Instituto Politécnico Formosa

Sistema de Gestión de Blog Personal con Autenticación

Objetivo:

Desarrollar desde cero un sistema completo de blog personal que incluya autenticación de usuarios, gestión de artículos y etiquetas. El proyecto debe implementar todas las tecnologías vistas en clase: JWT, cookies, bcrypt, validaciones con express-validator, relaciones de Sequelize (1:1, 1:N, N:M) y operaciones CRUD completas con eliminación cascada y lógica.

Criterios de Evaluación

1. Presentación del código

- **El código debe estar limpio, ordenado y bien indentado.**
- **Uso obligatorio de try-catch en todos los controladores para un manejo adecuado de errores.**
- **Organización correcta del proyecto en carpetas:**
  - **src/config** → configuración (conexión a BD).
  - **src/models** → definición de modelos Sequelize.
  - **src/routes** → definición de rutas.
  - **src/controllers** → controladores.
  - **src/middlewares** → middlewares.
  - **src/helpers** → utilidades (JWT, bcrypt).
- **Uso exclusivo de ESModules (import / export).**
- **El código debe ser funcional, modularizado y sin errores de ejecución.**

2. Validaciones y lógica

- **Validaciones correctamente implementadas dentro de los controladores con express-validator.**
- **Uso correcto de Sequelize** con conexión a MySQL, definición adecuada de modelos y relaciones.
- **Respuestas claras y códigos HTTP apropiados** en cada operación:
  - **201 Created** → al crear.
  - **200 OK** → en consultas y actualizaciones exitosas.
  - **400 Bad Request** → errores de validación.
  - **401 Unauthorized** → falta autenticación.
  - **403 Forbidden** → falta autorización.
  - **404 Not Found** → recurso inexistente.
  - **500 Internal Server Error** → errores inesperados.
- **Verificación de unicidad** al crear o editar recursos con campos únicos.
- **Verificación de existencia previa** antes de editar o eliminar un recurso.
- **Mostrar mensajes de éxito o error** al realizar operaciones CRUD.

3. Autenticación y Autorización

- **Sistema JWT completo:** Generación, verificación y manejo de tokens.
- **Implementación correcta de cookies seguras:** httpOnly.
- **Hasheo de contraseñas con bcrypt:** Registro y login seguros.
- **Middleware de autenticación:** Protección de rutas privadas.
- **Middleware de autorización:** Diferentes permisos para users y admins.
- **Controladores de autenticación completos:** Registro, login, logout, perfil.

4. Relaciones y Eliminaciones

- **Todas las relaciones implementadas correctamente:** 1:1, 1:N y N:M.
- **Asociaciones bidireccionales** con Sequelize (belongsTo, hasOne, hasMany, belongsToMany).
- **Eliminación en cascada** implementada en al menos dos relaciones.
- **Eliminación lógica** implementada en al menos un modelo.
- **Integridad referencial** mantenida en todas las operaciones.

Entrega mediante Git y GitHub – Uso de ramas y commits

Requisitos obligatorios para el control de versiones.

Crear un nuevo repositorio llamado "**trabajo-practico-integrador-1"** y trabajar con la siguiente estructura:

- **Crear repositorio** con README inicial en rama **main**.
- **Crear rama develop** desde **main**.
- **Crear rama proyecto-integrador** desde **develop** para el desarrollo.
- **Durante el desarrollo** en rama **proyecto-integrador**, realizar **mínimo 10 commits** con mensajes claros y descriptivos.
- **Al finalizar el trabajo:**
    - Hacer un **merge limpio** de **proyecto-integrador** hacia **develop**, sin conflictos.
    - Luego hacer un **merge limpio** de **develop** hacia **main**, para que ambas ramas queden sincronizadas nuevamente.

Consignas:

1. Configuración inicial del proyecto

Crear la estructura base desde cero:

- **Inicializar proyecto Node.js** con "**npm init"**.
- **Instalar dependencias necesarias:**
  - express, sequelize, mysql2, cors, dotenv
  - jsonwebtoken, bcrypt, cookie-parser, express-validator
- **Crear estructura de carpetas** según criterios de evaluación.
- **Configurar variables de entorno** (.env):
  - Conexión a MySQL (DB_HOST, DB_USER, DB_PASSWORD, DB_NAME)
  - JWT_SECRET para firmar tokens
  - Configuración del servidor (PORT)
- **Crear archivo de configuración de base de datos** con Sequelize.
- **Configurar servidor Express** con middlewares básicos (CORS, cookie-parser, JSON parser).

2. Implementación de modelos y relaciones

Crear todos los modelos especificados:

1. User (Usuario)

- id (Primary Key, auto increment)
- username (VARCHAR(20), único, 3-20 caracteres)
- email (VARCHAR(100), único, formato válido)
- password (VARCHAR(255), hasheada con bcrypt)
- role (ENUM: 'user', 'admin', default: 'user')
- created_at
- updated_at
- deleted_at (para eliminación lógica)

2. Profile (Perfil de Usuario)

- id (Primary Key, auto increment)
- user_id (Foreign Key → User, único)
- first_name (VARCHAR(50))
- last_name (VARCHAR(50))
- biography (TEXT, opcional)
- avatar_url (VARCHAR(255), opcional)
- birth_date (DATE, opcional)
- created_at
- updated_at

4. Article (Artículo)

- id (Primary Key, auto increment)
- title (VARCHAR(200), 3-200 caracteres)
- content (TEXT, mínimo 50 caracteres)
- excerpt (VARCHAR(500), resumen corto, opcional)
- status (ENUM: 'published', 'archived', default: published)
- user_id (Foreign Key → User)
- created_at
- updated_at

5. Tag (Etiqueta)

- id (Primary Key, auto increment)
- name (VARCHAR(30), único, 2-30 caracteres)
- created_at
- updated_at

6. ArticleTag (Tabla Intermedia para relación N:M)

- id (Primary Key, auto increment)
- article_id (Foreign Key → Article)
- tag_id (Foreign Key → Tag)
- created_at
- updated_at

Validaciones a implementar:

- **User:** Validación de email único, username único, password hasheada.
- **Tag:** Validación de nombre único.

Configurar relaciones entre modelos:

- **Relación 1:1:** User ↔ Profile (User.hasOne(Profile), Profile.belongsTo(User))
- **Relaciones 1:N:**
  - User → Article (User.hasMany(Article), Article.belongsTo(User))
- **Relación N:M:** Article ↔ Tag (Article.belongsToMany(Tag, {through: 'ArticleTag'}), Tag.belongsToMany(Article, {through: 'ArticleTag'}))

Configuración de alias (as) para las relaciones:

- **Definir alias descriptivos** en todas las asociaciones para mayor claridad en las consultas.
- **Usar includes con as** siempre que sea necesario mostrar datos relacionados en las respuestas.
- **Para la relación N:M Article ↔ Tag:** Crear el modelo ArticleTag como tabla intermedia con sus propias propiedades y configurar la relación through: ArticleTag.

- **Configurar alias:**
  - User ↔ Profile: 'profile' (User) y 'user' (Profile)
  - User → Article: 'articles' (User) y 'author' (Article)
  - Article ↔ Tag: 'tags' (Article) y 'articles' (Tag) usando through: ArticleTag

3. Eliminación en cascada y lógica

Configurar en los modelos existentes las siguientes reglas de eliminación:

- **Implementar eliminación lógica**:
  - User (paranoid: true)
- **Implementar eliminación en cascada**:
  - Article (Cuando se elimina un article eliminar las asociaciones con tags que posee)

4. Sistema de autenticación y autorización

Crear helpers de utilidad:

- **JWT Helper:** Funciones para generar y verificar tokens JWT.
- **Bcrypt Helper:** Funciones para hashear y comparar contraseñas.

Implementar middlewares:

- **authMiddleware:** Verificar JWT desde cookies y extraer usuario.
- **adminMiddleware:** Verificar que el usuario autenticado sea admin.
- **ownerMiddleware:** Verificar que el usuario sea propietario del recurso.

Desarrollar controladores de autenticación:

- **POST /api/auth/register:** Registro de usuario con creación automática de perfil. (público)
- **POST /api/auth/login:** Login con JWT enviado como cookie segura. (público)
- **GET /api/auth/profile:** Obtener perfil del usuario autenticado. (usuario autenticado)
- **PUT /api/auth/profile:** Actualizar perfil del usuario autenticado. (usuario autenticado)
- **POST /api/auth/logout:** Logout limpiando cookie de autenticación. (usuario autenticado)

5. Controladores con CRUD completo

Para todos los modelos existentes debe existir un CRUD completo:

Users (acceso admin):

- **GET /api/users** → Listar todos los usuarios con sus perfiles. (solo admin)
- **GET /api/users/:id** → Obtener usuario específico con perfil y artículos. (solo admin)
- **PUT /api/users/:id** → Actualizar usuario (solo admin).
- **DELETE /api/users/:id** → Eliminación lógica de usuario (solo admin).

Tags:

- **POST /api/tags** → Crear etiqueta (solo admin).
- **GET /api/tags** → Listar todas las etiquetas. (usuario autenticado)
- **GET /api/tags/:id** → Obtener etiqueta específica con artículos asociados(solo admin).
- **PUT /api/tags/:id** → Actualizar etiqueta (solo admin).
- **DELETE /api/tags/:id** → Eliminar etiqueta (solo admin).

Articles:

- **POST /api/articles** → Crear artículo. (usuario autenticado)
- **GET /api/articles** → Listar artículos publicados. (usuario autenticado)
- **GET /api/articles/:id** → Obtener artículo por su id. (usuario autenticado)
- **GET /api/articles/user** → Listar artículos publicados del usuario logueado. (usuario autenticado)
- **GET /api/articles/user/:id** → Obtener artículo del usuario logueado por su id. (usuario autenticado)
- **PUT /api/articles/:id** → Actualizar artículo (solo autor o admin).
- **DELETE /api/articles/:id** → Eliminación lógica (solo autor o admin).

Articles Tags:

- **POST /api/articles-tags** → Agregar etiqueta a artículo. (solo autor)
- **DELETE /api/articles-tags/:articleTagId** → Remover etiqueta de artículo. (solo autor)

6. Validaciones completas con express-validator

Asegurarse de que todas las rutas (POST, GET, PUT, DELETE) tengan validaciones consistentes:

- **Validar IDs** recibidas tanto en el body como por params (como mínimo que sean enteros y existan en la BD).
- **Validar campos obligatorios** (por ejemplo: notEmpty, isLength, etc.).
- **Agregar validaciones personalizadas (custom)** según cada modelo.

Validaciones específicas por modelo:

User/Auth:

- username: 3-20 caracteres, alfanumérico, único.
- email: formato válido, único.
- password: mínimo 8 caracteres, al menos una mayúscula, minúscula y número.
- role: solo valores permitidos ('user', 'admin').

Profile:

- first_name y last_name: 2-50 caracteres, solo letras.
- biography: máximo 500 caracteres.
- avatar_url: formato URL válido (opcional).

Article:

- title: 3-200 caracteres, obligatorio.
- content: mínimo 50 caracteres, obligatorio.
- excerpt: máximo 500 caracteres.
- status: solo valores permitidos ('published', 'archived').
- user_id: debe existir y coincidir con usuario autenticado (excepto admin).

Tag:

- name: 2-30 caracteres, único, obligatorio, sin espacios.

Validaciones personalizadas adicionales:

- Validar que solo el autor pueda editar sus artículos (excepto admin).
- Verificar existencia de etiquetas antes de asociarlas a artículos.