Nom i Cognoms:	Marc Arqués Marimón
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?**

   Permet dividir el codi segons responsabilitats, millorant la claredat, mantenibilitat i escalabilitat. Facilita la col·laboració entre desenvolupadors i redueix el risc d'errors 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?**

   La seqüència correcta és: carregar mòduls, crear l'app, afegir middlewares (com express.json()), definir rutes, afegir el middleware de gestió d'errors i finalment iniciar el servidor amb listen().
   L'ordre és important perquè Express executa els middlewares en l'ordre en què s'han declarat, i alguns depenen dels anteriors (per exemple, accedir al body d'una petició POST).

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 les variables des d'un fitxer .env. Això permet mantenir configuracions sensibles fora del codi font, facilitar el canvi d'entorn (producció, desenvolupament) i millorar la seguretat i flexibilitat.

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?**

Es fa servir un objecte Router d'Express per definir les rutes i associar-les a funcions del controlador amb .get() i .post().

GET s'utilitza per consultar dades (sense modificar-les), i POST per enviar o crear dades noves al servidor.

2. **En el fitxer chatController.js, per què és important separar la lògica del controlador de les rutes? Quins principis de disseny s'apliquen?**

   Ajuda a aplicar el principi de responsabilitat única, facilita les proves i millora l'organització. També permet reutilitzar la lògica en diferents rutes i mantenir el codi modular i net.

3. **Com gestiona el projecte els errors HTTP? Analitza el middleware errorHandler.js i explica com centralitza la gestió d'errors.**

   Tots els errors són capturats per un middleware d'error (errorHandler) que envia una resposta amb un missatge i un codi d'error. Això centralitza el tractament d'errors, evita repeticions i permet una resposta consistent a l'usuari.

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?**

   S'integra mitjançant comentaris especials a les rutes (`chatRoutes.js`) que Swagger llegeix per generar documentació automàtica. Ajuda a entendre i provar l'API fàcilment.

2. **Com es documenten els diferents endpoints amb els decoradors de Swagger? Per què és important documentar els paràmetres d'entrada i sortida?**

   Es defineixen paràmetres i respostes amb comentaris `@swagger`. És important per saber què s'ha d'enviar i què es rebrà.

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

   Swagger UI permet provar endpoints des del navegador. És útil per fer proves ràpides sense eines externes.

4.

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?**

   S'usen `hasMany` i `belongsTo` per relacionar models. El `UUID` garanteix identificadors únics.

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

   Es fan servir migracions. `sync()` pot esborrar dades en producció si no es fa servir bé.

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

   Sequelize facilita el treball amb la base de dades, evita SQL manual i permet treballar amb objectes.

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?**

   Es fa amb `winston`, que permet estructurar els logs. Nivells: `info`, `warn`, `error`, `debug`.

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

   Diversos transports permeten enviar els logs a consola o fitxer. Això es configura a `logger.js`.

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

   Ajuda a trobar errors. Cal registrar errors, accions importants i problemes de connexió.

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

Plantejament i proposta prèvia abans d'implementar el codi:

1. **Afegir la ruta a `chatRoutes.js`**
   Es crearà una nova ruta amb el mètode `POST /api/chat/sentiment-analysis`, que cridarà al mètode corresponent del controlador.

2. **Implementar el controlador a `chatController.js`**
   El controlador validarà que es rep un camp `text`, farà la crida a l'API d'Ollama per analitzar el sentiment, guardarà el resultat a la base de dades i retornarà el sentiment detectat com a resposta.

3. **Crear el model `SentimentAnalysis.js`**
   Es definirà una taula amb els camps:

- `id` (UUID)
- `text` (TEXT)
- `sentiment` (STRING)
- `createdAt` (DATE)

4. **Afegir la documentació a Swagger**
   Es documentarà el nou endpoint indicant el mètode `POST`, la petició d'entrada (`text`) i possibles respostes (`positive`, `negative`, `neutral`, `error`).

5. **Afegir logging amb winston**
   Es registrarà cada anàlisi al fitxer de logs: inclourà el text rebut, el sentiment obtingut i qualsevol error que es produeixi durant la petició.