models`. Este archivo contiene la definición de la clase del modelo. Veamos un ejemplo:

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    protected $fillable = ['nombre', 'precio', 'descripcion'];
}
```

Dentro del modelo se puede especificar la tabla de la base de datos a la que se asocia. Si el nombre de la tabla difiere del nombre plural del modelo es posible especificarlo manualmente. La propiedad `$fillable`, que se puede observar en el ejemplo anterior, obliga a que cada registro que se quiere incluir en la tabla tenga que estar completo para los atributos que tengan dicha propiedad.

Adicionalmente, los modelos permiten definir relaciones entre ellos, como "uno a uno", "uno a muchos", "muchos a muchos", lo que facilita el acceso a registros relacionados.

## Eloquent ORM

*Eloquent* es el *ORM (Object-Relational Mapping)* incluido en Laravel. Proporciona una serie de métodos que permiten a los desarrolladores interactuar con la base de datos de una manera sencilla y orientada a objetos a través de modelos. A continuación, paso a describir algunos de los métodos proporcionados por Eloquent.

## 1

## Recuperación de registros

- **`all()`:** recupera todos los registros de una tabla en la base de datos.
- **`find($id)`:** encuentra un registro por su clave primaria.
- **`first()`:** obtiene el primer registro que cumple con una condición dada.
- **`get()`:** recupera múltiples registros que cumplen con una condición.
- **`findOrFail($id)`:** similar a `find()`, pero lanza una excepción si el registro no se encuentra.
- **`findMany($ids)`:** recupera varios registros por sus claves primarias.
- **`whereIn($column, $values)`:** recupera los registros donde una columna está en una lista de valores.
- **`orderBy($column, $direction)`:** ordena los resultados por una columna específica.
- **`count()`:** devuelve el número de registros que cumplen con una condición.

## 2

## Creación y actualización de registros

- **`create($attributes)`:** crea un nuevo registro en la base de datos.
- **`update($attributes)`:** actualiza los registros existentes en la base de datos.
- **`save()`:** guarda un modelo después de crearlo o actualizarlo.
- **`delete()`:** elimina un registro de la base de datos.
- **`increment($column, $amount)`:** incrementa el valor de una columna en una cantidad específica.
- **`decrement($column, $amount)`:** decrementa el valor de una columna en una cantidad específica.

## 3

**Consultas avanzadas:**

- **`where($column, $operator, $value)`:** agrega una condición WHERE a la consulta.
- **`orWhere($column, $operator, $value)`:** agrega una condición OR a la consulta.
- **`select($columns)`:** selecciona columnas específicas en la consulta.
- **`join($table, $first, $operator, $second)`:** realiza una operación JOIN en la consulta.
- **`groupBy($column)`:** agrupa los resultados por una columna.
- **`having($column, $operator, $value)`:** agrega una condición HAVING a la consulta.
- **`distinct()`:** devuelve resultados únicos sin duplicados.

## 4

**Relaciones**

Eloquent además permite definir y trabajar con relaciones entre modelos, como "**`belongsTo`**", "**`hasMany`**", "**`belongsToMany`**", "**`hasOne`**", entre otros. Estas relaciones facilitan el acceso a registros relacionados en la base de datos.

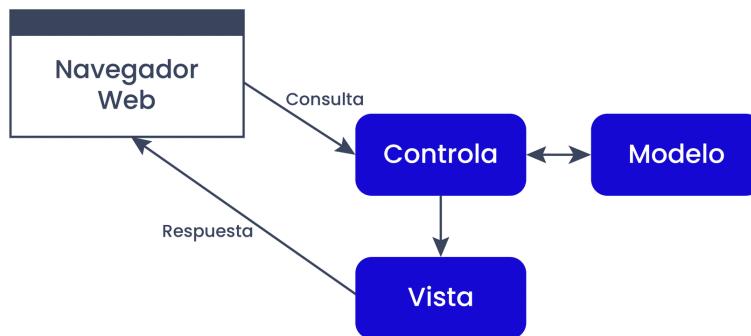
---

**Los modelos en Laravel son una parte fundamental de la interacción con la base de datos y facilitan la manipulación de datos. Puedes definir la estructura de la tabla, establecer relaciones, realizar consultas y mucho más con la ayuda de modelos en Laravel.**

# Controladores Laravel



Laravel implementa el **diseño modelo-vista-controlador (MVC)**. Esta arquitectura de tres componentes sigue la lógica que se puede observar en el siguiente diagrama:



Estudiemos ahora qué es cada uno de los elementos presentados en nuestro esquema:

- El **controlador** se encarga de manejar las solicitudes de los usuarios y se comunica con el modelo y la vista.
- El **modelo** es la representación de la capa de acceso a bases de datos.
- La **vista** es la capa encargada de representar la información de manera amigable.

Los controladores en Laravel son clases que desempeñan un papel fundamental en la organización y gestión de la lógica de una aplicación web. Estas clases proporcionan una estructura coherente para manejar solicitudes HTTP, interactuar con la base de datos y preparar datos para su presentación en las vistas.

## Creación de controladores

Para crear un controlador en Laravel, se utiliza el generador de controladores **Artisan**. Fíjate en este ejemplo:

```
php artisan make:controller NombreControlador
```

Tras la ejecución del comando se crea un archivo llamado *NombreControlador.php* en el directorio *app/Http/Controllers*. En este archivo, se pueden definir los métodos que llevarán a cabo diversas tareas.

Por otro lado, Laravel proporciona la opción de crear controladores de recursos que manejan rutas relacionadas con operaciones **CRUD** (crear, leer, actualizar, eliminar). Mediante el siguiente comando podemos crear un controlador para realizar estas operaciones CRUD:

```
php artisan make:controller NombreControlador --resource
```

## Métodos del controlador

Los métodos de un controlador se encargan de realizar acciones específicas dentro de la aplicación. Por ejemplo, un controlador podría tener un método para mostrar una lista de usuarios o para presentar un formulario de registro. Los métodos se definen en el archivo de controlador previamente creado.

```
public function index()
{
    // Lógica para mostrar una lista de usuarios
}

public function create()
{
    // Lógica para mostrar un formulario de registro
}
```

Laravel permite la inyección de dependencias en los controladores. Esto significa que se pueden solicitar instancias de clases o servicios específicos como argumentos en los métodos del controlador. Esto facilita la recuperación de datos de la base de datos, la autenticación y otras tareas.

```
use App\User;
use Illuminate\Http\Request;

public function show(User $user)
{
    // El objeto $user se inyecta automáticamente y contiene el usuario
    solicitado
}
```

Los métodos del controlador pueden devolver diferentes tipos de respuestas, como vistas, datos JSON o redirecciones. Por ejemplo, para devolver una vista, se puede ejecutar un código como el siguiente:

```
public function index()
{
    return view('usuarios.index');
```

Es posible aplicar *middleware* a un controlador completo o a métodos específicos para agregar capas de seguridad o autenticación. Esto se especifica en el constructor del controlador o en la definición de métodos. El siguiente ejemplo comprueba que el usuario que está intentando acceder a una ruta está autenticado en la aplicación.

```
public function __construct()
{
    $this->middleware('auth');
```

En definitiva, los controladores desempeñan un papel fundamental en Laravel, ayudando a mantener la lógica de la aplicación organizada y separada de las vistas y las rutas. Permiten un enfoque estructurado en el desarrollo de aplicaciones web.

# Vistas en Laravel



Las vistas son componentes cruciales que permiten presentar información a los usuarios en forma de páginas web.

En Laravel, estas vistas son archivos que contienen código HTML y, opcionalmente, instrucciones de PHP para mostrar datos de manera estructurada y visualmente atractiva.

## Creación de vistas y plantillas Blade

Las vistas en Laravel se almacenan en el directorio `resources/views`. Se pueden crear vistas simplemente creando archivos dentro de esta carpeta. Por ejemplo, para crear una vista llamada "`inicio.blade.php`", podemos utilizar el siguiente comando:

```
touch resources/views/inicio.blade.php
```

Laravel utiliza el motor de plantillas Blade para las vistas, que ofrece una sintaxis elegante y expresiva para integrar instrucciones de PHP en las vistas. Esto facilita la presentación de datos dinámicos y la creación de plantillas reutilizables. El código a continuación es un ejemplo para mostrar una variable en una vista Blade:

```
<h1>{{ $titulo }}</h1>
```

Blade ofrece directivas para condicionales, bucles y otras estructuras de control, lo que facilita la lógica en las vistas. Además, se pueden utilizar comentarios Blade para documentar las vistas sin afectar la salida final.

```
@if ($usuario->esAdmin)
    <p>Este usuario es un administrador.</p>
@else
    <p>Este usuario no es un administrador.</p>
@endif
```



Consulta la documentación de las Plantillas Blade y la documentación Laravel [aquí](#).

Por otro lado, se pueden pasar variables como parámetros a las vistas. Estas variables se especifican en un *array* que se pasa al método *view* o, en el caso de Blade, se pueden definir directamente en la vista.

```
@extends('layouts.app')
@section('content')


<h1>Perfil de Usuario</h1>
    <p>Nombre: {{ $user['name'] }}</p>
    <p>Correo Electrónico: {{ $user['email'] }}</p>
    <p>Edad: {{ $user['age'] }}</p>


```

En el ejemplo anterior vemos cómo a los párrafos de la plantilla se le introducen como variables los datos contenidos en el array “*user*”.

## Herencia de plantillas

Blade permite la herencia de plantillas, lo que significa que se pueden definir plantillas principales (*layouts*) con secciones que pueden ser reemplazadas por contenido específico en vistas secundarias. Esto es útil para mantener una estructura coherente en toda la aplicación. A continuación te comarto un ejemplo de una plantilla principal (*layout*) con una sección:

```
<!DOCTYPE html>
<html>
<head>
    <title>@yield('titulo')</title>
</head>
<body>
    <div class="contenido">
        @yield('contenido')
    </div>
</body>
</html>
```

En una vista secundaria se puede extender la plantilla principal y completar las secciones como se puede ver en el siguiente fragmento de código:

```
@extends('layout')

@section('titulo', 'Página de Inicio')

@section('contenido')
    <p>Bienvenido a la página de inicio.</p>
@endsection
```

## Generación de vistas desde los controladores

En los métodos de los controladores se pueden devolver vistas utilizando el método `view`. Este método toma el nombre de la vista y, opcionalmente, un arreglo de datos que se pueden utilizar en la vista. Aquí tienes un ejemplo:

```
public function index()
{
    $titulo = 'Página de Inicio';
    return view('inicio', ['titulo' => $titulo]);
}
```

---

**Las vistas son uno de los pilares del desarrollo web en Laravel, ya que permiten crear interfaces de usuario atractivas y dinámicas. Con la sintaxis Blade y la capacidad de heredar plantillas, conseguimos un código limpio y organizado; también nos aseguramos una presentación eficiente de los datos a los usuarios.**

# Operaciones en BBDD en Laravel



En Laravel, las bases de datos y los modelos son componentes esenciales en la interacción con la *data* de una manera óptima y clara. Y es que, como ya he comentado, proporciona un sistema de ORM (*Object-Relational Mapping*) que facilita la creación y manipulación de las tablas de bases de datos utilizando objetos orientados a objetos.

## Configuración de la base de datos

La configuración de la base de datos en una aplicación Laravel se realiza en dos etapas principales: la configuración en el archivo de entorno (`.env`) y, si es necesario, la configuración adicional en el archivo de configuración de Laravel.

El archivo `.env` se encuentra en el directorio raíz del proyecto Laravel, el lugar principal para configurar la conexión a la base de datos. Deben establecerse varias variables de entorno en este archivo. Vamos a ver cada una y definirlas.

- **DB\_CONNECTION:** indica el tipo de base de datos que se utilizará, como `mysql`, `pgsql`, `sqlite`, `sqlsrv`, entre otros.
- **DB\_HOST:** especifica la dirección IP o el nombre del host del servidor de la base de datos. Normalmente, es `"127.0.0.1"` o `"localhost"` si la base de datos se encuentra en el mismo servidor.
- **DB\_PORT:** el puerto en el que la base de datos escucha; por defecto, los valores comunes son `3306` para MySQL y `5432` para PostgreSQL.
- **DB\_DATABASE:** el nombre de la base de datos.
- **DB\_USERNAME:** el nombre de usuario para acceder a la base de datos.
- **DB\_PASSWORD:** la contraseña asociada al nombre de usuario.

Ahora vamos a ver un ejemplo de la configuración en el archivo `.env` para una base de datos MySQL:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=nombre_de_la_base_de_datos
DB_USERNAME=nombre_de_usuario
DB_PASSWORD=contraseña
```

---

**Recuerda este tip: si se necesitan configuraciones adicionales específicas de la base de datos, como el conjunto de caracteres o el tiempo de espera de la conexión, se pueden establecer en el archivo `config/database.php`. Sin embargo, en la mayoría de los casos, las configuraciones predeterminadas son adecuadas.**

## Migraciones

Las migraciones en Laravel son una característica poderosa que permite a los desarrolladores gestionar la estructura de la base de datos de una aplicación de manera organizada y controlada. Con las migraciones, es posible crear, modificar y eliminar tablas y columnas en la base de datos de una manera eficiente y rastreable.

Para crear una nueva migración en Laravel, se utiliza el comando `make:migration` de **Artisan**. Por ejemplo, para crear una migración que defina la estructura de una tabla de usuarios se ejecuta el siguiente comando:

```
php artisan make:migration nombre_migracion
```

Este comando genera un archivo de migración en el directorio `database/migrations`. Dentro del archivo de migración, se encuentran dos métodos principales: `up` y `down`.

### método up

Se utiliza para definir los cambios que se desean aplicar en la base de datos.

### método down

Se utiliza para deshacer los cambios que se han aplicado en la base con la función `up`. Dicho método es utilizado cuando se hace `rollback` de la migración. Veamos un ejemplo para entenderlo mejor:

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamps();
    });
}
// El método 'down', se define cómo deshacer los cambios realizados en 'up'
public function down()
{
    Schema::dropIfExists('users');
}
```

Para aplicar las migraciones y crear la estructura de la base de datos, se ejecuta el siguiente comando:

```
php artisan migrate
```

Este comando aplica todas las migraciones pendientes. Puede ejecutarse cada vez que se crean nuevas migraciones o se realizan modificaciones en las existentes.

Si es necesario deshacer una migración, se puede utilizar el comando `migrate:rollback`. Por ejemplo, para deshacer la última migración, se ejecuta:

```
php artisan migrate:rollback
```

Además de crear tablas, las migraciones permiten agregar columnas, modificar columnas existentes y llevar a cabo otras operaciones de manipulación de la base de datos.

 En la documentación de Laravel se pueden encontrar diversos métodos disponibles para definir los cambios necesarios en las migraciones.

Puedes consultarla [aquí](#).

También podemos combinar las migraciones con las semillas (`seeds`), así podemos insertar datos de prueba en la base de datos al mismo tiempo que se ejecutan las migraciones. Esto es útil para llenar la base de datos con datos iniciales.

# Formulario y Validación



Qualentum Lab

Como ya sabes, gestionar los formularios y la validación es un proceso clave en el desarrollo de aplicaciones web, y Laravel ofrece herramientas muy potentes con esta finalidad.

## Creación de formularios HTML

Para crear un formulario HTML en una vista de Laravel, se utiliza Blade, el motor de plantillas de Laravel. Además, se agrega el método `csrf` para generar un **token CSRF** que protege el formulario contra ataques de falsificación de solicitud entre sitios (CSRF). Un ejemplo de código podría ser el siguiente:

```
<form method="POST" action="/procesar-formulario">
    @csrf
    <!-- Campos del formulario -->
    <input type="text" name="nombre">
    <input type="email" name="correo">
    <button type="submit">Enviar</button>
</form>
```

## Recepción y validación de datos del formulario

En el controlador de Laravel, se puede acceder a los datos enviados a través del formulario utilizando la solicitud (*request*), por ejemplo:

```
public function procesarFormulario(Request $request)
{
    $nombre = $request->input('nombre');
    $correo = $request->input('correo');
    // Realizar acciones con los datos del formulario
}
```

Laravel proporciona un sistema de validación para asegurarse de que los datos que ha registrado el usuario sean válidos. Las reglas de validación se pueden definir en el controlador utilizando el método *validate*, o se pueden encapsular en una clase *Form Request* para mantener la lógica de validación separada. A continuación, muestro un ejemplo de validación en el controlador:

```
public function procesarFormulario(Request $request)
{
    $request->validate([
        'nombre' => 'required|string|max:255',
        'correo' => 'required|email|unique:usuarios',
    ]);

    // Si la validación es exitosa, continuar con el procesamiento de los
    // datos
}
```

Cuando la validación falla, Laravel redirige automáticamente al usuario de vuelta al formulario y proporciona una variable `$errors` que contiene los errores de validación. Estos errores se pueden mostrar en la vista para que el usuario pueda corregirlos.

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

Es posible personalizar los mensajes de error para proporcionar una experiencia de usuario más amigable. Esto se puede lograr definiendo mensajes personalizados en el controlador o en un archivo de lenguaje.

Después de un envío exitoso del formulario y validación, es común redirigir al usuario a una página de éxito o a una página relevante. Esto se puede hacer en el controlador utilizando el método `redirect`.

---

**Las características de validación integradas en Laravel  
permiten garantizar que los datos del usuario sean  
correctos y proporcionan una experiencia de usuario  
más segura y amigable.**

---

# Recursos de interés



Qualentum Lab

Como hemos visto Laravel es una herramienta que todo desarrollador especializado en PHP debe manejar con soltura y así optimizar sus tiempos y procesos de trabajo. Es un framework muy sencillo en su uso, que nos garantiza un código limpio y una estructura clara y ordenada.

Con esta lectura, he querido acercarte a sus **esenciales**, realizando un recorrido que partía de la creación de un nuevo proyecto para llegar a la gestión de formularios. Ahora te toca a ti, ahora es el turno de que practiques con esta plataforma y profundices en todas las posibilidades que nos ofrece. Para ello, te recojo aquí los enlaces directos a la documentación oficial y te animo a que sigas explorando el ecosistema PHP y Laravel.

- [Instalación PHP](#)
- [Documentación de Composer](#)
- [Documentación oficial Laravel](#)
- [Directivas en Laravel](#)
- [Migraciones Laravel](#)

¡Enhорabuena! Fastbook superado



[Qualentum.com](http://Qualentum.com)