



Nom i Cognoms	Pablo Vicente Roura
----------------------	---------------------

Al contrari que altres pràctiques, aquesta te, principalment, preguntes que s'han de respondre en aquest document

ACTIVITAT
Objectius: <ul style="list-style-type: none">- Practicar conceptes relacionats amb construcció, implementació, i integració de components de programari específics per a la persistència de dades, incloent la seva especificació, empaquetatge, i desplegament.
Instruccions: <ul style="list-style-type: none">- Responen a l'espai de cada pregunta, si ho feu amb diapositives enganxeu la diapositiva en aquest mateix espai.- Es valorarà la cura en la presentació del document i que segueixi l'estructura indicada.
Criteris d'avaluació: <ul style="list-style-type: none">- Cada pregunta té el mateix pes- Es valorarà la presentació i els comentaris al codi
Entrega: <ul style="list-style-type: none">- Aquest document anomenat memoria.pdf i el codi corresponent



Exercicis

Exercici 1. GitHub Actions (2 punts)

GitHub Action: Projecte de Java amb Maven

Fes una còpia d'aquest repositori

<https://github.com/jpala4-ieti/DAM2-MP06-UF04-Java-Base.git>

Entén com funcionen les Github Actions amb ajuda d'aquest recursos:

- <https://docs.google.com/presentation/d/1fGSGT8vWeaXBloWA3eu05sfJzqK8qZrWr12PdZRvMqE/edit?usp=sharing>
- <https://docs.github.com/es/actions/writing-workflows/quickstart>

Acaba modificant el README.md de la copia del teu projecte per tal que el badge mostri la informació del teu repositori i no de l'original.

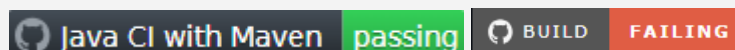
Assegura't que l'acció s'executa correctament.

URL del teu repositori: <https://github.com/pablovr05/PROXMOX>

Explica com funciona el badge i quin és el fitxer de configuració que fa servir i que GitHub acaba usant

Que es el badge?

El badge és una petita imatge que mostra l'estat de l'última execució de la GitHub Action. Quan el workflow s'executa amb èxit o falla, el badge canvia automàticament per reflectir l'estat actual del workflow.



El fitxer de configuració que defineix l'acció es troba dins del directori `.github/workflows/` i es diu `maven.yml`. Aquest fitxer conté la definició de les accions que es realitzen quan es puja codi al repositori.



Funcionament del Badge

1. Activa l'acció quan es fa un **push** o **pull request** a la branca **main**.
2. Executa el job **build** en un entorn **ubuntu-latest**.
3. Descarrega el codi del repositori (**checkout**).
4. Configura el JDK 11 per a l'execució del codi.
5. Executa la compilació del projecte amb Maven (**mvn package**).



```
name: Java CI with Maven
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven
      - name: Build with Maven
        run: mvn -B package --file pom.xml
```



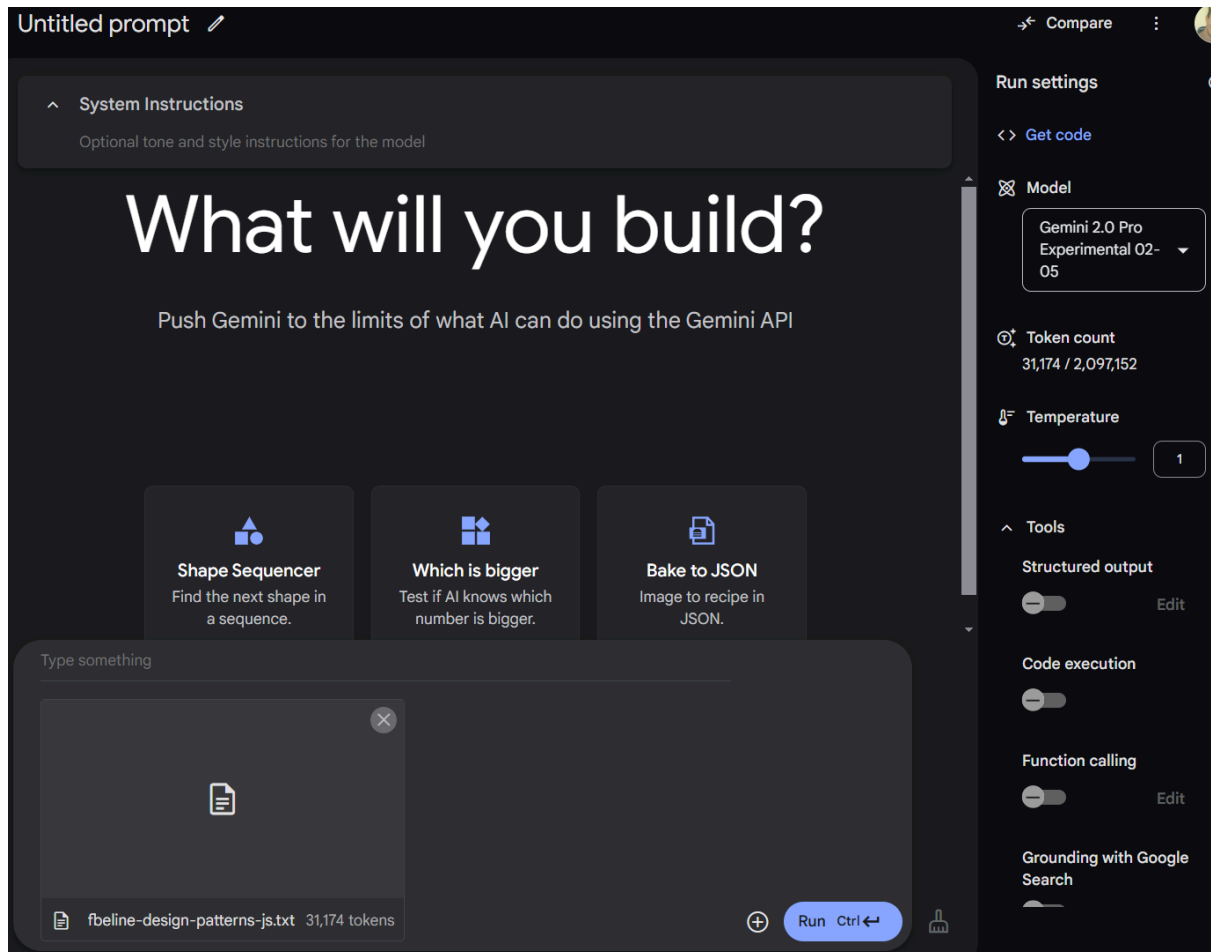
Exercici 2. Sessió d'estudi amb ajuda de IA (4 punts)

Et caldrà poder usar aquesta eina https://aistudio.google.com/prompts/new_chat (Compte personal). Model a usar: “**Gemini 2.0 Pro Experimental 02-05**”

Descarregar el fitxer que genera aquesta aplicació

<https://gitingest.com/fbeline/design-patterns-JS>

Inicia una nova conversa carregant el fitxer que s'ha generat amb gitingest





Començar amb la pregunta “Què puc aprendre a partir d'aquesta base de codi?” I, a partir d'aquesta resposta inicial tria 5 temes a desenvolupar i acaba explicant cadascun dels temes amb les teves paraules.

Resposta de Gemini:

Comparació de prototips i classes ES6:

- El codi mostra com implementar els mateixos patrons de disseny utilitzant tant prototips com classes ES6. Pots aprofundir en les diferències entre aquests dos enfocaments, els seus avantatges i desavantatges, i quan és millor utilitzar-ne un o l'altre.
- Aquest tema t'ajudarà a entendre millor com funciona JavaScript per sota del capó i a prendre decisions més informades sobre com estructurar el teu codi.

Patrons de disseny en aplicacions web:

- Pots explorar com aplicar aquests patrons de disseny en el context del desenvolupament web modern. Per exemple, pots investigar com utilitzar el patró Observer per gestionar les actualitzacions de la interfície d'usuari, o com utilitzar el patró Factory per crear components reutilitzables.
- Aquest tema t'ajudarà a construir aplicacions web més robustes, mantenibles i escalables.

Proves unitàries i qualitat del codi:

- El codi inclou proves unitàries per a cada patró de disseny. Pots aprofundir en com escriure proves efectives, com utilitzar marcs de proves com Jest o Mocha, i com mesurar la cobertura del codi.
- Aquest tema t'ajudarà a millorar la qualitat del teu codi i a prevenir errors.

Documentació i bones pràctiques:

- El fitxer `docs.md` és un exemple de com documentar el codi. Pots explorar diferents eines i tècniques per generar documentació automàticament, com JSDoc o TypeDoc, i com escriure documentació clara i concisa.
- Aquest tema t'ajudarà a comunicar millor el teu codi a altres desenvolupadors i a fer que sigui més fàcil de mantenir.

Refactorització de codi existent:

- Pots prendre codi JavaScript existent i refactoritzar-lo per aplicar patrons de disseny. Aquest és un exercici pràctic excel·lent per entendre com els patrons de disseny poden millorar la llegibilitat, la mantenibilitat i la flexibilitat del codi.
- Aquest tema t'ajudarà a desenvolupar habilitats de refactorització i a aprendre a millorar el codi sense canviar-ne la funcionalitat.



A partir d'aquesta resposta donada per Gemini, els temes dels que tractaré a continuació amb les meves paraules son:

- **Classes ES6**
- **Qualitat del codi**
- **Bones pràctiques**
- **Refactorització de codi existent**
- **Proves unitàries**

A més a més, he de dir que per poder realitzar aquesta activitat buscaré informació sobre que son les classes ES6 ja que mai he escoltat parlar sobre aquest terme i em sembla interessant.

Explicació amb les meves paraules

CLASSES ES6

Primer de tot m'agradaria parlar del nom ES6, ja que m'ha semblat curiós i he fet cerques sobre el seu nom. ES6 prové de ECMAScript, que és l'estàndard de JavaScript, i el 6 fa referència a la versió 6 d'aquest estàndard. ECMAScript prové de ECMA, que és una organització que defineix els estàndards per a diverses tecnologies de la web, incloent-hi JavaScript.

ES6 va aportar diverses millores a JavaScript, algunes de les quals són:

Classes: Abans d'ES6, JavaScript no tenia una manera oficial de crear classes. Amb ES6, es va introduir la sintaxi de class, que permet crear objectes amb estructures més clares i llegibles, apropant-se més a altres llenguatges de programació orientats a objectes como Java.

Arrow Functions: Va introduir les arrow functions o funcions amb fletxa. Aquestes funcions tenen una sintaxi més curta i també comporten un canvi en el funcionament de la paraula this en el seu interior.

Això és un exemple molt senzill:

```
const suma = (a, b) => a + b;
```

Let i Const: Anteriorment, només hi existia var per declarar variables, però amb ES6, es van introduir let i const. Let permet declarar variables amb un abast d'un bloc de codi, mentre que const s'utilitza per declarar constants que no es poden modificar (final).

Destructuració: La destructuració és una característica que permet desestructurar valors de arrays o objectes en variables individuals, és molt útil pero guardar informació estructura de manera local.

```
const persona = { nom: "Pablo", edat: 20 };  
const { nom, edat } = persona;  
console.log(nom); "Pablo"
```



QUALITAT DEL CODI

Abans de res, m'agradaria parlar una mica sobre el concepte de qualitat del codi, ja que em sembla fonamental i vaig voler investigar sobre què implica realment. En termes generals, la qualitat del codi es refereix a com de net, clar i eficaç és el codi que escrivim. Però no es tracta només que funcioni, sinó també que sigui fàcil de mantenir i de comprendre per altres programadors. Algunes coses que cal tenir en compte per millorar la qualitat del codi:

Llegibilitat: El codi ha de ser fàcil de llegir i entendre. A l'hora de programar, no només hem de pensar en què fa el codi, sinó també en com serà llegit per una altra persona (o fins i tot per tu mateix d'aquí a uns mesos). És important posar noms clars i descriptius per a les variables i funcions, de manera que qualsevol persona que llegeixi el codi sàpiga immediatament què fa.

Comentaris: Els comentaris són una eina molt útil per explicar el "per què" d'algunes decisions en el codi. Els comentaris només han d'estar on sigui necessari per aclarir conceptes o processos difícils de comprendre al primer cop d'ull. Han de ser breus però informatius, evitant explicar coses òbvies.

Modularitat: És molt important dividir el codi en petites parts independents. Això fa que el codi sigui més reutilitzable i fàcil de mantenir. Les funcions i les classes han de fer una sola cosa.

Evitar duplicació de codi: Si existeix la mateixa lògica en diverses parts del codi, és millor agrupar-la en una sola funció i reutilitzar-la. Això no redueix la duplicació només, sinó que també fa que sigui més fàcil detectar i arreglar errors, ja que qualsevol canvi ha de ser fet en un sol lloc.

Organització i normes d'estil: Seguir unes normes d'estil i mantenir una organització coherent al llarg del codi és fonamental per garantir la seva qualitat. Això inclou coses com utilitzar un estil de codi generalitzat, anomenar les variables seguint unes pautes comunes.



BONES PRÀCTIQUES

Les bones pràctiques de programació són les normes no escrites que ens ajuden a escriure codi fàcil de llegir, mantenir i ampliar. Quan es treballa en projectes, seguir certes regles o estructuració ajuda a gestionar i crear un projecte més net i clar. Algunes de les bones pràctiques més importants són:

Primer, és essencial utilitzar noms clars i coherents per a les variables, funcions i classes. Els noms han de descriure clarament el que fa cada element. Per exemple, en comptes d'utilitzar noms genèrics com `x` o `temp`, és millor utilitzar noms com `temperaturaActual` o `nombreUsuario`, que faciliten la comprensió del codi. A més, les classes en JavaScript, com a regla general, es solen declarar amb la primera lletra en majúscula, ja que això ajuda a distingir-les clarament dels altres tipus d'objectes o variables. Així, la classe `Persona` es diferenciarà fàcilment d'una variable com `persona`.

També és important no sobrecarregar funcions o mètodes. Cada funció hauria de fer una sola cosa i fer-ho bé. És millor dividir la funció en diverses més petites i manejables, perquè cada una realitzi una tasca específica i essencial. Això farà que el codi sigui més modular i fàcil de mantenir. A més, les funcions també s'han de nomenar de manera coherent, preferentment amb verbs que indiquin clarament què fa la funció, com per exemple `calcularSuma()` en lloc de noms vagues com `funcio1()`.

Quan es treballa amb objectes, és important recordar que els noms de les funcions i les variables han de seguir una convenció consistent. Els objectes poden ser declarats amb la primera lletra en minúscula, com `persona` o `coordinada`, perquè representen instàncies o entitats que tenen propietats i comportaments associats a elles. Aquesta convenció ajuda a separar visualment les classes d'objectes i evita confusions amb altres elements del codi.

El maneig d'errors també és una altra pràctica essencial. Quan el codi falla, és millor capturar l'error i mostrar un missatge clar que ajudi a identificar què ha passat. Utilitzar tècniques com les excepcions o els `try-catch` ajuda a gestionar els errors de manera ordenada.



REFACTORITZACIÓ DE CODI EXISTENT

La refactorització de codi existent és el procés de modificar el codi d'un programa per millorar-ne l'estructura, llegibilitat i eficiència sense alterar el comportament del programa. És essencial per mantenir un codi net, entendible i fàcil de mantenir a mesura que un projecte creix i evoluciona.

Una de les primeres raons per refactoritzar codi és reduir la seva complexitat. Amb el temps, un projecte pot acumular codi, funcions massa llargues o lògica poc clara. Amb la refactorització, es poden dividir funcions grans en altres més petites i específiques, eliminar duplicació de codi i reestructurar-lo per fer-lo més entenedor. Això fa que el manteniment sigui més senzill i que altres programadors puguin entendre el codi amb més facilitat.

Un altre aspecte clau de la refactorització és millorar la reutilització del codi. Quan s'agrupa funcionalitats repetides en funcions separades, es redueix la quantitat de codi duplicat i s'aconsegueix un sistema més organitzat. Això no només fa que el codi sigui més net y facilita que en un futur ho puguem entendre més ràpidament.

També és una pràctica fonamental per millorar el rendiment del codi. Sovint es poden trobar maneres d'optimitzar algorismes, reduir consultes innecessàries a bases de dades o evitar operacions costoses. Això pot resultar en un programa més bo i ràpid sense necessitat de modificar una funcionalitat.



REFACTORITZACIÓ DE CODI EXISTENT

Les proves unitàries són un tipus de test automatitzats que serveixen per verificar que unitats específiques del codi, com funcions, mètodes o classes, funcionin correctament de manera aïllada. L'objectiu principal és detectar errors en fases inicials del desenvolupament i garantir que els canvis en el codi no trenquin funcionalitats existents.

Una de les principals avantatges de les proves unitàries és que permeten identificar errors ràpidament. Si cada funció o mètode té el seu test associat, qualsevol modificació en el codi es pot validar automàticament, evitant que es propaguin errors a altres parts del projecte. Això és especialment útil en equips grans o en aplicacions que evolucionen constantment.

També ajuden a millorar la qualitat del codi, ja que obliguen els desenvolupadors a escriure codi més modular i desacoblat. Quan un mètode és difícil de testejar, sovint és una senyal que està fent massa coses o que depèn en excés d'altres components. Escriure proves unitàries força a dividir el codi en peces més manejables i reutilitzables.

Per implementar proves unitàries, s'utilitzen frameworks de testing específics segons el llenguatge de programació. Per exemple, en Java s'utilitza JUnit, en JavaScript Jest o Mocha, i en Python pytest. Aquests frameworks permeten definir tests de manera senzilla i automatitzar-ne l'execució per comprovar si el codi passa o falla en diferents escenaris.

Una bona pràctica és aplicar el concepte de *test-driven development* (TDD), que consisteix a escriure primer les proves unitàries abans de desenvolupar el codi. Això assegura que el codi nou es crea amb els requisits correctes i ajuda a evitar errors des del principi.



WEBGRAFIES D'ON HE TRET INFORMACIÓ QUE DESPRÉS QUE HE ESCRIT AMB LES MEVES PARAULES

1. CLASSES ES6

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

2. QUALITAT DEL CODI

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

3. BONES PRÀCTIQUES

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring>

4. REFACTORITZACIÓ DE CODI EXISTENT

<https://github.com/airbnb/javascript>

<https://refactoring.com/>

5. REFACTORITZACIÓ DE CODI EXISTENT

<https://jestjs.io/>

https://ca.wikipedia.org/wiki/Proves_unit%C3%A0ries



Exercici 3. Sessió de refactorització de la teva pràctica 4.2 (4 punts)

Amb gitingest, obtingues el codi de la teva pràctica 4.2, carregala en una nova sessió de xat.

Demana-li:

- Recomanacions/comentaris de millora de codi i implementa'n dues (amb ajuda de la IA).
- Quins són els errors més greus/males pràctiques en el desenvolupament de programari que has comès
- Una proposta de test unitari que cobreixi algun aspecte important i implementa-la

IMPORTANT

URL repositori refactoritzat: https://github.com/pablovr05/imagIA_server

Encara que el repositori existeix i és públic, el gitIngest no detectava el repositori i per això no podia aconseguir el resum del meu projecte, és per això que he utilitzat el repositori del servidor node de Imagia2 que té una estructura molt semblant al de la pràctica:

<https://github.com/pablovr05/PR4.2-24-25-PuntPartida>



Resum dels errors més greus que has comès:

Emmagatzematge de Contrasenyes en Text Pla: El problema més crític. Desar les contrasenyes sense "hashejar" (encriptar de forma no reversible) és una vulnerabilitat de seguretat molt greu. Si la base de dades es veiés compromesa, totes les contrasenyes quedarien exposades.

Ús de `sequelize.sync({ force: true })` a l'Arrencada: Aquesta configuració fa que totes les taules de la base de dades s'eliminin i es tornin a crear cada cop que s'inicia el servidor. Això garanteix la pèrdua de totes les dades existents, sent molt perillós per al desenvolupament i totalment inacceptable per a producció.

Emmagatzematge de Codis de Verificació en Memòria: Guardar els codis de verificació SMS en una variable global del servidor (`verificationCodes`) és insegur (podrien filtrar-se), poc fiable (es perden si el servidor reinicia) i impedeix escalar l'aplicació a múltiples instàncies.

Possible Inclusió del Fitxer `.env` al Control de Versions (Git): El comentari `# .env` al `.gitignore` suggereix que el fitxer `.env`, que conté secrets (contrasenyes de BD, tokens d'API), podria estar sent inclòs a Git. Això és un risc de seguretat enorme, ja que exposa les credencials.

Condicció de Cursa (Race Condition) a `useQuote`: La manera com es comprova i es resta la quota (`remainingQuote`) no és atòmica. Dues peticions simultànies podrien llegir la mateixa quota, passar la comprovació i restar-la dues vegades, portant a una comptabilitat incorrecta.



Quin test unitari t'ha suggerit? Per quin motiu?

Test Suggestit: Un test unitari per a la funció `registerUser` del controlador (`chatController.js`).

Motiu: Es va suggerir aquest test perquè `registerUser` és una funció fonamental i crítica de l'aplicació.

Implica:

- La creació de nous usuaris.
- La gestió d'informació sensible (la contrasenya).
- La interacció amb la base de dades (`Users.create`).
- L'inici del flux de verificació (generació del codi SMS).
- La resposta a l'usuari.

El test proposat se centra a verificar que la millora de seguretat implementada (el "hashing" de contrasenyes amb `bcrypt`) funciona correctament. També comprova que es crida a la base de dades amb les dades adequades (incloent la contrasenya "hashejada") i que la resposta HTTP donada al client en cas d'èxit és la correcta (status 201 i les dades esperades).



Com valores les recomanacions i comentaris que t'ha donat?

Les recomanacions que he rebut em semblen molt valuoses i rellevants perquè toquen punts crítics per al desenvolupament d'una API robusta i segura.

Pel que fa a seguretat, m'ha quedat clar que hi havia algunes vulnerabilitats greus, com l'ús de contrasenyes en text pla o la possibilitat de filtrar secrets a Git. També he vist que l'emmagatzematge dels codis no era el més segur. Les solucions proposades, com utilitzar **bcrypt** per encriptar contrasenyes, gestionar correctament el `.gitignore` i tenir un emmagatzematge més segur.

En quant a la integritat de dades, m'han fet adonar que el **force: true** pot provocar pèrdua de dades, i que a **useQuote** podria haver-hi condicions de cursa. Això reforça la importància d'utilitzar **migracions de base de dades** i operar de manera atòmica per evitar problemes inesperats.

També hi ha molts aspectes de **bones pràctiques i mantenibilitat** que puc millorar. Un punt clau és la validació d'entrades, que seria molt més eficient si centralitzo el middleware amb **express-validator** en lloc de fer validacions disperses pel codi. També m'ha semblat molt encertat el consell d'evitar valors "hardcoded" i utilitzar constants, així com millorar la gestió del **logging** per tenir una visió més clara dels errors i el funcionament de l'API.

Finalment, el tema del **testing** és una altra àrea on puc millorar molt. Les recomanacions que he rebut sobre tenir una configuració adequada de tests i incloure tests unitaris, com l'exemple que m'han donat, m'han fet veure la importància de dedicar més temps a assegurar que cada peça de codi funciona correctament abans de desplegar res en producció.



Com valores l'experiència que has tingut interactuant amb aquesta IA?

L'experiència amb aquesta IA ha estat molt positiva. Les seves anàlisis han estat encertades i útils, especialment en seguretat, assenyalant riscos com l'emmagatzematge de contrasenyes en text pla i la pujada de fitxers sensibles a Git. També ha identificat problemes d'integritat de dades, com l'ús de `force: true` i possibles condicions de cursa, proposant solucions adequades.

Les recomanacions en bones pràctiques m'han ajudat a millorar la validació de dades amb `express-validator`, evitar l'ús de cadenes literals i consolidar el sistema de logs. A més, m'ha fet veure la importància de tenir tests unitàries, proposant-ne un per validar el registre d'usuaris i el hashing de contrasenyes. M'ha fet plantejar-me la manera de guardar tokens i informació important i no posar-ho a `.env` que és una pràctica dolenta la que és visible per tothom.

En resum, el feedback ha estat pràctic, tècnicament sòlid i alineat amb les millors pràctiques. Aquesta interacció m'ha ajudat a detectar punts febles i aplicar solucions efectives, fent que l'experiència sigui molt bona.