La Sénia
Institut d'Educació Secundària
Paiporta

UD6. Mantenimiento del estado: Cookies y Sesiones

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Mantenimiento del estado.
- Cookies.
- Sesiones.

Mantenimiento del estado

- El mantenimiento del estado de la conexión entre cliente y servidor puede hacerse mediante dos formas: ***Cookies*** o ***Sesiones***. La primera, se almacenarán los datos en el cliente y en la segunda, los datos del estado se guardarán en el servidor
- Por motivos de persistencia y seguridad, es aconsejable usar sesiones ya que el cliente puede rechazar las cookies o bien pueden ser borradas.
- La sesión es como se denomina a la conexión entre cliente y servidor. Cada sesión puede abarcar múltiples páginas web y su seguimiento se realiza mediante la gestión del estado de la conexión y los datos.

Mantenimiento del estado

Muchas veces es necesario mantener el estado de una conexión entre distintas páginas o entre distintas visitas a un mismo sitio, de modo que se conserven los datos generados.

Algunas situaciones en las que se requiere mantener el estado son, por ejemplo, con la finalidad de:

- Acumular una cesta de la compra
- Control de acceso de los usuarios
- Conocer los pasos de la navegación y preferencias del usuario
- Actualizar una base de datos
- Etc.

Cookies

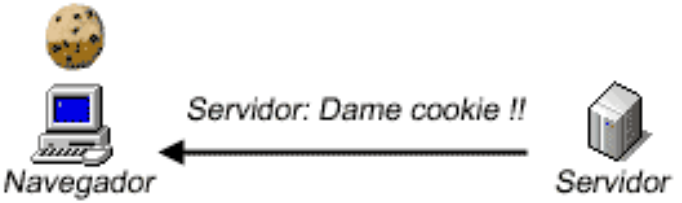
El manejo de las Cookies en PHP es sencillo:

- Primero se envía la Cookie, invocando a la función **setcookie()** para crear la Cookie en el cliente
- En las posteriores peticiones que recibamos de ese cliente vendrá incrustada la Cookie. Se consulta la información almacenada en ella accediendo a la superglobal **$_COOKIE**.
- Las Cookies se envían en las cabeceras de las transacciones HTTP y deben enviarse antes que cualquier otra cabecera HTML (restricción de las Cookies no de PHP)

Cookies - Proceso

1) La Cookie es enviada al navegador desde el servidor y, si la acepta, permanece en el cliente

2) Las páginas ejecutadas en el servidor piden la Cookie al navegador



3) El navegador envía la Cookie permitiendo la identificación del usuario por parte del servidor

Cookies - Campos

Las Cookies son bloques de texto sin formato, de tamaño máximo 4KB y formadas por varios campos:

- **Nombre** (requerido): Nombre de la Cookie
- **Valor**: Valor asociado con codificación URL
- **Fecha Expiración**: Momento en el que deja de ser válida
- **Path**: Subconjunto de URLs para los que la Cookie es válida
- **Dominio**: rango de dominios para los que la cookie es válida en el servidor
- **Segura**: Indica si la cookie se debe transmitir exclusivamente sobre https
- **Httponly**: Accesible sólo por el protocolo HTTP (no script JavaScript)

Ver RFC6265 http://www.faqs.org/rfcs/rfc6265.html

Cookies - Salvedades

- Las Cookies ***no son visibles hasta la próxima carga*** de la página en la que debieran serlo. Para probar si se ha creado, se debe buscar la cookie en alguna página cargada posteriormente y antes que la cookie expire.

  Podemos comprobar las Cookies desde PHP con:
  echo $_COOKIE["MyCookie"];** o **print_r($_COOKIE);
- Al ***borrar*** una Cookie hay que ***asegurar que ha expirado***, por ejemplo asignando $_COOKIE expires a un momento previo.
- Si a $_COOKIE value se asigna FALSE (o **‘’**) y los demás campos tienen los mismos valores que en la llamada anterior, intentará eliminar la cookie. Por ello, al configurar valor, no se debe usar valores booleanos sino indicar 0 para FALSE y 1 para TRUE.
- Múltiples llamadas a setcookie() se efectúan en el orden de llamada.

Cookies - Ejemplo

```
<?php
$cookie_name = "user";
$cookie_value = "Silvia";
$cookie_expires=time() + (60*60*24 * 30) //30 días
$cookie_path= "/" //Todo el sitio Web
setcookie($cookie_name, $cookie_value,$cookie_expires,$cookie_path);
// la cookie se debe crear previamente a cualquier cabecera o html
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
<p><strong>Note:</strong> You might have to reload the page to see the value of the cookie.</p>
</body>
</html>
```

Cookie 'user' is set!
Value is: Silvia
Note: You might have to reload the page to see the value of the cookie.

Cookies - Borrado

Podemos diferenciar las siguientes situaciones:

1) Queremos **borrar los parámetros** de la Cookie: Se invoca a setcookie() sin parámetros

```
<?php
    setcookie(“user”)
?>
```

2) Queremos **eliminar del servidor la variable cookie** ya leída: Usamos la función unset()

```
<?php
    unset($_COOKIE[“user”])
?>
```

3) Queremos **eliminar el archivo Cookie del cliente**: Indicaremos un tiempo de expiración 0 o anterior al actual

```
<?php
    setcookie(“user”,”Silvia”,0) // o time()-Valor_en_segundos
?>
```

Cookies - Ejemplo

- Para borrar la cookie invocamos setcookie con los parámetros value=”” y $expires a un tiempo anterior al presente:

```
<?php
$cookie_name = "user";
$cookie_value = """; //también funciona con FALSE
$cookie_expires=time() -3600 //Hace una hora
$cookie_path= "/" //Todo el sitio Web
setcookie($cookie_name, $cookie_value,$cookie_expires,
$cookie_path);
//También podríamos invocar:
setcookie($cookie_name,””,time() -3600);
?>
```

- Si ***no se indica fecha de expiración*** o el ***valor indicado es 0***, la cookie expirará (se borra) tras cerrar la sesión (cierre del navegador)

Sesiones

- Las sesiones son una manera de guardar información específica para cada usuario durante toda su visita.
- Cada usuario que entra en un sitio abre una sesión, independiente de la sesión de otros usuarios.
- En la sesión de un usuario podemos almacenar todo tipo de datos: nombre del usuario, productos de un carrito de la compra, preferencias de visualización o trabajo, páginas por las que ha navegado, etc.
- Todas estas informaciones se guardan en lo que denominamos variables de sesión.

Sesiones vs Cookies

- Las sesiones mantienen el valor de las variables a lo largo de toda la navegación.
- Las cookies permiten almacenar poca información (sólo 4KB), las sesiones no tienen esa limitación.
- Muchos usuarios/ navegadores desactivan las Cookies.
- Las sesiones almacenan la información en el servidor en lugar de almacenarla en el cliente como las Cookies. Su ubicación se indica en ***session.save_path*** en php.ini
- PHP internamente genera un identificador de sesión único **(SID, Session ID)**, que sirve para determinar las variables de sesión que pertenecen a cada usuario. Se accede desde la cookie **PHPSESSID**
- Para propagar el SID al cliente y que lo indique en su visita, bien se guarda en una Cookie o bien se pasa por parámetro en la URL (query string).

Sesiones vs Cookies

Las sesiones funcionan aunque las Cookies estén desactivadas. Si el servidor web detecta que las cookies no están activadas, añade ***automáticamente*** el id de sesión como un **query string** ***nombre_sesión=SID*** en todos los enlaces de la página. Para que esto funcione hay que cumplir estos requisitos:

- Todas las páginas tienen que tener la extensión **.php**, para que php pueda añadir la SID a las páginas.
- Es necesario que ***session.use_trans_sid*** esté activado y ***session.use_only_cookies*** esté desactivado en **php.ini.**
  https://www.php.net/manual/es/session.configuration.php#ini.session.use-trans-sid
- Se pueden generar en anchors (etiquetas <a>) y formularios pasando automáticamente la constante SID, pero no así con header(location) que debe propagase de forma manual.
- Sólo se aplica en **direcciones relativas** del servidor, no absolutas (servidores externos) por seguridad.
- Usar **htmlspecialchars(SID)** para prevenir ataques

Sesiones

- Los datos entre peticiones de la sesión se almacena en la variable superglobal **$_SESSION** y permanecen hasta que se cierra el navegador.
- Cuando un cliente accede a un sitio web PHP comprobará, automáticamente (si **session.auto_start** está establecido a 1) o sobre su petición (explícitamente a través de **session_start()**), si se ha enviado un id de sesión específico con la petición. Si éste es el caso, se recrea el entorno anteriormente guardado.

Ver aspectos de seguridad en sesiones:
https://www.php.net/manual/es/session.security.php

Sesiones: Funciones

session_start() crea una sesión o reanuda la actual basada en el SID pasado mediante una petición GET/POST o bien una cookie.

session_name() establece o devuelve el valor nombre de la sesión. Por defecto es PHPSESSID.

session_id() Devuelve y/o establece el identificador de la sesión.

Sesiones – Pasando el SID

PHP añade ***automáticamente*** el SID de forma transparente al programador según la configuración de ***session.trans_sid_tags*** (en las etiquetas HTML a=href, area=href, frame=src, input=src, form=). <input hidden="id_sesión" name="nombre_sesión"> se añade como variable de formulario .Por seguridad, los formularios deben enviarse con método POST de modo que el SID no aparezca en la URL.

Ejemplo:

- En nuestro archivo index.php indicamos el SID:

  <**a href**="otrapagina.php?**<?php print SID; ?>**">Otra página

  <a href="otrapagina.php?**<?php print session_name(); ?>**">Otra página // nos referimos a la sesión con un nombre

  <a href="otrapagina.php?**<?php print session_id(); ?>**">Otra página //Otro modo de recuperar el SID

- Al navegador le llegará de la siguiente forma:

Otra página

Sesiones – Pasando el SID

```
<?php
ini_set(“session.use_trans_sid”, “1”); //activamos sesión si fallan cookies
ini_set(“session.use_cookies”, “0”); //desactivamos el uso de cookies
ini_set(“session.use_only_cookies”, “0”); //desactivamos obligación de usar sólo cookies
session_start(); //iniciamos la sesión
$_SESSION['dato'] = “cliente nuevo”; //ponemos un dato de prueba
?>
<html>
<form action="nextpage.php?<?php echo htmlspecialchars(SID); ?>" method="POST">
    Introduce el dato: <input type="Text" name="dato">
</form>
</html>
```

Sesiones – Pasando el SID

En nextpage.php:

```
<?php
session_start(); //iniciamos la sesión
$_SESSION['dato'] = $_POST['dato'];

if (empty($_SESSION['dato'])) {
   echo "No hay dato introducido";
} else {
   echo "El dato introducido es: ", $_SESSION['dato'];
}
// Cambia "cliente nuevo" por el valor introducido en el Text
?>
```

En la url de la página nextpage.php a la que hemos enviado el dato nuevo, podéis observar el valor de PHPSESSID

El dato introducido es: Dato nuevo

Sesiones – Ejemplo Sesión

```
<?php
ini_set("session.use_trans_sid", "1"); //activamos sesión si fallan cookies
ini_set("session.use_cookies", "0"); //desactivamos el uso de cookies
ini_set("session.use_only_cookies", "0"); //No usar sólo cookies
session_start(); //iniciamos la sesión
if (empty($_SESSION['count'])) {    //contaremos las veces que visita la página
    $_SESSION['count'] = 1;
} else {
    $_SESSION['count']++;
}
?>
<html>
<body>
    <p>
        Hola visitante, ha visto esta página <?php echo $_SESSION['count']; ?>
veces.
    </p>
    <p>
        Para continuar, <a href="nextpage.php?<?php echo htmlspecialchars(SID);
?>">haga clic aquí</a>.
    </p>
</body>
</html>
```

Sesiones – Funciones de uso

Funciones de PHP para el manejo de sesiones:

- Registra o modifica una variable de sesión

$_SESSION['nombre'] = valor;

- Elimina una o todas las variables de sesión

unset ($_SESSION['nombre']);

Otra forma menos elegante, asignar vector vacío:

$_SESSION=array();

- Comprueba si la variable de sesión está creada

if (isset($_SESSION['nombre']))

Sesiones – Funciones de uso

Funciones de PHP para el manejo de sesiones:

- **session_cache_expire()**: Devuelve/asigna la caducidad de la caché actual.
- **session_cache_limiter()**: Obtener y/o establecer el limitador de caché actual.
- **session_commit()**: alias de session_write_close. Guarda los datos de la sesión y la cierra
- **session_decode()**: decodifica la información de sesión desde una cadena.
- **session_encode()**: codifica la información de la sesión actual y los devuelve como una cadena.
- **session_regenerate_id()**: Actualiza el id de sesión actual con un nuevo valor más reciente.
- **session_create_id()**: Genera un nuevo SID.

Sesiones – Funciones de uso

- **session_save_path()**: Obtiene y/o establece la ruta de almacenamiento de la sesión actual.
- **session_write_close()**: Escribe la información de sesión y finalizar la sesión.
- **session_reset()**: Reinicializa el array de sesión con los valores originales
- **session_unset()**: Libera las variables de la sesión
- **session_abort()**: Desecha los cambios en el array de sesión y la finaliza
- **session_destroy()**: Destruye toda la información asociada con la sesión actual pero no destruye las variables globales asociadas con la sesión, ni destruye la cookie de sesión. Para volver a utilizar las variables de sesión se debe llamar a session_start(). Para eliminar por completo la cookie con el SID se debe usar setcookie()

Sesiones – Funciones de uso

- **session_get_cookie_params()**: Devuelve una matriz asociativa con la información de la cookie de la sesión. Se puede usar para obtener los parámetros y así pasarlos para caducar la sesión.

Alguna información que contiene es:

  - 'lifetime': Tiempo de vida de la cookie de sesión
  - 'path': Ruta donde se guarda la cookie de sesión.
  - 'domain': Dominio del servidor que genera la cookie.
  - 'secure': Sólo se puede enviar en conexiones seguras
  - 'httponly': La cookie sólo se propaga por HTTP (no acceso desde lenguajes script como JS).

- **session_set_cookie_params()**: Se establecen los parámetros de la sesión antes de llamar a session_start(), Este efecto sólo dura para cada petición durante la ejecución del script

Sesiones – Funciones de uso

/* Ejemplo de uso para eliminar la cookie de sesión con los mismos parámetros de creación (recordad que session_destroy no elimina la cookie de sesión ni las variables globales asociadas a ella) */

```
// Obtenemos los datos de la sesión actual
$CookieInfo = session_get_cookie_params();

if ( (empty($CookieInfo['domain'])) && (empty($CookieInfo['secure'])) ) {
      //Si no hay valores específicos para domain y secure, no se indican
      setcookie(session_name(), '', time()-3600, $CookieInfo['path']);
} elseif (empty($CookieInfo['secure']))
      { //No se indican valores para secure
            setcookie(session_name(), '', time()-3600,$CookieInfo['path'],
                  $CookieInfo['domain']);
      } else { //se indican todos los valores que tenía la cookie de sesión
                  setcookie(session_name(), '', time()-3600, $CookieInfo['path'],
                        $CookieInfo['domain'], $CookieInfo['secure']);
                  }
session_destroy();
```

La Sénia
Institut d'Educació Secundària
Paiporta

UD7. Seguridad: Roles y Tokens

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Seguridad: usuarios, perfiles, roles.
- Autenticación de usuarios.
- Tokens de formulario

Seguridad: usuarios, perfiles, roles

- Para controlar el acceso de los usuarios, se implementan las ACL (Access Control List) en las que se determinan los permisos de acceso en base a **roles o perfiles** que son aplicables a usuarios y grupos de usuarios.
- Los pasos para implementar la seguridad con ACL son:

1) Crear la ACL que puede estar implementada en una BD, una estructura de datos, etc.

2) Usar la ACL en la página Web comprobando el rol al que pertenece el usuario para aplicarle los permisos correspondientes y visualizar contenido

Ejemplo roles

```
<?php      /* roles.php */
session_start();
     $_SESSION['usuario']=$_POST['usuario'];
     $_SESSION['pass']=$_POST['password'];
     $_SESSION['rol']=$_POST['rol'];

if (isset($_SESSION['usuario'])){
   switch ($_SESSION['rol']) {
      case 'Estudiante':
         $location = 'Location: indexEstudiante.php';
         break;
      case 'Profesor':
         $location = 'Location: indexProfesor.php';
         break;
      case 'Otro':
         $location = 'Location: indexDefault.php';
         break;
   }
}
header($location);
?>
```

Nota: No se han validado los datos, sólo la comprobación de hay un usuario en la sesión

Ejemplo roles

```
/* roles.php */
<html>
    <form  action="roles.php" method="post">
        Nombre: <input type="text" name="usuario"/>
        Password: <input type="password" name="password"/>
        <p>
        Estudiante <input type="radio" name="rol" value="Estudiante"/>
        Profesor <input type="radio" name="rol" value="Profesor"/>
        Otro <input type="radio" name="rol" value="Otro" />
        </p>
        <input type="submit" name="Acceder" value="Acceder"/>
    </form>
</html>
```

Ejemplo roles

/* En cada uno de los ficheros indexDefault.php, indexProfesor.php e indexEstudiante.php */

```
<?php
session_start();

print ("Bienvenido ".$_SESSION['usuario']."!\n");
print ("Tu rol es ".$_SESSION['rol']."\n");
print ("Tu password es ".$_SESSION['pass']);

?>
```

Autenticación de usuarios

1) Podemos crear nuestro propio formulario de autenticación de usuarios donde en base a las características del usuario (identificador y password) se pueda validar.
2) Por otra parte, la autenticación de usuarios puede realizarse en el propio servidor web. En el caso de Apache, los ficheros **.htaccess** permiten limitar el acceso a un determinado recurso del servidor. Basta con configurar las directivas adecuadas en el .htaccess en la carpeta de nuestro proyecto para que se apliquen en nuestro sitio web
3) Además, disponemos de la llamada ***autenticación universal*** (ejemplo: validarse con una cuenta de gmail, FB, Twitter, etc.) que nos facilita los siguientes métodos:
   - **OpenId**: Permite a un usuario entrar en una página web pudiendo ser verificado por otro servidor que soporte este protocolo
   - **OAuth**: Dispone de una API segura de autorización.

Autenticación de usuarios - Passwords

- Para encriptar los passwords es recomendable usar la función **password_hash()** que incluye el algoritmo, el coste y el salt (obsoleto) del hash devuelto y simplifica su comprobación con **password_verify()**

Ejemplo de creación de password:

```
<?php
/**
 * Se prefiere el modificador PASSWORD_DEFAULT a PASSWORD_BCRYPT
 * Actualmente BCRYPT produce un truncamiento de 72 caracteres . Pero con
 * DEFAULT la longitud puede variar, se debería poder almacenar hasta 255
 * caracteres
 */
    echo password_hash("silviavilar", PASSWORD_DEFAULT);
?>
```

Resultado:

```
$2y$10$aGPT0rVPoBo9161i6.uKf.Pxw0.BLmYjc85lypdOoxPJsbzWa3GXy
```

Autenticación de usuarios - Passwords

Ejemplo de comprobación de un password:

```
<?php
// Ver el ejemplo de password_hash() para ver de dónde viene este hash.
$hash
='$2y$10$aGPT0rVPoBo9161i6.uKf.Pxw0.BLmYjc85lypdOoxPJsbzWa3GXy';

if (password_verify('silviavilar', $hash)) {
    echo '¡La contraseña es válida!';
} else {
    echo 'La contraseña no es válida.';
}
if (password_verify('otrouser', $hash)) {
    echo '¡La contraseña es válida!';
} else {
    echo 'La contraseña no es válida.';
}
?>
```

Resultado:
¡La contraseña es válida!
La contraseña no es válida.

Autenticación con Sesiones - Ejemplo

```
<?php                          /*autentica.php*/
//creamos la variable de sesión de usuario autenticado para consultarla después
session_start(); //iniciamos la sesión

if (!isset($_SESSION["autenticado"])){
    if (isset($_POST["user"]) && isset($_POST["pass"])){
        if ($_POST["user"]==="silvia" && $_POST["pass"]==="123"){
            $_SESSION["autenticado"]="SI";
            header("Location: aplicacionsegura.php");
        } else // Credenciales erróneas, mostramos de nuevo index
            header("Location: index.php”);
    } else //No ha rellenado el formulario de autenticación
        header("Location: index.php");
} else // Ya está acreditado y no ha cerrado la sesión aún
    header("Location: aplicacionsegura.php");
?>
```

Problemas de seguridad - XSS

La principal motivación de los ataques es el robo de cookies y de sesiones, modificar el sitio web, redireccionar a otras páginas, etc. **Ataques XSS (Cross-Site Scripting)**: se basan en explotar la confianza que tiene un usuario en un determinado sitio web o aplicación. ***Se ejecutan en el navegador (cliente)***.La forma más común de ataque es introduciendo scripts en los controles de formularios o mediante subida de archivos con código malicioso.

La protección frente a este ataque es:

1) **Validación** de datos: que sean correctos y esperados
2) **Sanitización** de los datos: garantizar que son seguros eliminando parte indeseable y normalizándolos en la forma correcta. Ejemplo: evitar marcas de HTML en un string introducido por el usuario
3) Aplicar output **scaping**, esto es, evitar caracteres especiales al devolver el dato al cliente. Por ejemplo, usando funciones como htmlespecialchars() y htmlentities()

Ver: https://es.wikipedia.org/wiki/Cross-site_scripting

Problemas de seguridad - XSRF/CSRF

Ataques XSRF / CSRF (Cross-Site Request Forgery): se basan en explotar la confianza que un sitio web o aplicación tiene en un usuario en particular.

- La forma más común de ataque es el robo de una sesión iniciada del usuario y éste visita una página generada por el atacante durante la vigencia de la sesión. Entonces el atacante interactúa con el servidor con la sesión del usuario legítimo.
- La protección frente a este ataque es el uso de **token de formulario** de modo que se generará un token conteniendo un identificador único para la sesión del usuario usando funciones seguras como **bin2hex()** y **openssl_random_pseudo_bytes()**
- Cuando el procesador del formulario recibe datos, comprueba si el token recibido del formulario coincide con el de la sesión. Si no coincide descarta la petición y si coincide procesa la petición y borra la variable de sesión que contenía el valor del token.
- Evitan que se pueda reenviar un formulario y ataques CSRF/XSRF

Ver: https://es.wikipedia.org/wiki/Cross-site_request_forgery

Tokens de Formulario

Un token de formulario es un campo oculto que incluye un valor único y, a la vez, ese mismo valor se guarda en una variable de la sesión del usuario para después comprobar que sean el mismo.

```
// Ejemplo generación
session_start();
$_SESSION["token"] = bin2hex(openssl_random_pseudo_bytes(24));
// Inclusión en el formulario en un control oculto
<form action="process.php" method="post">
    <input type="hidden" name="token" value="<?php echo $_SESSION['token']; ?>">
    <input type="text" name="email" placeholder="Your email address..."><br>
    <input type="submit" name="submit_form">
</form>
// También podemos pasarlo como parámetro en la url cuando cierre sesión, por ejemplo
<!--Protege la URL de cierre de sesión de ataques CSRF-->
<a href="logout.php?token=<?php echo $_SESSION['token'];?>"> Logout </a>
```

Tokens de Formulario

```
//Procesamos el formulario en process.php
//Debemos recordar siempre iniciar la sesión para recuperarla.
session_start();

//Nos aseguramos de que el token se ha guardado en la variable $_POST.
if (!isset($_POST['token'])) {
    print('No se ha encontrado token!');
} else {

    //Si existe, debemos comprobar que el token recibido en  $_POST es
    //el que hemos almacenado en la variable de la sesión $_SESSION
    if (hash_equals($_POST['token'], $_SESSION['token']) === false) {
        print('El token no coincide!');
    } else {

        //El token es correcto y continúa el procesamiento con seguridad
        print('El token es correcto y podemos ejecutar acciones');


    }
}
```

La Sénia
Institut d'Educació Secundària
Paiporta

UD8. Programación Orientada a Objetos en PHP

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Clases en PHP.
- Clases y métodos abstractos
- Propiedades y métodos estáticos
- Operador de Resolución de Ámbito
- Herencia
- Interfaces
- Rasgos (Traits)
- Métodos mágicos
- Gestión de Excepciones

Clases en PHP

La clase establece las características, métodos y comportamiento de un objeto. El objeto es la instancia de una clase. Los métodos y atributos del objeto se acceden con **->** (**sin $**!!!)

```php
<?php   /*definición*/
class ClaseSencilla
{
    // Declaración de una propiedad
    public $var = 'valor1';

    // Declaración de un método
    public function mostrarVar() {
        echo $this->var;
    }
}
?>
```

```php
<?php   /*instanciación*/
$instancia = new ClaseSencilla();

// Esto también se puede hacer con una variable:
$nombreClase = 'ClaseSencilla';
$instancia = new $nombreClase();
?>
```

```php
<?php   /*invocación de métodos y propiedades*/
echo $instancia->var;    //devuelve valor1, el atributo var del objeto
$instancia->mostrarVar();    //invocamos al método y devuelve valor1
?>
```


Ver: https://www.php.net/manual/es/language.oop5.basic.php

Clases Abstractas

Una clase abstracta (creada con la palabra reservada **abstract**) es aquella que contiene métodos abstractos que sólo se declaran pero que se codifican en las clases descendientes.

Las clases abstractas no se pueden instanciar y sus descendientes están obligadas a implementar los métodos abstractos o a volver a definirlos como abstractos.

Si una clase tiene un método abstracto, la clase debe ser declarada como abstracta. Por otra parte, una clase abstracta puede contener métodos no abstractos.

Los métodos abstractos tienen dos requerimientos:

- No se pueden definir como privados puesto que necesitan ser heredados
- No pueden ser definidos como final porque deben ser redefinidos en las clases descendientes

Ver: https://www.php.net/manual/es/language.oop5.abstract.php

Clases Abstractas - Ejemplo

```
abstract class Database {
    abstract public function connect($server, $username, $password, $database);
    abstract public function query($sql);
    abstract public function fetch();
    abstract public function close();
}

class MySQL extends Database {
    protected $dbh;    // manejador de la BD
    protected $query;        // código SQL
    public function connect($server, $username, $password, $database) {
        $this->dbh = mysqli_connect($server, $username, $password, $database);
        // https://www.php.net/manual/es/function.mysqli-connect
    }
    public function query($sql) {
        $this->query = mysqli_query($this->dbh, $sql); // https://www.php.net/manual/es/mysqli.query.php
    }
    public function fetch() {
        return mysqli_fetch_row($this->dbh, $this->query); // https://www.php.net/manual/en/mysqli-result.fetch-row.php

    }
    public function close() {
        mysqli_close($this->dbh); // https://www.php.net/manual/es/mysqli.close.php
    }
}
```

Propiedades y métodos estáticos

Los métodos y propiedades **static** pueden ser invocados sin necesidad de instanciar objetos de la clase (muy útil para contadores, aplicar formatos, conversiones de medidas, etc.)
Por ello, se invocan usando **::** en lugar de **->** y en la misma clase se referencian con self (se refiere siempre a la clase en la que se crea) en lugar de usar $this (no existe referencia al objeto o instancia)

Nota: las propiedades static se invocan con **::** e incluyen el **$** de la variable.

```
<?php
class Format {
    public static $decimal=',';     // indica el formato para indicador de decimales
    public static $thousands='.';  // indica el formato para separador de miles
    public static function number($number, $decimals) {
        return number_format($number, $decimals, self::$decimal, self::$thousands);
    }
    public static function integer($number) {
        return self::number($number, 0); // se usa self en lugar de $this
    }
}
print Format::number(1234.567,2) . "\n"; // Los métodos static se invocan con :: y no usan ->
print Format::integer(1234.567) . "\n";
?>
```

```
/* resultado */
1.234,57
1.235
```

Ver: https://www.php.net/manual/es/language.oop5.static.php

Operador de Resolución de Ámbito ::

El operador de resolución de ámbito (Paamayim Nekudotayim) o el doble dos-puntos, es un token que permite acceder a elementos estáticos, constantes, y sobrescribir propiedades o métodos de una clase.
Para acceder a las constantes de una clase, se debe utilizar el nombre de la clase (o **self** si estamos dentro de la clase) y el operador de ámbito **::** como se muestra en el ejemplo:

```
class DB {
    const USUARIO = 'php';
    public function muestraUsuario() {
        echo self::USUARIO; // desde dentro de la clase }
    }

echo DB::USUARIO; // desde fuera de la clase
```


Ver: https://www.php.net/manual/es/language.oop5.paamayim-nekudotayim.php

Herencia

Para indicar que una subclase hereda desde la clase padre (herencia simple multinivel) se usa **extends**. Es necesario que la clase padre se defina antes que la subclase. Se heredan todos los métodos públicos y protegidos de la clase padre y se pueden sobrescribir los métodos heredados.

```
class forma {
    function dibujar() {  // pintar en pantalla }
}
class circulo extends forma {
    function dibujar($inicio, $radio) {
        // validar datos
        if ($radio > 0) {
            parent::dibujar();
            return true;
        }
        return false;
    }
}
```

Ver: https://www.php.net/manual/es/language.oop5.inheritance.php

Interfaces

Las interfaces permiten definir el comportamiento de los objetos mediante la definición de métodos que serán implementados en dichos objetos. Se utiliza la palabra reservada **implements**. Con class_implements() podemos saber si una clase implementa una interface concreta

```
interface NombreInterface {
    public function getNombre();      // Sólo se definen los métodos, no se implementan
    public function setNombre($nombre);
}

class Libro implements NombreInterface {
    private $nombre;
    public function getNombre() {     // Aquí es donde se implementan los métodos
        return $this->nombre;
    }
    public function setNombre($nombre) {
        return $this->nombre = $nombre;
    }
}
```

Ver: https://www.php.net/manual/es/language.oop5.interfaces.php

Rasgos (Traits)

Los rasgos o **traits** permiten reutilizar código entre objetos sin herencia. No se pueden instanciar y sus funciones se utilizan desde otras clases. La palabra reservada para incluir traits es **use**. Combinado los trait con interface se obtiene el máximo potencial de ambos. Una clase puede implementar la interface directamente o usar el trait que la implementa.

```
trait NombreTrait {
    private $nombre;
    public function getNombre() {
        return $this->nombre;
    }
    public function setNombre($nombre)
    {
        return $this->nombre = $nombre;
    }
}
```

```
/* instancia la clase y usa los métodos */
$l = new Libro();
$l->getNombre();
$l->setNombre("Mi libro");
```

```
/* usando únicamente trait en la clase */
class Libro {
    use NombreTrait;
    // Tiene los métodos getNombre y setNombre
}

/* combinando interface y trait en la clase*/
class Libro implements NombreInterface {
    use NombreTrait;
}
```


Ver: https://www.php.net/manual/es/language.oop5.traits.php

Rasgos (Traits) Múltiples

Ejemplo con múltiples traits:

```
trait Hola {
    public function saludo() {
        echo “Hola”;
    }
}
trait Adios {
    public function saludo() {
        echo “Adiós”;
    }
}
```

```
/* usamos los trait en la clase */
class MiSaludo {
    use Hola, Adios {
      // Ambos tienen el método saludo()
      // indicamos cuál queremos mostrar
      Hola::saludo insteadof Adios;
      Adios::saludo as despedida;
    }
}
```

Cuando varios traits comparten el nombre de los métodos, podemos elegir de qué trait vamos a usar el método con **insteadof**. También podemos usar los métodos de ambos traits indicando un alias con **as**

```
/* instanciamos la clase y usamos los métodos */
$misaludo = new MiSaludo();
$misaludo->saludo();  // imprime Hola
$misaludo->despedida(); // imprime Adiós
```


Ver: https://www.php.net/manual/es/language.oop5.traits.php

Métodos mágicos

Los métodos mágicos se invocan cuando ocurren determinados sucesos o eventos que los activan. Con nombre __**metodo()**, determinan cómo reaccionará el objeto ante dichos eventos o sucesos. A continuación veremos algunos métodos mágicos:

__**construct()**: permite inicializar el objeto al crearlo con **new**

__**destruct():** libera objetos sin referencias (unset) u otra finalización (exit, fin del script, etc.)

```
class usuario {
    public $nombreusuario;
    function __construct($nombreusuario, $password) {
        if ($this->validar_usuario($nombreusuario, $password)) {
            $this->nombreusuario = $nombreusuario;
        }
    }
    function __destruct() {
         echo “Ha finalizado su sesión”;
    }
}
```

```
// usamos constructor para crearlo con datos
$usuario = new usuario('Silvia', 'Dwes');
//El final del script invoca al destructor
```


Ver: https://www.php.net/manual/es/language.oop5.magic.php

Métodos mágicos II

__toString(): permite controlar cómo se muestra un objeto cuando se imprime. **Nota:** Si hay valores no string, hacer casting a string al devolverlo

```
class Persona {
    protected $nombre;
    protected $email;
    public function setNombre($nombre) {
        $this->nombre = $nombre;
    }
    public function setEmail($email) {
        $this->email = $email;
    }
    public function __toString() {
        return "$this->nombre <$this->email>";
    }
}
```

Ejecutamos:

```
$silvia = new Persona;
$silvia->setNombre('Silvia Vilar');
$silvia->setEmail('silvia@php.net');
print $silvia; // Muestra Silvia Vilar <silvia@php.net>
```

Métodos mágicos III

__get(): permite obtener propiedades private y protected del objeto
__set(): permite establecer propiedades private y protected del objeto
__isset(): invocado por isset() comprueba propiedad no public
__unset(): se invoca al usar la función unset() con propiedad no public

```
class Persona {
    private $vdatos = array();              //almacenaremos sus datos en un array
    public function __get($propiedad) {     //con get() no se puede acceder si no es public, __get sí puede
        if (isset($this->vdatos[$propiedad])) {
            return $this->vdatos[$propiedad];
        } else {
            return false;
        }
    }
    public function __set($propiedad, $valor) {    //con set() no se puede acceder si no es public, __set sí
        $this->vdatos[$propiedad] = $valor;
    }
    public function __isset($property) { //permite comprobar propiedades private o protected
        return isset($this->data[$property]);
    }
    public function __unset($property) {
        if (isset($this->data[$property])) {
            unset($this->data[$property]);
        }
    }
}
```

```
$silvia = new Persona;
$silvia->email = 'silvia@php.net'; // llama a
__set(email, 'silvia@php.net')
print $silvia->email; // llama a __get(email) y muestra
silvia@php.net
```

Métodos mágicos III-bis

Hay quien recomienda no usar los métodos mágicos __get y __set y usar en su lugar funciones para implementar get y set de las propiedades del objeto. Algunas razones son:

- Estos métodos sólo se ejecutan cuando falla el acceso a propiedades no existentes o inaccesibles (acceden a private o protected).
- Sin validación (property_exists o isset), se puede modificar la estructura del objeto en tiempo de ejecución agregando propiedades.
- Son más lentos que los métodos getX()/setX().
- Se requiere documentar explícitamente el uso de dichas funciones ya que imposibilita la documentación automática con herramientas como phpDocumentor. Además los IDE tienen dificultades para identificarlos y sugerir código (phpIntellisense).
- No se pueden utilizar con propiedades static.

Métodos mágicos IV

__call(): Permite que varios objetos se comporten como uno solo
__callStatic(): Realiza la funció de __call() con métodos estáticos

```
class Persona {
    protected $nombre;
    protected $direccion;
    public function __construct() {
        $this->direccion = new Direccion;
    }
    public function setNombre($nombre) {
        $this->nombre = $nombre;
    }
    public function getNombre() {
        return $this->nombre;
    }
    public function __call($metodo, $argumentos) {
        if (method_exists($this->direccion, $metodo)) { //method_exists($object,$method_name ):bool
            return call_user_func_array(array($this->direccion, $metodo), $argumentos);
        }
    }
}
```

```
class Direccion {
    protected $ciudad;
    public function setCiudad($ciudad) {
        $this->ciudad = $ciudad;
    }
    public function getCiudad() {
        return $this->ciudad;
    }
}
```

```
$silvia = new Persona;
$silvia->setNombre('Silvia Vilar');
$silvia->setCiudad('Valencia');  //llama a Direccion::setCiudad('Valencia') con __call
print $silvia->getNombre() . ' vive en ' . $silvia->getCiudad() . '.';
```

Métodos mágicos V

__sleep(): se invoca al serializar un objeto para almacenarlo como string. Se almacenan variables y nombre de clase pero no métodos

__wakeup(): se invoca al recuperar un objeto serializado y recupera conexiones de BD u otras tareas de reinicialización

```
class LogFile {
    protected $filename;
    protected $handle; // almacenará el puntero al fichero
    public function __construct($filename) {
        $this->filename = $filename; // Nombramos el fichero
        $this->open(); // Lo abrimos para trabajar con él
    }
    private function open() {
        $this->handle = fopen($this->filename, 'a'); // apertura para sólo escritura
    }
    public function __destruct($filename) {
        fclose($this->handle);
    }
    public function __sleep() { // invocado con serialized, obtendría array con propiedades a serializar
    return array('filename');
    }
    public function __wakeUp() { // invocado con unserialized, recupera el handle y nombre del
    $this->open();                fichero de las propiedades del objeto ya que no recibe argumentos
    }
}
```

Ver: https://www.php.net/manual/es/language.oop5.serialization.php

Gestión de Excepciones

Cuando se genera un error o un comportamiento inesperado en un script de PHP, se produce un Error o una Exception. Los errores pueden ser lanzados por múltiples funciones y clases de PHP y las excepciones son lanzadas por las funciones y clases definidas por el usuario y las excepciones de PHP.

La gestión de dichos errores y excepciones es el modo adecuado de tratar los errores, asegurar la consistencia de los datos y mostrar información útil al usuario para que pueda evitar volver a generar el error. Por otra parte, si no se maneja el error lanzado por PHP, éste provoca que el script se detenga

Para tratar Error y Exception, recurrimos al bloque **try**… **catch** con la posibilidad de lanzar la excepción a otro gestor con **throw**

IMPORTANTE: La jerarquía de Error no hereda de Exception con lo que deberemos dedicar un bloque catch (Exception $e) para las excepciones y un bloque catch(Error $e) para manejar los errores. Sin embargo, podemos acceder a ambas mediante la interfaz predefinida **Throwable**

Ver: https://www.php.net/manual/es/language.errors.php7.php

Throwable

La interfaz Throwable no puede ser implementada directamente (muestra un error), de modo que será necesario implementar una interfaz que herede de Throwable y crear una clase que herede de Exception e implemente los métodos de la interfaz de usuario que acabamos de crear:

```
Interface MiInterfazThrowable extends Throwable {
    //
}

class MiExcepcion extends Exception implements MiInterfazThrowable {
    //
}

/* Usamos MiExcepcion para capturar cualquier error o excepción */
try {
    Throw new MiExcepcion("System Error");
} catch (MiExcepcion $e) {
    echo 'Excepcion: '. $e->getMessage;
} finally {
    echo 'Acciones de Finally';
}
```

Nota: Cualquier catch(Error $e) o catch(Exception $e) no será accesible si antes está catch(Throwable $e) ya que Throwable trata tanto Error como Exception

Ver: https://www.php.net/manual/es/class.throwable.php

Ejemplos Gestión Excepciones

```
/* Ejemplo con Throwable */
class Mailer {
      private $transport;
      public function __construct(Transport
      $transport)
      {
            $this->transport = $transport;
      }
}
$transport = new stdClass();
//generará error porque no es de clase Transport

/* Usamos Throwable, interfaz para capturar Error y
Exception en el mismo bloque Catch */
try {
      $mailer = new Mailer($transport);
} catch (Throwable $e) {
      echo ‘Caught!’;
} finally {
      echo ‘Cleanup!’;
}
```

```
Try { /* Ejemplo con Exception */
    throw new Exception('Hello world');
} catch (Exception $e) {
    echo 'Exception: '. $e->getMessage();
}
```

Podemos comparar la ejecución con Error y Throwable:

```
/*con clase Error*/
try {
  $resultado = 4 / 0;
} catch (DivisionByZeroError $e) {
    echo $e->getMessage(), "\n";
}
```

```
/* con Throwable */
try {
  $resultado = 4 / 0;
} catch (Throwable $e) {
    echo $e->getMessage(), "\n";
}
```

La Sénia
Institut d'Educació Secundària
Paiporta

UD9. Utilización de Técnicas de Acceso a Datos

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Utilización de bases de datos relacionales.
- API de PHP para MySQL.
- Establecimiento de conexiones.
- Ejecución de sentencias SQL (DML).
- Consultas preparadas.
- Obtención de resultados.
- Uso de conjuntos de resultados.
- Otros orígenes de datos. Ficheros. XML. JSON
- Buenas prácticas

Utilización de BD relacionales

PHP tiene un soporte muy amplio para BD. Puede interactuar con cualquier base de datos, ya sea relacional o no. Incluso soporta ODBC que le permite acceder a BD de las que no tiene extensiones.

Las extensiones disponibles para MySQL son las siguientes:

- PHP Data Objects (PDO): Cada BD requiere el uso de un controlador de interfaz PDO específico para dicha BD. Por ejemplo, para MySQL es **PDO_MYSQL**.
- Extensión de BD específica del proveedor: que en el caso de MySQL se debe usar **mysqli**.

Como recordatorio, las DBR se estructuran en tablas relacionadas, cada una de ellas representada por tuplas o filas con los datos y columnas, campos o atributos que dan nombre al dato almacenado en la celda

Ver: https://www.php.net/manual/es/refs.database.php

Comparación API de MySQL (mysqli vs pdo)

Para probar la conexión devolveremos el primer DNI de la tabla clientes, el 00371569G

```
<?php
// conexión usando mysqli
try {
    $mysqli = new mysqli("localhost:33006",USERNAME, PASSWORD,"EMPRESA");
    $resultado = $mysqli->query("SELECT DNI AS ID_CLIENTE FROM CLIENTE");
    $fila = $resultado->fetch_assoc();
    echo "El ID de Cliente (mysqli) es " . htmlentities($fila['ID_CLIENTE'] . "<br>\n");
} catch (mysqli_sql_exception $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$mysqli->close();
```

El ID de Cliente (mysqli) es 00371569G
El ID de Cliente (pdo) es 00371569G

```
// conexión usando PDO
try {
    $pdo = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME, PASSWORD);
    $resultado = $pdo->query("SELECT DNI AS ID_CLIENTE FROM CLIENTE");
    $fila = $resultado->fetch(PDO::FETCH_ASSOC);
    echo "El ID de Cliente (pdo) es " . htmlentities($fila['ID_CLIENTE'] . "<br>\n");
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$pdo = null;
?>
```


Ver: https://www.php.net/manual/es/mysqlinfo.api.choosing.php

Elección API de MySQL

- La mayoría desarrolladores se decantan por la API **PDO** ya que la razón principal que aducen es que, gracias a los drivers que soporta, permite acceder a diversas bases de datos (12 en concreto) en lugar de tener que usar las API específicas de los proveedores lo que simplificaría un hipotético cambio de proveedor de la BD (Por ejemplo migrar de Oracle a MySQL)
- Ambas ofrecen una API orientada a objetos aunque msqli tiene interfaz procedimental que puede usarse con programación estructurada
- PDO ofrece mayor seguridad ante ataques de inyección de SQL ya que permite ejecutar sentencias preparadas en el cliente de modo que el cliente sólo introduce los parámetros y no crea la sentencia. Ver: https://www.php.net/manual/es/pdo.prepared-statements.php
- En este enlace podéis ver una comparativa de varios casos de uso de PDO y mysqli muy interesante: https://websitebeaver.com/php-pdo-vs-mysqli

En este tema se usará PDO_MYSQL para los ejemplos y ejercicios

Ver clase PDO: https://www.php.net/manual/es/class.pdo.php

Establecimiento de conexiones

Las conexiones se establecen creando una instancia de la clase PDO independientemente del driver que deseemos usar. Debemos indicar el nombre del origen de datos o DSN (DataBase Source Name) que es como se muestra:

mysql:host=198.0.4.221**;port**=3306**;dbname**=pruebaBD

//

mysql:host=127.0.0.1**:**3306**;dbname**=pruebaBD

//

mysql:host=bd.ejemplo.com**;dbname**=pruebaBD

Ver: https://www.php.net/manual/es/ref.pdo-mysql.connection.php

Además del DSN, podemos pasar el usuario y la contraseña de forma opcional y capturar cualquier error en la conexión:

```
<?php
try{
    $mbd = new PDO('mysql:host=localhost;dbname=pruebaBD', 'usuario', 'contraseña');
} catch (PDOException $e) {   //Ver: https://www.php.net/manual/en/class.pdoexception.php
        print "Error al conectar con la BD: " .  $e->getMessage();
}
?>
```


Ver: https://www.php.net/manual/es/pdo.connections.php

Atributos de la conexión

Las conexiones permiten establecer atributos de la misma con el método de PDO setAttribute. Por ejemplo, veamos las alternativas en cuanto a los valores de ***informe de Errores*** de **PDO::ATTR_ERRMODE**

- Modo Silencioso (por defecto): Si el método ejecutado en try falla en su ejecución, devuelve el valor false. Se debe comprobar el valor devuelto por el método y usar el método errorInfo() de PDO para obtener detalles del problema. El valor es **PDO::ERRMODE_SILENT**
- Mostrar Advertencias: En este modo las funciones se comportan como en el modo silencio (sin lanzar excepciones y devolviendo false al producirse un error) pero el motor PHP genera un mensaje de advertencia o warning. Dependiendo de cómo está configurado el manejo de errores, este mensaje puede mostrarse por pantalla o en un fichero de log. El valor es **PDO::ERRMODE_WARNING**
- Mostrar Excepciones: En este modo, se lanzan excepciones para que puedan ser gestionadas antes de mostrar un error o bien se detalle información del mismo. El valor es **PDO::ERRMODE_EXCEPTION**

Ver: https://www.php.net/manual/en/pdo.setattribute.php

Atributos de la conexión – Modo Silencioso

```
// El constructor siempre lanza una excepción si falla
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: "  . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
$result = $db->exec("INSERT INTO CLIENTE (DNI, NOMBRE, APELLIDOS, TIPO)
    VALUES ('11111111A','Ana', 'Acosta', 0)"); // TIPO no es columna de la tabla
if (false === $result) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
    /* El primer elemento de $error es el código de error SQLSTATE (standard en las
BD). El segundo elemento es el código específico del motor de BD (mysql). El tercer
elemento es el mensaje textual describiendo el error concreto */
}
```

/* resultado */

```
No se ha podido realizar la inserción!
SQL Error=42S22, DB Error=1054, Message=Unknown column 'TIPO' in 'field list'
```

Atributos de la conexión – Modo Advertencia

```
try {// El constructor siempre lanza una excepción si falla
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
$result = $db->exec("INSERT INTO CLIENTE (DNI, NOMBRE, APELLIDOS, TIPO)
    VALUES ('11111111A','Ana', 'Acosta', 0)"); // TIPO no es columna de la tabla
if (false === $result) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
    /* El primer elemento de $error es el código de error SQLSTATE (standard en las
BD). El segundo elemento es el código específico del motor de BD (mysql). El tercer
elemento es el mensaje textual describiendo el error concreto */
}
```

/* resultado */
PHP Warning: PDO::exec(): SQLSTATE[42S22]: Column not found: 1054 Unknown column 'TIPO' in 'field list' in C:\Users\Silvia\DWES\UD9_ADD\atributosConexion.php on line 30
No se ha podido realizar la inserción!
SQL Error=42S22, DB Error=1054, Message=Unknown column 'TIPO' in 'field list'

Atributos de la conexión – Modo Excepción

```
try { // El constructor siempre lanza una excepción si falla
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$result = $db->exec("INSERT INTO CLIENTE (DNI, NOMBRE, APELLIDOS, TIPO)
    VALUES ('11111111A','Ana', 'Acosta', 0)"); // TIPO no es columna de la tabla
if (false === $result) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
    /* $error[0]=SQLSTATE,  $error[1]=código mysql, $error[2]=error concreto */
}
```

/* resultado */

PHP Fatal error: Uncaught PDOException: SQLSTATE[42S22]: Column not found: 1054 Unknown column 'TIPO' in 'field list' in C:\Users\Silvia\DWES\UD9_ADD\atributosConexion.php:46
Stack trace:
#0 C:\Users\Silvia\Documents\2DWES\DWES\UD9_ADD\EjemplosUD9_AD\atributosConexion.php(46): PDO->exec()
#1 {main}
  thrown in C:\Users\Silvia\DWES\UD9_ADD\atributosConexion.php on line 46 **(Puede no mostrarse por pantalla dependiendo de php.ini)**

Nota: La visualización de Excepciones y Warnings depende de la configuración de error_reporting. Con Debug, sí podéis comprobar estos errores en cualquier caso

Ver configuración informe errores: https://www.php.net/manual/en/function.error-reporting

Ejecución de sentencias SQL

- Como se ha introducido en los ejemplos anteriores, para poder ejecutar las sentencias SQL (normalmente DML: INSERT, UPDATE, DELETE y SELECT), hacemos uso del método **exec()** de la instancia PDO.
- Dicho método exec() ejecuta la sentencia SQL ***devolviendo el número de filas afectadas por la sentencia***, no devuelve el resultado de la sentencia SQL ejecutada. Existe el método query() que sí devuelve el resultado de la sentencia SQL (ver https://www.php.net/manual/es/pdo.query.php)
- Las sentencias DDL y DCL no deben poder ser ejecutadas por los desarrolladores, deben ser ejecutadas (siempre de forma local en la máquina en la que reside el servidor de la BD) por los Administradores de la BD (DBA).

A continuación se verán ejemplos de ejecución de estas sentencias DML de SQL.

Ver: https://www.php.net/manual/es/pdo.exec

INSERT

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
//Ejemplo INSERT (Se deben haber insertado proveedores con anterioridad)
$affectedRows = $db->exec("INSERT INTO PRODUCTO (COD_PROD, NOMBRE,
PROVEEDOR,PVP)
        VALUES ('P0001', 'MONITOR','A12345678', 200.50),
        ('P0002', 'TECLADO','A12345678',25.49),
        ('P0003', 'RATÓN','A12345678', 15)");
if (false === $affectedRows) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la inserción!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
} else {
    print "Se han insertado " . $affectedRows . " filas.\n";
}
```

/* resultado */
Se han insertado 3 filas.

Ver: https://www.php.net/manual/en/pdo.exec

UPDATE

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}

//Ejemplo UPDATE Incrementar 10% el PVP de los productos que sea inferior a 50,5€
$affectedRows = $db->exec("UPDATE PRODUCTO SET PVP=(PVP*1.10)
      WHERE PVP < 50.50;");
if (false === $affectedRows) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar la actualización!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
} else {
    print "Se han actualizado " . $affectedRows . " filas.";
}
```

/* resultado */
Se han actualizado 2 filas.

DELETE

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME,
PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}

//Ejemplo DELETE: Eliminar los productos con precio mayor de 200€
$affectedRows = $db->exec("DELETE FROM PRODUCTO WHERE PVP>200;");
if (false === $affectedRows) { // Comprobamos si da falso el método usado
    $error = $db->errorInfo();
    print "No se ha podido realizar el borrado!\n";
    print "SQL Error={$error[0]}, DB Error={$error[1]}, Message={$error[2]}\n";
} else {
    print "Se han eliminado " . $affectedRows . " filas.";
}
```

/* resultado */
Se han eliminado 1 filas.

SELECT

```
try {
    $db = new PDO('mysql:host=localhost:33006;dbname=EMPRESA', USERNAME, PASSWORD);
} catch (PDOException $e) {
    print "No se ha podido realizar la conexión: " . $e->getMessage();
}
//Ejemplo SELECT Obtener los productos con PVP menor de 50€
$result = $db->query("SELECT * FROM PRODUCTO WHERE PVP<50;");
//necesitamos saber el número de filas afectadas en $numRows
$numRows = $db->query("SELECT COUNT(*) FROM PRODUCTO WHERE PVP<50;")-
>fetchColumn();
if ($numRows) {
    print "Se han obtenido " . $numRows . " filas.\n";
    print "El resultado de la consulta es: \n";
    printf("%-10s%-15s%-12s%-7s\n", "CODIGO", "NOMBRE", "PROVEEDOR", "PVP");
    printf("%-10s%-15s%-12s%-7s\n", "------", "------", "---------", "-----");
    foreach ($result->fetchAll() as $row) {
        printf("%-10s%-15s%-12s%-7s\n", $row['COD_PROD'], $row['NOMBRE'] ,
            $row['PROVEEDOR'], $row['PVP']);
    }
} else {
    print "No se han obtenido resultados a mostar!\n";
}
```

/* resultado */

Se han obtenido 2 filas.
El resultado de la consulta es:

CODIGO	NOMBRE	PROVEEDOR	PVP
P0002	TECLADO	A12345678	28.04
P0003	RATÓN	A12345678	16.50

Ver: https://www.php.net/manual/es/pdostatement.fetchcolumn

Consultas preparadas

Las consultas preparadas pueden ser ejecutadas múltiples veces con parámetros distintos y previenen la inyección de SQL en los formularios de forma que un usuario malintencionada pueda cambiar la estructura de tablas, eliminar datos, borrar tablas, etc. Para ello se construye la consulta parametrizando los atributos a tratar de modo que no se pueda ejecutar una sentencia SQL en un atributo ya que añade comillas automáticamente a los parámetros. Dicha sentencia, se crea con el método **prepare()** de PDO indicando con **?** o **:atributo** los atributos cuyo valor espera y se ejecuta con el método **execution()** de la instancia PDOStatement aportando únicamente los valores de los atributos esperados en un array.

Ver: https://www.php.net/manual/es/pdo.prepare.php y https://www.php.net/manual/es/pdostatement.execute.php

/*Ejemplo de SENTENCIA PREPARADA*/

```
<?php
require_once __DIR__.'\..\db.php';
try {
    $db = new PDO("mysql:host=127.0.0.1:33006;dbname=EMPRESA", USERNAME, PASSWORD);
} catch (PDOException $e) {
    die("Conexión fallida: " . $e->getMessage());
}
if (isset($_POST['btnEnviar'])) {
    $SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=?;"); //También
podemos usar PROVEEDOR=:proveedor
    if (isset($_POST['proveedor'])){
        $SQLstring->execute(array($_POST['proveedor']));//También se puede usar BindParam
        $result=$SQLstring->fetchAll(); //Obtenemos los productos
        $SQLCount=$db->prepare("SELECT COUNT(*) FROM PRODUCTO WHERE PROVEEDOR=?;");
        $SQLCount->execute(array($_POST['proveedor'])); //También se puede usar BindParam
        $numRows=$SQLCount->fetchColumn(); //obtenemos el número de filas devueltas
        if($numRows){
            print "Se han obtenido " . $numRows . " filas." . "<br>";
            print "<h2>Productos del proveedor ".$_POST['proveedor']."</h2><br>";
            print "<table border='1'>";
            print "<tr><th>COD_PROD</th><th>NOMBRE</th><th>PROVEEDOR</th><th>PVP</th></tr>";
            foreach ($result as $row){
                print "<tr><td>$row[COD_PROD]</td><td>$row[NOMBRE]</td><td>$row[PROVEEDOR]</td><td>$row[PVP]</td></tr>";
            }
            print "</table>";
        } else {
            print "No se han obtenido resultados a mostar!\n";
        }
    } else{
        print "<p>introduce un proveedor</p>";
    }
}     ?>
```

/*Ejemplo de SENTENCIA PREPARADA*/

```
<html>
<head>
    <meta charset="UTF-8">
    <title>Consulta de productos por proveedor</title>
</head>
<body>
    <form action="sentenciaPreparada.php" method="post">
        <h1>Consulta de productos por proveedor</h1>
        <label for="proveedor">Proveedor:</label>
        <select name="proveedor" id="proveedor">
            <?php
            $SQLstring = $db->prepare("SELECT DISTINCT PROVEEDOR FROM PRODUCTO;");
            $SQLstring->execute();
            $result = $SQLstring->fetchAll();
            foreach ($result as $row) {
                print "<option value='$row[PROVEEDOR]'>$row[PROVEEDOR]</option>";
            }
            ?>
            <input type="submit" name="btnEnviar" value="Enviar">
    </form>
</body>
```

/*Ejemplo de SENTENCIA PREPARADA*/

Se han obtenido 3 filas.

Productos del proveedor T98723467

COD_PROD	NOMBRE	PROVEEDOR	PVP
P0004	ALTAVOCES	T98723467	15.50
P0006	MINI PC	T98723467	200.99
P0009	PC PORTÁTIL	T98723467	750.99

Consulta de productos por proveedor

Proveedor: A12345678 Enviar

Obtención de resultados

En el ejemplo, los valores de los parámetros de la consulta SQL los hemos pasado usando un vector en el método execute (método lazy).

```
$SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=?;");
$SQLstring->execute(array($_POST['proveedor']))
```

Otro modo de pasar los valores a los parámetros es usando el método PDOStatement::bindParam() [o bindValue] y luego llamar a execute sin parámetros:

```
$SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=?;");
$SQLstring->BindParam(1,$_POST['proveedor'],PDO::PARAM_STR); //1-posición param.
$SQLstring->execute();
// O también si usamos parámetro con :nombre y no con ?
$SQLstring=$db->prepare("SELECT * FROM PRODUCTO WHERE PROVEEDOR=:proveedor;");
$SQLstring->BindParam(:proveedor,$_POST['proveedor'],PDO::PARAM_STR);
$SQLstring->execute();
```

Ver: https://www.php.net/manual/es/pdostatement.bindparam y https://www.php.net/manual/es/pdostatement.bindvalue.php

Uso de conjuntos de resultados

El conjunto de tuplas resultado se obtiene con fetch(). En el ejemplo con fetchAll():

```
array (size=3)
  0 =>
array (size=8)
      'COD_PROD' => string 'P0004' (length=5)
      0 => string 'P0004' (length=5)
      'NOMBRE' => string 'ALTAVOCES' (length=9)
      1 => string 'ALTAVOCES' (length=9)
      'PROVEEDOR' => string 'T98723467' (length=9)
      2 => string 'T98723467' (length=9)
      'PVP' => string '15.50' (length=5)
      3 => string '15.50' (length=5)
  1 =>
    array (size=8)
      'COD_PROD' => string 'P0006' (length=5)
      0 => string 'P0006' (length=5)
      'NOMBRE' => string 'MINI PC' (length=7)
      1 => string 'MINI PC' (length=7)
      'PROVEEDOR' => string 'T98723467' (length=9)
      2 => string 'T98723467' (length=9)
      'PVP' => string '200.99' (length=6)
      3 => string '200.99' (length=6)
  2 =>
    array (size=8)
      'COD_PROD' => string 'P0009' (length=5)
      0 => string 'P0009' (length=5)
      'NOMBRE' => string 'PC PORTÁTIL' (length=12)
      1 => string 'PC PORTÁTIL' (length=12)
      'PROVEEDOR' => string 'T98723467' (length=9)
      2 => string 'T98723467' (length=9)
      'PVP' => string '750.99' (length=6)
      3 => string '750.99' (length=6)
```

fetchAll() obtiene todas las filas completas del resultado de la consulta.

fetchColumn(): Devuelve sólo la columna indicada en el índice (0 por defecto, la primera) de la fila siguiente.

Ver: https://www.php.net/manual/es/pdostatement.fetch

Otros orígenes de datos - Ficheros

Las operaciones sobre ficheros suelen constar de tres fases:

1) Apertura del fichero Ver: https://www.php.net/manual/es/function.fopen.php

Se abre el fichero, indicando el modo (lectura, escritura o ambas). La función fopen() devuelve un descriptor de fichero en caso de éxito o false en otro caso.

```
<?php
    $df = fopen("c:\\folder\\resource.gif", "r"); //en Windows
    $df = fopen("/home/silvia/fichero.txt", "wb"); //sistemas Linux
    $df = fopen("http://www.example.com/", "r"); //fichero en red
?>
```

2) Procesamiento del fichero

Se realizarán las operaciones de lectura, escritura o ambas según permita la apertura del archivo. Tras finalizar, se debe liberar el descriptor para desbloquear el archivo

3) Cierre del fichero Ver: https://www.php.net/manual/es/function.fclose.php

Se cierra el fichero, devolviendo true en caso de éxito o false en caso de error

```
<?php
    fclose($df);
?>
```

Ficheros completos en una vez I

Leer y escribir un fichero completo a la vez: **file_get_contents() (lectura como string) y file_put_contents() (escritura)**

```
//Lectura del archivo plantilla.html obtenido completamente en una variable
$page = file_get_contents('plantilla.html'); //Obtiene el fichero html como string
$page = str_replace('{page_title}', 'Bienvenida', $page); // Indica título de la página
if (date('H' >= 12)) { //color de la página según si es por la mañana o la tarde
    $page = str_replace('{color}', 'blue', $page); // Azul para la mañana
} else {
    $page = str_replace('{color}', 'green', $page); //Verde para la tarde
}
// Escribimos la plantilla personalizada en un archivo llamado paginaSaludo.html
file_put_contents('paginaSaludo.html', $page);
```

Nota: en las plantillas HTML las variables entre llaves como {variable} son procesadas por php con el valor indicado en el script

fichero plantilla.html

```
<html>
    <head><title>{page_title}</title></head>
    <body bgcolor="{color}">
        <h1>Hola, eres bienvenido/a!!</h1>
    </body>
</html>
```

Ver: https://www.php.net/manual/es/function.file-get-contents.php
y https://www.php.net/manual/es/function.file-put-contents.php

Ficheros completos en una vez II

Obtener un fichero completo a la vez en un **array** donde cada elemento es una fila del fichero: **file()**

Las opciones disponibles en flags son:

FILE_USE_INCLUDE_PATH: Buscar el fichero en include_path

FILE_IGNORE_NEW_LINES: Omitir nueva línea al final de cada elemento del array

FILE_SKIP_EMPTY_LINES: Saltar las líneas vacías

```
//Obtenemos un fichero en un vector y recorremos los elementos del mismo para obtener cada una de las lineas del fichero e imprimir una lista de usuarios con su email
$vfile=file('usuarios.txt');
foreach ($vfile as $line) {
    $line = trim($line);
    $info = explode('-', $line);
    print '<li><b>' . $info[0] . '</b> tiene el email <b>' . $info[1] ."</b></li>\n";
}
```

fichero usuarios.txt

Silvia Vilar - silvia.vilar@example.com
Ana Andrés - ana.andres@example.com
Berto Barea - berto.barea@example.com
Carlos Calvo - carlos.calvo@example.com
David Díaz - david.diaz@example.com

- **Silvia Vilar** tiene el email **silvia.vilar@example.com**
- **Ana Andrés** tiene el email **ana.andres@example.com**
- **Berto Barea** tiene el email **berto.barea@example.com**
- **Carlos Calvo** tiene el email **carlos.calvo@example.com**
- **David Díaz** tiene el email **david.diaz@example.com**

Nota: resultado mostrado en el navegador

Ver: https://www.php.net/manual/es/function.file.php

Ficheros de forma parcial I

En este caso debemos llamar a fopen() ya que necesitamos el descriptor del fichero para operar con él. Veamos distintas funciones:

Función fgets(): obtiene la línea de fichero hasta alcanzar $length o eof

```
$df = fopen("usuarios.txt", "r");
while (!feof($df)){ //Mientras no alcancemos el final del archivo
    $linea = fgets($df); // $length es parámetro opcional, por defecto hasta EOF
    echo "USUARIO ",$linea, "</br>";
}
fclose($df);
```

Función fread(): obtiene conjunto de datos de ficheros binarios del tamaño indicado.Se suele indicar $length (filesize() para todo el fichero)

```
$df = fopen("usuarios.txt", "rb");
$datos = fread($df,filesize("usuarios.txt")); // leemos el fichero entero con filesize
var_dump($datos);
fclose($df);
```

fichero usuarios.txt

Silvia Vilar, silvia.vilar@example.com
Ana Andrés, ana.andres@example.com
Berto Barea, berto.barea@example.com
Carlos Calvo, carlos.calvo@example.com
David Díaz, david.diaz@example.com

Ver:
https://www.php.net/manual/es/function.fgets.php
https://www.php.net/manual/es/function.fread.php

Ficheros de forma parcial II

Función fscanf(): obtiene la línea de fichero con un formato dado

```
$df = fopen("usuarios.txt", "r");
while ($usuarioinfo = fscanf($df, "%s\t%s\t%s\t%s")){
    list($usuario, $email) = $usuarioinfo;
    echo $usuarioinfo[0] . " ".$usuarioinfo[1] . " tiene el email $usuarioinfo[3] <br>";
}
fclose($df);
```

Función fseek(): lee desde la posición exacta que se le indique

```
$df = fopen("usuarios.txt", "r");
fseek($df, 40); //comenzamos a leer en el inicio de la segunda línea (segundo usuario)
while(!feof($df)){
    $linea = fgets($df);
    echo $linea, "</br>";
}
fclose($df);
```

fichero usuarios.txt

Silvia Vilar, silvia.vilar@example.com
Ana Andrés, ana.andres@example.com
Berto Barea, berto.barea@example.com
Carlos Calvo, carlos.calvo@example.com
David Díaz, david.diaz@example.com

Ver:
https://www.php.net/manual/es/function.fscanf.php
https://www.php.net/manual/es/function.fseek.php

Otros orígenes de datos – Ficheros CSV

Función fgetcsv(): Busca campos csv en el fichero y devuelve un array con ellos

```
$numfila = 1; // la primera fila es el encabezado
if(($df = fopen("usuarios.csv", "r")) !== FALSE) {
    while (($fila = fgetcsv($df,0,",")) !== FALSE) { // 0 es sin límite de longitud
        if ($numfila>1){ //La primera fila son los nombres de campos del CSV
            echo "\nEl usuario $fila[0] tiene el email $fila[1] <br>";
        }
        $numfila++;
    }
fclose($df);
}
```

```
El usuario Silvia Vilar tiene el email silvia.vilar@example.com
El usuario Ana Andrés tiene el email ana.andres@example.com
El usuario Berto Barea tiene el email berto.barea@example.com
El usuario Carlos Calvo tiene el email carlos.calvo@example.com
El usuario David Díaz tiene el email david.diaz@example.com
```

fichero usuarios.csv

```
Nombre, Email
Silvia Vilar, silvia.vilar@example.com
Ana Andrés, ana.andres@example.com
Berto Barea, berto.barea@example.com
Carlos Calvo, carlos.calvo@example.com
David Díaz, david.diaz@example.com
```


Ver: https://www.php.net/manual/es/function.fgetcsv.php

Otros orígenes de datos - XML

Tenemos diversas maneras de obtener documentos XML en PHP (se requiere la extensión libxml, habilitada por defecto) como por ejemplo:

- **Parser** o Analizador de **XML**: Permite analizar documentos XML (pero no validarlos, si el documento no está bien formado, da error). Es más rápido porque no carga todo el documento en una vez, analiza nodo por nodo. Ver: https://www.php.net/manual/es/book.xml.php
- Usando la API **DOM** (Document Model Object) de PHP: Se basa en análisis en árbol cargando el documento XML entero, Permite construir, modificar, consultar, validar y transformar documentos XML. Ver: https://www.php.net/manual/es/book.dom.php
- Usando **SimpleXML**: SimpleXML permite añadir, modificar, comparar elementos, usar Xpath, etc. de forma rápida y fácil. En pocas ocasiones para algo un poco más complejo puede ser necesario usar PHP DOM. Existen dos funciones para pasar de uno a otro: de un nodo DOM a un objeto SimpleXML **_simplexml_importdom()** y de un objeto SimpleXML a un nodo DOM **_dom_importsimplexml()**.Ver: https://www.php.net/manual/es/book.simplexml.php

Usuarios.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE usuarios SYSTEM "usuarios.dtd"> //Esta línea sólo es necesaria para DOM
<usuarios>
   <usuario>
      <nombre>Silvia</nombre>
      <apellido>Vilar</apellido>
      <direccion>Calle Mar 12</direccion>
      <ciudad>Valencia</ciudad>
      <pais>España</pais>
      <contacto>
         <telefono>12345678</telefono>
         <url>http://silvia.ejemplo.com</url>
         <email>silvia@ejemplo.com</email>
      </contacto>
   </usuario>
   <usuario>
      <nombre>Paco</nombre>
      <apellido>Ruiz</apellido>
      <direccion>Calle La Fuente 22</direccion>
      <ciudad>Alicante</ciudad>
      <pais>España</pais>
      <contacto>
         <telefono>23456789</telefono>
         <url>http://paco.ejemplo.com</url>
         <email>paco@ejemplo.com</email>
      </contacto>
   </usuario>
</usuarios>
```

Ejemplo de XML Parser

EjemploXMLParser.php

```
<?php
include “funcionesParser.php”;
$parser = xml_parser_create();
// Especificar el handler de elementos
xml_set_element_handler($parser,"start","stop");
// Especificar el handler de datos
xml_set_character_data_handler($parser,"char");
// Abrir un archivo xml
$df = fopen("usuarios.xml","r");
// Leer los datos
while ($data = fread($df,4096)) {
    xml_parse($parser,$data,feof($df)) or
    die (sprintf("Error XML: %s en la línea %d",
        xml_error_string(xml_get_error_code($parser)),
        xml_get_current_line_number($parser)));
}
// Liberar el analizador
xml_parser_free($parser);
?>
```

Nota: Esta forma sólo se usa en casos donde se requiera analizar el XML con rapidez y sin cargar el documento completo, aunque en este caso también se recomienda XMLReader

Ver: https://www.php.net/manual/es/book.xml.php

/*funcionesParser.php*/

```
// Función a utilizar al inicio de un elemento:
function start($parser,$elemento,$atributos) {
    switch($elemento) {
      case "USUARIO":
        echo "Datos de usuario: <br>";
        break;
      case "NOMBRE":
        echo "Nombre: ";
        break;
      case "APELLIDO":
        echo "Apellido: ";
        break;
      case "CIUDAD":
        echo "Ciudad: ";
        break;
      case "PAIS":
        echo "País: ";
            break;
      case "TELEFONO":
        echo "Teléfono: ";
        break;
      case "URL":
        echo "URL: ";
        break;
      case "EMAIL":
        echo "Email: ";
    }
}
// Función para el final de un elemento:
function stop($parser,$elemento) {
    echo "<br>";
}
// Función al encontrar un carácter
function char($parser,$data) {
    echo $data;
}
```

Ejemplo de XML DOM

```
<?php
$dom = new DOMDocument; //Instanciamos el DOM
$dom->load('usuarios.xml'); //Cargamos el fichero a analizar
$listausuarios=$dom->getElementsByTagname('usuario');
$numusuarios=$listausuarios->length;
echo "Se van a listar los datos de un total de $numusuarios usuarios: </br></br>";
foreach ($dom->getElementsByTagname('usuario') as $usuario) {
    $nombre = $usuario->getElementsByTagname('nombre')->item(0)->nodeValue;
    $apellido = $usuario->getElementsByTagname('apellido')->item(0)->nodeValue;
    $direccion = $usuario->getElementsByTagname('direccion')->item(0)->nodeValue;
    $ciudad = $usuario->getElementsByTagname('ciudad')->item(0)->nodeValue;
    $pais = $usuario->getElementsByTagname('pais')->item(0)->nodeValue;
    $telefono = $usuario->getElementsByTagname('telefono')->item(0)->nodeValue;
    $url = $usuario->getElementsByTagname('url')->item(0)->nodeValue;
    $email = $usuario->getElementsByTagname('email')->item(0)->nodeValue;
    print "Datos del usuario $nombre $apellido</br>";
    print "Dirección: $direccion</br>";
    print "Ciudad: $ciudad</br>";
    print "País: $pais</br>";
    print "Teléfono: $telefono</br>";
    print "URL: $url</br>";
    print "Email: $email</br></br>";
}
?>
```

Ver: https://www.php.net/manual/es/book.dom.php

Ejemplo de SimpleXML

```
<?php
if(!$xml = simplexml_load_file('usuarios.xml')){ //Cargamos el archivo
    echo "No se ha podido cargar el archivo";
} else {
    $numusuarios=$xml->length;
    echo "Se van a listar los datos de un total de $numusuarios usuarios:
</br></br>";
    foreach ($xml as $usuario){
        echo 'Datos del usuario '.$usuario->nombre.' '.$usuario->apellido.'<br>';
        echo 'Dirección: '.$usuario->direccion.'<br>';
        echo 'Ciudad: '.$usuario->ciudad.'<br>';
        echo 'País: '.$usuario->pais.'<br>';
        echo 'Teléfono: '.$usuario->contacto->telefono.'<br>';
        echo 'Url: '.$usuario->contacto->url.'<br>';
        echo 'Email: '.$usuario->contacto->email.'<br><br>';
    }
}
?>
```


Ver: https://www.php.net/manual/es/book.simplexml.php

Otros orígenes de datos - JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y a la vez legible para humanos (como XML, pero sin el marcado). Su sintaxis es un subconjunto del lenguaje JavaScript que fue estandarizado en 1999.

Almacena combinaciones desordenadas de **clave:valor** en objetos { } o utiliza arrays [ ] para preservar el orden de los valores, por ello es fácil de analizar y de leer. Pero también tiene limitaciones, ya que JSON sólo define una cantidad pequeña de tipos de datos, por lo que para transmitir tipos como fechas deben ser transformadas en un string o en un unix timestamp como un integer.

Los tipos de datos que soporta JSON son: strings, números, booleanos y null, además de soportar objetos y arrays como valores.

Ver: https://www.php.net/manual/en/book.json.php

Usuarios.json

```
{"usuarios":
    [
        {
            "Nombre": "Silvia",
            "Apellido": "Vilar",
            "Direccion":  "Calle Mar 12",
            "Ciudad": "Valencia",
             "Pais": "España",
             "Contacto": [
                    {
                    "Telefono": "12345678",
                    "URL": "http://silvia.ejemplo.com",
                     "Email": "silvia@ejemplo.com"
                     }
             ]
        },
        {
            "Nombre": "Paco",
            "Apellido": "Ruiz",
            "Direccion":  "Calle La Fuente 22",
            "Ciudad": "Alicante",
            "Pais": "España",
            "Contacto": [
                    {
                    "Telefono": "23456789",
                    "URL": "http://paco.ejemplo.com",
                    "Email": "paco@ejemplo.com"
                    }
            ]
        }
    ]
}
```

Otros orígenes de datos - JSON

```
<?php
$data = file_get_contents("usuarios.json");
$json = json_decode($data); // Recoge el fichero json usuarios
$usuarios=$json->usuarios; //Cargamos los usuarios (primera etiqueta json)

$numusuarios=count($usuarios);
echo "Se van a listar los datos de un total de $numusuarios usuarios: </br></br>";
foreach ($usuarios as $usuario) { //accedemos a cada usuario
    echo 'Datos del usuario '.$usuario->Nombre.' '.$usuario->Apellido.'<br>';
    echo 'Dirección: '.$usuario->Direccion.'<br>';
    echo 'Ciudad: '.$usuario->Ciudad.'<br>';
    echo 'País: '.$usuario->Pais.'<br>';
    $contacto=$usuario->Contacto[0]; //Debemos acceder al nivel de contacto
    echo 'Teléfono: '.$contacto->Telefono.'<br>';
    echo 'URL: '.$contacto->URL.'<br>';
    echo 'Email: '.$contacto->Email.'<br><br>';
}
?>
```

{objeto**}** es un objeto JSON y puedes acceder a sus propiedades directamente: $objeto->propiedad.
[objeto**]** es un array JSON y puedes acceder a sus objetos/propiedades usando un bucle o indicando el índice, por ejemplo: $objeto[0]->propiedad.

JSON (json_decode)

La función **json_decode**($json) convierte un JSON a un vector asociativo o a un objeto. Esto depende de la opción ***associative*** que cuando su valor es ***false***, convierte el JSON a un **objeto** en lugar de a un vector asociativo. Si se especifica true, se devuelve el JSON como **vector asociativo**. Si no se indica nada (associative es null) depende de si la constante JSON_OBJECT_AS_ARRAY está establecida en los flags de la función json_decode()

```
<?php
$data = file_get_contents("usuarios.json");
$vector=json_decode($data,true);
$objeto=json_decode($data,false);

echo "\n<br>JSON decodificado como vector asociativo: \n<br>";
var_dump($vector);
echo "\n<br>JSON decodificado como objeto: \n<br>";
var_dump($objeto);
}
?>
```

JSON (json_decode) - Resultado

JSON decodificado como vector asociativo:

array(1) { ["usuarios"]=> array(2) { [0]=> array(6) { ["Nombre"]=> string(6) "Silvia" ["Apellido"]=> string(5) "Vilar" ["Direccion"]=> string(12) "Calle Mar 12" ["Ciudad"]=> string(8) "Valencia" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> array(3) { ["Telefono"]=> string(8) "12345678" ["URL"]=> string(25) "http://silvia.ejemplo.com" ["Email"]=> string(18) "silvia@ejemplo.com" } } } [1]=> array(6) { ["Nombre"]=> string(4) "Paco" ["Apellido"]=> string(4) "Ruiz" ["Direccion"]=> string(18) "Calle La Fuente 22" ["Ciudad"]=> string(8) "Alicante" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> array(3) { ["Telefono"]=> string(8) "23456789" ["URL"]=> string(23) "http://paco.ejemplo.com" ["Email"]=> string(16) "paco@ejemplo.com" } } } } }

JSON decodificado como objeto:

object(stdClass)#10 (1) { ["usuarios"]=> array(2) { [0]=> object(stdClass)#6 (6) { ["Nombre"]=> string(6) "Silvia" ["Apellido"]=> string(5) "Vilar" ["Direccion"]=> string(12) "Calle Mar 12" ["Ciudad"]=> string(8) "Valencia" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> object(stdClass)#7 (3) { ["Telefono"]=> string(8) "12345678" ["URL"]=> string(25) "http://silvia.ejemplo.com" ["Email"]=> string(18) "silvia@ejemplo.com" } } } [1]=> object(stdClass)#8 (6) { ["Nombre"]=> string(4) "Paco" ["Apellido"]=> string(4) "Ruiz" ["Direccion"]=> string(18) "Calle La Fuente 22" ["Ciudad"]=> string(8) "Alicante" ["Pais"]=> string(7) "España" ["Contacto"]=> array(1) { [0]=> object(stdClass)#9 (3) { ["Telefono"]=> string(8) "23456789" ["URL"]=> string(23) "http://paco.ejemplo.com" ["Email"]=> string(16) "paco@ejemplo.com" } } } } }

Buenas prácticas

Es altamente recomendable que todos los datos de configuración que sean susceptibles de cambiar en los distintos entornos (desarrollo, producción, test, etc.) se separen a uno o varios ficheros de configuración

Estos ficheros pueden codificarse en PHP o utilizar otro formato que después pueda ser leído por PHP

En el ejemplo, utilizaremos un fichero config.php

Buenas prácticas - Ejemplo

Contenido de BDConfig.php

```
<?php
const HOST = 'localhost';
const DBNAME = 'EMPRESA';
const USERNAME = 'dwes';
const PASSWORD = 'dbdwespass';
$options = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
?>
```

Contenido del script BDForm.php

```
<?php
require_once "BDConfig.php";
try{
$mbd = new PDO('mysql:host=HOST;dbname=DBNAME', USERNAME,
PASSWORD);
echo "Conectado con éxito en la BD ".DBNAME." alojada en ".HOST." con el
usuario ".USERNAME."\n";
} catch (PDOException $e) {
        print "Error al conectar con la BD: " .  $e->getMessage();
}
?>
```


Conectado con éxito en la BD EMPRESA alojada en localhost:33006 con el usuario dwes

La Sénia
Institut d'Educació Secundària
Paiporta

UD11. Uso framework PHP: LARAVEL

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Estructura proyecto Laravel
- Artisan
- Rutas
- Vistas. Motor de plantillas Blade
- Controladores
- Bases de datos
- Migraciones
- Modelos
- Operaciones con Modelos
- Relaciones entre Modelos
- Seeders y factories
- API REST

Estructura proyecto Laravel

1. **app**: contiene el código fuente de la aplicación. Subcarpetas console (definir comandos propios), exceptions (definir excepciones), **http** (controladores y el middleware), providers (proveedores de servicios de la aplicación y los propios).
2. **bootstrap**: contiene app.php (inicia aplicación), la carpeta cache (ficheros ya cargados), etc.
3. **config**: archivos de configuración de la aplicación (variables de entorno, desarrollo o producción, parámetros de conexión a la BD, etc.). En él se accede a las variables de entorno definidas en ***.env***.
4. **database**: elementos de gestión de la base de datos, tales como generadores de objetos, migraciones, etc.
5. **public**: contiene ***index.php***, punto de entrada de todas las peticiones a la web, además de carpetas donde ubicar el contenido estático del cliente (imágenes, hojas de estilo CSS, archivos JavaScript…)
6. **resources**: contiene las **vistas** de nuestra aplicación y archivos no compilados de CSS (archivos sass) y JavaScript (archivos sin minimizar). También almacena los archivos de traducción, en el caso de que queramos hacer sitios multi idioma.
7. **routes**: almacena las **rutas** de la aplicación, del contenido web normal (web.php), servicios web (api.php), comandos y otras opciones.
8. **storage**: contiene las vistas compiladas y otros archivos como los logs o las sesiones.
9. **test**: almacena los tests o pruebas sobre los componentes de nuestra aplicación
10. **vendor**: almacena las dependencias o librerías adicionales de terceros (o proveedores) que se requieren en nuestro proyecto Laravel.

Además Laravel tiene unas clases propias (Routes, Model, etc.) en el espacio de nombres Illuminate que se incluyen con **use**:

```
use Illuminate\Database\Eloquent\Model
```


Ver: https://laravel.com/docs/11.x/structure

Artisan

- Al instalar Laravel disponemos de la herramienta Artisan, una CLI para gestionar nuestro proyecto.
- Podemos comprobar los comando que podemos usar en esta consola ejecutando la opción list

```
php artisan list
```

- Podemos comprobar la versión instalada de Laravel aplicando la opción --version

```
php artisan --version
```

- La consola muestra ayuda de los comandos con help

```
php artisan help migrate
```


Ver: https://laravel.com/docs/11.x/artisan

Routes

- Las rutas (routes) establecen qué respuesta enviar a una petición que intenta acceder a una determinada URL.
- En el archivo routes/web.php definiremos las rutas web para recuperar contenidos típicamente en formato HTML.
- El archivo routes/api.php define servicios REST
- La ruta se define mediante Route::get($url, $funcionAEjecutar)

Ruta que carga la página de bienvenida de Laravel

```
<?php
use Illuminate\Support\Facades\Route;
Route::get('/', function() {
    return view('welcome');
});
?>
```


Ver: https://laravel.com/docs/11.x/routing

Tipos de rutas

- **Simples**: tienen un nombre de ruta fijo y una función que responde a dicho nombre emitiendo una respuesta

Se llama con la url localhost:8083/fecha

```
Route::get('fecha', function() {
    return date("d/m/y h:i:s");
});
```

- Con **parámetros**: la url incluye el nombre del parámetro entre llaves que también se pasa a la función del segundo parámetro.

localhost:8083/saludo/Silvia

```
Route::get('saludo/{nombre}', function($nombre) {
    return "Hola, " . $nombre;
});
```

  - Da error 404 si no se indica en la URL (obligatorio). Se añade **?** Para indicar que es opcional y se pasa valor por defecto en la función

localhost:8083/saludo/

```
Route::get('saludo/{nombre?}', function($nombre="invitado") {
    return "Hola, " . $nombre;
});
```

  - También se pueden validar parámetros con **where**

```
Route::get('saludo/{nombre?}', function($nombre = "Invitado") {
    return "Hola, " . $nombre;
})->where('nombre', "[A-Za-z]+");
```


Ver: https://laravel.com/docs/11.x/routing#basic-routing y https://laravel.com/docs/11.x/routing#route-parameters

Tipos de rutas

- **Nominadas**: la ruta recibe un nombre con la función name() para referenciarla en el código y optimizar modificaciones de la misma

En lugar de poner Contacto en el código, pondremos
Contacto

```
Route::get('contacto', function() {
        return "Página de contacto";
})->name('ruta_contacto');
```

- Se pueden combinar cláusulas where de validación de parámetros con el nombre de las rutas de la siguiente forma:

```
Route::get('saludo/{nombre?}/{id?}',
function($nombre="Invitado", $id=0)
{
        return "Hola $nombre, tu código es el $id";
})->where('nombre', "[A-Za-z]+")
   ->where('id', "[0-9]+")
   ->name('saludo');
```

URL	Respuesta
/saludo	Hola Invitado, tu código es el 0
/saludo/Silvia	Hola Silvia, tu código es el 0
/saludo/Silvia/3	Hola Silvia, tu código es el 3
/saludo/3	Error 404 (URL incorrecta)

Ver: https://laravel.com/docs/11.x/routing#named-routes

Vistas – motor de plantillas Blade

- Las vistas las encontramos en **resources/views**
- Se crean las vistas para que las rutas puedan devolver el código html de la respuesta. Tienen la extensión **.blade.php**
- Podemos crear una vista muy sencilla llamada ***inicio.blade.php***

```
html>
    <head>
        <title>Inicio</title>
    </head>
    <body>
        <h1>Página de inicio</h1>
    </body>
</html>
```

- Se llamará en ***routes/web.php***

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


Ver: https://laravel.com/docs/11.x/views y https://laravel.com/docs/11.x/blade

Vistas – pasar parámetros

Podemos pasar valores a las vistas:

- Con el método **with** en la ruta (con variable o vector)

```
Route::get('/', function() {
        $nombre = "Silvia";
        return view('inicio')->with('nombre', $nombre);
//también return view('inicio')->with(['nombre' => $nombre, ...]);
});
```

- Con array asociativo en la vista

```
Route::get('/', function() {
        $nombre = "Silvia";
        return view('inicio', ['nombre' => $nombre, ...]);
});
```

- Con compact en la vista

```
Route::get('/', function() {
        $nombre = "Silvia";
        return view('inicio', compact('nombre'));
});
```

- Directamente en la ruta si tiene poca lógica interna o poca información asociada

```
Route::view('/', 'inicio', ['nombre' => 'Silvia']);
```


Ver: https://laravel.com/docs/11.x/views#passing-data-to-views

Blade – elementos

Blade permite usar estructuras de control: @if -@else / @elseif - @endif, @isset - endisset, @forelse - @empty - @endforelse, @foreach - @endforeach, @while - @endwhile, @for - @endfor y @switch - @case - @defatult - @endswitch

- Cuando se recorre una estructura de datos, podemos usar **$loop** que tiene métodos index, count, first y last

```
<ul>
    @forelse($elementos as $elemento)
        <li>{{ $elemento }}
    {{ $loop->last ? "Ultimo elemento" : "" }}
        </li>
    @empty
        <li>No hay elementos que mostrar</li>
    @endforelse
</ul>
```

- En blade podemos llamar a las rutas usando la función **url()**

```
<a href="{{ url('/contacto') }}">Contacto</a>
```


Ver: https://laravel.com/docs/11.x/views#passing-data-to-views

Blade – creando plantillas

Al crear una plantilla se indica el contenido variable con la sección @yield (A la derecha definición de plantilla.blade.php)

@include: incluimos la vista con el menú de navegación que se llama partials.nav.blade.php

```
<html>
    <head>
        <title>
            @yield('titulo')
        </title>
    </head>
    <body>
        <nav>
            @include(‘partials.nav’)
        </nav>
        @yield('contenido')
    </body>
</html>
```

Podemos indicar el contenido que deseamos de la siguiente forma (abajo la vista inicio.blade.php)

```
@extends('plantilla')
@section('titulo', 'inicio')
@section('contenido')
    <h1>Página de inicio</h1>
    Bienvenido/a {{ $nombre }}
@endsection
```

@extends: indica plantilla a usar
@section seguida del nombre de la sección: contenido para cada uno de los @yield que se hayan indicado en la plantilla.
Finaliza cada sección con la directiva @endsection

Ver: https://laravel.com/docs/11.x/views#passing-data-to-views

Incluir CSS y JS en cliente

Las dependencias de librerías de la parte del cliente se encuentran en el archivo **package.json** en la raíz del proyecto que se instalan con el comando ***npm install***
En el fichero **resources/css/app.css** (o resources/sass/app.css) se definen estilos CSS propios o se incluyen librerías externas usando CSS plano o Sass. En el fichero **resources/js/app.js** se definen funciones propias en JavaScript.

Estos archivos se especifican en **webpack.mix.js** que usa la herramienta WebPack para compilar, empaquetar y optimizar el resultado que se guarda en **public/css** y **public/js** respectivamente. Tras ejecutar ***npm run dev*** se obtienen los ficheros css y js resultado que se pueden incluir en la plantilla

```
<html>
    <head>
        <title>
            @yield('titulo')
        </title>
        <link rel="stylesheet" type="text/css" href="/css/app.css">
        <script type="text/javascript" scr="/js/app.js">
        </script>
    </head>
...
```

Para incluir Bootstrap debemos incluir la librería ui (User Interface) con composer require laravel/ui y emplearla con artisan con **php artisan ui bootstrap** (añade dependencia bootstrap en webpackage.json) que enlaza a dicha librería en el archivo **resources/sass/app.scss** para generar un archivo CSS optimizado con Bootstrap. Se debe ejecutar ***npm install && npm run dev*** para instalar bootstrap y obtener el resultado

A partir de Laravel 9.x se puede usar **Vite** https://laravel.com/docs/11.x/vite#main-content

Ver: https://laravel.com/docs/11.x/mix#introduction

Controladores (Controller)

- Los controladores gestionan cierta lógica común de las peticiones (acceso a los datos, validación de formularios, etc).
- Se guardan en **app/Http/Controllers** y se genera vacío con el comando `php artisan make:controller PruebaController`
- El controlador generado con opción **-i** tiene el método **__invoke** para definir la lógica de generar u obtener los datos que necesita una vista, y renderizarla

```
php artisan make:controller PruebaController -i
```

```
<?php    //PruebaController.php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class PruebaController extends Controller {
        public function __invoke(Request $request){
                $datos = array(...);
                return view('miVista', compact('datos'));
        }
}
```

```
<?php    //routes/web.php
use App\Http\Controllers\PruebaController;
...
Route::get('prueba', PruebaController::class)->name('prueba');
```


Ver:https://laravel.com/docs/11.x/controllers y https://laravel.com/docs/11.x/controllers#single-action-controllers

Controlador de recursos

El controlador de recursos tiene los siguientes métodos:

- **index**: lista los elementos de la entidad o recurso
- **create**: muestra formulario de alta para nuevos elementos
- **store**: almacena en la BD el recurso creado en create
- **show**: muestra los datos de un recurso por su clave o id
- **edit**: muestra formulario para editar un recurso existente
- **update**: actualiza en la BD el recurso editado con edit
- **destroy**: elimina un elemento por su clave o id

```
php artisan make:controller PruebaController -r
```

- Al llamar a alguno de sus métodos hay que indicar cuál es el que queremos ejecutar

```
<?php    //routes/web.php
use App\Http\Controllers\PruebaController;
...
Route::get('prueba', [PruebaController::class, 'index'])->name('listado_prueba');
Route::get('prueba/{id}', [PruebaController::class, 'show']);
```


Ver: https://laravel.com/docs/11.x/controllers#resource-controllers

Controlador de api

El controlador de api tiene los siguientes métodos (similares a controlador de recursos pero sin create ni edit):

- **index**: lista los elementos de la entidad o recurso
- **store**: almacena en la BD el recurso
- **show**: muestra los datos de un recurso por su clave o id
- **update**: actualiza en la BD el recurso
- **destroy**: elimina un elemento por su clave o id

```
php artisan make:controller PruebaController --api
```

- Hay que indicar cuál es el método que queremos ejecutar. Podemos indicar los métodos que queremos tener accesibles o no. Se pueden generar todas las rutas usando apiReource

```
<?php    //routes/web.php
use App\Http\Controllers\PruebaController;
...
Route::resource('prueba', PruebaController::class)->only(['index','show']);
Route::resource('prueba', PruebaController::class)->except(['store','update','destroy']);
Route::apiResource('prueba',PruebaController::class)
```


Ver: https://laravel.com/docs/11.x/controllers#api-resource-routes

Request (petición del usuario)

El controlador recibe datos de la petición en el objeto Request

- Por ejemplo, si hemos pasado el parámetro id del elemento en nuestra ruta, este elemento se recoge en el objeto $request que se le pasa al método para poder operar con él

```
class LibroController extends Controller
{
    ...
    public function store(Request $request)
    {
        $id=$request->id;
        $name=$request->name;
       ...
    }
}
```

- Por otra parte, podemos obtener de la petición el contenido de un campo del formulario o de la query string o un fichero subido

```
$nombre= $request->input('nombre');
$nombre= $request->query('nombre');
$fichero= $request->file('foto');
```


Ver: https://laravel.com/docs/11.x/requests

Response (respuesta del servidor)

El método respuesta admite los siguientes parámetros:

- Contenido de la respuesta
- Código de estado HTTP (por defecto es 200)
- Array con las cabeceras de respuesta (vacío por defecto)

```
response("Mensaje de respuesta", 200)
        ->header('Cabecera1', 'Valor1')
        ->header('Content-Type', 'text/plain');
```

Se puede usar el método **json** para devolver un objeto de respuesta (los objetos del modelo y colecciones se devuelven automáticamente en formato json https://laravel.com/docs/11.x/eloquent-collections)

```
return response()->json(['dato1' => 'valor1', 'dato2'=>'valor2'], 201)
        ->header('Cabecera1', 'Valor1')
        ...;
```

También se pueden hacer redirecciones

```
redirect('/');    //ruta como parámetro
redirect()->route('inicio');    //indicando ruta con nombre
```


Ver: https://laravel.com/docs/11.x/responses

Bases de datos

La configuración de la base de datos se encuentra en **config/database.php**. En este fichero se realizan llamadas al fichero **.env** que es el que contiene las constantes de conexión. La función **env**('VALOR_ENV','valorDefault') donde el primer parámetro lo lee del fichero ,env y si no lo encuentra, lee el segundo parámetro.

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Para crear y modificar el esquema de la BD, se usan las **migraciones** que funcionan como un control de versiones de la BD. Se encuentran en **database/migrations**

Para acceder a los datos de las tablas, se define un **modelo** de datos asociado a cada una y se manipulan empleando el ORM (Object Relationar Mapping) Eloquent que incorpora Laravel. Se encuentran en **app/models**

Para poblar de datos de prueba la BD, se usan los **factories** y **seeders** cuyas carpetas se encuentran en la carpeta **database**

Ver: https://laravel.com/docs/11.x/database

Migraciones

La migración se crea con el comando php artisan make_migration nombre_migración: `php artisan make:migration crear_tabla_usuario`

La migración tiene dos métodos: **up** que permite agregar tablas, columnas o índices a la BD y **down** que revierte lo hecho por el método up

```
public function up()
{
	Schema::create('usuarios', function(Blueprint $tabla) {
		$tabla->id();
		$tabla->string('nombre');
	});
}
public function down()
{
	Schema::drop('usuarios');
}
```

Las migraciones se ejecutan con el comando `php artisan migrate`

Si se desea borrar la BD y volver a ejecutar las migraciones, se usa `php artisan migrate:fresh`

Ver:https://laravel.com/docs/11.x/migrations

Modelos

Los Modelos heredan de Illuminate/Database/Eloquent/Model.
Para generarlos usamos el comando php artisan make:model NombreModelo: php artisan **make:model** Usuario
De esta forma se obtiene el siguiente código en Usuario

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;
class Usuario extends Model
{
        protected $table = ‘usuario’;
}
```

Podemos crear un Modelo con su migración usando el comando con la opción --migration o -m: php artisan make:model Usuario **--migration**

Además podemos añadir un Controlador del modelo vacío con la opción --controller o -c: php artisan make:model Usuario **-mc**
Si queremos que sea un controlador de recursos hay que añadir la opción --recursos o -r: php artisan make:model Usuario **-mcr**

Ver:https://laravel.com/docs/11.x/eloquent#generating-model-classes

Operaciones con Modelos - Búsquedas

Se puede importar en Controller y usar los métodos de Model. Para listado usamos get() y para mostrar elemento usamos find(). Si el modelo está vacío podemos usar findOrFail() (muestra excepción)

```
use App\Models\Usuario; //UsuarioController
...
class UsuarioController extends Controller
{
	public function index()
	{
		$usuarios = Usuario::get();
		return view('usuarios.index', compact('usuarios'));
	}
	public function show($id)
	{
		$usuario = Usuario::find($id); // o findOrFail($id)
		return view('usuarios.show', compact('usuario'));
	}
}
```

```
//usuarios/index.blade.php
@forelse($usuarios as $usuario)
	{{ $usuario->nombre }}
	li><a href="{{ route('usuarios.show', $usuario) }}">
		{{ $usuario->nombre }}
	</a></li>
@endforelse
0
```

```
//route/web.php
Route::resource('usuarios', UsuarioController::class);
```


Ver: https://laravel.com/docs/11.x/eloquent#retrieving-single-models

Operaciones con Modelos - Inserción

Se usa el método **save** de Model para las inserciones. Cada campo de la petición debe tener asociado un campo del mismo nombre en el modelo. Se realiza en el método **store** del controlador

```
use App\Models\Libro;
...
class LibroController extends Controller
{
	public function store(Request $request)
	{
		$libro = new Libro();
		$libro->titulo = $request->titulo;
		$libro->editorial = $request->editorial";
		$libro->precio = $request->precio;
		$libro->save();
	}
```

Si el modelo, se indican los campos a insertar en la propiedad **$fillable**, se puede llamar directamente al método **create** del controlador.

```
class Libro extends Model
{
	protected $fillable = ['titulo', 'editorial', 'precio'];
}
```

```
Libro::create($request->all());
```


Ver:https://laravel.com/docs/11.x/eloquent#inserts

Operaciones con Modelos - Modificación

La modificación consiste en encontrar el objeto a modificar (**findOrFail**) y llamar al método **save**() del modelo tras modificar las propiedades

```
use App\Models\Libro;
...
class LibroController extends Controller
{
	public function update(Request $request, Libro $libro)
	{
		$libroAmodificar = Libro::findOrFail($libro->id);
		$libroAModificar->titulo=$request->titulo;
		$libroAModificar->save();
	}
}
```

Si el atributo **$fillable** está indicado, se puede utilizar el método **update** enlazado con **findOrFail**

```
class Libro extends Model
{
	protected $fillable = ['titulo', 'editorial', 'precio'];
}
```

```
Libro::findOrFail($id)->update($request->all());
```


Ver:https://laravel.com/docs/11.x/eloquent#updates

Operaciones con Modelos - Borrados

El borrado debe encontrar el objeto a eliminar (**findOrFail**) y llamar al método **delete**() de la instancia del modelo. Se lleva a cabo en el método **destroy** del controlador.
Tras el borrado, normalmente se redirige o renderiza alguna vista resultado como el listado de elementos tras borrar una de ellos.

```
use App\Models\Libro;
...
class LibroController extends Controller
{
	public function destroy(Libro $libro)
	{
		$libro->delete();
		$libros = Libro::get();
		return view('libros.index', compact('libros'));
	}
}
```


Ver:https://laravel.com/docs/11.x/eloquent#deleting-models

Relaciones entre Modelos – 1:1

Suponemos la relación usuario y teléfono con relación 1:1. Una de las tablas debe contener la referencia a la otra así que en al tabla telefono añadiremos el atributo usuario_id. En la clase Usuario debemos indicarle que tiene un telefono con un método con el nombre de la clase que la referencia con **hasOne**:

```
class Usuario extends Model
{
        ,,,
        public function telefono(): HasOne
        {
                return $this->hasOne(Telefono::class);
        }
}
```

Para obtener el teléfono del usuario, usaremos findOrFail

```
$telefono = Usuario::findOrFail(1)->telefono;
```


Ver:https://laravel.com/docs/11.x/eloquent-relationships#one-to-one

Relaciones entre Modelos – 1:1

Si deseamos que, a partir de un teléfono podamos obtener el usuario al que pertenece, indicamos la relación inversa, esto es, emplearemos en telefono el método **belongsTo**

```
class Telefono extends Model
{
        ,,,
        public function usuario(): BelongsTo
        {
                return $this->belongsTo(Usuario::class);
        }
}
```

Para obtener el usuario del teléfono, usaremos findOrFail

```
$usuario = Telefono::findOrFail($idTelefono)->usuario;
```


Ver: https://laravel.com/docs/11.x/eloquent-relationships#one-to-one

Relaciones entre Modelos – 1:N

Suponemos la relación autor y libro con relación 1:N. La tabla N (libro) debe tener una referencia a la tabla 1 (autor) por lo que libro tendrá un campo autor_id. En la clase Autor indicamos que tiene varios libros con **hasMany**:

```
class Autor extends Model
{       ,,,
        public function libros(): HasMany
        {
                return $this->hasMany(Libro::class);
        }
}
```

Para obtener los libros del autor, usaremos findOrFail (podemos añadir criterios de búsqueda)

```
$libros = Autor::findOrFail(1)->libros;
```

```
$libros = Autor::findOrFail(1)->libros()->where('titulo','Informática');
```


Ver: https://laravel.com/docs/11.x/eloquent-relationships#one-to-many

Relaciones entre Modelos – 1:N

Si deseamos que, a partir de un libro podamos obtener el autor al que pertenece, indicamos la relación inversa, esto es, emplearemos en libro el método **belongsTo**

```
class Libro extends Model
{
        ,,,
        public function autor(): BelongsTo
        {
                return $this->belongsTo(Autor::class);
        }
}
```

Para obtener el autor del libro, usaremos findOrFail

```
$nombreAutor = Libro::findOrFail($id)->autor->nombre;
```


Ver:https://laravel.com/docs/11.x/eloquent-relationships#one-to-many

Relaciones entre Modelos – N:M

Suponemos la relación usuario y roles con relación N:M de la que debe crearse la tabla rol_usuario (alfabéticamente) con los campos usuario_id y rol_id. Debemos crear un método en cada clase que devuelva la relación mediante **belongsToMany**:

```
class Usuario extends Model
{     ,,,
      public function roles(): BelongsToMany
      {
            return $this->belongsToMany(Rol::class);
      }
}
```

Para obtener el rol del usuario, usaremos findOrFail

```
$roles = Usuario::findOrFail($id)->roles();
```


Ver:https://laravel.com/docs/11.x/eloquent-relationships#many-to-many

Relaciones entre Modelos – N:M

Si deseamos que, a partir de un libro podamos obtener el autor al que pertenece, indicamos la relación inversa, esto es, emplearemos en rol el método **belongsToMany**

```
class Rol extends Model
{
        ,,,
        public function usuarios(): BelongsToMany
        {
                return $this->belongsToMany(Usuario::class);
        }
}
```

Para obtener los usuarios del rol, usaremos findOrFail

```
$usuarios = Rol::findOrFail($id)->usuarios();
```


Ver:https://laravel.com/docs/11.x/eloquent-relationships#many-to-many

Seeders

Los seeders son clases especiales que permiten poblar de datos la BD de forma automática. Se almacenan en la carpeta **database/seeders** y se crean con php artisan **make:seeder** NombreSeeder

```
php artisan make:seeder LibrosSeeder
```

```
Class LibrosSeeder extends Seeder{
    public function run()
    {
        $libro = new Libro();
        $libro->titulo = "El libro del Seeder";
        $libro->editorial = "Seeder S.A.";
        $libro->precio = 10;
        $libro->save();
    }
}
```

En el método **run** del seeder indicamos los datos a almacenar. El seeder debe darse de alta en la clase **DatabaseSeeders** con el método **call**

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(LibrosSeeder::class);
    }
}
```

El seeder se ejecuta añadiendo la opción --seed

```
php artisan migrate:fresh --seed
```


Ver: https://laravel.com/docs/11.x/seeding

Factories

Los factories permiten generar datos por lotes. Se almacenan en la carpeta **database/factories** y se crean con php artisan **make:factory** NombreFactory. Se debe indicar **use** en el Modelo.

```
php artisan make:factory UsuarioFactory
```

```
Class UsuarioFactory extends Factory{
    public function definition()
    {
        return [
            'nombre' => fake()->name(),
            'email=>fake()->unique()-safeEmail(),
            'password'=>fake()->password(),
            'creado'=>now(),
        ];
    }
}
```

En el método **definition** del factory indicamos los datos a generar. Se puede usar el objeto Faker para datos aleatorios. En el seeder podemos indicar en el método run cuántos elementos del factory debe crear

```
class UsuarioSeeder extends Seeder
{
    public function run()
    {
        Usuario::factory()->count(5)->create();
    }
}
```

El factory se ejecuta como el seeder, añadiendo la opción --seed

```
php artisan migrate:fresh --seed
```


Ver: https://laravel.com/docs/11.x/eloquent-factories y https://github.com/FakerPHP/Faker

API REST

Usaremos controladores de API para los servicios Web

```
php artisan make:controller Api/LibroController --api --model=Libro
```

En lugar de devolver vistas, devolveremos json. Si no queremos mostrar algunos campos, usamos la propiedad **$hidden** en el modelo, de forma similar a $fillable

```
class Usuario extends Model
{
    protected $hidden = ['password'];
}
```

HTTP	URI	Controller
GET	api/libros	libros.index
POST	api/libros	libros.store
GET	api/libros/{libro}	libros.show
PUT	api/libros/{libro}	libros.update
DELETE	api/libros/{libro}	libros.destroy

```
Class LibroController extends Controller{
    public function index()
    {
        $libros = Libro::get();
        return response()->json($libros, 200);
    }
    public function show(Libro $libro)
    {
        return response()->json($libro, 200);
    }
    public function destroy(Libro $libro)
    {
        $libro->delete();
        return response()->json(null, 204);
    }
}
```


Ver: https://laravel.com/docs/11.x/responses