Actividad: Paso a Paso Seguridad API

Objetivo

El objetivo de este trabajo práctico es que los alumnos desarrollen una API RESTful utilizando Express.js para la gestión de libros, almacenando los datos en una base de datos con MongoDB y que puedan implementar Autenticación y Autorización para garantizar la seguridad de la API.

La API permitirá realizar operaciones básicas como crear, leer, actualizar y eliminar libros, así como obtener una lista de todos los libros disponibles.

Requisitos

1. Instalar el motor MongoDB en la computadora:
    a. https://docs.google.com/document/d/1qam4mqHj14EABzf9GPwFR-jdB-ufOemU-XVKWe41wB4/edit?usp=sharing

Consigna

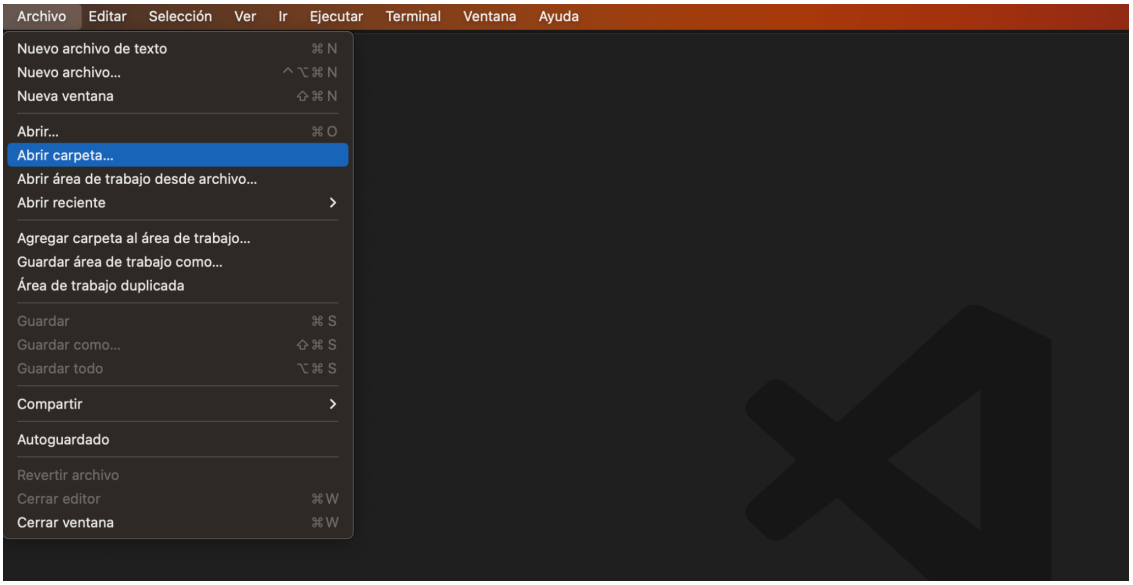
1. La API debe seguir los principios de arquitectura RESTful y utilizar los métodos HTTP adecuados para cada operación.
2. La API debe implementar las siguientes rutas y funcionalidades:
    a. GET /libros: Devuelve la lista completa de libros.
    b. GET /libros/:id: Devuelve los detalles de un libro específico según su ID.
    c. POST /libros: Crea un nuevo libro con la información proporcionada.
    d. PUT /libros/:id: Actualiza la información de un libro específico según su ID.
    e. DELETE /libros/:id: Elimina un libro específico según su ID.
3. La API debe utilizar una estructura de archivos y carpetas organizada para separar las rutas y los controladores de libros.
4. La API debe manejar adecuadamente los errores y devolver respuestas JSON apropiadas en caso de errores.
5. Se debe implementar mediante el ORM Mongoose el acceso a una base de datos de libros
6. Implementar autenticación en la API, validando Token OAuth 2.0 en cada petición HTTP de cualquier ruta.
7. Implementar autorización en la API, validando accesos a las rutas mediante SCOPES, a continuación para cada ruta el scope con el que se debe validar:
    a. GET /libros - Scope: **read:libros**
    b. GET /libros/:id - Scope: **read:libros**
    c. POST /libros - Scope: **write:libros**
    d. PUT /libros/:id - Scope: **write:libros**

e. DELETE /libros/:id - Scope: **write:libros**

Desarrollo Paso a Paso

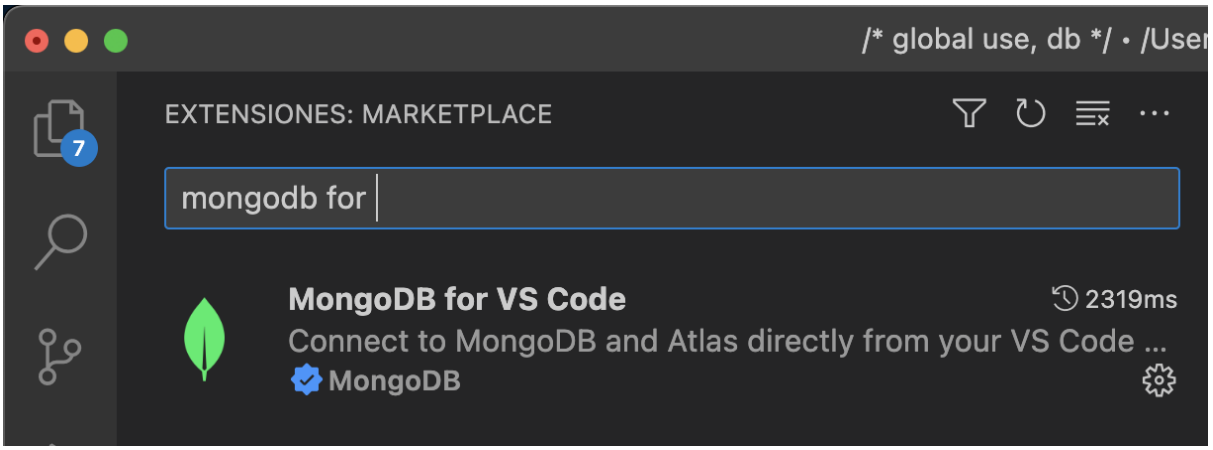
| Nota: En caso de tener ya desarrollada la API de Biblioteca, iniciar desde el punto 26.

1. Iniciamos la aplicación VSCode.
2. Ir al menú Archivo -> Abrir Carpeta

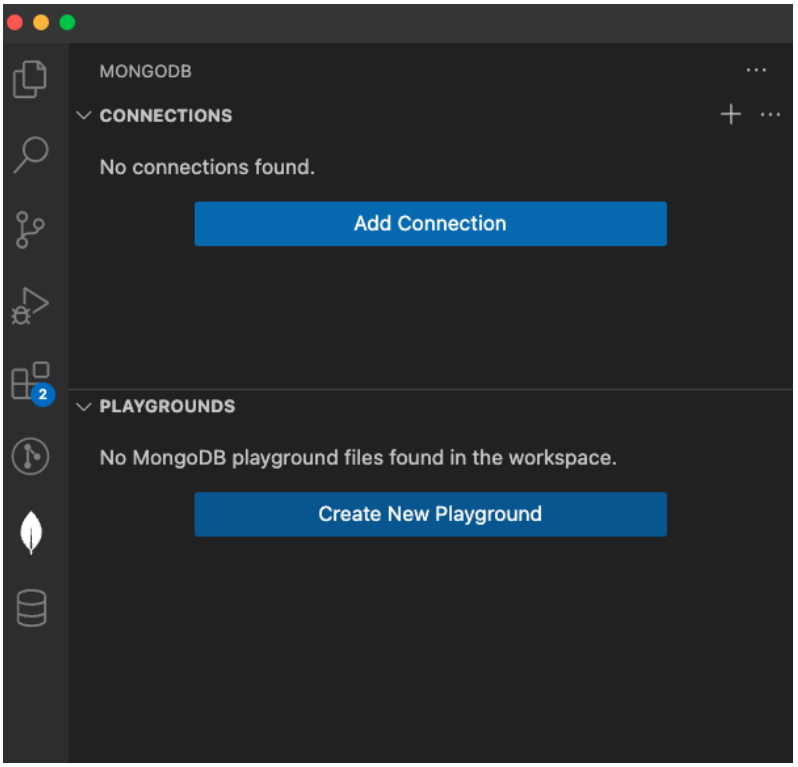


3. Del paso anterior se abre una ventana de búsqueda de carpetas, aquí buscamos la carpeta Documentos, seleccionamos y luego elegimos la opción “Nueva Carpeta”, indicando el nombre de la carpeta **api_biblioteca**.
4. Selecciona la carpeta **api_biblioteca** para que se abra en VSCode.

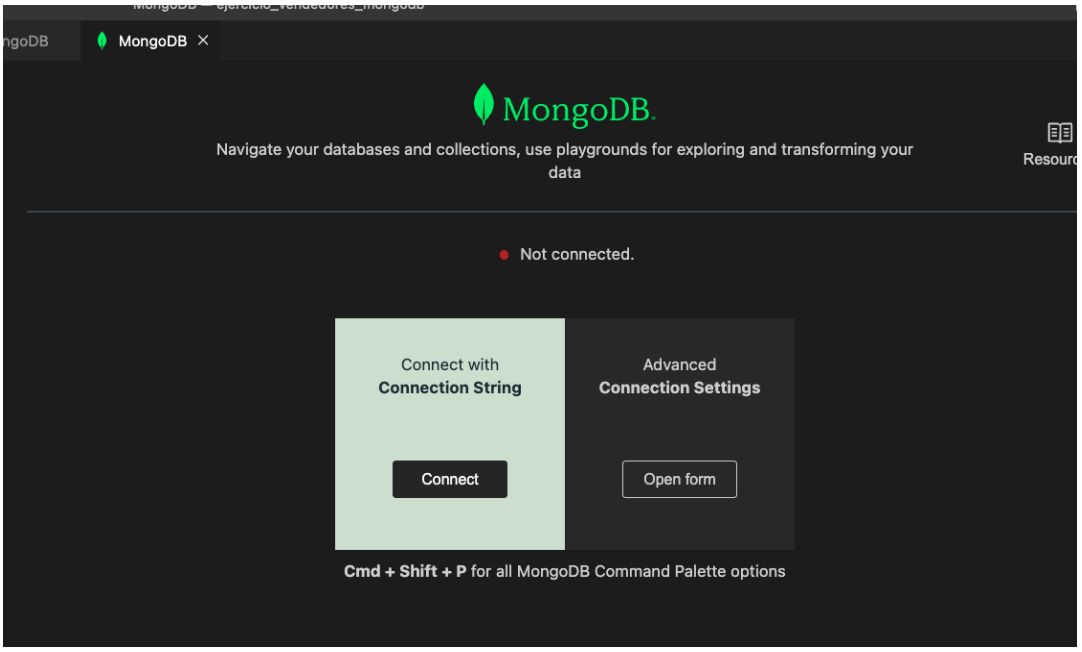
5. Antes de empezar vamos a crear la base de datos con la extensión MongoDB for VSCode:



6. Creamos una conexión al motor de base de datos local, haciendo click en la opción AddConnection:



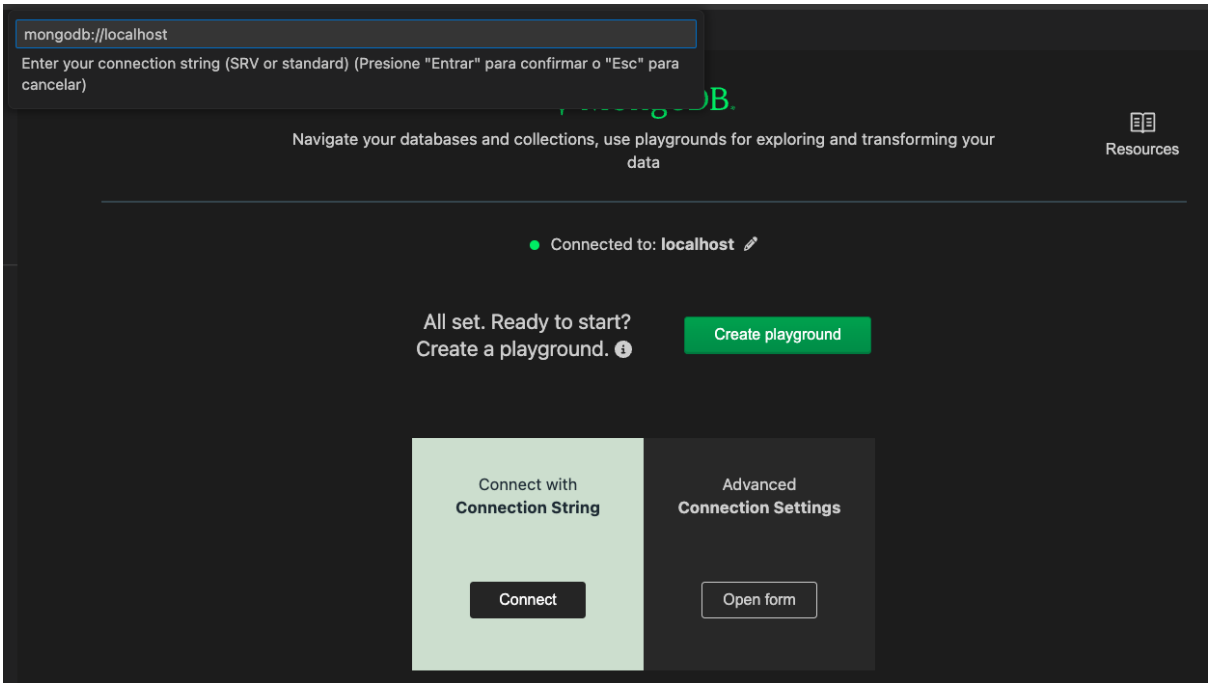
7. Luego hacemos click en la opción Connect



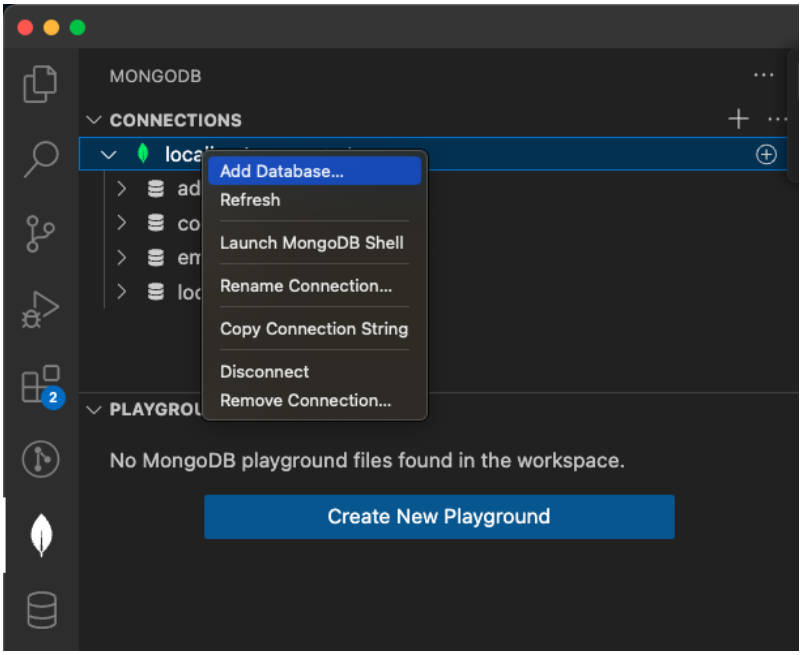
8. Ingresamos la siguiente cadena de conexión:

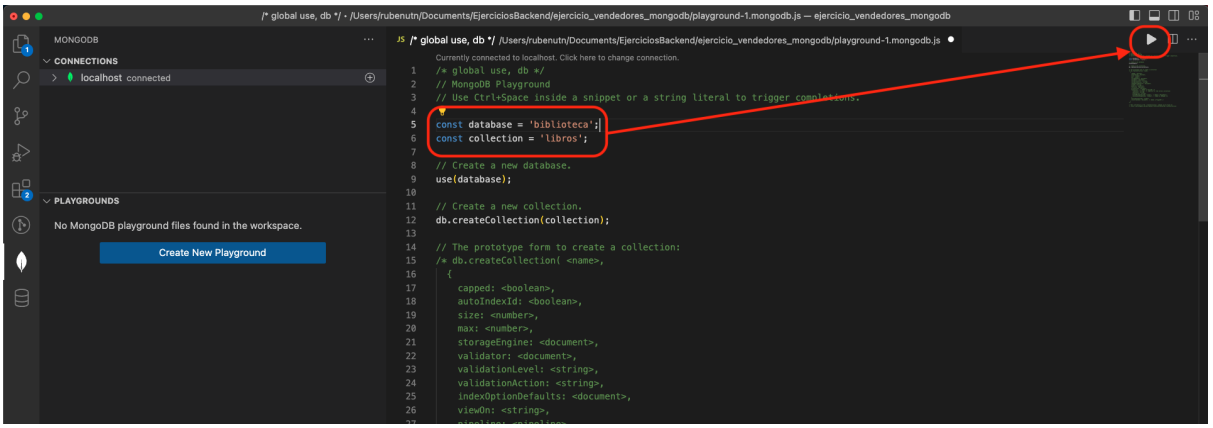
   mongodb://localhost

quedando así:

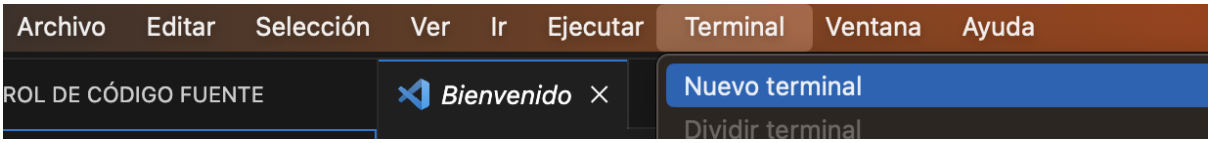


9. Ahora vamos a crear una base de datos de nombre “Biblioteca”:

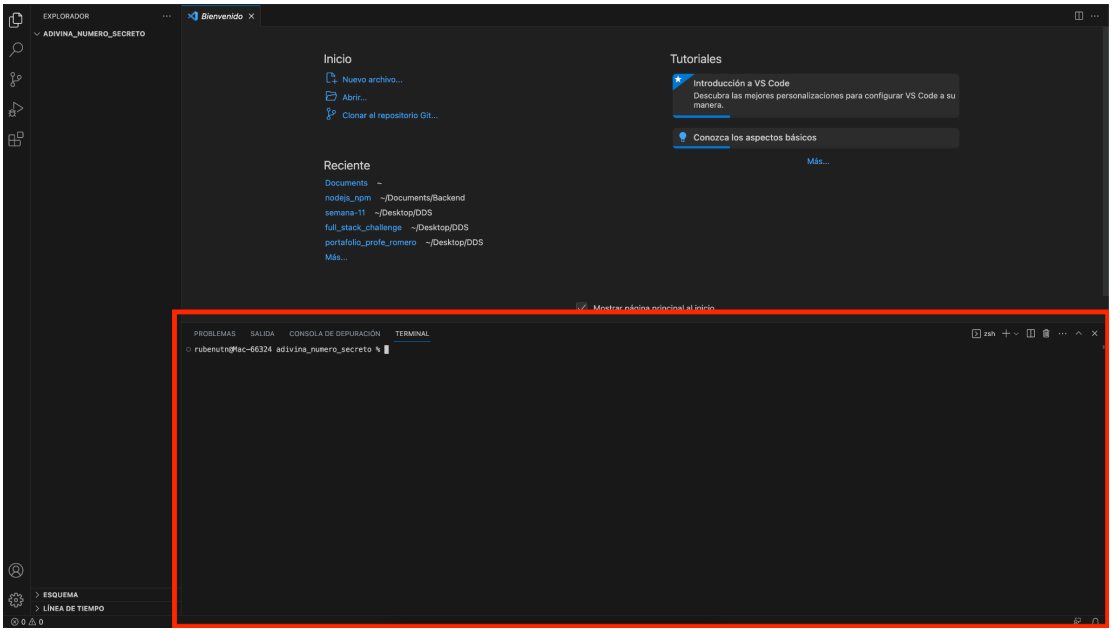




10. Ahora vamos a crear el proyecto en VSCode buscamos en el menú la opción **Terminal -> Nuevo Terminal:**



11. En la siguiente imagen podemos ver en VSCode la sección de Terminal:



12. En el Terminal vamos a escribir el siguiente comando y luego apretar la tecla enter (para que se ejecute el comando):

```
npm init -y
```

Resultado esperado del comando:

```
> npm init -y
Wrote to /Users/rubenutn/Documents/api_biblioteca/package.json:

{
  "name": "api_biblioteca",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

13. Vamos a instalar la librería **express**, que se utilizará para APIs. En el Terminal vamos a escribir el siguiente comando y luego apretar la tecla enter (para que se ejecute el comando):

```
npm install express
```

14. Vamos a instalar la librería **mongoose**, para acceder a la base de datos. En el Terminal vamos a escribir el siguiente comando y luego apretar la tecla enter (para que se ejecute el comando):

```
npm install mongoose
```

15. Seguimos instalando la librería **joi**, para esto en el Terminal vamos a escribir el siguiente comando y luego enter:

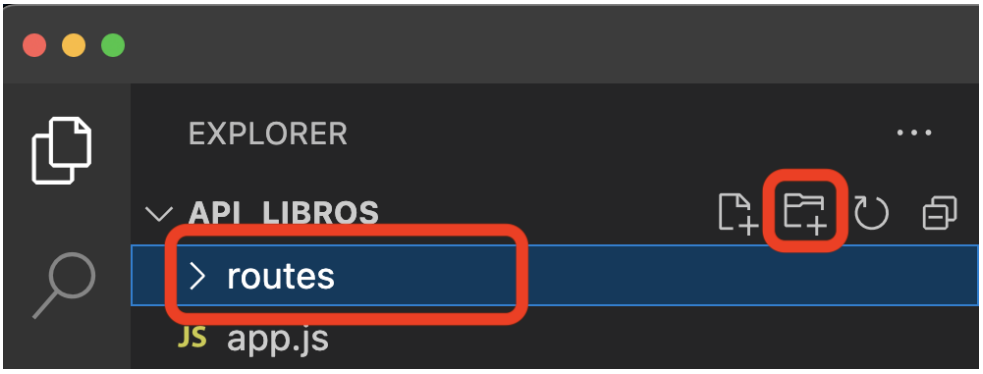
```
npm install joi
```

16. Continuamos creando en VSCode un archivo **app.js** con el siguiente código:

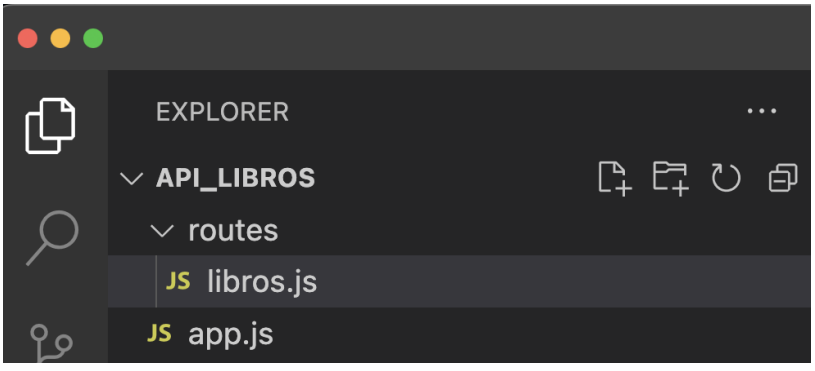
```
const express = require('express');
const app = express();
app.use(express.json());

// Importamos el Router de Libros
const librosRouter = require('./routes/libros');

// Importamos el Middleware Error Handler
const errorHandler = require('./middlewares/errorHandler');
```

```
app.use('/libros', librosRouter);

app.use(errorHandler);

app.listen(3000, () => {
  console.log('Servidor iniciado en el puerto 3000');
});
```

17. Ahora vamos a crear una carpeta **routes** para dejar todos los archivos de enrutamiento:



18. Creamos dentro de la carpeta **routes** el archivo **libros.js**:



19. Agregamos el siguiente código al archivo **libros.js**:

```
const express = require("express");
const router = express.Router();

const Libro = require("../models/Libro");
```

```
// Ruta para obtener todos los libros
router.get("/", async (req, res) => {
  try {
    const libros = await Libro.find();
    res.json(libros);
  } catch (error) {
    res.status(500).json({ error: "Error al obtener los libros" });
  }
});

// Ruta para crear un nuevo Libro
router.post("/", async (req, res) => {
  try {
    const nuevoLibro = new Libro(req.body);
    await nuevoLibro.save();
    res.json(nuevoLibro);
  } catch (error) {
    res.status(500).json({ error: "Error al crear el Libro" });
  }
});

// Ruta para actualizar un Libro existente
router.put("/:id", async (req, res) => {
  try {
    const Libro = await Libro.findByIdAndUpdate(req.params.id, req.body,
{
      new: true,
    });
    res.json(Libro);
  } catch (error) {
    res.status(500).json({ error: "Error al actualizar el Libro" });
  }
});

// Ruta para eliminar un Libro
router.delete('/:id', async (req, res) => {
    try {
      await Libro.findByIdAndDelete(req.params.id);
      res.json({ message: 'Libro eliminado correctamente' });
    } catch (error) {
      res.status(500).json({ error: 'Error al eliminar el Libro' });
    }
  });
```

```
module.exports = router;
```

20. Creamos una carpeta **models.**
21. Creamos el archivo **Libro.js** en la carpeta **models** con el siguiente código fuente:

```
const mongoose = require('mongoose');

mongoose.connect("mongodb://localhost:27017/biblioteca", {
  useUnifiedTopology: true,
  useNewUrlParser: true,
});

const LibroSchema = new mongoose.Schema({
  titulo: String,
  autor: String
}, { collection: 'libros' });

const Libro = mongoose.model('Libro', LibroSchema);

module.exports = Libro;
```

22. Abrimos una terminal en Visual Studio Code. Para hacerlo, selecciona "Terminal" en la barra de menú superior y luego selecciona "Nueva terminal".

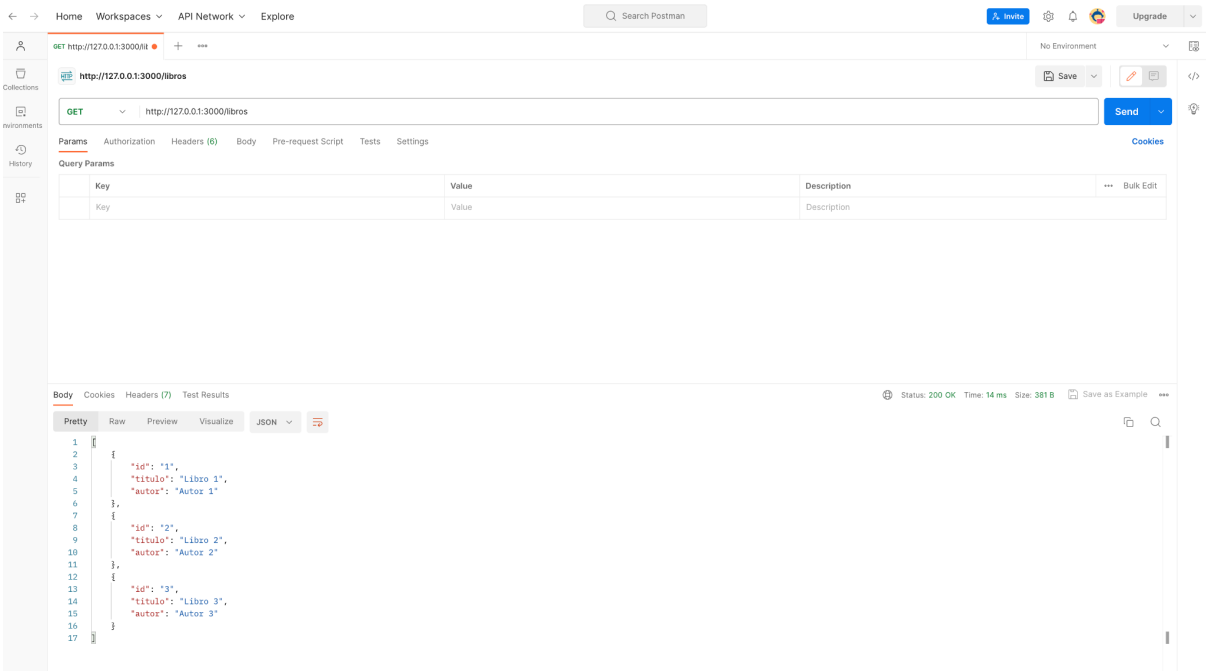
23. Ejecutamos la aplicación utilizando el Terminal de VSCode ejecutando el siguiente comando:

```
node app.js
```

Resultado esperado:

Si todo está bien, vas a ver un mensaje en la consola que dice "Servidor iniciado en el puerto 3000".

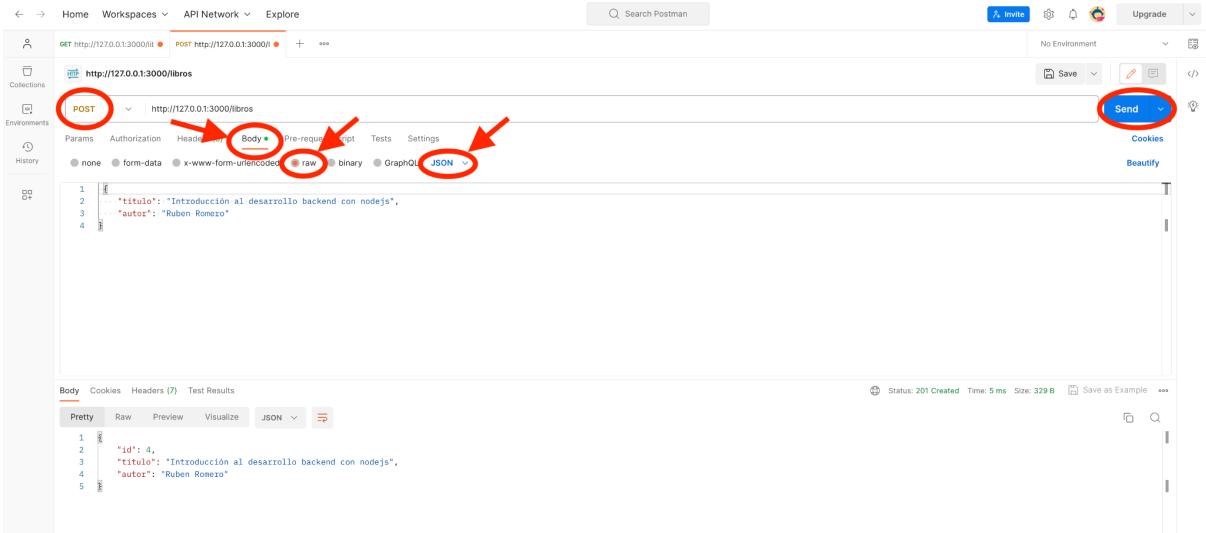
24. Abrimos la aplicación Postman y escribimos la dirección "**http://127.0.0.1:3000/libros**" en la barra de Url, hacemos click en **Send**. Deberíamos obtener el siguiente resultado:



25. Ahora vamos a probar dar de alta un nuevo libro, para eso vamos a:
    a. Utilizar el método HTTP POST
    b. En la sección **Body** vamos a seleccionar la opción **raw**, luego la opción **JSON.**
    c. Vamos a copiar y pegar el JSON en el body :

```
{
  "titulo": "Introducción al desarrollo backend con nodejs",
  "autor": "Ruben Romero"
}
```

    d. y finalmente vamos a enviar la petición con el **botón Send.**



Como vemos el resultado esperado en la respuesta es:

```
{
    "id": 4,
    "titulo": "Introducción al desarrollo backend con nodejs",
    "autor": "Ruben Romero"
}
```

Autenticación API

26. Seguimos instalando la librería **express-oauth2-jwt-bearer**, para esto en el Terminal vamos a escribir el siguiente comando y luego enter:

```
npm install express-oauth2-jwt-bearer
```

27. Creamos la carpeta **middleware**
28. Luego creamos el archivo **errorHandler.js** dentro de la carpeta **middleware**, para que las respuestas por error de Express sean en formato JSON:

```
 // middleware/errorHandler.js
const errorHandler = (err, req, res, next) => {
    // Verificar si el error tiene un código de estado definido, de lo contrario, establecer el código de estado predeterminado
    const statusCode = err.statusCode || 500;

    // Construir objeto de respuesta de error
    const errorResponse = {
      error: {
        message: err.message || "Error interno del servidor",
        code: err.code || "internal_error",
      },
    };

    // Enviar respuesta de error en formato JSON
    res.status(statusCode).json(errorResponse);
  };

  module.exports = errorHandler;
```

29. En app.js hacemos el require del errorHandler:

```
const errorHandler = require("./middleware/errorHandler")
```

30. Agregamos en app.js, el errorHandler como middleware luego de definir todas las rutas:

```
app.use(errorHandler);
```

31. En app.js:
    a. Hacemos require de express-oauth2-jwt-bearer.
    b. Agregamos la configuración del Servidor de Autorización.
    c. Asignamos el middleware la ruta “/libros”.

Quedando el archivo app.js con el siguiente código:

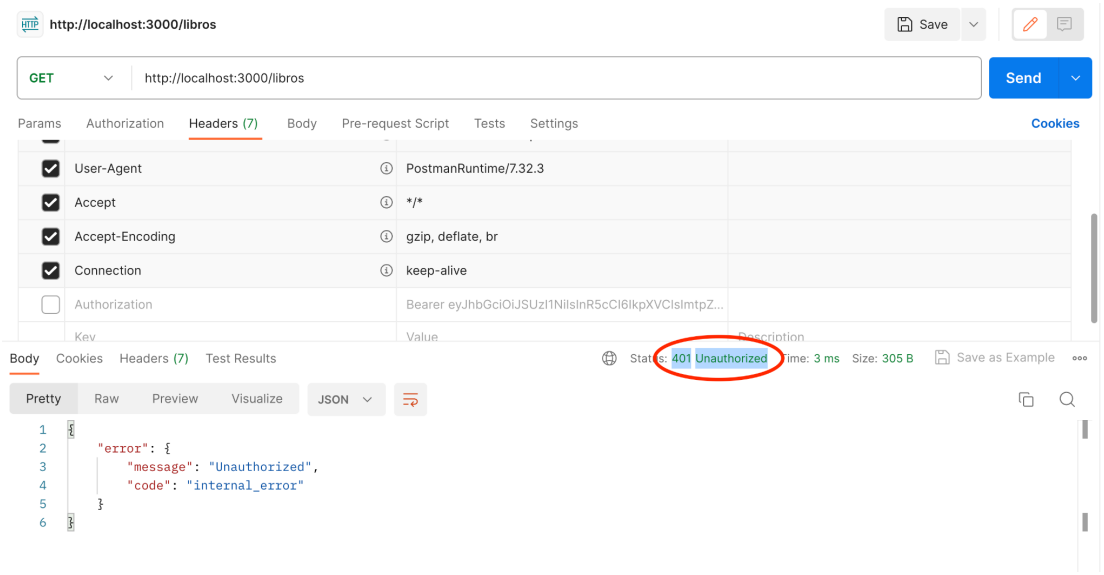
```
const express = require("express");

const { auth } = require("express-oauth2-jwt-bearer");
const errorHandler = require("./middleware/errorHandler");

// Configuracion Middleware con el Servidor de Autorización
const autenticacion = auth({
  audience: "http://localhost:3000/api/productos",
  issuerBaseURL: "https://dev-utn-frc-iaew.auth0.com/",
  tokenSigningAlg: "RS256",
});

const app = express();
app.use(express.json());

// Importamos el Router de Libros
const librosRouter = require("./routes/libros");

//Configuramos el middleware de autenticacion
app.use("/libros", autenticacion,  librosRouter);

app.use(errorHandler);

app.listen(3000, () => {
```

```
  console.log("Servidor iniciado en el puerto 3000");
});
```

32. Iniciamos nuevamente el servidor de NodeJs:

```
node app.js
```

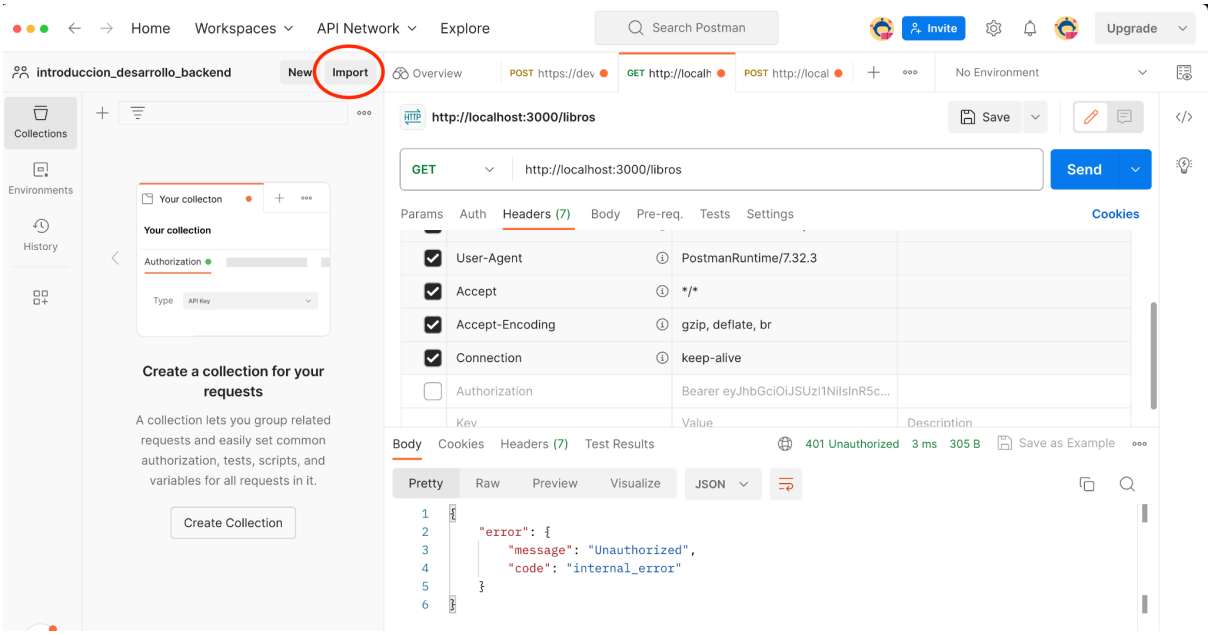
33. Verificamos si funciona la Autorización. Como no ingresamos un TOKEN deberíamos esperar una respuesta HTTP con Status 401 Unauthorized:

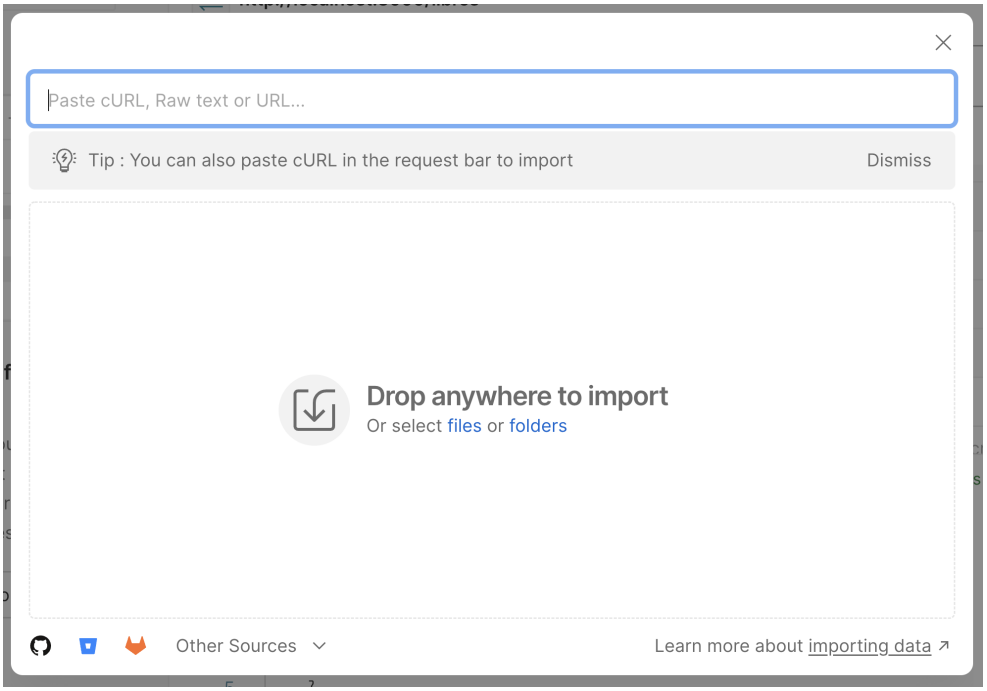


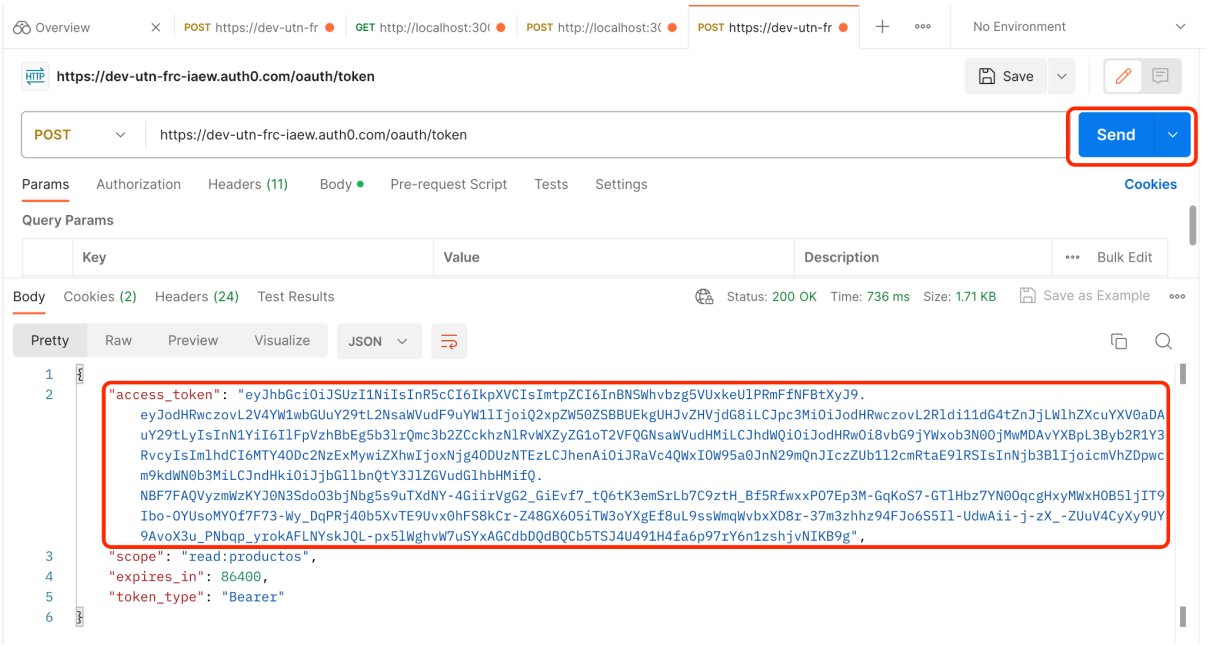
34. Solicitamos un Token al Servidor de Autorización.
    a. Copiamos lo siguiente:

```
curl --location 'https://dev-utn-frc-iaew.auth0.com/oauth/token' \
--header 'content-type: application/json' \
--header 'Cookie:
did=s%3Av0%3A76204e80-0ef1-11ee-8e12-b99ba3014b47.fKR174NkC7L0s9Z8riCysG37mMeRVakoS%2BTt
ye2lC3g;
did_compat=s%3Av0%3A76204e80-0ef1-11ee-8e12-b99ba3014b47.fKR174NkC7L0s9Z8riCysG37mMeRVak
oS%2BTtye2lC3g' \
--data '{
"client_id": "QiW8AlH9oykBg7ofBrHs6ToYvrdmhOeE",
"client_secret": "7kZPQqNnhRAuCXipSVSdHUsV9MzgfUzBB3AYbfemGPqxtpXI6j1GNxDBfYBSUume",
"audience": "http://localhost:3000/api/productos",
"grant_type": "client_credentials"
}'
```

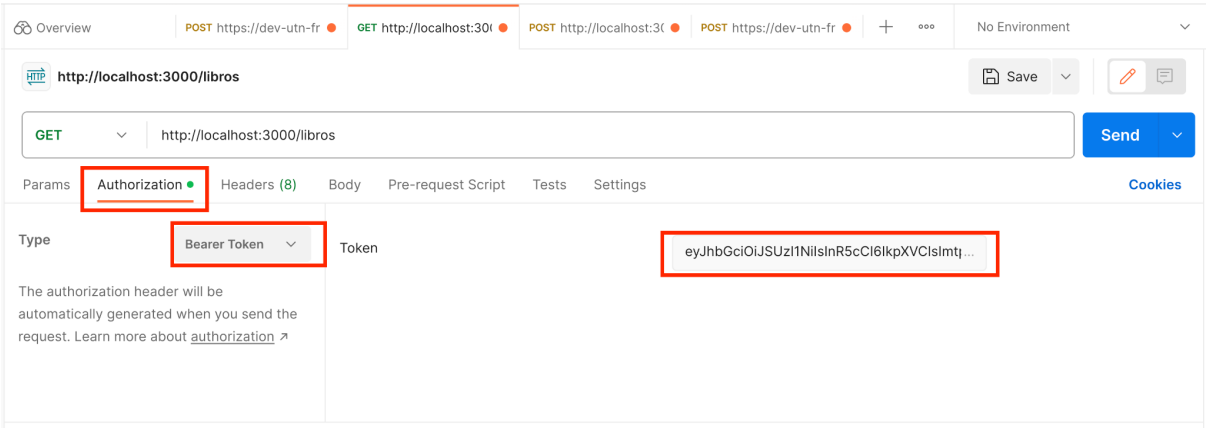
b. Vamos a Postman e importamos lo que copiamos anteriormente. Esto nos va a generar un nuevo request en Postman para solicitar el token al Servidor de Autorización:



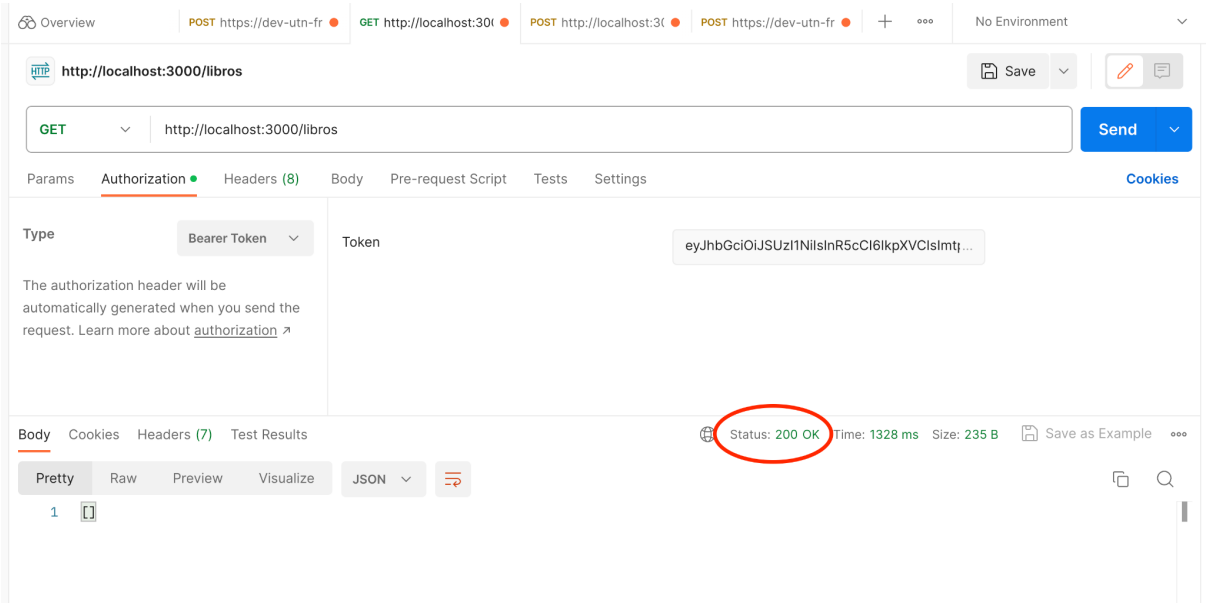




35. Copiamos el valor de access_token, y vamos al request de GET /libros buscamos la opción Authorization, elegimos el tipo Bearer Token y pegamos el access_token donde dice TOKEN:



36. Hacemos click en Send, y la respuesta esperada es 200 OK. AHORA SI ESTAMOS AUTENTICADOS!! YUPI!!!:



Autorización API

37. En el archivo routes/libros.js agregamos configuración de scopes quedando de la siguiente forma:

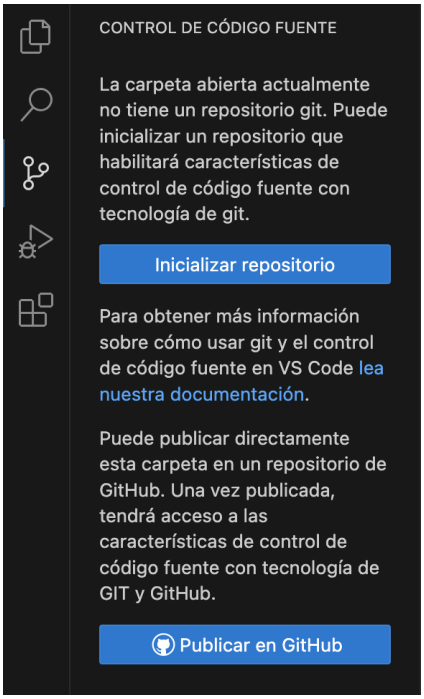
```
const express = require("express");
const router = express.Router();

const Libro = require("../models/Libro");

// Importamos la librería para validar scopes
const { requiredScopes } = require("express-oauth2-jwt-bearer");

// Ruta para obtener todos los libros
router.get("/", requiredScopes("read:libros"), async (req, res) => {
  try {
    const libros = await Libro.find();
    res.json(libros);
  } catch (error) {
    res.status(500).json({ error: "Error al obtener los libros" });
  }
});
```

```
// Ruta para crear un nuevo Libro
router.post("/", requiredScopes("write:libros"), async (req, res) => {
  try {
    const nuevoLibro = new Libro(req.body);
    await nuevoLibro.save();
    res.json(nuevoLibro);
  } catch (error) {
    res.status(500).json({ error: "Error al crear el Libro" });
  }
});

// Ruta para actualizar un Libro existente
router.put("/:id", requiredScopes("write:libros"), async (req, res) => {
  try {
    const Libro = await Libro.findByIdAndUpdate(req.params.id, req.body,
{
      new: true,
    });
    res.json(Libro);
  } catch (error) {
    res.status(500).json({ error: "Error al actualizar el Libro" });
  }
});

// Ruta para eliminar un Libro
router.delete("/:id", requiredScopes("write:libros"), async (req, res)
=> {
  try {
    await Libro.findByIdAndDelete(req.params.id);
    res.json({ message: "Libro eliminado correctamente" });
  } catch (error) {
    res.status(500).json({ error: "Error al eliminar el Libro" });
  }
});

module.exports = router;
```

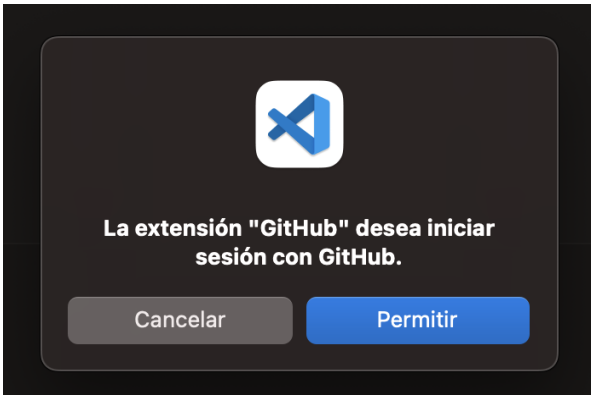
38. Probamos desde Postman nuevamente con GET /libros y deberíamos recibir la respuesta 200 OK.
39. LUEGO DE ESTO PODEMOS DECIR QUE NUESTRA API ES SEGURA!!!

Subir a GitHub

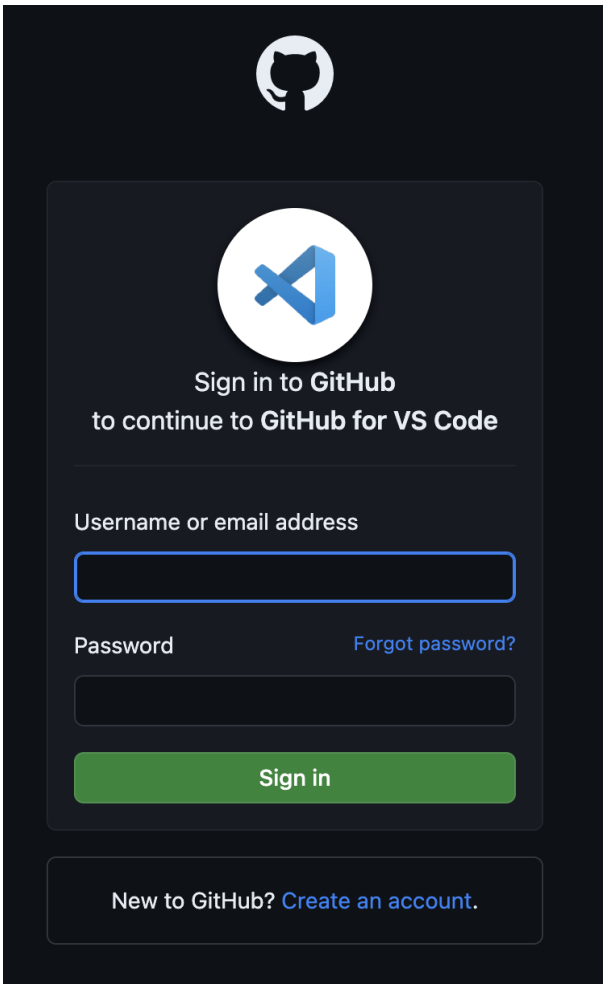
40. Ahora volvemos a VSCode y vamos a llevar a GitHub el código fuente generado usando la herramienta de git de VSCode. Hacemos click en la opción “Publicar en GitHub”:



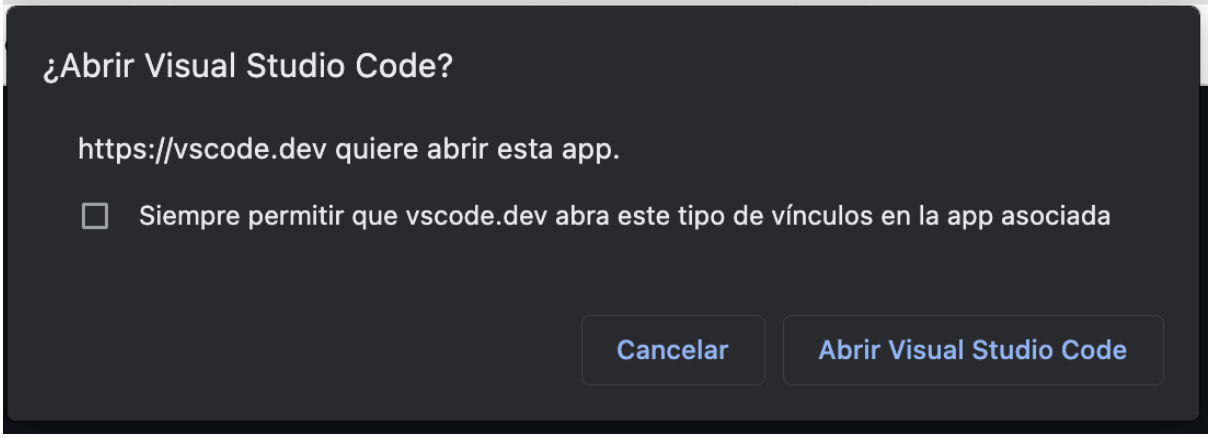
41. VSCode muestra este mensaje para que le darle permisos a iniciar sesión en GitHub a través de un navegador:



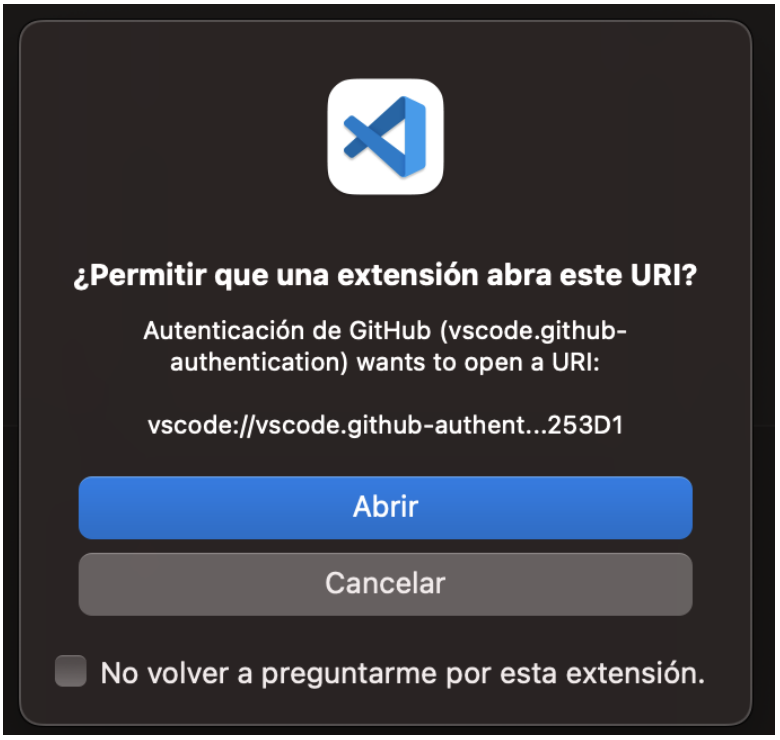
42. VSCode nos lleva al navegador web que tengamos por defecto y nos pide que ingresemos los datos de nuestra cuenta GitHub (el que no tenga cuenta se crea una nueva con su correo electrónico):



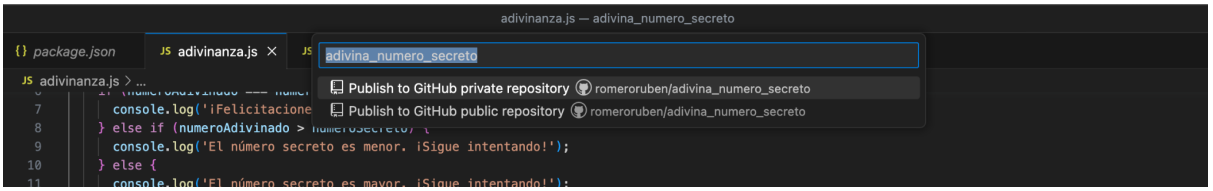
43. El navegador solicita que permitamos abrir VSCode, seleccionamos la opción “Abrir Visual Studio Code”



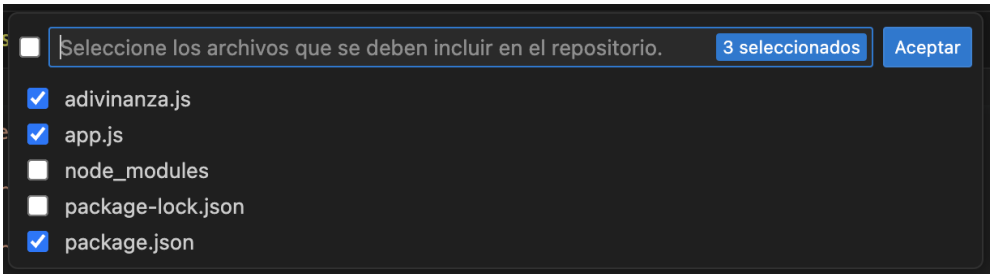
44. VSCode nos pide permisos para abrir una url, hacemos click en la opción “Abrir”:



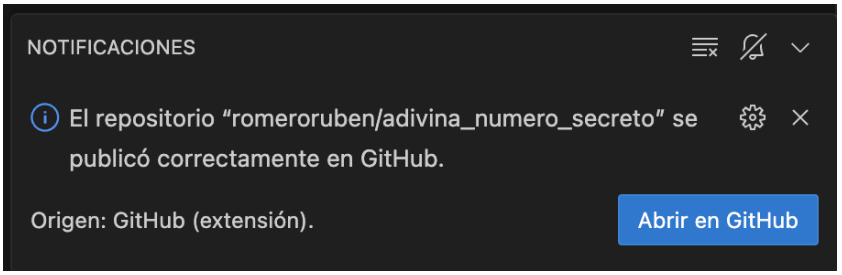
45. Ahora debemos seleccionar que tipo de repositorio queremos crear, seleccionar “Publish to GitHub **public** repository”, esto significa que vamos a publicar nuestro código en forma pública.



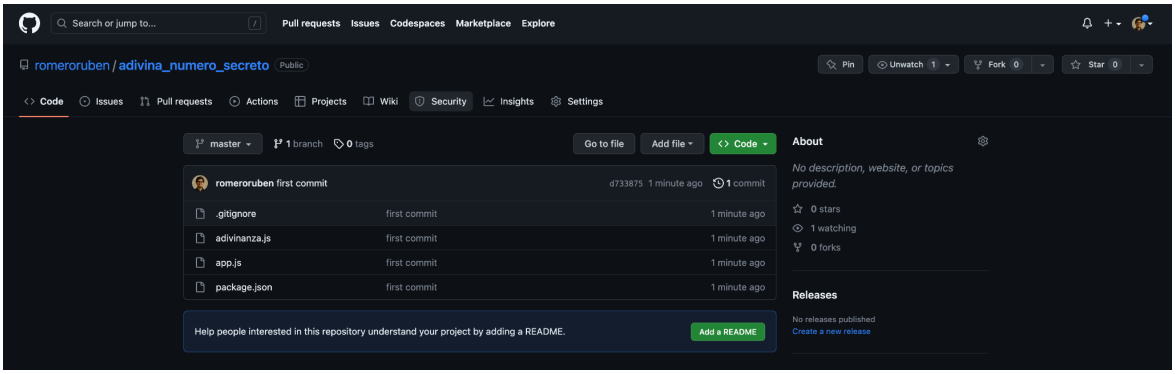
46. A continuación seleccionamos los archivos y carpetas que queremos “versionar” y llevar a GitHub.



47. Esperamos unos minutos hasta que en VSCode podemos ver la siguiente notificación (abajo a la derecha), hacemos click en “Abrir en GitHub”
:



48. Se abre un navegador con el repositorio de GitHub disponible para que cualquiera lo pueda acceder:



49. IMPORTANTE! Para dar por finalizada la actividad compartir la url de GitHub en la actividad del aula virtual en https://uve.frc.utn.edu.ar/.