

Ejercicios resueltos de Desarrollo Web en Entorno Servidor

Los 48 ejercicios de Desarrollo Web en Entorno Servidor de la web, tema a tema: cada uno con su enunciado, los datos que necesitas (código de partida, ejemplos de entrada y salida o la base de datos) y, al final, la solución explicada.

Índice

Arquitecturas y Tecnologías del Servidor

[1. Analiza una petición HTTP](#)

[3. Un enrutador con parámetros](#)

[2. Construye la respuesta](#)

Fundamentos de PHP

[4. Tabla de multiplicar en HTML](#)

[6. Encuentra el fallo: el saludo que suma](#)

[8. Ticket de compra](#)

[10. Encuentra el fallo: la subida de precios que no sube](#)

[12. Tu propio motor de plantillas](#)

[5. Encuentra el fallo: la frase que empieza por hola](#)

[7. Las palabras más repetidas](#)

[9. Convertir y validar los datos de un formulario](#)

[11. Calendario del mes en HTML](#)

Formularios, Sesiones y Seguridad

[13. Comentarios sin XSS](#)

[15. Un carrito guardado en la sesión](#)

[17. Inicio de sesión con bloqueo](#)

[19. Control de acceso por roles y propiedad](#)

[14. Validar los datos de un formulario](#)

[16. Encuentra el fallo: el comentario que ejecuta código](#)

[18. Cookies firmadas](#)

POO y el Patrón MVC

[20. Una entidad que se valida sola](#)

[22. Encuentra el fallo: los intereses que no llegan](#)

[24. Una cadena de middlewares](#)

[21. Nóminas con herencia e interfaces](#)

[23. Un motor de plantillas mínimo](#)

[25. Un contenedor de dependencias con autowiring](#)

Acceso a Datos con PDO

[26. Encuentra el fallo: el listado de un solo producto](#)

[28. Pagar resultados con LIMIT](#)

[30. Transferencias con transacciones](#)

[32. Importar un CSV: todo o nada](#)

[27. Buscar sin inyección SQL](#)

[29. Filtros dinámicos sin inyección](#)

[31. Un informe sin el problema N+1](#)

Servicios Web y APIs REST

[33. Resumir la respuesta de una API](#)

[35. Una API REST de libros](#)

[37. Limitar las peticiones: 429 Too Many Requests](#)

[34. Caché HTTP con ETag](#)

[36. Tokens JWT firmados a mano](#)

Aplicaciones Dinámicas e Híbridas

[38. El endpoint de un autocompletado](#)

[40. Un cortacircuitos para un servicio externo](#)

[39. Verificar la firma de un webhook](#)

Ejercicios de Desarrollo Web en Entorno Servidor

[41. Informe a partir de un CSV](#)

[43. Carrito con cupones](#)

[42. API de tareas con base de datos](#)

Ejercicios largos

44. Factura de un carrito con IVA y promociones

46. API REST de tareas con un router propio

48. Horas trabajadas con nocturnidad usando
DateTime

45. Préstamos de una biblioteca con PDO y
transacciones

47. Libro de visitas seguro: validación, spam y
escape de HTML

Arquitecturas y Tecnologías del Servidor

1. Analiza una petición HTTP

Medio · Arquitecturas y Tecnologías del Servidor · apuntesdam.com/subject/dwes/topic/arquitecturas-servidor

La entrada es una petición HTTP tal y como llega al servidor: la línea de petición, las cabeceras, una línea en blanco y, a veces, un cuerpo en formato de formulario (application/x-www-form-urlencoded). Muestra «Método: ...», «Ruta: ...» (sin la query string), una línea «GET nombre = valor» por cada parámetro de la URL, «Host: ...», «Cabeceras: N» y una línea «POST nombre = valor» por cada campo del cuerpo. Los valores deben aparecer decodificados (%C3%B1 es ñ y + es un espacio).

Código de partida (php)

```
<?php
$texto = stream_get_contents(STDIN);
$lineas = explode("\n", str_replace("\r", '', $texto));

// TODO: línea de petición, cabeceras (hasta la primera línea vacía) y cuerpo
```

Ejemplo: Petición GET con parámetros

ENTRADA

```
GET /buscar?q=teclado+mec%C3%A1nico&pagina=2
HTTP/1.1
Host: tienda.es
Accept: text/html
```

SALIDA ESPERADA

```
Método: GET
Ruta: /buscar
GET q = teclado mecánico
GET pagina = 2
Host: tienda.es
Cabeceras: 2
```

Ejemplo: Formulario con POST

ENTRADA

```
POST /registro HTTP/1.1
Host: academia.es
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

nombre=Ana+Mu%C3%B1oz&email=ana%40correo.es&acep
to=1
```

SALIDA ESPERADA

```
Método: POST
Ruta: /registro
Host: academia.es
Cabeceras: 3
POST nombre = Ana Muñoz
POST email = ana@correo.es
POST acepto = 1
```

2. Construye la respuesta

Medio · Arquitecturas y Tecnologías del Servidor · apuntesdam.com/subject/dwes/topic/arquitecturas-servidor

Cada línea de la entrada describe una respuesta con «código;tipo;cuerpo». Escribe la respuesta HTTP completa: la línea de estado «HTTP/1.1 código texto» (200 OK, 201 Created, 204 No Content, 301 Moved Permanently, 404 Not Found, 500 Internal Server Error), «Content-Type: tipo; charset=utf-8», «Content-Length: N» con la longitud del cuerpo en bytes, una línea en blanco y el cuerpo. Una respuesta 204 no lleva cuerpo, ni Content-Type ni Content-Length. Si el código no está en la lista, escribe «Código no soportado: N». Separa cada respuesta de la siguiente con una línea «--».

Código de partida (php)

```
<?php
$textos = [200 => 'OK']; // TODO: completa los códigos

$respuestas = [];
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$codigo, $tipo, $cuerpo] = explode(';', $linea, 3);
    // TODO: construye la respuesta y añádela a $respuestas
}
echo implode("\n---\n", $respuestas), "\n";
```

Ejemplo: Varias respuestas

ENTRADA

```
200;text/html;<h1>Hola</h1>
404;application/json>{"error":"No encontrado"}
```

SALIDA ESPERADA

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 13

<h1>Hola</h1>
---
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8
Content-Length: 25

{"error":"No encontrado"}
```

Ejemplo: Bytes, 204 y códigos raros

ENTRADA

```
201;text/plain;Creado: añadido; ok
204;text/plain;
418;text/plain;Soy una tetera
```

SALIDA ESPERADA

```
HTTP/1.1 201 Created
Content-Type: text/plain; charset=utf-8
Content-Length: 20

Creado: añadido; ok
---
HTTP/1.1 204 No Content
---
Código no soportado: 418
```

3. Un enrutador con parámetros

Difícil · Arquitecturas y Tecnologías del Servidor · apuntesdam.com/subject/dwes/topic/arquitecturas-servidor

Así deciden los frameworks qué código ejecutar para cada URL. La entrada empieza con rutas «MÉTODO patrón acción» (el patrón puede tener parámetros como /productos/{id}), una línea «---» y después peticiones «MÉTODO url». Para cada petición escribe la acción y sus parámetros en orden («verProducto id=42»), «404 No encontrado» si ninguna ruta encaja con la ruta de la URL, o «405 Método no permitido (GET, PUT)» si la ruta existe pero no para ese método (lista los métodos permitidos en el orden en que se definieron). Ignora la query string y una barra final (/productos/ equivale a /productos). Si varias rutas encajan, gana la primera definida.

Código de partida (php)

```
<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
$separador = array_search('---', $lineas);
$rutas = array_slice($lineas, 0, $separador);
$peticiones = array_slice($lineas, $separador + 1);

foreach ($peticiones as $peticion) {
    [$metodo, $url] = explode(' ', $peticion);
    // TODO
    echo "404 No encontrado\n";
}
```

Ejemplo: Rutas básicas

ENTRADA

```
GET /productos listar
GET /productos/{id} verProducto
POST /productos crear
---
GET /productos
GET /productos/42
POST /productos
GET /clientes
```

SALIDA ESPERADA

```
listar
verProducto id=42
crear
404 No encontrado
```

Ejemplo: Varios parámetros, 405 y barra final

ENTRADA

```
GET /usuarios/{usuario}/pedidos/{pedido}
verPedido
GET /productos/{id} ver
PUT /productos/{id} actualizar
---
GET /usuarios/ana/pedidos/7?detalle=1
DELETE /productos/3
GET /productos/3/
PUT /productos/3
```

SALIDA ESPERADA

```
verPedido usuario=ana pedido=7
405 Método no permitido (GET, PUT)
ver id=3
actualizar id=3
```

Fundamentos de PHP

4. Tabla de multiplicar en HTML

Fácil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

PHP genera HTML. Lee un número de la entrada y escribe su tabla de multiplicar del 1 al 10 como una tabla HTML, con una fila por línea y dos espacios de sangría: « `<tr><td>7 x 1</td><td>7</td></tr>` ». Si la entrada no es un entero entre 1 y 100, escribe «`<p>Número no válido</p>`».

Código de partida (php)

```
<?php
$texto = trim((string) fgets(STDIN));

// TODO
```

Ejemplo: Tabla del 7

ENTRADA

7

SALIDA ESPERADA

```
<table>
<tr><td>7 x 1</td><td>7</td></tr>
<tr><td>7 x 2</td><td>14</td></tr>
<tr><td>7 x 3</td><td>21</td></tr>
<tr><td>7 x 4</td><td>28</td></tr>
<tr><td>7 x 5</td><td>35</td></tr>
<tr><td>7 x 6</td><td>42</td></tr>
<tr><td>7 x 7</td><td>49</td></tr>
<tr><td>7 x 8</td><td>56</td></tr>
<tr><td>7 x 9</td><td>63</td></tr>
<tr><td>7 x 10</td><td>70</td></tr>
</table>
```

Ejemplo: Entrada no válida

ENTRADA

abc

SALIDA ESPERADA

<p>Número no válido</p>

5. Encuentra el fallo: la frase que empieza por hola

Fácil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

El programa dice qué frases contienen la palabra «hola». Con «hola mundo» responde que no la contiene, aunque es lo primero que aparece. Corrige la comprobación.

Código de partida (php)

```
<?php
$frases = ["hola mundo", "dice hola", "adiós"];
foreach ($frases as $f) {
    if (strpos($f, "hola")) {
        echo "$f: contiene hola\n";
    } else {
        echo "$f: no contiene hola\n";
    }
}
```

Ejemplo: Tres frases

SALIDA ESPERADA

```
hola mundo: contiene hola
dice hola: contiene hola
adiós: no contiene hola
```

6. Encuentra el fallo: el saludo que suma

Fácil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

El programa debería mostrar «Hola, Ana. Tienes 3 mensajes.», pero termina con un error. Corrígelo.

Código de partida (php)

```
<?php
$nombre = "Ana";
$mensajes = 3;
echo "Hola, " + $nombre + ". Tienes " + $mensajes + " mensajes.\n";
```

Ejemplo: Saludo

SALIDA ESPERADA

Hola, Ana. Tienes 3 mensajes.

7. Las palabras más repetidas

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Lee un texto (puede tener varias líneas) y muestra las tres palabras más frecuentes con el formato «palabra: veces», en minúsculas. Una palabra es una secuencia de letras (incluidas las tildes y la ñ). Si dos palabras se repiten lo mismo, va primero la que sea menor alfabéticamente.

Código de partida (php)

```
<?php
$texto = stream_get_contents(STDIN);

// TODO
```

Ejemplo: Refrán

ENTRADA

El que mucho abarca poco aprieta, y el que poco abarca mucho aprieta.

SALIDA ESPERADA

abarca: 2
aprieta: 2
el: 2

Ejemplo: Mayúsculas y signos

ENTRADA

¡PHP! php, Php... Java; java. SQL

SALIDA ESPERADA

php: 3
java: 2
sql: 1

8. Ticket de compra

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Cada línea de la entrada es «producto;precio;cantidad». Escribe una línea por producto con el formato «Teclado x 2 = 49,98 €» y al final la base, el IVA y el total: «Base: 62,48 €», «IVA (21%): 13,12 €», «Total: 75,60 €». Las cantidades usan coma decimal y punto de miles (1.234,50 €). Las líneas que no tengan tres campos, un precio numérico y una cantidad entera se ignoran mostrando «Línea ignorada: ...».

Código de partida (php)

```
<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);

// TODO
```

Ejemplo: Compra normal

ENTRADA

Teclado;24.99;2
Ratón;12.50;1

SALIDA ESPERADA

Teclado x 2 = 49,98 €
Ratón x 1 = 12,50 €
Base: 62,48 €
IVA (21%): 13,12 €
Total: 75,60 €

Ejemplo: Líneas erróneas

ENTRADA

Monitor;189;1
Cable;barato;3
Funda;9.90
Portátil;999;1

SALIDA ESPERADA

Monitor x 1 = 189,00 €
Línea ignorada: Cable;barato;3
Línea ignorada: Funda;9.90
Portátil x 1 = 999,00 €
Base: 1.188,00 €
IVA (21%): 249,48 €
Total: 1.437,48 €

9. Convertir y validar los datos de un formulario

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Todo lo que llega de un formulario es texto, y convertirlo bien es la primera defensa. Cada línea de la entrada es «campo=valor». Valida y convierte según el campo: edad (entero de 0 a 120), email (dirección válida), nota (decimal de 0 a 10; admite coma o punto decimal) y web (URL que empiece por http o https). Para cada línea escribe «campo: valor convertido (tipo)» usando el tipo de PHP (int, float o string), o «campo: no válido». La nota se muestra con dos decimales. Los campos desconocidos se ignoran.

Código de partida (php)

```
<?php
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$campo, $valor] = explode('=', $linea, 2) + [1 => ''];
    $valor = trim($valor);

    // TODO: convierte según el campo; false si no es válido
    $resultado = false;

    if ($resultado === false) {
        echo "$campo: no válido\n";
    } else {
        echo "$campo: $resultado (" . get_debug_type($resultado) . ")\n";
    }
}
```

Ejemplo: Datos correctos

ENTRADA

```
edad=34
email=ana@correo.es
nota=7,5
web=https://apuntesdam.com
```

SALIDA ESPERADA

```
edad: 34 (int)
email: ana@correo.es (string)
nota: 7.50 (float)
web: https://apuntesdam.com (string)
```

Ejemplo: Datos incorrectos

ENTRADA

```
edad=treinta
edad=150
email=ana@
nota=11
web=ftp://ficheros.es
color=rojo
nota=abc
```

SALIDA ESPERADA

```
edad: no válido
edad: no válido
email: no válido
nota: no válido
web: no válido
nota: no válido
```

10. Encuentra el fallo: la subida de precios que no sube

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

El programa debería subir todos los precios un 10 % (redondeado a dos decimales) y listarlos, pero salen los precios de antes. Corrígelo.

Código de partida (php)

```
<?php
$precios = ["pan" => 1.20, "leche" => 0.95, "café" => 4.50];
foreach ($precios as $p) {
    $p = round($p * 1.10, 2);
}
foreach ($precios as $producto => $p) {
    echo "$producto: $p €\n";
}
```

Ejemplo: Tres productos

SALIDA ESPERADA

```
pan: 1.32 €
leche: 1.05 €
café: 4.95 €
```

11. Calendario del mes en HTML

Difícil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

La entrada es un mes con el formato AAAA-MM. Genera su calendario como tabla HTML con la semana empezando en lunes: primero «<table>», una fila de cabecera «<tr><th>L</th><th>M</th><th>X</th><th>J</th><th>V</th><th>S</th><th>D</th></tr>», después una fila por semana con siete celdas (las celdas fuera del mes van vacías: «<td></td>»; los días, «<td>5</td>») y al final «</table>». Si la entrada no es un mes válido escribe «<p>Mes no válido</p>».

Código de partida (php)

```
<?php
$texto = trim((string) fgets(STDIN));

// TODO
```

Ejemplo: Septiembre de 2026

ENTRADA

2026-09

SALIDA ESPERADA

```
<table>
  <tr><th>L</th><th>M</th><th>X</th><th>J</th>
  <th>V</th><th>S</th><th>D</th></tr>
  <tr><td></td><td>1</td><td>2</td><td>3</td>
  <td>4</td><td>5</td><td>6</td></tr>
  <tr><td>7</td><td>8</td><td>9</td><td>10</td>
  <td>11</td><td>12</td><td>13</td></tr>
  <tr><td>14</td><td>15</td><td>16</td>
  <td>17</td><td>18</td><td>19</td><td>20</td></tr>
  <tr><td>21</td><td>22</td><td>23</td>
  <td>24</td><td>25</td><td>26</td><td>27</td></tr>
  <tr><td>28</td><td>29</td><td>30</td><td></td>
  <td></td><td></td><td></td></tr>
</table>
```

Ejemplo: Mes no válido

ENTRADA

2026-13

SALIDA ESPERADA

```
<p>Mes no válido</p>
```

12. Tu propio motor de plantillas

Muy difícil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Twig o Blade separan el HTML del código con un lenguaje de plantillas. Construye uno pequeño. La primera línea de la entrada son los datos en JSON y el resto, la plantilla. Soporta: {{ nombre }} y {{ usuario.email }} (valor escapado con htmlspecialchars; si no existe, cadena vacía), {{{ html }}} (sin escapar), {% for x in lista %}...{% endfor %} (sin anidar otros for) y {% if var %}...{% endif %} (se muestra si el valor es «verdadero» en PHP; puede ir dentro de un for). Escribe la plantilla ya procesada.

Código de partida (php)

```
<?php
$datos = json_decode(fgets(STDIN), true);
$plantilla = stream_get_contents(STDIN);

function renderizar(string $tpl, array $ctx): string
{
    // TODO
    return $tpl;
}

echo renderizar($plantilla, $datos);
```

Ejemplo: Variables y escape

ENTRADA

```
{ "nombre": "Ana <3", "usuario":  
  { "email": "ana@correo.es", "firma": "  
    <b>Equipo</b>" }  
<p>Hola, {{ nombre }} ({{ usuario.email }})</p>  
{{ firma }} {{ firma }} [{{ falta }}]
```

SALIDA ESPERADA

```
<p>Hola, Ana &lt;3 (ana@correo.es)</p>  
<b>Equipo</b> &lt;b>Equipo&lt;/b> [ ]
```

Ejemplo: Bucles y condiciones

ENTRADA

```
{ "titulo": "Pedidos", "pedidos":  
  [ { "id": 1, "urgente": true },  
    { "id": 2, "urgente": false },  
    { "id": 3, "urgente": 1 } ], "vacio": [ ] }  
<h1>{{ titulo }}</h1>  
<ul>{% for p in pedidos %}<li>#{{ p.id }}{% if  
  p.urgente %} ¡urgente!{% endif %}</li>{% endfor  
%}</ul>  
{% if vacio %}Hay datos{% endif %}{% for x in  
  vacio %}nunca{% endfor %}fin
```

SALIDA ESPERADA

```
<h1>Pedidos</h1>  
<ul><li>#1 ¡urgente!</li><li>#2</li><li>#3  
¡urgente!</li></ul>  
fin
```

Formularios, Sesiones y Seguridad

13. Comentarios sin XSS

Fácil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Cada línea es un comentario «autor|texto» que han escrito los usuarios. El código ya los muestra como , pero es vulnerable: un usuario podría colar HTML o JavaScript. Corrígelo para que los caracteres especiales se escapen (< > & " ').

Código de partida (php)

```
<?php  
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {  
    if (!str_contains($linea, '|')) {  
        continue;  
    }  
    [$autor, $texto] = explode('|', $linea, 2);  
    // TODO: escapa los datos antes de mostrarlos  
    echo "<li><strong>$autor</strong>: $texto</li>\n";  
}
```

Ejemplo: Comentarios normales

ENTRADA

```
Ana|Muy buen tema  
Luis|Gracias
```

SALIDA ESPERADA

```
<li><strong>Ana</strong>: Muy buen tema</li>  
<li><strong>Luis</strong>: Gracias</li>
```

Ejemplo: Intento de ataque

ENTRADA

```
Hacker|<script>robar(document.cookie)</script>  
Eva|Uso <b>negrita</b> & "comillas" y 'simples'
```

SALIDA ESPERADA

```
<li><strong>Hacker</strong>:  
&lt;script&gt;robar(document.cookie)&lt;/script&  
t;</li>  
<li><strong>Eva</strong>: Uso  
&lt;b&gt;negrita&lt;/b&gt; &amp;  
&quot;comillas&quot; y &#039;simples&#039;</li>
```

14. Validar los datos de un formulario

Medio · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

La entrada es el cuerpo de un formulario enviado por POST (nombre=Ana&email=...&edad=...&web=...). Valida y escribe un error por línea, en este orden: «nombre: obligatorio» o «nombre: máximo 50 caracteres»; «email: no válido»; «edad: debe estar entre 18 y 120» (entero); «web: no válida» (es opcional, solo se valida si no está vacía). Si no hay errores, escribe «Datos correctos».

Código de partida (php)

```
<?php
parse_str(trim((string) fgets(STDIN)), $datos);
$errores = [];

// TODO
```

Ejemplo: Formulario correcto

ENTRADA

```
nombre=Ana+García&email=ana%40correo.es&edad=20&web=
```

SALIDA ESPERADA

```
Datos correctos
```

Ejemplo: Todo mal

ENTRADA

```
nombre=++&email=ana&edad=17&web=hola
```

SALIDA ESPERADA

```
nombre: obligatorio
email: no válido
edad: debe estar entre 18 y 120
web: no válida
```

Ejemplo: Nombre largo y edad decimal

ENTRADA

```
nombre=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxx&email=a%40b.es&edad=20.5&web=https%3A%2F%2Fapuntesdam.com
```

SALIDA ESPERADA

```
nombre: máximo 50 caracteres
edad: debe estar entre 18 y 120
```

15. Un carrito guardado en la sesión

Medio · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Simulamos \$_SESSION con un array. Cada línea es una acción del usuario: «añadir id cantidad», «quitar id», «vaciar» o «ver». El catálogo tiene precios y stock. Al añadir, la cantidad debe ser un entero positivo y el total de ese producto en el carrito no puede superar el stock; si no se cumple, escribe «Error: ...» («cantidad no válida», «producto inexistente» o «stock insuficiente») y no cambies nada. «ver» muestra una línea por producto en el orden en que se añadieron («Teclado x 2 = 49,98 €») y «Total: ...», o «Carrito vacío».

Código de partida (php)

```
<?php
$catalogo = [
    't1' => ['nombre' => 'Teclado', 'precio' => 24.99, 'stock' => 5],
    'r1' => ['nombre' => 'Ratón', 'precio' => 12.50, 'stock' => 2],
    'm1' => ['nombre' => 'Monitor', 'precio' => 189.00, 'stock' => 1],
];
$_SESSION = ['carrito' => []];
$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    $partes = explode(' ', trim($linea));
    $accion = $partes[0];
    // TODO
}
```

Ejemplo: Añadir y ver

ENTRADA

```
anadir t1 2
anadir r1 1
anadir t1 1
ver
```

SALIDA ESPERADA

```
Teclado x 3 = 74,97 €
Ratón x 1 = 12,50 €
Total: 87,47 €
```

Ejemplo: Errores y quitar

ENTRADA

```
anadir x9 1
anadir t1 0
anadir t1 dos
anadir r1 2
anadir r1 1
anadir m1 1
quitar r1
ver
vaciar
ver
```

SALIDA ESPERADA

```
Error: producto inexistente
Error: cantidad no válida
Error: cantidad no válida
Error: stock insuficiente
Monitor x 1 = 189,00 €
Total: 189,00 €
Carrito vacío
```

16. Encuentra el fallo: el comentario que ejecuta código

Medio · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

La función pinta un comentario de un formulario. Si alguien escribe una etiqueta <script>, se mete en la página tal cual (XSS). Corrige la función para que el autor y el texto se muestren escapados.

Código de partida (php)

```
<?php
function tarjeta(string $autor, string $texto): string {
    return "<p><b>$autor</b>: $texto</p>";
}

echo tarjeta("Ana", "¡Me encanta!"); . "\n";
echo tarjeta("Pirata", "<script>alert('hola')</script>") . "\n";
```

Ejemplo: Comentario normal y comentario con script

SALIDA ESPERADA

```
<p><b>Ana</b>: ¡Me encanta!</p>
<p><b>Pirata</b>: &lt;script&gt;alert(&#039;hola&#039;)&lt;/script&gt;</p>
```

17. Inicio de sesión con bloqueo

Difícil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Las contraseñas se guardan cifradas con `password_hash`. Cada línea de la entrada es un intento «usuario contraseña» (la contraseña es lo que va después del primer espacio). Escribe «Bienvenido/a, usuario» si es correcto o «Usuario o contraseña incorrectos» si no (sin revelar si el usuario existe). Tras 3 fallos seguidos de un mismo usuario, su cuenta queda bloqueada: responde «Cuenta bloqueada» a cualquier intento posterior, aunque la contraseña sea correcta. Un acceso correcto reinicia sus fallos.

Código de partida (php)

```
<?php
$usuarios = [
    'ana' => password_hash('Secreta123', PASSWORD_DEFAULT),
    'luis' => password_hash('DAW 2025!', PASSWORD_DEFAULT),
];
$fallos = [];

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$usuario, $clave] = array_pad(explode(' ', $linea, 2), 2, '');
    // TODO
}
```

Ejemplo: Accesos correctos e incorrectos

ENTRADA

```
ana Secreta123
ana secreta123
luis DAW 2025!
pepe 1234
```

SALIDA ESPERADA

```
Bienvenido/a, ana
Usuario o contraseña incorrectos
Bienvenido/a, luis
Usuario o contraseña incorrectos
```

Ejemplo: Bloqueo tras tres fallos

ENTRADA

```
ana a
ana b
ana c
ana Secreta123
luis DAW 2025!
```

SALIDA ESPERADA

```
Usuario o contraseña incorrectos
Usuario o contraseña incorrectos
Usuario o contraseña incorrectos
Cuenta bloqueada
Bienvenido/a, luis
```

18. Cookies firmadas

Difícil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

El usuario puede cambiar sus cookies, pero sí podemos detectar si lo ha hecho. Firmaremos cada valor con un HMAC: la cookie guarda «valor.firma», donde firma es `hash_hmac('sha256', valor, clave secreta)`. Cada línea es «firmar texto» (escribe la cookie resultante) o «verificar cookie» (escribe «Válida: valor» o «Manipulada»). La clave secreta es la del código. Compara las firmas con `hash_equals`, que tarda lo mismo acierte o falle y no da pistas a un atacante. El valor puede contener puntos: la firma es lo que va tras el último punto.

Código de partida (php)

```
<?php
const CLAVE = 'b7f1c0a9e2d34f5a8c6b1e0d9f2a4c7e';

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$orden, $dato] = explode(' ', $linea, 2) + [1 => ''];
    // TODO
}
```

Ejemplo: Firmar y verificar

ENTRADA

```
firmar tema=oscuro
firmar idioma=es
verificar
tema=oscuro.4a4266b175ca852708e16e7683c741ffab3f8
89751364ef5d7fbc1f91ee6922
```

SALIDA ESPERADA

```
tema=oscuro.4a4266b175ca852708e16e7683c741ffab3f8
89751364ef5d7fbc1f91ee6922
idioma=es.da4e78e48d02ba709bc82f09327775d9dd6e896
a0c3c107193e13c25c9aba3a6
Válida: tema=oscuro
```

Ejemplo: Manipulaciones

ENTRADA

```
verificar
tema=claro.4a4266b175ca852708e16e7683c741ffab3f88
9751364ef5d7fbc1f91ee6922
verificar
tema=oscuro.4a4266b175ca852708e16e7683c741ffab3f8
89751364ef5d7fbc1f91ee6922
verificar sinpunto
verificar
v1.2.c481f604fe4fafcded64054280f87bbb1002ae4cd73e
5984b77b5de8275c2ac3
```

SALIDA ESPERADA

```
Manipulada
Manipulada
Manipulada
Válida: v1.2
```

19. Control de acceso por roles y propiedad

Muy difícil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

La entrada tiene tres bloques separados por «---». Primero, los roles: «rol NOMBRE: permiso1, permiso2» o «rol NOMBRE hereda PADRE: permisos» (un rol tiene sus permisos y todos los de sus antepasados). Después, los usuarios: «NOMBRE: ROL». Por último, las peticiones: «USUARIO ACCION» o «USUARIO ACCION PROPIETARIO». Un usuario puede hacer ACCION si su rol tiene el permiso ACCION, o si tiene ACCION_propio y el recurso es suyo (PROPIETARIO es él). Escribe «USUARIO ACCION: permitido» o «... denegado». Un usuario desconocido siempre recibe denegado. Así se evitan los fallos de control de acceso, los primeros del OWASP Top 10.

Código de partida (php)

```
<?php
$bloques = explode("---", stream_get_contents(STDIN));
$lineas = fn(string $b) => array_filter(array_map('trim', explode("\n", $b)));

// TODO: roles, usuarios y peticiones
```

Ejemplo: Herencia y propiedad

ENTRADA

```
rol lector: leer
rol autor hereda lector: crear, editar_propio,
borrar_propio
rol editor hereda autor: editar, publicar
---
ana: autor
luis: editor
eva: lector
---
ana leer
ana editar luis
ana editar ana
ana publicar
luis editar ana
luis borrar ana
eva crear
pepe leer
```

SALIDA ESPERADA

```
ana leer: permitido
ana editar: denegado
ana editar: permitido
ana publicar: denegado
luis editar: permitido
luis borrar: denegado
eva crear: denegado
pepe leer: denegado
```

Ejemplo: Varios niveles

ENTRADA

```
rol invitado: ver_catalogo
rol cliente hereda invitado: comprar,
ver_pedido propio, cancelar_pedido propio
rol soporte hereda cliente: ver_pedido
rol admin hereda soporte: cancelar_pedido,
gestionar_usuarios
---
marta: cliente
jorge: soporte
sara: admin
---
marta ver_catalogo
marta ver_pedido jorge
marta ver_pedido marta
jorge ver_pedido marta
jorge cancelar_pedido marta
jorge cancelar_pedido jorge
sara cancelar_pedido marta
sara gestionar_usuarios
```

SALIDA ESPERADA

```
marta ver_catalogo: permitido
marta ver_pedido: denegado
marta ver_pedido: permitido
jorge ver_pedido: permitido
jorge cancelar_pedido: denegado
jorge cancelar_pedido: permitido
sara cancelar_pedido: permitido
sara gestionar_usuarios: permitido
```

POO y el Patrón MVC

20. Una entidad que se valida sola

Medio · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Completa la clase Producto: el constructor usa promoción de propiedades (string \$nombre, float \$precio, int \$stock = 0) y lanza InvalidArgumentException con «El nombre es obligatorio», «El precio no puede ser negativo» o «El stock no puede ser negativo». precioConIva() devuelve el precio con el 21 %. Al convertirlo en texto (__toString) debe dar «Teclado: 24,99 € (30,24 € con IVA) · stock 5». El programa ya lee líneas «nombre;precio;stock» y muestra el producto o «Error: mensaje».

Código de partida (php)

```
<?php
class Producto
{
    // TODO: constructor con promoción de propiedades y validación, y precioConIva()

    public function __toString(): string
    {
        return ''; // TODO
    }
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$nombre, $precio, $stock] = array_pad(explode(';', $linea), 3, '0');
    try {
        echo new Producto(trim($nombre), (float) $precio, (int) $stock), "\n";
    } catch (InvalidArgumentException $e) {
        echo 'Error: ', $e->getMessage(), "\n";
    }
}
```

Ejemplo: Productos válidos

ENTRADA

```
Teclado;24.99;5
Monitor;189;0
```

SALIDA ESPERADA

```
Teclado: 24,99 € (30,24 € con IVA) · stock 5
Monitor: 189,00 € (228,69 € con IVA) · stock 0
```

Ejemplo: Errores de validación

ENTRADA

```
;10;1  
Ratón;-3;2  
Cable;2;-1
```

SALIDA ESPERADA

```
Error: El nombre es obligatorio  
Error: El precio no puede ser negativo  
Error: El stock no puede ser negativo
```

21. Nóminas con herencia e interfaces

Medio · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Completa la jerarquía. Empleado es abstracta: guarda el nombre y obliga a implementar salario(). Fijo cobra su sueldo base más 50 € por cada trienio (3 años completos de antigüedad); PorHoras cobra horas × precio, y las que pasen de 160 a precio × 1,5. Comercial extiende Fijo y además cobra un 5 % de sus ventas. Los que implementan la interfaz Bonificable (solo Comercial) suman bonus() a su salario en la nómina. Cada línea de la entrada es «tipo;nombre;datos...». Escribe «nombre: salario €» para cada uno, y al final «Total: ...».

Código de partida (php)

```
<?php
interface Bonificable
{
    public function bonus(): float;
}

abstract class Empleado
{
    public function __construct(public readonly string $nombre) {}

    abstract public function salario(): float;
}

// TODO: completa Fijo, PorHoras y Comercial (que implementa Bonificable: bonus de 100 € si vende más de
// 20.000 €)
class Fijo extends Empleado
{
    public function __construct(string $nombre, protected float $base, protected int $antiguedad)
    {
        parent::__construct($nombre);
    }

    public function salario(): float
    {
        return 0; // TODO
    }
}

class PorHoras extends Empleado
{
    public function __construct(string $nombre, private float $horas, private float $precio)
    {
        parent::__construct($nombre);
    }

    public function salario(): float
    {
        return 0; // TODO
    }
}

class Comercial extends Fijo // TODO: implements Bonificable
{
    public function __construct(string $nombre, float $base, int $antiguedad, private float $ventas)
    {
        parent::__construct($nombre, $base, $antiguedad);
    }
}

$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';
$total = 0;
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    $d = explode(';', $linea);
    $e = match ($d[0]) {
        'fijo' => new Fijo($d[1], (float) $d[2], (int) $d[3]),
        'horas' => new PorHoras($d[1], (float) $d[2], (float) $d[3]),
        'comercial' => new Comercial($d[1], (float) $d[2], (int) $d[3], (float) $d[4]),
    };
    $importe = $e->salario() + ($e instanceof Bonificable ? $e->bonus() : 0);
    $total += $importe;
    echo "{$e->nombre}: {$euros($importe)}\n";
}
echo "Total: {$euros($total)}\n";
```

Ejemplo: Plantilla

ENTRADA

```
fijo;Ana;1800;7
horas;Luis;170;12
comercial;Eva;1200;2;25000
```

SALIDA ESPERADA

```
Ana: 1.900,00 €
Luis: 2.100,00 €
Eva: 2.550,00 €
Total: 6.550,00 €
```

Ejemplo: Sin extras

ENTRADA

```
fijo;Marta;2000;2
horas;Pablo;100;15
comercial;Iker;1000;3;20000
```

SALIDA ESPERADA

```
Marta: 2.000,00 €
Pablo: 1.500,00 €
Iker: 2.050,00 €
Total: 5.550,00 €
```

22. Encuentra el fallo: los intereses que no llegan

Medio · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

CuentaAhorro hereda de Cuenta y añade intereses, pero tras ingresar 1000 € y aplicar un 5 % el saldo sigue en 1000 € y aparecen avisos. Corrígelo cambiando lo mínimo.

Código de partida (php)

```
<?php
class Cuenta {
    private float $saldo = 0;
    public function ingresar(float $c): void { $this->saldo += $c; }
    public function getSaldo(): float { return $this->saldo; }
}

class CuentaAhorro extends Cuenta {
    public function aplicarIntereses(float $tipo): void {
        $this->saldo += $this->saldo * $tipo / 100;
    }
}

$c = new CuentaAhorro();
$c->ingresar(1000);
$c->aplicarIntereses(5);
echo "Saldo: " . $c->getSaldo() . " €\n";
```

Ejemplo: 1000 € al 5 %

SALIDA ESPERADA

```
Saldo: 1050 €
```

23. Un motor de plantillas mínimo

Difícil · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Separar la presentación de la lógica es la base de MVC. Completa `render($plantilla, $datos)`: sustituye cada `{{ clave }}` por el valor de `$datos` escapado con `htmlspecialchars`, y `{{ clave|raw }}` por el valor sin escapar. Las claves pueden tener puntos para entrar en arrays anidados (`usuario.nombre`). Si una clave no existe, se sustituye por una cadena vacía. La primera línea de la entrada son los datos en JSON y el resto la plantilla.

Código de partida (php)

```
<?php
function render(string $plantilla, array $datos): string
{
    // TODO
    return $plantilla;
}

$datos = json_decode((string) fgets(STDIN), true) ?? [];
echo render(stream_get_contents(STDIN), $datos);
```

Ejemplo: Variables simples

ENTRADA

```
{"titulo": "Hola", "nombre": "Ana"}
<h1>{{ titulo }}</h1>
<p>Bienvenida, {{nombre}}</p>
```

SALIDA ESPERADA

```
<h1>Hola</h1>
<p>Bienvenida, Ana</p>
```

Ejemplo: Escapado, raw y anidados

ENTRADA

```
{"usuario": {"nombre": "<Luis>", "rol": "admin"},
 "aviso": "<b>Nuevo</b>"}
<p>{{ usuario.nombre }} ({{ usuario.rol }})</p>
{{ aviso|raw }} {{ aviso }}
```

SALIDA ESPERADA

```
<p>&lt;Luis&gt; (admin)</p>
<b>Nuevo</b> &lt;b&gt;Nuevo&lt;/b&gt;
```

24. Una cadena de middlewares

Difícil · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Los frameworks procesan cada petición con una cadena de middlewares: cada uno puede hacer algo antes, pasar la petición al siguiente con `$siguiente($peticion)` y hacer algo después, o cortar la cadena devolviendo su propia respuesta. Completa `ejecutar()`, que envuelve el controlador con los middlewares en orden (el primero es el más externo). Los middlewares ya están escritos: registro (anota la entrada y la salida), autenticación (401 si no hay usuario) y administración (403 si la ruta empieza por `/admin` y el rol no es admin). Cada línea de la entrada es «ruta usuario rol» (o «ruta - -» sin usuario).

Código de partida (php)

```
<?php
$log = [];

$registro = function (array $p, callable $siguiente) use (&$log) {
    $log[] = "→ {$p['ruta']}";
    $respuesta = $siguiente($p);
    $log[] = "← {$respuesta[0]}";
    return $respuesta;
};

$autenticacion = fn(array $p, callable $siguiente) => $p['usuario'] === null ? [401, 'Inicia sesión'] :
    $siguiente($p);
$administracion = fn(array $p, callable $siguiente) =>
    str_starts_with($p['ruta'], '/admin') && $p['rol'] !== 'admin' ? [403, 'Prohibido'] : $siguiente($p);
$controlador = fn(array $p) => [200, "Hola, {$p['usuario']}": {$p['ruta']}"];

function ejecutar(array $middlewares, callable $controlador, array $peticion): array
{
    // TODO
    return $controlador($peticion);
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$ruta, $usuario, $rol] = explode(' ', $linea);
    $peticion = ['ruta' => $ruta, 'usuario' => $usuario === '-' ? null : $usuario, 'rol' => $rol];
    [$codigo, $cuerpo] = ejecutar([$registro, $autenticacion, $administracion], $controlador, $peticion);
    echo "$codigo $cuerpo\n";
}
echo implode(' ', $log), "\n";
```

Ejemplo: Peticiones variadas

ENTRADA

```
/perfil ana usuario
/admin/usuarios ana usuario
/admin/usuarios luis admin
/perfil - -
```

SALIDA ESPERADA

```
200 Hola, ana: /perfil
403 Prohibido
200 Hola, luis: /admin/usuarios
401 Inicia sesión
→ /perfil ← 200 → /admin/usuarios ← 403 →
/admin/usuarios ← 200 → /perfil ← 401
```

Ejemplo: Solo anónimos

ENTRADA

```
/admin - -
/ - -
```

SALIDA ESPERADA

```
401 Inicia sesión
401 Inicia sesión
→ /admin ← 401 → / ← 401
```

25. Un contenedor de dependencias con autowiring

Muy difícil · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Laravel y Symfony crean los objetos por ti: miran el constructor de una clase, ven qué necesita y lo crean primero. Completa la clase Contenedor. get(\$clase) devuelve siempre la misma instancia de cada clase (singleton). Para crearla, usa ReflectionClass para leer los parámetros del constructor: si el tipo es una clase, la resuelve recursivamente con get; si es una interfaz, usa la implementación registrada con enlazar(); si es un tipo simple con valor por defecto, usa ese valor. Si detecta un ciclo (A necesita B y B necesita A) lanza LogicException «Dependencia circular: A → B → A». Si una interfaz no está enlazada, LogicException «Sin implementación para X». El programa ya muestra el orden en que se construyen los objetos.

Código de partida (php)

```
<?php
$orderen = [];

interface Almacen {}
final class AlmacenMemoria implements Almacen { public function __construct() { global $orden; $orden[] = 'AlmacenMemoria'; } }
final class Reloj { public function __construct(public string $zona = 'Europe/Madrid') { global $orden; $orden[] = 'Reloj'; } }
final class Registro { public function __construct(Reloj $reloj) { global $orden; $orden[] = 'Registro'; } }
final class Repositorio { public function __construct(Almacen $almacen, Registro $registro) { global $orden; $orden[] = 'Repositorio'; } }
final class Servicio { public function __construct(Repositorio $repo, Registro $registro, int $reintentos = 3) { global $orden; $orden[] = "Servicio($reintentos)"; } }
final class Huevo { public function __construct(Gallina $g) {} }
final class Gallina { public function __construct(Huevo $h) {} }
interface Correo {}
final class Avisos { public function __construct(Correo $c) {} }

final class Contenedor
{
    private array $instancias = [];
    private array $enlaces = [];

    public function enlazar(string $interfaz, string $clase): void
    {
        $this->enlaces[$interfaz] = $clase;
    }

    public function get(string $clase): object
    {
        // TODO
        return new $clase();
    }
}

$c = new Contenedor();
$c->enlazar(Almacen::class, AlmacenMemoria::class);
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $pedido) {
    $orden = [];
    try {
        $a = $c->get($pedido);
        echo "$pedido: " . (implode(', ', $orden) ?: 'ya existía') . ($a === $c->get($pedido) ? ' (misma instancia)' : '') . "\n";
    } catch (Throwable $e) {
        echo "$pedido: " . get_class($e) . ': ' . $e->getMessage() . "\n";
    }
}
```

Ejemplo: Árbol de dependencias

ENTRADA

Servicio
Repositorio
Almacen

SALIDA ESPERADA

Servicio: AlmacenMemoria, Reloj, Registro, Repositorio, Servicio(3) (misma instancia)
Repositorio: ya existía (misma instancia)
Almacen: ya existía (misma instancia)

Ejemplo: Errores

ENTRADA

Huevo
Avisos
Reloj

SALIDA ESPERADA

Huevo: LogicException: Dependencia circular:
Huevo → Gallina → Huevo
Avisos: LogicException: Sin implementación para Correo
Reloj: Reloj (misma instancia)

Acceso a Datos con PDO

26. Encuentra el fallo: el listado de un solo producto

Fácil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

La consulta devuelve tres productos ordenados por precio, pero el programa solo muestra el primero. Corrígelo para listarlos todos.

Código de partida (php)

```
<?php
$pdo = new PDO("sqlite::memory:");
$pdo->exec("CREATE TABLE productos (nombre TEXT, precio INTEGER)");
$pdo->exec("INSERT INTO productos VALUES ('Teclado', 25), ('Ratón', 12), ('Monitor', 150)");
$stmt = $pdo->query("SELECT nombre, precio FROM productos ORDER BY precio");
if ($fila = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "{$fila['nombre']}: {$fila['precio']} €\n";
}
```

Ejemplo: Tres productos

SALIDA ESPERADA

```
Ratón: 12 €
Teclado: 25 €
Monitor: 150 €
```

27. Buscar sin inyección SQL

Medio · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

El programa lee una categoría y lista sus productos ordenados por precio («Teclado - 24,99 €»), o «Sin resultados».

Funciona, pero construye la consulta pegando el texto del usuario: prueba a buscar ' OR '1'='1 y verás todos los productos. Reescríbelo con una consulta preparada.

Código de partida (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec("CREATE TABLE productos (id INTEGER PRIMARY KEY, nombre TEXT, categoria TEXT, precio REAL)");
$pdo->exec("INSERT INTO productos (nombre, categoria, precio) VALUES
    ('Teclado', 'perifericos', 24.99), ('Ratón', 'perifericos', 12.5),
    ('Monitor 27', 'pantallas', 189), ('Portátil', 'equipos', 749),
    ('Alfombrilla', 'perifericos', 6.9), ('Monitor 24', 'pantallas', 129)");

$categoria = trim((string) fgets(STDIN));

// TODO: esta consulta es vulnerable a inyección SQL
$filas = $pdo->query("SELECT nombre, precio FROM productos WHERE categoria = '$categoria' ORDER BY
    precio")->fetchAll();

if (!$filas) {
    echo "Sin resultados\n";
}
foreach ($filas as $f) {
    echo $f['nombre'], ' - ', number_format($f['precio'], 2, ',', '.'), " €\n";
}
```

Ejemplo: Búsqueda normal

ENTRADA

perifericos

SALIDA ESPERADA

Alfombrilla - 6,90 €
Ratón - 12,50 €
Teclado - 24,99 €

Ejemplo: Intento de inyección

ENTRADA

' OR '1'='1

SALIDA ESPERADA

Sin resultados

28. Pagar resultados con LIMIT

Medio · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

La tabla alumnos tiene 23 filas. La entrada es «página porPágina» (por ejemplo «2 5»). Muestra los nombres de esa página ordenados por id, uno por línea, y después «Página 2 de 5». Si la página o el tamaño no son enteros positivos, o la página no existe, muestra «Página fuera de rango».

Código de partida (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec("CREATE TABLE alumnos (id INTEGER PRIMARY KEY, nombre TEXT)");
$insertar = $pdo->prepare('INSERT INTO alumnos (nombre) VALUES (?)');
for ($i = 1; $i <= 23; $i++) {
    $insertar->execute(["Alumno $i"]);
}

[$pagina, $tamano] = array_pad(preg_split('/\s+/', trim((string) fgets(STDIN))), 2, '');
// TODO
```

Ejemplo: Segunda página

ENTRADA

2 5

SALIDA ESPERADA

Alumno 6
Alumno 7
Alumno 8
Alumno 9
Alumno 10
Página 2 de 5

Ejemplo: Última página incompleta

ENTRADA

3 10

SALIDA ESPERADA

Alumno 21
Alumno 22
Alumno 23
Página 3 de 3

Ejemplo: Fuera de rango

ENTRADA

6 5

SALIDA ESPERADA

Página fuera de rango

29. Filtros dinámicos sin inyección

Medio · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Cada línea de la entrada es un filtro opcional «clave=valor»: categoría, min (precio mínimo), max (precio máximo), texto (el nombre lo contiene) y orden (precio, precio_desc o nombre; cualquier otro valor se ignora y se ordena por nombre). Construye una única consulta preparada con solo los filtros recibidos y lista «nombre - precio €», o «Sin resultados». El texto se busca literalmente: los caracteres % y _ del usuario no deben funcionar como comodines de LIKE.

Código de partida (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC]);
$pdo->exec("CREATE TABLE productos (id INTEGER PRIMARY KEY, nombre TEXT, categoria TEXT, precio REAL)");
$pdo->exec("INSERT INTO productos (nombre, categoria, precio) VALUES
('Teclado mecánico', 'perifericos', 59.9), ('Ratón', 'perifericos', 12.5), ('Monitor 27',
'pantallas', 189),
('Monitor 24', 'pantallas', 129), ('Cable USB_C', 'cables', 9.95), ('Cable USBx_C', 'cables', 7.5),
('Pack descuento 100%', 'ofertas', 99), ('Pack 1000', 'ofertas', 45), ('Alfombrilla', 'perifericos',
6.9)");

$filtros = [];
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$clave, $valor] = explode('=', $linea, 2) + [1 => ''];
    $filtros[$clave] = $valor;
}

// TODO: construye la consulta con los filtros
$filas = $pdo->query('SELECT nombre, precio FROM productos ORDER BY nombre')->fetchAll();

echo $filas ? '' : "Sin resultados\n";
foreach ($filas as $f) {
    echo $f['nombre'], ' - ', number_format($f['precio'], 2, ',', '.'), " €\n";
}
```

Ejemplo: Categoría y precio

ENTRADA

```
categoria=perifericos
max=50
orden=precio_desc
```

SALIDA ESPERADA

```
Ratón - 12,50 €
Alfombrilla - 6,90 €
```

Ejemplo: Comodines literales

ENTRADA

```
texto=USB_
orden=precio
```

SALIDA ESPERADA

```
Cable USB_C - 9,95 €
```

Ejemplo: Inyección y orden raro

ENTRADA

```
categoria=' OR '1'='1
orden=precio; DROP TABLE productos
```

SALIDA ESPERADA

```
Sin resultados
```

30. Transferencias con transacciones

Difícil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Cada línea es una transferencia «origen destino importe». Para cada una escribe «Cuenta inexistente», «Importe no válido» (menor o igual que 0 o no numérico), «Saldo insuficiente» o «Transferencia realizada», en ese orden de comprobación. Las dos actualizaciones de saldo deben hacerse dentro de una transacción. Al final, lista las cuentas ordenadas por id: «ES01 Ana: 400,00 €».

Código de partida (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec("CREATE TABLE cuentas (id TEXT PRIMARY KEY, titular TEXT, saldo REAL)");
$pdo->exec("INSERT INTO cuentas VALUES ('ES01', 'Ana', 500), ('ES02', 'Luis', 120), ('ES03', 'Marta', 0)");

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$origen, $destino, $importe] = array_pad(preg_split('/\s+/', trim($linea)), 3, '');
    // TODO
}

foreach ($pdo->query('SELECT * FROM cuentas ORDER BY id') as $c) {
    echo "{$c['id']} {$c['titular']}: ", number_format($c['saldo'], 2, ',', '.'), " €\n";
}
```

Ejemplo: Varias operaciones

ENTRADA

```
ES01 ES02 100
ES02 ES03 300
ES03 ES01 0
ES09 ES01 10
```

SALIDA ESPERADA

```
Transferencia realizada
Saldo insuficiente
Importe no válido
Cuenta inexistente
ES01 Ana: 400,00 €
ES02 Luis: 220,00 €
ES03 Marta: 0,00 €
```

Ejemplo: Encadenadas

ENTRADA

```
ES01 ES03 250.5
ES03 ES02 50.25
ES02 ES01 abc
```

SALIDA ESPERADA

```
Transferencia realizada
Transferencia realizada
Importe no válido
ES01 Ana: 249,50 €
ES02 Luis: 170,25 €
ES03 Marta: 200,25 €
```

31. Un informe sin el problema N+1

Difícil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

El programa muestra, para un año dado, cuántos pedidos ha hecho cada cliente y cuánto ha gastado, ordenado de mayor a menor gasto (a igual gasto, por nombre). Deben salir todos los clientes, también los que no han comprado ese año («0 pedidos, 0,00 €»). El código actual funciona, pero hace una consulta por cliente: con 10.000 clientes serían 10.001 consultas. Reescríbelo con una sola consulta (LEFT JOIN y GROUP BY). El programa cuenta las consultas ejecutadas y lo muestra al final.

Código de partida (php)

```
<?php
final class SentenciaContada extends PDOStatement
{
    public static int $ejecutadas = 0;
    protected function __construct() {}
    public function execute(?array $params = null): bool
    {
        self::$ejecutadas++;
        return parent::execute($params);
    }
}

$pdo = new PDO('sqlite::memory:', null, null, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC]);
$pdo->exec("CREATE TABLE clientes (id INTEGER PRIMARY KEY, nombre TEXT)");
$pdo->exec("CREATE TABLE pedidos (id INTEGER PRIMARY KEY, cliente_id INTEGER, fecha TEXT, total REAL)");
$pdo->exec("INSERT INTO clientes (nombre) VALUES ('Ana'), ('Luis'), ('Marta'), ('Pablo'), ('Eva')");
$pdo->exec("INSERT INTO pedidos (cliente_id, fecha, total) VALUES
(1, '2025-03-10', 120), (1, '2025-11-02', 80.5), (1, '2026-01-15', 45),
(2, '2025-06-01', 200.5), (3, '2026-02-20', 99.99), (3, '2026-05-05', 10),
(4, '2025-12-31', 0.01), (5, '2025-01-01', 200.5)");
$pdo->setAttribute(PDO::ATTR_STATEMENT_CLASS, [SentenciaContada::class]);

$anio = trim((string) fgets(STDIN));
$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';

// TODO: una sola consulta
$clientes = $pdo->prepare('SELECT id, nombre FROM clientes');
$clientes->execute();
$informe = [];
$porCliente = $pdo->prepare("SELECT COUNT(*) AS pedidos, COALESCE(SUM(total), 0) AS gasto FROM pedidos
WHERE cliente_id = ? AND strftime('%Y', fecha) = ?");
foreach ($clientes->fetchAll() as $c) {
    $porCliente->execute([$c['id'], $anio]);
    $informe[] = ['nombre' => $c['nombre']] + $porCliente->fetch();
}
usort($informe, fn($a, $b) => [$b['gasto'], $a['nombre']] <=> [$a['gasto'], $b['nombre']]);

foreach ($informe as $fila) {
    echo "{$fila['nombre']}: {$fila['pedidos']} pedidos, {$euros($fila['gasto'])}\n";
}
echo 'Consultas: ', SentenciaContada::$ejecutadas, "\n";
```

Ejemplo: Año 2025

ENTRADA

2025

SALIDA ESPERADA

Ana: 2 pedidos, 200,50 €
Eva: 1 pedidos, 200,50 €
Luis: 1 pedidos, 200,50 €
Pablo: 1 pedidos, 0,01 €
Marta: 0 pedidos, 0,00 €
Consultas: 1

Ejemplo: Año 2026

ENTRADA

2026

SALIDA ESPERADA

Marta: 2 pedidos, 109,99 €
Ana: 1 pedidos, 45,00 €
Eva: 0 pedidos, 0,00 €
Luis: 0 pedidos, 0,00 €
Pablo: 0 pedidos, 0,00 €
Consultas: 1

32. Importar un CSV: todo o nada

Muy difícil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

La entrada es un CSV con cabecera «sku,nombre,precio,stock». Procesa las filas en orden dentro de una transacción: si el SKU ya existe en la tabla, actualiza nombre, precio y stock; si no, lo inserta. Valida cada fila antes de guardarla: 4 campos («número de campos incorrecto»), SKU con el formato AAA-000 («SKU no válido»), nombre no vacío («nombre vacío»), precio numérico ≥ 0 («precio no válido»), stock entero ≥ 0 («stock no válido») y que el SKU no se repita en el fichero («SKU repetido»). En cuanto una fila falle, escribe «Línea N: motivo» (la cabecera es la línea 1), deshaz la transacción y escribe «Importación cancelada». Si todo va bien, confirma y escribe «Nuevos: X, actualizados: Y». En ambos casos termina listando la tabla ordenada por SKU («TEC-001 Teclado 24.99 5»).

Código de partida (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC]);
$pdo->exec("CREATE TABLE productos (sku TEXT PRIMARY KEY, nombre TEXT NOT NULL, precio REAL NOT NULL,
stock INTEGER NOT NULL)");
$pdo->exec("INSERT INTO productos VALUES ('TEC-001', 'Teclado', 24.99, 5), ('RAT-001', 'Ratón', 12.5,
10)");

$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
array_shift($lineas); // cabecera

// TODO: validar e importar dentro de una transacción

foreach ($pdo->query('SELECT * FROM productos ORDER BY sku') as $p) {
    echo "{$p['sku']} {$p['nombre']} {$p['precio']} {$p['stock']}\n";
}
```

Ejemplo: Importación correcta

ENTRADA

```
sku,nombre,precio,stock
MON-027,"Monitor 27""",189,3
TEC-001,Teclado mecánico,59.9,8
CAB-010,"Cable, 2 m",4.5,100
```

SALIDA ESPERADA

```
Nuevos: 2, actualizados: 1
CAB-010 Cable, 2 m 4.5 100
MON-027 Monitor 27" 189 3
RAT-001 Ratón 12.5 10
TEC-001 Teclado mecánico 59.9 8
```

Ejemplo: Un error deshace todo

ENTRADA

```
sku,nombre,precio,stock
MON-027,Monitor,189,3
TEC-001,Teclado nuevo,30,2
ALF-01,Alfombrilla,6.9,1
```

SALIDA ESPERADA

```
Línea 4: SKU no válido
Importación cancelada
RAT-001 Ratón 12.5 10
TEC-001 Teclado 24.99 5
```

Servicios Web y APIs REST

33. Resumir la respuesta de una API

Medio · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

La entrada es la respuesta JSON de una API de repositorios: un array de objetos con name, language (puede ser null) y stars. Muestra los tres con más estrellas («1. nombre (★ 1.234)», con punto de miles), después «Total de estrellas: N» y «Lenguajes: ...» con los lenguajes distintos ordenados alfabéticamente y separados por comas, sin contar null. Si el JSON no es válido, muestra «JSON no válido».

Código de partida (php)

```
<?php
$texto = stream_get_contents(STDIN);

// TODO
```

Ejemplo: Repositorios

ENTRADA

```
[{"name": "apuntes", "language": "TypeScript",
"stars": 320}, {"name": "tienda-php", "language":
"PHP", "stars": 1450}, {"name": "dotfiles",
"language": null, "stars": 12}, {"name": "api-
rest", "language": "PHP", "stars": 870}, {"name":
"juego", "language": "JavaScript", "stars": 95}]
```

SALIDA ESPERADA

```
1. tienda-php (★ 1.450)
2. api-rest (★ 870)
3. apuntes (★ 320)
Total de estrellas: 2.747
Lenguajes: JavaScript, PHP, TypeScript
```

Ejemplo: JSON roto

ENTRADA

```
[{"name": "x",
```

SALIDA ESPERADA

```
JSON no válido
```

34. Caché HTTP con ETag

Medio · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Para no enviar una y otra vez los mismos datos, una API puede responder con una ETag (una huella del contenido). El cliente la guarda y, en la siguiente petición, la envía en la cabecera If-None-Match; si el recurso no ha cambiado, el servidor responde 304 sin cuerpo. Cada línea de la entrada es «id etag-recibida» (o «id -» si el cliente no envía ninguna). Para cada una: si el producto no existe, «404»; si no, calcula la ETag como «...» con el md5 del JSON del producto y escribe «304» si coincide con la recibida, o «200 ETag: «...» {json}» si no.

Código de partida (php)

```
<?php
$productos = [
    7 => ['id' => 7, 'nombre' => 'Teclado', 'precio' => 24.99],
    8 => ['id' => 8, 'nombre' => 'Ratón', 'precio' => 12.5],
];

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$id, $recibida] = explode(' ', $linea, 2);
    // TODO: 404, 304 o 200 con ETag
    echo '200 ', json_encode($productos[(int) $id] ?? null, JSON_UNESCAPED_UNICODE), "\n";
}
```

Ejemplo: Primera petición y repetición

ENTRADA

```
7 -
7 "20d295fd089a83559e889d2e4b093c30"
```

SALIDA ESPERADA

```
200 ETag: "20d295fd089a83559e889d2e4b093c30"
{"id":7,"nombre":"Teclado","precio":24.99}
304
```

Ejemplo: ETag antigua y producto inexistente

ENTRADA

```
8 "20d295fd089a83559e889d2e4b093c30"
9 -
8 "8ce973fc634b7edd437fd1757591f55f"
```

SALIDA ESPERADA

```
200 ETag: "8ce973fc634b7edd437fd1757591f55f"
{"id":8,"nombre":"Ratón","precio":12.5}
404
304
```

35. Una API REST de libros

Difícil · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Cada línea de la entrada es una petición «MÉTODO RUTA [cuerpo JSON]». Escribe para cada una el código de estado y, si hay cuerpo, un espacio y la respuesta en JSON (json_encode con JSON_UNESCAPED_UNICODE). GET /api/libros → 200 con la lista; GET /api/libros/{id} → 200 con el libro o 404 {"error":"Libro no encontrado"}; POST /api/libros con titulo y autor → 201 con el libro creado (id = mayor id + 1) o 422 {"error":"Faltan campos"}; DELETE /api/libros/{id} → 204 sin cuerpo o 404. Cualquier otro método sobre esas rutas → 405 {"error":"Método no permitido"} y cualquier otra ruta → 404 {"error":"Ruta no encontrada"}.

Código de partida (php)

```
<?php
$libros = [
    1 => ['id' => 1, 'titulo' => 'El Quijote', 'autor' => 'Cervantes'],
    2 => ['id' => 2, 'titulo' => 'La Regenta', 'autor' => 'Clarín'],
];

function responder(int $codigo, mixed $datos = null): void
{
    echo $codigo, $datos === null ? ' ' : ' ' . json_encode($datos, JSON_UNESCAPED_UNICODE), "\n";
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$metodo, $ruta, $cuerpo] = array_pad(explode(' ', $linea, 3), 3, '');
    // TODO
}
```

Ejemplo: Consultar

ENTRADA

```
GET /api/libros
GET /api/libros/2
GET /api/libros/9
```

SALIDA ESPERADA

```
200 [{"id":1,"titulo":"El Quijote","autor":"Cervantes"},
{"id":2,"titulo":"La Regenta","autor":"Clarín"}]
200 {"id":2,"titulo":"La Regenta","autor":"Clarín"}
404 {"error":"Libro no encontrado"}
```

Ejemplo: Crear y borrar

ENTRADA

```
POST /api/libros {"titulo": "Niebla", "autor": "Unamuno"}
POST /api/libros {"titulo": "Sin autor"}
DELETE /api/libros/1
GET /api/libros
```

SALIDA ESPERADA

```
201 {"id":3,"titulo":"Niebla","autor":"Unamuno"}
422 {"error":"Faltan campos"}
204
200 [{"id":2,"titulo":"La Regenta","autor":"Clarín"},
{"id":3,"titulo":"Niebla","autor":"Unamuno"}]
```

Ejemplo: Errores de método y ruta

ENTRADA

```
PUT /api/libros/2
PATCH /api/libros
GET /api/autores
```

SALIDA ESPERADA

```
405 {"error":"Método no permitido"}
405 {"error":"Método no permitido"}
404 {"error":"Ruta no encontrada"}
```

36. Tokens JWT firmados a mano

Difícil · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Un JWT tiene tres partes en base64url separadas por puntos: cabecera, datos (payload) y firma HMAC-SHA256 de «cabecera.payload» con una clave secreta. Completa crear(array \$datos): string y verificar(string \$token, int \$ahora): ?array. verificar devuelve los datos si el token tiene tres partes, la firma coincide (hash_equals) y no ha caducado (exp mayor que \$ahora); si no, null. La entrada tiene órdenes «crear usuario rol exp» (escribe el token) y «verificar token ahora» (escribe «usuario (rol)» o «Token no válido»).

Código de partida (php)

```
<?php
const CLAVE = 'clave-super-secreta-de-32-bytes!';

function b64url(string $s): string
{
    return rtrim(strtr(base64_encode($s), '+/', '-_'), '=');
}

function crear(array $datos): string
{
    // TODO
    return '';
}

function verificar(string $token, int $ahora): ?array
{
    // TODO
    return null;
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    $p = explode(' ', $linea);
    if ($p[0] === 'crear') {
        echo crear(['sub' => $p[1], 'rol' => $p[2], 'exp' => (int) $p[3]]), "\n";
    } else {
        $d = verificar($p[1], (int) $p[2]);
        echo $d ? "{$d['sub']} ({$d['rol']})\n" : "Token no válido\n";
    }
}
```

Ejemplo: Crear y verificar

ENTRADA

```
crear ana admin 2000
verificar
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhbmEiLCJyb2wiOiJhZG1pbiIsImV4cCI6MjAwMH0.IaJuoy2wd_oP6WTgrfU1Ny2SyxlpU5xvgqW0_DHmGoM 1000
```

SALIDA ESPERADA

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhbmEiLCJyb2wiOiJhZG1pbiIsImV4cCI6MjAwMH0.IaJuoy2wd_oP6WTgrfU1Ny2SyxlpU5xvgqW0_DHmGoM
ana (admin)
```

Ejemplo: Caducado y manipulado

ENTRADA

```
verificar
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJs
dWlzIiwicm9sIjoiY2xpZW50ZSI6ImV4cCI6MTUwMH0.Wm4XR
0ADR0f3fufjlvLLdVgmrqmGPjsM7kRNYPZ3Q4 1500
verificar
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJs
dWlzIiwicm9sIjoiY2xpZW50ZSI6ImV4cCI6MTUwMH0.Wm4XR
0ADR0f3fufjlvLLdVgmrqmGPjsM7kRNYPZ3Q4 1499
verificar
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJh
bmEiLCJyb2wiOiJhZG1pbiIsImV4cCI6OTk5MD0.IaJuoy2wd
_oP6WTgrfU1Ny2SyxlpU5xvqW0_DHmGoM 1000
verificar basura 1000
```

SALIDA ESPERADA

```
Token no válido
luis (cliente)
Token no válido
Token no válido
```

37. Limitar las peticiones: 429 Too Many Requests

Muy difícil · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Toda API pública limita cuántas peticiones acepta de cada cliente. Implementa una ventana deslizante: un cliente puede hacer como máximo 3 peticiones en cualquier intervalo de 10 segundos. Cada línea de la entrada es «segundo cliente» (los segundos no bajan). Si la petición se admite, escribe «segundo cliente: 200 (quedan N)»; si no, «segundo cliente: 429 Retry-After: S», donde S es cuántos segundos faltan para que la petición más antigua de su ventana salga de ella. Las peticiones rechazadas no cuentan. Al final, escribe cuántas peticiones se admitieron por cliente, en orden alfabético.

Código de partida (php)

```
<?php
const LIMITE = 3;
const VENTANA = 10;
$historial = [];
$admitidas = [];

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$ahora, $cliente] = explode(' ', $linea);
    $ahora = (int) $ahora;
    // TODO
    echo "$ahora $cliente: 200 (quedan 2)\n";
}

ksort($admitidas);
foreach ($admitidas as $cliente => $n) {
    echo "$cliente: $n admitidas\n";
}
```

Ejemplo: Un cliente insistente

ENTRADA

```
0 ana
1 ana
2 ana
3 ana
9 ana
10 ana
11 ana
```

SALIDA ESPERADA

```
0 ana: 200 (quedan 2)
1 ana: 200 (quedan 1)
2 ana: 200 (quedan 0)
3 ana: 429 Retry-After: 7
9 ana: 429 Retry-After: 1
10 ana: 200 (quedan 0)
11 ana: 200 (quedan 0)
ana: 5 admitidas
```

Ejemplo: Clientes independientes

ENTRADA

```
0 luis
0 eva
1 luis
2 luis
2 eva
5 luis
12 luis
```

SALIDA ESPERADA

```
0 luis: 200 (quedan 2)
0 eva: 200 (quedan 2)
1 luis: 200 (quedan 1)
2 luis: 200 (quedan 0)
2 eva: 200 (quedan 1)
5 luis: 429 Retry-After: 5
12 luis: 200 (quedan 2)
eva: 2 admitidas
luis: 4 admitidas
```

Aplicaciones Dinámicas e Híbridas

38. El endpoint de un autocompletado

Medio · Aplicaciones Dinámicas e Híbridas · apuntesdam.com/subject/dwes/topic/aplicaciones-hibridas

Un campo de búsqueda de municipios pide sugerencias al servidor mientras el usuario escribe. Cada línea de la entrada es la query string que llega (q=...). Responde con un array JSON (una línea por petición, sin escapar las tildes) de como mucho 5 objetos {"nombre", "provincia"}: primero los municipios que empiezan por el texto y después los que lo contienen en otra posición, cada grupo en orden alfabético. La búsqueda ignora mayúsculas, tildes y espacios a los lados. Con menos de 2 caracteres, responde [].

Código de partida (php)

```
<?php
$municipios = [
    ['Arnedo', 'La Rioja'], ['Logroño', 'La Rioja'], ['Calahorra', 'La Rioja'], ['Haro', 'La Rioja'],
    ['Alfaro', 'La Rioja'], ['Ávila', 'Ávila'], ['Almería', 'Almería'], ['Alicante', 'Alicante'],
    ['Albacete', 'Albacete'], ['Alcalá de Henares', 'Madrid'], ['Madrid', 'Madrid'], ['Málaga',
    'Málaga'],
    ['Salamanca', 'Salamanca'], ['Cádiz', 'Cádiz'], ['Jaén', 'Jaén'], ['León', 'León'], ['Lleida',
    'Lleida'],
    ['Logrosán', 'Cáceres'], ['Santander', 'Cantabria'], ['Zaragoza', 'Zaragoza'], ['Alcobendas',
    'Madrid'],
    ['Aranjuez', 'Madrid'],
];

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    parse_str($linea, $q);
    $texto = trim($q['q'] ?? '');
    // TODO
    $sugerencias = [];
    echo json_encode($sugerencias, JSON_UNESCAPED_UNICODE), "\n";
}
```

Ejemplo: Prefijos y límite de 5

ENTRADA

```
q=a1
```

SALIDA ESPERADA

```
[{"nombre":"Albacete","provincia":"Albacete"},
{"nombre":"Alcalá de
Henares","provincia":"Madrid"},
{"nombre":"Alcobendas","provincia":"Madrid"},
{"nombre":"Alfaro","provincia":"La Rioja"},
{"nombre":"Alicante","provincia":"Alicante"}]
```

Ejemplo: Mayúsculas, tildes y mínimo

ENTRADA

```
q=LOGRO  
q=%C3%A1vila  
q=a
```

SALIDA ESPERADA

```
[{"nombre":"Logroño","provincia":"La Rioja"},  
{"nombre":"Logrosán","provincia":"Cáceres"}]  
[{"nombre":"Ávila","provincia":"Ávila"}]  
[]
```

39. Verificar la firma de un webhook

Difícil · Aplicaciones Dinámicas e Híbridas · apuntesdam.com/subject/dwes/topic/aplicaciones-hibridas

Cuando un pago se completa, la pasarela avisa a tu servidor con una petición (un webhook). Cualquiera podría enviar una petición falsa a esa URL, así que la pasarela la firma: la cabecera Firma es «t=segundos,v1=hmac», donde hmac es hash_hmac('sha256', "\$t.\$cuerpo", secreto). Puede haber varios v1 (mientras se cambia de secreto) y basta con que coincida uno. Cada línea de la entrada es «ahora|cabecera|cuerpo». Responde «Aceptado: tipo» (el campo tipo del JSON del cuerpo), «Rechazado: cabecera» si falta t o no hay ningún v1, «Rechazado: caducado» si la firma tiene más de 300 segundos (evita que alguien reenvíe un aviso antiguo) o «Rechazado: firma».

Código de partida (php)

```
<?php  
const SECRETO = 'whsec_9f86d081884c7d659a2feaa0c55ad015';  
  
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {  
    [$ahora, $cabecera, $cuerpo] = explode('|', $linea, 3);  
    // TODO  
    echo 'Aceptado: ', json_decode($cuerpo, true)['tipo'] ?? '?', "\n";  
}
```

Ejemplo: Firma correcta y secretos en rotación

ENTRADA

```
1790000000|t=1790000000,v1=a66b90c2ce77ad964f837f  
f815c1bfb648250c7ae018143fc7deaf5bd70adc04|  
{"tipo":"pago.completado","pedido":42}  
1790000000|t=1790000000,v1=6d8e2dca4474029af715b9  
2b92446f8ccfb7482dbef34e7e74d6b66bdaab76e9,v1=3b8  
cfd66d6a6b47bbca67eecba09df7144505aaa108217791e82  
54bd1cff55cd|{"tipo":"pago.fallido","pedido":43}
```

SALIDA ESPERADA

```
Aceptado: pago.completado  
Aceptado: pago.fallido
```

Ejemplo: Ataques

ENTRADA

```
1790000000|t=1790000000,v1=a66b90c2ce77ad964f837f  
f815c1bfb648250c7ae018143fc7deaf5bd70adc04|  
{"tipo":"pago.completado","pedido":99}  
1790000301|t=1790000000,v1=a66b90c2ce77ad964f837f  
f815c1bfb648250c7ae018143fc7deaf5bd70adc04|  
{"tipo":"pago.completado","pedido":42}  
1790000000|v1=a66b90c2ce77ad964f837ff815c1bfb6482  
50c7ae018143fc7deaf5bd70adc04|  
{"tipo":"pago.completado","pedido":42}  
1790000000|t=1790000000|  
{"tipo":"pago.completado","pedido":42}
```

SALIDA ESPERADA

```
Rechazado: firma  
Rechazado: caducado  
Rechazado: cabecera  
Rechazado: cabecera
```

40. Un cortacircuitos para un servicio externo

Muy difícil · Aplicaciones Dinámicas e Híbridas · apuntesdam.com/subject/dwes/topic/aplicaciones-hibridas

Si la API de envíos se cae, cada petición a tu tienda esperaría el tiempo máximo antes de fallar, y la tienda entera se volvería lenta. El patrón cortacircuitos (circuit breaker) lo evita. Empieza cerrado (las llamadas pasan). Tras 3 fallos seguidos se abre: durante 30 segundos las peticiones se rechazan al momento sin llamar. Pasados 30 segundos, la siguiente petición es una prueba: si sale bien se cierra (y se reinicia la cuenta de fallos); si falla, vuelve a abrirse otros 30 segundos. Un éxito con el circuito cerrado también reinicia la cuenta. Cada línea es «segundo resultado», donde resultado es lo que respondería el servicio si se le llamara (ok o fallo). Escribe, según el caso: «t: ok (cerrado)», «t: fallo (cerrado, N fallos)», «t: fallo → abierto», «t: rechazada (abierto)», «t: prueba ok → cerrado» o «t: prueba fallo → abierto». Al final, «Llamadas reales: N».

Código de partida (php)

```
<?php
const UMBRAL = 3;
const ESPERA = 30;
$estado = 'cerrado';
$fallos = 0;
$abiertoDesde = 0;
$llamadas = 0;

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$t, $resultado] = explode(' ', $linea);
    $t = (int) $t;
    // TODO
    $llamadas++;
    echo "$t: $resultado (cerrado)\n";
}
echo "Llamadas reales: $llamadas\n";
```

Ejemplo: Se abre y la prueba falla

ENTRADA

```
0 ok
1 fallo
2 fallo
3 fallo
10 ok
32 ok
33 fallo
```

SALIDA ESPERADA

```
0: ok (cerrado)
1: fallo (cerrado, 1 fallos)
2: fallo (cerrado, 2 fallos)
3: fallo → abierto
10: rechazada (abierto)
32: rechazada (abierto)
33: prueba fallo → abierto
Llamadas reales: 5
```

Ejemplo: Un éxito reinicia y la prueba sale bien

ENTRADA

```
0 fallo
1 ok
2 fallo
3 fallo
4 fallo
40 ok
41 fallo
```

SALIDA ESPERADA

```
0: fallo (cerrado, 1 fallos)
1: ok (cerrado)
2: fallo (cerrado, 1 fallos)
3: fallo (cerrado, 2 fallos)
4: fallo → abierto
40: prueba ok → cerrado
41: fallo (cerrado, 1 fallos)
Llamadas reales: 7
```

Ejercicios de Desarrollo Web en Entorno Servidor

41. Informe a partir de un CSV

Medio · Ejercicios de Desarrollo Web en Entorno Servidor · apuntesdam.com/subject/dwes/topic/ejercicios-dwes

La entrada es un CSV con cabecera «fecha,comercial,importe» (los campos pueden ir entre comillas). Suma el importe por comercial y muestra una línea por comercial ordenada de mayor a menor importe («Ana: 3.200,50 €»), después «Total: ...» y, si alguna fila no tenía 3 campos o un importe numérico, «Filas descartadas: N».

Código de partida (php)

```
<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);

// TODO
```

Ejemplo: Ventas

ENTRADA

```
fecha,comercial,importe
2025-01-10,Ana,1200.50
2025-01-11,Luis,800
2025-01-15,Ana,2000
2025-02-01,"Marta",950
```

SALIDA ESPERADA

```
Ana: 3.200,50 €
Marta: 950,00 €
Luis: 800,00 €
Total: 4.950,50 €
```

Ejemplo: Filas erróneas

ENTRADA

```
fecha,comercial,importe
2025-03-01,Luis,100
2025-03-02,Ana
2025-03-03,Marta,mucho
2025-03-04,"Pérez, Juan",4500
```

SALIDA ESPERADA

```
Pérez, Juan: 4.500,00 €
Luis: 100,00 €
Total: 4.600,00 €
Filas descartadas: 2
```

42. API de tareas con base de datos

Difícil · Ejercicios de Desarrollo Web en Entorno Servidor · apuntesdam.com/subject/dwes/topic/ejercicios-dwes

Une una API REST con PDO. La tabla tareas (id, titulo, hecha) empieza vacía. Cada línea es «MÉTODO RUTA [JSON]» y la respuesta, «código JSON». POST /tareas {"titulo": ...} → 201 con la tarea ({ "id":1,"titulo":..., "hecha":false}) o 422 {"error":"El título es obligatorio"}; GET /tareas → 200 con todas por id; GET /tareas?hecha=1 o ?hecha=0 → filtradas; PATCH /tareas/{id} {"hecha": true} → 200 con la tarea actualizada; DELETE /tareas/{id} → 204 sin cuerpo. Un id inexistente → 404 {"error":"Tarea no encontrada"}. Todas las consultas, preparadas; hecha se guarda como 0/1 y se devuelve como booleano.

Código de partida (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec('CREATE TABLE tareas (id INTEGER PRIMARY KEY, titulo TEXT NOT NULL, hecha INTEGER NOT NULL
    DEFAULT 0)');

function responder(int $codigo, mixed $datos = null): void
{
    echo $codigo, $datos === null ? ' ' : ' ' . json_encode($datos, JSON_UNESCAPED_UNICODE), "\n";
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$metodo, $ruta, $cuerpo] = array_pad(explode(' ', $linea, 3), 3, '');
    // TODO
}
```

Ejemplo: Crear y listar

ENTRADA

```
POST /tareas {"titulo": "Estudiar PDO"}
POST /tareas {"titulo": "Hacer la API"}
POST /tareas {"titulo": " "}
GET /tareas
```

SALIDA ESPERADA

```
201 {"id":1,"titulo":"Estudiar
PDO","hecha":false}
201 {"id":2,"titulo":"Hacer la
API","hecha":false}
422 {"error":"El título es obligatorio"}
200 [{"id":1,"titulo":"Estudiar
PDO","hecha":false},{id":2,"titulo":"Hacer la
API","hecha":false}]
```

Ejemplo: Actualizar, filtrar y borrar

ENTRADA

```
POST /tareas {"titulo": "A"}
POST /tareas {"titulo": "B"}
PATCH /tareas/2 {"hecha": true}
GET /tareas?hecha=1
GET /tareas?hecha=0
DELETE /tareas/1
DELETE /tareas/1
GET /tareas
```

SALIDA ESPERADA

```
201 {"id":1,"titulo":"A","hecha":false}
201 {"id":2,"titulo":"B","hecha":false}
200 {"id":2,"titulo":"B","hecha":true}
200 [{"id":2,"titulo":"B","hecha":true}]
200 [{"id":1,"titulo":"A","hecha":false}]
204
404 {"error":"Tarea no encontrada"}
200 [{"id":2,"titulo":"B","hecha":true}]
```

43. Carrito con cupones

Difícil · Ejercicios de Desarrollo Web en Entorno Servidor · apuntesdam.com/subject/dwes/topic/ejercicios-dwes

La entrada tiene líneas «producto;precio;cantidad» y, opcionalmente, líneas «CUPON código». El envío cuesta 4,95 € si el subtotal es menor de 50 €. Cupones: DESC10 (10 % del subtotal), MENOS5 (5 € si el subtotal es de al menos 30 €) y ENVIOGRATIS (sin gastos de envío). Solo se aplica un cupón: el que más ahorre de los introducidos (los códigos desconocidos o que no cumplan su condición se ignoran). Muestra «Subtotal: ...», «Descuento: ...» (con el código entre paréntesis si hay cupón, «Descuento (DESC10): 5,00 €»; si no, «Descuento: 0,00 €»), «Envío: ...» y «Total: ...».

Código de partida (php)

```
<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);

// TODO
```

Ejemplo: Sin cupón

ENTRADA

Libro;19.90;1
Bolígrafo;1.20;5

SALIDA ESPERADA

Subtotal: 25,90 €
Descuento: 0,00 €
Envío: 4,95 €
Total: 30,85 €

Ejemplo: Elige el mejor cupón

ENTRADA

Auriculares;39.90;1
CUPON MENOS5
CUPON ENVIIOGRATIS
CUPON DESC10

SALIDA ESPERADA

Subtotal: 39,90 €
Descuento (MENOS5): 5,00 €
Envío: 4,95 €
Total: 39,85 €

Ejemplo: Cupón que no cumple y otro que sí

ENTRADA

Funda;12;2
CUPON MENOS5
CUPON FALSO
CUPON DESC10

SALIDA ESPERADA

Subtotal: 24,00 €
Descuento (DESC10): 2,40 €
Envío: 4,95 €
Total: 26,55 €

Ejercicios largos

44. Factura de un carrito con IVA y promociones

Difícil · PHP · 70 minutos · apuntesdam.com/ejercicios/php/factura-con-promociones

Una tienda online pequeña guarda el carrito de cada cliente como texto, con una línea por producto: nombre, categoría, precio sin IVA y unidades. Antes de cobrar, necesita un programa que convierta ese carrito en una factura clara, con las promociones aplicadas y el IVA desglosado.

Para simplificar, la tienda agrupa sus productos en tres categorías y cada una tiene un único tipo de IVA (en la realidad hay alimentos al 4 % y otros al 10 %, pero aquí no hace falta afinar tanto). Además tiene dos promociones: un 3x2 en alimentación y un 5 % de descuento en los productos de la categoría general cuando la compra es grande.

El carrito lo rellena un formulario antiguo y a veces llegan líneas incorrectas. El programa no debe fallar por ellas: avisa de cada una y factura el resto.

Como en cualquier programa que maneje dinero, los importes se calculan en céntimos enteros. Sumar euros con decimales en coma flotante acumula pequeños errores ($0.1 + 0.2$ no es exactamente 0.3), y en una factura un céntimo de diferencia es un error.

Requisitos

- Cada línea del carrito es `nombre;categoria;precio;unidades`. El precio no lleva IVA y usa punto decimal. Se ignoran las líneas vacías y los espacios al principio y al final de cada campo.
- Una línea no es válida si no tiene exactamente 4 campos, si la categoría no es `libros`, `alimentacion` ni `general`, si el precio no es un número mayor o igual que 0 o si las unidades no son un número entero mayor que 0. Por cada una escribe `Aviso: línea N ignorada`, con N el número de línea de la entrada (las vacías también cuentan). Los avisos salen antes que la factura.
- Si no queda ningún producto válido, escribe `Carrito vacío` y termina.

- La tabla empieza con la cabecera y tiene una línea por producto, en el orden del carrito: el nombre en 22 caracteres alineado a la izquierda (contando letras, no bytes: «Lámpara» ocupa 7), y alineados a la derecha las unidades en 5 caracteres, el precio unitario en 10 y el importe (unidades × precio) en 11. La cabecera usa las mismas anchuras: `Producto`, `Uds`, `Precio` e `Importe`.
- 3x2 en alimentación: por cada 3 unidades de un mismo producto de alimentación, una es gratis (unidades gratis = unidades ÷ 3, división entera). Por cada producto con unidades gratis, escribe `3x2 en nombre: -X €`.
- Descuento por importe: si la base de todo el carrito, ya con el 3x2 aplicado, supera los 100 € (estrictamente), los productos de la categoría general tienen un 5 % de descuento sobre su base, redondeado al céntimo. Si ese descuento es mayor que 0, escribe `Descuento del 5 % en general: -X €`.
- Si hay alguna promoción, se escriben bajo la línea `Promociones:` con dos espacios delante: primero los 3x2 en el orden del carrito y después el descuento.
- El resumen tiene la `Base imponible:` (la suma de las bases de las tres categorías tras las promociones), una línea `IVA T %:` por cada categoría con base mayor que 0, en el orden 4 %, 10 % y 21 %, y el `TOTAL:` (base más cuotas). La cuota de cada tipo es su base × T ÷ 100, redondeada al céntimo. Cada línea lleva la etiqueta en 16 caracteres a la izquierda, el importe en 12 a la derecha, un espacio y `€`.
- Todos los importes se escriben con dos decimales, coma decimal y punto de millar, por ejemplo `1.200,00`. Los redondeos son al céntimo más cercano y las mitades suben (como hace `round` en PHP).

Formato de la entrada

- Una línea por producto: `nombre;categoria;precio;unidades`. Puede haber líneas vacías o mal escritas.

Categorías de la tienda

Categoría	IVA	Promoción
libros	4 %	—
alimentacion	10 %	3x2 en cada producto
general	21 %	5 % de descuento si la base del carrito supera 100 €

Ejemplo: Un carrito con promociones

ENTRADA

Leche entera;alimentacion;0.95;6
 Pan de molde;alimentacion;1.80;2
 Auriculares;general;59.90;1
 Cargador USB-C;general;19.95;2
 El Quijote;libros;12.50;1

SALIDA ESPERADA

Producto	Uds	Precio	Importe
Leche entera	6	0,95	5,70
Pan de molde	2	1,80	3,60
Auriculares	1	59,90	59,90
Cargador USB-C	2	19,95	39,90
El Quijote	1	12,50	12,50

Promociones:
 3x2 en Leche entera: -1,90 €
 Descuento del 5 % en general: -4,99 €
 Base imponible: 114,71 €
 IVA 4 %: 0,50 €
 IVA 10 %: 0,74 €
 IVA 21 %: 19,91 €
 TOTAL: 135,86 €

Ejemplo: Con líneas incorrectas

ENTRADA

Yogur natural;alimentacion;0.45;7
 Cuaderno;general;2.40;3
 Revista;prensa;3.00;1
 Bolígrafos;general;abc;2
 Agua;alimentacion;0.30;0

 Diccionario;libros;24.00;1

SALIDA ESPERADA

Aviso: línea 3 ignorada
 Aviso: línea 4 ignorada
 Aviso: línea 5 ignorada

Producto	Uds	Precio	Importe
Yogur natural	7	0,45	3,15
Cuaderno	3	2,40	7,20
Diccionario	1	24,00	24,00

Promociones:
 3x2 en Yogur natural: -0,90 €
 Base imponible: 33,45 €
 IVA 4 %: 0,96 €
 IVA 10 %: 0,23 €
 IVA 21 %: 1,51 €
 TOTAL: 36,15 €

Código de partida (php)

```
<?php
// Tipos de IVA de cada categoría
const IVA = ["libros" => 4, "alimentacion" => 10, "general" => 21];

/** Céntimos → «1.234,50» (punto de millar y coma decimal). */
function dinero(int $centimos): string
{
    return number_format($centimos / 100, 2, ",", ".");
}

$lineas = explode("\n", stream_get_contents(STDIN));
$productos = [];
foreach ($lineas as $i => $linea) {
    // TODO: valida la línea (o escribe el aviso) y guarda el producto con su precio en céntimos
}

// TODO: tabla de productos, promociones, base imponible, IVA de cada tipo y total
```

45. Préstamos de una biblioteca con PDO y transacciones

Muy difícil · PHP · 100 minutos · apuntesdam.com/ejercicios/php/biblioteca-con-pdo-y-transacciones

La biblioteca de un instituto quiere dejar la hoja de cálculo y gestionar los préstamos con una aplicación web en PHP. Antes de hacer las pantallas, se escribe y se prueba la capa de datos: funciones que reciben órdenes y trabajan contra la base de datos con PDO.

Para que el ejercicio funcione sin instalar nada, la base de datos es SQLite en memoria: el código inicial la crea con sus tres tablas y unos datos de partida, que tienes abajo. Con MySQL el código sería el mismo cambiando solo la cadena de conexión (el DSN): esa es la ventaja de PDO.

Las reglas del préstamo son las habituales: 15 días de plazo, como mucho 3 libros a la vez, nada nuevo si se tiene algo vencido y 0,20 € de multa por día de retraso. Comprobar las reglas y registrar el préstamo debe ser una operación atómica: si dos bibliotecarios prestan a la vez el último ejemplar, solo uno puede conseguirlo. Eso es una transacción.

Requisitos

- Cada línea es una orden; antes de responder, escribe `>` y la línea tal como se ha leído (sin espacios en los extremos). Una orden desconocida o con argumentos que no tocan escribe `Orden no válida`.
- `BUSCAR texto`: libros cuyo título o autor contienen el texto (sin distinguir mayúsculas), por orden de título, como `código · título (autor) · disponibles D de E`, donde D son los ejemplares menos los prestados sin devolver. Si no hay ninguno, `Sin resultados para «texto»`.
- `PRESTAR código socio fecha`: comprueba que existen el libro (`No existe el libro X`) y el socio (`No existe el socio N`) y, dentro de una transacción y en este orden, que queda algún ejemplar (`No quedan ejemplares de «título»`), que el socio no tiene préstamos activos vencidos a esa fecha (`Nombre tiene préstamos vencidos`) y que no tiene ya 3 activos (`Nombre ya tiene 3 préstamos`). Si todo va bien, registra el préstamo y escribe `Préstamo N: «título» para Nombre (devolver antes del AAAA-MM-DD)`, con la fecha de salida más 15 días. Un préstamo vence cuando su fecha límite es anterior a la fecha de la operación.
- `DEVOLVER id fecha`: `No existe el préstamo N`, `El préstamo N ya se devolvió el AAAA-MM-DD`, o registra la devolución y escribe `Devuelto «título» a tiempo` o `Devuelto «título» con R días de retraso: multa X €` ($R = \text{días transcurridos} - 15$; «día» si es 1).
- `MOROSOS fecha`: `Préstamos vencidos el fecha:` y una línea por préstamo activo con retraso a esa fecha, por fecha de salida: `Nombre: «título» (R días de retraso)`; o `No hay préstamos vencidos el fecha`.
- `RESUMEN`: `Préstamos activos: N` y `Socio más lector: Nombre (N préstamos)` (contando también los devueltos; si empatan, el primero alfabéticamente).
- Todas las consultas que llevan datos de la entrada deben ser consultas preparadas.

Formato de la entrada

- Una orden por línea: `BUSCAR`, `PRESTAR`, `DEVOLVER`, `MOROSOS` o `RESUMEN`, con las fechas como `AAAA-MM-DD`.

libros

codigo	titulo	autor	ejemplares
L01	Cien años de soledad	Gabriel García Márquez	2
L02	1984	George Orwell	1
L03	Rebelión en la granja	George Orwell	2
L04	La sombra del viento	Carlos Ruiz Zafón	1
L05	El nombre del viento	Patrick Rothfuss	3
L06	El tercer policía	Flann O'Brien	1

socios

id	nombre
1	Ana Gil
2	Luis Mora
3	Eva Ruiz

prestamos

id	codigo	socio	salida	devolucion
1	L02	2	2026-09-01	NULL
2	L01	1	2026-09-20	2026-09-28
3	L05	1	2026-09-25	NULL

Ejemplo: Una mañana en la biblioteca

ENTRADA

```
BUSCAR viento
BUSCAR orwell
BUSCAR O'Brien
PRESTAR L02 3 2026-10-01
PRESTAR L04 2 2026-10-01
PRESTAR L04 3 2026-10-01
DEVOLVER 1 2026-10-01
PRESTAR L02 3 2026-10-02
MOROSOS 2026-10-20
RESUMEN
```

SALIDA ESPERADA

```
> BUSCAR viento
L05 · El nombre del viento (Patrick Rothfuss) ·
disponibles 2 de 3
L04 · La sombra del viento (Carlos Ruiz Zafón) ·
disponibles 1 de 1
> BUSCAR orwell
L02 · 1984 (George Orwell) · disponibles 0 de 1
L03 · Rebelión en la granja (George Orwell) ·
disponibles 2 de 2
> BUSCAR O'Brien
L06 · El tercer policía (Flann O'Brien) ·
disponibles 1 de 1
> PRESTAR L02 3 2026-10-01
No quedan ejemplares de «1984»
> PRESTAR L04 2 2026-10-01
Luis Mora tiene préstamos vencidos
> PRESTAR L04 3 2026-10-01
Préstamo 4: «La sombra del viento» para Eva Ruiz
(devolver antes del 2026-10-16)
> DEVOLVER 1 2026-10-01
Devuelto «1984» con 15 días de retraso: multa
3,00 €
> PRESTAR L02 3 2026-10-02
Préstamo 5: «1984» para Eva Ruiz (devolver antes
del 2026-10-17)
> MOROSOS 2026-10-20
Préstamos vencidos el 2026-10-20:
  Ana Gil: «El nombre del viento» (10 días de
retraso)
  Eva Ruiz: «La sombra del viento» (4 días de
retraso)
  Eva Ruiz: «1984» (3 días de retraso)
> RESUMEN
Préstamos activos: 3
Socio más lector: Ana Gil (2 préstamos)
```

Ejemplo: Órdenes y datos que no existen

ENTRADA

```
BUSCAR Quijote
PRESTAR L99 1 2026-10-01
PRESTAR L01 7 2026-10-01
DEVOLVER 2 2026-10-01
DEVOLVER 50 2026-10-01
PRESTAR L01
CATALOGO
```

SALIDA ESPERADA

```
> BUSCAR Quijote
Sin resultados para «Quijote»
> PRESTAR L99 1 2026-10-01
No existe el libro L99
> PRESTAR L01 7 2026-10-01
No existe el socio 7
> DEVOLVER 2 2026-10-01
El préstamo 2 ya se devolvió el 2026-09-28
> DEVOLVER 50 2026-10-01
No existe el préstamo 50
> PRESTAR L01
Orden no válida
> CATALOGO
Orden no válida
```

```

<?php
// Base de datos de la biblioteca (SQLite en memoria): tablas libros, socios y prestamos
$pdo = new PDO("sqlite::memory:");
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
$pdo->exec(<<<SQL
    CREATE TABLE libros (codigo TEXT PRIMARY KEY, titulo TEXT NOT NULL, autor TEXT NOT NULL, ejemplares
        INTEGER NOT NULL);
    CREATE TABLE socios (id INTEGER PRIMARY KEY, nombre TEXT NOT NULL);
    CREATE TABLE prestamos (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        codigo TEXT NOT NULL REFERENCES libros(codigo),
        socio INTEGER NOT NULL REFERENCES socios(id),
        salida TEXT NOT NULL,
        devolucion TEXT
    );
    INSERT INTO libros VALUES
        ('L01', 'Cien años de soledad', 'Gabriel García Márquez', 2),
        ('L02', '1984', 'George Orwell', 1),
        ('L03', 'Rebelión en la granja', 'George Orwell', 2),
        ('L04', 'La sombra del viento', 'Carlos Ruiz Zafón', 1),
        ('L05', 'El nombre del viento', 'Patrick Rothfuss', 3),
        ('L06', 'El tercer policía', 'Flann O'Brien', 1);
    INSERT INTO socios VALUES (1, 'Ana Gil'), (2, 'Luis Mora'), (3, 'Eva Ruiz');
    INSERT INTO prestamos (codigo, socio, salida, devolucion) VALUES
        ('L02', 2, '2026-09-01', NULL),
        ('L01', 1, '2026-09-20', '2026-09-28'),
        ('L05', 1, '2026-09-25', NULL);
SQL);

const PLAZO = 15;           // días de préstamo
const MULTA = 20;           // céntimos por día de retraso
const MAXIMO = 3;           // préstamos activos por socio

function euros(int $c): string { return number_format($c / 100, 2, ",", ".") . " €"; }

// TODO: buscar, prestar (en una transacción), devolver, morosos y resumen, siempre con consultas
// preparadas

foreach (explode("\n", stream_get_contents(STDIN)) as $linea) {
    $linea = trim($linea);
    if ($linea === "") continue;
    // TODO: escribe «> línea» y ejecuta la orden
}

```

46. API REST de tareas con un router propio

Difícil · PHP · 90 minutos · apuntesdam.com/ejercicios/php/api-rest-de-tareas

Una aplicación móvil de tareas necesita un backend. La API sigue el estilo REST: la URL identifica el recurso (`/api/tareas` es la colección, `/api/tareas/3` una tarea) y el método HTTP dice qué hacer con él. Las respuestas son JSON y el código de estado HTTP indica el resultado: 200 bien, 201 creado, 204 borrado sin contenido, 400 petición mal formada, 404 no existe, 405 método no permitido y 422 datos que no pasan la validación.

Los frameworks (Laravel, Symfony, Slim) traen un router que asocia cada ruta con su código. Aquí vas a escribir uno pequeño: cada ruta es un patrón como `/api/tareas/{id}` que se convierte en una expresión regular, y su acción es una closure.

Para probarlo sin servidor web, cada línea de la entrada es una petición HTTP simplificada (método, URL y cuerpo JSON), y el programa escribe la respuesta. Las tareas se guardan en memoria: una base de datos sería el siguiente paso.

Requisitos

- Cada línea es `MÉTODO URL [cuerpo JSON]` (el cuerpo es todo lo que va tras el segundo espacio). Si la línea no tiene al menos método y URL, escribe `Petición no válida`. La respuesta se escribe como `MÉTODO URL → estado cuerpo`, con el método en mayúsculas, la URL tal como venía y el cuerpo en JSON compacto sin escapar tildes ni barras; las respuestas 204 no llevan cuerpo. Una barra al final de la ruta no cuenta (`/api/tareas/` es `/api/tareas`).
- Una tarea es `{"id": n, "titulo": texto, "prioridad": alta|media|baja, "hecha": booleano}`. Los id empiezan en 1 y no se reutilizan.
- `GET /api/tareas` → 200 con el array de tareas por orden de id; admite los filtros `?hecha=true|false` y `?q=texto` (título que contiene el texto sin distinguir mayúsculas), combinables. `GET /api/tareas/{id}` → 200 con la tarea.
- `POST /api/tareas` crea una tarea → 201 con la tarea. El título es obligatorio y, como los demás campos, se valida: título de 3 a 60 caracteres (tras quitar espacios de los extremos, que no se guardan), prioridad alta, media o baja (por defecto media), hecha true o false (por defecto false), y ningún otro campo (`campo desconocido`). Los errores devuelven 422 con `{"errores": {campo: mensaje}}`: obligatorio, entre 3 y 60 caracteres, debe ser alta, media o baja, debe ser true o false.
- `PATCH /api/tareas/{id}` cambia solo los campos que vienen (con las mismas reglas, sin título obligatorio) → 200 con la tarea completa. `DELETE /api/tareas/{id}` → 204.
- Si el cuerpo de un POST o PATCH no es un objeto JSON → 400 `{"error": "JSON no válido"}`. Si la tarea no existe → 404 `{"error": "No existe la tarea N"}`. Si la ruta no existe → 404 `{"error": "Ruta no encontrada"}`; si existe pero no con ese método → 405 `{"error": "Método no permitido"}`.

Formato de la entrada

- Una petición por línea: `MÉTODO URL` y, en POST y PATCH, el cuerpo JSON.

Rutas de la API

Método	Ruta	Respuesta correcta
GET	/api/tareas?hecha=&q=	200 + lista
POST	/api/tareas	201 + tarea creada
GET	/api/tareas/{id}	200 + tarea
PATCH	/api/tareas/{id}	200 + tarea modificada
DELETE	/api/tareas/{id}	204 sin cuerpo

Ejemplo: Crear, consultar, modificar y borrar

ENTRADA

```
POST /api/tareas {"titulo": "Preparar el examen de DWES", "prioridad": "alta"}
POST /api/tareas {"titulo": " Comprar café "}
GET /api/tareas
PATCH /api/tareas/2 {"hecha": true}
GET /api/tareas?hecha=true
GET /api/tareas?q=EXAMEN
DELETE /api/tareas/1
GET /api/tareas/1
GET /api/tareas/
```

SALIDA ESPERADA

```
POST /api/tareas → 201 {"id":1,"titulo":"Preparar el examen de DWES", "prioridad":"alta", "hecha":false}
POST /api/tareas → 201 {"id":2,"titulo":"Comprar café", "prioridad":"media", "hecha":false}
GET /api/tareas → 200 [{"id":1,"titulo":"Preparar el examen de DWES", "prioridad":"alta", "hecha":false}, {"id":2,"titulo":"Comprar café", "prioridad":"media", "hecha":false}]
PATCH /api/tareas/2 → 200 {"id":2,"titulo":"Comprar café", "prioridad":"media", "hecha":true}
GET /api/tareas?hecha=true → 200 [{"id":2,"titulo":"Comprar café", "prioridad":"media", "hecha":true}]
GET /api/tareas?q=EXAMEN → 200 [{"id":1,"titulo":"Preparar el examen de DWES", "prioridad":"alta", "hecha":false}]
DELETE /api/tareas/1 → 204
GET /api/tareas/1 → 404 {"error":"No existe la tarea 1"}
GET /api/tareas/ → 200 [{"id":2,"titulo":"Comprar café", "prioridad":"media", "hecha":true}]
```

Ejemplo: Errores de la API

ENTRADA

```
POST /api/tareas {"titulo": "ab", "prioridad": "urgente", "color": "rojo"}
POST /api/tareas {titulo: mal}
POST /api/tareas [1, 2]
POST /api/tareas {"prioridad": "baja"}
PUT /api/tareas
GET /api/usuarios
PATCH /api/tareas/9 {"hecha": true}
POST
```

SALIDA ESPERADA

```
POST /api/tareas → 422 {"errores": {"titulo":"entre 3 y 60 caracteres", "prioridad":"debe ser alta, media o baja", "color":"campo desconocido"}}
POST /api/tareas → 400 {"error":"JSON no válido"}
POST /api/tareas → 400 {"error":"JSON no válido"}
POST /api/tareas → 422 {"errores": {"titulo":"obligatorio"}}
PUT /api/tareas → 405 {"error":"Método no permitido"}
GET /api/usuarios → 404 {"error":"Ruta no encontrada"}
PATCH /api/tareas/9 → 404 {"error":"No existe la tarea 9"}
Petición no válida
```

```

<?php
/** Una ruta: método, patrón con {parámetros} y la función que la atiende. */
final class Ruta {
    public function __construct(public string $metodo, public string $patron, public Closure $accion) {}

    /** Los parámetros de la ruta si la URL encaja con el patrón, o null. */
    public function encaja(string $ruta): ?array {
        // TODO: convierte el patrón en una expresión regular con grupos con nombre
        return null;
    }
}

final class Respuesta {
    public function __construct(public int $estado, public mixed $cuerpo = null) {}
}

final class Router {
    private array $rutas = [];

    public function add(string $metodo, string $patron, Closure $accion): void {
        $this->rutas[] = new Ruta($metodo, $patron, $accion);
    }

    public function despachar(string $metodo, string $ruta, array $consulta, ?string $cuerpo): Respuesta {
        // TODO: 404 si ninguna ruta encaja, 405 si encaja pero con otro método
        return new Respuesta(404, ["error" => "Ruta no encontrada"]);
    }
}

$tareas = [];
$siguiente = 1;

// TODO: registra las rutas GET, POST, PATCH y DELETE de /api/tareas y atiende cada línea de la entrada

```

47. Libro de visitas seguro: validación, spam y escape de HTML

Medio · PHP · 55 minutos · apuntesdam.com/ejercicios/php/libro-de-visitas-seguro

La web de una casa rural tiene un libro de visitas: los huéspedes dejan su nombre, su email, su web (opcional) y un mensaje, y los mensajes se publican en la página. Es el ejemplo perfecto de por qué no hay que fiarse de nada de lo que llega en un formulario.

Un visitante malintencionado puede escribir `<script>` en el mensaje o poner `javascript:alert(1)` como web: si se publica tal cual, el código se ejecuta en el navegador de todos los que vean la página (un ataque XSS). La defensa es doble: validar al recibir y escapar al mostrar con `htmlspecialchars`.

Además, los robots de spam rellenan todos los campos que encuentran. Un truco eficaz es el campo trampa (honeypot): un campo `telefono2` que se oculta con CSS, que ninguna persona ve ni rellena. Si llega con algo, el envío es de un robot.

Requisitos

- Cada línea no vacía es un envío del formulario codificado como lo envía el navegador (`nombre=Ana&email=ana%40correo.es&mensaje=Hola`), con los campos nombre, email, web, mensaje y telefono2. Los envíos se numeran desde 1. A todos los campos se les quitan los espacios de los extremos.
- Si `telefono2` no está vacío, escribe `Mensaje N descartado: spam` y no se valida nada más.
- Reglas, en este orden: nombre obligatorio y de 2 a 50 caracteres; email obligatorio y válido según `FILTER_VALIDATE_EMAIL`; web opcional, pero si se escribe tiene que ser una URL válida con esquema http o https (solo direcciones http o https); mensaje obligatorio y de 200 caracteres como mucho. Los mensajes de error son obligatorio, entre 2 y 50 caracteres, no es un email válido, solo direcciones http o https y como mucho 200 caracteres.

- Un email (sin distinguir mayúsculas) solo puede publicar 2 mensajes: si el envío es correcto pero ese email ya tiene 2 publicados, el error es `email: ya ha enviado 2 mensajes`.
- Por cada error, `Mensaje N rechazado: campo: motivo`. Al final, la lista HTML de los mensajes publicados en orden: `<ul class="visitas">`, una línea por mensaje `<li><strong>nombre</strong> (<a href="web" rel="nofollow ugc">web</a>): mensaje</li>` (sin el paréntesis si no hay web) y `</ul>`, con el nombre, la web y el mensaje escapados con `htmlspecialchars ( ENT_QUOTES | ENT_HTML5 )`. Después, `Publicados: P · Rechazados: R · Spam: S`.

Formato de la entrada

- Un envío del formulario por línea, como query string (`campo=valor&campo=valor`).

Ejemplo: Cinco envíos

ENTRADA

```
nombre=Ana+Gil&email=ana%40correo.es&web=https%3A%2F%2Fana.es&mensaje=Un+sitio+precioso%2C+volveremos%3C3&telefono2=
nombre=Hacker&email=h%40x.com&web=javascript%3Aalert(1)&mensaje=%3Cscript%3Ealert(1)%3C%2Fscript%3E&telefono2=
nombre=Bot&email=bot%40spam.com&mensaje=Compra+ya&telefono2=600123123
nombre=Seán+O'27Brien&email=SEAN%40mail.ie&mensaje=%3Cb%3EGreat%3C%2Fb%3E+%26+quiet&telefono2=
nombre=A&email=no-es-email&mensaje=
```

SALIDA ESPERADA

```
Mensaje 2 rechazado: web: solo direcciones http o https
Mensaje 3 descartado: spam
Mensaje 5 rechazado: nombre: entre 2 y 50 caracteres
Mensaje 5 rechazado: email: no es un email válido
Mensaje 5 rechazado: mensaje: obligatorio
<ul class="visitas">
  <li><strong>Ana Gil</strong> (<a href="https://ana.es" rel="nofollow ugc">web</a>): Un sitio precioso, volveremos &lt;3</li>
  <li><strong>Seán O'27Brien</strong>: &lt;b>Great&lt;/b> & &lt;b>quiet</li>
</ul>
Publicados: 2 · Rechazados: 2 · Spam: 1
```

Ejemplo: Límite de mensajes por email

ENTRADA

```
nombre=Eva&email=eva%40x.es&mensaje=Uno
nombre=Eva&email=EVA%40x.es&mensaje=Dos
nombre=Eva&email=eva%40x.es&mensaje=Tres
```

SALIDA ESPERADA

```
Mensaje 3 rechazado: email: ya ha enviado 2 mensajes
<ul class="visitas">
  <li><strong>Eva</strong>: Uno</li>
  <li><strong>Eva</strong>: Dos</li>
</ul>
Publicados: 2 · Rechazados: 1 · Spam: 0
```

Código de partida (php)

```
<?php
const MAX_POR_EMAIL = 2;

/** Errores de un mensaje (campo → motivo), en el orden de los campos. */
function validar(array $f): array {
    $e = [];
    // TODO: nombre, email, web y mensaje
    return $e;
}

/** Escapa para HTML: <, >, &, comillas dobles y simples. */
function h(string $texto): string {
    // TODO
    return $texto;
}

// TODO: lee cada envío con parse_str, descarta el spam, valida y escribe la lista HTML y el resumen
```

48. Horas trabajadas con nocturnidad usando DateTime

Difícil · PHP · 70 minutos · apuntesdam.com/ejercicios/php/horas-trabajadas-y-nocturnidad

Un hospital paga a su personal de guardias por horas, con un plus del 25 % por cada hora trabajada de noche (de 22:00 a 06:00). Los fichajes se exportan del control de presencia con la entrada y la salida de cada turno, y a fin de mes hay que calcular cuánto cobra cada persona.

Las fechas son la parte delicada: un turno de 22:00 a 06:30 cruza la medianoche, y uno de 20:00 a 08:00 tiene unas horas nocturnas y otras no. Calcularlo con horas sueltas lleva a errores; con objetos de fecha y la intersección de intervalos sale solo.

El control de presencia comete errores: fechas que no existen, salidas anteriores a la entrada, turnos que se olvidaron de cerrar (de más de 12 horas) o fichajes duplicados que se solapan. Esos turnos se rechazan con un mensaje y no se pagan.

Requisitos

- La primera línea es **TARIFA euros** (euros por hora, con hasta dos decimales); si no lo es, escribe **Falta la tarifa** y nada más. Cada línea siguiente es un turno: **nombre;AAAA-MM-DD HH:MM;AAAA-MM-DD HH:MM** (entrada y salida).
- Un turno se rechaza, escribiendo **Línea N: motivo** (N es el número de línea de la entrada), si: no tiene ese formato o alguna fecha no existe (**formato no válido**); la salida no es posterior a la entrada (**la salida no es posterior a la entrada**); dura más de 12 horas (**un turno no puede pasar de 12 horas**); o se solapa con un turno ya aceptado de la misma persona (**se solapa con otro turno de nombre**). Dos turnos que solo se tocan (uno termina cuando empieza el otro) no se solapan.
- Horas nocturnas: los minutos del turno que caen entre las 22:00 y las 06:00 de cualquier noche.
- Importe de cada persona: todos sus minutos a la tarifa más un 25 % de la tarifa por cada minuto nocturno, redondeado al céntimo.
- Escribe una línea por persona, por orden alfabético: **nombre: T turnos, D (nocturnas N) · importe**, con «1 turno» en singular y las duraciones como **38 h 30 min** o **8 h**. Al final, **Total a pagar: X**. Los importes con punto de millar, coma decimal y **€**.

Formato de la entrada

- Línea 1: **TARIFA euros**. Resto: **nombre;entrada;salida** con fechas **AAAA-MM-DD HH:MM**.

Ejemplo: Guardias de una semana

ENTRADA

```
TARIFA 14.50
Ana Gil;2026-09-28 22:00;2026-09-29 06:30
Luis Mora;2026-09-28 08:00;2026-09-28 15:00
Ana Gil;2026-09-30 20:00;2026-10-01 08:00
Luis Mora;2026-09-29 15:00;2026-09-29 23:15
```

SALIDA ESPERADA

```
Ana Gil: 2 turnos, 20 h 30 min (nocturnas 16 h) ·
355,25 €
Luis Mora: 2 turnos, 15 h 15 min (nocturnas 1 h
15 min) · 225,66 €
Total a pagar: 580,91 €
```

Ejemplo: Turnos rechazados

ENTRADA

```
TARIFA 12
Eva;2026-09-31 08:00;2026-09-31 14:00
Eva;2026-10-01 14:00;2026-10-01 08:00
Eva;2026-10-01 08:00;2026-10-01 21:00
Eva;2026-10-02 08:00;2026-10-02 14:00
Eva;2026-10-02 13:00;2026-10-02 18:00
Eva;2026-10-02 14:00;2026-10-02 18:00
Eva sin fechas
```

SALIDA ESPERADA

```
Línea 2: formato no válido
Línea 3: la salida no es posterior a la entrada
Línea 4: un turno no puede pasar de 12 horas
Línea 6: se solapa con otro turno de Eva
Línea 8: formato no válido
Eva: 2 turnos, 10 h (nocturnas 0 h) · 120,00 €
Total a pagar: 120,00 €
```

Código de partida (php)

```
<?php
/** Minutos de [$inicio, $fin) que caen entre las 22:00 y las 06:00. */
function minutosNocturnos(DateTimeImmutable $inicio, DateTimeImmutable $fin): int {
    // TODO
    return 0;
}

function euros(int $c): string {
    return number_format($c / 100, 2, ",", ".") . " €";
}

$lineas = explode("\n", stream_get_contents(STDIN));
// TODO: lee la tarifa y los turnos, válídalos y escribe las horas y el importe de cada persona
```

Soluciones

1. Analiza una petición HTTP

Medio · Arquitecturas y Tecnologías del Servidor · apuntesdam.com/subject/dwes/topic/arquitecturas-servidor

Solución (php)

```
<?php
$texto = stream_get_contents(STDIN);
$lineas = explode("\n", str_replace("\r", '', $texto));

// Línea de petición: MÉTODO URL VERSIÓN
[$metodo, $url] = explode(' ', $lineas[0]);
echo "Método: $metodo\n";
echo 'Ruta: ', parse_url($url, PHP_URL_PATH), "\n";
parse_str((string) parse_url($url, PHP_URL_QUERY), $get);
foreach ($get as $nombre => $valor) {
    echo "GET $nombre = $valor\n";
}

// Cabeceras: hasta la primera línea vacía
$cabeceras = [];
$i = 1;
while ($i < count($lineas) && $lineas[$i] !== '') {
    [$nombre, $valor] = explode(':', $lineas[$i], 2);
    $cabeceras[trim($nombre)] = trim($valor);
    $i++;
}
echo 'Host: ', $cabeceras['Host'] ?? '', "\n";
echo 'Cabeceras: ', count($cabeceras), "\n";

// Cuerpo: lo que queda tras la línea vacía
parse_str(trim(implode("\n", array_slice($lineas, $i + 1))), $post);
foreach ($post as $nombre => $valor) {
    echo "POST $nombre = $valor\n";
}
```

Una petición HTTP es texto con una estructura fija: la primera línea (método, URL y versión), las cabeceras «Nombre: valor» hasta una línea vacía y, después, el cuerpo. El código la recorre en ese orden, con un índice que avanza hasta la línea vacía.

`parse_url` separa la ruta de la query string y `parse_str` convierte una query string (o un cuerpo de formulario, que tiene el mismo formato) en un array, decodificando ya los `%C3%B1` y los `+`. Es lo que hace PHP por ti al rellenar `$_GET` y `$_POST`.

2. Construye la respuesta

Medio · Arquitecturas y Tecnologías del Servidor · apuntesdam.com/subject/dwes/topic/arquitecturas-servidor

Solución (php)

```
<?php
$textos = [
    200 => 'OK',
    201 => 'Created',
    204 => 'No Content',
    301 => 'Moved Permanently',
    404 => 'Not Found',
    500 => 'Internal Server Error',
];

$respuestas = [];
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$codigo, $tipo, $cuerpo] = explode(';', $linea, 3);
    $codigo = (int) $codigo;
    if (!isset($textos[$codigo])) {
        $respuestas[] = "Código no soportado: $codigo";
        continue;
    }
    $respuesta = "HTTP/1.1 $codigo {$textos[$codigo]}";
    if ($codigo !== 204) { // 204 no lleva cuerpo ni cabeceras de contenido
        $respuesta .= "\nContent-Type: $tipo; charset=utf-8"
            . "\nContent-Length: " . strlen($cuerpo)
            . "\n\n$cuerpo";
    }
    $respuestas[] = $respuesta;
}
echo implode("\n---\n", $respuestas), "\n";
```

La respuesta se construye en el orden en que viaja: línea de estado, cabeceras, una línea en blanco y el cuerpo. El texto de cada código sale de un array asociativo, y `isset` detecta los códigos que no están.

Content-Length cuenta bytes, no caracteres, y por eso se usa `strlen` (que cuenta bytes) y no `mb_strlen`: en UTF-8, la «ñ» ocupa dos. El `explode` con límite 3 deja el cuerpo entero aunque contenga punto y coma.

3. Un enrutador con parámetros

Difícil · Arquitecturas y Tecnologías del Servidor · apuntesdam.com/subject/dwes/topic/arquitecturas-servidor

Solución (php)

```
<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
$separador = array_search('---', $lineas);
$ rutas = array_slice($lineas, 0, $separador);
$ peticiones = array_slice($lineas, $separador + 1);

// Sin barra final, salvo la raíz
$normalizar = fn(string $ruta) => rtrim($ruta, '/') === '' ? '/' : rtrim($ruta, '/');

// Cada patrón se convierte en una expresión regular: {id} pasa a ser un grupo con nombre
$definidas = [];
foreach ($rutas as $r) {
    [$m, $patron, $accion] = explode(' ', $r);
    $regex = '#^' . preg_replace('#\{(\w+)\}#', '(?P<$1>[^\s/]+)', $normalizar($patron)) . '$#';
    $definidas[] = [$m, $regex, $accion];
}

foreach ($peticiones as $peticion) {
    [$metodo, $url] = explode(' ', $peticion);
    $ruta = $normalizar((string) parse_url($url, PHP_URL_PATH));
    $encontrada = null;
    $permitidos = [];
    foreach ($definidas as [$m, $regex, $accion]) {
        if (!preg_match($regex, $ruta, $grupos)) {
            continue;
        }
        $permitidos[] = $m;
        if ($m === $metodo && $encontrada === null) { // gana la primera que encaja
            $params = [];
            foreach ($grupos as $nombre => $valor) {
                if (is_string($nombre)) {
                    $params[] = "$nombre=$valor";
                }
            }
            $encontrada = trim("$accion " . implode(' ', $params));
        }
    }
    if ($encontrada !== null) {
        echo $encontrada, "\n";
    } elseif ($permitidos) {
        echo '405 Método no permitido (' . implode(' ', array_unique($permitidos)) . ")\n";
    } else {
        echo "404 No encontrado\n";
    }
}
```

Cada patrón se traduce a una expresión regular: `{id}` se sustituye por `(?P<id>[^\s/]+)`, un grupo con nombre que captura un trozo de ruta sin barras. Así `preg_match` comprueba si la URL encaja y, a la vez, extrae los parámetros con su nombre.

Para distinguir el 404 del 405 se recorren todas las rutas: las que encajan con la URL aportan su método a la lista de permitidos, y la primera que además coincide en método es la que se ejecuta. Si ninguna encaja con la URL, la ruta no existe; si encaja alguna pero con otro método, el problema es el método.

`preg_match` devuelve cada grupo con nombre dos veces (por nombre y por número), por eso al montar los parámetros solo se toman las claves de tipo texto.

4. Tabla de multiplicar en HTML

Fácil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$texto = trim((string) fgets(STDIN));

$n = filter_var($texto, FILTER_VALIDATE_INT, ['options' => ['min_range' => 1, 'max_range' => 100]]);
if ($n === false) {
    echo "<p>Número no válido</p>\n";
    exit;
}
echo "<table>\n";
for ($i = 1; $i <= 10; $i++) {
    echo "    <tr><td>$n x $i</td><td>, $n * $i, "</td></tr>\n";
}
echo "</table>\n";
```

`filter_var` con `FILTER_VALIDATE_INT` comprueba a la vez que el texto sea un entero y que esté en el rango, y devuelve `false` si no lo es; se compara con `=== false` porque un 0 válido también sería «falso» con `==`.

El HTML se genera escribiendo texto: PHP sustituye `$n` y `$i` dentro de las comillas dobles, y el producto se pasa a `echo` como otro argumento porque las operaciones no se pueden escribir dentro de la cadena.

5. Encuentra el fallo: la frase que empieza por hola

Fácil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$frases = ["hola mundo", "dice hola", "adiós"];
foreach ($frases as $f) {
    if (strpos($f, "hola") !== false) {
        echo "$f: contiene hola\n";
    } else {
        echo "$f: no contiene hola\n";
    }
}
```

`strpos` devuelve la posición donde aparece el texto, y en «hola mundo» esa posición es 0. Como el `if` convierte el 0 en falso, la frase parecía no contenerlo. `strpos` devuelve `false` (no 0) cuando no encuentra nada, así que hay que comparar con `!== false`, que distingue el tipo.

En PHP 8 es más claro `str_contains($f, "hola")`, que devuelve directamente un booleano.

6. Encuentra el fallo: el saludo que suma

Fácil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$nombre = "Ana";
$mensajes = 3;
echo "Hola, " . $nombre . ". Tienes " . $mensajes . " mensajes.\n";
```

En PHP el operador para unir cadenas es el punto (`.`), no el `+`. Con `+`, PHP intenta sumar números, y en PHP 8 sumar un texto que no es numérico como «Hola, » lanza un `TypeError`.

Otra forma es la interpolación: `"Hola, $nombre. Tienes $mensajes mensajes.\n"`.

7. Las palabras más repetidas

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$texto = stream_get_contents(STDIN);

// Palabras = secuencias de letras (la u hace que \p{L} entienda UTF-8: tildes y ñ)
preg_match_all('/\p{L}+/u', mb_strtolower($texto), $m);
$cuenta = array_count_values($m[0]);

$palabras = array_keys($cuenta);
usort($palabras, fn($a, $b) => $cuenta[$b] <=> $cuenta[$a] ?: strcmp($a, $b));

foreach (array_slice($palabras, 0, 3) as $p) {
    echo "$p: {$cuenta[$p]}\n";
}
```

`preg_match_all` con `\p{L}+` y el modificador `u` extrae todas las palabras, incluidas las que llevan tilde o ñ; antes se pasa el texto a minúsculas con `mb_strtolower`, que, a diferencia de `strtolower`, entiende UTF-8.

`array_count_values` cuenta las apariciones de cada una de una vez.

Para ordenar por dos criterios se usa el operador nave espacial: `$cuenta[$b] <=> $cuenta[$a]` ordena de más a menos y, si empatan (da 0), `?:` pasa al segundo criterio, el orden alfabético.

8. Ticket de compra

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);

$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';
$base = 0;
foreach ($lineas as $linea) {
    $campos = explode(';', $linea);
    if (count($campos) !== 3 || !is_numeric($campos[1]) || filter_var($campos[2], FILTER_VALIDATE_INT)
        === false) {
        echo "Línea ignorada: $linea\n";
        continue;
    }
    [$producto, $precio, $cantidad] = [$campos[0], (float) $campos[1], (int) $campos[2]];
    $importe = $precio * $cantidad;
    $base += $importe;
    echo "$producto x $cantidad = ", $euros($importe), "\n";
}
$iva = round($base * 0.21, 2);
echo 'Base: ', $euros($base), "\n";
echo 'IVA (21%): ', $euros($iva), "\n";
echo 'Total: ', $euros($base + $iva), "\n";
```

Cada línea se valida antes de usarla: tres campos, un precio numérico (`is_numeric`) y una cantidad entera (`FILTER_VALIDATE_INT`). Las que no cumplen se informan y se saltan con `continue`, sin romper el resto del ticket.

`number_format($n, 2, ',', '.')` da el formato español: dos decimales, coma decimal y punto de miles. Se guarda en una función flecha (`$euros`) para no repetirlo, y el IVA se redondea a céntimos antes de sumarlo, como en una factura real.

9. Convertir y validar los datos de un formulario

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$campo, $valor] = explode('=', $linea, 2) + [1 => ''];
    $valor = trim($valor);

    if (!in_array($campo, ['edad', 'email', 'nota', 'web'], true)) {
        continue; // campo desconocido
    }
    $resultado = match ($campo) {
        'edad' => filter_var($valor, FILTER_VALIDATE_INT, ['options' => ['min_range' => 0, 'max_range' => 120]]),
        'email' => filter_var($valor, FILTER_VALIDATE_EMAIL),
        'nota' => filter_var(str_replace(',', '.', $valor), FILTER_VALIDATE_FLOAT, ['options' => ['min_range' => 0, 'max_range' => 10]]),
        'web' => preg_match('#^https?://#i', $valor) ? filter_var($valor, FILTER_VALIDATE_URL) : false,
    };

    if ($resultado === false) {
        echo "$campo: no válido\n";
    } else {
        $texto = is_float($resultado) ? number_format($resultado, 2, '.', '') : $resultado;
        echo "$campo: $texto (" . get_debug_type($resultado) . ")\n";
    }
}
```

Cada campo tiene su filtro de `filter_var`: entero con rango, email, decimal con rango y URL. Los filtros no solo comprueban: también convierten, así que la edad sale como `int` y la nota como `float`, que es lo que muestra `get_debug_type`.

Dos detalles: la nota admite coma decimal, así que se cambia por un punto antes de validarla, y `FILTER_VALIDATE_URL` acepta también `ftp://`, por lo que la web se comprueba además con una expresión que exige http o https. El resultado se compara con `=== false`, porque una edad 0 es válida.

10. Encuentra el fallo: la subida de precios que no sube

Medio · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$precios = ["pan" => 1.20, "leche" => 0.95, "café" => 4.50];
foreach ($precios as $producto => $p) {
    $precios[$producto] = round($p * 1.10, 2);
}
foreach ($precios as $producto => $p) {
    echo "$producto: $p €\n";
}
```

`foreach` trabaja con una copia de cada valor: cambiar `$p` dentro del bucle no cambia el array. Para modificarlo hay que escribir en el propio array usando la clave (`$precios[$producto] = ...`).

La alternativa es recorrerlo por referencia (`foreach ($precios as &$p)`), pero entonces hay que hacer `unset($p)` al terminar: si no, `$p` sigue apuntando al último elemento y el siguiente `foreach` lo sobrescribe. Por eso la versión con la clave es más segura.

11. Calendario del mes en HTML

Difícil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$texto = trim((string) fgets(STDIN));

if (!preg_match('/^\d{4}-\d{2}$/', $texto, $m) || !checkdate((int) $m[2], 1, (int) $m[1])) {
    echo "<p>Mes no válido</p>\n";
    exit;
}
$primero = new DateTimeImmutable("$texto-01");
$dias = (int) $primero->format('t'); // días del mes
$huecos = (int) $primero->format('N') - 1; // N: 1 = lunes ... 7 = domingo

$celdas = array_merge(array_fill(0, $huecos, ''), range(1, $dias));
while (count($celdas) % 7 !== 0) {
    $celdas[] = ''; // completar la última semana
}

echo "<table>\n";
echo "  <tr><th>L</th><th>M</th><th>X</th><th>J</th><th>V</th><th>S</th><th>D</th></tr>\n";
foreach (array_chunk($celdas, 7) as $semana) {
    echo '  <tr>', implode(',', array_map(fn($d) => "<td>$d</td>", $semana)), "</tr>\n";
}
echo "</table>\n";
```

La validación tiene dos partes: la expresión regular comprueba el formato AAAA-MM y `checkdate` que el mes exista. Después, `DateTimeImmutable` da los dos datos que hacen falta: cuántos días tiene el mes (`t`) y en qué día de la semana cae el 1 (`N` , que ya empieza en lunes).

El calendario se construye como una lista plana de celdas: tantos huecos vacíos como días de la semana hay antes del 1, los días del mes y huecos al final hasta completar múltiplos de 7. `array_chunk` la parte en semanas, y cada semana es una fila.

12. Tu propio motor de plantillas

Muy difícil · Fundamentos de PHP · apuntesdam.com/subject/dwes/topic/php-fundamentos

Solución (php)

```
<?php
$datos = json_decode(fgets(STDIN), true);
$plantilla = stream_get_contents(STDIN);

// Valor de una ruta con puntos (usuario.email) dentro del contexto; null si no existe
function valor(string $ruta, array $ctx): mixed
{
    $v = $ctx;
    foreach (explode('.', $ruta) as $clave) {
        if (!is_array($v) || !array_key_exists($clave, $v)) {
            return null;
        }
        $v = $v[$clave];
    }
    return $v;
}

function renderizar(string $tpl, array $ctx): string
{
    // 1. Bucles: el cuerpo se procesa una vez por elemento, con la variable del bucle en el contexto
    $tpl = preg_replace_callback('/\{%s*for\s+(\w+)\s+in\s+([\w.]+)\s*%\}(.*)\{%s*endfor\s*%\}/s',
        function ($m) use ($ctx) {
            $salida = '';
            foreach ((array) (valor($m[2], $ctx) ?? []) as $elemento) {
                $salida .= renderizar($m[3], [$m[1] => $elemento] + $ctx);
            }
            return $salida;
        }, $tpl);
    // 2. Condiciones
    $tpl = preg_replace_callback('/\{%s*if\s+(\w+)\s*%\}(.*)\{%s*endif\s*%\}/s',
        fn($m) => valor($m[1], $ctx) ? renderizar($m[2], $ctx) : '', $tpl);
    // 3. Variables: primero las de tres llaves (sin escapar), después las de dos (escapadas)
    $tpl = preg_replace_callback('/\{\{\s*([\w.]+)\s*\}\}/', fn($m) => (string) valor($m[1], $ctx),
        $tpl);
    return preg_replace_callback('/\{\s*([\w.]+)\s*\}/', fn($m) => htmlspecialchars((string)
        valor($m[1], $ctx)), $tpl);
}

echo renderizar($plantilla, $datos);
```

El motor procesa la plantilla por capas con `preg_replace_callback`, que sustituye cada coincidencia de una expresión regular por lo que devuelva una función. El orden importa: primero los bucles (porque su cuerpo puede tener condiciones y variables que dependen del elemento), luego las condiciones y por último las variables.

El cuerpo de un bucle se procesa llamando otra vez a `renderizar` con un contexto ampliado: `[$m[1] => $elemento] + $ctx` añade la variable del bucle, y el `+` de arrays da prioridad a la de la izquierda. Las expresiones usan `.*?` (no codicioso) y el modificador `s` para que el cuerpo pueda ocupar varias líneas.

Las variables de tres llaves se sustituyen antes que las de dos; si no, la expresión de dos llaves encontraría `{{ firma }}` dentro de `{{{ firma }}}`. Escapar por defecto y exigir una sintaxis especial para no hacerlo es justo lo que hacen Twig y Blade para evitar XSS.

13. Comentarios sin XSS

Fácil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Solución (php)

```
<?php
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    if (!str_contains($linea, '|')) {
        continue;
    }
    [$autor, $texto] = explode('|', $linea, 2);
    $autor = htmlspecialchars($autor, ENT_QUOTES, 'UTF-8');
    $texto = htmlspecialchars($texto, ENT_QUOTES, 'UTF-8');
    echo "<li><strong>$autor</strong>: $texto</li>\n";
}
```

`htmlspecialchars` convierte los caracteres con significado en HTML (<, >, & y las comillas) en entidades, así que el navegador los muestra como texto en lugar de interpretarlos como etiquetas: un `<script>` escrito por un usuario se ve, pero no se ejecuta.

Se escapan los dos datos, también el autor, porque cualquier cosa que venga del usuario es sospechosa. `ENT_QUOTES` hace que se escapen también las comillas simples, necesarias si el dato acaba dentro de un atributo.

14. Validar los datos de un formulario

Medio · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Solución (php)

```
<?php
parse_str(trim((string) fgets(STDIN)), $datos);
$errores = [];

$nombre = trim($datos['nombre'] ?? '');
if ($nombre === '') {
    $errores[] = 'nombre: obligatorio';
} elseif (mb_strlen($nombre) > 50) {
    $errores[] = 'nombre: máximo 50 caracteres';
}
if (filter_var($datos['email'] ?? '', FILTER_VALIDATE_EMAIL) === false) {
    $errores[] = 'email: no válido';
}
if (filter_var($datos['edad'] ?? '', FILTER_VALIDATE_INT, ['options' => ['min_range' => 18, 'max_range' => 120]]) === false) {
    $errores[] = 'edad: debe estar entre 18 y 120';
}
$web = trim($datos['web'] ?? '');
if ($web !== '' && filter_var($web, FILTER_VALIDATE_URL) === false) { // opcional
    $errores[] = 'web: no válida';
}

echo $errores ? implode("\n", $errores) . "\n" : "Datos correctos\n";
```

Los datos de un formulario pueden no llegar, así que cada uno se lee con `?? ''`. El nombre se recorta antes de comprobarlo (un nombre de espacios no es un nombre) y se mide con `mb_strlen`, que cuenta caracteres y no bytes.

Los errores se acumulan en un array en lugar de parar en el primero: así el usuario ve todo lo que tiene que corregir de una vez. Un campo opcional, como la web, solo se valida si trae algo.

15. Un carrito guardado en la sesión

Medio · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Solución (php)

```
<?php
$catalogo = [
    't1' => ['nombre' => 'Teclado', 'precio' => 24.99, 'stock' => 5],
    'r1' => ['nombre' => 'Ratón', 'precio' => 12.50, 'stock' => 2],
    'm1' => ['nombre' => 'Monitor', 'precio' => 189.00, 'stock' => 1],
];
$_SESSION = ['carrito' => []];
$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    $partes = explode(' ', trim($linea));
    $accion = $partes[0];
    $id = $partes[1] ?? '';
    if ($accion === 'anadir') {
        $cantidad = filter_var($partes[2] ?? '', FILTER_VALIDATE_INT);
        $actual = $_SESSION['carrito'][$id] ?? 0;
        if (!isset($catalogo[$id])) {
            echo "Error: producto inexistente\n";
        } elseif ($cantidad === false || $cantidad <= 0) {
            echo "Error: cantidad no válida\n";
        } elseif ($actual + $cantidad > $catalogo[$id]['stock']) {
            echo "Error: stock insuficiente\n";
        } else {
            $_SESSION['carrito'][$id] = $actual + $cantidad;
        }
    } elseif ($accion === 'quitar') {
        unset($_SESSION['carrito'][$id]);
    } elseif ($accion === 'vaciar') {
        $_SESSION['carrito'] = [];
    } elseif ($accion === 'ver') {
        if (!$_SESSION['carrito']) {
            echo "Carrito vacío\n";
            continue;
        }
        $total = 0;
        foreach ($_SESSION['carrito'] as $clave => $n) {
            $importe = $catalogo[$clave]['precio'] * $n;
            $total += $importe;
            echo "{$catalogo[$clave]['nombre']} x $n = ", $euros($importe), "\n";
        }
        echo 'Total: ', $euros($total), "\n";
    }
}
```

El carrito guarda solo id → cantidad; los nombres y precios se toman siempre del catálogo, que es la fuente fiable. Guardar el precio en la sesión permitiría que un cambio de precio no se viera, y nunca hay que fiarse de un precio que llegue del cliente.

Antes de añadir se comprueba todo y solo si pasa se modifica la sesión, así un error deja el carrito como estaba. El stock se compara con lo que ya hay en el carrito más lo que se quiere añadir, no solo con lo nuevo. Los arrays de PHP conservan el orden de inserción, por eso «ver» lista en el orden en que se añadieron.

16. Encuentra el fallo: el comentario que ejecuta código

Medio · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Solución (php)

```
<?php
function tarjeta(string $autor, string $texto): string {
    $autor = htmlspecialchars($autor, ENT_QUOTES, 'UTF-8');
    $texto = htmlspecialchars($texto, ENT_QUOTES, 'UTF-8');
    return "<p><b>$autor</b>: $texto</p>";
}

echo tarjeta("Ana", "¡Me encanta!") . "\n";
echo tarjeta("Pirata", "<script>alert('hola')</script>") . "\n";
```

El fallo es meter en el HTML texto que viene de un usuario sin escaparlos. `htmlspecialchars` convierte `<`, `>`, `&` y las comillas en entidades (`<`; ...) que el navegador muestra como texto, así el `<script>` se ve pero no se ejecuta.

Con `ENT_QUOTES` también se escapan las comillas simples (`'`), lo que protege además cuando el dato acaba dentro de un atributo HTML.

17. Inicio de sesión con bloqueo

Difícil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Solución (php)

```
<?php
$usuarios = [
    'ana' => password_hash('Secreta123', PASSWORD_DEFAULT),
    'luis' => password_hash('DAW 2025!', PASSWORD_DEFAULT),
];
$fallos = [];

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$usuario, $clave] = array_pad(explode(' ', $linea, 2), 2, '');
    if (($fallos[$usuario] ?? 0) >= 3) {
        echo "Cuenta bloqueada\n";
        continue;
    }
    if (isset($usuarios[$usuario]) && password_verify($clave, $usuarios[$usuario])) {
        $fallos[$usuario] = 0; // un acceso correcto reinicia los fallos
        echo "Bienvenido/a, $usuario\n";
    } else {
        $fallos[$usuario] = ($fallos[$usuario] ?? 0) + 1;
        echo "Usuario o contraseña incorrectos\n";
    }
}
```

Las contraseñas nunca se comparan directamente: `password_verify` calcula el hash de lo que se ha escrito con la misma sal y el mismo coste que el guardado y los compara. El bloqueo se comprueba antes que la contraseña, para que ni siquiera una contraseña correcta entre en una cuenta bloqueada.

El mensaje de error es el mismo exista o no el usuario: si fueran distintos, un atacante podría averiguar qué usuarios existen probando nombres.

18. Cookies firmadas

Difícil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Solución (php)

```
<?php
const CLAVE = 'b7f1c0a9e2d34f5a8c6b1e0d9f2a4c7e';

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$orden, $dato] = explode(' ', $linea, 2) + [1 => ''];
    if ($orden === 'firmar') {
        echo $dato, '.', hash_hmac('sha256', $dato, CLAVE), "\n";
    } elseif ($orden === 'verificar') {
        $punto = strrpos($dato, '.');           // la firma va tras el ÚLTIMO punto
        if ($punto === false) {
            echo "Manipulada\n";
            continue;
        }
        $valor = substr($dato, 0, $punto);
        $firma = substr($dato, $punto + 1);
        echo hash_equals(hash_hmac('sha256', $valor, CLAVE), $firma) ? "Válida: $valor\n" :
            "Manipulada\n";
    }
}
```

Firmar es calcular un HMAC del valor con una clave que solo conoce el servidor. Para verificar se vuelve a calcular con el valor recibido: si alguien cambió el valor (o la firma), no coinciden, y sin la clave no puede fabricar una firma buena.

Como el valor puede contener puntos, la firma se separa con `strrpos`, que busca el último punto. La comparación se hace con `hash_equals`, que tarda lo mismo acierte o falle, para no dar pistas a quien intente adivinar la firma midiendo tiempos.

19. Control de acceso por roles y propiedad

Muy difícil · Formularios, Sesiones y Seguridad · apuntesdam.com/subject/dwes/topic/formularios-sesiones

Solución (php)

```
<?php
$bloques = explode("---", stream_get_contents(STDIN));
$lineas = fn(string $b) => array_filter(array_map('trim', explode("\n", $b)));

// Roles: nombre -> ['padre' => ..., 'permisos' => [...]]
$roles = [];
foreach ($lineas($bloques[0]) as $l) {
    if (preg_match('/^rol\s+(\w+)(?:\s+hereda\s+(\w+))?\s*:\s*(.*)$/', $l, $m)) {
        $roles[$m[1]] = [
            'padre' => $m[2] !== '' ? $m[2] : null,
            'permisos' => array_filter(array_map('trim', explode(',', $m[3]))),
        ];
    }
}

// Permisos de un rol: los suyos y los de sus antepasados (con protección frente a ciclos)
function permisos(string $rol, array $roles, array $vistos = []): array
{
    if (!isset($roles[$rol]) || in_array($rol, $vistos, true)) {
        return [];
    }
    $vistos[] = $rol;
    $proprios = $roles[$rol]['permisos'];
    $padre = $roles[$rol]['padre'];
    return $padre === null ? $proprios : array_merge($proprios, permisos($padre, $roles, $vistos));
}

$usuarios = [];
foreach ($lineas($bloques[1] ?? '') as $l) {
    [$nombre, $rol] = array_map('trim', explode(':', $l, 2));
    $usuarios[$nombre] = $rol;
}

foreach ($lineas($bloques[2] ?? '') as $l) {
    $p = preg_split('/\s+/', $l);
    [$usuario, $accion] = $p;
    $propietario = $p[2] ?? null;
    $permitido = false;
    if (isset($usuarios[$usuario])) {
        $suyos = permisos($usuarios[$usuario], $roles);
        $permitido = in_array($accion, $suyos, true)
            || (in_array($accion . '_propio', $suyos, true) && $propietario === $usuario);
    }
    echo "$usuario $accion: ", $permitido ? 'permitido' : 'denegado', "\n";
}
```

Los roles forman un árbol de herencia, y los permisos de un rol se calculan recursivamente: los suyos más los de su padre, que a su vez incluyen los del suyo. La lista `$vistos` corta la recursión si dos roles se heredan mutuamente, que de otro modo sería un bucle infinito.

La comprobación tiene dos caminos: tener el permiso general (editar cualquier artículo) o tener la versión `_propio` y ser el dueño del recurso. Ese segundo caso es el que más se olvida en las aplicaciones reales: comprobar que el usuario puede editar, pero no que lo que edita es suyo.

Todo lo que no se puede demostrar que está permitido se deniega: usuario desconocido, rol inexistente o permiso que no aparece.

20. Una entidad que se valida sola

Medio · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Solución (php)

```
<?php
class Producto
{
    public function __construct(
        public string $nombre,
        public float $precio,
        public int $stock = 0,
    ) {
        if (trim($nombre) === '') {
            throw new InvalidArgumentException('El nombre es obligatorio');
        }
        if ($precio < 0) {
            throw new InvalidArgumentException('El precio no puede ser negativo');
        }
        if ($stock < 0) {
            throw new InvalidArgumentException('El stock no puede ser negativo');
        }
    }

    public function precioConIva(): float
    {
        return $this->precio * 1.21;
    }

    public function __toString(): string
    {
        $euros = fn(float $n) => number_format($n, 2, ',', '.'). ' €';
        return "{$this->nombre}: {$euros($this->precio)} ({$euros($this->precioConIva())} con IVA) ·
            stock {$this->stock}";
    }
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$nombre, $precio, $stock] = array_pad(explode(';', $linea), 3, '0');
    try {
        echo new Producto(trim($nombre), (float) $precio, (int) $stock), "\n";
    } catch (InvalidArgumentException $e) {
        echo 'Error: ', $e->getMessage(), "\n";
    }
}
```

La promoción de propiedades declara y asigna los atributos directamente en los parámetros del constructor (`public string $nombre`), y el cuerpo del constructor solo tiene que validar. Si algo no es válido se lanza una excepción, así que nunca llega a existir un `Producto` con datos incorrectos: la entidad se protege a sí misma.

`__toString` se llama automáticamente cuando el objeto se usa como texto, como en el `echo` del programa. Dentro de una cadena con comillas dobles, `{ $euros(...) }` llama a la función flecha y mete el resultado en el texto.

21. Nóminas con herencia e interfaces

Medio · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Solución (php)

```
<?php
interface Bonificable
{
    public function bonus(): float;
}

abstract class Empleado
{
    public function __construct(public readonly string $nombre) {}

    abstract public function salario(): float;
}

class Fijo extends Empleado
{
    public function __construct(string $nombre, protected float $base, protected int $antiguedad)
    {
        parent::__construct($nombre);
    }

    public function salario(): float
    {
        return $this->base + 50 * intdiv($this->antiguedad, 3);    // 50 € por trienio completo
    }
}

class PorHoras extends Empleado
{
    public function __construct(string $nombre, private float $horas, private float $precio)
    {
        parent::__construct($nombre);
    }

    public function salario(): float
    {
        $normales = min($this->horas, 160);
        $extra = max($this->horas - 160, 0);
        return $normales * $this->precio + $extra * $this->precio * 1.5;
    }
}

class Comercial extends Fijo implements Bonificable
{
    public function __construct(string $nombre, float $base, int $antiguedad, private float $ventas)
    {
        parent::__construct($nombre, $base, $antiguedad);
    }

    public function salario(): float
    {
        return parent::salario() + $this->ventas * 0.05;    // lo de un fijo más la comisión
    }

    public function bonus(): float
    {
        return $this->ventas > 20000 ? 100 : 0;
    }
}

$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';
$total = 0;
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    $d = explode('; ', $linea);
    $e = match ($d[0]) {
        'fijo' => new Fijo($d[1], (float) $d[2], (int) $d[3]),
        'horas' => new PorHoras($d[1], (float) $d[2], (float) $d[3]),
    }
}
```

```

        'comercial' => new Comercial($d[1], (float) $d[2], (int) $d[3], (float) $d[4]),
    );
    $importe = $e->salario() + ($e instanceof Bonificable ? $e->bonus() : 0);
    $total += $importe;
    echo "{$e->nombre}: {$euros($importe)}\n";
}
echo "Total: {$euros($total)}\n";

```

Cada clase solo escribe lo que la diferencia. `Comercial` hereda de `Fijo`, así que su salario es `parent::salario()` (base y trienios) más la comisión: no repite el cálculo de los trienios. `intdiv` da los trienios completos con una división entera.

La interfaz `Bonificable` expresa una capacidad que no depende de la jerarquía: el código de la nómina pregunta `instanceof Bonificable` y suma el bonus solo a quien lo tiene, sin saber qué clase concreta es. Si mañana otro tipo de empleado tuviera bonus, bastaría con que implementara la interfaz.

22. Encuentra el fallo: los intereses que no llegan

Medio · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Solución (php)

```

<?php
class Cuenta {
    protected float $saldo = 0;
    public function ingresar(float $c): void { $this->saldo += $c; }
    public function getSaldo(): float { return $this->saldo; }
}

class CuentaAhorro extends Cuenta {
    public function aplicarIntereses(float $tipo): void {
        $this->saldo += $this->saldo * $tipo / 100;
    }
}

$c = new CuentaAhorro();
$c->ingresar(1000);
$c->aplicarIntereses(5);
echo "Saldo: " . $c->getSaldo() . " €\n";

```

Un atributo `private` solo es accesible desde la clase que lo declara, ni siquiera desde sus hijas. Dentro de `CuentaAhorro`, `$this->saldo` no era el saldo de `Cuenta`: PHP creaba una propiedad nueva y dinámica (de ahí los avisos), y el saldo real no cambiaba.

Con `protected` el atributo sigue oculto para el código de fuera, pero las subclases pueden usarlo. Es el cambio mínimo; otra opción sería añadir en `Cuenta` un método protegido para modificar el saldo.

23. Un motor de plantillas mínimo

Difícil · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Solución (php)

```
<?php
function render(string $plantilla, array $datos): string
{
    return preg_replace_callback('/\{\{s*([\w.]+)s*(\\|raw)?s*\}\}/', function (array $m) use ($datos):
        string {
            // Recorrer las claves con puntos: usuario.nombre → $datos['usuario']['nombre']
            $valor = $datos;
            foreach (explode('.', $m[1]) as $clave) {
                if (!is_array($valor) || !array_key_exists($clave, $valor)) {
                    return '';
                }
                $valor = $valor[$clave];
            }
            $texto = is_array($valor) ? '' : (string) $valor;
            return !empty($m[2]) ? $texto : htmlspecialchars($texto);
        }, $plantilla);
}

$datos = json_decode((string) fgets(STDIN), true) ?? [];
echo render(stream_get_contents(STDIN), $datos);
```

`preg_replace_callback` busca cada `{{ ... }}` con una expresión regular y lo sustituye por lo que devuelve la función. El grupo `(\\|raw)?` es opcional: si aparece, el valor se devuelve tal cual; si no, se escapa con `htmlspecialchars`. Escapar por defecto es la decisión segura.

Las claves con puntos se resuelven bajando nivel a nivel por el array; en cuanto una clave no existe, se devuelve la cadena vacía. `use ($datos)` hace que la función anónima pueda leer los datos de fuera.

24. Una cadena de middlewares

Difícil · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Solución (php)

```
<?php
$log = [];

$registro = function (array $p, callable $siguiente) use (&$log) {
    $log[] = "> {$p['ruta']}";
    $respuesta = $siguiente($p);
    $log[] = "< {$respuesta[0]}";
    return $respuesta;
};

$autenticacion = fn(array $p, callable $siguiente) => $p['usuario'] === null ? [401, 'Inicia sesión'] :
    $siguiente($p);
$administracion = fn(array $p, callable $siguiente) =>
    str_starts_with($p['ruta'], '/admin') && $p['rol'] !== 'admin' ? [403, 'Prohibido'] : $siguiente($p);
$controlador = fn(array $p) => [200, "Hola, {$p['usuario']}: {$p['ruta']}"];

function ejecutar(array $middlewares, callable $controlador, array $peticion): array
{
    // Se envuelve de dentro hacia fuera: el último middleware rodea al controlador, el primero a todos
    $siguiente = $controlador;
    foreach (array_reverse($middlewares) as $middleware) {
        $siguiente = fn(array $p) => $middleware($p, $siguiente);
    }
    return $siguiente($peticion);
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$ruta, $usuario, $rol] = explode(' ', $linea);
    $peticion = ['ruta' => $ruta, 'usuario' => $usuario === '-' ? null : $usuario, 'rol' => $rol];
    [$codigo, $cuerpo] = ejecutar([$registro, $autenticacion, $administracion], $controlador, $peticion);
    echo "$codigo $cuerpo\n";
}
echo implode(' ', $log), "\n";
```

La cadena se construye como unas muñecas rusas: se empieza por el controlador y, recorriendo los middlewares al revés, cada uno se envuelve en una función que lo llama pasándole el «siguiente» que ya había. Al final, `$siguiente` es el primer middleware, que contiene a todos los demás.

La clave es que la función flecha captura `$middleware` y `$siguiente` por valor en el momento de crearse, así cada capa recuerda la suya aunque las variables cambien en la siguiente vuelta del bucle. Un middleware que devuelve su propia respuesta sin llamar a `$siguiente` corta la cadena, y el registro sigue viendo la salida porque es la capa más externa.

25. Un contenedor de dependencias con autowiring

Muy difícil · POO y el Patrón MVC · apuntesdam.com/subject/dwes/topic/poo-mvc

Solución (php)

```
<?php
$orderen = [];

interface Almacen {}
final class AlmacenMemoria implements Almacen { public function __construct() { global $orden; $orden[] = 'AlmacenMemoria'; } }
final class Reloj { public function __construct(public string $zona = 'Europe/Madrid') { global $orden; $orden[] = 'Reloj'; } }
final class Registro { public function __construct(Reloj $reloj) { global $orden; $orden[] = 'Registro'; } }
final class Repositorio { public function __construct(Almacen $almacen, Registro $registro) { global $orden; $orden[] = 'Repositorio'; } }
final class Servicio { public function __construct(Repositorio $repo, Registro $registro, int $reintentos = 3) { global $orden; $orden[] = "Servicio($reintentos)"; } }
final class Huevo { public function __construct(Gallina $g) {} }
final class Gallina { public function __construct(Huevo $h) {} }
interface Correo {}
final class Avisos { public function __construct(Correo $c) {} }

final class Contenedor
{
    private array $instancias = [];
    private array $enlaces = [];
    private array $enConstruccion = []; // pila de clases que se están creando (para detectar ciclos)

    public function enlazar(string $interfaz, string $clase): void
    {
        $this->enlaces[$interfaz] = $clase;
    }

    public function get(string $clase): object
    {
        $real = $this->enlaces[$clase] ?? $clase;
        if (isset($this->instancias[$real])) {
            return $this->instancias[$real];
        }
        $reflexion = new ReflectionClass($real);
        if ($reflexion->isInterface()) {
            throw new LogicException("Sin implementación para $clase");
        }
        $pos = array_search($real, $this->enConstruccion, true);
        if ($pos !== false) {
            $ciclo = [...array_slice($this->enConstruccion, $pos), $real];
            throw new LogicException('Dependencia circular: ' . implode(' → ', $ciclo));
        }

        $this->enConstruccion[] = $real;
        try {
            $argumentos = [];
            foreach ($reflexion->getConstructor()?->getParameters() ?? [] as $param) {
                $tipo = $param->getType();
                if ($tipo instanceof ReflectionNamedType && !$tipo->isBuiltin()) {
                    $argumentos[] = $this->get($tipo->getName()); // otra clase: se resuelve antes
                } elseif ($param->isDefaultValueAvailable()) {
                    $argumentos[] = $param->getDefaultValue();
                } else {
                    throw new LogicException("No se puede resolver \${$param->getName()} de $real");
                }
            }
            return $this->instancias[$real] = $reflexion->newInstanceArgs($argumentos);
        } finally {
            array_pop($this->enConstruccion);
        }
    }
}
```

```

}

$c = new Contenedor();
$c->enlazar(Almacen::class, AlmacenMemoria::class);
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $pedido) {
    $orden = [];
    try {
        $a = $c->get($pedido);
        echo "$pedido: " . (implode(' ', $orden) ?: 'ya existía') . ($a === $c->get($pedido) ? ' (misma instancia)' : '') . "\n";
    } catch (Throwable $e) {
        echo "$pedido: " . get_class($e) . ': ' . $e->getMessage() . "\n";
    }
}

```

`ReflectionClass` permite leer el constructor de una clase en tiempo de ejecución: `getParameters()` da cada parámetro y `getType()` su tipo. Si el tipo es una clase, el contenedor la pide recursivamente con `get`; si es un tipo simple con valor por defecto, usa el valor. Así se construye todo el árbol de dependencias sin escribir un solo `new`.

Las instancias ya creadas se guardan en un array y se devuelven directamente (singleton), y las interfaces se traducen a su clase con los enlaces. Para detectar ciclos se lleva una pila de las clases que se están construyendo: si una vuelve a aparecer, hay un ciclo, y el trozo de pila desde su primera aparición es justo el mensaje.

El `finally` saca la clase de la pila tanto si se construye bien como si salta una excepción; si no, un error dejaría la pila sucia para las peticiones siguientes.

26. Encuentra el fallo: el listado de un solo producto

Fácil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Solución (php)

```

<?php
$pdo = new PDO("sqlite::memory:");
$pdo->exec("CREATE TABLE productos (nombre TEXT, precio INTEGER)");
$pdo->exec("INSERT INTO productos VALUES ('Teclado', 25), ('Ratón', 12), ('Monitor', 150)");
$st = $pdo->query("SELECT nombre, precio FROM productos ORDER BY precio");
while ($fila = $st->fetch(PDO::FETCH_ASSOC)) {
    echo "{$fila['nombre']}: {$fila['precio']} €\n";
}

```

`fetch` devuelve una sola fila cada vez que se llama (y `false` cuando ya no quedan). Con `if` se llama una vez y solo se muestra la primera; con `while` se repite hasta que se acaban las filas.

Si se quieren todas de golpe, `fetchAll()` devuelve un array con todas las filas para recorrerlo con `foreach`.

27. Buscar sin inyección SQL

Medio · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Solución (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec("CREATE TABLE productos (id INTEGER PRIMARY KEY, nombre TEXT, categoria TEXT, precio REAL)");
$pdo->exec("INSERT INTO productos (nombre, categoria, precio) VALUES
    ('Teclado', 'perifericos', 24.99), ('Ratón', 'perifericos', 12.5),
    ('Monitor 27', 'pantallas', 189), ('Portátil', 'equipos', 749),
    ('Alfombrilla', 'perifericos', 6.9), ('Monitor 24', 'pantallas', 129)");

$categoria = trim((string) fgets(STDIN));

$consulta = $pdo->prepare('SELECT nombre, precio FROM productos WHERE categoria = ? ORDER BY precio');
$consulta->execute([$categoria]);
$filas = $consulta->fetchAll();

if (!$filas) {
    echo "Sin resultados\n";
}
foreach ($filas as $f) {
    echo $f['nombre'], ' - ', number_format($f['precio'], 2, ',', '.'), " €\n";
}
```

Al pegar el texto del usuario en la consulta, sus comillas forman parte del SQL: con `' OR '1'='1` la condición pasa a ser siempre verdadera y salen todos los productos. Una consulta preparada envía la sentencia con un `?` y, aparte, el dato; la base de datos lo trata siempre como un valor, nunca como código.

Por eso la búsqueda maliciosa simplemente busca una categoría que se llama literalmente «' OR '1'='1» y no encuentra nada. `prepare + execute(...)` es la forma de pasar cualquier dato que venga del usuario.

28. Pagar resultados con LIMIT

Medio · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Solución (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec("CREATE TABLE alumnos (id INTEGER PRIMARY KEY, nombre TEXT)");
$insertar = $pdo->prepare('INSERT INTO alumnos (nombre) VALUES (?)');
for ($i = 1; $i <= 23; $i++) {
    $insertar->execute(["Alumno $i"]);
}

[$pagina, $tamano] = array_pad(preg_split('/\s+/', trim((string) fgets(STDIN))), 2, '');
$pagina = filter_var($pagina, FILTER_VALIDATE_INT, ['options' => ['min_range' => 1]]);
$tamano = filter_var($tamano, FILTER_VALIDATE_INT, ['options' => ['min_range' => 1]]);
$total = (int) $pdo->query('SELECT COUNT(*) FROM alumnos')->fetchColumn();
$paginas = $tamano ? (int) ceil($total / $tamano) : 0;

if ($pagina === false || $tamano === false || $pagina > $paginas) {
    echo "Página fuera de rango\n";
    exit;
}
$consulta = $pdo->prepare('SELECT nombre FROM alumnos ORDER BY id LIMIT ? OFFSET ?');
$consulta->bindValue(1, $tamano, PDO::PARAM_INT);
$consulta->bindValue(2, ($pagina - 1) * $tamano, PDO::PARAM_INT);
$consulta->execute();
foreach ($consulta->fetchAll(PDO::FETCH_COLUMN) as $nombre) {
    echo $nombre, "\n";
}
echo "Página $pagina de $paginas\n";
```

`LIMIT n OFFSET m` pide a la base de datos solo `n` filas a partir de la `m`: la página `p` empieza en `(p - 1) * tamaño`. Así nunca se traen a PHP las 10.000 filas de una tabla grande para mostrar 10.

El número de páginas es el total dividido entre el tamaño, redondeando hacia arriba con `ceil`. Los dos números se validan como enteros positivos antes de usarlos, y se pasan con `bindValue(..., PDO::PARAM_INT)` para que lleguen a la consulta como números y no como texto.

29. Filtros dinámicos sin inyección

Medio · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Solución (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC]);
$pdo->exec("CREATE TABLE productos (id INTEGER PRIMARY KEY, nombre TEXT, categoria TEXT, precio REAL)");
$pdo->exec("INSERT INTO productos (nombre, categoria, precio) VALUES
    ('Teclado mecánico', 'perifericos', 59.9), ('Ratón', 'perifericos', 12.5), ('Monitor 27',
    'pantallas', 189),
    ('Monitor 24', 'pantallas', 129), ('Cable USB_C', 'cables', 9.95), ('Cable USBxC', 'cables', 7.5),
    ('Pack descuento 100%', 'ofertas', 99), ('Pack 1000', 'ofertas', 45), ('Alfombrilla', 'perifericos',
    6.9)");

$filtros = [];
foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$clave, $valor] = explode('=', $linea, 2) + [1 => ''];
    $filtros[$clave] = $valor;
}

// Cada filtro añade una condición con su marcador y su valor
$condiciones = [];
$valores = [];
if (isset($filtros['categoria'])) {
    $condiciones[] = 'categoria = ?';
    $valores[] = $filtros['categoria'];
}
if (isset($filtros['min'])) {
    $condiciones[] = 'precio >= ?';
    $valores[] = (float) $filtros['min'];
}
if (isset($filtros['max'])) {
    $condiciones[] = 'precio <= ?';
    $valores[] = (float) $filtros['max'];
}
if (isset($filtros['texto'])) {
    $condiciones[] = "nombre LIKE ? ESCAPE '\\\"";
    $valores[] = '%' . addslashes($filtros['texto'], '%_\\') . '%'; // % y _ del usuario, literales
}

// El orden no puede ir en un marcador: se elige de una lista cerrada
$ordenes = ['precio' => 'precio', 'precio_desc' => 'precio DESC', 'nombre' => 'nombre'];
$orden = $ordenes[$filtros['orden']] ?? '' ?? 'nombre';

$sql = 'SELECT nombre, precio FROM productos'
    . ($condiciones ? ' WHERE ' . implode(' AND ', $condiciones) : '')
    . " ORDER BY $orden";
$consulta = $pdo->prepare($sql);
$consulta->execute($valores);
$filas = $consulta->fetchAll();

echo $filas ? '' : "Sin resultados\n";
foreach ($filas as $f) {
    echo $f['nombre'], ' - ', number_format($f['precio'], 2, ',', '.'), " €\n";
}
```

La consulta se construye a trozos, pero los datos nunca se pegan en ella: cada filtro añade una condición con un `?` y, en paralelo, su valor a un array que se pasa a `execute`. Así se puede tener una consulta con cero, uno o cuatro filtros y seguir siendo inmune a la inyección.

Lo que no puede ir en un marcador, como el nombre de la columna del `ORDER BY`, se elige de una lista blanca: el usuario solo escoge una clave, y el texto que llega a la consulta lo escribe el programa. Cualquier otro valor se ignora.

En `LIKE`, `%` y `_` son comodines. Para buscarlos literalmente se escapan con una barra (`addslashes`) y se indica con `ESCAPE` cuál es el carácter de escape; así «USB_» no encuentra «USBxC».

30. Transferencias con transacciones

Difícil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Solución (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec("CREATE TABLE cuentas (id TEXT PRIMARY KEY, titular TEXT, saldo REAL)");
$pdo->exec("INSERT INTO cuentas VALUES ('ES01', 'Ana', 500), ('ES02', 'Luis', 120), ('ES03', 'Marta', 0)");

$saldoDe = $pdo->prepare('SELECT saldo FROM cuentas WHERE id = ?');
$mover = $pdo->prepare('UPDATE cuentas SET saldo = saldo + ? WHERE id = ?');

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$origen, $destino, $importe] = array_pad(preg_split('/\s+/', trim($linea)), 3, '');

    $saldoDe->execute([$origen]);
    $saldoOrigen = $saldoDe->fetchColumn();
    $saldoDe->execute([$destino]);
    $existeDestino = $saldoDe->fetchColumn() !== false;
    if ($saldoOrigen === false || !$existeDestino) {
        echo "Cuenta inexistente\n";
        continue;
    }
    if (!is_numeric($importe) || (float) $importe <= 0) {
        echo "Importe no válido\n";
        continue;
    }
    $importe = (float) $importe;
    if ($saldoOrigen < $importe) {
        echo "Saldo insuficiente\n";
        continue;
    }

    $pdo->beginTransaction();
    try {
        $mover->execute([- $importe, $origen]);
        $mover->execute([$importe, $destino]);
        $pdo->commit(); // las dos o ninguna
        echo "Transferencia realizada\n";
    } catch (PDOException $e) {
        $pdo->rollBack();
        echo "Error: la transferencia se ha deshecho\n";
    }
}

foreach ($pdo->query('SELECT * FROM cuentas ORDER BY id') as $c) {
    echo "{$c['id']} {$c['titular']}: ", number_format($c['saldo'], 2, ',', '.'), " €\n";
}
```

Una transferencia son dos actualizaciones que tienen que ocurrir juntas: si el programa fallara entre la primera y la segunda, el dinero desaparecería. `beginTransaction` agrupa las dos, `commit` las confirma a la vez y, si algo lanza una excepción, `rollBack` deshace lo que se hubiera hecho.

Las comprobaciones van antes y en el orden pedido. `fetchColumn` devuelve `false` si no hay ninguna fila, lo que sirve para saber si una cuenta existe; las sentencias se preparan una sola vez fuera del bucle y se ejecutan con datos distintos en cada vuelta. El saldo se modifica con `saldo = saldo + ?`, en la propia base de datos.

31. Un informe sin el problema N+1

Difícil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Solución (php)

```
<?php
final class SentenciaContada extends PDOStatement
{
    public static int $ejecutadas = 0;
    protected function __construct() {}
    public function execute(?array $params = null): bool
    {
        self::$ejecutadas++;
        return parent::execute($params);
    }
}

$pdo = new PDO('sqlite::memory:', null, null, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC]);
$pdo->exec("CREATE TABLE clientes (id INTEGER PRIMARY KEY, nombre TEXT)");
$pdo->exec("CREATE TABLE pedidos (id INTEGER PRIMARY KEY, cliente_id INTEGER, fecha TEXT, total REAL)");
$pdo->exec("INSERT INTO clientes (nombre) VALUES ('Ana'), ('Luis'), ('Marta'), ('Pablo'), ('Eva')");
$pdo->exec("INSERT INTO pedidos (cliente_id, fecha, total) VALUES
(1, '2025-03-10', 120), (1, '2025-11-02', 80.5), (1, '2026-01-15', 45),
(2, '2025-06-01', 200.5), (3, '2026-02-20', 99.99), (3, '2026-05-05', 10),
(4, '2025-12-31', 0.01), (5, '2025-01-01', 200.5)");
$pdo->setAttribute(PDO::ATTR_STATEMENT_CLASS, [SentenciaContada::class]);

$anio = trim((string) fgets(STDIN));
$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';

// Una sola consulta: el filtro del año va en el ON para no perder a los clientes sin pedidos
$consulta = $pdo->prepare("
    SELECT c.nombre, COUNT(p.id) AS pedidos, COALESCE(SUM(p.total), 0) AS gasto
    FROM clientes c
    LEFT JOIN pedidos p ON p.cliente_id = c.id AND strftime('%Y', p.fecha) = ?
    GROUP BY c.id, c.nombre
    ORDER BY gasto DESC, c.nombre");
$consulta->execute([$anio]);
$informe = $consulta->fetchAll();

foreach ($informe as $fila) {
    echo "{$fila['nombre']}: {$fila['pedidos']} pedidos, {$euros($fila['gasto'])}\n";
}
echo 'Consultas: ', SentenciaContada::$ejecutadas, "\n";
```

El problema N+1 aparece al hacer una consulta para la lista y otra por cada elemento. Se resuelve dejando que la base de datos haga el trabajo en una sola consulta: `LEFT JOIN` une cada cliente con sus pedidos, `GROUP BY` los agrupa por cliente, y `COUNT` y `SUM` calculan cada fila del informe. El orden también se hace en SQL.

El detalle importante es dónde va el filtro del año. Si estuviera en el `WHERE`, eliminaría las filas de los clientes sin pedidos ese año (en ellas `p.fecha` es `NULL`) y esos clientes desaparecerían. En la condición del `ON` solo decide qué pedidos se unen, y el `LEFT JOIN` conserva a todos los clientes.

`COUNT(p.id)` cuenta solo los pedidos que existen (no cuenta los `NULL`), y `COALESCE(SUM(...), 0)` convierte en 0 la suma vacía, que sería `NULL`.

32. Importar un CSV: todo o nada

Muy difícil · Acceso a Datos con PDO · apuntesdam.com/subject/dwes/topic/acceso-datos-php

Solución (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC]);
$pdo->exec("CREATE TABLE productos (sku TEXT PRIMARY KEY, nombre TEXT NOT NULL, precio REAL NOT NULL,
stock INTEGER NOT NULL)");
$pdo->exec("INSERT INTO productos VALUES ('TEC-001', 'Teclado', 24.99, 5), ('RAT-001', 'Ratón', 12.5,
10)");

$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
array_shift($lineas); // cabecera

// Devuelve el motivo del error de una fila, o null si es válida
function validar(array $c, array $vistos): ?string
{
    if (count($c) !== 4) return 'número de campos incorrecto';
    [$sku, $nombre, $precio, $stock] = $c;
    if (!preg_match('/^[A-Z]{3}-\d{3}$/', $sku)) return 'SKU no válido';
    if (trim($nombre) === '') return 'nombre vacío';
    if (!is_numeric($precio) || (float) $precio < 0) return 'precio no válido';
    if (filter_var($stock, FILTER_VALIDATE_INT, ['options' => ['min_range' => 0]]) === false) return
'stock no válido';
    if (isset($vistos[$sku])) return 'SKU repetido';
    return null;
}

$existe = $pdo->prepare('SELECT 1 FROM productos WHERE sku = ?');
$actualizar = $pdo->prepare('UPDATE productos SET nombre = ?, precio = ?, stock = ? WHERE sku = ?');
$insertar = $pdo->prepare('INSERT INTO productos (nombre, precio, stock, sku) VALUES (?, ?, ?, ?)');
$nuevos = $actualizados = 0;
$vistos = [];
$error = null;

$pdo->beginTransaction();
foreach ($lineas as $i => $linea) {
    $c = str_getcsv($linea, ',', '"', '');
    $error = validar($c, $vistos);
    if ($error !== null) {
        echo 'Línea ', $i + 2, ": $error\n"; // la cabecera es la línea 1
        break;
    }
    [$sku, $nombre, $precio, $stock] = $c;
    $vistos[$sku] = true;
    $existe->execute([$sku]);
    $datos = [trim($nombre), (float) $precio, (int) $stock, $sku];
    if ($existe->fetchColumn()) {
        $actualizar->execute($datos);
        $actualizados++;
    } else {
        $insertar->execute($datos);
        $nuevos++;
    }
}
if ($error !== null) {
    $pdo->rollBack(); // nada de lo importado se queda
    echo "Importación cancelada\n";
} else {
    $pdo->commit();
    echo "Nuevos: $nuevos, actualizados: $actualizados\n";
}

foreach ($pdo->query('SELECT * FROM productos ORDER BY sku') as $p) {
    echo "{$p['sku']} {$p['nombre']} {$p['precio']} {$p['stock']}\n";
}
```

«Todo o nada» es exactamente lo que da una transacción: se abre antes de la primera fila, cada fila válida se guarda dentro de ella, y en cuanto una falla se hace `rollback`, que deshace también las filas anteriores. Solo si todas pasan se confirma con `commit`.

`str_getcsv` interpreta el CSV de verdad: respeta las comillas (una coma dentro de «Cable, 2 m» no separa campos) y convierte `""` en una comilla. La validación está en su propia función, que devuelve el primer motivo de error o `null`, y el mismo array de datos sirve para el UPDATE y el INSERT porque los marcadores van en el mismo orden.

33. Resumir la respuesta de una API

Medio · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Solución (php)

```
<?php
$texto = stream_get_contents(STDIN);

$repos = json_decode($texto, true);
if (!is_array($repos)) {
    echo "JSON no válido\n";
    exit;
}
$miles = fn(int $n) => number_format($n, 0, ',', '.');

usort($repos, fn($a, $b) => $b['stars'] <=> $a['stars']);
foreach (array_slice($repos, 0, 3) as $i => $r) {
    echo $i + 1, ". {$r['name']} (★ ", $miles($r['stars']), ")\n";
}
echo 'Total de estrellas: ', $miles(array_sum(array_column($repos, 'stars'))), "\n";

$lenguajes = array_unique(array_filter(array_column($repos, 'language'), fn($l) => $l !== null));
sort($lenguajes, SORT_STRING);
echo 'Lenguajes: ', implode(' ', $lenguajes), "\n";
```

`json_decode($texto, true)` convierte el JSON en arrays de PHP y devuelve `null` si el texto no es JSON válido, que es como se detecta el error. A partir de ahí, las funciones de arrays hacen casi todo: `usort` ordena por estrellas, `array_slice` toma los tres primeros y `array_column` extrae una columna de todos los objetos.

Los lenguajes se obtienen encadenando funciones: la columna `language`, sin los nulos (`array_filter`), sin repetidos (`array_unique`) y ordenada. `number_format` con 0 decimales pone el punto de miles.

34. Caché HTTP con ETag

Medio · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Solución (php)

```
<?php
$productos = [
    7 => ['id' => 7, 'nombre' => 'Teclado', 'precio' => 24.99],
    8 => ['id' => 8, 'nombre' => 'Ratón', 'precio' => 12.5],
];

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$id, $recibida] = explode(' ', $linea, 2);
    $producto = $productos[(int) $id] ?? null;
    if ($producto === null) {
        echo "404\n";
        continue;
    }
    $json = json_encode($producto, JSON_UNESCAPED_UNICODE);
    $etag = '"' . md5($json) . '"'; // huella del contenido
    if (trim($recibida) === $etag) {
        echo "304\n"; // el cliente ya lo tiene: sin cuerpo
    } else {
        echo "200 ETag: $etag $json\n";
    }
}
```

La ETag es una huella del contenido: el md5 del JSON que se enviaría. Si el producto no cambia, la huella tampoco, así que basta con compararla con la que manda el cliente en If-None-Match: si coincide, se responde 304 sin cuerpo y el cliente usa lo que ya tiene.

La huella se calcula sobre exactamente los mismos bytes que se envían (con `JSON_UNESCAPED_UNICODE`), porque cualquier diferencia en la representación cambiaría el md5 aunque los datos fueran los mismos.

35. Una API REST de libros

Difícil · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Solución (php)

```
<?php
$libros = [
    1 => ['id' => 1, 'titulo' => 'El Quijote', 'autor' => 'Cervantes'],
    2 => ['id' => 2, 'titulo' => 'La Regenta', 'autor' => 'Clarín'],
];

function responder(int $codigo, mixed $datos = null): void
{
    echo $codigo, $datos === null ? ' ' : ' ' . json_encode($datos, JSON_UNESCAPED_UNICODE), "\n";
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$metodo, $ruta, $cuerpo] = array_pad(explode(' ', $linea, 3), 3, '');
    if (!preg_match('#^/api/libros(?:/(\d+))?$#', $ruta, $m)) {
        responder(404, ['error' => 'Ruta no encontrada']);
        continue;
    }
    $id = isset($m[1]) ? (int) $m[1] : null;

    if ($id === null) {
        // la colección
        if ($metodo === 'GET') {
            responder(200, array_values($libros));
        } elseif ($metodo === 'POST') {
            $datos = json_decode($cuerpo, true);
            if (!is_array($datos) || empty($datos['titulo']) || empty($datos['autor'])) {
                responder(422, ['error' => 'Faltan campos']);
                continue;
            }
            $nuevoId = $libros ? max(array_keys($libros)) + 1 : 1;
            $libros[$nuevoId] = ['id' => $nuevoId, 'titulo' => $datos['titulo'], 'autor' =>
                $datos['autor']];
            responder(201, $libros[$nuevoId]);
        } else {
            responder(405, ['error' => 'Método no permitido']);
        }
        continue;
    }

    if ($metodo !== 'GET' && $metodo !== 'DELETE') {
        // un libro
        responder(405, ['error' => 'Método no permitido']);
    } elseif (!isset($libros[$id])) {
        responder(404, ['error' => 'Libro no encontrado']);
    } elseif ($metodo === 'GET') {
        responder(200, $libros[$id]);
    } else {
        unset($libros[$id]);
        responder(204);
    }
}
```

Una sola expresión regular reconoce las dos rutas: `/api/libros` y `/api/libros/{id}`, con el id en un grupo opcional. Si no encaja, la ruta no existe (404); si encaja, el método decide la operación, y uno que no tiene sentido para esa ruta da 405.

La lista se devuelve con `array_values` para que `json_encode` produzca un array JSON (`[...]`) y no un objeto con claves numéricas. El nuevo id es el mayor existente más uno, y un 204 se responde sin cuerpo.

36. Tokens JWT firmados a mano

Difícil · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Solución (php)

```
<?php
const CLAVE = 'clave-super-secreta-de-32-bytes!';

function b64url(string $s): string
{
    return rtrim(strtr(base64_encode($s), '+/', '-_'), '=');
}

function firmar(string $cabecera, string $payload): string
{
    return b64url(hash_hmac('sha256', "$cabecera.$payload", CLAVE, true));
}

function crear(array $datos): string
{
    $cabecera = b64url(json_encode(['alg' => 'HS256', 'typ' => 'JWT']));
    $payload = b64url(json_encode($datos));
    return "$cabecera.$payload." . firmar($cabecera, $payload);
}

function verificar(string $token, int $ahora): ?array
{
    $partes = explode('.', $token);
    if (count($partes) !== 3) {
        return null;
    }
    [$cabecera, $payload, $firma] = $partes;
    if (!hash_equals(firmar($cabecera, $payload), $firma)) {
        return null; // manipulado o firmado con otra clave
    }
    $datos = json_decode(base64_decode(strtr($payload, '-_', '+/')), true);
    if (!is_array($datos) || ($datos['exp'] ?? 0) <= $ahora) {
        return null; // caducado
    }
    return $datos;
}

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    $p = explode(' ', $linea);
    if ($p[0] === 'crear') {
        echo crear(['sub' => $p[1], 'rol' => $p[2], 'exp' => (int) $p[3]]), "\n";
    } else {
        $d = verificar($p[1], (int) $p[2]);
        echo $d ? "{$d['sub']} ({$d['rol']})\n" : "Token no válido\n";
    }
}
```

Un JWT no cifra los datos (cualquiera puede decodificar el payload), pero los firma. La firma es un HMAC-SHA256 de «cabecera.payload» con la clave del servidor, en binario (`true` en `hash_hmac`) y codificado en base64url. Se extrae a la función `firmar` porque crear y verificar la calculan igual.

Al verificar, el orden es importante: primero la firma y solo después se leen los datos para mirar la caducidad, porque hasta que la firma no se comprueba no hay motivo para creer lo que dicen. La firma se compara con `hash_equals`, en tiempo constante.

37. Limitar las peticiones: 429 Too Many Requests

Muy difícil · Servicios Web y APIs REST · apuntesdam.com/subject/dwes/topic/servicios-web

Solución (php)

```
<?php
const LIMITE = 3;
const VENTANA = 10;
$historial = [];
$admitidas = [];

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$ahora, $cliente] = explode(' ', $linea);
    $ahora = (int) $ahora;
    // Peticiones admitidas en los últimos VENTANA segundos
    $ventana = array_values(array_filter($historial[$cliente] ?? [], fn($t) => $t > $ahora - VENTANA));
    if (count($ventana) < LIMITE) {
        $ventana[] = $ahora;
        $admitidas[$cliente] = ($admitidas[$cliente] ?? 0) + 1;
        echo "$ahora $cliente: 200 (quedan ", LIMITE - count($ventana), ")\n";
    } else {
        $espera = $ventana[0] + VENTANA - $ahora; // cuando salga la más antigua
        echo "$ahora $cliente: 429 Retry-After: $espera\n";
    }
    $historial[$cliente] = $ventana;
}

ksort($admitidas);
foreach ($admitidas as $cliente => $n) {
    echo "$cliente: $n admitidas\n";
}
```

La ventana deslizante guarda, por cliente, los instantes de sus peticiones admitidas. En cada petición se descartan las que ya tienen más de 10 segundos y se cuentan las que quedan: si son menos de 3, se admite y se apunta; si no, se rechaza con 429.

Retry-After dice cuánto falta para que la petición más antigua de la ventana caduque, que es cuando habrá sitio: su instante más 10 menos el actual. Las rechazadas no se apuntan, para que un cliente que insiste no se bloquee a sí mismo para siempre.

38. El endpoint de un autocompletado

Medio · Aplicaciones Dinámicas e Híbridas · apuntesdam.com/subject/dwes/topic/aplicaciones-hibridas

Solución (php)

```
<?php
$municipios = [
    ['Arnedo', 'La Rioja'], ['Logroño', 'La Rioja'], ['Calahorra', 'La Rioja'], ['Haro', 'La Rioja'],
    ['Alfaro', 'La Rioja'], ['Ávila', 'Ávila'], ['Almería', 'Almería'], ['Alicante', 'Alicante'],
    ['Albacete', 'Albacete'], ['Alcalá de Henares', 'Madrid'], ['Madrid', 'Madrid'], ['Málaga',
    'Málaga'],
    ['Salamanca', 'Salamanca'], ['Cádiz', 'Cádiz'], ['Jaén', 'Jaén'], ['León', 'León'], ['Lleida',
    'Lleida'],
    ['Logrosán', 'Cáceres'], ['Santander', 'Cantabria'], ['Zaragoza', 'Zaragoza'], ['Alcobendas',
    'Madrid'],
    ['Aranjuez', 'Madrid'],
];

// Minúsculas y sin tildes, para comparar como lo haría una persona (la ñ, como n: así ordena bien)
$normalizar = fn(string $s) => strtolower(trim($s), ['á' => 'a', 'é' => 'e', 'í' => 'i', 'ó' =>
'o', 'ú' => 'u', 'ü' => 'u', 'ñ' => 'n']);

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    parse_str($linea, $q);
    $texto = $normalizar($q['q'] ?? '');
    $sugerencias = [];
    if (mb_strlen($texto) >= 2) {
        $empiezan = $contienen = [];
        foreach ($municipios as [$nombre, $provincia]) {
            $pos = mb_strpos($normalizar($nombre), $texto);
            if ($pos === 0) {
                $empiezan[] = ['nombre' => $nombre, 'provincia' => $provincia];
            } elseif ($pos !== false) {
                $contienen[] = ['nombre' => $nombre, 'provincia' => $provincia];
            }
        }
        $alfabetico = fn($a, $b) => strcmp($normalizar($a['nombre']), $normalizar($b['nombre']));
        usort($empiezan, $alfabetico);
        usort($contienen, $alfabetico);
        $sugerencias = array_slice(array_merge($empiezan, $contienen), 0, 5);
    }
    echo json_encode($sugerencias, JSON_UNESCAPED_UNICODE), "\n";
}
```

Para que «LOGRO» encuentre «Logroño» y «ávila» encuentre «Ávila», el texto buscado y cada municipio se normalizan igual: sin espacios a los lados, en minúsculas con `mb_strtolower` (que entiende UTF-8) y sin tildes con `strtr`. La comparación se hace siempre entre versiones normalizadas, pero se devuelve el nombre original.

La ñ también se convierte en n al normalizar: `strcmp` compara bytes, y en UTF-8 la ñ va detrás de todas las letras sin tilde, así que «Logroño» saldría después de «Logrosán».

`mb_strpos` dice dónde aparece el texto: en la posición 0, el municipio empieza por él; en otra, lo contiene; y `false` si no está. Cada grupo se ordena por separado y se unen con los que empiezan primero, porque son las sugerencias más útiles. El límite de 5 y el mínimo de 2 caracteres evitan respuestas enormes con cada pulsación.

39. Verificar la firma de un webhook

Difícil · Aplicaciones Dinámicas e Híbridas · apuntesdam.com/subject/dwes/topic/aplicaciones-hibridas

Solución (php)

```
<?php
const SECRETO = 'whsec_9f86d081884c7d659a2feaa0c55ad015';

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$ahora, $cabecera, $cuerpo] = explode('|', $linea, 3);

    // Cabecera «t=...,v1=...,v1=...»
    $t = null;
    $firmas = [];
    foreach (explode(' ', $cabecera) as $parte) {
        [$clave, $valor] = array_pad(explode('=', $parte, 2), 2, '');
        if ($clave === 't') {
            $t = $valor;
        } elseif ($clave === 'v1') {
            $firmas[] = $valor;
        }
    }
    if ($t === null || !ctype_digit($t) || !$firmas) {
        echo "Rechazado: cabecera\n";
        continue;
    }
    if ((int) $ahora - (int) $t > 300) {
        echo "Rechazado: caducado\n";
        continue;
    }
    $esperada = hash_hmac('sha256', "$t.$cuerpo", SECRETO);
    $valida = false;
    foreach ($firmas as $firma) {
        if (hash_equals($esperada, $firma)) {
            $valida = true;
        }
    }
    echo $valida ? "Aceptado: " . (json_decode($cuerpo, true)['tipo'] ?? '?') . "\n" : "Rechazado: firma\n";
}
```

La firma se calcula sobre el instante y el cuerpo juntos («t.cuerpo»): así no se puede cambiar ni el contenido del aviso ni su fecha sin invalidarla. El servidor la recalcula con su secreto y la compara con cada `v1` recibido con `hash_equals`; basta con que coincida una, para poder cambiar de secreto sin cortar el servicio.

Comprobar la antigüedad evita los ataques de repetición: alguien que capture un aviso válido no puede reenviarlo días después. Como `t` está dentro de lo firmado, tampoco puede cambiar la fecha para que parezca nuevo.

40. Un cortacircuitos para un servicio externo

Muy difícil · Aplicaciones Dinámicas e Híbridas · apuntesdam.com/subject/dwes/topic/aplicaciones-hibridas

Solución (php)

```
<?php
const UMBRAL = 3;
const ESPERA = 30;
$estado = 'cerrado';
$fallos = 0;
$abiertoDesde = 0;
$llamadas = 0;

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$t, $resultado] = explode(' ', $linea);
    $t = (int) $t;
    if ($estado === 'abierto' && $t - $abiertoDesde < ESPERA) {
        echo "$t: rechazada (abierto)\n";          // ni siquiera se llama al servicio
        continue;
    }
    $llamadas++;
    $ok = $resultado === 'ok';
    if ($estado === 'abierto') {                    // ya pasó la espera: esta llamada es la prueba
        if ($ok) {
            $estado = 'cerrado';
            $fallos = 0;
            echo "$t: prueba ok → cerrado\n";
        } else {
            $abiertoDesde = $t;
            echo "$t: prueba fallo → abierto\n";
        }
        continue;
    }
    if ($ok) {
        $fallos = 0;
        echo "$t: ok (cerrado)\n";
    } elseif (++$fallos >= UMBRAL) {
        $estado = 'abierto';
        $abiertoDesde = $t;
        $fallos = 0;
        echo "$t: fallo → abierto\n";
    } else {
        echo "$t: fallo (cerrado, $fallos fallos)\n";
    }
}
echo "Llamadas reales: $llamadas\n";
```

El cortacircuitos es una máquina de estados. Abierto, rechaza al momento las peticiones hasta que pasan 30 segundos; la primera después de esa espera se deja pasar como prueba, y su resultado decide si se cierra o se vuelve a abrir.

Cerrado, cuenta los fallos seguidos (un éxito la reinicia) y se abre al llegar a 3. Lo importante es que las peticiones rechazadas no llegan a llamar al servicio: la tienda responde rápido aunque la API esté caída, y la API no recibe más carga mientras intenta recuperarse.

41. Informe a partir de un CSV

Medio · Ejercicios de Desarrollo Web en Entorno Servidor · apuntesdam.com/subject/dwes/topic/ejercicios-dwes

Solución (php)

```
<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);

array_shift($lineas); // cabecera
$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';
$ventas = [];
$descartadas = 0;
foreach ($lineas as $linea) {
    $campos = str_getcsv($linea, ',', '"', '');
    if (count($campos) !== 3 || !is_numeric($campos[2])) {
        $descartadas++;
        continue;
    }
    $comercial = $campos[1];
    $ventas[$comercial] = ($ventas[$comercial] ?? 0) + (float) $campos[2];
}
arsort($ventas); // de mayor a menor importe
foreach ($ventas as $comercial => $importe) {
    echo "$comercial: ", $euros($importe), "\n";
}
echo 'Total: ', $euros(array_sum($ventas)), "\n";
if ($descartadas > 0) {
    echo "Filas descartadas: $descartadas\n";
}
```

`str_getcsv` interpreta cada línea como CSV, así que «"Pérez, Juan"» llega como un solo campo aunque tenga una coma dentro; con `explode(',', '')` se partiría en dos. Las filas con un número de campos distinto o un importe no numérico se cuentan y se saltan.

Las ventas se acumulan en un array asociativo comercial → importe y se ordenan con `arsort`, que ordena por valor de mayor a menor conservando las claves. `number_format` redondea al mostrar, por eso 0,1 + 0,2 sale como 0,30 aunque en coma flotante sea 0,300000000000000004.

42. API de tareas con base de datos

Difícil · Ejercicios de Desarrollo Web en Entorno Servidor · apuntesdam.com/subject/dwes/topic/ejercicios-dwes

Solución (php)

```
<?php
$pdo = new PDO('sqlite::memory:', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
]);
$pdo->exec('CREATE TABLE tareas (id INTEGER PRIMARY KEY, titulo TEXT NOT NULL, hecha INTEGER NOT NULL
    DEFAULT 0)');

function responder(int $codigo, mixed $datos = null): void
{
    echo $codigo, $datos === null ? ' ' : ' ' . json_encode($datos, JSON_UNESCAPED_UNICODE), "\n";
}

// Fila de la base de datos → JSON (hecha como booleano)
$aJson = fn(array $f) => ['id' => (int) $f['id'], 'titulo' => $f['titulo'], 'hecha' => (bool)
    $f['hecha']];
$buscar = $pdo->prepare('SELECT * FROM tareas WHERE id = ?');

foreach (file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES) as $linea) {
    [$metodo, $ruta, $cuerpo] = array_pad(explode(' ', $linea, 3), 3, '');
    $camino = (string) parse_url($ruta, PHP_URL_PATH);
    parse_str((string) parse_url($ruta, PHP_URL_QUERY), $query);
    $datos = json_decode($cuerpo, true) ?? [];

    if ($camino === '/tareas') {
        if ($metodo === 'GET') {
            if (isset($query['hecha'])) {
                $consulta = $pdo->prepare('SELECT * FROM tareas WHERE hecha = ? ORDER BY id');
                $consulta->execute([$query['hecha'] ? 1 : 0]);
            } else {
                $consulta = $pdo->prepare('SELECT * FROM tareas ORDER BY id');
                $consulta->execute();
            }
            responder(200, array_map($aJson, $consulta->fetchAll()));
        } elseif ($metodo === 'POST') {
            $titulo = trim((string) ($datos['titulo'] ?? ''));
            if ($titulo === '') {
                responder(422, ['error' => 'El título es obligatorio']);
                continue;
            }
            $pdo->prepare('INSERT INTO tareas (titulo) VALUES (?)')->execute([$titulo]);
            responder(201, ['id' => (int) $pdo->lastInsertId(), 'titulo' => $titulo, 'hecha' => false]);
        } else {
            responder(405, ['error' => 'Método no permitido']);
        }
        continue;
    }

    if (preg_match('#^/tareas/(\d+)$#', $camino, $m)) {
        $buscar->execute([$m[1]]);
        $tarea = $buscar->fetch();
        if (!$tarea) {
            responder(404, ['error' => 'Tarea no encontrada']);
        } elseif ($metodo === 'GET') {
            responder(200, $aJson($tarea));
        } elseif ($metodo === 'PATCH') {
            if (array_key_exists('hecha', $datos)) {
                $pdo->prepare('UPDATE tareas SET hecha = ? WHERE id = ?')->execute([$datos['hecha'] ? 1 :
                    0, $m[1]]);
            }
            if (isset($datos['titulo']) && trim($datos['titulo']) !== '') {
                $pdo->prepare('UPDATE tareas SET titulo = ? WHERE id = ?')->execute([trim($datos['titulo']), $m[1]]);
            }
            $buscar->execute([$m[1]]);
            responder(200, $aJson($buscar->fetch()));
        }
    }
}
```

```

    } elseif ($metodo === 'DELETE') {
        $pdo->prepare('DELETE FROM tareas WHERE id = ?')->execute([$m[1]]);
        responder(204);
    } else {
        responder(405, ['error' => 'Método no permitido']);
    }
    continue;
}

responder(404, ['error' => 'Ruta no encontrada']);
}

```

La API tiene dos tipos de rutas: la colección (`/tareas`), donde se lista y se crea, y cada elemento (`/tareas/{id}`), donde se consulta, modifica o borra. Para los elementos, lo primero es buscar la tarea: si no existe, la respuesta es 404 sea cual sea el método.

La base de datos guarda `hecha` como 0/1 y la API la devuelve como booleano: la función `toJson` hace esa traducción en un solo sitio, y se usa para todas las respuestas. PATCH solo cambia los campos que llegan en el cuerpo y responde con la tarea leída otra vez de la base de datos, que es su estado real.

Todas las consultas que llevan datos van preparadas, incluido el filtro `?hecha=`, que se lee de la query string con `parse_url` y `parse_str`.

43. Carrito con cupones

Difícil · Ejercicios de Desarrollo Web en Entorno Servidor · apuntesdam.com/subject/dwes/topic/ejercicios-dwes

Solución (php)

```

<?php
$lineas = file('php://stdin', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);

$euros = fn(float $n) => number_format($n, 2, ',', '.') . ' €';
$subtotal = 0;
$cupones = [];
foreach ($lineas as $linea) {
    if (str_starts_with($linea, 'CUPON ')) {
        $cupones[] = trim(substr($linea, 6));
        continue;
    }
    [, $precio, $cantidad] = explode(';', $linea);
    $subtotal += (float) $precio * (int) $cantidad;
}
$envio = $subtotal < 50 ? 4.95 : 0;

// Lo que ahorraría cada cupón (0 si no existe o no cumple su condición)
$ahorro = fn(string $c) => match ($c) {
    'DESC10' => round($subtotal * 0.10, 2),
    'MENOS5' => $subtotal >= 30 ? 5.0 : 0.0,
    'ENVIAGRATIS' => $envio,
    default => 0.0,
};
$mejor = null;
foreach ($cupones as $c) {
    if ($ahorro($c) > 0 && ($mejor === null || $ahorro($c) > $ahorro($mejor))) {
        $mejor = $c;
    }
}

$descuento = 0;
if ($mejor === 'ENVIAGRATIS') {
    $envio = 0;
} elseif ($mejor !== null) {
    $descuento = $ahorro($mejor);
}
echo 'Subtotal: ', $euros($subtotal), "\n";
echo 'Descuento', $mejor !== null ? " ($mejor)" : '', ': ', $euros($descuento), "\n";
echo 'Envío: ', $euros($envio), "\n";
echo 'Total: ', $euros($subtotal - $descuento + $envio), "\n";

```

Para elegir el mejor cupón hay que poder comparar cupones muy distintos (un porcentaje, una cantidad fija y el envío). La función `$ahorro` los reduce todos a lo mismo, los euros que ahorran, y devuelve 0 si el código no existe o no cumple su condición. Después basta con quedarse con el que más ahorra.

El envío se calcula antes que los cupones porque ENVIOGRATIS ahorra justo lo que cuesta el envío (0 si ya era gratis). `match` compara con `===` y obliga a contemplar todos los casos, con `default` para los códigos desconocidos.

44. Factura de un carrito con IVA y promociones

Difícil · PHP · 70 minutos · apuntesdam.com/ejercicios/php/factura-con-promociones

Solución (php)

```
<?php
// Tipos de IVA de cada categoría
const IVA = ["libros" => 4, "alimentacion" => 10, "general" => 21];

/** Céntimos → «1.234,50» (punto de millar y coma decimal). */
function dinero(int $centimos): string
{
    return number_format($centimos / 100, 2, ",", ".");
}

$lineas = explode("\n", stream_get_contents(STDIN));
$productos = [];
foreach ($lineas as $i => $linea) {
    if (trim($linea) === "") {
        continue;
    }
    $campos = array_map("trim", explode(";", $linea));
    if (count($campos) !== 4) {
        echo "Aviso: línea " . ($i + 1) . " ignorada\n";
        continue;
    }
    [$nombre, $categoria, $precio, $unidades] = $campos;
    if (!isset(IVA[$categoria]) || !is_numeric($precio) || $precio < 0 || !ctype_digit($unidades) ||
        (int) $unidades === 0) {
        echo "Aviso: línea " . ($i + 1) . " ignorada\n";
        continue;
    }
    // Todo se calcula en céntimos enteros: así los totales cuadran siempre
    $productos[] = [
        "nombre" => $nombre,
        "categoria" => $categoria,
        "unidades" => (int) $unidades,
        "precio" => (int) round((float) $precio * 100),
    ];
}

if (count($productos) === 0) {
    echo "Carrito vacío\n";
    exit;
}

echo "Producto" . str_repeat(" ", 14) . sprintf("%5s%10s%11s", "Uds", "Precio", "Importe") . "\n";
$base = ["libros" => 0, "alimentacion" => 0, "general" => 0];
$promociones = [];
foreach ($productos as $p) {
    $importe = $p["unidades"] * $p["precio"];
    // mb_str_pad cuenta caracteres, no bytes: así los nombres con tildes también quedan alineados
    echo mb_str_pad($p["nombre"], 22) . sprintf("%5d%10s%11s", $p["unidades"], dinero($p["precio"]),
        dinero($importe)) . "\n";
    $gratis = $p["categoria"] === "alimentacion" ? intdiv($p["unidades"], 3) : 0;
    if ($gratis > 0) {
        $promociones[] = "3x2 en {$p['nombre']}: -" . dinero($gratis * $p["precio"]) . " €";
    }
    $base[$p["categoria"]] += $importe - $gratis * $p["precio"];
}

if (array_sum($base) > 10000) {
    $descuento = (int) round($base["general"] * 5 / 100);
    if ($descuento > 0) {
        $base["general"] -= $descuento;
        $promociones[] = "Descuento del 5 % en general: -" . dinero($descuento) . " €";
    }
}

if ($promociones) {
    echo "Promociones:\n";
}
```

```

    foreach ($promociones as $texto) {
        echo " $texto\n";
    }
}

$total = array_sum($base);
printf("%-16s%12s €\n", "Base imponible:", dinero($total));
foreach (IVA as $categoria => $tipo) {
    if ($base[$categoria] > 0) {
        $cuota = (int) round($base[$categoria] * $tipo / 100);
        $total += $cuota;
        printf("%-16s%12s €\n", "IVA $tipo %:", dinero($cuota));
    }
}
printf("%-16s%12s €\n", "TOTAL:", dinero($total));

```

La lectura y la validación van primero y por separado: cada línea se comprueba campo a campo y, si algo falla, se avisa y se pasa a la siguiente con `continue`. Como los avisos se escriben mientras se lee, aparecen antes que la factura sin necesidad de guardarlos.

Todos los importes son céntimos enteros. Solo hay redondeo donde aparece un porcentaje (el descuento y cada cuota de IVA), y se redondea una vez por categoría, igual que en una factura real, donde el IVA se calcula sobre la base de cada tipo y no producto a producto.

El array asociativo `$base` es el corazón del programa: acumula la base de cada categoría ya con el 3x2, el descuento modifica solo la de general y el bucle del IVA lo recorre en el orden de la constante `IVA`, que define a la vez los tipos y el orden de las líneas del resumen.

La presentación se apoya en `sprintf` para las columnas y en una única función `dinero` para el formato de los importes. `mb_str_pad` cuenta caracteres en lugar de bytes, así que los nombres con tildes también quedan alineados.

45. Préstamos de una biblioteca con PDO y transacciones

Muy difícil · PHP · 100 minutos · apuntesdam.com/ejercicios/php/biblioteca-con-pdo-y-transacciones

Solución (php)

```
<?php
// Base de datos de la biblioteca (SQLite en memoria): tablas libros, socios y prestamos
$pdo = new PDO("sqlite::memory:");
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
$pdo->exec(<<<SQL
    CREATE TABLE libros (codigo TEXT PRIMARY KEY, titulo TEXT NOT NULL, autor TEXT NOT NULL, ejemplares
        INTEGER NOT NULL);
    CREATE TABLE socios (id INTEGER PRIMARY KEY, nombre TEXT NOT NULL);
    CREATE TABLE prestamos (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        codigo TEXT NOT NULL REFERENCES libros(codigo),
        socio INTEGER NOT NULL REFERENCES socios(id),
        salida TEXT NOT NULL,
        devolucion TEXT
    );
    INSERT INTO libros VALUES
        ('L01', 'Cien años de soledad', 'Gabriel García Márquez', 2),
        ('L02', '1984', 'George Orwell', 1),
        ('L03', 'Rebelión en la granja', 'George Orwell', 2),
        ('L04', 'La sombra del viento', 'Carlos Ruiz Zafón', 1),
        ('L05', 'El nombre del viento', 'Patrick Rothfuss', 3),
        ('L06', 'El tercer policía', 'Flann O'Brien', 1);
    INSERT INTO socios VALUES (1, 'Ana Gil'), (2, 'Luis Mora'), (3, 'Eva Ruiz');
    INSERT INTO prestamos (codigo, socio, salida, devolucion) VALUES
        ('L02', 2, '2026-09-01', NULL),
        ('L01', 1, '2026-09-20', '2026-09-28'),
        ('L05', 1, '2026-09-25', NULL);
SQL);

const PLAZO = 15;           // días de préstamo
const MULTA = 20;           // céntimos por día de retraso
const MAXIMO = 3;           // préstamos activos por socio

function euros(int $c): string { return number_format($c / 100, 2, ",", ".") . " €"; }

function dias(string $desde, string $hasta): int {
    return (int) (new DateTimeImmutable($desde)->diff(new DateTimeImmutable($hasta))->format("%r%a"));
}

function vence(string $salida): string {
    return (new DateTimeImmutable($salida))->modify("+" . PLAZO . " days")->format("Y-m-d");
}

/** Ejemplares que quedan en la estantería: los totales menos los prestados sin devolver. */
function disponibles(PDO $pdo, string $codigo): int {
    $st = $pdo->prepare("SELECT l.ejemplares - COUNT(p.id) FROM libros l
        LEFT JOIN prestamos p ON p.codigo = l.codigo AND p.devolucion IS NULL
        WHERE l.codigo = ? GROUP BY l.codigo");
    $st->execute([$codigo]);
    return (int) $st->fetchColumn();
}

function buscar(PDO $pdo, string $texto): void {
    // El texto va como parámetro: un apóstrofo (O'Brien) no rompe la consulta ni permite inyectar SQL
    $st = $pdo->prepare("SELECT codigo, titulo, autor, ejemplares FROM libros
        WHERE titulo LIKE :t OR autor LIKE :t ORDER BY titulo");
    $st->execute([":t" => "%$texto%"]);
    $libros = $st->fetchAll();
    if (!$libros) {
        echo "Sin resultados para «{$texto}»\n";
        return;
    }
    foreach ($libros as $l) {
        echo "{$l['codigo']} · {$l['titulo']} ({$l['autor']}) · disponibles " . disponibles($pdo,
            $l['codigo']) . " de {$l['ejemplares']}\n";
    }
}
```

```

    }
}

function prestar(PDO $pdo, string $codigo, int $socio, string $fecha): void {
    $libro = $pdo->prepare("SELECT titulo FROM libros WHERE codigo = ?");
    $libro->execute([$codigo]);
    $titulo = $libro->fetchColumn();
    $persona = $pdo->prepare("SELECT nombre FROM socios WHERE id = ?");
    $persona->execute([$socio]);
    $nombre = $persona->fetchColumn();
    if ($titulo === false) { echo "No existe el libro $codigo\n"; return; }
    if ($nombre === false) { echo "No existe el socio $socio\n"; return; }

    // Las comprobaciones y el INSERT van en una transacción: nadie puede llevarse el último ejemplar
    // entre medias
    $pdo->beginTransaction();
    try {
        if (disponibles($pdo, $codigo) <= 0) throw new RuntimeException("No quedan ejemplares de «{$titulo}»");
        $activos = $pdo->prepare("SELECT salida FROM prestamos WHERE socio = ? AND devolucion IS NULL");
        $activos->execute([$socio]);
        $salidas = $activos->fetchAll(PDO::FETCH_COLUMN);
        foreach ($salidas as $s) {
            if (vence($s) < $fecha) throw new RuntimeException("$nombre tiene préstamos vencidos");
        }
        if (count($salidas) >= MAXIMO) throw new RuntimeException("$nombre ya tiene " . MAXIMO . " préstamos");
        $pdo->prepare("INSERT INTO prestamos (codigo, socio, salida) VALUES (?, ?, ?)");
        $pdo->execute([$codigo, $socio, $fecha]);
        $id = $pdo->lastInsertId();
        $pdo->commit();
        echo "Préstamo $id: «{$titulo}» para $nombre (devolver antes del " . vence($fecha) . ")\n";
    } catch (RuntimeException $e) {
        $pdo->rollBack();
        echo $e->getMessage() . "\n";
    }
}

function devolver(PDO $pdo, int $id, string $fecha): void {
    $st = $pdo->prepare("SELECT p.salida, p.devolucion, l.titulo FROM prestamos p JOIN libros l USING (codigo) WHERE p.id = ?");
    $st->execute([$id]);
    $p = $st->fetch();
    if (!$p) { echo "No existe el préstamo $id\n"; return; }
    if ($p['devolucion'] !== null) { echo "El préstamo $id ya se devolvió el {$p['devolucion']}\n"; return; }
    $pdo->prepare("UPDATE prestamos SET devolucion = ? WHERE id = ?");
    $pdo->execute([$fecha, $id]);
    $retraso = max(0, dias($p['salida'], $fecha) - PLAZO);
    if ($retraso === 0) {
        echo "Devuelto «{$p['titulo']}» a tiempo\n";
    } else {
        echo "Devuelto «{$p['titulo']}» con $retraso " . ($retraso === 1 ? "día" : "días") . " de retraso: multa " . euros($retraso * MULTA) . "\n";
    }
}

function morosos(PDO $pdo, string $fecha): void {
    $st = $pdo->query("SELECT s.nombre, l.titulo, p.salida FROM prestamos p JOIN socios s ON s.id = p.socio JOIN libros l ON l.codigo = p.codigo WHERE p.devolucion IS NULL ORDER BY p.salida, p.id");
    $lista = [];
    foreach ($st as $fila) {
        $retraso = dias($fila['salida'], $fecha) - PLAZO;
        if ($retraso > 0) $lista[] = "{$fila['nombre']}: «{$fila['titulo']}» ($retraso " . ($retraso === 1 ? "día" : "días") . " de retraso)";
    }
    echo $lista ? "Préstamos vencidos el $fecha:\n" . implode("\n", $lista) . "\n" : "No hay préstamos vencidos el $fecha\n";
}

function resumen(PDO $pdo): void {
    $activos = $pdo->query("SELECT COUNT(*) FROM prestamos WHERE devolucion IS NULL");
}

```

```

echo "Préstamos activos: $activos\n";
$top = $pdo->query("SELECT s.nombre, COUNT(*) AS n FROM prestamos p JOIN socios s ON s.id = p.socio
    GROUP BY s.id ORDER BY n DESC, s.nombre LIMIT 1")->fetch();
echo $top ? "Socio más lector: {$top['nombre']} ({$top['n']} préstamos)\n" : "Socio más lector:
    ninguno\n";
}

foreach (explode("\n", stream_get_contents(STDIN)) as $linea) {
    $linea = trim($linea);
    if ($linea === "") continue;
    $p = preg_split('/\s+/', $linea, 2);
    $orden = strtoupper($p[0]);
    $args = isset($p[1]) ? preg_split('/\s+/', $p[1]) : [];
    echo "> $linea\n";
    match (true) {
        $orden === "BUSCAR" && isset($p[1]) => buscar($pdo, $p[1]),
        $orden === "PRESTAR" && count($args) === 3 => prestar($pdo, $args[0], (int) $args[1], $args[2]),
        $orden === "DEVOLVER" && count($args) === 2 => devolver($pdo, (int) $args[0], $args[1]),
        $orden === "MOROSOS" && count($args) === 1 => morosos($pdo, $args[0]),
        $orden === "RESUMEN" => resumen($pdo),
        default => print("Orden no válida\n"),
    };
}

```

Toda consulta con datos del usuario es preparada. Además de proteger frente a la inyección de SQL, evita los errores de comillas (O'Brien) y permite reutilizar la consulta con distintos valores.

El préstamo es el caso de libro de una transacción: varias lecturas (disponibles, préstamos activos) y una escritura que solo tiene sentido si las lecturas siguen siendo ciertas. Con `beginTransaction / commit / rollBack` todo ocurre o nada ocurre; en MySQL se añadiría `SELECT ... FOR UPDATE` para bloquear las filas leídas.

Los ejemplares disponibles no se guardan en ninguna columna: se calculan con un `LEFT JOIN` a partir de los préstamos. Así nunca pueden quedar desincronizados, que es lo que pasaría si cada préstamo tuviera que acordarse de restar uno.

Las reglas se expresan como excepciones con el mensaje para el usuario, de modo que la transacción tiene un único camino de éxito y un único `catch` que deshace. `DateTimeImmutable` hace las cuentas de fechas sin errores de meses o años bisiestos.

46. API REST de tareas con un router propio

Difícil · PHP · 90 minutos · apuntesdam.com/ejercicios/php/api-rest-de-tareas

Solución (php)

```
<?php
/** Una ruta: método, patrón con {parámetros} y la función que la atiende. */
final class Ruta {
    public function __construct(public string $metodo, public string $patron, public Closure $accion) {}

    /** Los parámetros de la ruta si la URL encaja con el patrón, o null. */
    public function encaja(string $ruta): ?array {
        $regex = "#^" . preg_replace('#\{(\w+)\}#', '(<P$1>[^/]+)', $this->patron) . "$#";
        if (!preg_match($regex, $ruta, $m)) return null;
        return array_filter($m, "is_string", ARRAY_FILTER_USE_KEY);
    }
}

final class Respuesta {
    public function __construct(public int $estado, public mixed $cuerpo = null) {}
}

final class Router {
    /** @var Ruta[] */
    private array $rutas = [];

    public function add(string $metodo, string $patron, Closure $accion): void {
        $this->rutas[] = new Ruta($metodo, $patron, $accion);
    }

    public function despachar(string $metodo, string $ruta, array $consulta, ?string $cuerpo): Respuesta
    {
        $rutaExiste = false;
        foreach ($this->rutas as $r) {
            $params = $r->encaja($ruta);
            if ($params === null) continue;
            $rutaExiste = true;
            if ($r->metodo === $metodo) return ($r->accion)($params, $consulta, $cuerpo);
        }
        // Si la ruta existe pero no con ese método, es 405, no 404
        return $rutaExiste ? new Respuesta(405, ["error" => "Método no permitido"]) : new Respuesta(404,
            ["error" => "Ruta no encontrada"]);
    }
}

$tareas = [];
$siguiente = 1;
const PRIORIDADES = ["alta", "media", "baja"];

/** Valida los campos que vienen en $datos; con $nueva, el título es obligatorio. Devuelve los errores.
 */
function validar(array $datos, bool $nueva): array {
    $errores = [];
    if ($nueva && !array_key_exists("titulo", $datos)) {
        $errores["titulo"] = "obligatorio";
    } elseif (array_key_exists("titulo", $datos) && (!is_string($datos["titulo"]) ||
        mb_strlen(trim($datos["titulo"])) < 3 || mb_strlen(trim($datos["titulo"])) > 60)) {
        $errores["titulo"] = "entre 3 y 60 caracteres";
    }
    if (array_key_exists("prioridad", $datos) && !in_array($datos["prioridad"], PRIORIDADES, true)) {
        $errores["prioridad"] = "debe ser alta, media o baja";
    }
    if (array_key_exists("hecha", $datos) && !is_bool($datos["hecha"])) {
        $errores["hecha"] = "debe ser true o false";
    }
    $permitidos = ["titulo", "prioridad", "hecha"];
    foreach (array_diff(array_keys($datos), $permitidos) as $campo) $errores[$campo] = "campo desconocido";
    return $errores;
}
```

```

/** El cuerpo JSON como array asociativo, o null si no es un objeto JSON. */
function json(?string $cuerpo): ?array {
    $datos = json_decode($cuerpo ?? "", true);
    return is_array($datos) && !array_is_list($datos) ? $datos : (($cuerpo ?? "") === "{}" ? [] : null);
}

$router = new Router();

$router->add("GET", "/api/tareas", function (array $p, array $q) use (&$tareas): Respuesta {
    $lista = array_values($tareas);
    if (isset($q["hecha"])) $lista = array_values(array_filter($lista, fn($t) => $t["hecha"] === ($q["hecha"] === "true")));
    if (isset($q["q"])) $lista = array_values(array_filter($lista, fn($t) => mb_stripos($t["titulo"], $q["q"]) !== false));
    return new Respuesta(200, $lista);
});

$router->add("POST", "/api/tareas", function (array $p, array $q, ?string $cuerpo) use (&$tareas, &$siguiente): Respuesta {
    $datos = json($cuerpo);
    if ($datos === null) return new Respuesta(400, ["error" => "JSON no válido"]);
    if ($errores = validar($datos, true)) return new Respuesta(422, ["errores" => $errores]);
    $tarea = ["id" => $siguiente++, "titulo" => trim($datos["titulo"]), "prioridad" => $datos["prioridad"] ?? "media", "hecha" => $datos["hecha"] ?? false];
    $tareas[$tarea["id"]] = $tarea;
    return new Respuesta(201, $tarea);
});

$router->add("GET", "/api/tareas/{id}", function (array $p) use (&$tareas): Respuesta {
    return isset($tareas[$p["id"]]) ? new Respuesta(200, $tareas[$p["id"]]) : new Respuesta(404, ["error" => "No existe la tarea {$p['id']}"]);
});

$router->add("PATCH", "/api/tareas/{id}", function (array $p, array $q, ?string $cuerpo) use (&$tareas): Respuesta {
    if (!isset($tareas[$p["id"]])) return new Respuesta(404, ["error" => "No existe la tarea {$p['id']}"]);
    $datos = json($cuerpo);
    if ($datos === null) return new Respuesta(400, ["error" => "JSON no válido"]);
    if ($errores = validar($datos, false)) return new Respuesta(422, ["errores" => $errores]);
    if (isset($datos["titulo"])) $datos["titulo"] = trim($datos["titulo"]);
    $tareas[$p["id"]] = array_merge($tareas[$p["id"]], $datos);
    return new Respuesta(200, $tareas[$p["id"]]);
});

$router->add("DELETE", "/api/tareas/{id}", function (array $p) use (&$tareas): Respuesta {
    if (!isset($tareas[$p["id"]])) return new Respuesta(404, ["error" => "No existe la tarea {$p['id']}"]);
    unset($tareas[$p["id"]]);
    return new Respuesta(204);
});

foreach (explode("\n", stream_get_contents(STDIN)) as $linea) {
    $linea = trim($linea);
    if ($linea === "") continue;
    $partes = explode(" ", $linea, 3);
    if (count($partes) < 2) {
        echo "Petición no válida\n";
        continue;
    }
    [$metodo, $url] = $partes;
    $ruta = parse_url($url, PHP_URL_PATH) ?? "";
    $consulta = parse_str(parse_url($url, PHP_URL_QUERY) ?? "", $consulta);
    $r = $router->despachar(strtoupper($metodo), rtrim($ruta, "/") ? "/" : "", $consulta, $partes[2] ?? null);
    $json = $r->cuerpo === null ? "" : " " . json_encode($r->cuerpo, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_SLASHES);
    echo strtoupper($metodo) . " $url → {"$r->estado}$json\n";
}

```

El router separa el «a dónde va cada petición» del «qué hace cada una»: registrar una ruta nueva no toca nada más. Convertir los patrones en expresiones regulares con grupos con nombre es exactamente lo que hacen los routers de los frameworks.

Los códigos de estado son parte de la API: un cliente decide qué hacer según el código antes de leer el cuerpo. Distinguir 400 (no entiendo la petición), 404 (no existe), 405 (existe, pero no así) y 422 (entiendo la petición, pero los datos no valen) es lo que hace una API predecible.

Las closures con `use (&$tareass)` comparten el almacén en memoria; en una aplicación real ese estado viviría en una base de datos y se inyectaría un repositorio, pero la estructura de las acciones sería la misma.

La validación es una función que acumula errores por campo en lugar de parar en el primero: el cliente puede mostrar todos los problemas del formulario de una vez. PATCH reutiliza las mismas reglas sin exigir el título.

47. Libro de visitas seguro: validación, spam y escape de HTML

Medio · PHP · 55 minutos · apuntesdam.com/ejercicios/php/libro-de-visitas-seguro

Solución (php)

```
<?php
const MAX_POR_EMAIL = 2;

/** Errores de un mensaje (campo → motivo), en el orden de los campos. */
function validar(array $f): array {
    $e = [];
    $nombre = trim($f["nombre"] ?? "");
    if ($nombre === "") $e["nombre"] = "obligatorio";
    elseif (mb_strlen($nombre) < 2 || mb_strlen($nombre) > 50) $e["nombre"] = "entre 2 y 50 caracteres";

    $email = trim($f["email"] ?? "");
    if ($email === "") $e["email"] = "obligatorio";
    elseif (filter_var($email, FILTER_VALIDATE_EMAIL) === false) $e["email"] = "no es un email válido";

    // La web es opcional, pero solo http o https: «javascript:» en un enlace es una puerta a XSS
    $web = trim($f["web"] ?? "");
    if ($web !== "" && (filter_var($web, FILTER_VALIDATE_URL) === false ||
        !in_array(strtolower(parse_url($web, PHP_URL_SCHEME)), ["http", "https"], true))) {
        $e["web"] = "solo direcciones http o https";
    }

    $mensaje = trim($f["mensaje"] ?? "");
    if ($mensaje === "") $e["mensaje"] = "obligatorio";
    elseif (mb_strlen($mensaje) > 200) $e["mensaje"] = "como mucho 200 caracteres";
    return $e;
}

/** Escapa para HTML: <, >, &, comillas dobles y simples. */
function h(string $texto): string {
    return htmlspecialchars($texto, ENT_QUOTES | ENT_HTML5, "UTF-8");
}

$publicados = [];
$porEmail = [];
$rechazados = $spam = 0;
$n = 0;
foreach (explode("\n", stream_get_contents(STDIN)) as $linea) {
    if (trim($linea) === "") continue;
    $n++;
    parse_str(trim($linea), $f);
    // Campo trampa: está oculto con CSS, así que solo lo rellenan los robots
    if (($f["telefono2"] ?? "") !== "") {
        $spam++;
        echo "Mensaje $n descartado: spam\n";
        continue;
    }
    $errores = validar($f);
    $email = strtolower(trim($f["email"] ?? ""));
    if (!$errores && ($porEmail[$email] ?? 0) >= MAX_POR_EMAIL) $errores["email"] = "ya ha enviado " .
        MAX_POR_EMAIL . " mensajes";
    if ($errores) {
        $rechazados++;
        foreach ($errores as $campo => $motivo) echo "Mensaje $n rechazado: $campo: $motivo\n";
        continue;
    }
    $porEmail[$email] = ($porEmail[$email] ?? 0) + 1;
    $publicados[] = $f;
}

echo "<ul class=\"visitas\">\n";
foreach ($publicados as $f) {
    $autor = "<strong>" . h(trim($f["nombre"])) . "</strong>";
    $web = trim($f["web"] ?? "");
    if ($web !== "") $autor .= " (<a href=\"" . h($web) . "\" rel=\"nofollow ugc\">web</a>);";
    echo "  <li>$autor: " . h(trim($f["mensaje"])) . "</li>\n";
}
```

```
echo "</ul>\n";
echo "Publicados: " . count($publicados) . " . Rechazados: $rechazados . Spam: $spam\n";
```

La seguridad tiene dos momentos: al recibir (validar que cada dato es lo que dice ser) y al mostrar (escapar para el contexto donde se escribe). Validar no sustituye a escapar: un nombre como `O'Brien <3` es válido y aun así hay que escaparlo.

`htmlspecialchars` con `ENT_QUOTES` escapa también las comillas simples, necesarias cuando el valor va dentro de un atributo como `href`. Por eso el enlace se escribe con el valor escapado y, además, solo se aceptan esquemas `http` y `https`.

El campo trampa detecta robots sin molestar a las personas (a diferencia de un CAPTCHA). No es infalible, pero combinado con el límite por email frena la mayor parte del spam automático.

`rel="nofollow ugc"` le dice a los buscadores que el enlace lo ha puesto un usuario: así los spammers no ganan posicionamiento enlazando desde tu web.

48. Horas trabajadas con nocturnidad usando DateTime

Difícil · PHP · 70 minutos · apuntesdam.com/ejercicios/php/horas-trabajadas-y-nocturnidad

Solución (php)

```
<?php
/** Minutos de [$inicio, $fin) que caen entre las 22:00 y las 06:00. */
function minutosNocturnos(DateTimeImmutable $inicio, DateTimeImmutable $fin): int {
    $total = 0;
    // Cada noche va de las 22:00 de un día a las 06:00 del siguiente; se empieza por la noche anterior
    // al inicio
    $noche = $inicio->modify("-1 day")->setTime(22, 0);
    while ($noche < $fin) {
        $desde = max($inicio, $noche);
        $hasta = min($fin, $noche->modify("+8 hours"));
        if ($hasta > $desde) $total += intval($hasta->getTimestamp() - $desde->getTimestamp(), 60);
        $noche = $noche->modify("+1 day");
    }
    return $total;
}

function duracion(int $minutos): string {
    $h = intval($minutos, 60);
    $m = $minutos % 60;
    return $m === 0 ? "$h h" : "$h h $m min";
}

function euros(int $c): string {
    return number_format($c / 100, 2, ",", ".") . " €";
}

$lineas = explode("\n", stream_get_contents(STDIN));
if (!preg_match('/^TARIFA (\d+(\.\d{1,2})?)$/ ', trim($lineas[0]), $m)) {
    echo "Falta la tarifa\n";
    exit;
}

$tarifa = (int) round((float) $m[1] * 100); // céntimos por hora

$turnos = []; // nombre → lista de [inicio, fin]
foreach (array_slice($lineas, 1) as $i => $linea) {
    $linea = trim($linea);
    if ($linea === "") continue;
    $num = $i + 2;
    $campos = array_map("trim", explode(";", $linea));
    $inicio = count($campos) === 3 ? DateTimeImmutable::createFromFormat("!Y-m-d H:i", $campos[1]) :
        false;
    $fin = count($campos) === 3 ? DateTimeImmutable::createFromFormat("!Y-m-d H:i", $campos[2]) : false;
    // createFromFormat acepta el 31 de septiembre (lo pasa al 1 de octubre): se comprueba que la fecha
    // no cambie
    if (!$inicio || !$fin || $inicio->format("Y-m-d H:i") !== $campos[1] || $fin->format("Y-m-d H:i") !==
        $campos[2] || $campos[0] === "") {
        echo "Línea $num: formato no válido\n";
        continue;
    }
    $nombre = $campos[0];
    $minutos = intval($fin->getTimestamp() - $inicio->getTimestamp(), 60);
    if ($minutos <= 0) { echo "Línea $num: la salida no es posterior a la entrada\n"; continue; }
    if ($minutos > 12 * 60) { echo "Línea $num: un turno no puede pasar de 12 horas\n"; continue; }
    foreach ($turnos[$nombre] ?? [] as [$a, $b]) {
        if ($inicio < $b && $a < $fin) { echo "Línea $num: se solapa con otro turno de $nombre\n";
            continue 2; }
    }
    $turnos[$nombre][] = [$inicio, $fin];
}

ksort($turnos, SORT_STRING);
$total = 0;
foreach ($turnos as $nombre => $lista) {
    $minutos = $nocturnos = 0;
    foreach ($lista as [$a, $b]) {
        $minutos += intval($b->getTimestamp() - $a->getTimestamp(), 60);
    }
    $nocturnos = minutosNocturnos($a, $b);
    $total += $minutos + $nocturnos;
}
```

```

        $nocturnos += minutosNocturnos($a, $b);
    }
    // Todas las horas a la tarifa, y las nocturnas con un 25 % más
    $importe = (int) round(($minutos * $tarifa + $nocturnos * $tarifa * 0.25) / 60);
    $total += $importe;
    $turnosTexto = count($lista) === 1 ? "1 turno" : count($lista) . " turnos";
    echo "$nombre: $turnosTexto, " . duracion($minutos) . " (nocturnas " . duracion($nocturnos) . ") . "
        . euros($importe) . "\n";
}
echo "Total a pagar: " . euros($total) . "\n";

```

La clave del ejercicio es tratar los turnos y las noches como intervalos de tiempo: la parte nocturna de un turno es su intersección con cada noche. El mismo razonamiento sirve para tarifas eléctricas por franjas, aparcamientos o reservas de salas.

`DateTimeImmutable` hace las cuentas de calendario (cambio de día, de mes o de año) y su inmutabilidad evita los errores de modificar sin querer una fecha que se sigue usando. Comparar objetos de fecha con `<` y usar `max/min` funciona directamente.

La validación de fechas con la vuelta al texto original detecta lo que `createFromFormat` corrige en silencio. Es el mismo problema que con `Date` en JavaScript: las librerías de fechas prefieren «arreglar» antes que fallar.

El importe se calcula en céntimos con un único redondeo por persona, para que la suma de las nóminas sea exactamente el total que se escribe.