ETP Xavier	MÒDUL 0613 – Desenvolupament web en entorn servidor			
	De l'HTML estàtic al servidor dinàmic: llenguatges, motors i frameworks			
	DATA	22/09/2025	GRUP	2n
	NOM	Sana Rut Sabir París		

Exercici 1 — Mecanismes d'execució al servidor web

Objectiu: entendre diferència estàtic/dinàmic i el paper del servidor web.

A la sessió anterior ja vam veure que no tots els fitxers web es processen de la mateixa manera. Ara volem que distingiu clarament què fa el servidor amb cada tipus de fitxer (HTML, CSS, JS, PHP…) i com arriba la resposta final al navegador.

Enunciat 1. Crea tres fitxers al

servidor local:

- index.html → pàgina amb un text i un enllaç a data.php.

AC2-Sana > index.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Això es un test</title>
    <a href="data.php">Accedir a data</a>
</head>
<body>
    <p>L'entrega s'allarga fins diumenge.</p>
</body>
</html>
```

- data.php → mostra la data actual i el teu nom amb echo.

AC2-Sana > index.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Això es un test</title>

</head>
<body>
    <p>L'entrega s'allarga fins diumenge.</p>
    <a href="data.php">Accedir a data</a>
</body>
</html>
```

- script.js → mostra a l'alerta "Hola des de JavaScript".

```
JS script.js
htdocs > a1 > JS script.js
1   window.alert("Hola des de JavaScript.");
```

AVÍS: Per tal de que tingui sentit, he de vincular l'arxiu JS a HTML. He procedit a fer això:

```
index.html
htdocs > a1 > index.html > html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Això es un test</title>
7
8   </head>
9   <body>
10      <p>L'entrega s'allarga fins diumenge.</p>
11      <a href="data.php">Accedir a data</a>
12      <script src="script.js"></script>
13  </body>
14  </html>
```

I el resultat serà:

2. Obre cada fitxer al navegador i:

- Consulta el codi font (Ctrl+U).

HTML

Ajuste de línea

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Això es un test</title>

</head>
<body>
    <p>L'entrega s'allarga fins diumenge.</p>
    <a href="data.php">Accedir a data</a>
    <script src="script.js"></script>
</body>
</html>
```

PHP

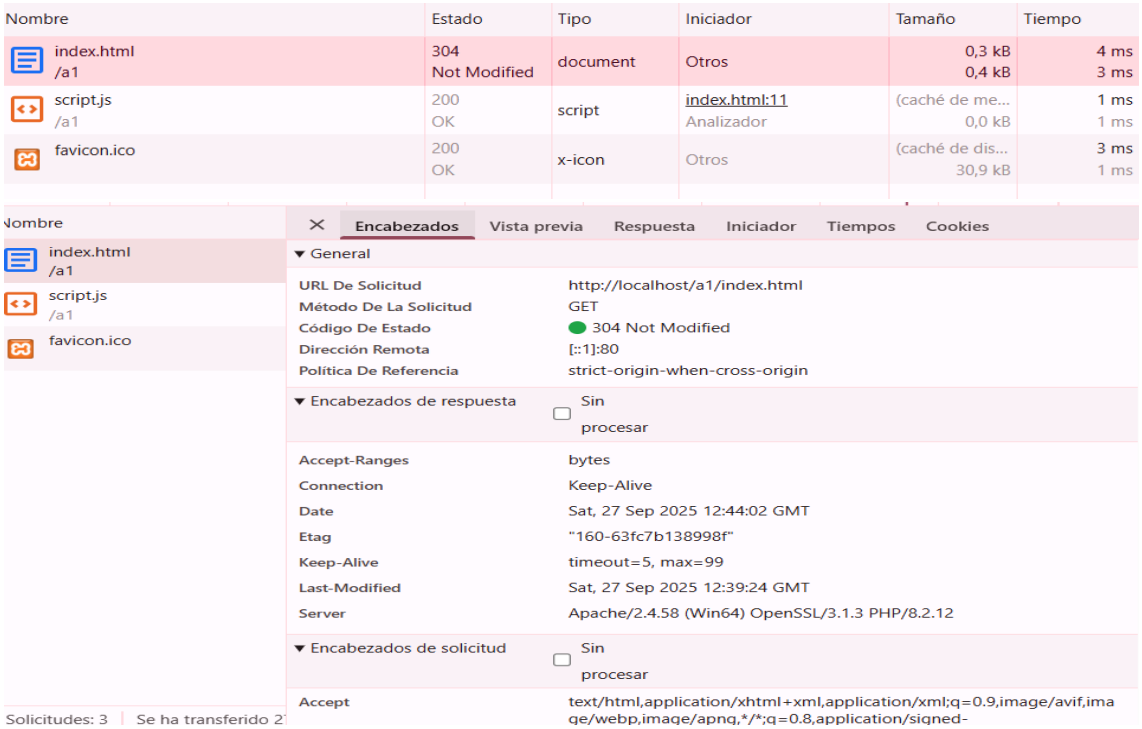
```
Sana, avui és: 22/09/2025
```

JS

```
window.alert("Hola des de JavaScript.");
```

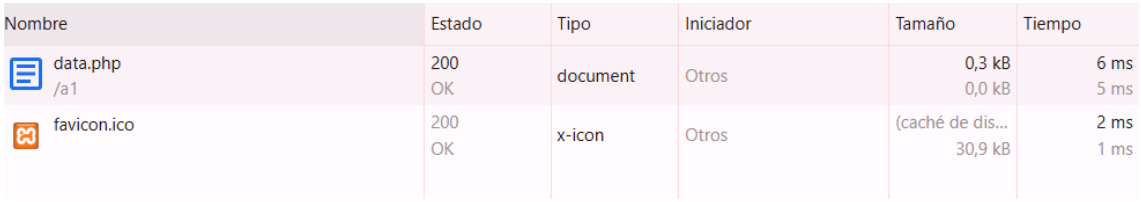
- Consulta la pestanya Network a DevTools.

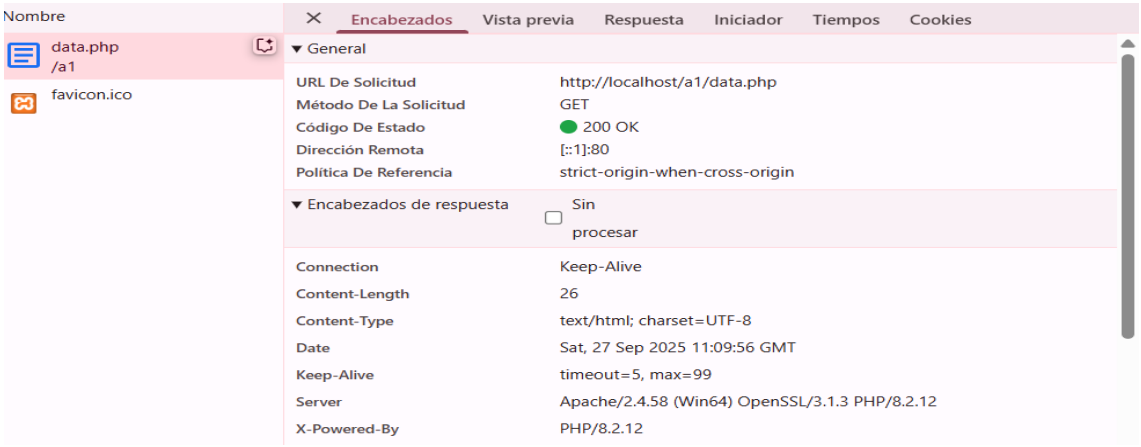
HTML

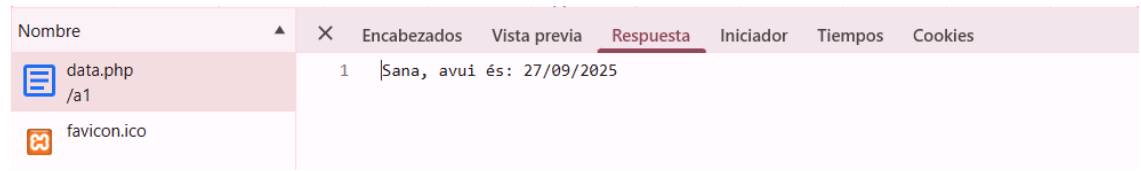



Nombre
index.html
/a1
script.js
/a1
favicon.ico
Encabezados
Vista previa
Respuesta
Iniciador
Tiempos
Cookies
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Això es un test</title>
7
8 </head>
9 <body>
10 L'entrega s'allarga fins diumenge.
11 Accedir a data
12 <script src="script.js"></script>
13 </body>
14 </html>

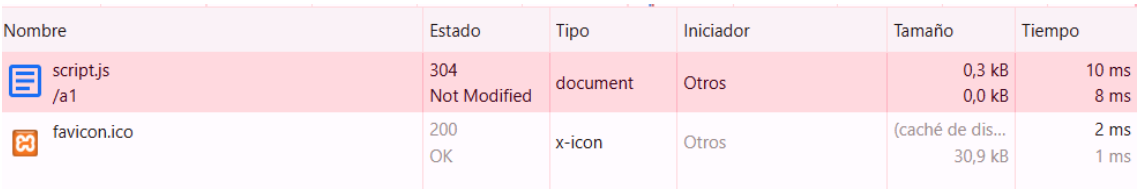
PHP


Nombre
Estado
Tipo
Iniciador
Tamaño
Tiempo
data.php
/a1
200
OK
document
Otros
0,3 kB
0,0 kB
6 ms
5 ms
favicon.ico
200
OK
x-icon
Otros
(caché de dis...
30,9 kB
2 ms
1 ms

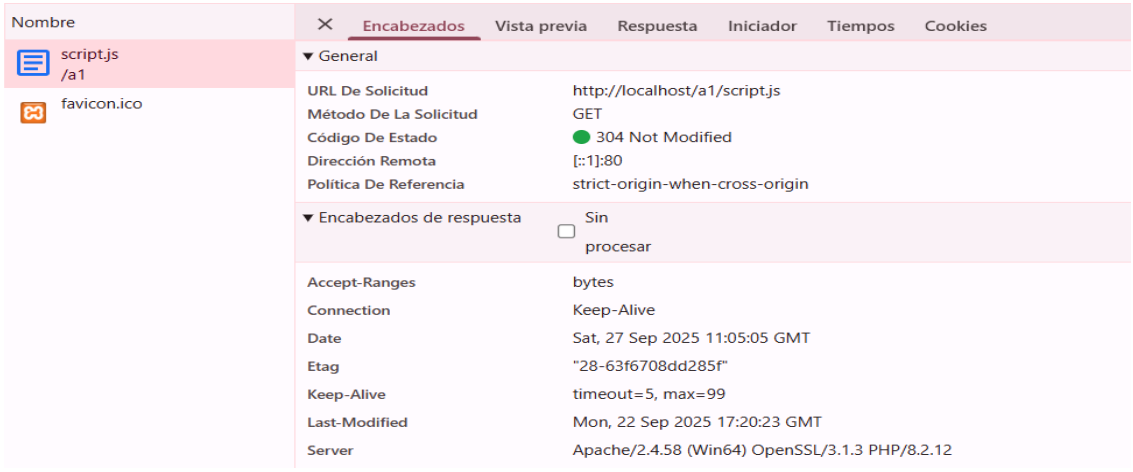

Nombre
data.php
/a1
favicon.ico
Encabezados
Vista previa
Respuesta
Iniciador
Tiempos
Cookies
General
URL De Solicitud
http://localhost/a1/data.php
Método De La Solicitud
GET
Código De Estado
200 OK
Dirección Remota
[::1]:80
Política De Referencia
strict-origin-when-cross-origin
Encabezados de respuesta
Sin procesar
Connection
Keep-Alive
Content-Length
26
Content-Type
text/html; charset=UTF-8
Date
Sat, 27 Sep 2025 11:09:56 GMT
Keep-Alive
timeout=5, max=99
Server
Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12
X-Powered-By
PHP/8.2.12

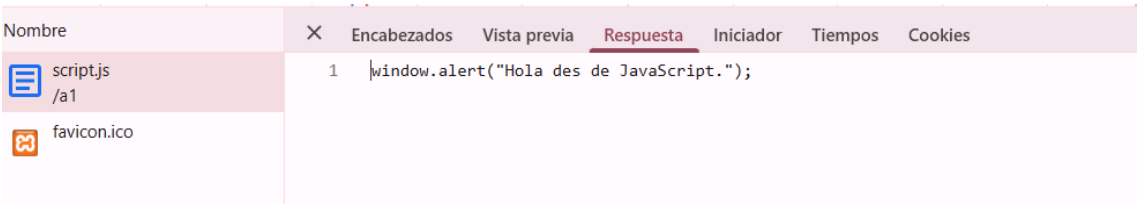

Nombre
data.php
/a1
favicon.ico
Encabezados
Vista previa
Respuesta
Iniciador
Tiempos
Cookies
1 Sana, avui és: 27/09/2025

JS



Nombre	Estado	Tipo	Iniciador	Tamaño	Tiempo
script.js /a1	304 Not Modified	document	Otros	0,3 kB 0,0 kB	10 ms 8 ms
favicon.ico	200 OK	x-icon	Otros	(caché de dis... 30,9 kB	2 ms 1 ms





3. Respon:

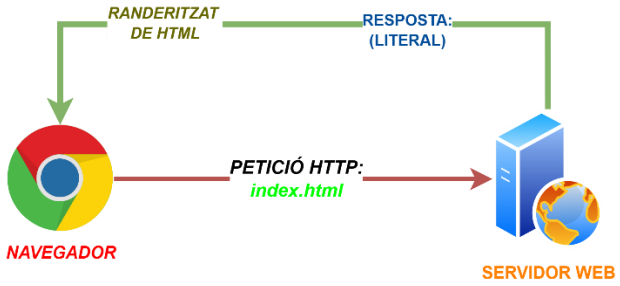
- Quins fitxers ha enviat el servidor sense processar?
  Els fitxers ***index.html*** i ***script.js.***
  Ho sé perquè la resposta d'aquests és literalment el seu codi sencer (ho podem veure al fer el Control+ U o bé amb F12 → Xarxa→Resposta.
  En canvi, amb l'arxiu PHP no passa això, ja que la part "echo" no apareix. És a dir, que sí es processa.

- Quin fitxer ha passat per un motor d'execució?
  Com ja he explicat en el punt anterior, el que sí es processa és el ***data.php*** ,
  ja que no es veu el codi, només el resultat.

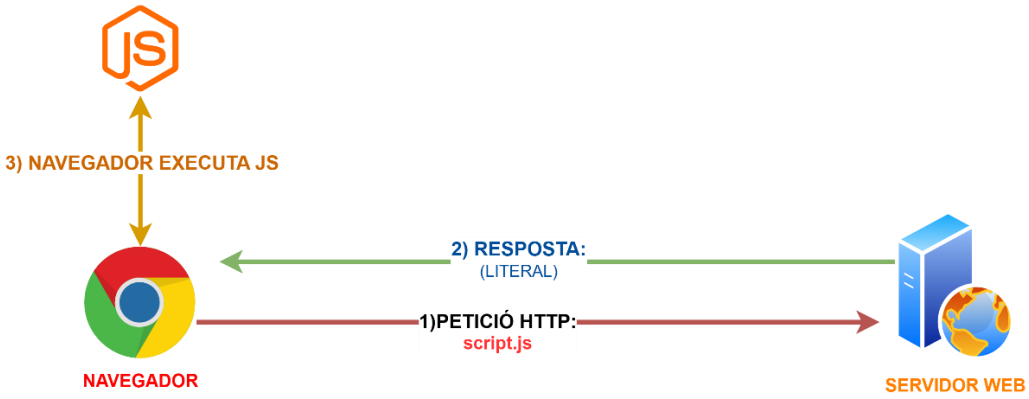
- Quin fitxer ha estat processat pel navegador i no pel servidor?
  El fitxer que és processat pel navegador és el ***script.js*** , ja que si jo trec el *<script src="script.js">* de ***l'index.html***, l'efecte de finestra emergent desapareix. És a dir, que l'execució depèn del navegador totalment, al fer la càrrega del fitxer.

- Explica amb un esquema qui ha fet cadascuna de les tasques (navegador / servidor).

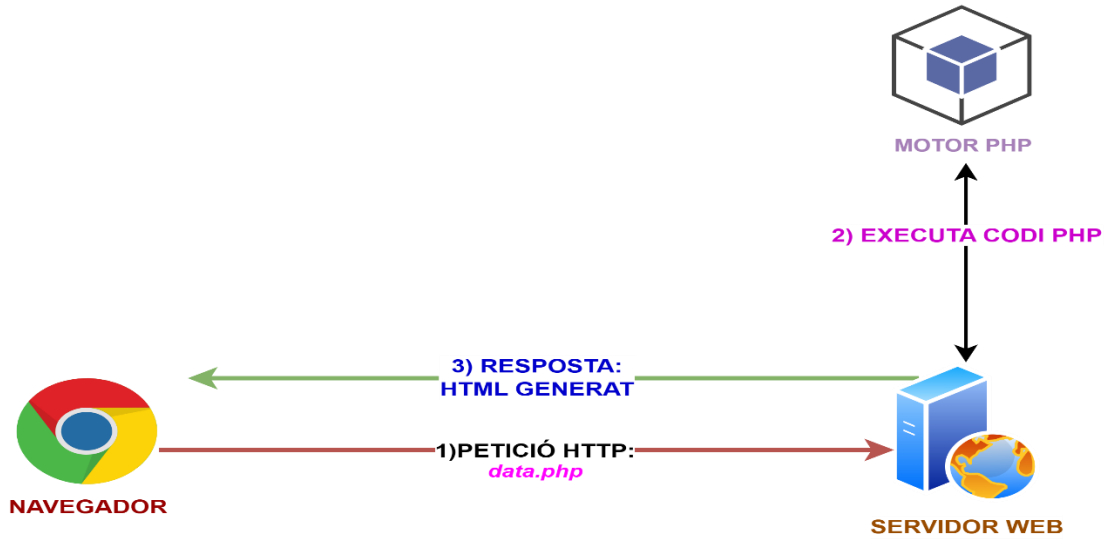
HTML:



JS:



PHP:



NOTA: He fet un diagrama de cadascun per a que sigui més entenedor.

Exercici 2 — Servidor d'aplicacions i integració

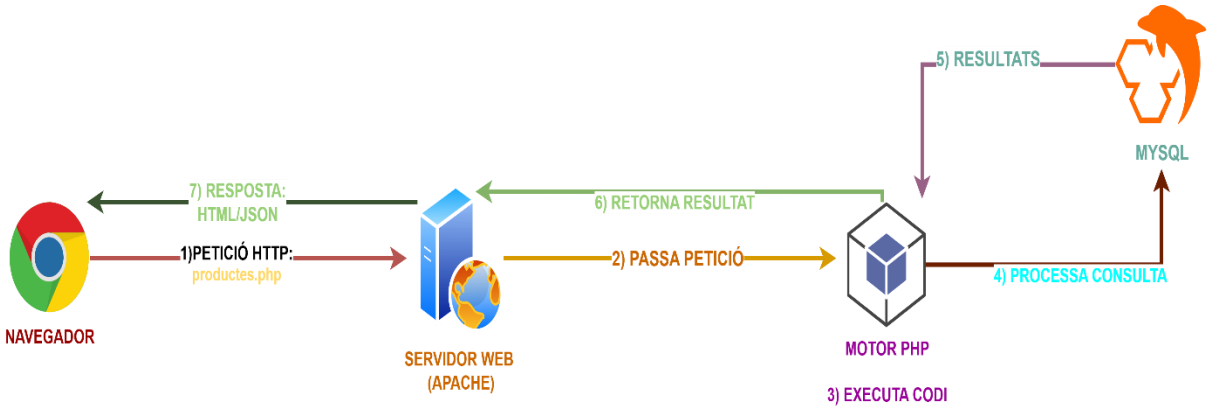
Objectiu: entendre el flux entre servidor web i servidor d'aplicacions.

A l'activitat anterior només vam fer un esquema global del camí d'una petició web. Ara anirem un pas més enllà: volem descriure amb detall què passa dins de cada component (Apache, motor PHP, MySQL) i qui s'encarrega de cada part del procés.

Enunciat 1. Escriu en un full o esquema digital el camí complet d'una

petició:

- Navegador demana productes.php.
- Apache decideix què fer.
- Motor PHP executa el codi.
- PHP fa una consulta a MySQL i obté 3 productes.
- Apache retorna el resultat final al navegador.



2. Escriu tots els passos amb detall:

1) *Navegador envia petició HTTP : /productes.php*

2) *Servidor web (Apache) rep la petició i veu que és .php. →deriva la petició al motor PHP.*

3) *Motor PHP carrega i executa el codi de productes.php.*

4) *Fa una consulta a MYSQL (la base de dades).*

5) *MySQL rep la consulta, l'executa i retorna els resultats.*

6) *El motor PHP rep els resultats, els processa i genera la resposta final. Retorna la sortida generada a Apache .*

7) *Apache envia la resposta HTTP al navegador i aquest mostra resultat final (HTML/JSON)*

- Quin component rep la petició?
  La rep el servidor web (APACHE). Aquest és qui decideix què fer amb la petició (si derivar-ho o no al servidor d'aplicacions).

- Quin component executa el codi?
  És el motor PHP qui executa el codi, per després passar-li la petició a la base de dades (MYSQL) i que aquesta executi la consulta.

- Quin component retorna el resultat al client?
  Resultats MYSQL → motor PHP genera resposta→APACHE (SERVIDOR WEB) envia resposta HTTP(JSON/HTML) a NAVEGADOR (client).
  Per tant, és el servidor web qui donarà el resultat al client, després de tot el procés que he detallat.

3. Afegiu un cas de fitxer mixt: part HTML + part PHP. Explica qui processa què.

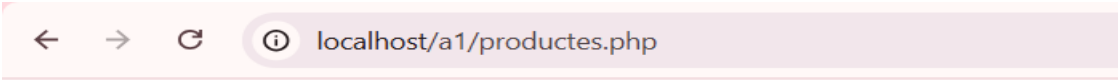
htdocs > a1 > productes.php

```php
<?php
  $productes = ["Poma", "Papaya", "Pressec"];
  $ara = date(format: 'd/m/Y H:i');
?>
<!doctype html>
<html lang="ca">
<head>
  <meta charset="utf-8">
  <title>Productes</title>
</head>
<body>
  <h1>Qué bó!</h1>

  <ul>
    <?php foreach ($productes as $p) { ?>
      <li><?php echo $p; ?></li>
    <?php } ?>
  </ul>

  <p>Generat el: <?php echo $ara; ?></p>
</body>
</html>

```



Qué bó!

- Poma
- Papaya
- Pressec

Generat el: 28/09/2025 00:48

Explicació:

- **Servidor (Apache)→ motor PHP**
  - Executa el codi entre <?php … ?>:
  - Defineix l'array $productes
  - Calcula la data $ara
  - Recorre el foreach i genera els … de la llista.
  - Envia al navegador només l'HTML final (sense codi PHP).
- **Navegador (client)**
  - Renderitza l'HTML rebut:

    veu <h1>, la <ul> amb 3 <li>, i el paràgraf amb la data.

I això com ho puc comprovar? Fàcil. Fent un F12 → Xarxa → Resposta:

Ni rastre de la part PHP. Per tant, això ja m'indica que PHP s'ha executat al servidor, i que el navegador només mostra el resultat.

Exercici 3 — Llenguatges i tecnologies al servidor

Objectiu: conèixer motors d'execució i comparar tecnologies.

Ja sabem que el servidor no pot executar qualsevol codi directament, necessita un motor específic. Aquest exercici us farà pensar en diferents llenguatges de programació i com s'integren amb el servidor web.

Enunciat

1. Fes una taula comparativa amb tres llenguatges: PHP, Python i Node.js.

Adjunto els links:

https://es.wikipedia.org/wiki/PHP
https://es.wikipedia.org/wiki/Python
https://es.wikipedia.org/wiki/Node.js

- Quin motor o entorn executa el codi?
- Quin paper juga Apache o Nginx en cada cas?
- Un cas d'ús real típic (ex: CMS, ciència de dades, apps en temps real).

Llenguatge	Motor / Entorn ex.codi	Apache / Nginx	Cas d'ús típic
PHP	**Zend Engine** (implementació de referència de PHP; el codi PHP s'interpreta al servidor)	Fan de servidor web i passen les URL .php al procés **PHP-FPM**; també serveixen fitxers estàtics.	**CMS / webs clàssiques** (ex.: WordPress, Moodle), CRUD ràpid. (associat a entorns LAMP/XAMPP.)
Python	L'intèrpret de Python a través d'un servidor **WSGI/ASGI** (Gunicorn/uWSGI/Uvicorn)	Sovint de **reverse proxy** davant el servidor WSGI (Gunicorn/uWSGI), gestionant TLS, estàtics..	**Backends amb Django/Flask/FastAPI**, integració amb **dades/IA** (ecosistema Python).
Node.js	El **runtime Node.js,** que usa el **motor V8** per executar JavaScript al servidor (fora del navegador).	Sovint com a **reverse proxy** davant del procés Node (escala, TLS, caché, estàtics).	**Apps en temps real** (xat, websockets), **APIs** i servidors web escalables.

NOTA:

- ***"Passar a PHP-FPM/WSGI"****: el servidor web (Apache/Nginx) rep la petició i la deriva al procés que realment executa el codi (PHP-FPM per a PHP; Gunicorn/uWSGI per a Python). Després, el servidor web retorna la resposta al client.*

- ***"Reverse Proxy"****: peça que afegeix TLS, caché, equilibri de càrrega i protegeix els serveis interns (molt comú amb Node i Python)*

2. A partir de la taula:

- Explica en quins casos és més avantatjós PHP, en quins Python i en quins Node.js.

PHP:

- **Webs de contingut i CMS: blogs, portals, intranets, e-commerce amb WordPress/Prestashop.**
- **CRUD ràpid (llistats, formularis, panell d'admin) amb terminis curts.**
- **Hosting barat i simple (shared hosting, XAMPP/LAMP): pujar fitxers i funcionar.**
- **Equip novell o mixt: aprendre i desplegar és molt directe.**
- **Projectes web clàssics on la prioritat és arrancar ràpid amb pressupost ajustat.**

PHYTON:

- **APIs "netes" i mantenibles (Django/Flask/FastAPI) amb bones pràctiques i tests.**
- **Projectes amb dades/IA: analítica, ML, integracions amb notebooks, scripts.**
- **Backend + processos (ETL, tasques programades, cues) dins el mateix ecosistema.**
- **Equips que volen qualitat de codi i llarg recorregut (claror, tipatge opcional, testing).**

NODE.JS :

- **Temps real i alta concurrència: xats, notificacions, jocs, live dashboards, websockets.**
- **Una sola llengua (JS) en front + back : compartir models/validacions i ser més àgil.**
- **APIs lleugeres i microserveis amb moltes connexions E/S (streams, proxys).**
- **Ideal quan vols temps real i moltes connexions amb desenvolupament ràpid en JavaScript.**

- Reflexiona: si haguessis de programar una app de xat en temps real, quin triaries i per què?

  Triaria *Node.js* :

- Model d'E/S asíncrona *(operacions de disc/xarxa no bloquegen; el servidor pot atendre altres usuaris mentre espera)* i event loop *(bucle que processa una cua d'esdeveniments i executa callbacks un a un)*.
  És ideal per mantindre moltes connexions obertes amb WebSockets *(connexió bidireccional i persistent entre client i servidor)*.

- Ecosistema madur per a temps real: Socket.IO *(llibreria WebSocket amb sales, reconexió i broadcast "out-of-the-box")*, ws *(implementació WebSocket minimalista i molt ràpida)*, Redis adapters *(connector que usa Redis com a bus pub/sub per coordinar sales i missatges entre múltiples instàncies)* per a sales, historial, escala horitzontal *(afegir més processos/servers en paral·lel per atendre més usuaris)*…

- Una sola llengua (JS) a client i servidor: comparteix models/validacions i accelera el desenvolupament.

- Desplegament estàndard: Nginx de reverse proxy *(servidor frontal que rep les connexions, fa TLS/estàtics i reparteix tràfic a les teues instàncies)* al davant i Node gestionant el temps real; escalar és senzill.

  Per què no les altres dues?

- Python (FastAPI + websockets / Starlette) *(frameworks ASGI, és a dir, interfície moderna per apps async en Python)* és bona opció si el projecte integra IA/analítica o ja tens l'equip en Python.

- PHP pot fer-ho (Ratchet, Swoole — *servidor/event loop per a PHP*) però no és el seu terreny natural; requerix més enginyeria per a temps real sostingut.

  Conclusió: per a un xat en temps real, Node.js és la tria més directa i eficient per concurrència *(moltes connexions simultànies)*, ecosistema i productivitat.

Exercici 4 — Eines i frameworks

Objectiu: entendre utilitats pràctiques dels frameworks.

Un cop entès el funcionament bàsic d'una aplicació web, toca veure com els frameworks ens ajuden a organitzar i simplificar el codi. En aquest exercici compararem la manera de treballar sense framework amb l'estructura que ofereix un framework modern.

Enunciat

1. Tria un dels llenguatges anteriors i investiga el seu framework més popular.

- Ex: PHP → Laravel, Python → Django, Node.js → Express.

  Bé, com aquest curs treballarem amb Laravel, he decidit escollir-lo. Així em vaig familiaritzant. He tornat a fer ús de wikipedia.

  https://es.wikipedia.org/wiki/Laravel

2. Busca i explica:

- Com gestiona les rutes (posa un exemple).

  Laravel té un sistema d'enrutament:
  a ***routes/web.php*** defineixes quin codi o controlador respon a cada URL. Pots usar funcions anònimes (closures) o controladors:

```
// routes/web.php
use Illuminate\Support\Facades\Route;

Route::get('/hola', function () {
    return 'Hola món';
});
```

  Això mapeja GET /hola a aquesta funció i retorna el text.

- Com organitza carpetes (Models, Vistes, Controladors).

  Laravel segueix **MVC** i una estructura de directoris clara:

- **Models** (p. ex. app/Models/): representen taules i encapsulen l'accés a dades (Eloquent).
- **Vistes** (resources/views/): són les vistes Blade (plantilles HTML amb una sintaxi curta, com {{ $nom }}).
- **Controladors** (app/Http/Controllers/): contenen la lògica per rebre peticions i retornar respostes/vistes.
- **Database/migrations/**: migracions (versions de l'esquema de BD).

- Quina eina dona per a la base de dades (ORM, migracions…).

- **Eloquent ORM**: és l'ORM de Laravel (patró *Active Record*). Et deixa treballar amb la BD com objectes i defineix relacions (1:N, N:M…). Exemple de consulta: Libro::all().

  ***Què és un ORM**?*
  Un ORM ("Object-Relational Mapper") és una eina que et deixa treballar amb la base de dades com si foren objectes del teu llenguatge, en lloc d'escriure SQL a mà.

  Per deixar-ho més clar...
  *> Sense ORM: escrius SELECT * FROM products WHERE id = 5; i després converteixes el resultat (files/columnes) a variables/estructures.*
  *> Amb ORM: fas Product::find(5) i el que reps és directament un **objecte Product** amb propietats ($product->name, $product->price).*

  Per què és útil?
- *Més ràpid de programar: menys SQL repetitiu.*
- *Codi més net i mantenible: lògica en models, no barrejada amb vistes/controladors.*
- *Portabilitat: canvi de SGBD (MySQL, PostgreSQL...) amb pocs canvis.*
- *Seguretat: consultes preparades i escape automàtic (reduïx risc d'SQL injection).*

- **Migracions**: Laravel integra migracions per versionar l'esquema (crear/modificar taules per codi). Podem dir que es un "control de versions" de l'esquema.
  S'apliquen amb ***"php artisan migrate".***

- Quina eina dona per a la seguretat (hashing, middleware, permisos…).

- **Hashing de contrasenyes**: Laravel usa Hash::make(...) (bcrypt/argon) i comprova amb Hash::check(...). Venen integrats amb el sistema d'autenticació.

- **Middleware**: filtres que s'executen abans/després d'una ruta. Per ex:

  · *auth* (obliga a estar autenticat),

  · *throttle* (limita peticions),

  · *protecció* CSRF per a rutes web. S'associen a rutes o grups de rutes.

- **Autorització (permisos)**: Gates i Policies per decidir "qui pot fer què" (per exemple, "l'usuari X pot editar el producte Y?"). Pots aplicar-ho des de controladors o com a middleware.

3. Reflexiona: si féssim el CRUD de productes sense framework, quines parts serien més difícils de mantenir?

1) **Rutes i organització de fitxers**

- Sense framework: acabes amb molts .php barrejats (e.g., llista.php, nou.php, guarda.php, edita.php, esborra.php), redireccions manuales i include per tot arreu.
- Per què costa: duplicació de codi i camins "trencats" quan canvies una URL.
- Amb Laravel: rutes clares a routes/web.php i controladors per accions.

2) Validació de formularis i missatges d'error

- Sense: valides "a mà" (required, numèric, longitud…) a cada fitxer; tornes errors amb echo o variables ad hoc.
- Per què costa: repeteixes regles; fàcil oblidar-te'n en alguna acció.
- Amb Laravel: FormRequest/$request->validate() i missatges consistents.

3) **Accés a base de dades i SQL repetit**

- Sense: mysqli/PDO a mà, escrivint SELECT/INSERT/UPDATE/DELETE a cada pàgina.
- Per què costa: molt codi boilerplate, errors de còpia/enganxa, canviar l'esquema és feixuc.
- Amb Laravel: Eloquent ORM per CRUD i migracions versionades.

4) **Migracions i canvis d'esquema**

- Sense: canvies taules "a mà" (phpMyAdmin/SQL), i cada company té una BD diferent.
- Per què costa: inconsistència entre entorns; difícil "desfer" un canvi.
- Amb Laravel: fitxers de migració i php artisan migrate per sincronitzar.

5) **Plantilles i reutilització d'HTML**

- Sense: còpies d'header.php/footer.php; canviar el menú implica tocar 10 fitxers.
- Per què costa: inconsistència visual i temps perdut.
- Amb Laravel: Blade layouts (@extends, @section) i components.

6) **Paginació, filtrat i ordenació**

- Sense: càlcul de LIMIT/OFFSET, paràmetres a mà, links de pàgines construïts manualment.
- Per què costa: codi llarg i propens a errors.
- Amb Laravel: $products->paginate() i helpers de vistes.

7) **Autenticació i permisos**

- Sense: sessions "a mà", hashing de contrasenyes, middleware casolà per restringir rutes.
- Per què costa: molt codi sensible i repetit, difícil d'auditar.
- Amb Laravel: php artisan make:auth (o Breeze/Jetstream), polítiques (Policies) i gates.

CONCLUSIÓ:

Si fem ús del framework, tindrem menys codi repetit, **més seguretat** i un **projecte mantenible a mesura que creix.**