PHP 8: Programación de aplicaciones web en el servidor

PHP 8: Programación de aplicaciones web en el servidor

Guía paso a paso con ejemplos y proyectos prácticos

Jose Vicente Carratalá Sanchis

Tabla de contenidos

1. Prólogo....16
2. Introducción....18
2.1. Breve historia de la informática.... 19
2.2. Breve historia de internet....19
2.3. Breve historia de la web - desde su creación hasta la aparición de PHP....20
2.4. PHP.... 21
2.5. Breve historia de Internet - desde la aparición de php hasta el dia de hoy.... 22
2.6. Cómo instalar un entorno de desarrollo local para comenzar a desarrollar con PHP.... 23
2.7. Estructura de este libro....26
2.8. Sobre el autor....29
3. Fundamentos de PHP.... 31
3.1. Sintaxis....32
3.2. Comentarios.... 36
3.2.1. Linea unica.... 36
3.2.2. Multilinea....37
3.3. Entradas y salidas.... 38
3.3.1. Salidas....39
Echo....39
HTML....40
Print.... 41
3.3.2. Entradas.... 42
3.4. Operadores....43
3.4.1. Aritméticos....43
3.4.2. Comparacion.... 44
3.4.3. Lógicos.... 45
3.5. Variables....46
3.5.1. Declaracion....47
3.5.2. Usar variable....48

3.5.3. Encadenamiento....49
3.5.4. Incrementos....50
3.5.5. Operadores abreviados....52
3.5.6. Palabras reservadas....53
3.5.7. Identificadores....54
3.5.8. Constantes....55
3.6. Estructuras de datos....56
3.6.1. Arrays....56
Declaracion....57
Error....58
Var_dump....59
Push....60
Pop....61
Splice....62
Indices alfanumericos....63
Multidimensionales....64
3.7. Estructuras de control....65
3.7.1. Bucle....66
For....66
While....67
While inicio....68
Incremento....69
Do while....70
3.7.2. Condicionales....71
If....71
Else....72
Anidacion....73
Else if....74
Switch....75
3.7.3. Try....76
3.8. Funciones....79
3.8.1. Declaración....80

3.8.2. Uso de la función....80
3.8.3. Return en la función....81
3.8.4. Parámetros en las funciones....82
3.8.5. Parámetros....83
3.8.6. Multiples parametros....84
3.9. POO....85
3.9.1. Clases....85
3.9.2. Propiedades....86
3.9.3. Instancias....87
3.9.4. Acceso....88
3.9.5. Privado....89
3.9.6. Set y get....90
3.9.7. Validación....91
3.9.8. Constructor con parámetros....93
3.9.9. Clase pagina....95
3.9.10. Herencia....98
3.9.11. Abstracción....99
3.10. Includes....101
3.10.1. Include....103
3.10.2. Fallo....104
3.10.3. Require....105
3.11. Librerias....106
3.11.1. Fechas....106
3.11.2. Codificacion....108
3.12. Sesiones....114
3.12.1. Problema....115
3.12.2. Solución....116
3.13. Super Variables....118
3.13.1. GET....118
3.13.2. Session....119
3.13.3. Server....120
3.13.4. Environment....121

3.14. Formularios.....121
3.14.1. Formulario.....122
3.14.2. Atrapo datos.....123
3.14.3. Isset.....124
3.14.4. Get.....125
3.15. Persistencia.....126
3.15.1. Archivos.....126
4. Blog en PHP.....129
4.1. Proyecto-Definición de clases.....130
4.1.1. Estructura del proyecto.....131
4.1.2. Qué se está haciendo.....131
4.1.3. Archivo clases/Publicacion.php.....132
4.1.4. Archivo index.php.....132
4.1.5. Conclusión.....133
4.2. Proyecto-Método devolver entrada.....136
4.2.1. Estructura del proyecto.....137
4.2.2. Qué se está haciendo.....137
4.2.3. Cambios en clases/Publicacion.php.....137
4.2.4. Cambios en index.php.....138
4.2.5. Justificación de los cambios.....138
4.2.6. Conclusión.....139
4.3. Proyecto-Include entradas.....142
4.3.1. Estructura del proyecto.....143
4.3.2. Qué se está haciendo.....143
4.3.3. Archivo datos/arrayobjetos.php.....144
4.3.4. Cambios en index.php.....144
4.3.5. Cambios en clases/Publicacion.php.....145
4.3.6. Justificación de los cambios.....145
4.3.7. Conclusión.....145
4.4. Proyecto-Cabecera y pie de página.....149
4.4.1. Estructura del proyecto.....150
4.4.2. Qué se está haciendo.....150

4.4.3. Cambios en index.php..... 151
4.4.4. Archivos clases/Publicacion.php y datos/arrayobjetos.php..... 152
4.4.5. Justificación de los cambios..... 152
4.4.6. Conclusión..... 153
4.5. Proyecto-Include.....157
4.5.1. Estructura del proyecto.....158
4.5.2. Qué se está haciendo.....158
4.5.3. Cambios en index.php..... 159
4.5.4. Contenido de inc/cabecera.php..... 159
4.5.5. Contenido de inc/piedepagina.php..... 160
4.5.6. Archivos clases/Publicacion.php y datos/arrayobjetos.php..... 160
4.5.7. Justificación de los cambios..... 160
4.5.8. Conclusión..... 161
4.6. Proyecto-Estilo..... 166
4.6.1. Estructura del proyecto.....167
4.6.2. Qué se está haciendo.....168
4.6.3. Archivo css/estilo.css.....168
4.6.4. Archivo inc/cabecera.php..... 169
4.6.5. Archivos index.php, clases/Publicacion.php, datos/arrayobjetos.php y inc/piedepagina.php..... 170
4.6.6. Justificación de los cambios..... 170
4.6.7. Conclusión..... 170
4.7. Proyecto-Entradas en JSON..... 176
4.7.1. Estructura del proyecto.....177
4.7.2. Qué se está haciendo.....178
4.7.3. Contenido de datos/entradas.json..... 178
4.7.4. Cambios en index.php..... 179
4.7.5. Cambios en clases/Publicacion.php..... 180
4.7.6. Justificación de los cambios..... 180
4.7.7. Conclusión..... 181

4.8. Proyecto-JSON a objetos..... 187
4.8.1. Estructura del proyecto.....188
4.8.2. Qué se está haciendo.....189
4.8.3. Cambios en index.php..... 189
4.8.4. Cambios en clases/Publicacion.php..... 190
4.8.5. Cambios en datos/entradas.json..... 191
4.8.6. Justificación de los cambios..... 191
4.8.7. Conclusión..... 192
4.9. Proyecto-Imágenes y fecha..... 199
4.9.1. Estructura del proyecto.....200
4.9.2. Qué se está haciendo.....201
4.9.3. Cambios en clases/Publicacion.php.....201
4.9.4. Archivo datos/entradas.json..... 202
4.9.5. Archivo index.php..... 203
4.9.6. Justificación de los cambios..... 203
4.9.7. Conclusión..... 203
4.10. Proyecto-Estilo.....211
4.10.1. Estructura del proyecto.....212
4.10.2. Qué se está haciendo.....213
4.10.3. Cambios en css/estilo.css..... 213
4.10.4. Archivos clases/Publicacion.php, index.php, entradas.json..... 215
4.10.5. Justificación de los cambios..... 215
4.10.6. Conclusión..... 215
4.11. Proyecto-JSON páginas..... 222
4.11.1. Estructura del proyecto..... 223
4.11.2. Qué se está haciendo..... 224
4.11.3. Contenido de datos/paginas.json.....224
4.11.4. Cambios en index.php..... 225
4.11.5. Cambios en clases/Publicacion.php..... 227
4.11.6. Justificación de los cambios..... 227
4.11.7. Conclusión..... 227

4.12. Proyecto-Estilo..... 237
4.12.1. Estructura del proyecto.....238
4.12.2. Qué se está haciendo.....239
4.12.3. Cambios en clases/Publicacion.php.....239
4.12.4. Cambios en css/estilo.css..... 240
4.12.5. Archivos index.php y *.json.....242
4.12.6. Justificación de los cambios..... 242
4.12.7. Conclusión.....242
4.13. Proyecto-Panel de control..... 252
4.13.1. Estructura del proyecto.....254
4.13.2. Qué se está haciendo.....254
4.13.3. Contenido de panel.php.....255
4.13.4. Cambios en otros archivos..... 257
4.13.5. Justificación de los cambios..... 257
4.13.6. Conclusión.....257
4.14. Proyecto-Procesar entrada.....270
4.14.1. Estructura del proyecto.....271
4.14.2. Qué se está haciendo.....272
4.14.3. Contenido de panel.php (actualizado).....273
4.14.4. Contenido de procesar.php.....273
4.14.5. Consideraciones técnicas.....274
4.14.6. Justificación de los cambios..... 275
4.14.7. Conclusión.....275
4.15. Proyecto-Refactorización..... 288
4.15.1. Estructura del proyecto.....289
4.15.2. Qué se está haciendo.....290
4.15.3. Refactorización en la práctica.....290
1. Creación de una función de utilidad..... 290
2. Aplicación en index.php.....291
3. Aplicación en panel.php.....291
4.15.4. Qué se logra con esta refactorización..... 292
4.15.5. Justificación de los cambios..... 292

4.15.6. Conclusión....293
4.16. Proyecto-Cargar paginas....306
4.16.1. Estructura del proyecto....307
4.16.2. Qué se está haciendo....308
4.16.3. Acceso desde la URL....308
4.16.4. Contenido de pagina.php....309
4.16.5. Cambios en clases/Publicacion.php....310
4.16.6. Justificación de los cambios....310
4.16.7. Consideraciones futuras....311
4.16.8. Conclusión....311
4.17. Proyecto-Cargar una página....325
4.17.1. Estructura del proyecto....327
4.17.2. Qué se está haciendo....327
4.17.3. Ejemplo de enlace en el menú....328
4.17.4. Código de pagina.php (refinado)....328
4.17.5. Mejoras aplicadas....329
4.17.6. Justificación de los cambios....330
4.17.7. Consideraciones futuras....330
4.17.8. Conclusión....330
4.18. Proyecto-Refactorizar....345
4.18.1. Estructura del proyecto....347
4.18.2. Qué se está haciendo....347
4.18.3. Código para funciones.php....348
4.18.4. Aplicación en pagina.php....349
4.18.5. Ventajas de esta refactorización....350
4.18.6. Justificación de los cambios....350
4.18.7. Conclusión....351
5. Epílogo....366

1. Prólogo

La programación web es una de las habilidades más demandadas y versátiles en el mundo actual. Desde sus inicios, PHP ha sido una de las tecnologías más utilizadas para construir sitios y aplicaciones que forman parte de la vida diaria de millones de personas. En esta nueva edición, centrada en PHP 8, mi objetivo es acercar a cualquier persona interesada en el desarrollo web a las bases y prácticas esenciales para crear aplicaciones modernas y funcionales.

A lo largo de estas páginas, he reunido años de experiencia como programador y docente, transformando el proceso de aprendizaje en un recorrido gradual, claro y práctico. El libro está estructurado para que avances paso a paso, desde los primeros conceptos del lenguaje hasta la construcción de proyectos completos y funcionales, como la creación de tu propio blog. Cada capítulo está acompañado de ejemplos y fragmentos de código diseñados para que puedas experimentar y comprender los fundamentos en acción, fomentando así un aprendizaje activo y participativo.

PHP 8 introduce novedades y mejoras que hacen que programar en este lenguaje sea más potente y seguro. Aquí encontrarás no solo la explicación de estos conceptos, sino también su aplicación práctica, abordando los problemas habituales que surgen en el desarrollo real y cómo resolverlos con buenas prácticas.

Espero que este libro se convierta en tu guía de referencia durante tus primeros pasos y también en esos momentos en los que necesitas consultar una solución concreta. Mi deseo es que al terminarlo, no solo entiendas cómo funciona PHP 8, sino que seas capaz de construir tus propias aplicaciones web, enfrentando cada nuevo reto con confianza y curiosidad.

Bienvenido al fascinante mundo de la programación en PHP.

2. Introducción

2.1. Breve historia de la informática

La historia de la informática es el relato de cómo la humanidad ha buscado formas cada vez más eficientes de procesar, almacenar y comunicar información. Aunque sus raíces se remontan a la antigüedad, con dispositivos como el ábaco o las primeras herramientas de cálculo, el verdadero impulso llegó con la revolución científica y el desarrollo de las matemáticas en los siglos XVII y XVIII.

A mediados del siglo XIX, Charles Babbage diseñó la máquina analítica, considerada el primer concepto de un ordenador programable, y Ada Lovelace creó el primer algoritmo destinado a ser procesado por una máquina. Sin embargo, no fue hasta el siglo XX cuando la informática comenzó a desarrollarse de manera acelerada. Durante la Segunda Guerra Mundial, se construyeron las primeras computadoras electrónicas, como la Colossus británica y la ENIAC estadounidense, que ocuparon habitaciones enteras y funcionaban con válvulas de vacío.

La invención del transistor en los años 40 y su evolución en los años 50 permitió reducir el tamaño y el consumo de los ordenadores, facilitando su comercialización y el nacimiento de la industria informática. En las décadas siguientes, la aparición de los circuitos integrados y los microprocesadores impulsó la miniaturización y el abaratamiento, abriendo paso a la informática personal y doméstica.

El desarrollo de lenguajes de programación, sistemas operativos y redes de computadoras, especialmente con la llegada de Internet en los años 90, transformó la informática en una herramienta fundamental para la vida cotidiana, la educación, la empresa y el ocio. Hoy en día, la informática está presente en todos los ámbitos de la sociedad, desde los dispositivos móviles hasta la inteligencia artificial, y continúa evolucionando a un ritmo vertiginoso, marcando el rumbo del futuro tecnológico.

2.2. Breve historia de internet

Internet nació como un proyecto militar en la década de 1960, cuando el Departamento de Defensa de Estados Unidos impulsó la creación de ARPANET, una red que permitiera comunicar ordenadores de diferentes universidades y centros de investigación de forma descentralizada y resistente a fallos. En 1969 se realizó la primera conexión entre dos nodos, marcando el inicio de una nueva era en las comunicaciones.

Durante los años 70 y 80, la red fue creciendo y adoptando protocolos de comunicación comunes, como TCP/IP, que permitieron la interconexión de múltiples redes y la expansión a nivel internacional. El verdadero salto hacia la popularización llegó en 1991, cuando Tim Berners-Lee, científico del CERN, creó la World Wide Web: un sistema de hipertexto que facilitaba la navegación, la consulta de información y la publicación de páginas web accesibles para cualquier usuario.

A partir de la década de 1990, Internet experimentó una expansión explosiva, primero en el ámbito académico y después en los hogares y empresas, gracias a la aparición de los navegadores web, el correo electrónico y las primeras tiendas en línea. El desarrollo de nuevas tecnologías, como la banda ancha, el wifi y los dispositivos móviles, convirtió Internet en una herramienta cotidiana e imprescindible.

Hoy en día, Internet conecta a miles de millones de personas en todo el mundo, permitiendo la comunicación instantánea, el acceso a información ilimitada, el comercio electrónico, la educación a distancia y la colaboración global. Su evolución continúa a un ritmo acelerado, transformando profundamente la sociedad y la forma en que vivimos y trabajamos.

2.3. Breve historia de la web - desde su creación hasta la aparición de PHP

La World Wide Web, conocida simplemente como la web, fue creada a finales de los años ochenta por Tim Berners-Lee, un científico británico

del CERN. Su objetivo era facilitar el intercambio de información entre investigadores de diferentes lugares mediante un sistema de documentos conectados mediante enlaces de hipertexto. En 1990, Berners-Lee desarrolló el primer navegador web y el primer servidor web, sentando las bases de lo que hoy conocemos como Internet.

Al principio, la web era un medio estático: las páginas estaban compuestas únicamente por texto e imágenes y solo podían ser consultadas, no interactuadas. Sin embargo, este nuevo paradigma revolucionó la forma de acceder a la información y pronto se expandió más allá del ámbito académico.

Durante la primera mitad de los años 90, la web se popularizó con la llegada de navegadores como Mosaic y Netscape, que facilitaron el acceso a los contenidos. En esa época, el contenido seguía siendo mayoritariamente estático, pero pronto surgió la necesidad de crear páginas dinámicas, capaces de responder a las acciones de los usuarios y mostrar información personalizada.

Para ello, comenzaron a aparecer los primeros lenguajes de programación orientados al servidor, como CGI (Common Gateway Interface) y Perl, que permitían generar contenido dinámico, aunque de manera compleja y poco eficiente. Fue en este contexto, en 1995, cuando Rasmus Lerdorf desarrolló PHP (Personal Home Page), un lenguaje de programación especialmente diseñado para la creación de páginas web dinámicas de forma sencilla e integrada con HTML. PHP marcó el inicio de una nueva etapa en la evolución de la web, permitiendo a los desarrolladores crear aplicaciones interactivas y sentando las bases de la web moderna.

2.4. PHP

PHP es un lenguaje de programación de propósito general, especialmente diseñado para el desarrollo de aplicaciones web que se ejecutan del lado del servidor. Su nombre originalmente significaba “Personal Home Page”, aunque en la actualidad se interpreta como “PHP: Hypertext Preprocessor”. Fue creado en 1995 por Rasmus Lerdorf, quien buscaba una manera sencilla de gestionar y mostrar información personal en la web.

Desde sus primeras versiones, PHP destacó por su facilidad de integración con HTML y su sintaxis sencilla, lo que permitió que miles de desarrolladores de todo el mundo comenzaran a crear páginas web dinámicas de forma rápida y accesible. A medida que la web crecía y se hacía más compleja, PHP evolucionó para incorporar características avanzadas, soporte para bases de datos, manejo de sesiones, orientación a objetos y una gran variedad de funciones integradas para resolver necesidades comunes en la programación web.

Una de las grandes ventajas de PHP es su naturaleza de código abierto, lo que ha impulsado una comunidad global muy activa y una constante evolución del lenguaje. PHP es compatible con la mayoría de los servidores web y sistemas operativos, y es utilizado por millones de sitios y aplicaciones, incluyendo proyectos tan conocidos como WordPress, Facebook (en sus orígenes), Wikipedia y muchos otros.

Hoy en día, PHP continúa siendo una de las tecnologías más utilizadas para el desarrollo web, gracias a su rendimiento, versatilidad y a la gran cantidad de recursos y herramientas disponibles para los desarrolladores.

2.5. Breve historia de Internet - desde la aparición de php hasta el dia de hoy

Desde la aparición de PHP en 1995, Internet ha experimentado una transformación vertiginosa tanto en tecnología como en alcance social y económico. A finales de los años noventa, la web pasó de ser un entorno mayoritariamente estático a convertirse en un espacio dinámico e interactivo, gracias al auge de lenguajes como PHP, ASP y posteriormente JavaScript en el navegador. Esta evolución permitió la creación de foros, tiendas en línea, sistemas de gestión de contenidos y aplicaciones web cada vez más sofisticadas.

La llegada del nuevo milenio trajo consigo la llamada Web 2.0, caracterizada por la participación activa de los usuarios. Nacieron redes sociales como MySpace, Facebook y Twitter, plataformas de vídeo como YouTube y sistemas colaborativos como Wikipedia. Durante este periodo, PHP se consolidó como una de las tecnologías clave para el desarrollo de

sitios dinámicos y de grandes proyectos, impulsando sistemas como WordPress, Drupal o Moodle.

En la década de 2010, la generalización del acceso a Internet de alta velocidad, la proliferación de dispositivos móviles y la aparición de nuevas tecnologías como la computación en la nube y las APIs transformaron la manera de crear y consumir servicios online. La web se volvió ubicua, personalizada e instantánea, integrando redes sociales, servicios de streaming, comercio electrónico y comunicación en tiempo real.

En los últimos años, Internet ha seguido evolucionando con la expansión de la inteligencia artificial, el Internet de las cosas, las plataformas de desarrollo sin servidor (serverless) y la preocupación por la seguridad y la privacidad de los datos. El desarrollo web se ha hecho más profesional, con frameworks y herramientas modernas que conviven con tecnologías clásicas como PHP, que ha seguido actualizándose para responder a las nuevas necesidades del desarrollo moderno.

Hoy, Internet es una infraestructura fundamental para la sociedad global, imprescindible en la vida cotidiana, la economía, la educación y la comunicación, y sigue creciendo y adaptándose a los desafíos y oportunidades que plantea el futuro.

2.6. Cómo instalar un entorno de desarrollo local para comenzar a desarrollar con PHP

Para comenzar a desarrollar con PHP en tu propio ordenador, es necesario instalar un entorno de desarrollo local que incluya tanto PHP como un servidor web (como Apache o Nginx) y, opcionalmente, un sistema gestor de bases de datos como MySQL o MariaDB. El conjunto más popular para esto es conocido como “stack”, y uno de los más utilizados es el llamado **LAMP** (Linux, Apache, MySQL, PHP) en Linux o **XAMPP/WAMP/MAMP** según tu sistema operativo. Aquí tienes una guía general para instalarlo:

1. Elige el stack adecuado según tu sistema operativo

- **Windows:**
  Puedes instalar XAMPP o WAMP.

- **macOS:**
  Puedes instalar MAMP o XAMPP.

- **Linux (Ubuntu, Debian, etc.):**
  Se recomienda instalar el stack LAMP usando el gestor de paquetes.

---

2. Instalación paso a paso en cada sistema operativo

a) Windows con XAMPP:

1. Descarga el instalador desde la web oficial de XAMPP.

2. Ejecuta el instalador y sigue los pasos, seleccionando los componentes necesarios (al menos Apache y PHP).

3. Finaliza la instalación y abre el panel de control de XAMPP.

4. Inicia los servicios de Apache (y MySQL si lo necesitas).

5. Coloca tus archivos PHP en la carpeta `htdocs` dentro de la carpeta de instalación de XAMPP (por ejemplo, `C:\xampp\htdocs\`).

6. Accede desde el navegador a `http://localhost/`.

b) macOS con MAMP:

1. Descarga MAMP desde la web oficial.
2. Instala el paquete arrastrando la aplicación a la carpeta “Aplicaciones”.
3. Abre MAMP y haz clic en “Start Servers”.
4. Coloca tus archivos PHP en la carpeta `htdocs` de MAMP (por ejemplo, `/Applications/MAMP/htdocs/`).
5. Accede desde el navegador a `http://localhost:8888/`.

c) Linux (Ubuntu) con LAMP:

1. Abre la terminal.

Instala Apache, PHP y MySQL con el siguiente comando:

```
sudo apt update
```

```
sudo apt install apache2 php libapache2-mod-php mysql-server
```

2.

(Opcional) Instala phpMyAdmin para gestionar MySQL desde el navegador:

```
sudo apt install phpmyadmin
```

3.
4. Tus archivos PHP deben ir en `/var/www/html/`.

Reinicia Apache si haces cambios de configuración:

```
sudo systemctl restart apache2
```

5.
6. Accede desde el navegador a `http://localhost/`.

---

3. Comprueba tu instalación

Crea un archivo llamado `info.php` en la carpeta correspondiente con este contenido:

```
<?php

phpinfo();

?>
```

Abre `http://localhost/info.php` en tu navegador. Si ves la información de PHP, tu entorno está listo.

---

4. Editores de código recomendados

Para escribir tus scripts PHP puedes usar editores como Visual Studio Code, Sublime Text, Atom, Notepad++ o el que prefieras.

Con este entorno tendrás todo lo necesario para empezar a desarrollar y probar aplicaciones PHP en tu propio ordenador, sin necesidad de estar conectado a Internet.

2.7. Estructura de este libro

Este libro está estructurado de manera progresiva y práctica, pensada para acompañarte desde los conceptos más básicos de PHP hasta la realización de proyectos completos de desarrollo web. A continuación, se describe la organización general del contenido:

1. Introducción y contexto

- Breve historia de la informática
- Breve historia de Internet y la web
- ¿Qué es PHP?
- Cómo instalar un entorno de desarrollo local

2. Fundamentos de PHP

- Sintaxis básica del lenguaje
- Comentarios, variables y constantes
- Tipos de datos y operadores
- Estructuras de control (condicionales y bucles)
- Funciones y parámetros

3. Estructuras de datos

- Arrays y manipulación de colecciones
- Estructuras multidimensionales

- Funciones útiles para el manejo de datos

4. Programación orientada a objetos en PHP

- Conceptos fundamentales de la POO
- Clases, objetos, propiedades y métodos
- Herencia, abstracción y encapsulamiento
- Métodos especiales y buenas prácticas

5. Gestión de archivos y persistencia de datos

- Lectura y escritura de archivos
- Uso de sesiones y superglobales
- Introducción al manejo de bases de datos

6. Desarrollo web con PHP

- Manejo de formularios y validación de datos
- Generación dinámica de páginas HTML
- Gestión de estilos y recursos externos

7. Proyecto práctico: Creación de un blog

- Definición y organización de clases y archivos
- Implementación de funcionalidades paso a paso

- Mejora de la interfaz y experiencia de usuario

8. Buenas prácticas y recursos adicionales

- Recomendaciones para seguir aprendiendo
- Herramientas útiles para el desarrollo
- Comunidad y documentación

A lo largo de cada capítulo, encontrarás ejemplos prácticos, fragmentos de código y explicaciones claras, pensadas para que puedas avanzar a tu ritmo y poner en práctica lo aprendido desde el primer momento. El objetivo es que al terminar el libro, tengas una base sólida en PHP y la experiencia necesaria para abordar tus propios proyectos web.

2.8. Sobre el autor

Jose Vicente Carratalá Sanchis es desarrollador, formador y divulgador tecnológico con más de dos décadas de experiencia en el sector de la programación y el diseño web. Ha trabajado tanto en el ámbito empresarial como educativo, colaborando en la creación de aplicaciones y soluciones digitales para empresas, instituciones y proyectos personales.

A lo largo de su carrera, ha combinado su labor profesional con la enseñanza, impartiendo cursos y talleres de programación, bases de datos, inteligencia artificial y tecnologías web en centros de formación, universidades y empresas. Su enfoque pedagógico se caracteriza por la claridad, la sencillez y el sentido práctico, facilitando el aprendizaje de la tecnología a estudiantes de todos los niveles.

Autor de varios libros técnicos, Jose Vicente destaca por su capacidad para explicar conceptos complejos de manera accesible, ayudando a miles de lectores y alumnos a dar sus primeros pasos en el mundo de la informática y la programación. Apasionado de la innovación y la

divulgación, sigue explorando nuevas herramientas y compartiendo su conocimiento para contribuir al crecimiento de la comunidad tecnológica.

3. Fundamentos de PHP

PHP es un lenguaje de programación diseñado para generar contenido dinámico en servidores web. Nació en 1995 con el objetivo de simplificar la creación de páginas que reaccionan a las acciones del usuario sin necesidad de recargar manualmente cada elemento. Desde entonces ha evolucionado hacia un ecosistema completo, ofreciendo un intérprete robusto, una sintaxis accesible y una amplísima comunidad que contribuye con librerías, marcos de trabajo y herramientas de despliegue.

A diferencia de los lenguajes que se compilan antes de ejecutarse, PHP se interpreta en tiempo real: el servidor recibe la petición del cliente, procesa el script y devuelve directamente el HTML resultante. Esto facilita el ciclo de desarrollo, permite comprobar al instante los cambios y reduce la brecha entre la escritura del código y su salida visual en el navegador.

Aunque su uso original se centró en tareas sencillas —como mostrar datos de formularios o acceder a bases de datos—, hoy PHP ofrece un amplio abanico de posibilidades: frameworks para estructurar proyectos de gran envergadura, sistemas de plantillas que aíslan la lógica de presentación, extensiones para optimizar el rendimiento y compatibilidad con prácticamente cualquier sistema de bases de datos y servidor web. Estos “fundamentos de PHP” le dan la versatilidad necesaria para construir desde blogs personales hasta grandes plataformas de comercio electrónico.

3.1. Sintaxis

PHP es un lenguaje que permite mezclar instrucciones dentro de páginas HTML. Un archivo PHP puede comenzar con contenido HTML puro, continuar con bloques de código PHP y, tras estos, volver a incluir HTML. Esta flexibilidad permite que el desarrollador controle de forma precisa qué parte del contenido se genera de forma dinámica y cuál permanece fija.

Para comenzar a escribir código PHP dentro de un archivo, se utiliza la etiqueta `<?php`, que marca el inicio del bloque, y `?>`, que indica su final. El intérprete de PHP ejecutará únicamente el código que se encuentre entre estas dos etiquetas. El resto del contenido del archivo se enviará tal cual al navegador, como HTML.

Es posible abrir y cerrar varias veces el bloque de PHP dentro del mismo archivo. Esto permite alternar entre HTML y PHP según convenga, facilitando la estructura del documento y la generación de contenido dinámico en secciones concretas.

El código PHP se escribe línea a línea, terminando cada instrucción con punto y coma `;`. Aunque el lenguaje permite cierta flexibilidad con los espacios en blanco y los saltos de línea, es buena práctica mantener una estructura clara mediante sangrados e indentaciones que hagan el código más legible.

```
001-extension.php
<p>Un archivo PHP puede contener html</p>
```

En PHP, la extensión más común para los archivos es `.php`. Esta extensión indica al servidor que debe procesar el archivo con el motor de PHP antes de enviarlo al navegador. Aunque técnicamente es posible usar otras extensiones, como `.phtml` o incluso `.html`, lo habitual y recomendable es usar `.php` para evitar confusiones y asegurar la correcta interpretación del código.

Cuando el servidor encuentra un archivo con esta extensión, analiza el contenido y ejecuta cualquier bloque que esté delimitado por `<?php` y `?>`. El resultado de esa ejecución, normalmente en forma de HTML generado dinámicamente, es lo que se envía al navegador del usuario.

Es importante asegurarse de que el servidor esté configurado para reconocer y procesar los archivos con extensión `.php`. Esto suele venir

activado por defecto en la mayoría de servidores web como Apache o Nginx cuando tienen instalado PHP.

```
002-php.php
<p>Un archivo PHP puede contener html</p>

<?php

    echo "Lo que escriba aqui dentro es PHP pero se convierte
a HTML";

?>

<p>Puedo continuar con HTML puro</p>
```

El uso de PHP dentro de un archivo permite alternar entre HTML y bloques de código que se ejecutan en el servidor. Por ejemplo, se puede iniciar el documento con HTML, insertar un bloque de PHP para generar contenido dinámico, y continuar luego con más HTML. Esta característica es una de las principales ventajas del lenguaje, ya que facilita la integración de lógica de programación dentro del flujo natural de una página web.

Cuando el navegador solicita la página, el servidor ejecuta únicamente el código contenido entre las etiquetas `<?php` y `?>`, y sustituye ese bloque por el resultado producido. Todo el contenido HTML que no esté dentro de estas etiquetas se envía directamente sin modificaciones.

Esto permite crear estructuras mixtas en las que se puede mantener un diseño web estático (HTML) con fragmentos dinámicos generados por PHP, como por ejemplo textos personalizados, resultados de bases de datos o condiciones de visualización.

```
003-varias veces.php
<p>Un archivo PHP puede contener html</p>

<?php

    echo "Lo que escriba aqui dentro es PHP pero se convierte
a HTML";

?>

<p>Puedo continuar con HTML puro</p>

<?php

    echo "Esto vuelve a estar escrito por PHP";

?>

<p>Y sigo con HTML puro</p>
```

PHP permite insertar múltiples bloques de código dentro de un mismo archivo. Esto significa que se pueden abrir y cerrar secciones PHP tantas veces como sea necesario a lo largo del documento, alternando con contenido HTML según convenga.

Esta capacidad resulta útil cuando se necesita incluir fragmentos de lógica o salida dinámica en puntos concretos del HTML, sin tener que generar toda la página exclusivamente desde el interior de un único bloque PHP.

Cada bloque se delimita con `<?php` para abrir y `?>` para cerrar, y dentro de estos se pueden realizar operaciones, imprimir contenido o incluir estructuras de control. Al volver al HTML fuera del bloque, el flujo del documento continúa de forma natural. Esta alternancia se convierte en una herramienta poderosa para construir páginas web dinámicas de manera clara y organizada.

```
004-Sangrado.php
<?php

    echo "Hola";
echo "Hola de nuevo";

?>
```

3.2. Comentarios

Los comentarios en PHP permiten al programador incluir anotaciones dentro del código que no serán interpretadas ni ejecutadas por el servidor. Son útiles para explicar el propósito de una sección, dejar recordatorios o desactivar temporalmente partes del código sin eliminarlas.

PHP admite varios tipos de comentarios. Uno de ellos es el comentario de una sola línea, que comienza con `//`. Todo lo que se escriba después de estos caracteres, hasta el final de la línea, será ignorado por el intérprete. También se puede utilizar `#` para este mismo propósito, aunque es menos habitual y se hereda de otros lenguajes como Perl.

Además de los comentarios de línea única, PHP permite escribir comentarios multilínea, que comienzan con `/*` y terminan con `*/`. Todo el contenido intermedio será tratado como comentario, sin importar cuántas líneas ocupe. Este formato resulta útil para dejar explicaciones más extensas o para desactivar bloques completos de código durante el desarrollo.

3.2.1. Línea única

Los comentarios de una sola línea en PHP se escriben anteponiendo `//` al texto. Esta notación permite añadir observaciones o explicaciones breves al lado de una instrucción o en una línea independiente. Es común usarla

para describir qué hace una línea de código o para dejar indicaciones rápidas durante el desarrollo.

También se puede emplear el símbolo # para lograr el mismo efecto, aunque su uso es menos frecuente en PHP y más común en otros lenguajes como Bash o Perl.

Los comentarios no afectan la ejecución del programa: son completamente ignorados por el intérprete de PHP. Por eso, también se pueden usar para desactivar temporalmente líneas de código sin borrarlas, facilitando las pruebas o depuración.

```
001-Una única linea.php
<?php

    // Esto es un comentario de una única linea

?>
```

3.2.2. Multilínea

Los comentarios multilínea en PHP se utilizan cuando se desea añadir explicaciones más extensas o cuando se necesita desactivar varias líneas de código al mismo tiempo. Para ello se emplean los delimitadores /* para abrir el comentario y */ para cerrarlo.

Todo el texto que se encuentre entre estos dos símbolos será ignorado por el intérprete, sin importar cuántas líneas ocupe. Este formato resulta especialmente útil para documentar funciones, describir bloques de lógica compleja o dejar instrucciones internas al equipo de desarrollo.

Es importante asegurarse de no anidar comentarios multilínea dentro de otros comentarios multilínea, ya que PHP no lo permite y provocaría un error de interpretación.

```
002-multilinea.php
<?php

   /*
        Esto es un comentario de primera linea
        Segunda linea
        Tercera linea
    */

?>
```

3.3. Entradas y salidas

PHP permite tanto mostrar como recibir información. Las salidas se utilizan para enviar datos al navegador del usuario, ya sea como parte del contenido HTML o como respuesta a una petición. Las entradas, por su parte, permiten capturar datos que el usuario introduce mediante formularios, parámetros en la URL o variables de sesión, entre otros.

El mecanismo más directo para mostrar información es utilizar instrucciones como `echo` o `print`, que insertan texto en la página generada. Estas instrucciones pueden incluir tanto contenido estático como resultados de operaciones o valores de variables.

Para capturar entradas, PHP proporciona supervariables como `$_GET`, `$_POST` o `$_REQUEST`, que almacenan los datos enviados desde formularios o incluidos en la URL. Estas variables se utilizan para adaptar el comportamiento del programa según las acciones del usuario o los parámetros recibidos.

3.3.1. Salidas

Las salidas en PHP permiten enviar información desde el servidor al navegador. Esta información puede ser texto plano, etiquetas HTML, resultados de cálculos o cualquier combinación de los anteriores. El objetivo principal es generar contenido dinámico que se adapte a cada usuario o situación.

La instrucción más común para producir salidas es `echo`, que permite imprimir directamente en pantalla el resultado de una expresión o una cadena de texto. También se puede usar `print`, aunque tiene un comportamiento ligeramente distinto, ya que devuelve un valor, lo que permite integrarlo dentro de expresiones más complejas.

Además de mostrar texto simple, estas funciones pueden incluir etiquetas HTML para dar formato a la salida. De esta forma, se pueden generar estructuras completas como listas, tablas o formularios, todo ello desde código PHP.

Echo

La instrucción `echo` es una de las formas más utilizadas para mostrar contenido desde PHP al navegador. Su sintaxis es sencilla y permite imprimir tanto cadenas de texto como el resultado de variables o expresiones.

Se puede usar con o sin paréntesis, aunque lo más habitual es escribirla directamente seguida del contenido entre comillas. Cuando se desea concatenar variables y texto, se utiliza el operador `.` para unir los elementos que se desean mostrar.

`echo` es muy útil para generar HTML desde PHP. Al incluir etiquetas HTML dentro de las comillas, el contenido enviado al navegador será interpretado como parte de la estructura de la página, permitiendo así mezclar programación con presentación de forma dinámica.

```
001-echo.php
<?php

    echo "Contenido HTML escrito desde PHP";


?>
```

HTML

PHP permite incluir etiquetas HTML dentro de las cadenas que se imprimen con `echo`. De este modo, se puede generar contenido dinámico que mantiene el formato visual definido por las etiquetas.

Al utilizar comillas dentro de otras comillas, es importante prestar atención al tipo de comillas usadas o escaparlas adecuadamente para evitar errores de sintaxis. Por ejemplo, se puede usar comillas dobles para la cadena PHP y comillas simples dentro del HTML, o viceversa.

Este mecanismo es especialmente útil para crear elementos como encabezados, párrafos, listas o cualquier otro componente HTML desde el propio código PHP. Así, el servidor puede adaptar el contenido visual según los datos disponibles o las acciones del usuario.

```
002-etiquetas html.php
<?php

    echo "<h1>Contenido HTML escrito desde PHP</h1>";


?>
```

Print

Otra instrucción que se puede utilizar para mostrar contenido en PHP es `print`. Funciona de forma muy similar a `echo`, aunque tiene la particularidad de devolver un valor, concretamente el entero `1`, lo que permite integrarla dentro de expresiones o estructuras condicionales si se desea.

En la práctica, `print` se usa casi igual que `echo`, con la diferencia de que admite solo un argumento, mientras que `echo` puede recibir varios separados por comas. Al igual que con `echo`, también se pueden incluir etiquetas HTML dentro del texto para controlar la apariencia del contenido en la página resultante.

Aunque `echo` es ligeramente más rápido y más común en la mayoría de los proyectos PHP, `print` sigue siendo completamente válido y útil en contextos donde se espera una expresión que devuelva un valor.

```
003-print.php
<?php

    print "<h1>Contenido HTML escrito desde PHP</h1>";


?>
```

3.3.2. Entradas

PHP permite acceder a los datos que el usuario introduce a través de formularios o URLs utilizando las supervariables. Una de las más utilizadas es `$_GET`, que recoge los datos enviados mediante el método GET. Estos datos se pasan como parámetros en la URL y pueden ser capturados y utilizados directamente desde el código PHP.

Para acceder a un valor, se indica el nombre del parámetro entre corchetes, como en `$_GET['nombre']`. Esto permite leer la información introducida por el usuario y generar contenido personalizado en función de esos datos.

Es importante verificar que los parámetros existen antes de usarlos, ya que de lo contrario se pueden generar errores. Además, se recomienda validar y filtrar estos datos para evitar posibles problemas de seguridad como la inyección de código o el acceso a recursos no autorizados.

```
001-supervariable get.php
<?php

    echo "tu nombre es: ".$_GET['nombre'];


?>
```

3.4. Operadores

Los operadores en PHP permiten realizar distintas operaciones entre valores o variables. Son símbolos que representan acciones como sumar, restar, comparar, asignar o combinar condiciones lógicas. Su uso es esencial para construir expresiones que resuelvan cálculos, tomen decisiones o modifiquen valores según determinadas reglas.

PHP incluye varios grupos de operadores, entre ellos los operadores aritméticos, los operadores de comparación y los operadores lógicos. Cada grupo permite realizar tareas específicas dentro del código, como operaciones matemáticas, verificación de condiciones o evaluación de expresiones booleanas.

Comprender y utilizar correctamente estos operadores es clave para construir estructuras dinámicas y funcionales en cualquier programa PHP.

3.4.1. Aritméticos

Los operadores aritméticos en PHP se utilizan para realizar operaciones matemáticas básicas entre valores numéricos o variables. Estos operadores incluyen la suma (`+`), la resta (`-`), la multiplicación (`*`), la división (`/`) y el módulo (`%`), que devuelve el resto de una división.

Cada uno de estos operadores puede aplicarse directamente a constantes numéricas o a variables que contengan valores numéricos.

También se pueden combinar en expresiones más complejas, utilizando paréntesis para controlar el orden de evaluación.

Además de los operadores básicos, PHP respeta la jerarquía matemática estándar, por lo que primero se resuelven las multiplicaciones y divisiones, y luego las sumas y restas, a menos que se especifique otro orden mediante paréntesis.

```
001-Aritméticos.php
<?php

    echo 4+3;
    echo "<br>";
3;
    echo "<br>";
    echo 4*3;
    echo "<br>";
    echo 4/3;
    echo "<br>";
    echo 4%3;

?>
```

3.4.2. Comparacion

Los operadores de comparación en PHP permiten evaluar si dos valores son iguales, diferentes o si uno es mayor o menor que otro. Estos operadores se utilizan habitualmente en estructuras condicionales para tomar decisiones en función del resultado de la comparación.

Entre los más comunes se encuentran `==` (igualdad de valor), `!=` (diferente), `>` (mayor que), `<` (menor que), `>=` (mayor o igual que) y `<=` (menor o igual que). También existe `===`, que compara tanto el valor como el tipo de dato, y `!==`, que verifica si los valores o los tipos son distintos.

El resultado de una comparación es siempre un valor booleano: `true` si la condición se cumple, y `false` en caso contrario. Este resultado puede usarse para controlar el flujo del programa, por ejemplo, dentro de una sentencia `if` para ejecutar un bloque de código solo si se cumple cierta condición.

```
002-comparacion.php
<?php

    echo 4<3;
    echo "<br>";
    echo 4>3;
    echo "<br>";
    echo 4<=3;
    echo "<br>";
    echo 4>=3;
    echo "<br>";
    echo 4==3;
    echo "<br>";
    echo 4!=3;
    echo "<br>";
    echo 4===4;
    echo "<br>";
    echo 4=="4";
    echo "<br>";

?>
```

3.4.3. Lógicos

Los operadores lógicos en PHP permiten combinar múltiples condiciones para evaluar expresiones complejas. Estos operadores devuelven un valor booleano (`true` o `false`) en función del resultado de las condiciones implicadas.

El operador `&&` (y lógico) devuelve `true` solo si todas las condiciones son verdaderas. Se utiliza cuando se desea que varias condiciones se cumplan simultáneamente. Por ejemplo, una expresión que verifica que un número sea mayor que 10 y menor que 20.

El operador `||` (o lógico) devuelve `true` si al menos una de las condiciones es verdadera. Es útil cuando basta con que se cumpla una de varias opciones.

Finalmente, el operador `!` (no lógico) invierte el valor de una condición. Si una expresión es verdadera, `!` la convertirá en falsa, y viceversa. Estos operadores se pueden combinar con los de comparación para construir estructuras condicionales más sofisticadas.

```
001-Lógicos.php
<?php

    echo 4>3 && 3>2 && 2>1;
    echo "<br>";
    echo 4>3 && 3>2 && 2>2;
    echo "<br>";

    echo 4>3 || 3>2 || 2>2;
    echo "<br>";
    echo 4>3 || 3>3 || 2>2;
    echo "<br>";
    echo 4>4 || 3>3 || 2>2;
    echo "<br>";


?>
```

3.5. Variables

Las variables en PHP son contenedores que permiten almacenar valores para ser utilizados a lo largo del programa. Se definen utilizando el

símbolo `$` seguido del nombre de la variable, y se les asigna un valor mediante el operador `=`.

Una característica importante de PHP es que es un lenguaje de tipado dinámico, lo que significa que no es necesario declarar el tipo de dato de la variable; este se determina automáticamente según el valor que se le asigne. Así, una variable puede contener un número, una cadena de texto, un valor booleano, un array o incluso un objeto, dependiendo del contexto.

El nombre de la variable debe comenzar por una letra o un guion bajo, seguido de letras, números o guiones bajos. No se permite comenzar con un número ni usar espacios. Además, el lenguaje distingue entre mayúsculas y minúsculas, por lo que `$edad` y `$Edad` serían variables distintas.

3.5.1. Declaracion

Para declarar una variable en PHP, se utiliza el símbolo `$` seguido del nombre de la variable y el operador de asignación `=` para establecer su valor. Por ejemplo: `$edad = 45;`. Esta instrucción crea una variable llamada `edad` y le asigna el valor numérico 45.

No es necesario especificar el tipo de dato, ya que PHP lo detecta automáticamente. Se pueden almacenar números, cadenas de texto, valores booleanos, arrays, objetos y otros tipos, simplemente asignándolos a la variable correspondiente.

Cada instrucción de asignación debe finalizar con un punto y coma `;`. Además, se puede modificar el valor de la variable más adelante en el código, sobrescribiendo el anterior sin necesidad de volver a declararla.

```
001-Declaracion.php
<?php

    $edad = 45;

?>
```

3.5.2. Usar variable

Una vez declarada una variable, se puede utilizar en cualquier parte del código para mostrar su valor, realizar operaciones o construir nuevas expresiones. Para ello, simplemente se llama por su nombre precedido del símbolo `$`.

PHP permite concatenar texto con variables usando el operador `.`. Por ejemplo, si se tiene una variable `$edad`, se puede generar una salida como `"Tu edad es de ".$edad." años"`, combinando texto fijo con el valor dinámico de la variable.

Este mecanismo permite personalizar la información que se presenta al usuario o que se procesa internamente, adaptando la ejecución del programa en función de los datos almacenados.

```
002-Usar una variable.php
<?php

    $edad = 45;
    echo "Tu edad es de ".$edad." años";

?>
```

3.5.3. Encadenamiento

El encadenamiento en PHP se refiere a la combinación de varias partes de texto y valores de variables en una misma expresión, utilizando el operador de concatenación `.`. Este operador permite unir fragmentos de cadenas de texto con el contenido de variables para formar mensajes dinámicos.

Por ejemplo, si una variable `$edad` contiene el valor `45`, se puede construir una frase como `"Tu edad es de ".$edad." años"`. Cada parte se concatena en orden, y el resultado final es una única cadena que puede mostrarse con `echo` o utilizarse en otras operaciones.

Este procedimiento es muy habitual en la generación de respuestas personalizadas, nombres de archivos, rutas, consultas a bases de datos o cualquier situación donde el contenido deba ajustarse a datos variables.

```
003-encadenamiento.php
<?php

    $edad = 45;
    echo "Tu edad es de ".$edad." años";

?>
```

3.5.4. Incrementos

PHP permite modificar el valor de una variable incrementándola o decrementándola con operadores específicos. El operador `++` incrementa su valor en una unidad, mientras que `--` lo reduce en uno. Estos operadores se pueden usar antes o después del nombre de la variable, aunque su posición puede alterar el comportamiento en contextos más complejos.

Por ejemplo, si una variable `$edad` contiene el valor `45`, al aplicar `$edad++`, su valor pasa a ser `46`. Este tipo de operación es muy útil en bucles o cuando se desea llevar un conteo.

Además de los incrementos y decrementos unitarios, PHP permite utilizar operadores abreviados como `+=`, `-=`, `*=`, y `/=`, que combinan la operación con la asignación. Así, `$edad += 2` suma 2 al valor actual y lo guarda en la misma variable, todo en una sola línea.

```
004-incrementos.php
<?php

    $edad = 45;
    echo "Tu edad es de ".$edad." años<br>";
    $edad++;
    echo "Tu edad es de ".$edad." años<br>";
-;
    echo "Tu edad es de ".$edad." años<br>";

?>
```

Los operadores abreviados en PHP permiten realizar una operación y asignar el resultado a la misma variable de forma compacta. Son especialmente útiles para simplificar el código y mejorar su legibilidad.

Por ejemplo, en lugar de escribir `$edad = $edad + 2;`, se puede escribir directamente `$edad += 2;`. Esta expresión suma 2 al valor actual de `$edad` y almacena el resultado nuevamente en esa variable. De forma similar, se pueden usar `-=` para restar, `*=` para multiplicar, `/=` para dividir y `%=` para calcular el módulo.

Estos operadores se utilizan frecuentemente en bucles, contadores o cálculos acumulativos, donde una variable necesita ser actualizada repetidamente en función de su valor anterior.

```
005-operadores abreviados.php
<?php

    $edad = 45;
    echo "Tu edad es de ".$edad." años<br>";
    $edad += 2;
    echo "Tu edad es de ".$edad." años<br>";
= 2;
    echo "Tu edad es de ".$edad." años<br>";
    $edad *= 2;
    echo "Tu edad es de ".$edad." años<br>";
    $edad /= 2;
    echo "Tu edad es de ".$edad." años<br>";


?>
```

3.5.5. Operadores abreviados

PHP, como muchos lenguajes de programación, posee una serie de palabras reservadas que no pueden utilizarse como nombres de variables, funciones o clases. Estas palabras forman parte del lenguaje y tienen un significado especial para el intérprete, por lo que usarlas incorrectamente puede generar errores o comportamientos inesperados.

Entre las palabras reservadas se encuentran estructuras de control como `if`, `else`, `switch`, `while`, `for`, funciones del lenguaje como `echo`, `print`, `isset`, y conceptos propios de la programación orientada a objetos como `class`, `public`, `private`, `protected`, `extends`, `interface`, entre muchas otras.

Es importante tener en cuenta que estas palabras no distinguen entre mayúsculas y minúsculas, ya que PHP no permite redefinirlas bajo ninguna forma. Por ello, es recomendable utilizar nombres descriptivos y únicos para las variables, evitando coincidencias con estas palabras clave.

3.5.6. Palabras reservadas

Las palabras reservadas son términos que PHP utiliza internamente para definir su sintaxis y funcionamiento. No se pueden emplear como identificadores para variables, funciones, clases o constantes, ya que están asociadas a instrucciones del lenguaje.

Ejemplos de palabras reservadas incluyen `function`, `return`, `class`, `echo`, `new`, `foreach`, `try`, `catch`, `public`, `private`, entre muchas otras. También se consideran reservadas algunas funciones especiales como `isset()`, `empty()`, `include`, `require`, etc.

Usar una palabra reservada como nombre de variable generaría un error de interpretación o podría hacer que el código se comporte de forma incorrecta. Por eso, al nombrar identificadores se recomienda usar términos que no estén en la lista de palabras clave del lenguaje.

```
006-palabras reservadas.php

PHP Keywords
__halt_compiler()   abstract     and    array()      as
break  callable     case   catch  class
clone  const  continue     declare      default
die()  do     echo   else   elseif
empty()       enddeclare   endfor endforeach   endif
endswitch     endwhile     eval() exit() extends
final  finally      fn (as of PHP 7.4)  for    foreach
function      global goto   if     implements
include       include_once instanceof   insteadof
interface
isset()       list() match (as of PHP 8.0)      namespace
new
or     print  private      protected    public
readonly (as of PHP 8.1.0) *     require      require_once
return static
switch throw  trait  try    unset()
use    var    while  xor    yield
yield from
```

3.5.7. Identificadores

En PHP, los identificadores son los nombres que se asignan a variables, funciones, clases y otros elementos definidos por el programador. Para ser válidos, deben cumplir ciertas reglas sintácticas.

Un identificador debe comenzar con una letra (a-z o A-Z) o con un guion bajo _, seguido de cualquier número de letras, dígitos o guiones bajos. No puede comenzar con un número ni contener espacios. Además, aunque PHP no distingue entre mayúsculas y minúsculas en los nombres de funciones, sí lo hace en los nombres de variables: `$edad` y `$Edad` son variables distintas.

También se recomienda evitar el uso de palabras reservadas como identificadores, y se aconseja adoptar convenciones de nomenclatura claras y coherentes para facilitar la lectura y mantenimiento del código.

```
007-identificadores.php
<?php

    $edad = 45;
    $edad2 = 45;
    $e2dad = 45;
    //$2edad = 45;
    $edad_nueva = 45;
nueva = 45;
    //$edad nueva = 45;
    //$_edad = 45;

?>
```

3.5.8. Constantes

Las constantes en PHP son identificadores cuyo valor no puede cambiar durante la ejecución del programa. Se definen con la palabra clave `const` seguida del nombre de la constante y su valor. A diferencia de las variables, no se antepone el símbolo `$` y deben declararse con un nombre válido de identificador.

Una vez definida, la constante puede utilizarse en cualquier parte del código, incluso fuera de funciones o clases, siempre que esté dentro del ámbito de visibilidad adecuado. Su valor es inmutable, lo que significa que cualquier intento de reasignación generará un error.

Las constantes son útiles para representar valores que no deben modificarse, como el número pi, rutas de directorio, claves de configuración, o límites máximos de ejecución. También ayudan a mejorar la legibilidad del código, al expresar con claridad el propósito de ciertos valores fijos.

```
008-constante.php
<?php

    const PI = 3.1415;
    echo "El valor de PI es: ".PI;
    PI = 4

?>
```

3.6. Estructuras de datos

PHP permite almacenar múltiples valores en una sola variable mediante estructuras de datos llamadas arrays. Los arrays son colecciones ordenadas en las que cada elemento puede identificarse por un índice numérico o una clave alfanumérica.

Para declarar un array, se utiliza la sintaxis con corchetes, asignando uno o varios valores a una variable. Esta estructura resulta especialmente útil cuando se necesita manejar listas de elementos, como nombres, resultados, productos o cualquier conjunto homogéneo o heterogéneo de datos.

Los arrays permiten recorrer sus elementos mediante bucles, acceder a valores concretos según su posición o clave, y modificarlos en tiempo de ejecución. Además, PHP proporciona numerosas funciones para manipular arrays de forma eficiente.

3.6.1. Arrays

Existen varias formas de declarar un array en PHP. La más común es mediante corchetes, asignando directamente los valores deseados. Por ejemplo, se puede crear un array llamado `$agenda` y asignar a cada posición un nombre, como `$agenda[0] = "Jose";`.

También se puede utilizar la función `array()` para declarar un array con varios elementos de una vez, aunque esta forma ha sido desplazada en versiones modernas por el uso de corchetes.

Cada elemento del array se almacena en una posición identificada por un índice, que por defecto es numérico y comienza en cero. Esta estructura facilita el acceso secuencial y el tratamiento automático de conjuntos de datos.

Declaracion

La declaración implícita de un array en PHP consiste en asignar directamente valores a posiciones del array sin necesidad de inicializarlo previamente. Por ejemplo, al escribir `$agenda[0] = "Jose Vicente";`, PHP crea automáticamente el array `$agenda` y le asigna el valor correspondiente en la posición 0.

Este método permite construir un array paso a paso, añadiendo elementos en las posiciones deseadas sin tener que definir la estructura completa desde el principio. Además, PHP asigna los índices de forma secuencial si no se especifican explícitamente.

Esta flexibilidad facilita la creación dinámica de colecciones de datos, adaptándose al flujo lógico del programa sin requerir una declaración estructurada al inicio.

```
001-Declaracion implicita.php
<?php

    $agenda[0] = "Jose Vicente";
    $agenda[1] = "Juan";


?>
```

Error

Cuando se intenta mostrar un array directamente con `echo`, PHP produce un error o una advertencia, ya que no puede convertir automáticamente una estructura compleja como un array en una cadena de texto. Este comportamiento se debe a que `echo` está diseñado para trabajar con valores simples como textos o números.

Si se desea visualizar el contenido de un array, es necesario utilizar funciones como `print_r()` o `var_dump()`, que están preparadas para recorrer y representar la estructura interna del array de forma legible. Estas funciones muestran tanto los índices como los valores, lo que resulta útil para depurar el código o comprobar el estado de los datos en tiempo de ejecución.

```
002-echo error.php
<?php

    $agenda[0] = "Jose Vicente";
    $agenda[1] = "Juan";
    echo $agenda;

?>
```

Var_dump

La función `var_dump()` en PHP permite inspeccionar el contenido de una variable con gran detalle. Es especialmente útil para visualizar arrays, ya que muestra tanto los índices como los tipos de dato y los valores almacenados.

Al aplicar `var_dump()` a un array, el resultado incluye la cantidad de elementos que contiene, los índices utilizados (numéricos o alfanuméricos) y los datos asociados a cada uno, incluyendo su tipo (por ejemplo, `string`, `int`, `bool`, etc.).

Esta herramienta es muy empleada durante el desarrollo para depurar el código y entender mejor cómo se están almacenando y organizando los datos en estructuras complejas como arrays multidimensionales u objetos.

```
003-var dump.php
<?php

    $agenda[0] = "Jose Vicente";
    $agenda[1] = "Juan";
    var_dump($agenda);

?>
```

Push

PHP proporciona funciones para añadir elementos a un array de forma dinámica. Una de las más comunes es `array_push()`, que permite insertar uno o más elementos al final del array.

Esta función modifica directamente la estructura del array original, expandiéndola con los nuevos valores. Es útil cuando se trabaja con listas de longitud variable, como registros obtenidos desde una base de datos o datos introducidos por el usuario.

Al combinar `array_push()` con `var_dump()`, se puede comprobar fácilmente cómo evoluciona el array tras cada inserción. Esta técnica es frecuente en algoritmos que requieren acumulación progresiva de información.

```
004-array push.php
<?php

    $agenda[0] = "Jose Vicente";
    $agenda[1] = "Juan";
    var_dump($agenda);
    echo "<br>";
    array_push($agenda,"Jorge");
    var_dump($agenda);
?>
```

Pop

La función `array_pop()` en PHP permite eliminar el último elemento de un array. Esta operación no solo modifica el array original, reduciendo su tamaño, sino que también devuelve el valor eliminado, lo cual puede ser útil si se desea almacenarlo o procesarlo.

Al aplicar `array_pop()`, el índice correspondiente al último elemento desaparece, y el array queda con un elemento menos. Si se ejecuta repetidamente, se puede vaciar el array por completo, uno a uno.

Este comportamiento es útil en estructuras tipo pila (stack), donde el último elemento añadido es el primero en salir (LIFO: Last In, First Out). Es una técnica habitual en algoritmos de retroceso o control de estados temporales.

```
005-array pop.php
<?php

    $agenda[0] = "Jose Vicente";
    $agenda[1] = "Juan";
    var_dump($agenda);
    echo "<br>";
    array_push($agenda,"Jorge");
    var_dump($agenda);
    echo "<br>";
    array_pop($agenda);
    var_dump($agenda);
?>
```

Splice

La función `array_splice()` permite eliminar uno o más elementos de un array en cualquier posición, no solo al final o al principio. Es una herramienta muy versátil, ya que además de eliminar elementos, también permite reemplazarlos o insertar nuevos en su lugar.

Para usarla, se indica el array original, el índice a partir del cual se desea modificar, y la cantidad de elementos a eliminar. Si se omite el tercer parámetro, eliminará todos los elementos desde la posición indicada hasta el final.

Tras aplicar `array_splice()`, el array se reordena automáticamente, reasignando los índices numéricos para mantener la continuidad. Esto la hace ideal para reestructurar arrays durante el procesamiento de datos o al aplicar filtros condicionales.

```
006-array splice.php
<?php

    $agenda[0] = "Jose Vicente";
    $agenda[1] = "Juan";
    var_dump($agenda);
    echo "<br>";
    array_push($agenda,"Jorge");
    var_dump($agenda);
    echo "<br>";
    array_pop($agenda);
    var_dump($agenda);
    echo "<br>";
    array_splice($agenda,0,1);
    var_dump($agenda);
?>
```

Indices alfanumericos

En PHP, los arrays no tienen por qué usar solo índices numéricos. También se pueden utilizar claves alfanuméricas para identificar cada elemento, lo que da lugar a arrays asociativos.

Este tipo de arrays permite establecer una relación clara entre una clave y su valor, como por ejemplo: `$agenda["nombre"] = "Jose Vicente";`. En este caso, `"nombre"` actúa como identificador del valor asignado, lo cual mejora la legibilidad y organización del código.

Los arrays asociativos son especialmente útiles cuando se necesita representar conjuntos de datos con campos nombrados, como registros de usuarios, configuraciones o parámetros. Además, funcionan de manera transparente con funciones de recorrido como `foreach()`, que permiten acceder a la vez a claves y valores.

```
007-Indices alfanumericos.php
<?php

    $agenda["nombre"] = "Jose Vicente";
    $agenda["email"] = "info@josevicentecarratala.com";
    $agenda["telefono"] = "6345646";
    var_dump($agenda);

?>
```

Multidimensionales

Los arrays multidimensionales en PHP son estructuras que contienen otros arrays como elementos. Esto permite organizar los datos en múltiples niveles, como si se tratara de una tabla con filas y columnas.

Por ejemplo, se puede crear un array donde cada elemento representa una persona, y cada persona es a su vez un array con claves como `"nombre"`, `"email"` y `"telefono"`. Esta organización es ideal para representar listas de objetos, resultados de bases de datos o configuraciones jerárquicas.

Para acceder a un valor dentro de un array multidimensional, se encadenan los índices, como en `$agenda[0]["nombre"]`. Esta estructura potencia la expresividad del código y permite manejar datos complejos de forma ordenada y eficiente.

```
008-arrays multidimensionales.php
<?php

    $agenda[0]["nombre"] = "Jose Vicente";
    $agenda[0]["email"] = "info@josevicentecarratala.com";
    $agenda[0]["telefono"] = "6345646";

    $agenda[1]["nombre"] = "Juan";
    $agenda[1]["email"] = "juan@josevicentecarratala.com";
    $agenda[1]["telefono"] = "6345646";

    $agenda[2]["nombre"] = "Jorge";
    $agenda[2]["email"] = "jorge@josevicentecarratala.com";
    $agenda[2]["telefono"] = "6345646";

    var_dump($agenda);

?>
```

3.7. Estructuras de control

Las estructuras de control en PHP permiten dirigir el flujo de ejecución del programa, tomando decisiones, repitiendo instrucciones o reaccionando ante ciertas condiciones. Estas estructuras son fundamentales para desarrollar aplicaciones dinámicas que respondan a distintos escenarios de forma automatizada.

Entre las más comunes se encuentran los bucles y las condicionales. Los bucles permiten ejecutar un bloque de código varias veces, mientras que las estructuras condicionales permiten decidir qué instrucciones ejecutar en función de ciertas condiciones.

PHP incluye diferentes tipos de bucles (`for`, `while`, `do...while`, `foreach`) y condicionales (`if`, `else`, `elseif`, `switch`), cada uno con su sintaxis y particularidades. Su correcto uso es clave para construir aplicaciones lógicas, eficientes y organizadas.

3.7.1. Bucle

Los bucles en PHP permiten repetir un bloque de instrucciones varias veces, lo que resulta útil para procesar listas de elementos, realizar cálculos repetitivos o generar contenido dinámico. PHP ofrece varios tipos de bucles, cada uno adecuado para distintos contextos: `for`, `while`, `do...while` y `foreach`.

El bucle `for` se utiliza cuando se conoce de antemano el número de repeticiones. El bucle `while` repite las instrucciones mientras se cumpla una condición. El `do...while` garantiza al menos una ejecución antes de comprobar la condición. Y `foreach`, pensado específicamente para recorrer arrays, permite acceder a cada elemento sin necesidad de manejar un contador.

Elegir el bucle adecuado para cada situación facilita la legibilidad del código y mejora su rendimiento.

For

El bucle `for` en PHP se utiliza cuando se conoce el número exacto de repeticiones que deben ejecutarse. Su sintaxis incluye tres partes: inicialización, condición y actualización. Por ejemplo, `for($i = 0; $i < 10; $i++)` ejecuta el bloque de instrucciones diez veces, incrementando el valor de `$i` en cada iteración.

Este tipo de bucle es ideal para recorrer arrays mediante índices numéricos, generar listados secuenciales o repetir instrucciones un número determinado de veces. Su estructura compacta y clara permite controlar todos los aspectos del ciclo en una sola línea, lo que lo convierte en una herramienta muy eficiente para tareas repetitivas controladas.

```
001-for.php
<?php

    for($dia = 1;$dia<=31;$dia++){
        echo "hoy es el dia ".$dia." del mes <br>";
    }

?>
```

While

El bucle `while` en PHP ejecuta un bloque de instrucciones mientras se cumpla una condición lógica. A diferencia del `for`, no requiere una estructura compacta con inicialización y actualización; en su lugar, evalúa una condición antes de cada iteración.

Para evitar bucles infinitos, es importante asegurarse de que la condición cambiará en algún momento dentro del cuerpo del bucle. Por ejemplo, si se desea repetir una acción mientras una variable no supere cierto valor, es necesario incrementar esa variable dentro del bucle.

Este tipo de bucle es útil cuando no se conoce de antemano cuántas veces se debe repetir la operación, y se quiere mantener la ejecución hasta que ocurra una condición específica.

```
002-while.php
<?php

    while($dia<=31){
        echo "hoy es el dia ".$dia." del mes <br>";
    }

?>
```

While inicio

Para que un bucle `while` funcione correctamente, es necesario que la variable utilizada en la condición esté inicializada antes de comenzar el bucle. Por ejemplo, si se quiere repetir una acción mientras `$dia <= 31`, primero se debe asignar un valor inicial a `$dia`, como `$dia = 1`.

Esta variable actúa como contador o condición de control, y si no se inicializa, el programa podría generar un error o comportarse de forma inesperada. Además, dentro del cuerpo del bucle es habitual modificar esta variable para asegurar que eventualmente la condición deje de cumplirse y el bucle termine.

Inicializar correctamente la variable permite que el ciclo tenga un punto de partida claro y que su evolución sea predecible.

```
003-while con variable de inicio.php
<?php
    $dia = 1;
    while($dia<=31){
        echo "hoy es el dia ".$dia." del mes <br>";
    }

?>
```

Incremento

Dentro del cuerpo del bucle `while`, es necesario modificar la variable que actúa como condición de control para evitar bucles infinitos. En muchos casos, esto se hace mediante un incremento, como `$dia++;`, que aumenta su valor en una unidad con cada iteración.

Este incremento permite que la condición que evalúa el bucle (`$dia <= 31`, por ejemplo) se acerque progresivamente a dejar de cumplirse. Si no se realiza este ajuste, el valor de la variable no cambiará, la condición siempre será verdadera y el bucle se ejecutará indefinidamente.

Incluir el incremento como parte del cuerpo del bucle es una práctica común cuando se utiliza `while`, ya que otorga al programador un control más explícito del ritmo de ejecución.

```
004-incremento.php
<?php
    $dia = 1;
    while($dia<=31){
        echo "hoy es el dia ".$dia." del mes <br>";
        $dia++;
    }

?>
```

Do while

El bucle `do...while` en PHP se diferencia de los demás en que garantiza que el bloque de instrucciones se ejecutará al menos una vez, independientemente de si la condición se cumple inicialmente o no. La evaluación de la condición se realiza después de la ejecución del bloque.

Su estructura comienza con la palabra clave `do`, seguida del bloque de código entre llaves, y concluye con la cláusula `while` y la condición entre paréntesis, terminada con punto y coma.

Este tipo de bucle es útil cuando se desea que el código se ejecute una primera vez antes de validar si se debe continuar repitiéndolo, como en menús interactivos o validaciones de entrada que requieren mostrarse al menos una vez.

```
005-do while.php
<?php
    $dia = 1;
    do{
        echo "te digo hola";
    }while($dia > 5);

?>
```

3.7.2. Condicionales

Las estructuras condicionales en PHP permiten ejecutar bloques de código distintos según se cumplan o no ciertas condiciones. Son fundamentales para dotar a los programas de comportamiento dinámico y adaptativo.

La forma más común de condicional es la instrucción `if`, que evalúa una expresión lógica y ejecuta un bloque de instrucciones si el resultado es verdadero. Si se desea ejecutar una alternativa cuando la condición no se cumple, se puede añadir una cláusula `else`. Para evaluar múltiples condiciones secuenciales, se utiliza `elseif`.

Otra estructura condicional disponible en PHP es `switch`, que permite comparar una misma variable contra varios valores posibles, lo que resulta útil cuando se manejan múltiples casos claramente definidos. Estas herramientas permiten controlar el flujo del programa y tomar decisiones basadas en datos.

If

La instrucción `if` en PHP permite ejecutar un bloque de código solo cuando se cumple una condición específica. La condición se escribe entre paréntesis, y si su resultado es `true`, se ejecuta el código encerrado entre llaves `{}`.

Por ejemplo, si se desea comprobar si una persona es menor de 30 años, se puede escribir una condición como `if($edad < 30)`. Si la variable `$edad` cumple esa condición, el bloque se ejecutará; en caso contrario, será ignorado.

Esta estructura es la base de toda lógica condicional en PHP, y puede combinarse con otras instrucciones para construir comportamientos complejos según distintas situaciones.

```
001-if.php
<?php
    $edad = 45;
    if($edad < 30){
        echo "Eres un joven";
    }

?>
```

Else

La cláusula `else` en PHP se utiliza para definir un bloque de código que se ejecutará cuando la condición del `if` no se cumpla. Es decir, si la expresión evaluada en el `if` resulta falsa, el control pasa directamente al bloque `else`.

Esta estructura permite establecer alternativas de ejecución claras, en las que solo uno de los dos bloques se ejecutará dependiendo del valor de la condición. Por ejemplo, si se comprueba si `$edad < 30`, el bloque `if` se ejecuta si la condición es verdadera, y el bloque `else` si es falsa.

El uso de `else` mejora la legibilidad del código y permite manejar casos contrarios sin necesidad de escribir una segunda condición.

```
002-else.php
<?php
    $edad = 45;
    if($edad < 30){
        echo "Eres un joven";
    }else{
        echo "ya no eres un joven";
    }

?>
```

Anidacion

La anidación de condicionales en PHP permite evaluar condiciones dentro de otras condiciones, creando estructuras más complejas y específicas. Esto se logra incluyendo una instrucción `if` dentro del bloque de otra instrucción `if` o `else`.

Este enfoque es útil cuando se necesita tomar decisiones en varios niveles, por ejemplo, distinguir entre distintas franjas de edad o validar múltiples propiedades de un mismo dato. Cada nivel de anidación evalúa una nueva condición, y solo se ejecuta si se ha cumplido la anterior.

Para mantener la claridad, es importante usar indentación adecuada y no abusar del número de niveles anidados, ya que un exceso puede dificultar la lectura y el mantenimiento del código.

```
003-anidacion.php
<?php
    $edad = 45;
    if($edad < 30){
        if($edad < 10){

            echo "Eres un niño";
        }else{
            echo "Eres un joven";
        }

    }else{
        if($edad < 70){

            echo "Ya no eres un joven";
        }else{
            echo "Eres un senior";
        }
    }

?>
```

Else if

La estructura `else if` en PHP permite encadenar múltiples condiciones que se evaluarán de forma secuencial. Si la primera condición `if` no se cumple, se prueba la siguiente con `else if`, y así sucesivamente hasta encontrar una verdadera. Si ninguna se cumple, se puede incluir un `else` final como caso por defecto.

Esta forma de condicional es útil cuando se tienen varios rangos o categorías, y se desea ejecutar un bloque distinto para cada uno. Cada `else if` representa una posibilidad adicional a considerar, evaluada solo si las anteriores fallan.

PHP evalúa las condiciones en orden, y en cuanto una de ellas devuelve `true`, se ejecuta su bloque y se ignoran las siguientes. Esto hace que la estructura sea eficiente y clara para tomar decisiones múltiples.

```
004-else if.php
<?php
    $edad = 45;
    if($edad < 10){
        echo "eres un niño";
    }else if($edad >= 10 && $edad < 30){
        echo "eres un joven";
    }else if($edad >= 30 && $edad < 70){
        echo "ya no eres un niño";
    }else if($edad >= 70){
        echo "eres un senior";
    }

?>
```

Switch

La estructura `switch` en PHP permite evaluar una variable frente a múltiples valores posibles, y ejecutar un bloque de código distinto para cada caso. Es una alternativa más limpia y ordenada al uso repetido de `else if` cuando se trabaja con comparaciones iguales.

Cada posible valor se define con la palabra clave `case`, seguida del valor a comparar y dos puntos. Si la variable coincide con ese valor, se ejecutan las instrucciones correspondientes hasta encontrar un `break`, que detiene la ejecución del `switch`.

Si ningún `case` coincide, se puede incluir una cláusula `default`, que actúa como caso por defecto. Esta estructura es especialmente útil cuando se espera que una variable tome uno de varios valores bien definidos, como días de la semana, comandos o estados.

```
005-switch.php
<?php
    $diadelasemana = "lunes";
   switch($diadelasemana){
       case "lunes":
           echo "hoy es el peor día de la semana";
           break;
        case "martes":
           echo "hoy es el segundo peor día de la semana";
           break;
        case "miercoles":
           echo "Ya estamos a mitad de semana";
            break;
        case "jueves":
           echo "Ya casi es viernes";
           break;
        case "viernes":
           echo "Por fin es viernes";
           break;
        case "sabado":
           echo "hoy es el mejor día de la semana";
           break;
        case "domingo":
           echo "Parece mentira que mañana ya sea lunes";
           break;

   }

?>
```

3.7.3. Try

En PHP, el uso de `try` permite manejar errores de forma controlada mediante el mecanismo de excepciones. Dentro del bloque `try`, se colocan las instrucciones que podrían generar un error en tiempo de ejecución, como divisiones por cero, acceso a archivos o conexiones a bases de datos.

Si se produce un error dentro del `try`, el flujo se interrumpe y se transfiere al bloque `catch`, donde se puede capturar el objeto de la excepción y reaccionar adecuadamente. Esto evita que el programa se detenga de forma abrupta y permite ofrecer una respuesta más informativa o realizar acciones correctivas.

El sistema de excepciones mejora la robustez del código, permitiendo separar la lógica principal del manejo de errores.

```
001-try.php
<?php

    echo 1/0;

    echo "Pero quiero que mi programa continúe";
?>
```

El bloque `catch` en PHP se utiliza junto al `try` para capturar y manejar excepciones cuando se produce un error en tiempo de ejecución. Dentro del `catch`, se define una variable que contiene el objeto de la excepción lanzada, lo cual permite acceder al mensaje de error y a otros detalles relevantes.

Por ejemplo, si dentro de un `try` se lanza una excepción con `throw new Exception("mensaje")`, el `catch(Exception $e)` intercepta esa excepción y permite acceder a su contenido mediante métodos como `$e->getMessage()`.

Este mecanismo proporciona una forma estructurada y segura de gestionar errores, evitando interrupciones inesperadas en la ejecución del programa y facilitando la implementación de respuestas adecuadas según el tipo de fallo detectado.

```
002-try.php
<?php
    try{
        $numerador = 1;
        $denominador = 0;


        if ($denominador === 0) {
            throw new Exception("No permitido.");
        }
    }catch(Exception $e){
>getMessage();
    }

    echo "Pero quiero que mi programa continúe";
?>
```

El bloque `finally` en PHP se utiliza junto con `try` y `catch` para definir un conjunto de instrucciones que se ejecutarán siempre, independientemente de si se ha producido una excepción o no. Es útil para liberar recursos, cerrar archivos, desconectar bases de datos o realizar cualquier tarea que deba ejecutarse pase lo que pase.

La estructura completa consta de un bloque `try` donde se ejecuta el código principal, un bloque `catch` donde se gestiona la excepción en caso de error, y finalmente un bloque `finally` donde se colocan las acciones finales.

El uso de `finally` garantiza que ciertas operaciones no se omitan, incluso si ocurre un error, lo cual es clave para asegurar la integridad del programa.

```
003-finally.php
<?php
    try{
        $numerador = 1;
        $denominador = 0;


        if ($denominador === 0) {
            throw new Exception("No permitido.");
        }
    }catch(Exception $e){
>getMessage();
    }finally{
        echo "Y la vida sigue";
    }

    echo "Pero quiero que mi programa continúe";
?>
```

3.8. Funciones

Las funciones en PHP permiten encapsular un conjunto de instrucciones bajo un nombre específico, lo que facilita su reutilización en distintos puntos del programa. Definir una función implica utilizar la palabra clave `function`, seguida del nombre que se le desea asignar, y un bloque de código entre llaves que contiene las instrucciones que ejecutará.

Este mecanismo permite organizar el código de forma modular, separar responsabilidades y reducir la repetición. Una vez definida, la función puede llamarse tantas veces como sea necesario, lo que mejora la eficiencia y la claridad del programa.

Las funciones pueden recibir parámetros para trabajar con datos externos y devolver resultados utilizando la instrucción `return`. Esta flexibilidad permite construir desde utilidades simples hasta bloques complejos de lógica reutilizable.

3.8.1. Declaración

Para declarar una función en PHP se utiliza la palabra clave `function`, seguida del nombre de la función y un par de paréntesis. Dentro de los paréntesis pueden colocarse parámetros, aunque no son obligatorios. El cuerpo de la función se escribe entre llaves `{}` e incluye las instrucciones que se ejecutarán al llamarla.

Por ejemplo, una función sin parámetros puede utilizarse para mostrar un mensaje o ejecutar un bloque fijo de código. Declarar funciones permite encapsular tareas repetitivas, mejorar la organización del programa y facilitar su mantenimiento. Una vez definida, la función puede llamarse por su nombre en cualquier parte del código.

```
001-declaracion de funciones.php
<?php
    function saluda(){
        echo "Yo te saludo";
    }
?>
```

3.8.2. Uso de la función

Una vez declarada una función, se puede utilizar llamándola por su nombre seguido de paréntesis. Si la función no requiere parámetros, los paréntesis se escriben vacíos. Esta llamada provoca la ejecución del bloque de instrucciones definido dentro de la función.

El uso de funciones mejora la claridad del código y evita repeticiones, ya que permite centralizar una tarea en un único lugar y reutilizarla cuantas veces sea necesario. También facilita el mantenimiento, ya que los cambios realizados en la definición de la función se aplican automáticamente en todos los lugares donde se invoque.

```
002-uso de la funcion.php
<?php
    function saluda(){
        echo "Yo te saludo";
    }

    saluda();
?>
```

3.8.3. Return en la función

Las funciones en PHP pueden devolver un valor utilizando la instrucción `return`. Al hacerlo, la ejecución de la función finaliza en ese punto y el valor especificado es entregado como resultado a quien haya llamado la función.

Este valor puede ser asignado a una variable, mostrado con `echo` o utilizado como parte de una expresión más compleja. El uso de `return` permite que las funciones no solo ejecuten tareas, sino que también generen resultados que puedan aprovecharse en otras partes del programa.

Además, aunque se puede incluir más código después de un `return`, no se ejecutará, ya que el flujo de la función se detiene en ese punto. Por ello, debe colocarse al final del bloque o en las ramas donde se desea finalizar la ejecución.

```
003-return.php
<?php
    function saluda(){
        return "Yo te saludo";
    }

    echo saluda();
?>
```

3.8.4. Parámetros en las funciones

Las funciones en PHP pueden recibir datos de entrada mediante parámetros. Estos parámetros se definen dentro de los paréntesis en la declaración de la función, separados por comas si son varios. Cada parámetro actúa como una variable interna que adopta el valor proporcionado al llamar la función.

Por ejemplo, si se define una función con un parámetro `$nombre`, al invocarla como `saluda("Ana")`, la variable `$nombre` dentro de la función tomará el valor `"Ana"`. Esto permite personalizar el comportamiento de la función en función de los datos recibidos.

El uso de parámetros hace que las funciones sean más versátiles y reutilizables, ya que pueden trabajar con distintos valores sin necesidad de reescribir el código.

```
004-parametros de las funciones.php
<?php
    function saluda($nombre){
        return "Hola, ".$nombre.", yo te saludo";
    }

    echo saluda();
?>
```

3.8.5. Parámetros

Al definir funciones en PHP, se pueden incluir uno o varios parámetros para proporcionar distintos valores de entrada. Cada parámetro se comporta como una variable interna que toma su valor en el momento de la llamada.

Cuando se utilizan múltiples parámetros, deben definirse en el orden en que se espera recibirlos y separarse con comas. Por ejemplo, una función que reciba un nombre y una edad podría declararse como `function saluda($nombre, $edad)`.

Al llamar la función, es importante respetar el mismo orden y número de argumentos. Esta estructura permite construir funciones flexibles que adapten su comportamiento según los datos proporcionados por el usuario o por otras partes del programa.

```
005-parametros.php
<?php
    function saluda($nombre){
        return "Hola, ".$nombre.", yo te saludo";
    }

    echo saluda("Jose Vicente");
?>
```

3.8.6. Multiples parametros

Una función puede aceptar múltiples parámetros para procesar varios datos al mismo tiempo. Estos parámetros se separan por comas en la declaración y permiten que la función reciba información diversa en cada llamada.

Por ejemplo, una función que saluda a una persona puede recibir su nombre y su edad como argumentos, lo que permite generar mensajes personalizados. Al invocar la función, se deben proporcionar los valores en el mismo orden en que fueron definidos.

Esta capacidad de manejar varios parámetros hace que las funciones sean herramientas potentes para construir lógica más compleja, permitiendo operaciones condicionadas, cálculos o generación de contenido dinámico en función de múltiples datos de entrada.

```
006-multiples parametros.php
<?php
    function saluda($nombre,$edad){
        return "Hola, ".$nombre.", que tienes ".$edad." años,
yo te saludo";
    }

    echo saluda("Jose Vicente",45);
?>
```

3.9. POO

La programación orientada a objetos (POO) en PHP permite estructurar el código en torno a entidades llamadas clases, que agrupan datos (propiedades) y comportamientos (métodos) relacionados. Esta aproximación facilita la organización del código, mejora su reutilización y hace más sencilla la evolución de aplicaciones complejas.

Las clases actúan como moldes a partir de los cuales se crean objetos. Cada objeto es una instancia concreta de una clase, con sus propios valores internos. Dentro de una clase se pueden definir propiedades para almacenar información y métodos para realizar acciones sobre ella.

PHP admite los principales conceptos de la POO, como encapsulación, herencia, polimorfismo y abstracción, lo que permite construir aplicaciones robustas, extensibles y fáciles de mantener.

3.9.1. Clases

Para declarar una clase en PHP se utiliza la palabra clave `class`, seguida del nombre que se le quiera dar. A continuación se abren llaves `{}` dentro de las cuales se definen las propiedades y métodos que formarán parte de esa clase.

Las propiedades representan los datos que cada objeto de esa clase almacenará, mientras que los métodos definen las acciones que pueden realizarse con dichos datos. Al crear una instancia (un objeto) a partir de esa clase, se hereda toda su estructura y funcionalidad, y se puede trabajar con ella de forma individualizada.

Definir clases es el primer paso para aplicar el paradigma orientado a objetos en PHP, permitiendo encapsular la lógica y manejar estructuras más complejas de forma ordenada y reutilizable.

```
001-Declaracion de una clase.php
<?php

    class Entrada{

    }

?>
```

3.9.2. Propiedades

Dentro de una clase en PHP se pueden definir propiedades, que son variables internas asociadas a cada objeto creado a partir de esa clase. Estas propiedades se declaran dentro de la clase utilizando palabras clave como `public`, `protected` o `private`, seguidas del símbolo `$` y el nombre de la propiedad.

Las propiedades permiten almacenar información propia de cada objeto, como un título, una fecha o un contenido. Para asignarles un valor al crear el objeto, se puede utilizar un método especial llamado constructor, o bien establecer los valores directamente después de la instanciación.

Cada objeto creado a partir de la clase tendrá sus propias copias de las propiedades, lo que permite trabajar con múltiples instancias de manera independiente.

```
002-Propiedades.php
<?php

    class Entrada{
        function __construct(){
            $this->titulo;
            $this->fecha;
            $this->autor;
            $this->contenido;
            $this->imagen;
        }

    }

?>
```

3.9.3. Instancias

Para crear una instancia de una clase en PHP se utiliza la palabra clave `new`, seguida del nombre de la clase y paréntesis. Este proceso genera un objeto basado en la estructura definida por la clase, con sus propias propiedades y métodos.

Cada vez que se crea una instancia, se ejecuta automáticamente el constructor de la clase, si está definido. A partir de ese momento, se puede acceder a las propiedades y métodos del objeto mediante el operador de flecha `->`.

Crear varias instancias de una misma clase permite representar diferentes entidades con la misma estructura, cada una con sus propios datos. Esto es fundamental en la programación orientada a objetos, ya que proporciona modularidad y flexibilidad en la gestión de datos y comportamientos.

```
003-Instanciar la clase.php
<?php

    class Entrada{
        function __construct(){
            $this->titulo = "";
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = "";
            $this->imagen = "";
        }

    }

    $entrada1 = new Entrada();
    var_dump($entrada1);
    $entrada2 = new Entrada();
    var_dump($entrada2);
?>
```

3.9.4. Acceso

Una vez que se ha creado una instancia de una clase, se puede acceder a sus propiedades utilizando el operador de flecha `->` seguido del nombre de la propiedad. Esto permite consultar o modificar los valores almacenados dentro del objeto.

Por ejemplo, si una clase tiene una propiedad llamada `$titulo`, se puede acceder a ella con `$miObjeto->titulo`. Para asignarle un nuevo valor, se utiliza una expresión como `$miObjeto->titulo = "Nuevo título";`.

Este mecanismo permite manipular de forma directa la información contenida en cada objeto, siempre que las propiedades sean públicas o se acceda a ellas mediante métodos específicos (`get` y `set`) en el caso de propiedades privadas o protegidas.

```
004-Acceder a propiedades.php
<?php

    class Entrada{
        function __construct(){
            $this->titulo = "";
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = "";
            $this->imagen = "";
        }

    }

    $entrada1 = new Entrada();
    $entrada1->titulo = "Título de la primera entrada";
    echo $entrada1->titulo;

?>
```

3.9.5. Privado

En PHP, se puede restringir el acceso a las propiedades de una clase mediante el modificador `private`. Esto significa que dichas propiedades solo podrán ser accedidas o modificadas desde dentro de la propia clase, y no desde fuera ni desde instancias externas.

El uso de propiedades privadas fomenta la encapsulación, un principio clave de la programación orientada a objetos, que busca proteger el estado interno del objeto. Para permitir el acceso controlado a estas propiedades, se suelen utilizar métodos públicos llamados `get` y `set`.

Si se intenta acceder directamente a una propiedad privada desde fuera de la clase, PHP generará un error. Por eso es recomendable usar este tipo de protección cuando se desea mantener el control sobre cómo y cuándo se accede a los datos de un objeto.

```
005-propiedades y metodos privados.php
<?php

    class Entrada{
        private $titulo;

        function __construct(){
            $this->titulo = "";
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = "";
            $this->imagen = "";
        }

    }

    $entrada1 = new Entrada();
    $entrada1->titulo = "Título de la primera entrada";
    echo $entrada1->titulo;

?>
```

3.9.6. Set y get

Los métodos `set` y `get` permiten acceder y modificar propiedades privadas de una clase desde fuera de ella, de forma controlada. El método `set` se utiliza para asignar un valor a una propiedad, mientras que el método `get` sirve para recuperar su valor.

Estos métodos suelen incluir lógica adicional para validar o transformar los datos antes de asignarlos o devolverlos. Por ejemplo, se puede comprobar si el valor recibido cumple ciertos requisitos antes de almacenarlo, y devolver un mensaje de error si no es así.

El uso de `set` y `get` refuerza la encapsulación y permite mantener la coherencia de los datos internos del objeto, ya que se evita el acceso directo a las propiedades y se centraliza su manipulación en funciones específicas.

```
006-Metodos set y get.php
<?php

    class Entrada{
        private $titulo;

        function __construct(){
            $this->titulo = "";
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = "";
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            $this->titulo = $nuevotitulo;
        }
        function getTitulo(){
            return $this->titulo;
        }
    }

    $entrada1 = new Entrada();
    $entrada1->setTitulo("Título de la primera entrada");
    echo $entrada1->getTitulo();

?>
```

3.9.7. Validación

Los métodos `set` en PHP pueden incluir validación para asegurar que los datos asignados a una propiedad cumplen ciertos criterios. Esta técnica permite proteger la integridad del objeto y evitar estados inconsistentes.

Por ejemplo, se puede verificar que la longitud de una cadena sea adecuada antes de asignarla. Si el valor no cumple la condición, se puede asignar un valor alternativo o emitir un mensaje de error. Esta lógica de

validación se coloca dentro del método `set`, antes de modificar la propiedad.

De este modo, cada vez que se intenta cambiar el valor de una propiedad mediante su método correspondiente, se garantiza que el dato introducido es aceptable, mejorando la robustez y fiabilidad del código.

```
007-validacion.php
<?php

    class Entrada{
        private $titulo;

        function __construct(){
            $this->titulo = "";
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = "";
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
    }

    $entrada1 = new Entrada();
    $entrada1->setTitulo("Título");
    echo $entrada1->getTitulo();

?>
```

3.9.8. Constructor con parámetros

Un constructor con parámetros en PHP permite inicializar las propiedades de un objeto en el momento de su creación. Para ello, se define el método especial `__construct()` con una lista de parámetros, que serán los valores iniciales a asignar a las propiedades del objeto.

Dentro del constructor, estos parámetros se utilizan para establecer el estado inicial de la instancia, lo que facilita la creación de objetos completamente configurados desde el principio. Por ejemplo, se puede pasar un título y un contenido para asignarlos directamente a las propiedades correspondientes.

Este enfoque ahorra pasos y mejora la claridad del código, ya que cada objeto se construye de forma explícita con los datos necesarios. Además, se puede seguir utilizando métodos `set` para modificar los valores posteriormente si se desea aplicar validaciones u otras reglas de negocio.

```
008-constructor con parametros.php
<?php

    class Entrada{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
    }

    $entrada1 = new Entrada("Título de la entrada","Contenido
de la entrada");

    echo $entrada1->getTitulo();
    $entrada1->setTitulo("Titulo nuevo");
    echo $entrada1->getTitulo();

?>
```

3.9.9. Clase pagina

Una clase puede contener múltiples métodos además del constructor, y puede coexistir junto a otras clases dentro del mismo archivo. Esto permite definir estructuras completas y coherentes, en las que cada clase

representa un tipo distinto de entidad, con sus propios atributos y comportamientos.

Por ejemplo, se puede definir una clase `Pagina` con sus propiedades y métodos, y otra clase `Entrada` con una estructura similar. Ambas pueden compartir un constructor con parámetros, métodos `set` para validar datos y métodos `get` para acceder a las propiedades privadas.

Tener varias clases definidas en un mismo archivo facilita la organización del código cuando están estrechamente relacionadas, como ocurre en aplicaciones que trabajan con distintos tipos de contenidos. Cada objeto creado a partir de estas clases puede funcionar de forma autónoma, reutilizando los métodos comunes y respetando las validaciones definidas.

```
009-clase pagina.php
<?php
    class Pagina{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
    }
    class Entrada{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
```

```
            return $this->titulo;
        }
    }

    $entrada1 = new Entrada("Título de la entrada","Contenido
de la entrada");

    echo $entrada1->getTitulo();
    $entrada1->setTitulo("Titulo nuevo");
    echo $entrada1->getTitulo();

?>
```

3.9.10. Herencia

La herencia en PHP permite que una clase adquiera las propiedades y métodos de otra, lo que favorece la reutilización de código y la organización jerárquica. Para declarar que una clase hereda de otra, se utiliza la palabra clave `extends`.

Por ejemplo, una clase `Entrada` puede heredar de una clase `Publicacion`, accediendo automáticamente a todos sus atributos y comportamientos públicos o protegidos, sin necesidad de redefinirlos. Esto permite crear especializaciones de una clase base, manteniendo un núcleo común de funcionalidad.

Además, es posible sobrescribir métodos heredados o añadir otros nuevos en la clase hija. Esta flexibilidad permite construir sistemas complejos a partir de componentes básicos, con un alto grado de coherencia y eficiencia en el diseño del código.

```
010-Herencia.php
<?php
    class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

    $entrada1 = new Entrada("Título de la entrada","Contenido
de la entrada");

    $entrada2 = new Publicacion("Título de la
entrada","Contenido de la entrada");

?>
```

3.9.11. Abstracción

La abstracción en PHP permite definir clases que no pueden ser instanciadas directamente, pero que sirven como modelo para otras clases. Para declarar una clase abstracta se utiliza la palabra clave `abstract` antes de `class`.

Una clase abstracta puede contener propiedades y métodos implementados, así como métodos abstractos, que se definen sin cuerpo y deben ser implementados obligatoriamente por las clases hijas. Esto obliga a las subclases a completar la funcionalidad, garantizando una estructura mínima común.

El uso de clases abstractas es útil cuando se desea establecer una base general para un conjunto de clases relacionadas, sin permitir la creación de objetos genéricos que carezcan de sentido por sí mismos. Así se refuerza la coherencia y se facilita la expansión del sistema mediante herencia.

```
011-Abstracto.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

    $entrada1 = new Entrada("Título de la entrada","Contenido
de la entrada");



?>
```

3.10. Includes

PHP permite dividir el código en archivos separados y reutilizarlos mediante instrucciones como `include` o `require`. Esto es especialmente útil cuando se quiere organizar el proyecto en módulos, como clases, plantillas o configuraciones externas.

La instrucción `include` inserta el contenido de otro archivo en el punto donde se llama. Si el archivo no se encuentra, PHP genera una advertencia pero continúa ejecutando el script. Es ideal para añadir código que no sea esencial para el funcionamiento crítico.

Este mecanismo promueve la reutilización y el mantenimiento del código, ya que permite definir elementos como clases o funciones una sola vez y utilizarlos en distintos lugares sin repetir su contenido. También facilita mantener una estructura clara y modular en proyectos de mayor tamaño.

```
clase.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

3.10.1. Include

La instrucción `include` en PHP se utiliza para incorporar el contenido de otro archivo dentro del script actual. Esto permite dividir el código en módulos reutilizables, como clases, funciones o fragmentos HTML.

Cuando se ejecuta `include "archivo.php";`, PHP lee el contenido de ese archivo y lo inserta en ese punto del código. Si el archivo no existe o hay un error al incluirlo, se genera una advertencia, pero el programa continúa su ejecución.

Esta técnica es común en proyectos organizados, donde se separan las definiciones de clases o los bloques de interfaz en archivos distintos, facilitando la reutilización, el mantenimiento y la claridad del código.

```
001-Include.php
<?php

    include "clase.php";

    $entrada1 = new Entrada("Título de la entrada","Contenido
de la entrada");
    echo $entrada1->getTitulo();

?>
```

3.10.2. Fallo

Si se utiliza `include` para cargar un archivo que no existe o que contiene errores de sintaxis, PHP generará una advertencia (`Warning`) y continuará ejecutando el resto del script. Esto puede provocar fallos posteriores si el código esperaba que el contenido del archivo estuviera disponible.

Este comportamiento hace que `include` sea adecuado cuando el archivo externo no es imprescindible para el funcionamiento del programa. Sin embargo, si el archivo es esencial, se recomienda utilizar `require`, ya que detendrá la ejecución inmediatamente en caso de error, evitando comportamientos inesperados.

```
002-fallo en include.php
<?php

    include "clase2.php";

    $entrada1 = new Entrada("Título de la entrada","Contenido
de la entrada");
    echo $entrada1->getTitulo();

?>
```

3.10.3. Require

La instrucción `require` en PHP funciona de forma similar a `include`, pero con una diferencia importante: si el archivo especificado no se encuentra o contiene errores, la ejecución del script se detiene inmediatamente.

Esto lo convierte en la opción preferida cuando el archivo es fundamental para el correcto funcionamiento del programa, como la inclusión de clases esenciales o archivos de configuración.

Utilizar `require` garantiza que el código no continuará ejecutándose en un estado incompleto o incorrecto, lo que aporta mayor seguridad y control en aplicaciones que dependen de estructuras bien definidas desde el inicio.

```
003-require.php
<?php

    require "clase.php";

    $entrada1 = new Entrada("Título de la entrada","Contenido
de la entrada");
    echo $entrada1->getTitulo();

?>
```

3.11. Librerias

PHP ofrece un amplio conjunto de librerías que extienden sus capacidades, permitiendo realizar tareas comunes de forma más sencilla y eficiente. Estas librerías abarcan funcionalidades como manejo de fechas, codificación y decodificación de datos, gestión de sesiones, acceso a bases de datos, envío de correos y mucho más.

El uso de librerías facilita el desarrollo al evitar que el programador tenga que implementar desde cero funciones que ya están optimizadas y probadas. Algunas vienen integradas con el lenguaje por defecto, mientras que otras pueden añadirse mediante instalación de extensiones o frameworks externos.

Integrar librerías en un proyecto no solo acelera la programación, sino que también promueve buenas prácticas y mejora la compatibilidad con otros sistemas.

3.11.1. Fechas

PHP proporciona funciones nativas para trabajar con fechas y horas, permitiendo obtener y mostrar información temporal en distintos

formatos. La función más común es `date()`, que recibe un formato como parámetro y devuelve una cadena con la fecha actual según ese patrón.

Por ejemplo, se pueden obtener el año (`Y`), mes (`m`), día (`d`), hora (`H`), minutos (`i`), segundos (`s`) o la marca de tiempo Unix (`U`). Estas letras se combinan dentro del string del formato para construir salidas personalizadas como `date('Y-m-d H:i:s')`.

Estas funciones son útiles para registrar eventos, ordenar datos por fecha, mostrar la hora actual al usuario o generar nombres únicos basados en el tiempo. Además, PHP permite modificar zonas horarias y trabajar con objetos `DateTime` para una gestión más avanzada.

```
001-date.php
<?php

    echo date('Y');
    echo "<br>";
    echo date('m');
    echo "<br>";
    echo date('d');
   echo "<br>";
    echo date('H');
   echo "<br>";
    echo date('i');
   echo "<br>";
    echo date('s');
   echo "<br>";
    echo date('U');
   echo "<br>";


?>
```

3.11.2. Codificacion

PHP incluye funciones para codificar y decodificar datos en distintos formatos, siendo `base64_encode()` y `base64_decode()` dos de las más utilizadas. Estas funciones permiten transformar cadenas de texto en una representación ASCII segura para transmisión o almacenamiento, y luego restaurarlas a su forma original.

Codificar en base64 es útil cuando se necesita incluir datos binarios en entornos que solo aceptan texto, como correos electrónicos, URLs o archivos JSON. La función `base64_encode()` toma una cadena y la convierte en su equivalente codificado, mientras que `base64_decode()` realiza la operación inversa.

Estas herramientas facilitan la manipulación de datos que deben conservarse íntegros al pasar por medios que no admiten caracteres especiales o binarios.

```
001-base64.php
<?php

    $nombre = "Jose Vicente";
    echo $nombre;
    echo "<br>";
    $codificado = base64_encode($nombre);
    echo $codificado;
    echo "<br>";
    $descodificado = base64_decode($codificado);
    echo $descodificado;
    echo "<br>";
?>
```

Además de las funciones básicas de codificación y decodificación, es común implementar funciones personalizadas que aplican múltiples ciclos de codificación o decodificación, incrementando la seguridad o la complejidad del proceso.

Por ejemplo, una función puede codificar una cadena varias veces seguidas usando `base64_encode()` en un bucle, y otra función correspondiente la decodifica aplicando `base64_decode()` el mismo número de veces. Esto dificulta la interpretación directa del contenido sin conocer el número de iteraciones.

Este enfoque puede utilizarse en sistemas sencillos de ofuscación de datos, aunque no debe considerarse un método de cifrado seguro para proteger información sensible.

```
002-funcion codificar.php
<?php

    function codifica($cadena,$veces){
        $interno = $cadena;
        for($i = 0;$i<$veces;$i++){
            $interno = base64_encode($interno);
        }
        return $interno;
    }

    function descodifica($cadena,$veces){
        $interno = $cadena;
        for($i = 0;$i<$veces;$i++){
            $interno = base64_decode($interno);
        }
        return $interno;
    }

    $nombre = "Jose Vicente";
    $codificado = codifica($nombre,7);
    $descodificado = descodifica($codificado,7);
    echo $descodificado;
?>
```

PHP también ofrece funciones para generar hashes, que son valores únicos y fijos derivados de una cadena de entrada. Dos de las funciones más utilizadas son `md5()` y `sha1()`, que calculan un resumen criptográfico de 128 y 160 bits respectivamente.

Los hashes se emplean para verificar integridad de datos, almacenar contraseñas de forma segura o identificar información de manera unívoca sin revelar su contenido original. Al aplicar `md5()` o `sha1()` a una cadena, se obtiene una secuencia alfanumérica fija que cambia si la entrada se modifica.

Aunque `md5()` y `sha1()` ya no se consideran seguros para proteger contraseñas debido a vulnerabilidades, siguen siendo útiles para comprobaciones rápidas y generación de identificadores. Para seguridad

avanzada, se recomiendan algoritmos más robustos como `bcrypt` o `argon2`.

```
003-Hasheo.php
<?php

    $nombre = "Jose Vicente";
    $hash = md5($nombre);
    echo "<br>";
    echo $hash;
?>
```

Las funciones de hash como `md5()` y `sha1()` generan valores únicos para una cadena de texto dada, lo que permite identificar o verificar datos sin revelar su contenido original. Estas funciones son deterministas: la misma entrada siempre produce el mismo hash.

Estos hashes se utilizan comúnmente para comparar archivos, validar integridad o almacenar contraseñas de forma segura cuando se combinan con técnicas adicionales como el salting. Sin embargo, debido a vulnerabilidades conocidas, no se recomienda su uso para proteger información sensible sin complementos adicionales.

La salida de estas funciones es una cadena hexadecimal que representa el resumen criptográfico, y puede utilizarse en bases de datos, sistemas de autenticación o mecanismos de control de versiones.

```
004-SHA1.php
<?php

    $nombre = "Jose Vicente";
    $hash = md5($nombre);
    echo "<br>";
    echo $hash;
    $hash2 = sha1($nombre);
    echo "<br>";
    echo $hash2;
?>
```

PHP permite combinar estas funciones de hash con ciclos o procesos recursivos para generar listas extensas de valores hash a partir de combinaciones de caracteres. Este enfoque es útil en análisis criptográficos o generación de tablas de búsqueda.

Por ejemplo, se puede crear un script que genere todas las combinaciones posibles de caracteres de un conjunto dado, y para cada una calcule su hash con `sha1()`. Estos resultados se almacenan o muestran para su posterior uso.

Aunque este tipo de técnicas consume recursos y tiempo, es útil para tareas como la generación de tablas arcoíris o pruebas de resistencia en sistemas de seguridad, siempre dentro de un contexto controlado y ético.

```
005-Libreriadehashes.php
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <?php

            $letrasynumeros =
"abcdefghijklmnopqrstuvwxyzñçáéíóúàèìòù1234567890$";
            $cadena = "";
            for($i = 0;$i<strlen($letrasynumeros);$i++){
                $cadena .=
$letrasynumeros[$i].",".sha1($letrasynumeros[$i]).."<br>";
            }
            echo $cadena;
        ?>
        </body>
</html>
```

Las librerías de hashes en PHP permiten generar resúmenes criptográficos para conjuntos de caracteres o cadenas completas, facilitando su uso en aplicaciones de seguridad o validación.

Por ejemplo, se puede construir una lista que combine cada carácter de un conjunto con su correspondiente hash SHA-1, mostrando tanto el original como su resumen. Esto es útil para crear referencias rápidas o validar datos.

Estos procesos pueden automatizarse con bucles que recorran los caracteres y calculen sus hashes, mostrando los resultados en formato legible, como tablas o listas HTML, para facilitar su interpretación y uso.

```
006-recursivo.php
<html>
<head>
    <meta charset="utf-8">
</head>
<body>
    <?php
        $letrasynumeros =
"abcdefghijklmnopqrstuvwxyz1234567890$";
        $cadena = "";

        for ($i = 1; $i <= 3; $i++) {  // You can change '3'
to generate longer combinations
            generateCombinations($letrasynumeros, "", $i);
        }

        function generateCombinations($charset, $prefix,
$length) {
            global $cadena;
            if ($length === 0) {
                $cadena .= $prefix . "," . sha1($prefix) .
"<br>";
                return;
            }
            for ($i = 0; $i < strlen($charset); $i++) {
                generateCombinations($charset, $prefix .
$charset[$i], $length - 1);
            }
        }

        echo $cadena;
    ?>
</body>
</html>
```

3.12. Sesiones

Las sesiones en PHP permiten mantener información persistente entre distintas peticiones del mismo usuario, posibilitando la creación de

aplicaciones web con estado, como sistemas de autenticación o carritos de compra.

Al iniciar una sesión con `session_start()`, PHP genera un identificador único que se almacena en una cookie del navegador. Este identificador permite asociar los datos guardados en `$_SESSION` con el usuario correspondiente en futuras solicitudes.

Las variables de sesión actúan como un almacén temporal en el servidor, accesible desde cualquier página durante la duración de la sesión. Esto facilita compartir información personalizada y mantener el contexto de la interacción del usuario.

3.12.1. Problema

Cuando se intenta compartir una variable entre diferentes páginas sin utilizar sesiones, el valor no se mantiene porque cada petición HTTP es independiente y no conserva el estado de las variables declaradas en una página anterior.

Por ejemplo, si en una página se define `$edad = 45;` y se navega a otra página que intenta acceder a esa variable, ésta no estará definida a menos que se utilicen mecanismos de persistencia como sesiones, cookies o paso de parámetros mediante URL o formularios.

Este comportamiento es inherente al protocolo HTTP y requiere que el programador implemente métodos para conservar el estado cuando sea necesario mantener datos entre distintas páginas o interacciones.

```
001-origen.php
<?php

    $edad = 45;

?>
<a href="002-destino.php">Vamos al destino</a>
```

Para solucionar el problema de mantener valores entre diferentes páginas en PHP, se utilizan las sesiones. Al llamar a `session_start()`, se inicia una sesión o se continúa una existente, lo que permite almacenar datos en el arreglo superglobal `$_SESSION`.

Por ejemplo, en la primera página se puede asignar un valor a una variable de sesión como `$_SESSION['edad'] = 45;`. Luego, en la página destino, después de llamar también a `session_start()`, se puede acceder a este valor con `echo $_SESSION['edad'];`.

Las sesiones permiten mantener información personalizada y persistente durante toda la interacción del usuario con la aplicación, superando la naturaleza sin estado del protocolo HTTP.

```
002-destino.php
<?php

    echo $edad;

?>
```

3.12.2. Solución

Para mantener y compartir datos entre páginas en PHP, se utiliza la supervariable de sesión `$_SESSION`. Primero, en la página origen, se

debe iniciar la sesión con `session_start()` y asignar valores a `$_SESSION`, por ejemplo, `$_SESSION['edad'] = 45;`.

En la página destino, también se debe llamar a `session_start()` para continuar la sesión y luego acceder a los datos almacenados, como `echo $_SESSION['edad'];`.

Este mecanismo permite conservar información específica del usuario durante la navegación, facilitando funcionalidades como autenticación, carritos de compra o preferencias personalizadas en aplicaciones web.

```
001-origen.php
<?php

    session_start();
    $_SESSION['edad'] = 45;

?>
<a href="002-destino.php">Vamos al destino</a>
```

Para acceder a los datos almacenados en una sesión en PHP, es necesario llamar a `session_start()` al inicio del script. Esto asegura que se reanude la sesión activa y se pueda acceder a la supervariable global `$_SESSION`.

Una vez iniciada la sesión, se puede leer cualquier dato almacenado previamente, por ejemplo, `echo $_SESSION['edad'];`. Si el dato existe, se mostrará su valor; de lo contrario, se deberá manejar el caso para evitar errores.

Este acceso permite mantener información persistente del usuario durante toda la interacción con la aplicación web, a pesar de la naturaleza sin estado del protocolo HTTP.

```
002-destino.php
<?php
    session_start();

    echo $_SESSION['edad'];

?>
```

3.13. Super Variables

Las supervariables en PHP son arrays predefinidos que contienen información útil para la gestión de la aplicación y la interacción con el usuario. Se accede a ellas en cualquier parte del código sin necesidad de declararlas previamente.

Algunas supervariables comunes son `$_GET`, `$_POST`, `$_SESSION`, `$_SERVER` y `$_ENV`. Cada una tiene un propósito específico: por ejemplo, `$_GET` recoge datos enviados por URL, mientras que `$_SESSION` almacena información persistente entre páginas.

Estas supervariables facilitan el manejo de entradas, salidas, configuraciones del servidor y otras funcionalidades esenciales para el desarrollo web dinámico con PHP.

3.13.1. GET

La supervariable `$_GET` en PHP contiene un array asociativo con los datos enviados mediante el método GET, es decir, los parámetros que se incluyen en la URL de la petición.

Por ejemplo, si la URL es `pagina.php?nombre=Juan`, entonces `$_GET['nombre']` tendrá el valor `"Juan"`. Esto permite capturar información enviada por el usuario o desde enlaces y utilizarla dentro del script.

Esta variable es fundamental para construir aplicaciones que reaccionen a parámetros externos, como filtros, búsquedas o páginas dinámicas. Es importante validar y sanitizar los datos obtenidos para evitar vulnerabilidades de seguridad.

```
001-GET.php
<?php

    var_dump($_GET);

?>
```

3.13.2. Session

La supervariable `$_SESSION` en PHP es un array asociativo que almacena datos específicos de una sesión iniciada por un usuario. Estos datos permanecen disponibles a lo largo de múltiples páginas y solicitudes mientras la sesión esté activa.

Para utilizar `$_SESSION`, es necesario llamar a `session_start()` al inicio del script. A partir de ese momento, se pueden asignar valores a esta supervariable, como `$_SESSION['nombre'] = "Jose Vicente";`, y acceder a ellos en otras páginas durante la sesión.

Este mecanismo es esencial para mantener el estado y personalizar la experiencia del usuario en aplicaciones web, permitiendo guardar información como autenticación, preferencias o datos temporales.

```
002-Session.php
<?php
    session_start();
    $_SESSION['nombre'] = "Jose Vicente";
    $_SESSION['apellidos'] = "Carratala Sanchis";
    var_dump($_SESSION);

?>
```

3.13.3. Server

La supervariable `$_SERVER` en PHP es un array asociativo que contiene información sobre el servidor web y la ejecución actual del script. Proporciona datos como las cabeceras HTTP, rutas, nombres de archivos, direcciones IP, y otros detalles relacionados con la petición y el entorno.

Por ejemplo, `$_SERVER['HTTP_HOST']` devuelve el nombre del host desde donde se accede al script, y `$_SERVER['REQUEST_METHOD']` indica el método HTTP utilizado (GET, POST, etc.).

Esta supervariable es fundamental para obtener contexto sobre la solicitud del usuario y adaptar el comportamiento del servidor o del script según las condiciones específicas del entorno.

```
003-Server.php
<?php

    var_dump($_SERVER);

?>
```

3.13.4. Environment

La supervariable `$_ENV` en PHP es un array asociativo que contiene las variables de entorno del sistema operativo en el que se ejecuta el servidor web. Estas variables proporcionan información sobre la configuración del entorno, como rutas, configuración regional, usuario actual y otras variables definidas en el sistema.

Se pueden utilizar para acceder a parámetros de configuración externa al script, facilitando la adaptación del programa según el entorno en que se despliega sin necesidad de modificar el código fuente.

El contenido de `$_ENV` depende de la configuración del servidor y del sistema operativo, por lo que su disponibilidad y valores pueden variar entre diferentes entornos.

```
004-Environment.php
<?php

    var_dump($_ENV);

?>
```

3.14. Formularios

Los formularios en PHP son elementos HTML que permiten al usuario enviar datos al servidor para ser procesados. Utilizan los métodos `GET` o

POST para transmitir la información introducida en campos como texto, botones, listas o casillas de verificación.

Al enviar un formulario, los datos se reciben en PHP a través de las supervariables $_GET o $_POST, dependiendo del método usado. Esto permite capturar y utilizar la información proporcionada para generar contenido dinámico, realizar cálculos, almacenar en bases de datos, o cualquier otra acción necesaria.

El uso de formularios es fundamental para interactuar con el usuario y construir aplicaciones web dinámicas y personalizadas.

3.14.1. Formulario

Un formulario básico en HTML para PHP consta de la etiqueta `<form>` con los atributos `method` y `action`. El atributo `method` indica cómo se enviarán los datos (GET o POST), y `action` especifica la URL o el archivo que procesará la información enviada.

Dentro del formulario se colocan elementos de entrada como `<input>`, `<textarea>`, `<select>`, entre otros, que permiten al usuario introducir datos. Por ejemplo, un campo de texto para capturar un nombre se define con `<input type="text" name="nombre">`.

Al enviar el formulario, los datos se envían al servidor para ser procesados por el script PHP indicado en `action`, facilitando la interacción y el procesamiento dinámico de la información.

```
001-formulario.php
<form method="POST" action="?">
    <input type="text" name="nombre">
    <input type="submit">
</form>
```

3.14.2. Atrapo datos

Para capturar los datos enviados desde un formulario en PHP, se utilizan las supervariables `$_POST` o `$_GET`, según el método de envío configurado. Estas variables son arrays asociativos que contienen los valores introducidos por el usuario, accesibles mediante el nombre asignado a cada campo.

Por ejemplo, si el formulario incluye un campo con `name="nombre"`, el valor ingresado se puede obtener en PHP con `$_POST['nombre']` si se usó el método POST, o con `$_GET['nombre']` si fue GET.

Es importante verificar si el dato existe utilizando funciones como `isset()` para evitar errores y validar o sanitizar la información antes de procesarla, garantizando la seguridad y la integridad de la aplicación.

```
002-atrapo datos.php
<?php
    echo "Tu nombre es: ".$_POST['nombre']."<br>";
?>
<form method="POST" action="?">
    <input type="text" name="nombre">
    <input type="submit">
</form>
```

3.14.3. Isset

Para asegurarse de que un dato enviado desde un formulario existe antes de usarlo, se utiliza la función `isset()` en PHP. Esta función verifica si una variable está definida y no es `null`.

Por ejemplo, antes de acceder a `$_POST['nombre']`, se puede hacer una comprobación como `if(isset($_POST['nombre']))` para evitar errores en caso de que el campo no haya sido enviado o esté vacío.

Usar `isset()` es una buena práctica para controlar la validez de los datos recibidos y prevenir fallos en la ejecución del programa debido a referencias a variables inexistentes.

```
003-isset.php
<?php
    if(isset($_POST['nombre'])){
        echo "Tu nombre es: ".$_POST['nombre']."<br>";
   }
?>
<form method="POST" action="?">
    <input type="text" name="nombre">
    <input type="submit">
</form>
```

3.14.4. Get

La supervariable `$_GET` en PHP recoge los datos enviados mediante el método GET, es decir, los parámetros que se incluyen en la URL de la petición. Para acceder a un valor específico, se utiliza el nombre del parámetro como índice, por ejemplo, `$_GET['nombre']`.

Antes de usar estos datos, es recomendable comprobar su existencia con `isset()` para evitar errores. Esto es especialmente importante porque el usuario puede modificar o eliminar parámetros en la URL manualmente.

Los datos obtenidos a través de `$_GET` se utilizan para generar contenido dinámico, realizar búsquedas, filtrar resultados o personalizar la respuesta según la información enviada en la URL.

```
004-get.php
<?php
    if(isset($_GET['nombre'])){
        echo "Tu nombre es: ".$_GET['nombre']."<br>";
   }
?>
<form method="GET" action="?">
    <input type="text" name="nombre">
    <input type="submit">
</form>
```

3.15. Persistencia

La persistencia en PHP se refiere a la capacidad de almacenar datos de manera que permanezcan disponibles entre diferentes ejecuciones del programa o entre distintas solicitudes web. Esto es esencial en aplicaciones web para mantener información del usuario, configuraciones, registros o cualquier dato que no deba perderse al cerrar la sesión o cambiar de página.

Existen diferentes mecanismos de persistencia, como el uso de archivos, bases de datos, cookies o sesiones. Cada uno tiene sus características y casos de uso específicos, y la elección depende de la naturaleza de los datos, la seguridad requerida y la arquitectura de la aplicación.

Gestionar correctamente la persistencia garantiza una experiencia de usuario coherente y permite construir aplicaciones complejas y funcionales que mantienen su estado a lo largo del tiempo.

3.15.1. Archivos

PHP permite almacenar información en archivos para conservar datos entre distintas ejecuciones. Para escribir en un archivo, se utiliza la

función `fopen()` con el modo `'a'` (append) para añadir contenido al final del archivo sin borrar lo existente.

Una vez abierto el archivo, se emplea `fwrite()` para escribir datos, y finalmente `fclose()` para cerrar el archivo y liberar los recursos. Este proceso es útil para registrar información como logs, entradas de usuario o cualquier dato que se quiera preservar.

Es importante validar y sanitizar los datos antes de almacenarlos, y gestionar adecuadamente los permisos de archivo para evitar problemas de seguridad o pérdida de información.

```
001-escribir.php
<?php
    if(isset($_GET['nombre'])){
        echo "Tu nombre es: ".$_GET['nombre']."<br>";
        $archivo = fopen("documento.txt",'a');
        fwrite($archivo,$_GET['nombre']."\n");
   }
?>
<form method="GET" action="?">
    <input type="text" name="nombre">
    <input type="submit">
</form>
```

Para leer el contenido almacenado en un archivo en PHP, se abre el archivo en modo lectura con `fopen()` usando el modo `'r'`. Luego, se puede recorrer el archivo línea a línea utilizando un bucle con la función `fgets()`, que lee hasta el salto de línea.

Durante la lectura, cada línea puede procesarse o mostrarse, por ejemplo, imprimiéndola en pantalla para visualizar el contenido completo. Finalmente, se debe cerrar el archivo con `fclose()` para liberar los recursos asociados.

Este método permite recuperar y utilizar datos almacenados previamente, facilitando la persistencia y gestión de información entre distintas ejecuciones del programa o solicitudes web.

```
002-leer.php
<?php
    if(isset($_GET['nombre'])){
        echo "Tu nombre es: ".$_GET['nombre']."<br>";
        $archivo = fopen("documento.txt",'a');
        fwrite($archivo,$_GET['nombre']."\n");
        fclose($archivo);
   }
?>
<form method="GET" action="?">
    <input type="text" name="nombre">
    <input type="submit">
</form>
<h3>Listado de los nombres guardados:</h3>
<?php
    $archivo = fopen("documento.txt",'r');
    while(!feof($archivo)) {
      echo fgets($archivo) . "<br>";
    }
    fclose($archivo);
?>
```

4. Blog en PHP

El proyecto del blog en PHP se basa en la definición de clases que representan las distintas entidades del sistema, como las entradas o publicaciones. Estas clases encapsulan la información y los comportamientos necesarios para gestionar el contenido del blog.

Se estructura el código en archivos organizados, donde se incluyen las definiciones de clases y scripts que manejan la lógica de presentación, acceso a datos y generación dinámica de contenido. Esta organización facilita la escalabilidad y el mantenimiento del proyecto.

El uso de objetos permite modelar de forma clara y coherente cada elemento del blog, facilitando la extensión y personalización futura de sus funcionalidades.

4.1. Proyecto-Definición de clases

El primer paso en la creación del proyecto es definir las clases que representarán las entidades principales, como las publicaciones o entradas del blog. Estas clases incluyen propiedades para almacenar atributos como título, contenido, autor y fecha, así como métodos para manipular y mostrar esta información.

Esta definición se realiza en archivos separados para mantener el código organizado y facilitar su reutilización. Las clases forman la base estructural sobre la cual se construirá el resto de la aplicación, permitiendo manejar los datos de forma ordenada y coherente.

Estructura del proyecto:

```
001-Proyecto-Definición de clases
├── clases
│   └── Publicacion.php
└── index.php
```

En este primer subcapítulo del proyecto del blog, titulado **Proyecto - Definición de clases**, se establece la arquitectura inicial sobre la que se va a construir todo el sistema de publicación. Es el punto de partida del proyecto, y se centra en definir la estructura básica de datos que representará una entrada de blog.

4.1.1. Estructura del proyecto

En esta fase, el proyecto consta de dos archivos fundamentales distribuidos en una estructura mínima de carpetas:

```
001-Proyecto-Definición de clases
├── clases
│   └── Publicacion.php
└── index.php
```

4.1.2. Qué se está haciendo

Se definen las bases de la programación orientada a objetos que utilizará el blog. En particular, se define una **clase abstracta** llamada `Publicacion` y dos clases hijas: `Pagina` y `Entrada`. La clase `Publicacion` actúa como modelo base a partir del cual se podrán crear diferentes tipos de contenido.

Además, en el archivo `index.php` se crea un conjunto de instancias de la clase `Entrada`, lo cual representa entradas concretas del blog. Estas instancias se agrupan dentro de un array, utilizando `array_push`, y se hace un volcado con `var_dump` para confirmar que el sistema puede crear y almacenar correctamente los objetos.

4.1.3. Archivo clases/Publicacion.php

Este archivo contiene la definición de las clases que serán la piedra angular del sistema:

- `Publicacion` es una clase abstracta. Esta clase define propiedades comunes como el título, la fecha, el autor, el contenido y la imagen. Estas propiedades se definen como privadas y se inicializan en el constructor.
- Se proporciona un método `setTitulo` con validación: sólo se permite establecer un título si su longitud es mayor de 10 caracteres. Si no se cumple esta condición, se asigna la cadena "error".
- Se proporciona un método `getTitulo` para recuperar el valor del título.
- Se definen dos clases vacías que heredan de `Publicacion`: `Pagina` y `Entrada`. Aunque aún no tienen comportamiento propio, su existencia anticipa que más adelante podrán diferenciarse en funcionalidad o representación.

Este enfoque refuerza el uso de la herencia como mecanismo para compartir código común entre diferentes tipos de publicaciones.

4.1.4. Archivo index.php

Este archivo incluye la clase desde `clases/Publicacion.php` y crea un array llamado `$entradas`. A continuación, se insertan tres objetos de la clase `Entrada` con diferentes títulos y contenidos, lo que simula la creación de tres entradas de blog. Finalmente, se muestra el contenido del array con `var_dump`.

Este primer paso cumple una función doble:

1. **Verificar la correcta construcción de objetos**, su almacenamiento y manipulación.
2. **Sentar las bases del modelo de datos**, alrededor del cual girará todo el resto del proyecto del blog.

4.1.5. Conclusión

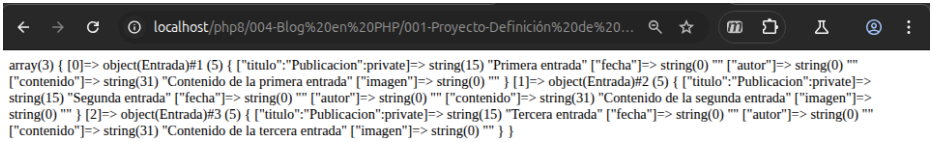
Esta etapa inicial no produce salida HTML ni presenta todavía una interfaz visual. Su objetivo es establecer el modelo lógico del proyecto: definir cómo se estructura y representa la información en el sistema. En los próximos pasos se avanzará hacia la presentación visual y la funcionalidad interactiva, pero todo ello descansará sobre las clases definidas en esta etapa. Esta decisión de comenzar por el modelo de datos es habitual en proyectos bien estructurados, pues garantiza escalabilidad y mantenibilidad.

```
index.php
<?php

    include "clases/Publicacion.php";

    $entradas = [];

    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
var_dump($entradas);

?>
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

Resultado:



array(3) { [0]=> object(Entrada)#1 (5) { ["titulo":"Publicacion":private]=> string(15) "Primera entrada" ["fecha"]=> string(0) "" ["autor"]=> string(0) "" ["contenido"]=> string(31) "Contenido de la primera entrada" ["imagen"]=> string(0) "" } [1]=> object(Entrada)#2 (5) { ["titulo":"Publicacion":private]=> string(15) "Segunda entrada" ["fecha"]=> string(0) "" ["autor"]=> string(0) "" ["contenido"]=> string(31) "Contenido de la segunda entrada" ["imagen"]=> string(0) "" } [2]=> object(Entrada)#3 (5) { ["titulo":"Publicacion":private]=> string(15) "Tercera entrada" ["fecha"]=> string(0) "" ["autor"]=> string(0) "" ["contenido"]=> string(31) "Contenido de la tercera entrada" ["imagen"]=> string(0) "" } }

4.2. Proyecto-Método devolver entrada

Para mostrar las entradas del blog, se crea un array que almacenará objetos de la clase `Entrada`. A continuación, se instancian varias entradas con sus respectivos títulos y contenidos, y se agregan al array usando la función `array_push()`.

Una vez que el array contiene todas las entradas deseadas, se recorre con un bucle para llamar a un método que devuelve la representación HTML de cada publicación, mostrando así el contenido en la página web.

Esta estructura permite gestionar dinámicamente las entradas y facilita su presentación ordenada y coherente en el sitio.

Estructura del proyecto:

```
002-Proyecto-Metodo devolver entrada
├── clases
│   └── Publicacion.php
└── index.php
```

En este segundo subcapítulo, titulado **Proyecto - Método devolver entrada**, se amplía la lógica interna de las clases del proyecto incorporando la capacidad de cada entrada de blog de generar su propia representación en HTML. Esta funcionalidad permitirá mostrar las entradas en la interfaz web más adelante.

4.2.1. Estructura del proyecto

La estructura de archivos se mantiene idéntica a la del paso anterior:

```
002-Proyecto-Metodo devolver entrada
├── clases
│   └── Publicacion.php
└── index.php
```

4.2.2. Qué se está haciendo

El objetivo principal en este punto es que los objetos creados a partir de la clase `Entrada` (que hereda de `Publicacion`) sean capaces de generar una salida HTML bien estructurada que represente visualmente una publicación. Para ello se introduce un nuevo método dentro de la clase `Publicacion`, llamado `damePublicacion`.

4.2.3. Cambios en clases/Publicacion.php

Se añade el método `damePublicacion`, que encapsula una plantilla HTML básica. Este método devuelve una cadena HTML que incluye:

- La etiqueta `<article>` como contenedor principal de la entrada.
- Una cabecera `<h3>` con el título de la publicación.
- Una etiqueta `<time>` con la fecha (aunque por el momento el valor es estático).
- Un párrafo `<p>` con el contenido de la entrada.

El método utiliza la concatenación de cadenas para insertar el valor de `$this->titulo` y `$this->contenido` dentro de la plantilla. Se trata de una aproximación simple pero efectiva para generar contenido HTML desde clases PHP.

Además, este cambio fortalece la separación de responsabilidades, ya que delega en cada objeto la responsabilidad de saber cómo debe representarse visualmente.

4.2.4. Cambios en index.php

El archivo principal del proyecto se modifica para aprovechar el nuevo método `damePublicacion`. Ahora, tras crear las entradas mediante `new Entrada(...)`, se recorre el array `$entradas` con un bucle `for`, y en cada iteración se llama a `$entradas[$i]->damePublicacion()`. El resultado de cada llamada se imprime en pantalla usando `echo`.

Esto significa que, en vez de simplemente inspeccionar las entradas con `var_dump`, como en el paso anterior, ahora el proyecto ya produce una **salida HTML funcional**, visible desde el navegador, representando cada entrada como un artículo completo.

4.2.5. Justificación de los cambios

Los cambios introducidos tienen un impacto significativo:

- **Encapsulación visual**: cada entrada sabe cómo debe presentarse, lo que permite reusar y mantener la lógica de presentación en un solo lugar.

- **Preparación para el frontend**: al generar HTML desde las clases, se sientan las bases para una interfaz web dinámica y bien estructurada.
- **Modularidad**: separar el modelo (`Publicacion`) de la vista (plantilla HTML dentro del método) hace posible futuras ampliaciones como tematización, plantillas alternativas o generación de otros formatos.

4.2.6. Conclusión

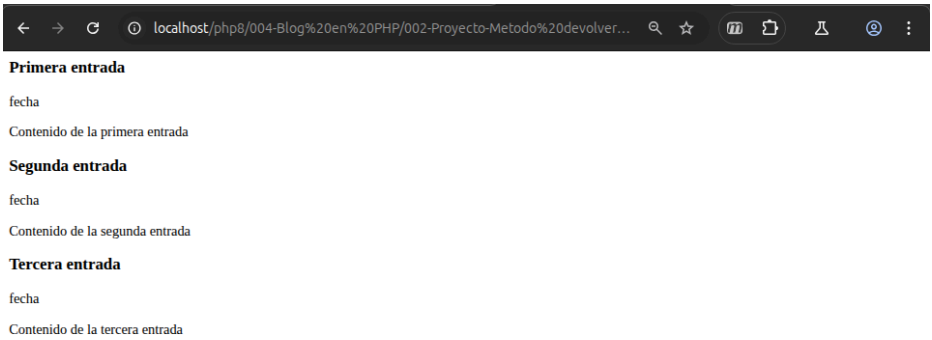
Esta etapa representa un paso esencial en la evolución del proyecto, transformando simples objetos en componentes activos del sistema capaces de construir su propia representación. Con este avance, el blog empieza a producir HTML real, acercándose a una aplicación web funcional. En los siguientes pasos se introducirá la separación de responsabilidades mediante la inclusión de archivos, y se continuará organizando el contenido para lograr una arquitectura escalable y profesional.

```
index.php
<?php

    include "clases/Publicacion.php";

    $entradas = [];

    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

?>
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <h3>'.$this->titulo.'</h3>
                    <time>fecha</time>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

Resultado:



4.3. Proyecto-Include entradas

Para organizar el código, se puede separar la definición de las entradas en un archivo aparte que contenga únicamente la creación de los objetos y su inserción en el array. Este archivo se incluye luego en el script principal con la instrucción `include`.

Esta técnica facilita la gestión y actualización del contenido sin alterar la lógica principal del programa. Además, permite reutilizar o modificar las entradas de forma independiente, mejorando la modularidad del proyecto y manteniendo un código más limpio y mantenible.

Estructura del proyecto:

```
003-Proyecto-Include entradas
├── clases
│   └── Publicacion.php
├── datos
│   └── arrayobjetos.php
└── index.php
```

En este tercer subcapítulo del proyecto, titulado **Proyecto - Include entradas**, se introduce un principio clave del desarrollo de aplicaciones web modulares: la **separación del contenido del flujo de control principal**, mediante la inclusión de archivos. Es un paso estratégico que permite escalar el proyecto de forma ordenada y facilitar la reutilización del código.

4.3.1. Estructura del proyecto

Aquí se introduce una nueva carpeta `datos` donde se almacena el contenido de las entradas:

```
003-Proyecto-Include entradas
├── clases
│   └── Publicacion.php
├── datos
│   └── arrayobjetos.php
└── index.php
```

4.3.2. Qué se está haciendo

El principal cambio en esta etapa es que se externaliza la creación de las entradas de blog. En lugar de construir el array `$entradas` directamente en `index.php`, se crea un nuevo archivo —`datos/arrayobjetos.php`— que contiene exclusivamente las llamadas a `array_push` para insertar objetos `Entrada` en el array `$entradas`.

Esto permite:

- **Desacoplar los datos de las entradas del flujo principal**.
- **Mejorar la organización**: ahora los datos pueden mantenerse, editarse o incluso ser reemplazados por otras fuentes (como un archivo JSON o una base de datos) sin tocar el archivo principal.
- **Facilitar la escalabilidad**: a medida que el contenido crezca, se podrá segmentar, versionar o generar dinámicamente.

4.3.3. Archivo datos/arrayobjetos.php

Este archivo define las entradas estáticas mediante `array_push`, utilizando la misma estructura que en los pasos anteriores. Importante: este archivo **no declara el array** `$entradas`; se asume que ese array ya ha sido definido en el archivo que lo incluye. Este detalle obliga a que, en `index.php`, el array `$entradas` se declare antes de hacer el `include`.

El archivo contiene instrucciones como estas:

```
array_push(
    $entradas,
    new Entrada(
        "Primera entrada",
        "Contenido de la primera entrada"
    )
);
```

Se repiten varias veces para simular un conjunto de entradas disponibles en el blog.

4.3.4. Cambios en index.php

El archivo principal sigue cargando la clase desde `clases/Publicacion.php`. Luego, se declara el array `$entradas`, y **se incluye** el archivo `datos/arrayobjetos.php` que lo rellena.

Finalmente, se recorre el array con un bucle y se llama a `damePublicacion` sobre cada objeto para mostrar las entradas en

HTML, exactamente igual que en el paso anterior. Es decir, el resultado visual es el mismo, pero la estructura interna del código es mucho más mantenible.

```
$entradas = [];
include "datos/arrayobjetos.php";

for($i = 0; $i < count($entradas); $i++) {
    echo $entradas[$i]->damePublicacion();
}
```

4.3.5. Cambios en clases/Publicacion.php

Este archivo no sufre modificaciones en esta etapa. Se sigue utilizando la clase `Publicacion` como clase abstracta, y `Entrada` como subclase con la funcionalidad de representación HTML a través de `damePublicacion`.

4.3.6. Justificación de los cambios

- **Mantenimiento**: editar el contenido del blog ahora solo requiere modificar un archivo externo.
- **Separación de responsabilidades**: `index.php` se encarga del flujo de control y del renderizado, mientras que `arrayobjetos.php` se ocupa únicamente del contenido.
- **Escalabilidad**: permite que el contenido sea sustituido fácilmente por un archivo JSON, una API externa o una base de datos.

4.3.7. Conclusión

Este paso no cambia la funcionalidad del proyecto desde el punto de vista del usuario, pero sí supone un salto cualitativo en cuanto a arquitectura del código. Introducir `include` para separar la lógica del contenido es una práctica fundamental en PHP y en el desarrollo web en general. Este patrón será recurrente a lo largo del proyecto y se irá perfeccionando con

la separación también de cabeceras, pie de página, hojas de estilo, datos JSON, formularios y paneles de control.

```
index.php
<?php

    include "clases/Publicacion.php";


    $entradas = [];
    include "datos/arrayobjetos.php";

   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

?>
```

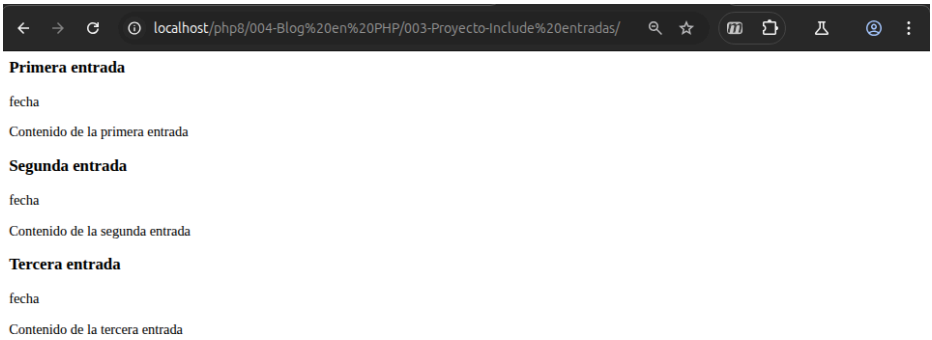
.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <h3>'.$this->titulo.'</h3>
                    <time>fecha</time>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

Resultado:



4.4. Proyecto-Cabecera y pie de página

Para mejorar la estructura del proyecto y facilitar su mantenimiento, es recomendable separar las distintas partes de la página en archivos independientes, como la cabecera, el pie de página y los datos. Estos archivos se incluyen en el script principal mediante `include` o `require`.

Esta organización permite reutilizar componentes comunes en varias páginas, evitando duplicación de código y facilitando actualizaciones centralizadas. Además, ayuda a mantener un código más claro y modular, favoreciendo la colaboración y escalabilidad del proyecto.

Estructura del proyecto:

```
004-Proyecto-Cabecera y pie de pagina
├── clases
│   └── Publicacion.php
├── datos
│   └── arrayobjetos.php
└── index.php
```

En este cuarto subcapítulo, titulado **Proyecto - Cabecera y pie de página**, se avanza en la organización estructural y visual del proyecto mediante la introducción de las secciones *cabecera* y *pie de página* en el HTML generado. Esto no solo mejora la estética y funcionalidad de la interfaz web, sino que también introduce un principio clave de desarrollo modular: la reutilización de plantillas comunes para distintas páginas.

4.4.1. Estructura del proyecto

La estructura de carpetas y archivos en esta fase es la siguiente:

```
004-Proyecto-Cabecera y pie de pagina
├── clases
│   └── Publicacion.php
├── datos
│   └── arrayobjetos.php
└── index.php
```

Aunque en este paso todavía no se usan `include` separados para cabecera y pie, el HTML completo ya se ha integrado en el archivo principal `index.php`.

4.4.2. Qué se está haciendo

El archivo `index.php` se modifica para:

- Añadir una estructura completa de documento HTML (`<!doctype html>`, `<html>`, `<head>`, `<body>`).

- Incluir la cabecera visual del blog con una etiqueta `<header>`, en la que aparece un título con `<h1>El blog de Jose Vicente</h1>`.
- Introducir un `<main>` donde se renderizan las entradas mediante el bucle PHP.
- Agregar un `<footer>` con los créditos o derechos de autor.

Este diseño prepara el terreno para poder aplicar estilos CSS más adelante, y además permite reutilizar estas secciones comunes en otras páginas del sitio (como la página de creación de entradas, el panel de administración, o las páginas estáticas).

4.4.3. Cambios en index.php

Ahora el contenido del archivo está completamente incrustado dentro de una plantilla HTML. El bloque PHP que muestra las entradas se encuentra dentro de la sección `<main>`, respetando la separación entre estructura HTML y lógica de presentación.

Además de lo ya conocido (inclusión de clases y datos, bucle que recorre `$entradas`), se agregan las siguientes secciones:

```
<!doctype html>
<html>
<head>
    <title>El blog de Jose Vicente</title>
    <link rel="stylesheet" href="css/estilo.css">
</head>
<body>
    <header>
        <h1>El blog de Jose Vicente</h1>
    </header>
    <main>
        <!-- Aquí se renderizan las entradas -->
    </main>
    <footer>
        <h6>(c) 2023 Jose Vicente Carratalá</h6>
    </footer>
</body>
</html>
```

Dentro de `<main>`, el bucle PHP sigue iterando sobre las entradas y llamando a `damePublicacion()` como antes.

4.4.4. Archivos clases/Publicacion.php y datos/arrayobjetos.php

Ambos se mantienen sin cambios respecto al paso anterior. Siguen cumpliendo sus funciones de definición de clases y provisión de datos respectivamente.

4.4.5. Justificación de los cambios

- **Mejora visual**: la página comienza a tener un formato de blog real, con cabecera, cuerpo y pie de página.
- **Preparación para el estilo CSS**: se introduce un enlace a una hoja de estilo (`css/estilo.css`), aunque aún no se ha creado.
- **Modularidad conceptual**: aunque la cabecera y el pie todavía están escritos directamente en `index.php`, la estructura ya

permite dividirlos en archivos externos mediante `include` en el siguiente paso.
- **Reutilización futura**: se sientan las bases para que estos elementos comunes se puedan usar en cualquier página del proyecto.

4.4.6. Conclusión

Este paso marca el inicio de la transformación del proyecto desde un simple generador de HTML hacia una aplicación web completa, con diseño y estructura visual coherente. Aunque no se han tocado las clases ni los datos, se ha dado un paso crucial en la organización de la interfaz y en la preparación para la futura inclusión de estilos y componentes reutilizables. Esta organización en bloques semánticos facilitará enormemente las siguientes fases de desarrollo.

```
index.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
        </header>
        <main>

        <?php

            include "clases/Publicacion.php";


            $entradas = [];
            include "datos/arrayobjetos.php";

           for($i = 0;$i<count($entradas);$i++){
                echo $entradas[$i]->damePublicacion();
            }

        ?>

        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

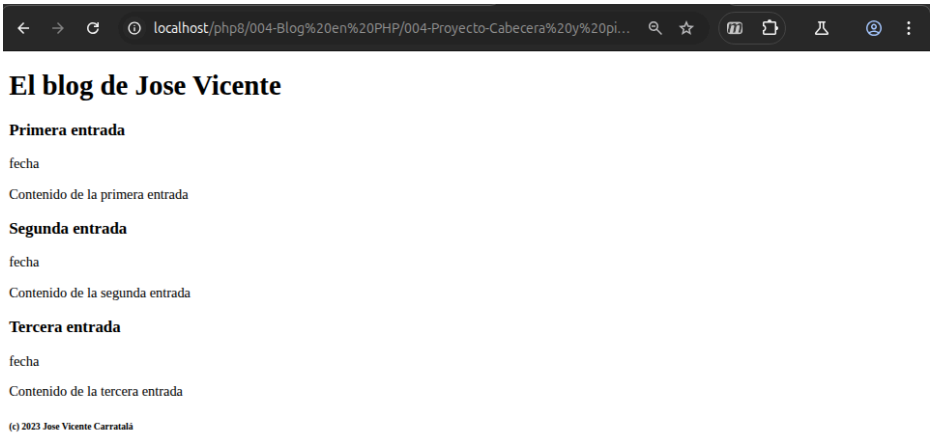
.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <h3>'.$this->titulo.'</h3>
                    <time>fecha</time>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

Resultado:



4.5. Proyecto-Include

Para optimizar la organización del proyecto, se pueden crear archivos específicos para fragmentos reutilizables como la cabecera y el pie de página, almacenándolos en una carpeta dedicada, por ejemplo, `inc/`. Luego, estos archivos se incluyen en el archivo principal con la instrucción `include`.

Esta práctica facilita la gestión del diseño común en todas las páginas, permitiendo modificar la apariencia o estructura desde un único lugar. Además, mejora la modularidad del código y su mantenimiento a medida que el proyecto crece.

Estructura del proyecto:

```
005-Proyecto-Include
├── clases
│   └── Publicacion.php
├── datos
│   └── arrayobjetos.php
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

En este quinto subcapítulo, titulado **Proyecto - Include**, se produce una reorganización fundamental de la estructura del proyecto mediante la **separación física de la cabecera y el pie de página en archivos independientes**. Este paso afianza el uso del principio de modularidad y reutilización, característico de las buenas prácticas en desarrollo web con PHP.

4.5.1. Estructura del proyecto

A partir de este punto, la estructura del proyecto incorpora una nueva carpeta `inc`, que contiene los fragmentos HTML comunes del sitio:


```
005-Proyecto-Include
├── clases
│   └── Publicacion.php
├── datos
│   └── arrayobjetos.php
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```


4.5.2. Qué se está haciendo

Hasta ahora, tanto la cabecera (`<head>`, `<header>`) como el pie de página (`<footer>`) estaban codificados directamente dentro de `index.php`. En esta fase se externalizan esos bloques en dos archivos nuevos:

- `inc/cabecera.php`: contiene toda la estructura de apertura HTML y el encabezado de la página.
- `inc/piedepagina.php`: cierra el bloque `<main>` y contiene el pie de página y el cierre del documento.

4.5.3. Cambios en index.php

El archivo principal queda mucho más limpio y enfocado únicamente en el flujo lógico de la página. Se sustituyen los bloques largos de HTML por dos sentencias `include`:

```
include "inc/cabecera.php";
include "clases/Publicacion.php";

$entradas = [];
include "datos/arrayobjetos.php";

for($i = 0; $i < count($entradas); $i++) {
    echo $entradas[$i]->damePublicacion();
}

include "inc/piedepagina.php";
```

Este enfoque deja claro que `index.php` actúa como **controlador de la vista**, mientras que el contenido de los includes representa la **presentación**. Además, reduce el ruido visual y mejora la mantenibilidad.

4.5.4. Contenido de inc/cabecera.php

Contiene el comienzo del documento HTML:

```
<!doctype html>
<html>
<head>
    <title>El blog de Jose Vicente</title>
    <link rel="stylesheet" href="css/estilo.css">
</head>
<body>
    <header>
        <h1>El blog de Jose Vicente</h1>
    </header>
    <main>
```

4.5.5. Contenido de inc/piedepagina.php

Contiene el cierre de la sección principal y el documento:

```
   </main>
    <footer>
        <h6>(c) 2023 Jose Vicente Carratalá</h6>
    </footer>
</body>
</html>
```

4.5.6. Archivos clases/Publicacion.php y datos/arrayobjetos.php

Permanecen sin modificaciones. La clase `Entrada` sigue siendo responsable de construir la estructura HTML de cada entrada, y `arrayobjetos.php` sigue creando el contenido de prueba.

4.5.7. Justificación de los cambios

- **Reutilización**: las secciones comunes de la web ahora pueden incluirse en múltiples páginas (por ejemplo, en el panel de administración o en futuras vistas de entradas).

- **Organización**: se reduce la longitud de `index.php` y se centraliza el diseño en un solo lugar.

- **Facilidad de mantenimiento**: cualquier cambio en el título, los enlaces a hojas de estilo o los créditos del pie se puede hacer en un solo archivo.

- **Escalabilidad**: esta modularización permite fácilmente añadir cabeceras alternativas, plantillas condicionales o internacionalización.

4.5.8. Conclusión

Esta etapa transforma el archivo principal en un orquestador que incluye piezas separadas según su responsabilidad. Este patrón se consolidará aún más con la introducción de datos en formatos externos (como JSON), paneles de control y estructuras más complejas. La técnica de `include` es esencial para cualquier proyecto en PHP, y su uso temprano asegura que el desarrollo posterior se mantenga claro, reutilizable y profesional.

```
index.php
<?php
    include "inc/cabecera.php";
    include "clases/Publicacion.php";

    $entradas = [];
    include "datos/arrayobjetos.php";

   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }
    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <h3>'.$this->titulo.'</h3>
                    <time>fecha</time>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

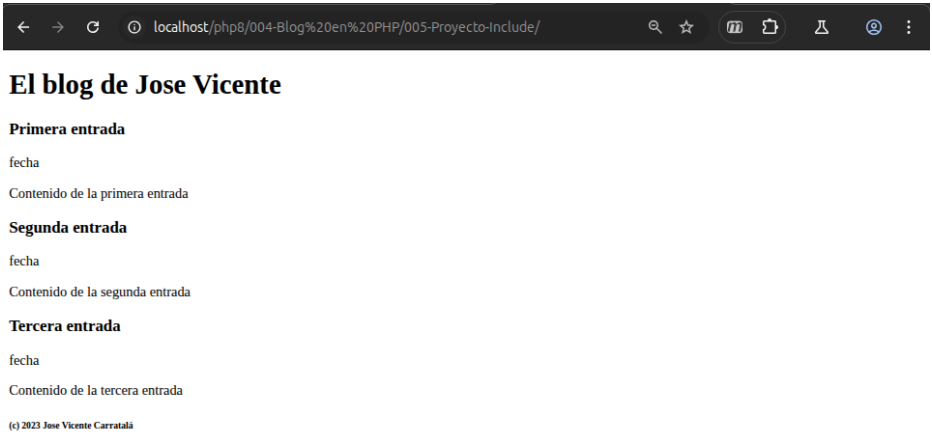
.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:



4.6. Proyecto-Estilo

Para dar estilo a la aplicación web, se utiliza una hoja de estilos CSS que define la apariencia visual de los elementos HTML. El archivo CSS se incluye en la cabecera del documento mediante la etiqueta `<link>`.

El estilo controla aspectos como colores de fondo, tipografías, márgenes, bordes y disposición de los elementos. Por ejemplo, se puede establecer un ancho fijo para el contenido principal, centrarlo en la página y aplicar sombras para darle profundidad.

Aplicar un diseño coherente y atractivo mejora la experiencia del usuario y hace que la interfaz sea más profesional y fácil de usar. Además, separar el estilo del contenido permite modificar la apariencia sin afectar la lógica del programa.

Estructura del proyecto:

```
006-Proyecto-Estilo
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   └── arrayobjetos.php
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

En este sexto subcapítulo, titulado **Proyecto - Estilo**, se incorpora por primera vez una hoja de estilos CSS al proyecto. Este paso no afecta a la lógica de la aplicación, pero transforma por completo su apariencia, haciendo que el blog adopte un diseño visual profesional, estructurado y agradable para el usuario. Se introduce así la noción de separación de contenido (HTML), lógica (PHP) y presentación (CSS), uno de los principios más sólidos del desarrollo web moderno.

4.6.1. Estructura del proyecto

Ahora se añade una nueva carpeta llamada `css` que contiene el archivo de estilos:

```
006-Proyecto-Estilo
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   └── arrayobjetos.php
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

4.6.2. Qué se está haciendo

La funcionalidad del proyecto no cambia en términos de código PHP ni en la generación de contenido dinámico. Lo que sí cambia es la **forma en que ese contenido se presenta** al usuario. Se define un conjunto de reglas CSS que afectan a todo el documento HTML para aplicar:

- Colores de fondo
- Tipografías
- Ancho del contenedor
- Espaciados
- Sombra en el área principal
- Bordes redondeados en cabecera y pie de página
- Z-index para mantener jerarquía visual

4.6.3. Archivo css/estilo.css

Este archivo contiene las reglas CSS que estilizan el sitio. El diseño general aplica una tipografía `sans-serif`, centra el contenido y define una paleta de colores neutros (fondo gris claro con zonas blancas para los contenedores).

Las reglas principales son:

```
body, html {
    background: rgb(200, 200, 200);
    font-family: sans-serif;
}
header, main, footer {
    background: white;
    width: 800px;
    margin: auto;
    padding: 20px;
}
main {
    box-shadow: 0px 0px 20px rgba(0,0,0,0.5);
}
header, footer {
    position: relative;
    z-index: 1000;
}
header {
    border-radius: 10px 10px 0px 0px;
}
footer {
    border-radius: 0px 0px 10px 10px;
}
```

Este estilo mejora significativamente la presentación visual sin necesidad de modificar el HTML.

4.6.4. Archivo inc/cabecera.php

El único cambio respecto al paso anterior es la inclusión explícita de la hoja de estilos:

```
<link rel="stylesheet" href="css/estilo.css">
```

Con esto, todas las reglas de `estilo.css` se aplican automáticamente al contenido generado por PHP.

4.6.5. Archivos index.php, clases/Publicacion.php, datos/arrayobjetos.php y inc/piedepagina.php

Estos archivos **no se modifican** en esta etapa. Siguen funcionando como en el paso anterior, lo que demuestra que la aplicación de estilos puede realizarse de forma no invasiva, afectando únicamente a la presentación sin tocar la lógica del sistema.

4.6.6. Justificación de los cambios

- **Separación de responsabilidades**: el estilo se gestiona de forma completamente independiente de la lógica y estructura del contenido.
- **Reutilización y consistencia**: con una única hoja de estilos, se asegura un diseño coherente en toda la aplicación.
- **Preparación para el crecimiento**: se podrán añadir nuevos elementos (formularios, paneles, botones) reutilizando la misma hoja de estilos.
- **Mejora visual inmediata**: la legibilidad y estética del sitio mejora sustancialmente sin modificar el código PHP.

4.6.7. Conclusión

Este paso marca la transición de una maqueta funcional a una aplicación visualmente cuidada. Aunque conceptualmente sencillo, es fundamental para que el blog sea atractivo y usable. A partir de aquí, el desarrollo podrá continuar sin preocuparse por el estilo en cada nuevo componente, porque la base de presentación ya ha sido establecida. Además, esta separación permitirá delegar el diseño a perfiles especializados sin interferir con el núcleo del código PHP.

```
index.php
<?php
    include "inc/cabecera.php";
    include "clases/Publicacion.php";

    $entradas = [];
    include "datos/arrayobjetos.php";

   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }
    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <h3>'.$this->titulo.'</h3>
                    <time>fecha</time>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

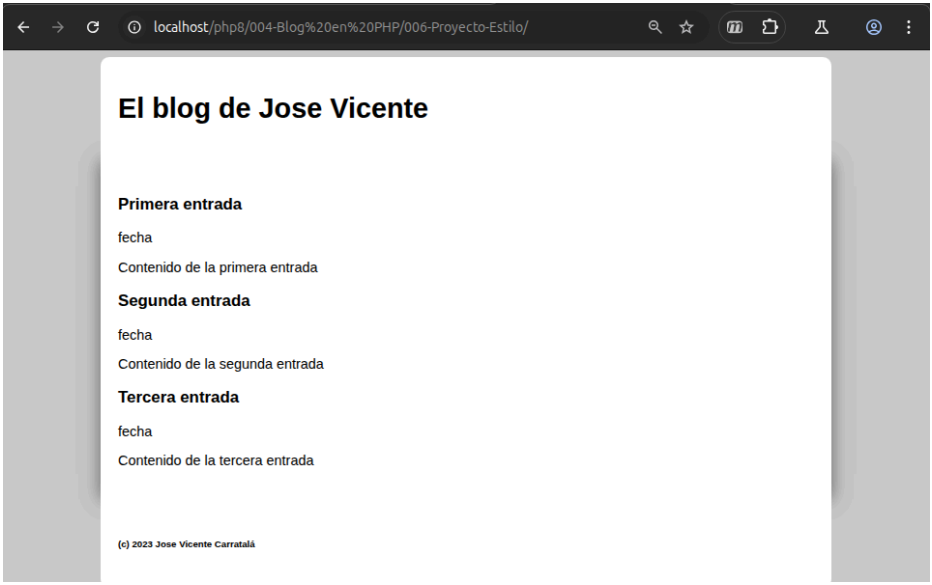
.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:



4.7. Proyecto-Entradas en JSON

Para gestionar las entradas del blog de forma más flexible, se puede almacenar la información en archivos JSON. Estos archivos contienen los datos estructurados en formato texto, que pueden ser fácilmente leídos y procesados por PHP.

Al cargar el archivo JSON, se decodifican sus contenidos para convertirlos en arrays o objetos que representan cada entrada. Esto permite actualizar o añadir contenido sin modificar el código fuente, facilitando la gestión y escalabilidad del proyecto.

Además, utilizar JSON mejora la interoperabilidad, ya que el mismo formato puede ser consumido por otras aplicaciones o servicios externos si es necesario.

Estructura del proyecto:

```
007-Proyecto-Entradas en JSON
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

En este séptimo subcapítulo, titulado **Proyecto - Entradas en JSON**, se introduce un cambio esencial en la forma en que se cargan los datos del blog: se reemplaza la generación manual del contenido con `array_push` por una fuente de datos estructurada en un archivo JSON. Este paso representa un punto de inflexión en el proyecto, al convertir el sistema en una aplicación basada en datos externos, más flexible, escalable y compatible con futuros desarrollos dinámicos (como APIs o paneles de administración).

4.7.1. Estructura del proyecto

Se mantiene la estructura modular y se añade un nuevo archivo de datos:

```
007-Proyecto-Entradas en JSON
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

El archivo `arrayobjetos.php` todavía está presente, pero dejará de usarse. El nuevo archivo `entradas.json` contiene las entradas de blog en formato JSON.

4.7.2. Qué se está haciendo

Se sustituyen las líneas que construyen las entradas en PHP por una lectura dinámica desde el archivo `datos/entradas.json`. Este archivo contiene un array de objetos con los campos `titulo`, `autor`, `fecha`, `contenido` e `imagen`.

Esto permite:

- **Desacoplar el contenido del código**.
- **Modificar fácilmente las entradas sin editar código PHP**.
- **Preparar el terreno para crear nuevas entradas desde formularios o administradores**.

4.7.3. Contenido de datos/entradas.json

El archivo contiene una colección de objetos con la información de cada entrada. Por ejemplo:

```
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    ...
]
```

Cada objeto representa una entrada del blog.

4.7.4. Cambios en index.php

El archivo PHP ahora realiza los siguientes pasos:

1. Incluye la cabecera.
2. Incluye la clase `Publicacion`.
3. Carga y decodifica el JSON con `file_get_contents` y `json_decode`.
4. Recorre los objetos JSON y los convierte en instancias de la clase `Entrada`, añadiéndolos al array `$entradas`.
5. Muestra cada entrada llamando a `damePublicacion`.
6. Incluye el pie de página.

El array `$entradas` ya no se rellena desde `arrayobjetos.php`, sino que se crea con un bucle como este:

```
$datos = json_decode(file_get_contents("datos/entradas.json"), true);

foreach($datos as $elemento){

        array_push($entradas, new Entrada($elemento["titulo"], $elemento["contenido"]));

}
```

Aunque el JSON incluye `autor`, `fecha` e `imagen`, de momento solo se utilizan `titulo` y `contenido`, ya que la clase `Entrada` aún no ha sido modificada para recibir o mostrar los demás campos.

4.7.5. Cambios en clases/Publicacion.php

Este archivo **no cambia todavía**. La clase `Entrada` sigue heredando de `Publicacion` y acepta únicamente título y contenido. Los otros campos del JSON se ignoran por ahora.

4.7.6. Justificación de los cambios

- **Escalabilidad**: el contenido del blog puede crecer sin tocar el código PHP.
- **Separación de datos**: los datos se mantienen en un formato independiente y estandarizado.
- **Compatibilidad futura**: el uso de JSON acerca el proyecto a estructuras de backend modernas basadas en APIs REST o AJAX.
- **Simplicidad editorial**: el contenido puede ser generado, editado o importado desde otros sistemas.

4.7.7. Conclusión

Con este paso, el proyecto da el salto de un sistema estático a uno dinámico basado en datos externos. Aunque los objetos se crean aún con un constructor básico, ya se ha logrado la independencia entre el contenido y el código, lo que abre posibilidades de edición, personalización y conexión con sistemas externos. Esta transformación pone los cimientos para un blog real, con contenido gestionado de forma flexible y profesional.

```
index.php
<?php
    include "inc/cabecera.php";
    include "clases/Publicacion.php";

    $entradas = [];
    include "datos/arrayobjetos.php";

   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }
    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo":"Primera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la primera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Segunda entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la segunda
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Tercera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la tercera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Cuarta entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la cuarta
entrada",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <h3>'.$this->titulo.'</h3>
                    <time>fecha</time>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

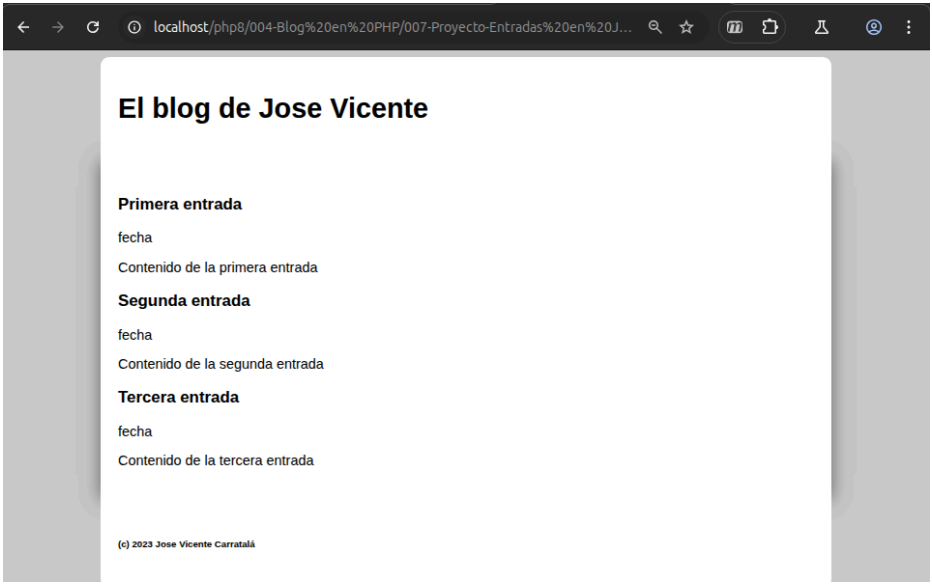
.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:



4.8. Proyecto-JSON a objetos

Para convertir las entradas almacenadas en un archivo JSON en objetos utilizables en PHP, se lee el contenido del archivo con `file_get_contents()` y luego se decodifica con `json_decode()`, obteniendo un array asociativo.

A continuación, se recorre este array y, para cada elemento, se crea una instancia de la clase `Entrada` pasando los datos correspondientes al constructor. Estas instancias se almacenan en un array que luego se utiliza para mostrar o manipular las entradas como objetos.

Este proceso permite aprovechar la estructura orientada a objetos manteniendo la flexibilidad de un formato externo editable como JSON.

Estructura del proyecto:

```
008-Proyecto-JSON a objetos
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

En este octavo subcapítulo, titulado **Proyecto - JSON a objetos**, se refina y fortalece el sistema de carga de datos desde archivos externos, consolidando el proceso iniciado en el apartado anterior. A diferencia del paso anterior, donde se cargaban los campos `titulo` y `contenido` desde un JSON ignorando el resto de propiedades, en esta fase se aprovecha **toda la información contenida en el archivo JSON** para **rellenar completamente cada objeto Entrada**, haciendo uso del constructor completo de la clase y sentando así las bases para una carga de datos estructurada y coherente.

4.8.1. Estructura del proyecto

La estructura de carpetas se mantiene:

```
008-Proyecto-JSON a objetos
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

El archivo `arrayobjetos.php` permanece, pero ya no se utiliza. El foco está ahora en `entradas.json`, desde el cual se crean los objetos `Entrada` de forma completa.

4.8.2. Qué se está haciendo

Se mejora la lógica de carga del archivo JSON en `index.php` para que cada propiedad del objeto JSON (`titulo`, `contenido`, `fecha`, `autor`, `imagen`) se pase correctamente al constructor de la clase `Entrada`, el cual ha sido adaptado previamente para aceptar esos parámetros. Ya no se descartan valores como antes, sino que se utiliza la información completa para poblar cada objeto.

4.8.3. Cambios en index.php

La carga de datos se realiza así:

```
$datos =
json_decode(file_get_contents("datos/entradas.json"), true);
foreach($datos as $elemento){
    $entrada = new Entrada(
        $elemento["titulo"],
        $elemento["contenido"]
    );
    $entrada->fecha = $elemento["fecha"];
    $entrada->autor = $elemento["autor"];
    $entrada->imagen = $elemento["imagen"];
    array_push($entradas, $entrada);
}
```

Este fragmento lee el JSON, crea un nuevo objeto `Entrada` con el título y el contenido, y a continuación **rellena las propiedades públicas o protegidas directamente** con el resto de los valores del JSON.

> Nota: Esto implica que las propiedades `fecha`, `autor` e `imagen` **deben estar accesibles** desde fuera de la clase. En esta fase del proyecto aún se están estableciendo directamente, aunque más adelante es probable que se refactoricen con `set`/`get`.

4.8.4. Cambios en clases/Publicacion.php

Aunque no se modifica explícitamente en esta fase, la funcionalidad `damePublicacion()` ahora refleja la incorporación de los datos adicionales (fecha, autor, imagen). Es decir, su HTML se actualiza para mostrar los nuevos campos:

```
function damePublicacion(){
    return '
        <article>
            <h3>'.$this->titulo.'</h3>
            <time>'.$this->fecha.'</time>
            <p>'.$this->contenido.'</p>
            <img src="imagenes/'.$this->imagen.'" alt="">
            <address>'.$this->autor.'</address>
        </article>
    ';
}
```

Este cambio hace que los objetos `Entrada` ya no sean visualmente planos, sino que expresan todos los atributos definidos en el JSON.

4.8.5. Cambios en datos/entradas.json

Este archivo ya estaba estructurado adecuadamente desde el apartado anterior, con propiedades como `titulo`, `contenido`, `autor`, `fecha` e `imagen`. No necesita modificaciones, pero ahora **todas sus propiedades se aprovechan** al crear los objetos `Entrada`.

4.8.6. Justificación de los cambios

- **Riqueza semántica**: cada entrada representa de forma completa su contenido, incluyendo autor, fecha e imagen.
- **Desacoplamiento real**: los datos se gestionan totalmente fuera del código PHP.
- **Preparación para validación**: al trabajar con objetos completos, se facilita la implementación futura de validaciones, métodos `set` y encapsulamiento.

- **Aumento del profesionalismo visual**: el blog ahora se parece más a una publicación real, con metainformación estructurada.

4.8.7. Conclusión

Este paso convierte las entradas en verdaderos objetos ricos en contenido, conectando de forma precisa los datos externos del JSON con la lógica de presentación PHP. A partir de aquí, el blog no solo es modular, sino que también es **dinámico y expresivo**, con contenido que puede gestionarse desde sistemas externos o interfaces de administración. Esta transformación permite que el código se escale sin duplicación ni dependencia entre contenido, estructura o presentación.

```
index.php
<?php
    include "inc/cabecera.php";
    include "clases/Publicacion.php";
    // Creo un array de entradas
    $entradas = [];
    // Cargo el archivo json que contiene datos
    $archivo = file_get_contents("datos/entradas.json");
    $json = json_decode($archivo,true);
    // Convierto los elementos JSON en Objetos de mi
aplicacion
    foreach($json as $entrada){
        array_push(
            $entradas,
            new Entrada(
                $entrada["titulo"],
                $entrada["contenido"]
            )
        );
    }
    // Recorro los objetos y los pinto en pantalla
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo":"Primera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la primera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Segunda entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la segunda
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Tercera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la tercera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Cuarta entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la cuarta
entrada",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function __construct($nuevotitulo,$nuevocontenido){
            $this->titulo = $nuevotitulo;
            $this->fecha = "";
            $this->autor = "";
            $this->contenido = $nuevocontenido;
            $this->imagen = "";
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <h3>'.$this->titulo.'</h3>
                    <time>fecha</time>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{

    }

?>
```

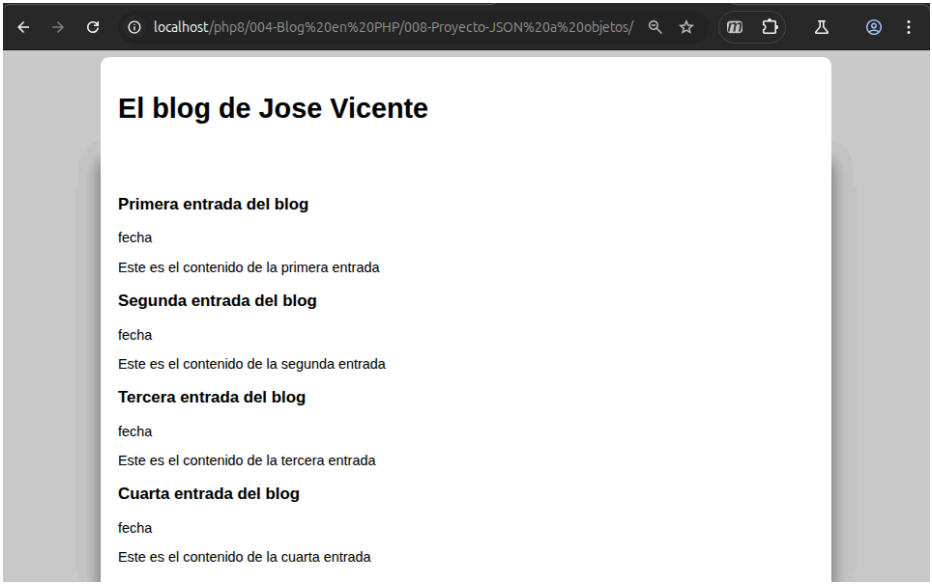
.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:



4.9. Proyecto-Imágenes y fecha

Para enriquecer las entradas con imágenes y fechas, se amplía la clase `Entrada` y su constructor para incluir propiedades que almacenen esta información adicional.

Al leer los datos desde el archivo JSON, se pasan también los valores correspondientes a la fecha y el nombre de la imagen al crear cada objeto. Esto permite mostrar no solo el texto, sino también imágenes asociadas y la fecha de publicación, ofreciendo una presentación más completa y atractiva.

Esta mejora aporta mayor dinamismo y profesionalidad al blog, facilitando la gestión visual y temporal de las publicaciones.

Estructura del proyecto:

```
009-Proyecto-Imagenes y fecha
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este noveno subcapítulo, titulado **Proyecto - Imágenes y fecha**, se completa la representación visual de cada entrada del blog incorporando dos elementos clave: **una imagen ilustrativa** y **la fecha de publicación**. Esto mejora tanto la estética como la funcionalidad del blog, acercándolo a un formato profesional y reconocible por cualquier lector habitual de publicaciones digitales.

4.9.1. Estructura del proyecto

La estructura general se mantiene, pero en esta fase se asume que se han añadido recursos visuales (por ejemplo, imágenes .jpg) en una carpeta como `imagenes/`, la cual no figura explícitamente pero se infiere por el uso dentro del código HTML.

```
009-Proyecto-Imagenes y fecha
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

4.9.2. Qué se está haciendo

Se actualiza la clase `Publicacion` para que cada entrada sea capaz de mostrar:

1. **Una imagen asociada** al contenido.
2. **La fecha de publicación**, extraída del JSON.

Estos datos ya estaban incluidos en el archivo `entradas.json` desde apartados anteriores, pero ahora se usan efectivamente dentro del método `damePublicacion()` para ser visibles en el navegador.

4.9.3. Cambios en clases/Publicacion.php

Se modifica el método `damePublicacion()` para que devuelva una estructura HTML más rica y semántica. Se añaden los siguientes elementos:

- Una etiqueta `<img>` que carga la imagen correspondiente desde la carpeta `imagenes/`.

- Una etiqueta `<time>` que muestra la fecha de publicación real.
- Se mantiene el `<h3>` para el título y el `<p>` para el contenido.
- Se añade un `<address>` para el nombre del autor.

Ejemplo de la nueva salida HTML por entrada:

```
function damePublicacion(){
    return '
        <article>
            <img src="imagenes/'.$this->imagen.'" alt="">
            <h3>'.$this->titulo.'</h3>
            <time>'.$this->fecha.'</time>
            <p>'.$this->contenido.'</p>
            <address>'.$this->autor.'</address>
        </article>
    ';
}
```

Este código construye un bloque visual coherente con los estándares de un blog.

4.9.4. Archivo datos/entradas.json

No requiere modificaciones, ya que ya contiene los campos `fecha`, `imagen` y `autor`. Este paso consiste en **aprovechar** esos campos, no en añadir nuevos.

Ejemplo de entrada en el JSON:

```
{
    "titulo": "Primera entrada del blog",
    "autor": "Jose Vicente Carratala",
    "fecha": "2023-11-30",
    "contenido": "Este es el contenido de la primera
entrada",
    "imagen": "josevicente.jpg"
}
```

4.9.5. Archivo index.php

No cambia respecto al paso anterior. Continúa cargando el JSON, instanciando objetos `Entrada`, y asignando manualmente sus propiedades secundarias (`fecha`, `autor`, `imagen`), como ya se hacía en el apartado 4.8.

4.9.6. Justificación de los cambios

- **Presentación visual completa**: se introducen los elementos visuales fundamentales de cualquier blog.
- **Aprovechamiento pleno de los datos**: ahora todos los campos del JSON tienen reflejo en pantalla.
- **Estandarización semántica**: se utilizan etiquetas HTML correctas (`<time>`, `<address>`, `<img>`) que mejoran la accesibilidad y posicionamiento SEO.
- **Preparación para personalización de estilo**: al tener todos los elementos definidos, es posible estilizar específicamente la imagen, la fecha o el autor.

4.9.7. Conclusión

Este paso finaliza el proceso de transformación de las entradas en objetos visuales ricos. Con la imagen, la fecha y el autor, cada publicación se presenta como una unidad estética y semántica completa. A nivel técnico, se consolida la sinergia entre contenido estructurado (JSON), lógica de aplicación (PHP orientado a objetos) y presentación HTML. Desde este punto, el proyecto ya está listo para avanzar hacia la interacción, edición o filtrado de contenidos.

```
index.php
<?php
    include "inc/cabecera.php";
    include "clases/Publicacion.php";
    // Creo un array de entradas
    $entradas = [];
    // Cargo el archivo json que contiene datos
    $archivo = file_get_contents("datos/entradas.json");
    $json = json_decode($archivo,true);
    // Convierto los elementos JSON en Objetos de mi
aplicacion
    foreach($json as $entrada){
        array_push(
            $entradas,
            new Entrada(
                $entrada["titulo"],
                $entrada["contenido"],
                $entrada["fecha"],
                $entrada["autor"],
                $entrada["imagen"]
            )
        );
    }
    // Recorro los objetos y los pinto en pantalla
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo":"Primera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la primera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Segunda entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la segunda
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Tercera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la tercera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Cuarta entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la cuarta
entrada",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{
```

```
    }

?>
```

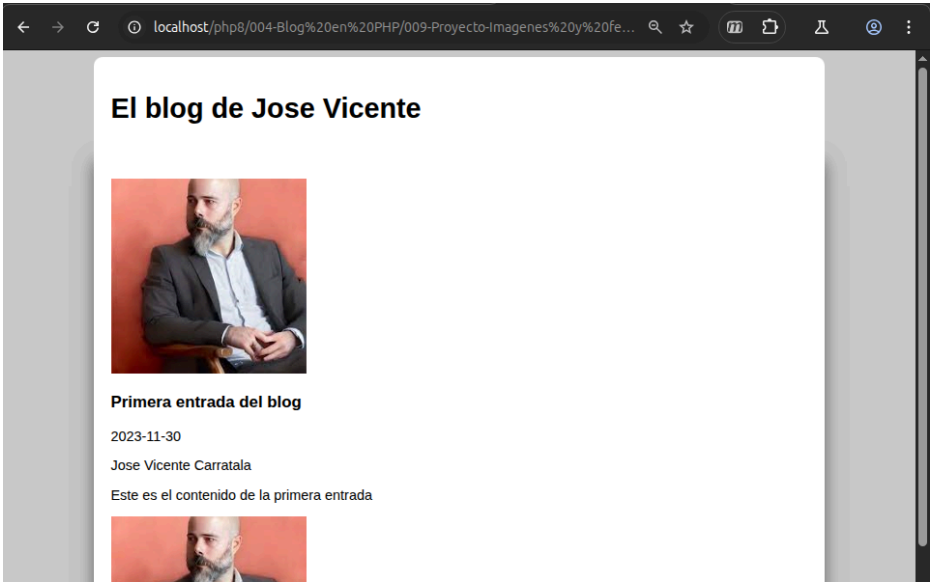
.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:



4.10. Proyecto-Estilo

Para mejorar la presentación visual del blog, se crea un archivo CSS que define estilos específicos para los distintos elementos HTML. Esto incluye el diseño del cuerpo, los encabezados, el contenido principal, los artículos, las imágenes y los menús de navegación.

Se utilizan propiedades como fondos, fuentes, márgenes, sombras y bordes para dar un aspecto limpio, moderno y coherente a la página. Por ejemplo, se puede aplicar un ancho fijo al contenedor principal y añadir sombras para resaltar las secciones.

Separar el estilo del código PHP y HTML facilita el mantenimiento y la adaptación del diseño, permitiendo modificar la apariencia sin afectar la funcionalidad del sitio.

Estructura del proyecto:

```
010-Proyecto-Estilo
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este décimo subcapítulo, titulado **Proyecto - Estilo**, se refuerza la presentación visual del blog incorporando **estilos específicos para los nuevos elementos HTML introducidos** en la fase anterior, como las imágenes, fechas y autores. También se consolida el diseño general del sitio, alineándolo con prácticas modernas de usabilidad y estética visual en entornos web.

4.10.1. Estructura del proyecto

La estructura permanece idéntica a la anterior, con todos los archivos organizados modularmente:

```
010-Proyecto-Estilo
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

El archivo central en esta fase es `css/estilo.css`, que se actualiza para reflejar el nuevo contenido visual que aparece en cada entrada.

4.10.2. Qué se está haciendo

En este punto, el proyecto ya genera entradas con título, contenido, fecha, autor e imagen. El objetivo ahora es aplicar estilos CSS que mejoren la presentación de estos elementos, tanto en lo visual como en lo estructural. Se hace lo siguiente:

- Se ajusta el estilo de las imágenes para que se adapten al diseño y no rompan el layout.
- Se estilizan las fechas (`<time>`) para que queden más discretas.
- Se formatea el autor (`<address>`) para que tenga un estilo diferenciador.
- Se refuerza el diseño del contenedor `<article>` para que cada entrada quede visualmente separada de las demás.

4.10.3. Cambios en css/estilo.css

Se añaden nuevas reglas al final del archivo o se modifican las existentes para adaptar el diseño. Estas reglas pueden incluir:

```
article {
    margin-bottom: 40px;
    padding-bottom: 20px;
    border-bottom: 1px solid #ccc;
}

article img {
    max-width: 100%;
    border-radius: 10px;
    margin-bottom: 10px;
}

article time {
    display: block;
    font-size: 0.9em;
    color: #777;
    margin-bottom: 5px;
}

article address {
    font-style: normal;
    font-size: 0.9em;
    color: #444;
    margin-top: 10px;
}
```

Estas reglas:

- Aseguran que cada entrada esté visualmente separada (`margin-bottom`, `border-bottom`).
- Hacen que las imágenes se escalen correctamente y no se desborden del contenedor.

- Aplican una tipografía discreta a la fecha.
- Presentan el nombre del autor con claridad pero sin robar protagonismo al título o contenido.

4.10.4. Archivos clases/Publicacion.php, index.php, entradas.json

No se modifican en esta fase. El objetivo no es cambiar la lógica de datos ni la estructura, sino únicamente mejorar la estética a partir de los elementos ya presentes.

4.10.5. Justificación de los cambios

- **Profesionalización del diseño**: se abandona el estilo genérico y se aplica uno más cuidado, jerárquico y legible.
- **Adaptación visual a contenido multimedia**: las imágenes ahora se comportan adecuadamente y se integran en el diseño.
- **Accesibilidad y legibilidad**: se emplean tamaños, colores y márgenes que favorecen la lectura y navegación.
- **Mantenibilidad**: todos los cambios estéticos se centralizan en la hoja de estilos, permitiendo futuras modificaciones sin tocar el código PHP.

4.10.6. Conclusión

Con esta fase, el blog adquiere una identidad visual clara, atractiva y estructurada. Todos los elementos —texto, imágenes, metadatos— quedan correctamente presentados, permitiendo que el lector navegue el contenido con comodidad. El código sigue limpio y modular, y la separación entre datos, lógica y presentación es total, cumpliendo con las mejores prácticas del desarrollo web moderno. El blog está listo para ser

extendido con nuevas páginas, funcionalidades interactivas o integración con sistemas de gestión de contenidos.

```
index.php
<?php
    include "inc/cabecera.php";
    include "clases/Publicacion.php";
    // Creo un array de entradas
    $entradas = [];
    // Cargo el archivo json que contiene datos
    $archivo = file_get_contents("datos/entradas.json");
    $json = json_decode($archivo,true);
    // Convierto los elementos JSON en Objetos de mi
aplicacion
    foreach($json as $entrada){
        array_push(
            $entradas,
            new Entrada(
                $entrada["titulo"],
                $entrada["contenido"],
                $entrada["fecha"],
                $entrada["autor"],
                $entrada["imagen"]
            )
        );
    }
    // Recorro los objetos y los pinto en pantalla
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo":"Primera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la primera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Segunda entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la segunda
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Tercera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la tercera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Cuarta entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la cuarta
entrada",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        private $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }
    class Pagina extends Publicacion{

    }
    class Entrada extends Publicacion{
```

```
    }

?>
```

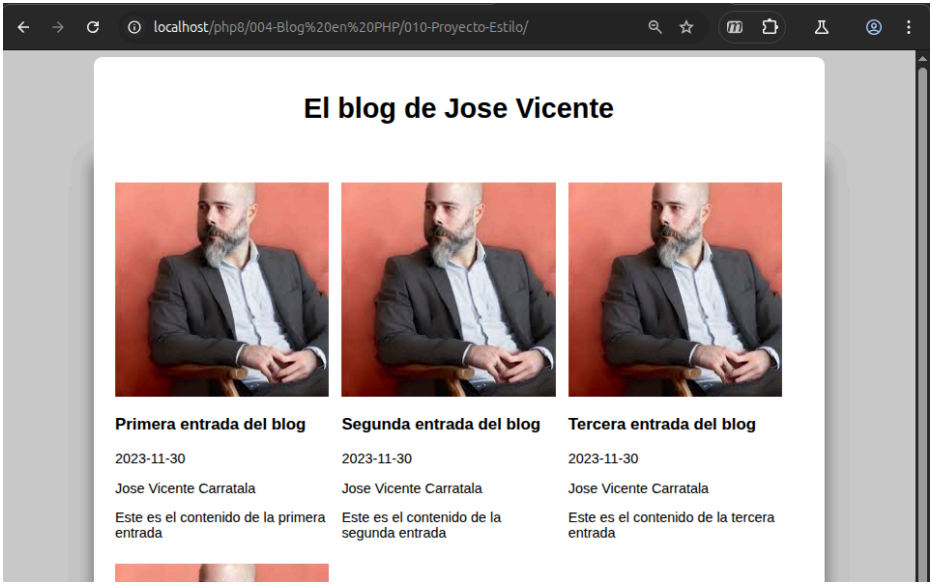
.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:



4.11. Proyecto-JSON páginas

Para gestionar múltiples tipos de contenido, como páginas además de entradas, se puede crear un archivo JSON adicional que almacene los datos de las páginas estáticas. Este archivo contiene objetos con propiedades similares a las entradas, como título, autor, fecha, contenido e imagen.

Al cargar ambos archivos JSON en el script PHP, se convierten en objetos correspondientes mediante la creación de instancias de clases específicas, por ejemplo, `Pagina` para páginas y `Entrada` para entradas.

Esta separación permite organizar y mostrar distintos tipos de contenido de forma independiente, manteniendo una estructura clara y facilitando la ampliación o modificación del sitio web.

Estructura del proyecto:

```
011-Proyecto-JSON paginas
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este undécimo subcapítulo, titulado **Proyecto - JSON páginas**, el proyecto amplía su arquitectura de datos para permitir no solo la gestión de entradas de blog, sino también la incorporación de **páginas estáticas** (como “Sobre mí”, “Contacto”, etc.) que se almacenan en un archivo JSON aparte. Se establece así un sistema multisección donde tanto entradas como páginas comparten la misma lógica de visualización, pero con distinta categoría de contenido.

4.11.1. Estructura del proyecto

En esta etapa se añade un nuevo archivo de datos y se reutilizan las clases y plantillas existentes:

```
011-Proyecto-JSON paginas
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

El nuevo archivo clave es `datos/paginas.json`, que contiene el contenido de las páginas estáticas.

4.11.2. Qué se está haciendo

El cambio principal en esta fase es que el sistema PHP ya no solo lee `entradas.json` y genera objetos de tipo `Entrada`, sino que **también lee `paginas.json` y genera objetos de tipo `Pagina`**, otra clase hija de `Publicacion`. Esto permite mantener ambos tipos de contenido en estructuras separadas, pero renderizarlos con la misma lógica de visualización, utilizando el método `damePublicacion()`.

Este paso representa un avance en la generalización del sistema, preparando el terreno para que el blog evolucione hacia una estructura de sitio web completo.

4.11.3. Contenido de datos/paginas.json

Este archivo es estructuralmente idéntico a `entradas.json`, pero con contenido semánticamente distinto. Por ejemplo:

```
[
    {
        "titulo": "Sobre mí",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-12-01",
        "contenido": "Esta página describe al autor del
blog.",
        "imagen": "autor.jpg"
    },
    {
        "titulo": "Contacto",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-12-02",
        "contenido": "Formulario y datos de contacto del
autor.",
        "imagen": "contacto.jpg"
    }
]
```

4.11.4. Cambios en index.php

El archivo PHP se modifica para cargar y renderizar **dos tipos de contenido**:

1. Entradas desde `entradas.json` usando la clase `Entrada`.

2. Páginas desde `paginas.json` usando la clase `Pagina`.

Ambas clases heredan de `Publicacion` y utilizan el mismo método `damePublicacion()` para mostrarse.

Fragmento ilustrativo:

```
$entradas = [];
$paginas = [];

$datosEntradas =
json_decode(file_get_contents("datos/entradas.json"), true);
foreach($datosEntradas as $elemento){
    $entrada = new Entrada($elemento["titulo"],
$elemento["contenido"]);
    $entrada->fecha = $elemento["fecha"];
    $entrada->autor = $elemento["autor"];
    $entrada->imagen = $elemento["imagen"];
    array_push($entradas, $entrada);
}

$datosPaginas =
json_decode(file_get_contents("datos/paginas.json"), true);
foreach($datosPaginas as $elemento){
    $pagina = new Pagina($elemento["titulo"],
$elemento["contenido"]);
    $pagina->fecha = $elemento["fecha"];
    $pagina->autor = $elemento["autor"];
    $pagina->imagen = $elemento["imagen"];
    array_push($paginas, $pagina);
}
```

Después se pueden mostrar todas las publicaciones, primero entradas y luego páginas (o al revés), según el criterio deseado:

```
foreach($entradas as $entrada){
    echo $entrada->damePublicacion();
}

foreach($paginas as $pagina){
    echo $pagina->damePublicacion();
}
```

4.11.5. Cambios en clases/Publicacion.php

No es necesario modificar el archivo de clases en esta etapa, ya que `Pagina` y `Entrada` ya están definidas como clases hijas de `Publicacion`. Ambas reutilizan el constructor heredado y el método `damePublicacion`, lo que simplifica el proceso.

4.11.6. Justificación de los cambios

- **Generalización del sistema**: el blog ya no se limita a publicaciones cronológicas; ahora puede mostrar secciones fijas.
- **Separación de tipos de contenido**: entradas y páginas se gestionan y almacenan por separado, lo que permite lógicas distintas en el futuro.
- **Reutilización de lógica existente**: se aprovecha el sistema de clases sin necesidad de duplicar código.
- **Escalabilidad**: esta estructura facilita agregar categorías, filtros, menús o enlaces directos a páginas específicas.

4.11.7. Conclusión

Este paso transforma el blog en un sistema de gestión de contenidos más completo. Gracias a la introducción de páginas como entidades diferenciadas, el sitio puede incorporar secciones estáticas permanentes, sin perder la flexibilidad de las entradas dinámicas. La estructura es extensible y lista para integrar navegación, enlaces internos, o plantillas personalizadas por tipo de contenido.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    // Creo un array de entradas
    $entradas = [];
    // Cargo el archivo json que contiene datos
    $archivo = file_get_contents("datos/entradas.json");
    $json = json_decode($archivo,true);
    // Convierto los elementos JSON en Objetos de mi
aplicacion
    foreach($json as $entrada){
        array_push(
            $entradas,
            new Entrada(
                $entrada["titulo"],
                $entrada["contenido"],
                $entrada["fecha"],
                $entrada["autor"],
                $entrada["imagen"]
            )
        );
    }
    // Recorro los objetos y los pinto en pantalla
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo":"Primera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la primera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Segunda entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la segunda
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Tercera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la tercera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Cuarta entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la cuarta
entrada",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
```

```
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

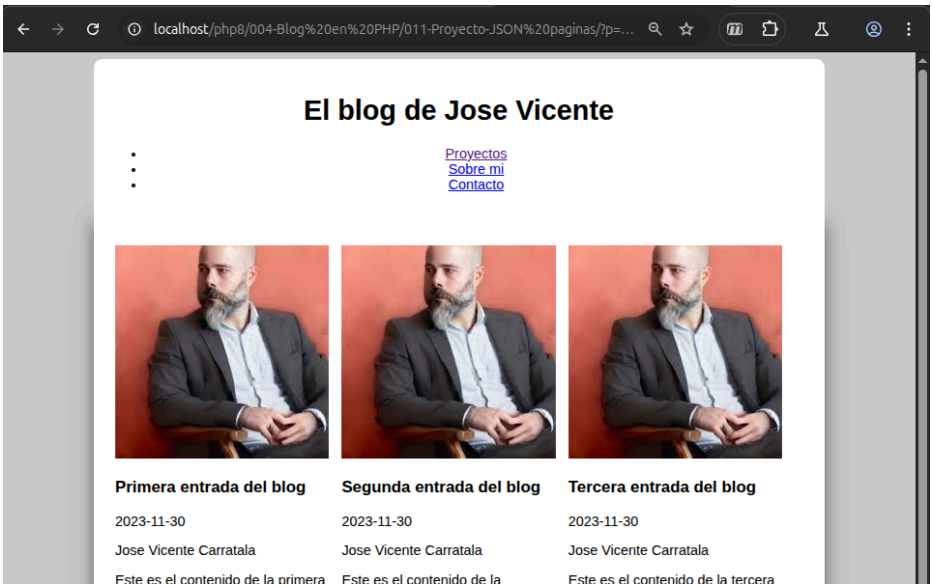
```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:



4.12. Proyecto-Estilo

Para lograr una navegación dinámica entre páginas, se puede implementar un menú que se construya a partir de un array de objetos

`Pagina`. Este menú se genera recorriendo el array y llamando a un método que devuelve el código HTML correspondiente a cada enlace.

De esta forma, el menú refleja automáticamente las páginas disponibles, facilitando la actualización y ampliación sin modificar el código estático. El usuario puede navegar entre las distintas secciones del sitio, y el contenido mostrado se ajusta según la página seleccionada.

Este enfoque mejora la experiencia del usuario y simplifica la gestión del contenido, permitiendo un diseño modular y escalable del sitio web.

Estructura del proyecto:

```
012-Proyecto-Estilo
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este duodécimo subcapítulo, titulado **Proyecto - Estilo**, se culmina la presentación estética del blog incorporando reglas CSS que distinguen visualmente las **entradas** de las **páginas estáticas**, ahora que ambas coexisten en el sistema. El objetivo es aportar claridad visual al usuario,

reforzar la estructura de navegación del sitio y preparar el proyecto para una experiencia de lectura más refinada y segmentada.

4.12.1. Estructura del proyecto

No hay cambios estructurales en carpetas o archivos respecto al paso anterior. La intervención principal se realiza exclusivamente sobre la hoja de estilos:

```
012-Proyecto-Estilo
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
└── index.php
```

4.12.2. Qué se está haciendo

En este punto del desarrollo, tanto entradas como páginas comparten la misma estructura HTML, pero ahora se desea **diferenciarlas visualmente** usando CSS. Para lograr esto, el método `damePublicacion()` de cada clase puede emitir una clase específica en el HTML (`<article class="entrada">` o `<article class="pagina">`). A partir de ahí, se aplican estilos distintos a cada tipo de publicación.

De esta forma:

- Las **entradas** mantienen un aspecto tradicional de blog, con protagonismo en el contenido cronológico.

- Las **páginas** reciben una presentación diferenciada, más institucional, útil para secciones como “Sobre mí”, “Contacto”, etc.

4.12.3. Cambios en clases/Publicacion.php

Se actualiza el método `damePublicacion()` de ambas subclases (`Entrada`, `Pagina`) para incorporar una clase CSS específica en la etiqueta `<article>`:

```
// En clase Entrada
function damePublicacion(){
    return '
        <article class="entrada">
            <img src="imagenes/'.$this->imagen.'" alt="">
            <h3>'.$this->titulo.'</h3>
            <time>'.$this->fecha.'</time>
            <p>'.$this->contenido.'</p>
            <address>'.$this->autor.'</address>
        </article>
    ';
}

// En clase Pagina
function damePublicacion(){
    return '
        <article class="pagina">
            <img src="imagenes/'.$this->imagen.'" alt="">
            <h3>'.$this->titulo.'</h3>
            <time>'.$this->fecha.'</time>
            <p>'.$this->contenido.'</p>
            <address>'.$this->autor.'</address>
        </article>
    ';
}
```

4.12.4. Cambios en css/estilo.css

Se añaden reglas específicas para cada clase, permitiendo personalizar la apariencia de entradas y páginas. Por ejemplo:

```
article.entrada {
    background-color: #ffffff;
    border-left: 6px solid #0088cc;
    padding: 20px;
    margin-bottom: 30px;
}

article.pagina {
    background-color: #f9f9f9;
    border-left: 6px solid #444444;
    padding: 25px;
    margin-bottom: 40px;
}

article.pagina h3 {
    font-size: 1.6em;
    color: #333;
}

article.entrada h3 {
    font-size: 1.4em;
    color: #0088cc;
}
```

Estos estilos:

- Aplican colores de fondo distintos para diferenciar contextos.
- Usan bordes laterales de diferente color para crear un código visual.
- Ajustan los márgenes y paddings para reforzar el espacio visual.
- Estilizan títulos para cada tipo de contenido, haciendo que el lector perciba claramente si está leyendo una entrada cronológica o una sección institucional.

4.12.5. Archivos index.php y *.json

No requieren cambios. Ambos tipos de objetos (`Entrada`, `Pagina`) ya se instancian correctamente y su método `damePublicacion()` es responsable de aplicar las clases CSS adecuadas.

4.12.6. Justificación de los cambios

- **Claridad de navegación**: los usuarios pueden identificar fácilmente el tipo de contenido que están leyendo.
- **Refuerzo visual**: se introduce una jerarquía estética que refleja la estructura lógica del sitio.
- **Preparación para el crecimiento**: permite escalar el diseño añadiendo nuevos estilos condicionales, como categorías o etiquetas.
- **Separación total de responsabilidades**: los objetos generan clases específicas y el CSS decide cómo se presentan, cumpliendo el principio de separación de contenido y estilo.

4.12.7. Conclusión

Con esta fase, el blog ya no solo es funcional y estructuralmente modular, sino que también es visualmente jerárquico. Se consolida un sistema de diseño basado en clases que permite aplicar temas, distinguir tipos de contenido y adaptar el sitio a diferentes públicos o contextos. Se trata de un paso decisivo hacia un producto final profesional, accesible y preparado para integrarse en un entorno web real.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    // Creo un array de entradas
    $entradas = [];
    // Cargo el archivo json que contiene datos
    $archivo = file_get_contents("datos/entradas.json");
    $json = json_decode($archivo,true);
    // Convierto los elementos JSON en Objetos de mi
aplicacion
    foreach($json as $entrada){
        array_push(
            $entradas,
            new Entrada(
                $entrada["titulo"],
                $entrada["contenido"],
                $entrada["fecha"],
                $entrada["autor"],
                $entrada["imagen"]
            )
        );
    }
    // Recorro los objetos y los pinto en pantalla
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo":"Primera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la primera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Segunda entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la segunda
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Tercera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la tercera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Cuarta entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la cuarta
entrada",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
nav ul li{
    display:inline-block;
}
nav ul li a{
    text-decoration: none;
    color:inherit;
    text-transform:uppercase;
```

```
    font-size:10px;
    margin-left:10px;
    margin-right:10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
```

```
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:

4.13. Proyecto-Panel de control

El archivo CSS asociado al proyecto define estilos para los elementos del menú de navegación, como los enlaces de la lista. Se establecen

propiedades para mostrar los elementos en línea (`inline-block`), eliminar el subrayado (`text-decoration: none`), aplicar mayúsculas (`text-transform: uppercase`) y ajustar márgenes y tamaño de fuente.

Estas reglas permiten que el menú tenga una apariencia limpia, ordenada y fácilmente legible, contribuyendo a la usabilidad y estética del sitio. Separar el estilo en un archivo externo facilita su mantenimiento y la coherencia visual en todas las páginas.

Estructura del proyecto:

```
013-Proyecto-Panel de control
├── admin
│   └── index.php
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este decimotercer subcapítulo, titulado **Proyecto - Panel de control**, el proyecto da un salto cualitativo hacia la administración interactiva del

contenido. Por primera vez, se introduce una **interfaz de gestión** desde la cual el usuario puede visualizar o controlar internamente las entradas del blog, preparando el terreno para futuras operaciones como añadir, modificar o eliminar contenido. Aunque en esta etapa el panel todavía no interactúa con el sistema de archivos o bases de datos, representa el primer esbozo de una **zona privada de administración**.

4.13.1. Estructura del proyecto

Se amplía el conjunto de archivos con una nueva página llamada `panel.php`, que actúa como interfaz de control:

```
013-Proyecto-Panel de control
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
└── panel.php
```

4.13.2. Qué se está haciendo

Se crea una nueva vista (archivo `panel.php`) con las siguientes características:

- Comparte la misma estructura visual que `index.php` (mediante `include` de cabecera y pie).
- Carga el contenido desde `entradas.json`.

- Muestra la lista de entradas, pero con una **presentación más técnica y orientada al control**: en lugar de HTML decorado, se usa `var_dump` o se imprimen los valores directamente con `echo`.

- Refuerza la separación entre la **zona pública del blog** y la **zona de administración**.

4.13.3. Contenido de panel.php

El archivo sigue esta estructura:

```
<?php
include "inc/cabecera.php";
include "clases/Publicacion.php";

$entradas = [];
$datos =
json_decode(file_get_contents("datos/entradas.json"), true);

foreach($datos as $elemento){
    $entrada = new Entrada($elemento["titulo"],
$elemento["contenido"]);
    $entrada->fecha = $elemento["fecha"];
    $entrada->autor = $elemento["autor"];
    $entrada->imagen = $elemento["imagen"];
    array_push($entradas, $entrada);
}

// Visualización básica (modo lista)
echo "<h2>Panel de control: Entradas actuales</h2>";
echo "<ul>";
foreach($entradas as $entrada){
    echo "<li>".$entrada->getTitulo()."</li>";
}
echo "</ul>";

include "inc/piedepagina.php";
?>
```

Este código:

- Carga las entradas del JSON igual que `index.php`.
- En lugar de renderizarlas como artículos HTML, **las muestra como elementos de lista** (`<li>`), lo que es más apropiado para un panel de control.

- Refuerza el concepto de área privada, sin decoración ni estilos visuales llamativos.

4.13.4. Cambios en otros archivos

- **`clases/Publicacion.php`**: no se modifica. Se reutiliza la clase `Entrada` para acceder a los títulos.

- **`estilo.css`**: puede permanecer sin cambios, aunque eventualmente se podrá añadir un tema visual diferenciado para el panel de control.

- **`cabecera.php`**: al ser incluida también por `panel.php`, es posible que se le añada en el futuro una navegación condicional para distinguir entre la vista pública y la administrativa.

4.13.5. Justificación de los cambios

- **Inicio del sistema de administración**: el proyecto deja de ser solo una salida de datos para convertirse en un sistema interactivo.

- **Preparación para edición**: se crea la base sobre la cual se podrán añadir formularios, botones de acción, validaciones y escritura en el JSON.

- **Separación conceptual**: se establece una diferencia clara entre lo que ve el lector (`index.php`) y lo que ve el administrador (`panel.php`).

- **Reutilización de clases**: se confirma que el modelo de datos es lo bastante sólido como para alimentar múltiples vistas.

4.13.6. Conclusión

Este paso marca el nacimiento del backend del proyecto. Aunque todavía es estático y solo muestra datos, la creación de un panel de control indica una clara intención de gestionar el blog desde la web. Es el comienzo de un sistema de gestión de contenidos propio, y permitirá, en los siguientes pasos, implementar formularios, autenticación, edición dinámica de archivos JSON y un verdadero flujo de publicación.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    // Creo un array de entradas
    $entradas = [];
    // Cargo el archivo json que contiene datos
    $archivo = file_get_contents("datos/entradas.json");
    $json = json_decode($archivo,true);
    // Convierto los elementos JSON en Objetos de mi
aplicacion
    foreach($json as $entrada){
        array_push(
            $entradas,
            new Entrada(
                $entrada["titulo"],
                $entrada["contenido"],
                $entrada["fecha"],
                $entrada["autor"],
                $entrada["imagen"]
            )
        );
    }
    // Recorro los objetos y los pinto en pantalla
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo":"Primera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la primera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Segunda entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la segunda
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Tercera entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la tercera
entrada",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Cuarta entrada del blog",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"Este es el contenido de la cuarta
entrada",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
admin/index.php
<h3>Nueva entrada</h3>
<form action="?" method="POST">
    <input type="text" name="titulo" placeholder="titulo">
    <input type="text" name="autor" placeholder="autor">
    <input type="date" name="fecha" placeholder="fecha">
    <input type="text" name="contenido"
placeholder="contenido">
    <input type="text" name="imagen" placeholder="imagen">
    <input type="submit">
</form>
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
nav ul li{
    display:inline-block;
}
nav ul li a{
    text-decoration: none;
    color:inherit;
    text-transform:uppercase;
```

```
    font-size:10px;
    margin-left:10px;
    margin-right:10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
```

```
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:

4.14. Proyecto-Procesar entrada

El archivo `Publicacion.php` contiene la definición de una clase abstracta `Publicacion` que incluye propiedades protegidas para almacenar información común a todas las publicaciones, como título, contenido, fecha, autor e imagen.

Esta clase proporciona métodos para establecer y obtener el título, con validaciones incluidas, y sirve como base para clases derivadas como `Pagina` y `Entrada`. Estas clases extienden `Publicacion` y pueden agregar funcionalidades específicas.

El método `damePublicacion()` en la clase `Entrada` genera una representación HTML completa de la publicación, incluyendo la imagen, título, fecha, autor y contenido, facilitando así la presentación dinámica de los datos en la página web.

Estructura del proyecto:

```
014-Proyecto-Procesar entrada
├── admin
│   └── index.php
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   ├── entradas2.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este decimocuarto subcapítulo, titulado **Proyecto - Procesar entrada**, el proyecto introduce la capacidad de **crear nuevas entradas de blog desde el navegador**, utilizando un **formulario HTML** y almacenando los datos enviados en el archivo `entradas.json`. Este es un paso clave: transforma el sitio de un visor estático de contenidos a una **plataforma interactiva con persistencia dinámica**, aunque en esta etapa inicial todavía con control limitado.

4.14.1. Estructura del proyecto

Se añade un nuevo archivo llamado `procesar.php`, que será el encargado de recibir los datos del formulario y actualizar el archivo de entradas:

```
014-Proyecto-Procesar entrada
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
├── panel.php
└── procesar.php
```

4.14.2. Qué se está haciendo

Se implementa un flujo completo de alta de contenido:

1. En `panel.php`, se presenta un **formulario** al usuario para introducir los campos de una nueva entrada (`titulo`, `contenido`, `autor`, `fecha`, `imagen`).

2. Al enviar el formulario, los datos llegan a `procesar.php` mediante `POST`.

3. En `procesar.php`, los datos del formulario se integran con las entradas existentes en `entradas.json`.

4. Se reescribe el archivo JSON con el nuevo conjunto de entradas, **incluyendo la nueva**.

Esto significa que **por primera vez se escribe sobre disco**, utilizando `file_put_contents`.

4.14.3. Contenido de panel.php (actualizado)

Ahora contiene un formulario como este:

```
<h2>Nueva entrada</h2>
<form method="POST" action="procesar.php">
    <input type="text" name="titulo"
placeholder="Título"><br>
    <input type="text" name="autor" placeholder="Autor"><br>
    <input type="text" name="fecha" placeholder="Fecha
(YYYY-MM-DD)"><br>
    <input type="text" name="imagen" placeholder="Nombre de
la imagen"><br>
    <textarea name="contenido"
placeholder="Contenido"></textarea><br>
    <input type="submit" value="Guardar entrada">
</form>
```

4.14.4. Contenido de procesar.php

Este archivo recibe los datos del formulario, los valida mínimamente, los agrega al archivo `entradas.json` y redirige al panel:

```
<?php
// Leer entradas actuales
$datos =
json_decode(file_get_contents("datos/entradas.json"), true);

// Crear nuevo elemento
$nuevaEntrada = [
    "titulo" => $_POST["titulo"],
    "autor" => $_POST["autor"],
    "fecha" => $_POST["fecha"],
    "contenido" => $_POST["contenido"],
    "imagen" => $_POST["imagen"]
];

// Añadir al array
array_push($datos, $nuevaEntrada);

// Guardar de nuevo en JSON
file_put_contents("datos/entradas.json", json_encode($datos,
JSON_PRETTY_PRINT));

// Redirigir de nuevo al panel
header("Location: panel.php");
exit;
?>
```

4.14.5. Consideraciones técnicas

- **No hay validación** en este punto: se confía plenamente en lo que el usuario introduce (esto se mejorará más adelante).
- **El guardado es en formato JSON plano**, sin base de datos.
- **No hay control de acceso**, cualquier visitante podría agregar contenido si conoce la URL.

4.14.6. Justificación de los cambios

- **Interactividad real**: el blog se convierte en un sistema de gestión de contenido (CMS) básico.
- **Persistencia dinámica**: los cambios quedan guardados entre sesiones.
- **Arquitectura robusta**: se separa la interfaz (`panel.php`) del controlador (`procesar.php`) y del modelo (`entradas.json`).
- **Expansión natural**: este mecanismo será la base para implementar edición y borrado de entradas, subida de imágenes, validaciones, etc.

4.14.7. Conclusión

Este subcapítulo marca el momento en que el blog deja de ser una colección de archivos estáticos para convertirse en una aplicación web funcional. La lógica de escritura permite al usuario generar contenido directamente desde el navegador, abriendo la puerta a un panel administrativo real. Aunque todavía básico, este flujo `formulario → POST → escritura → redirección` es el patrón base sobre el que se construirán las futuras capacidades de administración.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    // Creo un array de entradas
    $entradas = [];
    // Cargo el archivo json que contiene datos
    $archivo = file_get_contents("datos/entradas.json");
    $json = json_decode($archivo,true);
    // Convierto los elementos JSON en Objetos de mi
aplicacion
    foreach($json as $entrada){
        array_push(
            $entradas,
            new Entrada(
                $entrada["titulo"],
                $entrada["contenido"],
                $entrada["fecha"],
                $entrada["autor"],
                $entrada["imagen"]
            )
        );
    }
    // Recorro los objetos y los pinto en pantalla
   for($i = 0;$i<count($entradas);$i++){
        echo $entradas[$i]->damePublicacion();
    }

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Quinta entrada del blog",
        "autor": "Jose Vicente",
        "fecha": "2023-11-15",
        "contenido": "Esto es una prueba",
        "imagen": "josevicente.jpg"
    },
```

```
    {
        "titulo": "Sexta entrada del blog",
        "autor": "JOse Vicente",
        "fecha": "2023-11-13",
        "contenido": "Este es un contenido",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/entradas2.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
admin/index.php
<?php
    if(isset($_POST['titulo'])){
        $archivo =
file_get_contents("../datos/entradas.json");
        $json = json_decode($archivo);
        array_push($json,$_POST);
        $nuevojson = json_encode($json, JSON_PRETTY_PRINT);

file_put_contents("../datos/entradas.json",$nuevojson);
    }
?>
<h3>Nueva entrada</h3>
<form action="?" method="POST">
    <input type="text" name="titulo" placeholder="titulo">
    <input type="text" name="autor" placeholder="autor">
    <input type="date" name="fecha" placeholder="fecha">
    <input type="text" name="contenido"
placeholder="contenido">
    <input type="text" name="imagen" placeholder="imagen">
    <input type="submit">
</form>
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
nav ul li{
    display:inline-block;
}
nav ul li a{
    text-decoration: none;
    color:inherit;
    text-transform:uppercase;
```

```
    font-size:10px;
    margin-left:10px;
    margin-right:10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
```

```
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:

4.15. Proyecto-Refactorización

El archivo `cabecera.php` contiene el código HTML que define la estructura inicial de la página, incluyendo la declaración del tipo de documento, la apertura de las etiquetas `<html>`, `<head>` y `<body>`. En la sección `<head>`, se incluye el título de la página y la referencia al archivo CSS que define los estilos.

Dentro del `<body>`, se encuentra la cabecera visual de la página con un título principal y un menú de navegación dinámico. Este menú se genera leyendo un archivo JSON que contiene las páginas disponibles, convirtiéndolas en objetos y recorriéndolos para crear enlaces que permitan navegar entre las distintas secciones del sitio.

Estructura del proyecto:

```
015-Proyecto-Refactorizacion
├── admin
│   └── index.php
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   ├── entradas2.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   ├── funciones.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este decimoquinto subcapítulo, titulado **Proyecto - Refactorización**, se aborda una etapa crucial del ciclo de vida del software: la reorganización interna del código para mejorar su **legibilidad, reutilización y mantenibilidad**, sin alterar su comportamiento externo. En este punto del proyecto del blog, el código ya es funcional y permite leer entradas, mostrar páginas y añadir nuevos contenidos desde un formulario. Sin embargo, ciertas tareas —como cargar los objetos desde JSON— se repiten en varios archivos. Este paso se centra en **centralizar esa lógica común** para evitar duplicación y facilitar futuras extensiones.

4.15.1. Estructura del proyecto

Se mantiene sin cambios visibles en la estructura de carpetas. Sin embargo, se introducen **nuevas funciones reutilizables** en un archivo que puede estar ubicado dentro de `clases` u otro espacio lógico, o directamente integradas temporalmente en los scripts.

```
015-Proyecto-Refactorización
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   └── entradas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
├── panel.php
└── procesar.php
```

4.15.2. Qué se está haciendo

Hasta este momento, los archivos `index.php`, `panel.php` y `procesar.php` **duplicaban código** para leer el archivo JSON, decodificarlo, crear objetos `Entrada` o `Pagina`, y agregarlos a arrays. En esta fase se procede a:

- Encapsular la carga de entradas en una **función reutilizable**.
- Usar dicha función en los distintos archivos en lugar de repetir la lógica.
- Preparar la base para centralizar también otras operaciones como escritura o filtrado.

4.15.3. Refactorización en la práctica

1. Creación de una función de utilidad

Puede añadirse en `clases/Publicacion.php` o en un archivo auxiliar llamado `funciones.php`, con algo como:

```
function cargarEntradasDesdeJSON($ruta){
    $entradas = [];
    $datos = json_decode(file_get_contents($ruta), true);

    foreach($datos as $elemento){
        $entrada = new Entrada($elemento["titulo"],
$elemento["contenido"]);
        $entrada->fecha = $elemento["fecha"];
        $entrada->autor = $elemento["autor"];
        $entrada->imagen = $elemento["imagen"];
        array_push($entradas, $entrada);
    }

    return $entradas;
}
```

2. Aplicación en `index.php`

```
include "clases/Publicacion.php";
include "funciones.php";

$entradas = cargarEntradasDesdeJSON("datos/entradas.json");

foreach($entradas as $entrada){
    echo $entrada->damePublicacion();
}
```

3. Aplicación en `panel.php`

```
include "clases/Publicacion.php";
include "funciones.php";

$entradas = cargarEntradasDesdeJSON("datos/entradas.json");

echo "<ul>";
foreach($entradas as $entrada){
    echo "<li>".$entrada->getTitulo()."</li>";
}
echo "</ul>";
```

4.15.4. Qué se logra con esta refactorización

- **Reutilización del código**: un único bloque gestiona la lectura de entradas desde JSON.
- **Reducción del error humano**: al centralizar la lógica, si se necesita cambiar algo (por ejemplo, validar los datos o transformar fechas), solo se hace en un lugar.
- **Mejora de la organización**: separa claramente el modelo de los controladores y vistas.
- **Facilita la extensión**: futuras funciones como `cargarPaginasDesdeJSON()` o `guardarEntradasEnJSON()` seguirán este patrón.

4.15.5. Justificación de los cambios

- **Buena práctica profesional**: refactorizar cuando una funcionalidad se repite más de una vez es un principio básico de diseño limpio.

- **Escalabilidad futura**: esta mejora prepara al proyecto para estructuras más complejas, como APIs REST o MVC.

- **Mantenibilidad**: si el sistema crece, será más fácil localizar y modificar funciones.

4.15.6. Conclusión

Este paso representa una maduración del proyecto: ya no se limita a "hacer que funcione", sino que empieza a centrarse en "hacerlo bien". La refactorización convierte el código en una base sólida, clara y ordenada sobre la que se puede construir sin que cada nueva funcionalidad duplique código o genere inconsistencias. Es un hito silencioso pero fundamental en la evolución de cualquier proyecto sostenible.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    include "inc/funciones.php";
    bucleEntradas();

    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Quinta entrada del blog",
        "autor": "Jose Vicente",
        "fecha": "2023-11-15",
        "contenido": "Esto es una prueba",
        "imagen": "josevicente.jpg"
    },
```

```
    {
        "titulo": "Sexta entrada del blog",
        "autor": "JOse Vicente",
        "fecha": "2023-11-13",
        "contenido": "Este es un contenido",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/entradas2.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
admin/index.php
<?php
    if(isset($_POST['titulo'])){
        $archivo =
file_get_contents("../datos/entradas.json");
        $json = json_decode($archivo);
        array_push($json,$_POST);
        $nuevojson = json_encode($json, JSON_PRETTY_PRINT);

file_put_contents("../datos/entradas.json",$nuevojson);
    }
?>
<h3>Nueva entrada</h3>
<form action="?" method="POST">
    <input type="text" name="titulo" placeholder="titulo">
    <input type="text" name="autor" placeholder="autor">
    <input type="date" name="fecha" placeholder="fecha">
    <input type="text" name="contenido"
placeholder="contenido">
    <input type="text" name="imagen" placeholder="imagen">
    <input type="submit">
</form>
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
nav ul li{
    display:inline-block;
}
nav ul li a{
    text-decoration: none;
    color:inherit;
    text-transform:uppercase;
```

```
    font-size:10px;
    margin-left:10px;
    margin-right:10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
```

```
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/funciones.php
<?php

    function bucleEntradas(){
        // Creo un array de entradas
        $entradas = [];
        // Cargo el archivo json que contiene datos
        $archivo = file_get_contents("datos/entradas.json");
        $json = json_decode($archivo,true);
        // Convierto los elementos JSON en Objetos de mi
aplicacion
        foreach($json as $entrada){
            array_push(
                $entradas,
                new Entrada(
                    $entrada["titulo"],
                    $entrada["contenido"],
                    $entrada["fecha"],
                    $entrada["autor"],
                    $entrada["imagen"]
                )
            );
        }
        // Recorro los objetos y los pinto en pantalla
       for($i = 0;$i<count($entradas);$i++){
            echo $entradas[$i]->damePublicacion();
        }
    }

?>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:

4.16. Proyecto-Cargar paginas

El archivo `piedepagina.php` contiene el código HTML que cierra las secciones abiertas en la cabecera, como la etiqueta `<main>`, y define el pie de página del sitio. En este pie de página se incluye un texto con derechos de autor, generalmente con el año y el nombre del autor o la organización.

Además, se cierran las etiquetas `<body>` y `<html>`, completando la estructura del documento HTML. Este archivo se incluye en las páginas para mantener una presentación coherente y facilitar su mantenimiento centralizado.

Estructura del proyecto:

```
016-Proyecto-Cargar paginas
├── admin
│   └── index.php
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   ├── entradas2.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   ├── funciones.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este decimosexto subcapítulo, titulado **Proyecto - Cargar páginas**, se implementa la funcionalidad que permite al sistema **mostrar de forma dinámica páginas estáticas individuales**, seleccionadas por el usuario desde la URL. Hasta ahora, las páginas definidas en `paginas.json` se cargaban como un bloque junto al resto del contenido, pero sin posibilidad de navegación individual. En este paso, el blog evoluciona para permitir que cada página estática pueda ser accedida como si fuese una sección independiente del sitio.

4.16.1. Estructura del proyecto

Se introduce un nuevo archivo que actúa como plantilla para visualizar páginas individuales:

```
016-Proyecto-Cargar paginas
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
├── panel.php
├── procesar.php
└── pagina.php
```

El nuevo archivo clave es `pagina.php`, encargado de cargar una página concreta en base a un parámetro GET (`?id=`).

4.16.2. Qué se está haciendo

El sistema ya dispone de un conjunto de páginas definidas en `paginas.json`. En esta fase se permite:

1. **Acceder a una página específica** a través de la URL.
2. **Filtrar el array JSON para encontrar la página correspondiente** por su índice o identificador.
3. **Crear un objeto `Pagina` y renderizarlo individualmente** usando su método `damePublicacion()`.

4.16.3. Acceso desde la URL

El acceso a una página se hace mediante una URL como:

pagina.php?id=1

Donde el valor `1` hace referencia a la posición (índice) de la página dentro del array `paginas.json`.

4.16.4. Contenido de pagina.php

Este archivo realiza los siguientes pasos:

1. Recupera el parámetro `id` desde `$_GET`.
2. Carga el archivo `paginas.json`.
3. Extrae la página correspondiente si existe.
4. La convierte en un objeto `Pagina`.
5. La muestra dentro de la estructura general del sitio.

Ejemplo funcional:

```
<?php
include "inc/cabecera.php";
include "clases/Publicacion.php";

$id = $_GET["id"];

$datos = json_decode(file_get_contents("datos/paginas.json"),
true);

if(isset($datos[$id])){
    $elemento = $datos[$id];
    $pagina = new Pagina($elemento["titulo"],
$elemento["contenido"]);
    $pagina->fecha = $elemento["fecha"];
    $pagina->autor = $elemento["autor"];
    $pagina->imagen = $elemento["imagen"];

    echo $pagina->damePublicacion();
} else {
    echo "<p>Página no encontrada.</p>";
}

include "inc/piedepagina.php";
?>
```

4.16.5. Cambios en clases/Publicacion.php

No se requieren cambios. La clase `Pagina` ya hereda de `Publicacion` y puede utilizar `damePublicacion()` para renderizarse visualmente.

4.16.6. Justificación de los cambios

- **Navegación real**: por primera vez, el sistema permite acceder a contenidos individuales mediante URL.

- **Versatilidad**: esta técnica puede extenderse fácilmente a `entrada.php?id=`, para entradas individuales.

- **Separación de rutas**: el sistema distingue claramente entre índice (vista general), panel (zona administrativa) y contenido puntual.

- **Preparación para menús y enlaces**: ahora se pueden generar listados de enlaces a páginas individuales.

4.16.7. Consideraciones futuras

- **Mejorar el identificador**: en vez de un índice numérico, podría usarse una clave (`slug`) o campo único.

- **Validación y seguridad**: sanitizar `$_GET["id"]` para evitar errores o ataques.

- **404 personalizado**: en lugar de un mensaje simple, mostrar una página de error con estilo consistente.

4.16.8. Conclusión

Con este paso, el proyecto adquiere navegación individualizada de páginas, transformándose en un sistema de publicación más completo. El sitio ya puede mostrar secciones estáticas independientes, como "Sobre mí", "Contacto", o "Política de privacidad", accesibles directamente desde menús o enlaces, y renderizadas dinámicamente desde los datos JSON. Es una mejora clave en funcionalidad, navegación y experiencia de usuario.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    include "inc/funciones.php";
    if(!isset($_GET['p'])){
        bucleEntradas();
    }else{
        echo "Pues vamos a cargar una pagina";
    }


    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Quinta entrada del blog",
        "autor": "Jose Vicente",
        "fecha": "2023-11-15",
        "contenido": "Esto es una prueba",
        "imagen": "josevicente.jpg"
    },
```

```
    {
        "titulo": "Sexta entrada del blog",
        "autor": "JOse Vicente",
        "fecha": "2023-11-13",
        "contenido": "Este es un contenido",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/entradas2.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
admin/index.php
<?php
    if(isset($_POST['titulo'])){
        $archivo =
file_get_contents("../datos/entradas.json");
        $json = json_decode($archivo);
        array_push($json,$_POST);
        $nuevojson = json_encode($json, JSON_PRETTY_PRINT);

file_put_contents("../datos/entradas.json",$nuevojson);
    }
?>
<h3>Nueva entrada</h3>
<form action="?" method="POST">
    <input type="text" name="titulo" placeholder="titulo">
    <input type="text" name="autor" placeholder="autor">
    <input type="date" name="fecha" placeholder="fecha">
    <input type="text" name="contenido"
placeholder="contenido">
    <input type="text" name="imagen" placeholder="imagen">
    <input type="submit">
</form>
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
nav ul li{
    display:inline-block;
}
nav ul li a{
    text-decoration: none;
    color:inherit;
    text-transform:uppercase;
```

```
    font-size:10px;
    margin-left:10px;
    margin-right:10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
```

```
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/funciones.php
<?php

    function bucleEntradas(){
        // Creo un array de entradas
        $entradas = [];
        // Cargo el archivo json que contiene datos
        $archivo = file_get_contents("datos/entradas.json");
        $json = json_decode($archivo,true);
        // Convierto los elementos JSON en Objetos de mi
aplicacion
        foreach($json as $entrada){
            array_push(
                $entradas,
                new Entrada(
                    $entrada["titulo"],
                    $entrada["contenido"],
                    $entrada["fecha"],
                    $entrada["autor"],
                    $entrada["imagen"]
                )
            );
        }
        // Recorro los objetos y los pinto en pantalla
       for($i = 0;$i<count($entradas);$i++){
            echo $entradas[$i]->damePublicacion();
        }
    }

?>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:

4.17. Proyecto-Cargar una página

El archivo `index.php` es el punto de entrada principal del proyecto y combina todas las piezas para generar la página web completa. Incluye la cabecera (`inc/cabecera.php`), carga las clases necesarias y lee los datos de las entradas desde un archivo JSON.

Luego, crea un array de objetos `Entrada` a partir de los datos JSON y recorre ese array para mostrar cada publicación utilizando el método

`damePublicacion()`. Finalmente, incluye el pie de página (`inc/piedepagina.php`), cerrando así la estructura HTML y completando la página.

Esta organización modular permite separar la lógica, el contenido y la presentación, facilitando el mantenimiento y la escalabilidad del proyecto.

Estructura del proyecto:

```
017-Proyecto-Cargar una pagina
├── admin
│   └── index.php
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   ├── entradas2.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   ├── funciones.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

En este decimoséptimo subcapítulo, titulado **Proyecto - Cargar una página**, se completa y refina el sistema de navegación individual iniciado en el apartado anterior, optimizando el proceso para **mostrar el contenido de una única página estática** desde un archivo

independiente, con un diseño estructurado y preparado para enlaces navegables. Aunque el objetivo es similar al de *Cargar páginas*, en esta fase se enfoca en **acceder a una página concreta desde enlaces del menú u otras secciones**, consolidando así el comportamiento de una web multisección real.

4.17.1. Estructura del proyecto

No se crean nuevos archivos en esta fase; se trabaja principalmente sobre el archivo `pagina.php`, que ya fue introducido en la fase anterior.

```
017-Proyecto-Cargar una pagina
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
├── panel.php
├── procesar.php
└── pagina.php
```

4.17.2. Qué se está haciendo

Se mejora la funcionalidad de `pagina.php` para que sea **completamente operativa y lista para integrarse con enlaces navegables**, es decir, para que el usuario pueda pinchar sobre un enlace como `pagina.php?id=0` y ver directamente esa sección específica (por ejemplo, “Sobre mí”).

Los objetivos principales de esta fase son:

- Consolidar la carga de **una única página estática**.
- Asegurar una presentación **visual, semántica y coherente**.
- Preparar el archivo `pagina.php` para su uso desde menús u otras llamadas internas.

4.17.3. Ejemplo de enlace en el menú

Este paso da sentido a incluir, por ejemplo, en `cabecera.php` algo como:

```
<nav>
    <a href="index.php">Inicio</a>
    <a href="pagina.php?id=0">Sobre mí</a>
    <a href="pagina.php?id=1">Contacto</a>
</nav>
```

Así, el usuario puede navegar entre secciones reales del sitio.

4.17.4. Código de pagina.php (refinado)

La lógica general sigue siendo la misma que en el subcapítulo anterior, pero se presenta ahora de forma robusta, lista para el entorno de producción:

```
<?php
include "inc/cabecera.php";
include "clases/Publicacion.php";

$id = isset($_GET["id"]) ? (int)$_GET["id"] : -1;
$datos = json_decode(file_get_contents("datos/paginas.json"),
true);

if (isset($datos[$id])) {
    $elemento = $datos[$id];
    $pagina = new Pagina($elemento["titulo"],
$elemento["contenido"]);
    $pagina->fecha = $elemento["fecha"];
    $pagina->autor = $elemento["autor"];
    $pagina->imagen = $elemento["imagen"];

    echo $pagina->damePublicacion();
} else {
    echo "<h2>Página no encontrada</h2>";
    echo "<p>La página solicitada no existe o el parámetro es
incorrecto.</p>";
}

include "inc/piedepagina.php";
?>
```

4.17.5. Mejoras aplicadas

- Se fuerza el valor de `id` a entero para evitar errores de tipo.
- Se gestiona el caso en que el parámetro no existe o apunta a un índice no válido (`else` con mensaje amigable).
- La visualización sigue utilizando `damePublicacion()` para mantener el estilo visual uniforme.

4.17.6. Justificación de los cambios

- **Navegación semántica**: el visitante puede visitar páginas individuales como secciones del sitio.
- **Experiencia de usuario**: se muestra un mensaje claro si la página no se encuentra.
- **Preparación para menú dinámico**: cada página ahora puede integrarse con enlaces navegables.
- **Separación de vistas**: se distingue el tratamiento de entradas (en `index.php`) del de páginas (en `pagina.php`).

4.17.7. Consideraciones futuras

- **Sistema de rutas amigables**: para mejorar la URL (`pagina.php?id=1` → `/sobre-mi`).
- **Identificadores únicos por slug**: usando `"slug": "sobre-mi"` en lugar de usar un índice.
- **Encabezado dinámico**: personalizar el `<title>` de la página en función del contenido cargado.
- **Autogeneración del menú**: leer directamente de `paginas.json` para construir la navegación.

4.17.8. Conclusión

Este paso finaliza la integración real de las páginas estáticas como secciones navegables del sitio web. El usuario ya puede interactuar con el sitio como si fuese una web corporativa o personal, con menús que enlazan a páginas independientes. Se afianza así la transición del

proyecto desde un simple blog hacia un sistema web completo, gestionado desde archivos JSON y renderizado dinámicamente mediante clases PHP.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    include "inc/funciones.php";
    if(!isset($_GET['p'])){
        bucleEntradas();
    }else{
        // Creo un array de entradas
        $paginas = [];
        // Cargo el archivo json que contiene datos
        $archivo = file_get_contents("datos/paginas.json");
        $json = json_decode($archivo,true);
        // Convierto los elementos JSON en Objetos de mi
aplicacion
        foreach($json as $pagina){

            if($pagina["titulo"] == $_GET['p']){
                array_push(
                    $paginas,
                    new Pagina(
                        $pagina["titulo"],
                        $pagina["contenido"],
                        $pagina["fecha"],
                        $pagina["autor"],
                        $pagina["imagen"]
                    )
                );
            }
        }
        // Recorro los objetos y los pinto en pantalla
       for($i = 0;$i<count($paginas);$i++){
            echo $paginas[$i]->damePagina();
        }
    }


    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Quinta entrada del blog",
        "autor": "Jose Vicente",
        "fecha": "2023-11-15",
        "contenido": "Esto es una prueba",
        "imagen": "josevicente.jpg"
    },
```

```
    {
        "titulo": "Sexta entrada del blog",
        "autor": "JOse Vicente",
        "fecha": "2023-11-13",
        "contenido": "Este es un contenido",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/entradas2.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
admin/index.php
<?php
    if(isset($_POST['titulo'])){
        $archivo =
file_get_contents("../datos/entradas.json");
        $json = json_decode($archivo);
        array_push($json,$_POST);
        $nuevojson = json_encode($json, JSON_PRETTY_PRINT);

file_put_contents("../datos/entradas.json",$nuevojson);
    }
?>
<h3>Nueva entrada</h3>
<form action="?" method="POST">
    <input type="text" name="titulo" placeholder="titulo">
    <input type="text" name="autor" placeholder="autor">
    <input type="date" name="fecha" placeholder="fecha">
    <input type="text" name="contenido"
placeholder="contenido">
    <input type="text" name="imagen" placeholder="imagen">
    <input type="submit">
</form>
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
nav ul li{
    display:inline-block;
}
nav ul li a{
    text-decoration: none;
    color:inherit;
    text-transform:uppercase;
```

```
    font-size:10px;
    margin-left:10px;
    margin-right:10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
        function damePagina(){
             return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
```

```
                </article>
                ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/funciones.php
<?php

    function bucleEntradas(){
        // Creo un array de entradas
        $entradas = [];
        // Cargo el archivo json que contiene datos
        $archivo = file_get_contents("datos/entradas.json");
        $json = json_decode($archivo,true);
        // Convierto los elementos JSON en Objetos de mi
aplicacion
        foreach($json as $entrada){
            array_push(
                $entradas,
                new Entrada(
                    $entrada["titulo"],
                    $entrada["contenido"],
                    $entrada["fecha"],
                    $entrada["autor"],
                    $entrada["imagen"]
                )
            );
        }
        // Recorro los objetos y los pinto en pantalla
       for($i = 0;$i<count($entradas);$i++){
            echo $entradas[$i]->damePublicacion();
        }
    }


?>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

Resultado:

4.18. Proyecto-Refactorizar

El archivo `arrayobjetos.php` contiene código PHP que inserta manualmente objetos de la clase `Entrada` en un array llamado `$entradas`. Cada objeto se crea con un título y contenido específicos, y luego se agrega al array mediante `array_push()`.

Este archivo sirve como una fuente estática de datos para alimentar el blog, permitiendo probar la presentación y funcionamiento de las entradas sin necesidad de utilizar una base de datos o archivos externos.

Separar los datos en un archivo independiente facilita la gestión y actualización del contenido, además de mantener el código principal más limpio y organizado.

Estructura del proyecto:

```
018-Proyecto-Refactorizar
├── admin
│   └── index.php
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── arrayobjetos.php
│   ├── entradas.json
│   ├── entradas2.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   ├── funciones.php
│   └── piedepagina.php
├── index.php
└── photo
    ├── josevicente.jfif
    └── josevicente.jpg
```

.En este decimoctavo subcapítulo, titulado **Proyecto - Refactorizar**, se realiza una mejora estructural clave: se reorganiza el código para eliminar duplicaciones, centralizar funcionalidades repetidas y preparar el sistema para futuras expansiones, todo **sin alterar su comportamiento visible**.

Este paso se centra en **refactorizar el manejo de páginas estáticas**, de forma análoga a la refactorización previa hecha con las entradas del blog.

4.18.1. Estructura del proyecto

La estructura externa no cambia, pero internamente se introduce una **función específica para cargar páginas** desde el archivo `paginas.json`. Esta función se puede ubicar en un archivo de utilidades como `funciones.php`, junto a la ya existente `cargarEntradasDesdeJSON()`.

```
018-Proyecto-Refactorizar
├── clases
│   └── Publicacion.php
├── css
│   └── estilo.css
├── datos
│   ├── entradas.json
│   └── paginas.json
├── inc
│   ├── cabecera.php
│   └── piedepagina.php
├── index.php
├── panel.php
├── procesar.php
├── pagina.php
└── funciones.php
```

4.18.2. Qué se está haciendo

Hasta ahora, cada archivo que necesitaba cargar páginas (`pagina.php`, o incluso `index.php` si listaba páginas) contenía su propia lógica para:

1. Leer `paginas.json`.

2. Extraer un elemento por índice.

3. Crear un objeto `Pagina` manualmente.

Esa lógica se traslada ahora a una **función reutilizable** llamada, por ejemplo, `cargarPaginasDesdeJSON()`, o `cargarPaginaPorId($id)`, para evitar duplicación de código y mejorar la organización.

4.18.3. Código para funciones.php

Se incluye esta función nueva:

```
function cargarPaginasDesdeJSON($ruta){
    $paginas = [];
    $datos = json_decode(file_get_contents($ruta), true);
    foreach($datos as $elemento){
        $pagina = new Pagina($elemento["titulo"],
$elemento["contenido"]);
        $pagina->fecha = $elemento["fecha"];
        $pagina->autor = $elemento["autor"];
        $pagina->imagen = $elemento["imagen"];
        array_push($paginas, $pagina);
    }
    return $paginas;
}

function cargarPaginaPorId($ruta, $id){
    $datos = json_decode(file_get_contents($ruta), true);
    if(isset($datos[$id])){
        $e = $datos[$id];
        $pagina = new Pagina($e["titulo"], $e["contenido"]);
        $pagina->fecha = $e["fecha"];
        $pagina->autor = $e["autor"];
        $pagina->imagen = $e["imagen"];
        return $pagina;
    } else {
        return null;
    }
}
```

4.18.4. Aplicación en pagina.php

La lógica de carga se simplifica notablemente:

```
<?php
include "inc/cabecera.php";
include "clases/Publicacion.php";
include "funciones.php";

$id = isset($_GET["id"]) ? (int)$_GET["id"] : -1;
$pagina = cargarPaginaPorId("datos/paginas.json", $id);

if ($pagina) {
    echo $pagina->damePublicacion();
} else {
    echo "<h2>Página no encontrada</h2>";
}

include "inc/piedepagina.php";
?>
```

4.18.5. Ventajas de esta refactorización

- **Código limpio y reutilizable**: las funciones se pueden invocar desde cualquier parte del proyecto.
- **Mantenimiento centralizado**: si cambia el formato del JSON o se añaden nuevos campos, solo se modifica la función.
- **Menor probabilidad de errores**: se evita repetir estructuras largas y propensas a inconsistencias.
- **Mayor claridad conceptual**: cada archivo hace lo que debe: mostrar contenido, no reconstruir lógica de datos.

4.18.6. Justificación de los cambios

- **Principio DRY**: “Don’t Repeat Yourself” — se elimina duplicación de lógica en todos los scripts.
- **Preparación para crecimiento**: ahora se pueden añadir nuevas vistas o rutas que utilicen las mismas funciones reutilizables.
- **Modularidad real**: el sistema se aproxima a un patrón MVC básico (Modelo, Vista, Controlador), aunque de forma informal.

4.18.7. Conclusión

Con esta última refactorización, el proyecto alcanza una madurez estructural. Ya no solo funciona correctamente desde el punto de vista visual, sino que además está **organizado de forma profesional, modular y escalable**. Esta base robusta permitirá implementar nuevas características sin comprometer la estabilidad del código existente, y facilitará la colaboración, la depuración y el mantenimiento a largo plazo.

```
index.php
<?php
    include "clases/Publicacion.php";
   include "inc/cabecera.php";
    include "inc/funciones.php";
    if(!isset($_GET['p'])){
        bucleEntradas();
    }else{
        cargaPaginaActual();
    }
    include "inc/piedepagina.php";
?>
```

.

```
datos/arrayobjetos.php
<?php
    array_push(
        $entradas,
        new Entrada(
            "Primera entrada",
            "Contenido de la primera entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Segunda entrada",
            "Contenido de la segunda entrada"
        )
    );
    array_push(
        $entradas,
        new Entrada(
            "Tercera entrada",
            "Contenido de la tercera entrada"
        )
    );
?>
```

.

```
datos/entradas.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Quinta entrada del blog",
        "autor": "Jose Vicente",
        "fecha": "2023-11-15",
        "contenido": "Esto es una prueba",
        "imagen": "josevicente.jpg"
    },
```

```
    {
        "titulo": "Sexta entrada del blog",
        "autor": "JOse Vicente",
        "fecha": "2023-11-13",
        "contenido": "Este es un contenido",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/entradas2.json
[
    {
        "titulo": "Primera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la primera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Segunda entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la segunda
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Tercera entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la tercera
entrada",
        "imagen": "josevicente.jpg"
    },
    {
        "titulo": "Cuarta entrada del blog",
        "autor": "Jose Vicente Carratala",
        "fecha": "2023-11-30",
        "contenido": "Este es el contenido de la cuarta
entrada",
        "imagen": "josevicente.jpg"
    }
]
```

.

```
datos/paginas.json
[
    {
        "titulo":"Proyectos",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de
proyectos",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Sobre mi",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagina te voy a hablar de sobre
mi",
        "imagen":"josevicente.jpg"
    },
    {
        "titulo":"Contacto",
        "autor":"Jose Vicente Carratala",
        "fecha":"2023-11-30",
        "contenido":"En esta pagnia encontramos el menu de
navegacion",
        "imagen":"josevicente.jpg"
    }
]
```

.

```
admin/index.php
<?php
    if(isset($_POST['titulo'])){
        $archivo =
file_get_contents("../datos/entradas.json");
        $json = json_decode($archivo);
        array_push($json,$_POST);
        $nuevojson = json_encode($json, JSON_PRETTY_PRINT);

file_put_contents("../datos/entradas.json",$nuevojson);
    }
?>
<h3>Nueva entrada</h3>
<form action="?" method="POST">
    <input type="text" name="titulo" placeholder="titulo">
    <input type="text" name="autor" placeholder="autor">
    <input type="date" name="fecha" placeholder="fecha">
    <input type="text" name="contenido"
placeholder="contenido">
    <input type="text" name="imagen" placeholder="imagen">
    <input type="submit">
</form>
```

.

```
css/estilo.css
body,html{
    background:rgb(200,200,200);
    font-family:sans-serif;
}
header,main,footer{
    background:white;
    width:800px;
    margin:auto;
    padding:20px;
}
main{
    box-shadow:0px 0px 20px rgba(0,0,0,0.5);
}
header,footer{
    position:relative;
    z-index:1000;
    text-align:center;
}
header{
    border-radius:10px 10px 0px 0px;
}
footer{
    border-radius:0px 0px 10px 10px;
}
article{
    display:inline-block;
    width:32%;
    box-sizing: border-box;
    padding:5px;
}
article img{
    width:100%;
}
nav ul li{
    display:inline-block;
}
nav ul li a{
    text-decoration: none;
    color:inherit;
    text-transform:uppercase;
```

```
    font-size:10px;
    margin-left:10px;
    margin-right:10px;
}
```

.

```
clases/Publicacion.php
<?php
    abstract class Publicacion{
        protected $titulo;

        function
__construct($nuevotitulo,$nuevocontenido,$nuevafecha,$nuevoau
tor,$nuevaimagen){
            $this->titulo = $nuevotitulo;
            $this->fecha = $nuevafecha;
            $this->autor = $nuevoautor;
            $this->contenido = $nuevocontenido;
            $this->imagen = $nuevaimagen;
        }
        function setTitulo($nuevotitulo){
            if(strlen($nuevotitulo) > 10){
                $this->titulo = $nuevotitulo;
            }else{
                $this->titulo = "error";
            }
        }
        function getTitulo(){
            return $this->titulo;
        }

    }
    class Pagina extends Publicacion{
        function dameMenu(){
            return '
                <li><a
href="?p='.$this->titulo.'">'.$this->titulo.'</a></li>
            ';
        }
        function damePagina(){
             return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
```

```
                </article>
                ';
        }
    }
    class Entrada extends Publicacion{
        function damePublicacion(){
            return '
                <article>
                    <img src="photo/'.$this->imagen.'">
                    <h3>'.$this->titulo.'</h3>
                    <time>'.$this->fecha.'</time>
                    <p>'.$this->autor.'</p>
                    <p>'.$this->contenido.'</p>
                </article>
                ';
        }
    }

?>
```

.

```
inc/cabecera.php
<!doctype html>
<html>
    <head>
        <title>El blog de Jose Vicente</title>
        <link rel="stylesheet" href="css/estilo.css">
    </head>
    <body>
        <header>
            <h1>El blog de Jose Vicente</h1>
            <nav>
                <ul>
                    <?php
                        $paginas = [];
                        // Cargo el archivo json que contiene
datos
                        $archivo =
file_get_contents("datos/paginas.json");
                        $json = json_decode($archivo,true);
                        // Convierto los elementos JSON en
Objetos de mi aplicacion
                        foreach($json as $pagina){
                            array_push(
                                $paginas,
                                new Pagina(
                                    $pagina["titulo"],
                                    $pagina["contenido"],
                                    $pagina["fecha"],
                                    $pagina["autor"],
                                    $pagina["imagen"]
                                )
                            );
                        }
                        // Recorro los objetos y los pinto en
pantalla
                       for($i = 0;$i<count($paginas);$i++){
                            echo $paginas[$i]->dameMenu();
                        }
                    ?>
                </ul>
            </nav>
```

```
        </header>
        <main>
```

.

```
inc/funciones.php
<?php

    function bucleEntradas(){
        // Creo un array de entradas
        $entradas = [];
        // Cargo el archivo json que contiene datos
        $archivo = file_get_contents("datos/entradas.json");
        $json = json_decode($archivo,true);
        // Convierto los elementos JSON en Objetos de mi
aplicacion
        foreach($json as $entrada){
            array_push(
                $entradas,
                new Entrada(
                    $entrada["titulo"],
                    $entrada["contenido"],
                    $entrada["fecha"],
                    $entrada["autor"],
                    $entrada["imagen"]
                )
            );
        }
        // Recorro los objetos y los pinto en pantalla
       for($i = 0;$i<count($entradas);$i++){
            echo $entradas[$i]->damePublicacion();
        }
    }
function cargaPaginaActual(){
    // Creo un array de entradas
        $paginas = [];
        // Cargo el archivo json que contiene datos
        $archivo = file_get_contents("datos/paginas.json");
        $json = json_decode($archivo,true);
        // Convierto los elementos JSON en Objetos de mi
aplicacion
        foreach($json as $pagina){

            if($pagina["titulo"] == $_GET['p']){
                array_push(
                    $paginas,
```

```
                    new Pagina(
                        $pagina["titulo"],
                        $pagina["contenido"],
                        $pagina["fecha"],
                        $pagina["autor"],
                        $pagina["imagen"]
                    )
                );
            }
        }
        // Recorro los objetos y los pinto en pantalla
       for($i = 0;$i<count($paginas);$i++){
            echo $paginas[$i]->damePagina();
        }
}
?>
```

.

```
inc/piedepagina.php
        </main>
        <footer>
            <h6>(c) 2023 Jose Vicente Carratalá</h6>
        </footer>
    </body>
</html>
```

5. Epílogo

Has llegado al final de este recorrido por la programación en PHP 8, y espero que ahora sientas la confianza suficiente para abordar tus propios proyectos web. A lo largo de este libro, hemos caminado desde los conceptos más básicos hasta la realización de aplicaciones funcionales, repasando tanto la teoría esencial como los desafíos prácticos que surgen en el día a día de cualquier desarrollador.

La programación no es solo una cuestión de escribir líneas de código; es una herramienta para crear, para resolver problemas y para dar vida a ideas que pueden llegar a miles de personas. PHP sigue siendo una tecnología vigente y en constante evolución, presente en gran parte de la web, y aprender a manejarla es abrir la puerta a un sinfín de oportunidades.

No olvides que la mejor manera de afianzar lo aprendido es practicar. Experimenta, equivócate, busca nuevas formas de resolver los retos que encuentres y mantente siempre curioso. Cada nuevo proyecto será una oportunidad para crecer y descubrir algo nuevo.

Gracias por confiar en este libro como tu guía. Espero que continúes avanzando en tu camino como desarrollador, y que PHP te ayude a hacer realidad tus propios proyectos y metas. Recuerda que el aprendizaje nunca termina: la tecnología cambia, los lenguajes evolucionan, y cada día trae consigo nuevos desafíos y posibilidades.

¡Sigue programando!