Nom i Cognoms:	LAURA TORO
URL Repositori Github:	https://github.com/laurattt/DAM2MP06-Acceso-Datos.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 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?

   RTA/ Esta organización ayuda a que el código sea legible junto a su organización, de esta manera si llega a ocurrir una incidencia dentro del programa se podrá saber el lugar del error con más facilidad

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

   RTA/ Para inicializar una aplicación express se necesita

   - Crear la instancia de Express
   - Se configuran los middlewares principales (CORS, parser JSON)
   - Se configuran middlewares auxiliares (documentación, logging o personalizados)
   - Registrar rutas de la aplicación
   - Configuración de gestión de errores
     - app.use(...)

   Middlewares: función que se ejecuta en el flujo de una petición HTTP (procesa, modifica, registra, maneja errores, etc)

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

   RTA/ Estas variables se encuentran en el archivo .env, donde se definen las configuraciones sensibles como claves, tokens, puertos, etc. Las ventajas de usar dotenv es que permite acceder a estas a través de process.env, de esta forma las claves reales no están hardcodeadas directamente en el code, sino almacenadas en un archivo externo teniendo así mayor seguridad

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?

   RTA/ Se implementa utilizando express.Router() y funciona como un contenedor de rutas donde son separadas en otro archivo, de esta manera se verá más legible y organizado el código

   - GET: Este método sirve para obtener datos (consulta)
     - .get(/conversation/:id) -> devuelve id

   - POST: Este método sirve para crear o enviar info nueva al server
     - .post('/prompt', registerPrompt) -> envia prompt

   ROUTING: Decidir qué code se ejecuta según la URL y su método HTTP

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

   RTA/ Es importante separarlos, ya que en chatController.js encontramos la lógica de ejecución la cual puede ser reutilizada y en chatRoutes.js solo las rutas que gestionan el tránsito, cada archivo tiene una responsabilidad diferente y esta organización asegura un mejor entendimiento

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

   RTA/ Dentro del archivo errorHandler.js se encuentra la lógica la cual se comporta como una red de seguridad donde captura y controla los posibles errores que pueden aparecer tras ejecutar nuestro code principal

   - Recibe los errores desde cualquier parte de la aplicación
   - Indica el código de estado HTTP adecuado (404 - 500 - etc)
   - Registra errores para depurar

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?

   RTA/ La documentación se integra mediante comentarios especiales en los archivos de rutas, esta integración es importante, ya que nos brinda una simulación de lo que podría llegar a ser la interacción con el servidor

   SWAGGER: Se puede ver como un manual interactivo de una API

Cada endpoint consta con un bloque de comentarios @swagger, estos comentarios describen puntos como:

- URL endpoint
- Método HTTP
- Parámetros de entrada
- Cuerpo de la petición
- Posibles RTA HTTP

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

   RTA/ Se documentan debajo de la anotación @swagger de tal manera que su sintaxis se ve así:

```
/**
 * @swagger
 * /api/chat/conversation/{id}:
 *   get:
 *     summary: Obtenir una conversa per ID
 *     tags: [Conversations]
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: string
 *         required: true
 *         description: UUID de la conversa
 *     responses:
 *       200:
 *         description: Conversa trobada
 *       404:
 *         description: Conversa no trobada
 */
```

   Por parte de los parámetros de entrada y salida, son importantes de documentar para que la API sea clara, segura, mantenible y fácil de utilizar

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

   RTA/ Se puede probar utilizando la interfaz web de Swagger donde

   - Se selecciona endpoint que se quiere probar
   - Se complementan los parámetros de entrada
   - Se presiona el botón de probar donde swagger envía la petición al servidor
   - Obtenemos la respuesta

Esto tiene como ventajas el rápido testeo sin herramientas externas, también se ve como una validación inmediata de la API, detección de errores, brinda documentación interactiva. Ofrece todo esto estando siempre sincronizada con la API

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?

   RTA/ Se ve en la siguiente imagen

   ```
   // Definim les relacions aquí, un cop tots els models estan carregats
   Conversation.hasMany(Prompt, { foreignKey: 'ConversationId', onDelete: 'CASCADE' });
   Prompt.belongsTo(Conversation, { foreignKey: 'ConversationId' });
   ```

   Se utiliza la clave foránea ConversationId en la tabla prompts y se establece una relación 1:N

   Se utiliza UUID como clave primaria para sostener una buena seguridad entre los datos y su accesibilidad

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

   RTA/ La base de datos se gestiona con sequelize el cual ofrece el método nombrado en la pregunta sync( ), este crea tablas que no existen y dependiendo la forma en que se configuren se pueden reiniciar tablas existentes. Los riesgos identificados en su utilización son la pérdida de datos, cambios no controlados, falta de historial de cambios, en sí se recomienda usar migraciones controladas en prod

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

   RTA/ Sequelize permite trabajar con la bbdd usando objetos y métodos JS, gestiona modelos y relaciones de manera intuitiva también soporta múltiples motores de bases de datos, añade validaciones y mantiene un flujo de migraciones y sincronización controlado. Todo esto reduce errores y hace el desarrollo más rápido y seguro en comparación con consultas SQL directas

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?

   RTA/ La estructura que veremos por logging es

   - Timestamp
   - Nivel del log (error, info,etc)
   - Mensaje
   - Metadata
   - Stack Trace (en caso de errores)

   Los logs se guardan en consolas y archivos rotativos, también existe un middleware que registra las peticiones HTTP

   Los niveles que se encuentran son error, warn, info y debug los cuales permiten clasificar la severidad de los eventos

   WINSTON: librería de logging para NodeJS

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

   RTA/ ES importante porque permite separar logs por uso, esto garantiza la persistencia, facilita monitorización y dentro del proyecto están configuradas usando transports.Console - transports.DailyRotateFile cada uno con su formato y opciones

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

   RTA/ Los loggins registran eventos y errores críticos mientras la app sigue funcionando, permite recrear lo que pasó antes de un fallo y facilita encontrar la causa principal de errores sin interrumpir el servicio. Esto siempre ayudará a la depuración efectiva

   Un ejemplo de uso se puede ver tras una petición HTTP, si un endpoint falla se podrá ver exactamente qué petición causó el error, entre otras características

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.

1. ENDPOINT: /api/chat/sentiment-analysis

2. JSON DE ENTRADA:
{
"conversationId": 1234,
"text": "bla bla bla, esto lo debe analizar"
}

3. JSON DE SALIDA:
{
"conversationId": 1234 (UUID),
"text": "bla bla bla, esto lo debe analizar",
"sentiment": "positivo",
"timestamp": "2026-02-01T14:30:00Z"
}

4. BBDD data almacenada - id (UUID) - conversationId (UUID) - text - sentiment - createdAt
5. LOGGER logger.info('Sentiment analysis', { conversationId, text, sentiment, timestamp: new Date().toISOString() });
6. SWAGGER - Mostrar método utilizado (POST) - Request body (id y text) - Respuestas posibles (201 creado - 400 error)