UF2 - Generació dinàmica de pàgines web

Definició i característiques principals

Què és la generació dinàmica de pàgines web?

- Consisteix en la creació de contingut personalitzat i adaptat al moment d'accés per part d'un usuari o en funció de dades actualitzades o esdeveniments en temps real.
- A diferència de les pàgines estàtiques (HTML pur), les dinàmiques utilitzen llenguatges com PHP o Python per generar contingut en temps real.

Característiques principals:

- Interacció amb bases de dades.
- Capacitat d'adaptar el contingut a l'usuari.
- Processament de formularis i entrada d'usuari.
- Integració amb altres serveis com API o plataformes de tercers.

Avantatges principals

Contingut actualitzat en temps real:

- Permeten que el contingut es generi automàticament en funció de dades actualitzades o esdeveniments en temps real.

- Especialment útil en casos on la informació canvia constantment, com botigues en línia, notícies, xarxes socials o previsions meteorològiques.

- Exemples:
  - Un lloc web de comerç electrònic mostra l'estoc disponible en temps real per evitar vendes de productes esgotats.
  - Una pàgina de notícies mostra titulars actualitzats des d'una base de dades, evitant la necessitat d'editar manualment el contingut.

Avantatges principals

Personalització:

- Una pàgina dinàmica pot personalitzar-se segons les necessitats i preferències de cada usuari
- La personalització es pot aconseguir utilitzant informació com la seva ubicació, historial de compres o interessos.
- Exemples pràctics:
  - Pàgina de recomanacions personalitzades:
    - ➢ Amazon mostra recomanacions basades en l'historial de compres o cerques de l'usuari
  - Contingut personalitzat per ubicació:
    - ➢ Una pàgina meteorològica mostra el temps local en funció de l'IP de l'usuari o la seva ubicació.

Avantatges principals

Interacció amb l'usuari:

- Permet interactuar directament amb els usuaris, captant dades en temps real i mostrant resultats personalitzats.
- El contingut es pot modificar segons les accions de l'usuari, com omplir un formulari, seleccionar una opció o fer clic a un botó.
- Exemples pràctics:
  - Formularis interactius:
    - ➢ Un formulari de subscripció que verifica si el correu ja està registrat abans d'acceptar-lo.
  - Actualització de contingut en funció d'una acció:
    - ➢ Mostrar resultats en una pàgina de cerca.

Avantatges principals

Integració amb altres aplicacions i serveis:

- Permeten interactuar amb altres sistemes, serveis o aplicacions mitjançant APIs (Interfícies de Programació d'Aplicacions).
- Això les fa capaces d'obtenir, enviar o processar informació d'una manera fluida i automatitzada.
- Exemples pràctics:
  - Integració amb xarxes socials:
    - ➢ Un lloc web pot mostrar els últims tweets o publicacions d'Instagram d'una empresa.
  - Pagaments amb serveis com PayPal o Stripe:
    - ➢ Una botiga en línia pot processar pagaments de manera segura utilitzant APIs de passarel·les de pagament.
  - Mapes interactius:
    - ➢ Mostrar una ubicació utilitzant l'API de Google Maps.

Com funciona la generació dinàmica?

Flux bàsic d'una pàgina dinàmica

- Pas 1: L'usuari sol·licita una pàgina web.

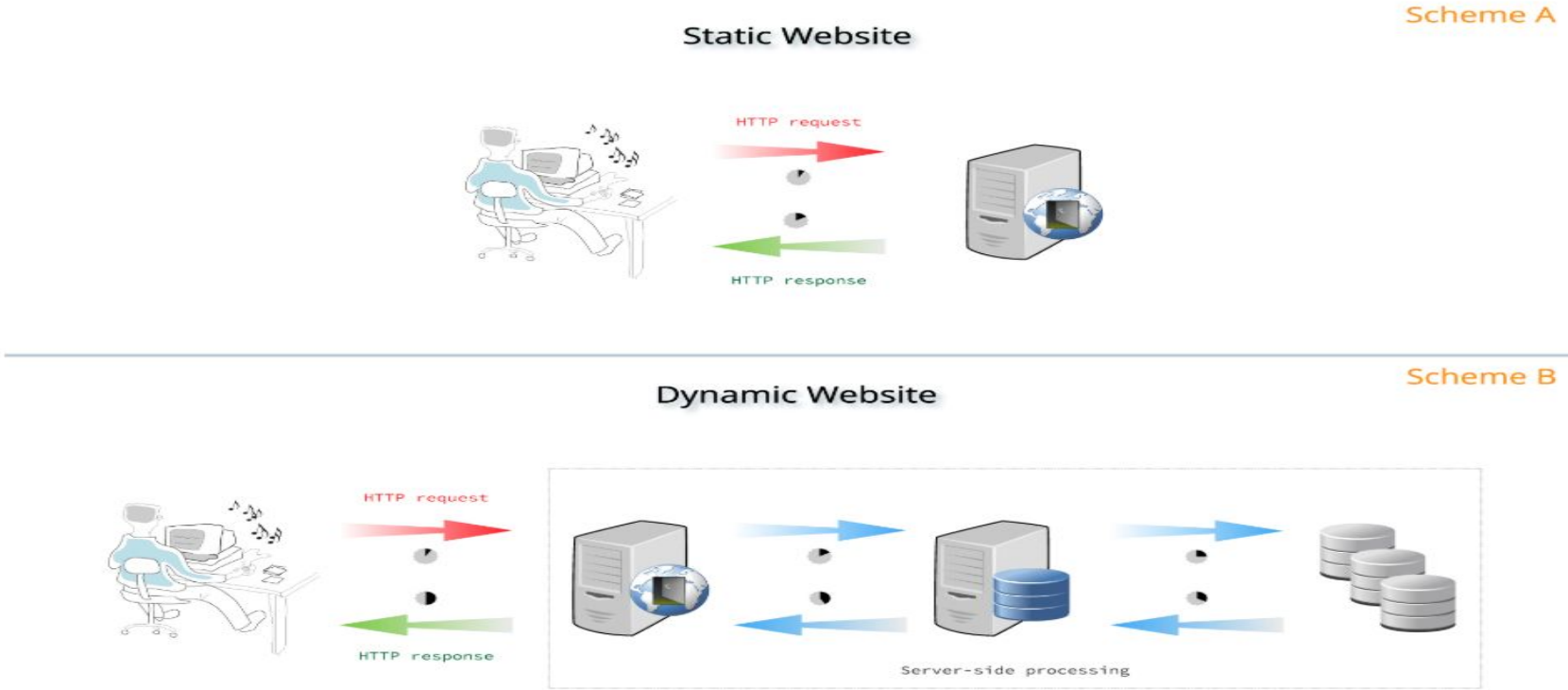
- Pas 2: El servidor processa la sol·licitud:
  - Recol·lecta dades d'una base de dades o d'altres fonts.
  - Processa la lògica de negoci (càlculs, filtres, regles).
  - Genera el contingut en format HTML dinàmicament.

- Pas 3: El servidor envia el contingut generat al navegador de l'usuari.

- Pas 4: El navegador mostra la pàgina final.

Com funciona la generació dinàmica?

Flux bàsic d'una pàgina dinàmica



Mecanismes de separació de la lògica de negoci

Introducció

Context i importància

Per què separar la lògica de negoci en una aplicació web?

En una aplicació web es combinen tres components principals:

- Interfície (HTML, CSS, JS) → El que veu l'usuari.
- Lògica de negoci → Les regles que defineixen el que fa l'aplicació.
- Emmagatzematge de dades (Base de dades).

Problemes comuns del codi sense separació:

- Barreja de codi HTML amb lògica → Fa que sigui difícil mantenir i depurar.
- Falta de reutilització → Cada pàgina es crea des de zero.
- Complicacions per treballar en equip → Desenvolupadors front-end i back-end interfereixen.

Introducció

Context i importància

Per què és important la separació de responsabilitats?

Avantatges principals de la separació de la lògica de negoci:

- Manteniment fàcil: Cada part del codi s'actualitza per separat.
- Reutilització del codi: La lògica es pot reutilitzar en altres parts del projecte.
- Col·laboració més fluida: Front-end i back-end treballen en paral·lel.
- Millor escalabilidad: Fàcil afegir noves funcionalitats.

Introducció

Objectius de la sessió

- Comprendre què és la lògica de negoci
- Com aquesta lògica de negoci es relaciona amb la resta de l'aplicació web.
- Reconèixer els problemes de barrejar codi i com evitar-los.
- Introduir el patró MVC com a solució per estructurar el codi.
- Analitzar casos pràctics senzills que mostren la separació de la lògica i la presentació.

Introducció

Conceptes clau

- Què és la lògica de negoci?
  - La lògica de negoci és el conjunt de reglas, processos i càlculs que defineixen el que fa l'aplicació.
  - Són exemples:
    - ➢ Si el client ha comprat més de 100 €, aplica un 10% de descompte.
    - ➢ Només els usuaris administradors poden accedir a certs continguts.

- On es troba la lògica de negoci?
  - Normalment, en el servidor (per exemple, en fitxers PHP, Python, Java, etc.).

- Exemple de regla de negoci:

```
function calculaDescompte($total) {
  return $total > 100 ? 10 : 0;
}
```

Introducció

Conceptes clau

- Codi barrejat:
  - HTML, CSS i lògica junts en el mateix fitxer → Difícil de mantenir.
  - Exemple senzill d'un fragment de codi barrejat (HTML + PHP).

- Codi separat:
  - Cada responsabilitat en un fitxer o mòdul diferent.
  - Introducció bàsica al concepte de Model, Vista i Controlador (MVC) → Desenvolupat més endavant.

Lògica de negoci

Lògica de negoci

1. Què és la lògica de negoci?

- Definició:
  - És el conjunt de regles, càlculs i processos que governen el funcionament d'una aplicació.
  - Decideix què cal fer en base a unes regles preestablertes.
  - Separa les operacions funcionals de la interfície visual.
  - Determina com els usuaris interactuen amb el sistema i quines respostes es generen.
  - Per exemple, una botiga online:
    - Regla: "Si el total de la compra supera els 100 €, aplica un descompte del 10%."
    - Processos: Càlcul de preus, gestió d'inventari, validació de pagaments.

Lògica de negoci

1. Què és la lògica de negoci?

- Per què és important?
  - Centralitza el funcionament principal de l'aplicació en un lloc específic.
  - Permet mantenir i actualitzar fàcilment les regles del negoci sense afectar altres parts del sistema.

- Exemples:
  - Regla de descompte en una compra: Si el client compra més de 100 €, aplica un 10% de descompte.
    - Pseudocodi:

```
if totalCompra > 100:
    descompte = totalCompra * 0.10
```

  - Control d'accés: Permetre que només certs rols (ex. administrador) puguin accedir a continguts protegits.

Lògica de negoci

2. Per què està en el servidor?

La lògica de negoci se sol implementar al servidor per diverses raons fonamentals, que milloren tant la seguretat com la funcionalitat de l'aplicació. Aquí tens un detall dels motius principals:

- Seguretat
    - Protecció contra manipulacions: Si les regles de negoci (descomptes, permisos d'accés, càlculs crítics) estiguessin al client (per exemple, en JavaScript), els usuaris podrien modificar-les fàcilment manipulant el codi.
        - Exemple: Un descompte aplicat incorrectament per un usuari que modifica un paràmetre en una sol·licitud HTTP.

    - Control centralitzat: Al servidor, les regles es poden controlar i actualitzar sense exposar-les públicament. Això minimitza el risc d'abusos o errors.

Lògica de negoci

2. Per què està en el servidor?

- Reutilització:
  - Multi-plataforma: Quan la lògica de negoci està al servidor, pot ser compartida entre diferents interfícies d'usuari:
    - ➢ Aplicacions web.
    - ➢ Aplicacions mòbils.
    - ➢ APIs per a serveis externs.
    - ➢ Exemple: Un sistema de gestió de comandes que s'utilitza tant des d'una pàgina web com des d'una app mòbil pot fer servir la mateixa lògica de càlcul d'impostos o gestió d'inventari.
  - Evolutius: Quan la lògica s'actualitza al servidor, totes les interfícies d'usuari es beneficien del canvi automàticament.

Lògica de negoci

2. Per què està en el servidor?

- Capacitat de processament:
  - Execució més potent: Els servidors tenen més capacitat per executar càlculs complexos o gestionar grans volums de dades, cosa que seria problemàtica al client, especialment en dispositius mòbils o amb menys recursos.
  - Gestió d'operacions crítiques: Operacions com accedir a bases de dades, validar credencials, o executar processos massius s'executen al servidor

- Consistència dels resultats
  - Unificació de regles: Les regles aplicades al servidor garanteixen que tothom rebi el mateix resultat davant d'una mateixa sol·licitud. Això elimina discrepàncies entre el que pot fer un client o un altre.
    - ➢ Exemple: Els preus d'una botiga online no poden variar perquè un usuari tingui una versió antiga del codi al client.

Lògica de negoci

3. Relació amb la resta de l'aplicació

- Relació amb les altres capes:
  - Vista
    - ➢ Rep les accions de l'usuari (formularis, clics, etc.)
    - ➢ Mostra resultats provinents de la lògica de negoci (p.ex., "Descompte aplicat: 10%")
  - Controlador
    - ➢ Fa de pont entre la Vista i la Lògica de Negoci.
    - ➢ Envia dades de l'usuari a la Lògica de Negoci i rep el resultat per retornar-lo a la Vista.
  - Base de dades (Model):
    - ➢ La lògica de negoci consulta i actualitza les dades necessàries per aplicar les regles.

Lògica de negoci

3. Relació amb la resta de l'aplicació

- Exemples pràctics de flux:
  - Cas senzill: Carret de compra amb descompte:
    1. L'usuari introdueix un producte al carret (Vista).
    2. El controlador gestiona aquesta sol·licitud i passa el total de la compra a la Lògica de Negoci.
    3. La Lògica de Negoci aplica un descompte si cal i actualitza la Base de Dades.
    4. El controlador envia el total actualitzat a la Vista per mostrar-ho a l'usuari.

Lògica de negoci

3. Relació amb la resta de l'aplicació

- Exemples visualitzat en codi:

```
// Vista: L'usuari envia la sol·licitud
<form action="controlador.php" method="post">
    <input type="text" name="product" value="Tassa">
    <input type="submit" value="Afegir"></form>
// Controlador: Processa la sol·licitud
$producte = $_POST['producte'];
$preuFinal = aplicarDescompte($producte);
// Lògica de Negoci: Aplica les reglas
function aplicarDescompte($producte) {
    $preu = obtenirPreu($producte);
    return ($preu > 100) ? $preu * 0.90 : $preu;
}
```

Codi barrejat vs. codi separat

Codi barrejat vs. codi separat

1. Què és el codi barrejat?

- Definició: Codi on diferents responsabilitats (lògica de negoci, interfície d'usuari, accés a dades) estan barrejades en els mateixos fitxers.
  - Exemple senzill: Una aplicació PHP que inclou en un mateix fitxer HTML, CSS i càlculs de lògica.

```
<h1>Compra</h1>
<p>Total: <?php echo $total; ?></p>
<?php
if ($total > 100) {
  echo "<p>Descompte aplicat: 10%</p>";
} else {
  echo "<p>No hi ha descompte.</p>";
}
?>
```

Codi barrejat vs. codi separat

2. Problemes del codi barrejat:

- Dificultat per mantenir-lo: Afegir noves funcionalitats o corregir errors és complicat, ja que qualsevol canvi pot afectar altres parts del codi.
- Escalabilitat limitada: Difícil adaptar l'aplicació a nous requeriments o ampliar funcionalitats.
- Baixa reutilització: Fragments de codi difícilment reutilitzables en altres projectes.
- Augment de riscos: Barrejar HTML amb càlculs de negoci pot introduir vulnerabilitats, com l'injecció de codi (XSS, SQL injection).

3. Codi separat: Avantatges

- Modularitat i manteniment: Cada component té la seva responsabilitat definida.
- Reutilització: Fragments de codi reutilitzables en altres aplicacions o projectes.
- Col·laboració: Facilita el treball en equip, ja que desenvolupadors poden treballar en diferents components alhora.

Introducció al patró MVC

Introducció al patró MVC

1. Per què necessitem MVC?

- Separar responsabilitats ajuda a fer el codi més net, escalable i fàcil de mantenir.
- MVC és un patró utilitzat àmpliament en desenvolupament web per gestionar aquesta separació.

2. Components del patró MVC:

- Model:
  - Responsable de la lògica de negoci i la gestió de dades.
  - Interactua amb bases de dades o altres fonts de dades.
  - Exemple: Una funció per calcular un descompte o per obtenir articles d'una base de dades.

```
function calcularDescompte($total) {
  return $total > 100 ? 10 : 0;
}
```

Introducció al patró MVC

2. Components del patró MVC:

- Vista:
  - Genera el contingut que es mostra a l'usuari (HTML, CSS).
  - No conté lògica de negoci, només la presentació.

```
<h1>Benvingut</h1>
<p>Total de la compra: <?= $total ?> €</p>
<p>Descompte aplicat: <?= $descompte ?>%</p>
```

- Controlador:
  - Gestiona les sol·licituds de l'usuari.
  - Coordina les accions entre el Model i la Vista.

Introducció al patró MVC

3. Flux de dades en MVC:

- L'usuari realitza una acció (per exemple, clicar un botó).
- El controlador processa la sol·licitud i interactua amb el Model.
- El Model retorna les dades al controlador.
- El controlador passa les dades a la Vista.
- La Vista genera el contingut que es mostra a l'usuari.

4. Beneficis del patró MVC:

- Organització: Clarifica les responsabilitats de cada component.
- Escalabilitat: Facilita l'addició de noves funcionalitats.
- Reutilització: Les Vistes i Models es poden reutilitzar fàcilment.