



Universidad Tecmilenio

Campus Las Torres

Materia: Desarrollo FullStack

Actividad 3 – API RESTful utilizando Node.js y Express.js

Alumno: Patricio Calvo

Matricula: 07097795

Febrero / 2026

Link de Github:

<https://github.com/Diablo650cc/Desarrollo-FullStack/tree/main/ACT%203>

REPORTE

Configuración Inicial

Se creó la carpeta "api-tareas" y se inicializó con npm init -y. Se instalaron las dependencias: express, body-parser, jsonwebtoken y bcryptjs. Esta base permitió construir el servidor y las funcionalidades de seguridad.

Servidor con Express.js

Se configuró el archivo server.js con un servidor Express escuchando en puerto 3000. Se incluyó middleware para parsear JSON y un sistema de logging que registra cada petición (método, URL, headers y body), facilitando el debugging.

Almacenamiento con Módulo fs

Se utilizó fs.promises para operaciones asincrónicas no bloqueantes. Se crearon funciones obtenerTareas() y guardarTareas() para leer/escribir en tareas.json. Similarmente, obtenerUsuarios() y guardarUsuarios() para manejar usuarios.json. Esto garantiza persistencia sin afectar el Event Loop.

Rutas CRUD

- GET /tareas: Devuelve todas las tareas del archivo JSON.
- POST /tareas: Crea nueva tarea con ID único (timestamp), fecha de creación y la guarda.
- PUT /tareas/:id: Actualiza tarea existente, agregando fecha de actualización. Error 404 si no existe.
- DELETE /tareas/:id: Elimina tarea por ID. Error 404 si no se encuentra.

Sistema de Autenticación

- POST /register: Recibe email y contraseña. Verifica que el email no exista, encripta contraseña con bcryptjs (10 rondas de sal) y guarda el usuario.
- POST /login: Verifica credenciales. Si son válidas, genera token JWT con ID y email del usuario, válido por 2 horas. El token debe incluirse en peticiones posteriores.

Middleware de Autenticación

La función autenticarToken extrae el token del header Authorization, verifica su validez con jwt.verify() y adjunta los datos del usuario a req.user. Si no hay token, responde 401; si es inválido, 403.

Todas las rutas de tareas incorporan este middleware, quedando protegidas. Al crear tareas, se asocia el userId del token para futuras personalizaciones.

Manejo de Errores y Debugging

Se implementó un sistema de errores de tres capas:

- Middleware de logging: Registra cada petición con detalles completos.
- Try-catch en rutas: Captura errores y los pasa al manejador con next(error).
- Middleware 404: Captura rutas no definidas.
- Manejador central de errores: Con cuatro parámetros, registra el error en consola, determina código HTTP (400, 401, 404, 500) y envía respuesta JSON estructurada. En desarrollo incluye stack trace.

Se agregó ruta /error-test para probar el sistema.

Validación de Datos

- Registro: Valida email y contraseña no vacíos, y email no duplicado.
- Login: Valida que ambos campos estén presentes.
- Las rutas CRUD pueden extenderse fácilmente para requerir título y descripción.

Herramientas de Debugging

- console.log() estratégicos
- Middleware de logging automático
- Node.js Inspector (node --inspect server.js)
- Postman para pruebas de endpoints

Resultado Final

- La API cumple todos los requisitos:
- Servidor Express en puerto 3000
- CRUD completo con almacenamiento JSON
- Autenticación segura con bcrypt y JWT
- Rutas protegidas por middleware
- Manejo centralizado de errores
- Validación básica de datos
- Herramientas de debugging integradas

Conclusión

El proyecto demostró la implementación práctica de conceptos clave de Node.js: Event Loop, asíncronía con `async/await`, manejo de archivos, middleware en Express, y autenticación segura. La estructura modular y el manejo de errores facilitan el mantenimiento y escalabilidad futura.

A. CONFIGURAR EL PROYECTO NODE.JS

The screenshot shows a Windows desktop environment. On the left is a File Explorer window titled 'api-tareas' showing the contents of a folder named 'ACT 3'. Inside 'ACT 3' is a folder named 'api-tareas' which contains three files: 'package.json', 'package-lock.json', and a 'node_modules' folder. The 'node_modules' folder is highlighted. On the right is a Command Prompt window with a black background and white text. It shows the command 'npm init -y' being run, followed by the generated package.json file content. Then, the command 'npm install express body-parser jsonwebtoken bcryptjs' is run, and the output shows 80 packages added, 81 audited, 23 packages looking for funding, 0 vulnerabilities found, and the command 'C:\Users\PatoCV\Desktop\ACT 3\api-tareas>' at the end.

```
C:\Users\PatoCV\Desktop\ACT 3\api-tareas>npm init -y
Write to C:\Users\PatoCV\Desktop\ACT 3\api-tareas\package.json:
{
  "name": "api-tareas",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}

C:\Users\PatoCV\Desktop\ACT 3\api-tareas>npm install express body-parser
jsonwebtoken bcryptjs
added 80 packages, and audited 81 packages in 5s
23 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
C:\Users\PatoCV\Desktop\ACT 3\api-tareas>
```

B. CREACIÓN DE UN SERVIDOR BÁSICO CON EXPRESS.JS.

The screenshot shows a Visual Studio Code interface with a dark theme. The main area displays the 'server.js' file under the 'api-tareas' project. The code is written in JavaScript and uses the Express framework to create a basic API. It includes routes for getting a welcome message and handling errors, and it listens on port 3000, logging the server's address to the console.

```
api-tareas > JS server.js > ...
1 const express = require('express');
2 const app = express();
3 const PORT = 3000;
4
5 app.use(express.json());
6
7 app.get('/', (req, res) => {
8   res.json({ mensaje: 'Bienvenido a la API de Tareas' });
9 });
10
11 app.use((err, req, res, next) => {
12   const statusCode = err.status || 500;
13   res.status(statusCode).json({
14     success: false,
15     message: err.message || 'Error interno del servidor',
16     status: statusCode
17   });
18 });
19
20 app.listen(PORT, () => {
21   console.log(`Servidor corriendo en http://localhost:${PORT}`);
22 });
```

C. CREAR LAS RUTAS PARA LA API RESTFUL

```
151 // RUTAS PARA LA API RESTFUL
152 // LAS RUTAS A CONTINUACION EXPONEN LOS ENDPOINTS DE LA API (PELÍCULAS)
153
154
155 peliculas.push(nuevaPelicula);
156 await guardarPeliculas(peliculas);
157 res.status(201).json(nuevaPelicula);
158 } catch (error) {
159 next(error);
160 }
161 });
162
```

D. MANEJO DE DATOS CON EL MÓDULO FS

```
25 // MANEJO DE DATOS CON EL MÓDULO FS
26 // LAS FUNCIONES ABAJO REALIZAN LECTURA/ESCRITURA EN ARCHIVOS JSON
27
28 async function obtenerPeliculas() {
29   try {
30     const data = await fs.readFile(PELICULAS_FILE, 'utf8');
31     return JSON.parse(data);
32   } catch {
33     return [];
34   }
35 }
```

E. IMPLEMENTACIÓN DE AUTENTICACIÓN Y SESIONES

```
44 // Usuarios (auth)
45 async function obtenerUsuarios() {
46   try {
47     const data = await fs.readFile(USERS_FILE, 'utf8');
48     const parsed = JSON.parse(data);
49     return Array.isArray(parsed) ? parsed : [];
50   } catch {
51     return [];
52   }
53 }
54
55 async function guardarUsuarios(usuarios) {
56   await fs.writeFile(USERS_FILE, JSON.stringify(usuarios, null, 2));
57 }
58
59 function authenticateToken(req, res, next) {
60   const authHeader = req.headers['authorization'];
61   const token = authHeader && authHeader.split(' ')[1];
62   if (!token) return res.status(401).json({ error: 'Token requerido' });
63   try {
64     const payload = jwt.verify(token, JWT_SECRET);
65     req.user = payload;
66     next();
67   } catch (err) {
68     return res.status(403).json({ error: 'Token inválido' });
69   }
70 }
```

F. MANEJO DE ERRORES Y DEBUGGING

```
211 // MANEJO DE ERRORES Y DEBUGGING
212 // EL MIDDLEWARE SIGUIENTE CAPTURA ERRORES Y DEVUELVE CÓDIGO HTTP
213 app.use((err, req, res, next) => {
214   const statusCode = err.status || 500;
215   res.status(statusCode).json({ error: err.message });
216 });
217
218 app.listen(PORT, () => {
219   console.log(`Servidor de Películas en http://localhost:\${PORT}`);
220 });
```

PRUEBAS DE FUNCIONAMIENTO

```
npm start  
Microsoft Windows [Version 10.0.26100.7840]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\PatoCV\Desktop\ACT 3\api-tareas>npm start  
  
> api-tareas@1.0.0 start  
> node server.js  
  
Servidor de Películas en http://localhost:3000
```

LOGIN

Usuario

Contraseña

[Entrar](#) [Registrarse](#)

Error: Credenciales inválidas

REGISTRO

Usuario

Contraseña

[Registrarse](#) [Volver a Login](#)

Registro correcto. Puedes iniciar sesión.

LOGIN

Usuario

Contraseña

[Entrar](#) [Registrarse](#)

Login correcto. Redirigiendo...

Películas Vistas

Película agregada [Logout](#)

Agregar Película

Nombre de la película:
Ej: Inception

Calificación (1-10):
Ej: 8

Comentarios:
Ej: Excelente película, muy recomendada

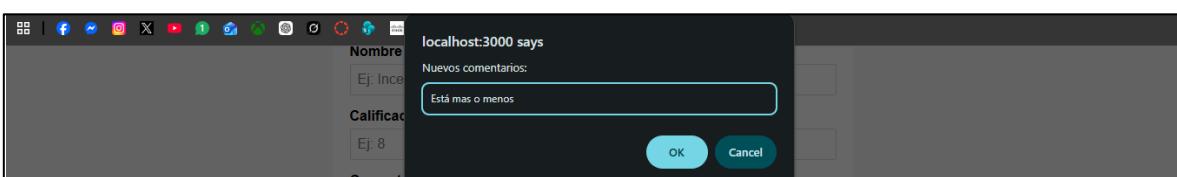
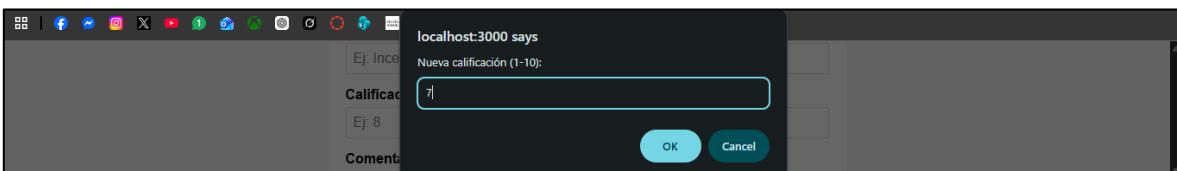
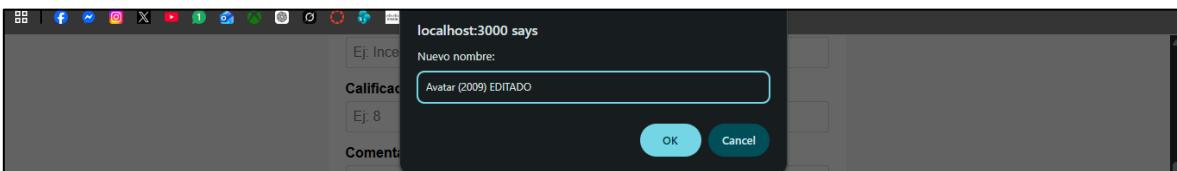
[Agregar Película](#)

Mis Películas

Avatar (2009)
Calificación: 9/10
Revolucionó el cine en 3D y efectos visuales. Visualmente impresionante.

Agregado: 2/18/2026, 4:45:08 PM

[Editar](#) [Eliminar](#)

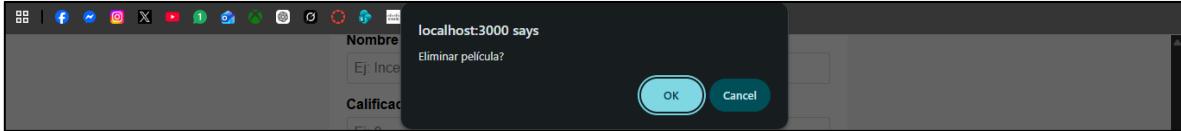


Mis Películas

Avatar (2009) EDITADO
Calificación: 7/10
Está mas o menos

Agregado: 2/18/2026, 4:45:08 PM

[Editar](#) [Eliminar](#)



Mis Películas

No hay películas agregadas

Mis Películas

To exit full screen, press and hold Esc

Avatar (2009)
Calificación: 8/10
Revolucionó el cine en 3D y efectos visuales. Visualmente impresionante.
Agregado: 2/18/2026, 4:53:10 PM
[Editar](#) [Eliminar](#)

Avengers: Endgame (2019)
Calificación: 9/10
Cierre épico de 10 años del UCM. Evento cultural masivo.
Agregado: 2/18/2026, 4:53:27 PM
[Editar](#) [Eliminar](#)

Avatar: The Way of Water (2022)
Calificación: 8/10
Expansión visual impresionante del mundo de Pandora.
Agregado: 2/18/2026, 4:53:44 PM
[Editar](#) [Eliminar](#)