INFORMÁTICA Y COMUNICACIONES
G.S. Desarrollo de Aplicaciones Web

Desarrollo web en entorno servidor

Xabier Ganzábal García

EDITORIAL SINTESIS

Escaneado con CamScanner

Desarrollo web en entorno servidor

Material complementario

En **www.sintesis.com** puede descargar el archivo libro_servidor.rar, que incluye los recursos necesarios para trabajar los ejemplos que aparecen en el libro.

Puede descargarlo a través del siguiente código: 1827-DCRJ-7666

1. Entre en **www.sintesis.com** y regístrese como usuario.
2. Seleccione "Complementos web" en el menú superior e introduzca el código de acceso.

Escaneado con CamScanner

Consulte nuestra página web: **www.sintesis.com**
En ella encontrará el catálogo completo y comentado



Queda prohibida, salvo excepción prevista en la ley, cualquier forma de reproducción, distribución, comunicación pública y transformación de esta obra sin contar con autorización de los titulares de la propiedad intelectual. La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (arts. 270 y sigs. Código Penal). El Centro Español de Derechos Reprográficos (www.cedro.org) vela por el respeto de los citados derechos.

Escaneado con CamScanner

Desarrollo web en entorno servidor

Xabier Ganzábal García

EDITORIAL SINTESIS

Escaneado con CamScanner

Asesor editorial:

Juan Carlos Moreno Pérez

© Xabier Ganzábal García

© EDITORIAL SÍNTESIS, S. A.
Vallehermoso, 34. 28015 Madrid
Teléfono 91 593 20 98
http://www.sintesis.com

ISBN: 978-84-9171-349-4
Depósito Legal: M-11.684-2019

Impreso en España - Printed in Spain

Reservados todos los derechos. Está prohibido, bajo las sanciones penales y el resarcimiento civil previstos en las leyes, reproducir, registrar o transmitir esta publicación, íntegra o parcialmente, por cualquier sistema de recuperación y por cualquier medio, sea mecánico, electrónico, magnético, electroóptico, por fotocopia o por cualquier otro, sin la autorización previa por escrito de Editorial Síntesis, S. A.

Escaneado con CamScanner

Índice

PRESENTACIÓN 11

1. INTRODUCCIÓN AL DESARROLLO DE APLICACIONES WEB 13

Objetivos 13
Mapa conceptual 14
Glosario 14
1.1. Modelos de programación en entornos cliente-servidor 15
1.2. Generación dinámica de páginas web 16
1.2.1. Lenguajes de programación en entornos servidor 17
1.2.2. Integración con los lenguajes de marcas 17
1.2.3. Servidores de aplicaciones 18
1.3. Instalación del entorno de trabajo 18
1.3.1. Servidores web y de base de datos 19
1.3.2. Ejemplos y bases de datos 20
1.3.3. Entorno de desarrollo 20
Resumen 21
Ejercicios propuestos 21
Actividades de autoevaluación 22

2. INTRODUCCIÓN AL LENGUAJE PHP 25

Objetivos 25
Mapa conceptual 26
Glosario 26

Escaneado con CamScanner

2.1. PHP y HTML. Código incrustado 2
2.2. Sintaxis de PHP 2
2.3. Variables y tipos de dato 2
2.3.1. Declaración de variables 2
2.3.2. Asignación por copia y por referencia 2
2.3.3. Variables no inicializadas 2
2.3.4. Constantes 3
2.3.5. Tipos de datos escalares 3
2.3.6. Ámbito de las variables 3
2.3.7. Variables predefinidas 3
2.4. Comentarios 3
2.5. Estructuras de control 3
2.5.1. Estructuras condicionales 3
2.5.2. Estructuras de repetición 3
2.5.3. Otras estructuras de control 4
2.6. Operadores 4
2.7. *Arrays* 4
2.8. Funciones y librerías 4
2.8.1. Funciones predefinidas 4
2.8.2. Funciones definidas por el usuario 5
2.8.3. Paso de argumentos por copia y por valor 5
2.8.4. Funciones como argumentos 5
2.9. Excepciones y errores 5
2.9.1. Errores 5
2.9.2. Excepciones 5
2.9.3. Excepciones Error 5
2.10. Clases y objetos 5
Resumen 5
Ejercicios propuestos 5
Actividades de autoevaluación 5

3. DESARROLLO DE APLICACIONES WEB CON PHP 6

Objetivos 6
Mapa conceptual 6
Glosario 6
3.1. Paso de parámetros 6
3.2. Formularios 6
3.2.1. Formulario de *login* 6
3.2.2. Formulario y procesamiento en un solo fichero 6
3.2.3. Subida de ficheros 6
3.3. *Cookies* 7
3.4. Sesiones. Seguridad: usuarios y roles 7
3.5. Envío de correo electrónico 7
3.6. Bases de datos relacionales 7
3.6.1. Conexión a la base de datos 7
3.6.2. Recuperación y presentación de datos 8
3.6.3. Inserción, borrado y actualización 8
3.6.4. Transacciones 8
3.7. Bases de datos no relacionales 8
3.7.1. Instalación y puesta marcha de MongoDB 8
3.7.2. Conexión desde PHP 8

Escaneado con CamScanner

3.8. Ficheros 86
3.8.1. Ficheros XML 89
3.9. Pruebas 92
3.10. Depuración de errores 94
Resumen 95
Ejercicios propuestos 95
Actividades de autoevaluación 96

4. EJEMPLO DE APLICACIÓN COMPLETA EN PHP 99

Objetivos 99
Mapa conceptual 100
Glosario 100
4.1. Definición del proyecto 100
4.2. Análisis de requisitos 102
4.2.1. Esquema entidad-relación 102
4.2.2. Limitaciones de la aplicación 103
4.3. Diseño de la aplicación 103
4.3.1. Diseño lógico de la base de datos 103
4.3.2. Diseño físico de la base de datos 104
4.3.3. Diagrama de flujo de pantallas 105
4.3.4. El carrito de la compra 106
4.3.5. Control de acceso 107
4.3.6. Ficheros de la aplicación 108
4.4. Implementación 109
4.4.1. *Login* 109
4.4.2. Control de acceso 110
4.4.3. La cabecera 110
4.4.4. Lista de categorías 111
4.4.5. Tabla de productos 112
4.4.6. Añadir productos 113
4.4.7. El carrito de la compra 114
4.4.8. Eliminar productos 115
4.4.9. Procesamiento del pedido 116
4.4.10. La base de datos 117
4.4.11. Envío de correos 121
Resumen 122
Ejercicios propuestos 123
Actividades de autoevaluación 123

5. APLICACIONES WEB DINÁMICAS CON AJAX 125

Objetivos 125
Mapa conceptual 126
Glosario 126
5.1. Separación de la lógica de negocio 126
5.2. Tecnologías y librerías asociadas 127
5.3. Obtención remota de información 128
5.3.1. Peticiones síncronas y asíncronas 129
5.4. Respuesta del servidor 131

Escaneado con CamScanner

5.5. Modificación de la estructura de la página web 132
5.6. Captura de eventos 134
5.7. Aplicaciones de una sola página 135
Resumen 135
Ejercicios propuestos 135
Actividades de autoevaluación 136

6. APLICACIÓN DE PEDIDOS CON AJAX 139

Objetivos 139
Mapa conceptual 140
Glosario 140
6.1. Diseño de la aplicación 140
6.2. Estructura de la página web 141
6.3. Cambios en la estructura 142
6.4. En el servidor 143
6.5. Implementación 144
6.6. Lado del servidor 145
6.6.1. *Login* 145
6.6.2. Control de acceso 146
6.6.3. La cabecera 146
6.6.4. Las categorías 147
6.6.5. Los productos 147
6.6.6. El carrito de la compra 147
6.6.7. Añadir y eliminar productos 148
6.6.8. Cerrar la sesión 148
6.6.9. Procesar el pedido 149
6.6.10. Funciones auxiliares 149
6.7. El lado del cliente 149
6.7.1. *Login* 149
6.7.2. Las categorías 150
6.7.3. Los productos 151
6.7.4. El carrito 153
6.7.5. Añadir y eliminar 154
6.7.6. Realizar el pedido 155
Resumen 156
Ejercicios propuestos 156
Actividades de autoevaluación 156

7. MAPEO OBJETO-RELACIONAL (ORM) 159

Objetivos 159
Mapa conceptual 160
Glosario 160
7.1. Mapeo objeto-relacional 160
7.2. Doctrine 161
7.2.1. Instalación y configuración 161
7.2.2. Entidades 162
7.2.3. Inserción y borrado 165
7.3. Asociaciones 166
7.3.1. Asociaciones muchos a uno unidireccionales 167
7.3.2. Asociaciones muchos a uno bidireccionales 169

Escaneado con CamScanner

7.4. Consultas básicas 170
7.5. DQL 171
7.6. Repositorios propios 173
Resumen 174
Ejercicios propuestos 174
Actividades de autoevaluación 175

8. DESARROLLO DE APLICACIONES EN SYMFONY 177

Objetivos 177
Mapa conceptual 178
Glosario 178
8.1. El patrón MVC 178
8.2. Symfony 179
8.2.1. Visión general 179
8.2.2. Instalación 180
8.2.3. Estructura de directorios 181
8.3. Controladores 182
8.4. Rutas 182
8.4.1. Paso de parámetros 182
8.4.2. Valores por defecto 183
8.4.3. Redirección 184
8.4.4. Rutas a nivel de clase 184
8.4.5. Rutas disponibles 185
8.5. Plantillas 185
8.5.1. Introducción a Twig 185
8.5.2. Rutas en plantillas 188
8.5.3. Inclusión y herencia 188
8.6. Servicios 189
8.7. Bases de datos 190
8.8. Formularios 191
8.9. Envío de correo 192
8.10. Seguridad. Usuarios y roles 193
8.10.1. Control de acceso 197
8.10.2. Abrir sesión 197
8.10.3. Cerrar sesión 199
Resumen 199
Ejercicios propuestos 200
Actividades de autoevaluación 200

9. APLICACIÓN DE PEDIDOS EN SYMFONY 203

Objetivos 203
Mapa conceptual 204
Glosario 204
9.1. Diseño de la aplicación 204
9.1.1. Plantillas 205
9.1.2. Entidades 205
9.1.3. Rutas de la aplicación 206
9.2. Implementación 207

Escaneado con CamScanner

9.3. Plantillas
9.3.1. *Login*
9.3.2. Plantilla base
9.3.3. Cabecera
9.3.4. Lista de categorías
9.3.5. Tabla de productos
9.3.6. El carrito de la compra
9.3.7. Confirmación del pedido
9.3.8. Correo
9.4. Entidades
9.5. Controladores
9.5.1. Abrir sesión
9.5.2. Lista de categorías
9.5.3. Tabla de productos
9.5.4. Carrito
9.5.5. Añadir y eliminar
9.5.6. Realizar pedido
9.6. Seguridad
Resumen
Ejercicios propuestos
Actividades de autoevaluación

10. SERVICIOS WEB Y APLICACIONES HÍBRIDAS

Objetivos
Mapa conceptual
Glosario
10.1. Arquitecturas de programación orientadas a servicios
10.2. Protocolos y lenguajes implicados. SOAP
10.2.1. SOAP
10.2.2. Descripción de servicios web. WSDL
10.3. Librerías de PHP para servicios web
10.3.1. Generación de servicios web
10.3.2. Utilización de servicios web
10.4. Aplicaciones híbridas
10.4.1. Interfaces de programación y repositorios
Ejercicios propuestos
Resumen
Actividades de autoevaluación

WEBGRAFÍA 247

Escaneado con CamScanner

Presentación

Este libro trata sobre desarrollo de aplicaciones web. Está pensado como libro de texto para el módulo *Desarrollo web en entorno servidor*, del ciclo formativo de grado superior Desarrollo de Aplicaciones Web, pero puede ser útil para cualquier persona interesada en aprender PHP o Symfony. Se asume que el lector tiene una base de programación y HTML. Además, para aprovechar todo el contenido, hay que conocer el modelo relacional de bases de datos.

El libro sigue un enfoque totalmente práctico. Los contenidos se presentan con ejemplos, que el alumno puede descargar para probarlos mientras trabaja con el libro. Al final de cada capítulo se proponen más ejercicios para practicar.

Se pone especial énfasis en el desarrollo de aplicaciones completas. A lo largo del libro, se desarrollan tres aplicaciones que incorporan los contenidos de los capítulos anteriores. En el capítulo 4, se implementa una aplicación de pedidos utilizando solo PHP. En el capítulo 6, se rehace utilizando AJAX. Y después, en el 9, se desarrolla en Symfony.

Symfony es uno de los temas más interesantes del libro. Es un *framework* para desarrollo web en PHP muy extendido. Symfony plantea las aplicaciones web de una manera determinada que, una vez dominada, simplifica y acelera el desarrollo.

Estructura de los contenidos

- El capítulo 1 sirve como introducción al desarrollo de aplicaciones web. Se presentan las arquitecturas básicas, los protocolos implicados y los lenguajes más extendidos. También se instala el entorno de trabajo para probar los ejemplos del libro.
- En el capítulo 2, se explican los elementos básicos de PHP. Sintaxis, tipos de datos, funciones y objetos.
- En el capítulo 3, se muestra cómo utilizar PHP para las tareas más habituales en una aplicación web: paso de parámetros, formularios, sesiones y *cookies*. También se ve cómo manejar bases de datos y ficheros.

Escaneado con CamScanner

- En el capítulo 4, se desarrolla una aplicación completa, incluyendo análisis y diseño. Es uno de los temas más interesantes porque explica cómo diseñar una aplicación web que maneja una base de datos. Esta aplicación se rehace en los capítulos 6 y 9.
- En el capítulo 5, se presenta AJAX. Con esta técnica de diseño de aplicaciones web se busca desacoplar la lógica de negocio de la de presentación y mejorar la experiencia de usuario.
- En el capítulo 6, se rehace la aplicación de pedidos utilizando AJAX. Se podrán apreciar las ventajas que conlleva.
- El capítulo 7 se ocupa del mapeo objeto-relacional, ORM, mediante el que se establece una asociación entre la base de datos y los objetos de la aplicación.
- El capítulo 8 introduce Symfony. Es un *framework* para desarrollo web en PHP que implementa el patrón MVC. Incluye componentes para muchas de las tareas habituales, que se pueden reutilizar en proyectos que no usen Symfony.
- En el capítulo 9, se rehace la aplicación de pedidos. Como se podrá ver, con un buen diseño y conociendo el *framework*, el desarrollo con Symfony es verdaderamente rápido.
- El capítulo 10 presenta la arquitectura orientada a servicios y las opciones para implementarla. También se introducen aplicaciones híbridas o *mashups*, que integran contenidos y funcionalidad de diversos orígenes.

En www.sintesis.com puedes descargar el archivo **libro_servidor.rar**, que incluye los recursos necesarios para trabajar los ejemplos que aparecen en el libro. Puedes descargarlo a través del código y las indicaciones incluidas en la primera página del libro y encontrarás más información sobre ello en el apartado 1.3.2.

Escaneado con CamScanner

1

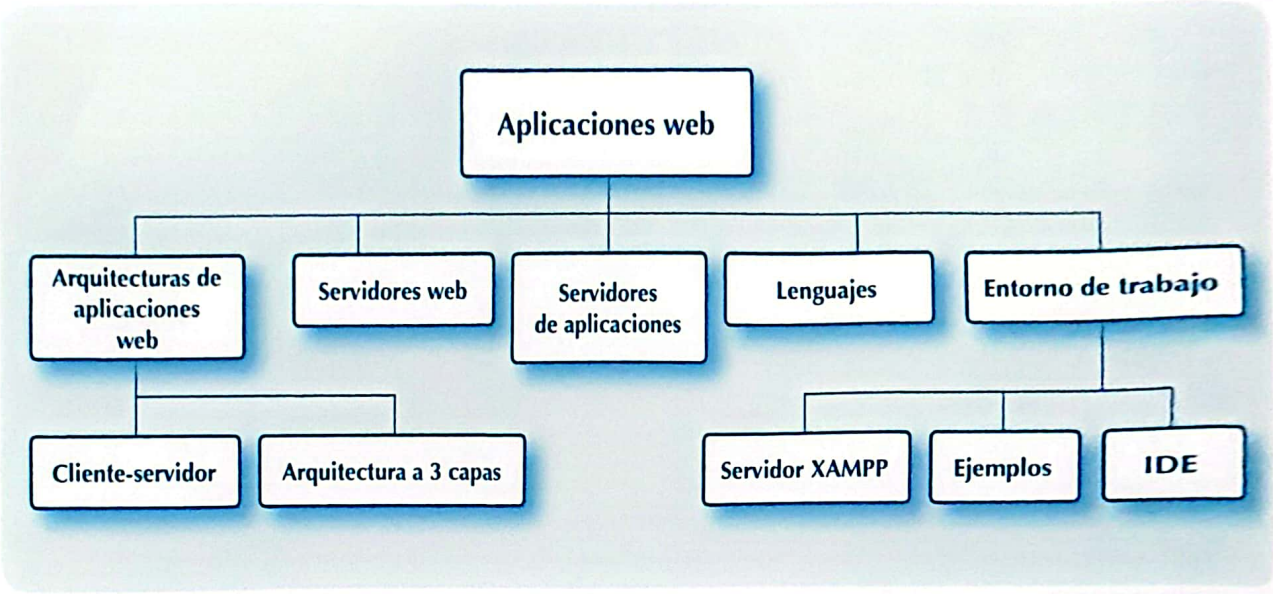
Introducción al desarrollo de aplicaciones web

Objetivos

- ✓ Conocer la arquitectura cliente-servidor.
- ✓ Presentar los modelos de programación en entornos cliente-servidor.
- ✓ Identificar los lenguajes más importantes para el desarrollo de aplicaciones web.
- ✓ Introducir los principales lenguajes y *frameworks* para el desarrollo en el lado del servidor.
- ✓ Entender cómo se integran los lenguajes de programación con los lenguajes de marcas y los servidores.
- ✓ Instalar el entorno de trabajo para ejecutar los ejemplos del libro.

Escaneado con CamScanner

Mapa conceptual



Glosario

Aplicación web. Aplicación informática a la que se accede mediante una interfaz web utilizando un navegador.

Cliente. En el modelo cliente-servidor, los clientes solicitan funcionalidad a los servidores.

Framework. Un *framework* es una plataforma para el desarrollo de aplicaciones. Puede incluir librerías y metodologías.

HTML. *Hyper Text Markup Language*, el lenguaje básico para la creación de páginas web. Es un estándar del W3C.

HTTP. *Hyper Text Transfer Protocol*, protocolo de transferencia de hipertexto. Es el protocolo que utilizan clientes y servidores web para comunicarse. Es un estándar del W3C.

HTTPS. Versión segura del HTTP.

IDE. *Integrated Development Envoroment*, entorno de desarrollo integrado. Programa que integra herramientas útiles para programar, como editores, compiladores o control de versiones.

Protocolo. Según la RAE, conjunto de reglas que se establecen en el proceso de comunicación entre dos sistemas

Servidor. En el modelo cliente-servidor, los servidores proveen servicios a los clientes.

W3C. *World Wide Web Consortium*, organismo que elabora y mantiene varios de los estándares más importantes en Internet, como el HTTP o el HTML.

Escaneado con CamScanner

1.1. Modelos de programación en entornos cliente-servidor

Las aplicaciones web se basan en el modelo cliente-servidor. Es un modelo distribuido en el que hay dos tipos de elementos con funciones diferenciadas: clientes y servidores. Los servidores proveen servicios, como información o funcionalidad, a los clientes.

Es el cliente el que inicia el proceso al enviar una solicitud al servidor que, a su vez, envía un mensaje de respuesta.

Cliente y servidor se comunican mediante un protocolo que define el formato de solicitudes y respuestas y, en general, se ejecutan en máquinas diferentes conectadas por red, aunque también pueden ejecutarse en el mismo ordenador.

En el caso de los servidores web, los clientes son, la mayoría de las veces, los navegadores que realizan peticiones de páginas al servidor. Este responde con la página web solicitada o con un mensaje de error si no la encuentra o el acceso no está autorizado.



Figura 1.1
Modelo cliente-servidor.

La comunicación entre clientes y servidores web se realiza mediante protocolo HTTP o su versión segura, HTTPS, que tienen reservados los puertos 80 y 553 respectivamente en la lista de puertos bien conocidos. Las características del modelo cliente-servidor y del protocolo HTTP condicionan el desarrollo de aplicaciones web.

La arquitectura a tres capas es una ampliación del modelo cliente-servidor. La lógica de la aplicación se separa en:

a) *Capa de presentación*. Para la interfaz de usuario. Muestra información y permite interactuar con el sistema al usuario.
b) *Capa de negocio*. Para la lógica propia de la aplicación. Se comunica con las otras dos capas.
c) *Capa de datos*. Para gestionar la base de datos.

La capa de presentación no se comunica directamente con la de datos, sino que se hace a través de la capa de negocio. La capa de negocio recibe las acciones del usuario de la capa de presentación y, si es necesario, se comunica con la capa de datos para consultar o modificar la base de datos. De la misma manera, pasa la salida de la capa de datos a la de presentación.

En una aplicación web, la capa de presentación se ejecuta en un navegador, en el ordenador del usuario; la capa de negocio, en el servidor web, y la de datos, en el servidor de bases de datos. Estas dos pueden estar en equipos diferentes o en el mismo.

El objetivo es desacoplar la lógica de negocio de la de presentación para conseguir código más reutilizable. Además, mediante la separación entre componentes se obtienen aplicaciones más fáciles de mantener y ampliar.

Escaneado con CamScanner



Figura 1.2
Modelo a 3 capas.

Por ejemplo, al hacer *login* en una aplicación web:

- Desde el cliente (navegador) se envía un formulario. Es la capa de presentación.
- En servidor, capa de negocio, se consulta con la base datos (capa de datos) si los datos son correctos.
- En función de la respuesta, se pasan las instrucciones correspondientes a la capa de presentación (se permite el acceso o se muestra un mensaje de error).

Actividad propuesta 1.1

Visita la página del W3C (www.w3.org) para ver una lista de sus estándares. A lo largo del libro se utilizarán varios.

Investiga qué es un Estándar Web *(Web Standard)* y qué es una Recomendación *(Recommendation)*.

1.2. Generación dinámica de páginas web

El lenguaje básico para la web es el HTML. Una página que esté escrita usando solo HTML será estática, es decir, mostrará siempre el mismo contenido. Cuando se utiliza un lenguaje de programación en el servidor se pueden generar páginas en función de la solicitud del cliente.

Por ejemplo, muchas páginas muestran anuncios diferentes según dónde esté el usuario. Es decir, no se ofrece el mismo contenido a todos los usuarios, sino que parte del contenido se genera dinámicamente. En este caso se habla de *páginas web dinámicas*.

Para ejecutar código en un servidor web hay varias opciones. Las más habituales son:

1. *Common Gateway Interface (CGI).* Las peticiones de los clientes se pasan a un ejecutable en el servidor. Este programa genera la salida y el servidor se la pasa al cliente.
2. *Como módulo del servidor web.* En lugar de utilizar un programa externo, el propio servidor cuenta con un módulo, generalmente un intérprete, para ejecutar código.
3. *Servlets.* Los *servlets* de Java son objetos que reciben una petición y devuelven una respuesta en función de la petición. Para utilizar *servlets* hace falta un contenedor web, que es el que interactúa con ellos.

Escaneado con CamScanner

1.2.1. Lenguajes de programación en entornos servidor

Hay muchos lenguajes para el lado del servidor. Los más habituales son:

a) PHP. Sin duda, el lenguaje más extendido en el lado del servidor. Es el lenguaje del que trata este libro. Normalmente se ejecuta como un módulo del servidor.
b) JSP. La versión Java de PHP. Para utilizarlo hace falta un contenedor web.
c) ASP.NET. La alternativa a PHP de Microsoft, integrada en la plataforma.NET.
d) PERL. Muy utilizado para CGI. Es un lenguaje especialmente pensado para el procesamiento de expresiones regulares.
e) Ruby. Es un lenguaje orientado a objetos muy apreciado por desarrolladores web.



Actividad propuesta 1.2

Busca información sobre otros lenguajes del lado del servidor y explica para qué se recomiendan.

También hay muchos *frameworks* para desarrollo de aplicaciones web (cuadro 1.1).

CUADRO 1.1
***Frameworks* para desarrollo web**

Framework	Descripción
Spring	El *framework* más extendido para JEE.
Ruby on Rails	Muy popular para el desarrollo MVC, ha influenciado otros muchos *frameworks* extendidos. En Ruby.
Django	Basado en Python, sigue el modelo MVT.
AngularJS	*Framework* de Google para el lado del cliente basado en JavaScript.
Symfony	*Framework* para el desarrollo MVC en PHP.

1.2.2. Integración con los lenguajes de marcas

Las páginas dinámicas se componen de una parte estática en HTML y una parte dinámica en algún lenguaje de programación. Por eso se habla de lenguajes de plantillas.

Cuando se solicita una página, el servidor busca en ella bloques de código. Si los encuentra, los ejecuta y los sustituye por su salida. En PHP los bloques se delimitan por `<?php" y" ?>`.

Por ejemplo, en el fichero hay un bloque PHP en las líneas 7-9:

```
1  <!DOCTYPE html>
2  <html>
```


Escaneado con CamScanner

```
3       <head>
4           <title>Número aleatorio</title>
5       </head>
6       <body>
7           <?php
8               echo rand(0, 100);
9           ?>
10      </body>
11  </html>
```

Al solicitar la página, el servidor ejecuta el bloque y lo sustituye por su salida, en este caso un número aleatorio entre 0 y 100. Si sale un 8, enviará al cliente:

```
<!DOCTYPE html>
<html>
    <head>
        <title> Número aleatorio </title>
    </head>
    <body>
        8
    </body>
</html>
```

Actividad propuesta 1.3

Averigua qué caracteres se utilizan para insertar código JSP y ASP dentro de HTML.

1.2.3. Servidores de aplicaciones

Los servidores de aplicaciones ofrecen un entorno de ejecución para las aplicaciones. Su objetivo es liberar al programador de algunas tareas relacionadas con la infraestructura de la aplicación, como seguridad o balanceo de carga. Normalmente incluyen un servidor web.

Posiblemente, los más conocidos sean los servidores de aplicaciones para JEE, como JBoss o Websphere. Estos servidores tienen contenedores para ejecutar los componentes JEE, como contenedores de *servlets* y Java Entreprise Beans o persistencia. No todos los servidores dan soporte a todos los componentes.

1.3. Instalación del entorno de trabajo

Para poder probar los ejemplos del libro hay que:

1. Instalar un servidor web.
2. Instalar un servidor de bases de datos.
3. Copiar los ejemplos en el servidor e importar las bases de datos.

Escaneado con CamScanner

1.3.1. Servidores web y de base de datos

La instalación se puede realizar sencillamente con XAMPP. El paquete XAMPP incluye, entre otras cosas, un servidor web Apache con PHP y un servidor de bases de datos MySQL (ahora, MariaDB).

Recurso web

Puedes descargar XAMPP desde www.apachefriends.org/es/index.html.

Para instalarlo solo hay que ejecutar el instalador con las opciones por defecto.

Una vez instalado hay que arrancar los servidores. XAMPP también incluye un panel de control para arrancar y parar los servidores (figura 1.3).

Después de arrancar el servidor web se puede comprobar si la instalación ha sido correcta accediendo desde el navegador a "localhost" (figura 1.4).

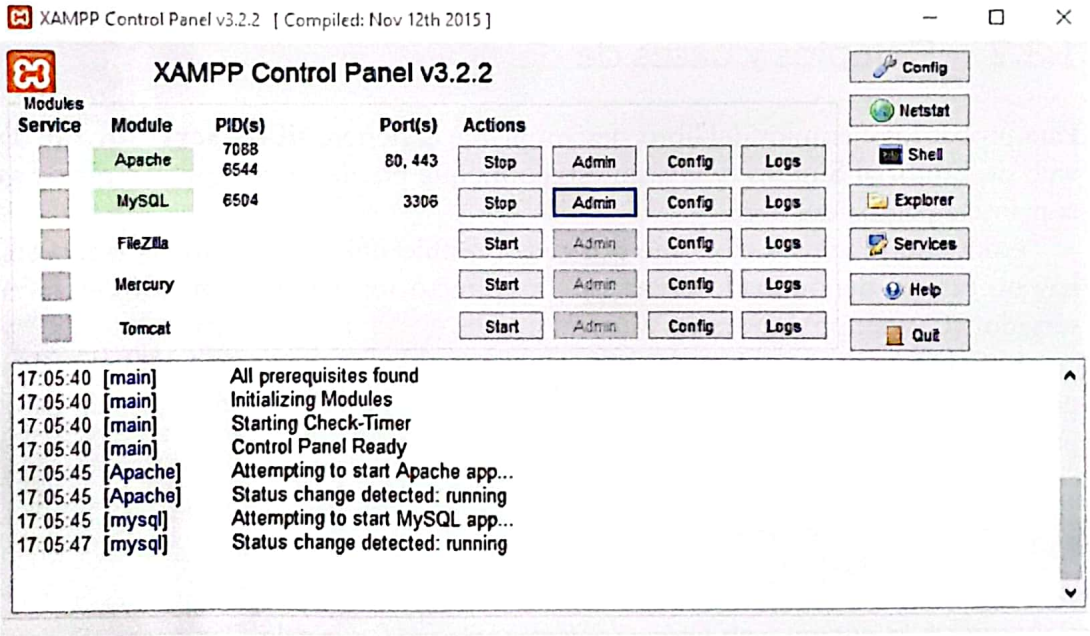
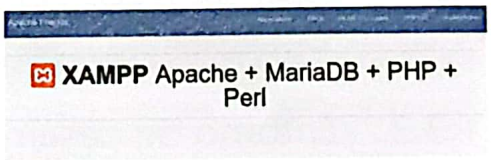


Figura 1.3 Panel de control de XAMPP.

Para comprobar si se ha instalado bien la base de datos, se arranca desde el panel de control y se accede a http://localhost/phpmyadmin/. Es una aplicación web para manejar la base datos incluida en XAMPP. Si no ha habido ningún problema, se mostrará una página como la recogida en la figura 1.5, pero con menos bases de datos en la sección de la izquierda.



Welcome to XAMPP for Windows 7.2.4

Figura 1.4 Instalación correcta.

Escaneado con CamScanner

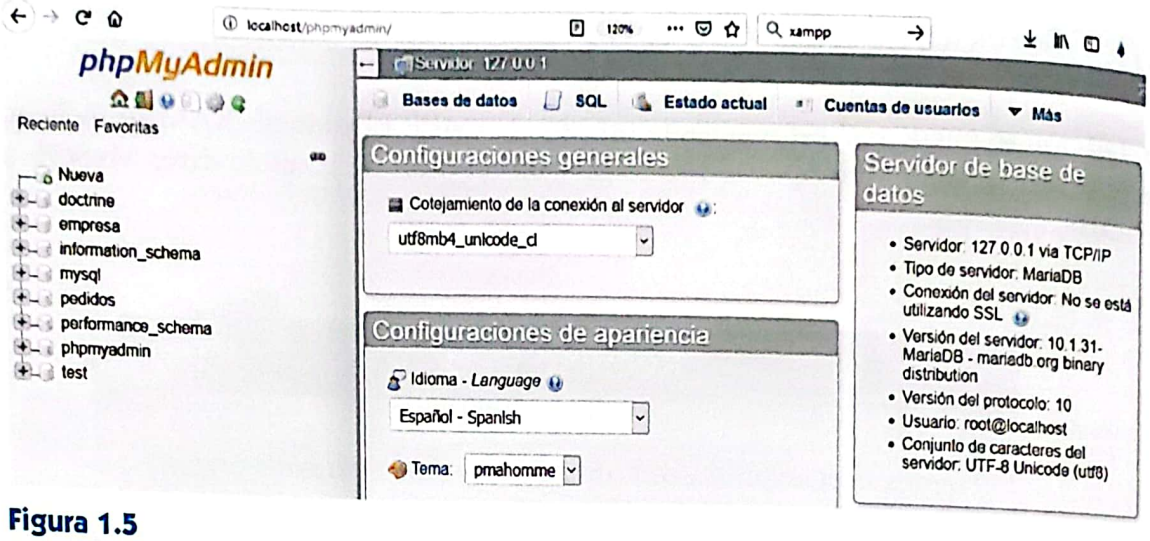


Figura 1.5
La aplicación phpMyAdmin.

Actividad propuesta 1.4

Investiga cómo se utiliza phpMyAdmin. Prueba a crear bases de datos, tablas y restricciones.

1.3.2. Ejemplos y bases de datos

Para probar los ejemplos del libro, descomprime el fichero **libro_servidor.rar** disponible en la web de Editorial Síntesis (www.sintesis.com), que puedes descargar con el código señalado en la primera página del libro.

Contiene 11 directorios, uno por cada capítulo del libro y uno llamado **vendor**. También hay un fichero denominado **crear.sql**. Los directorios hay que copiarlos en la carpeta raíz del servidor (C:\xampp\htdocs en Windows).

El fichero sirve para crear las bases de datos. Puedes importarlo desde phpMyAdmin, en la pestaña de importar. Si se ejecuta correctamente, debería crear tres bases de datos nuevas: empresa, pedidos y doctrine.

Actividad propuesta 1.5

Accede a la página web https://getcomposer.org/ e instala Composer, un gestor de dependencias de librerías de PHP. Una vez instalado, se pueden añadir nuevas librerías con un simple comando.

1.3.3. Entorno de desarrollo

Como IDE es posible utilizar un entorno ligero como Notepad++. Si se prefiere un IDE más completo, algunas opciones son:

Escaneado con CamScanner

1. *Eclipse PDT*. La versión de Eclipse para PHP.
2. *Aptana*. Puede ser buena opción si se tiene interés en integrar el desarrollo en la parte del cliente. Está basado en Eclipse y está disponible como *plugin* y como aplicación independiente.
3. *PHPStorm*. Un entorno muy completo, pero es de pago. Se puede probar gratis durante un mes.

Aunque puedan parecer excesivos para los primeros ejemplos, a medida que los ejercicios se hacen más complejos conviene usar un buen IDE.

Recursos web



Puedes descargar estos IDEs y algunos tutoriales básicos en los siguientes enlaces:

Eclipse PDT: https://www.eclipse.org/pdt/
Aptana: http://www.aptana.com/
PHPStorm: https://www.jetbrains.com/phpstorm/

Resumen

- Las aplicaciones web siguen la arquitectura cliente-servidor.
- El modelo a 3 capas es una evolución del modelo cliente-servidor.
- El modelo a 3 capas separa la lógica de las aplicaciones en tres partes para conseguir código más reusable.
- Las capas son: presentación, negocio y datos.
- El W3C mantiene varios de los estándares más importantes para el desarrollo web.
- La comunicación entre clientes y servidores se hace con el protocolo HTTP.
- Se pueden generar páginas dinámicas usando lenguajes de programación.
- El lenguaje de programación más extendido en el lado del servidor es PHP.
- El código del lenguaje se inserta dentro del HTML. El servidor se encarga de ejecutarlo antes de enviar la respuesta.
- XAMPP instala Apache con PHP y MySQL.

Ejercicios propuestos



1. Instala XAMPP siguiendo las indicaciones del apartado 1.3.1.

Escaneado con CamScanner

2. Copia los directorios del fichero libro_servidor.rar al servidor web e importa el script crear.sql al servidor de bases de datos.
3. Busca información sobre IDEs para PHP. Descarga e instala algunos para ir probándolos con los ejemplos.
4. Analiza las ventajas del modelo a 3 capas respecto al enfoque cliente-servidor básico.
5. Piensa ejemplos de aplicaciones que utilices o conozcas que sigan el modelo.
6. Busca información sobre el modelo a N capas, una generalización del modelo a 3 capas.

ACTIVIDADES DE AUTOEVALUACIÓN

1. El W3C:
   - ☐ *a)* Elabora servidores de aplicaciones.
   - ☐ *b)* Elabora y mantiene estándares de tecnologías web.
   - ☐ *c)* Elabora regulación de obligado cumplimiento para los desarrolladores.
2. El lenguaje básico para la elaboración de páginas web es:
   - ☐ *a)* HTTP.
   - ☐ *b)* HTML.
   - ☐ *c)* HTTPS.
3. En el modelo a tres capas, la lógica de la interfaz gráfica se sitúa en la capa de:
   - ☐ *a)* Presentación.
   - ☐ *b)* Datos.
   - ☐ *c)* Negocio.
4. En el modelo a tres capas no hay comunicación directa entre:
   - ☐ *a)* La capa de datos y la de negocio.
   - ☐ *b)* La capa de presentación y la de negocio.
   - ☐ *c)* La capa de datos y la de presentación.
5. En el modelo cliente-servidor, la comunicación la inicia:
   - ☐ *a)* El cliente.
   - ☐ *b)* El servidor.
   - ☐ *c)* Cualquiera de los dos.
6. ¿Cuál es el puerto asignado al protocolo HTTP?
   - ☐ *a)* 80.
   - ☐ *b)* 21.
   - ☐ *c)* 8000.

Escaneado con CamScanner

7. ¿Cuál es el puerto asignado al protocolo HTTPS?
   - ☐ *a)* 80.
   - ☐ *b)* 553.
   - ☐ *c)* 8000.

8. XAMPP incluye:
   - ☐ *a)* Servidor web y PHP.
   - ☐ *b)* Servidor web, PHP y base datos.
   - ☐ *c)* Servidor de bases de datos y PHP.

9. ¿Cuál de los siguientes no es un estándar del W3C?
   - ☐ *a)* HTML.
   - ☐ *b)* CGI.
   - ☐ *c)* HTTP.

10. Si se utiliza CGI, las peticiones se responden:
   - ☐ *a)* Usando un módulo del servidor web.
   - ☐ *b)* Usando un ejecutable en el servidor web.
   - ☐ *c)* Usando un *servlet* en el servidor web.

SOLUCIONES:

1.	a	**b**	c	**5.**	**a**	b	c	
2.	a	**b**	c	**6.**	**a**	b	c	**9.** a **b** c
3.	**a**	b	c	**7.**	a	**b**	c	**10.** a **b** c
4.	a	b	**c**	**8.**	a	**b**	c	

Escaneado con CamScanner

2

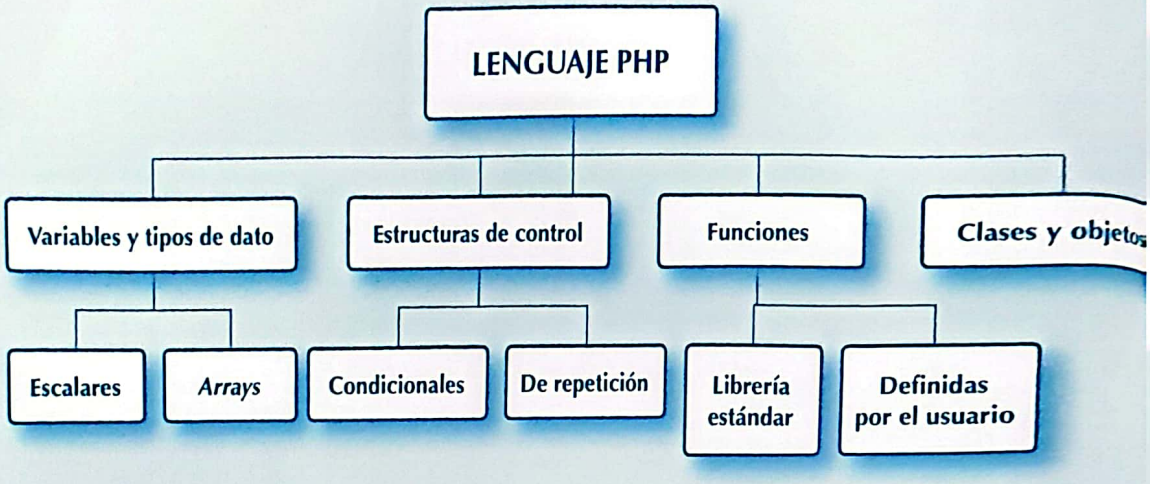
Introducción al lenguaje PHP

Objetivos

- ✓ Conocer la sintaxis básica de PHP.
- ✓ Entender cómo se integran PHP y HTML.
- ✓ Describir los tipos de datos existentes en PHP.
- ✓ Manejar las estructuras de control básicas.
- ✓ Aprender a utilizar los *arrays* asociativos.
- ✓ Presentar la notación de objetos en PHP.

Escaneado con CamScanner

Mapa conceptual



Glosario

Array. Es un tipo de dato compuesto habitual en los lenguajes de programación. Aunq los detalles varían entre lenguajes, en general, se parecen a vectores o listas orden das.

Bucle. Estructura de programación que permite repetir instrucciones.

Estructura condicional. Posibilita ejecutar o no una instrucción según se cumpla ur condición.

Función. Conjunto de instrucciones que realiza una tarea concreta.

Librería. Las funciones relacionadas entre sí se agrupan en librerías o bibliotecas.

Programación orientada a objetos. Paradigma de programación basado en la idea de ob jetos, elementos que agrupan variables y funciones.

Script. Programa sencillo, habitualmente ejecutado por un intérprete en lugar de ser com pilado.

Variable. Posición en la memoria del ordenador identificada por un nombre. La variable almacena datos. Estos datos son el valor de la variable.

2.1. PHP y HTML. Código incrustado

PHP es el lenguaje de programación para desarrollo web en el lado del servidor. Desde su apar ción en 1994 ha tenido gran aceptación y se puede decir que es lenguaje más extendido para desarrollo en el lado del servidor. Aunque no es la única opción, lo normal es que el intérpre de PHP sea un módulo del servidor web.

Escaneado con CamScanner

El lenguaje PHP es flexible y permite programar pequeños *scripts* con rapidez. Comparado con lenguajes como Java, requiere escribir menos código y, en general, resulta menos engorroso. La sintaxis de los elementos básicos es bastante parecida a la de lenguajes muy extendidos como Java y C. Por estos motivos, es un lenguaje rápido de aprender para las personas con alguna experiencia en programación.

En este capítulo se presentan la sintaxis y los elementos básicos del lenguaje PHP. Se espera que el lector esté familiarizado con los conceptos básicos de programación estructurada y orientada a objetos.

En el desarrollo web es muy habitual utilizar PHP incrustado dentro ficheros HTML. El código PHP se introduce dentro del HTML utilizando la etiqueta `<?php` para abrir el bloque de PHP y la etiqueta `?>` para cerrarlo.

El ejemplo **hola_mundo.php** muestra una página HTML completa con un bloque PHP incrustado en las líneas 7-10. El bloque tiene una única sentencia que sirve para mostrar la cadena "Hola mundo".

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Hola mundo</title>
5      </head>
6      <body>
7          <?php
8              echo "Hola mundo";
9          ?>
10     </body>
11 </html>
```

Al solicitar la página al servidor web, el resultado es:

Hola mundo

Si se consulta el código fuente de la página (pulsando Ctrl + u), se obtiene:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hola mundo</title>
    </head>
    <body>
        Hola mundo
    </body>
</html>
```

Se puede observar que el servidor web ha modificado una parte del fichero. El bloque PHP ha desaparecido y en su lugar aparece "Hola mundo". El servidor web procesa los bloques de PHP y los sustituye por su salida, es decir, por lo que muestran las sentencias del bloque. En este caso se utiliza `echo`, que muestra el valor de una variable o de una cadena de texto.

Escaneado con CamScanner



TOMA NOTA

Para probar los ejemplos del libro hay que instalar el entorno de trabajo como se expli en el capítulo 1. Para probar este ejemplo accede a http://localhost/cap2/hola_mund php. Para el resto de los ejemplos solo hay que cambiar el nombre del fichero. Recuer arrancar el servidor web.

2.2. Sintaxis de PHP

El elemento básico en PHP es el bloque. Un bloque PHP está delimitado por las etiquetas rrespondientes y contiene una serie de sentencias separadas por punto y coma.

```
<?php
   sentencia1;
   sentenciaN;
? >
```

RECUERDA

✓ Cuando un fichero contiene solo PHP se recomienda no cerrar la etiqueta del último bloque. Puede dar problemas si la respuesta del servidor involucra varios ficheros.

2.3. Variables y tipos de dato

Una de las características principales de PHP es que es un lenguaje *no fuertemente tipado*. Est quiere decir que no es necesario indicar el tipo de dato al declarar una variable. De hecho, l variables no se declaran, se crean la primera vez que se les asigna un valor. El tipo de dato de pende del valor con que se inicialicen.

Esto agiliza la escritura de programas, pero también tiene inconvenientes. Si no se prest atención, puede dar lugar a código de baja de calidad y, a medida que las aplicaciones crecen pueden darse errores difíciles de depurar.

2.3.1. Declaración de variables

En PHP los identificadores de las variables van siempre precedidos por el carácter '$'. El identificador de la variable debe comenzar por una letra o un guion bajo ('_'), y puede estar formado por números, letras y guiones bajos.

Para declarar una variable, solo hay que asignarle un valor:

```
$nombre = valor;
```


Escaneado con CamScanner

Por ejemplo, esta sentencia declara la variable `$entero`, que será de tipo `integer` porque se inicializa con un entero.

```
$entero = 4;
```

También es posible cambiar el tipo de dato de una variable simplemente asignándole un valor de otro tipo de dato, como se puede ver en el ejemplo **tipos_dato.php** (líneas 8-12). El ejemplo utiliza la función `gettype()`, que devuelve el tipo de dato de una variable.

```
1  <?php
2  /* declaración de variables */
3  $entero = 4; // tipo integer
4  $numero = 4.5; // tipo coma flotante
5  $cadena = "cadena"; // tipo cadena de caracteres
6  $bool = TRUE; //tipo booleano
7  /* cambio de tipo de una variable */
8  $a = 5; // entero
9  echo gettype($a); // imprime el tipo de dato de a
10 echo "<br>";
11 $a = "Hola"; // cambia a cadena
12 echo gettype($a); // se comprueba que ha cambiado
```

La salida del ejemplo confirma que la variable cambia de tipo de dato.

```
integer
string
```

2.3.2. Asignación por copia y por referencia

En principio, la asignación de variables se realiza mediante copia. Es decir, si hacemos:

```
$a = $b;
```

se crea una nueva variable *a* y se le asigna el valor que tenga *b*. Las variables *a* y *b* representan posiciones diferentes de memoria, aunque tengan el mismo valor después de la asignación.

También es posible definir una referencia a una variable utilizando el operador *ampersand*:

```
$var2 = &$var1;
```

En este caso `$var2` no es una nueva variable con el valor de `$var1`. Por el contrario, `$var2` apunta a la misma dirección de memoria que `$var1`, de manera que `$var1` y `$var2` son en realidad dos nombres para el mismo dato. En el ejemplo **copia.php** se muestra que, al cambiar el valor de una referencia, se modifica también el de la variable referenciada.

```
<?php
    $var1 = 100;
    $var2 = &$var1; // asignación por referencia
```


Escaneado con CamScanner

```
$var3 = $var1; // asignación por copia
echo "$var2<br>";// muestra 100
$var2 = 300; // cambia el valor de $var2
echo "$var1<br>";// $var1 también cambia
$var3 = 400; // este cambio no afecta a $var1
echo $var1;
```

La salida de este ejemplo será:

100
300
300

2.3.3. Variables no inicializadas

Si se intenta utilizar una variable antes de asignarle un valor, se genera un error de tipo E_N TICE, pero no se interrumpe la ejecución del *script*. Si una variable no inicializada apare dentro de una expresión, dicha expresión se calcula tomando el valor por defecto para ese ti de dato. En el ejemplo **no_init.php**, se puede ver lo que ocurre al utilizar una variable no in cializada dentro de una expresión.

```
1  <?php
2  $var1 = 100;
3  $var3 = 100 + $var2;//$var2 no existe, se toma como 0
4  echo "$var3 <br>"; // muestra 100
5  $var3 = 100 * $var2; // $var2 no existe, se toma como 0
6  echo "$var3 <br>"; // muestra 0
```

En la línea 3, se intenta sumar una variable no inicializada. Como el valor por defecto par un entero es 0, el resultado de la suma es 100. En la línea 5, se intenta multiplicar por una va riable no inicializada y, al multiplicar por 0, el resultado es 0.

La salida muestra un mensaje de error por cada intento de utilización de una variable no inicializada.

Notice: Undefined variable: var2 in C:\xampp\htdocs\cap2\ejeinicializaciones.php on line 3
100
Notice: Undefined variable: var2 in C:\xampp\htdocs\cap2\ejeinicializaciones.php on line 5
0

2.3.4. Constantes

Para definir constantes se utiliza la función define(), que recibe el nombre de la constante y el valor que queremos darle.

```
define("LIMITE", 1000);
```

Es habitual utilizar identificadores en mayúsculas para las constantes.

Escaneado con CamScanner

2.3.5. Tipos de datos escalares

PHP ofrece cuatro tipos de datos escalares: *integer*, *float*, *boolean* y *string*.

A) Números

Para representar números enteros se usa el tipo de dato *integer*. El tamaño y los valores máximo y mínimo de un entero dependen de la plataforma, y se pueden conocer mediante los constantes PHP_INT_SIZE, PHP_INT_MAX y PHP_INT_MIN, respectivamente.

Para números reales, se utiliza el tipo *float*. El tamaño también depende de la plataforma, pero suele ofrecer una precisión de 14 decimales. En cualquier caso, la precisión de los *float* presenta los mismos problemas que en otros lenguajes y los redondeos pueden dar sorpresas.

La conversión entre *integer* y *float* es automática. Si se recibe un *float* cuando se espera un *integer*, se trunca. Si al realizar una operación sobre un entero el resultado supera los valores límite o tiene decimales, se convierte a *float*. También se pueden utilizar los operadores de conversión, `(int)` o `(float)`.

El ejemplo **tipos_numericos.php** muestra las diferentes opciones disponibles para literales *integer* o *float* y algunas operaciones entre ellos.

```
<?php
    echo PHP_INT_SIZE.'<br>';
    echo PHP_INT_MIN.'<br>';
    echo PHP_INT_MAX.'<br>';
    $a = 0b100; // en binario
    $a = 0100; // octal
    $a = 0x100; // hexadecimal
    $a = 3/2; // la división entre enteros no da problemas
    echo $a.'<br>'; // 1.5
    $b = 7.5;
    $a = (int)$b; //casting a int
    echo $a.'<br>'; // 7, se trunca
    $b = 7e2; // notación científica
    $b = 7E2;
```

B) Cadenas

El tipo de dato *string* permite almacenar cadenas de caracteres. Para delimitar una cadena es posible utilizar comillas simples o dobles, pero hay una diferencia. Si se utilizan comillas dobles (también llamadas *comillas mágicas*), las variables que aparezcan dentro de la cadena se sustituirán por su valor. Las comillas dobles son muy prácticas, ya que es más rápido insertar directamente las variables que montar las cadenas con varias concatenaciones.

```
<?php
    $var = "Paco";
    $a = "Hola $var <br>";
    $b = 'Hola $var';
    echo $a;
    echo $b;
```

La salida será:

Escaneado con CamScanner

```
Hola Paco
Hola $var
Hola Paco
```

En el primer caso, `$var` se sustituye por su valor, "Paco". En el segundo, se interp como texto normal. La última línea del *script* muestra el operador de concatenación d cadenas, `"."`.



TOMA NOTA

A la hora de formatear la salida hay que tener en cuenta que esta va a ser procesada como HTML por un navegador web, es decir, los saltos de línea se ignoran y los espacios en blanco consecutivos colapsan. Los caracteres de escape como '\n', '\r' y '\t' son ignorados por el navegador. Para introducir un salto de línea la opción más sencilla es incluir la etiqueta de salto de línea de HTML ("
") en la salida, como se puede ver en los ejemplos anteriores.

C) Booleanos

Este tipo de dato es para variables *booleanas*. Solo pueden tomar los valores TRUE y FALS verdadero y falso. Este es el tipo de dato que se obtiene, entre otros casos, como resultado de l operadores de comparación y se utiliza en sentencias condicionales y bucles.

Cuando se espera un valor *booleano* y se recibe otro tipo de dato, se aplican las reglas d conversión recogidas en el cuadro 2.1.

CUADRO 2.1
Conversión implícita a *boolean*

Tipo	Valor como *boolean*
integer	Si es 0 se toma FALSE, en otro caso como TRUE
float	Si es 0.0 se toma FALSE, en otro caso como TRUE
string	Si es una cadena vacía o "0", se toma como FALSE, en otro caso como TRUE
variables no inicializadas	FALSE
null	FALSE
array	Si no tiene elementos se toma FALSE, en otro caso como TRUE

D) Otros tipos de dato

Además de los tipos de datos dato escalares en PHP también existen los siguientes tipos de datos:

Escaneado con CamScanner

1. *array*. Para representar colecciones de elementos. Los *arrays* de PHP son muy potentes y se explican con detalle en el apartado 2.2.5.
2. *object*, PHP tiene soporte completo para la programación orientada a objetos. Se explica en el apartado 2.2.8.
3. *callable*. Un tipo de dato especial para representar funciones de *callback*, funciones que se pasan a otras funciones.
4. *null*. El tipo de dato *null* representa una variable que no ha sido asignada. Solo puede tomar un único valor, NULL. Se considera que una variable es de tipo *null* si se le asigna el valor NULL o no está inicializada.
5. *resource*. Este tipo de dato representa recursos externos, como una conexión a una base de datos.

CUADRO 2.2
Tipos de dato en PHP

Tipo	Descripción
integer	Números enteros
float	Números reales en coma flotante
string	Cadenas de caracteres
boolean	Booleanos, TRUE o FALSE
array	Colección de elementos identificados
object	Un objeto es una instancia de una clase
callable	Para las funciones de *callback*
null	Para representar variables no asignadas
resource	Representa recursos externos

2.3.6. Ámbito de las variables

El ámbito de una variable es la parte del código en que esta es visible. Una variable declarada en un fichero PHP está disponible en ese fichero y en los ficheros que se incluyan desde este.

Por otro lado, las funciones definen un ámbito local, de manera que las variables que se declaran en las mismas solo son accesibles desde la propia función. Además, desde la función no se puede acceder a otras variables que no sean las locales o sus argumentos.

Para definir variables globales hay dos opciones. La palabra reservada `global` y la variable predefinida `$_GLOBALS`. Las variables globales son accesibles desde cualquier función o fichero de la aplicación.

2.3.7. Variables predefinidas

En PHP hay muchas variables predefinidas disponibles. Contienen información sobre el servidor, datos enviados por el cliente o variables de entorno. Dentro de las variables predefinidas

Escaneado con CamScanner

hay un grupo de ellas, las *superglobales*, que están disponibles en cualquier ámbito. Cada una de ellas guarda información de un tipo.

Por ejemplo, en `$_SERVER` hay información sobre el servidor en el que está alojada la página. El *script* **global_server.php** muestra algunos de los datos disponibles.

```
<?php
    echo "Ruta dentro de htdocs: ". $_SERVER['PHP_SELF'];
    echo "Nombre del servidor: ". $_SERVER['SERVER_NAME'];
    echo "Software del servidor: ". $_SERVER['SERVER_SOFTWARE'];
    echo "Protocolo: ". $_SERVER['SERVER_PROTOCOL'];
    echo "Método de la petición: ". $_SERVER['REQUEST_METHOD'];
```

Las variables *superglobales* (cuadro 2.3) son muy relevantes para el desarrollo de aplicaciones web y se irán poniendo en práctica a lo largo del libro.

CUADRO 2.3
Variables *superglobales*

Nombre	Descripción
$GLOBALS	Variables globales definidas en la aplicación
$_SERVER	Información sobre el servidor
$_GET	Parámetros enviados con el método GET (en la URL)
$_POST	Parámetros enviados con el método POST (formularios)
$_FILES	Ficheros subidos al servidor
$_COOKIE	*Cookies* enviadas por el cliente
$_SESSION	Información de sesión
$_REQUEST	Contiene la información de $_GET, $_POST y $_COOKIE
$_ENV	Variables de entorno

2.4. Comentarios

En PHP se pueden utilizar comentarios:

- De bloque, encerrados entre "`/*`" y "`*/`".
- De línea, comenzando por "`//`" o por "`#`".

```
<?php
    //comentario de línea
    $a = 0; // comentario de línea
    /* comentario
    de bloque
    */
    # comentario de una sola línea
```


Escaneado con CamScanner

Como en cualquier lenguaje, comentar adecuadamente el código se considera una buena práctica de programación.

2.5. Estructuras de control

PHP cuenta con las estructuras de control habituales en la programación estructurada para realizar sentencias condicionales y de repetición. La sintaxis es muy parecida a la de Java o C.

2.5.1. Estructuras condicionales

Las estructuras condicionales de PHP son `if`, `if-else`, `if-elseif` y `switch`.

La sentencia condicional más sencilla es la estructura `if`. La sintaxis general es:

```
if (condición)
    instrucción
```

Si se cumple la condición, se ejecuta la instrucción. Si no se cumple, no se ejecuta. Para agrupar dentro del *if* más de una sentencia, se encierran entre llaves.

```
if (condición){
    instrucción 1
    instrucción n
}
```

La condición es una expresión que se evalúa a verdadero a falso, siguiendo las normas de conversión a *boolean* si es necesario.

En el siguiente ejemplo solo se cumple el segundo `if` y, por tanto, la salida del programa sería "Es mayor que cero".

```
<?php
    $var = 3;
    if($var < 0) echo "Es menor que cero";
    if ($var > 0){
        echo "Es mayor que cero";
    }
```

Cuando se utiliza un `if` se puede añadir un `else`. Las instrucciones dentro del `else` solo se ejecutan cuando la condición del `if` no se cumple.

```
<?php
    $var = 3;
    if($var < 0){
        echo "Es menor que cero";
    }else{
        echo "Es mayor o igual que cero";
    }
```


Escaneado con CamScanner

Si se anidan varias sentencias condicionales, se puede usar `elseif`, que es equivalente a `else`

```
<?php
    $var = 3;
    if ($var == 1) {
        echo "Es un uno";
    }elseif ($var == 2) {
        echo "Es un dos";
    }elseif ($var == 3) {
        echo "Es un tres";
    }else{
        echo "No es un uno, ni un dos, ni un tres"
    }
```

La primera condición que se cumpla es la que se ejecuta. Si no se cumple ninguna, se ejecuta el `else` final (si lo hay).

Para agrupar varios `if` puede ser útil la estructura `switch`, también habitual en otros lenguajes. El ejemplo switch.php es equivalente al anterior, pero más fácil de leer.

```
<?php
    $var = 3;
    switch($var){
        case 1:
            echo "Es un 1";
        break;
        case 2:
            echo "Es un 2";
        break;
        case 3:
            echo "Es un 3";
        break;
        default:
            echo "No es un 1, ni un 2, ni un 3";
    }
```

Según el valor de `$var`, se ejecutará un `case` u otro. La sección `default` se ejecuta cuando no se da ninguno de los `case`.

RECUERDA

✓ Si no hay un *break* al final de un *case*, la ejecución continúa con el siguiente.

2.5.2. Estructuras de repetición

Las estructuras de repetición o bucles sirven para repetir un conjunto de instrucción mientras se dé una condición. PHP cuenta con las estructuras habituales: `for`, `while` y `do-while`, que tienen la misma sintaxis que en Java o C.

Escaneado con CamScanner

Para el bucle `for`, la sintaxis es:

```
for(instrucciones de inicialización; condición; instrucciones de iteración) {
    instrucciones del bucle;
}
```

Las instrucciones de inicialización se realizan una única vez al llegar al bucle. Las instrucciones dentro del cuerpo del bucle se repetirán mientras se cumpla la condición.

Después de ejecutar las instrucciones de inicialización se evalúa la condición. Si se cumple, se ejecutan las instrucciones del bucle y las instrucciones de iteración. Después se vuelve a comprobar la condición. El proceso se repite hasta que la condición deja de cumplirse y a partir de ahí la ejecución continúa con las instrucciones que estén después del bucle.

El ejemplo **bucle_for.php** muestra un bucle `for` básico que se repite cinco veces. La variable `$i` se inicializa a cero y su valor se incrementa en uno tras cada iteración. El bucle se repite mientras `$i` valga menos que cinco, es decir, de cero a cuatro.

```
<?php
    for($i = 0; i < 5; $i = $i + 1){
        echo "$i <br>";
    }
```

El bucle `while` también sigue la sintaxis habitual:

```
while (condición) {
    instrucciones;
}
```

Igual que con el `for`, las instrucciones se ejecutan mientras se cumpla la condición. El `while` no cuenta con instrucciones de inicialización o iteración, pero se pueden añadir antes y al final del bucle, respectivamente. El ejemplo **bucle_while.php** tiene la misma salida que el anterior.

```
<?php
    $i = 0;
    while($i < 5){
        echo "$i <br>";
        $i = $i +1;
    }
```

El bucle `do-while` es similar, pero la condición se evalúa después de ejecutar las ejecuciones del bucle.

```
do {
    instrucciones;
} while (condición);
```

Este ejemplo es equivalente a los anteriores.

```
<?php
    $i = 0;
    do{
        echo "$i <br>";
        $i = $i +1;
    } while ($i < 5);
```

Escaneado con CamScanner

El bucle `do-while` se diferencia de los anteriores en que las instrucciones dentro del bu se ejecutan por lo menos una vez, como se puede ver en el siguiente ejemplo:

```
<?php
    $i = 0;
    do {
        echo "En el do-while: $i <br>";
        $i = $i + 1;
    } while ($i < 0);
    while ($i < 0) {
        echo "En el while: $i <br>";
        $i = $i + 1;
    }
```

La salida de este ejemplo será:

En el do-while: 0

Dentro de un bucle es posible usar las sentencias `break` y `continue`. La primera sirve par salir del bucle o `switch` en el que aparece. En el ejemplo **break.php**, el bucle acaba antes d llegar a cinco porque al llegar a tres se ejecuta el `break`.

```
<?php
    $i = 0;
    while ($i < 5){
        echo "$i <br>";
        $i++; // es lo mismo que $i = $ i + 1;
        if ($i == 3){
            break;
        }
    }
```

En PHP la sentencia `break` admite un número, que representa el número de niveles de anidación de los que debe salir. Así se puede salir de un bucle anidado utilizando una única secuencia `break` como se puede ver a continuación.

```
<?php
    echo "Primer for anidado: <br>";
    for ($i = 0; $i < 3; $i++) {
        for ($j = 0; $j < 3; $j++) {
            echo "i: $i j: $j <br>";
            if ($j == 1) {
                break; //es lo mismo que poner break 1
            }
        }
    }
    echo "Segundo for anidado: <br>";
    for ($i = 0; $i < 3; $i++) {
        for ($j = 0; $j < 3; $j++) {
            echo "i: $i j: $j <br>";
                if ($j == 1){
                    break 2;
            }
        }
    }
```


Escaneado con CamScanner

En el primer bucle anidado se utiliza un `break` sin pasarle ningún número, que es lo mismo que pasarle un uno, y por tanto solo sale del bucle interior. El bucle exterior se repetirá tres veces y el interior seis, dos por cada vez que se ejecute el exterior, con los valores de $j=0 y $j=1. En el segundo bucle anidado, al ejecutarse la línea x se acaban los dos bucles. Esto ocurre en la primera iteración del bucle externo y la segunda del interno. La salida será:

Primer for anidado:

```
i: 0 j: 0
i: 0 j: 1
i: 1 j: 0
i: 1 j: 1
i: 2 j: 0
i: 2 j: 1
```

Segundo for anidado:

```
i: 0 j: 0
i: 0 j: 1
```

Esto también se aplica a la sentencia condicional `switch`. Por ejemplo, si un `switch` está dentro de un bucle, utilizando `break 2` se sale de ambos.

```
<?php
    $i = 0;
    echo "Primer switch anidado: <br>";
    while ($i< 2) {
        switch ($i) {
            case 0:
                echo "Es un cero <br>";
                break;
            case 1:
                echo "Es un uno <br>";
                break;
        }
        $i++;
    }
    $i = 0;
    echo "Segundo switch anidado: <br>";
    while ($i< 2) {
        switch ($i) {
            case 0:
                echo "Es un cero <br>";
                break 2;
            case 1:
                echo "Es un uno <br>";
                break;
        }
        $i++;
    }
```

Escaneado con CamScanner

La salida de este ejemplo es:

```
Primer switch anidado:
Es un cero
Es un uno
Segundo switch anidado:
Es un cero
```

La sentencia `continue` fuerza una nueva iteración del bucle. Las instrucciones que est dentro del bucle, pero después de `continue` no se ejecutan y, si se cumple la condición del b cle, se ejecuta una nueva iteración. Si se trata de un bucle `for`, se ejecutan las instrucciones autoincremento antes de evaluar la condición.

En el ejemplo **continue.php** se puede observar el funcionamiento de `continue`.

```
<?php
    for($i = 0; $i < 5; $i = $i + 1){
        if ($i == 3){
            continue;
        }
        echo "$i <br>";
    }
```

Cuando `$i` vale tres se ejecuta el `continue`, por lo que el `echo` a continuación no ejecu esa iteración. La salida será:

```
0
1
2
4
```

Actividad propuesta 2.1



Escribe un programa que calcule el factorial de un número.
Recuerda que el factorial solo está definido para números enteros mayores o iguales que cero.

2.5.3. Otras estructuras de control

Es posible incluir otros ficheros utilizando las sentencias `include` y `require`.

```
include "mifichero.php";
require "mifichero.php";
```

Se diferencian solo en el tratamiento de errores. Si no se encuentra el fichero especificado, `include` genera un error de tipo E_NOTICE, pero `require` genera un error E_FATAL, lo que implica el fin de la ejecución del *script*. Si no hay error, en ambos casos el fichero especificado se

Escaneado con CamScanner

ejecuta. Si el fichero es una librería con funciones, estas se declaran y quedan disponibles para el fichero que lo incluye. Si el fichero contiene sentencias fuera de una función, estas se ejecutan.

El ejemplo **requerir.php** incluye otro fichero.

```
<?php
    $a = "variable del principal";
    require "ejerequerido.php";
    $b = "otra variable del principal";
    echo "En el script principal";
```

El fichero requerido es el siguiente:

```
<?php
    echo "En el fichero requerido <br>";
    echo $a;
    echo $b;
```

Al solicitar al servidor **cap2/requerir.php** se obtiene esta salida:

```
En el fichero requerido
variable del principal
Notice: Undefined variable: b in C:\xampp\htdocs\cap2\ejerequerido.php on line 4
En el script principal
```

Se puede observar que:

- Las instrucciones del fichero requerido se ejecutan.
- Las variables del fichero principal están disponibles en el requerido si se habían inicializado antes de requerirlo. Esto ocurre con $a, pero no con $b, que se inicializa después y, por tanto, se muestra un mensaje de error. Es decir, el ámbito de las variables del fichero principal incluye los ficheros requeridos.

También están disponibles las sentencias `require_once` e `include_once`, análogas a las anteriores, pero con la diferencia de que solo se ejecutan los ficheros especificados si no han sido incluidos o requeridos con anterioridad.

TOMA NOTA

La inclusión de ficheros tiene una diferencia con otros lenguajes. Supongamos que el fichero A incluye al fichero B. Las rutas relativas que aparezcan en B se interpretarán a partir del directorio de A.

Para solucionarlo, en el fichero B se utiliza dir_name(__FILE__) que devuelve la ruta del fichero:

```
include( dirname(__FILE__)."\".'fichero.php');
```


Escaneado con CamScanner

Para acabar con las estructuras de control, la sentencia `return` finaliza la ejecución del fichero cuando se encuentra fuera de una función. En el siguiente ejemplo, el `echo final` no llega a ejecutar, porque se ha ejecutado antes el `return`.

```
<?php
    $a = 0;
    if ($a == 0){
        return;
    }
    echo "Después del if";
```

Si el *script* se estaba ejecutando mediante un `require` o `include`, la ejecución continúa en el fichero que lo incluyó. Si se pasa como argumento un número entero, este se devuelve como valor de retorno.

Si se encuentra dentro de una función, finaliza la ejecución de esta y la función devuelve el argumento del `return`. La ejecución continua en el fichero o función que llama a la función.

2.6. Operadores

PHP cuenta con los operadores habituales para operaciones aritméticas, lógicas, de manipulación de cadenas y demás.

Entre los operadores de comparación cabe señalar los operadores "===" y "!==", llamados Idéntico y No Idéntico, que no están presentes en todos los lenguajes. El operador Idéntico se usa para comparar dos expresiones y se evalúa como verdadero cuando las dos expresiones tienen el mismo valor y además el mismo tipo de dato. Se diferencia del operado Igual, "==", en que este, cuando las expresiones no tienen el mismo tipo de dato, intenta convertirlas antes de compararlas. El operador Idéntico es útil para evitar sorpresas inesperadas en la conversión de datos. En el ejemplo **identico.php** se puede comprobar la diferencia.

```
<?php
    $a = 3;
    $b = "3";
    if ($a == $b){
        echo "Son iguales <br>";
    }else{
        echo "No son iguales <br>";
    }
    if ($a === $b){
        echo "Son idénticos <br>";
    }else{
        echo "No son idénticos <br>";
    }
```

La primera comparación permite la conversión de tipos, y la cadena "3" se convierte al entero 3. La salida será:

```
Son iguales
No son idénticos
```


Escaneado con CamScanner

Cuadro 2.4
Operadores

Operadores de comparación	
e1 === e2	Idéntico. Verdadero si las dos expresiones son del mismo tipo y tienen el mismo valor.
e1 == e2	Igual. Verdadero si las dos expresiones son iguales tras la conversión de tipos, si es necesaria.
e1 !== e2	No idéntico.
e1 != e2, e1 <> e2	No igual.
e1 >= e2, e1 > e2, e1 <= e2, e1 < e2	Mayor o igual, mayor, menor o igual, menor.
e1 ?? e2 ?? e3	Comenzando por la izquierda, devuelve la primera expresión no nula.
Operadores aritméticos	
+e1	Como operador unario, sirve para convertir la expresión a *integer* o *float*, según corresponda.
–e1	Como operador unario, cambio de signo.
e1 + e2, e1 - e2, e1 * e2, e1 / e2	Suma, resta, multiplicación, división.
e1 % e2	Módulo.
e1 ** e2	Potencia.
Operadores lógicos	
e1 and e2, e1 && e2	Y. Verdadero si las dos expresiones se evalúan a TRUE.
e1 or e2, e1 || e2	O. Verdadero si una o las dos expresiones se evalúan a TRUE.
e1 xor e2	O exclusivo. Verdadero si solo una de las expresiones se evalúa a TRUE.
!e1	Devuelve TRUE si la expresión es FALSE y viceversa.
Operadores a nivel de bit	
e1 & e2, e1 | e2, e1 ^ e2, ~e1	Y, O, O exclusivo y negación.
$var» N	Desplaza N bits de $var hacia la izquierda.
$var «N	Desplaza N bits de $var hacia la derecha.
Operadores de asignación	
$var1 = e1	Asignación por valor.
$var2 = &$var2	Asignación por referencia.
$var += e1, $var -= e1, $var *= e1, $var /= e1	Equivalentes a $var = $var + e1, $var = $var – e1… Válido para cualquier operador binario aritmético, de cadenas o *arrays*.
Otros operadores	
$var++, $var--	Devuelve $var, luego le suma (resta) 1.
++$var, --$var	Suma (resta) 1 a $var, devuelve el valor actualizado.
$cad1. $cad2	Concatena dos cadenas.

Escaneado con CamScanner

2.7. *Arrays*

Los *arrays* en PHP son una estructura muy flexible y potente. Unifica en un solo tipo lo q en otros lenguajes se consigue con *arrays* básicos, vectores, listas o diccionarios. Los elem tos de un *array* se identifican por una clave, que puede ser un entero o una cadena. Los e mentos guardan un orden dentro del *array*. Este orden está determinado por el orden de elementos al declarar el *array* o al añadir nuevos.

Para declarar un *array* se puede utilizar la función `array()`.

```
$var = array(
    clave1 => valor1,
    ...
    claven => valorn
);
```

También es posible utilizar la notación de corchetes

```
$arr = [
    clave1 => valor1,
    ...
    claven => valorn
];
```

Se puede acceder a cada elemento del *array* por su clave utilizando los corchetes, como habitual en otros lenguajes:

```
$arr[clave] = valor;
```

El ejemplo **arrays1.php** muestra cómo declarar y utilizar los *arrays*. En el ejemplo se utili la función `print_r()`, que muestra información sobre la variable que se le pase como argumen to. Si se le pasa un *array*, muestra las claves y valores de todos los elementos.

```
<?php
    $arr1 = [
        0 => 444,
        1 => 222,
        2 => 333,
    ];
    print_r($arr1);
    echo "<br>". "pos 0: ". $arr1[0]. "<br>";
    $arr1[0] = 555;
    print_r($arr1);
    echo "<br>";
    $arr2 = array (
        "1111A" => "Juan Vera Ochoa",
        "1112A" => "Maria Mesa Cabeza",
        "1113A" => "Ana Puertas Peral"
    );
    $arr2["1113A"] = "Ana Puertas Segundo";
```

Para recorrer un *array* lo habitual es utilizar el bucle `foreach`.

Escaneado con CamScanner

```
foreach($arr as $valor){
    ...
}
```

El cuerpo del bucle se ejecuta una vez por cada elemento del *array* `$arr`. En cada iteración del bucle `$valor` almacena el elemento correspondiente.

También se puede utilizar especificando una variable para la clave,

```
foreach($arr as $clave => $valor){ ... }
```

En este caso, en cada iteración también se dispondrá de la clave del elemento correspondiente en la variable `$clave`.

```
<?php
    $arr2 = array (
        "1111A" => "Juan Vera Ochoa",
        "1112A" => "Maria Mesa Cabeza",
        "1113A" => "Ana Puertas Peral"
    );
    foreach ($arr2 as $nombre) {
        echo "$nombre <br>";
    }
    foreach ($arr2 as $codigo => $nombre) {
        echo "Código: $codigo Nombre: $nombre <br>";
    }
```

Si se pretende modificar el contenido del *array* al recorrerlo con un bucle `foreach`, hay que utilizar una referencia al declarar las variables del bucle. En el ejemplo **foreach_modificar.php**, el primer bucle no utiliza referencias y, por tanto, no modifica el *array*. El segundo bucle es la forma correcta de hacerlo.

```
<?php
    $arr1 = array(
        "Viernes" => 22,
        "Sábado" => 34
    );
    /* no modifica el array */
    foreach ($arr1 as $cantidad) {
        $cantidad = $cantidad * 2;
    }
    print_r($arr1);
    echo "<br>";
    /* modifica el array */
    foreach ($arr1 as &$cantidad) {
        $cantidad = $cantidad * 2;
    }
    print_r($arr1);
```

La salida del programa será:

```
Array ( [Viernes] => 22 [Sábado] => 34 )
Array ( [Viernes] => 44 [Sábado] => 68 )
```

También es posible declarar un *array* sin usar claves. En ese caso, se asignan como claves por defecto enteros consecutivos empezando por 0. Si se añade un nuevo elemento sin especificar clave, la clave será el número siguiente a la mayor clave entera presente en el *array*.

Escaneado con CamScanner

```php
<?php
    $arr1 = array(10, 20, 30, 40);
    print_r($arr1);
    echo "<br>";
    $arr1[] = 5;
    print_r($arr1);
    echo "<br>";
    $arr1[10] = 6;
    $arr1[] = 5;
    print_r($arr1);
    echo "<br>";
```

Salida:

```
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 )
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 [4] => 5 )
Array ( [0] => 10 [1] => 20 [2] => 30 [3] => 40 [4] => 5 [12] => 6 [11] => 5 [13] => 5 )
```

Algunos operadores también están disponibles para los *arrays*, pero su significado no exactamente el mismo. Por ejemplo, el operador "Idéntico" solo será verdadero si los dos ar tienen todos los elementos iguales tanto en clave como en valor y además están en el mis orden. Para el operador "Igual", el orden de las claves no importa.

```php
<?php
    $arr1 = array(
        1 => "3000",
        2 => "4000",
    );
    $arr2 = array(
        1 => 3000,
        2 => 4000,
    );
    $arr3 = array(
        2 => "4000",
        1 => "3000",
    );
    if($arr1 == $arr2){
    echo "arr1 y arr2 son iguales <br>";
    }else{
        echo "arr1 y arr2 no son iguales <br>";
    }
    if($arr1 == $arr3){
        echo "arr1 y arr3 son iguales <br>";
    }else{
        echo "arr1 y arr3 no son iguales <br>";
    }
    if($arr1 === $arr2){
        echo "arr1 y arr2 son idénticos <br>";
    }else{
        echo "arr1 y arr2 no son idénticos <br>";
    }
    if($arr1 === $arr3){
        echo "arr1 y arr3 son idénticos <br>";
    }else{
        echo "arr1 y arr3 no son idénticos <br>";
    }
```


Escaneado con CamScanner

La salida será:

```
arr1 y arr2 son iguales
arr1 y arr3 son iguales
arr1 y arr2 no son idénticos
arr1 y arr3 son idénticos
```

El operador "+" utilizado con dos *arrays* devuelve su unión. El *array* resultante contendrá primero los elementos del primer *array* (el que aparece a la izquierda del operador) y a continuación los del segundo, pero sin repetir claves. Si hay elementos con la misma clave en los dos *arrays*, el del segundo no se añade al resultado.

```
<?php
    $arr1 = array(
        10 => "3000",
        20 => "4000",
        30 => "6000",
    );
    print_r($arr1);
    echo "<br>";
    $arr2 = array(
        10 => "8000",
        15 => "6000",
        20 => "4000",
    );
    print_r($arr2);
    echo "<br>";
    $arr3 = $arr1 + $arr2;
    print_r($arr3);
```

El resultado será:

```
Array ( [10] => 3000 [20] => 4000 [30] => 6000 )
Array ( [10] => 8000 [15] => 6000 [20] => 4000 )
Array ( [10] => 3000 [20] => 4000 [30] => 6000 [15] => 6000 )
```

CUADRO 2.5
Operadores para *arrays*

Operador	Descripción
$a1 === $a2	Idéntico. Verdadero si los dos *arrays* tienen las claves y valores iguales, en el mismo orden y del mismo tipo.
$a1 !== $a2	No idéntico.
$a1 == $a2	Igual. Verdadero si los dos *arrays* tienen las claves y valores iguales.
$a1 != $a2, $a1 <> $a2	No igual.
$a1 + $a2	Unión. Devuelve un *array* con los elementos de ambos.

Escaneado con CamScanner

2.8. Funciones y librerías

Una función es un conjunto de instrucciones que realiza una tarea concreta. Las funciones pueden recibir argumentos, los datos que necesitan para llevar a cabo su cometido. Por ejemplo, una función que abre un fichero puede recibir como argumento el nombre del fichero que tiene que abrir.

Las funciones pueden devolver o no un valor. A diferencia de lo que ocurre en otros lenguajes, en PHP no es necesario declarar el tipo de dato que devuelve una función y, de hecho, puede devolver tipos de datos diferentes según el caso. Por ejemplo, en PHP hay muchas funciones que devuelven FALSE en caso de error y no devuelven un *boolean* si no hay error.

2.8.1. Funciones predefinidas

PHP cuenta con una gran cantidad de funciones predefinidas para las tareas más habituales.

Entre las más utilizadas están las relacionadas con las variables. Como en PHP las variables no se declaran explícitamente, hay una serie de funciones que permiten conocer el estado y el tipo de dato de una variable. Las más importantes son:

- `is_null($var)`. Devuelve TRUE si `$var` es NULL, FALSE en otro caso.
- `isset($var)`. Devuelve TRUE si `$var` ha sido inicializada y su valor no es NULL, FALSE en otro caso.
- `unset($var)`. Elimina la variable. Ya no contará como inicializada.
- `empty($var)`. Devuelve TRUE si `$var` no ha sido inicializada o su valor es FALSE, FALSE en otro caso.
- `is_int($var)`, `is_float($var)`, `is_bool($var)`, `is_array($var)`. Devuelven TRUE si `$var` es entero, *float*, *booleano* o *array* respectivamente, y FALSE en otro caso.
- `print_r($var)` y `var_dump($var)`. Muestran información sobre `$var`.

En el ejemplo **funciones_variables.php** se pueden ver las diferencias entre estas funciones.

```
<?php
    $var1 = 4;
    $var2 = NULL;
    $var3 = FALSE;
    $var4 = 0;
    echo "var 1";
    var_dump(isset($var1)); // TRUE
    var_dump(is_null($var1)); // FALSE
    var_dump(empty($var1)); // FALSE
    echo "var 2";
    var_dump(isset($var2)); // FALSE
    var_dump(is_null($var2)); // TRUE
    var_dump(empty($var2)); // TRUE
    echo "var 3";
    var_dump(isset($var3)); // TRUE
    var_dump(is_null($var3)); // FALSE
    var_dump(empty($var3)); // TRUE
    echo "var 4";
```


Escaneado con CamScanner

```
var_dump(empty($var4)); // TRUE, EL 0 COMO BOOLEAN ES FALSE
echo "unset";
unset($var1);
var_dump(isset($var1)); // FALSE
```

A lo largo de los siguientes capítulos se irán presentando más funciones según sean necesarias. En el cuadro 2.6 se resumen algunas de las más útiles.

CUADRO 2.6
Algunas funciones útiles en PHP

Funciones de variables	
isset($var)	TRUE si la variable está inicializada y no es NULL
is_null($var)	TRUE si la variable es NULL
empty($var)	TRUE si la variable no está inicializada o su valor es FALSE
is_int($var), is_float($var), is_bool($var), is_array($var)	Para comprobar el tipo de dato de $var
intval($var), floatval($var), boolvar($var), strval($var)	Para obtener el valor de $var como otro tipo de dato
Funciones de cadenas	
strlen($cad)	Devuelve la longitud de $cad
explode($cad, $token)	Parte una cadena utilizando $token como separador. Devuelve un *array* de cadenas
implode($token, $array)	Crea una cadena larga a partir de un *array* de cadenas, entre cadena y cadena se introduce $token
strcmp($cad1, $cad2)	Compara las dos cadenas. Devuelve 0 si son iguales, -1 si $cad1 es menor y 1si $cad1 es mayor
strtolower($cad), strtoupper($cad)	Devuelven $cad en mayúsculas o minúsculas, respectivamente
str($cad1, $cad2)	Busca la primera ocurrencia de $cad2 en $cad1. Si no aparece devuelve FALSE, si aparece devuelve $cad1 desde donde comienza la ocurrencia
Funciones de *arrays*	
ksort($arr), krsort($arr)	Ordena el *array* por clave en orden ascendente o descendente
sort($arr), rsort($arr)	Ordena el *array* por valor en orden ascendente o descendente
array_values($arr)	Devuelve los valores de $arr
array_keys($arr)	Devuelve las claves de $arr
array_key_exists($arr, $cla)	Devuelve verdadero si algún elemento de $arr tiene clave $cla
count($arr)	Devuelve el número de elementos del *array*

Escaneado con CamScanner

2.8.2. Funciones definidas por el usuario

La sintaxis para definir una función es:

```
function nombre(argumentos){
    <instrucciones>
}
```

La función puede tener o no argumentos. Se declaran en la cabecera de la función, entre paréntesis, separados por comas. Los argumentos son variables locales a la función y solo existen mientras se esté ejecutando. En PHP no hace falta declarar el tipo de dato que devuelve la función. De hecho, es posible que una función devuelva tipos de datos diferentes según el caso.

Por ejemplo, la siguiente función recibe dos números y devuelve su suma.

```
<?php
    function suma($a, $b)  {
        return $a + $b;
    }
    echo suma(4,8).'<br>';
    $var1 = 35;
    $var2 = 5;
    $var3 = suma($var1, $var2);
    echo $var3.'<br>';
```

Es posible especificar valores por defecto para los argumentos. Si al llamar a la función no se usa el argumento, se toma el valor por defecto.

```
<?php
    function saludar($nombre = 'usuario'){
        echo "Hola $nombre <br>";
    }
    saludar();
    saludar("Ana");
```

La salida será:

```
Hola usuario
Hola Ana
```

Actividad propuesta 2.2

Escribe una función para calcular potencias. Recibirá como argumentos la base y el exponente, que es opcional y tiene valor por defecto 2 (elevar al cuadrado).

Escaneado con CamScanner

2.8.3. Paso de argumentos por copia y por valor

En PHP los argumentos se pasan por copia. Esto quiere decir que, cuando se llama a una función con una variable como argumento, se crea una variable local a la función en la que se copia el valor del argumento. Por tanto, si la función modifica el argumento, estos cambios no tienen efecto en la variable original.

Esto es lo que ocurre en la función `duplicarMal()` del siguiente ejemplo. El argumento se multiplica por dos, pero el valor de $var1 no cambia. Para solucionarlo hay dos opciones. Devolver el nuevo valor y reasignar, como se hace con la función `duplicar()`, o utilizar una referencia, como se hace en `duplicar2()`. Para utilizar una referencia solo hace falta añadir el símbolo "&" antes del argumento.

```
<?php
    function duplicarMal($a){
        $a = $a *2;
    }
    function duplicar($a){
        return $a *2;
    }
    function duplicar2(&$a){
        $a = $a *2;
    }
    $var1 = 5;
    duplicarMal($var1);
    echo "$var1 <br>";
    $var1 = duplicar($var1);
    echo "$var1 <br>";
    duplicar2($var1);
    echo "$var1 <br>";
```

La salida del programa será:

```
5
10
20
```

2.8.4. Funciones como argumentos

En PHP es posible pasar funciones como argumentos a otras funciones. Es una característica avanzada que se utiliza entre otras cosas para las funciones de *callback*. Solo hay que pasar el nombre de la función entre comillas como argumento. La función que lo recibe podrá utilizarla si le pasan los argumentos adecuados.

En el ejemplo **calculador.php** se utiliza esta característica. La función `calculador()` recibe como argumentos dos números y también la función que debe aplicarles. Según qué función se le pase como argumento, devolverá un valor u otro.

```
<?php
    function calculador($operacion, $numa, $numb){
        $resul = $operacion($numa, $numb);
```


Escaneado con CamScanner

```
    return $resul;
}
function sumar($a, $b){
    return $a + $b;
}
function multiplicar($a, $b){
    return $a * $b;
}
$a = 4;
$b = 5;
$r1 = calculador("multiplicar", $a, $b);
echo "$r1 <br>";
$r2 = calculador("sumar", $a, $b);
echo "$r2 <br>";
```

Actividad propuesta 2.3

Escribe una función para calcular el factorial de un número, que recibirá como argumento. Devolverá el factorial o -1 si el argumento no es válido.

2.9. Excepciones y errores

El sistema de control de errores de PHP ha ido evolucionando a lo largo de las versiones. E el sistema básico, se generan errores de diferentes tipos, representados por un número. Por otr lado, desde PHP 5 hay un sistema de excepciones similar al de Java y otros lenguajes utilizand bloques `try/catch/finally`. Finalmente, en PHP 7 aparecieron las excepciones de clase Erro

2.9.1. Errores

En el sistema básico, ante determinadas condiciones (por ejemplo, utilizar una variable no inicializada) PHP genera un error. Hay diferentes tipos de errores, cada uno asociado con un número y una constante predefinida. Se puede controlar cómo se comporta PHP antes los errores mediante tres directivas del fichero php.ini:

- *error_reporting*: indica qué errores deben reportarse. Lo normal es utilizar E_ALL, es decir, todos.
- *display_errors*: señala si los mensajes de error deben aparecer en la salida del *script*. Esta opción es apropiada durante el desarrollo, pero no en producción.
- *log_errors*: indica si los mensajes de error deben almacenarse en un fichero. Es especialmente útil en producción, cuando no se muestran los errores en la salida.
- *error_log*: si la directiva anterior está activada, es la ruta en la que se guardan los mensajes de error.

Escaneado con CamScanner

El valor de la directiva *error_reporting* es un número, pero para especificarlo lo habitual es utilizar las constantes predefinidas y el operador or a nivel de bit.

CUADRO 2.7
Tipos de error

Código	Constante	Descripción
1	E_ERROR	Error fatal en tiempo de ejecución. La ejecución del *script* se detiene
2	E_WARNING	Advertencia en tiempo de ejecución. El *script* no se detiene
4	E_PARSE	Error de sintaxis al compilar
8	E_NOTICE	Notificación. Puede indicar error o no
16	E_CORE_ERROR	Error fatal al iniciar PHP
32	E_CORE_WARNING	Advertencia al iniciar PHP
64	E_COMPILE_ERROR	Error fatal al compilar
128	E_COMPILE_WARNING	Advertencia fatal al compilar
256	E_USER_ERROR	Error generado por el usuario
512	E_USER_WARNING	Advertencia generada por el usuario
1024	E_USER_NOTICE	Notificación generada por el usuario
2048	E_STRICT	Sugerencias para mejorar la portabilidad
4096	E_RECOVERABLE_ERROR	Error fatal capturable
8192	E_DEPRECATED	Advertencia de código obsoleto
16384	E_USER_DEPRECATED	Como la anterior, generada por el usuario
32767	E_ALL	Todos los errores



Actividad propuesta 2.4

Modifica el fichero php.ini para que los errores E_NOTICE no se muestren por pantalla (display_errors).

Comprueba la diferencia accediendo a localhost/cap2/no_init.php.

- **Funciones relacionadas**

La función `error_reporting()` permite cambiar el valor de la directiva *error_reporting* en tiempo de ejecución.

Escaneado con CamScanner

También es posible definir una función propia para que se encargue de los errores utilizando `set_error_handler()`. La función que se ocupe de los errores tendrá que tener la siguiente firma:

bool handler (int $errno, string $errstr [, string $errfile [, int $errline [, array $errcontext]]]);

El siguiente ejemplo muestra cómo utilizar `set_error_handler()` para manejar los errores con una función propia.

```
<?php
    function manejadorErrores($errno, $str, $file, $line){
        echo "Ocurrió el error: $errno";
    }
    set_error_handler("manejadorErrores");
    $a = $b; // causa error, $b no está inicializada
```

La salida será:

Ocurrió el error: 8

2.9.2. Excepciones

Otra opción para indicar un error es lanzar una excepción. Para controlar las excepciones se utilizan bloques `try/catch/finally`, como en Java. Cuando se lanza una excepción y no es capturada por un bloque `catch`, la ejecución del programa se detiene. Si es capturada, se ejecuta el código del bloque correspondiente.

Para capturar una excepción se introduce la instrucción que puede causarla dentro de un bloque `try` y se añade el bloque `catch` correspondiente. Se puede añadir un bloque `finally`, que se ejecuta después del `try/catch`, haya habido excepción o no.

```
try{
    instrucciones;
}catch(Exception e){
    instrucciones;
}finally{
    instrucciones;
}
```

En el ejemplo **excepciones.php** se utiliza una función que recibe dos argumentos y lanza una excepción si el segundo es cero. Para lanzar una excepción se utiliza `throw`, que recibe como argumento un objeto de clase `Exception` o de alguna subclase.

```
1  <?php
2      function dividir($a, $b){
3          if ($b==0){
4              throw new Exception('El segundo argumento es 0');
5          }
```


Escaneado con CamScanner

```
6           return $a/$b;
7       }
8       try{
9           $result1 = dividir(5, 0);
10          echo "Resul 1 $result1". "<br>";
11      }catch(Exception e){
12          echo "Excepción: ". $e->getMessage(). "<br>";
13      }finally{
14          echo "Primer finally";
15      }
16      try{
17          $resul2 = dividir(5, 2);
18          echo "Resul 2 $resul2". "<br>";
19      }catch(Exception e){
20          echo "Excepción: ". $e->getMessage(). "<br>";
21      }finally{
22          echo "Segundo finally";
23      }
```

En la primera llamada a `dividir()`, línea 9, se produce una excepción. Por tanto, el resto del bloque `try`, el `echo`, no se ejecuta. Sí se ejecutan el `catch` y el `finally` correspondientes.

En la segunda llamada a `dividir()`, línea 17, no se produce excepción. En este caso se ejecutan el `echo` del `try` y a continuación el del bloque `finally`. El bloque `catch` no se ejecuta.

Actividad propuesta 2.5

Adapta la actividad 2.3 para que controle si el argumento es negativo utilizando una excepción.

2.9.3. Excepciones Error

En PHP 7 aparecieron las excepciones de tipo Error. No heredan de la clase Exception, así que para capturarlas hay que usar:

```
catch (Error $e){
    ...
}
```

o, alternativamente, la clase Throwable, de la que se derivan tanto Error como Exception:

```
catch (Throwable $e){
    ...
}
```


Escaneado con CamScanner

En el cuadro 2.8. se muestran las excepciones Error predefinidas.

CUADRO 2.8
Excepciones Error

Nombre	Descripción	Hereda de
Error	Clase base para las excepciones Error	
ArithmeticError	Error en operaciones matemáticas. Hereda del anterior	Error
DivisionByZeroError	Intento de división por cero. Hereda del anterior	ArithmeticError
AssertionError	Ocurre cuando falla una llamada a assert()	Error
ParseError	Error al compilar	Error
TypeError	Ocurre cuando una expresión no tiene el tipo de dato que se espera	Error
ArgumentCountError	Ocurre al llamar a una función con menos argumentos de los necesarios. Hereda del anterior	TypeError

2.10. Clases y objetos

PHP tiene soporte completo para la programación orientada a objetos. Permite definir clas herencia, interfaces y los demás elementos habituales. Para declarar una clase se utiliza la palab reservada `class`. Los atributos se declaran utilizando su nombre, los modificadores que pue tener y opcionalmente un valor por defecto.

```
class Clase{
    private $att1 = 10; // con valor por defecto
    private $atr2; // sin valor por defecto
    private static $atr3 = 0; // estático
    ...
}
```

Los atributos y métodos se pueden declarar como estáticos, de manera que no habrá uno por objeto, sino uno por clase. Se utiliza la palabra reservada `static`.

Se puede crear un constructor para la clase con el método `__construct()`. Este método forma parte de los *métodos mágicos*, nombres reservados a tareas concretas y que comienzan por dos guiones bajos, "__".

Para crear un nuevo objeto se utiliza el operador new seguido del nombre de la clase:

```
$obj = new Clase();
```

Para los atributos y métodos se pueden utilizar los siguientes modificadores de visibilidad:

- `public`. Se pueden utilizar desde dentro y fuera de la clase.
- `private`. Pueden emplearse desde la propia clase.
- `protected`. Se pueden utilizar dentro de la propia clase, las derivadas y las antecesoras.

Escaneado con CamScanner

Normalmente los atributos se declaran como privados y se crean métodos públicos para acceder a ellos. Para acceder a los métodos y atributos se utiliza:

```
$objeto->propiedad;
$objeto->método(argumentos);
```

Los métodos son funciones definidas dentro de una clase. En los métodos no estáticos se puede utilizar la palabra reservada `this`, que representa el objeto desde el que se invoca el método.

El ejemplo **PersonayCliente.php** muestra cómo crear una clase y crear y manejar objetos de esta. Para empezar, declara la clase Persona, con atributos para nombre, apellido y DNI. Los tres son privados. La clase tiene un constructor, el método `__construct()`, que recibe valores para los tres atributos. A continuación, se declaran una serie de métodos públicos para leer y escribir los atributos, los llamados *getters* y *setters* (líneas 11-23).

```
1  <?php
2  class Persona {
3      private $DNI;
4      private $nombre;
5      private $apellido;
6      function __construct($DNI, $nombre, $apellido) {
7          $this->DNI = $DNI;
8          $this->nombre = $nombre;
9          $this->apellido = $apellido;
10     }
11     public function getNombre() {
12         return $this->nombre;
13     }
14     public function getApellido() {
15         return $this->apellido;
16     }
17     public function setNombre($nombre) {
18         $this->nombre = $nombre;
19     }
20
21     public function setApellido($apellido) {
22         $this->apellido = $apellido;
23     }
24     public function __toString() {
25         return "Persona: ".$this->nombre." ".$this->apellido;
26     }
27 }
```

En esta clase también se utiliza `__toString()` (líneas 24-26), otro método mágico que se usa para generar una cadena de texto con la información del objeto. Cuando se pasa un objeto a `echo`, se representa usando el valor de retorno de `__toString()`.

En las líneas 46-52 se crea un objeto de la clase Persona y se manipula.

```
45 // crear una persona
46 $per = new Persona("1111111A", "Ana", "Puertas");
47 // mostrarla, usa el método __toString()
48 echo $per. "<br>";
49 // cambiar el apellido
50 $per->setApellido("Montes");
```


Escaneado con CamScanner

```
51 // volver a mostrar
52 echo $per. "<br>";
```

La salida de este fragmento será:

Persona: Ana Puertas
Persona: Ana Montes

Para crear una clase que herede de otra, se utiliza la palabra clave **extends**. La clase derivada tendrá los mismos atributos y métodos que la clase base y podrá añadir nuevos o sobrescribirlos.

En el ejemplo anterior se declara también la clase Cliente, que hereda de persona y añade un atributo saldo. Incluye también los métodos `getSaldo()` y `setSaldo()` y sobrescribe el método `__toString()` de la clase base. También hay un constructor que incluye el nuevo atributo y utiliza el constructor de la clase base.

```
class Cliente extends Persona{
    private $saldo = 0;
    function __construct($DNI, $nombre, $apellido, $saldo){
        parent::__construct($DNI, $nombre, $apellido);
        $this->$saldo = $saldo;
    }
    public function getSaldo(){
        return $this->saldo;
    }
    public function setSaldo($saldo){
        $this->saldo = $saldo;
    }
    public function __toString(){
        return "Cliente: ". $this->getNombre();
    }
}
```

En las líneas 54–56 se crea un objeto de la clase cliente y se muestra. La salida será "Cliente: Pedro".

```
$cli = new Cliente("22222245A", "Pedro", "Sales", 100);
// lo muestra
echo $cli. "<br>";
```

Resumen

- PHP es el lenguaje más extendido para el desarrollo web en el lado del servidor.
- Los ficheros de PHP mezclan HTML y PHP. El servidor sustituye los bloques de PHP por su salida.
- En PHP no hace falta declarar las variables, se declaran la primera vez que se usan.
- Tampoco es necesario definir su tipo de dato, se infiere del valor de inicialización.

Escaneado con CamScanner

- El paso de parámetros y la asignación se realizan por defecto por copia, pero se pueden usar referencias.
- Las funciones de PHP pueden ser parámetros de otras funciones. Es una característica muy usada en las librerías PHP.
- PHP cuenta con muchas funciones integradas para las tareas más habituales: manipulación de cadenas, *arrays*, bases de datos, ficheros...
- Los *arrays* de PHP son un tipo de dato muy potente similar a los que en otros lenguajes se llaman *mapas* o *diccionarios*.
- Hay tres sistemas de control de errores en PHP: errores, excepciones y excepciones de clase Error.
- PHP tiene soporte completo para la programación orientada a objetos.

Ejercicios propuestos

1. Escribe un *script* para resolver ecuaciones de segundo grado, ax2 + bx + c = 0. Si la ecuación no tiene soluciones reales, hay que mostrar un mensaje de error.
2. Crea una función para resolver la ecuación de segundo grado. Esta función recibe los coeficientes de la ecuación y devuelve un *array* con las soluciones. Si no hay soluciones reales, devuelve FALSE.
3. Almacena la función anterior en el fichero **matemáticas.php**. Crea un fichero que la incluya y la utilice.
4. Escribe una función que reciba una cadena y comprueba si es un palíndromo.
5. Escribe una función que reciba un *array* de números, y un número, el límite. La función tiene que devolver un nuevo *array* que contenga solo los elementos del *array* original menores que el límite.
6. Escribe un *script* para probar las funciones del cuadro 2.6.

Actividades de autoevaluación

1. En PHP, las variables:
   - ☐ *a)* No tienen tipo de dato.
   - ☐ *b)* Tienen un tipo de dato que puede cambiar.
   - ☐ *c)* Tienen un tipo de dato que no puede cambiar.

2. En PHP, las funciones:
   - ☐ *a)* Tienen que declarar el tipo de dato que devuelven.
   - ☐ *b)* Pueden devolver diferentes tipos de datos según el caso.
   - ☐ *c)* No tienen que declarar el tipo de dato que devuelven, pero siempre tienen que devolver el mismo.

Escaneado con CamScanner

3. En los *arrays* de PHP, si no se asigna clave al insertar un elemento:
   - ☐ *a)* Da error.
   - ☐ *b)* Se inserta sin clave.
   - ☐ *c)* Se le asigna una clave numérica.

4. El orden de los elementos dentro de un *array* está determinado por:
   - ☐ *a)* Su clave.
   - ☐ *b)* Su valor.
   - ☐ *c)* El orden de inserción.

5. Al utilizar una variable no inicializada:
   - ☐ *a)* Se genera un error de tipo E_NOTICE y la ejecución del *script* se detiene.
   - ☐ *b)* Se genera un error de tipo E_NOTICE y la ejecución del *script* continúa.
   - ☐ *c)* No se genera ningún error.

6. La salida de estas líneas de código es:

```
$arr = array(2,2,2,2);
foreach($arr as $elemento){
    $elemento = $elemento *2;
}
foreach($arr as $elemento){
    echo $elemento. " ";
}
```

   - ☐ *a)* 2 2 2 2.
   - ☐ *b)* 4 4 4 4.
   - ☐ *c)* Hay un error.

7. La salida de estas líneas de código es:

```
$a = 4;
$b = &$a;
$a = 6;
echo $b;
```

   - ☐ *a)* 4
   - ☐ *b)* 6
   - ☐ *c)* 8.

8. En PHP, los argumentos:
   - ☐ *a)* Se pasan por copia.
   - ☐ *b)* Se pasan por valor.
   - ☐ *c)* Se pueden pasar por copia o por valor.

9. Sobre los bloques de PHP:
   - ☐ *a)* Se deben cerrar siempre.
   - ☐ *b)* Si el fichero acaba con un bloque, no se cierra.
   - ☐ *c)* Si el fichero contiene solo bloques de PHP, el último no se cierra.

Escaneado con CamScanner

10. Dos *arrays* son idénticos si:

- ☐ *a)* Todos los elementos tienen el mismo valor y están en el mismo orden.
- ☐ *b)* Todos los elementos son iguales en clave y valor y además están en el mismo orden.
- ☐ *c)* Todos los elementos son iguales en clave y valor.

SOLUCIONES:

1.	a **b** c	**5.**	a **b** c
2.	a **b** c	**6.**	**a** b c
3.	a b **c**	**7.**	a **b** c
4.	a b **c**	**8.**	a b **c**
9.	a b **c**		
10.	a **b** c		

Escaneado con CamScanner

Escaneado con CamScanner

3

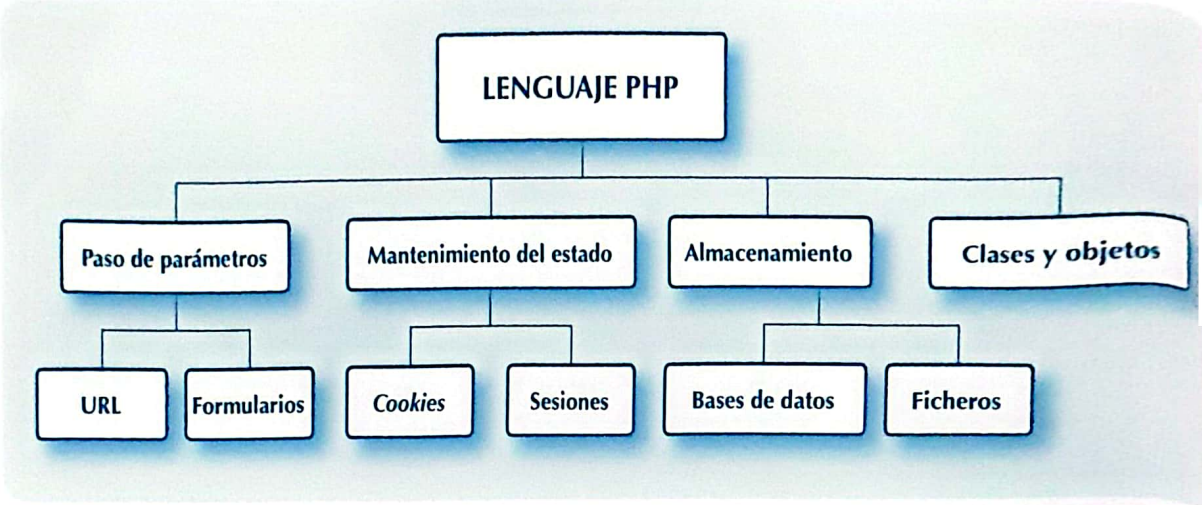
Desarrollo de aplicaciones web con PHP

Objetivos

- ✓ Conocer los mecanismos de paso de parámetros a un *script*.
- ✓ Procesar y validar formularios.
- ✓ Utilizar *cookies*.
- ✓ Aprender a emplear las sesiones y las variables de sesión.
- ✓ Manejar bases de datos.
- ✓ Enviar correos desde PHP.
- ✓ Dominar las herramientas de pruebas y depuración.

Escaneado con CamScanner

Mapa conceptual



Glosario

Base de datos no relacional. Sistema gestor de bases de datos que no sigue el modelo relacional. Por ejemplo, bases de datos XML.

Cookie. Fichero que almacenan los servidores en los clientes. Puede almacenar información sobre la última visita o las preferencias del usuario, entre otras cosas.

Formulario. Elemento que permite al usuario introducir datos que se envían servidor.

Sesión. Las sesiones permiten que las páginas de una misma aplicación compartan información.

XML. *eXtensible Markup Languange*, "lenguaje de marcas extensibles". Es un estándar del W3C.

XPath. El lenguaje XPath se utiliza para buscar información en ficheros XML. Es un estándar del W3C.

XSD. Lenguaje basado en XML para validar documentos XML. Validar consiste en verificar que un fichero tiene una estructura determinada. Es un estándar del W3C.

XSL/XSLT. Lenguaje basado en XML para realizar transformaciones sobre documentos XML. El resultado puede ser XML, HTML... Es un estándar del W3C.

3.1. Paso de parámetros

El protocolo HTPP define ocho métodos o verbos para establecer una comunicación entre cliente y servidor. Los más habituales son GET (obtener) y POST (enviar). El método GET es el que se utiliza habitualmente para solicitar páginas web a un servidor, por ejemplo, al seguir

Escaneado con CamScanner

un vínculo o introducir una dirección a mano en la barra del navegador. El método POST se utiliza sobre todo para enviar formularios al servidor.

Cuando se usa el método GET se pueden pasar parámetros al servidor en la URL. A la ruta normal para acceder a una página se le añade el carácter "?"como indicador de que empieza la lista de parámetros. Cada parámetro tiene un nombre, a la izquierda del igual, y un valor, a la derecha. Los argumentos están separados entre sí por el carácter *ampersand*. Por ejemplo, si se accede con el navegador a:

http://localhost/cap3/hola_nombre.php?nombre=Ana

Se solicita al servidor el fichero **/cap3/hola_nombre.php** del servidor *localhost* y se le pasa un parámetro llamado "nombre" con valor "Ana".

Con la siguiente URL se añadiría otro parámetro para el apellido:

http://localhost/cap3/hola_nombre.php?nombre=Ana&apellido=Luna

Se puede acceder a los argumentos de la URL desde un bloque PHP usando el *array* superglobal `$_GET`, que tiene un elemento por cada argumento presente en la URL. El nombre del argumento será la clave del elemento del *array*.

El fichero **hola_nombre.php** muestra un mensaje personalizado usando el valor del parámetro nombre.

```
<?php
    echo "Hola ". $_GET["nombre"];
```

Si se accede con la ruta

http://localhost/cap3/hola_nombre.php?nombre=Ana,

se obtendrá el mensaje "Hola Ana".

Si se accede con

http://localhost/cap3/hola_nombre.php,

se obtendrá:

Notice: Undefined index: nombre in C:\xampp\htdocs\cap3\eje3_1.php on line 2 Hola

Para controlar si los parámetros se han pasado correctamente se pueden utilizar las funciones `empty()` o `is_null()`. Las dos devuelven TRUE cuando el parámetro no está presente en la URL. La diferencia está en los parámetros presentes, pero sin valor, por ejemplo:

http://localhost/cap3/hola_nombre.php?nombre

En este caso, `empty($_GET["nombre"])` devuelve TRUE, pero `is_null(($_GET["nombre"])` devuelve FALSE.

El ejemplo **hola_comprobacion.php** mejora el anterior para mostrar un mensaje de error si no se pasa el parámetro nombre.

Escaneado con CamScanner

```php
<?php
    if (empty($_GET["nombre"])) {
        echo "Error, falta el parámetro nombre";
    }else {
        echo "Hola ". $_GET["nombre"];
    }
```

Actividad propuesta 3.1

Escribe un fichero que reciba dos parámetros, num1 y num2, y muestre su suma. Hay que comprobar que los dos argumentos existan y sean números.

3.2. Formularios

Los formularios HTML son la forma más habitual de enviar datos a un servidor. Permit que el usuario rellene varios campos mediante diferentes tipos de controles (campos de tex botones de radio...) y lo envíe al servidor al pulsar un botón. El servidor procesa los datos c formulario y genera la respuesta.

Un formulario sencillo de *login* en HTML se escribiría así:

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de login</title>
        <meta charset = "UTF-8">
    </head>
    <body>
        <form action = "login_basico.php" method = "POST">
            <input name = "usuario" type = "text">
            <input name = "clave" type = "password">
            <input type = "submit">
        </form>
    </body>
</html>
```

El atributo `action` del formulario especifica la ruta del *script* al que se enviará el formulario para que lo procese. Si se usa una ruta relativa, se toma como referencia la localización en e servidor del fichero que contenga el formulario.

El atributo `method` especifica el método HTPP, que se usará para la petición. En los formularios lo habitual es utilizar POST, pero también es posible utilizar GET. Si se utiliza POST, los parámetros no aparecen en la URL.

Dentro del elemento `form` se introducen los campos, que tendrá que rellenar el usuario y los botones de envío y para limpiar los campos. Para los campos que se envían hay que usar el atributo `name`, que sirve para identificarlo dentro del *script*. El envío se produce al pulsar e botón.

Escaneado con CamScanner

En el *script* al que se envía el formulario los parámetros están disponibles en el *array* superglobal `$_POST`. La clave de cada argumento dentro del *array* es el atributo `name` del elemento correspondiente en el formulario.

El fichero **login_basico.php**, al que se envía el formulario anterior, muestra los valores introducidos en él:

```
<?php
    echo "Usuario introducido: ". $_POST['usuario']. "<br>";
    echo "Clave introducida: ". $_POST['clave'];
```

Al usar el método POST los parámetros no se muestran en la URL, pero se pueden consultar desde la consola del navegador. En Firefox está disponible en Menú > Desarrollador Web > Red. Una vez abierta, muestra las peticiones que se realizan desde el navegador. Si se selecciona la correspondiente al envío del formulario, se pueden consultar los parámetros en la parte derecha.

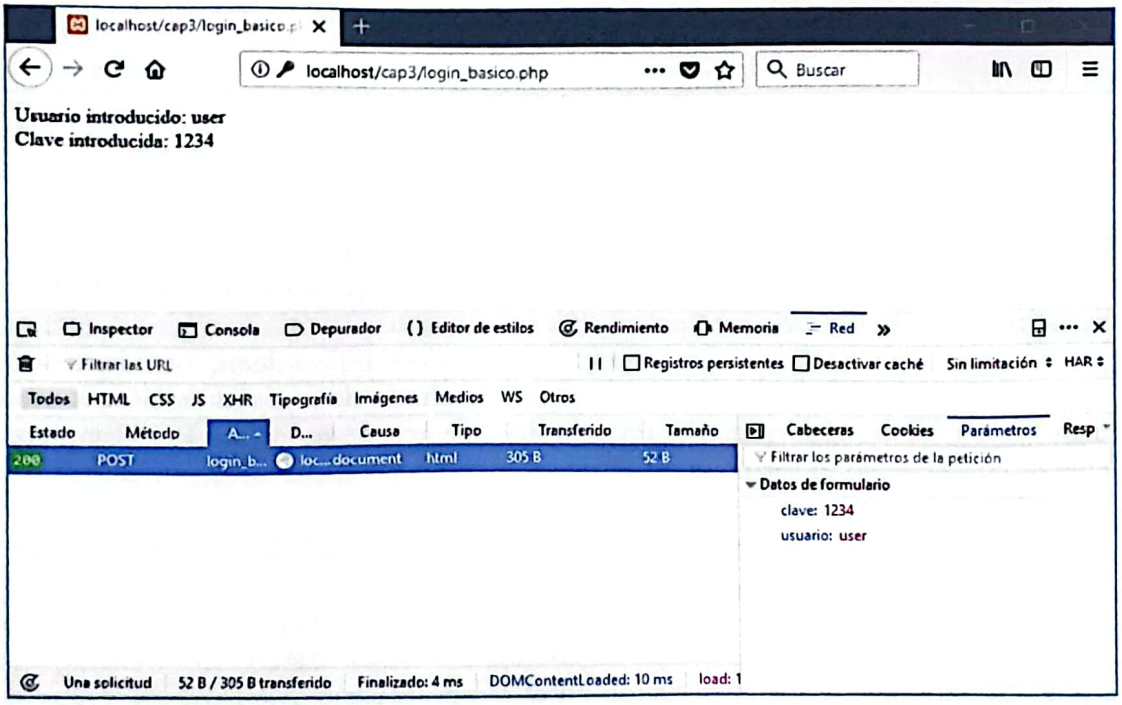


Figura 3.1
Herramientas de desarrollador en Firefox.

3.2.1. Formulario de *login*

En general un formulario de *login* se encarga de comprobar los datos introducidos y, según sean correctos o no, da acceso al sistema al usuario o muestra un mensaje de error. El fichero **login_falso.php** se encarga de procesar un formulario de *login*. Para simular un formulario de *login*, hay que comprobar que el usuario y la contraseña sean correctos. Si el usuario es "usuario" y la clave es "1234", se redirige a la página de bienvenida. En caso contrario, lo hace a una página de error. Para la redirección se usa la función `header`, que sirve para escribir en la cabecera de la respuesta HTTP.

Escaneado con CamScanner

```
<?php
/*si va bien redirige a bienvenido.html
si va mal, mensaje de error */
if ($_POST['usuario']=="usuario" and $_POST["clave"]=="1234"){
   header("Location:bienvenido.html");
} else {
   header("Location:error.html");
}
```

RECUERDA

✓ Hay que enviar las cabeceras antes de empezar con el cuerpo de la respuesta. Esto implica que hay que utilizar la función header() antes de que se empiece a escribir la salida. Si se intenta llamar a header() después de haber realizado un echo, se producirá un error.

3.2.2. Formulario y procesamiento en un solo fichero

En ocasiones el formulario HTML y el bloque PHP que lo procesa se integran en un solo fichero, en el que hay que distinguir entre dos casos. Cuando se accede al formulario para rellenarlo y cuando se envía para procesarlo.

Cuando se accede a la página usando el método GET, es decir, introduciendo la dirección en el navegador, al seguir un vínculo o como resultado de una redirección con `header(Location:)`, se muestra el formulario. En cambio, si se accede mediante POST quiere decir que el cliente está enviado el formulario. Se puede diferenciar entre los dos métodos de acceso consultando `$_SERVER["REQUEST_METHOD"]`.

El ejemplo **form_en_uno.php** une los dos ficheros del anterior en uno solo y, en caso de error:

- Muestra un mensaje apropiado encima del formulario.
- Mantiene el valor introducido en el campo usuario.

```
<?php
/* si va bien redirige a principal.php si va mal, mensaje de error */
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   if($_POST['usuario']=="usuario" and $_POST["clave"]== "1234"){
      header("Location: principal.php");
   }else{
      $err = true;
   }
}
?>
<!DOCTYPE html>
   <html>
```


Escaneado con CamScanner

```
<head>
    <title>Formulario de login</title>
    <meta charset = "UTF-8">
</head>
<body>
    <?php if(isset($err)){
        echo "<p> Revise usuario y contraseña</p>";
    }?>
    <form method = "POST"
        action="<?php echohtmlspecialchars($_SERVER["PHP_SELF"]);?>">
        <label for = "usuario">Usuario</label>
        <input value = "<?php if(isset($usuario))echo $usuario;?>"
        id = "usuario" name = "usuario" type = "text">
        <label for = "clave">Clave</label>
        <input id = "clave" name = "clave" type = "password">
        <input type = "submit">
    </form>
</body>
</html>
```

TOMA NOTA

Cuando el formulario llama al mismo fichero se recomienda usar:

action = "<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"

en lugar del nombre del fichero como aparece en el ejemplo. La variable *superglobal* $_SERVER["PHP_SELF"]contiene el nombre del fichero y la función htmlspecialchars() sirve para filtrar los caracteres por seguridad.

El bloque inicial de PHP con la comprobación de usuario y contraseña se ejecuta solo cuando se accede por el método POST, es decir, al enviar el formulario.

En ese caso, se comprueban los datos y, si son correctos, se reenvía a la página de bienvenida. Con el reenvío termina la ejecución de **form_en_uno.php**. Si no son correctos, se crea la variable `$err` con valor TRUE y el *script* continúa. El bloque de HTML que contiene el formulario se mostrará tanto si el método es GET como si es POST y la comprobación de datos falló.

Hay un segundo bloque de PHP antes del formulario para imprimir un mensaje de error si la variable `$err` existe, es decir, si se ha enviado el formulario con datos incorrectos. Si se accede por GET el bloque se ejecuta, pero la condición no se cumple y no se muestra el mensaje.

3.2.3. Subida de ficheros

Un caso especial en los formularios es la subida de ficheros al servidor. En el formulario hay que usar el atributo `enctype="multipart/form-data"` y el método POST. Para el fichero se usa una etiqueta `<input type = "file">`. Con este control se abre una ventana para que el usuario pueda escoger un fichero en su equipo.

Escaneado con CamScanner

```
<form action="procesar_subida.php" method="post"
    enctype = "multipart/form-data">
    Escoja un fichero
    <input type="file" name="fichero">
    <input type="submit" value="Subir fichero">
</form>
```

En el *script* que recibe el formulario, la variable global `$_FILES` contiene información sobre el fichero que se está subiendo. Se trata de un *array* bidimensional. La primera dimensión identifica el fichero según el atributo `name` en el formulario, en la segunda las claves son:

- `name`: el nombre del fichero en el cliente.
- `size`: el tamaño del fichero en *bytes*.
- `type`: el tipo MIME del fichero.
- `tmp_name`: el nombre temporal con el que se ha subido al servidor.
- `error`: código de error asociado a la subida.

El fichero se almacena en principio en el directorio temporal del servidor y se puede mover al directorio que se desee con la función:

```
bool move_uploaded_file ( $fichero, $destino )
```

Si el fichero no se mueve del directorio temporal, el servidor se encargará de eliminarlo.

El ejemplo **procesar_subida.php** se encarga de comprobar que el fichero no pase cierto límite de tamaño. Si se cumple la condición, se mueve al directorio "subidos" (hay que crear el directorio para que el ejercicio funcione).

```
<?php
$tam = $_FILES["fichero"]["size"];
if($tam > 256 *1024){
    echo "<br>Demasiado grande";
    return;
}
echo "Nombre del fichero: ". $_FILES["fichero"]["name"];
echo "<br>Nombre temporal del fichero en el servidor: ".
    $_FILES["fichero"]["tmp_name"];
$res = move_uploaded_file($_FILES["fichero"]["tmp_name"],
    "subidos/".$_FILES["fichero"]["name"]);
if($res){
    echo "<br>Fichero guardado";
} else {
    echo "<br>Error";
}
```

El formulario de envío sería el siguiente:

Escaneado con CamScanner

```
<!DOCTYPE html>
<html>
        <body>
            <form action="procesar_subida.php" method="post"
            enctype="multipart/form-data">
                    Escoja un fichero
                    <input type="file" name="fichero">
                    <input type="submit" value="Subir fichero">
            </form>
        </body>
</html>
```

3.3. Cookies

Las *cookies* son pequeños ficheros que dejan los servidores web en los ordenadores de los clientes. Pueden almacenar información sobre la fecha de la última visita o preferencias de idioma, por ejemplo. Cuando un cliente realiza una petición web, envía al servidor las *cookies* que pudiera tener de este.

Para manejar las *cookies* se usa la función `setcookie()`, que tiene la siguiente cabecera:

bool setcookie (string $name [, string $value = "" [, int $expire = 0 [, string $path = "" [, string $domain = "" [, bool $secure = FALSE [, bool $httponly = FALSE]]]]]])

Los tres primeros argumentos son los más importantes:

1. El primer argumento es el nombre de la *cookie*.
2. El segundo, el valor que se le quiere dar.
3. El tercero es la fecha en la que expira la *cookie*. Se especifica como una fecha Unix, es decir, el número de segundos pasados desde el comienzo de 1970. Normalmente se utiliza la función `time()`, que devuelve la fecha actual y se le suma un periodo de tiempo expresado en segundos.

Por ejemplo, para crear la *cookie* "visitas" con valor "1" y duración de un día se utiliza:

```
setcookie('visitas', '1', time() + 3600 * 24);
```

Para la fecha de caducidad se toma la actual, que es el valor que devuelve `time()` y se le suma 3.600 * 24, el número de segundo que hay en 24 horas.

Para destruir una *cookie* se usa la función `setcookie()` con una fecha límite anterior a la actual:

```
setcookie('visitas', '1', time() - 3600 * 24);
```

RECUERDA

✓ Las *cookies* se envían como cabeceras de las peticiones HTTP. Hay que enviar las cabeceras antes de empezar con el cuerpo de la respuesta. Esto implica que hay que utilizar la función `setcookie()` antes de que se empiece a escribir la salida. Si se intenta llamar a `setcookie()` después de haber realizado un `echo`, se producirá un error.

Escaneado con CamScanner

Las *cookies* que envía el cliente están disponibles en `$_COOKIES`, tomando como clave el nombre que se les dio con `setcookie()`. Es importante señalar que, después de crear una *cookie*, esta no está disponible en `$_COOKIES` hasta la siguiente petición del cliente. El siguiente ejemplo daría error:

```
<?php
setcookie('nueva', "valor", time() + 3600 * 24);
echo $_COOKIES["nueva"];
```

El ejemplo **contador_visitas.php** utiliza una *cookie* para almacenar el número de veces que un usuario ha visitado la página. Si la *cookie* no existe, la crea con valor "1". Si ya existe, lee su valor para mostrar el mensaje y reescribe la *cookie* sumando 1 al valor. Como el valor es una cadena, hay que transformarlo a entero.

```
<?php
    if (!isset($_COOKIE['visitas'])) { // si no existe
        setcookie('visitas', '1', time() + 3600 * 24);
        echo "Bienvenido por primera vez";
    } else { // si existe
        $visitas = (int) $_COOKIE['visitas'];
        $visitas++; // se reescribe incrementada
        setcookie('visitas', $visitas, time() + 3600 * 24);
        echo "Bienvenido por $visitas vez";
    }
```

La primera vez que se acceda al *script* se obtendrá la salida "Bienvenido por primera vez". La siguiente, "Bienvenido por 2 vez", y así sucesivamente.

Actividad propuesta 3.2

Añade un vínculo para borrar la *cookie* al ejemplo anterior.

Los navegadores ofrecen la posibilidad de ver las *cookies* almacenadas en el ordenador. En Firefox se accede a ellas mediante el menú Opciones, en la sección Privacidad y Seguridad. Hay un apartado

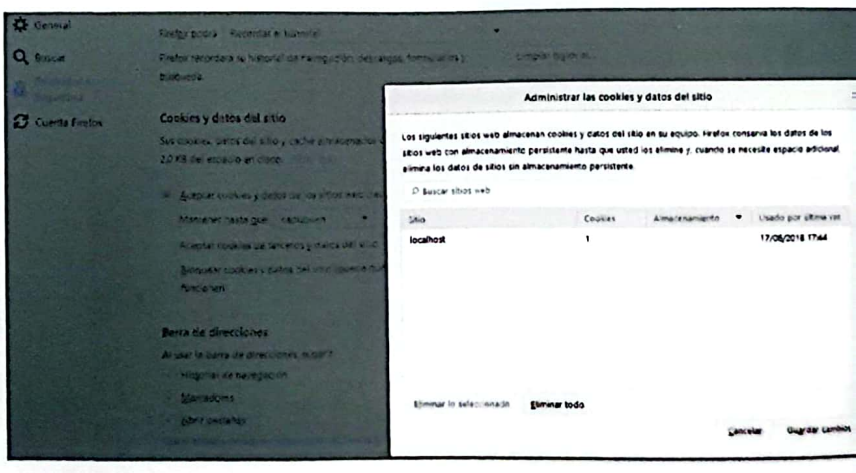


Figura 3.2
Administrar *cookies* en Firefox.

Escaneado con CamScanner

llamado "Cookies y datos del sitio" con un botón "Administrar datos". Al pulsarlo se muestra la lista de *cookies* agrupadas por el servidor de origen. Después de acceder al ejemplo **contador_visitas.php**, habrá una *cookie* del servidor *localhost*. También es posible eliminar las *cookies*, o borrar el historial de navegación.

Desde las herramientas de desarrollador se pueden ver las *cookies* que envía el cliente y la que crea el servidor con el valor incrementado en uno:

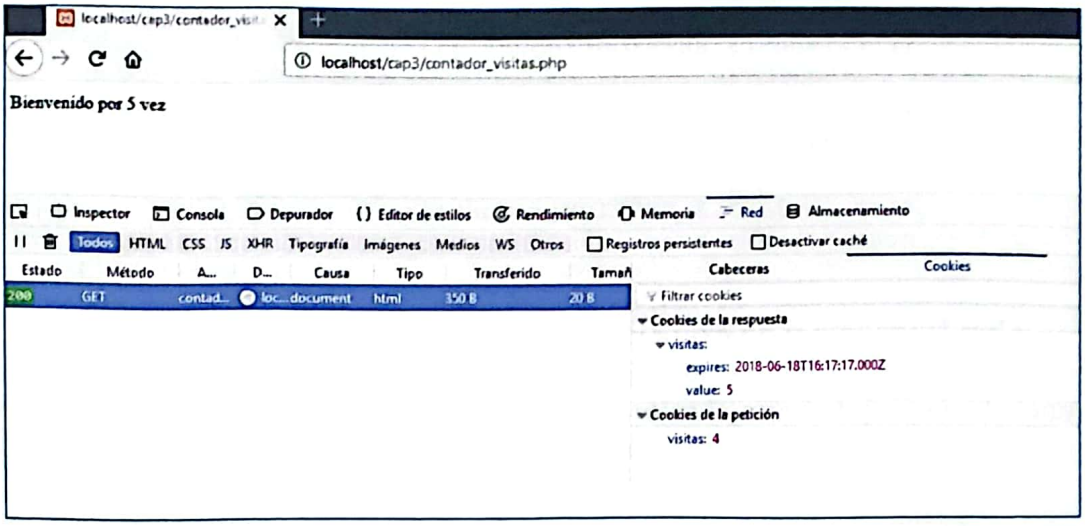


Figura 3.3
Cookies desde las herramientas de desarrollador.

Actividad propuesta 3.3

Crea una página web con un formulario para elegir el idioma en el que se muestra, inglés o español. Almacena la elección del usuario con una *cookie* para que la siguiente vez que el usuario se conecte la página aparezca directamente en su idioma. Si la *cookie* no existe, la página se mostrará en español.

3.4. Sesiones. Seguridad: usuarios y roles

Como HTTP es un protocolo sin estado, las diferentes peticiones de un cliente al servidor son independientes, no están relacionadas entre sí. Para asociarlas, se utilizan las sesiones.

Al iniciar una sesión el servidor asigna y envía al usuario un identificador de sesión. En las siguientes peticiones el usuario envía al servidor ese identificador, de manera que el servidor sabe que se trata del mismo usuario.

Para controlar la sesión el servidor deja una *cookie* con el id de sesión en el cliente, que se elimina al cerrarla. Si se borran manualmente las *cookies* del navegador, se cierran las sesiones abiertas. Se puede configurar el servidor para controlar las sesiones sin *cookies*, pero no es lo habitual.

Para crear una sesión se utiliza la función `start_session()`. Si no hay una sesión activa la crea, si la hay, el *script* que llama a la función se une a ella. Una vez creada la sesión es posible

Escaneado con CamScanner

utilizar la variable superglobal `$_SESSION` para compartir información entre los *scripts* que compartan sesión. Es un *array* que almacena las variables de sesión definidas por el usuario, basta con añadir elementos de la manera usual.

```
$_SESSION["nombre"] = valor;
```

El ejemplo **sesiones_uso_basico.php** es muy sencillo.

```
<?php
    session_start();
    if (!isset($_SESSION['count'])) {
        $_SESSION['count'] = 0;
    }else {
        $_SESSION['count']++;
    }
    echo "hola ".$_SESSION['count'];
    echo "<br><a href='sesiones_uso_basico2.php'>Siguiente</a>";
```

Crea la sesión y, si no existe ya, una variable de sesión `$_SESSION["count"]` con valor 0. Si existe, le suma 1. Muestra un vínculo a **sesiones_uso_basico2.php**. Este segundo fichero se une a la sesión y muestra el valor de la variable. Accediendo a ambos se puede comprobar que realmente se trata de la misma variable.

```
<?php
    session_start();
    echo "La variable count vale: ". $_SESSION['count'];
```

En las aplicaciones web es habitual que se cree una sesión al hacer *login*. El ejemplo **sesiones1_login.php** amplía el del apartado 3.2.1. Si el *login* es correcto, se crea una sesión, una variable de sesión para el nombre de usuario y redirige **sesiones1_principal.php**.

Como en el ejemplo anterior, se usa una función para simular la base de datos. Admite dos usuarios, "usuario" y "admin", ambos con clave "1234". Si los datos son correctos, devuelve un *array* con el nombre y el rol del usuario, si no devuelve FALSE.

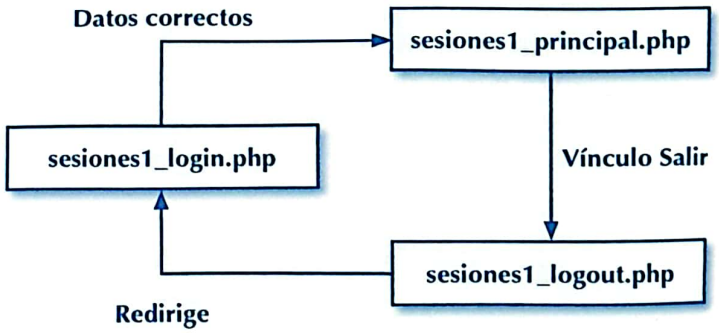


Figura 3.4
Diagrama para el ejemplo de sesiones.

```
<?php
/*formulario de login
habitual
si va bien abre sesión, guarda el nombre de usuario y redirige a principal.
php
si va mal, mensaje de error */
    function comprobar_usuario($nombre, $clave){
    if($nombre === "usuario" and $clave === "1234"){
```

Escaneado con CamScanner

```
        $usu['nombre'] = "usuario";
        $usu['rol'] = 0;
        return $usu;
    }elseif($nombre === "admin" and $clave === "1234"){
        $usu['nombre'] = "admin";
        $usu['rol'] = 1;
        return $usu;
    }else return FALSE;
}
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $usu = comprobar_usuario($_POST['usuario'], $_POST['clave']);
    if($usu==FALSE){
        $err = TRUE;
        $usuario = $_POST['usuario'];
    }else{
        session_start();
        $_SESSION['usuario'] = $_POST['usuario'];
        header("Location: sesiones1_principal.php");
    }
}
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de login</title>
        <meta charset = "UTF-8">
    </head>
    <body>
        <?php if(isset($_GET["redirigido"])){
            echo "<p>Haga login para continuar</p>";
        }?>
        <?php if(isset($err) and $err == true){
            echo "<p> revise usuario y contraseña</p>";
        }?>
        <form method = "POST" action = "<?php echo htmlspecialchars($_
        SERVER["PHP_SELF"]);?>" >
            Usuario
            <input value = "<?php if(isset($usuario))echo $usuario;?>"
            id = "usuario" name = "usuario" type = "text">
            Clave
            <input id="clave" name = "clave" type = "password">
            <input type = "submit">
        </form>
    </body>
</html>
```

Lo primero que hace **sesiones1_principal.php** es unirse a la sesión. Todos los ficheros que quieran acceder a las variables de sesión tienen que llamar a `session_start()`. Después,

Escaneado con CamScanner

comprueba que realmente se haya hecho *login* viendo si la variable `$_SESSION['usuari` está definida. Si no lo está redirige al formulario de *login*. De esta manera se bloquea el acc a **sesiones1_principal.php** si no se ha hecho *login* antes. Si la variable está definida, mue un mensaje personalizado.

```
<?php
    session_start();
    if(!isset($_SESSION['usuario'])){
        header("Location:sesiones1_login.php?redirigido=true");
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Página principal</title>
        <meta charset = "UTF-8">
    </head>
    <body>
        <?php echo "Bienvenido ".$_SESSION['usuario'];?>
        <br><a href = "sesiones1_logout.php"> Salir <a>
    </body>
</html>
```

Para cerrar la sesión se utiliza la función `session_destroy()`. El vínculo "Salir" lleva a **se siones1_logout.php**, que cierra la sesión y redirige a la página de *login*.

```
<?php
    session_start(); // unirse a la sesión
    $_SESSION = array();

    session_destroy(); // eliminar la sesión
    // eliminar la cookie

    setcookie(session_name(), 123, time() - 1000);
    header("Location: sesiones1_login.php");
```

Para cerrar la sesión no basta con llamar a `session_destroy()`. También hay que borrar las variables de sesión y la *cookie*.

3.5. Envío de correo electrónico

Aunque la función `mail()` permite enviar correos directamente, es habitual usar alguna librería que se ocupe de los detalles del formato. Una de las más habituales es phpMailer. Se puede instalar utilizando *composer*:

```
composer require phpmailer/phpmailer
```

En principio es posible enviar un correo utilizando la configuración de *sendmail* (en Linux) o un servidor SMTP local, pero en la práctica los filtros *antispam* hacen que no lleguen los correos enviados desde servidores que no estén registrados correctamente.

Escaneado con CamScanner

Si no se dispone de un servidor de correo en Internet, la opción más cómoda para enviar un correo electrónico es utilizar una cuenta de Gmail o similar. En Gmail hay que activar la opción "Permitir aplicaciones menos seguras" en la sección de ajustes de la cuenta.

Recurso web

Para permitir las aplicaciones menos seguras, consulta:

https://support.google.com/accounts/answer/6010255?hl=es

En **ejemplo_correo.php** se muestra cómo enviar un correo a través del servidor de Google.

- Hay que introducir usuario y clave de Google (líneas 13 y 18).
- Los destinatarios se añaden con `AddAddress()` (líneas 24-25)

```
1  <?php
2  use PHPMailer\PHPMailer\PHPMailer;
3  require "vendor/autoload.php";
4  $mail = new PHPMailer();
5  $mail->IsSMTP();
6  // cambiar a 0 para no ver mensajes de error
7  $mail->SMTPDebug = 2;
8  $mail->SMTPAuth = true;
9  $mail->SMTPSecure = "tls";
10 $mail->Host = "smtp.gmail.com";
11 $mail->Port = 587;
12 // introducir usuario de google
13 $mail->Username = "";
14 // introducir clave
15 $mail->Password = "";
16 $mail->SetFrom('user@gmail.com', 'Test');
17 // asunto
18 $mail->Subject = "Correo de prueba";
19 // cuerpo
20 $mail->MsgHTML('Prueba');
21 // adjuntos
22 $mail->addAttachment("empleado.xsd");
23 // destinatario
24 $address = "destino@servidor.com";
25 $mail->AddAddress($address, "Test");
26 // enviar
27 $resul = $mail->Send();
28 if(!$resul) {
29     echo "Error". $mail->ErrorInfo;
```

Escaneado con CamScanner

```
30 } else {
31     echo "Enviado";
32 }
```

3.6. Bases de datos relacionales

En PHP hay *drivers* para manejar los sistemas gestores de bases de datos más extendidos. También hay varias extensiones que proveen una capa de abstracción sobre la base de datos, entre las que destaca PHP Data Objects (PDO). Permite manejar diferentes bases de datos con una interfaz común. Tiene la ventaja de que si se cambia de base de datos no hay que modificar el código.

3.6.1. Conexión a la base de datos

El primer paso para trabajar con una base datos es obtener una conexión a la misma. Para representar la conexión se usa un objeto de clase PDO. El constructor es:

```
public PDO::__construct ( string $dsn [, string $username [, string $passwd [, array $options ]]]
```

El primer parámetro es una cadena que especifica qué *driver* hay que usar, la localización y el nombre de la base de datos. Para una base de datos MySQL se usaría:

```
mysql:dbname=<base de datos>;host=<ip o nombre>
```

Los siguientes parámetros son el nombre de usuario y clave para acceder a la base de datos. El último parámetro, opcional, es un *array* de opciones.

Si se puede establecer la conexión, se usará el nuevo objeto PDO para manejar la base de datos. Si no se puede conectar con la base de datos, el constructor lanza una excepción PDOException.

```
<?php
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadena_conexion, $usuario, $clave);
    $bd->close();
} catch (PDOException $e) {
    echo 'Error con la base de datos: '. $e->getMessage();
}
```

En principio, al acabar el *script* se cierra la conexión a la base de datos, pero también es posible usar conexiones persistentes, que no se cierran automáticamente al terminar el *script*. Quedan abiertas y si se vuelven a utilizar no es necesario reestablecer la conexión, por lo que pueden ser más eficientes en determinadas ocasiones. Para crear una conexión persistente se utiliza la opción PDO::ATTR_PERSISTENT en el constructor:

Escaneado con CamScanner

```
$bd = new PDO($cadena_conexion, $usuario, $clave, array(PDO::ATTR_PERSISTENT =>
    true));
```

TOMA NOTA

Los ejemplos de esta sección utilizan la base datos empresa, que contiene la tabla:
Usuarios(Código, Nombre, Clave, Rol)

- Código es la clave primaria, un campo autonumérico.
- Nombre es el nombre de usuario y está marcado como *unique*.
- Clave es una cadena con la clave de acceso al sistema.
- Rol es un entero que representa el rol del usuario dentro del sistema.

3.6.2. Recuperación y presentación de datos

El método `query($cad)` de la clase PDO ejecuta la cadena que recibe como argumento en la base de datos, como se haría desde la línea de comando SQL. El argumento `$cad` tiene que ser una instrucción SQL válida. Devuelve FALSE si hubo algún error o un objeto PDOStatement si la cadena se ejecutó con éxito.

Si se trata de una consulta, es posible recorrer las filas devueltas con un `foreach`. En cada iteración del bucle se tendrá una fila, representada como un *array* en que las claves son los nombres que aparecen en la cláusula `select`.

```
<?php
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadena_conexion, $usuario, $clave);
    echo "Conexión realizada con éxito";
    $sql = 'SELECT nombre, clave, rol FROM usuarios';
    $usuarios = $bd->query($sql);
    echo $usuarios->rowCount(). "<br>";
    foreach ($usuarios as $row) {
        print $row['nombre']. "\t";
        print $row['clave']. "\t";
    }
        }
```

Actividad propuesta 3.4

Escribe un fichero que reciba el código de un usuario y muestre por pantalla todos sus datos.

Escaneado con CamScanner

También es posible obtener instrucciones preparadas que permiten utilizar parámetros. Se inicializan una sola vez con el método `prepare()` y luego se ejecutan las veces que sea necesario con `execute()`, con diferentes valores para los parámetros. Las instrucciones preparadas permiten reutilizar las consultas, previenen la inyección de código y mejoran el rendimiento.

Hay dos opciones para indicar los parámetros de la consulta, por posición y por nombre. En el primer caso se utiliza el símbolo de interrogación para indicar un parámetro. Al ejecutarla, se asocian por orden los símbolos de interrogación con los valores del *array* que se pasa como argumento a `execute()`.

```
$preparada = $bd->prepare("select nombre from usuarios where
    rol = ?");
$preparada->execute( array(0));
echo "Usuarios con rol 0: ". $preparada->rowCount(). "<br>";
foreach ($preparada as $usu) {
    print "Nombre: ". $usu['nombre']. "<br>";
}
```

Si se utilizan nombres en la instrucción preparada, utilizando: *nombre*, se deberán usar esos nombres como claves del *array* de `execute()`.

```
$preparada_nombre=$bd->prepare("select nombre from usuarios where rol
=:rol");
$preparada_nombre->execute(array(':rol' => 0));
echo "Usuarios con rol 0: ". $preparada->rowCount(). "<br>";
foreach ($preparada_nombre as $usu) {
    print "Nombre: ". $usu['nombre']. "<br>";
}
```

Actividad propuesta 3.5

Modifica el formulario de *login* para que compruebe usuario y contraseña usando la tabla *usuarios* de la base de datos *empresa*.

3.6.3. Inserción, borrado y actualización

Para insertar, borrar o actualizar simplemente hay que ejecutar la sentencia SQL correspondiente. Puede ser una sentencia preparada o no.

```
<?php
// datos conexión
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    // conectar
    $bd = new PDO($cadena_conexion, $usuario, $clave);
```


Escaneado con CamScanner

```
    echo "Conexión realizada con éxito<br>";
    // insertar nuevo usario
    $ins = "insert into usuarios(nombre, clave, rol)
       values('Alberto', '33333', '1');";
    $resul = $bd->query($ins);
    //comprobar errores
    if($resul) {
       echo "insert correcto <br>";
       echo "Filas insertadas: ". $resul->rowCount(). "<br>";
    }else print_r( $bd -> errorinfo());
    // para los autoincrementos
    echo "Código de la fila insertada".$bd->lastInsertId()."<br>";
    // actualizar
    $upd = "update usuarios set rol = 0 where rol = 1";
    $resul = $bd->query($upd);
    //comprobar errores
    if($resul){
       echo "update correcto <br>";
       echo "Filas actualizadas: ". $resul->rowCount(). "<br>";
    }else print_r($bd -> errorinfo());
    // borrar
    $del = "delete from usuarios where nombre = 'Luisa'";
    $resul = $bd->query($del);
    //comprobar errores
    if($resul){
       echo "delete correcto <br>";
       echo "Filas borradas: ". $resul->rowCount(). "<br>";
    }else print_r($bd -> errorinfo());
} catch (PDOException $e) {
    echo 'Error con la base de datos: '. $e->getMessage();
}
```

Para que el ejemplo funcione, tiene que estar activada la opción de *autocommit* en la base datos.

3.6.4. Transacciones

Una transacción consiste en un conjunto de operaciones que deben realizarse de forma atómica. Es decir, o se realizan todas o no se realiza ninguna. Por ejemplo, una transferencia de saldo entre dos clientes implica dos operaciones:

- Quitar saldo al cliente que envía la transferencia.
- Aumentar el saldo del cliente que recibe la transferencia.

Si por cualquier motivo la segunda operación falla, hay que deshacer la primera operación para que el sistema quede en un estado consistente. Las dos operaciones forman una única transacción.

Para indicar el comienzo de una transacción se utiliza el método `beginTransaction()`. La transacción se salva usando el método `commit()`. Con el método `rollBack()` se deshacen las operaciones realizadas desde que se inició la transacción.

En el ejemplo **transaccion.php** se realizan dos inserciones como una transacción. Como la segunda falla (tiene un valor repetido para un campo *unique*), la primera se deshace.

Escaneado con CamScanner

```php
<?php
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadena_conexion, $usuario, $clave);
    echo "Conexión realizada con éxito<br>";
    // comenzar la transacción
    $bd->beginTransaction();
    $ins = "insert into usuarios (nombre, clave, rol)
        values('Fernando', '33333', '1')";
    $resul = $bd->query($ins);
    // se repite la consulta
    // falla porque el nombre es unique
    $resul = $bd->query($ins);
    if(!$resul){
        echo "Error: ". print_r($bd->errorinfo());
        // deshace el primer cambio
        $bd->rollback();
        echo "<br>Transacción anulada<br>";
    }else{
        // si hubiera ido bien
        $bd->commit();
    }
} catch (PDOException $e) {
    echo 'Error al conectar: '. $e->getMessage();
}
```

3.7. Bases de datos no relacionales

Las bases de datos no relacionales o no-SQL están cada vez más extendidas. Como su propi nombre indica, no siguen el modelo relacional. En las bases de datos no relacionales los modelo de datos son, en general, más flexibles y es más fácil realizar cambios en el esquema a lo larg del proceso de desarrollo.

En el caso de MongoDB, el elemento básico de almacenamiento es el documento. Los documentos se agrupan dentro de colecciones. Una base de datos cuenta con una o más colecciones.

Los documentos en MongoDB siguen un formato similar a JSON (incluye algunos tipos de datos adicionales) llamado BSON, Binary JSON. Por ejemplo,

```
{ nombre: "Pedro", edad: 22 }
```

Buscando una analogía con el modelo relacional, los documentos serían filas y las colecciones, tablas. Pero, al contrario que en el modelo relacional, no todos los documentos dentro de una colección tienen que tener la misma estructura. Dentro de una tabla, todas las filas tienen que tener las mismas columnas, aunque algunos valores puedan ser nulos. Las colecciones de MongoDB no imponen restricciones de ese tipo.

3.7.1. Instalación y puesta marcha de MongoDB

Hay que instalar:

- MongoDB Community Server, la versión libre de MongoDB.

Escaneado con CamScanner

- MongoDB Compass, una aplicación para manejar la base de datos con interfaz gráfica.
- El *driver* de PHP para MongoDB.

Recursos web

Para descargar MongoDB Community Server:

https://docs.mongodb.com/manual/administration/install-community/

Para MongoDB Compass:

https://www.mongodb.com/products/compass

Para empezar, hay que instalar el servidor MongoDB. En la configuración de la instalación, por defecto:

- El servidor escucha en el puerto 27017.
- La ruta de instalación en Windows es C:\Program Files\MongoDB.
- No es necesario utilizar usuario y contraseña para conectarse.

Una vez instalado, el servidor se pone en marcha ejecutando mongod.exe, que se encuentra en el subdirectorio bin dentro de la ruta de instalación del programa (en Windows, la ruta por defecto es C:\Program Files\MongoDB\Server\3.6\bin).

A continuación, se instala MongoDB Compass, que ofrece una interfaz gráfica para manejar el servidor. Al arrancar la aplicación, lo primero que hay que hacer es conectarse al servidor. Si se ha seguido la configuración por defecto, no hace falta cambiar nada, solo pulsar el botón Connect, en la esquina inferior derecha.

Al conectar, la aplicación muestra las bases de datos del servidor (llamado MyCluster). Hay tres bases de datos ya creadas: admin, config y local.

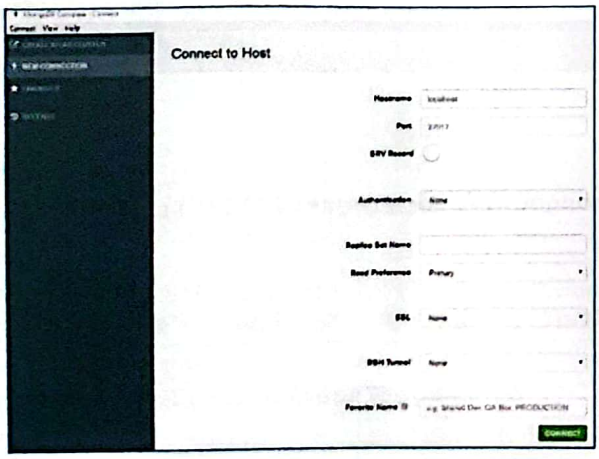


Figura 3.5
Conexión a MongoDB.

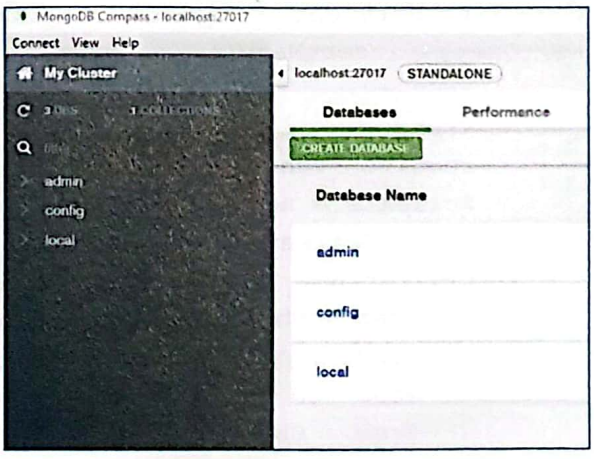


Figura 3.6
Bases de datos predefinidas.

Para crear una nueva se usa el botón "CREATE DATABASE". Al pulsarlo, se abre una ventana que pide el nombre de la nueva base de datos y también el de una colección. Es obligatorio

Escaneado con CamScanner

introducir ambos campos. Para poder utilizar los ejemplos posteriores hay que crear una base de datos llamada *libroservidor* y una colección llamada *usuarios*.

Al volver a la pantalla principal aparecerá la nueva base de datos, y si se pulsa sobre su nombre se verá también la colección.

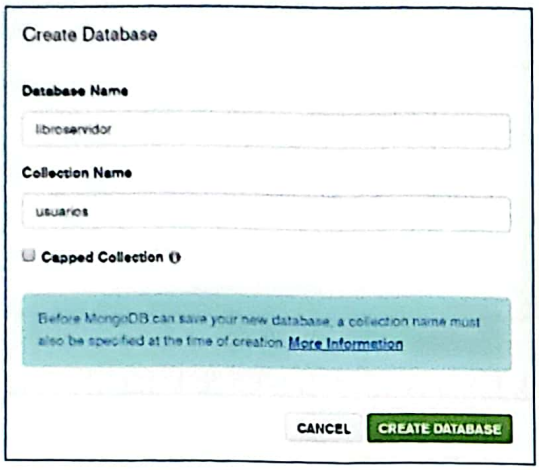


Figura 3.7
Creación de una nueva base de datos.

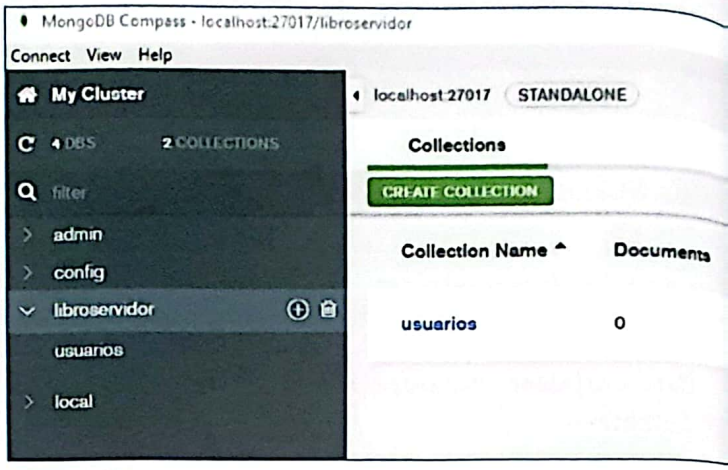


Figura 3.8
Colección creada.

Finalmente, para poder conectarse a un servidor MongoDB desde PHP hay que instalar el *driver* correspondiente, lo que se hace utilizando `composer`. Se debe ejecutar el comando:

```
composer require mongodb/mongodb
```

3.7.2. Conexión desde PHP

Con la librería de MongoDB para PHP es muy fácil realizar las operaciones habituales. El siguiente ejemplo inserta documentos en la colección creada en el apartado anterior.

Primero se conecta al servidor usando la clase `MongoDB\Client()` y se selecciona la base de datos `libroservidor`. Luego se insertan documentos en `usuarios` utilizando los métodos `insertOne()` e `insertMany()`.

```
<?php
require 'vendor/autoload.php';
$cliente = new MongoDB\Client("mongodb://localhost:27017");
$bd = $cliente->libroservidor;
try {
    $res = $bd->usuarios->insertOne( [ 'nombre' => 'Ana', 'clave' =>
    '1234', 'saldo' => 1000 ] );
    echo "Id del último registro: ". $res->getInsertedId(). "<br>";
    $res = $bd->usuarios->insertMany( [
        ['nombre' => 'Paco', 'clave' => '1234', 'saldo' => 100],
        ['nombre' => 'Nuria', 'clave' => '1234', 'saldo' => 30],
    ] );
    echo "Documentos insertados: ". $res->getInsertedCount(). "<br>";
    print_r($res->getInsertedIds());
```


Escaneado con CamScanner

```
} catch (Exception $e) {
    print ($e);
}
```

Para consultar se usan los métodos `find()` y `findOne()`. El segundo devuelve solo un resultado, se suele utilizar cuando se sabe que solo habrá uno, como al buscar por clave primaria. Admiten criterios de búsqueda utilizando un *array*, como se puede ver a continuación.

```
<?php
require 'vendor/autoload.php';
try {
    $cliente = new MongoDB\Client("mongodb://localhost:27017");
    $bd = $cliente->libroservidor;
    // devuelve todos los usuarios
    echo "Todos los usuarios". "<br>";
    $usuarios = $bd->usuarios->find();
    foreach($usuarios as $usuario){
        var_dump($usuario);
    }
    // usuarios con nombre Ana
    echo "Usuarias con nombre 'Ana'". "<br>";
    $usuarios = $bd->usuarios->find(['nombre' => 'Ana']);
    foreach($usuarios as $usuario){
        var_dump($usuario);
    }
    // solo devuelve el primero que encuentre
    echo "Usuaria con nombre 'Ana'". "<br>";
    $ana = $bd->usuarios->findOne(['nombre' => 'Ana']);
    var_dump($ana);
}catch (Exception $e) {
    print ($e);
}
```

Para actualizar se usan los métodos `update()` y `updateOne()`.

```
<?php
require 'vendor/autoload.php';
try {
    $cliente = new MongoDB\Client("mongodb://localhost:27017");
    $bd = $cliente->libroservidor;
    /* pone a 7000 el saldo del usuario con nombre 'Ana'*/
    $updateResult = $bd->usuarios->updateOne(
    [ 'nombre' => 'Ana' ],
    [ '$set' => [ 'saldo' => '7000' ]]
    );
}catch (Exception $e) {
    print ($e);
}
```

Escaneado con CamScanner

Para borrar se usan los métodos `delete()` y `deleteOne()`.

```
<?php
    require 'vendor/autoload.php';
    try {
        $cliente = new MongoDB\Client("mongodb://localhost:27017");
        $bd = $cliente->libroservidor;
        /* pone a 7000 el saldo del usuario con nombre 'Ana'*/
        $updateResult = $bd->usuarios->deleteOne(
        [ 'nombre' => 'Paco' ]);
        echo "Documentos restantes después de borrar: ".$bd->
        usuarios->count();
    }catch (Exception $e) {
        print ($e);
    }
```

3.8. Ficheros

En PHP hay varias librerías para manejo de ficheros. La más importante es Filesystem, que incluye funciones para leer y escribir ficheros y para manejar el sistema de archivos. Son funciones muy similares a las de manejo de ficheros en C.

Para abrir un fichero se usa la función `fopen()`, indicando la ruta del fichero y el modo en el que se abre. Si hay algún error, la función devuelve FALSE; en otro caso devuelve un puntero para manejar el fichero. En el siguiente ejemplo se muestran ambos casos.

```
<?php
$fich = fopen("fichero_ejemplo.txt", "r");
if ($fich === FALSE){
    echo "No se encuentra fichero_ejemplo.txt<br>";
}else{
    echo "fichero_ejemplo.txt se abrió con éxito<br>";
}
$fich = fopen("fichero_no_existe.txt", "r");
if ($fich === FALSE){
    echo "No se encuentra fichero_no_existe.txt<br>";
}else{
    echo " fichero_no_existe.txt se abrió con éxito<br>";
}
```

El primer fichero sí existe, pero el segundo no. Al ejecutarlo la salida será:

```
fichero_ejemplo.txt se abrió con éxito Warning: fopen(fichero_no_existe.txt): failed to open
    stream: No such file or directory in C:\xampp\htdocs\cap3\ficheros_fopen.php on line No
    se encuentra fichero_no_existe.txt
```

El mensaje de Warning lo pone el propio servidor y, en principio, no se mostraría en producción. Los modos de apertura más importantes se recogen en el cuadro 3.1.

Escaneado con CamScanner

CUADRO 3.1
Modos de apertura de ficheros

Modo	Descripción
r	Solo lectura. Si el fichero no existe, devuelve FALSE
r+	Lectura y escritura. Si el fichero no existe, devuelve FALSE
w	Solo escritura. Si el fichero no existe, se crea; si existe, se trunca (es decir, se borra el contenido anterior). Si no puede crearlo, devuelve FALSE
w+	Lectura y escritura. Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE
a	Solo escritura. Las escrituras se realizan siempre al final de fichero (append). Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE
a+	Lectura y escritura. Las escrituras se realizan siempre al final de fichero. Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE

Lectura y escritura se realizan a partir del indicador de posición del fichero. Cuando se abre un fichero, el indicador se sitúa al inicio de este, salvo si se ha abierto en modo "a" o "a+", que se sitúa al final. Cuando se lee o escribe, el puntero avanza para situarse al final del último carácter o *byte* leído o escrito

En el siguiente ejemplo, se abre un fichero en modo lectura y se lee carácter a carácter usando la función `fgetc()`, que devuelve el siguiente carácter a partir del indicador de posición.

```
<?php
$fich = fopen("fichero_ejemplo.txt", "r");
if ($fich === FALSE){
   echo "No se encuentra el fichero o no se pudo leer<br>";
}else{
   while( !feof($fich) ){
      $car = fgetc($fich);
      echo $car;
   }
}
fclose($fich);
```

Para detectar el final del fichero se usa la función `feof()`, que devuelve TRUE cuando ya se ha llegado al final. Cuando se termina de trabajar con el fichero, se cierra con `fclose()`.

Para leer un fichero que sigue un formato determinado se puede usar la función `fscanf()`, que lee una línea del fichero y le aplica un formato. Hay dos maneras de usarla. En la primera, se le pasan dos parámetros y devuelve un *array* con los valores leídos.

```
$valores = fscanf($fichero, $formato);
```

También se le pueden pasar variables adicionales para que almacene en ellas los valores leídos en lugar de devolverlos. En este caso devuelve el número de valores leídos correctamente.

Escaneado con CamScanner

```
$valores = fscanf($fichero, $formato, $var1, ...);
```

Ejemplo de formato

Supongamos que tenemos un fichero de texto que representa una matriz de 4 filas y 4 columnas.

```
  23 234 611  5
1233 565 123  5
 123  54 757 12
  77  88   9 99
```

Para indicar que cada línea está formada por 4 números separados por espacios se usa como formato:

```
"%d %d %d %d"
```

Cada %d representa un número en formato decimal, y se deja un espacio entre ellas.

El siguiente ejemplo muestra cómo recorrer un fichero con el formato anterior utilizando `fscanf()` de las dos maneras posibles. Antes de empezar con la segunda vuelta, sitúa el indicador de posición de nuevo al principio del fichero usando `rewind()`.

```
<?php
$fich = fopen("matriz.txt", "r");
if ($fich === False){
   echo "No se encuentra el fichero o no se pudo leer<br>";
}else{
   while( !feof($fich) ){
      $num = fscanf($fich, "%d %d %d %d");
      echo "$num[0] $num[1] $num[2] $num[3] <br>";
   }
}
rewind($fich);
while( !feof($fich) ){
   $num = fscanf($fich, "%d %d %d %d", $num1,$num2,$num3,$num4);
   echo "$num1 $num2 $num3 $num4 <br>";
}
fclose($fich);
```

También son útiles las funciones `file_get_contents()` y `file_put_contents()`. La primera devuelve una cadena con el contenido de un fichero. La segunda hace lo contrario, escribe datos en un fichero. La ventaja que tienen es que funcionan directamente a partir de la ruta del fichero, así que no hace falta llamar a `fopen()` y `fclose()`.

Escaneado con CamScanner

```
<?php
$contenido = file_get_contents("fichero_ejemplo.txt");
echo "Contenido del fichero: $contenido<br>";
$res = file_put_contents("fichero_salida.txt",
    "Fichero creado con file_put_contents");
if($res){
    echo "Fichero creado con éxito";
}else{
    echo "Error al crear el fichero";
}
```

En el cuadro 3.2 se resumen las funciones más útiles para trabajar con ficheros.

CUADRO 3.2
Funciones de ficheros

Función	Descripción
fgets($fich)	Devuelve una cadena con los caracteres desde el indicador de posición
fputs($fich, $cad)	Escribe una cadena en el fichero
fseek($fich, $pos)	Sitúa el cursor del fichero en la posición indicada
ftell($fich)	Devuelve el indicador de posición del fichero
rewind($fich)	Sitúa el indicador de posición al principio del fichero
fopen($ruta, modo)	Abre un fichero
fclose($fich)	Cierra un fichero
fread($fich, $longitud)	Lee $longitud *bytes*
fwrite($fich, $cadena)	Escritura una cadena en un fichero
copy($origen, $destino)	Copia un fichero
unlink($borra)	Borra un fichero
move($actual, $nuevo)	Mueve un fichero
filesize($ruta)	Devuelve el tamaño del fichero en *bytes*
Filetype($ruta)	Devuelve el tipo de fichero
is_file($ruta) is_dir($ruta)	Para comprobar si la ruta corresponde a un fichero Para comprobar si la ruta corresponde a un directorio
rename($actual,$nuevo)	Cambia el nombre a un fichero

3.8.1. Ficheros XML

También hay librerías específicas para facilitar el trabajo con ficheros XML. La librería SimpleXML viene activada por defecto y tiene funciones para filtrar y recorrer ficheros XML.

Escaneado con CamScanner

La función `simplexml_load_file()` lee un fichero XML y devuelve un objeto de clase `SimpleXMLElement`. El fichero se manipula a través de este objeto. Por ejemplo, si se recorre como un *array*, se obtienen los hijos del elemento raíz.

```
<?php
$datos = simplexml_load_file("empleados.xml");
echo "<br>";
if($datos===false){
    echo "Error al leer el fichero";
}
foreach($datos as $valor){
    print_r($valor);
    echo "<br>";
}
```

El método `Xpath()` permite seleccionar ciertos elementos usando una expresión XPath. En el siguiente ejemplo se selecciona solo los elementos *<edad>*.

```
<?php
$datos = simplexml_load_file("empleados.xml");
$edades = $datos->xpath("//edad");
foreach($edades as $valor){
    print_r($valor);
    echo "<br>";
}
```

Validar con esquemas XSD es muy sencillo con la clase `DOMDocument`. El siguiente ejemplo valida **empleados.xml** usando el esquema **departamento.xsd**.

```
<?php
$dept = new DOMDocument();
$dept->load( 'empleados.xml');
$res = $dept->schemaValidate('departamento.xsd');
if ($res){
    echo "El fichero es válido";
}
else {
    echo "Fichero no válido";
}
```

También es posible utilizar transformaciones XSLT con la librería del mismo nombre. El siguiente ejemplo utiliza una transformación para mostrar un fichero XML como una tabla HTML. Tanto la transformación como el fichero XML se cargan utilizando la clase `DOMDocument`. La transformación se realiza con un objeto `XSLTProcessor`, que se asocia con la transformación usando el método `importStylsheet()`.

```
<?php
    // cargar el fichero a transformar
    $dept = new DOMDocument();
    $dept->load('empleados.xml' );
    // cargar la transformacion
    $transformacion = new DOMDocument();
```

Escaneado con CamScanner

```
$transformacion->load( 'departamento.xslt' );
// crear el procesador
$procesador = new XSLTProcessor();
// asociar el procesador y la transformacion
$procesador-> importStylesheet($transformacion);
// transformar
$transformada = $procesador->transformToXml($dept);
echo $transformada;
```

Los ejemplos de este apartado usan el fichero **empleados.xml**. Representa la información de un departamento y sus empleados.

```
<?xml version="1.0" encoding="UTF-8"?>
<departamento nombre="Marketing" jefeDep="e101">
    <empleado codEmple="e101">
        <nombre>Ana</nombre>
        <apellidos>Frutos Jarama</apellidos>
        <edad>28</edad>
        <carnetConducir/>
    </empleado>
    <empleado codEmple="e102">
        <nombre>Antonio</nombre>
        <apellidos>Miguel Bustos</apellidos>
        <edad>23</edad>
    </empleado>
</departamento>
```

El esquema para validarlo es este:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDe-
fault="qualified">
    <xs:element name="departamento">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="empleado" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="jefeDep" type="xs:string" use="required"/>
            <xs:attribute name="nombre" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="empleado">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellidos" type="xs:string"/>
                <xs:element name="edad" type="xs:string"/>
                <xs:element name="carnetConducir" minOccurs = "0"
                type="xs:string"/>
            </xs:sequence>
```

Escaneado con CamScanner

```
            <xs:attribute name="codEmple" type="xs:string" use="requi-
            red"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

La transformación es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver-
sion="1.0">
    <xsl:template match="/">
        <html>
            <head>
                <title>Empleados del departamento</title>
            </head>
            <body>
                <table>
                <tr>
                    <th>Nombre</th>
                    <th>Apellidos</th>
                    <th>Edad</th>
                </tr>
                <xsl:for-each select="departamento/empleado">
                    <tr>
                        <td>
                            <xsl:value-of select="nombre"/>
                        </td>
                        <td>
                            <xsl:value-of select="apellidos"/>
                        </td>
                        <td>
                            <xsl:value-of select="edad"/>
                        </td>
                    </tr>
                </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

3.9. Pruebas

Para automatizar las pruebas de código, la herramienta más utilizada en PHP es PHPUnit, que sigue una línea similar a JUnit. Se puede instalar con *composer*:

```
composer require --dev phpunit/phpunit
```


Escaneado con CamScanner

El elemento base de PHPUnit es el caso de prueba, una clase que hereda de la clase `TestCase` de la librería. Dentro de un caso de prueba hay varias pruebas.

Para probar una clase de nombre `Clase` se creará una clase llamada `ClaseTest`. Esta clase contendrá una serie de métodos, que contendrán las pruebas que se quieren realizar. Estos métodos tienen que ser públicos. La clase también puede tener otros métodos auxiliares que no sean pruebas.

Para indicar que un método es una prueba hay dos opciones:

- Que el nombre empiece por "test".

```
public testExcepcion(){ ... }
```

- Utilizar un bloque de comentarios específico con la anotación `@test`. Son bloques de comentario que empiezan por "/**". Estos bloques se sitúan justo encima del método al que se refieren.

```
/**
* @test
*/
public static function excepcion();
```

Dentro de estos métodos, la mayoría de las comprobaciones se realizan utilizando aserciones, métodos definidos en la clase `TestCase`, que comprueban si se cumplen o no ciertas condiciones. Si alguna de las aserciones del método no se cumple, se considera que la prueba ha fallado.

Por ejemplo, la clase `MatematicasTest` sirve para probar la clase `Matematicas`, en concreto el método factorial. El primer método, `testCero()`, se encarga de verificar si la función calcula correctamente el factorial de 0. Si la función devuelve 1, la aserción se cumple y se considerará que la prueba ha ido bien. En caso contrario, la prueba falla.

El segundo método se encarga de comprobar que, si se le pasa un argumento negativo, la función genera una excepción. Si la función no genera la excepción, la prueba falla. Se puede hacer con el método `expectException()` o con la anotación `@expectedException`.

```
<?php
    require "vendor/autoload.php";
    require "Matematicas.php";
    use PHPUnit\Framework\TestCase;
    class MatematicasTest extends TestCase{
        public function testCero(){
        $this->assertEquals(1,Matematicas::factorialEx(0));
        }
        /**
        * @test
        * @expectedException InvalidArgumentException
        */
        public static function Excepcion(){
            return Matematicas::factorialEx(-1);
        }
    }
```

Escaneado con CamScanner

Para ejecutar un caso de prueba, se ejecuta PHPUnit desde la línea de comandos.

```
C:\xampp\htdocs\cap3\> \vendor\bin\phpunit MatematicasTest.php
```

La salida en este caso es:

```
2/2 100%
2 test / 2 assertions
```

Es decir, se han realizado dos pruebas (que contienen dos aserciones) y las dos han sido correctas.

La clase `TestCase` incluye muchos más métodos para hacer comprobaciones. Algunos de los más útiles están recogidos en el cuadro 3.3.

CUADRO 3.3
Algunos métodos básicos de PHPUnit

Aserción / método	Descripción
assertLessThan($limite, $valor)	Comprueba que $valor sea menor que $limite
assertGreaterThan($limite, $valor)	Comprueba que $valor sea mayor que $limite
assertStringStartsWith($prefijo, $cadena)	Comprueba que $cadena empiece por $prefijo
assertArrayHasKey($clave, $array)	Comprueba que $array tenga un elemento con clave $clave
assertInstanceOf(Clase, $objeto)	Comprueba que $objeto sea una instancia de Clase
assertFileExists($ruta)	Comprueba que la ruta exista
assertNull($var)	Comprueba que $var sea *null*
assertCount($num, $conjunto)	Comprueba que $conjunto tenga $num elementos
expectException(Clase)	Comprueba que el método genere una excepción de clase Clase

3.10. Depuración de errores

Aunque PHP incluye un depurador a partir de la versión 5.6, lo habitual es usar un depurador externo integrado en un IDE. Los depuradores más extendidos son ZendDebugger y Xdebug. Ambos están disponibles sin instalaciones adicionales en Eclipse PDT (ver capítulo 1).

El proceso de depuración es similar al de otros lenguajes:

- Para depurar un *script*, hay que establecer al menos un punto de ruptura (*breakpoint*).
- Al comenzar la depuración, el *script* se ejecutará hasta el primero. A partir de ahí, es posible ejecutar la siguiente línea de código pulsando F6.
- Si se trata de una función, pulsando F5 se entra en el código de esta.

Escaneado con CamScanner

- A la derecha se muestran las variables del *script*; sus valores se van actualizando al ejecutar más instrucciones.

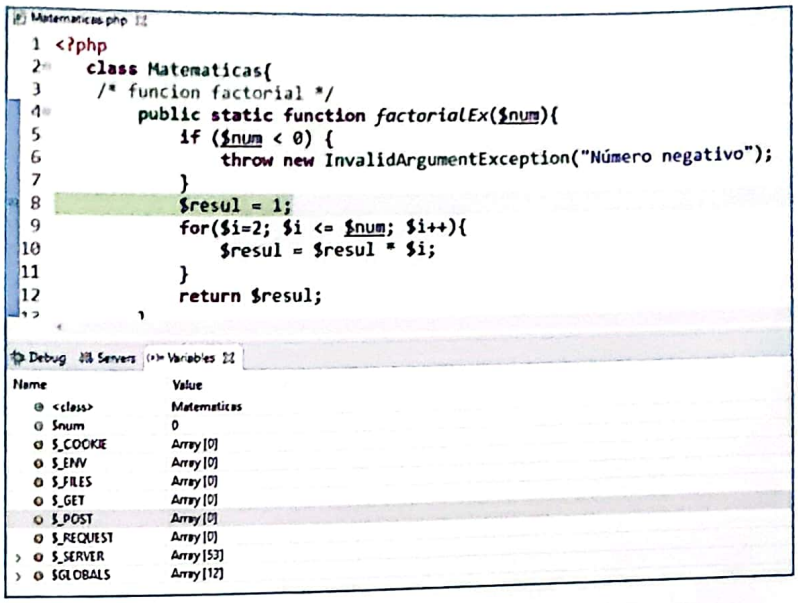


Figura 3.9
Depurador de Eclipse PDT.

Resumen

- Los métodos HTTP más habituales son GET y POST. Ambos permiten enviar parámetros.
- GET es el método que se usa para solicitar una página al servidor. Los parámetros enviados por GET se añaden a la URL y están disponibles en $_GET.
- En los formularios es habitual usar POST para enviar los datos. El *script* que reciba el formulario podrá acceder a ellos en el *array* $_POST.
- Las claves de $_POST coinciden con los atributos *name* de los controles del formulario.
- Hay que tener cuidado de no ejecutar funciones que afecten a la cabecera de la respuesta, como setcookie(), después de haber comenzado a generar la salida del *script*.
- Las *cookies* permiten al servidor almacenar información en cliente. Este puede borrarlas cuando quiera, ya que son parte del historial de navegación.
- Las páginas de una misma aplicación pueden compartir información mediante las sesiones. Las variables de sesión se almacenan en el *array* $_SESSION.
- PHP cuenta con *drivers* para las bases de datos más extendidas. La librería PDO se encarga de los detalles específicos de la base de datos.
- Hay varias librerías para trabajar con ficheros XML. Desde PHP es posible utilizar esquemas XSD, transformaciones XSL/XSLT y búsquedas con XPath.
- Para automatizar las pruebas la librería más extendida es PHPUnit.

Ejercicios propuestos

1. Crea una página web con un formulario para elegir el color de fondo. Almacena la elección del usuario con una *cookie* para que la siguiente vez que el usuario se conecte la página aparezca directamente con ese color.

Escaneado con CamScanner

2. Amplia el ejemplo de la sección 3.4 para que la página principal incluya un nuevo vínculo, "Zona Admin", solo visible para los usuarios con rol 1.
3. Escribe un programa que muestre una tabla con los datos de la tabla Usuarios.
4. Confecciona un formulario para insertar usuarios en la tabla Usuarios.
5. A partir del ejemplo del apartado 3.5, crea una función para enviar correos.
6. Realiza un formulario para enviar correos que utilice esa función.

ACTIVIDADES DE AUTOEVALUACIÓN

1. Los parámetros que se pasan por la URL están disponibles en:
   - ☐ *a)* $_GET.
   - ☐ *b)* $_POST.
   - ☐ *c)* Ambos.

2. En $_FILES['fichero']['name'] se almacena:
   - ☐ *a)* El nombre en el cliente del archivo que se está subiendo.
   - ☐ *b)* El nombre temporal en el servidor del archivo que se está subiendo.
   - ☐ *c)* El nombre en el que se almacenará el fichero en el servidor.

3. En Mongo DB los documentos se almacenan como:
   - ☐ *a)* BSON.
   - ☐ *b)* XML.
   - ☐ *c)* Se convierten a una fila de la tabla correspondiente.

4. Para borrar una *cookie* se utiliza la función:
   - ☐ *a)* deletecookie().
   - ☐ *b)* setcookie().
   - ☐ *c)* erasecookie().

5. ¿Cuál de estos elementos no se envía en las cabeceras?
   - ☐ *a)* Redirecciones.
   - ☐ *b)* Datos de formulario.
   - ☐ *c)* *Cookies*.

6. El cliente envía al servidor:
   - ☐ *a)* Todas las *cookies* que tiene.
   - ☐ *b)* Todas las *cookies* que tiene de ese servidor.
   - ☐ *c)* Las *cookies* que el servidor le solicita.

7. Para cerrar la sesión hay que:
   - ☐ *a)* Llamar a sesion_destroy().
   - ☐ *b)* Llamar a sesion_destroy(), borrar las variables de sesión y la *cookie* de sesión.
   - ☐ *c)* Llamar a sesion_destroy() o borrar las variables de sesión.

Escaneado con CamScanner

8. Para automatizar las pruebas se utiliza la librería:
   - ☐ *a)* PHPUnit.
   - ☐ *b)* PHPMailer.
   - ☐ *c)* PDO.

9. Al abrir un fichero con modo de apertura "r+":
   - ☐ *a)* Si el fichero no existe, lo crea.
   - ☐ *b)* Si el fichero no existe, devuelve error.
   - ☐ *c)* Si el fichero existe, devuelve error.

10. Al abrir un fichero con modo de apertura "w":
   - ☐ *a)* Si el fichero no existe, lo crea.
   - ☐ *b)* Si el fichero no existe, devuelve error.
   - ☐ *c)* Si el fichero existe, devuelve error.

SOLUCIONES:

1. **a** b c
2. **a** b c
3. **a** b c
4. a **b** c
5. a **b** c
6. a **b** c
7. a **b** c
8. **a** b c
9. a **b** c
10. **a** b c

Escaneado con CamScanner

4

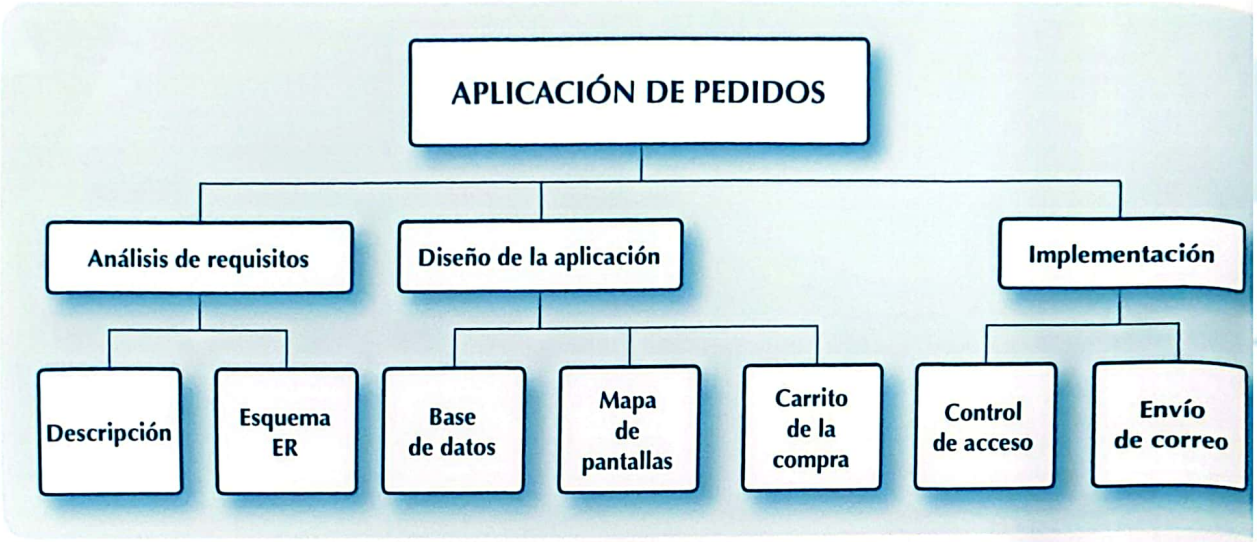
Ejemplo de aplicación completa en PHP

Objetivos

- ✓ Crear una aplicación completa de pedidos.
- ✓ Cargar dinámicamente las categorías y productos disponibles.
- ✓ Controlar el estado del pedido mediante una variable de sesión.
- ✓ Almacenar pedidos en la base de datos.
- ✓ Controlar el acceso con una tabla de usuarios.
- ✓ Enviar correos de confirmación.

Escaneado con CamScanner

Mapa conceptual



Glosario

Carrito de la compra. Abstracción habitual en las tiendas online. En el carrito se almacenan los productos que el cliente va escogiendo antes de realizar el pedido.

Diseño físico de la base de datos. Paso final para crear la base de datos. Se definen los tipos de datos e índices. Se pueden realizar cambios sobre el diseño lógico para optimizar el rendimiento.

Diseño lógico de la base de datos. En el modelo relacional consiste en obtener el conjunto de tablas que formarán la base de datos. Habitualmente se parte de un esquema ER o un diagrama de clases.

Especificación de requisitos de software. Descripción detallada de la aplicación que se va a realizar.

Esquema ER. Representación gráfica del esquema de datos de una aplicación. Es el primer paso para desarrollar una base de datos.

Mapa de flujo de pantallas. Diagrama que representa las vistas disponibles para el usuario y cómo se relacionan.

4.1. Definición del proyecto

En este capítulo se desarrolla una aplicación web para realizar pedidos, similar a una tienda web. Se pretende:

- Unir los elementos de los capítulos anteriores para plantear una aplicación web completa.
- Mostrar cómo plantear un proyecto de aplicación web con base de datos desde cero siguiendo una metodología apropiada.

Escaneado con CamScanner

Es un proyecto pequeño pero completo. Se desarrollan las fases de análisis, diseño e implementación. Se puede considerar como la primera iteración dentro de un proyecto más completo.

Actividad propuesta 4.1

Si no lo conoces, busca información sobre el modelo de desarrollo en espiral.

En el análisis, se define la funcionalidad de la aplicación y sus limitaciones. Se detallan los datos que se quieren almacenar y se realiza un esquema entidad relación para la base de datos.

El diseño es la parte más importante del capítulo, se definen:

- Las pantallas que verá el usuario.
- Los ficheros que formarán la aplicación y cómo se pasarán los parámetros entre ellos.
- La estructura de datos para el carrito de la compra y cómo manipularla, uno de los puntos más complicados de la aplicación.
- La base de datos.

Finalmente, en la implementación, se escriben los ficheros de la aplicación. La parte relacionada con la base de datos es la más larga.

Como la base de datos es un elemento fundamental para la aplicación, para poder entender bien el capítulo es necesario recordar los elementos básicos del modelo entidad-relación y, sobre todo, del modelo relacional y del SQL.

La aplicación es básicamente una tienda web sencilla. Se espera que tenga la funcionalidad habitual en una tienda online. La principal diferencia está en que, como los restaurantes son de la misma empresa, el pedido no requiere pago. Tampoco es necesario especificar la dirección de envío. La empresa sabe dónde está cada restaurante.

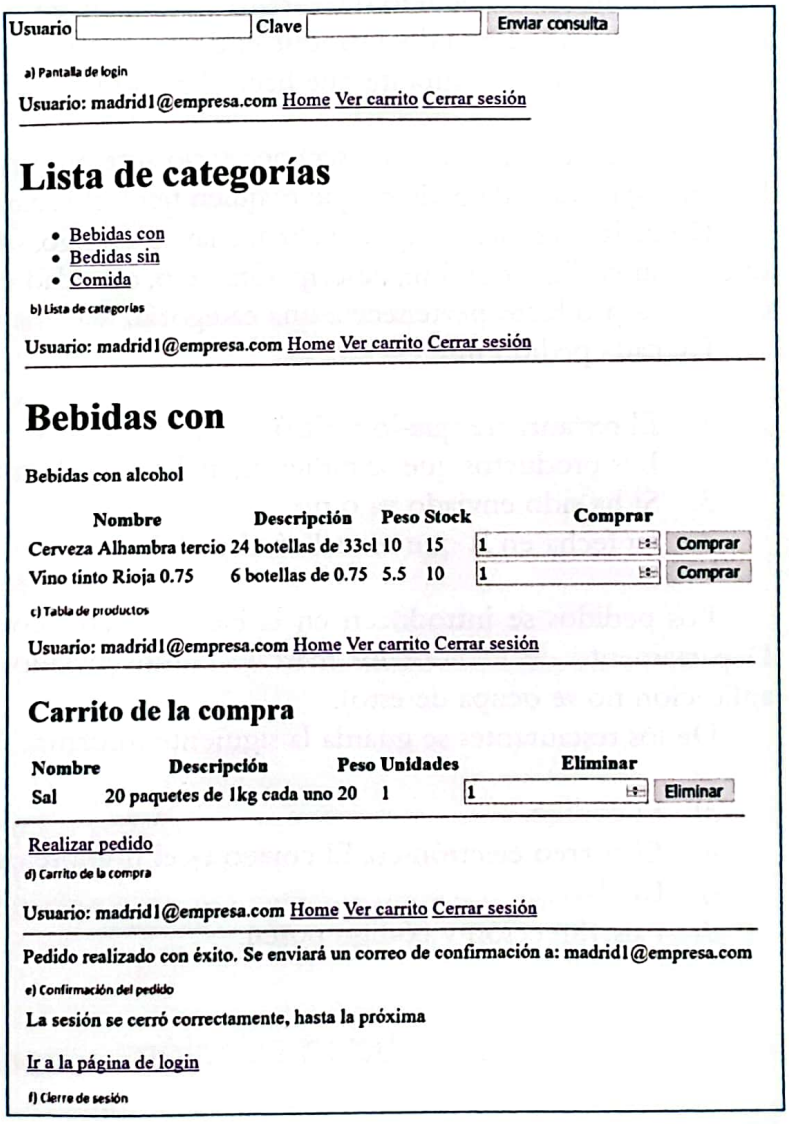


Figura 4.1
Resultado final de la aplicación.

Escaneado con CamScanner

RECUERDA

✓ Asegúrate de haber importado la base de datos **pedidos**, como se explica en el capítulo 1, y de arrancar la base de datos.

4.2. Análisis de requisitos

Se quiere realizar una aplicación para el Departamento de Pedidos para una cadena de restaurantes. Los restaurantes de la cadena utilizarán la aplicación web para realizar pedidos de comida, bebida y materiales.

La aplicación debe permitir:

- Consultar las categorías.
- Consultar los productos.
- Añadir una o más unidades de un producto al pedido.
- Consultar el pedido del carrito y eliminar productos de este.
- Realizar el pedido, introduciéndolo en la base de datos y enviando correos de confirmación al restaurante que hace el pedido y al Departamento de Pedidos de la empresa.

Para acceder a la aplicación será necesario autentificarse. Se supone que en cada restaurante habrá un responsable de pedidos que es quien tiene el usuario y la clave para acceder a la aplicación.

De cada categoría se quiere almacenar su código, su nombre y su descripción. De los productos, su código, nombre, descripción, peso, cantidad en *stock* y la categoría a la que pertenecen. Cada producto pertenece a una categoría.

De cada pedido interesa saber:

1. El restaurante que lo realizó.
2. Los productos que se pidieron, incluyendo la cantidad de unidades de cada producto.
3. Si ha sido enviado ya o no.
4. La fecha en la que se realizó el pedido.

Los pedidos se introducen en la base de datos como no enviados. Cuando se envíen el Departamento de Pedidos los marcará como enviados (directamente en la base de datos, la aplicación no se ocupa de esto).

De los restaurantes se guarda la siguiente información:

a) El código.
b) El correo electrónico. El correo es el nombre de usuario para acceder a la aplicación.
c) La clave.
d) País, dirección y código postal.

4.2.1. Esquema entidad-relación

De la descripción anterior se obtiene este esquema E-R. La relación *Incluye* es *muchos a muchos* porque un pedido puede incluir varios productos y un producto puede aparecer en varios pedidos

Escaneado con CamScanner

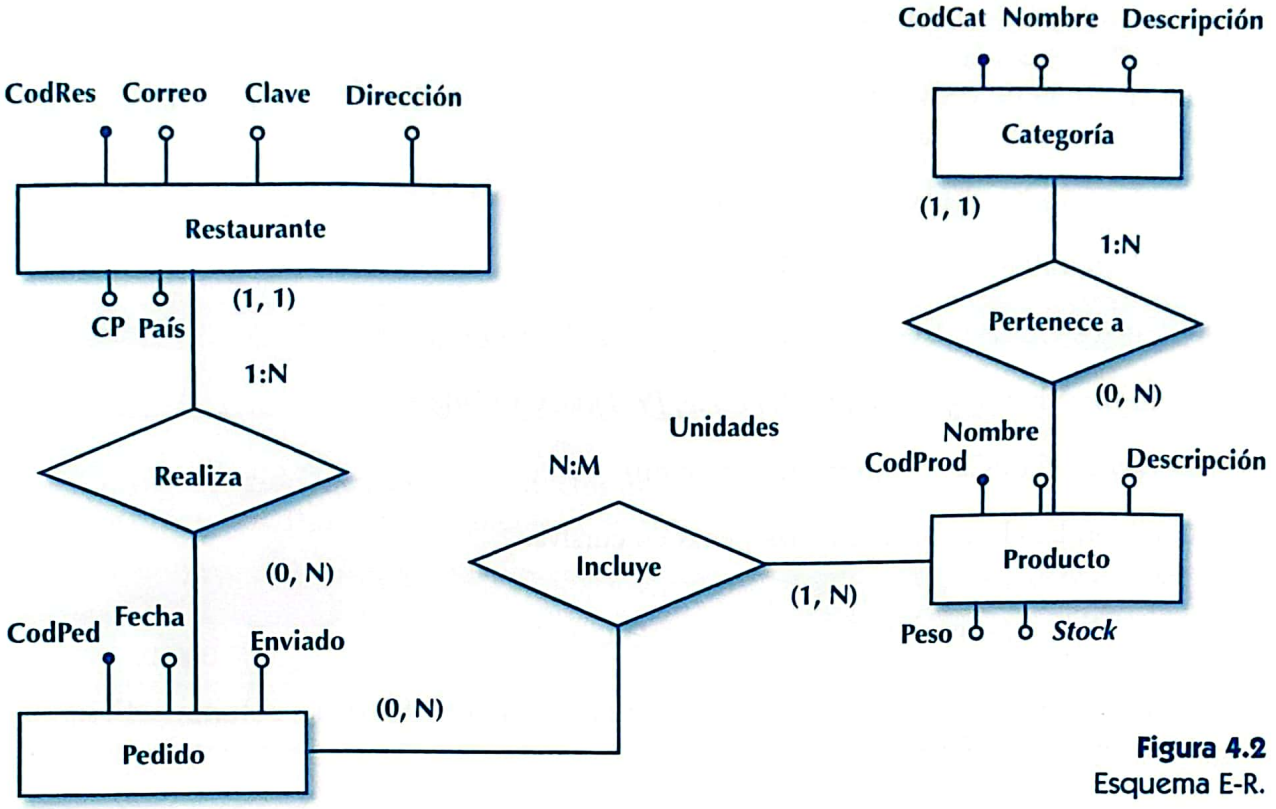


Figura 4.2
Esquema E-R.

4.2.2. Limitaciones de la aplicación

No hay panel de administración. Los usuarios, categorías y productos se tienen que introducir directamente en la base de datos.

No hay posibilidad de autorregistro.

No se controla el *stock*. Si al realizar un pedido algún producto queda con stcok negativo, el pedido se tramita igualmente.

4.3. Diseño de la aplicación

A partir del análisis anterior, se puede proceder al diseño de la aplicación. Los elementos más importantes son:

a) La base de datos.
b) El flujo de pantallas para realizar un pedido.
c) La estructura de datos para el carrito de la compra.
d) Los ficheros que forman la aplicación y cómo se pasan parámetros entre ellos.
e) El control de acceso.

4.3.1. Diseño lógico de la base de datos

Para obtener las tablas de la base de datos se parte del esquema E-R anterior. Para empezar, se crea una tabla por cada entidad. Las relaciones *Realiza* y *Pertenece a* implican un intercambio de

Escaneado con CamScanner

claves. La tabla producto recibirá la clave de la categoría, y la tabla pedido, la del restaurante que la realiza. Por otro lado, la relación *Incluye* es N:M y se resuelve mediante una tercera tabla que incluirá las claves de Pedido y Producto y los atributos de la relación.

Se obtienen las siguientes tablas:

Restaurantes(CodRes, Correo, Clave, Pais, CP, Ciudad, Dirección)

Pedidos(CodPed, Fecha, Enviado, *Restaurante*)

Productos(CodProd, Nombre, Descripción, Peso, Stock, *Categoria*)

PedidosProductos(CodPedProd, *Pedido*, *Producto*, Unidades)

Categorias(CodCat, Nombre, Descripción)

'Cod' son las claves primarias, las ajenas en cursiva.

4.3.2. Diseño físico de la base de datos

Para finalizar con la base de datos hay que decidir los tipos de datos de las columnas. El siguiente diagrama representa el resultado final.

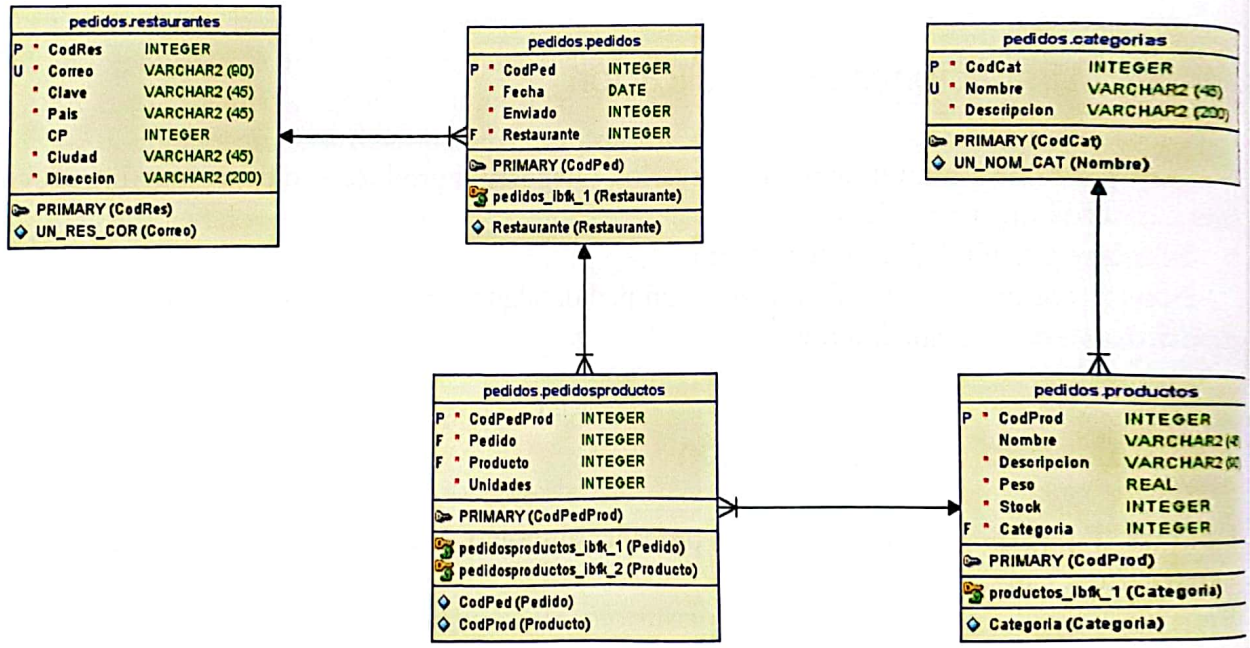


Figura 4.3
Tablas de la base de datos.

Conviene señalar que:

- Los códigos serán enteros con la opción de autoincremento.
- El correo de los restaurantes y el nombre de las categorías se marcan como *unique*.

Para insertar un pedido en la base de datos hay que insertar una fila en la tabla pedidos y una en PedidosProductos por cada producto diferente que incluya el pedido.

Escaneado con CamScanner

Ejemplo

Inserción de un pedido

El usuario con código 1 hace un pedido de 100 unidades del producto con código 16 y 150 del producto con código 20. El pedido recibe el código 1.000. En la tabla pedidos se insertará:

CodPed	Fecha	Enviado	Ejemplo
1.000	01/09/2019	0	1

Como el pedido incluye dos productos diferentes, hay que insertar dos filas en la tabla PedidosProductos. Estas filas reciben los códigos 20.000 y 20.001.

CodPedProd	CodPed	CodProd	Unidades
20.000	1.000	16	100
20.001	1.000	20	150

4.3.3. Diagrama de flujo de pantallas

El objetivo de este tipo de diagramas es representar las pantallas por las que pasa el usuario al realizar una operación. Los cuadrados representan las pantallas, y las flechas, acciones que llevan de unas a otras. El siguiente diagrama es una posible solución para este caso.

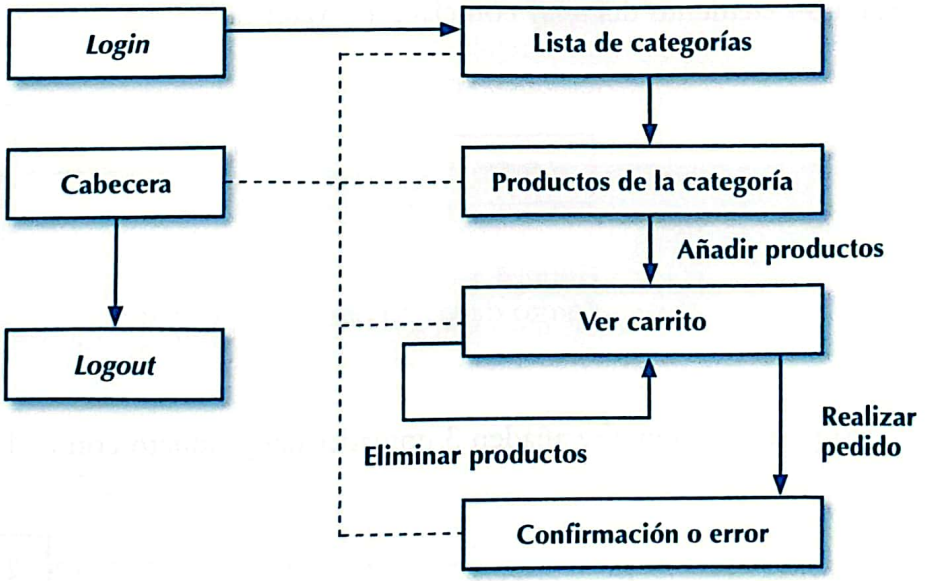


Figura 4.4
Diagrama de flujo de pantallas.

El punto de entrada a la aplicación es *login*, donde el usuario debe introducir un usuario y contraseña válidos para poder acceder a la aplicación.

Tras hacer *login* con éxito, se redirige al usuario a la página principal, que muestra las categorías existentes. Al seleccionar una categoría, se accede a sus productos. Como las categorías pueden ir cambiando a lo largo del tiempo, hay que cogerlos de las bases de datos.

Escaneado con CamScanner

En "Productos de la categoría" se muestran los datos de los productos de cierta categoría y se permite añadir un número variable de unidades al pedido. Si se añade algún producto, se redirige al usuario a "Ver carrito".

En "Ver carrito" se muestra en detalle el estado del pedido, se ofrece la posibilidad de eliminar productos y se puede confirmar el pedido. Al confirmar el pedido se muestra un mensaje de confirmación o error, según el caso.

En todas las páginas (salvo *login* y *logout*) habrá una cabecera con el nombre del restaurante y vínculos para:

- Ver carrito.
- Lista de categorías
- Cerrar la sesión.

4.3.4. El carrito de la compra

La estructura de datos utilizada para el carrito de la compra es uno de los puntos más importantes de la aplicación. Para almacenarlo se utilizará una variable de sesión.

El carrito será un *array* asociativo en el que las claves de los elementos representan el código de un producto, y el valor, el número de unidades pedidas de este producto.

El *array* comienza vacío. Cuando se añade un producto al pedido, hay que comprobar si ya hay en el *array* algún elemento que tenga como clave el código del producto. Si no lo hay, se añade un nuevo elemento tomando como clave el código y como valor el número de unidades.

Por ejemplo, si el carrito está vacío y se añaden 2 unidades del producto con código 4, se creará un elemento del *array* con clave 4 y valor 2.



Figura 4.5
Carrito de la compra con un elemento.

Si posteriormente se añaden 3 unidades del producto con código 5, el carrito será:

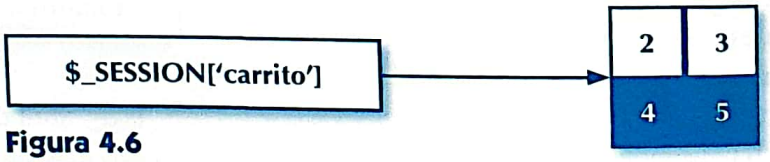


Figura 4.6
Se añade un nuevo producto.

Si ya hay un elemento con ese código, se suman las unidades al valor actual del elemento. Si se añaden otras cinco unidades del producto con código 4, el resultado será:

Escaneado con CamScanner

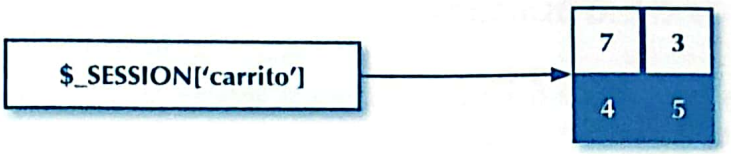


Figura 4.7
Se añade un nuevo producto.

De la misma manera, al eliminar productos del carrito habrá que buscar el elemento correspondiente en el *array* y restarle las unidades. Tras eliminar 3 unidades del producto con código 4, el carrito quedará así:

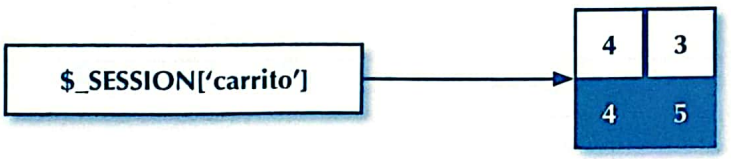


Figura 4.8
Se eliminan algunas unidades de un producto.

Cuando se eliminan todas las unidades de un producto, se suprime el elemento correspondiente en el *array*. Si para finalizar se eliminan 4 unidades del producto con código 4, se obtendrá:



Figura 4.9
Se eliminan todas las unidades de un producto.



Actividad propuesta 4.2

¿Crees que sería buena idea almacenar en el carrito todos los datos de los productos, y no solo el código? Analiza las ventajas e inconvenientes.

4.3.5. Control de acceso

Cuando se realiza *login* con éxito, se crea una nueva sesión y dos variables de sesión:

- Un *array* con dos campos. Uno para guardar el nombre del usuario (el correo del restaurante) y otro para su código, así no hay que buscarlo en la base de datos más adelante.
- La variable para el carrito de la compra.

El resto de los ficheros de la aplicación comienza uniéndose a la sesión y comprobando que la primera de estas variables existe. Si no se han creado, es que el usuario no ha hecho *login* y por tanto no puede acceder. En ese caso se le redirige a la página de *login*.

Escaneado con CamScanner

4.3.6. Ficheros de la aplicación

El siguiente paso es decidir qué ficheros formarán la aplicación y cómo se comunicarán entre ellos. Se resumen en la siguiente tabla.

CUADRO 4.1
Ficheros de la aplicación

Ruta	Descripción	Parámetros	Redirige a
login.php	Formulario de *login*	$_GET['redirigido'] $_POST['usuario'] $_POST['clave']	login.php?error=TRUE categorias.php
logout.php	Cierra la sesión		
sesiones.php	Comprueba que el usuario haya iniciado sesión correctamente		login.php (si no se ha iniciado sesión)
categorias.php	Muestra la lista de categorías con vínculos a productos.php?categoria=codigo		
cabecera.php	Cabecera con vínculos para ver el carrito, las categorías o cerrar sesión		
productos.php	Muestra los productos de la categoría, permite añadir al carro de la compra	$_GET['categoria'], el código de la categoría	
carrito.php	Muestra el carro de la compra, permite quitar productos y confirmar el pedido		
anadir.php	Añade productos al carrito	$_POST['cod'] $_POST['unidades']	carrito.php
eliminar.php	Elimina productos del carrito	$_POST['cod'] $_POST['unidades']	carrito.php
procesar_pedido.php	Inserta el pedido en la base de datos, envía correos de confirmación y muestra mensajes de error o éxito		
bd.php	Para agrupar las funciones de la base de datos		
correo.php	Funciones para enviar correo		

Escaneado con CamScanner

4.4. Implementación

Una vez completado el diseño de la aplicación, se puede comenzar con la implementación. Tomando como base la tabla anterior, en este apartado se van implementando todos los ficheros. Probablemente, el punto más complicado sean las funciones de acceso a la base de datos, que están agrupadas en bd.php.

4.4.1. *Login*

El fichero **login.php** incluye tanto el formulario HTML como el código para procesarlo. Al principio está el bloque PHP, que se ejecuta solo cuando el método es POST, es decir, cuando se envía el formulario.

```
1  <?php
2  require_once 'bd.php';
3  /*formulario de login habitual
4  si va bien abre sesión, guarda el nombre de usuario y redirige a princi-
   pal.php
5  si va mal, mensaje de error */
6  if ($_SERVER["REQUEST_METHOD"] == "POST") {
7
8      $usu = comprobar_usuario($_POST['usuario'], $_POST['clave']);
9      if($usu===FALSE){
10         $err = TRUE;
11         $usuario = $_POST['usuario'];
12     }else{
13         session_start();
14         //     $usu tiene campos correo y codRes, correo
15         $_SESSION['usuario'] = $usu;
16         $_SESSION['carrito'] = [];
17         header("Location: categorias.php");
18         return;
19     }
20 }
21 ?>
22 <!DOCTYPE html>
23 <html>
24     <head>
25         <title>Formulario de login</title>
26         <meta charset = "UTF-8">
27     </head>
28     <body>
29         <?php if(isset($_GET["redirigido"])){
30             echo "<p>Haga login para continuar</p>";
31         }?>
32         <?php if(isset($err) and $err == TRUE){
33             echo "<p> Revise usuario y contraseña</p>";
34         }?>
35         <form action = "<?php echo htmlspecialchars($_SERVER["PHP_
           SELF"]);?>" method = "POST">
36             <label for = "usuario">Usuario</label>
37             <input value = "<?php if(isset($usuario))echo $usuario;?>"
38             id = "usuario" name = "usuario" type = "text">
```


Escaneado con CamScanner

```
39              <label for = "clave">Clave</label>
40              <input id = "clave" name = "clave" type = "password">
41              <input type = "submit">
42          </form>
43      </body>
44  </html>
```

En la línea 8 comprueba usuario y contraseña. Si son correctos, crea una nueva sesión y la variables de sesión para el carrito y para el usuario. Para el usuario se crea una variable $_SESSION['usuario'] con dos campos, 'correo' y 'codRes'. Para finalizar, reenvía al usuario a ca **tegorias.php**, la página principal.

Si el *login* no es correcto o se accede al fichero por GET, se muestra el formulario HTML. Además de los elementos básicos, se pueden mostrar dos mensajes de error:

- Si el parámetro "redirigido" está presente en la URL, se muestra un mensaje indicando que se debe iniciar sesión para continuar (línea 29). El *script* que comprueba si se ha iniciado sesión, **sesiones.php**, redirige a login.php?redirigido=true cuando no encuentra la variable `$_SESSION['usuario']`.
- Si se ha enviado el formulario, pero los datos no son correctos, el primer bloque de PHP crea la variable `$err` con valor TRUE. Esto se comprueba en la línea 32 y se muestra el mensaje correspondiente.

4.4.2. Control de acceso

Para comprobar que solo puedan acceder a la aplicación los usuarios que hayan hecho *login*, se utiliza la función `comprobar_sesion()` del fichero **sesiones.php**.

```
<?php
function comprobar_sesion(){
    session_start();
    if(!isset($_SESSION['usuario'])){
    header("Location: login.php?redirigido=true");
    }
}
```

Simplemente se une a la sesión existente y comprueba que la variable `$_SESSION['usuario']` exista. Si no es el caso, quiere decir que el usuario no ha hecho *login* correctamente y por tanto, se le redirige al formulario de *login*. El parámetro redirigido hace que se muestre un mensaje de error junto con el formulario.

El resto de las páginas de la aplicación (excepto **login.php** y **logout.php**) llaman a esta para restringir el acceso de manera que solo puedan acceder los usuarios que hayan hecho *login* con éxito.

4.4.3. La cabecera

Esta es la cabecera común a todas las páginas una vez se hace *login*. Muestra el nombre del usuario utilizando la variable de sesión y vínculos para cerrar la sesión y para volver a la página principal

Escaneado con CamScanner

```
<header>
    Usuario: <?php echo $_SESSION['usuario']['correo']?>
    <a href="categorias.php">Home</a>
    <a href="carrito.php">Ver carrito</a>
    <a href="logout.php">Cerrar sesión</a>
</header>
<hr>
```

4.4.4. Lista de categorías

Este fichero es la página principal de la aplicación. En el cuerpo de la página se muestra una lista con las categorías. Cada elemento de la lista es un vínculo con el nombre de la categoría y un vínculo a productos.php?categoria=<cod>.

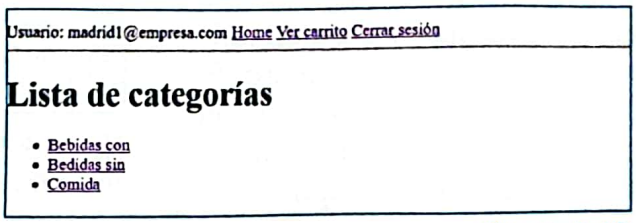


Figura 4.10
Lista de categorías.

```
<?php
/*comprueba que el usuario haya abierto sesión o redirige*/
require 'sesiones.php';
require_once 'bd.php';
comprobar_sesion();
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset = "UTF-8">
    <title>Lista de categorías</title>
</head>
    <body>
        <?php require 'cabecera.php';?>
        <h1>Lista de categorías</h1>
        <!--lista de vínculos con la forma
        productos.php?categoria=1-->
        <?php
        $categorias = cargar_categorias();
        if($categorias===FALSE){
            echo "<p class='error'>Error al conectar con la base datos</
            p>";
        }else{
            echo "<ul>"; //abrir la lista
            foreach($categorias as $cat){
                /*$cat['nombre] $cat['codCat']*/
                $url = "productos.php?categoria=".$cat['codCat'];
                echo "<li><a href='$url'>".$cat['nombre']."</a></li>";
            }
            echo "</ul>";
        }
        ?>
```


Escaneado con CamScanner

```
    </body>
</html>
```

4.4.5. Tabla de productos

El fichero **productos.php** muestra una tabla con todos los elementos de una categoría y permite añadirlos al carrito. Al mostrar la tabla de productos, se añade a cada fila un formulario para añadir 1 o más unidades de ese producto al carrito. El formulario contiene campos para el código del producto, las unidades y un botón de envío. El campo con el código está oculto, no se muestra al usuario. Se envía a **anadir.php**.

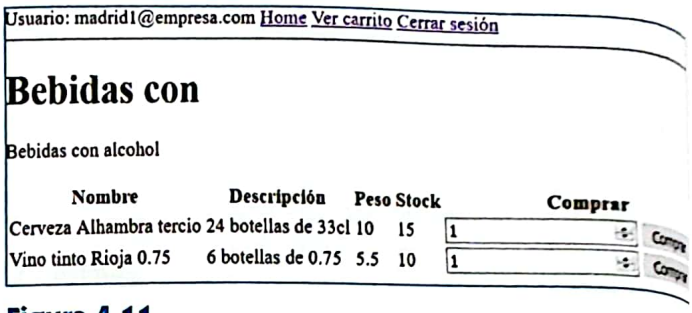


Figura 4.11
Tabla de productos.

```
<?php
    /*comprueba que el usuario haya abierto sesión o redirige*/
    require 'sesiones.php';
    require_once 'bd.php';
    comprobar_sesion();
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>Tabla de productos por categoría</title>
    </head>
    <body>
    <?php
        require 'cabecera.php';
        $cat = cargar_categoria($_GET['categoria']);
        $productos = cargar_productos_categoria($_GET['categoria']);
        if($cat=== FALSE or $productos === FALSE){
            echo "<p class='error'>Error al conectar con la base datos
            p>";
            exit;
        }
        echo "<h1>". $cat['nombre']. "</h1>";
        echo "<p>". $cat['descripcion']."</p>";
        echo "<table>"; //abrir la tabla
        echo "<tr><th>Nombre</th><th>Descripción</th>
              <th>Peso</th><th>Stock</th><th>Comprar</th></tr>";
        foreach($productos as $producto){
            $cod = $producto['CodProd'];
```


Escaneado con CamScanner

```
            $nom = $producto['Nombre'];
            $des = $producto['Descripcion'];
            $peso = $producto['Peso'];
            $stock = $producto['Stock'];
        echo "<tr><td>$nom</td><td>$des</td><td>$peso</td><td>$stock</td>
            <td>
              <form action = 'anadir.php' method = 'POST'>
               <input name = 'unidades' type='number' min = '1'value = '1'>
               <input type = 'submit' value='Comprar'>
               <input name = 'cod' type='hidden' value = '$cod'>
              </form>
            </td>
        </tr>";
        }
      echo "</table>"
  ?>
  </body>
</html>
```

Actividad propuesta 4.3

¿Qué opciones se te ocurren para añadir productos en lugar de un formulario? Analiza sus ventajas e inconvenientes.

4.4.6. Añadir productos

El fichero **anadir.php** se encarga de añadir elementos al carrito. No tiene salida, modifica la variable de sesión y redirige a **carrito.php**. Recibe los datos del formulario de **productos.php** y los parámetros `cod` y `unidades` que recibe para modificar el carrito. Primero comprueba si el código ya existe en el *array*. Si existe, se suma `unidades` al valor existente. Si no existe, se crea con valor `unidades`.

```
<?php
    /*comprueba que el usuario haya abierto sesión o redirige*/
    require_once 'sesiones.php';
    comprobar_sesion();
    $cod = $_POST['cod'];
    $unidades = (int)$_POST['unidades'];
    /*si existe el código sumamos las unidades*/
    if(isset($_SESSION['carrito'][$cod])){
       $_SESSION['carrito'][$cod] += $unidades;
    }else{
       $_SESSION['carrito'][$cod] = $unidades;
    }
    header("Location: carrito.php");
```


Escaneado con CamScanner

Actividad propuesta 4.4

El fichero anterior permite añadir más unidades de las disponibles para un producto. ¿Qué alternativas se te ocurren para impedirlo?

4.4.7. El carrito de la compra

Muestra una tabla con una fila por cada producto (diferente) del carrito. La fila muestra los datos del producto y el número de unidades pedidas. Además, en cada fila hay un formulario para eliminar ese producto del carrito. El funcionamiento es análogo al del formulario para añadir de **productos.php**.

El formulario incluye el número de unidades que hay que eliminar y el código del producto, este último como campo oculto. Se envía a **eliminar.php**.

```
<?php
    /*comprueba que el usuario haya abierto sesión o redirige*/
    require_once 'sesiones.php';
    require_once 'bd.php';
    comprobar_sesion();
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>Carrito de la compra</title>
    </head>
    <body>
        <?php
        require 'cabecera.php';
        echo "<h2>Carrito de la compra</h2>";
        $productos = cargar_productos(array_keys($_SESSION['carrito']));
        if($productos === FALSE){
            echo "<p>No hay productos en el pedido</p>";
            exit;
        }
        echo "<h2>Carrito de la compra</h2>";
        echo "<table>"; //abrir la tabla
        echo "<tr><th>Nombre</th><th>Descripción</th><th>Peso</th>
        <th>Unidades</th><th>Eliminar</th></tr>";
        foreach($productos as $producto){
            $cod = $producto['CodProd'];
            $nom = $producto['Nombre'];
            $des = $producto['Descripcion'];
            $peso = $producto['Peso'];
            $unidades = $_SESSION['carrito'][$cod];
```

Escaneado con CamScanner

```
            echo "<tr><td>$nom</td><td>$des</td><td>$peso</td><td>$unida-
            des</td>
            <td>
                <form action = 'eliminar.php' method = 'POST'>
                    <input name = 'unidades' type='number' min = '1'value =
                    '1'>
                    <input type = 'submit' value='Eliminar'></td>
                    <input name = 'cod' type='hidden' value ='$cod'>
                </form>
            </td>
        </tr>";
        }
            echo "</table>";
        ?>
        <hr>
        <a href = "procesar_pedido.php">Realizar pedido</a>
    </body>
</html>
```

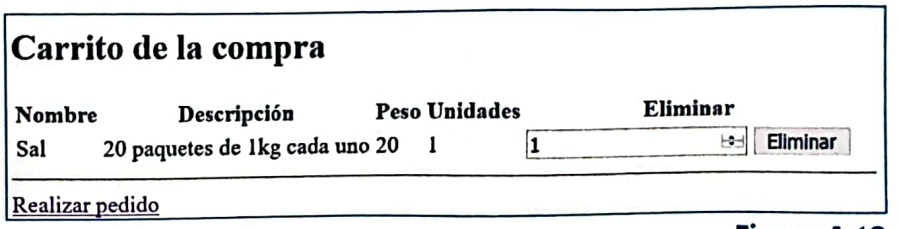


Figura 4.12
Carrito de la compra.

4.4.8. Eliminar productos

Este *script* se encarga de eliminar elementos del carrito. No tiene salida, modifica la variable de sesión y redirige a **carrito.php**. Lo primero que hace es comprobar que los parámetros `cod` y `unidades` estén presentes. Para modificar el carrito, primero comprueba si el código ya existe en el *array*. Si existe, se resta unidades al valor existente. Si el valor resultante es menor o igual que cero, se elimina ese elemento del *array*.

Si no existe, no hace nada, porque no hay nada que eliminar. Como a **eliminar.php** se accede desde los vínculos de **carrito.php**, que se generan a partir del carrito, en principio este caso no debería darse.

```
<?php
/*comprueba que el usuario haya abierto sesión o redirige*/
require_once 'sesiones.php';
comprobar_sesion();
$cod = $_POST['cod'];
$unidades = $_POST['unidades'];
/*si existe el código restamos las unidades, con mínimo de 0*/
if(isset($_SESSION['carrito'][$cod])){
    $_SESSION['carrito'][$cod] -= $unidades;
    if($_SESSION['carrito'][$cod] <= 0){
        unset($_SESSION['carrito'][$cod]);
```


Escaneado con CamScanner

```
        }
    }
header("Location: carrito.php");
```

4.4.9. Procesamiento del pedido

Este proceso consiste en:

- Insertar al pedido usando la función `insertar_pedido()`.
- Si el pedido se inserta correctamente:
  - Llamar a `enviar_correos()` para mandar los correos de confirmación.
  - Vaciar el carrito.
- Mostrar mensajes de confirmación o error.

```
<?php
    /*comprueba que el usuario haya abierto sesión o redirige*/
    require '..\correo\enviar_correo.php';
    require 'sesiones.php';
    require_once 'bd.php';
    comprobar_sesion();
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>Pedidos</title>
    </head>
    <body>
        <?php
        require 'cabecera.php';
        $resul = insertar_pedido($_SESSION['carrito'],
            $_SESSION['usuario']['codRes']);
        if($resul === FALSE){
            echo "No se ha podido realizar el pedido<br>";
        }else{
            $correo = $_SESSION['usuario']['correo'];
            echo "Pedido realizado correctamente<br>";
            //vaciar carrito
            $compra = $_SESSION['carrito'];
            $_SESSION['carrito'] = [];
            echo "Pedido realizado con éxito.
            Se enviará un correo de confirmación a: $correo ";
                enviar_correos($compra, $pedido, $correo);
        }
    ?>
```


Escaneado con CamScanner

```
    </body>
</html>
```

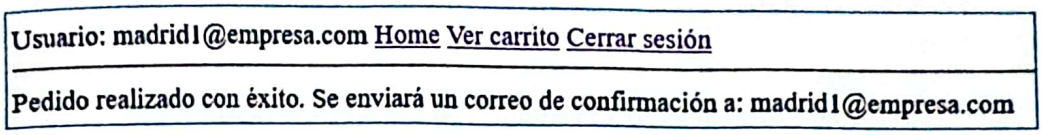


Figura 4.13
Confirmación del pedido.



Actividad propuesta 4.5

Las tiendas online suelen pedir la confirmación del usuario antes de procesar el envío. ¿Qué cambios habría que hacer en la aplicación para incluir ese paso?

4.4.10. La base de datos

Las funciones para manejar la base de datos se agrupan en el fichero **bd.php**. La base de datos se utiliza para controlar el accesos al sistema, al ver las categorías y productos y al realizar el pedido.

La siguiente tabla resume las funciones para que relacionadas con la base de datos.

CUADRO 4.2
Funciones de bd.php

Función	Descripción
leer_config($nombre, $esquema)	Devuelve la configuración de la base de datos
cargar_categorias()	Devuelve un cursor con las categorías
cargar_productos_categoria($codCat)	Devuelve un cursor con los productos de la categoría
cargar_categoria($codCat)	Devuelve los datos de la categoría
comprobar_usuario($nombre, $clave);	Comprueba usuario y clave en la base de datos
cargar_productos($codigosProductos)	Devuelve un cursor de productos a partir de sus códigos
insertar_pedido($carrito, $codRes)	Inserta el pedido en la base de datos

A) Configuración

Para almacenar los datos de configuración de la base de datos se usa el fichero **configuracion.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuracion>
```


Escaneado con CamScanner

```
    <ip>127.0.0.1</ip>
    <nombre>pedidos</nombre>
    <usuario>root</usuario>
    <clave></clave>
</configuracion>
```

Este fichero tiene que cumplir el esquema `configuracion.xsd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDe-
fault="qualified">
    <xs:element name="configuracion">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ip" type="xs:string"/>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="usuario" type="xs:string"/>
                <xs:element name="clave" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Todas las funciones que acceden a la base de datos utilizan los datos de este fichero (a tra de la función `leer_config()`.).

B) Descripción de las funciones

1. function leer_config($nombre, $esquema);

- $nombre es la ruta del fichero con los datos de conexión a la base de datos.
- $esquema es la ruta del fichero XSD para validar el anterior.

Si el fichero de configuración existe y es válido, devuelve un *array* con 3 valores: la cad de conexión, el nombre de usuario y la clave.

Si no encuentra el fichero o no es válido, lanza una excepción.

2. function cargar_categorias();

Devuelve un cursor con el código y nombre de las categorías de la base de datos.

```
function cargar_categorias(){
    $res = leer_config(dirname(__FILE__)."/configuracion.xml",
    dirname(__FILE__)."/configuracion.xsd");
    $bd = new PDO($res[0], $res[1], $res[2]);
```


Escaneado con CamScanner

```
    $ins = "select codCat, nombre from categorias";
    $resul = $bd->query($ins);
    if (!$resul) {
        return FALSE;
    }
    if ($resul->rowCount() === 0) {
        return FALSE;
    }
    //si hay 1 o más
    return $resul;
}
```

3. function cargar_categoria($codCat);

Recibe como argumento el código de una categoría y devuelve un *array* con su nombre y descripción. Si hay algún error con la base de datos o la categoría no existe, devuelve FALSE.

4. function comprobar_usuario($nombre, $clave);

Esta es la función que se usa para comprobar los datos del formulario de *login*. Si los datos son correctos, devuelve un *array* con dos campos: 'codRes', el código del restaurante y 'correo', su correo. Si hay algún error con la base de datos o los datos no son correctos, devuelve FALSE.

5. function cargar_productos_categoria($codCat);

Recibe como argumento el código de una categoría y devuelve un cursor con sus productos. Incluye todas las columnas de la base de datos. Si hay algún error con la base de datos, la categoría no existe o no tiene productos, devuelve FALSE.

6. function cargar_productos($codigosProductos);

Recibe como argumento un *array* de códigos de productos y devuelve un cursor con todas las columnas de estos. Si hay algún error con la base de datos, devuelve FALSE. Esta función se usa al mostrar el carrito de la compra.

```
function cargar_productos($codigosProductos){
    $res = leer_config(dirname(__FILE__)."/configuracion.xml",
    dirname(__FILE__)."/configuracion.xsd");
    $bd = new PDO($res[0], $res[1], $res[2]);
    $texto_in = implode(",", $codigosProductos);
    $ins = "select * from productos where codProd in($texto_in)";
    $resul = $bd->query($ins);
    if (!$resul) {
```


Escaneado con CamScanner

```
            return FALSE;
        }
        return $resul;
    }
```

7. function insertar_pedido($carrito, $codRes);

Esta función es la que se encarga de insertar el pedido en la base de datos. Recibe el c de la compra y el código del restaurante que realiza el pedido. Si todo va bien, devuelve digo del nuevo pedido. Si hay algún error devuelve FALSE.

Se encarga de transformar el carrito de la compra en un pedido en la base de datos, para l

- Crea una nueva fila en la tabla pedidos.
- Crea una fila en la tabla PedidosProductos por cada producto diferente que se pid que usar la clave del nuevo pedido.

Las inserciones deben ocurrir como una transacción. Si alguna falla, hay que desha anteriores.

```
function insertar_pedido($carrito, $codRes){
    $res = leer_config(dirname(__FILE__)."/configuracion.xml", dirname(_
    FILE__)."/configuracion.xsd");
    $bd = new PDO($res[0], $res[1], $res[2]);
    $bd->beginTransaction();
    $hora = date("Y-m-d H:i:s", time());
    // insertar el pedido
    $sql = "insert into pedidos(fecha, enviado, restaurante)
        values('$hora',0, $codRes)";
    $resul = $bd->query($sql);
        if (!$resul) {
        return FALSE;
    }
    // coger el id del nuevo pedido para las filas detalle
    $pedido = $bd->lastInsertId();
    // insertar las filas en pedidoproductos
    foreach($carrito as $codProd=>$unidades){
        $sql = "insert into pedidosproductos(Pedido, Producto,
        Unidades)values($pedido, $codProd, $unidades)";
        echo $sql;
        $resul = $bd->query($sql);
        if (!$resul) {
            $bd->rollback();
            return FALSE;
        }
        $sql = "update productos set stock = stock - $unidades
where codProd = $codProd";
```

Escaneado con CamScanner

```
$resul = $bd->query($sql);
if (!$resul) {
    $bd->rollback();
    return FALSE;
}
    }
    $bd->commit();
    return $pedido;
}
```

4.4.11. Envío de correos

Si todo ha ido bien, envía un correo de confirmación al restaurante que lo ha realizado y al Departamento de Pedidos. Es el mismo correo para los dos. El correo incluirá el número del pedido, el restaurante que lo realiza y una tabla HTML con los productos del pedido, bastante parecida a la del carrito.

Las funciones para enviar los correos están en el fichero **correo.php**:

1. crear_correo($carrito, $pedido, $correo);

```
function crear_correo($carrito, $pedido, $correo){
    $texto = "<h1>Pedido nº $pedido </h1><h2>Restaurante:
    $correo </h2>";
    $texto.= "Detalle del pedido:";
    $productos = cargar_productos(array_keys($carrito));
    $texto.= "<table>"; //abrir la tabla
    $texto.= "<tr><th>Nombre</th><th>Descripción</th><th>Peso</th>
    <th>Unidades</th><th>Eliminar</th></tr>";
    foreach($productos as $producto){
        $cod = $producto['CodProd'];
        $nom = $producto['Nombre'];
        $des = $producto['Descripcion'];
        $peso = $producto['Peso'];
        $unidades = $_SESSION['carrito'][$cod];
        $texto.= "<tr><td>$nom</td><td>$des</td><td>$peso</td>
        <td>$unidades</td><td></tr>";
    }
    $texto.= "</table>";
    return $texto;
}
```

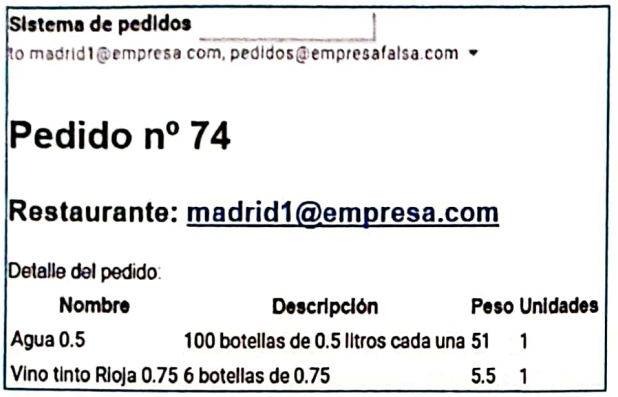
2. enviar_correo_multiples ($lista_correos, $cuerpo, $asunto = "");

Recibe un *array* de direcciones de correo, el cuerpo del correo y opcionalmente el asunto. Envía el correo a todas las direcciones.

Escaneado con CamScanner

3. enviar_correos($carrito, $pedido, $correo);

Recibe el carrito de la compra, el número de pedido y el correo del restaurante que lo hace. Prim
llama a la función `crear_correo()` para crear el cuerpo, y luego llama `enviar_correo_multiples`


Sistema de pedidos
to madrid1@empresa.com, pedidos@empresafalsa.com

Pedido nº 74

Restaurante: madrid1@empresa.com

Detalle del pedido:

Nombre	Descripción	Peso	Unidades
Agua 0.5	100 botellas de 0.5 litros cada una	51	1
Vino tinto Rioja 0.75	6 botellas de 0.75	5.5	1

Figura 4.14
Confirmación del pedido.



TOMA NOTA

Hay que tener instalado PHPMai
ler, también hay que introducir lo
datos de la cuenta de Gmail y per
mitir el acceso a aplicaciones me
nos seguras, como se explica en el
capítulo 3.

Resumen

- La aplicación permite que los restaurantes de una cadena realicen pedidos al Departamento de Pedidos.
- Muestra los productos disponibles organizados por categorías.
- El carrito de la compra es una variable de sesión. Es un *array* en que la clave de cada elemento representa el código del producto, y su valor, las unidades pedidas.
- El carrito se manipula a través de formularios que envían el código del producto y las unidades que hay que añadir o eliminar.
- Al añadir productos al carrito, si no hay ya un elemento con ese código, se crea. Si lo hay, se suman las unidades.
- Al eliminar productos del carrito hay que comprobar si quedan 0 o menos unidades para eliminar el elemento correspondiente.
- Al pulsar el vínculo de realizar pedido, los datos del carrito se utilizan para modificar la base de datos.
- El pedido se realiza como una transacción; si hay algún error con la base de datos, se deshacen las operaciones anteriores.
- Si el pedido se realiza con éxito, se envían correos de confirmación al Departamento de Pedidos y al restaurante que lo realiza.
- Hay muchas ampliaciones sencillas por hacer: control de *stock*, panel de administración, contraseñas encriptadas...

Escaneado con CamScanner

Ejercicios propuestos



1. Al añadir un producto, la aplicación redirige al carrito. Modifícala para que redirija a la tabla de productos, con la misma categoría.
2. Haz los cambios necesarios para encriptar las contraseñas de la tabla de usuarios (utiliza la función bcrypt()).
3. Modifica la tabla de productos para que no muestre los productos sin *stock*.
4. Añade el peso total del pedido en el contenido del correo de confirmación.
5. Almacena la información del servidor de correo en un fichero de XML como se hace con el de la base de datos. Modifica las funciones de envío de correo para que utilicen ese fichero.
6. Crea un esquema XSD para la configuración de correo y valida con él el fichero de configuración antes de usarlo.

ACTIVIDADES DE AUTOEVALUACIÓN

1. Para el carrito de la compra se utiliza:
   - ☐ *a)* Una variable de sesión que almacena todos los datos de los productos pedidos.
   - ☐ *b)* Una variable de sesión que almacena el código de los productos pedidos.
   - ☐ *c)* Una variable *cookie*.
2. La tabla de pedidos tiene:
   - ☐ *a)* Una clave ajena.
   - ☐ *b)* Dos claves ajenas.
   - ☐ *c)* Ninguna clave ajena.
3. El sistema controla el *stock* de los productos:
   - ☐ *a)* Al añadirlos al carrito.
   - ☐ *b)* Al procesar el pedido.
   - ☐ *c)* No lo comprueba.
4. Para realizar un pedido hay que llevar a cabo inserciones o actualizaciones en:
   - ☐ *a)* Tres tablas.
   - ☐ *b)* Dos tablas.
   - ☐ *c)* Una tabla.
5. La variable de sesión para el carrito:
   - ☐ *a)* Se crea al iniciar sesión.
   - ☐ *b)* Se crea al añadir el primer producto.
   - ☐ *c)* Se crea al procesar el pedido.

Escaneado con CamScanner

6. La tabla de productos enlaza con anadir.pnp:
   - ☐ *a)* Con un vínculo.
   - ☐ *b)* Con JavaScript.
   - ☐ *c)* Con un formulario.

7. anadir.php y eliminar.php reciben como parámetros POST:
   - ☐ *a)* El carrito, el código y el número de unidades.
   - ☐ *b)* El carrito.
   - ☐ *c)* El código y el número de unidades.

8. Los datos de conexión a la base de datos se introducen:
   - ☐ *a)* En un fichero XML.
   - ☐ *b)* En un fichero de texto plano.
   - ☐ *c)* Directamente en las funciones que usan la base de datos.

9. Al abrir sesión:
   - ☐ *a)* Se crea una variable de sesión para guardar el correo del restaurante.
   - ☐ *b)* Se crean dos variables de sesión para guardar el código y el correo del restaurante.
   - ☐ *c)* Se crea una variable de sesión que almacena código y correo en un *array*.

10. Los datos del servidor de correo electrónico se introducen:
    - ☐ *a)* En un fichero XML.
    - ☐ *b)* En un fichero de texto plano.
    - ☐ *c)* Directamente en la función de correo.

SOLUCIONES:

1.	a **b** c	**5.**	**a** b c
2.	a **b** c	**6.**	a b **c**
3.	a b **c**	**7.**	a b **c**
4.	**a** b c	**8.**	**a** b c
9.	a b **c**		
10.	a b **c**		

Escaneado con CamScanner

5

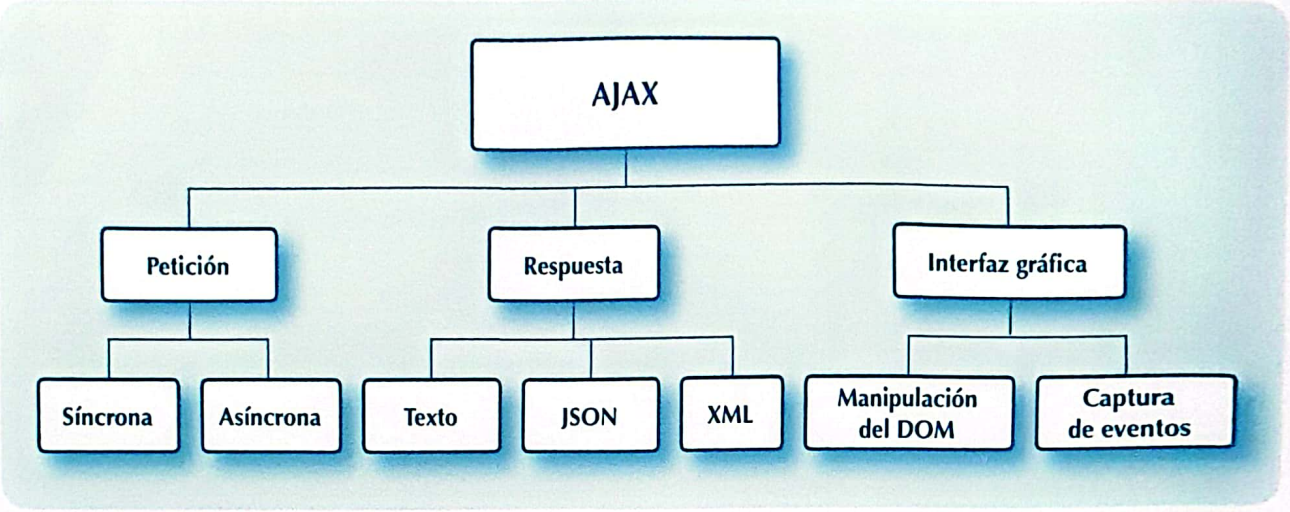
Aplicaciones web dinámicas con AJAX

Objetivos

- ✓ Comprender las ventajas de separar la lógica de presentación y la de negocio.
- ✓ Entender el papel de JavaScript dentro de las aplicaciones web.
- ✓ Presentar la técnica AJAX.
- ✓ Entender el funcionamiento de las peticiones asíncronas.
- ✓ Utilizar JSON para las respuestas del servidor.
- ✓ Conocer los principios de las aplicaciones de una sola página.

Escaneado con CamScanner

Mapa conceptual



Glosario

AJAX. *Asyncronous JavaScript and XML*. Técnica de diseño de aplicaciones web que se basa en realizar peticiones al servidor desde JavaScript.

Aplicación web de una sola página. Aplicación en la que todas las peticiones al servidor, salvo la inicial, se hacen de forma dinámica utilizando AJAX.

DOM. Interfaz que permite acceder y manipular el contenido de un documento.

JavaScript. Lenguaje de programación en el lado del cliente. Es uno de los elementos básicos del desarrollo web.

JSON. *JavaScript ObjectNotation*. Lenguaje para intercambio de datos en formato texto muy extendido en desarrollo web.

Lógica de negocio. Es la lógica específica de la aplicación, donde realmente se desarrolla la funcionalidad de esta. Procesa la información que introduce el usuario y maneja la base de datos.

Lógica de presentación. Parte de la aplicación que se ocupa de mostrar la información al usuario.

Petición asíncrona. Cuando se realiza una petición asíncrona, el *script* sigue ejecutándose mientras se espera una respuesta.

Petición síncrona. Cuando se realiza una petición síncrona, la ejecución del *script* se detiene hasta que se recibe la respuesta.

5.1. Separación de la lógica de negocio

Uno de los temas principales en el diseño de aplicaciones es la reusabilidad del código. Para conseguirlo, las aplicaciones se dividen en partes que se relacionan entre sí, pero que también tienen sentido de manera independiente.

Escaneado con CamScanner

A pequeña escala, se crean funciones y clases que, si están bien diseñadas, se pueden reutilizar en otras aplicaciones. A nivel de diseño de aplicaciones, se plantea una arquitectura con varios componentes desacoplados que se ocupan de las diferentes partes de la aplicación.

En los ejemplos realizados hasta ahora, la lógica de negocio y la de presentación están completamente mezcladas. Un ejemplo está en la tabla de productos de la aplicación del capítulo anterior. Hay varios bloques de PHP, etiquetas de HTML en la plantilla y otras etiquetas HTML generadas mediante `echo`. Incluso en un ejemplo sencillo, el código se complica rápidamente.

Para mejorar esta situación, el primer paso sería aislar la salida en funciones específicas que se ocuparan exclusivamente de la salida. Mejor aún, se podría utilizar una librería de plantillas, como se verá en el capítulo 7.

Otro enfoque consiste en llevar la lógica de presentación al cliente. Al fin y al cabo, en el cliente es donde se muestra la información. El lenguaje más extendido en el lado del cliente es JavaScript, que se ejecuta dentro del navegador. Aunque no ha aparecido hasta ahora, es un elemento fundamental en el desarrollo de aplicaciones web. Tiene muchas utilidades, pero en este capítulo se utiliza para realizar peticiones al servidor y mostrar los datos recibidos.

AJAX es una técnica de diseño de aplicaciones web que se basa en hacer peticiones al servidor desde JavaScript. En los capítulos anteriores la salida de la página web se genera por completo en el servidor. Por ejemplo, la tabla de productos de la aplicación de pedidos. En AJAX, simplificando, el servidor devuelve solo los datos que hay que mostrar. Es decir, los datos de los productos. En el cliente, la interfaz se actualiza según los datos recibidos, también utilizando JavaScript.

De esta manera se desacopla la lógica de negocio de la de presentación. Tiene varias ventajas:

- El código es más fácil de mantener y modificar, al no estar mezclado.
- El código del servidor se puede reutilizar con otros clientes, como una aplicación de móvil.
- Se puede trabajar en la parte del cliente y en la del servidor en paralelo.

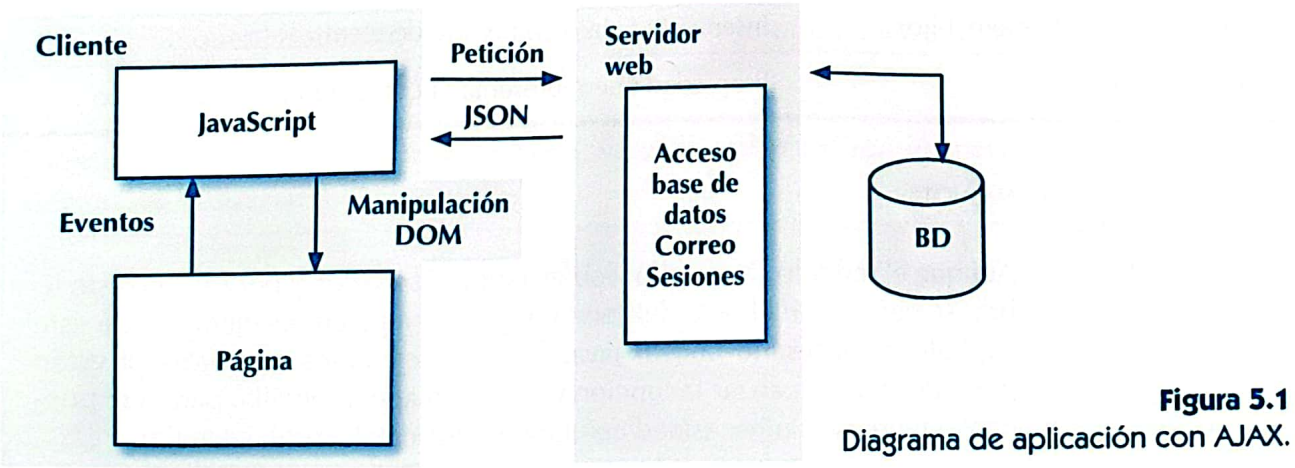


Figura 5.1
Diagrama de aplicación con AJAX.

5.2. Tecnologías y librerías asociadas

El código JavaScript también llega al cliente desde el servidor incluido en páginas HTML. Se ejecuta en el navegador y puede manipular la página para cambiar su aspecto, mostrar mensajes de alerta y establecer comunicaciones con el servidor.

Escaneado con CamScanner

Para modificar la página web, JavaScript utiliza el DOM, un estándar del W3C. Es una interfaz que permite acceder y modificar el contenido, la estructura y el estilo de un documento HTML. Los documentos se representan mediante una estructura de árbol en la que los nodos son los elementos HTML. Los elementos forman parte de una jerarquía y se puede hablar de elementos padres, hijo, descendientes y antecesores

Por ejemplo, para una página sencilla:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ejemplo DOM</title>
    </head>
    <body>
    </body>
</html>
```

El elemento `html` es el nodo raíz, y es el padre de `head` y `body`. También es antecesor del resto de elementos de la página. De la misma manera, `title` es hijo de `head` y descendiente de `html`.

Desde JavaScript se puede modificar el contenido de los elementos, sus atributos y su estilo. En el cuadro 5.1 se resumen las funciones que se usan a lo largo del capítulo para trabajar con el DOM.

CUADRO 5.1
Funciones y atributos básicos para manipular el DOM

Función	Descripción
document.createElement(<etiqueta>)	Crea un nuevo elemento
document.getElementById(id)	Devuelve el elemento de la página que tiene ese id
elem.appendChild(elem_hijo)	Inserta elem_hijo como hijo de elem
elem.innerHTML	Propiedad que representa el contenido de un elemento



TOMA NOTA

Aunque el módulo Desarrollo web en entorno servidor, y por lo tanto este libro, se centran en el lado del servidor, para completar los ejemplos de este capítulo es necesario utilizar JavaScript. Los ejemplos de JavaScript están pensados para alcanzar la funcionalidad de manera sencilla, para que puedan entenderse sin necesidad de entrar en los detalles del lenguaje.

5.3. Obtención remota de información

Hasta ahora, se ha visto que se solicita una página al servidor:

- Cuando se accede a una página introduciendo la URL en el navegador.
- Al seguir un vínculo.

Escaneado con CamScanner

- Al enviar un formulario.
- Como consecuencia de una redirección.

En todos los casos, el navegador cambia el contenido que esté mostrando con la respuesta que obtiene del servidor. Este modelo es bastante restrictivo y condiciona el diseño de las páginas, ya que cada interacción con el servidor requiere que este acabe devolviendo una página web completa.

Es posible superar esta limitación utilizando JavaScript, en concreto AJAX. Por ejemplo, una página web puede incluir un *script* que cada minuto consulte las últimas noticias y muestre los titulares en una sección lateral. El *script* se encarga de obtener los datos y de modificar el contenido solo de esa sección, sin tener que recargar la página.

El elemento central de AJAX es el objeto `XMLHttpRequest`, que representa una petición al servidor. Utilizándolo es posible hacer una petición al servidor sin que su respuesta se muestre automáticamente como una nueva página en el navegador. En lugar de eso, la respuesta se procesa mediante JavaScript.

5.3.1. Peticiones síncronas y asíncronas

Las peticiones pueden ser *síncronas* o *asíncronas*. En el primer caso, la ejecución se detiene hasta que se recibe la respuesta. En el segundo, la ejecución continúa y, cuando se recibe una respuesta, se procesa.

Los métodos básicos para enviar una petición son:

1. open(método, destino, asíncrona). Recibe el método HTTP con el que se realiza la petición, la ruta que se solicita y si la petición es asíncrona o no.
2. send([params]). Realiza el envío. Tiene un argumento opcional para añadir parámetros a la petición.
3. setRequestHeader(cabecera). Para establecer las cabeceras de la petición. Se usa, si es necesario, entre `open()` y `send()`.

Por ejemplo, para pedir la ruta **hora_servidor.php** usando el método GET de manera síncrona, se utilizaría:

```
var xhttp = new XMLHttpRequest();
xhttp.open("GET", "hora_servidor.php", false);
xhttp.send();
```

Al ser una petición síncrona, la ejecución del *script* se detiene hasta que se recibe una respuesta o pasa el tiempo máximo de espera. Una vez recibida, se almacena en la propiedad `response`. El código HTTP de la respuesta se almacena en `status`.

```
if (xhttp.status == 200){
   alert("OK");
}else {
   alert("Error");
}
alert(xhttp.response);
```


Escaneado con CamScanner

RECUERDA

Si la petición utiliza el método POST, hay que incluir una cabecera específica y añadir una cadena de parámetros.

```
xhttp.open("POST", ruta, true);
xhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xhttp.send("param1=value1&param2=value2");
```

En las peticiones asíncronas, que son las más habituales, la situación se complica un poco. Al realizar la petición, el *script* sigue ejecutándose sin esperar a la respuesta. Por lo tanto, hay que indicar de alguna manera qué código se encargará de gestionar la respuesta cuando llegue.

Para hacerlo se utiliza la propiedad `onreadystatechange`, que permite asociar una función a un cambio en el estado de la petición. El estado de la petición se representa mediante el atributo `readystate`, que puede tomar los valores recogidos en el cuadro 5.2.

CUADRO 5.2
Estado de una petición

Valor	Nombre	Descripción
0	UNSET	No se ha llamado a open()
1	OPENED	Se ha llamado a open(), pero no a send()
2	HEADERS_RECEIVED	Se han recibido las cabeceras de la respuesta
3	LOADING	Se está recibiendo el cuerpo de la respuesta
4	DONE	Se ha recibido la respuesta

El estado de la petición va cambiando según se desarrolla la comunicación con el servidor. En cada cambio, se llama a la función indicada en `onreadystatechange`.

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        alert(this.response);
    }
};
xhttp.open("GET", "hora_servidor.php", true);
xhttp.send();
```

Es bastante habitual que estas funciones comiencen comprobando que el estado es 4, comunicación finalizada, y que el servidor ha devuelto el código 200, que indica que no ha habido errores. La primera comprobación es necesaria para que no se ejecute el código cada vez que cambia el estado.

Escaneado con CamScanner

TOMA NOTA

Si la petición se resuelve sin problemas, el servidor devuelve el código 200 (*OK*). Otros códigos habituales son 404 (*Not Found*), 400 (*Bad Request*) o 500 (*Internal Server Error*).

Actividad propuesta 5.1

Modifica el código precedente para que compruebe el código de error 404 y muestre una alerta cuando ocurra.

5.4. Respuesta del servidor

Como no se espera que la respuesta sea una página, no tiene por qué consistir en HTML. En casos sencillos puede ser texto sin formato. En los ejemplos anteriores se llama a **hora_servidor.php**. Este fichero devuelve simplemente una cadena de texto con la hora.

```
<?php
echo date("h:m:s");
```

Para devolver datos estructurados (como una lista de productos) se puede utilizar JSON o XML. Cuando se popularizó AJAX era habitual devolver los datos como XML, de ahí el nombre. Actualmente, en el desarrollo web está muy extendido JSON porque, en el lado del cliente, se pueden transformar en una variable de JavaScript con la función `JSON.parse()`.

El ejemplo **datos_categorias_json.php** devuelve los datos de un *array* codificados como JSON.

```
<?php
$cat1 = array("cod" => 1, "nombre" => "Comida");
$cat2 = array("cod" => 2, "nombre" => "Bebida");
$array = array($cat1, $cat2);
$json = json_encode($array);
echo $json;
```

Si se accede al fichero directamente desde el navegador, se obtiene:

```
[{"cod":1,"nombre":"Comida"},{"cod":2,"nombre":"Bebida"}]
```

Actividad propuesta 5.2

Modifica el ejemplo **datos_categorias_json.php** para que cargue las categorías de la base de datos de la aplicación de pedidos (capítulo 4).

Escaneado con CamScanner

5.5. Modificación de la estructura de la página web

En muchos casos, la respuesta del servidor consiste en una serie de datos que hay que mostrar de una forma u otra en la página. Puede ser que simplemente haya que actualizar una sección o que haya que modificar la estructura de la página, introduciendo y eliminando elementos.

En el ejemplo **hora.php** se muestra un caso básico. Se trata de una página que simplemente muestra la hora del servidor. En lugar de usar una alerta, como en el caso anterior, modifica el contenido de la página utilizando el DOM. Cada cinco segundos solicita el fichero **hora_servidor.php**.

- La hora del servidor se muestra en la sección con `id="hora"`, línea 24. Inicialmente está vacía.
- En las líneas 7-18 se declara la función `loadDoc()`. Se encarga de solicitar **hora_servidor.php** al servidor y de poner la respuesta en la página (líneas 11-12).
- Para seleccionar la sección utiliza `document.getElementById("hora")`. El contenido se modifica con la propiedad `innerHTML`.
- La petición se realiza en la línea 15. Los parámetros son el método HTTP mediante el que se realizará la petición, la ruta solicitada y si la petición es asíncrona o no. Con `true` se especifica que la petición es asíncrona.
- En la línea 19, la función `setInterval()` hace que se llame a `loadDoc()` cada 5 segundos.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Hora en el servidor</title>
5          <meta charset = "UTF-8">
6          <script>
7          function loadDoc() {
8              var xhttp = new XMLHttpRequest();
9              xhttp.onreadystatechange = function() {
10                 if (this.readyState == 4 && this.status == 200) {
11                     document.getElementById("hora").innerHTML =
12                     "Hora en el servidor:" + this.responseText;
13                 }
14             };
15             xhttp.open("GET", "hora_servidor.php", true);
16             xhttp.send();
17             return false;
18         }
19         setInterval(loadDoc, 5000);
20         </script>
21     </head>
22     <body>
23         <h1>Hora en el servidor</h1>
24         <section id="hora"></section>
25     </body>
26 </html>
```


Escaneado con CamScanner

Si se accede a **hora.php** desde el navegador, al principio solo se verá el título. Cinco segundos después aparecerá la hora.

El siguiente ejemplo muestra cómo procesar datos en JSON. Solicita al servidor **datos_categorias_json.php** y, con los datos que recibe, crea una lista de categorías.

```
1  function categorias(){
2      var xhttp = new XMLHttpRequest();
3      xhttp.onreadystatechange = function() {
4          if (this.readyState == 4 && this.status == 200) {
5              // crear lista
6              var lista = document.createElement("ul");
7              // meter los datos de la respuesta en un array
8              var cats = JSON.parse(this.response);
9              // para cada elemento del array
10             for(var i = 0; i < cats.length; i++){
11                 //se crea un elemento ul con el campo nombre
12                 var elem = document.createElement("li");
13                 elem.innerHTML = cats[i]["nombre"];
14                 // se añade a la lista
15                 lista.appendChild(elem);
16             }
17             var body = document.getElementById("principal");
18             // eliminar el contenido actual
19             body.innerHTML = "";
20             body.appendChild(lista);
21         }
22     };
23     xhttp.open("GET", "datos_categorias_json.php", true);
24     xhttp.send();
25     // para que no se siga el link que llama a esta función
26     return false;
27 }
```

La función solicita el fichero y cuando se recibe la respuesta:

- Línea 6: se crea un elemento `ul`.
- Línea 8: se transforma la respuesta en un *array*.
- Líneas 10-16: se recorre la respuesta creando un elemento `li` por cada elemento del *array*. El texto será el campo nombre. Estos elementos se añaden a la lista.
- Línea 19: se elimina el contenido que pueda haber en la sección principal.
- Línea 20: inserta la lista dentro del elemento principal.

La página es muy sencilla:

- Incluye el fichero en el que está la función con la etiqueta `script`.
- Define la sección en la que irá la lista.
- Llama a la función.

Escaneado con CamScanner

```
<!DOCTYPE html>
<html>
    <head>
        <title>AJAX</title>
        <meta charset = "UTF-8">
        <script type = "text/JavaScript" src = "funciones.js"></script>
    </head>
    <body>
        <section id = "principal"></section>
        <script type = "text/JavaScript" >categorias();</script>
    </body>
</html>
```

Actividad propuesta 5.3



Modifica el ejemplo anterior para que los elementos de la lista sean vínculos. El texto de los vínculos tiene que ser el nombre de cada categoría. Tienen que apuntar a *productos.php?categoria?<código>*, donde código es el código de la categoría.

5.6. Captura de eventos

Muchas veces se llama a una función JavaScript cuando el usuario sigue un vínculo o envía un formulario. Como ya se ha comentado, en estos casos el navegador por defecto carga la respuesta como una nueva página.

Para evitarlo, lo más sencillo es utilizar el atributo `onclick` (para los vínculos). Con este atributo se asocia el evento *click* con una función. Si esta función devuelve FALSE, no se sigue el vínculo. En los formularios se utiliza el atributo `onsubmit`.

El siguiente ejemplo es similar al anterior. En lugar de mostrar la lista de categorías directamente, muestra un vínculo que llama a la función.

```
<!DOCTYPE html>
<html>
    <head>
        <title>AJAX</title>
        <meta charset = "UTF-8">
        <script type = "text/JavaScript" src = "funciones.js"></script>
    </head>
        <body>
        <section id = "principal"></section>
        <a href = "#" onclick = "return categorias();">Categorías</a>
    </body>
</html>
```

Actividad propuesta 5.4



Modifica el ejemplo anterior para añadir un vínculo que cargue en la sección principal la hora en el servidor (utilizando **hora_servidor.php**).

Escaneado con CamScanner

5.7. Aplicaciones de una sola página

Una aplicación web, como la aplicación de pedidos, implica una serie de intercambios entre cliente y servidor. En las aplicaciones de una sola página todas estas comunicaciones, salvo la inicial, se hacen utilizando JavaScript.

En estas aplicaciones la página nunca se recarga por completo. Se va actualizando desde JavaScript como se ha visto a lo largo del capítulo. Desde el punto de vista del usuario, como no hay recargas, estas páginas son más parecidas a aplicaciones de escritorio.

Este enfoque se ha popularizado en los últimos años y se utiliza en varios *frameworks* muy extendidos, como AngularJs o React.



Actividad propuesta 5.5

Investiga qué librerías de JavaScript pueden ser útiles para AJAX y para manipular el DOM. Una de ellas es JQuery: https://jquery.com/.

Resumen

- Repartir la lógica entre cliente y servidor permite obtener código más reutilizable.
- La lógica de presentación se desplaza al cliente, donde se utiliza JavaScript.
- El servidor devuelve los datos en JSON, XML o cualquier otro formato.
- Las peticiones al servidor se realizan utilizando el objeto XMLHttpRequest.
- Para peticiones POST hay que añadir una cabecera y pasar una cadena de parámetros.
- Las peticiones pueden ser síncronas o asíncronas.
- En las peticiones asíncronas hay una función encargada de procesar la respuesta.
- El código en el cliente se encarga de mostrar los datos recibidos. También modifica la estructura de la página.
- JavaScript realiza cambios en la página utilizando el DOM.
- En las aplicaciones de una sola página, todas las solicitudes al servidor, salvo la primera, se realizan desde JavaScript.

Ejercicios propuestos



1. Utilizando AJAX y PHP escribe una página que muestre un número aleatorio. Cada cinco segundos, la página tiene que solicitar al servidor un nuevo número y mostrarlo.
2. Escribe una página web con un formulario que permita sumar dos números. Utiliza AJAX para enviar el formulario y mostrar el resultado.
3. Escribe un fichero PHP que devuelva en JSON los datos de la tabla de productos de la aplicación de pedidos.

Escaneado con CamScanner

4. A partir del ejercicio anterior, escribe una página que cree una tabla con los productos.
5. Añade un vínculo al ejercicio anterior para recargar la página.
6. Escribe un formulario de *login* para la tabla de Restaurantes del capítulo anterior. Muestra el resultado en una alerta.

ACTIVIDADES DE AUTOEVALUACIÓN

1. El DOM es:
   - ☐ *a)* Un lenguaje de programación en el lado del cliente.
   - ☐ *b)* Un lenguaje de programación en el lado del servidor.
   - ☐ *c)* Una interfaz para manipular documentos desde un lenguaje de programación.

2. El código JavaScript:
   - ☐ *a)* Se ejecuta en el servidor.
   - ☐ *b)* Se ejecuta en el cliente.
   - ☐ *c)* Se ejecuta en ambos.

3. La diferencia entre una petición síncrona y una asíncrona es:
   - ☐ *a)* Que las asíncronas bloquean la ejecución del *script* hasta que se recibe la respuesta y las otras no.
   - ☐ *b)* Que las síncronas bloquean la ejecución del *script* hasta que se recibe la respuesta y las otras no.
   - ☐ *c)* Que las síncronas generan una excepción si no se recibe respuesta y las otras no.

4. Las peticiones AJAX utilizan el objeto:
   - ☐ *a)* XMLHTTPRequest
   - ☐ *b)* HTTPRequest
   - ☐ *c)* DOM

5. El formato de las respuestas a las peticiones AJAX es:
   - ☐ *a)* XML
   - ☐ *b)* JSON
   - ☐ *c)* HTML, XML, JSON o cualquier otro formato.

6. Cuando la respuesta implica modificar la estructura de la página, en general:
   - ☐ *a)* El nuevo HTML se genera en el servidor y se envía al cliente
   - ☐ *b)* El nuevo HTML se genera en el cliente a partir de la respuesta del servidor.
   - ☐ *c)* El nuevo HTML tiene que estar presente ya en la página, pero oculto.

7. Una de las siguientes afirmaciones sobre las aplicaciones web de una sola página es incorrecta:
   - ☐ *a)* Realizan todas las peticiones al servidor, salvo la inicial, mediante JavaScript.

Escaneado con CamScanner

☐ *b)* Tienen un aspecto más parecido a las aplicaciones de escritorio.
☐ *c)* Se utilizan para aplicaciones pequeñas.

8. ¿Cuál de las siguientes afirmaciones sobre las aplicaciones con AJAX es incorrecta?
   ☐ *a)* Se separa la lógica de negocio de la de presentación.
   ☐ *b)* La lógica de presentación se desplaza al cliente.
   ☐ *c)* El cliente se conecta directamente a la base de datos usando JavaScript.

9. En la función que procesa la respuesta a una petición asíncrona, en general:
   ☐ *a)* Se comprueba el código de respuesta del servidor.
   ☐ *b)* Se comprueba el estado de la respuesta del servidor.
   ☐ *c)* Se comprueban ambos.

10. Si se quiere asociar una función JavaScript a un vínculo:
    ☐ *a)* La función tiene que devolver TRUE para que el navegador no cargue el vínculo.
    ☐ *b)* La función tiene que devolver FALSE para que el navegador no cargue el vínculo.
    ☐ *c)* Al utilizar el atributo onclick el comportamiento por defecto del navegador se anula, da igual lo que devuelva la función.

SOLUCIONES:

1.	a b **c**	**5.**	a b **c**
2.	a **b** c	**6.**	a **b** c
3.	a **b** c	**7.**	a b **c**
4.	**a** b c	**8.**	a b **c**
9.	a b **c**		
10.	a **b** c		

Escaneado con CamScanner

6

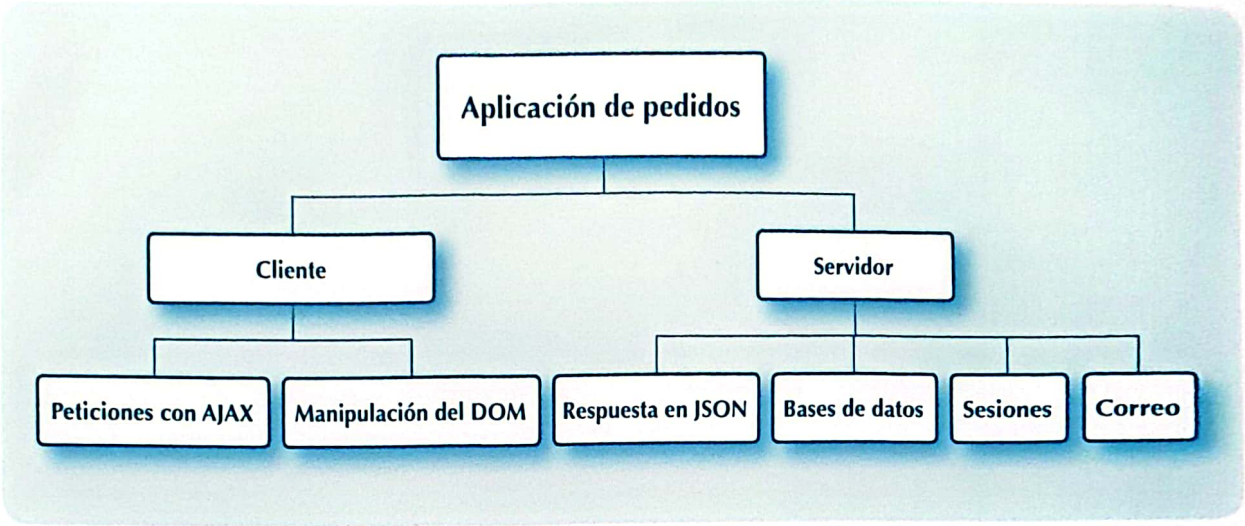
Aplicación de pedidos con AJAX

Objetivos

- ✓ Rediseñar la aplicación de pedidos como una aplicación de una sola página.
- ✓ Utilizar AJAX para las comunicaciones con el servidor y la interfaz gráfica.
- ✓ Identificar los cambios necesarios en el lado del servidor.
- ✓ Valorar las mejoras de este enfoque.
- ✓ Analizar las posibilidades de reutilización por separado del código de cliente y servidor.
- ✓ Reutilizar en lo posible el proyecto anterior.

Escaneado con CamScanner

Mapa conceptual



Glosario

Evento. Los eventos se usan, entre otras cosas, para representar las acciones del usuario. Por ejemplo, que pase el ratón por encima de un elemento, o que haga *click*.

iterator_to_array(). Función de PHP para convertir a un *array* cualquier objeto que implemente la interfaz Iterator.

json_encode(). Función de PHP para convertir un *array* en un documento JSON.

JSON.parse(). Función de JavaScript para convertir un documento JSON a un objeto.

onclick. Atributo HTML que permite asociar código al evento *click* (sobre el elemento que tiene el atributo) a una función JavaScript

onsubmit. Atributo para formularios HTML que permite asociar código JavaScript al evento *submit*.

6.1. Diseño de la aplicación

En este capítulo se rediseña la aplicación de pedidos del capítulo 4 como una aplicación de una sola página utilizando AJAX. La nueva aplicación tendrá el mismo aspecto y funcionalidad que la anterior, pero la lógica de presentación se pasa al cliente. Este se encargará de:

- Solicitar los datos al servidor y mostrarlos.
- Modificar la estructura de la página para pasar de una pantalla a otra (por ejemplo, de la lista de categorías a la tabla de productos).

La lógica en el lado del servidor se ocupará de las cosas que no se pueden hacer en el cliente:

Escaneado con CamScanner

- Control de sesiones.
- Manejo de la base de datos.
- Envío de correo.

RECUERDA

✓ Si no recuerdas bien la aplicación, revisa el apartado 4.3.

Gran parte del diseño de la aplicación anterior se mantiene sin cambios:

1. La base de datos. Utiliza la misma que las otras aplicaciones
2. El mapa de pantallas.
3. La variable para el carrito.

Pero una aplicación de una sola página implica cambios importantes:

a) Hay que modificar los ficheros que devuelven HTML para que devuelvan en JSON (o XML) solo los datos que se quieren mostrar. Esto afecta a la lista de categorías, a la tabla de productos y al carrito de la compra.
b) El cliente se ocupa de mostrar los datos recibidos o realizar cambios en la estructura de la página web JavaScript.
c) Todas las peticiones al servidor (salvo la inicial) se realizan mediante JavaScript.

Actividad propuesta 6.1

Al utilizar AJAX, manejar la interfaz gráfica es más sencillo. ¿Qué cambios harías en la aplicación original para que fuera más cómoda para el usuario? Modifica el mapa de pantallas si es necesario.

6.2. Estructura de la página web

Aunque el mapa de pantallas no cambia, ahora en realidad solo va a haber un documento HTML que se va modificando. Esto incluye el formulario de *login*. Hay que pensar la estructura de este documento.

La página tendrá dos secciones principales, una con el formulario de *login* y otra, la principal, para el resto de la aplicación. La sección principal comienza oculta. Al hacer *login* con éxito se oculta la sección del formulario y se muestra la principal.

A su vez, la sección principal tendrá dos secciones, como la aplicación original, una cabecera y otra sección en la que se mostrarán la lista de categorías, la tabla de productos, el carrito y el mensaje de confirmación. Las transiciones entre pantallas consisten en eliminar el contenido de esta sección y cargar el nuevo.

Escaneado con CamScanner

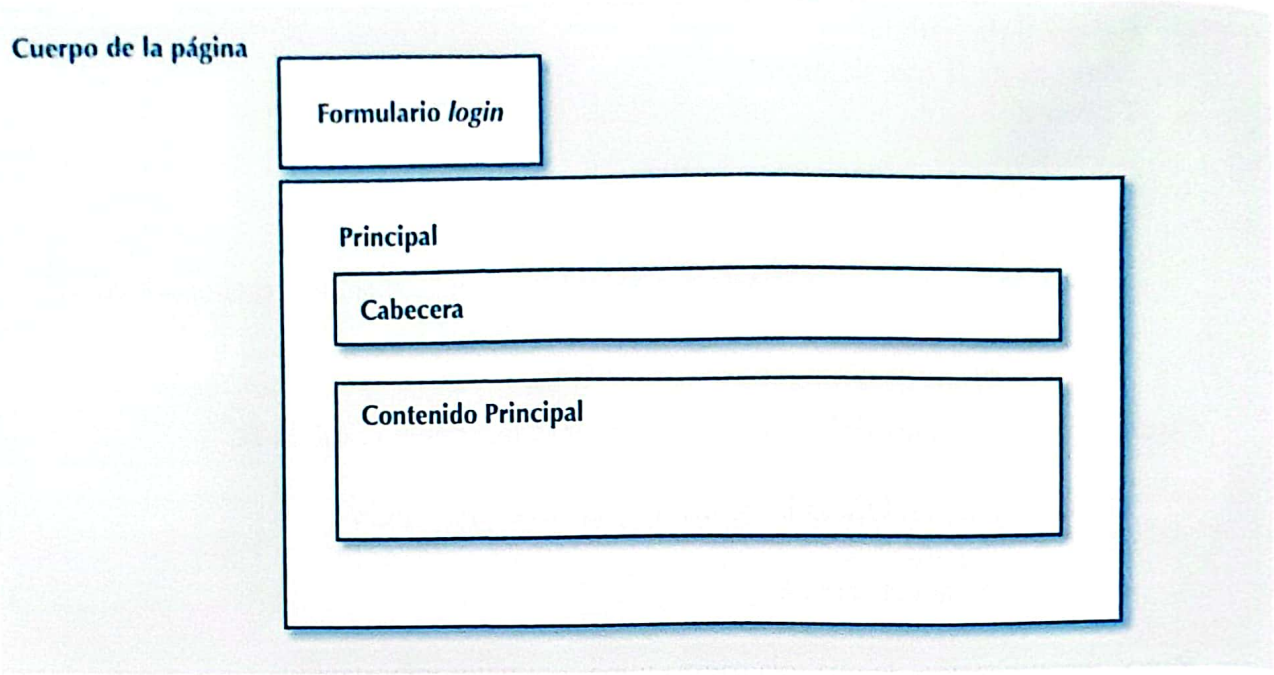


Figura 6.1
Estructura base de la página.

6.3. Cambios en la estructura

En el lado del cliente la principal dificultad está en las transiciones entre pantallas:

- Al hacer *login*, se oculta el formulario y se muestra la sección principal. El contenido inicial de la sección es la lista de categorías.
- Al cerrar sesión, se oculta la sección principal y se muestra de nuevo la del formulario.
- Al seguir un vínculo en la lista de categorías, se muestra la tabla de productos.
- Al añadir o eliminar un producto, se muestra el carrito.
- Al procesar un pedido, se muestra el mensaje de confirmación.

En todos los casos hay que establecer comunicación con el servidor y modificar el contenido de la página con la respuesta. Habrá una función para cada caso (cuadro 6.1).

CUADRO 6.1
Funciones más importantes en el cliente

Función JavaScript	Descripción
login()	Valida el formulario, muestra la sección principal
cerrarSesionUnaPagina()	Cierra la sesión, muestra el formulario de *login*
cargarCategorias()	Solicita los datos de las categorías, crea la lista de vínculos
cargarProductos()	Solicita los datos de los productos, crea la tabla
cargarCarrito()	Solicita los datos del carrito, crea la tabla

[.../...]

Escaneado con CamScanner

CUADRO 6.1 (CONT.)

anadirProductos()	Modifica el carrito y lo muestra
eliminarProductos()	Modifica el carrito y lo muestra
procesarPedido()	Envía la orden de procesar el pedido y muestra el resultado

Para organizar mejor el código, se utilizan algunas funciones auxiliares para la creación de la interfaz (cuadro 6.2).

CUADRO 6.2
Funciones auxiliares para la interfaz gráfica

Función	Descripción
crearVinculoCategorias()	Crea los vínculos para lista de categorías
crearTablaCarrito()	Crea la tabla del carrito de la compra
crearTablaProductos()	Crea la tabla de productos
crearFila()	Crea una fila de las tablas de productos y carrito
crearFormulario()	Crea los formularios de añadir y eliminar

6.4. En el servidor

La parte del servidor es bastante parecida a la del capítulo 4. Las diferencias son:

- Los ficheros que devolvían HTML ahora devuelven JSON.
- Se eliminan las redirecciones, que no tienen sentido en una aplicación de una sola página. En su lugar, la respuesta será un código indicando si la operación se realizó con éxito o no.

CUADRO 6.3
Ficheros en el servidor

Ruta	Descripción	Parámetros
login_json.php	Valida el formulario de entrada	$_POST['usuario'] $_POST['clave']
logout_json.php	Cierra la sesión	
categorías_json.php	Devuelve un documento JSON con los datos de las categorías	
productos_json.php	Devuelve un documento JSON con los datos de los productos de una categoría	$_GET['categoria'], el código de la categoría

[.../...]

Escaneado con CamScanner

CUADRO 6.3 (CONT.)

carrito_json.php	Devuelve un documento JSON con los datos del carrito de la compra	
anadir_json.php	Añade productos al carrito	$_POST['cod'] $_POST['unidades']
eliminar_json.php	Elimina productos del carrito	$_POST['cod'] $_POST['unidades']
procesar_pedido_json.php	Inserta el pedido en la base de datos, envía correos de confirmación y devuelve error o éxito	
una_sola_pagina.html	Página principal y única de la aplicación	
cabecera_json.php	Cabecera con vínculos para ver el carrito, las categorías o cerrar sesión	
sesiones_json.php	Para comprobar que el usuario haya iniciado sesión correctamente	
bd.php	Para agrupar las funciones de la base de datos	
correo.php	Funciones para enviar correo	

Los siete primeros ficheros están pensados para ser accedidos desde JavaScript. Para cada uno de ellos habrá una función de JavaScript que se encargue de llamarlos y procesar la respuesta.

Actividad propuesta 6.2

¿Crees que JSON es el mejor formato para devolver los datos de categorías y productos? ¿Sería mejor devolver directamente la lista o la tabla HTML? ¿Y devolver los datos en XML? Analiza las ventajas e inconvenientes de cada opción.

6.5. Implementación

El fichero inicial de la aplicación es **una_sola_pagina.php**.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Formulario de login</title>
5          <meta charset = "UTF-8">
6          <script type = "text/JavaScript" src = "js/cargarDatos.js"></
           script>
7          <script type = "text/JavaScript" src = "js/sesion.js"></script>
8      </head>
9      <body>
```

Escaneado con CamScanner

```
10          <section id = "login">
11              <form onsubmit="return login()" method = "POST">
12                  Usuario <input id = "usuario" type = "text">
13                  Clave <input id = "clave" type = "password">
14                  <input type = "submit">
15              </form>
16          </section>
17          <section id = "principal" style="display:none">
18              <header>
19                  <?php require 'cabecera_json.php' ?>
20              </header>
21              <h2 id = "titulo"></h2>
22              <section id = "contenido">
23              </section>
24          </section>
25      </body>
26  </html>
```

Se puede observar:

- Líneas 6-7: incluye los ficheros JavaScript.
- Línea 11: el envío del formulario está asociado a la función JavaScript `login()`.
- Líneas 12-13: los campos del formulario tienen `id` en lugar de `name`, para usarlos desde JavaScript.
- Línea 17: la sección con id principal comienza oculta.



Actividad propuesta 6.3

En lugar de utilizar una página con una estructura base, se podría crear toda la página desde JavaScript. ¿Qué ventajas e inconvenientes tiene esta opción?

6.6. Lado del servidor

En esta versión de la aplicación los ficheros del servidor son más sencillos porque no hay que ocuparse de la salida.

6.6.1. *Login*

Del *login* se encarga **login_json.php**. Es parecido al fichero de *login* original, pero solo tiene la parte de procesamiento. El formulario está en página principal (y única). Además, no redirige. Si hay error, devuelve la cadena FALSE; y si no crea las variables de sesión, devuelve la cadena TRUE.

Escaneado con CamScanner

```php
<?php
require_once '..\cap4\bd.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $usu = comprobar_usuario($_POST['usuario'], $_POST['clave']);
    if($usu===FALSE){
        echo "FALSE";
    }else{
        session_start();
        // $usu tiene campos correo y codRes, correo
        $_SESSION['usuario'] = $usu;
        $_SESSION['carrito'] = [];
        echo "TRUE";
    }
}
```

6.6.2. Control de acceso

Hay cambios en el control de acceso, que antes se basaba en una redirección. Ahora, la función `comprobar_sesion()` devuelve TRUE o FALSE según haya sesión abierta o no.

```php
<?php
    function comprobar_sesion(){
        session_start();
        if(!isset($_SESSION['usuario'])){
        return FALSE;
        }else return TRUE;
    }
```

Los demás ficheros hacen esta comprobación:

```php
require_once '..\cap4\bd.php';
if(!comprobar_sesion()) return;
```

6.6.3. La cabecera

La cabecera, aunque corta, cambia completamente. Los vínculos se sustituyen por llamadas a las funciones correspondientes. El nombre del usuario no aparece, en su lugar hay un elemento **span** con id = "nombre". Esto es así porque la cabecera se sirve antes de que se haya hecho *login*, así que no puede saberse el usuario. Si se hace *login* correctamente, se introduce el nombre desde JavaScript.

```html
<header>
    <span id="cab_usuario"></span>
    <a href="#" onclick="loadCategorias();">Home</a>
    <a href="#" onclick="cargarCarrito();">Carrito</a>
    <a href="#" onclick="cerrarSesionUnaPagina();">Cerrar sesión</a>
</header>
<hr>
```


Escaneado con CamScanner

6.6.4. Las categorías

El fichero **categorias_json.php** devuelve los datos de la tabla de categorías en formato JSON. Utiliza la función `cargar_categorias()`, la misma del capítulo 4. La función `json_enconde()` transforma un *array* a formato JSON. Como `cargar_categorias()` devuelve un cursor, se utiliza `iterator_to_array()` para convertirlo.

```
<?php
require_once 'sesiones_json.php';
require_once '..\cap4\bd.php';
if(!comprobar_sesion()) return;
$categorias = cargar_categorias();
$cat_json = json_encode(iterator_to_array($categorias);
echo $cat_json;
```

Si se accede directamente desde el navegador, se obtendrá:

[{"codCat":"3","0":"3","nombre":"Bebidas con","1":"Bebidas con"},
{"codCat":"2","0":"2","nombre":"Bedidas sin","1":"Bedidas sin"},
{"codCat":"1","0":"1","nombre":"Comida","1":"Comida"}]

6.6.5. Los productos

El fichero **productos_json.php** devuelve los datos de los productos de una categoría en formato JSON. El código de la categoría se pasa en la URL. Funciona de manera análoga al anterior.

```
<?php
require_once '..\cap4\bd.php';
/*comprueba que el usuario haya abierto sesión o devuelve*/
require 'sesiones_json.php';
if(!comprobar_sesion()) return;
$productos_array = [];
$productos = cargar_productos_categoria($_GET['categoria']);
$cat_json = json_encode(iterator_to_array($productos);
echo $cat_json;
```

6.6.6. El carrito de la compra

Para obtener los productos del carrito se usa **carrito_json.php**. Devuelve un *array* JSON con los datos de los productos presentes en el carrito y las unidades pedidas. Es muy parecido a **carrito.php** del capítulo 4.

```
<?php
require 'sesiones_json.php';
require_once '..\cap4\bd.php';
if(!comprobar_sesion()) return;
$productos = cargar_productos(array_keys($_SESSION['carrito']));
// hay que añadir las unidades
```


Escaneado con CamScanner

```
$productos = iterator_to_array($productos);
foreach($productos as &$producto){
    $cod = $producto['CodProd'];
    $producto['unidades'] = $_SESSION['carrito'][$cod];
}
echo json_encode($productos);
```

6.6.7. Añadir y eliminar productos

Los ficheros **anadir_json.php** y **eliminar_json.php**, son prácticamente iguales a los de la aplicación anterior. El único cambio es que no redirigen.

```
<?php
/*comprueba que el usuario haya abierto sesión o devuelve*/
require 'sesiones_json.php';
if(!comprobar_sesion()) return;
$cod = $_POST['cod'];
$unidades = (int)$_POST['unidades'];
/*si existe el código sumamos las unidades*/
if(isset($_SESSION['carrito'][$cod])){
    $_SESSION['carrito'][$cod] += $unidades;
}else{
    $_SESSION['carrito'][$cod] = $unidades;
}
<?php
/*comprueba que el usuario haya abierto sesión*/
require_once 'sesiones_json.php';
if(!comprobar_sesion()) return;
$cod = $_POST['cod'];
$unidades = $_POST['unidades'];
/*si existe el código restamos las unidades, con mínimo de 0*/
if(isset($_SESSION['carrito'][$cod])){
    $_SESSION['carrito'][$cod] -= $unidades;
    if($_SESSION['carrito'][$cod] <= 0){
        unset($_SESSION['carrito'][$cod]);
    }
}
```

6.6.8. Cerrar la sesión

Lo mismo ocurre con **logut_json.php**.

```
<?php
if(!comprobar_sesion()) return;
$_SESSION = array();
session_destroy(); // eliminar la sesion
setcookie(session_name(), 123, time() - 1000); // eliminar la cookie
```


Escaneado con CamScanner

6.6.9. Procesar el pedido

El fichero **procesar_pedido.php** también es muy parecido al de la primera versión. En lugar de devolver un mensaje, devuelve las cadenas TRUE o FALSE según se haya podido insertar el pedido o no. Utiliza las funciones de correo del capítulo 4.

```
<?php
/*comprueba que el usuario haya abierto sesión o redirige*/
require '..\cap4\correo.php';
require_once '..\cap4\bd.php';
/*comprueba que el usuario haya abierto sesión o devuelve*/
require 'sesiones_json.php';
if(!comprobar_sesion()) return;
$resul = insertar_pedido($_SESSION['carrito'], $_SESSION['usuario']['co-
dRes']);
if($resul === FALSE){
    echo "FALSE";
}else{
    $correo = $_SESSION['usuario']['correo'];
    $conf = enviar_correos($_SESSION['carrito'], $resul, $correo);
    echo "TRUE";
    //vaciar carrito
    $_SESSION['carrito'] = [];
}
```

6.6.10. Funciones auxiliares

Los ficheros **bd.php** y **correo.php** son los del capítulo 4. Se reutilizan sin cambios.

6.7. El lado del cliente

Las funciones de JavaScript se reparten en dos ficheros:

- **sesion.js.** Con las funciones para hacer *login* y cerrar sesión.
- **cargarDatos.js.** Con las funciones que piden los datos al servidor y modifican la estructura de la página.

6.7.1. *Login*

El formulario de *login* está asociado con la función `login()`, que envía a `login_json.php` los datos del formulario. Si son correctos:

- Muestra la sección principal (línea 8).
- Oculta la sección del formulario (línea 9).
- Introduce el nombre del usuario en la parte apropiada de la cabecera (línea 11).

Escaneado con CamScanner

Si no son correctos, muestra un mensaje de error con una alerta (línea 6).

```
1  function login(formu){
2      var xhttp = new XMLHttpRequest();
3      xhttp.onreadystatechange = function() {
4          if (this.readyState == 4 && this.status == 200) {
5              if(this.responseText==="FALSE"){
6                  alert("Revise usuario y contraseña");
7              }else{
8                  document.getElementById("principal").style.display= "block";
9                  document.getElementById("login").style.display= "none";
10                 /*ponemos el usuario devuelto en el hueco correspondiente*/
11                 document.getElementById("cab_usuario").innerHTML = "Usua-
                   rio: " + usuario;
12                 cargarCategorias();
13             }
14         }
15     }
16     var usuario = document.getElementById("usuario").value;
17     var clave = document.getElementById("clave").value;
18     var params = "usuario="+usuario+"&clave="+clave;
19     xhttp.open("POST", "login_json.php", true);
20     // envío con POST requiere cabecera y cadena de parámetros
21     xhttp.setRequestHeader("Content-type","application/
           x-www-form-urlencoded");
22     xhttp.send(params);
23     return false;
24 }
```

La petición se realiza en la línea 22. Como se trata de enviar un formulario se utiliza el método POST. Por tanto, hay que añadir la cabecera específica (línea 21) y enviar una cadena con los parámetros.

6.7.2. Las categorías

La función `cargarCategorias()` solicita al servidor los datos de las categorías (**categorias_json.php**) y crea la lista de vínculos. Cuando recibe la respuesta del servidor:

- La convierte en un objeto JavaScript, que será un *array* (línea 5). Cada elemento de *array* será un objeto con campos `codCat` y `nombre` (los nombres de los campos en el *array* JSON que devuelve el servidor).
- Crea un elemento `ul`, la lista que tiene que crear (línea 6).
- Por cada elemento del *array* crea un vínculo usando el nombre y el código (línea 9).
- Ese elemento se introduce en un elemento `li` (línea 10).
- El elemento `li` se introduce en la lista (línea 11).
- Elimina el contenido de la sección "contenido" y luego introduce la lista en ella (líne 13-17).

Escaneado con CamScanner

```
function cargarCategorias() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        var cats = JSON.parse(this.responseText);
        var lista = document.createElement("ul");
        for(var i = 0; i < cats.length; i++){
            var elem = document.createElement("li");
            vinculo = crearVinculoCategorias(cats[i].nombre, cats[i].codCat);
            elem.appendChild(vinculo);
            lista.appendChild(elem);
        }
        var contenido = document.getElementById("contenido");
        contenido.innerHTML = "";
        var titulo = document.getElementById("titulo");
        titulo.innerHTML ="Categorías";
        contenido.appendChild(lista);
    }
};
xhttp.open("GET", "categorias_json.php", true);
xhttp.send();
return false;
```

Para crear los vínculos utiliza una función auxiliar.

```
function crearVinculoCategorias(nom, cod){
    var vinculo = document.createElement("a");
    var ruta = "productos_json.php?categoria=" +cod;;
    vinculo.href = ruta;
    vinculo.innerHTML = nom;
    vinculo.onclick = function(){return cargarProductos(this);}
    return vinculo;
}
```

Mediante el atributo `onclick`, estos vínculos están asociados a la función `cargarProductos()`. En `this` está la ruta del vínculo.



Actividad propuesta 6.4

Investiga la función on() de jQuery(), que permite asociar eventos a funciones.

6.7.3. Los productos

La función `cargarProductos()` es muy parecida a `cargarCategorias()`.

- Pide los datos al servidor (**productos_json.php**).
- Crea la tabla de productos.

Escaneado con CamScanner

- Elimina el contenido de la sección "contenido" y luego introduce la tabla en ella.

```
function cargarProductos(destino){
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var prod = document.getElementById("contenido");
            var titulo = document.getElementById("titulo");
            titulo.innerHTML ="Productos";
            try{
                var filas = JSON.parse(this.responseText);
                var tabla = crearTablaProductos(filas);
                prod.innerHTML = "";
                prod.appendChild(tabla);
            }catch(e){
                var mensaje = document.createElement("p");
                mensaje.innerHTML = "Categoría sin productos";
                prod.innerHTML = "";
                prod.appendChild(mensaje);
            }
        }
    };
    xhttp.open("GET", destino, true);
    xhttp.send();
    return false;
}
```

Para crear la tabla utiliza la función auxiliar `crearTablaProductos(filas)`. Esta función recibe el *array* de productos y devuelve un elemento `<table>` con una fila por producto.

```
function crearTablaProductos(productos){
    var tabla = document.createElement("table");
    var cabecera = crear_fila(["Código", "Nombre", "Descripción",
        "Stock", "Comprar"], "th");
    tabla.appendChild(cabecera);
    for(var i = 0; i < productos.length; i++){
        /*formulario*/
        formu = crearFormulario( "Añadir",
            productos[i]['CodProd'],anadirProductos);
        fila = crear_fila([productos[i]['CodProd'],
            productos[i]['Nombre'],
            productos[i]['Descripcion'],
            productos[i]['Stock']], "td");
        celda_form = document.createElement("td");
        celda_form.appendChild(formu);
        fila.appendChild(celda_form);
        tabla.appendChild(fila);
    }
```


Escaneado con CamScanner

```
    return tabla;
}
```

Para crear el formulario de cada fila se usa la función `crearFormulario(texto, cod, funcion)`. Esta función recibe tres parámetros:

1. El texto del botón del formulario.
2. El código del producto.
3. La función que se encarga de enviar el formulario. En la tabla de productos el formulario se envía con la función `anadirProductos()`. El formulario queda asociado con la función a través del atributo `onsubmit`. Es decir, cuando se pulse el botón de envío del formulario, se llamará a la función.

```
1  function crearFormulario(texto, cod, funcion){
2      var formu = document.createElement("form");
3      var unidades = document.createElement("input");
4      unidades.value = 1;
5      unidades.name = "unidades";
6      var codigo = document.createElement("input");
7      codigo.value = cod;
8      codigo.type = "hidden";
9      codigo.name = "cod";
10     var bsubmit = document.createElement("input");
11     bsubmit.type = "submit";
12     bsubmit.value = texto;
13     formu.onsubmit = function(){return funcion(this);}
14     formu.appendChild(unidades);
15     formu.appendChild(codigo);
16     formu.appendChild(bsubmit);
17     return formu;
18 }
```

6.7.4. El carrito

Esta función solicita al servidor los datos de los productos del carrito (**carrito_json.php**) y crea la tabla de productos. Elimina el contenido de la sección "contenido" y luego introduce la tabla en ella. También muestra un vínculo para realizar el pedido, que está asociado con la función `realizarPedido()`.

Para crear la tabla utiliza la función auxiliar `crearTablaCarrito(filas)`, que es muy parecida a la de crear productos.

```
function crearTablaCarrito(productos){
    var tabla = document.createElement("table");
    var cabecera = crear_fila(["Código", "Nombre", "Descripción",
        "Unidades", "Eliminar"], "th");
    tabla.appendChild(cabecera);
    for(var i = 0; i < productos.length; i++){
```

Escaneado con CamScanner

```
            /*formulario*/
            formu = crearFormulario("Eliminar",productos[i]['CodProd'],elimi-
            narProductos);
            fila = crear_fila([productos[i]['CodProd'],
                productos[i]['Nombre'],
                productos[i]['Descripcion'],
                productos[i]['unidades']], "td"
            );
            celda_form = document.createElement("td");
            celda_form.appendChild(formu);
            fila.appendChild(celda_form);
            tabla.appendChild(fila);
        }
        return tabla;
    }
```

Para crear el formulario de cada fila también se usa la función `crearFormulario(texto, cod, funcion)`, pero como función se pasa `eliminarProductos()`.

Actividad propuesta 6.5

Las funciones para crear la tabla de productos y la del carrito son muy parecidas. ¿Crees que sería buena idea unificarlas? ¿Qué cambios habría que hacer?

6.7.5. Añadir y eliminar

La función de añadir productos está asociada al formulario de la tabla de productos. Cuando se envía el formulario se ejecuta la función. El argumento de la función es el propio formulario. La función se encarga de coger los datos del formulario y enviarlos a **anadir_json.php**. Como es un envío POST hay que crear una cabecera y una cadena con los parámetros.

Al recibir la respuesta muestra una alerta y llama a `cargarCarrito()`.

La función eliminar es igual, pero llama a **eliminar_json.php**.

```
function anadirProductos(formulario){
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            alert("Producto añadido con éxito");
            cargarCarrito();
        }
    };
    var params = "cod="+formulario.elements['cod'].value+
    "&unidades="+formulario.elements['unidades'].value;
    xhttp.open("POST", "anadir_json.php", true);
```

Escaneado con CamScanner

```
xhttp.setRequestHeader("Content-type",
"application/x-www-form-urlencoded");
xhttp.send(params);
return false;
}
function eliminarProductos(formulario){
   var xhttp = new XMLHttpRequest();
   xhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
      alert("Producto eliminado con éxito");
      cargarCarrito();
   }
   };
   var params = "cod="+formulario.elements['cod'].value+
      "&unidades="+formulario.elements['unidades'].value;
   xhttp.open("POST", "eliminar_json.php", true);
   xhttp.setRequestHeader("Content-type",
      "application/x-www-form-urlencoded");
   xhttp.send(params);
   return false;
}
```

6.7.6. Realizar el pedido

Esta función solicita **procesar_pedido_json.php** al servidor. Muestra un mensaje de confirmación o error según reciba las cadenas TRUE o FALSE en la respuesta.

```
function procesarPedido(){
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
      var contenido = document.getElementById("contenido");
      contenido.innerHTML = "";
      var titulo = document.getElementById("titulo");
      titulo.innerHTML ="Estado del pedido";
      if(this.responseText=="TRUE"){
         contenido.innerHTML = "Pedido realizado";
      }else{
         contenido.innerHTML = "Error al procesar el pedido";
      }
   }};
   xhttp.open("GET", "procesar_pedido_json.php", true);
   xhttp.send();
   return false;
}
```

Escaneado con CamScanner

Resumen

- La aplicación tiene el mismo aspecto y funcionalidad que la del capítulo 4, pero está elaborada utilizando AJAX.
- Después de la petición inicial, todas las peticiones al servidor se realizan a través de JavaScript.
- La página inicial contiene el formulario de *login* y una sección principal para el resto de la aplicación. Incluye los ficheros JavaScript.
- Al hacer *login* con éxito, se oculta el formulario y se muestra la sección principal.
- La lógica de presentación se desplaza al cliente. El servidor se ocupa del control de sesiones, base de datos y envío de correo.
- El servidor devuelve los datos en JSON en lugar de utilizar HTML.
- El código en el cliente se encarga de mostrar los datos recibidos. También modifica la estructura de la página.
- Hay que asociar los eventos de envío de formularios con las funciones de JavaScript correspondientes.
- Lo mismo ocurre con los vínculos de la lista de categorías, la cabecera y realizar pedido.
- Se reutilizan sin cambios las funciones de bases de datos y correo del capítulo anterior.



Ejercicios propuestos

Modifica la aplicación para que:

1. No redirija al carrito de la compra al añadir o eliminar productos.
2. No muestre los productos sin *stock*.
3. Pida confirmación al usuario antes de realizar el pedido.
4. El carrito de la compra esté siempre visible.
5. Muestre un vínculo "Zona admin" solo a los usuarios con rol 1.
6. Para reflexionar: ventajas e inconvenientes de manejar el carrito de la compra desde JavaScript en lugar de desde el servidor.

ACTIVIDADES DE AUTOEVALUACIÓN

1. Al hacer *login* con éxito, se pasa a la pantalla principal:
   - ☐ *a)* Solicitando la página al servidor por JavaScript.
   - ☐ *b)* Haciendo visible la sección correspondiente.
   - ☐ *c)* El servidor envía una redirección.

Escaneado con CamScanner

3. El correo electrónico:
   - ☐ *a)* Se envía desde el cliente.
   - ☐ *b)* Se envía desde el servidor.
   - ☐ *c)* El correo se genera en el cliente, que lo pasa al servidor para que lo envíe.

4. Los datos de conexión a la base de datos se introducen:
   - ☐ *a)* En un fichero XML.
   - ☐ *b)* En un fichero de texto plano.
   - ☐ *c)* Directamente en las funciones que usan la base de datos.

5. Los datos del servidor de correo electrónico se introducen:
   - ☐ *a)* En un fichero XML.
   - ☐ *b)* En un fichero de texto plano.
   - ☐ *c)* Directamente en la función de correo.

6. La cabecera de la página:
   - ☐ *a)* Se inserta en el cliente.
   - ☐ *b)* Se inserta en el servidor.
   - ☐ *c)* Se inserta en el servidor, pero algunos campos se completan en el cliente.

7. La conexión con la base de datos se realiza:
   - ☐ *a)* Desde el cliente.
   - ☐ *b)* Desde el servidor.
   - ☐ *c)* Desde el cliente para el *login*, desde el servidor para los pedidos.

8. ¿Cuál de estos elementos ha cambiado respecto a la aplicación inicial?
   - ☐ *a)* Diseño de la base de datos.
   - ☐ *b)* Carrito de la compra.
   - ☐ *c)* Formato de salida de los *scripts* del servidor.

9. Al cerrar sesión:
   - ☐ *a)* El servidor redirige a la página de *login*.
   - ☐ *b)* Se oculta la sección principal y se vuelve a mostrar la de *login*.
   - ☐ *c)* Se solicita mediante JavaScript la página de despedida.

10. Las sesiones se manejan:
    - ☐ *a)* Desde el servidor.
    - ☐ *b)* Desde el cliente.
    - ☐ *c)* La variable para el carrito en el servidor, la que almacena los datos del usuario en el cliente.

SOLUCIONES:

1.	a	**b**	c	**5.**	a	b	**c**	
2.	a	**b**	c	**6.**	a	b	**c**	**9.** a **b** c
3.	a	**b**	c	**7.**	a	**b**	c	**10.** **a** b c
4.	**a**	b	c	**8.**	a	b	**c**	

Escaneado con CamScanner

7

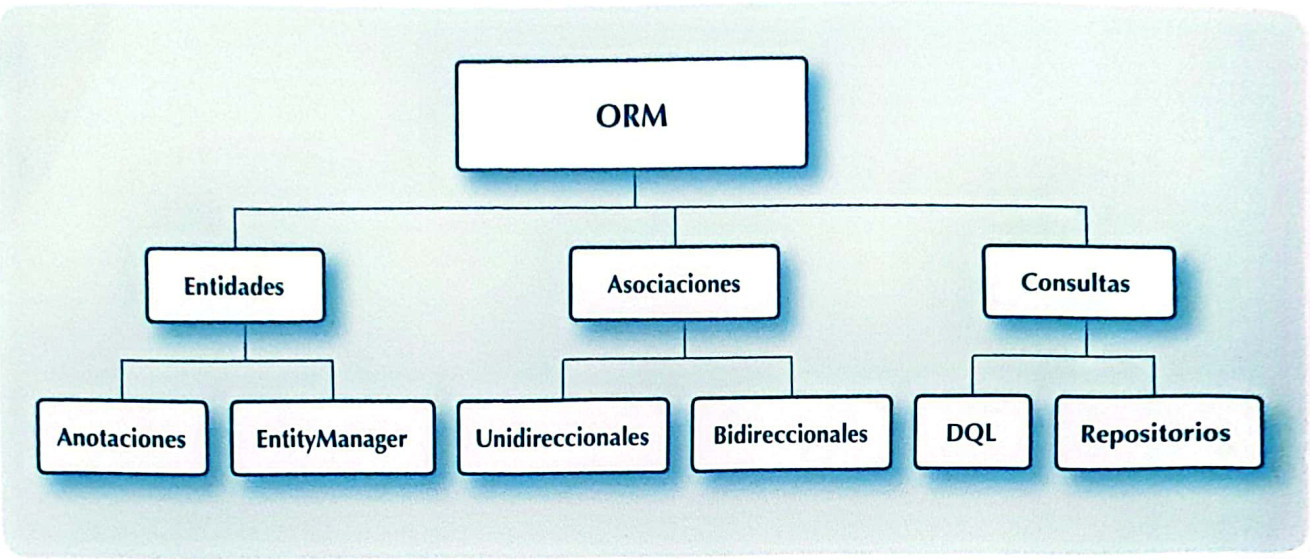
Mapeo objeto-relacional (ORM)

Objetivos

- ✓ Entender la utilidad de los ORM.
- ✓ Comprender las dificultades para almacenar objetos en el modelo relacional.
- ✓ Aprender a utilizar el ORM Doctrine.
- ✓ Crear entidades para representar tablas de la base de datos.
- ✓ Conocer las anotaciones más importantes en Doctrine.
- ✓ Diferenciar entre los distintos tipos de asociaciones.

Escaneado con CamScanner

Mapa conceptual



Glosario

Anotación. Metadatos que se añaden como comentarios al código fuente.

Asociación. Las entidades se relacionan mediante asociaciones. Son parecidas a las claves ajenas en el modelo relacional.

Desajuste por impedancia objeto-relacional. Con esta expresión se hace referencia a las dificultades que surgen al almacenar objetos en bases de datos relacionales.

Doctrine. ORM para PHP. Es el ORM por defecto de Symfony.

DQL. *Doctrine Query Language*. Lenguaje parecido al SQL para hacer búsquedas en Doctrine.

Entidad. Denominación que reciben las clases sincronizadas con la base de datos.

Hibernate. Uno de los ORM más extendidos para JEE. Doctrine está inspirado en él.

Mapeo objeto-relacional. El mapeo objeto-relacional consiste en asociar las clases que se manejan en la aplicación y la base de datos para simplificar el manejo de esta. También se denomina ORM al software que lo implementa.

7.1. Mapeo objeto-relacional

Mediante el mapeo objeto-relacional es posible asociar los elementos de la base de datos con los objetos de la aplicación de manera que la gestión de la base de datos sea más sencilla. La gestión no se realiza directamente sobre la base de datos, sino sobre una serie de clases que la replican.

Dentro del patrón MVC, que se verá en el próximo capítulo, los ORM se encargan del modelo. Es importante señalar que los ORM no se utilizan solo dentro del patrón MVC, también se emplean en otros tipos de aplicaciones.

Escaneado con CamScanner

En los capítulos anteriores hemos realizado el acceso a datos enviando las sentencias correspondientes como cadenas de texto que hay que construir cuidadosamente, un proceso en el que es fácil cometer errores. Además, las operaciones con las bases de datos se repiten en todas las aplicaciones cambiando el nombre de las tablas y las columnas.

Los ORM liberan al programador de muchas tareas repetitivas, generando el código necesario para comunicarse con la base de datos. Proporcionan un nivel de abstracción adicional que permite integrar diferentes orígenes de datos con un lenguaje común y facilita la reusabilidad del código. Por ejemplo, si se desarrolla una aplicación utilizando SQL Server como gestor de la base de datos, pero más adelante se decide migrar a Oracle, no habrá que adaptar el código que gestiona la base de datos a las diferencias entre fabricantes, ya que de esa parte se ocupa el ORM. Por supuesto, siempre que el ORM utilizado sea compatible con ambos sistemas.

Por ejemplo, para cargar el departamento 3 usando el ORM Doctrine, se usaría el siguiente código:

```
$dep = $entityManager->find("Departamento", 3);
```

Con la librería PDO se utilizaba:

```
$sql = 'SELECT * FROM departamentos where codDept = 3';
$usuarios = $bd->query($sql);
```

En este caso sencillo ya se aprecian las ventajas. Es el método `find()` el que se encarga de construir y ejecutar la cadena SQL correspondiente. La clase `Departamentos` contiene los métodos necesarios para realizar las operaciones de la base de datos. Siguiendo con el ejemplo anterior, es posible actualizar los departamentos modificando las propiedades del objeto y guardándolo:

```
$dep->setPresupuesto(70000);
$entityManager->flush();
```

Además de los métodos, los ORM incluyen un leguaje para hacer consultas sobre el modelo para realizar consultas complejas.

En general, cada tabla de la base de datos tiene una clase asociada, y el conjunto de estas clases constituye el modelo de la aplicación. Es posible crear las clases a partir de la base de datos utilizando asistentes y viceversa.

7.2. Doctrine

Doctrine es un ORM muy extendido para PHP. Está inspirado en Hibernate, uno de los ORM más extendidos, muy utilizado en JEE.

7.2.1. Instalación y configuración

Se puede instalar Doctrine utilizando `composer` mediante el siguiente comando:

```
composer require doctrine/orm
```


Escaneado con CamScanner

El siguiente paso consiste en crear el fichero **bootstrap.php**. Este fichero sirve para:

- Configurar la conexión con la base de datos. Los datos se introducen en el *array* `$dbparams`, en las líneas 8-13.
- Obtener un objeto de la clase `EntityManager`. Esta clase expone la interfaz pública de Doctrine. En la última línea se crea el objeto `$entityManager`, que estará disponible en los ficheros que incluyan a **bootstrap.php**.

```
<?php
require_once "../vendor/autoload.php";
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;
$paths = array("./src");
$isDevMode = true;
// configuración de la base de datos
$dbParams = array(
'driver' => 'pdo_mysql',
'user' => 'root',
'password' => '',
'dbname' => 'doctrine',
'host' => 'localhost',
);
$config = Setup::createAnnotationMetadataConfiguration($paths,
    $isDevMode, null, null, false);
$entityManager = EntityManager::create($dbParams, $config);
```



TOMA NOTA

En la segunda línea se incluye el fichero **autoload.php**, que carga Doctrine. Hay que ajustar la ruta según dónde esté la carpeta.

7.2.2. Entidades

El elemento básico de Doctrine son las entidades. Las entidades son clases que están asociadas con la base de datos. Los atributos (al menos algunos de ellos) de estas clases son persistentes. Es decir, se almacenan en una base de datos.

La relación entre las entidades y la base de datos se describe mediante metadatos que pueden añadirse con anotaciones en la propia entidad o con un fichero externo en formato XML. En este capítulo se utiliza la primera opción.

Las anotaciones se introducen dentro de bloques de comentarios DocBlock. Aparecen precedidas por una arroba (@) y pueden tener atributos. Por ejemplo, para asociar una clase con una tabla se utilizan las anotaciones `@Entity` y `@Table`. El bloque se sitúa justo encima de la declaración de la clase.

Escaneado con CamScanner

```
/**
* @ORM\Entity
* @ORM\Table(name="tabla")
*/
class Clase
{
```

La anotación `@Entity` simplemente dice que la clase es una entidad. La anotación `@Table` sirve para indicar con qué tabla se relaciona (con el atributo `name`). Ambas anotaciones tienen otros atributos que no se usan en el ejemplo.

En general, en las anotaciones hay que especificar:

- La tabla con la que está mapeada la entidad.
- La columna con la que está mapeado cada atributo persistente.
- Los modificadores de cada atributo, por ejemplo, que se trata un valor autogenerado.
- Las asociaciones con otras entidades. Las asociaciones son la contraparte de las claves ajenas.

Las entidades son clases normales con las siguientes restricciones:

a) Sus atributos tienen que ser `protected` o `private`.
b) En general hay que hacer *getters* y *setters* para todos los atributos salvo los que se correspondan con una columna autogenerada, por ejemplo, los campos de autoincremento en MySQL. Para estos campos no se hace *setter*.

TOMA NOTA

En este capítulo se usan las siguientes tablas de la base de datos "doctrine".

Equipo(Id, Nombre, Socios, Fundacion, Ciudad)
Jugador(Id, nombre, Apellidos, Edad, Equipo)

Los atributos id son clave primaria. La columna Equipo en Jugador es clave ajena.

La clase Equipo es una entidad sincronizada con la tabla Equipo. Tiene un atributo por cada columna de la tabla. Como se puede ver, sobre cada atributo hay un bloque con la anotación `@Column` para indicar el tipo de dato.

Para el primer atributo hay además otras dos anotaciones:

- `@Id` indica que el atributo es clave primaria.
- `@GeneratedValue` indica que es un valor autogenerado.

En este caso los nombres de los atributos de la clase coinciden con los de las columnas de la tabla. Si no fuera así, se utiliza el atributo `name` dentro de la anotación `@Column` para indicar con qué columna se asocia el atributo.

Escaneado con CamScanner

@ORM\Column(type="integer", name ="columna")

```php
<?php
// src/Equipo.php
use Doctrine\ORM\EntityRepository;
use Doctrine\ORM\Mapping as ORM;;
/**
* @ORM\Entity
* @ORM\Table(name="equipo")
@ORM\Entity(repositoryClass="EquipoRepository")
*/
class Equipo{
    /** @ORM\Id @ORM\Column(type="integer") @ORM\GeneratedValue **/
    private $id;
    /** @ORM\Column(type="string") **/
    private $nombre;
    /** @ORM\Column(type="integer") **/
    private $fundacion;
    /** @ORM\Column(type="integer") **/
    private $socios;
    /** @ORM\Column(type="string") **/
    private $ciudad;
    public function getId()
    {
        return $this->id;
    }
    public function getNombre()
    {
        return $this->nombre;
    }
    public function setNombre($nombre)
    {
        $this->nombre = $nombre;
    }
    public function getFundacion()
    {
        return $this->fundacion;
    }
    public function setFundacion($fundacion)
    {
        $this->fundacion = $fundacion;
    }
    public function getSocios()
    {
        return $this->socios;
    }
    public function setSocios($socios)
    {
        $this->socios = $socios;
    }
    public function getCiudad()
    {
        return $this->ciudad;
    }
    public function setCiudad($ciudad)
    {
        $this->ciudad = $ciudad;
    }
}
```

Escaneado con CamScanner

En el ejemplo siguiente se puede ver cómo manejar las entidades. En las primeras líneas se incluyen tanto la entidad que se quiere usar como el fichero **bootstrap.php**. En ese fichero se declara `$entityManager`, que tiene una serie de métodos para, como su nombre indica, gestionar las entidades.

Por ejemplo, el método `find()` sirve para buscar una entidad por clave. Recibe el nombre de la clase y un valor. Para buscar el equipo con id 1 se usaría:

```
$eq = $entityManager->find("Equipo", 1);
```

Como la entidad equipo está asociada, mediante las anotaciones, con la tabla Equipos, `find()` buscará en esa tabla algún equipo con código uno. Si existe, devuelve un objeto de la clase Equipos en la que los atributos tendrán el valor de las columnas correspondientes.

El objeto devuelto se puede manipular a través de sus métodos, como muestra el siguiente ejemplo:

```
1  <?php
2  require_once './src/Equipo.php';
3  require_once "bootstrap.php";
4  // buscar por clave primaria
5  $eq = $entityManager->find("Equipo", 1);
6  // mostrar datos
7  echo $eq->getSocios();
8  // cambiar el número de socios
9  $eq->setSocios(70000);
10 $entityManager->flush();
11 // si el equipo no existe devuelve NULL
12 $eq = $entityManager->find("Equipo", 4);
13 if(!$eq){
14     echo "Equipo no encontrado";
15 }
```

En la línea 9 se modifica el número de socios. En la línea 10 se utiliza el método `flush()`. Este método hace que se salven en la base de datos los cambios que se hayan producido en las entidades. Es en este momento cuando realmente se modifica la base de datos. Doctrine se encarga de ejecutar la sentencia SQL correspondiente.

En la línea 12 se busca un equipo con código 4, que no existe. En este caso, el método devuelve NULL.



Actividad propuesta 7.1

Escribe un programa que reciba por la URL el código de un equipo y muestre sus datos. Si el equipo no existe, tiene que mostrar un mensaje apropiado.

7.2.3. Inserción y borrado

Para realizar una inserción:

- Se crea un nuevo objeto.

Escaneado con CamScanner

- Se indica que debe almacenarse con la base de datos con el método `persist()`.
- Se llama al método `flush()` para que realmente se produzca la inserción.

```
<?php
require_once './src/Equipo.php';
require_once "bootstrap.php";
$nuevo = new Equipo();
$nuevo->setNombre('Real Madrid');
$nuevo->setFundacion(1900);
$nuevo->setSocios(50000);
$nuevo->setCiudad('Madrid');
$entityManager->persist($nuevo);
$entityManager->flush();
echo "Equipo insertado ". $nuevo->getId(). "\n";
```

Actividad propuesta 7.2

Escribe un formulario para insertar un nuevo equipo. Si hay algún error al insertar, hay que mostrar un mensaje apropiado.

Para borrar un registro:

- Se selecciona el objeto correspondiente.
- Se elimina con el método `remove()`.
- Se llama al método `flush()` para que realmente se produzca el borrado.

El ejemplo **borrar.php** borra un equipo a partir de su id, que recibe en la URL.

```
<?php
require_once "bootstrap.php";
require_once './src/Equipo.php';
$id = $_GET['id'];
/*buscar el jugador con el id indicado*/
$equipo = $entityManager->find("Equipo", $id);
if(!$equipo){
    echo "Equipo no encontrado";
}else{
    $entityManager->remove($equipo);
    $entityManager->flush();
    echo "Equipo borrado";
}
```

7.3. Asociaciones

Las asociaciones vinculan unos objetos con otros. Son la contraparte en la base de datos de objetos de las claves ajenas de las tablas, pero con diferencias importantes.

Hay varios tipos de asociaciones, se diferencian por:

- Su cardinalidad: muchos a uno, uno a muchos, uno a uno, muchos a muchos...

Escaneado con CamScanner

- Direccionalidad: unidireccionales o bidireccionales.
- Si incluyen una referencia a la propia entidad.

En este apartado se presentan las más utilizadas, especialmente si se parte de un modelo relacional ya creado.

7.3.1. Asociaciones muchos a uno unidireccionales

Este es el caso típico para una clave ajena relacional. Por ejemplo, muchos jugadores tienen un mismo equipo. Si una tabla tiene una clave ajena, en la entidad correspondiente habrá una referencia a un objeto de la entidad correspondiente a la otra tabla.

En el caso de las tablas de jugadores y equipos, esto se traduce en que la entidad Jugador tendrá un atributo que será una referencia a un objeto de la clase Equipo.

Esta es una de las diferencias principales entre las asociaciones y las claves ajenas del modelo relacional:

- La tabla de jugadores tiene una columna Equipo, que representa el equipo del jugador. Contiene el id del equipo correspondiente, que en este caso es un entero.
- En cambio, el atributo correspondiente en la clase Jugador es un objeto de clase Equipo. Es decir, el atributo equipo en la clase no contiene el código del equipo, sino una referencia al equipo correspondiente.

La clase es bastante parecida a Equipo, pero hay una novedad. El atributo equipo está anotado con:

- `@ManyToOne`, que indica que se trata de una relación *muchos a uno*. El atributo `targetEntity` especifica la entidad asociada.
- `@JoinColumn`, para indicar qué columnas se utilizan en la unión de las tablas. El atributo `name` es el nombre en la tabla que tiene la referencia (en este caso, Jugador) y `referencedColumn` es la columna de la otra tabla (Equipo).

```
<?php
use Doctrine\ORM\Mapping as ORM;
/**
* @ORM\Entity @ORM\Table(name="jugador")
**/
class Jugador{
    /** @ORM\Id @ORM\Column(type="integer") @ORM\GeneratedValue **/
    private $id;
    /** @ORM\Column(type="string") **/
    private $nombre;
    /** @ORM\Column(type="string") **/
    private $apellidos;
    /** @ORM\Column(type="integer") **/
    private $edad;
    /**
    * @ORM\ManyToOne(targetEntity="Equipo")
```

Escaneado con CamScanner

```
     * @ORM\JoinColumn(name="equipo",
        referencedColumnName="id")
     **/
     private $equipo;
     public function getId(){
        return $this->id;
     }
     public function getNombre(){
        return $this->nombre;
     }
     public function setNombre($nombre){
        $this->nombre = $nombre;
     }
     public function getApellidos(){
        return $this->apellidos;
     }
     public function setApellidos($apellidos){
        $this->apellidos = $apellidos;
     }
     public function getEdad(){
        return $this->edad;
     }
     public function setEdad($edad){
        $this->edad = $edad;
     }
     public function getEquipo(){
        return $this->equipo;
     }
     public function setEquipo($equipo)
     {
        $this->equipo = $equipo;
     }
}
```

En el ejemplo **jugador_equipo.php** se utiliza la asociación para acceder al equipo de un jugador. Recibe el id de un jugador a través de la URL y lo busca. Si el jugador existe, muestra su nombre y el de su equipo.

```
1  <?php
2  require_once "bootstrap.php";
3  require_once './src/Equipo.php';
4  require_once './src/Jugador.php';
5  $id = $_GET['id'];
6  /*buscar el jugador con el id indicado*/
7  $jugador = $entityManager->find("Jugador", $id);
8  if(!$jugador){
9     echo "Jugador no encontrado";
10 }else{
```


Escaneado con CamScanner

```
11      echo "Nombre del jugador: ". $jugador->getNombre()."<br>";
12      $equipo = $jugador->getEquipo();
13      echo "Nombre del equipo: ". $equipo->getNombre()."<br>";
14  }
```

En la línea 7, se obtiene el objeto Jugador a partir del parámetro de la URL.

En la línea 12, `getEquipo()` devuelve una referencia al objeto correspondiente, que es de lase Equipo.

En la línea 13, se muestra el nombre del equipo utilizando el método `getNombre()`.

RECUERDA

✓ Las dos entidades tienen un método getNombre().

7.3.2. Asociaciones muchos a uno bidireccionales

En las asociaciones bidireccionales, ambas entidades reciben referencias de la entidad asociada. Esta es otra de las diferencias importantes con el modelo relacional.

En el caso de los Equipos y Jugadores, la bidireccionalidad consistiría en añadir un atributo ugadores a la clase Equipos. Como en un equipo hay muchos jugadores, hay que almacenar nuchas referencias. En lugar de un *array* normal de PHP, se usa una clase de Doctrine, `Array-Collection`.

Veamos cómo implementarlo con las clases `EquipoBidireccional` y `JugadorBidireccional`.

La clase `EquipoBidireccional` es como equipo, pero añade este campo. También hay que añadir un constructor para inicializar el objeto `ArrayCollection`.

```
/**
* Un equipo tiene muchos jugadores
* @ORM\OneToMany(targetEntity="JugadorBidireccional",
    mappedBy="equipo")
*/
private $jugadores;
public function __construct() {
    $this->jugadores = new ArrayCollection();
}
```

En la clase Jugador la anotación del atributo `$equipo` se modifica para indicar que es una asociación inversa. Con el atributo `inversedBy` se indica el atributo de Equipo que almacenará los jugadores.

```
/**
* @ORM\ManyToOne(targetEntity="EquipoBidireccional",
    inversedBy = "jugadores")
* @ORM\JoinColumn(name="equipo", referencedColumnName="id")
*/
    private $equipo;
```

Escaneado con CamScanner

El ejemplo **probar_bidireccional.php** recibe en la URL el código de un equipo y muestra los nombres de los jugadores. Para cargar los jugadores usa el método `getJugadores()`, así que no hay que hacer una consulta. Doctrine carga los datos a través de la información contenida en las anotaciones.

```
<?php
require_once "bootstrap.php";
require_once './src/EquipoBidireccional.php';
require_once './src/JugadorBidireccional.php';
$id = $_GET['id'];
$equipo = $entityManager->find("EquipoBidireccional", $id);
if(!$equipo){
    echo "Equipo no encontrado";
}else{
    echo "Nombre del equipo: ". $equipo->getNombre()."<br>";
    $jugadores = $equipo->getJugadores();
    echo "Lista de jugadores"."<br>";
    foreach($jugadores as $jugador){
        echo "Nombre: ". $jugador->getNombre()."<br>";
    }
}
```

7.4. Consultas básicas

En Doctrine un repositorio es una clase que contiene consultas relacionadas. Se puede obtener un repositorio básico con métodos útiles para cualquier clase con el método `getRepository()` de `EntityManager`. Recibe el nombre de una entidad y devuelve un objeto de clase `EntityRepository`.

El ejemplo **findBy.php** usa algunos:

- `findBy()`. Para utilizar criterios de búsqueda, que se introducen en un *array*.
- `findOneBy()`. Como el anterior, pero devuelve solo un resultado.
- `findAll()`. Devuelve todas las filas de la tabla asociada.

```
<?php
require_once "bootstrap.php";
require_once './src/Jugador.php';
require_once './src/Equipo.php';
/*Con findBy/findOneBy:
-Jugadores con exactamente XX años..*/
echo "Jugadores con 12 años<br>";
$jugadores = $entityManager->getRepository('Jugador')
    ->findBy(array('edad' => 12));
foreach($jugadores as $jugador){
    echo "Nombre: ". $jugador->getNombre().
    " ". $jugador->getApellidos(). "<br>";
}
```


Escaneado con CamScanner

```
//Equipos de Madrid fundados en 1900.
echo "Equipos de Madrid fundados en 1900<br>";
$equipos = $entityManager->getRepository('Equipo')
    ->findBy(array(
    'fundacion' => 1900,
    'ciudad'=>'Madrid')
    );
foreach($equipos as $equipo){
echo "Nombre: ". $equipo->getNombre()."<br>";
}
/*Equipo cuyo nombre es "Real Madrid"*/
echo "Equipos cuyo nombre es 'Real Madrid'<br>";
$equipo = $entityManager->getRepository('Equipo')
    ->findOneBy(array('nombre' => 'Real Madrid'));
echo "Nombre: ". $equipo->getNombre(). " ".
    $equipo->getFundacion(). " ".
    $equipo->getCiudad()."<br>";
```

Actividad propuesta 7.3

Escribe un programa que reciba por la URL el nombre de una ciudad y muestre los datos de los equipos de esa ciudad.

7.5. DQL

Los métodos vistos hasta ahora para realizar consultas son bastante limitados. Doctrine tiene su propio lenguaje de consultas, llamado Doctrine Query Language, DQL.

Es bastante parecido a SQL, como se puede ver en estos ejemplos. Para obtener todos los datos de todos los jugadores se usaría:

```
SELECT j FROM jugador j
```

Para seleccionar el nombre los jugadores mayores de 30 años:

```
SELECT j.nombre FROM jugador j WHERE j.edad > 30
```

Para contar cuántos jugadores tienen más de 30 años:

```
SELECT COUNT(j.id) as num FROM jugador j WHERE j.edad > 30
```

Para ordenar los resultados:

```
SELECT j FROM jugador j ORDER BY j.edad ASC
```

Es necesario utilizar un alias para todas las entidades del FROM, no como en SQL, donde es opcional.

Escaneado con CamScanner

Actividad propuesta 7.4

Escribe una consulta DQL que devuelva los nombres de los equipos con más de 10.000 socios.

Si la consulta incluye más de una tabla y estas están asociadas, se pueden unir con JOIN Por ejemplo, para mostrar el nombre de todos los jugadores junto con el de su equipo se pued utilizar:

```
SELECT j.nombre as nombre, e.nombre as equipo FROM Jugador j JOIN j.equipo e
```

En el FROM hay dos alias:

- "j", para la entidad Jugador.
- "e", para el atributo equipo de j.

Como las entidades están asociadas, "e" representa al equipo del jugador. Es un objeto de entidad Equipo y en las consultas se puede acceder a sus atributos usando el alias.

Actividad propuesta 7.5

Escribe una consulta DQL que devuelva los nombres de los jugadores de los equipos de Madrid.

También se puede utilizar para actualizar:

```
UPDATE jugador j SET j.edad = j.edad + 1 WHERE j.edad > 20
```

Y para borrar:

```
DELETE jugador j WHERE j.edad > 30
```

Para ejecutar una consulta desde PHP:

- Se crea con `$entityManager->createQuery("SELECT …")`.
- Se ejecuta con el método `getResult()`, que devuelve un `ArrayCollection` co resultados.

```
/*todos los jugadores, todos los datos*/
$query= $entityManager->createQuery("SELECT j FROM Jugador j");
$jugadores = $query->getResult();
```


Escaneado con CamScanner

```
foreach($jugadores as $jugador){
    echo "Nombre: ". $jugador->getNombre()."<br>";
}
```

Si la consulta devuelve un único valor, también se puede utilizar `getSingleScalarResult()`.

```
echo "Contar los jugadores mayores de 30 años<br>";
$query_contar = $entityManager->createQuery(
    "SELECT COUNT(j.id) as num
    FROM jugador j
    WHERE j.edad > 30");
$num_jugadores = $query_contar->getSingleScalarResult();
echo $num_jugadores."<br>";
```

7.6. Repositorios propios

Es posible crear clases repositorio propias, derivadas de la `EntityRepository`. Puede ser una opción para agrupar consultas que se repitan en la aplicación.

La clase `EquipoRepository` tiene un método `getLista()` que recibe el nombre de un equipo y devuelve sus jugadores.

```
class EquipoRepository extends EntityRepository{
/* devuelve una colección con los jugadores del equipo, -1 si no encuentra
el equipo*/
    public function getLista($nombre_equipo) {
        $equipo = $entityManager->getRepository('Equipo')
                                ->findOneBy(array('nombre' =>
                                        $nombre_equipo));
        if(!$equipo){
            return -1;
        }else{
            $query = $entityManager ->createQuery(
                        "SELECT j
                        FROM jugador j JOIN j.equipo e
                        WHERE e.nombre = '$nombre_equipo'");
            return $query->getResult();
        }
    }
}
```

Para asociar esta clase con la clase jugador, se utiliza la anotación el atributo `repositoryClass` de `@Entity`.

```
/**
* @ORM\Table(name="equipo")
@ORM\Entity(repositoryClass="EquipoRepository")
*/
class Equipo{
```


Escaneado con CamScanner

Recurso web

En este capítulo se ven las anotaciones más habituales, pero hay más de 30. Puedes consultar una lista completa de anotaciones y atributos en la documentación de Doctrine.

https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/annotations-reference.html

Resumen

- Para comenzar a trabajar con Doctrine hay que obtener un objeto de la clase EntityManager asociado a la base de datos.
- Cada tabla de la base de datos se asocia con una entidad. Los atributos de la clase estarán asociados a las columnas de la tabla.
- Los metadatos se introducen mediante anotaciones en las clases o con ficheros XML externos.
- Las claves ajenas del modelo relacional se representan como asociaciones en la base de datos de objetos.
- El EntityManager permite obtener un objeto repositorio asociado a una clase con getRepository().
- Este objeto tiene métodos para recuperar información de la base de datos sin tener que escribir la consulta.
- El lenguaje DQL sirve para hacer consultas más complejas.
- En general, cuando se devuelve una sola fila, se devuelve un objeto de la entidad correspondiente.
- Las consultas devuelven un objeto ArrayCollection.
- La comunicación con la base de datos se produce cuando se ejecuta flush().

Ejercicios propuestos

1. Escribe un formulario para borrar equipos usando Doctrine.
2. Escribe una consulta que devuelva todos los equipos ordenados por año de fundación.
3. Añade esta consulta a un repositorio y escribe un programa para probarlo.
4. La base de datos Doctrine también incluye la tabla:

   - Partido(Id, Local, Visitante, Goles_local, Goles_visitante, Fecha)
     Id es la clave primaria y Local y Visitante son claves ajenas de Equipo.

Escaneado con CamScanner

Actividades de autoevaluación

1. ¿Cuál de estas afirmaciones sobre los ORM no es cierta?
   - ☐ *a)* Asocian los objetos de la aplicación con la base de datos.
   - ☐ *b)* No se basan en el modelo relacional.
   - ☐ *c)* Liberan al programador de tareas repetitivas.

2. En general, al utilizar un ORM habrá una entidad por cada:
   - ☐ *a)* Base de datos.
   - ☐ *b)* Tabla de la base de datos.
   - ☐ *c)* Usuario de la base de datos.

3. Una entidad es una clase:
   - ☐ *a)* Normal.
   - ☐ *b)* Que hereda de la clase Entity.
   - ☐ *c)* Sincronizada con la base de datos.

4. Las anotaciones son:
   - ☐ *a)* Comentarios para explicar el programa.
   - ☐ *b)* Metadatos que se incluyen en bloques de comentarios.
   - ☐ *c)* Un tipo de entidad definido por Doctrine.

5. El DQL se utiliza para:
   - ☐ *a)* Hacer consultas sobre la base de datos de objetos.
   - ☐ *b)* Definir las entidades.
   - ☐ *c)* Controlar el acceso a la base de datos.

6. En Doctrine los repositorios agrupan:
   - ☐ *a)* Entidades relacionadas.
   - ☐ *b)* Consultas relacionadas.
   - ☐ *c)* Anotaciones relacionadas.

7. Para las asociaciones se utiliza la anotación:
   - ☐ *a)* @Asociation.
   - ☐ *b)* @ID.
   - ☐ *c)* @JoinColumn.

Escaneado con CamScanner

8. El método flush() se emplea para:

- ☐ *a)* Deshacer los cambios realizados.
- ☐ *b)* Salvar los cambios realizados.
- ☐ *c)* Resetear la conexión con la base de datos.

9. Los datos de conexión con la base de datos se introducen:

- ☐ *a)* Como argumento del constructor de EntityManager.
- ☐ *b)* En fichero bootstrap.php.
- ☐ *c)* En las anotaciones de las entidades.

10. Para borrar una entidad:

- ☐ *a)* Se utiliza remove().
- ☐ *b)* Se utiliza erase().
- ☐ *c)* Hay que utilizar una consulta DQL.

SOLUCIONES:

1.	a	**b**	c	**5.**	**a**	b	c	
2.	a	**b**	c	**6.**	a	**b**	c	**9.** a **b** c
3.	a	b	**c**	**7.**	**a**	b	c	**10.** **a** b c
4.	a	**b**	c	**8.**	a	**b**	c	

Escaneado con CamScanner

8

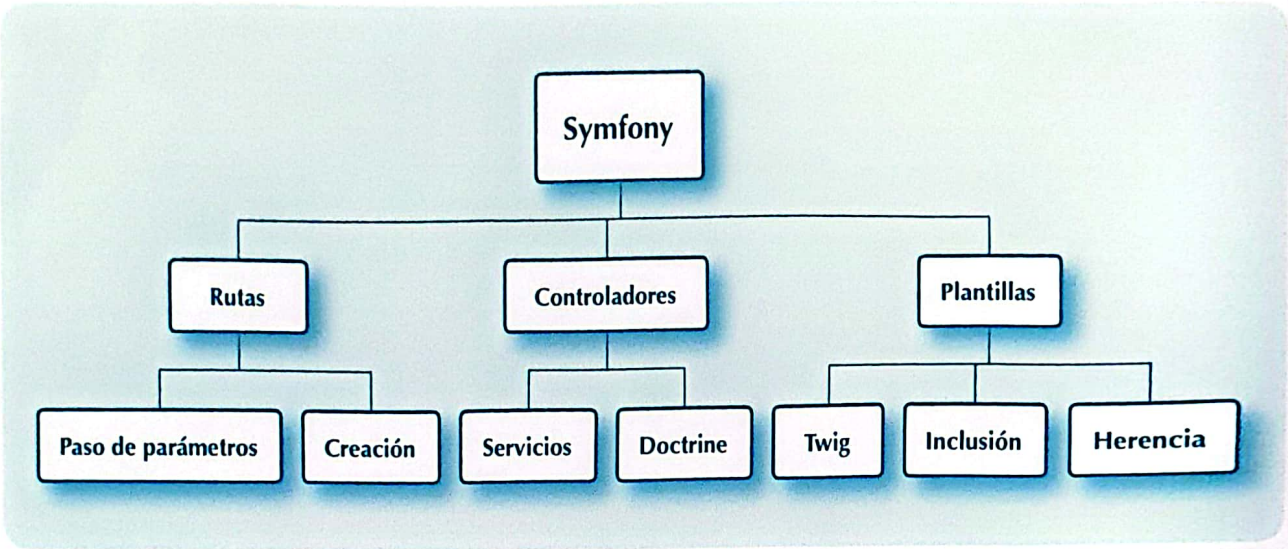
Desarrollo de aplicaciones en Symfony

Objetivos

- ✓ Conocer el patrón MVC y su utilidad para el desarrollo web.
- ✓ Identificar los *frameworks* más importantes que implementan el patrón MVC.
- ✓ Entender la arquitectura de las aplicaciones Symfony.
- ✓ Comprender el formato de rutas de Symfony.
- ✓ Utilizar anotaciones para asociar rutas con controladores.
- ✓ Aprender a utilizar el sistema de plantillas Twig.

Escaneado con CamScanner

Mapa conceptual



Glosario

Controlador. En Symfony, los controladores son las funciones encargadas de procesar la petición del cliente.

Patrón MVC. Patrón de diseño que divide las aplicaciones en tres partes: modelo, vista y controlador.

Plantilla. Fichero en el que se delega la salida. Tiene una parte estática y otra dinámica.

Ruta. Asocian las URL solicitadas por el cliente con el controlador correspondiente.

Servicio. En Symfony, es un objeto que provee alguna funcionalidad útil para el desarrollo, como enviar un correo.

Symfony. *Framework* para desarrollo web en PHP que sigue el patrón MVC.

Twig. Es el sistema de plantillas por defecto de Symfony.

YAML. Es uno de los formatos disponibles para los metadatos y archivos de configuración, junto con las anotaciones y los ficheros XML.

8.1. El patrón MVC

El patrón MVC divide la aplicación en tres capas: modelo, vista y controlador. Al desacoplar los elementos de la aplicación, se consigue código reusable. Además, permite el desarrollo en paralelo, con equipos independientes para cada capa.

El patrón MVC es uno de los más extendidos para el desarrollo de aplicaciones, no solo en desarrollo web. En la actualidad hay muchos *frameworks* que utilizan este patrón o algunas de las muchas variantes que han surgido.

Escaneado con CamScanner

CUADRO 8.1
Frameworks MVC

Framework	Lenguaje
Spring MVC	Java (JEE)
Symfony	PHP
ASP.NET MVC	ASP
Ruby-on-Rails	Ruby
Angular	JavaScript
TreeFrog	C++

Las tres capas de la aplicación son:

1. *Modelo:* en el modelo está la lógica de negocio. Se encarga de manejar la base de datos de la aplicación.
2. *Vista:* la interfaz gráfica. Una vista muestra una parte del modelo al usuario.
3. *Controlador:* se encarga de recoger las acciones del usuario y, en función de estas, interactúa con el modelo.

Como se puede apreciar, la descripción básica del patrón es muy genérica. Los diferentes *frameworks* difieren en los detalles de implementación.

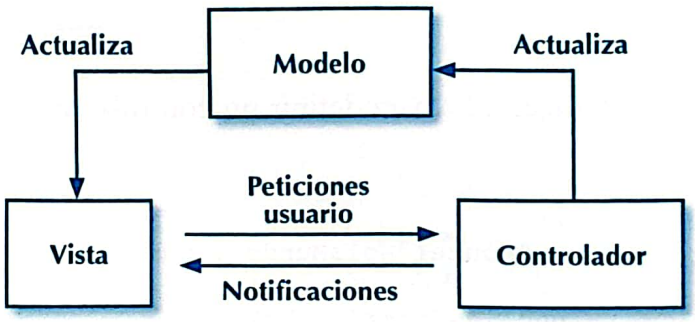


Figura 8.1
Patrón MVC.

8.2. Symfony

Symfony es un *framework* para desarrollo de aplicaciones web en PHP utilizando el patrón MVC. Como todos los *frameworks*, su objetivo es facilitar el desarrollo ofreciendo soluciones para las tareas más habituales.

Por un lado, Symfony plantea las aplicaciones web de una forma determinada a la que el desarrollador debe adaptarse.

Por otro, incluye componentes para muchas de las tareas habituales, como formularios o seguridad. Además, estos componentes son librerías independientes de Symfony y pueden usarse en otros proyectos.

Aunque está impulsado por una empresa, Symfony es un proyecto de código abierto y la comunidad de usuarios ha desarrollado varios componentes interesantes. Muchas aplicaciones conocidas, como Drupal, utilizan Symfony o alguno de sus componentes.

8.2.1. Visión general

La diferencia principal entre las aplicaciones realizadas hasta ahora y Symfony está en cómo se interpretan las URL. Hasta el momento, la interacción entre cliente y servidor se basa en que

Escaneado con CamScanner

el cliente solicita explícitamente al servidor los ficheros que necesita. Por ejemplo, al acceder a localhost/holamundo.php se ejecuta el *script* **holamundo.php**. De esta manera, los ficheros (y los parámetros que reciben) son la interfaz entre cliente y servidor.

En Symfony, las URL son procesadas por un controlador de *front-end* que analiza las solicitudes para redirigir la petición al controlador adecuado. Un controlador es un método que recibe la petición del cliente, la procesa y genera una respuesta o un reenvío. Los controladores están asociados a una o más rutas: las URL en las que están disponibles.

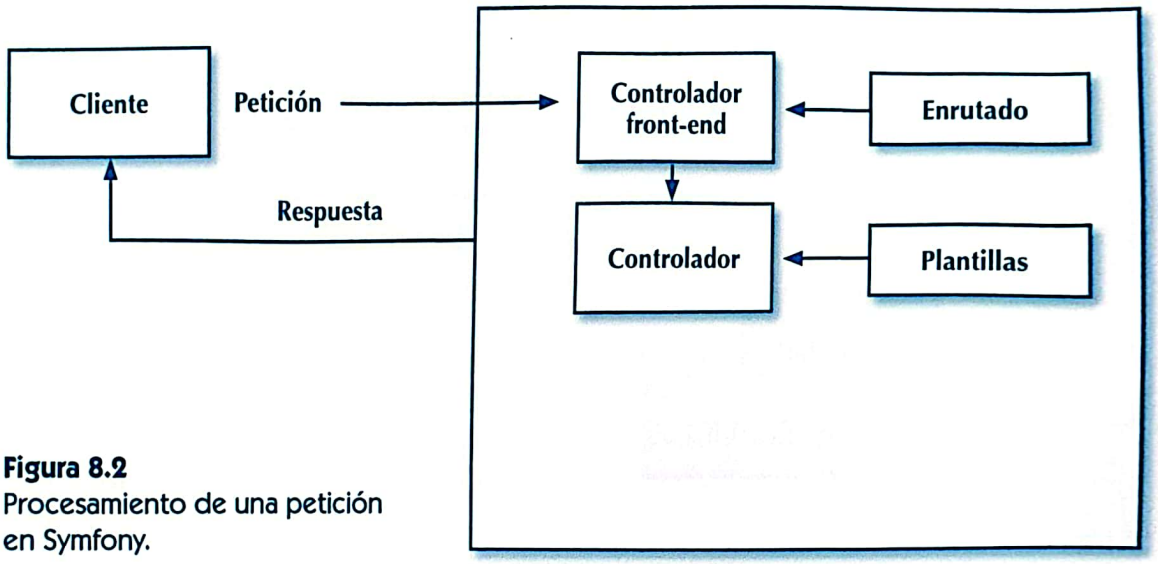


Figura 8.2
Procesamiento de una petición en Symfony.

Por ejemplo, para definir un controlador que simplemente muestre por pantalla "Hola", se haría:

```
/**
* @Route("HolaMundo", name="hola")
*/
public function hola(){
    return new Response('<html><body>Hola</body></html>');
}
```

Como en Doctrine, en Symfony se utilizan anotaciones en bloques de comentarios. En el bloque de comentarios hay una anotación `@route` que sirve para establecer la ruta en la que estará disponible el servidor. Se añade a la ruta base del servidor. Por lo tanto, para acceder al controlador se usaría:

localhost:8000/HolaMundo

De los controladores se espera que devuelvan algo o causen una redirección. En este ejemplo es una cadena de HTML, pero lo normal es utilizar una plantilla.

8.2.2. Instalación

Para crear un nuevo proyecto Symfony se utiliza `composer`:

composer create-project symfony/website-skeleton <nombre>

Escaneado con CamScanner

Se creará un directorio <nombre> en el que se descargarán los componentes de Symfony. La descarga puede llevar varios minutos.

TOMA NOTA

> Si en lugar de "website-skleton" se utiliza "skleton", se descargan menos componentes. Luego se pueden añadir los que se vayan necesitando.

Los proyectos Symfony incluyen su propio servidor web. Se pone en marcha ejecutando este comando desde el directorio del proyecto:

```
php bin/console server:run
```

La salida del comando informa si se ha podido arrancar el servidor o no. Hay que asegurarse de que no haya otro servidor escuchando previamente en el puerto 8.000.

Si todo ha ido bien, al acceder desde el navegador *a* se debería ver algo parecido a la captura recogida en la figura 8.3.

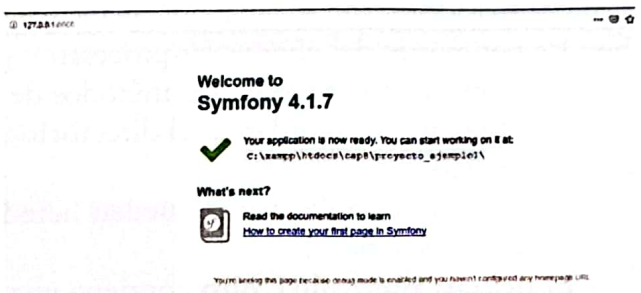


Figura 8.3
Instalación de Symfony.

8.2.3. Estructura de directorios

Al crear un nuevo proyecto se crea una jerarquía de directorios. En Symfony, la ubicación de los ficheros es muy importante. Se espera que cada tipo de componente esté en un directorio concreto. Al principio puede resultar un poco pesado, pero si se utiliza Symfony para varias aplicaciones, se observan las ventajas de estandarizar la organización del proyecto.

La estructura de directorios de un proyecto recién creado puede observarse en la figura 8.4.

Los directorios más importantes son:

- El directorio raíz del proyecto. Aquí está el fichero .env, donde se almacena la configuración de base de datos y correo.
- /config/packages. Está el fichero security.yaml, que contiene la configuración de seguridad.
- /src/Controller. Para los controladores.
- /src/Entity. Se emplea para guardar las entidades.
- /src/Templates. Se utiliza para las plantillas.

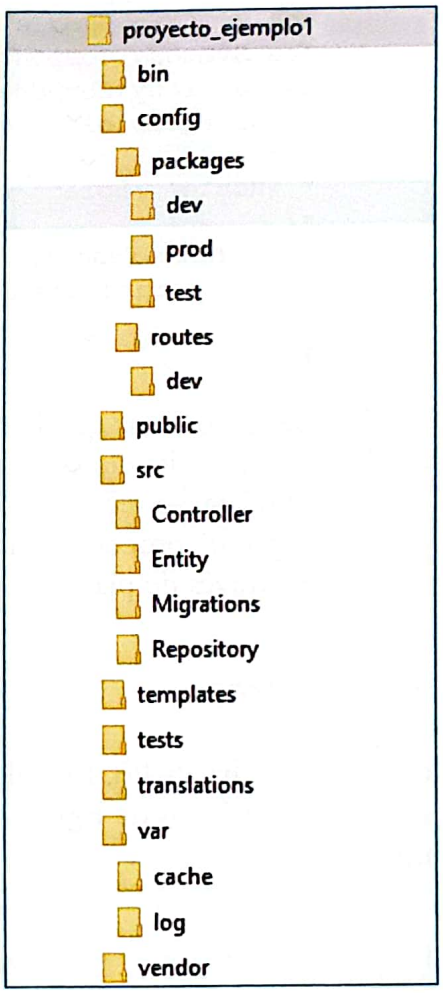


Figura 8.4
Estructura de directorios de un proyecto Symfony.

Escaneado con CamScanner

Actividad propuesta 8.1

Crea un proyecto para probar los ejemplos de este capítulo. Comprueba que el servidor se arranca y revisa la estructura de directorios.

8.3. Controladores

Los controladores son el elemento principal del desarrollo en Symfony. Son métodos que reciben las peticiones del cliente, las procesan y generan una salida o redirección.

Los controladores suelen ser métodos de una clase. Estas clases se denominan *clases controladoras*. Se tienen que guardar en el directorio `/src/Controller` y pueden crearse subdirectorios si es necesario.

Aunque no es obligatorio, pueden heredar de la clase `AbstractController`, que tiene varios métodos prácticos.

El fichero **Ejemplo1.php** contiene una de estas clases, que tiene un método controlador.

```
<?php
// src/Controller/Ejemplo1.php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
class Ejemplo1 extends AbstractController{
/**
* @Route("/hola", name="hola")
*/
    public function home_admin(){
        return new Response('<html><body>Hola</body></html>');
    }
}
```

Se puede observar que la clase está en el espacio de nombres App\Controller. Esto hay que añadirlo en todas las clases controladoras.

En Symfony se recomienda que los controladores sean lo más ligeros posible y contengan poco código. El procesamiento debería delegarse en funciones o clases auxiliares y la salida realizarse a través de plantillas.

8.4. Rutas

Como ya se ha dicho, las aplicaciones Symfony usan un formato de ruta diferente al de las aplicaciones básicas de PHP. Las rutas de los ficheros se sustituyen por rutas que se asocian con un controlador.

8.4.1. Paso de parámetros

El paso de parámetros es más sencillo. Simplemente se añaden a la URL base separados por un carácter '/'. Por ejemplo:

Escaneado con CamScanner

nombre_fichero.php?param1=value1&param2=value2

se sustituye por:

nombre_ruta/valor1/valor2

Para utilizar esos parámetros hay que incluirlos en la lista de parámetros del controlador, como se hace en este ejemplo, que recibe dos números y muestra su producto.

```
/**
* @Route("/producto/{num1}/{num2}", name="producto")
*/
public function producto($num1, $num2){
    $producto = $num1 * $num2;
    return new Response("<html><body>". $producto. "</body></html>");
}
```

Como se puede ver en el ejemplo anterior, la anotación `@Route` puede tener un atributo `name`. Sirve para referirse a esta ruta desde otra parte de la aplicación.



Actividad propuesta 8.2

Escribe un controlador que reciba un número y muestre su factorial. Hay que comprobar que el parámetro sea realmente un número y que no sea negativo.

8.4.2. Valores por defecto

Es posible dar valores opcionales a los argumentos de dos maneras. Una es poniendo el valor por defecto en la lista de argumentos de la función:

```
/**
* @Route("/defecto1/{num}", name="defecto1")
*/
public function defecto1($num = 3){
    return new Response("<html><body>".$num."</body></html>");
}
```

La otra forma es, en la anotación, añadiendo el símbolo '?' y el valor después del argumento.

```
/** * @Route("/defecto2/{num?4}", name="defecto2") */
public function defecto2($num){
    return new Response("<html><body>". $num."</body></html>");
}
```


Escaneado con CamScanner

8.4.3. Redirección

Los controladores pueden redirigir a otra ruta en lugar de devolver una página. Si se trata de una ruta sin parámetros, se utiliza simplemente:

```
return $this->redirectToRoute('nombre_ruta');
```

Para indicar la ruta a la que se redirige se utiliza su atributo `name`. Si la ruta tiene parámetros se pasan en un *array* indicando nombre y valor.

```
return $this->redirectToRoute('nombre_ruta', array('nombre1' => valor1,
'nombre1' => valor2));
```

En el siguiente ejemplo, se calcula el cuadrado de un número que se pasa como argumento utilizando el controlador del ejemplo anterior. A partir del parámetro num, se redirige a la ruta /producto/num/num.

```
/**
* @Route("/cuadrado/{num}", name="cuadrado")
*/
public function cuadrado($num){
    return $this->redirectToRoute('producto', array('num1' => $num,
        'num2' => $num));
}
```

8.4.4. Rutas a nivel de clase

La anotación `@Route` también se puede utilizar sobre las clases, en lugar de sobre los métodos individuales. La ruta indicada se antepone a la de todos los controladores de la clase. En el siguiente ejemplo, para acceder al controlador `hola()`, habrá que utilizar localhost/base/hola.

```
<?php
// src/Controller/EjemploRutaBase.php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
/**
* @Route("/base")
*/
class EjemploRutaBase extends AbstractController{
    /**
    * @Route("/hola")
    */
    public function hola(){
        return new Response('<html><body>Hola</body></html>');
    }
}
```


Escaneado con CamScanner

8.4.5. Rutas disponibles

Para ver todas las rutas definidas en la aplicación, se puede utilizar el siguiente comando desde el directorio del proyecto (figura 8.5).

```
php bin/console debug:router
```

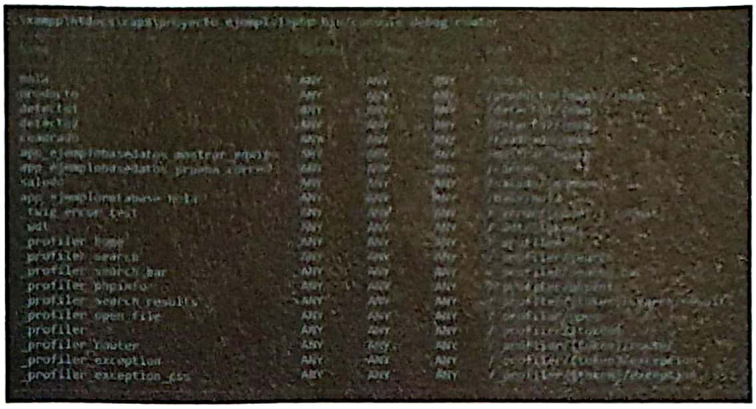


Figura 8.5
Rutas disponibles en la aplicación.

8.5. Plantillas

En los ejemplos anteriores los controladores devuelven un objeto Response que se crea con una cadena de HTML. En general, la salida de los controladores en Symfony se realiza a través de plantillas, ya sea HTML, XML o cualquier otro formato.

Estas plantillas contienen la parte estática de la página y también la lógica de presentación. De esta manera, se desacopla la lógica de negocio de la de presentación. Pueden recibir argumentos, que son los datos que hay que mostrar.

Por ejemplo, para mostrar una tabla de empleados se puede utilizar una plantilla que reciba como argumento un *array* de empleados. La plantilla se encarga de generar las etiquetas correspondientes a la tabla.

8.5.1. Introducción a Twig

Se pueden utilizar plantillas en PHP o utilizar la librería Twig, que es la opción por defecto. El siguiente ejemplo es una plantilla HTML con Twig.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Saludo</title>
    </head>
    <body>
        Hola {{ nombre }}
    </body>
</html>
```

La única parte dinámica es `{{ nombre }}`. Con esta notación se hace referencia a un parámetro de la plantilla. Al mostrar la plantilla, esa parte se sustituirá por el valor del parámetro nombre.

RECUERDA

✓ La extensión de las plantillas es <formato_generado>.twig. Por ejemplo, xml.twig o html.twig.

Escaneado con CamScanner

El controlador `saludo()` recibe como parámetro un nombre y devuelve una página web con el texto "Hola <nombre>", utilizando la plantilla anterior.

```
<?php
// src/Controller/EjemploPlantillas.php
namespace App\Controller;
Use symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
   class EjemploPlantillas extends AbstractController{
   /**
   * @Route("/saludo/{nombre}", name="saludo")
   */
      public function saludo($nombre){
         return $this->render('saludo.html.twig', array ( 'nombre' =>
         $nombre));
      }
   }
```

El controlador pasa el parámetro a la plantilla, que genera la página completa. De esta manera, si se accede a localhost/saludo/Paco, el navegador mostrará "Hola Paco".

Para mostrar una plantilla desde un controlador se utiliza el método `render` (se hereda de `AbstractController`). Este método recibe la ruta de la plantilla y, opcionalmente, un *array* con parámetros. Las claves del *array* tienen que coincidir con los nombres de los parámetros que se usan en la plantilla.

En Twig hay tres tipos de etiquetas:

- {{ ... }}. Para introducir el valor de un parámetro o expresión.
- {% ... %}. Para introducir lógica o definir secciones.
- {# ... #}. Para comentarios.

En las plantillas puede haber estructuras condicionales y bucles. Por ejemplo, la siguiente plantilla recibe como parámetro un número y muestra un mensaje según sea positivo o no.

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Saludo</title>
   </head>
   <body>
      {% if numero > 0 %}
      Positivo
      {% else %}
      Menor o igual que 0
      {%endif%}
   </body>
</html>
```

Para probarla, se puede utilizar este controlador.

Escaneado con CamScanner

```
/**
* @Route("/positivo/{num}", name="positivo")
*/
public function positivo($num){
    return $this->render('if.html.twig', array('numero'=> $num));
}
```



Actividad propuesta 8.3

Escribe un controlador que reciba un número y muestre su factorial utilizando una plantilla. La plantilla recibirá el resultado y un parámetro llamado error. Si error es TRUE, en lugar del resultado hay que mostrar un mensaje apropiado.

El siguiente ejemplo es más complicado. Utiliza un bucle para crear las filas de una tabla. Recibe el argumento filas, que se espera que sea un *array* o similar (iterable). Para cada elemento del *array* introduce una fila usando los campos de ese elemento. Estos elementos tienen que tener campos `codigo` y `nombre`.

```
!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Saludo</title>
    </head>
    <body>
        <table>
            <tr><th>Código</th><th>Nombre</th></tr>
            {%for fila in filas %}
                <tr><td>{{ fila.codigo }}</td>
                <td>{{ fila.nombre }}</td></tr>
            {% endfor %}
        </table>
    </body>
</html>
El controlador tabla usa la plantilla con un array.
/**
* @Route("/tabla/", name="tabla")
*/
public function tabla(){
    $filas = array(array('codigo'=> '1', 'nombre' =>'Sevilla' ),
    array('codigo'=> '2', 'nombre' =>'Madrid' ));
        return $this->render('tabla.html.twig', array ( 'filas' => $fi-
        las));
}
```


Escaneado con CamScanner

8.5.2. Rutas en plantillas

Para introducir una ruta dentro de la plantilla, por ejemplo, en el atributo `href` de un vínculo, se usa la función `path()`. Si no tiene parámetros, es simplemente:

```
{{ path('nombre_ruta') }}
```

Si hay parámetros se añaden en un array.

```
{{ path('nombre_ruta', {'param1': valor1, 'param2': valor2) }}
```

También existe la función `url()`, que devuelve una ruta absoluta y se usa de la misma forma.

8.5.3. Inclusión y herencia

Es posible combinar varias plantillas mediante inclusión y herencia. La inclusión consiste simplemente en insertar una plantilla dentro de otra. Por ejemplo, si varias páginas tienen una cabecera común, se puede hacer una plantilla cabecera:

```
<a href="{{ path('categorias') }}">Home</a>
<a href="{{ path('carrito') }}">Carrito</a>
<a href="{{ path('logout') }}">Cerrar sesión</a>
```

e incluirla en las demás plantillas con:

```
{{ include('cabecera.html.twig ') }}
```

La herencia entre plantillas es parecida a la herencia entre objetos. La plantilla que hereda puede modificar o ampliar la plantilla base. Una plantilla base podría ser así:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Aplicación de pedidos{% endblock %}</title>
    </head>
    <body>
        <header>
            {{ include('cabecera.html.twig') }}
        </header>
        <hr>
            {% block body %}{% endblock %}
    </body>
</html>
```

Esta plantilla:

- Incluye la plantilla cabecera.
- Define dos bloques con las etiquetas {% block ... %} y {% endblock ... %} para el título y el cuerpo.

Escaneado con CamScanner

Las plantillas que hereden de esta podrán sustituir los bloques, algo similar a la sobreescritura de métodos. Por ejemplo, una plantilla que utilice:

```
{% extends 'base.html.twig %}
{% block title %}Lista de categorías{% endblock %}
```

sería como la plantilla base pero cambiaría el título.

RECUERDA

En las plantillas se pueden usar las siguientes variables:

- app.user. El usuario actual de la aplicación. Disponible si usa el sistema de usuarios de Symfony (ver más adelante).
- app.session. Las variables de sesión.
- app.request. El objeto Request al que se está respondiendo.

8.6. Servicios

En Symfony hay una serie de objetos llamados *servicios*, que proveen funcionalidad útil para el desarrollo, como registro o envío de correos. Para utilizarlos dentro de un controlador, hay que añadir un parámetro con la clase adecuada (*type-hinting*).

Uno de estos objetos es `Request`, que tiene información sobre la petición que recibe el servidor. Se utiliza en el siguiente ejemplo:

```
/**
* @Route("/testRequest", name = "testRequest")
*/
public function testRequest(Request $request){
    $ip = $request->getClientIp();
    return new Response(
        '<html><body>IP: '.$ip.'</body></html>');
}
```

Otro servicio importante es `SessionInterface`, que sirve para manejar las variables de sesión, como se puede ver en este ejemplo que utiliza dos controladores. El primer controlador crea la variable de sesión y redirige al segundo, que la muestra.

```
/**
* @Route("/sesion1", name = "sesion1")
*/
public function sesion1(SessionInterface $session){
    $session->set("variable", 100);
    return $this->redirectToRoute('sesion2');
}
```

Escaneado con CamScanner

```
/**
* @Route("/sesion2", name = "sesion2")
*/
public function sesion2(SessionInterface $session){
    $var = $session->get("variable");
    return new Response('<html><body>'.$var.'</body></html>');
}
```

- Hay que incluir un parámetro `SessionInterface` en los dos controladores. Representa la sesión actual.
- Con el método `set()`, se asigna valor a una variable de sesión. El primer argumento es el nombre de la variable y el segundo, su valor.
- El método `get()` devuelve el valor de la variable cuyo nombre se pasa como argumento.

Se puede ver la lista de los servicios disponibles ejecutando el comando:

```
php bin/console debug:autowiring
```

desde el directorio del proyecto. Con la instalación *website-skeleton* se instalan muchos servicios, así que la salida del comando es muy larga.

8.7. Bases de datos

Aunque se pueden usar otros, Doctrine es el ORM por defecto de Symfony. La configuración de la base de datos se introduce en el fichero **.env**. Para utilizar la base de datos del capítulo anterior (Doctrine), se utilizaría:

```
DATABASE_URL=mysql://root:@127.0.0.1:3306/doctrine
```

Hay que colocar las entidades de la aplicación en el directorio /scr/Entity. Se pueden utilizar subcarpetas para organizarlas.

Para usar Doctrine en un controlador, hay que conseguir el `$entityManager` usando:

```
$entityManager = $this->getDoctrine()->getManager();
```

Este es el mismo `$entityManager` del capítulo anterior y se maneja de la misma manera, como se puede comprobar en el siguiente controlador, que utiliza Doctrine para cargar el equipo con código 1. Utiliza la entidad del capítulo anterior.

```
/**
* @Route("/mostrar_equipo")
*/
public function mostrar_equipo(){
    $entityManager = $this->getDoctrine()->getManager();
    $eq = $entityManager->find(Equipo::class, 1);
    $nombre = $eq->getNombre();
    return new Response('<html><body>'. $nombre. '</body></html>');
}
```


Escaneado con CamScanner



Actividad propuesta 8.4

Escribe un controlador que reciba el código de un equipo y muestre sus datos utilizando una plantilla. Si el equipo no existe hay que mostrar un mensaje de error.

8.8. Formularios

Los formularios son una de las tareas más habituales en el desarrollo web. Symfony tiene un componente específico. Al principio puede resultar complejo, pero se puede utilizar también en proyectos que no usen Symfony.

Para crear un formulario:

- Se obtiene un objeto `FormBuilder` a partir del método `CreateFormBuilder()`, de `AbstractController`.
- Los controles del formulario se añaden a este objeto utilizando el método `add()`.
- Una vez añadidos todos los campos del formulario, se obtiene el formulario con el método `getForm()`.

Por defecto, los formularios se envían a su propia ruta. Para diferenciar entre cuándo se solicitan los datos y cuándo se envían, se puede utilizar `$form->isSubmitted()`.

El controlador `formuHola` muestra un formulario para que el usuario introduzca nombre y apellido. Al enviarlo, muestra un saludo con esos datos.

```
class EjemploFormularios extends AbstractController{
/**
* @Route("/formuHola", name = "formuHola")
*/
public function formuHola(Request $request) {
    $form = $this->createFormBuilder()
        ->add('nombre', TextType::class)
        ->add('apellido', TextType::class)
        ->add('Enviar', SubmitType::class, array('label'=>'Sumar'))
        ->getForm();
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $datos = $form->getData();
            $nombre = $datos['nombre'];
            $apellido = $datos['apellido'];
            return new Response('<html><body>Hola '. $nombre.' '. $apelli-
            do
                .'</body></html>');
        }
        return $this->render('formuHola.html.twig',
            array('form' => $form->createView()));
    }
}
```


Escaneado con CamScanner

La plantilla para mostrar un formulario es simplemente:

```
{{ form_start(form) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

Symfony se encarga de generar el HTML a partir del objeto `$form`.

TOMA NOTA



Por supuesto, también se pueden utilizar formularios HTML directamente en las plantillas TWIG, como cualquier elemento de HTML.

8.9. Envío de correo

Symfony utiliza la librería SwiftMailer para enviar correos. Está incluida en la instalación con opción website-skeleton. El servidor de correo se configura con la variable MAILER_URL, que está en fichero .env. Para usar una cuenta de Gmail se utiliza:

```
MAILER_URL=gmail://<usuario>:<contraseña>@localhost
```

RECUERDA

✓ Hay que permitir el acceso a "Aplicaciones menos seguras" en la configuración de cuenta de Gmail.

Los correos se crean utilizando la clase SwiftMessage. Enviar un correo es muy fácil, como se puede ver en este ejemplo:

```
/**
* @Route("/correo")
*/
public function prueba_correo(\Swift_Mailer $mailer){
    $message = new \Swift_Message();
    $message->setFrom('pruebas@consymfony.com');
    $message->setTo('direccion@consymfony.com');
    $message->setBody("Pruebas");
    $mailer->send($message);
    return new Response('<html><body>Enviado</body></html>');
}
```

Para el cuerpo del correo también se puede utilizar una plantilla.

Escaneado con CamScanner

Actividad propuesta 8.5

Escribe un controlador para enviar un correo electrónico que reciba como parámetros (al menos) la dirección de destino y el cuerpo del correo.

8.10. Seguridad. Usuarios y roles

Symfony cuenta con un componente de seguridad basado en la idea de usuarios y roles. Aunque no es obligatorio, resulta muy útil que las aplicaciones lo utilicen.

El componente de seguridad se configura a través del fichero **/config/packages/security.yaml**. Inicialmente tiene esta estructura:

```
security:
    providers:
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
    access_control:
```

TOMA NOTA

En los ficheros YAML la tabulación es parte de la sintaxis.

- Dentro de la sección `providers` se define el *proveedor de usuarios*, es decir, de dónde obtiene Symfony los usuarios. Hay varias opciones, pero la más habitual es que este proveedor sea una base de datos.
- La sección `firewalls` es para la autenticación de usuarios. El *firewall* `dev` es para uso interno de Symfony. Con `anonymous: true`, se permite el acceso a usuarios que no hayan hecho *login*.
- La sección `access_control` permite limitar el acceso a determinadas rutas según el rol del usuario.

Para relacionar el sistema de usuarios de Symfony con una tabla de usuarios hay que indicar a Symfony qué entidad los representa. Veamos cómo configurar el fichero para utilizar la tabla de restaurantes de la aplicación de pedidos del capítulo 4.

```
security:
    providers:
        pedidos:
            entity:
              class: App\Entity\Restaurante
              property: correo
```


Escaneado con CamScanner

```
encoders:
    App\Entity\Restaurante:
    algorithm: plaintext
```

Se define un proveedor con nombre *pedidos*. Se asocia con la entidad Restaurante (que representa la tabla de restaurantes). En `property` se indica el atributo que se usa como nombre de usuario para hacer login. La sección `encoders` se usa para especificar cómo se encripta la clave; en este caso, texto plano.

La entidad que representa a los usuarios debe implementar las interfaces:

1. UserInterface, con los métodos:

   a) `getUsername()`. Devuelve el nombre de usuario, que será uno de los atributos de la clase.
   b) `getPassword()`. Devuelve la clave del usuario, que será otro atributo.
   c) `getRoles()`. Devuelve un *array* con los roles del usuario.
   d) `getSalt()`. La sal es un parámetro que usan algunos algoritmos de encriptación. Si se usó, hay que indicarlo.
   e) `eraseCredentials()`. Para borrar datos privados de la entidad antes de serializarla.

2. Serializable:

   a) `serialize()`. Tiene que serializar al menos el nombre de usuario, la clave y, si se usó, la sal.
   b) `unserialize()`. Realiza la operación inversa.

La entidad Restaurante se crea como se vio en el capítulo anterior y se le añaden los métodos de las interfaces. Los métodos `getUsername()` y `getPassword()` tienen que devolver el correo y la clave del restaurante, respectivamente.

RECUERDA

✓ En el capítulo 9 se utiliza el componente de seguridad de Symfony para rehacer la aplicación de pedidos.

```
<?php
// src/Entity/Restaurante.php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;
/**
* @ORM\Entity @ORM\Table(name="restaurantes")
*/
class Restaurante implements UserInterface, \Serializable{
    /**
    * @ORM\Id
    * @ORM\GeneratedValue
```


Escaneado con CamScanner

```
* @ORM\Column(type="integer",name="CodRes")
*/
private $codRes;
/**
* @ORM\Column(type="string", name = "Correo")
*/
private $correo;
/**
* @ORM\Column(type="string", name = "Clave")
*/
private $clave;
/**
* @ORM\Column(type="string", name = "Pais")
*/
private $pais;
/**
* @ORM\Column(type="string", name = "CP")
*/
private $CP;
/**
* @ORM\Column(type="string", name = "Ciudad")
*/
private $ciudad;
/**
* @ORM\Column(type="string", name = "Direccion")
*/
private $direccion;
/**
* @ORM\Column(type="integer", name = "rol")
*/
private $rol;
public function getRol(){
    return $this->rol;
}
/**
* @param mixed $rol
*/
public function setRol($rol){
    $this->rol = $rol;
}
public function getCodRes(){
    return $this->codRes;
}
public function getCorreo(){
    return $this->correo;
}
public function setCorreo($correo){
    $this->correo = $correo;
}
public function getClave(){
    return $this->clave;
}
public function setClave($clave){
    $this->clave = $clave;
```

Escaneado con CamScanner

```
}
public function getPais(){
    return $this->pais;
}
public function setPais($pais){
    $this->pais = $pais;
}
public function getCP(){
    return $this->CP;
}
public function setCP($CP) {
    $this->CP = $CP;
}
public function getCiudad(){
    return $this->ciudad;
}
public function setCiudad($ciudad){
    $this->ciudad = $ciudad;
}
public function getDireccion(){
    return $this->direccion;
}
public function serialize(){
    return serialize(array(
        $this->codRes,
        $this->correo,
        $this->clave,
    ));
}
public function unserialize($serialized){
    list (
        $this->codRes,
        $this->correo,
        $this->clave,
    ) = unserialize($serialized);
}
public function setDireccion($direccion){
    $this->direccion = $direccion;
}
public function getRoles(){
    return array('ROLE_USER');
}
public function getPassword(){
    return $this->getClave();
}
public function getSalt(){
    return;
}
public function getUsername(){
    return $this->getCorreo();
}
public function eraseCredentials(){
```

Escaneado con CamScanner

```
            return;
        }
    }
```

Como se puede ver, hay un método `getRoles()`. Este método devuelve un *array* con los roles asignados al usuario. Si la aplicación no define roles, conviene utilizar el rol predefinido ROLE_USER. El programador puede definir más siempre que empiecen por ROLE_.

8.10.1. Control de acceso

La sección `access_control` del fichero **security.yaml** permite limitar el acceso a determinadas rutas según el rol del usuario. La siguiente configuración restringiría el acceso a todas las rutas que empiecen por "admin".

```
- { path: ^/admin, roles: ROLE_ADMIN }
```

Solo podrían acceder a las mismas los usuarios con rol ROLE_ADMIN.

Otra posibilidad es controlar el acceso mediante código en los controladores. Por ejemplo, para asegurarse de que el usuario haya hecho *login*.

```
$this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');
```

Si no ha hecho *login*, se le redirige al formulario de login.

También se puede usar para comprobar roles:

```
$this->denyAccessUnlessGranted(ROLE_ADMIN');
```

La última opción es usar una anotación. Puede ponerse sobre una clase o sobre un método. La siguiente anotación hace que el controlador solo esté disponible para los usuarios con ROLE_USER. Es el rol por defecto, así que sirve para que solo puedan acceder quienes hayan abierto sesión.

```
/**
* @Security("has_role('ROLE_USER')")
*/
```

8.10.2. Abrir sesión

Para asociar un formulario de *login* con el sistema de usuarios de Symfony, se añade un elemento `form_login` al *firewall* `main`.

```
security:
    providers:
        pedidos:
        entity:
```


Escaneado con CamScanner

```
            class: App\Entity\Restaurante
            property: correo
    encoders:
        App\Entity\Restaurante:
        algorithm: plaintext
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
            form_login:
                login_path: login
                check_path: login
                default_target_path: ruta_defecto
            provider: pedidos
```

Hay que indicar:

- *login_path:* la ruta a la que se envía a un usuario que no ha hecho *login* si intenta acceder a una ruta que lo requiera.
- *check_path:* la ruta a la que se envía el formulario de *login*.
- *default_target_path:* ruta por defecto a la que enviar tras hacer *login* con éxito. Opcional.
- *provider:* el proveedor de usuarios. Symfony se encargará de validar usuario y contraseña usando la tabla asociada.

Con la configuración del ejemplo, hay que crear un controlador en la ruta *login* que se encargue de mostrar el formulario. No tiene que consultar a la base de datos para ver si los datos son correctos, lo hace Symfony. Puede ser tan sencillo como:

```
class PedidosLogin extends AbstractController{
    /**
    * @Route("/login", name="login")
    */
    public function login(){
        return $this->render('login.html.twig');
    }
}
```

La plantilla con el formulario también es muy sencilla, solo hay que ocuparse de que **action** apunte a la ruta correcta.

```
{# templates/login.html.twig #}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Formulario de login</title>
    </head>
    <body>
        <form action="{{ path('login') }}" method="post">
            Usuario
```

Escaneado con CamScanner

```
            <input type="text" id="username" name="_username" />
            Clave
            <input type="password" id="password" name="_password" />
            <button type="submit">login</button>
        </form>
    </body>
</html>
```

8.10.3. Cerrar sesión

Con el sistema de usuarios de Symfony no hace falta escribir un controlador para cerrar sesión, se puede configurar en el fichero de seguridad. Dentro del *firewall* se añade un elemento `logout`.

```
security:
…
firewalls:
    dev:
        …
    main:
        …
        logout:
            path: /logout
            target: /login
```

De esta manera, si se redirige al usuario a la ruta indicada en `path` (en este caso, /logout), Symfony cierra la sesión y le redirige a la ruta indicada en `target` (en este caso, /login).

Aunque no haga falta escribir un controlador, la ruta tiene que existir. Para definir una ruta sin controlador se usa el fichero **/config/routes.yaml**. Hay que añadir:

```
logout:
    path: /logout
```

Con esta configuración, para cerrar sesión desde cualquier punto de la aplicación basta con redirigir a *logout*.

Resumen

- El patrón MVC reparte la lógica de la aplicación en tres capas: modelo, vista y controlador.
- La arquitectura de las aplicaciones en Symfony está basada en este patrón.
- Los proyectos de Symfony tienen una estructura de directorios marcada. Cada tipo de componente tiene un directorio asignado.
- La configuración de los servidores de base de datos y correo se hace en el fichero .env.

Escaneado con CamScanner

- Los controladores son el componente central de Symfony. Se encargan de procesar las peticiones del cliente y devolver una respuesta.
- La configuración de rutas asocia las URL que solicita el cliente con el controlador correspondiente. Se puede introducir mediante anotaciones.
- La salida se realiza habitualmente mediante una plantilla. Las plantillas contienen la lógica de presentación.
- El ORM por defecto en Symfony es Doctrine, pero se pueden usar otros.
- El componente de seguridad de Symfony se puede integrar con una base de datos de usuarios.
- La configuración de seguridad se guarda en el fichero security.yaml.



Ejercicios propuestos

Para estos ejercicios utiliza la base de datos Doctrine, del capítulo anterior.

1. Escribe un controlador que reciba dos números y muestre el resto al dividir el primero por el segundo. El segundo parámetro es opcional, con valor por defecto 2.
2. Escribe un controlador que muestre una tabla con los datos de todos los jugadores.
3. Escribe un controlador que reciba el código de un equipo y muestre una tabla con los datos de sus jugadores.
4. Escribe un controlador con un formulario para el ejercicio anterior.
5. Escribe un controlador con un formulario para enviar correo.
6. Escribe un controlador que devuelva una lista con vínculos a los controladores de los ejercicios propuestos 4 y 5.

Actividades de autoevaluación

1. Las rutas de la aplicación se definen:
   - ☐ *a)* Con anotaciones.
   - ☐ *b)* En el fichero .env.
   - ☐ *c)* En el directorio de los controladores.
2. Twig permite:
   - ☐ *a)* Inclusión de plantillas.
   - ☐ *b)* Herencia de plantillas.
   - ☐ *c)* Ambas.

Escaneado con CamScanner

3. El rol por defecto de los usuarios en Symfony es:
   - ☐ *a)* ROLE_BASIC
   - ☐ *b)* ROLE_USER
   - ☐ *c)* ROLE_DEFAULT

4. Las plantillas se encargan de:
   - ☐ *a)* La salida.
   - ☐ *b)* El enrutado.
   - ☐ *c)* Manejar la base de datos.

5. Los controladores son:
   - ☐ *a)* Objetos que controlan el acceso no autorizado.
   - ☐ *b)* Métodos asociados a una ruta.
   - ☐ *c)* Métodos que validan la información introducida por el usuario.

6. Para redireccionar se utiliza el método:
   - ☐ *a)* toURL().
   - ☐ *b)* header("Location:").
   - ☐ *c)* redirectToRoute().

7. Para incluir una ruta en una plantilla se utiliza:
   - ☐ *a)* path().
   - ☐ *b)* url().
   - ☐ *c)* Ambas.

8. En el fichero .env se configura:
   - ☐ *a)* La base de datos.
   - ☐ *b)* La seguridad de la aplicación.
   - ☐ *c)* El directorio de las plantillas.

9. El ORM por defecto de Symfony es:
   - ☐ *a)* Propel.
   - ☐ *b)* Doctrine.
   - ☐ *c)* Hibernate.

10. Las entidades se guardan en el directorio:
    - ☐ *a)* /src/Entity.
    - ☐ *b)* /src/Controller.
    - ☐ *c)* Cualquiera, se especifica en .env.

SOLUCIONES:

1. **a** b c
2. a b **c**
3. a **b** c
4. **a** b c
5. a **b** c
6. a b **c**
7. a b **c**
8. **a** b c
9. a **b** c
10. **a** b c

Escaneado con CamScanner

9

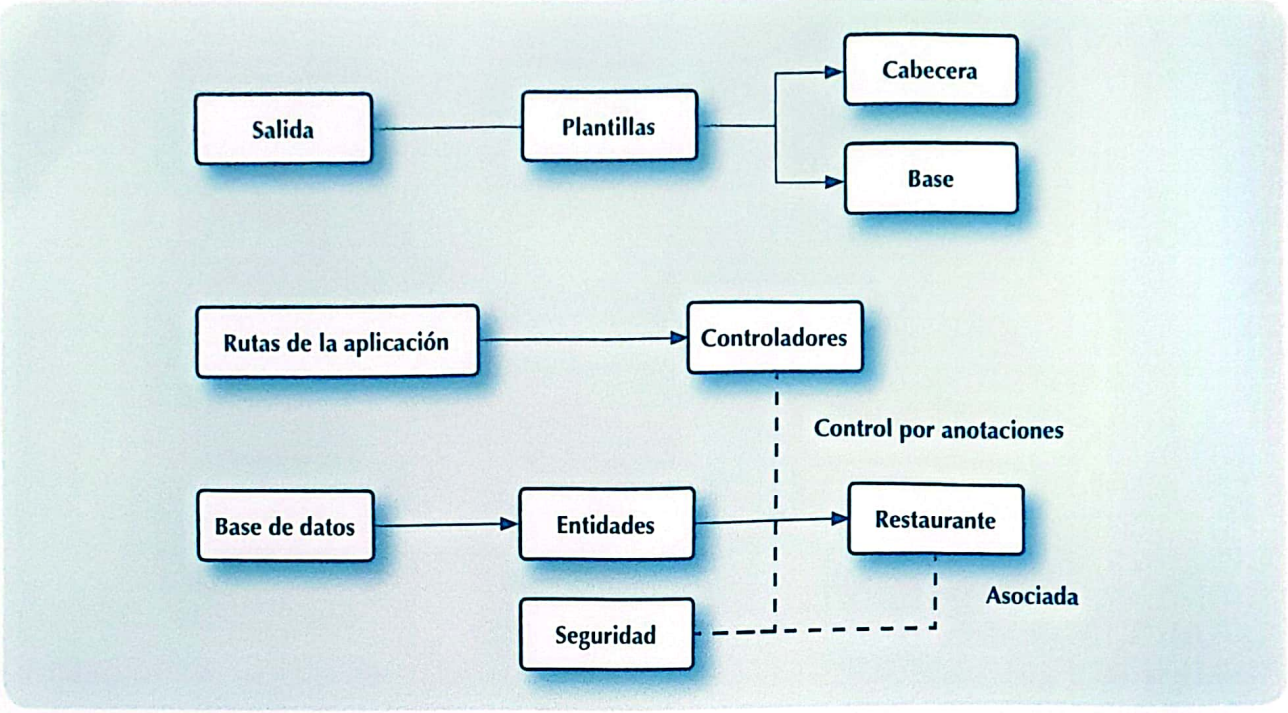
Aplicación de pedidos en Symfony

Objetivos

- ✓ Rediseñar la aplicación de pedidos con Symfony.
- ✓ Utilizar el sistema de plantillas Twig.
- ✓ Integrar el sistema de usuarios de Symfony.
- ✓ Manejar la base de datos de pedidos utilizando Doctrine.
- ✓ Valorar las ventajas de este enfoque.

Escaneado con CamScanner

Mapa conceptual



Glosario

@Security("has_role()"). Anotación para restringir el acceso a una ruta a los usuarios con un rol determinado.

app.user. Si se usa el sistema de usuarios de Symfony, esta variable está disponible en las plantillas. Tiene información sobre el usuario.

Proveedor de usuarios. El origen de los datos de los usuarios para el sistema de seguridad Symfony.

Rol. Asignando roles a los usuarios se puede restringir el acceso a determinadas partes de la aplicación.

UserInterface. En el sistema de usuarios de Symfony, la entidad que representa a los usuarios tiene que implementar esta interfaz.

9.1. Diseño de la aplicación

En este capítulo se rediseña la aplicación de pedidos del capítulo 4 como una aplicación de una sola página utilizando Symfony. La nueva aplicación tendrá el mismo aspecto y funcionalidad que la anterior. Como se podrá apreciar, con un buen diseño y conociendo el *framework*, el desarrollo con Symfony es verdaderamente rápido.

Gran parte del diseño de la aplicación se mantiene sin cambios:

Escaneado con CamScanner

- La base de datos. Utiliza la misma que las otras aplicaciones.
- El mapa de pantallas.
- La variable para el carrito.

Los cambios son:

a) La salida se hace con plantillas Twig.
b) La base de datos se maneja con Doctrine.
c) Los ficheros de la aplicación original son sustituidos por métodos controladores.

RECUERDA

✓ Si no recuerdas bien la aplicación, revisa el apartado 4.3.

9.1.1. Plantillas

Como se recordará, después del *login* todas las páginas de la aplicación tienen una estructura común. Una cabecera en la parte superior y una sección principal cuyo contenido varía. Para representarla se utilizará esta jerarquía de plantillas.

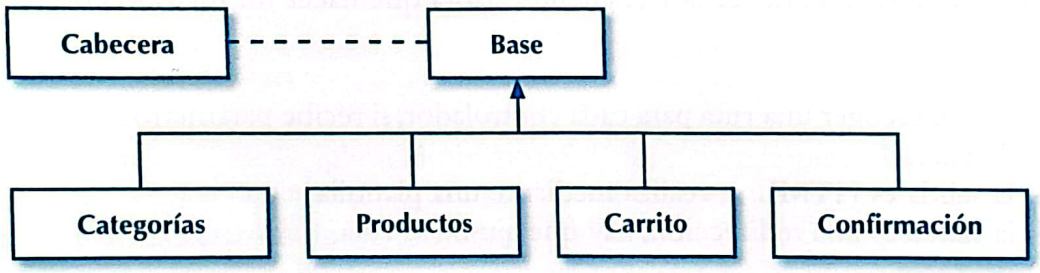


Figura 9.1
Jerarquía de plantillas.

- Una plantilla contiene la cabecera de la página.
- La plantilla base incluye la cabecera y define la estructura de la página.
- El resto de las plantillas heredan de la plantilla base y la sobrescriben para incluir su contenido específico.

Además, hay plantillas para:

— Formulario de *login*.
— El correo de confirmación de pedido.

9.1.2. Entidades

Para poder utilizar Doctrine hay que crear las entidades de la tabla de base de datos pedidos como se vio en el capítulo 7. Habrá que hacer una entidad por cada tabla. Es importante definir las asociaciones:

Escaneado con CamScanner

- Entre Categorías y Productos habrá una asociación bidireccional. Es decir, cada producto contendrá una referencia a su categoría y cada categoría tendrá un atributo con los productos de esa categoría. Así, se podrán obtener los productos de la categoría sin tener que hacer consultas.
- Entre Pedidos y Restaurantes habrá una asociación unidireccional. Es decir, cada pedido contendrá una referencia al restaurante que lo realiza, pero los restaurantes no tendrán un atributo con sus pedidos.
- Para la tabla de PedidosProductos, que se relaciona con Pedidos y Productos, usaremos dos asociaciones unidireccionales. En PedidosProductos habrá referencias a Pedidos y Productos, pero al contrario no.

La entidad Restaurantes es especial, ya que representa a los usuarios de la aplicación y se va a integrar en el componente de seguridad de Symfony. Como se comentó en el capítulo anterior, el sistema de usuarios de Symfony utiliza el concepto de *roles*, pero en esta aplicación no se diferencia entre tipos de usuarios. En la tabla de restaurantes hay un campo rol, pero está presente para futuras ampliaciones.

Por lo tanto, el único rol que se utiliza es el rol por defecto, ROLE_USER.

9.1.3. Rutas de la aplicación

Hay que transformar los ficheros de la aplicación del capítulo 4 en controladores. Para cada fichero (solicitado directamente por el cliente) habrá que hacer un método controlador equivalente:

- Hay que escoger una ruta para cada controlador; si recibe parámetros, hay que incluirlos en la ruta.
- Si la salida es HTML, se realiza mediante una plantilla.
- Si la salida es una redirección, hay que ajustar la ruta.

CUADRO 9.1
Rutas de la aplicación

Ruta	Descripción	Parámetros	Plantilla/redirección
/login name = 'login'	Formulario de *login*	$_POST['usuario'] $_POST['clave']	login.html.twig /categorias
/logout name = 'logout'	Cierra la sesión		/login
/categorias name = 'categorias'	Muestra la lista de categorías con vínculos a productos/{id}		categorias.html.twig
/productos/{id} name = 'productos'	Muestra los productos de la categoría, permite añadir al carro de la compra	El código de la categoría	productos.html.twig

[.../...]

Escaneado con CamScanner

CUADRO 9.1 (CONT.)

/carrito name = 'carrito'	Muestra el carro de la compra, permite quitar productos y confirmar el pedido		carrito.html.twig
/anadir name = 'anadir'	Añade productos al carrito	$_POST['cod'] $_POST['unidades']	/carrito
/eliminar name = 'eliminar'	Elimina productos del carrito	$_POST['cod'] $_POST['unidades']	/carrito
/realizarPedido name = 'realizarPedido'	Inserta el pedido en la base de datos, envía correos de confirmación y muestra mensajes de error o éxito		pedido.html.twig correo.html.twig

TOMA NOTA

Las rutas "login" y "logout" estarán asociadas con el sistema de usuarios de Symfony.

La ruta "logout" no está asociada a ningún controlador. Existe para el sistema de usuarios de Symfony. Al redirigir a un usuario a esa ruta, Symfony cerrará la sesión.

9.2. Implementación

Con todos los elementos de la aplicación bien definidos, se puede proceder a la implementación.

El primer paso es crear un nuevo proyecto:

```
composer create-project symfony/website-skeleton PedidosSymfony
```

Luego hay que modificar el fichero **.env** para incluir la configuración de la base de datos y el servidor de correo.

```
DATABASE_URL=mysql://root:@127.0.0.1:3306/pedidos
MAILER_URL=gmail://<usuario>:<clave>@localhost
```

Actividad propuesta 9.1

Configura el fichero.env para utilizar una cuenta de Gmail y poder probar el correo de confirmación de pedidos de la aplicación.

En los siguientes apartados se explicará cómo implementar las plantillas, las entidades, los controladores y la seguridad.

Escaneado con CamScanner

9.3. Plantillas

Para comenzar hay que hacer las plantillas, así se pueden ir probando los controladores.

9.3.1. *Login*

Un formulario sencillo que apunta a la ruta *login*.

```
{# templates/login.html.twig #}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Aplicación de pedidos</title>
    </head>
    <body>
        <form action="{{ path('login') }}" method="post">
            Usuario
            <input type="text" id="username" name="_username" />
            Clave
            <input type="password" id="password" name="_password" />
            <button type="submit">login</button>
        </form>
    </body>
</html>
```

9.3.2. Plantilla base

Una vez abierta sesión, todas las páginas comparten una estructura base definida por esta plantilla.

```
1   {# templates/base.html.twig #}
2   <!DOCTYPE html>
3   <html>
4       <head>
5           <meta charset="UTF-8">
6           <title>{% block title %}Aplicación de pedidos{% endblock %}</title>
7       </head>
8       <body>
9       {% block body %}
10          <header>
11          {% block header %}
12              {{ include('cabecera.html.twig') }}
13              <hr>
14          {%endblock %}
15          </header>
16          {% block contenido%}
```

Escaneado con CamScanner

```
17          {% endblock%}
18          {% endblock %}
19      </body>
20  </html>
```

Se definen varios bloques que serán sobrescritos por las otras plantillas:

- Línea 6: `title`, para el título de la página.
- Líneas 11-14: `header`, que incluye la cabecera.
- Líneas 16-17: `contenido`. Aquí es donde se sitúa el contenido específico de cada sección.

9.3.3. Cabecera

La cabecera está en la incluida en la plantilla base.

```
Usuario: {{ app.user.correo }}
<a href="{{ path('categorias') }}">Home</a>
<a href="{{ path('carrito') }}">Carrito</a>
<a href="{{ path('logout') }}">Cerrar sesión</a>
```

Incluye:

- El correo del usuario a través de app.user.
- Los vínculos para la lista de categorías, el carrito y cerrar sesión. Utiliza la función `path()` con el nombre de la ruta correspondiente.

9.3.4. Lista de categorías

Extiende la plantilla base y modifica el bloque `contenido` para mostrar la lista de categorías.

```
{# templates/categorias.html.twig #}
{% extends 'base.html.twig' %}
{% block title %}Lista de categorías{% endblock %}
{% block contenido %}
<ul>
{% for cat in categorias %}
    <li>
        <a href="{{path('productos',{'id':cat.CodCat})}}">
            {{cat.Nombre}}</a>
    </li>
    {% endfor %}
</ul>
{% endblock %}
```

Esta plantilla recibe como parámetro un *array* con las categorías. Como en las versiones anteriores, los elementos de la lista son vínculos cuyo texto es el nombre de la categoría y que apuntan a la tabla de productos de esa categoría; en este caso, a la ruta `productos/{id}`.

Escaneado con CamScanner

9.3.5. Tabla de productos

Muestra la tabla de productos de una categoría, incluyendo los formularios para añadir. Son como los del capítulo 4, solo cambia el valor de `action`. Ahora, en lugar de `anadir.php` se utiliza el nombre de la ruta correspondiente, `anadir`. Recibe como argumento un *array* de productos.

```
{# templates/productos.html.twig #}
{% extends 'base.html.twig' %}
{% block title %}Tabla de productos{% endblock %}
{% block contenido %}
<table>
    <tr><th>Nombre</th><th>Descripción</th><th>Stock</th><th>Peso</th><-
    th>Comprar</th></tr>
    {% for prod in productos %}
    <tr>
        <td>{{ prod.nombre }}</td>
        <td>{{ prod.descripcion }}</td>
        <td>{{ prod.stock }}</td>
        <td>{{ prod.peso }}</td>
        <td>
        <form action = {{path('anadir')}} method = 'POST'>
            <input name='unidades' type='number' min ='1' value='1'>
            <input type = 'submit' value='Comprar'>
            <input name = 'cod' type= 'hidden' value = {{prod.codProd}}>
        </form>
        </td>
    </tr>
    {% endfor %}
</table>
{% endblock %}
```

9.3.6. El carrito de la compra

Muestra una tabla con los datos del carrito de la compra. Recibe un *array* con los datos de los productos y las unidades pedidas. Incluye los formularios para eliminar productos, que también son como los del capítulo 4. En este caso la ruta es `eliminar`.

```
{# templates/carrito.html.twig #}
{% extends 'base.html.twig' %}
{% block title %}Carrito de la compra{% endblock %}
{% block contenido %}
{% if productos is empty %}
<p> El carrito está vacío </p>
{% else %}
<table>
    <tr>
        <th>Nombre</th><th>Descripción</th>
        <th>Stock</th><th>Peso</th><th>Unidades</th><th>Eliminar</th>
    </tr>
    {% for prod in productos %}
```


Escaneado con CamScanner

```
    <tr>
        <td>{{ prod.nombre }}</td>
        <td>{{ prod.descripcion }}</td>
        <td>{{ prod.stock }}</td>
        <td>{{ prod.peso }}</td>
        <td>{{ prod.unidades }}</td>
        <td>
            <form action={{path('eliminar')}} method = 'POST'>
                <input name='unidades' type='number' min ='1' value= '1'>
                <input type = 'submit' value='Eliminar'>
                <input name = 'cod' type='hidden'
                value = {{prod.codProd}}>
            </form>
        </td>
    </tr>
    {% endfor %}
</table>
<a href = {{ path('realizarPedido') }}>Realizar Pedido</a>
{% endif %}
{% endblock %}
```

9.3.7. Confirmación del pedido

Este controlador es muy sencillo, muestra mensajes de error o confirmación. Recibe un código de error y el código del pedido.

```
{# templates/pedido.html.twig #}
{% extends 'base.html.twig' %}
{% block title %}Confirmación de pedido{% endblock %}
{% block contenido %}
    {% if error == 1 %}
        No hay productos en el carrito
    {% elseif error == 2%}
        Error de la base de datos al procesar el pedido,
    consulte con el administrador
    {% else %}
        Pedido {{ id }} realizado con éxito
    {% endif %}
{% endblock %}
```

9.3.8. Correo

Esta plantilla se utiliza para crear el correo. Es una tabla con los datos de los productos del pedido y las unidades. Es como la del carrito de la compra, pero sin formularios.

```
{# templates/correo.html.twig #}
Pedido {{ id }} realizado con éxito
```


Escaneado con CamScanner

```
<table>
    <tr><th>Nombre</th><th>Descripción</th>
        <th>Peso</th><th>Unidades</th>
    </tr>
    {% for prod in productos %}
    <tr>
        <td>{{ prod.nombre }}</td>
        <td>{{ prod.descripcion }}</td>
        <td>{{ prod.peso }}</td>
        <td>{{ prod.unidades }}</td>
    </tr>
    {% endfor %}
</table>
```

Actividad propuesta 9.2



Las plantillas de la tabla de productos, el carrito y el correo tienen muchos elementos en común. ¿Crees que sería buena idea reescribirlas usando herencia e inclusión? Analiza las ventajas e inconvenientes.

9.4. Entidades

Habrá una entidad por cada tabla. La entidad Restaurante representa a los usuarios de la aplicación. Se utiliza la entidad desarrollada en el capítulo anterior.

Se emplea la entidad Producto para la tabla de productos, asociada con Categoria.

```
<?php
// src/Entity/Producto.php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;
/**
* @ORM\Entity @ORM\Table(name="productos")
*/
class Producto {
    /**
    * @ORM\Id
    * @ORM\GeneratedValue
    * @ORM\Column(type="integer", name="CodProd")
    */
    private $codProd;
    /**
    * @ORM\Column(type="string")
    */
```

Escaneado con CamScanner

```
 private $nombre;
 /**
 * @ORM\Column(type="string")
 */
 private $descripcion;
 /**
 * @ORM\Column(type="float")
 */
 private $peso;
 /**
 * @ORM\Column(type="integer")
 */
 private $stock;
 /**
 * @ORM\ManyToOne(targetEntity="Categoria", inversedBy = "productos")
 * @ORM\JoinColumn(name="Categoria", referencedColumnName="CodCat")
 */
 private $categoria;
 public function getCodProd() {
     return $this->codProd;
 }
 public function setCodProd($codProd) {
     $this->codProd = $codProd;
 }
 public function getNombre() {
     return $this->nombre;
 }
 public function setNombre($nombre) {
     $this->nombre = $nombre;
 }
 public function getDescripcion() {
     return $this->descripcion;
 }
 public function setDescripcion($descripcion) {
     $this->descripcion = $descripcion;
 }
 public function getPeso() {
     return $this->peso;
 }
 public function setPeso($peso) {
     $this->peso = $peso;
 }
 public function getStock(){
     return $this->stock;
 }
 public function setStock($stock) {
     $this->stock = $stock;
 }
```


Escaneado con CamScanner

```
    public function getCategoria() {
        return $this->categoria;
    }
    public function setCategoria($categoria) {
        $this->categoria = $categoria;
    }
}
```

La asociación con Categoria es bidireccional. La clase Categoria tiene un campo productos en el que se cargan los productos de la categoría. Esto lo hace Doctrine cuando se accede al campo.

```
<?php
// src/Entity/Categoria.php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;
/**
* @ORM\Entity @ORM\Table(name="categorias")
*/
class Categoria {
    /**
    * @ORM\Id
    * @ORM\GeneratedValue
    * @ORM\Column(type="integer",name="CodCat")
    */
    private $codCat;
    /**
    * @ORM\Column(type="string")
    */
    private $nombre;
    /**
    * @ORM\Column(type="string")
    */
    private $descripcion;
    /**
    * Bidireccional
    * @ORM\OneToMany(targetEntity="Producto", mappedBy="categoria")
    */
    private $productos;
    public function getCodCat() {
        return $this->codCat;
    }
    public function getProductos() {
        return $this->productos;
    }
    public function getNombre() {
        return $this->nombre;
    }
```

Escaneado con CamScanner

```
    public function setNombre($nombre) {
        $this->nombre = $nombre;
    }
    public function getDescripcion() {
        return $this->descripcion;
    }
    public function setDescripcion($descripcion) {
        $this->descripcion = $descripcion;
    }
}
```

Se emplea la entidad Pedido para la tabla de pedidos, asociada con Restaurante.

```
<?php
// src/Entity/Pedido.php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
* @ORM\Entity @ORM\Table(name="pedidos")
*/
class Pedido
{
    /**
    * @ORM\Id
    * @ORM\GeneratedValue
    * @ORM\Column(type="integer", name="CodPed")
    */
    private $codPed;
    /**
    * @ORM\Column(type="datetime", name = "Fecha")
    */
    private $fecha;
    /**
    * @ORM\Column(type="integer", name = "Enviado")
    */
    private $enviado;
    /**
    * @ORM\ManyToOne(targetEntity="Restaurante")
    * @ORM\JoinColumn(name="Restaurante", referencedColumnName="CodRes")
    */
    private $restaurante;
    public function getCodPed() {
        return $this->codPed;
    }
    public function getFecha() {
        return $this->fecha;
    }
    public function setFecha($fecha) {
```


Escaneado con CamScanner

```
        $this->fecha = $fecha;
    }
    public function getEnviado() {
        return $this->enviado;
    }
    public function setEnviado($enviado) {
        $this->enviado = $enviado;
    }
    public function getRestaurante() {
        return $this->restaurante;
    }
    public function setRestaurante($restaurante) {
        $this->restaurante = $restaurante;
    }
}
```

Y finalmente, se utiliza la entidad PedidosProducto, que tiene dos asociaciones: con Pedido y con Productos.

```
<?php
// src/Entity/PedidoProducto.php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;
/**
* @ORM\Entity @ORM\Table(name="pedidosproductos")
*/
class PedidoProducto{
    /**
    * @ORM\Id
    * @ORM\GeneratedValue
    * @ORM\Column(type="integer", name="CodPedProd")
    */
    private $codPedProd;
    /**
    * @ORM\ManyToOne(targetEntity="Pedido")
    * @ORM\JoinColumn(name="Pedido", referencedColumnName="CodPed")
    */
    private $codPed;
    /**
    * @ORM\ManyToOne(targetEntity="Producto")
    * @ORM\JoinColumn(name="Producto", referencedColumnName="CodProd")
    */
    private $codProd;
    /**
    * @ORM\Column(type="integer", name = "unidades")
    */
    private $unidades;
```

Escaneado con CamScanner

```
    public function getCodPedProd() {
        return $this->codPedProd;
    }
    public function getCodPed() {
        return $this->codPed;
    }
    public function setCodPed($codPed) {
        $this->codPed = $codPed;
    }
    public function getCodProd() {
        return $this->codProd;
    }
    public function setCodProd($codProd) {
        $this->codProd = $codProd;
    }
    public function getUnidades() {
        return $this->unidades;
    }
    public function setUnidades($unidades) {
        $this->unidades = $unidades;
    }
}
```

9.5. Controladores

El controlador de *login* está en la clase `PedidosLogin`; el resto, en `PedidosBase`. Esta clase tendrá una anotación de seguridad para que solo puedan acceder a sus controladores los usuarios que hayan abierto sesión.

9.5.1. Abrir sesión

Este controlador estará asociado al sistema de usuarios de Symfony. Por tanto, no tiene que ocuparse de las comprobaciones, solo de mostrar el formulario.

```
class PedidosLogin extends AbstractController{
    /**
    * @Route("/login", name="login")
    */
    public function login(){
        return $this->render('login.html.twig');
    }
}
```

9.5.2. Lista de categorías

Carga los datos de la tabla de categorías con el método `findAll()` y pasa el resultado a la plantilla.

Escaneado con CamScanner

```
/**
* @Route("/categorias", name="categorias")
*/
public function mostrarCategorias() {
    $categorias = $this->getDoctrine()
        ->getRepository(Categoria::class)
        ->findAll();
    return $this->render("categorias.html.twig",
        array('categorias'=>$categorias));
}
```

9.5.3. Tabla de productos

Este controlador es el que devuelve la tabla de productos. Recibe el código de la categoría como parámetro. Busca la categoría con el método `find()` y accede a sus productos con el método `getProductos()`, que utiliza la asociación bidireccional. Pasa los productos a la plantilla.

```
/**
* @Route("/productos/{id}", name="productos")
*/
public function mostrarProductos($id) {
    $productos = $this->getDoctrine()
    ->getRepository(Categoria::class)
    ->find($id)
    ->getProductos();
    if (!$productos) {
        throw $this->createNotFoundException('Categoría no encontrada');
    }
    return $this->render("productos.html.twig", array('productos'=>
       $productos));
}
```

9.5.4. Carrito

A partir de la variable de sesión de carrito, monta un *array* con los datos que hay que mostrar y se los pasa a la plantilla. Para cada producto, busca con `find()` todos sus datos y añade las unidades pedidas.

```
/**
* @Route("/carrito", name="carrito")
*/
public function mostrarCarrito(SessionInterface $session){
    /* para cada elemento del carrito se consulta la base de datos y se
    recuperan sus datos*/
    $productos = [];
    $carrito = $session->get('carrito');
```

Escaneado con CamScanner

```
    /* si el carrito no existe se crea como un array vacío*/
    if(is_null($carrito)){
        $carrito = array();
        $session->set('carrito', $carrito);
    }
    /* se crea array con todos los datos de los productos y la canti-
    dad*/
    foreach ($carrito as $codigo => $cantidad){
        $producto = $this->getDoctrine()
            ->getRepository(Producto::class)
            ->find((int)$codigo);
        $elem = [];
        $elem['codProd'] = $producto->getCodProd();
        $elem['nombre'] = $producto->getNombre();
        $elem['peso'] = $producto->getPeso();
        $elem['stock'] = $producto->getStock();
        $elem['descripcion'] = $producto->getDescripcion();
        $elem['unidades'] = implode($cantidad);
        $productos[] = $elem;
    }
    return $this->render("carrito.html.twig",
    array('productos'=>$productos));
}
```

9.5.5. Añadir y eliminar

Estos controladores son muy parecidos a los de la aplicación original. Solo cambia que acceden a la sesión con el objeto de Symfony. Modifican la variable de sesión y redirigen a carrito.

```
/**
* @Route("/anadir", name="anadir")
*/
public function anadir(SessionInterface $session) {
    $id = $_POST['cod'];
    $unidades= $_POST['unidades'];
    $carrito = $session->get('carrito');
    if(is_null($carrito)){
        $carrito = array();
    }
    if(isset($carrito[$id])){
        $carrito[$id]['unidades'] += intval($unidades);
    }else{
        $carrito[$id]['unidades'] = intval($unidades);
    }
    $session->set('carrito', $carrito);
    return $this->redirectToRoute('carrito');
}
/**
```


Escaneado con CamScanner

```
* @Route("/eliminar", name="eliminar")
*/
public function eliminar(SessionInterface $session){
    $id = $_POST['cod'];
    $unidades= $_POST['unidades'];
    $carrito = $session->get('carrito');
    if(is_null($carrito)){
        $carrito = array();
    }
    if(isset($carrito[$id])){
        $carrito[$id]['unidades'] -= intval($unidades);
        if($carrito[$id]['unidades'] <= 0) {
            unset($carrito[$id]);
        }
    }
    $session->set('carrito', $carrito);
    return $this->redirectToRoute('carrito');
}
```

Actividad propuesta 9.3

¿Qué cambios habría que hacer para que el controlador de añadir verifique que no se incluyan más unidades de las disponibles para ningún producto?

9.5.6. Realizar pedido

Este es el controlador más complicado. Modifica la base de datos y envía los correos. Para enviar los correos utiliza la plantilla `correo.twig.html`; para mostrar si ha habido error o confirmar el pedido, `pedido.html.twig`.

```
/**
* @Route("/realizarPedido", name="realizarPedido")
*
*/
public function realizarPedido(SessionInterface $session, \Swift_Mailer
$mailer) {
    $entityManager = $this->getDoctrine()->getManager();
    $carrito = $session->get('carrito');
    /* si el carrito no existe, o está vacío*/
    if(is_null($carrito) ||count($carrito)==0){
        return $this->render("pedido.html.twig",
        array('error'=>1));
    }else{
        #crear un nuevo pedido
```


Escaneado con CamScanner

```
    $pedido = new Pedido();
    $pedido->setFecha(new \DateTime());
    $pedido->setEnviado(0);
    $pedido->setRestaurante($this->getUser());
    $entityManager->persist($pedido);
    #recorrer carrito creando nuevos pedidoproducto
    foreach ($carrito as $codigo => $cantidad){
        $producto = $this->getDoctrine()
            ->getRepository(Producto::class)
            ->find($codigo);
        $fila = new PedidoProducto();
        $fila->setCodProd($producto);
        $fila->setUnidades( implode($cantidad));
        $fila->setCodPed($pedido);
        //actualizar el stock
        $cantidad = implode($cantidad);
        $query = $entityManager->createQuery(
            "UPDATE App\Entity\Producto p
            SET p.stock = p.stock - $cantidad
            WHERE p.codProd = $codigo");
        $resul = $query->getResult();
        $entityManager->persist($fila);
    }
}
/*si hay error con la BD,
Muestra plantilla con el código adecuado*/
try{
    $entityManager->flush();
}catch (Exception $e) {
    return $this->render("pedido.html.twig",
    array( 'error'=>2));
}
/*prepara el array de productos para la plantilla*/
foreach ($carrito as $codigo => $cantidad){
    $producto = $this->getDoctrine()
        ->getRepository(Producto::class)
        ->find((int)$codigo);
    $elem = [];
    $elem['codProd'] = $producto->getCodProd();
    $elem['nombre'] = $producto->getNombre();
    $elem['peso'] = $producto->getPeso();
    $elem['stock'] = $producto->getStock();
    $elem['descripcion'] = $producto->getDescripcion();
    $elem['unidades'] = implode($cantidad);
    $productos[] = $elem;
}
//vaciar el carrito
$session->set('carrito', array());
```

Escaneado con CamScanner

```
        /* mandar el correo */
        $message = (new \Swift_Message())
            ->setFrom('noreply@empresafalsa.com', 'Sistema de pedidos')
            ->setTo($this->getUser()->getCorreo())
            ->setSubject("Pedido ". $pedido->getCodPed(). "confirmado")
            ->setBody($this->renderView('correo.html.twig',
                array('id'=>$pedido->getCodPed(),
                'productos'=> $productos)),
                'text/html');
        $mailer->send($message);
        return $this->render("pedido.html.twig",
            array('error'=>0,'id'=>$pedido->getCodPed(),
            'productos'=> $productos));
    }
```

Actividad propuesta 9.4



¿Qué cambios habría que hacer para que el controlador de realizar pedido verifique que no se incluyan más unidades de las disponibles para ningún producto? ¿Crees que es necesario hacerlo si ya se controló al añadir los productos al carrito?

9.6. Seguridad

Hay que modificar el fichero `/config/packages/security.yaml` para asociar el sistema de usuarios de Symfony con la entidad Restaurante y especificar las rutas de *login* y *logout*.

```
security:
    providers:
        pedidos:
        entity:
            class: App\Entity\Restaurante
            property: correo
    encoders:
        App\Entity\Restaurante:
        algorithm: plaintext
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: true
            form_login:
                login_path: login
                check_path: login
                default_target_path: categorias
```

Escaneado con CamScanner

```
            provider: pedidos
        logout:
            path: /logout
            target: /login
```

También hay que crear la ruta para "logout" en el fichero `/config/routes.yaml`:

```
logout:
  path: /logout
```

En lugar de utilizar la sección `access_control`, la clase `PedidosBase` está protegida con la anotación:

```
/**
* @Security("has_role('ROLE_USER')")
*/
class PedidosBase extends AbstractController{
```

De esta manera, todos los controladores de la clase están restringidos a los usuarios que hayan abierto sesión.



Actividad propuesta 9.5

¿Cómo se utilizaría la sección access_control para controlar el acceso?

Resumen

- El diseño de la base de datos y la estructura para el carrito de la compra se mantienen igual que en las versiones anteriores.
- Las funciones de acceso a la base de datos y envío de correo cambian para adaptarse al *framework*.
- La base de datos se maneja a través de Doctrine. Hay una entidad por cada tabla.
- Para la salida, los controladores utilizan plantillas Twig.
- La estructura de la página se implementa a través de una jerarquía de plantillas.
- La aplicación utiliza el sistema de usuarios de Symfony. La entidad Restaurante tiene que implementar UserInterface.
- La asociación entre Productos y Categorias es bidireccional, así no hace falta hacer una consulta para encontrar los productos de una categoría.
- El resto de las asociaciones son unidireccionales.
- Los ficheros de la aplicación original se sustituyen por controladores.
- El control de acceso se realiza con una anotación en la clase PedidosBase, que incluye todos los controladores menos los de *login* y *logout*.

Escaneado con CamScanner

Ejercicios propuestos

1. Al añadir un producto, la aplicación redirige al carrito. Modifícala para que redirija a la tabla de productos, con la misma categoría.
2. Modifica la tabla de productos para que no muestre los productos sin *stock*.
3. Añade formularios para que los administradores (usuarios con rol 1) puedan insertar nuevas categorías y productos. Protege el acceso a ellos.
4. Modifica la cabecera para que muestre a los administradores vínculos a los formularios del ejercicio 3.
5. Modifica las tablas de productos y carrito para utilizar el componente de formularios de Symfony.
6. En este capítulo se utilizan plantillas HTML, pero también podría haberse utilizado JSON, como en el capítulo 6. ¿Qué modificaciones habría que hacer para utilizar AJAX? Rediseña la aplicación intentando reutilizar el código del cliente del capítulo 6.

ACTIVIDADES DE AUTOEVALUACIÓN

1. La asociación entre Producto y Categoria es:
   - ☐ *a)* No hay.
   - ☐ *b)* Unidireccional.
   - ☐ *c)* Bidireccional.

2. Sobre la plantilla cabecera.html.twig se puede decir que:
   - ☐ *a)* Hereda de base.html.twig.
   - ☐ *b)* base.html.twig hereda de ella.
   - ☐ *c)* Ninguna de las anteriores.

3. La entidad que representa a los usuarios de la aplicación es:
   - ☐ *a)* Usuario.
   - ☐ *b)* Restaurante.
   - ☐ *c)* Pedido.

4. El control de acceso se realiza con:
   - ☐ *a)* La sección access_control del fichero security.yaml.
   - ☐ *b)* Una anotación en la clase PedidosBase.
   - ☐ *c)* Una anotación en la clase PedidosLogin.

5. La configuración para la base de datos se ha introducido en:
   - ☐ *a)* El fichero security.yaml.
   - ☐ *b)* El fichero .env.
   - ☐ *c)* No hay base de datos.

Escaneado con CamScanner

6. La ruta *eliminar* redirige a:
   - ☐ *a)* carrito.
   - ☐ *b)* categorias.
   - ☐ *c)* cerrarserion.

7. Para el envío de correo:
   - ☐ *a)* Se reutilizan las funciones de la aplicación original.
   - ☐ *b)* Se utiliza el componente de Symfony.
   - ☐ *c)* No está configurado.

8. Para manejar la base de datos:
   - ☐ *a)* Se utiliza Doctrine.
   - ☐ *b)* Se reutilizan las funciones de la aplicación original.
   - ☐ *c)* Se reutilizan las funciones de la aplicación original con pequeñas modificaciones.

9. Los formularios de añadir y eliminar productos:
   - ☐ *a)* Se envían con JavaScript.
   - ☐ *b)* Usan el componente de Symfony.
   - ☐ *c)* Son formularios HTML normales.

10. En la aplicación se utilizan los roles:
    - ☐ *a)* ROLE_USER y ROLE_ADMIN.
    - ☐ *b)* ROLE_USER.
    - ☐ *c)* No se utilizan roles.

SOLUCIONES:

1.	a b **c**	**5.**	a **b** c
2.	a b **c**	**6.**	**a** b c
3.	a **b** c	**7.**	a **b** c
4.	a **b** c	**8.**	**a** b c
9.	a b **c**		
10.	a **b** c		

Escaneado con CamScanner

10

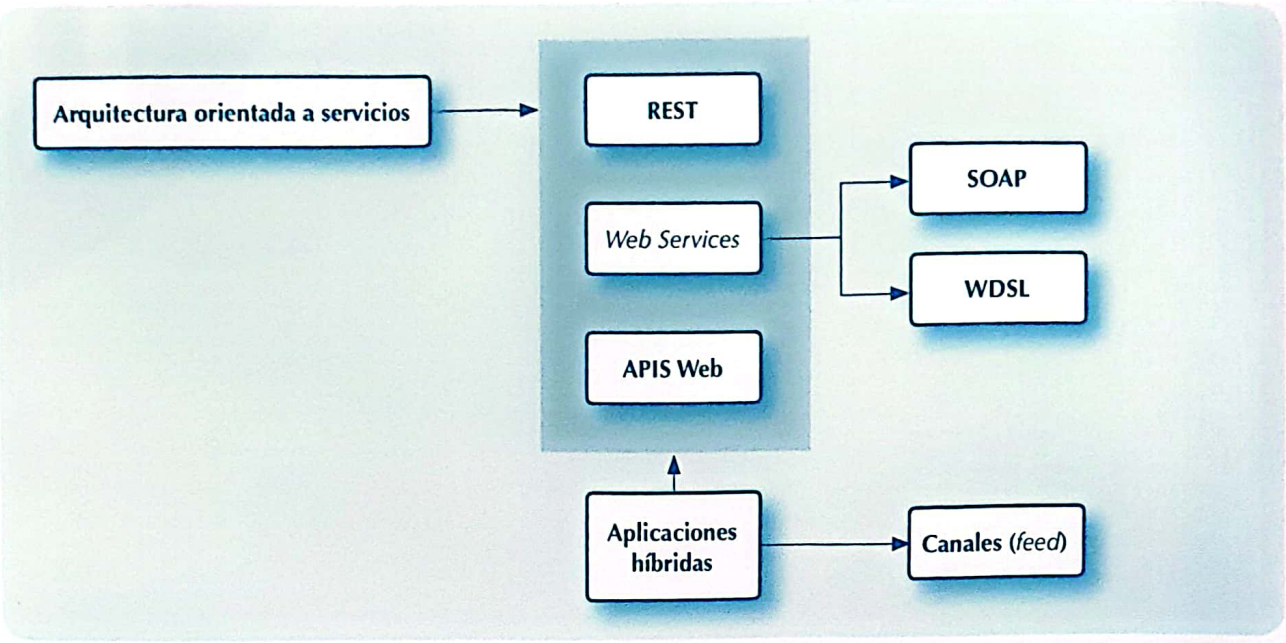
Servicios web y aplicaciones híbridas

Objetivos

- ✓ Presentar la arquitectura orientada a servicios.
- ✓ Conocer los elementos del protocolo SOAP.
- ✓ Entender y crear documentos WDSL.
- ✓ Crear servicios web.
- ✓ Utilizar servicios web.
- ✓ Comprender el funcionamiento de las aplicaciones híbridas.
- ✓ Desarrollar una aplicación híbrida.

Escaneado con CamScanner

Mapa conceptual



Glosario

Aplicación híbrida o *mashup*. Las aplicaciones híbridas o *mashups* integran datos y funcionalidad de diferente origen en una misma interfaz gráfica.

***Endpoint*.** Los servicios web exponen uno o más *endpoints* a los que se pueden enviar solicitudes.

Fuente o canal web. Las fuentes o canales web (*feed*) son un medio de sindicación de contenido al que los usuarios pueden suscribirse.

Servicio. Los servicios son componentes independientes que proveen alguna funcionalidad a otros componentes o sistemas.

Servicios web. Puede referirse a un servicio genérico accesible por la web o a arquitectura *Web Services*, servicios web, un estándar del W3C. Para diferenciar, en este capítulo el segundo caso se escribirá con mayúscula.

Sindicación. Consiste en publicar contenidos para que puedan ser utilizados en otras webs.

SOA. *Service Oriented Architecture*. En la arquitectura orientada a servicios se utilizan varios servicios en conjunto para obtener la funcionalidad de la aplicación

SOAP. *Simple Object Access Protocol*. Es un protocolo para intercambio de mensajes entre clientes y proveedores de servicios. Es un estándar del W3C.

Web API. Conjunto de *endpoints* accesible a través de la web. Proveen funcionalidad que se puede integrar en otras aplicaciones.

WSDL. *Web Service Description Language*. Lenguaje basado en XML para describir la funcionalidad ofrecida por un Servicio Web. Es un estándar del W3C.

Escaneado con CamScanner

10.1. Arquitecturas de programación orientadas a servicios

Un servicio es un componente que ofrece alguna funcionalidad. Por ejemplo, la previsión meteorológica o el tipo de cambio de una moneda. Los servicios son elementos independientes y accesibles de forma remota mediante una interfaz. Para cumplir su función, un servicio puede utilizar otros servicios.

Los servicios reciben peticiones y envían una respuesta. Un servicio puede exponer varios *endpoints*, cada uno con una funcionalidad. Por ejemplo, un servicio puede exponer un *endpoint* para obtener el tipo de cambio del euro respecto al dólar y otro *endpoint* para el tipo inverso.

En las arquitecturas orientadas a servicios, las aplicaciones se diseñan utilizando estos componentes. Con esta arquitectura modular las aplicaciones son más flexibles y se favorece la reutilización del código. Además, permite integrar componentes de diferente procedencia, de diferentes sistemas y en diferentes lenguajes.

Dentro del desarrollo web, el modelo SOA se puede implementar de varias maneras. Entre ellas:

- Arquitectura *Web Services*. Es un estándar del W3C. Los servicios tienen una interfaz en WSDL y otros sistemas realizan solicitudes al servicio utilizando SOAP.
- REST. *Representational State Transfer*. La arquitectura REST utiliza una interfaz uniforme que se apoya en los métodos HTTP.
- API Web. Conjunto de *endpoints* accesible a través de la web. Proveen funcionalidad que se puede integrar en otras aplicaciones. Un ejemplo típico es el API de Google Maps, para integrar los mapas de Google en otras páginas. Pueden ser del lado del cliente o del servidor.

10.2. Protocolos y lenguajes implicados. SOAP

El estándar *Web Services* se basa en una serie de protocolos:

- Para los mensajes se utiliza el SOAP, basado en XML.
- Para la descripción de los servicios, WSDL, también basado en XML.
- Para el descubrimiento de servicios, es decir, para que los clientes puedan encontrarlos, UDDI, también basado en XML. Su uso no está muy extendido.
- La comunicación entre los servicios y sus clientes se puede establecer por medio de varios protocolos. Algunos de los más habituales son HTTP, SMTP y FTP.

CUADRO 10.1
Pila de protocolos para *Web Services*

Protocolos	Función
UDDI	Descubrimiento de servicios
WSDL	Descripción de servicios
SOAP	Mensajes
HTTP, FTP, SMTP...	Comunicaciones

Escaneado con CamScanner

10.2.1. SOAP

SOAP define el formato de los mensajes que se intercambian entre el cliente y el servicio web. Un mensaje SOAP es un fichero XML con la siguiente estructura:

```
<Envelope>
    <Header>
        ...
    </Header>
    <Body>
        ...
    </Body>
    <Fault>
        ...
    </Fault>
</Envelope>
```

Los elementos `header` y `fault` son opcionales.
Por ejemplo, un mensaje de petición podría ser así:

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:ns1="http://localhost/cap10/servidorsoapmate.php">
    <env:Header/>
    <env:Body>
        <ns1:potencia>
            <base>2</base>
            <exponente>3</exponente>
        </ns1:potencia>
    </env:Body>
</env:Envelope>
```

Dentro del elemento `Body` hay un elemento potencia, que contiene elementos `base` y `exponente`. Con este mensaje se solicita el *endpoint* `potencia`, y se pasan los parámetros `base` y `exponente` con los valores indicados.

10.2.2. Descripción de servicios web. WSDL

EL WDSL es un lenguaje basado en XML para describir la funcionalidad ofrecida por un servicio web. Aunque el formato es bastante descriptivo, también es complejo. Afortunadamente, los programadores, en la mayoría de los casos, no tienen que ocuparse de los detalles.

En WSDL un servicio es un conjunto de *endpoints*. En WSDL 1 para los *endpoints* se usa el elemento `port`. En WSDL 2, se sustituyó por `endpoint`.

Los elementos de un fichero WSDL son:

a) `types`: define los tipos de datos que se utilizan en los mensajes utilizando XSD.
b) `message`: define el formato de los mensajes de solicitud y respuesta.

Escaneado con CamScanner

`binding`: especifica los detalles del protocolo SOAP.
`service`: define los *endpoints* expuestos. Contiene los elementos `port`, cada uno es un *endpoint* de la aplicación. De hecho, en WSDL 2.0, se llaman *endpoint*.
`portType`: Detalla los mensajes de solicitud y respuesta de cada `port` (*endpoint*).

mejor verlo con un ejemplo. La siguiente imagen muestra las secciones principales de ero WSDL. Este documento describe un servicio que expone un *endpoint* para calcular ias. Recibe dos argumentos, la base y el exponente, y devuelve el resultado de elevar la exponente.

```
itions name="Mate" targetNamespace="http://localhost/cap10/servidorsoapmate.php">
pes></types>
rtType name="MatePort"></portType>
nding name="MateBinding" type="tns:MatePort"></binding>
rvice name="MateService"></service>
essage name="potenciaIn"></message>
essage name="potenciaOut"></message>
finitions>
```

Figura 10.1
Secciones
de un fichero WSDL.

Con `types` se definen los tipos de datos que se utilizan en los mensajes utilizando XSD. En ejemplo se definen el elemento `potencia`, formado por dos elementos de tipo `float`, `base` ponente, y el elemento `potenciaResponse`, que solo tiene un elemento `float`. El primero tipo de dato de los mensajes de petición; el segundo, el de los mensajes de respuesta.

```
<types>
    <xsd:schema targetNamespace="http://localhost/cap10/matewsdl.php">
        <xsd:element name="potencia">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="base" type="xsd:float"/>
                    <xsd:element name="exponente" type="xsd:float"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="potenciaResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="potenciaResult" type="xsd:float"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:schema>
</types>
```

Los elementos `message` especifican el formato de los mensajes de solicitud y respuesta. El mensaje `PotenciaIn`, el que se usa para solicitar el servicio, incluye un elemento potencia.

```
<message name="potenciaIn">
    <part name="parameters" element="tns:potencia"/>
</message>
```

Escaneado con CamScanner

Con portType se describen los mensajes de solicitud y respuesta de cada *endpoint*. El portType `MatePort` recibe un mensaje de tipo `potenciaIn` y devuelve un mensaje de tipo `potenciaOut` (definidos como elementos `message`).

```
<portType name="MatePort">
    <operation name="potencia">
        <documentation>/**</documentation>
        <input message="tns:potenciaIn"/>
        <output message="tns:potenciaOut"/>
    </operation>
</portType>
```

El elemento `binding` especifica los detalles del protocolo SOAP.

```
<binding name="MateBinding" type="tns:MatePort">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="potencia">
        <soap:operation soapAction="http://localhost/cap10/matewsdl.
        php#potencia"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
```

En el elemento `service` se definen los *endpoints* expuestos a través de elementos `port`. Cada elemento `port` es un *endpoint* de la aplicación. De hecho, en WSDL 2.0 se llaman *endpoint*. En el ejemplo, el elemento `port` utiliza `name="MatePort"`. Esto quiere decir que es el de tipo `MatePort`, definido con el elemento `portType`.

```
<service name="MateService">
    <port name="MatePort" binding="tns:MateBinding">
        <soap:address location="http://localhost/cap10/matewsdl.php"/>
    </port>
</service>
```

Recurso web

Puedes ver el documento completo accediendo a http://localhost/cap10/matewsdl.php.

Escaneado con CamScanner

10.3. Librerías de PHP para servicios web

La extensión SOAP de PHP permite crear clientes y servidores SOAP. La librería Zend\SOAP está desarrollada a partir de esta extensión y añade clases útiles para el desarrollo. Entre otras cosas, para generar automáticamente ficheros WDSL.

Para activar la extensión SOAP hay que "descomentar" la línea

```
extension=soap
```

del fichero **php.ini** y reiniciar el servidor para que la cargue.

La librería Zend\Soap se instala utilizando `composer`:

```
composer require zendframework/zend-soap
```

10.3.1. Generación de servicios web

Para crear un servicio web se utiliza un objeto de la clase Zend\Soap\Server. Este objeto se asocia con una clase cuyos métodos serán los métodos expuestos por el servicio. Por ejemplo, para un servicio con operaciones matemáticas se podría usar la siguiente clase:

```
class Mate{
    /**
    * @param float $base
    * @param float $exponente
    * @return float $resul
    */
    public function potencia($base, $exponente)
    {
        return pow($base, $exponente);
    }
}
```

El primer paso es generar un fichero WSDL con la descripción del servicio. Con la clase Zend\Soap\AutoDiscover se puede generar automáticamente mediante introspección del código, lo que ahorra la tarea al programador. Para que funcione bien es importante anotar los métodos con bloques DocBlock indicando al menos nombre y tipo de datos de los argumentos y tipo de dato devuelto, como se puede ver en la clase `Mate`.

El siguiente ejemplo genera el WSDL necesario para exponer el método de la clase `Mate`:

```
<?php
require_once __DIR__. '/vendor/autoload.php';
require "Mate.php";
$serverUrl = "http://localhost/cap10/matewsdl.php";
$soapAutoDiscover = new \Zend\Soap\AutoDiscover(new \Zend\Soap\Wsdl\
ComplexTypeStrategy\ArrayOfTypeSequence());
$soapAutoDiscover->setBindingStyle(array('style' => 'document'));
$soapAutoDiscover->setOperationBodyStyle(array('use' => 'literal'));
$soapAutoDiscover->setClass('Mate');
```


Escaneado con CamScanner

```
$soapAutoDiscover->setUri($serverUrl);
header("Content-Type: text/xml");
echo $soapAutoDiscover->generate()->toXml();
```

Si se accede desde el navegador, se obtiene:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://localhost/cap10/matewsdl.php"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    name="Mate"
    targetNamespace="http://localhost/cap10/matewsdl.php">
    <types>
        <xsd:schema targetNamespace="http://localhost/cap10/matewsdl.php">
            <xsd:element name="potencia">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="base" type="xsd:float"/>
                        <xsd:element name="exponente" type="xsd:float"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="potenciaResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="potenciaResult" type="xsd:float"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </types>
    <portType name="MatePort">
        <operation name="potencia">
            <documentation>/**</documentation>
            <input message="tns:potenciaIn"/>
            <output message="tns:potenciaOut"/>
        </operation>
    </portType>
    <binding name="MateBinding" type="tns:MatePort">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="potencia">
                <soap:operation
                    soapAction="http://localhost/cap10/matewsdl.php#poten-
                    cia"/>
```

Escaneado con CamScanner

```
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="MateService">
        <port name="MatePort" binding="tns:MateBinding">
        <soap:address location="http://localhost/cap10/matewsdl.php"/>
    </port>
    </service>
    <message name="potenciaIn">
        <part name="parameters" element="tns:potencia"/>
    </message>
    <message name="potenciaOut">
        <part name="parameters" element="tns:potenciaResponse"/>
</message>
</definitions>
```

Como se puede observar, utiliza los métodos y anotaciones de la clase para crear los tipos de datos, mensajes y *bindings*.

Partiendo de esta base, el siguiente fichero se encarga tanto de generar el fichero WSDL como de crear el servidor y pasarle las peticiones. Si se accede al ejemplo añadiendo el parámetro `wsdl` a la ruta (líneas 5-14), se devuelve el WSDL, como en el ejemplo anterior. Si no, crea el servidor SOAP y le pasa la petición.

```
1  <?php
2  require_once __DIR__. '/vendor/autoload.php';
3  require "Mate.php";
4  $serverUrl = "http://localhost/cap10/servidorsoapmate.php";
5  if (isset($_GET['wsdl'])) {
6      $soapAutoDiscover = new \Zend\Soap\AutoDiscover
7      (new \Zend\Soap\Wsdl\ComplexTypeStrategy\ArrayOfTypeSequence());
8      $soapAutoDiscover->setBindingStyle(array('style' => 'document'));
9      $soapAutoDiscover->setOperationBodyStyle(array('use' => 'literal'));
10     $soapAutoDiscover->setClass('Mate');
11     $soapAutoDiscover->setUri($serverUrl);
12     header("Content-Type: text/xml");
13     echo $soapAutoDiscover->generate()->toXml();
14 } else {
15     $soap = new \Zend\Soap\Server($serverUrl. '?wsdl');
16     $soap->setObject(new \Zend\Soap\Server\DocumentLiteralWrapper
           (new Mate()));
17     $soap->handle();
18 }
```

Escaneado con CamScanner

Para crear el objeto servidor, hay que llamar al constructor con la ruta del fichero WSDL (línea 15). La ruta en este caso es la del propio fichero, pero con el parámetro `wdsl`.

Una vez creado, se asocia con la clase `Mate` (línea 16) y se le pasa la petición con `handle()` (línea 17). Aquí se produce el envío de la solicitud.

Actividad propuesta 10.1



Añade un método para calcular el factorial de un número a la clase Mate. Anótalo correctamente y comprueba el nuevo fichero WSDL generado.

10.3.2. Utilización de servicios web

Para utilizar un servicio SOAP se emplea la clase `Client`. Se inicializa también a partir del fichero WSDL. El objeto que se obtiene tendrá como métodos los *endpoints* expuestos.

```
$cliente = new Zend\Soap\Client(ruta);
```

Si se crea un cliente para el fichero WSDL anterior, tendrá un método potencia, correspondiente al *endpoint* potencia. Para pasarle argumentos se usa un *array*. Las claves son los nombres que figuran en el fichero WSDL.

```
<?php
require_once __DIR__. '../vendor/autoload.php';
    $cliente = new Zend\Soap\Client('http://localhost/cap10/servidor-
    soapmate.php?wsdl');
$result = $cliente->potencia(['base' => 2, 'exponente' => 3]);
echo $result->potenciaResult;
```

Si se accede desde el navegador, mostrará un 8 (el resultado de elevar 2 a 3).

Como se puede ver en el ejemplo, el objeto también tiene un atributo `potenciaResult`, que guarda el resultado para el *endpoint* `potencia`. Si hubiera más *endpoints*, habría más atributos `<método>Result`.

Al llamar al método potencia, el cliente monta un mensaje SOAP utilizando las definiciones del fichero WSDL. Accede al *endpoint* potencia.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:ns1="http://localhost/cap10/servidorsoapmate.php">
<env:Header/>
<env:Body>
    <ns1:potencia>
        <base>2</base>
        <exponente>3</exponente>
    </ns1:potencia>
</env:Body>
</env:Envelope>
```

Escaneado con CamScanner

En el servidor, el *endpoint* potencia está asociado con el método potencia de la clase `Mate`. El servidor SOAP la llama utilizando los datos del mensaje. Con el valor devuelto, monta el mensaje de respuesta.

Actividad propuesta 10.2

Escribe un programa para probar el *endpoint* creado para el factorial de la actividad propuesta 10.1.

10.4. Aplicaciones híbridas

Las aplicaciones híbridas o *mashups* integran datos y funcionalidad de diferente origen en una misma interfaz gráfica. Por ejemplo, un agrupador de noticias que incluya las de varios periódicos o que use la API de Google Maps para mostrar donde han ocurrido. Se suelen clasificar según su utilidad:

1. *De consumidor.* Combina datos de fuentes públicas y las muestra en una interfaz gráfica sencilla.
2. *De empresa.* Aplicaciones que combinan sus propios recursos con otros servicios externos.
3. *De datos.* En este caso, se combinan datos de diferentes orígenes, pero el resultado no es necesariamente gráfico. Constituyen un nuevo servicio que puede ser utilizado desde otros sistemas.

10.4.1. Interfaces de programación y repositorios

En general, el contenido integrado proviene de:

- Servicios web.
- Fuentes o canales *(feeds)*.
- APIs web.

A) Fuentes o canales web

Las fuentes o canales web *(feed)* son un medio de sindicación de contenido muy extendido. En los canales, las páginas web comparten el nuevo contenido que van incorporando. Los usuarios se pueden suscribir a estos canales a través de un programa apropiado (puede ser el propio navegador). Los *podcasts*, por ejemplo, funcionan así. Los formatos más utilizados son RSS y ATOM.

El Instituto Geográfico Nacional tiene varios canales RSS. Uno de ellos muestra información sobre los últimos terremotos detectados en España: http://www.ign.es/ign/RssTools/sismologia.xml.

Si se accede desde el navegador, se verá:

Escaneado con CamScanner

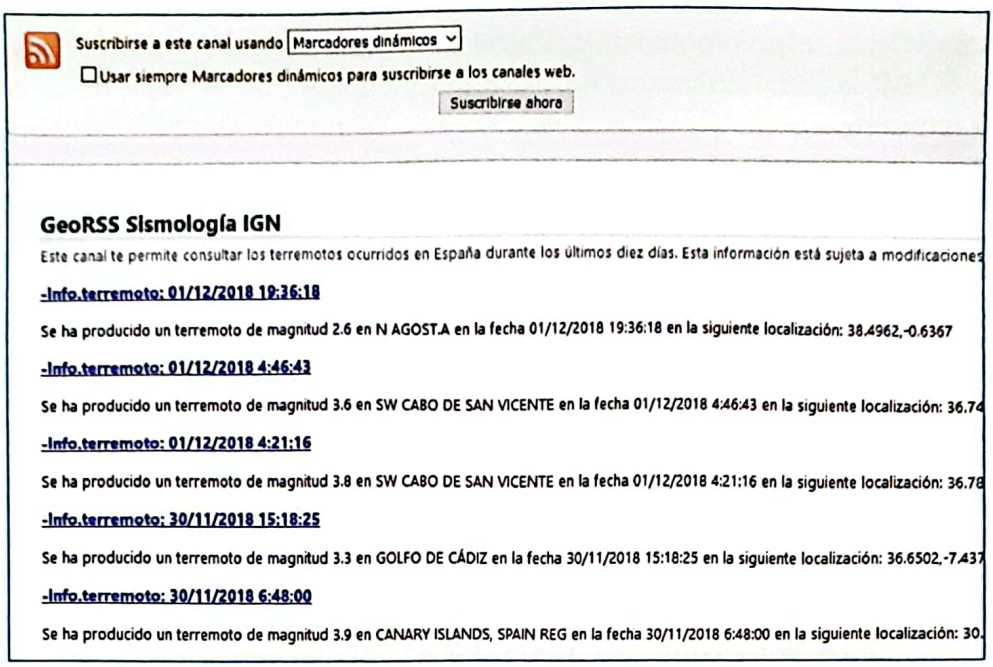


Figura 10.2
Canal RSS en Firefox.

El navegador aplica sus propias reglas de estilo al documento. Se puede observar que ofrece la posibilidad de suscribirse al canal. De esta manera, se notificarían los cambios en el canal. Para ver el fichero propiamente dicho hay que acceder al código fuente.

El fichero tiene la estructura habitual del formato RSS. El elemento raíz es `rss` y contiene un elemento `channel`, con los datos del canal. Cada terremoto es un elemento `ítem`, dentro del cual hay varios campos, entre ellos `title`, con la descripción del evento, y `geo:lat` y `geo:long`, con la latitud y longitud del epicentro.

```
<?xml version="1.0" encoding="UTF-8"?>
<rss …>
  <channel>
    <title>GeoRSS Sismología IGN</title>
    <link>http://www.ign.es/ign/rss</link>
    <description>Este canal te permite …</description>
    <language>es</language>
    <dc:language>es</dc:language>
    <item>
      <title>-Info.terremoto: 01/12/2018 19:36:18</title>
      <link>…</link>
      <description>…</description>
      <guid>… </guid>
      <geo:lat>38.4962</geo:lat>
      <geo:long>-0.6367</geo:long>
    </item>
    <item>
      …
    </item>
  </channel>
</rss>
```


Escaneado con CamScanner

El siguiente ejemplo accede al canal y muestra una tabla con la descripción y las coordenadas de cada terremoto. En el bloque inicial de PHP carga los datos y los pasa a un *array*. En el segundo bloque, muestra el *array* en una tabla.

```
<?php
$fichero = new DOMDocument();
$fichero->load( "http://www.ign.es/ign/RssTools/sismologia.xml");
$salida = array();
$terremotos = $fichero->getElementsByTagName("item");
foreach($terremotos as $entry) {
    $nuevo = array();
    $nuevo["title"] = $entry->getElementsByTagName("title")
        [0]->nodeValue;
    $nuevo["lat"] = $entry->getElementsByTagName("lat")[0]->nodeValue;
    $nuevo["lng"] = $entry->getElementsByTagName("long")[0]->nodeValue;
    $salida[] = $nuevo;
}
?>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>Últimos terremotos</title>
        <style> table, td, th { border-style: solid} </style>
    </head>
    <body>
        <table>
            <tr><td>Título</td><td>Longitud</td><td>Latitud</td></tr>
            <?php
            foreach($salida as $elemento) {
                $titulo = $elemento["title"];
                $lat = $elemento["lat"];
                $lon = $elemento["lng"];
                echo "<tr><td>$titulo</td><td>$lon</td><td>$lat</td></tr>";
            }
            ?>
        </table>
    </body>
</html>
```

El resultado es el siguiente:

Título	Longitud	Latitud
–Info.terremoto: 01/12/2018 19:36:18	-0,6367	38,4962
–Info.terremoto: 01/12/2018 4:46:43	-9,7053	36,7433
–Info.terremoto: 01/12/2018 4:21:16	-11,4621	36,7835
–Info.terremoto: 30/11/2018 15:18:25	-7,437	36,6502
–Info.terremoto: 30/11/2018 6:48:00	-16,1625	30,7435
–Info.terremoto: 29/11/2018 19:25:02	-10,3245	39,2279
–Info.terremoto: 29/11/2018 9:09:36	-0,0978	43,024

Figura 10.3
Tabla de terremotos.

Escaneado con CamScanner

Ejemplo

La mayoría de los periódicos tienen uno o más canales RSS para sus noticias. Un ejemplo es el canal de últimas noticias de *El País*:

http://ep00.epimg.net/rss/tags/ultimas_noticias.xml.

Escribe un programa que cargue la fuente y muestre una lista con los titulares. Los elementos de la lista tienen que ser los vínculos a la noticia en el periódico.

```
<?php
$fichero = new DOMDocument();
$fichero->load("http://ep00.epimg.net/rss/tags/ultimas_noticias.
xml");
$salida = array();
$noticias = $fichero->getElementsByTagName("item");
foreach($noticias as $noticia) {
    $nuevo = array();
    $nuevo["titular"] = $noticia
        ->getElementsByTagName("title")[0]
        ->nodeValue;
    $nuevo["url"]=$noticia->getElementsByTagName("link")[0]->nodeVa-
    lue;
    $salida[] = $nuevo;
}
?>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>Últimas noticias</title>
    </head>
    <body>
        <ul>
            <?php
            foreach($salida as $noticia) {
                $url = $noticia["url"];
                $titular = $noticia["titular"];
                echo "<li><a href = '$url'>$titular</a>";
            }
            ?>
        </ul>
    </body>
</html>
```

Para integrar estos en una aplicación híbrida, una opción es utilizar AJAX. Así se podría tener una barra lateral con los datos de los últimos terremotos o noticias que se actualice periódicamente. En este caso es mejor devolver los datos en JSON, como hace el ejemplo **terremotos_json.php**.

Escaneado con CamScanner

```php
<?php
$fichero = new DOMDocument();
$fichero->load( "http://www.ign.es/ign/RssTools/sismologia.xml");
$salida = array();
$terremotos = $fichero->getElementsByTagName("item");
foreach($terremotos as $entry) {
    $nuevo = array();
    $nuevo["title"] = $entry->getElementsByTagName("title")
    [0]->nodeValue;
    $nuevo["lat"] = $entry->getElementsByTagName("lat")[0]->nodeValue;
    $nuevo["lng"] = $entry->getElementsByTagName("long")[0]->nodeValue;
    $salida[] = $nuevo;
}
echo json_encode($salida);
```

Es como el anterior, pero devuelve el *array* en JSON en lugar de mostrar la tabla.

[{"title":"-Info.terremoto: 01\/12\/2018 19:36:18","lat":"38.4962","lng":"-0.6367"}, {"title":"-Info.terremoto: 01\/12\/2018 :46:43","lat":"36.7433","lng":"-9.7053"},...]

Actividades propuestas

10.3. Busca en internet canales RSS de alguna temática que te interese.

10.4. A partir del ejemplo anterior, crea una página que utilice JavaScript para mostrar la información de los terremotos que se actualice cada cinco minutos.

B) *Google Maps*

La API de Google Maps es una de las más utilizadas en aplicaciones híbridas. Permite integrar un mapa en una aplicación web. Se puede escoger el nivel de *zoom*, donde está centrado y poner marcadores. Se utiliza desde JavaScript.

Actividad propuesta 10.5

Para utilizar la API de Google Maps hace falta una clave (API *key*) que proporciona Google. Consigue una para poder probar el ejemplo:

https://developers.google.com/maps/documentation/javascript/get-api-key

Para incluirlo en una página, se utiliza un elemento *script* como este:

```
<script async defer src="https://maps.googleapis.com/maps/api/
js?key=<CLAVE>&callback=<FUNCION>">
</script>
```


Escaneado con CamScanner

Hay que sustituir <CLAVE> por la clave de la API, y <FUNCION> por el nombre de la función JavaScript a la que se llama al cargar el mapa.

La página tiene que tener un `div` con `id = 'map'`, que es donde se mostrará el mapa y estas opciones de estilo.

```
<style>
    #map {
        height: 100%;
    }
    html, body {
        height: 100%;
        margin: 0;
        padding: 0;
    }
</style>
```

El ejemplo **mapa_terremotos.php** utiliza **terremotos_json.php** para obtener los datos de los últimos terremotos y los visualiza en un mapa.

- La función `initMap()` (líneas 18-46) se encarga de inicializar el mapa.
- Es la función que se especifica al incluir el *script* (líneas 48-50). Para que el ejemplo funcione, hay que introducir una clave válida.
- El mapa se crea en las líneas 21-26 indicando el nivel de *zoom*, donde está centrado y el tipo de mapa.
- En la línea 44 se hace una petición AJAX a **terremotos_json.php**. La respuesta será un *array* de objetos con campos `title`, `long` y `lat`, los nombres que aparecen en el documento JSON.
- Cuando se recibe la respuesta (líneas 28-42), con cada uno de los elementos se añade un nuevo marcador al mapa (líneas 30-37). Se utilizan los campos `lat` y `long` para situarlo, y `title` como título. Se muestra al pasar el ratón por encima.

```
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <style>
5           #map {
6               height: 100%;
7           }
8               html, body {
9               height: 100%;
10              margin: 0;
11              padding: 0;
12          }
13          </style>
14      </head>
15      <body>
16          <div id="map"></div>
17          <script>
```


Escaneado con CamScanner

```
18        function initMap() {
19              var map;
20              var results;
21                map = new google.maps.Map(
22                 document.getElementById('map'),
23                 {zoom: 5.5,
24                 center: new google.maps.LatLng(40,-4),
25                 mapTypeId: 'terrain'
26                 });
27              var xhttp = new XMLHttpRequest();
28              xhttp.onreadystatechange = function() {
29                  if (this.readyState == 4 && this.status == 200) {
30                     results = JSON.parse(this.responseText);
31                     for (var i = 0; i < results.length; i++) {
32                          var title = results[i].title;
33                          var latLng = new google.maps.LatLng(
34                              results[i].lat, results[i].lng
35                          );
36                         var marker = new google.maps.Marker({
37                         position: latLng,
38                         title:title,
39                         map: map
40                         });
41                     }
42                   }
43              }
44              xhttp.open("GET", "terremotos_json.php", true);
45              xhttp.send();
46              }
47        </script>
48        <script async defer
49              src="https://maps.googleapis.com/maps/api/js?key= &callback=
                  initMap">
50        </script>
51      </body>
52  </html>
```

El resultado en el navegador es el que se muestra en la figura 10.4.



Figura 10.4
Mapa de terremotos.

Escaneado con CamScanner

Resumen

- Los servicios son componentes independientes que se pueden utilizar desde diferentes aplicaciones o sistemas.
- El W3C mantiene una serie de estándares relacionados con los servicios web, entre ellos SOAP y WDSL.
- PHP tiene una extensión, SOAP, con la funcionalidad básica. Otras librerías, como Zend\Soap, ofrecen más facilidades de uso.
- El servidor de Zend\Soap se asocia con una clase, cuyos métodos serán los *endpoints* del servicio.
- Con la librería Zend\Soap los ficheros WSDL se pueden generan a partir de la clase asociada.
- El cliente Zend\Soap se inicializa con la ruta del fichero WSDL del servicio.
- Las aplicaciones híbridas integran datos y funcionalidad a través de servicios, canales y APIs web.
- Un ejemplo muy extendido es la API de Google Maps, para integrar mapas de Google en cualquier página web.
- Las fuentes o canales web son un medio de sindicación de contenidos. Los formatos más extendidos son ATOM y RSS.
- Muchos organismos públicos y páginas web utilizan RSS para publicar información.



Ejercicios propuestos

1. Escribe una página que utilice el servicio del apartado 10.4. El usuario introducirá base y exponente en un formulario.
2. Modifica el ejercicio anterior para que el formulario se envíe con AJAX.
3. Realiza las siguientes tareas:

   a) Escribe un servicio web con un *endpoint* que devuelva los datos de la tabla de categorías de la base de datos de pedidos en formato JSON. Puedes reutilizar la función del capítulo 6.

   b) Escribe un cliente para probarlo.

4. El Ministerio de Ciencia tiene un canal RSS para publicar noticias:

   http://www.ciencia.gob.es/portal/site/MICINN/rss

   Escribe un programa que muestre una tabla a partir de los datos del canal.
5. Elabora una página que integre los ejercicios 2, 3 y 4 utilizando AJAX.
6. Busca alguna API web con funcionalidad interesante e intenta ponerla en práctica. Por ejemplo:

   - Wikipedia: https://www.mediawiki.org/wiki/API:Main_page
   - Flicker: https://www.flickr.com/services/api/

Escaneado con CamScanner

Actividades de autoevaluación

1. La descripción de los servicios web se realiza mediante:
   - ☐ *a)* REST.
   - ☐ *b)* SOAP.
   - ☐ *c)* WSDL.

2. ¿Cuál de los siguientes términos no es un estándar del W3C?
   - ☐ *a)* SOAP.
   - ☐ *b)* WSDL.
   - ☐ *c)* SOA.

3. Las aplicaciones híbridas:
   - ☐ *a)* Integran datos de diversas fuentes.
   - ☐ *b)* Integran funcionalidad de diversas fuentes.
   - ☐ *c)* Integran funcionalidad y datos de diversas fuentes.

4. La extensión SOAP de PHP y la librería Zend\Soap utilizan:
   - ☐ *a)* WSDL 1.
   - ☐ *b)* WSDL 2.
   - ☐ *c)* Según se configure al instalar.

5. La clase Zend\Soap\Server:
   - ☐ *a)* Se asocia con una clase cuyos métodos serán los *endpoints* del servicio.
   - ☐ *b)* Contiene una serie de métodos que serán los *endpoints* del servicio.
   - ☐ *c)* Define una serie de *endpoints* por defecto que se pueden sobrescribir.

6. ¿Cuál de las siguientes afirmaciones es correcta?
   - ☐ *a)* Un servicio se compone de varios *endpoints*.
   - ☐ *b)* Un *endpoint* se compone de varios servicios.
   - ☐ *c)* Un servicio se compone de un *endpoint* y un *frontpoint*.

7. El constructor de la clase Zend\Soap\Client recibe:
   - ☐ *a)* Un fichero WSDL.
   - ☐ *b)* Una clase cuyos métodos serán los *endpoints* del servicio.
   - ☐ *c)* Un fichero RSS.

8. Los canales o fuentes RSS sirven para:
   - ☐ *a)* Compartir contenido.
   - ☐ *b)* Publicar servicios web.
   - ☐ *c)* Encontrar servicios web.

9. ¿Cuál de los siguientes no es un formato para canales web?
   - ☐ *a)* UDDI.
   - ☐ *b)* ATOM.
   - ☐ *c)* RSS.

Escaneado con CamScanner

10. La API de Google Maps se utiliza:

- ☐ *a)* Desde el servidor.
- ☐ *b)* Desde el cliente.
- ☐ *c)* Se puede elegir.

SOLUCIONES:

1. a b **c**
2. a b **c**
3. a b **c**
4. **a** b c
5. **a** b c
6. **a** b c
7. **a** b c
8. **a** b c
9. **a** b c
10. a **b** c

Escaneado con CamScanner

Webgrafía

Aplicaciones híbridas

https://www.programmableweb.com

Doctrine

https://www.doctrine-project.org/

PHP

http://www.php.net/

SOAP

https://www.w3.org/TR/soap/

Symfony

https://symfony.com/

Web Services

https://www.w3.org/TR/ws-arch/

WSDL

https://www.w3.org/TR/wsdl/

Escaneado con CamScanner

Desarrollo web en entorno servidor



INFORMÁTICA Y COMUNICACIONES

G.S. Desarrollo de Aplicaciones Web

Este libro explica de forma clara, didáctica y muy práctica cómo desarrollar aplicaciones web. Contiene multitud de ejemplos para descargar y poder practicar, además de actividades y ejercicios para afianzar todo lo aprendido. Se pone especial énfasis en el desarrollo de aplicaciones completas. A lo largo del libro, se desarrollan tres aplicaciones que incorporan los contenidos de los capítulos anteriores. Primero utilizando solo PHP, luego incorporando AJAX y finalmente con Symfony.

El libro, concebido especialmente como manual para el módulo Desarrollo web en entorno servidor, puede ser útil también para cualquier persona interesada en aprender PHP o Symfony.

Esta obra incluye los siguientes recursos didácticos:

Objetivos • Mapa conceptual • Glosario • Recursos didácticos ("Toma nota", "Recuerda", "Recursos web") • Actividades propuestas
Cuadros y figuras • Ejercicios propuestos • Resumen
Actividades de autoevaluación (tipo test)

ISBN: 978-84-9171-349-4
21780
9 788491 713494

EDITORIAL SÍNTESIS

Escaneado con CamScanner