Instituto Politécnico Formosa

Primer Parcial - Sistema de Gestión de Patrimonio Judicial

Nota importante sobre comentarios: Para facilitar la identificación y lectura de los comentarios en el código, es necesario que instalen la extensión "Better Comments" en Visual Studio Code.

Pasos para instalar la extensión:

1. Abrir Visual Studio Code.
2. Ir a la pestaña de extensiones: a. Hagan clic en el ícono de extensiones en la barra lateral izquierda, o presione Ctrl+Shift+X (Windows/Linux).
3. Buscar la extensión "Better Comments": a. Escriban "Better Comments" en la barra de búsqueda.
4. Instalar la extensión: a. Seleccionen la extensión desarrollada por Aaron Bond y hagan clic en "Install".
5. Reiniciar Visual Studio Code si es necesario.

Esta extensión les permitirá visualizar los comentarios con distintos colores y formatos, facilitando la comprensión de las tareas que deben realizar.

Instrucciones Generales

El presente examen consiste en completar la implementación de una API REST utilizando Node.js, Express, Sequelize (MySQL) y Mongoose (MongoDB). **Se les proporcionará una carpeta base con la estructura completa del proyecto, incluyendo controladores, configuración de base de datos, modelos base, validaciones y middlewares ya implementados.**

Importante: Los archivos base proporcionados **NO deben ser modificados**, excepto aquellos específicamente indicados en las consignas. El trabajo se enfoca en:

- **Completar las relaciones de los modelos** (tanto Sequelize como Mongoose)
- **Completar la implementación de los controladores** (solo para Mongoose)
- **Refactorizar y organizar las rutas**
- **Mover las validaciones a sus archivos correspondientes**

Una vez finalizado, se debe subir el trabajo a GitHub y entregar el enlace en Google Classroom antes del horario límite establecido.

Criterios de Evaluación

La corrección del examen se regirá por los siguientes criterios:

1. Presentación del código

- Código limpio, ordenado y bien indentado.
- Uso obligatorio de try-catch en todos los controladores.
- Respeto de la estructura de carpetas proporcionada: src/config, src/models, src/routes, src/controllers, src/middlewares, src/helpers.
- Uso exclusivo de módulos ESModules (import / export).
- Código funcional, modularizado y sin errores de ejecución.
- **No modificación de archivos base proporcionados** (excepto los indicados en las consignas).

2. Validaciones y lógica

- Validaciones correctamente organizadas en archivos separados con express-validator.
- **Para Sequelize:** Definición adecuada de relaciones entre modelos.
- **Para Mongoose:** Definición adecuada de relaciones **(embebidas y referenciadas)** y uso correcto de populate.
- Respuestas con mensajes claros y códigos HTTP apropiados **(400, 404, 201, etc.).**
- Verificación de unicidad en creación y edición.
- Verificación de existencia previa antes de editar o eliminar un recurso.

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 según roles del sistema judicial.
- Controladores de autenticación completos: Registro, login, logout, perfil.

4. Relaciones y Eliminaciones

- **Para Sequelize**: Todas las relaciones implementadas correctamente (1:1, 1:N, N:M) con asociaciones bidireccionales y alias apropiados.
- **Para Mongoose**: Implementación correcta de documentos embebidos Y referencias, con populate y populate reverso.
- Eliminación en cascada implementada.
- Eliminación lógica implementada.
- Integridad referencial mantenida en todas las operaciones.

5. Refactorización y Organización

- **Rutas correctamente organizadas** en archivos separados.
- **Validaciones movidas a archivos específicos** según la estructura requerida.

- **Controladores completados** siguiendo los patrones establecidos.
- Mantenimiento de la modularidad y separación de responsabilidades.

6. Autonomía y originalidad del trabajo

Está estrictamente prohibido:

- Usar herramientas automáticas como IA generativa o autocompletado avanzado.
- Compartir archivos o código con otros compañeros durante la evaluación.
- Modificar los archivos base proporcionados **(excepto los específicamente indicados)**.
- También se tendrán en cuenta los requisitos del siguiente Gist.

Cualquier incumplimiento será motivo de desaprobación directa.

Entrega mediante Git y GitHub – Uso de ramas y commits

Requisitos obligatorios para el control de versiones

- El proyecto debe ser versionado con Git desde el inicio.
- Se deben crear y utilizar dos ramas:
  - **main**: rama principal para la entrega final.
  - **develop**: rama de desarrollo donde se realiza el examen.
- Todo el desarrollo debe hacerse en la rama develop.
- Al finalizar, se debe hacer un merge limpio de develop hacia main, sin conflictos.
- Se deben realizar al menos 5 commits con mensajes claros y descriptivos.

Requisitos del repositorio remoto

- El nombre del repositorio en GitHub debe seguir la siguiente convención: **primer-parcial-tlp2-apellido-nombre**. Ejemplo: primer-parcial-tlp2-perez-juan
- El repositorio debe contener:
  - Todos los archivos del proyecto (app.js, package.json, src/ completo).
  - El archivo .env.example con los nombres de las variables utilizadas.
  - El archivo .env y la carpeta node_modules no deben subirse.
- El enlace al repositorio debe entregarse en Google Classroom antes del horario límite establecido.

Instituto Politécnico Formosa

Consignas del Sistema de Gestión de Patrimonio del Poder Judicial de Formosa

1. Configuración inicial del proyecto

Nota: La estructura base y configuración inicial ya están proporcionadas en la carpeta del proyecto.

Completar la configuración:

- Configurar variables de entorno (.env) según el archivo .env.example:
  - Para MongoDB: **MONGODB_URI**
  - **JWT_SECRET** para firmar tokens
  - Configuración del servidor **PORT**
- Instalar dependencias con **npm install**
- Verificar que la conexión a la base de datos funcione correctamente

2. Implementación de relaciones en modelos/esquemas

Los modelos base ya están creados con sus propiedades (excepto lo embebido). Su tarea es COMPLETAR LAS RELACIONES Y DATOS EMBEBIDOS FALTANTES.

Datos embebidos a completar:

- profile en User (mongoose):
  - employee_number: String, unique, required.
  - first_name y last_name: String, required, minLength: 2, maxLength: 50
  - phone: String, opcional.

Relaciones a completar:

- **1:1**: User ↔ Profile
- **1:N**: User → Asset **(cómo responsible)**
- **N:M**: Asset ↔ Category

Alias a definir:

- User ↔ Profile: 'profile' (User) y 'user' (Profile)
- User → Asset: 'assets' (User) y 'responsible' (Asset)
- Asset ↔ Category: 'categories' (Asset) y 'assets' (Category) usando through: AssetCategory para Sequelize

3. Eliminación en cascada y lógica

- **Eliminación lógica**: User (paranoid: true para Sequelize, deletedAt para Mongoose)
- **Eliminación en cascada**:
  - Al eliminar un Asset, eliminar todas las asociaciones AssetCategory
  - Al eliminar una Category, eliminar todas las asociaciones AssetCategory

4. Completar controladores (Solo para Mongoose)

Los controladores base ya están proporcionados con la estructura. Su tarea es COMPLETAR la implementación faltante.

Controladores de autenticación a completar:

- POST /api/auth/register: Registro con creación automática de profile (público)
- POST /api/auth/login: Login con JWT en cookie (público)
- GET /api/auth/profile: Obtener profile del usuario autenticado (usuario autenticado)
- POST /api/auth/logout: Logout limpiando cookie (usuario autenticado)

Controladores con CRUD completo

Users (solo administrators):

- **GET /api/users** → Listar funcionarios con profiles y assets a cargo (solo admin)
- **DELETE /api/users/:id** → Eliminación lógica (solo admin)

Assets:

- **POST /api/assets** → Crear asset (usuario autenticado)
- **GET /api/assets** → Listar assets con categories (solo admin)
- **GET /api/assets/my-assets** → Listar assets del responsible autenticado (usuario responsible)
- **DELETE /api/assets/:id** → Eliminar asset (usuario responsible)

Categories:

- **POST /api/categories** → Crear category (solo admin)
- **GET /api/categories** → Listar categories (solo admin)
- **DELETE /api/categories/:id** → Eliminar category (solo admin)

5. Middlewares de validación a completar:

registerValidation:

- username: 3-20 caracteres, alfanumérico, único
- email: formato válido, único
- password: mínimo 8 caracteres, mayúscula, minúscula y número
- role: valores permitidos ('secretary', 'administrator')
- employee_number: formato específico, único, obligatorio
- first_name y last_name: 2-50 caracteres, solo letras
- phone: formato válido (opcional)

loginValidation:

- email: formato válido, obligatorio
- password: obligatorio, mínimo 8 caracteres

createUserValidation:

- username: 3-20 caracteres, alfanumérico, único
- email: formato válido, único
- password: mínimo 8 caracteres, mayúscula, minúscula y número
- role: valores permitidos ('secretary', 'administrator')
- employee_number: formato específico, único, obligatorio
- first_name y last_name: 2-50 caracteres, solo letras
- phone: formato válido (opcional)

createAssetValidation:

- inventory_number: formato específico, único, obligatorio
- description: 10-500 caracteres, obligatorio
- brand y model: 2-100 caracteres, obligatorio
- status: valores permitidos ('good', 'regular', 'bad', 'out_of_service')
- acquisition_date: fecha válida, no futura, obligatorio
- acquisition_value: número positivo, obligatorio
- responsible_id: debe existir y ser funcionario activo
- categories: array de IDs válidos de categorías existentes

createCategoryValidation:

- name: 3-100 caracteres, único, obligatorio
- description: máximo 500 caracteres (opcional)

Validaciones personalizadas a implementar:

- Verificar unicidad de username, email, employee_number, inventory_number
- Validar que el responsible_id corresponda a un usuario activo (no eliminado lógicamente)
- Verificar que las categorías asignadas a un asset existan

Aplicación en rutas:

- Integrar los middlewares de validación en cada ruta correspondiente
- Asegurar que todas las rutas tengan sus validaciones aplicadas antes de ejecutar la lógica del controlador**.**

7. Documentación requerida

En el README.md incluir:

Documentación técnica:

- **Instrucciones de instalación y configuración**: Pasos detallados para configurar y ejecutar el proyecto.