Nom i Cognoms:	MONICA DOMINGUEZ AGUIRRE
URL Repositori Github:	https://github.com/monica438/AMS2_M0486-Tema3-PR42.git

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 repositori

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. És important estructurar el codi en aquests directoris perquè facilita molt la llegibilitat del projecte i l'enteniment del seu funcionament.
   Alguns dels avantatges que té aquesta organització serien: la separació clara del codi segons la seva funcionalitat, facilita trobar el segment de codi que vols modificar, és més fàcil entrar a modificar un arxiu concret que està clarament separat que no un arxiu que tingui tot el projecte…

2. La seqüència correcta és: carregar variables d'entorn, fer les importacions requerides, crear instància d'Express, configuració dels middlewares principals, configuració de Swagger per la documentació de l'API, configuració de middleware de logging personalitzat, configuració del logger d'Express, registre de les rutes principals, gestió centralitzada d'errors, indicar port per defecte, gestió d'errors no controlats, gestió del senyal SIGTERM, inicialitzar el servidor (connectar amb la base de dades, sincronitzar els models, iniciar el servidor HTTP) i exportar l'app per tests.
   L'ordre dels middlewares és important perquè s'executen en l'ordre en què s'afegeixen, és a dir si es posés el middleware de rutes abans del express.json() no es podrien llegir els cossos de les peticions correctament.

3. El projecte gestiona les variables d'entorn utilitzant el paquet dotenv per carregar les variables definides en .env.
   Els avantatges que ofereix serien que és més segur ja que no has d'incloure credencials al codi font, facilita la modificació de la configuració i permet controlar les configuracions de diferents entorns (dev, test, prod).

API REST i Express

4. El routing s'implementa utilitzant express.Router() per definir les rutes /prompt, /conversation/{id} i /models. I cada ruta apunta a una funció del controlador.
   La diferència entre GET i POST és que GET recupera dades del servidor sense modificar-les i POST crea o modifica dades al servidor. S'ha d'utilitzar GET per llegir i POST per crear o modificar.

5. És important separar la lògica del controlador de les rutes perquè manté el codi net i modular i permet reutilitzar la lògica si és necessari evitant la duplicació del codi.
   Els principis aplicats serien Separation of Concerns i Single Responsibility Principle.

6. Els errors es gestionen passant pel middleware errorHandler(err, req, res, next).
   Hi ha molts tipus d'errors i en comptes de tractar-se tots per separat, errorHandler és capaç de diferenciar els errors de Sequelize retornant el codi HTTP que correspongui i per errors no controlats mostra la informació detallada en l'entorn development. També, registra stack trace path i mètode amb el logger.

Documentació amb Swagger

7. Per integrar la documentació amb el codi s'utilitza swagger-jsdoc que genera la documentació a partir de comentaris JSDoc.
   Els beneficis que aporta són que la documentació sempre està sincronitzada amb el codi i evita errors al definir l'API.

8. Per documentar els diferents endpoints s'utilitza la sintaxi JSDoc amb tags com swagger, summary, tags, requestBody i responses.
   És important documentar paràmetres d'entrada i sortida per tal que els clients puguin consumir correctament l'API i redueix errors a l'indicar clarament quins paràmetres s'han d'introduir.

9. Per provar els endpoints es pot fer des de la interfície de Swagger introduint valors de paràmetres i la petició i s'envia i es pot veure la resposta JSON.
   Els avantatges són que els endpoints es poden testejar fàcilment sense necessitat d'utilitzar res més, es pot verificar la documentació i implementació i és útil per debuggejar.

Base de Dades i Models

10. S'implementen les relacions entre models amb Conversation.hasMany… i Prompt.belongsTo…
    S'utilitza UUID com a clau primària perquè evita dependre de IDs autoincrementats en diferents instàncies i és millor per identificar registres únics a nivell global.

11. Per gestionar la sincronització s'utilitzar sequelize.sync() a server.js.
    Els riscos que té és que si s'utilitza force: true s'eliminen totes les dades existents i amb alter: true es poden causar errors si hi ha dades incompatibles.

12. Els avantatges són que facilita el manteniment i llegibilitat del codi, es gestiona de forma integrada les validacions, relacions, transaccions i migracions i es permet treballar amb models i objectes.

Logging i Monitorització

13. El logging estructurat s'implementa amb Winston, incloent timestamp, nivell, stack i metadata.
    Els nivells de logging són debug (per donar detalls per desenvolupament), info (per donar informació general de per on va el codi), warn (per avisar de coses importants) i error (per informar d'errors que poden afectar a la funcionalitat).

14. És important tenir diferents transports perquè la consola et dona velocitat i el fitxer permet tenir un registre del que ha passat per poder analitzar problemes en conjunt i per altres equips.
    Es configuren amb winston.transports.Console i winston.transports.DailyRotateFile.

15. El logging ajuda a debugar perquè permet reconstruir tot el que ha passat i veure exactament que ha passat en cada moment.
    S'haurien de loggejar errors amb els stack traces, data i hora de cada esdeveniment, inputs i outputs de les funcions i estats de connexió amb serveis externs com Ollama.

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 seria fer la petició seguint el següent procés: enviar petició POST al endpoint /api/chat/sentiment-analysis, el model analitza el text, es genera un resultat en format json que mostra el sentiment que el model prediu pel text concret, es guarda la informació a la base de dades, es registra el que s'ha fet als logs i es retorna el resultat de l'anàlisi.
S'utilitzaran Express.js, Sequelize, Winston i Swagger.

La prosposta d'entrada seria:

```
{
    "text": "Aquest és un comentari positiu",
    "userId": "12345",
    "model":"qwen2.5v1:7b"
}
```

I la de sortida:

```
{
    "text": "Aquest és un comentari positiu",
    "userId": "12345",
    "sentiment": "positive",
    "confidence": 0.92,
    "model": "qwen2.5v1:7b",
    "timestamp": "2026…"
}
```

A la base de dades es guardarien els següents camps:
id (UUID)
userId(VARCHAR(255))
text(TEXT)
sentiment(VARCHAR(50))
confidence(FLOAT)
model(VARCHAR(100))
timestamp(DATETIME)