Nom i Cognoms:	
URL Repositori Github:	

ACTIVITAT
Objectius: ● Familiaritzar-se amb el desenvolupament d'APIs REST utilitzant Express.js ● Aprendre a integrar serveis de processament de llenguatge natural i visió artificial ● Practicar la implementació de patrons d'accés a dades i gestió de bases de dades ● Desenvolupar habilitats en documentació d'APIs i logging ● Treballar amb formats JSON i processament de dades estructurades
Criteris d'avaluació: - Cada pregunta indica la puntuació corresponent
Entrega: - Repositori git que contingui el codi que resol els exercicis i, en el directori "doc", aquesta memòria resposta amb nom "memoria.pdf"

Punt de partida
https://github.com/jpala4-ieti/DAM-M06-UF04-PR4.2-24-25-Punt-Partida.git

Preparació de l'activitat

- Clonar el repositori de punt de partida
- Llegir els fitxers README.md que trobaràs en els diferents directoris
- Assegurar-te de tenir una instància de MySQL/MariaDB funcionant
- Tenir accés a una instància d'Ollama
- Completar els quatre exercicis proposats
- Lliurar el codi segons les instruccions d'entrega

Exercicis

Exercici 1 (2.5 punts)

L'objectiu de l'exercici és familiaritzar-te amb **xat-api**. Respon la les preguntes dins el requadre que trobaràs al final de l'exercici.

Configuració i Estructura Bàsica:

1. Per què és important organitzar el codi en una estructura de directoris com controllers/, routes/, models/, etc.? Quins avantatges ofereix aquesta organització?

És important organitzar el codi per què ajuda a mantenir un codi més net, comprensible i escalable. Aquesta organització ofereix els següents avantatges:

- Facilita la mantenibilitat, ja que cada component (lògica de negoci, rutes, models de dades, etc.) està separat per funció.
- Millora la col·laboració en equips, permetent que diversos desenvolupadors treballin en diferents parts del projecte sense interferir entre ells.
- Proporciona una estructura clara, fàcil d'entendre i navegar, especialment en projectes grans.

2. Analitzant el fitxer server.js, quina és la seqüència correcta per inicialitzar una aplicació Express? Per què és important l'ordre dels middlewares?

   L'ordre importa perquè Express processa les peticions en seqüència. Si un middleware, com el de CORS o el parser de JSON, s'executa després d'una ruta, la petició podria no processar-se correctament. Això garanteix que cada petició passi pels processos correctes abans d'arribar a la seva ruta final.

3. Com gestiona el projecte les variables d'entorn? Quins avantatges ofereix usar dotenv respecte a hardcodejar els valors?

   El projecte utilitza dotenv per carregar variables d'entorn des d'un fitxer `.env`.

Avantatges d'usar dotenv:

- Seguretat: Manté les dades sensibles fora del codi, evitant que s'exposin si es comparteix el projecte.
- Flexibilitat: Permet canviar les configuracions fàcilment sense modificar el codi.
- Consistència: Ajuda a mantenir un sol lloc on definir aquestes configuracions, evitant errors humans.

API REST i Express:

1. Observant chatRoutes.js, com s'implementa el routing en Express? Quina és la diferència entre els mètodes HTTP GET i POST i quan s'hauria d'usar cadascun?

   En el fitxer chatRoutes.js, el routing en Express es fa mitjançant el mètode router d'Express per definir les diferents rutes URL que l'aplicació ha de gestionar. Per cada ruta, s'especifica el tipus de sol·licitud HTTP GET o POST i la funció corresponent del controlador que s'executarà quan es cridi a aquella ruta.

   Diferència entre GET i POST:

- GET: Es fa servir per obtenir informació del servidor. No modifica cap dada, només retorna informació.
- POST: Es fa servir per enviar dades al servidor i normalment per crear o modificar recursos.

Quan utilitzar cada mètode: Utilitza GET quan necessites accedir a informació ja existent. Utilitza POST quan necessites enviar dades al servidor, com crear o modificar.

2. En el fitxer chatController.js, per què és important separar la lògica del controlador de les rutes? Quins principis de disseny s'apliquen?
   Perquè aixis es mes facil a la hora de comprendre les coses, també ens ayuda amb la forma de reutilització del codi i també per la claredat amb el que es poden veure les coses.
3. Com gestiona el projecte els errors HTTP? Analitza el middleware errorHandler.js i explica com centralitza la gestió d'errors.

   Aquest middleware centralitza la gestió d'errors, fent que qualsevol error en qualsevol part de l'aplicació es tracti de forma consistent. Això millora la seguretat i facilita la detecció i solució de problemes.

   Gestións: Quan es produeix un error, es registra la informació de l'error (missatge, mètode HTTP, URL, stack trace) amb un logger,per errors de validació o duplicats de la base de dades (Sequelize), es gestionen amb missatges i codis de resposta específics com el 400 (sol·licitud incorrecta) i per els errors no controlats, es retorna el codi 500.

Documentació amb Swagger:

1. Observant la configuració de Swagger a swagger.js i els comentaris a chatRoutes.js, com s'integra la documentació amb el codi? Quins beneficis aporta aquesta aproximació?

   El Swagger s'integra amb el codi utilitzant comentaris especials en els arxius de rutes, com chatRoutes.js, que descriuen els endpoints i les seves especificacions. Swagger genera automàticament la documentació a partir d'aquests comentaris. Això aporta claredat, ja que la documentació i el codi estan sincronitzats.
2. Com es documenten els diferents endpoints amb els decoradors de Swagger? Per què és important documentar els paràmetres d'entrada i sortida?

   Els endpoints es documenten amb comentaris `@swagger` que defineixen els detalls com el tipus de mètode HTTP (GET/POST), els paràmetres d'entrada i els possibles resultats. Documentar els paràmetres d'entrada i sortida és important perquè ajuda els desenvolupadors a entendre com interactuar amb l'API de forma correcta i què esperar com a resposta.

3. Com podem provar els endpoints directament des de la interfície de Swagger? Quins avantatges ofereix això durant el desenvolupament?

   Des de la interfície de Swagger, es poden provar directament els endpoints enviant peticions reals des del navegador. Això facilita molt les proves durant el desenvolupament, ja que permet veure les respostes de l'API en temps real i identificar problemes sense la necessitat d'una eina externa com Postman.

Base de Dades i Models:

1. Analitzant els models Conversation.js i Prompt.js, com s'implementen les relacions entre models utilitzant Sequelize? Per què s'utilitza UUID com a clau primària?

   Les relacions s'implementen utilitzant funcions com belongsTo i hasMany. Per exemple, un model Prompt pertany a una Conversation, i una Conversation pot tenir molts Prompts. Això ajuda a definir les associacions entre les taules de la base de dades.

   L'UUID és utilitzat com a clau primària perquè proporciona identificadors únics universals, cosa que millora la seguretat i escalabilitat en entorns distribuïts.

2. Com gestiona el projecte les migracions i sincronització de la base de dades? Quins riscos té usar `sync()` en producció?

   El projecte utilitza Sequelize per gestionar les migracions i sincronitzar la base de dades. La funció sync() crea o actualitza les taules automàticament segons els models. En producció, usar sync() és arriscat perquè pot sobrescriure dades o alterar l'estructura sense control.

3. Quins avantatges ofereix usar un ORM com Sequelize respecte a fer consultes SQL directes?

   Simplifica la gestió de la base de dades permetent treballar amb objectes en lloc de consultes SQL directes. Això augmenta la productivitat, millora la llegibilitat del codi i proporciona abstracció de diferents bases de dades.

Logging i Monitorització:

1. Observant logger.js, com s'implementa el logging estructurat? Quins nivells de logging existeixen i quan s'hauria d'usar cadascun?

   El logging s'implementa amb Winston, creant logs amb formats consistents que inclouen timestamps, errors, metadades, i traces d'stack. Això facilita la interpretació automàtica dels logs.
   Els nivells inclouen error, warn, info, debug, entre d'altres. S'utilitzen segons la gravetat: error per fallades greus, warn per advertiments, info per operacions generals, i debug per depuració durant el desenvolupament.

2. Per què és important tenir diferents transports de logging (consola, fitxer)? Com es configuren en el projecte?

   S'usen transports com la consola i fitxers per distribuir els logs. La consola és útil durant el desenvolupament, mentre que els fitxers s'usen en producció per auditar incidents. Es configuren amb Winston afegint diferents transports al logger.

3. Com ajuda el logging a debugar problemes en producció? Quina informació crítica s'hauria de loguejar?

   El logging permet detectar problemes en producció registrant errors, durades de sol·licituds, estats HTTP, i IPs. Es crítica loguejar errors amb traces d'stack, informació d'execució, i metadades rellevants per traçar el context dels problemes.

Exercici 2 (2.5 punts)

Dins de **practica-codi** trobaràs **src/exercici2.js**

Modifica el codi per tal que, pels dos primers jocs i les 2 primeres reviews de cada jocs crei una estadística que indiqui el nombre de reviews positives, negatives o neutres.

Modifica el prompt si cal.

Guarda la sortida en el directori data amb el nom **exercici2_resposta.json**

Exemple de sortida


```
{
  "timestamp": "2025-01-09T12:30:45.678Z",
  "games": [
    {
      "appid": "730",
      "name": "Counter-Strike 2",
      "statistics": {
        "positive": 1,
        "negative": 0,
        "neutral": 1,
        "error": 0
      }
    },
    {
      "appid": "570",
      "name": "Dota 2",
      "statistics": {
        "positive": 1,
        "negative": 1,
        "neutral": 0,
        "error": 0
      }
    }
  ]
}
```

Exercici 3 (2.5 punts)

Dins de **practica-codi** trobaràs **src/exercici3.js**

Modifica el codi per tal que retorni un anàlisi detallat sobre l'animal.
Modifica el prompt si cal.

La informació que volem obtenir és:

- Nom de l'animal.
- Classificació taxonòmica (mamífer, au, rèptil, etc.)
- Hàbitat natural
- Dieta
- Característiques físiques (mida, color, trets distintius)
- Estat de conservació

Guarda la sortida en el directori **data** amb el nom **exercici3_resposta.json**

```
{
    "analisis": [
        {
            "imatge": {
                "nom_fitxer": "nom_del_fitxer.jpg",
            },
            "analisi": {
                "nom_comu": "nom comú de l'animal",
                "nom_cientific": "nom científic si és conegut",
                "taxonomia": {
                    "classe": "mamífer/au/rèptil/amfibi/peix",
                    "ordre": "ordre taxonòmic",
                    "familia": "família taxonòmica"
                },
                "habitat": {
                    "tipus": ["tipus d'hàbitats"],
                    "regioGeografica": ["regions on viu"],
                    "clima": ["tipus de climes"]
                },
                "dieta": {
                    "tipus": "carnívor/herbívor/omnívor",
                    "aliments_principals": ["llista d'aliments"]
                },
                "caracteristiques_fisiques": {
                    "mida": {
                        "altura_mitjana_cm": "altura mitjana",
                        "pes_mitja_kg": "pes mitjà"
                    },
                    "colors_predominants": ["colors"],
                    "trets_distintius": ["característiques"]
                },
                "estat_conservacio": {
                    "classificacio_IUCN": "estat",
                    "amenaces_principals": ["amenaces"]
                }
            }
        }
    ]
}
```

Exercici 4 (2.5 punts)

Implementa un nou endpoint a xat-api per realitzar anàlisi de sentiment

Haurà de complir els següents requisits

- Estar disponible a l'endpoint POST /api/chat/sentiment-analysis
- Disposar de documentació swagger
- Emmagatzemar informació a la base de dades
- Usar el logger a fitxer

Abans d'implementar la tasca, explica en el requadre com plantejaràs i fes una proposta de json d'entrada, de sortida i de com emmagatzemaràs la informació a la base de dades.

Procés:

- Es rep un text en format JSON.
- Es realitza l'anàlisi del sentiment (positiu, negatiu o neutral) utilitzant una llibreria NLP o una API externa.
- Es guarda el text i el resultat a la base de dades, incloent-hi la puntuació del sentiment i la data.
- El sistema fa servir un logger per registrar el procés en un fitxer.
- Documentem l'endpoint a Swagger per tenir una referència clara.

Base de dades: Es crea una taula sentiment_analysis per emmagatzemar el text, el sentiment, la puntuació i la data.

JSON Entrada:

{ "text": "Aquest producte és increïble, m'ha encantat!"}

JSON Sortida:

```
{"sentiment": "positiu",
  "puntuació": 0.85,
  "data_hora": "2025-02-20T14:45:32.000Z",
  "text": "Aquest producte és increïble, m'ha encantat!"}
```