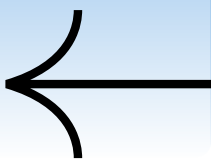
Actividad 3

Evelyn Michelle de la Torre Cornejo
T02989246
Miércoles 4 de Febrero del 2026

UNIVERSIDAD TECMILENIO



Descripción

En esta actividad, desarrollarás una API RESTful utilizando Node.js y Express.js para gestionar una lista de tareas (*to-do list*). La API permitirá realizar operaciones básicas de CRUD (crear, leer, actualizar y eliminar) sobre tareas almacenadas en un archivo JSON, utilizando los módulos nativos de Node.js como fs y http. La API debe incluir un sistema básico de autenticación y sesiones para asegurar el acceso a las rutas protegidas y se deberá implementar *middleware* personalizado para el manejo de errores y la validación de datos.

Durante la actividad, deberás poner en práctica conceptos como el *Event Loop*, la asincronía, la creación de servidores con Node.js y el manejo de rutas, y middleware con Express.js. Además, la actividad incluye la gestión de peticiones y respuestas, asegurando un manejo adecuado de los errores y aplicando técnicas de *debugging* para identificar y corregir problemas.

Objetivo

Implementar una API RESTful utilizando Node.js y Express.js que permita realizar operaciones CRUD y gestionar tareas almacenadas en archivos, integrando autenticación y sesiones para la protección de rutas, utilizando middleware personalizado para la validación de datos y el manejo adecuado de errores.

Instrucciones

- Configurar el proyecto Node.js.
  - Crea una carpeta para tu proyecto llamada api-tareas.
  - Inicializa el proyecto ejecutando el comando npm init -y para generar el archivo package.json.

    - Instala las dependencias necesarias con el siguiente comando:
        - npm install express body-parser jsonwebtoken bcryptjs.
- Creación de un servidor básico con Express.js.
    - Crea un archivo llamado server.js y configura un servidor básico usando Express.js.
    - Asegúrate de que el servidor escuche el puerto 3000.
    - Consulta un ejemplo básico de configuración de un servidor con Express.js que incluya manejo de errores utilizando una herramienta de inteligencia artificial (IA) como ChatGPT, Gemini AI, GitHub Copilot o cualquier otra que consideres adecuada para obtener ejemplos prácticos. Adapta la respuesta proporcionada y asegúrate de que cumpla con los requerimientos de la actividad. Esto te permitirá familiarizarte con soluciones generadas por IA y aprender a integrarlas de manera efectiva.
- Crear las rutas para la API RESTful.
    - Crea las siguientes rutas para gestionar las tareas:
        - GET /tareas: devuelve todas las tareas almacenadas en un archivo JSON.
        - POST /tareas: permite agregar una nueva tarea. La tarea debe incluir un título y una descripción.
        - PUT /tareas/: actualiza una tarea existente según el id proporcionado.
        - DELETE /tareas/: elimina una tarea según el id proporcionado.
- Manejo de datos con el módulo fs.
    - Utiliza el módulo fs de Node.js para leer y escribir las tareas en un archivo llamado tareas.json.
    - Asegúrate de que todas las operaciones de lectura y escritura sean asincrónicas utilizando fs.promises para evitar el bloqueo del Event Loop.
    - Consulta un ejemplo básico de implementación de operaciones de lectura y escritura de datos utilizando fs.promises mediante una herramienta de inteligencia artificial (IA), como ChatGPT, Gemini AI, GitHub Copilot, o cualquier otra que consideres adecuada para generar ejemplos claros y adaptables. Ajusta el código sugerido para trabajar con el archivo tareas.json de la actividad. Esto permitirá reforzar la comprensión de las operaciones asincrónicas y mejorar el manejo de datos en tu API.
- Implementación de autenticación y sesiones.
    - Crea una ruta para el registro de usuarios (POST /register) y otra para el inicio de sesión (POST /login).
    - Utiliza bcryptjs para encriptar las contraseñas de los usuarios y jsonwebtoken para generar *tokens* de autenticación.
    - Protege las rutas de la API (por ejemplo, /tareas) para que solo los usuarios autenticados puedan acceder a ellas utilizando middleware de autenticación.
    - Consulta un ejemplo práctico para encriptar contraseñas utilizando bcryptjs y generar tokens de autenticación con JWS, mediante una herramienta de inteligencia artificial (IA) como ChatGPT, Gemini AI, GitHub Copilot, o cualquier otra que consideres adecuada. Integra el código sugerido en las rutas /register y /login del proyecto. Esto te permitirá obtener soluciones prácticas y personalizables según las necesidades de la API, además de aprender a implementar medidas de seguridad de manera efectiva y flexible.
- Manejo de errores y debugging.
    - Implementa un middleware personalizado para capturar y gestionar errores en toda la aplicación. Asegúrate de devolver respuestas con el código de estado HTTP

adecuado (404, 500, etc.).
    - Utiliza herramientas como console.log y la opción --inspect de Node.js para depurar la API y corregir errores.
- Entrega de actividad.
    - En un archivo comprimido en formato .zip o .rar, o la URL de GitHub, incluye lo siguiente:
        - Archivo server.js que contenga la configuración del servidor, rutas y lógica de autenticación.
        - Archivo tareas.json con el conjunto de tareas almacenadas por la API.
        - En un documento Word explica cómo implementaste las rutas, el manejo de archivos y la autenticación en la API.
        - Elabora un video demostrativo donde muestres el funcionamiento de las diferentes rutas de la API, por ejemplo, la respuesta de las operaciones CRUD, la autenticación y la protección de rutas.

Explicación general del la actividad

Esta actividad aprendí a como desarrollar una API RESTful utilizando Node.js y Express.js para la gestión de tareas, donde implemente un sistema por medio de operaciones CRUD completo. Para así poder alcanzar el nivel de “Altamente competente", es fundamental que el almacenamiento de datos en archivos se realice mediante el módulo *fs* de forma asíncrona, asegurando que este no se bloquee en el *Event Loop* y que el servidor responda de forma eficiente en el puerto configurado en este caso el puerto 3000.

En la seguridad, en la actividad nos pide un sistema de autenticación basado en JWT (JSON Web Token) y el uso de bcryptjs para el cifrado de contraseñas. Esta configuración me ayuda garantizar que las credenciales se manejen de forma segura y que el acceso a las rutas de gestión de tareas esté restringido solamente a usuarios que ya se han autenticado, protegiendo así la integridad de la información y de las personas.

Al final utilice la una técnica en la cual implemente un middleware personalizado para la gestión de errores, el cual debe devolver códigos HTTP adecuados y mensajes claros. Al igual que el código debe destacar por la organización, la modularización y la documentación, demostrando el uso de las herramientas de depuración como lo son console.log y --inspect para así poder identificar y corregir los problemas de forma mucho más efectiva.

Repositorio de GitHub

https://github.com/m4d31n3v3n/DESARROLLO-FULL-STACK

Video de YouTube

https://www.youtube.com/watch?v=s26kwCq-sQI&t=12s

Código JavaScript

```
const express = require('express');
const fs = require('fs').promises;
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const app = express();
const PORT = 3000;
const SECRET_KEY = '140506'; // En producción, usa variables de entorno

// Archivos de datos
const PATH_TAREAS = './tareas.json';
const PATH_USUARIOS = './usuarios.json';

app.use(express.json());

// --- FUNCIONES AUXILIARES PARA FS ---
async function leerArchivo(path) {
    try {
        const data = await fs.readFile(path, 'utf8');
        return JSON.parse(data);
    } catch (error) {
        return []; // Si el archivo no existe o está vacío
    }
}

async function escribirArchivo(path, data) {
```

```
    await fs.writeFile(path, JSON.stringify(data, null, 2));
}

// --- MIDDLEWARE DE AUTENTICACIÓN ---
const verificarToken = (req, res, next) => {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];

    if (!token) return res.status(401).json({ error: 'Acceso denegado, token inexistente' });

    try {
        const verificado = jwt.verify(token, SECRET_KEY);
        req.user = verificado;
        next();
    } catch (err) {
        res.status(403).json({ error: 'Token no es válido' });
    }
};

// --- RUTAS DE AUTENTICACIÓN ---

app.post('/register', async (req, res, next) => {
    try {
        const { email, password } = req.body;
        const usuarios = await leerArchivo(PATH_USUARIOS);

        if (usuarios.find(u => u.email === email)) {
            return res.status(400).json({ error: 'El usuario ya existe' });
        }
```

```
        const hashedPassword = await bcrypt.hash(password, 10);
        const nuevoUsuario = { id: Date.now(), email, password: hashedPassword };

        usuarios.push(nuevoUsuario);
        await escribirArchivo(PATH_USUARIOS, usuarios);

        res.status(201).json({ message: 'Usuario registrado con éxito' });
    } catch (error) {
        next(error);
    }
});

app.post('/login', async (req, res, next) => {
    try {
        const { email, password } = req.body;
        const usuarios = await leerArchivo(PATH_USUARIOS);
        const usuario = usuarios.find(u => u.email === email);

        if (!usuario || !(await bcrypt.compare(password, usuario.password))) {
            return res.status(400).json({ error: 'Credenciales inválidas' });
        }

        const token = jwt.sign({ id: usuario.id, email: usuario.email }, SECRET_KEY, { expiresIn: '1h' });
        res.json({ token });
    } catch (error) {
        next(error);
    }
});
```

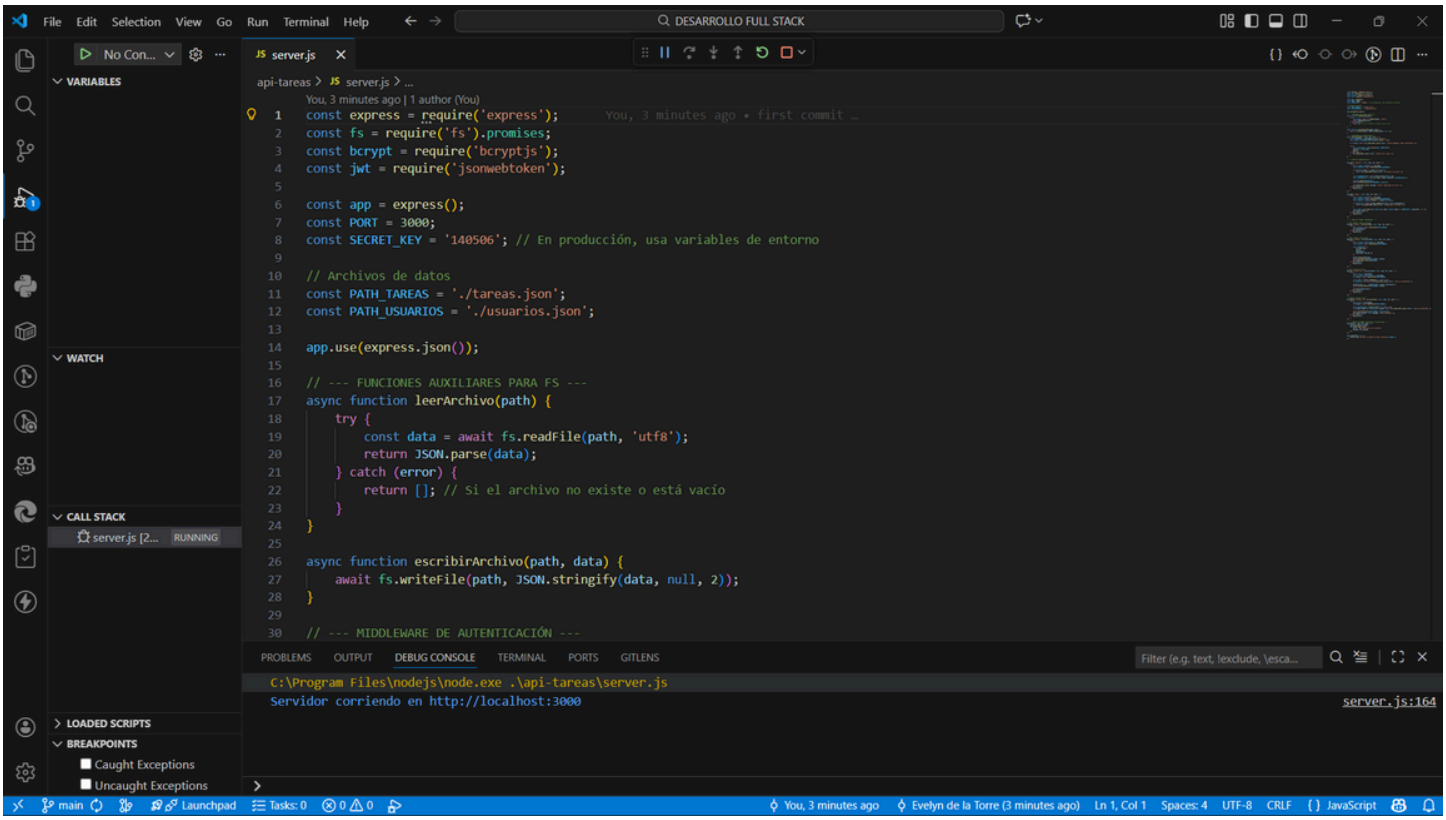
```
// --- RUTAS DE TAREAS (PROTEGIDAS) ---

// GET: Obtener todas las tareas
app.get('/tareas', verificarToken, async (req, res, next) => {
    try {
        const tareas = await leerArchivo(PATH_TAREAS);
        res.json(tareas);
    } catch (error) {
        next(error);
    }
});

// POST: Agregar nueva tarea
app.post('/tareas', verificarToken, async (req, res, next) => {
    try {
        const { titulo, descripcion } = req.body;
        const tareas = await leerArchivo(PATH_TAREAS);

        const nuevaTarea = {
            id: Date.now(),
            titulo,
            descripcion,
            usuarioId: req.user.id
        };

        tareas.push(nuevaTarea);
        await escribirArchivo(PATH_TAREAS, tareas);
        res.status(201).json(nuevaTarea);
```

```
    } catch (error) {
        next(error);
    }
});

// PUT: Actualizar tarea
app.put('/tareas/:id', verificarToken, async (req, res, next) => {
    try {
        const { id } = req.params;
        const { titulo, descripcion } = req.body;
        let tareas = await leerArchivo(PATH_TAREAS);

        const index = tareas.findIndex(t => t.id == id);
        if (index === -1) return res.status(404).json({ error: 'Tarea no encontrada' });

        tareas[index] = { ...tareas[index], titulo, descripcion };
        await escribirArchivo(PATH_TAREAS, tareas);

        res.json(tareas[index]);
    } catch (error) {
        next(error);
    }
});

// DELETE: Eliminar tarea
app.delete('/tareas/:id', verificarToken, async (req, res, next) => {
    try {
        const { id } = req.params;
        let tareas = await leerArchivo(PATH_TAREAS);
```

```
        const nuevasTareas = tareas.filter(t => t.id != id);
        if (tareas.length === nuevasTareas.length) return res.status(404).json({ error: 'Tarea no encontrada' });

        await escribirArchivo(PATH_TAREAS, nuevasTareas);
        res.json({ success: true, message: 'Tarea eliminada' });
    } catch (error) {
        next(error);
    }
});

// --- MANEJO DE ERRORES (Middleware personalizado) ---
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).json({
        error: 'Algo salió mal en el servidor',
        message: err.message
    });
});

app.listen(PORT, () => {
    console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

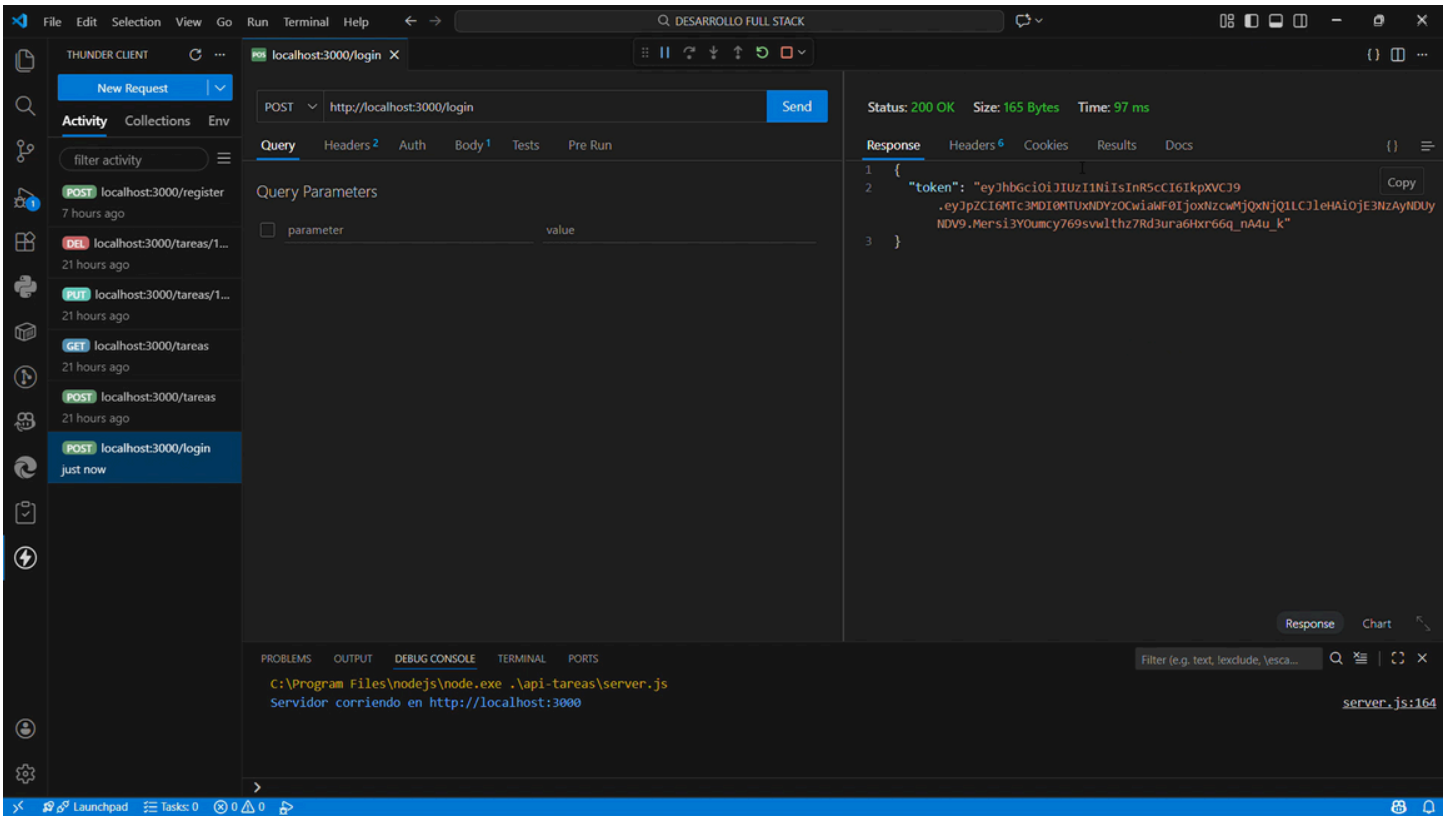
Capturas del funcionamiento del programa

SERVER

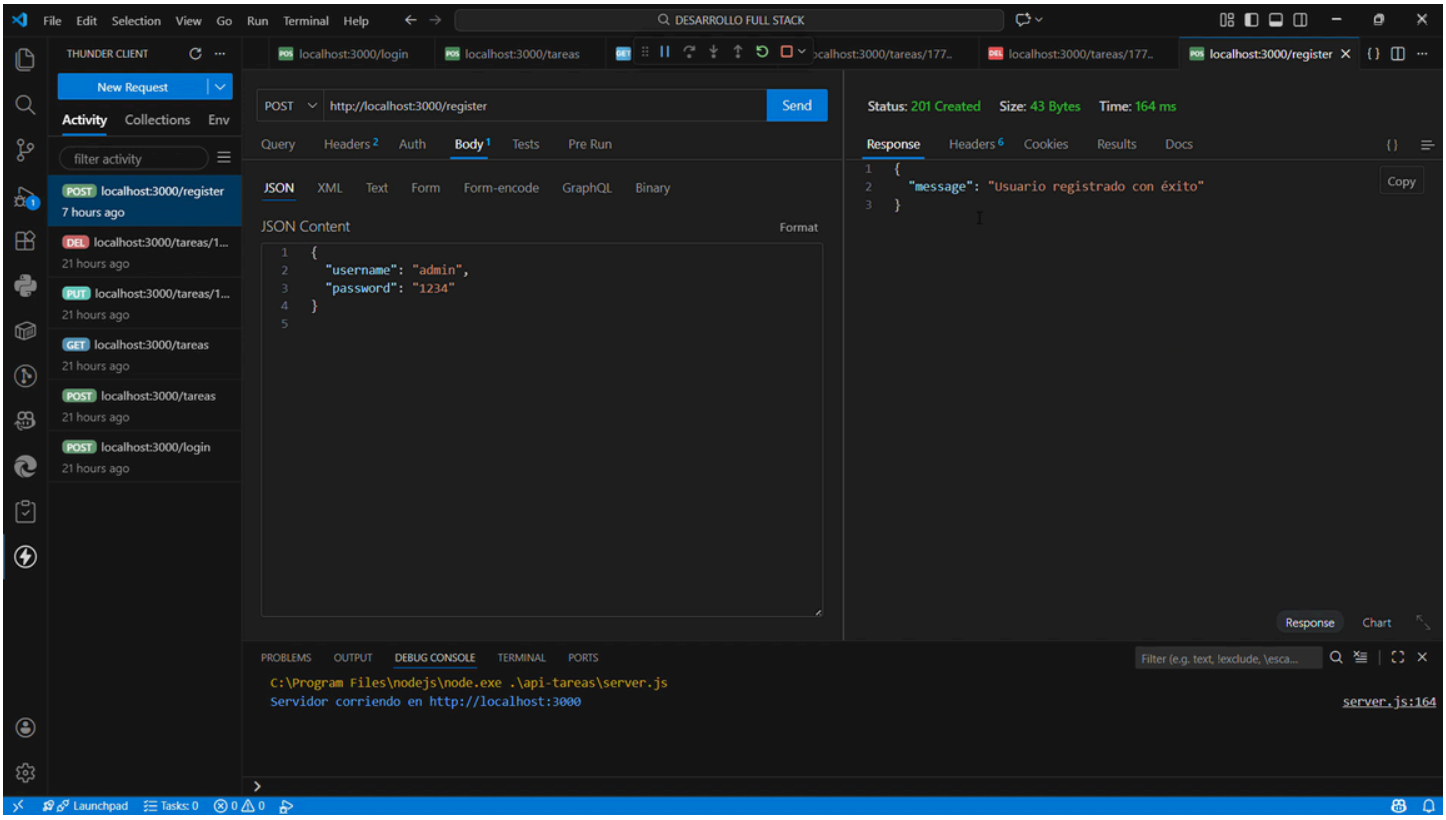


THUNDER CLIENT

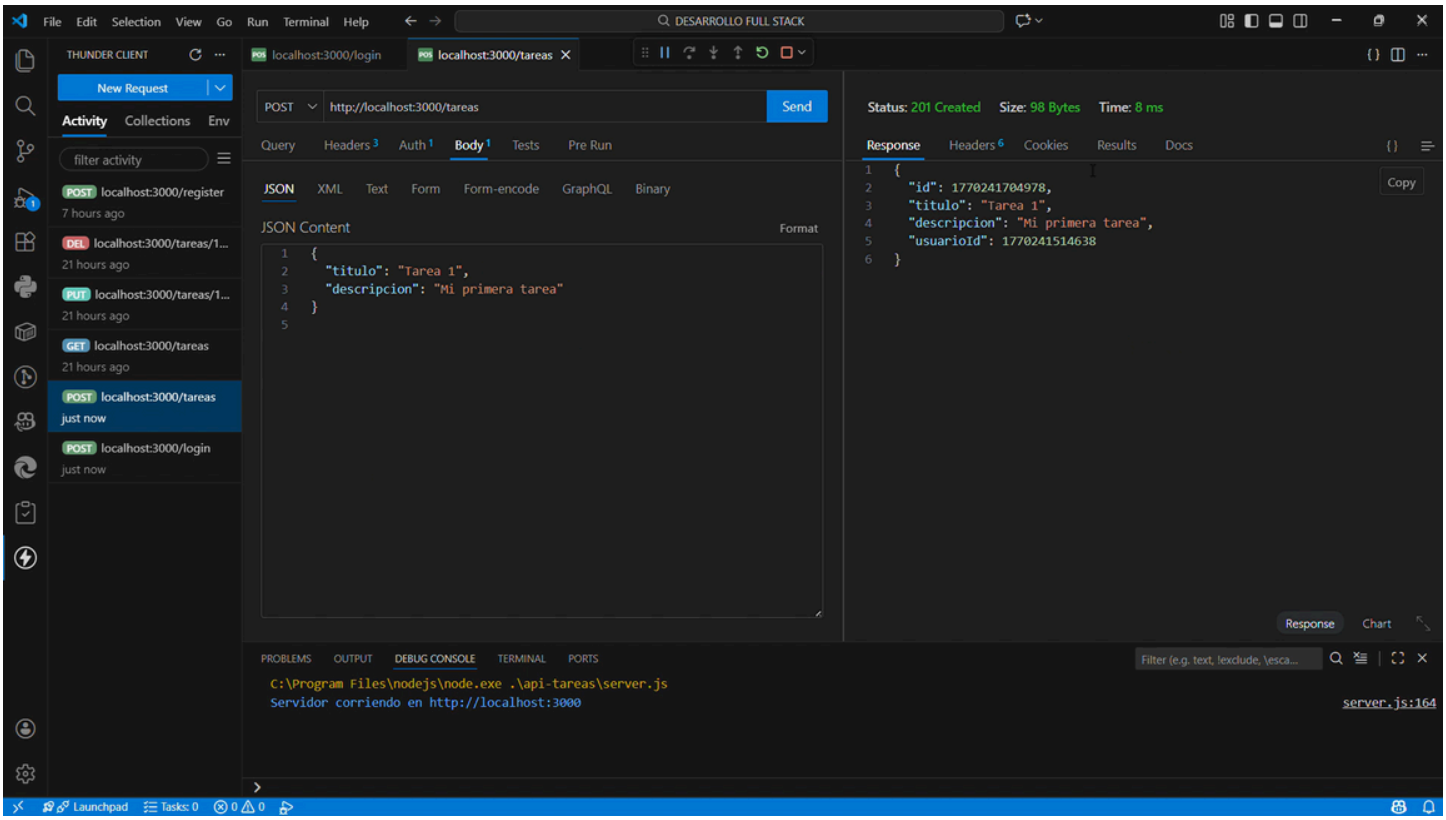
- **POST Login**



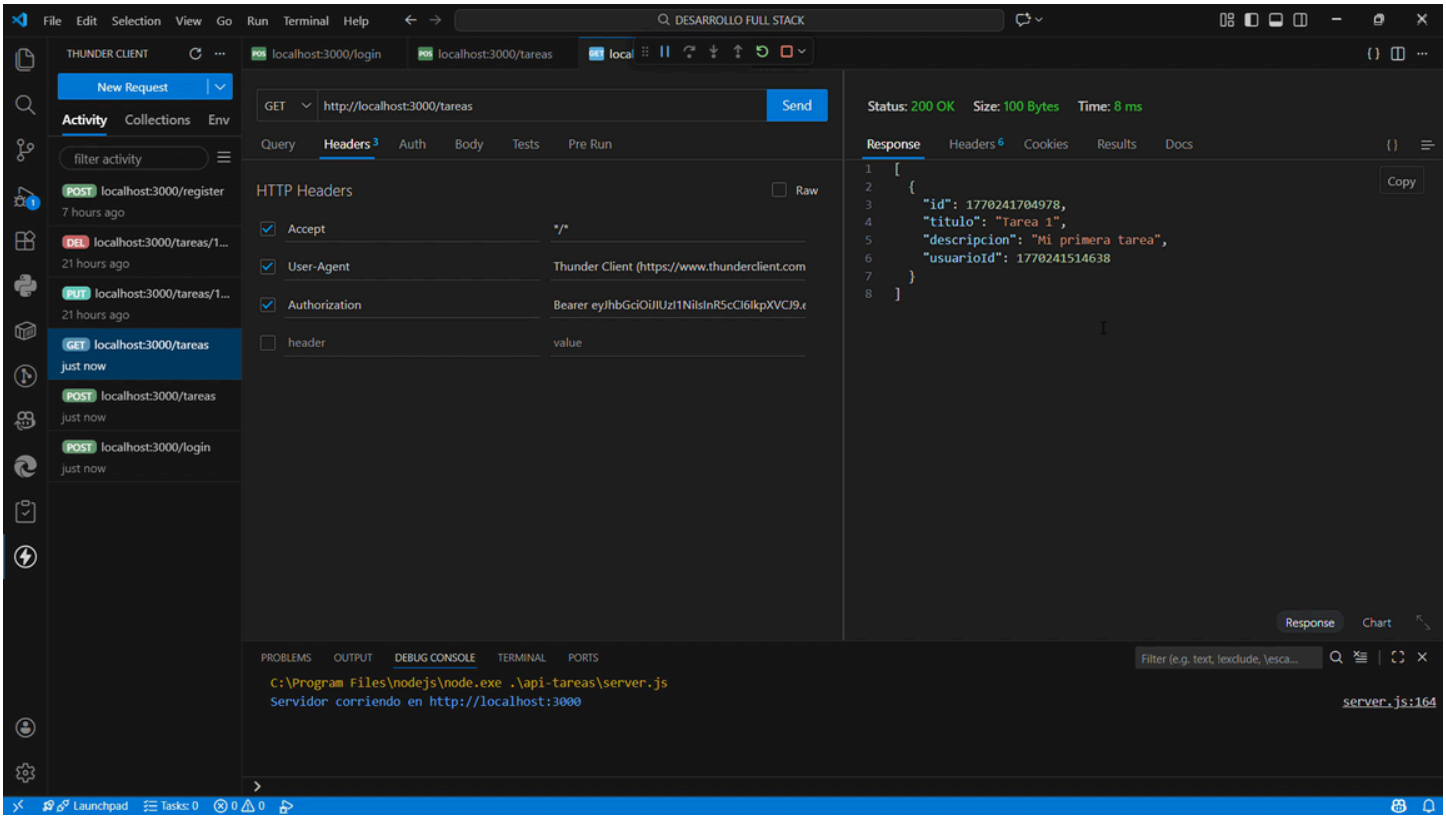
- **POST Register**



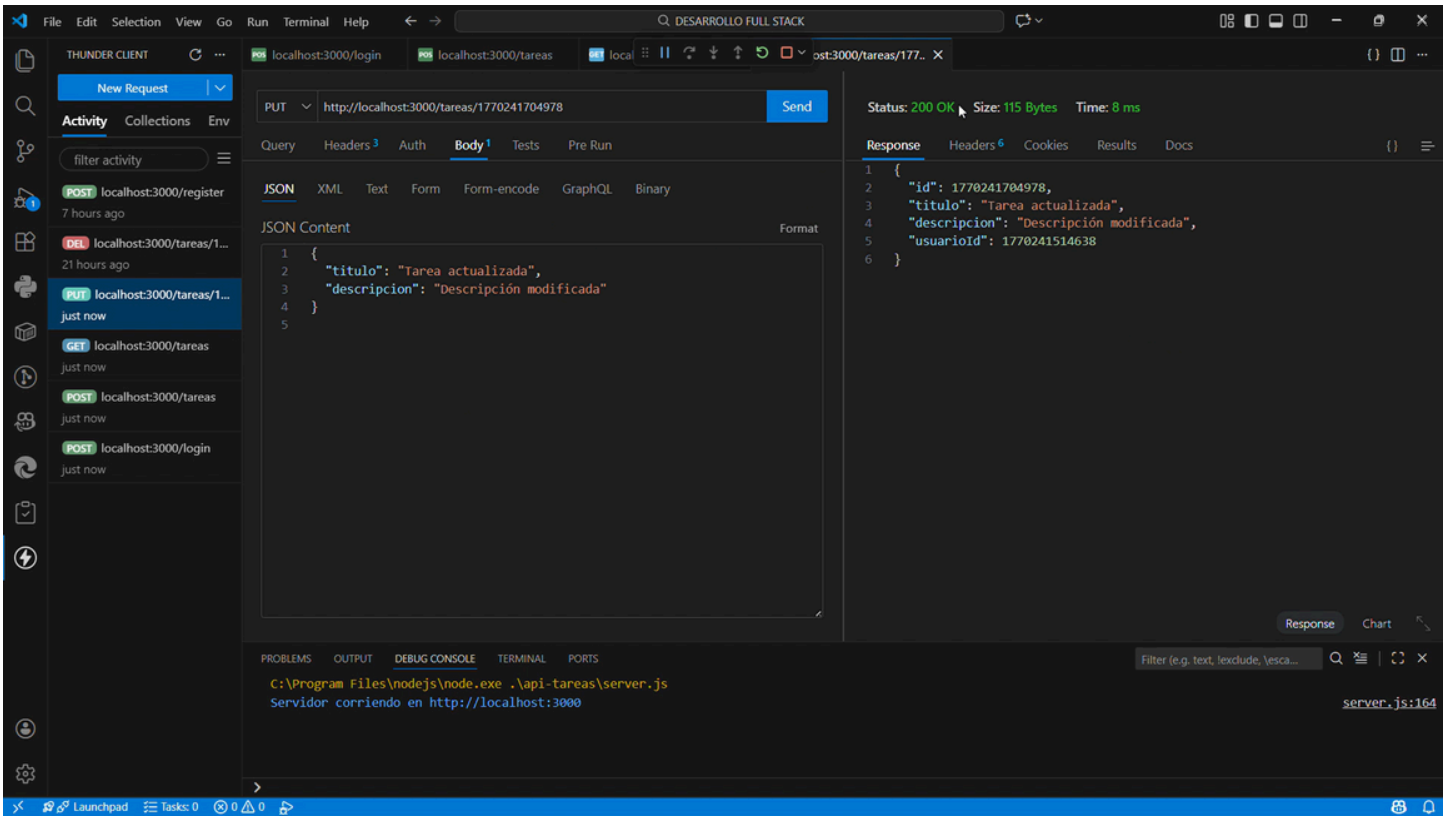
- **POST Tareas**



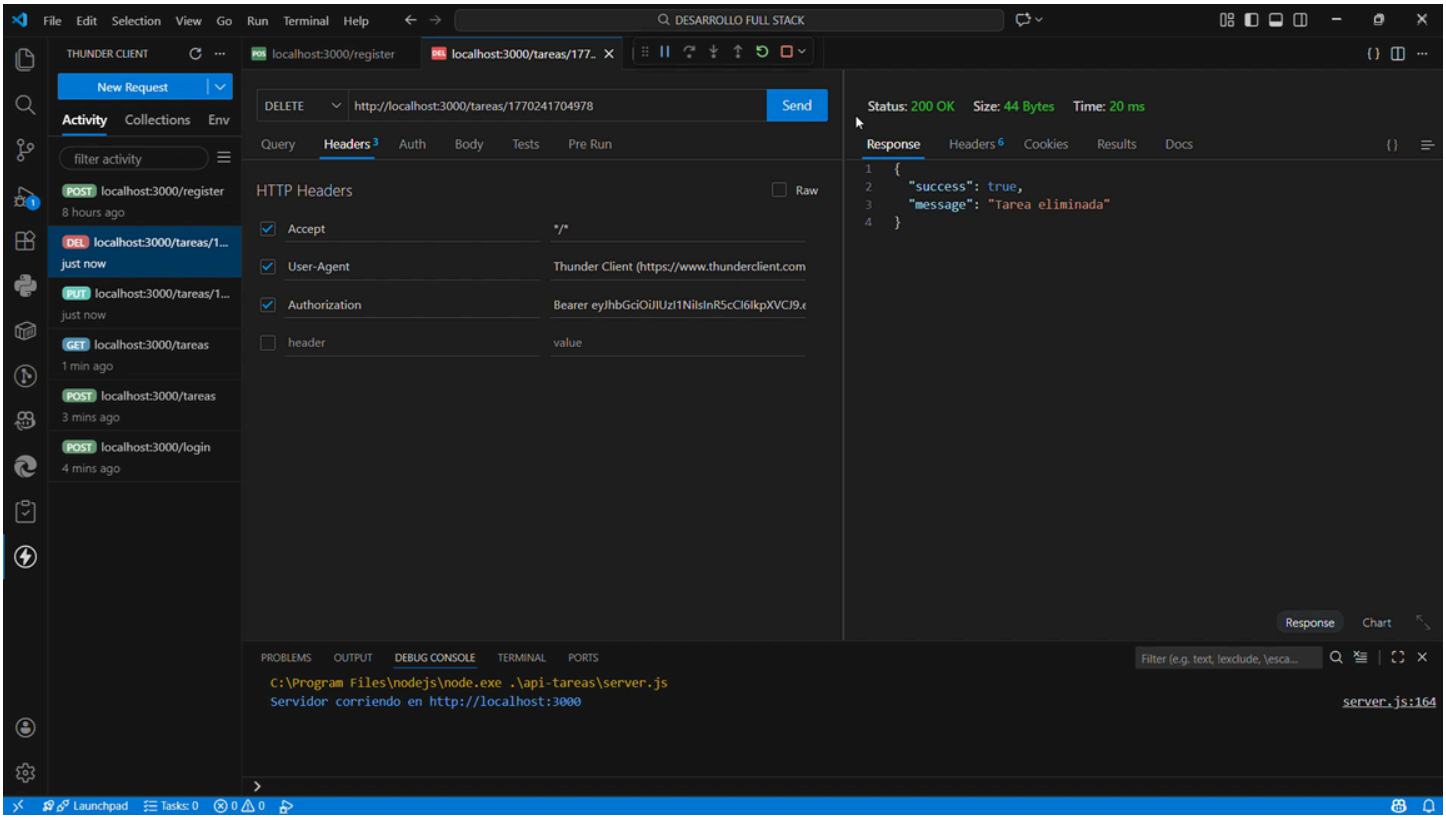
- **GET Tareas**



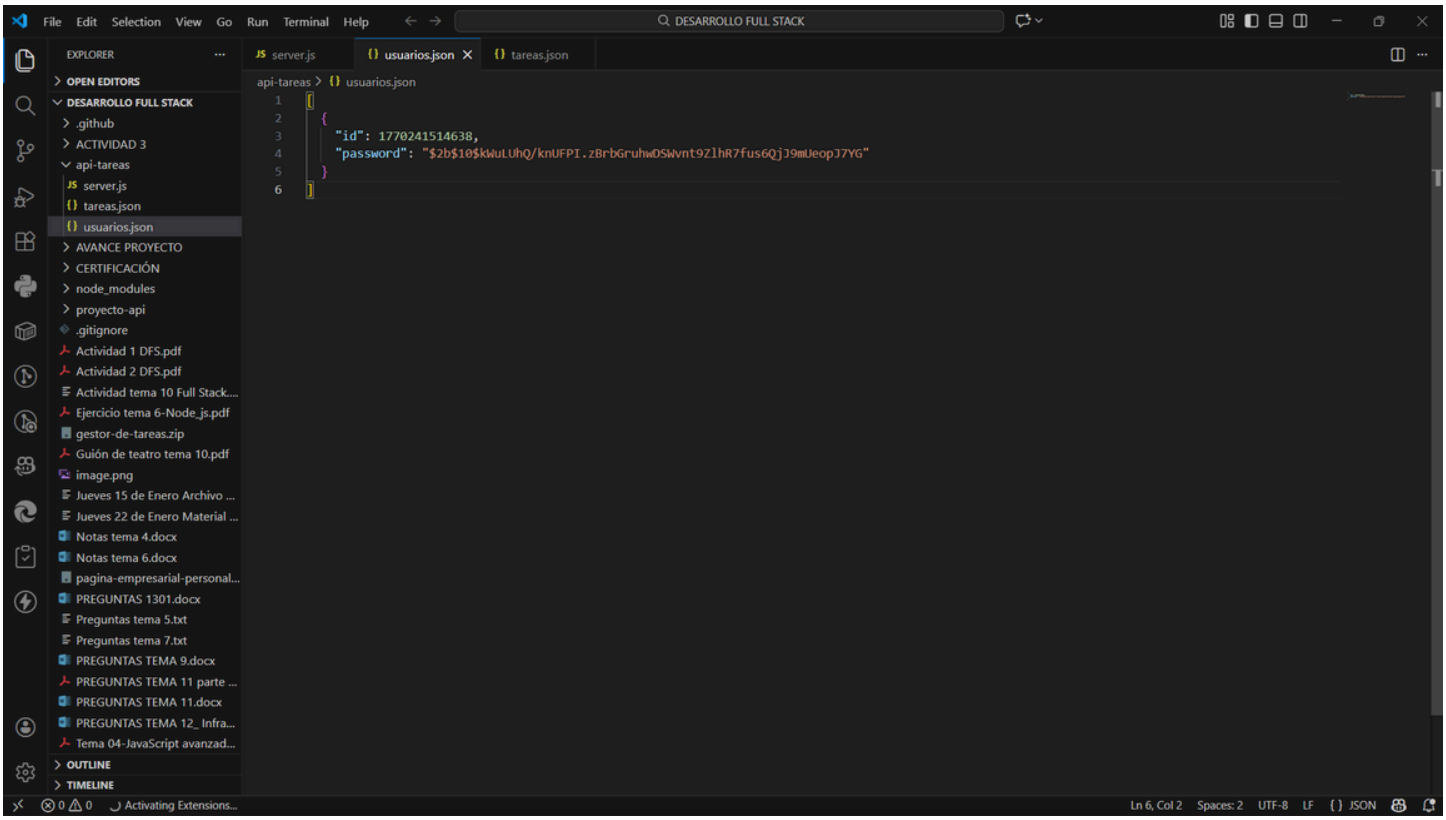
- **PUT Tareas**



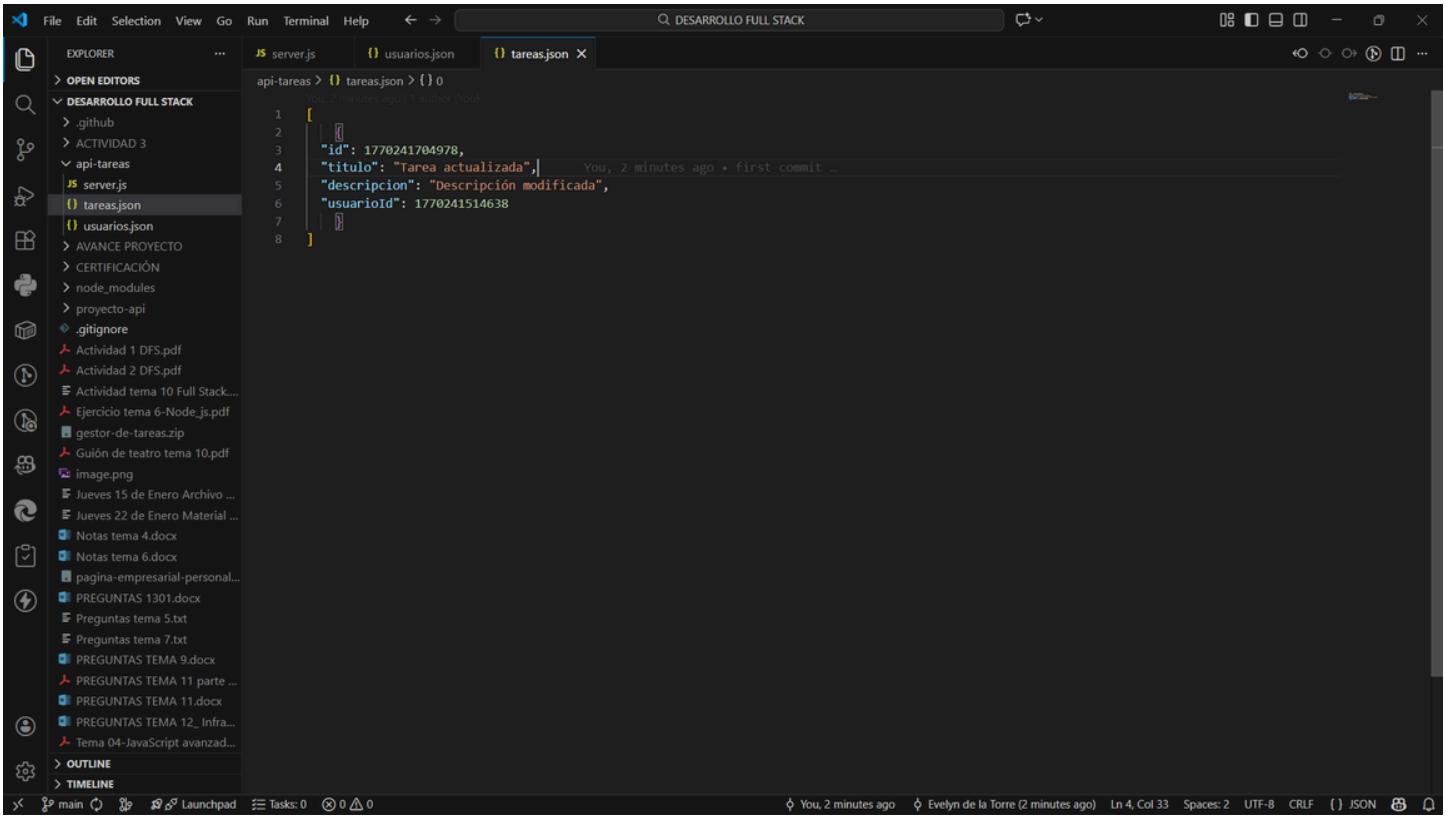
- **DELETE Tareas**



USUARIOS


File Edit Selection View Go Run Terminal Help
DESARROLLO FULL STACK
EXPLORER
server.js
usuarios.json
tareas.json
api-tareas > usuarios.json
{
"id": 1770241514638,
"password": "$2b$10$kWuLUhQ/knUFPI.zBrbGruhwOSWvnt9ZlhR7fus6QjJ9mUeopJ7YG"
}
]
OPEN EDITORS
DESARROLLO FULL STACK
.github
ACTIVIDAD 3
api-tareas
server.js
tareas.json
usuarios.json
AVANCE PROYECTO
CERTIFICACIÓN
node_modules
proyecto-api
.gitignore
Actividad 1 DFS.pdf
Actividad 2 DFS.pdf
Actividad tema 10 Full Stack....
Ejercicio tema 6-Node_js.pdf
gestor-de-tareas.zip
Guión de teatro tema 10.pdf
image.png
Jueves 15 de Enero Archivo ...
Jueves 22 de Enero Material ...
Notas tema 4.docx
Notas tema 6.docx
pagina-empresarial-personal...
PREGUNTAS 1301.docx
Preguntas tema 5.txt
Preguntas tema 7.txt
PREGUNTAS TEMA 9.docx
PREGUNTAS TEMA 11 parte ...
PREGUNTAS TEMA 11.docx
PREGUNTAS TEMA 12_ Infra...
Tema 04-JavaScript avanzad...
OUTLINE
TIMELINE
Activating Extensions...
Ln 6, Col 2 Spaces: 2 UTF-8 LF { } JSON

TAREAS


File Edit Selection View Go Run Terminal Help
DESARROLLO FULL STACK
EXPLORER
server.js
usuarios.json
tareas.json
api-tareas > tareas.json > { } 0
You, 2 minutes ago | 1 author (You)
[
{
"id": 1770241704978,
"titulo": "Tarea actualizada",
You, 2 minutes ago • first commit ...
"descripcion": "Descripción modificada",
"usuarioId": 1770241514638
}
]
OPEN EDITORS
DESARROLLO FULL STACK
.github
ACTIVIDAD 3
api-tareas
server.js
tareas.json
usuarios.json
AVANCE PROYECTO
CERTIFICACIÓN
node_modules
proyecto-api
.gitignore
Actividad 1 DFS.pdf
Actividad 2 DFS.pdf
Actividad tema 10 Full Stack....
Ejercicio tema 6-Node_js.pdf
gestor-de-tareas.zip
Guión de teatro tema 10.pdf
image.png
Jueves 15 de Enero Archivo ...
Jueves 22 de Enero Material ...
Notas tema 4.docx
Notas tema 6.docx
pagina-empresarial-personal...
PREGUNTAS 1301.docx
Preguntas tema 5.txt
Preguntas tema 7.txt
PREGUNTAS TEMA 9.docx
PREGUNTAS TEMA 11 parte ...
PREGUNTAS TEMA 11.docx
PREGUNTAS TEMA 12_ Infra...
Tema 04-JavaScript avanzad...
OUTLINE
TIMELINE
main Launchpad Tasks: 0
You, 2 minutes ago Evelyn de la Torre (2 minutes ago) Ln 4, Col 33 Spaces: 2 UTF-8 CRLF { } JSON