UD1 Actividad 1

Andrés López Barrera

Actividad 1: Investigación de frameworks backend

1. Express.js

- **Lenguaje**: JavaScript
- **Año de creación**: 2010
- **Última versión**: 4.18.2 (2022)
- **Popularidad**: Muy popular, especialmente entre startups y proyectos pequeños que buscan rapidez en el desarrollo de APIs y microservicios. Al ser parte del ecosistema de Node.js, es ampliamente utilizado en aplicaciones full-stack junto con frameworks frontend como React o Angular.
- **Facilidad de uso**: Express.js es conocido por su diseño minimalista y no impone una estructura estricta, lo que lo hace altamente flexible. Es fácil para desarrolladores familiarizados con JavaScript y Node.js.
- **Características**:
    - Soporte para middleware que permite manejar las peticiones y respuestas HTTP de manera flexible.
    - Ideal para crear APIs REST y aplicaciones ligeras.
    - Facilita la integración con bases de datos y otros servicios a través de módulos y bibliotecas.
    - Comunidad activa y extensa documentación.
- **Desventajas**: Al ser un framework minimalista, carece de características integradas como ORM o sistemas de autenticación, lo que requiere que los desarrolladores elijan herramientas adicionales(daily.dev)(Back4App Blog).
- **URL**: expressjs.com

2. Django

- **Lenguaje**: Python
- **Año de creación**: 2005
- **Última versión**: 4.2.5 (2023)
- **Popularidad**: Amplia popularidad entre desarrolladores que trabajan con Python, especialmente en proyectos que requieren escalabilidad y seguridad. Usado por grandes empresas como Instagram y Pinterest.
- **Facilidad de uso**: Aunque tiene una curva de aprendizaje moderada, es muy apreciado por su excelente documentación y su enfoque en “no reinventar la rueda”. Facilita el desarrollo rápido de aplicaciones web complejas.
- **Características**:
    - **ORM (Object-Relational Mapping)**: Permite interactuar con bases de datos sin escribir SQL, lo que acelera el desarrollo.
    - **Admin panel**: Un sistema de administración integrado que permite manejar fácilmente usuarios, bases de datos y otros recursos.
    - **Seguridad**: Protección contra ataques comunes como inyecciones SQL, XSS y CSRF. Es una de las razones por las que Django es popular para aplicaciones con altos requisitos de seguridad.
    - **Escalabilidad**: Ideal para aplicaciones que crecen en tamaño y tráfico, con soporte para arquitectura de microservicios(Back4AppBlog)(WeAreDevelopers).
- **URL**: djangoproject.com

3. Laravel

- **Lenguaje**: PHP
- **Año de creación**: 2011
- **Última versión**: 10.1.5 (2023)
- **Popularidad**: Es uno de los frameworks más utilizados en el mundo PHP. Es muy apreciado por su elegancia, simplicidad y productividad, especialmente en desarrollos de aplicaciones con bases de datos complejas.
- **Facilidad de uso**: Laravel tiene una sintaxis expresiva y fácil de entender, lo que permite a los desarrolladores crear aplicaciones rápidamente. Su curva de aprendizaje es moderada, pero ofrece muchas herramientas que simplifican el desarrollo.
- **Características**:
    - **Blade template engine**: Un motor de plantillas sencillo pero potente que facilita la creación de vistas reutilizables.
    - **Eloquent ORM**: Facilita la interacción con bases de datos usando una sintaxis sencilla, lo que permite crear y manejar relaciones entre tablas con facilidad.
    - **Artisan CLI**: Herramienta de línea de comandos que simplifica tareas rutinarias como la creación de controladores, migraciones de base de datos y más.
    - **Seguridad**: Laravel tiene autenticación y autorización integradas, facilitando el manejo de sesiones, roles de usuario y permisos(Back4App Blog)(WeAreDevelopers).
- **URL**: laravel.com

4. Spring Boot

- **Lenguaje**: Java
- **Año de creación**: 2014
- **Última versión**: 3.1.2 (2023)
- **Popularidad**: Es el framework más utilizado en el mundo empresarial para aplicaciones basadas en Java. Netflix, Uber y otras grandes compañías utilizan Spring Boot para sus microservicios y aplicaciones cloud-native.
- **Facilidad de uso**: Aunque puede ser complejo para principiantes, Spring Boot facilita la configuración inicial de aplicaciones Java al eliminar la necesidad de configuraciones manuales extensas. Es ideal para desarrollar microservicios y aplicaciones empresariales.
- **Características**:
    - **Microservicios**: Spring Boot está diseñado para crear aplicaciones modulares y escalables, lo que lo hace perfecto para arquitecturas de microservicios.
    - **Integración con herramientas populares**: Funciona bien con bibliotecas como Hibernate para bases de datos y otros servicios empresariales.
    - **Spring Cloud**: Extensiones que permiten la creación y gestión de aplicaciones en la nube(Radixweb)(WeAreDevelopers).
- **URL**: spring.io

5. Ruby on Rails

- **Lenguaje**: Ruby
- **Año de creación**: 2005
- **Última versión**: 7.0.6 (2023)
- **Popularidad**: Aunque ha perdido algo de terreno frente a otros frameworks, sigue siendo popular en startups que buscan un desarrollo rápido y ágil, gracias a su principio "Convention over Configuration" (Convención sobre Configuración).
- **Facilidad de uso**: Rails es conocido por permitir a los desarrolladores construir aplicaciones rápidamente gracias a su filosofía DRY (Don't Repeat Yourself) y herramientas preinstaladas.
- **Características**:
    - **Active Record**: ORM que simplifica la interacción con bases de datos y permite escribir código Ruby en lugar de SQL.
    - **Scaffolding**: Generación automática de código básico para CRUD.
    - **Action Controller**: Maneja la autenticación y respuestas del servidor de manera eficiente.
    - **Ruby Gems**: Una vasta colección de bibliotecas que extienden sus funcionalidades, facilitando el desarrollo de cualquier tipo de aplicación(WeAreDevelopers).
- **URL**: rubyonrails.org

Actividad 2: Cliente y servidor web

El modelo **cliente-servidor** es una arquitectura fundamental en la web moderna que describe cómo los dispositivos (clientes) interactúan con servicios remotos (servidores) para intercambiar información. En este modelo, el cliente y el servidor tienen roles y funciones claramente diferenciados.

Cliente

- El cliente es el dispositivo o la aplicación que solicita servicios o recursos al servidor. Generalmente, en el contexto de la web, el cliente es un navegador (como Chrome, Firefox, o Safari) que envía solicitudes para acceder a una página web o un recurso.
- **Ejemplos de clientes**: Computadoras, teléfonos móviles, tablets, e incluso otros servidores que actúan como intermediarios.

Servidor

- El servidor es una máquina remota que recibe las solicitudes del cliente, procesa esas solicitudes, accede a los recursos necesarios (por ejemplo, bases de datos o archivos), y luego devuelve una respuesta.
- Los servidores suelen manejar la lógica de negocio, gestionar bases de datos y asegurarse de que el contenido solicitado esté disponible para el cliente.

Interacción entre cliente y servidor (Proceso de petición y respuesta)

El ciclo básico de interacción entre cliente y servidor en el proceso de petición y respuesta incluye los siguientes pasos:

1. **El cliente realiza una solicitud**: Cuando un usuario accede a una página web, el navegador envía una solicitud HTTP al servidor. Esta solicitud puede ser un **HTTP GET** para obtener datos o un **HTTP POST** para enviar información (por ejemplo, en un formulario).
2. **Servidor procesa la solicitud**: El servidor recibe la solicitud, identifica el recurso solicitado (por ejemplo, una página HTML, datos de una API, o una imagen) y ejecuta el código necesario en el backend. Dependiendo del

servidor, este código puede estar escrito en lenguajes como PHP, Python, Java, Node.js, entre otros.

3. **Servidor envía una respuesta**: Después de procesar la solicitud, el servidor envía una respuesta al cliente. Esto suele ser una página web completa en formato HTML, pero también puede ser datos en formato JSON o XML, archivos, u otro tipo de contenido.
4. **El cliente recibe y procesa la respuesta**: El navegador del cliente recibe el contenido HTML, lo renderiza y ejecuta el JavaScript en el lado del cliente (si existe), mostrando la página o interactuando con los datos recibidos.

Característica	Lado del Cliente (JavaScript)	Lado del Servidor (PHP, etc.)
Ubicación de Ejecución	Se ejecuta en el navegador del usuario (cliente).	Se ejecuta en el servidor remoto.
Objetivo Principal	Controlar la interacción del usuario y la presentación.	Procesar lógica de negocio y manejar datos.
Velocidad	Generalmente más rápido para la interacción del usuario.	Más lento, ya que requiere comunicación entre cliente y servidor.
Seguridad	Menos seguro, ya que el código es visible para el usuario.	Más seguro, ya que el usuario no puede acceder al código del servidor.
Acceso a Recursos	Acceso limitado a recursos locales (por ejemplo, almacenamiento en el navegador).	Acceso completo a bases de datos y otros recursos del servidor.
Interactividad	Manipula el DOM directamente, permite actualizaciones en tiempo real sin recargar la página.	No es interactivo directamente; se utiliza para generar contenido y procesar datos.

Caso práctico: Simulación de una solicitud de página web

Paso 1: Solicitud del cliente

1. El usuario escribe la URL `www.ejemplo.com` en su navegador web y presiona Enter.

2. El **navegador (cliente)** genera una **solicitud HTTP GET** para obtener la página. Esta solicitud viaja a través de Internet hacia el **servidor** asociado con el dominio `www.ejemplo.com`.
    - **Solicitud HTTP**:
        - Método: `GET`
        - URL: `www.ejemplo.com`
        - Protocolo: `HTTPS` (si es una conexión segura)

Paso 2: Procesamiento en el servidor

3. El servidor web que aloja `www.ejemplo.com` recibe la solicitud.
    - El servidor podría estar ejecutando software como Apache, Nginx, o un framework backend como Node.js o Laravel (PHP).
4. El servidor analiza la solicitud y ejecuta el código correspondiente en el lado del servidor. En este ejemplo, el servidor está utilizando PHP para generar contenido dinámico.
    - **Código del lado del servidor (PHP)**:
        - El servidor podría ejecutar una consulta a una **base de datos** MySQL para obtener datos (como noticias recientes, productos o comentarios).
        - Genera un archivo **HTML** que será enviado de vuelta al cliente.
    - **Consultas a la base de datos**: El servidor recupera los datos solicitados de la base de datos y los inserta en la página HTML de manera dinámica.
5. Después de procesar la lógica del servidor y acceder a los recursos, el servidor genera una **respuesta HTTP** que contiene:
    - El código **HTML** de la página.
    - Recursos asociados como archivos **CSS** y **JavaScript** para estilos y funcionalidad.
6. La respuesta se envía de vuelta al navegador del cliente.

Paso 3: Renderización en el cliente

6. El navegador recibe la respuesta HTTP del servidor y comienza a procesar el código **HTML**.
    - **HTML**: Estructura y muestra el contenido de la página.
    - **CSS**: Aplica el diseño y los estilos (colores, fuentes, etc.).
    - **JavaScript**: Ejecuta cualquier funcionalidad interactiva (como formularios dinámicos o menús desplegables).
7. Si el código JavaScript incluye solicitudes adicionales (como una llamada AJAX o a una API), el cliente puede hacer nuevas solicitudes sin recargar la página, obteniendo datos de manera dinámica.
    - **Código del lado del cliente (JavaScript)**:
        - Valida formularios.
        - Añade interactividad (menús dinámicos, carruseles de imágenes).
        - Realiza solicitudes adicionales (API) para obtener datos sin recargar la página.

Paso 4: Visualización de la página en el navegador

8. Finalmente, el navegador renderiza la página completa en la pantalla del usuario. Ahora el usuario puede interactuar con la página, como hacer clic en enlaces, llenar formularios o navegar a otras secciones del sitio.

Actividad 3: Generación dinámica de páginas web.

1. ¿Qué es la generación dinámica de páginas web?

La **generación dinámica de páginas web** se refiere a la creación de contenido web en tiempo real, de acuerdo con las solicitudes específicas de los usuarios. A diferencia de las páginas estáticas (que siempre muestran el mismo contenido sin importar quién las visite), una página dinámica **adapta su contenido** según las interacciones del usuario, los datos almacenados en bases de datos, o factores como la fecha y la hora.

En la generación dinámica, el servidor genera una página web **bajo demanda**, usando lenguajes de programación del lado del servidor como PHP, Python, Ruby o Node.js, y puede consultar una base de datos para personalizar el contenido que se envía al navegador.

Cómo funciona:

1. El usuario realiza una solicitud (por ejemplo, al hacer clic en un enlace o enviar un formulario).
2. El servidor recibe la solicitud y ejecuta un código en el backend.
3. El código del servidor genera un documento HTML dinámico, incorporando datos según la solicitud.
4. El servidor envía la página generada al navegador del usuario, donde se muestra.

---

2. Ventajas frente a las páginas estáticas

- **Contenido Personalizado**: Una página dinámica puede generar contenido adaptado al usuario (por ejemplo, mostrar un nombre de usuario, recomendaciones de productos basadas en preferencias, etc.).
- **Actualización en Tiempo Real**: Los datos dinámicos, como las noticias, comentarios o publicaciones de redes sociales, pueden actualizarse constantemente sin requerir que se modifique el código HTML de la página.
- **Interactividad**: Las páginas dinámicas pueden incorporar formularios, autenticación de usuarios, carritos de compra, y otros elementos que requieren interacción entre cliente y servidor.
- **Conexión a Bases de Datos**: Permiten acceder y manipular datos almacenados en bases de datos, lo que es útil para aplicaciones que dependen de gran cantidad de información estructurada (por ejemplo, e-commerce, redes sociales, etc.).
- **Escalabilidad**: Las páginas dinámicas permiten administrar grandes cantidades de contenido de manera eficiente, ya que el contenido puede generarse en función de las solicitudes, en lugar de estar predefinido.

---

3. Ejemplo de una página dinámica en PHP

```php
<?php

$productos = [

    ['nombre' => 'Producto 1', 'precio' => 20],

    ['nombre' => 'Producto 2', 'precio' => 35],

    ['nombre' => 'Producto 3', 'precio' => 15],

];

echo "<h1>Lista de Productos</h1>";

echo "<ul>";

foreach ($productos as $producto) {

      echo "<li>{$producto['nombre']} - Precio: {$producto['precio']}
USD</li>";

}

echo "</ul>";

?>
```

Cómo funciona:

- El script PHP accede a un arreglo de productos (que podría estar en una base de datos).
- Cada vez que el servidor procesa la página, el contenido de la lista es generado dinámicamente, mostrando la información actualizada de los productos.

---

4. Cuándo es más eficiente usar una página dinámica en lugar de una estática

Las páginas dinámicas son más eficientes cuando:

- **El contenido cambia con frecuencia**: Por ejemplo, sitios de noticias, blogs con comentarios, y plataformas de redes sociales, donde los datos se actualizan constantemente.
- **Personalización del contenido**: Cuando necesitas adaptar el contenido según el usuario, como en sitios de comercio electrónico donde los productos recomendados cambian según el historial de compras del usuario.
- **Interacción con bases de datos**: Si el sitio web depende de información que se almacena en bases de datos y varía según la consulta (como los sitios de búsqueda o las tiendas en línea).
- **Autenticación de usuarios**: Para manejar sistemas de login, perfiles, y contenido protegido, es necesario generar páginas dinámicas que respondan a las credenciales de los usuarios.

Cuándo usar una página estática:

- **Contenido fijo**: Para sitios pequeños o portafolios que no requieren actualizaciones constantes.
- **Mejor rendimiento**: Las páginas estáticas suelen cargarse más rápido, ya que no dependen de consultas a bases de datos ni procesamiento en el servidor.

Actividad 4: Diferencias entre servidor web y servidor de aplicaciones.

1. ¿Qué es un servidor web y un servidor de aplicaciones?

Servidor Web:

Un **servidor web** es un software que maneja las solicitudes HTTP y sirve contenido estático como páginas HTML, archivos CSS, imágenes, y documentos. Su función

principal es recibir las solicitudes del navegador del cliente, procesarlas, y devolver los recursos solicitados.

- **Ejemplos comunes**: Apache HTTP Server, Nginx, Microsoft IIS.

El servidor web **no** ejecuta lógica compleja ni interactúa con bases de datos de forma directa; simplemente entrega archivos al cliente. Si necesita generar contenido dinámico, suele pasar la solicitud a una aplicación o a un servidor de aplicaciones.

Servidor de Aplicaciones:

Un **servidor de aplicaciones** es un software que ejecuta aplicaciones empresariales y sirve tanto contenido dinámico como lógico. Va más allá de un servidor web al manejar operaciones de negocio, procesamiento de datos y ejecución de scripts, accediendo a bases de datos, APIs y otros servicios backend.

- **Ejemplos comunes**: Apache Tomcat, GlassFish, JBoss.

Un servidor de aplicaciones puede gestionar todo el flujo de una aplicación web, incluyendo la ejecución de lenguajes de programación (Java, PHP, Python) y la lógica de negocio necesaria para interactuar con bases de datos y otras fuentes de datos.

---

2. Diferencias clave entre Servidor Web y Servidor de Aplicaciones

Aspecto	Servidor Web	Servidor de Aplicaciones
Función principal	Manejar solicitudes HTTP y servir contenido estático.	Ejecutar aplicaciones, manejar lógica de negocio y generar contenido dinámico.
Tipos de contenido	Contenido estático (HTML, CSS, imágenes, etc.).	Contenido dinámico (HTML generado, lógica de negocio, acceso a bases de datos).
Procesamiento	No ejecuta código de servidor, solo entrega archivos.	Ejecuta scripts y programas de servidor (Java, PHP, Python, etc.).
Lenguajes soportados	Maneja archivos estáticos (HTML, CSS, JavaScript).	Soporta varios lenguajes de programación (Java, Python, etc.).
Ejemplos	Apache HTTP Server, Nginx.	Apache Tomcat, GlassFish, JBoss.

Ejemplo de Servidor Web:

- **Nginx**: Es un servidor web de alto rendimiento, diseñado para manejar una gran cantidad de conexiones concurrentes. Sirve principalmente contenido estático como archivos HTML, CSS, y JavaScript.

Ejemplo de Servidor de Aplicaciones:

- **Apache Tomcat**: Es un servidor de aplicaciones diseñado para ejecutar aplicaciones Java, como servlets y JSP (Java Server Pages), generando contenido dinámico a partir de interacciones con bases de datos y otras aplicaciones.

---

3. ¿Por qué este concepto se considera algo obsoleto?

En la actualidad, la distinción entre servidores web y servidores de aplicaciones tiende a diluirse. Muchas tecnologías modernas pueden manejar ambos roles de manera eficiente. Un ejemplo común es **Node.js**, que es capaz de manejar tanto el servidor web como la lógica de la aplicación en un solo entorno.

Además, los **contenedores** y las **arquitecturas basadas en microservicios** han hecho menos relevante la diferenciación tradicional entre estos servidores. Plataformas como **Docker** permiten empaquetar una aplicación completa (con servidor web y lógica de aplicación) en una única unidad que puede ser desplegada de manera independiente. Por otro lado, los frameworks modernos como **Spring Boot** permiten ejecutar aplicaciones con capacidades tanto de servidor web como de servidor de aplicaciones, reduciendo la necesidad de utilizar dos tecnologías distintas.

Actividad 5: Integración de PHP con lenguajes de marcas

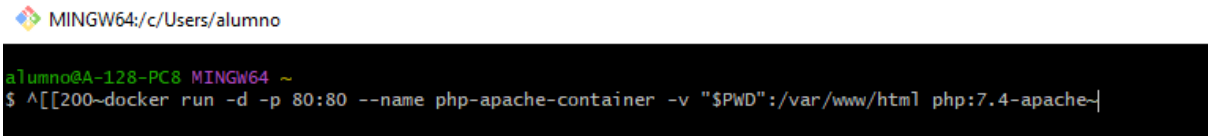
1. Arrancar el servicio PHP usando un entorno como XAMPP o Docker

XAMPP:

1. Descarga e instala XAMPP desde su sitio oficial.
2. Inicia el panel de control de XAMPP y activa los servicios de **Apache** y **MySQL**.
3. Verifica que PHP esté funcionando abriendo tu navegador y accediendo a http://localhost.

Docker:

1. Instala Docker y crea un contenedor con PHP y Apache utilizando la siguiente instrucción:



2. El contenedor servirá archivos PHP en el directorio actual ($PWD) en http://localhost.

2. Crear un archivo PHP llamado info.php

1. Crea un archivo en el directorio raíz del servidor web llamado info.php.
2. Escribe el siguiente código en el archivo:

```php
<?php

phpinfo();

?>
```

3. Guarda el archivo y accede a http://localhost/info.php en el navegador.

3. Verificar y anotar los valores

Al abrir info.php, aparecerá una página con la configuración detallada de PHP. A continuación, verifica y anota los siguientes valores:

- **Versión de PHP**: Indica la versión exacta de PHP que estás utilizando.
- **Loaded Configuration File**: Este campo muestra la ubicación del archivo php.ini cargado, que contiene la configuración de PHP.
- **memory_limit**: El límite de memoria asignado a los scripts PHP.
- **DOCUMENT_ROOT**: La ruta del directorio raíz del servidor web donde se encuentran los archivos servidos.

4. Integrar PHP con HTML para mostrar datos dinámicos

1. Crea un archivo PHP llamado fecha.php en el directorio raíz del servidor web.
2. Escribe el siguiente código para integrar PHP y HTML:

```
<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <title>Página Dinámica con PHP</title>

</head>

<body>

    <h1>Bienvenido a mi página dinámica</h1>

    <p>La fecha y hora actual es:

        <?php

        echo date('Y-m-d H:i:s');

        ?>

    </p>
```

```
</body>

</html>
```

3. Guarda el archivo y accede a http://localhost/fecha.php en tu navegador. El resultado debería mostrar la fecha y hora actualizada cada vez que recargues la página.

Actividad 6: Modificación del archivo php.ini

1. Acceder al archivo `php.ini-production`

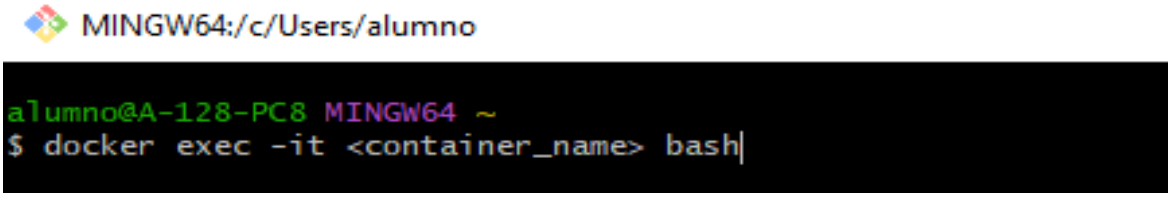
Para abrir y modificar el archivo `php.ini-production`, sigue los siguientes pasos dependiendo del entorno:

XAMPP:

1. Abre el Panel de Control de XAMPP.
2. Haz clic en el botón **Config** del módulo Apache y selecciona **PHP (php.ini)**. Esto abrirá el archivo de configuración en tu editor de texto.
3. Busca el archivo **php.ini-production**, generalmente ubicado en el directorio `xampp/php/`.

Docker:

1. Si estás usando Docker, puedes acceder al archivo php.ini dentro del contenedor PHP.
2. Usa el siguiente comando para acceder a una terminal del contenedor:



3. Dentro del contenedor, el archivo `php.ini` estará en la ruta `/usr/local/etc/php/`.

2. Propiedades clave de `php.ini`

file_uploads:

- **Descripción**: Esta directiva permite habilitar o deshabilitar la funcionalidad de subir archivos desde el navegador a través de formularios HTML.
- **Valores**:
    - `On`: Habilita la capacidad de subir archivos.
    - `Off`: Deshabilita la capacidad de subir archivos.

Ejemplo:

```
file_uploads = On
```

max_execution_time:

- **Descripción**: Especifica el tiempo máximo, en segundos, que se permite que un script PHP se ejecute antes de ser detenido por el servidor.
- **Valores**:
    - El valor predeterminado es `30` segundos, pero puede ajustarse según la necesidad de procesamiento del script.

Ejemplo:

```
max_execution_time = 60
```

short_open_tag:

- **Descripción**: Permite el uso de las etiquetas cortas de apertura `<?` en lugar de las etiquetas estándar `<?php` para abrir bloques de código PHP.
- **Valores**:
    - `On`: Habilita el uso de las etiquetas cortas.
    - `Off`: Deshabilita el uso de las etiquetas cortas, obligando a usar `<?php`.

Ejemplo:

```
short_open_tag = Off
```

3. Modificación de valores y su impacto

- **file_uploads**: Si está en `On`, permite que los usuarios suban archivos desde formularios HTML, lo que es crucial para aplicaciones como sistemas de gestión de archivos o sitios con formularios de contacto que permiten adjuntar archivos. Si se establece en `Off`, se deshabilita esta funcionalidad, lo que puede ser útil por razones de seguridad en sitios que no necesitan permitir subidas de archivos.
- **max_execution_time**: Si este valor es demasiado bajo, los scripts largos o pesados podrían no completarse, resultando en errores de tiempo de ejecución. Aumentarlo permite que scripts complejos, como la generación de informes o consultas a bases de datos grandes, se ejecuten completamente.
- **short_open_tag**: Si está en `On`, los desarrolladores pueden usar etiquetas cortas, lo que puede hacer que el código sea más compacto. Sin embargo, es mejor mantenerlo en `Off` para evitar conflictos con otros lenguajes de marcas o problemas de compatibilidad con sistemas que no permiten el uso de etiquetas cortas.

4. Comparación de `php.ini-development` y `php.ini-production`

php.ini-development:

- Está diseñado para un entorno de desarrollo donde los errores deben mostrarse de manera explícita y el rendimiento no es una prioridad.
- **Características clave**:
    - **display_errors**: `On` (muestra los errores en la pantalla).
    - **error_reporting**: Nivel máximo (`E_ALL`), para identificar todos los posibles problemas.
    - **memory_limit**: Generalmente más alto para pruebas y depuración.

php.ini-production:

- Está optimizado para entornos de producción donde los errores no deben ser visibles para los usuarios por razones de seguridad.
- **Características clave**:
    - **display_errors**: `Off` (los errores no se muestran públicamente, solo se registran en archivos de log).
    - **error_reporting**: Configurado para reportar solo los errores críticos (`E_ALL & ~E_DEPRECATED & ~E_STRICT`).
    - **memory_limit**: Es menor que en el entorno de desarrollo para evitar el consumo excesivo de recursos.