Sistema de Gestión de Blog Personal con MongoDB y Mongoose

Objetivo:

Desarrollar desde cero un sistema completo de blog personal que incluya autenticación de usuarios, gestión de artículos y etiquetas utilizando MongoDB como base de datos NoSQL y Mongoose como ODM. El proyecto debe implementar todas las tecnologías vistas en clase: JWT, cookies, bcrypt, validaciones con express-validator, relaciones embebidas y referenciadas, 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 MongoDB)
    - **src/models** → definición de esquemas Mongoose
    - **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 y validaciones de Mongoose.
- Uso correcto de Mongoose con conexión a MongoDB, definición adecuada de esquemas 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.
- Uso correcto de **populate y populate reverso** para traer datos relacionados.

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 eliminación lógica y en cascada

- Implementación correcta de documentos embebidos y referencias.
- Al menos una relación embebida (1:1 o array embebido).
- Al menos una relación referenciada (1:N).
- Al menos una relación muchos a muchos (N:M) usando referencias.
- Uso correcto de populate y populate reverso.
- Eliminación lógica implementada en al menos un modelo.
- Eliminación en cascada implementada en al menos dos relaciones.

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-2"** y trabajar con la siguiente estructura:

- Crear repositorio con README inicial en rama main.
- Crear rama develop desde main.
- Crear rama proyecto-integrador-mongodb desde develop para el desarrollo.
- Durante el desarrollo en rama proyecto-integrador-mongodb, realizar mínimo 10 commits con mensajes claros y descriptivos.
- Al finalizar el trabajo:
    - Hacer un merge limpio de **proyecto-integrador-mongodb** 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, mongoose, cors, dotenv**
    - **jsonwebtoken, bcrypt, cookie-parser, express-validator**
- Crear estructura de carpetas según criterios de evaluación.
- Configurar variables de entorno (.env):
    - Conexión a MongoDB **(MONGODB_URI)**
    - **JWT_SECRET** para firmar tokens
    - Configuración del servidor **(PORT)**
- Crear archivo de configuración de base de datos con Mongoose.
- Configurar servidor Express con middlewares básicos (**CORS, cookie-parser, JSON parser**).

2. Implementación de esquemas y relaciones

Crear todos los modelos especificados:

1. User (Usuario)

- _id (ObjectId automático)
- username (String, único, 3-20 caracteres)
- email (String, único, formato válido)
- password (String, hasheada con bcrypt)
- role (String, enum: 'user', 'admin', default: 'user')
- profile (Documento embebido - **relación 1:1**):
    - firstName (String, 2-50 caracteres)
    - lastName (String, 2-50 caracteres)
    - biography (String, máximo 500 caracteres, opcional)
    - avatarUrl (String, formato URL, opcional)
    - birthDate (Date, opcional)
- createdAt (Date)
- updatedAt (Date)

2. Article (Artículo)

- _id (ObjectId automático)
- title (String, 3-200 caracteres)
- content (String, mínimo 50 caracteres)
- excerpt (String, máximo 500 caracteres, opcional)

- status (String, enum: 'published', 'archived', default: 'published')
- author (ObjectId, referencia a User)
- tags (Array de ObjectIds, referencias a Tag - **relación N:M**)
- createdAt (Date)
- updatedAt (Date)

3. Tag (Etiqueta)

- _id (ObjectId automático)
- name (String, único, 2-30 caracteres, sin espacios)
- description (String, opcional, máximo 200 caracteres)
- createdAt (Date)
- updatedAt (Date)

4. Comment (Comentario) - Nuevo modelo

- _id (ObjectId automático)
- content (String, 5-500 caracteres)
- author (ObjectId, referencia a User)
- article (ObjectId, referencia a Article - **relación 1:N**)
- createdAt (Date)
- updatedAt (Date)

Validaciones a implementar en los esquemas:

- **User**: Validación de email único, username único, password con requisitos de seguridad.
- **Article**: Validación de título y contenido requeridos, status válido.
- **Tag**: Validación de nombre único, sin espacios.
- **Comment**: Validación de contenido requerido, referencias válidas.

Configurar relaciones:

- **Relación 1:1 embebida**: User ↔ Profile (profile embebido en User)
- **Relación 1:N referenciada**: User → Article (author referenciado)
- **Relación 1:N referenciada**: Article → Comment (article referenciado)
- **Relación N:M referenciada**: Article ↔ Tag (array de ObjectIds)

Eliminación lógica

Implementar eliminación lógica en **al menos un modelo**:

- **User**: Agregar campo **deletedAt (Date, opcional)** para soft delete
- Configurar el modelo para ignorar documentos con **deletedAt** en consultas normales

Eliminación en cascada

Implementar eliminación en cascada en **al menos dos relaciones**:

- **Article**: Al eliminar un artículo, eliminar todos sus comentarios automáticamente
- **Tag**: Al eliminar una etiqueta, removerla de todos los artículos que la referencian

3. 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.
- **ownerOrAdminMiddleware**: Verificar que el usuario sea propietario del recurso O sea admin.

Desarrollar controladores de autenticación:

- **POST /api/auth/register:** Registro de usuario con perfil embebido. (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 embebido del usuario autenticado. (usuario autenticado)
- **POST /api/auth/logout:** Logout limpiando cookie de autenticación. (usuario autenticado)

4. Controladores con CRUD completo

Users (acceso admin):

- **GET /api/users** → Listar todos los usuarios con populate de artículos. (solo admin)
- **GET /api/users/:id** → Obtener usuario específico con artículos y comentarios. (solo admin)
- **PUT /api/users/:id** → Actualizar usuario (solo admin).
- DELETE /api/users/:id → Eliminación física 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 con populate de artículos asociados (usuario autenticado).
- **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 con populate de author y tags. (usuario autenticado)
- **GET /api/articles/:id** → Obtener artículo por ID con populate completo. (usuario autenticado)
- **GET /api/articles/my** → Listar artículos del usuario logueado. (usuario autenticado)
- **PUT /api/articles/:id** → Actualizar artículo (solo autor o admin).
- **DELETE /api/articles/:id** → Eliminación física (solo autor o admin).

Comments:

- **POST /api/comments** → Crear comentario en artículo. (usuario autenticado)
- **GET /api/comments/article/:articleId** → Listar comentarios de un artículo con populate de author. (usuario autenticado)
- **GET /api/comments/my** → Listar comentarios del usuario logueado. (usuario autenticado)
- **PUT /api/comments/:id** → Actualizar comentario (solo autor o admin).
- **DELETE /api/comments/:id** → Eliminación física de comentario (solo autor o admin).

Article Tags (relación N:M):

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

5. Validaciones completas con express-validator y Mongoose

Asegurarse de que todas las rutas tengan validaciones consistentes:

- Validar ObjectIds recibidos tanto en el body como por params.
- Validar campos obligatorios con express-validator y esquemas Mongoose.
- Agregar validaciones personalizadas 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 (embebido):

- firstName y lastName: 2-50 caracteres, solo letras.
- biography: máximo 500 caracteres.
- avatarUrl: 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').
- author: ObjectId válido que debe coincidir con usuario autenticado (excepto admin).

Tag:

- name: 2-30 caracteres, único, obligatorio, sin espacios.
- description: máximo 200 caracteres.

Comment:

- content: 5-500 caracteres, obligatorio.
- author: ObjectId válido.
- article: ObjectId válido que debe existir.

Validaciones personalizadas adicionales:

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

6. Documentación requerida

En el README.md incluir:

- Explicar con sus propias palabras por qué se eligió **embebido** o **referenciado** para cada relación. Analizar ventajas y desventajas de cada decisión tomada en el diseño del sistema.
- Documentación de endpoints con ejemplos de request/response.
- Instrucciones de instalación y configuración.
- Explicación de las validaciones personalizadas implementadas.

Entrega:

- Repositorio en GitHub con todas las ramas requeridas.
- Código funcional sin errores.
- README.md completo con documentación.
- Archivo .env.example con variables requeridas.
- Al menos 10 commits descriptivos en la rama de desarrollo.