Nombre: Ana Vertedor **Fecha:** *30/10/2025*

Ciclo: *Desarrollo de Aplicaciones Web* / ***Asignatura:*** *Desarrollo Web en Entorno Servidor*

UNIDAD 1 - CASO 1:
ARQUITECTURA DE TRES CAPAS EN UN SERVIDOR WEB

1. OBJETIVO

El objetivo de este ejercicio es comprender y aplicar la arquitectura web en tres capas mediante la creación de una aplicación funcional desarrollada con HTML, CSS, JavaScript y PHP.
Se busca demostrar cómo el flujo de datos se gestiona entre las distintas capas de la aplicación:

- Capa de presentación: interfaz visual que permite al usuario interactuar con la aplicación.
- Capa de negocio: lógica intermedia que procesa las solicitudes del cliente y gestiona la comunicación con los datos.
- Capa de datos: responsable de almacenar o simular la información que será consultada desde el servidor.

A través de este ejercicio, pretendo consolidar el concepto de separación de responsabilidades dentro de una aplicación web y evidenciar cómo cada capa cumple una función específica dentro del proceso de respuesta de un servidor web.

2. DESCRIPCIÓN GENERAL DEL PROYECTO

La aplicación desarrollada consiste en un catálogo de productos interactivo que permite realizar búsquedas dinámicas a través de un formulario web. Su estructura se basa en la arquitectura web en tres capas, que separa la presentación, la lógica del negocio y los datos, facilitando la comprensión y mantenimiento del código.
Para la capa de presentación, he utilizado una página web diseñada previamente, creada con fines de práctica y experimentación. Este diseño lo he adaptado y ampliado para integrar el buscador del catálogo, manteniendo el estilo visual y coherencia estética. La página incorpora HTML, CSS y JavaScript, y utiliza la función *fetch()* para enviar solicitudes al servidor sin recargar la página.
En la capa de negocio, implementada en PHP, se gestionan las peticiones del cliente. Esta capa recibe los términos de búsqueda, procesa la solicitud y devuelve una respuesta en formato JSON, lo que permite al navegador interpretar fácilmente la información y mostrar los resultados en pantalla.
La capa de datos está compuesta por clases PHP que simulan una base de datos mediante un conjunto de productos predefinidos. Aunque no se utiliza un gestor real (como MySQL), esta aproximación permite demostrar el funcionamiento del flujo completo entre cliente, servidor y datos, manteniendo la separación lógica entre capas.
En conjunto, el proyecto demuestra la arquitectura en tres capas utilizando una página base propia como entorno de pruebas para la integración en PHP.

3. ESTRUCTURA DEL PROYECTO

La siguiente imagen muestra la estructura de carpetas y archivos utilizada en el proyecto. Cada nivel refleja la división lógica en tres capas (presentación, negocio y datos), organizadas dentro del directorio principal *dwes/ud1_caso1/*.

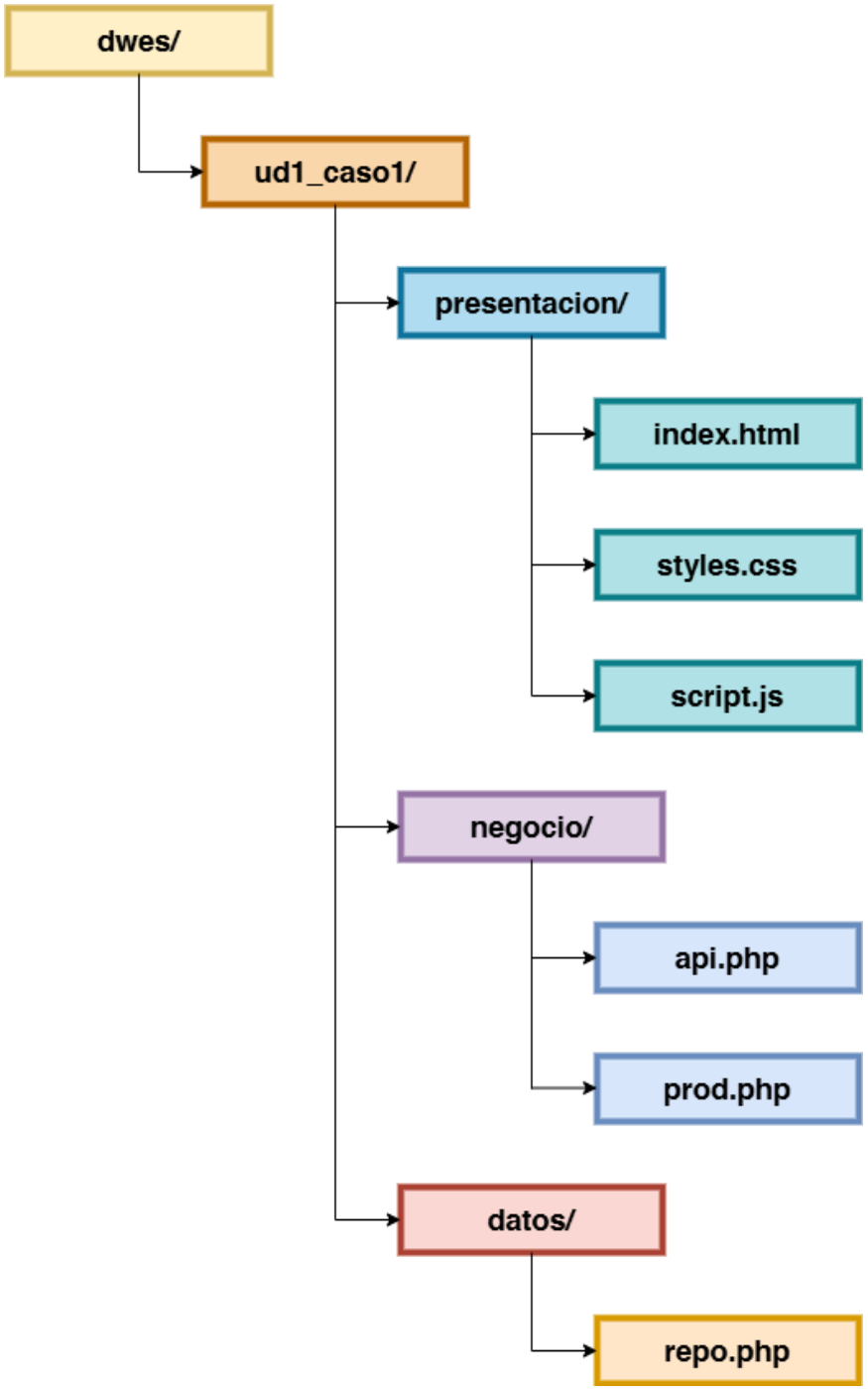

Fig. 1 - Diagrama de la estructura de las carpetas y archivos del proyecto.

- **presentacion/** → contiene la interfaz y la lógica del cliente.
- **negocio/** → procesa las peticiones y devuelve datos.
- **datos/** → simula la base de datos mediante PHP.

4. DIAGRAMA DE ARQUITECTURA

En el siguiente esquema se puede ver cómo se estructura la aplicación en tres capas y cómo se comunican entre sí. Cada capa tiene una función específica dentro del proceso: la presentación envía las solicitudes, la lógica de negocio las procesa y la capa de datos devuelve la información simulada.

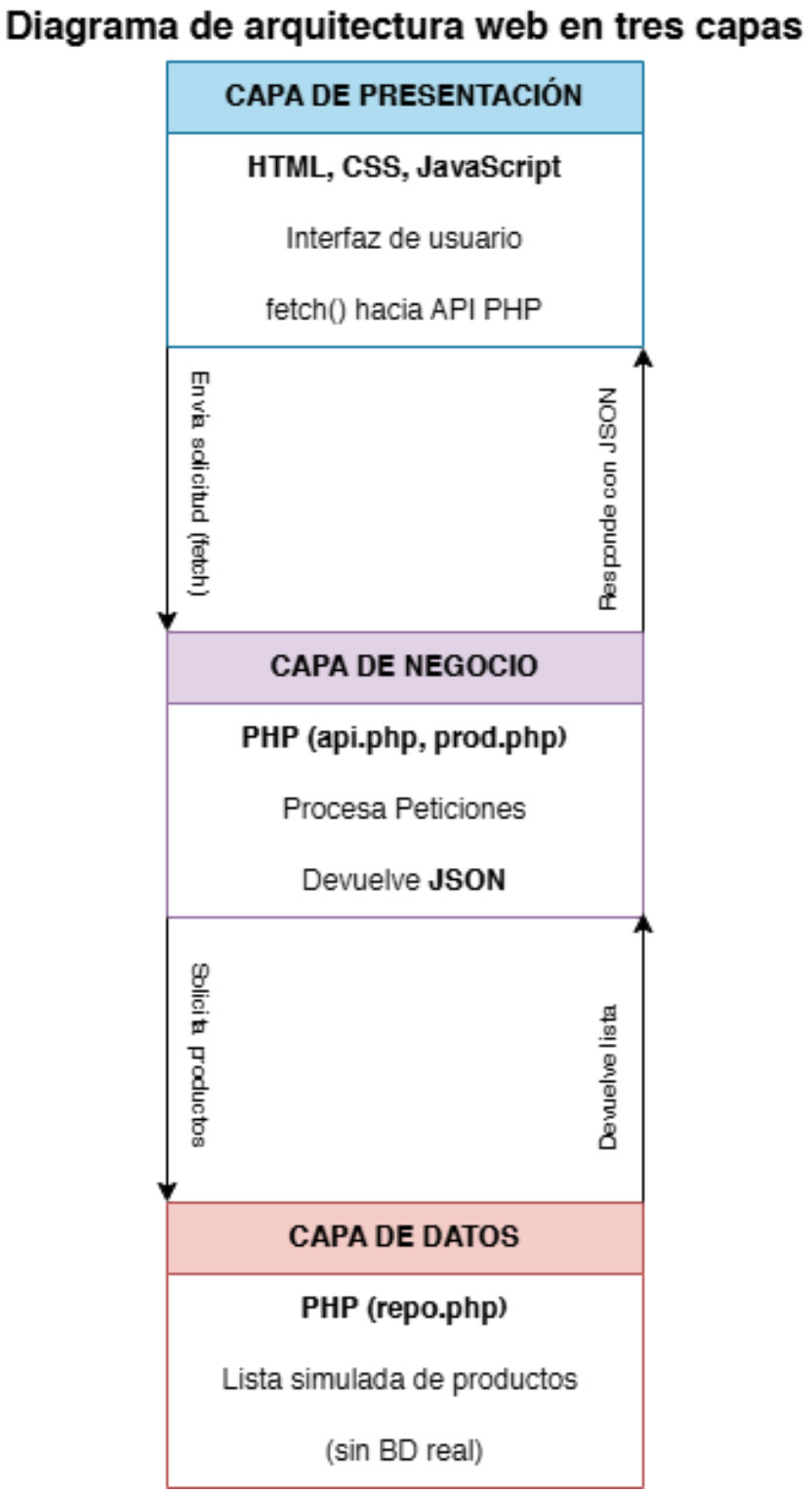

Fig. 2 - Diagrama de arquitectura

5. EJEMPLOS DE CÓDIGO

A continuación, se muestran fragmentos representativos del código desarrollado para cada una de las capas del proyecto.
Estos ejemplos ilustran cómo interactúan entre sí los distintos componentes: la interfaz de usuario realiza peticiones mediante JavaScript, el servidor las gestiona con PHP en la capa de negocio, y los datos se obtienen desde una fuente simulada en la capa de datos.

Ejemplo 1 - llamada asincrónica desde JavaScript:

Ejemplo de función buscar() de la capa de presentación (JavaScript). Gestiona la petición hacia la API PHP, muestra el estado de la búsqueda y renderiza los resultados en pantalla.

```
async function buscar() {
  $estado.textContent = 'Buscando...';
  $ul.innerHTML = '';
  const term = encodeURIComponent(($search.value || '').trim());
  try {
    const res = await fetch(`../negocio/api.php?search=${term}`);
    if (!res.ok) throw new Error('Error en la API');
    const data = await res.json();
    if (!Array.isArray(data) || data.length === 0) {
      $estado.textContent = 'Sin resultados.';
      return;
    }
    data.forEach(p => {
      const li = document.createElement('li');
      const price = typeof p.price === 'number' ? p.price.toFixed(2) :
p.price;
      li.textContent = `${p.name} — ${p.category} — ${price} €`;
      $ul.appendChild(li);
    });
    $estado.textContent = `${data.length} resultado(s).`;
  } catch (e) {
    console.error(e);
    $estado.textContent = 'Ha ocurrido un error consultando la API.';
  }
}
```

Ejemplo 2 - API PHP (api.php):

Gestiona las peticiones recibidas desde el cliente, consulta los datos del repositorio y devuelve la información en formato JSON para que pueda mostrarse en la interfaz.

```php
<?php
header('Content-Type: application/json; charset=utf-8');
require_once __DIR__.'/../datos/repo.php';
require_once __DIR__.'/prod.php';

$repo = new Repo();
$prod = new Prod($repo);
$search = $_GET['search'] ?? '';

echo json_encode(
  $prod->buscar($search),
  JSON_UNESCAPED_UNICODE | JSON_PRETTY_PRINT
);
```

Ejemplo 3 - Repositorio simulado (repo.php):

Contiene una lista de productos definida en PHP que actúa como base de datos simulada, permitiendo probar la lógica de consulta sin necesidad de un sistema real de almacenamiento.

```php
<?php
class Repo {
  private $data = [
    ['id'=>1,'name'=>'Teclado
mecánico','price'=>59.90,'category'=>'Periféricos'],
    ['id'=>2,'name'=>'Ratón
inalámbrico','price'=>24.50,'category'=>'Periféricos'],
    ['id'=>3,'name'=>'Monitor
24"','price'=>139.00,'category'=>'Pantallas'],
    ['id'=>4,'name'=>'USB-C Hub','price'=>29.95,'category'=>'Accesorios'],
  ];

  public function buscar(string $term): array {
    $term = mb_strtolower(trim($term));
    if ($term === '') return $this->data;
    return array_values(array_filter($this->data, fn($p)=>
      str_contains(mb_strtolower($p['name']),$term) ||
      str_contains(mb_strtolower($p['category']),$term)
    ));
  }
}
```

6. CAPTURAS DE FUNCIONAMIENTO

A continuación se muestran las capturas de la interfaz final de la aplicación.
En ellas puede observarse el flujo básico de funcionamiento: primero, la página inicial del catálogo con el campo de búsqueda, y después el resultado obtenido tras consultar la API.
Estas imágenes ilustran el recorrido completo desde la capa de presentación hasta la respuesta del servidor en formato JSON, ya procesada y mostrada en pantalla.

Fig. 3. Interfaz inicial del catálogo (capa de presentación).

Fig. 4. Resultado mostrado tras la búsqueda de "teclado".

7. BIBLIOGRAFÍA Y FUENTES CONSULTADAS

- *PHP Course for Beginners* - Youtube https://www.youtube.com/watch?v=m52ljs78S24&list=PL0eyrZgxdwhwwQQZA79OzYwl5ewA7HQih
- *How to Fetch data from an API using JavaScript* - Youtube https://www.youtube.com/watch?v=37vxWr0WgQk
- *Learn Fetch API in 6 minutes* - Youtube https://www.youtube.com/watch?v=cuEtnrL9-H0
- Curso de Udemy
- Apuntes personales de 1º
- Diseño de base propia, creado por mí para pruebas de maquetación y testeo de estilo.