MÓDULO 7. DESARROLLO AVANZADO DE BACKEND Y APIS

Tarea 2: Librerías de Backend con Node.js

Diego Matilla De La Cierva

Descripción del Desarrollo

En este proyecto se ha desarrollado una API REST para gestionar un recurso de “productos” cumpliendo con un CRUD completo, validación de datos, autenticación por JWT y manejo de errores. La aplicación está construida sobre Express, utiliza Joi para garantizar la calidad de los datos y jsonwebtoken para asegurar que solo usuarios autenticados puedan crear, modificar o eliminar productos. La persistencia se simula con un archivo JSON, y la configuración se centraliza en un archivo .env.

.env

PORT = 3000

JWT_SECRET = clave

Estructura del Proyecto

app.js

Carga variables de entorno, configura Express con parseo de JSON y logging (morgan), monta los routers de autenticación y productos, y aplica un middleware global de errores. Finalmente, inicia el servidor en el puerto configurado.

routes/

- authenticationRoutes.js: define POST /api/auth/login, valida credenciales mediante express-validator y devuelve un JWT firmado.
- productRoutes.js: expone los endpoints públicos (GET /api/productos y GET /api/productos/:id) y los protegidos (POST, PUT, DELETE), aplicando en estos últimos el middleware de autenticación y el de validación de producto.

controllers/

Contiene la lógica de negocio para cada endpoint:

- productController.js: funciones asíncronas que llaman al modelo, gestionan los resultados y devuelven códigos HTTP apropiados.

models/

- productModel.js: abstrae la persistencia en JSON, con métodos getAll, getById, create, update y remove.

middlewares/

- validationMiddleware.js: usa Joi para asegurar que nombre, precio, fechaCreacion, etc. cumplan el esquema.

- authMiddleware.js: extrae y verificar el token JWT del header Authorization.

- errorHandlerMiddleware.js: captura excepciones no manejadas y responde un 500 Internal Server Error uniforme.

utils/fileController.js

Métodos síncronos para leer y escribir el archivo data.json, creando automáticamente un array vacío si el fichero no existe.

Conclusiones y Observaciones

En conclusión, durante este ejercicio has visto cómo funciona el token de autenticación: en el endpoint /login se generan y firman con tu JWT_SECRET un pequeño “payload” (por ejemplo, { username: 'admin' }) que el cliente recibe en la respuesta; a partir de ahí, cada petición crítica (POST, PUT, DELETE) debe incluir en su cabecera Authorization: Bearer <token> para que el authMiddleware lo verifique con jsonwebtoken.verify(), extraiga el payload y autorice el acceso. Con ello has aprendido a integrar JWT en un flujo Express, a validar esquemas de entrada con Joi, a estructurar tu proyecto en rutas, controladores, modelos y middlewares, y a simular persistencia ligera leyendo y escribiendo en un archivo JSON, lo que refuerza todo el ciclo petición–respuesta de una API REST.