Nom i Cognoms:	Victor Prieto Gallegos
URL Repositori Github:	https://github.com/vipriga06/DAM2-AccesDadesVPrieto/tree/main/RA4/PT4.2

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-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?

   Per una millor organització, mantenibilitat i escabilitat. Després també millora amb

   els tests.

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

   -Importar dependències

   -Configurar middlewares bàsics (express.json(), express.urlencoded())

   -Configurar middlewares de tercers (cors, helmet, morgan)

   -Configurar rutes

   -Configurar maneig d'errors

   -Iniciar servidor

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

   // .env

   CHAT_API_OLLAMA_URL=http://localhost:11434

   CHAT_API_OLLAMA_MODEL=qwen2.5vl:7b

   // chatController.js

   const OLLAMA_API_URL = process.env.CHAT_API_OLLAMA_URL;

   ——

   Per el tema avantatges, ofereix seguretat, portabilitat, flexibilitat i col·laboració.

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?

   // Definició de rutes amb verbs HTTP

   router.post('/prompt', registerPrompt); // CREATE

   router.get('/conversation/:id', getConversation); // READ

   Diferència GET vs POST:

   -GET: Sol·licitar dades (idempotent, cachable)

   -POST: Crear/actualitzar recursos (no idempotent)

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

   Principis aplicats:

   Single Responsibility: Cada funció té una única responsabilitat

   Separation of Concerns: Rutes gestionen HTTP, controladors gestionen lògica

   Reusabilitat: Controladors poden ser usats per múltiples rutes

   Exemple:

   javascript
   // chatRoutes.js - Només routing
   router.post('/prompt', registerPrompt);

   // chatController.js - Tota la lògica
   const registerPrompt = async (req, res, next) => {
   // Lògica complexa aquí
   };

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

   errorHandler.js implementa:

   Captura global: Tots els errors passen per aquí

   Tipificació: Tracta diferents tipus d'errors (Sequelize, etc.)

   Logging: Registra errors amb context

   Resposta consistent: Formateig uniforme d'errors

   ——————————————————————————————————————

   // Gestió específica per tipus d'error

   if (err.name === 'SequelizeValidationError') {

   return res.status(400).json({ message: 'Error de validació' });

   }

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?

   Beneficis:

   Documentació viva: S'actualitza amb el codi

   Prova interactiva: Proves directes des del navegador

   Contracte clar: Defineix exactament què espera cada endpoint

   Com s'integra:

```
/**
 * @swagger
 * /api/chat/prompt:
 *   post:
 *     summary: Crear un nou prompt
 */
router.post('/prompt', registerPrompt);
```

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

   Elements importants:
   Paràmetres: Tipus, obligatorietat, valors permesos
   Respostes: Estructura esperada per cada codi HTTP
   Exemples: Casos d'ús reals

   Exemple:

```
/**
 * @swagger
 * /api/chat/sentiment-analysis:
 *   post:
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - text
 *             properties:
 *               text:
 *                 type: string
 *                 maxLength: 5000
 */
```

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?
   Estructura:

   // models/index.js - Defineix relacions

   Conversation.hasMany(Prompt, { foreignKey: 'ConversationId' });

   Prompt.belongsTo(Conversation, { foreignKey: 'ConversationId' });

   UUID vs ID autoincremental:

   UUID: Únic universalment, segur per distribució

   Autoincremental: Més simple però exposat

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

   Riscs de sync() en producció:

   Pèrdua de dades: Si es canvia l'estructura

   Bloqueig: Pot bloquejar la base de dades

   Inconsistències: Diferents entorns poden divergir

   Millor pràctica:

   // Desenvolupament (amb control)

   sequelize.sync({ alter: true });

   // Producció (sempre migracions)

   npx sequelize-cli db:migrate

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

   Seguretat: Prevenció d'SQL injection

   Portabilitat: Suport múltiples BBDD

   Productivitat: Sintaxi més intuïtiva

   Validacions: A nivell de model

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?

   Nivells (de més a menys sever):

   error: Errors crítics que necessiten acció immediata

   warn: Situacions anòmales que podrien causar errors

   info: Informació general del funcionament

   debug: Informació detallada per debugging

   verbose: Informació molt detallada

   Exemple del projecte:

   ```
   logger.error('Error en streaming', { error: error.message });

   logger.info('Conversa recuperada', { promptCount: prompts.length });
   ```

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

   Configuració típica:

   ```
   // logger.js
   transports: [
      new winston.transports.Console(),  // Desenvolupament
      new winston.transports.File({      // Producció
         filename: 'errors.log',
         level: 'error'
      })
   ]
   ```

   Avantatges:

   Consola: Desenvolupament ràpid

   Fitxers: Històric per producció

   Sistemes externs: ELK, Datadog, etc.

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

Informació Crítica per Logging

Què incloure:

-Timestamp: Quan va passar

-Nivell: Severitat

-Missatge: Descripció llegible

-Context:

```
{
   userId: req.user?.id,
   endpoint: req.path,
   method: req.method,
   errorCode: err.code
}
```

-Stack trace: Per debugging

-Correlation ID: Per seguir una transacció

Debugging en Producció

Com ajuda:

Tracing: Seguir una petició entre microserveis

Metrics: Identificar patrons (errors recurrents)

Alertes: Detectar problemes abans que els usuaris

Auditoria: Compliment legal

Exemple d'ús:

```
// chatController.js - Logging informatiu
logger.debug('Generant resposta no streaming', {
   conversationId: conversation.id,
   model: model });
```

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.

He fet SentimentAnalysis.js per definir aquests jsons i poder exporta-los com a models

hauria d'haver un json d'entrada:

-text (obligatori): Text a analitzar
-language (opcional, default: "ca"): Idioma del text
-model (opcional, default: valor de .env): Model d'Ollama a usar

```
{
"text": "Aquest producte és fantàstic! Estic molt content amb la compra.",
"language": "ca",
"model": "qwen2.5vl:7b"
}
```

i un de sortida:

-id: UUID de l'anàlisi
-text: Text analitzat
-sentiment: Categoria (positive/negative/neutral)
-score: Puntuació de -1 a 1
-confidence: Confiança (low/medium/high)
-keywords: Paraules clau detectades
-analysis: Explicació detallada del sentiment
-language: Idioma usat
-model: Model d'Ollama usat
-createdAt: Data de creació:

```
{
"id": "a8f5f167-524f-4c23-aa7b-9cf123456789",
"text": "Aquest producte és fantàstic! Estic molt content amb la compra.",
"sentiment": "positive",
"score": 0.85,
"confidence": "high",
"keywords": ["fantàstic", "content", "compra"],
```

```
 "analysis": "El text expressa una emoció clarament positiva amb un alt grau de
satisfacció...",
 "language": "ca",
 "model": "qwen2.5vl:7b",
 "createdAt": "2026-01-14T10:30:00.000Z"
}
```

I s'emmagatzema en una nova taula "SentimentAnalyses":

```
{
 id: UUID (PK),
 text: TEXT (NOT NULL) - Text original analitzat
 sentiment: ENUM('positive', 'negative', 'neutral') - Categoria del sentiment
 score: FLOAT - Puntuació de -1 a 1
 confidence: ENUM('low', 'medium', 'high') - Confiança de l'anàlisi
 keywords: JSON - Array de paraules clau
 analysis: TEXT - Explicació detallada generada per l'IA
 language: STRING (default: 'ca') - Idioma del text
 model: STRING - Model d'Ollama utilitzat
 createdAt: DATE
 updatedAt: DATE
}
```

La taula és independent (no relacionada amb Conversation/Prompt) ja que l'anàlisi de sentiment és un servei diferent
Si en el futur es vol relacionar amb converses, es podria afegir un ***ConversationId*** opcional