Nom i Cognoms:	Jonathan Rueda Neiro
URL Repositori Github:	https://github.com/Zonoik09/M06-UF04-PR4.2-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?

Organizar el código en una estructura de directorios es importante para mantener una separación clara para facilitar la localización y modificación de los archivos. Al tenerlo más organizado da facilidad para escalar y mantener el proyecto, haciendo que el trabajo en equipo sea más sencillo con la posibilidad de modificar sin intervenir en el trabajo de otros.

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 secuencia correcta es la siguiente:

1. Cargar variables de entorno: Se utiliza dotenv.config() para cargar las variables introducidas en el .env antes de cualquier otra configuración para garantizar que los valores estén disponibles desde el principio.

2. Importar dependencias: Se cargan módulos esenciales, la configuración de la base de datos (Sequelize), las rutas, y los middlewares.

3. Crear la instancia Express: Se crea la aplicación con "const app = express();".

4. Configurar middlewares globales:

- cors()
- express.json()
- swaggerUI

- expressLogger

5. Registrar las rutas principales: Define las rutas de la API (app.use('/api/chat', chatRoutes);)

6. Gestión centralizada de errores: Se coloca después de las rutas para interceptar errores.

7. Inicialización del servidor: Se conecta a la base de datos, se sincronizan los modelos y se inicia el servidor.

8. Gestión de errores globales: Se interceptan errores no controlados y la señal SIGTERM para un cierre seguro del servidor.

El orden de los middlewares es crucial porque se procesan de manera secuencial. Si se coloca de una posición distinta por ejemplo, colocar "express.json()" después de las rutas puede causar errores al procesar el cuerpo de las peticiones, y definir la gestión errores antes de las rutas impediría captar fallos en ella.

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

El proyecto gestiona las variables utilizando dotenv, que carga los valores desde un archivo .env al inicio de la aplicación. Es muy útil porque sirve para mantener la información sensible fuera del código y evitar hardcodear.

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 se hace con el objeto "router" de Express, que permite gestionar las rutas para el servidor.

La principal diferencia entre los métodos GET y POST, es que GET se utiliza para obtener información desde el servidor, como recuperar recursos o consultar datos. Es menos seguro porque a través del enlace se pueden exponer datos importantes.
Por otro lado, POST se usa para enviar datos al servidor con el fin de crear,actualizar o procesar información y estos datos se envían en el cuerpo de la solicitud, lo que lo hace más seguro que el GET.

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

Separar la lógica del controlador de las rutas facilita la gestión del código, siguiendo principios de diseño como la responsabilidad única, la modularidad y la reutilización. Las rutas se centran en gestionar las solicitudes, mientras que los controladores se dedican a la lógica de negocio, lo que hace que cada parte del código tenga una responsabilidad clara. Esta separación permite una mejor escalabilidad, facilita la actualización o la adición de nuevas funcionalidades sin afectar otras partes de la aplicación, y mejora la estabilidad del sistema, ya que los controladores se pueden probar de manera independiente.

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

errorHandler.js centraliza la gestión de errores en el proyecto, asegurando que los errores sean capturados y tratados de manera uniforme. Registra los detalles de cada error, como el mensaje y el stack trace, para facilitar la depuración. Si el error está relacionado con la validación de datos o duplicación de registros en Sequelize, responde con un código de error 400 y los detalles específicos del error. Para otros tipos de errores, devuelve un error 500, mostrando un mensaje genérico al usuario, pero detallado en desarrollo. Esta centralización mejora la mantenibilidad, coherencia, seguridad y facilidad de depuración del sistema.

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?

La integración de Swagger en el código se realiza a través de comentarios en las rutas de la API, los cuales siguen el formato OpenAPI 3.0 para describir operaciones, parámetros, respuestas y códigos de estado. En el archivo swagger.js se configura Swagger, especificando detalles como el título y la versión de la API, y la ubicación de las rutas. En chatRoutes.js, cada ruta tiene comentarios que detallan las solicitudes y respuestas, lo que permite generar documentación automática. Esta aproximación ofrece varios beneficios: asegura que la documentación esté siempre sincronizada con el código, facilita el entendimiento y uso de la API, mejora la mantenibilidad, permite la generación de clientes y pruebas automáticas, y fomenta una mayor calidad del código y transparencia.

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

Los endpoints se documentan con los decoradores de Swagger mediante comentarios específicos que describen las operaciones, como las rutas, los parámetros de entrada y las respuestas esperadas. Estos comentarios se añaden directamente al código, siguiendo la sintaxis OpenAPI para proporcionar información sobre cada método HTTP, los parámetros que se pasan a las solicitudes y las posibles respuestas con sus respectivos códigos de estado. Documentar los parámetros de entrada y salida es fundamental porque ayuda a garantizar que los usuarios de la API comprendan claramente qué esperan las solicitudes y qué devolverá la API, mejorando la comprensión, la seguridad y la usabilidad de la interfaz, así como facilitando la integración y las pruebas automáticas.

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

Podemos probar los endpoints directamente desde la interfaz de Swagger utilizando Swagger UI, que permite interactuar con la API de manera sencilla. A través de esta herramienta, los desarrolladores pueden ver todos los endpoints disponibles, introducir parámetros de entrada, ejecutar las solicitudes y ver las respuestas en tiempo real, lo que facilita la validación de su funcionamiento. Durante el desarrollo, esto ofrece varias ventajas, como una mayor facilidad de uso al evitar herramientas externas, una visualización clara de la documentación interactiva, mayor rapidez en las pruebas y una verificación inmediata de las respuestas, lo que facilita la detección de errores y mejora la eficiencia en el proceso de desarrollo y depuración.

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?

En los modelos Conversation.js y Prompt.js, las relaciones entre modelos se gestionan mediante los métodos belongsTo y hasMany de Sequelize. El modelo Prompt está relacionado con Conversation a través de Prompt.belongsTo(Conversation), lo que significa que cada registro de Prompt tiene una clave foránea que apunta a Conversation. A su vez, Conversation.hasMany(Prompt) establece que una conversación puede tener múltiples prompts asociados. El uso de UUID como clave primaria se debe a que ofrece varias ventajas, como ser único a nivel global, lo que facilita la integración entre sistemas distribuidos, además de ser difícil de predecir, lo que mejora la seguridad, y no depender del orden de inserción, lo que es útil en bases de datos distribuidas. Esto hace que los UUID sean ideales para aplicaciones escalables y seguras.

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

El proyecto utiliza Sequelize para gestionar la conexión con la base de datos MySQL y configurar los parámetros de la conexión, como el nombre de la base de datos, el usuario, la contraseña y el dialecto de la base de datos. La sincronización de la base de datos se puede realizar mediante el uso de sequelize.sync(), que crea o actualiza las tablas en función de los modelos definidos. Aunque este método puede ser útil durante el desarrollo, su uso en producción conlleva ciertos riesgos, ya que puede provocar la pérdida de datos si se realiza una actualización del esquema de la base de datos sin tener en cuenta los datos existentes. Además, sync() puede sobrescribir tablas o cambiar la estructura de la base de datos, afectando la estabilidad de la aplicación. Por esta razón, en entornos de producción, se recomienda utilizar migraciones controladas manualmente, que permiten aplicar cambios de manera segura y gestionada a través de archivos de migración, evitando así sorpresas o pérdidas de datos en la base de datos.

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

Usar un ORM como Sequelize ofrece ventajas significativas frente a las consultas SQL directas. Facilita la abstracción de la base de datos, permitiendo interactuar con ella a través de modelos en lugar de escribir consultas SQL manualmente, lo que hace el código más legible y fácil de mantener. Además, mejora la seguridad al proteger contra ataques de SQL injection, ya que genera las consultas de forma segura. Sequelize también simplifica la gestión de relaciones entre tablas y facilita la portabilidad entre diferentes bases de datos. A través de migraciones y sincronización, permite gestionar los cambios en el esquema de forma controlada. Asimismo, mejora la modularidad y escalabilidad del código, favorece la reutilización de modelos y facilita la realización de pruebas unitarias e integradas.

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?

   El logging estructurado en el proyecto se gestiona con winston, registrando eventos con información detallada como fecha, nivel de log, mensaje y errores. Los niveles de logging son: **'info'** para eventos normales, **'warn'** para advertencias, **'error'** para fallos y **'debug'** para detalles de desarrollo. Los logs se guardan en archivos con rotación y compresión para facilitar su gestión y análisis.

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

   Tener diferentes transportes de logging (consola y archivo) es clave para adaptarse a distintos entornos: la consola es útil para el desarrollo, mientras que los archivos permiten almacenar logs a largo plazo, esenciales en producción. En el proyecto, se configura un transporte de consola activo y, si se especifica, también se almacenan los logs en archivos con rotación diaria para una mejor gestión y análisis.

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

   El logging ayuda a depurar problemas en producción al proporcionar un registro detallado de las actividades de la aplicación, facilitando la identificación de errores y el seguimiento de la ejecución. La información crítica que se debe loguear incluye errores, excepciones, peticiones HTTP, tiempos de respuesta, así como información del sistema como la IP, el usuario y el agente de usuario. Esto permite detectar anomalías y rastrear el flujo de la aplicación para solucionar problemas de manera más rápida.

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

Enfoqué el planteamiento para que coincidiera con el ejercicio 2, utilizando el modelo, el prompt y la opción de stream, manteniendo una salida consistente.

Entrada:
El usuario debe proporcionar los siguientes campos:

- **model**: Especifica el modelo a utilizar, por ejemplo, "llama3:latest".
- **prompt**: Contiene el texto a analizar.
- **stream**: Indica si la respuesta se transmite en tiempo real. Aunque es opcional, se mantiene para coherencia con el ejercicio 2.

Ejemplo de entrada:

```
{
  "model": "llama3:latest",
  "prompt": "Analyze the sentiment of this text and respond with only one word (positive/negative/neutral) Is a good game!",
  "stream": false
}
```

Salida:
La respuesta conserva el formato de entrada, añadiendo el campo sentiment con el resultado del análisis.

Ejemplo de salida:

```
{
  "model": "llama3:latest",
  "prompt": "Analyze the sentiment of this text and respond with only one word (positive/negative/neutral) Is a good game!",
  "sentiment": "positive",
  "stream": false
}
```