Nom i Cognoms:	Adrián Casado Aguilera
URL Repositori Github:	https://github.com/AdrianCasado-IETI/DAM-M06-UF04-PR4.2-24-25-Punt-Partida

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?
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 (o una altra) organització de codi garantitza que el codi sigui escalable amb facilitat, a part de ser fàcil d'entendre per futurs desenvolupadors i per tu mateix més endavant. És molt més fàcil trobar un error o fer una actualització sobre un codi ordenat que no sobre una llista interminable d'arxius sense sentit.
2. Primer haurem de fer les exportacions, després fer les comprobacions que necessitem amb els middlewares, més tard fer el desenvolupament dels endpoints, i finalment iniciar l'aplicació. És important respectar l'ordre dels middlewares per assegurar-se des del principi de complir amb els requisits que hi imposem a l'accés a l'aplicació.
3. Les variables d'entorn ens ofereixen utilització més segura de valors que potser no volem que estiguin disponibles per a tothom. Ens pot servir per claus, URLs o altres conceptes semblants. També les podem utilitzar per aquelles coses que necessitem aplicar a tot el sistema com a constant i que en principi no canviaran, per a poder fer més fàcil canviar-les més endavant si fos necessari.

API REST i Express

1. El routing en express s'implementa indicant quins son els endpoints que utilitzarem, i passant-hi com a funció les accions que volem fer quan l'usuari accedeixi a aquests endpoints. Els endpoints GET serveixen per agafar informació publicada al servidor, mentres que els POST serveixen per actualitzar-hi informació al servidor. El mètode POST és més segur per fer transferència de dades.
2. És important fer aquesta separació per separar la lògica del programa amb les rutes, que no es necessiten entre elles per funcionar. Amb això, seguim el model d'abstracció que necessitem per fer un bon programa.
3. L'error handler agafa els error i els mostra amb un logger. El logger és capaç d'ensenyar els errors en temps real a la pantalla, però també els guarda a un arxiu per poder accedir-hi més endavant, per si necessitem fer alguna revisió. Més tard, retorna l'error amb un status HTTP d'error, per notificar aquest error a l'usuari, que el gestionarà l'aplicació client.

Documentació amb swagger

1. El swagger aplica un comentari que entendrà la configuració per crear una documentació. Al codi és només un comentari que ajuda a entendre el funcionament del codi.
2. El tipus d'endpoint es notificquen amb el decorador @swagger, seguit de l'endpoint i més tard el mètode. A partir d'aquí, s'explica tot allò que referencia al codi. És important deixar clar els paràmetres d'entrada i sortida per entendre el codi sense necessitat d'analitzar-lo.
3. A la interficie de swagger podem utilitzar els endpoints per probar-los inserint directament els paràmetres que necessitem. Això ofereix evidenment unes millores perquè no necessitem programar res per fer proves.

Bases de dades i models

1. Utilitzant sequelize, es formen relacions amb les funcions belongsTo i hasMany, que estableixen el tipus de relació que formen. S'utilitzen UUID per generar un identificador únic que no es pot repetir entre diferents tipus d'entitats.
2. Sequelize fa una connexió automàtica amb la base de dades segons la seva configuració, pel que només hem de garantitzar que guardem les dades a sequelize perquè aquests canvis s'apliquin. El risc d'utilitzar sync() és bàsicament que podem sofrir pèrdues de dades segons com ho tinguem configurat.
3. Primer de tot, ens simplifica la feina. També garanteix una protecció front a segons quins atacs, com per exemple atacs d'inyecció SQL. I finalment, en cas de migració del servei de Bases de Dades o d'actualització, no ens afectarà al nostre programa, ja que Sequelize fa aquesta abstracció.

Logging i monitorització

1. El logging estructurat genera logs que no son text lliure directament, sino que genera fitxers fàcilment interpretables. Existeixen nivells de debug, info (per informació d'events, per exemple), error (per problemes al codi), warn (per errors no crítics), http (per peticions rebudes i enviades), verbose i silly (per fer comentaris que ens poden ser útils).
2. És important per diversos motius. La consola és important per trobar errors en temps real. És especialment útil quan estem en procés de debug, però també per si estem seguint un problema que succeeix en producció. Però també podem necessitar un fitxer per a garantitzar la traçabilitat d'un error. Per exemple, podem imaginar que un usuari té un problema, podrem dirigir-nos al moment exacte en que ha sofert aquest problema (que potser no ho comunica de forma inmediata) sense risc a que la consola hagi esborrat el log.
3. El logging en producció ens pot ajudar per solucionar bugs que no hem pogut localitzar en desenvolupament, però també per errors que no es poden reproduir pel medi en que es reprodueix la producció. La informació que s'ha de loggejar és sobretot, els missatges d'accions, warnings i errors.

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.

L'endpoint /api/chat/sentiments-analysis analitza el sentiment d'un missatge. Es rep un text, es processa el text per determinar si és positiu, neutre o negatiu, i es guarda el resultat a la base de dades. Retorna el sentiment.

JSON d'entrada:

```
{
  "text": "Estic molt content avui!"
}
```

JSON de sortida:

```
{
  "id": "uuid",
  "text": "Estic molt content avui!",
  "sentiment": "positive",
  "message": "Anàlisi de sentiment registrada correctament"
}
```