UF2 - Generació dinàmica de pàgines web

Mecanismes de separació de la lògica de negoci II

Introducció

Context i importància

El desenvolupament d'aplicacions web ha evolucionat cap a estructures més organitzades i escalables.

A mesura que els projectes creixen en complexitat, separar la lògica de negoci de la presentació esdevé fonamental per millorar:

- Manteniments
- Reutilització
- Col·laboracions entre equips.

Introducció

Context i importància

El patró MVC (Model-Vista-Controlador) és una de les arquitectures més utilitzades per aconseguir aquesta separació.

Tanmateix, no és l'única estratègia: l'ús de **serveis i llibreries** ofereix mecanismes complementaris que poden simplificar i organitzar encara més el codi.

Introducció

Objectius de la sessió

- Comprendre com estructurar aplicacions PHP utilitzant el patró MVC.
- Introduir l'ús de serveis i llibreries com a mecanismes complementaris per a la separació de la lògica de negoci.
- Aprendre a diferenciar les responsabilitats de cada component i mecanisme per escriure codi més organitzat i reutilitzable.
- Aplicar exemples pràctics que integren aquestes tècniques en escenaris reals.

Introducció

Conceptes clau

- Model (M): Gestiona la lògica de negoci i el tractament de dades (interacció amb bases de dades, càlculs o aplicació de regles).

- Vista (V): S'encarrega de la presentació de la informació i la interacció amb l'usuari (HTML i interfície).

- Controlador (C): Connecta models i vistes. Processa sol·licituds d'usuari, gestiona la lògica d'aplicació i decideix quina vista mostrar.

- Separació de responsabilitats: El principi fonamental del patró MVC és mantenir cada component enfocat en una responsabilitat específica, millorant l'organització i la col·laboració.

Introducció

Conceptes clau

- Serveis:
  - Encapsulen lògica específica de negoci que pot ser utilitzada per diversos components dins del mateix projecte.
  - Es relacionen estretament amb les necessitats de l'aplicació i ajuden a estructurar funcionalitats complexes de manera modular i reutilitzable.
  - Exemples: calcular el total d'una comanda, gestionar notificacions, o enviar correus electrònics des de diferents mòduls de l'aplicació.

- Llibreries:
  - Agrupen funcions genèriques i independents de l'aplicació, pensades per ser reutilitzades en diversos projectes.
  - Proporcionen eines utilitàries que poden ser usades en diferents contextos sense dependre de la lògica específica d'un projecte concret.
  - Exemples: funcions de validació de formularis, tractament d'arxius o càlcul d'IVA.

Patró MVC: Components i Aplicació

Model (M):

- Responsabilitats principals:
  - Gestionar les dades de l'aplicació i la lògica de negoci.
  - Accedir, manipular i guardar dades en fonts d'emmagatzematge (bases de dades).
  - Aplicar regles i restriccions pròpies de l'aplicació (validacions o càlculs).

- Característiques clau:
  - El model no interactua directament amb l'usuari ni genera interfícies.
  - Proporciona dades al controlador (arrays o objectes).
  - És reutilitzable i independent del controlador o de la vista.

- Exemples:
  - Recuperar dades: Obtenir una llista de productes d'una base de dades i retornar-los al controlador.
  - Guardar dades: Emmagatzemar informació d'una comanda a la base de dades.
  - Aplicar regles de negoci: Determinar si un usuari té dret a un descompte específic.

Patró MVC: Components i Aplicació

Vista (V):

- Responsabilitats principals:
  - Presentar informació a l'usuari de manera clara i atractiva.
  - Mostrar dades que provenen del model (generalment a través del controlador).
  - Acceptar interaccions bàsiques de l'usuari, com fer clics o omplir formularis.

- Característiques clau:
  - La vista no conté lògica de negoci ni interacciona directament amb el model.
  - Pot incloure plantilles dinàmiques per integrar dades del model en HTML.

- Exemples:
  - Llistat de dades: Mostrar una taula amb una llista d'usuaris o productes.
  - Missatges d'error: Mostrar un missatge com "El formulari no és vàlid" si les dades no es poden processar.

Patró MVC: Components i Aplicació

Controlador (C):

- Responsabilitats principals:
  - Actuar com a intermediari entre el model i la vista.
  - Processar les sol·licituds de l'usuari (normalment a través d'una URL o accions en una interfície).
  - Coordinar quines dades s'han d'obtenir del model i quina vista s'ha de mostrar.

- Característiques clau:
  - El controlador no guarda dades ni és responsable del seu emmagatzematge.
  - Està enfocat en processar sol·licituds i coordinar les respostes.

- Exemples:
  - Rebre sol·licituds: Gestionar una petició HTTP POST per crear un nou usuari.
  - Sol·licitar al model que guardi un nou producte en una base de dades.
  - Processar errors
  - Decidir quina vista mostrar

Patró MVC: Components i Aplicació

Flux de Dades en MVC:

- Seqüència típica:
  1. L'usuari realitza una acció (com enviar un formulari o fer clic a un botó).
  2. El controlador processa la sol·licitud i decideix quines operacions s'han de realitzar.
  3. El controlador invoca el model per gestionar les dades necessàries.
  4. El controlador envia el resultat a la vista, que ho presenta a l'usuari.

- Exemple: Llistat de Productes
  1. Usuari: Clica un botó per veure un llistat de productes.
  2. Controlador: Rep la sol·licitud i demana al model que obtingui els productes de la base de dades.
  3. Model: Executa una consulta SQL per recuperar el llistat de productes i retorna les dades.
  4. Vista: Genera una taula HTML amb els productes obtinguts i la mostra a l'usuari.

Patró MVC: Components i Aplicació

Avantatges Addicionals del Patró MVC:

- Organització clara: Cada component té una responsabilitat definida, reduint la complexitat del codi.
- Facilitat de manteniment: És més senzill localitzar i corregir errors gràcies a la separació de funcions.
- Escalabilitat: Permet afegir funcionalitats sense afectar els components ja existents.
- Reutilització: Els models i les vistes poden ser reutilitzats en diferents parts de l'aplicació.
- Col·laboració eficient: Equips diferents poden treballar simultàniament en models, vistes i controladors.

Patró MVC: Exemple pràctic

1. Model: model/user_model.php

```php
<?php
function getUserById($id) {
    $users = [
        1 => ['nom' => 'Anna', 'edat' => 31],
        2 => ['nom' => 'Joan', 'edat' => 26],
    ];
    return isset($users[$id]) ? $users[$id] : null;
}
```

- Gestiona la font de dades de l'aplicació.
- Defineix la funció getUserById($id), que retorna un usuari segons el seu ID.
- El model encapsula les dades i la lògica per obtenir-les.
- Aquesta separació permet reutilitzar-lo en altres parts de l'aplicació sense dependre de la vista o el controlador.

Patró MVC: Exemple pràctic

2. Controlador: controller/user_controller.php

```
<?php
include_once 'model/user_model.php';
function handleUserRequest($id) {
    $user = getUserById($id);
    include 'view/user_view.php';
}
```

- Actua com a intermediari entre la sol·licitud de l'usuari (entrada) i el model/vista.
- Processa la sol·licitud d'un perfil d'usuari amb la funció handleUserRequest($id):
    1. Inclou el fitxer del model per accedir a getUserById($id).
    2. Obté l'usuari corresponent amb la funció del model.
    3. Passa les dades obtingudes del model a la vista (view/user_view.php).
- Manté la lògica separada de la presentació.
- Decideix quina vista utilitzar i amb quines dades, segons la sol·licitud.

Patró MVC: Exemple pràctic

3. Vista: view/user_view.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Perfil d'Usuari</title>
</head>
<body>
    <?php if ($user): ?>
        <h1>Perfil de <?= htmlspecialchars($user['nom']) ?></h1>
        <p>Edat: <?= htmlspecialchars($user['edat']) ?></p>
    <?php else: ?>
        <p>L'usuari no existeix.</p>
    <?php endif; ?>
</body>
</html>
```

Patró MVC: Exemple pràctic

3. Vista: view/user_view.php

Què fa?

- Genera la interfície d'usuari.
- Mostra la informació passada pel controlador:
  - Si l'usuari existeix, mostra el nom i l'edat.
  - Si no, mostra el missatge que "L'usuari no existeix."

Per què és important?

- Separa completament la lògica de negoci (model/controlador) de la interfície gràfica.
- Permet personalitzar la presentació sense alterar el model o el controlador.

Patró MVC: Exemple pràctic

4. Punt d'Entrada: index.php

```
<?php
include_once 'controller/user_controller.php';
// Verifica si s'ha passat un ID d'usuari mitjançant la URL
$id = isset($_GET['id']) ? intval($_GET['id']) : null;
// Gestiona la sol·licitud
handleUserRequest($id);
```

- Actua com el punt central de l'aplicació
    1. Inclou el controlador (controller/user_controller.php).
    2. Recull l'ID de l'usuari de la URL ($_GET['id']).
    3. Crida al controlador perquè gestioni la sol·licitud (handleUserRequest($id)).

Patró MVC: Exemple pràctic

4. Punt d'Entrada: index.php

```
<?php
include_once 'controller/user_controller.php';
// Verifica si s'ha passat un ID d'usuari mitjançant la URL
$id = isset($_GET['id']) ? intval($_GET['id']) : null;
// Gestiona la sol·licitud
handleUserRequest($id);
```

- Exemple de sol·licituds possibles:
  - http://localhost/projecte-mvc/index.php?id=1: Mostra el perfil de l'usuari amb ID 1.
  - http://localhost/projecte-mvc/index.php?id=2: Mostra el perfil de l'usuari amb ID 2.
  - http://localhost/projecte-mvc/index.php: Mostra el missatge que “L'usuari no existeix.” perquè no s'ha passat cap ID.

- Centralitza la gestió de sol·licituds, permetent que totes les peticions passin per un sol punt abans d’arribar al controlador.

Patró MVC: Exemple pràctic

5. Estructura del Projecte

La separació en carpetes reflecteix les responsabilitats de cada component:

projecte-mvc/

```
├── controller/       # Conté el controlador (gestió de sol·licituds).
│   └── user_controller.php
├── model/            # Conté el model (gestió de dades).
│   └── user_model.php
├── view/             # Conté la vista (interfície d'usuari).
│   └── user_view.php
└── index.php         # Punt d'entrada principal.
```

Patró MVC: Exemple pràctic

6. Explicació del Flux

Exemple: Sol·licitud per id=1

1. L'usuari accedeix a index.php?id=1.
2. El fitxer index.php recull l'ID i crida al controlador (handleUserRequest(1)).

   ['nom' => 'Anna', 'edat' => 31]
3. El controlador crida al model (getUserById(1))
4. El controlador carrega la vista (user_view.php) i passa les dades de l'usuari.
5. La vista mostra la pàgina amb el perfil d'Anna.

Serveis

Què són?

- Els serveis encapsulen una **part de la lògica de negoci** específica d'una aplicació en un mòdul reutilitzable.
- Normalment es fan servir per realitzar tasques concretes que poden ser utilitzades per diferents parts de l'aplicació.

Quan utilitzar-los?

- Quan tens una funcionalitat relacionada amb l'aplicació, però que ha de ser reutilitzada per diversos controladors o mòduls.
- Exemple:
  - Interactuar amb una API externa.
  - Processar pagaments.
  - Gestionar l'enviament de correus electrònics.

Serveis: Exemple pràctic

Servei d’enviament de correus

Tens una aplicació que ha de gestionar l'enviament de correus electrònics en diferents situacions:

1. Quan un usuari es registra (confirmació de compte).
2. Quan un usuari sol·licita recuperar la meva contrasenya.
3. Quan es vol enviar una notificació automàtica sobre una activitat.

En lloc de duplicar la funcionalitat en diverses parts del projecte, encapsulem la lògica en un **servei d’enviament de correus**.

Serveis: Exemple pràctic

Servei: servei_email.php

```
function enviarEmail($destinatari, $assumpte, $missatge) {

    // Simulació d'enviament de correu

    $headers = "From: noreply@aplicacio.com\r\n";
    $headers .= "Content-Type: text/plain; charset=UTF-8\r\n";

    // Simulem que el correu s'envia

    echo "S'ha enviat un correu a $destinatari amb l'assumpte '$assumpte'.\n";
    echo "Missatge: $missatge\n";

        // Retornem true per indicar que l'enviament ha estat correcte
    return true;
}
```

Serveis: Exemple pràctic

Ús en diferents parts de l'aplicació:

1. Quan un usuari es registra:

```
// Incloem el servei

include 'servei_email.php';

// Exemple d'ús

$destinatari = "usuari@example.com";
$assumpte = "Confirmació de registre";
$missatge = "Gràcies per registrar-te a la nostra aplicació! Fes clic al següent enllaç per confirmar el teu compte.";

enviarEmail($destinatari, $assumpte, $missatge);
```

Serveis: Exemple pràctic

Ús en diferents parts de l'aplicació:

2. Quan un usuari sol·licita recuperar la seva contrasenya:

```
// Incloem el servei

include 'servei_email.php';

// Exemple d'ús

$destinatari = "usuari@example.com";
$assumpte = "Recuperació de contrasenya";
$missatge = "Fes clic al següent enllaç per restablir la teva contrasenya";


enviarEmail($destinatari, $assumpte, $missatge);
```

Serveis: Exemple pràctic

Ús en diferents parts de l'aplicació:

3. Per enviar una notificació automàtica sobre una activitat:

```
// Incloem el servei

include 'servei_email.php';

// Exemple d'ús

$destinatari = "usuari@example.com";
$assumpte = "Nova activitat al teu compte";
$missatge = "Hola! Hi ha hagut una nova activitat al teu compte. Visita l'aplicació per a més informació.";

enviarEmail($destinatari, $assumpte, $missatge);
```

Serveis: Exemple pràctic

Punts clau:

- El servei **enviarEmail()** encapsula la funcionalitat d'enviament de correus.
- Pot ser cridat des de diferents parts de l'aplicació, evitant la duplicació de codi.
- Si en un futur es decideix canviar la implementació de l'enviament de correus, només caldrà modificar el servei, no les parts que el fan servir.

Llibreries

Què són?

- Una col·lecció de funcions genèriques i independents que es poden utilitzar en diverses parts d'un projecte o fins i tot en projectes diferents.
- Les llibreries no tenen dependències específiques amb l'aplicació. Estan dissenyades per ser flexibles i reutilitzables en qualsevol context.

Quan utilitzar-los?

- Quan necessites implementar funcionalitats genèriques, com:
  - Validació de dades.
  - Càlculs matemàtics.
  - Manipulació de textos o dates.
- Quan la funcionalitat **no està directament lligada a la lògica de negoci** de l'aplicació.

Llibreries: Exemple pràctic

Llibreria de càlcul d'IVA i conversió de moneda:

Context: Necessitem calcular l'IVA d'un producte i convertir preus entre diferents monedes. Aquestes funcionalitats són genèriques i es poden utilitzar en diversos projectes.

```
// Llibreria: utilitats.php

function calcularIVA($preu, $percentatgeIVA) {
    return $preu * ($percentatgeIVA / 100);
}

function convertirMoneda($preu, $taxaConversio) {
    return $preu * $taxaConversio;
}
```

Llibreries: Exemple pràctic

Ús en l'aplicació::

```
// Incloem la llibreria
include 'utilitats.php';

// Exemple d'ús
$preu = 100;
$percentatgeIVA = 21;
$taxaConversio = 1.1;

// Càlcul de l'IVA
$iva = calcularIVA($preu, $percentatgeIVA);
echo "IVA: $iva €\n";


// Conversió de moneda
$preuConvertit = convertirMoneda($preu, $taxaConversio);
echo "Preu en altra moneda: $preuConvertit €\n";
```

Llibreries: Exemple pràctic

Punts clau:

- Les funcions són totalment independents de l'aplicació.
- Es poden reutilitzar en qualsevol projecte.

Llibreries vs Serveis

Comparativa:

Característica	Llibreries	Serveis
Objectiu	Funcions genèriques i reutilitzables.	Encapsular lògica específica de negoci.
Relació amb l'aplicació	Independents de l'aplicació.	Estretament relacionats amb l'aplicació.
Exemple funcional	calcularIVA() o convertirMoneda().	calcularTotalComanda() per diverses parts del projecte.
Reutilització	Entre diversos projectes.	Dins del mateix projecte, en diversos mòduls.