Nom i Cognoms:	Joan Martinez Marin
URL Repositori Github:	https://github.com/xuanor/M06-UF04-PR4.2

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?**

   Facilita el manteniment, escalabilitat i col·laboració al separar responsabilitats clarament. Permet trobar i modificar components específics amb més facilitat.

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

   Seqüència: carregar dependències, configurar middlewares globals, definir rutes, gestionar errors i iniciar el servidor. L'ordre és crucial perquè els middlewares s'executen seqüencialment i poden afectar els següents.

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

   Usa dotenv per carregar variables des d'un fitxer .env. Això permet separar configuracions sensibles del codi i facilitar canvis entre entorns sense modificar el codi.

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?**

   El routing es defineix amb mètodes com router.get() o router.post(). GET s'utilitza per obtenir dades, mentre que POST s'utilitza per enviar dades al servidor, com formularis o cossos JSON.

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

   Facilita la reutilització, proves i manteniment del codi. S'apliquen principis com la separació de responsabilitats i la modularitat.

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

   Centralitza la captura d'errors personalitzant respostes segons el tipus d'error i evita duplicar codi en les rutes.

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 afegint comentaris específics al codi i configurant Swagger per generar documentació automàtica. Facilita la comprensió de l'API i redueix errors en el consum d'endpoints.

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

   Els decoradors defineixen la descripció, paràmetres i respostes esperades. Això ajuda a clarificar com utilitzar l'API i millora la col·laboració entre desenvolupadors.

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

   Swagger UI permet enviar sol·licituds directament des del navegador. Això simplifica les proves i detecta problemes sense eines externes.

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?**

   Sequelize utilitza mètodes com hasMany i belongsTo per definir relacions. UUID garanteix identificadors únics universals, útils en entorns distribuïts.

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

   Usa migracions per controlar canvis estructurals. sync() pot sobrescriure dades en producció si no es configura correctament.

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

   Proporciona abstracció, evita errors comuns i facilita treballar amb múltiples bases de dades sense escriure SQL específic.

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?**

   Usa eines com winston per estructurar logs. Nivells comuns: info (informació general), warn (advertències), error (errors crítics), debug (detalls tècnics per depuració).

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

   Permet registrar logs en diferents llocs segons la necessitat (fitxers per a anàlisi històrica, consola per depuració immediata). Configurat a través de winston amb múltiples transports.

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

   Proporciona un registre d'esdeveniments per identificar l'origen d'errors. S'haurien de loguejar errors, sol·licituds importants i punts crítics de l'aplicació.

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.

1. **Afegir la ruta a chatRoutes.js**
    - Crear una nova ruta POST /api/chat/sentiment-analysis que cridi el controlador corresponent.
2. **Implementar el controlador a chatController.js**
    - Validar que es rep un text en la petició.
    - Enviar el text a l'API d'Ollama per obtenir l'anàlisi de sentiment.
    - Guardar el resultat a la base de dades.
    - Retornar la resposta amb el sentiment detectat.
3. **Crear el model Sequelize SentimentAnalysis.js**
    - Definir una taula amb els camps: id, text, sentiment i createdAt.
    - Assegurar que cada anàlisi es desa correctament.
4. **Afegir la documentació a Swagger**
    - Especificar el nou endpoint, els paràmetres d'entrada i les possibles respostes.
5. **Afegir logging amb winston**
    - Registrar les peticions i respostes en un fitxer per facilitar el seguiment i la depuració d'errors.