DESARROLLO Y DISEÑO WEB

Miguel Cabrera

Tabla de contenido

World Wide Web

La **WWW (World Wide Web)**, conocida comúnmente como "la web", es un sistema de información global que permite a los usuarios acceder, compartir y consultar información a través de Internet. Funciona mediante hipervínculos y contenido multimedia, como texto, imágenes, videos y más.

Origen y combinaciones clave: La WWW surgió a partir de varias combinaciones de tecnologías y conceptos, entre ellos:

1. **HTTP (Hypertext Transfer Protocol):** Protocolo usado para transferir datos en forma de hipervínculos.
2. **HTML (HyperText Markup Language):** Lenguaje para estructurar y presentar contenido en las páginas web.
3. **URL (Uniform Resource Locator):** Dirección única que identifica cada recurso en la web.
4. **Internet:** La infraestructura que conecta redes de computadoras a nivel global.
5. **Hipervínculos:** Concepto de enlazar contenido mediante texto clicable.

Fue creada en 1989 por Tim Berners-Lee mientras trabajaba en el CERN, como una forma de compartir y acceder información fácilmente entre investigadores. Desde entonces, ha evolucionado y transformado cómo interactuamos con el mundo digital.

Modelo cliente-servidor:

- Es un diseño en el que las aplicaciones se dividen en dos partes: cliente y servidor.
- El cliente solicita datos o servicios (por ejemplo, tu navegador web).
- El servidor responde procesando la solicitud y enviando los datos (por ejemplo, el servidor web donde se aloja una página).

- Es la base de muchas aplicaciones en red, como sitios web, correos electrónicos, y servicios de streaming.

Modelo vista-controlador (MVC):

- Es un patrón de diseño para organizar aplicaciones dividiendo sus componentes en tres partes:
  - **Modelo:** Maneja la lógica de los datos y la comunicación con la base de datos.
  - **Vista:** Presenta la interfaz gráfica para que el usuario interactúe.
  - **Controlador:** Gestiona la interacción del usuario y los eventos, comunicándose entre la vista y el modelo.
- Es muy popular en el desarrollo de aplicaciones web debido a su modularidad y capacidad para separar responsabilidades.

CMS

CMS (Content Management System): Un CMS es un sistema de gestión de contenidos que permite crear, gestionar y modificar contenido en un sitio web sin necesidad de conocimientos avanzados de programación. Ejemplos populares incluyen WordPress, Joomla y Drupal. Los CMS son ideales para blogs, sitios corporativos y tiendas en línea.

Framework

Framework: Un framework es un conjunto de herramientas, bibliotecas y buenas prácticas que facilitan el desarrollo de aplicaciones. Proporciona una estructura base para que los desarrolladores puedan construir software de manera más eficiente. Ejemplos incluyen frameworks web como Django (Python), Laravel (PHP) y Angular (JavaScript).

EVA

EVA (Entorno Virtual de Aprendizaje): Un EVA, también conocido como Ambiente Virtual de Aprendizaje, es una plataforma digital diseñada para facilitar la enseñanza y el aprendizaje en línea. Incluye herramientas como foros, evaluaciones, recursos educativos y comunicación entre estudiantes y profesores. Ejemplos de EVA son Moodle y Blackboard.

Métodos HTTP principales:

1. **GET**: Recupera datos de un servidor.
2. **POST**: Envía datos al servidor para crear un recurso.
3. **PUT**: Actualiza o reemplaza un recurso existente.
4. **DELETE**: Elimina un recurso en el servidor.
5. **PATCH**: Modifica parcialmente un recurso.

Ejemplos con una API de prueba (https://jsonplaceholder.typicode.com):

1. GET

- **Descripción**: Recuperar una lista de publicaciones.
- **Comando**:

curl -X GET https://jsonplaceholder.typicode.com/posts

2. POST

- **Descripción**: Crear una nueva publicación.
- **Comando**:

```
1. curl -X POST https://jsonplaceholder.typicode.com/posts -H "Content-Type:
application/json" -d “{\"title\": \"Nuevo título\", \"body\": \"Contenido de prueba\",
\"userId\": 1}”
2.
```

3. PUT

- **Descripción**: Actualizar una publicación existente.
- **Comando**:

```
1. curl -X PUT https://jsonplaceholder.typicode.com/posts/1 -H "Content-Type:
application/json" -d “{\"title\": \"Título actualizado\", \"body\": \"Contenido
actualizado\", \"userId\": 1}”
```

```
2.
```

4. DELETE

- **Descripción**: Eliminar una publicación.
- **Comando**:

```
1. curl -X DELETE https://jsonplaceholder.typicode.com/posts/1
2.
```

5. PATCH

- **Descripción**: Modificar parcialmente una publicación.
- **Comando**:

```
1. curl -X PATCH https://jsonplaceholder.typicode.com/posts/1 -H "Content-Type:
application/json" -d "{\"title\": \"Título modificado\"}"
2.
```

Notas importantes:

- `-X`: Especifica el método HTTP.
- `-H`: Define los encabezados (headers).
- `-d`: Envía datos en el cuerpo de la solicitud.
- Asegúrate de reemplazar las URLs y datos con los de tu API de prueba.

EVOLUCIÓN DE LA WEB

La evolución de la web ha sido un viaje fascinante que ha transformado la manera en que interactuamos con la tecnología y la información. Aquí te presento un resumen de las principales etapas:

Web 1.0: La web estática (1991-2003)

- Características: Páginas simples y estáticas, con contenido unidireccional. Los usuarios eran consumidores pasivos de información.
- Ejemplo: Sitios informativos y corporativos sin interacción.
- Tecnología clave: HTML y navegadores básicos.

Web 2.0: La web social (2004-presente)

- Características: Introducción de la interactividad y colaboración. Los usuarios pueden crear contenido (blogs, redes sociales).
- Ejemplo: Facebook, YouTube, Wikipedia.
- Tecnología clave: AJAX, JavaScript y plataformas colaborativas.

Web 3.0: La web semántica (en desarrollo)

- Características: Uso de inteligencia artificial, adaptable a cualquier dispositivo, uso de datos semánticos, computación en la nube y datos estructurados para personalizar la experiencia del usuario.
- Ejemplo: Asistentes virtuales como Siri y Alexa.
- Tecnología clave: RDF, OWL y blockchain.

Web 4.0: La web inteligente (futuro emergente)

- Características: Conectividad total, Internet de las cosas (IoT) e inteligencia artificial avanzada.
- Ejemplo: Dispositivos interconectados y aprendizaje automático.
- Tecnología clave: IoT, IA y redes 5G.

Hosting

Un **hosting**, o alojamiento web, es un servicio que permite que un sitio web esté disponible en Internet. Básicamente, cuando contratas un hosting, estás alquilando un espacio en un servidor donde se almacenan todos los archivos y datos de tu sitio web, como imágenes, textos y bases de datos.

Cómo funciona:

1. **Servidor:** Es una computadora que está siempre activa para que tu sitio esté disponible las 24 horas.
2. **Proveedor de hosting:** Se encarga de mantener el servidor, aplicar medidas de seguridad y garantizar que los datos se transfieran correctamente a los navegadores de los visitantes.
3. **Dominio:** Cuando alguien escribe tu dominio (por ejemplo, www.tusitio.com), el servidor envía los archivos necesarios para mostrar tu página.

Tipos de hosting:

- **Hosting compartido:** Varios sitios web comparten el mismo servidor.
- **Hosting VPS:** Un servidor virtual privado con más recursos dedicados.
- **Hosting dedicado:** Un servidor completo para un solo sitio web.
- **Cloud hosting:** Utiliza múltiples servidores en la nube para mayor flexibilidad.

CSS

Selectores de etiqueta

En CSS, los selectores son herramientas que permiten aplicar estilos a elementos HTML específicos. Aquí tienes los principales tipos de selectores que puedes usar:

1. Selector de etiquetas (o tipo)

- Aplica estilos a todas las etiquetas de un tipo específico.
- Ejemplo:

  css

```
1. p {
2.   color: blue;
3. }
4.
```

Esto aplicará el color azul a todos los párrafos `<p>`.

2. Selector de clase

- Aplica estilos a elementos con una clase específica.
- Ejemplo:

```
1. css
2. .miClase {
3.   font-size: 16px;
4. }
5.
```

Esto afectará a cualquier elemento con `class="miClase"`.

3. Selector de ID

- Aplica estilos a un elemento con un ID único.
- Ejemplo:

```
1. css
2. #miID {
3.   background-color: yellow;
4. }
5.
```

Esto afectará al elemento con `id="miID"`.

4. Selector universal

- Aplica estilos a todos los elementos de la página.
- Ejemplo:

```
1. css
2. * {
3.   margin: 0;
4.   padding: 0;
5. }
6.
```

5. Selector de descendientes

- Aplica estilos a elementos dentro de un contenedor específico.
- Ejemplo:

```
1. css
2. div p {
3.   color: green;
4. }
5.
```

Esto afectará a los párrafos `<p>` *dentro de un* `<div>`.

6. Selector de hijos directos

- Aplica estilos solo a los hijos directos de un elemento.
- Ejemplo:

```
1. css
2. ul > li {
3.   list-style-type: square;
4. }
5.
```

7. Selector de atributos

- Aplica estilos a elementos con un atributo específico.
- Ejemplo:

```
1. css
2. input[type="text"] {
3.   border: 1px solid black;
4. }
5.
```

8. Selectores avanzados

- **Selector de pseudo-clases:** Aplica estilos según el estado del elemento.

  css

```
1. a:hover {
2.   color: red;
3. }
4.
```

- **Selector de pseudo-elementos:** Aplica estilos a partes específicas de un elemento.

  css

```
1. p::first-line {
2.   font-weight: bold;
3. }
4.
```

Estos selectores te permiten personalizar el diseño de tu página de manera precisa y eficiente.

Bootstrap

Bootstrap es un framework de código abierto diseñado para **facilitar el desarrollo de sitios web y aplicaciones responsivas. Proporciona una colección de herramientas predefinidas, como estilos CSS, componentes HTML y funciones JavaScript, que permiten crear diseños modernos y adaptables sin necesidad de escribir todo desde cero.** Es especialmente útil para garantizar que los sitios web se vean bien en diferentes dispositivos y tamaños de pantalla.

Breakpoints

Breakpoints, en el contexto de Bootstrap y diseño web, son puntos específicos definidos en el ancho de la pantalla donde el diseño de la página cambia o "se adapta" para ofrecer una mejor experiencia al usuario. Estos puntos se implementan mediante **media queries** en CSS. Bootstrap incluye breakpoints predefinidos que corresponden a tamaños comunes de dispositivos, como teléfonos, tabletas y computadoras de escritorio.

Breakpoints en Bootstrap (Bootstrap 5):

1. **Extra pequeño (xs):** Menos de 576px.
2. **Pequeño (sm):** 576px o más.
3. **Mediano (md):** 768px o más.
4. **Grande (lg):** 992px o más.
5. **Extra grande (xl):** 1200px o más.
6. **Extra extra grande (xxl):** 1400px o más.

Por ejemplo, puedes usar clases como `.col-sm-6` para definir que un elemento ocupe 6 columnas en pantallas pequeñas (576px o más) y `.col-lg-4` para que ocupe 4 columnas en pantallas grandes (992px o más).

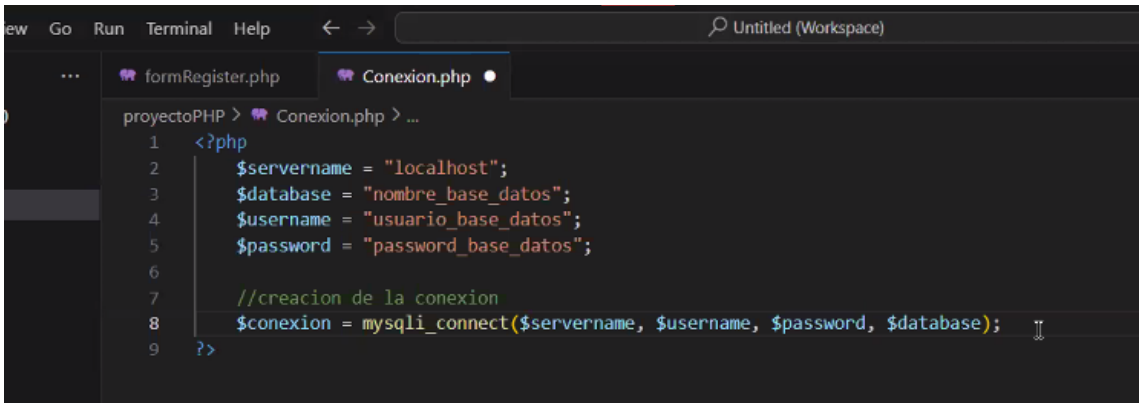
HTML

Los **atributos globales** en HTML son aquellos que pueden aplicarse a cualquier elemento HTML, independientemente de su tipo. Aquí tienes una lista de los principales atributos globales:

1. `class`: Define una o más clases para el elemento, que pueden ser utilizadas por CSS o JavaScript.
2. `id`: Proporciona un identificador único para el elemento.
3. `style`: Permite aplicar estilos CSS directamente al elemento.
4. `title`: Proporciona información adicional sobre el elemento, que suele mostrarse como un tooltip al pasar el cursor.
5. `lang`: Especifica el idioma del contenido del elemento.
6. `dir`: Define la dirección del texto (`ltr` para izquierda a derecha, `rtl` para derecha a izquierda).
7. `hidden`: Indica que el elemento no es relevante y debe estar oculto.
8. `accesskey`: Define una tecla de acceso rápido para interactuar con el elemento.
9. `draggable`: Especifica si el elemento puede ser arrastrado.
10. `spellcheck`: Indica si el contenido del elemento debe ser revisado ortográficamente.
11. `tabindex`: Define el orden de tabulación del elemento.
12. `translate`: Indica si el contenido del elemento debe ser traducido.

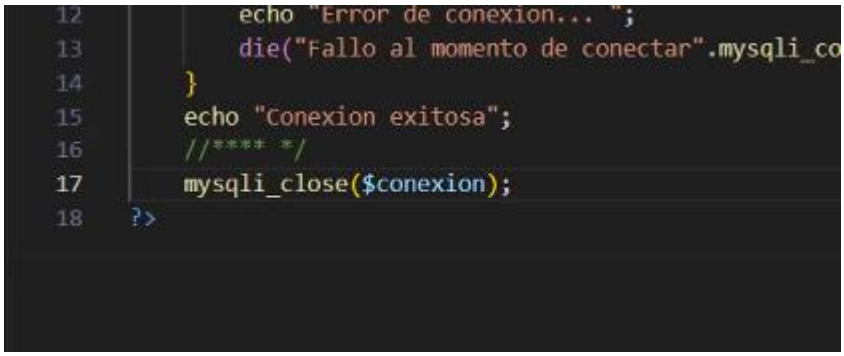
Estos atributos son útiles para mejorar la accesibilidad, personalización y funcionalidad de los elementos HTML.

Conexión a base de datos



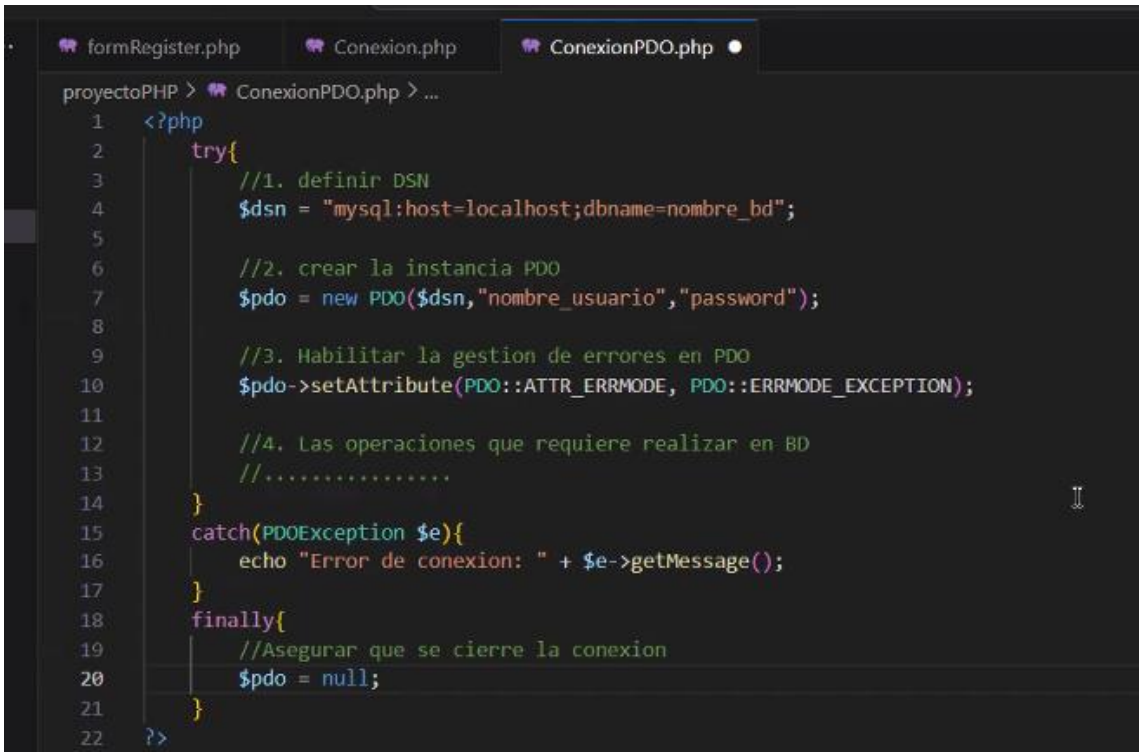
Como conectarse a una base de datos.

Esto Cierra todo cuando no hay conexión, como lanzar una excepción.



Esto cierra si hubo conexión.

Conexión a base de datos usando PDO



Forma de conectar a una base de datos usando PDO, con un bloque try-catch en php.

Eficiencia en la Conexión de Datos

La eficiencia de mantener una conexión abierta versus abrir y cerrar por cada requerimiento depende del contexto y de los requisitos de tu aplicación. Aquí hay un análisis detallado:

1. Mantener la conexión abierta (Connection Pooling)

- **Ventajas:**
    - **Menor overhead:** Evita el costo de abrir y cerrar conexiones repetidamente.
    - **Mejor rendimiento en aplicaciones con alta concurrencia:** Ideal para sistemas con muchas solicitudes frecuentes (ej. APIs, aplicaciones web).

- **Reutilización de conexiones:** Las conexiones se administran mediante un *pool* (grupo de conexiones listas para usarse).

- **Desventajas:**
  - **Consumo de recursos:** Cada conexión ocupa memoria en el servidor de la base de datos.
  - **Riesgo de conexiones inactivas:** Si no se gestiona bien, pueden acumularse conexiones sin uso.
  - **Problemas en entornos con límites estrictos:** Algunos sistemas (como MySQL) tienen un límite máximo de conexiones.

2. Abrir y cerrar por cada requerimiento

- **Ventajas:**
  - **Ahorro de recursos:** Cada conexión se libera inmediatamente después de usarse.
  - **Evita problemas de conexiones "zombies":** Útil en aplicaciones con bajo tráfico o ejecuciones esporádicas (ej. scripts).

- **Desventajas:**
  - **Alto overhead:** Abrir y cerrar conexiones consume tiempo y recursos (especialmente en bases de datos remotas).
  - **Lentitud en alta concurrencia:** Cada operación requiere un *handshake* nuevo con la BD.

Recomendación general

- **Usa Connection Pooling** en la mayoría de los casos, especialmente en:
  - Aplicaciones web o APIs con múltiples solicitudes por segundo.
  - Sistemas donde la latencia de abrir/cerrar conexiones afecta el rendimiento.
- **Abre y cierra manualmente** solo en:
  - Scripts puntuales o procesos batch que se ejecutan ocasionalmente.

- Entornos con restricciones de memoria o conexiones limitadas.

Ejemplo de configuración típica

- **Pooling en Node.js (con** mysql2 **o** pg**):**

```
javascript

Copy

const pool = require('mysql2').createPool({
  host: 'localhost',
  user: 'user',
  database: 'test',
  waitForConnections: true,
  connectionLimit: 10, // Número máximo de conexiones en el pool
});


// Uso:
pool.query('SELECT * FROM users', (err, results) => {
  // La conexión se devuelve automáticamente al pool.
});
```

- **Sin Pooling (no recomendado para producción):**

```
javascript

Copy

const connection = require('mysql2').createConnection({ /* config */ });
connection.query('SELECT * FROM users', (err, results) => {
  connection.end(); // Cierra la conexión manualmente (ineficiente si hay muchas queries).
});
```

Conclusión

La opción más eficiente en la mayoría de los casos es usar un *pool* de conexiones, ya que balancea el rendimiento y el uso de recursos. Solo considera abrir/cerrar manualmente en escenarios muy específicos de bajo tráfico.

Tarea Virtual 6: Framework Laravel-

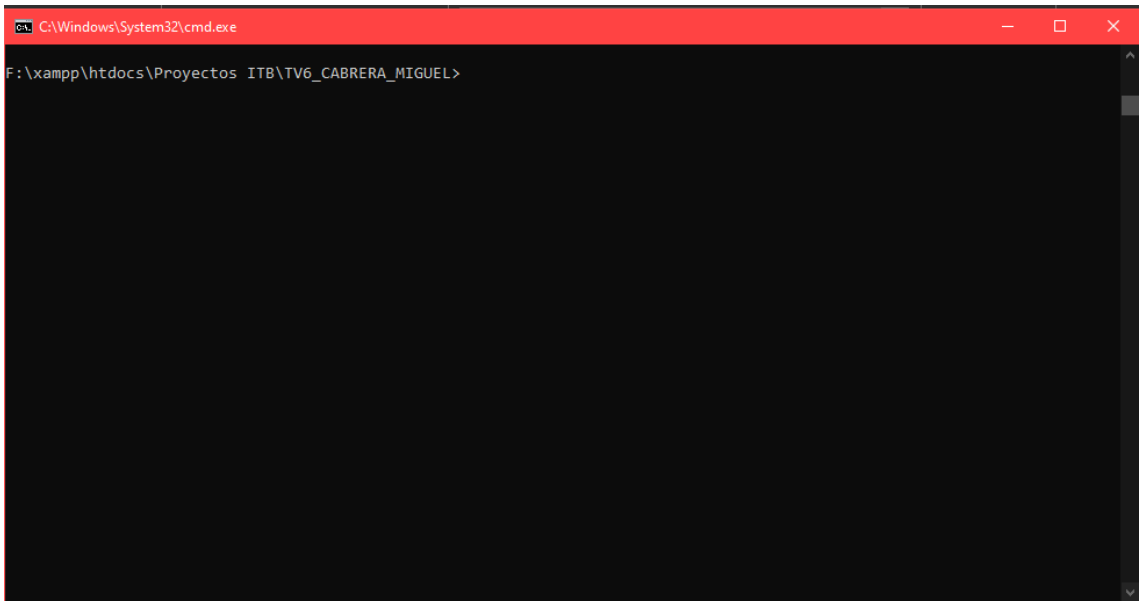
Explicación y Guía

Esta tarea está diseñada para enseñarte los conceptos fundamentales del framework Laravel, que es uno de los frameworks PHP más populares para desarrollo web. A través de esta práctica, aprenderás:

1. Rutas básicas en Laravel: Cómo definir rutas y manejar parámetros.
2. Vistas y plantillas: Cómo crear vistas y usar layouts (plantillas maestras) para reutilizar código.
3. Controladores: Cómo crear y usar controladores básicos y de tipo Resource.
4. Validación de datos: Cómo validar parámetros en las rutas.
5. Formularios y bases de datos: Cómo procesar formularios y guardar datos en la base de datos.

Pasos para realizar la tarea:

1. Crear el proyecto Laravel desde la ruta de la carpeta de tu tarea, ejemplo:




```bash
composer create-project laravel/laravel app-laravel2
cd app-laravel2

```

2. Configurar la base de datos

- Editar el archivo `.env` (que se encuentra en la carpeta app-laravel2) y configurar la conexión a tu base de datos.

```
# DB_CONNECTION=sqlite
# DB_HOST=127.0.0.1
# DB_PORT=3306
# DB_DATABASE=laravel
# DB_USERNAME=root
# DB_PASSWORD=

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=app_laravel2  # Nombre de la BD que crearás en phpMyAdmin
DB_USERNAME=root          # Usuario por defecto de XAMPP
DB_PASSWORD=              # Contraseña (vacía por defecto en XAMPP)

```

3. Crear las rutas (routes/web.php)

```php
// Ruta /movie con parámetro opcional
Route::get('/movie/{titulo?}', function ($titulo = null) {
    return view('movie', ['titulo' => $titulo]);
```

```
})->where('titulo', '.*');

```

// Ruta /series con parámetro obligatorio (solo minúsculas a-z)

```
Route::get('/series/{nombre}', function ($nombre) {
    return view('series', ['nombre' => $nombre]);
})->where('nombre', '[a-z]+');

```

```
// Ruta para el controlador NoticiasController
Route::get('/noticias', [NoticiasController::class, 'index']);

// Ruta resource para UsuarioController
Route::resource('usuarios', UsuarioController::class);

```

```

# 4. Crear el layout maestro (resources/views/_plantillamaster.blade.php)

```
```html
<!DOCTYPE html>
<html>
<head>
    <title>@yield('title')</title>
</head>
<body>
    @yield('content')
</body>
</html>
```

```

# 5. Crear las vistas:

```
- movie.blade.php:
```html
@extends('_plantillamaster')

@section('title', 'Película')
@section('content')
    @if(isset($titulo))
        <h1>Película seleccionada: {{ $titulo }}</h1>
    @else
        <h1>No hay película seleccionada</h1>
    @endif
@endsection
```

```

```
- series.blade.php:
```html
@extends('_plantillamaster')

@section('title', 'Series')
```

```
@section('content')
    <h1>Serie: {{ $nombre }}</h1>
@endsection
```

```
- noticias.blade.php:
```html
@extends('_plantillamaster')

@section('title', 'Noticias')
@section('content')
    <h1>Página de Noticias</h1>
@endsection
```

6. Crear los controladores:

```
```bash
php artisan make:controller NoticiasController
php artisan make:controller UsuarioController --resource
```

```
- NoticiasController.php:
```php
public function index()
{
    return view('noticias');
}
```

7. Crear el formulario de suscripción:

- Primero, crear la migración:

```
```bash
php artisan make:migration create_suscripciones_table
```

- Editar la migración:

```
```php
Schema::create('suscripciones', function (Blueprint $table) {
    $table->id();
    $table->string('nombre');
    $table->string('email');
    $table->timestamps();
});
```
```

- Ejecutar la migración:

```
1. ```bash
2. php artisan migrate
3. ```
4.
```

- Crear modelo:

```
1. ```bash
2. php artisan make:model Suscripcion
3. ```
4.
```

- Crear ruta [1] y controlador para el formulario:

```
1. ```php
2. use App\Http\Controllers\NoticiasController;
3. use App\Http\Controllers\UsuarioController;
4. use App\Http\Controllers\SuscripcionController;
5. // En routes/web.php
6. Route::view('/suscribirse', 'suscribirse');
7. Route::post('/suscribirse', [SuscripcionController::class, 'store']);
8.
```

- Crear vista `suscribirse.blade.php`:

```
 1. ```html
 2. @extends('_plantillamaster')
 3.
 4. @section('title', 'Suscribirse')
 5. @section('content')
 6.     <form method="POST" action="/suscribirse">
 7.         @csrf
 8.         <input type="text" name="nombre" placeholder="Nombre">
 9.         <input type="email" name="email" placeholder="Email">
10.         <button type="submit">Suscribirse</button>
11.     </form>
12. @endsection
13. ```
14.
```

- Crear controlador para manejar el formulario:

```
1. ```bash
2. php artisan make:controller SuscripcionController
3. ```
4.
```

[1] Con este comando “php artisan route:list”, sin comillas, se puede ver las rutas creadas

- Método store en SuscripcionController:

```
```php
use App\Models\Suscripcion; //IMPORTANTE IMPORTAR SUSCRIPCION
public function store(Request $request)
{
    $validated = $request->validate([
        'nombre' => 'required|max:255',
        'email' => 'required|email|unique:suscripciones',
    ]);

    Suscripcion::create($validated);

    return redirect('/')->with('success', '¡Te has suscrito exitosamente!');
}
```
```

8. Probar la aplicación

```
```bash
php artisan serve
```
```

Estructura final del proyecto:

```

app-laravel2/

├── app/

|   ├── Http/

|   |   ├── Controllers/

|   |   |   ├── NoticiasController.php

|   |   |   ├── UsuarioController.php

|   |   |   └── SuscripcionController.php

|   ├── Models/

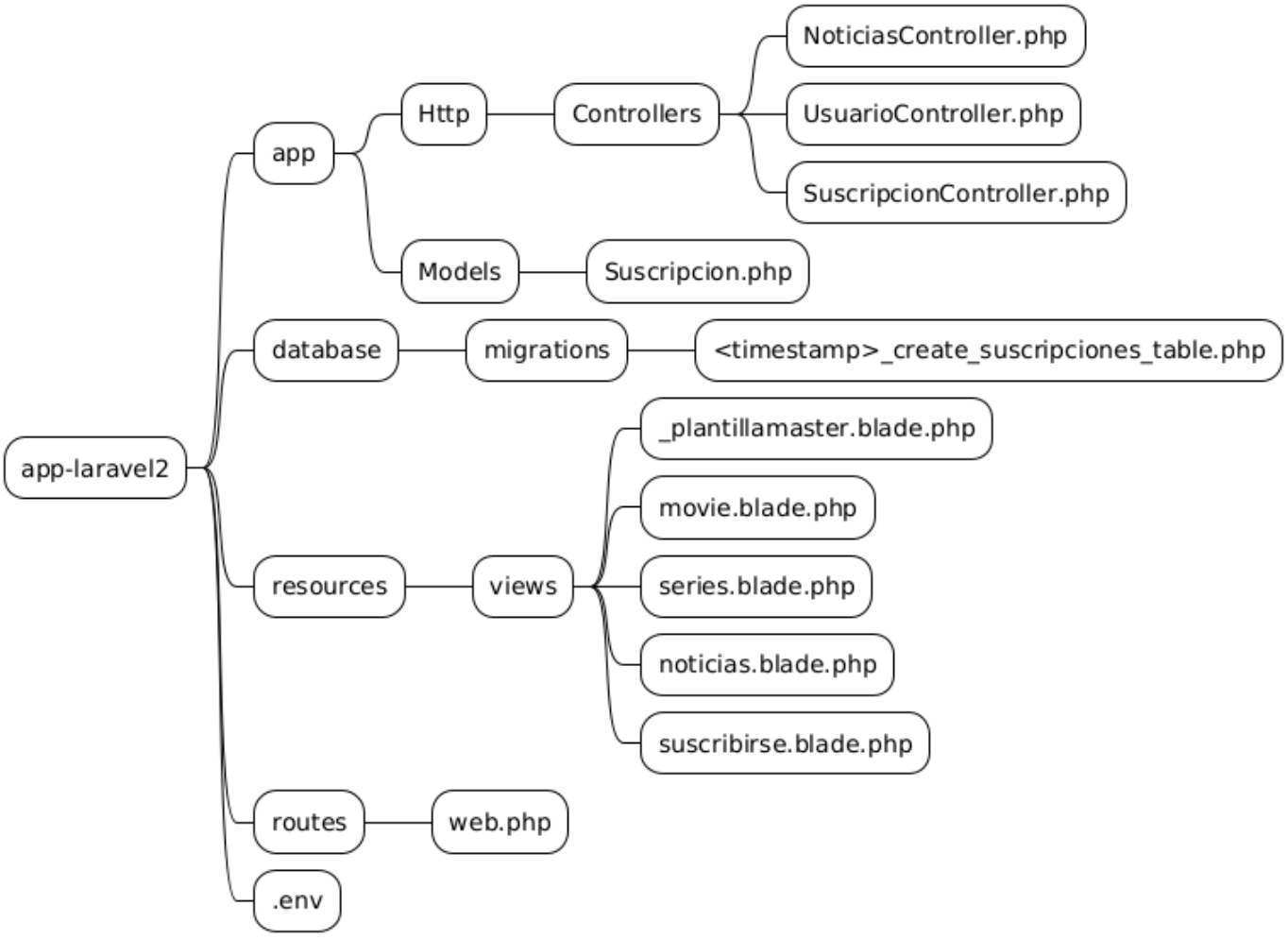
|   |   └── Suscripcion.php

├── database/

|   ├── migrations/

|   |   └── [timestamp]_create_suscripciones_table.php
```

```
├── resources/
|   ├── views/
|   |   ├── _plantillamaster.blade.php
|   |   ├── movie.blade.php
|   |   ├── series.blade.php
|   |   ├── noticias.blade.php
|   |   └── suscribirse.blade.php
|   ├── routes/
|   |   └── web.php
|   └── .env
```



Esta tarea te da una introducción práctica a los componentes esenciales de Laravel que usarás en cualquier proyecto real.

Para volver a crear este proyecto es necesario borrar la base de datos “app-laravel2” de la base de datos de xampp en http://localhost/phpmyadmin

Problemas y Soluciones:

1. No funciona ruta /suscribirse

Suele ser un error comun no importar las clases en web.php

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\NoticiasController;
use App\Http\Controllers\UsuarioController;
use App\Http\Controllers\SuscripcionController;

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

// Ruta /movie con parámetro opcional
Route::get('/movie/{titulo?}', function ($titulo = null) {
    return view('movie', ['titulo' => $titulo]);
})->where('titulo', '.*');

// Ruta /series con parámetro obligatorio (solo minúsculas a-z)
Route::get('/series/{nombre}', function ($nombre) {
    return view('series', ['nombre' => $nombre]);
})->where('nombre', '[a-z]+');

// Ruta para el controlador NoticiasController
Route::get('/noticias', [NoticiasController::class, 'index']);

// Ruta resource para UsuarioController
Route::resource('usuarios', UsuarioController::class);

Route::view('/suscribirse', 'suscribirse');
Route::post('/suscribirse', [SuscripcionController::class, 'store']);

```

2. No se guarda nada en la tabla.

Para solucionar el problema cuando **no se guardan los datos en la tabla** suscripciones, sigue estos pasos clave:

🔍 **Causas comunes y soluciones**

1. **Error en el nombre de la tabla o modelo**

- **Verifica** que el modelo Suscripcion esté asociado a la tabla correcta:

php

Copy

```
// En app/Models/Suscripcion.php

protected $table = 'suscripciones'; // Asegúrate de que coincida con tu migración
```

- **Confirma** que la migración tenga el nombre correcto:

php

Copy

```
Schema::create('suscripciones', function (...) {...}); // Debe ser el mismo nombre
```

2. **Problema con** $fillable **en el modelo**

- El modelo debe permitir la asignación masiva de los campos:

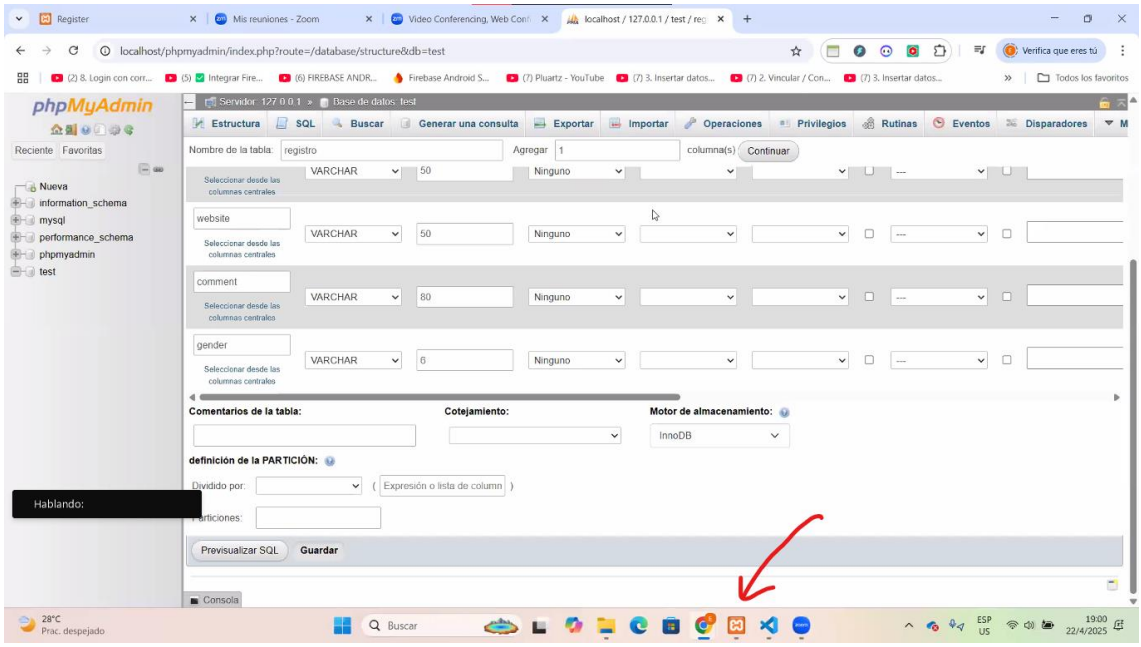
php

Copy

```
// En app/Models/Suscripcion.php

protected $fillable = ['nombre', 'email'];
```

Crear tabla en la base de datos de MYSQL



Desde xampp en ADMIN de la base de datos MYSQL podemos entrar al navegador de la base de datos y poder incluir múltiples campos.

TIPOS DE DESARROLLO WEB

Una aplicación web puede clasificarse como estática o dinámica dependiendo de cómo interactúa con los usuarios y cómo se genera su contenido.

Aplicación web estática:

- Es una web sencilla que presenta contenido fijo y predeterminado.
- No interactúa activamente con el usuario ni permite personalización del contenido.
- Ejemplo: Una página informativa con textos e imágenes (como una página de presentación de empresa).

- El contenido se guarda directamente en el servidor y se entrega tal cual al navegador.

Aplicación web dinámica:

- Permite interacción activa entre el usuario y el contenido.
- Se basa en bases de datos y scripts del servidor para generar contenido adaptado a cada usuario.
- Ejemplo: Redes sociales, sitios de comercio electrónico, blogs que permiten comentarios y actualizaciones.
- Utiliza tecnologías como PHP, Python o JavaScript para crear contenido según las acciones del usuario.

APÉNDICE

Layouts

En Laravel, los layouts son archivos Blade que funcionan como plantillas maestras para tus vistas. Son especialmente útiles para reutilizar elementos comunes en diferentes páginas de tu aplicación, como encabezados, menús, pies de página, entre otros.

Cómo funcionan:

Creas un layout maestro: Este archivo incluye la estructura general (HTML) y se define utilizando áreas dinámicas o secciones con @yield para que las vistas específicas puedan insertar contenido.

Ejemplo de layout:

```
blade
<!DOCTYPE html>
<html>
<head>
    <title>@yield('title')</title>
</head>
<body>
    <header>Encabezado común</header>
    <div class="content">
        @yield('content') <!-- Aquí va el contenido de cada vista -->
    </div>
    <footer>Pie de página común</footer>
</body>
</html>

```

Extiendes el layout: En cada vista, usas @extends para heredar la estructura del layout maestro, y defines el contenido para las secciones con @section.

Ejemplo de vista extendiendo el layout:

```
blade
@extends('_plantillamaster') <!-- Indica el archivo de layout -->
@section('title', 'Página de Noticias')

@section('content')
    <h1>Noticias Recientes</h1>
    <p>Esto es el contenido de la página de noticias.</p>
@endsection

```

Ventajas de usar layouts:

- Mantienes el código más limpio y organizado.
- Facilitas el mantenimiento, ya que los elementos comunes se modifican solo en el archivo maestro.

- Ayuda a crear aplicaciones consistentes en diseño.