Introducción

En el mundo real, las aplicaciones web modernas dependen de APIs REST para intercambiar datos entre clientes y servidores. Estas APIs no solo deben proveer información, sino también garantizar seguridad, consistencia en los formatos de respuesta y manejo adecuado de errores.

Este laboratorio busca que el estudiante experimente el proceso completo de diseñar e implementar una API en Node.js con Express, aplicando estándares de la industria. Con este ejercicio, se refuerza la importancia de diseñar APIs robustas, seguras y bien documentadas, que son fundamentales en cualquier proyecto de desarrollo web moderno.

Objetivo general

Desarrollar una API REST con Node.js y Express que implemente buenas prácticas de seguridad, manejo de errores, validación de datos y estandarización de respuestas, utilizando archivos JSON como base de datos simulada y documentando los endpoints en Postman.

Objetivos específicos

- Implementar endpoints REST que utilicen correctamente los códigos de estado HTTP en operaciones de consulta, creación, actualización y eliminación.
- Diseñar una estructura estándar de respuestas para casos de éxito y error, incluyendo metadatos como timestamp y path.
- Incorporar content negotiation, permitiendo devolver datos en formato JSON o XML según el encabezado Accept.
- Proteger la API mediante API Key para acceso básico y JWT para autenticación y autorización basada en roles.
- Validar las entradas de los usuarios, devolviendo errores estructurados con código 422 en caso de datos inválidos.
- Simular persistencia de datos utilizando archivos JSON, asegurando integridad de información y control de duplicados.
- Elaborar y entregar una colección Postman con la documentación de los endpoints, contemplando ejemplos de casos exitosos y de error.

Contenido

En este laboratorio el estudiantado debe desarrollar una API REST con Node.js y Express que cumpla con las siguientes características:

- **Rutas requeridas**
  - POST /auth/login: genera un **JWT** válido al recibir credenciales correctas, protegido con **API Key**
  - GET /products: listado de productos, protegido con **API Key**. Soporta paginación (page, limit) y negociación de contenido (JSON o XML).
  - GET /products/:id: detalle de producto, protegido con **API Key.**
  - POST /products: crea un producto nuevo, protegido con **JWT** y rol editor o admin.
  - PUT /products/:id: actualiza un producto existente, protegido con **JWT** y rol editor o admin.
  - DELETE /products/:id: elimina un producto, protegido con **JWT** y rol admin.
- **Seguridad**
  - Uso de **API Key** para rutas públicas (listado y detalle de productos).
  - Uso de **JWT** para rutas protegidas (crear, actualizar, eliminar).
  - Validación de roles: editor y admin.
- **Validaciones y persistencia**
  - La información de usuarios y productos se almacenará en **archivos JSON** dentro de la carpeta /db.
  - La creación y actualización de productos deben validar: name, sku único, price > 0, stock ≥ 0, category.
  - Si el sku ya existe, debe responder con **409 Conflict**.
- **Manejo de errores**
  - Implementar un **middleware central** que responda con formato estándar:

```json
{
  "error": {
    "code": "NOT_FOUND",
    "message": "Product not found",
    "details": {},
    "timestamp": "...",
    "path": "/products/999"
  }
}
```

    - Usar códigos HTTP correctos: 200, 201, 204, 400, 401, 403, 404, 409, 422, 500.
- **Respuestas y formatos**
    - Respuestas exitosas y de error deben incluir timestamp y path.
    - La API debe soportar **JSON** y **XML** según el encabezado Accept.
- **Documentación en Postman**
    - Crear una **colección Postman** con todos los endpoints.
    - Incluir ejemplos de casos exitosos y de error (401, 403, 404, 409, 422).
    - Exportar la colección (.json) y ubicarla en la carpeta /docs.

Rubrica de evaluación

La evaluación del laboratorio se realizará bajo los siguientes criterios:

Criterio	Insuficiente (0.5)	Aceptable (1.0)	Excelente (1.5)
Estructura del proyecto (carpetas, uso de .env, scripts)	Proyecto desordenado, sin guía de ejecución	Estructura básica, se puede ejecutar	Estructura clara, modular, con README y .env.example
Rutas implementadas (/auth/login, /products)	Faltan varias rutas o no funcionan	Todas las rutas principales existen y funcionan con limitaciones	Todas las rutas funcionan correctamente (GET, POST, PUT, DELETE) con respuestas consistentes
Seguridad (API Key y JWT con roles)	No implementa seguridad o está incorrecta	Implementa al menos un mecanismo (API Key o JWT)	Implementa API Key + JWT + roles correctamente
Validaciones y persistencia en JSON	No valida entradas o no guarda cambios en JSON	Valida parcialmente y persiste en JSON con errores	Valida todos los campos y persiste en JSON de forma correcta (con control de duplicados)
Manejo de errores y códigos HTTP	Respuestas genéricas, códigos incorrectos	Respuestas parcialmente estandarizadas, usa la mayoría de los códigos correctos	Middleware central, mensajes claros, usa todos los códigos HTTP esperados
Negociación de contenido (JSON/XML)	Solo responde en JSON	Responde en JSON y XML pero con limitaciones	Negociación completa, respuestas equivalentes en JSON y XML
Documentación en Postman	Incompleta o ausente	Colección con casos básicos de éxito	Colección completa con casos de éxito y error (401, 403, 404, 409, 422), bien organizada
Calidad técnica (legibilidad, modularización, logs básicos)	Código difícil de seguir o sin modularidad	Código entendible con algunos problemas	Código modular, claro, con middlewares y logs funcionales

Puntos extra

Se podrá podrá optar por puntos adicionales si implementa el uso de una **base de datos real en lugar de archivos JSON**, con la condición de que la **BD sea accesible públicamente sin necesidad de ejecutar nada localmente**. Esto permite probar la API desde cualquier equipo sin configuraciones extra.

Requisitos

- La API debe conectarse a una **BD pública en la nube**.
- No debe requerir instalación local de servidores (ej. MySQL, MongoDB, Postgres en la PC).
- El esquema de productos y usuarios debe mantenerse coherente con lo solicitado en el laboratorio.
- La conexión debe configurarse mediante variables de entorno en .env.

Entrega

Cada estudiante deberá entregar un archivo comprimido (.zip) con el proyecto completo, siguiendo estas instrucciones:

1. Una carpeta con el repositorio completo
2. Un archivo denominado enlaces.txt que contenga
    a. URL del repositorio público en GitHub:
    b. URL de la documentación de postman desplegada

Fecha límite: viernes 3 de octubre 7:00 p.m.

Forma de entrega: El archivo .zip debe subirse a la plataforma TEC Digital, en el espacio correspondiente al curso.

Portafolio del curso: Este laboratorio forma parte del portafolio individual del curso.