Nom i Cognoms:	Daniel Pons Solera
URL Repositori Github:	https://github.com/dps32/M06_RA6_PR4.2_PonsDaniel

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-M0486-Tema3-RA6-PR4.2-Punt-Partida-25-26

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 funcionant (al centre te'n facilitem una)

Entrega

- URL de resopitori

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?
2. Analitzant el fitxer server.js, quina és la seqüència correcta per inicialitzar una aplicació Express? Per què és important l'ordre dels middlewares?
3. Com gestiona el projecte les variables d'entorn? Quins avantatges ofereix usar dotenv respecte a hardcodejar els valors?

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?
2. En el fitxer chatController.js, per què és important separar la lògica del controlador de les rutes? Quins principis de disseny s'apliquen?
3. Com gestiona el projecte els errors HTTP? Analitza el middleware errorHandler.js i explica com centralitza la gestió d'errors.

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?
2. Com es documenten els diferents endpoints amb els decoradors de Swagger? Per què és important documentar els paràmetres d'entrada i sortida?
3. Com podem provar els endpoints directament des de la interfície de Swagger? Quins avantatges ofereix això durant el desenvolupament?

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?

2. Com gestiona el projecte les migracions i sincronització de la base de dades? Quins riscos té usar sync() en producció?
3. Quins avantatges ofereix usar un ORM com Sequelize respecte a fer consultes SQL directes?

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?
2. Per què és important tenir diferents transports de logging (consola, fitxer)? Com es configuren en el projecte?
3. Com ajuda el logging a debugar problemes en producció? Quina informació crítica s'hauria de loguejar?

Configuració i Estructura Bàsica

1. Aquesta estructura de directoris ajuda a la separació de responsabilitats, facilitant el manteniment i lectura del codi.
2. L'ordre és important ja que una llibreria pot dependre d'una altre, com per exemple **cors** que depèn d' **express** per funcionar. La seqüència actual es importar **express** i **cors** (per poder rebre peticions), **swagger** per facilitar la documentació de les peticions **REST API**, el **handler d'errors**, les **rutes** per fer peticions i finalment el **logger**.
3. El projecte utilitza un arxiu **.env** per emmagatzemar les variables d'entorn, l'avantatge principal davant de hardcodejar es que tens els valors importants a un mateix lloc i no has de sustituir cada repetició que hi hagi al codi si vols canviar un valor. També permet que puguis pujar tot el codi a un repositori i que només falti el contingut del **.env**, això evita pujar variables crítiques, com podríen ser credencials de la **DB** o **claus d'API** que puguin estar a arxius hardcodejats.

API REST i Express

1. Chatroutes defineix una ruta i la funció que es trucará quan hi hagi una petició d'un tipus a la ruta definida, el mètode **GET** està pensat per recuperar informació del servidor, mentre que **POST** hi és per enviar l'informació. Tècnicament pots fer qualsevol cosa utilitzant només un dels dos, pero tenen diferències com que GET té els paràmetres directament a la URL i es pot accedir desde un navegador, mentre que POST envía els paràmetres al body i no es queda a la caché.
2. Separar la lògica del controlador de les rutes fa que el codi sigui més net i fàcil de mantenir, intentant que sigui tot modular per poder reutilitzar codi i evitar repeticions.
3. El projecte utilitza el errorHandler per centralitzar la gestió dels errors http, capturant qualsevol error que es doni a les rutes o els controllers, evitant així la repetició de codi per capturar errors.

Documentació amb Swagger

1. Swagger s'integra al projecte utilitzant la configuració de swagger.js i els comentaris de chatRoutes.js, aquests serveixen per a que Swagger pugui generar automàticament una documentació de la API.
2. Els endpoints es documenten amb comentaris de Swagger a sobre de cada ruta, indicant la descripció, paràmetres i la resposta.
3. Des de la interfície es poden provar els endpoints directament, per facilitar el desenvolupament validant ràpidament que tot funciona i detectar problemes sense haver d'utilitzar altres eines.

Base de Dades i Models

1. Les relacions entre models a Sequelize s'implementen amb mètodes com belongsTo i hasMany, així pots definir que una conversa té molts prompts o que un prompt es de una conversa, s'utilitza l'UUID com a clau primària per garantir que identifiquem correctament cada prompt i conversa.
2. El projecte gestiona la base de dades amb sync(), que crea o actualitza les taules segons els models, utilitzar sync() a producció és arriscat perquè pot esborrar dades o fer canvis inesperats.
3. Utilitzar un ORM com Sequelize simplifica l'interacció amb la base de dades, permet escriure codi més llegible, facilita les modificacions al projecte i evita els errors que pugui donar una consulta SQL habitual.

Logging i Monitorització

1. El logging a logger.js s'implementa amb la llibrería winston, que permet registrar missatges amb format JSON, els nivells son info, warn, error i debug.
2. Tindre diferents llocs on mostrar logs es útil perque pots voler veure uns logs en temps real a la consola, pero també es possible que vulguis guardar altres més importants que no vols que es perdin a la consola dins d'un fitxer.
3. El logging ajuda a debugar perquè deixa un rastre de per on està passant l'aplicació, així pots asegurarte de que una part del codi s'està executant.

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 joc, 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 la plantejaràs i fes una proposta de json d'entrada, de sortida i de com emmagatzemaràs la informació a la base de dades.

La meva proposta sería simplement enviar d'entrada el text d'una opinió d'un producte, i que després es retorni una puntuació del 0 al 10, sent 0 una opinió molt negativa i un 10 una molt positiva, basat en aquesta puntuació treure el sentiment (positiu/neutre/negatiu).

Entrada:

```
{
  "text": "El producte és de una molt bona qualitat"
}
```

Sortida:

```
{
  "id": "9356d836-5fb0-4662-87cd-9302d477bdde",
  "text": "El producte és de una molt bona qualitat",
  "score": 9.2,
  "sentiment": "positiu",
  "model": "qwen2.5:7b",
  "createdAt": "2026-01-26T16:00:20.734Z"
}
```

Finalment retornem l'id que es crea a la base de dades per identificar la petició, el text d'entrada, la puntuació de positivitat, el sentiment, el model utilitzat per extreure el sentiment, i la data de creació de la petició. A la base de dades es guarda aquesta mateixa informació amb Sequelize, afegint també la resposta completa del model per poder revisar la resposta en cas de rebre un error o que no es pugui processar correctament el text.