Tecnologies de Desenvolupament per a Internet i Web

Curs 2024-2025

Pràctica: *Botiga virtual.* Sessió 2

Índex

1 Dates **1**

2 Objectius **1**

3 Feina prèvia abans de la sessió **1**

4 Feina durant la sessió i abans de la següent sessió **1**
4.1 MVC 1
4.2 Base de dades 6
4.3 Llistat dinàmic de categories 6

Apèndixs **8**

A Pas de paràmetres amb el protocol HTTP **8**

B ___DIR___ **9**

C *Debug* **10**
C.1 `print_r` 11
C.2 `var_dump` 12

D `include`, `include_once`, `require`, `require_once` **13**

E Base de dades **16**
E.1 Connexió 16
E.2 Consultes 17
E.2.1 Consultes amb múltiples resultats 17
E.2.2 Consulta amb un únic resultat 19

F == i === **20**

Referències **24**

1 Dates

Grups	**Dia**
Grups A, B, C	21/10
Grup D	21/10
Grups E, F, G, H	16/10
Grups I, J, K	17/10
Grups L, M, N	18/10

2 Objectius

Pes	**Preferent?**	**Funcionalitat**
Sistema de productes		
0'2	✓	Llistat de categories
General		
0'8	✓	MVC

3 Feina prèvia abans de la sessió

- Repasseu un tutorial de PHP bàsic.[9]
- Repasseu el paradigma de programació `MVC`.
- Porteu, en un fitxer o en paper, el diagrama lògic de la vostra base de dades perquè el professor us el pugui validar.

4 Feina durant la sessió i abans de la següent sessió

4.1 MVC

Abans de començar amb aquesta tasca, si no teniu clar com funciona el pas de paràmetres en una aplicació web, reviseu l'apèndix A.

Heu de fer tot el vostre web seguint l'arquitectura MVC. Per la vostra pràctica us recomanem tenir a l'arrel del vostre web —el directori `public_html`— un fitxer `index.php` que faci d'encaminador —en anglès, `router`—, és a dir, que rebi totes les peticions del web i executi les accions corresponents. Els vostres url, per tant, seran de la forma:

```
index.php?accio=llistar-categories
```

Amb els paràmetres de la *query request* de la operació `GET` especifiqueu l'acció a executar i tota la informació addicional que necessiteu per dur-la a terme —identificadors per fer filtres, paginacions... En aquest cas, amb el paràmetre `accio` especifiqueu que voleu obtenir el llistat de categories. Aleshores, al vostre fitxer `index.php` tindreu una cosa similar a això:

```php
<?php
// index.php

$accio = $_GET['accio'] ?? NULL;

switch ($accio) {
    case 'llistar-categories':
        include __DIR__.'/resource_llistar_categories.php';
        break;
    default:
        include __DIR__.'/resource_portada.php';
        break;
}
```

Com veieu, el que fa l'encaminador és incloure el recurs adient en funció de l'acció sol · licitada. Les accions que hi ha definides són les del llistat de categories i la de la portada però, segons aneu afegint més funcionalitats pràctica, haureu d'afegir tantes accions com necessiteu.

Un fitxer de recurs conté l'estructura de la pàgina sol · licitada, i inclou els controladors adients, com podeu veure a l'esquema de la imatge següent:

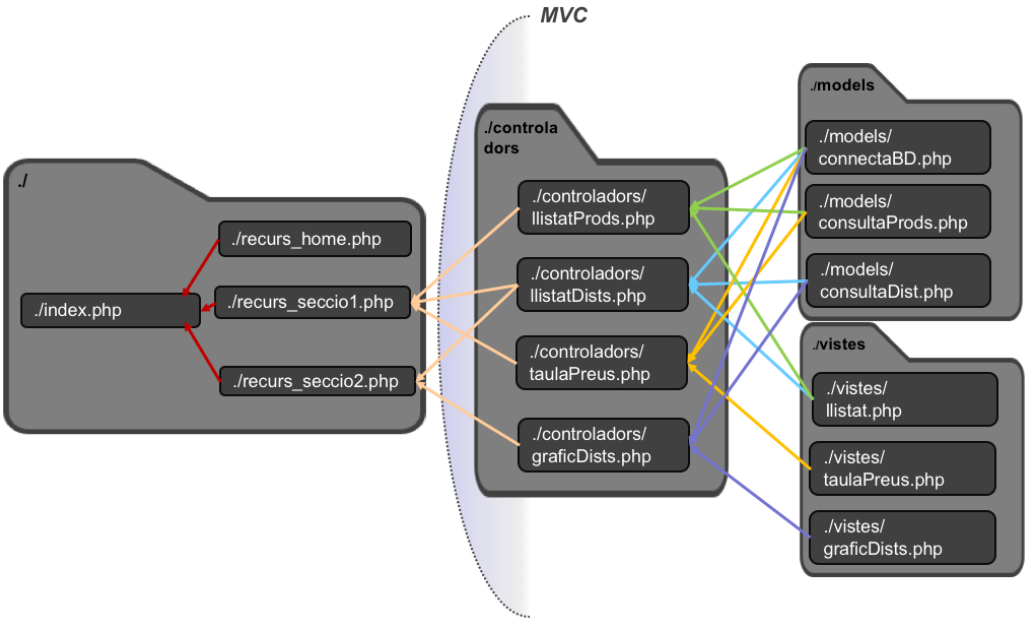

Figura 1: Funcionament dels fitxers de recursos

Un fitxer de recurs que mostrés un menú i el llistat de categories podria tenir l'estructura següent:

```
<!-- resource_llistar_categories.php -->

<html lang="ca">
<head>
<title>Llistat de categories - TDIW</title>
</head>
<body>

<header>
    <?php require __DIR__.'/controller/menu_superior.php'; ?>
</header>

<div class="container">
    <?php require __DIR__.'/controller/llistar_categories.php'; ?>
</div>

</body>
</html>
```

A partir d'aquí, se segueix el patró MVC estudiat a classe: el controlador demana al model les dades corresponents i les passa a la vista, que les renderitza i mostra per pantalla. Seguint amb l'exemple de llistar les categories, podríem tenir el següent controlador:

```php
<?php
// controller/llistar_categories.php

require_once __DIR__.'/../model/connectaDb.php';
require_once __DIR__.'/../model/categories.php';

$categories = getCategories(); // Aquesta crida és al model

include __DIR__.'/../views/llistar_categories.php';
```

I la vista podria ser la següent:

```php
<!-- // views/llistar_categories.php -->
<ul>
<?php foreach ($categories as $categoria): ?>
    <li>
        <h3><?php echo $categoria['nom'] ?></h3>
        <p><?php echo $categoria['descripcio'] ?></p>
    </li>
<?php endforeach; ?>
</ul>
```

Tingueu en compte que aquests exemples no són complets, només són una guia, i que hi manquen coses per ser completament funcionals: incloure els fitxers que corresponen pel model, tenir la vista completa amb els estils CSS...

Un cop teniu més clar el funcionament de MVC amb un encaminador, el que heu de fer a continuació és:

- Crear el vostre encaminador, `index.php`.
- Crear els fitxers de recursos per les pàgines que ja heu desenvolupat.
- Crear els directoris `model`, `view` i `controller`

- Modificar les pàgines que ja heu desenvolupat perquè s'adaptin al patró MVC.

Per saber si esteu seguint correctament el patró MVC, us detallem a a la taula 1 quins llenguatges de programació són vàlids a cadascun dels elements de la vostra pràctica.

	Encaminador	**Recurs**	**Controlador**	**Model**	**Vista**
PHP	✓	✓	✓	✓	✓
Postgres	✗	✗	✗	✓	✗
HTML	✗	✓	✗	✗	✓
Javascript	✗	✓[1]	✗	✗	✓
CSS	✗	✓[2]	✗	✗	✓

Taula 1: llenguatges programació vàlids a cadascun dels elements del patró MVC amb enrutador i recursos

Si us trobeu algun cas en què utilitzeu un llenguatge de programació en un dels elements que no correspon amb el que hi ha a la taula, no esteu seguint correctament el patró.

Per últim, recordar-vos que tingueu cura amb *hardcodejar* dades al vostre codi font. *Hardcodejar* vol dir escriure directament al codi dades que no són pròpies de la implementació, sinó que són dades dinàmiques que es coneixen en temps d'execució, com poden ser dades de la base de dades. Si *hardcodegeu* dades, el vostre web deixa de ser dinàmic.

I, en qualsevol cas, *hardcodejar* es sempre una **mala pràctica** que heu d'evitar a qualsevol llenguatge de programació que feu servir.

Uns exemples comuns de *hardcoding* a pràctiques de cursos anteriors són:

- Escriure directament l'identificador de les categories i dels productes al codi font. Pregunteu-vos què passaria si tinguéssiu de sobte 10.000 productes nous a afegir.

- Tenir un fitxer de controlador per cadascuna de les categories de la base de dades. Penseu què hauria de fer una persona no tècnica per afegir una nova categoria.

[2]És recomenat que estigui en fitxers a part.
[2]Hauria d'estar en fitxers a part.

4.2 Base de dades

Un cop tingueu el disseny validat pel vostre professor, heu de crear les taules a la base de dades i les relacions necessàries entre elles. Teniu a la vostra disposició el PHPMyAdmin, però podeu fer servir qualsevol altra eina o escriure vosaltres mateixos els scripts.

Per a la vostra pràctica us recomanem que cada taula tingui un camp **id** autoincremental. Us llistem també els camps mínims que han de tenir els productes:

- Nom
- Imatge (de moment, deseu un camp de text per fer-hi referència)
- Descripció
- Preu

Penseu, a més a més, si us és convenient tenir un camp booleà que indiqui si el producte és actiu o no, és a dir, si és visible o no a la botiga. Això us permetrà fer un esborrat lògic de productes, o publicar productes a una data concreta —per promocions, per exemple—, però aleshores heu de tenir en compte que només heu de mostrar els productes actius al *frontend*.

També heu d'afegir dades de productes i categories a la base de dades perquè es visualitzin a la part pública del vostre web. Per fer-ho, de nou, podeu fer servir el PHPMyAdmin o qualsevol altra eina, o escriure vosaltres mateixos scripts que us afegeixin les dades a la base de dades. És important remarcar que no cal que us hi escarrasseu massa en les dades que afegiu, no cal que hi hagi moltíssimes dades ni que siguin molt completes, ja que aquest no és l'objectiu de la pràctica; és a dir, si, per exemple, féssiu un web on es venen videojocs, no cal que hi afegiu 1.000 títols amb la seva descripció completa i precisa, sinó que, amb uns 10 títols —o el que us permeti navegar pel web— ja n'hi ha prou.

Pel model de dades de l'usuari, teniu tots els camps que necessiteu desar a l'enunciat de la sessió 1. També necessitareu un camp de text per desar la imatge de perfil de l'usuari. A la sessió 6 veurem com permetre a l'usuari pujarla imatge de perfil.

4.3 Llistat dinàmic de categories

Un cop tingueu el patró MVC aplicat i la base de dades creada i amb dades inserides, heu de fer el llistat dinàmic de categories, és a dir, mostrar a una pàgina la informació de les categories agafada de la base de dades. Tingueu

en compte que no podeu fer servir subcategories, és a dir, no podeu tenir més d'un nivell de categories.

A la figura 2 teniu un exemple del llistat de categories d'un dels webs del curs 2017-2018.

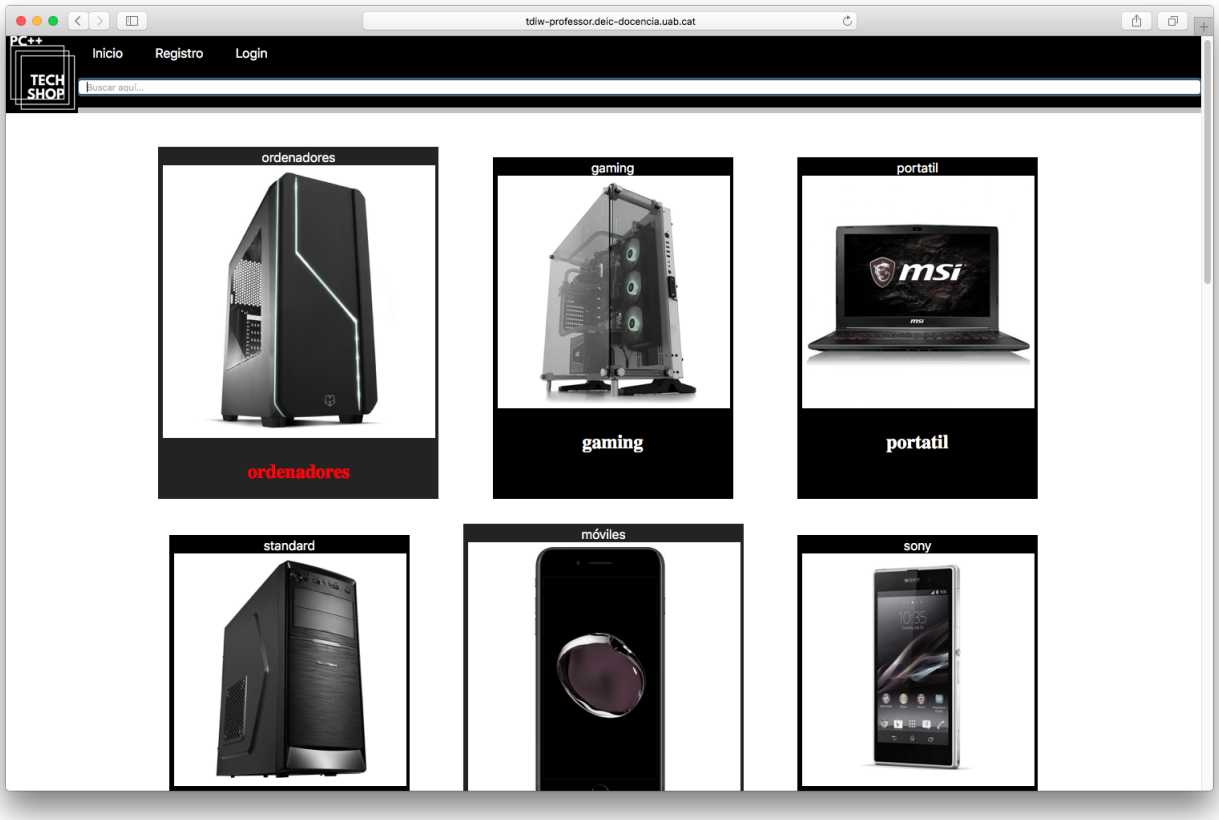

Figura 2: Exemple del llistat de categories d'un dels webs del curs 2017-2018

Apèndixs

A Pas de paràmetres amb el protocol HTTP

HTTP és un protocol *stateless*, és a dir, sense estat, de manera que el client —el vostre navegador— i el servidor són agnòstics entre ells, no es manté cap informació d'estat. Això vol dir que, entre la càrrega d'una pàgina i la següent, no es manté cap variable de les que hàgiu definit prèviament[3]. En conseqüència, cada cop que vulgueu anar d'una pàgina a una altra, haureu de passar tota la informació necessària.

Imagineu que teniu al vostre web cinc categories, cadascuna d'elles amb un identificador que va de l'1 al 5, i que teniu un llistat on mostreu el nom de cadascuna d'aquestes categories. Aquesta pàgina seria el `llistat de categories`.

Tenim també una altra pàgina amb el llistat de productes d'una categoria, a la qual anomenarem `llistat de productes`. Què caldria fer per mostrar els productes de la categoria amb `id=3`?

Està clar que el que necessitem és agafar tots els productes de la base de dades que pertanyin a aquesta categoria i que, per tant, ens cal aquest identificador de categoria. A més a més, com que HTTP és un protocol sense estat, cal tornar a carregar la pàgina.

Però com passem aquest identificador des de la pàgina de `llistat de categories` a la pàgina de `llistat de productes`?

La manera més simple és passar-lo com a paràmetre a l'url. En una petició web, tot el que hi hagi després de `?` és considerat com a paràmetre que es pot recollir i utilitzar a l'aplicació. Així doncs, podríem tenir al final del nostre url una cosa com:

`?categoria_id=3`

D'aquesta manera, al nostre *script* PHP podríem recuperar el valor de l'identificador de la categoria des de la variable superglobal `$_GET`, de la següent manera, i ja podríem agafar tots els productes corresponents:

```php
<?php

// El valor de la variable $categoriaId serà de '3'
$categoriaId = $_GET['categoria_id'];
```

[3]Sovint sí que cal desar un estat i, amb aquest fi, s'utilitzen sessions, que veurem a la sessió 5.

Recordeu que també es poden fer peticions a escripts mitjançant AJAX o bé a partir del submit d'un formulari utilitzant el mètode POST. En aquest cas, utilitzarieu la super variable `$_POST` per accedir als paràmetres. També teniu disponible la super variable `$_REQUEST` que accedeix tant als paràmetres enviats per POST com per GET, actuant com a comodí.

Tant `$_POST` com `$_GET` com `$_REQUEST` son diccionaris que PHP implementa com arrays associatius.

B ___DIR___

A PHP, el directori actual és sempre el directori del fitxer que s'està executant. Això pot donar peu a confusions a l'hora d'incloure fitxers, ja que és possible que us trobeu un cas en què esteu programant en un fitxer inclòs, i n'heu d'incloure un altre des d'aquest fitxer, de manera que no sabeu ben bé quina és la ruta que heu d'afegir.

Per resoldre aquest problema, teniu la constant «màgica» `__DIR__`[8] de PHP, que sempre conté el valor del directori del fitxer actual, sense la barra final. Fixeu-vos que té dos guions baixos `_` davant i dos darrere.

Aquesta constant us és útil quan treballeu amb el patró MVC, ja que heu d'incloure fitxers des del controlador, com el model o les vistes, i no sempre sabreu des d'on esteu executant l'*script* actual, de manera que podeu combinar la constant `__DIR__` amb rutes relatives als vostres fitxers.

Seguint l'exemple de l'apartat 4.1, i suposant que els fitxers són al directori `/home/tdiw/tdiw-a1/public_html`, els valors del directori actual i de la constant `__DIR__` serien els següents:

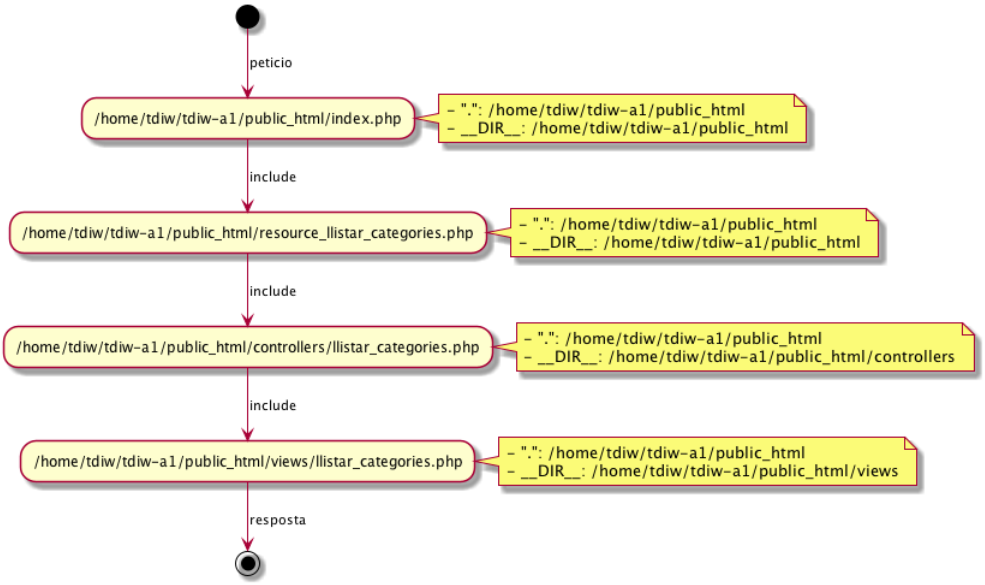

Figura 3: Exemple de valor de la constant ___DIR___

Fixeu-vos que el valor de `.` sempre és el mateix, però varia en funció de l'*script* que el crida, de manera que no és fiable fer-lo servir.

Nosaltres us recomanem fer ús de la constant `__DIR__` per construir el vostre web seguint el patró MVC, ja que fent-ne ús sempre podreu controlar des de quin directori heu d'incloure altres fitxers.

C *Debug*

A PHP existeixen depuradors de codi com els que coneixeu de llenguatges compilats, com poden ser Xdebug[2] o Zend Debugger[16]. Això no obstant, no són senzills de configurar, ja que la seva configuració s'ha de fer tant al servidor com al client, obrint ports, configurant serveis, els navegadors... La infraestructura de xarxa del departament no permet aquesta configuració sense comprometre'n la seguretat, i és per aquest motiu que no n'hi ha cap de configurat.

Així doncs, per depurar el vostre codi us proposem dues alternatives equivalents entre elles, més «manuals», que consisteixen a mostrar el valor de les variables per pantalla.

C.1 `print_r`

La primera opció que us proposem per depurar el vostre codi és la funció `print_r()`[11]. Aquesta funció mostra per pantalla de manera llegible els valors que contingui la variable que li passeu per paràmetre.

Suposant que voleu veure què hi ha a la variable superglobal `$_GET`, podríeu fer servir aquestes instruccions:

```
1 <?php
2
3 echo '<pre>';
4 print_r($_GET);
5 die;
```

La línia `3` mostra per pantalla l'etiqueta HTML `<pre>`[3], que mostra el text que vindrà a continuació amb format.

La línia `4` crida a la funció `print_r` passant-li com a paràmetre la variable superglobal `$_GET`. Aquest és només un exemple, però l'hi podeu passar qualsevol variable que tingueu definida al vostre codi.

La línia `5` crida a la funció `die`[5] de PHP, que el que fa és tallar l'execució del programa, de manera que ja no s'executarà cap codi que vingui a continuació.

Fixeu-vos que, com que és un codi que fem servir només per depurar, no tanquem l'etiqueta `<pre>`, però sí que seria necessari fer-ho si aquest codi fos part del nostre web.

Aquest codi mostraria per pantalla una cosa similar a això:

```
1 Array
2 (
3 [action] => categoria
4 [category_id] => 1
5 )
```

També podeu tenir el codi en una sola línia, perquè us sigui més fàcil de comentar per fer proves:

```
1 <?php
2
3 echo '<pre>';print_r($_GET);die;
```

Alternativament a la utilització del tag `<pre>`, utilitzeu el *web inspector*, el menú *network*, la pestanya *Response* i busqueu el que heu imprés amb la funció `print_r` que us ha tornat en servidor.

C.2 `var_dump`

La segona proposta que us fem per depurar el vostre codi és la funció `var_dump`[15]. El seu funcionament és anàleg a la funció `print_r` però, a més a més, també imprimeix per pantalla la llargària i el tipus de dada de la variable. Us pot ser útil en casos en què tingueu dubtes amb el tipus de dada amb què treballeu.

Així doncs, igual que en l'exemple anterior, suposant que volem depurar els valors de la variable superglobal `$_GET`, podríeu fer servir el següent codi:

```php
<?php

echo '<pre>';
var_dump($_GET);
die;
```

O, en una única línia:

```php
<?php

echo '<pre>';var_dump($_GET);die;
```

La sortida seria alguna cosa similar a això:

```
array (size=2)
'action' => string 'categoria' (length=9)
'category_id' => string '1' (length=1)
```

Recordeu que, igual que en el cas anterior, podeu passar qualsevol variable que tingueu definida en el vostre codi.

D `include`, `include_once`, `require`, `require_once`

Les funcions `include`[6] i `require` [12] són equivalents, i la seva funcionalitat és exactament la mateixa. El que fan és incloure el codi definit en un altre fitxer dins del codi que es crida.

Suposem que tenim un fitxer anomenat `connectaDb.php` que conté les variables de connexió a la base de dades com el següent:

```php
<?php

$databaseName = 'tdiw';
$databaseUser = 'user';
$databasePassword = 'password';
$databaseHost = '127.0.0.1';
```

I que en el fitxer que fa de model de la nostra aplicació, `model.php`, tenim el següent:

```php
<?php

include __DIR__.'/connectaDb.php';

$action = $_GET['action'] ?? null;
```

El codi que estaria executant realment el fitxer `model.php` seria el següent:

```php
<?php

$databaseName = 'tdiw';
$databaseUser = 'user';
$databasePassword = 'password';
$databaseHost = '127.0.0.1';

$action = $_GET['action'] ?? null;
```

Com podeu veure, el que fan tant les funcions `include` com `require` és afegir al codi que s'està executant el que codi del fitxer que s'inclou.

L'única cosa diferent entre `include` i `require` és el seu comportament en cas de trobar algun error al codi:

- `include` llença un *warning* si no troba el fitxer a incloure, però continua l'execució de l'*script* i llençarà un error si intenta fer ús d'alguna variable definida al codi a incloure.
- `require` llença un *error* si no troba el fitxer a incloure, i avorta l'execució de l'*script*.

També teniu les funcions `include_once`[7] i `require_once`[13]: la seva funcionalitat és la mateixa que les funcions `include` i `require`, però només inclouen el fitxer una vegada, encara que siguin cridades més d'un cop.

Seguint l'exemple anterior, si tinguéssim el mateix fitxer `config.php` i el següent fitxer `index.php`:

```php
<?php

require_once __DIR__.'/connectaDb.php';

$action = $_GET['action'] ?? null;

require_once __DIR__.'/connectaDb.php';
```

El codi que realment s'executaria seria:

```php
<?php

$databaseName = 'tdiw';
$databaseUser = 'user';
$databasePassword = 'password';
$databaseHost = '127.0.0.1';

$action = $_GET['action'] ?? null;
```

De manera anàloga, l'única cosa diferent entre `include_once` i `require_once` és el seu comportament en cas de trobar algun error al codi:

- `include_once` llença un *warning* si no troba el fitxer a incloure, però continua l'execució de l'*script* i llençarà un error si intenta fer ús d'alguna variable definida al codi a incloure.
- `require_once` llença un *error* si no troba el fitxer a incloure, i avorta l'execució de l'*script*.

Penseu quan utilitzar unes funcions o unes altres. Segurament les dades de connexió a la base de dades només les necessitareu incloure una vegada. I si teniu un fitxer que renderitza les dades d'un producte, segurament l'haureu de cridar més d'una vegada.

En qualsevol cas, nosaltres us recomanem fer ús de les funcions `require` i `require_once`, ja que us permetran trobar errors al vostre codi de manera més ràpida i senzilla.

E Base de dades

E.1 Connexió

Per fer les consultes a la base de dades, heu de fer servir pg_connect, que és una funció integrada en PHP per accedir a bases de dades postgreSQL en PHP. Nosaltres farem servir el controlador específic de postgreSQL, però a l'hora d'utilitzar les funcions de la interfície no hi hauria gaires diferències si volguéssiu utilitzar una altra base de dades, com MySQL o SQLite.

Per utilitzar aquesta funció, li passarem com a paràmetres el *Data Source Name* o `DSN` de connexió, l'usuari i la contrasenya d'accés a la base de dades.

Suposant que les dades que tenim són les següents:

- Servidor de la base de dades: `127.0.0.1`
- Nom de la base de dades: `tdiw-db`
- Usuari de la base de dades: `tdiw-user`
- Contrasenya d'accés a la base de dades: `tdiw-password`

La manera d'efectuar la connexió seria la següent:

```
1 <?php
2
3 $conn = pg_connect("host=127.0.0.1 dbname=tdiw-db user=tdiw-user
4 password=tdiw-password");
5
```

La nostra recomanació és que tingueu la connexió de base de dades definida a un fitxer anomenat `/model/connectaDB.php`, que podeu incloure des dels vostres models. Recordeu substituir les dades d'exemple que us mostrem per dades reals.

E.2 Consultes

Per fer consultes a la base de dades amb la funció descrita, el procés és el següent:

1. Es realitza la connexió amb la BBDD
2. Es prepara la consulta.
3. S'executa la consulta.
4. S'agafen els resultats.

Hi ha maneres alternatives de fer les consultes, però aquesta us servirà per a totes les consultes que heu de fer a la vostra pràctica.

E.2.1 Consultes amb múltiples resultats

A la vostra pràctica haureu de fer consultes que retornin múltiples resultats, com el llistat de categories o de productes. Per a fer-ho, php ens proporciona una funció, `pg_fetch_all` que ens permet obtenir tots els resultats de cop, sense haver d'iterar manualment sobre cadascuna de les files.

Suposant que volem agafar totes les categories de la nostra base de dades, podríem executar el següent codi PHP:

```php
<?php

// Pas 1: preparem la consulta.
$sql = 'SELECT id, `name`
FROM category';

// Pas 2: Enviem la query a la BBDD. La variable $conn
// és la definida al pas anterior.
$result = pg_query($conn, $sql);

// Pas 3: agafem els resultats de la consulta.
$categories = pg_fetch_all($result);

```

Depurant amb `print_r`, el valor de la variable `$categories` seria similar al següent:

```
Array
(
    [0] => Array
        (
            [id] => 1
            [0] => 1
            [name] => Filosofia
            [1] => Filosofia
        )

    [1] => Array
        (
            [id] => 3
            [0] => 3
            [name] => Narrativa
            [1] => Narrativa
        )

)
```

Fixeu-vos en l'estructura de dades retornada, es tracta d'un *array* de dues dimensions:

- La primera dimensió conté cada fila de la base de dades.
- La segona dimensió conté cada columna de cada fila de la base de dades, indexades amb números i amb noms de columnes.

Aquesta estructura la podeu fer servir per mostrar el llistat de categories per pantalla, ja que en teniu tota la informació necessària.

En cas que la consulta no retorni cap resultat, el valor de la variable `$categories` seria un *array* buit:

```
Array
(
)
```

E.2.2 Consulta amb un únic resultat

Hi ha casos en què només voldreu obtenir un resultat de la base de dades, com quan demaneu el detall d'un producte o quan un usuari inicia sessió. En aquests casos no té sentit demanar les dades com un *array* de dues dimensions, ja que només ens en cal una. Afortunadament, a PDO existeix una funció que fa això, `pg_fetch_row`, que retorna el següent resultat d'una consulta preparada.

Seguint l'exemple de l'apartat anterior, el seu ús seria així:

```php
<?php

// Pas 1: preparem la consulta.
$sql = 'SELECT id, `name`
FROM category';

// Pas 2: Enviem la query a la BBDD. La variable $conn
// és la definida al pas anterior.
$result = pg_query($conn, $sql);

// Pas 3: agafem els resultats de la consulta.
$categories = pg_fetch_row($result);

```

Depurant amb `print_r`, el valor de la variable `$categories` seria similar al següent:

```
Array
(
    [id] => 1
    [0] => 1
    [name] => Filosofia
    [1] => Filosofia
)
```

Fixeu-vos que, utilitzant `pg_fetch_row`, la consulta ens retorna un *array* d'una dimensió amb cada camp de la fila de la base de dades, indexades amb números i amb noms de columnes.

En cas que la consulta no retorni cap resultat, el valor de la variable `$category` seria `false`.

F == i ===

PHP és un llenguatge de programació feblement tipat, és a dir, les variables d'un tipus concret poden ser tractades com variables d'un altre tipus. Per exemple, podem tenir el següent codi, que és vàlid a PHP però que no ho seria a altres llenguatges fortament tipats, com Java o C++:

```php
<?php

$action = 1; // int
$action = 'llistar-productes'; // string
$action = true; // boolean
$action = []; // array
```

Per comparar els valors de les variables, a PHP existeixen dos operadors de comparacions d'igualtat[4]:

- `==` igualtat, per veure si els valors avaluats són iguals. També existeix el seu invers, `!=`.
- `===` identitat, per veure si els valors i els tipus de les variables són idèntics. També existeix el seu invers, `!==`.

Podeu veure què suposa això en el codi següent:

```php
<?php

var_dump('1' == 1); // true
var_dump('1' === 1); // false
var_dump('0' == 0); // true
var_dump('0' === 0); // false
var_dump(true == 1); // true
var_dump(true === 1); // false
var_dump(false == 0); // true
var_dump(false === 0); // false
var_dump(false == null); // true
var_dump(0 == null); // true
var_dump(false === null); // false
```

A la documentació de PHP[10] podeu consultar-hi una taula amb totes les comparacions de tipus.

Figura 4: Comparativa visual dels valors `0`, a l'esquerra, i `null`, a la dreta

Com podeu veure als tres darrers exemples de comparacions, a PHP també existeix el tipus `null`, que és diferent de `false` i de `0`. La diferència semàntica és que un valor `0` o `false` vol dir que l'element existeix, però que no té res; en canvi, un valor `null` vol dir que l'element no existeix. En teniu una analogia visual a la figura 4, que corre per Internet des de fa un temps, i que mostra la diferència entre `0` i `null` amb una imatge d'un dispensador de paper de vàter. El valor del dispensador de l'esquerra seria `0`, i el de la dreta, `null`.

La nostra recomanació és que sempre feu servir el comparador estricte (`===`), i que convertiu els valors prèviament amb *castings*[14]. A PHP, els `castings` es fan de la següent manera:

```php
<?php
$a = 1;
$a = (bool) $a; // $a === false
$b = '0';
$b = (int) $b; // $b === 0
```

Com a última cosa a tenir en compte en relació als tipus de dades a PHP, recordeu que des de la versió 7 de PHP, publicada el desembre de 2015, les funcions de PHP accepten la definició del tipus de dades tant dels paràmetres de les funcions com del valor de retorn. És a dir, podeu tenir una funció com la següent:

```php
<?php

function isValidUser(
    string $username,
    string $password,
    bool $isAdmin,
    int $age
): bool {
// ...
}

```

A més a més, podeu forçar la comprovació de tipus i que PHP llenci un error tant si els tipus de dades dels paràmetres passats a la funció no són els corresponents, com si el tipus de valor de retorn no és correcte. Per fer-ho, cal afegir la directiva següent a l'inici de cadascun dels fitxers PHP que feu servir:

```php
<?php
declare(strict_types=1);
```

És important que ho tingueu present quan cerqueu informació a Internet, ja que molts dels tutorials o manuals que hi trobareu són de versions antigues, sobretot de les versions 5.X. Intenteu sempre cercar informació actualitzada, per exemple fent servir les «Eines de cerca» del cercador de Google per limitar la data de publicació dels enllaços perquè només inclogui els del darrer any. Teniu un exemple de com fer-ho a la figura 5

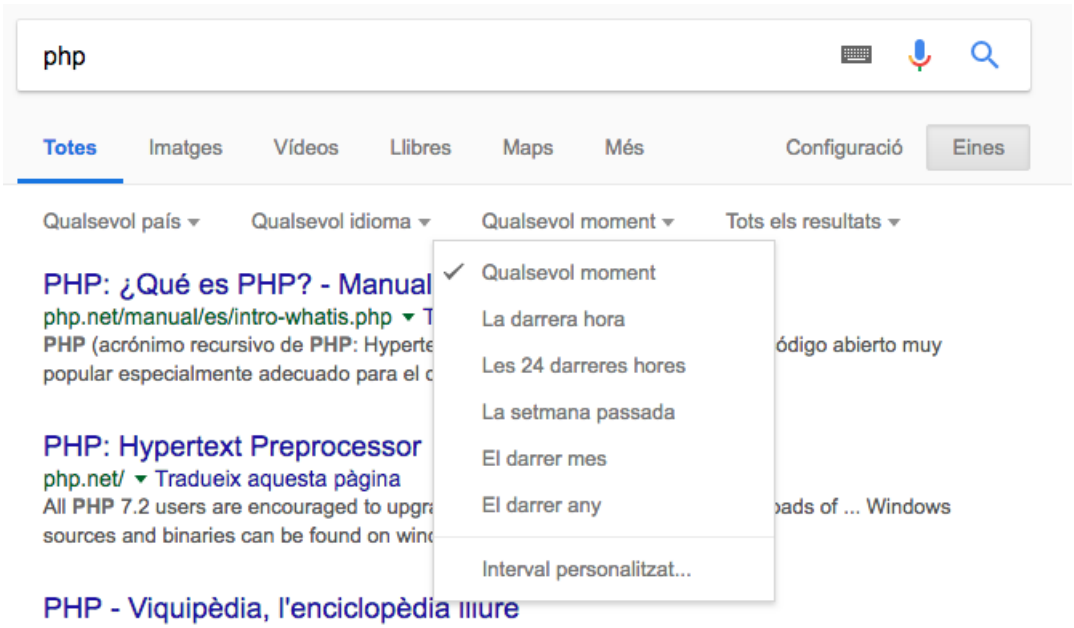

Figura 5: Exemple d'ús de les «Eines de cerca» del cercador de Google

Referències

[1] Live Templates - Help | PhpStorm. https://www.jetbrains.com/help/phpstorm/using-live-templates.html.

[2] Xdebug - Debugger and Profiler Tool for PHP. https://xdebug.org/.

[3] MDN. <pre> - HTML (HyperText Markup Language) | MDN. https://developer.mozilla.org/ca/docs/Web/HTML/Element/pre.

[4] PHP.net. PHP: Comparison Operators. http://php.net/manual/en/language.operators.comparison.php.

[5] PHP.net. PHP: die. http://php.net/manual/en/function.die.php.

[6] PHP.net. PHP: include. http://php.net/manual/en/function.include.php.

[7] PHP.net. PHP: include_once. http://php.net/manual/en/function.include-once.php.

[8] PHP.net. PHP: Magic constants. http://php.net/manual/en/language.constants.predefined.php.

[9] PHP.net. PHP: Manual. https://www.php.net/manual/es/.

[10] PHP.net. PHP: PHP type comparison tables. http://php.net/manual/en/types.comparisons.php.

[11] PHP.net. PHP: print_r. http://php.net/manual/en/function.print-r.php.

[12] PHP.net. PHP: require. http://php.net/manual/en/function.require.php.

[13] PHP.net. PHP: require_once. http://php.net/manual/en/function.require-once.php.

[14] PHP.net. PHP: Type Juggling - Manual. http://php.net/manual/en/language.types.type-juggling.php#language.types.typecasting.

[15] PHP.net. PHP: var_dump. http://php.net/manual/en/function.var-dump.php.

[16] Inc. Zend Technologies. Installation Guide Zend Debugger. https://www.zend.com/downloads/zend-studio-web-debugger.