# Documentación proyecto 1ª –NODE-js

## 1º El servidor debe utilizar Express.js

**// Crear el servidor**

**const express = require('express');**

**const app = express();**

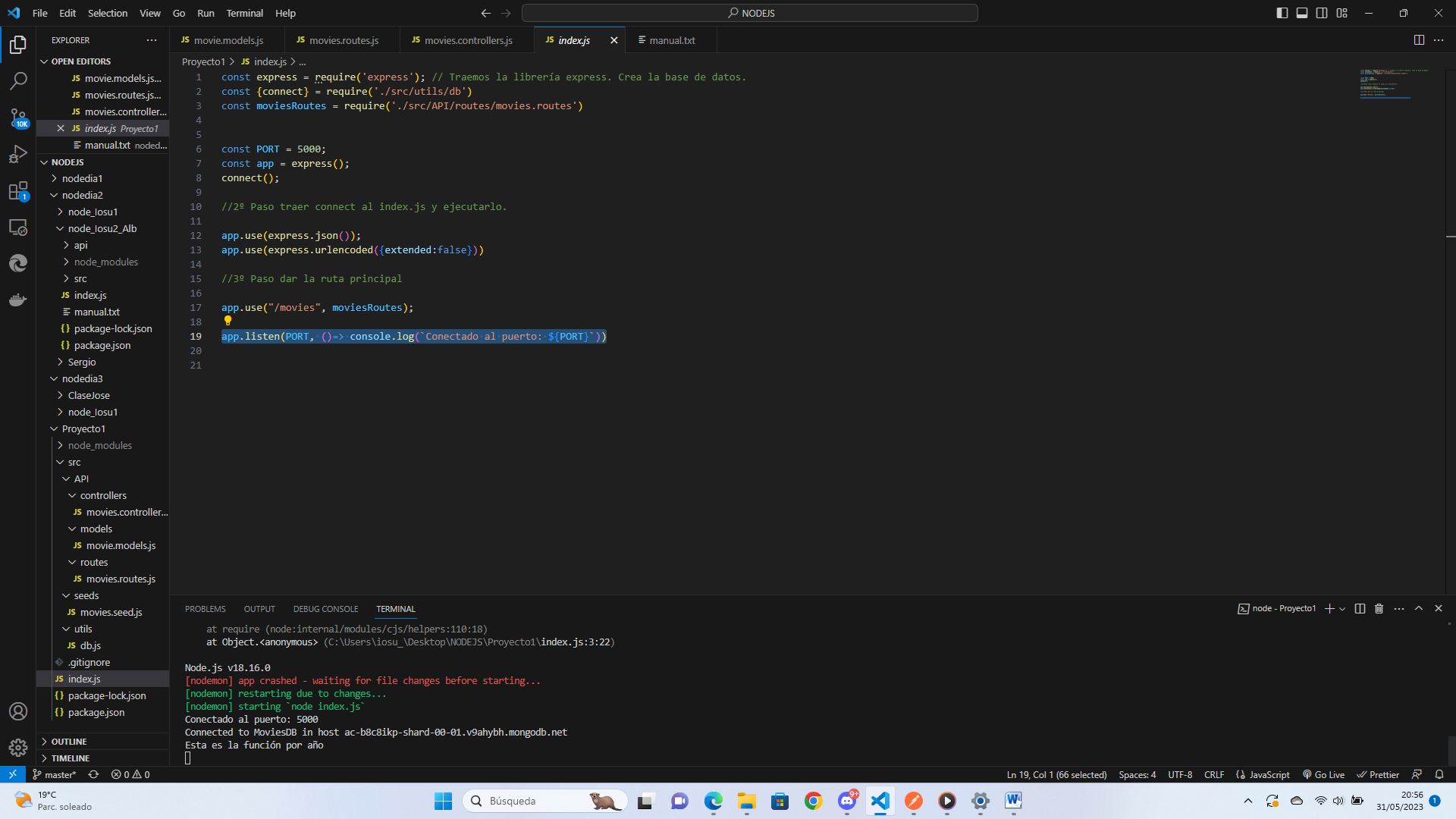
**const PORT = 5000;**

**// Iniciar el servidor**

**app.listen(PORT, ()=> console.log(`Conectado al puerto: ${PORT}`))});**

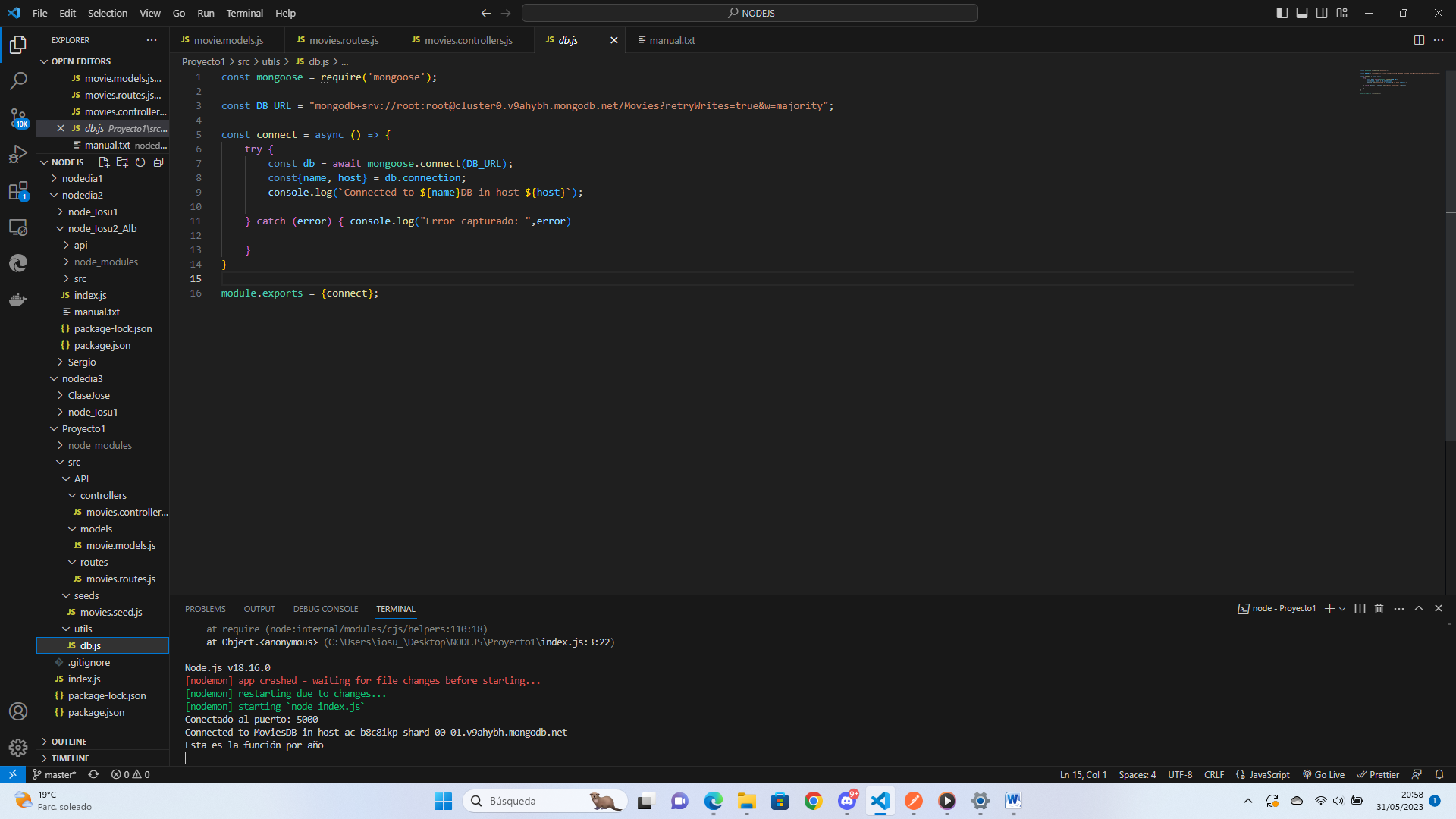
1º Se requiere y se activa el módulo express.

2º Iniciamos express y le damos un puerto que servirá para iniciar el servidor. La línea final nos permite verificar por consola que se está usando el puerto X para el servidor.



En la imagen vemos con el nodemon activo la respuesta programada en la última línea del cuadro rojo anterior.

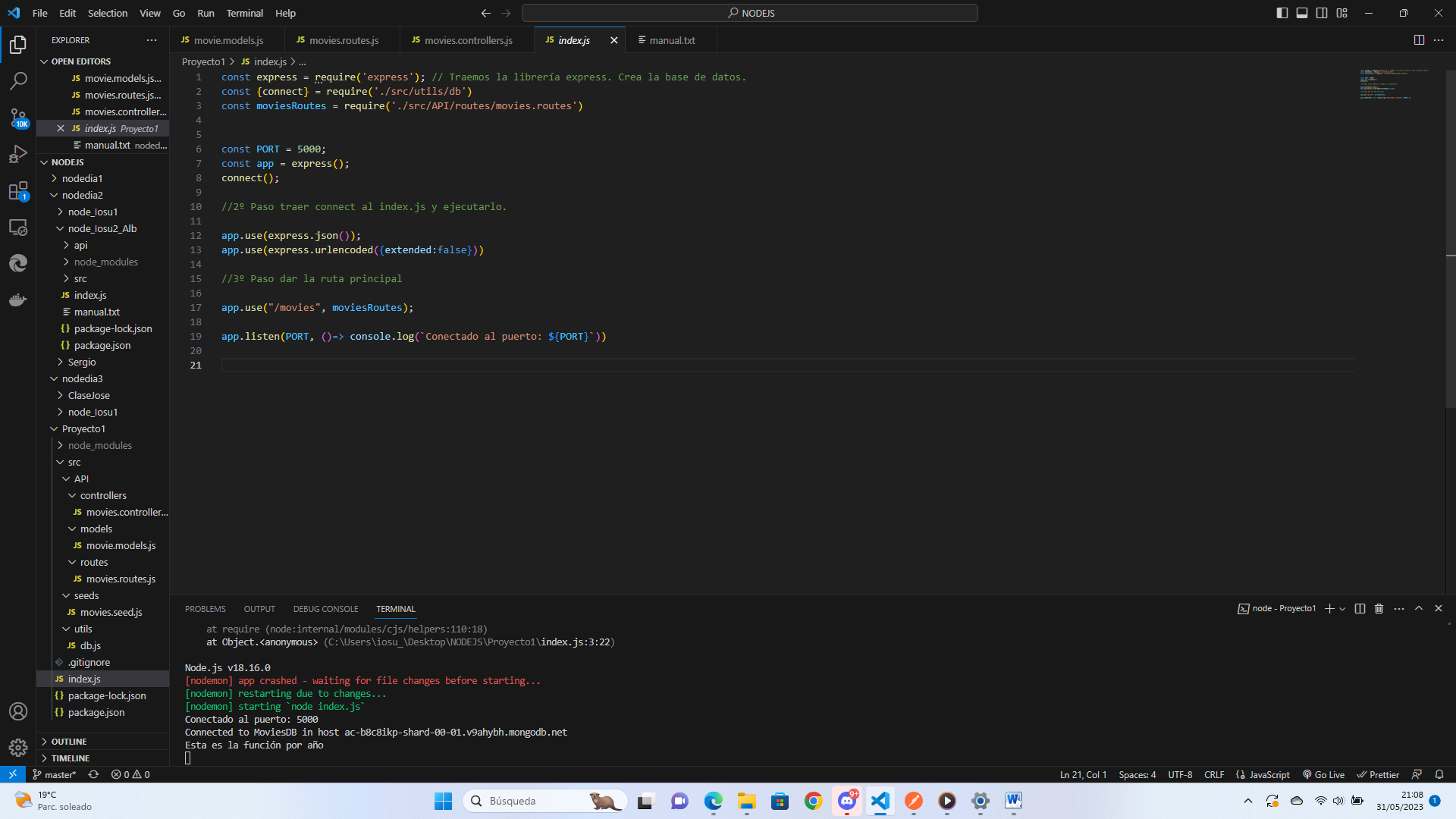
## 2º D**ebe conectarse a una base de datos MongoDB, en local o mediante MongoAtlas, utilizando mongoose.**



*Para usar Mongoose se hace igual que con la librería Express*. Requerirá del uso de “require”. Después se coge la dirección que cogemos de la sección drivers de Mongo Atlas (se crea con ella la variable DB\_URL). En este caso el user y la contraseña que se han utilizado han sido: “root:root”.

La función connect es una función asíncrona que intentará conectarse a la url que aportamos (DB\_URL). Para ello utilizará la librería mongoose. De la conexión recuperamos con destructurin el nombre y el host y generamos un segundo console.log() que nos permite comprobar que la conexión con Atlas se ha realizado correctamente. En la imagen superior aporto la función tras su escritura.

**OJO: La función se debe exportar para ser usada en el index.js.**

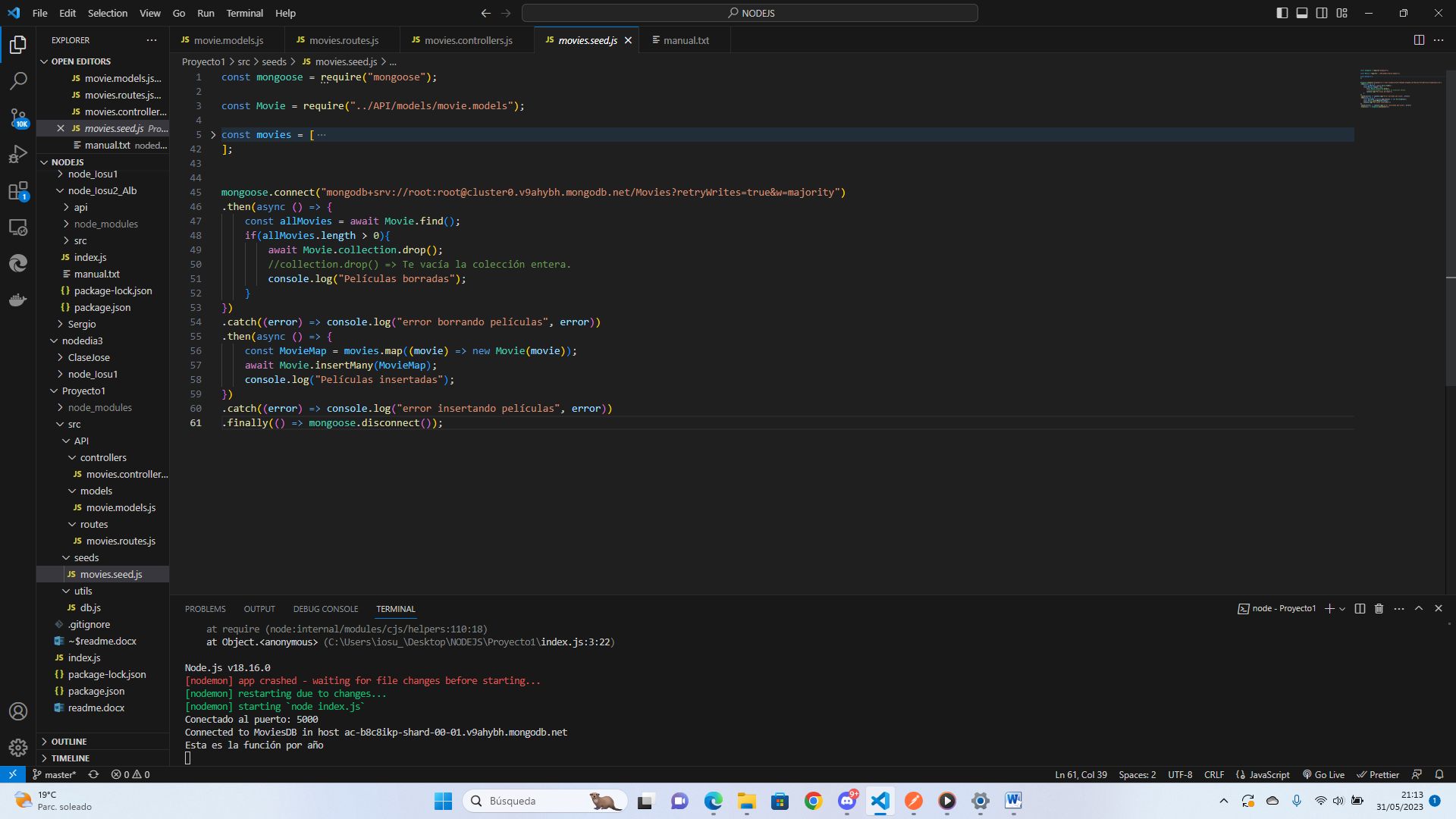


En la imagen se muestra ya el aspecto del index.js terminado. En este punto no debería aparecer el enrutador. Las líneas que nos interesan son:

const {connect} = require('./src/utils/db')

connect();

## 3º. Crear una semilla con el nombre \*\*`movies.seed.js`\*\* que cargue en la base de datos las siguientes películas.



1º Gestionar la conexión con mongoose. TODAS LAS CONEXIONES SE VAN A GESTIONAR CON ESTA LIBRERÍA.

2ºImportar el modelo o esquema con el que van a entrar las películas dentro del sistema.

3º Array de seeds que hace las veces de datos iniciales.

4º Mongoose gestiona la conexión hacia la base de datos. OJO: Poner la misma URL que se ha puesto a la hora de gestionar la función connect() (no está de más verificar por si en user o en contraseña hay algún error taquigráfico.

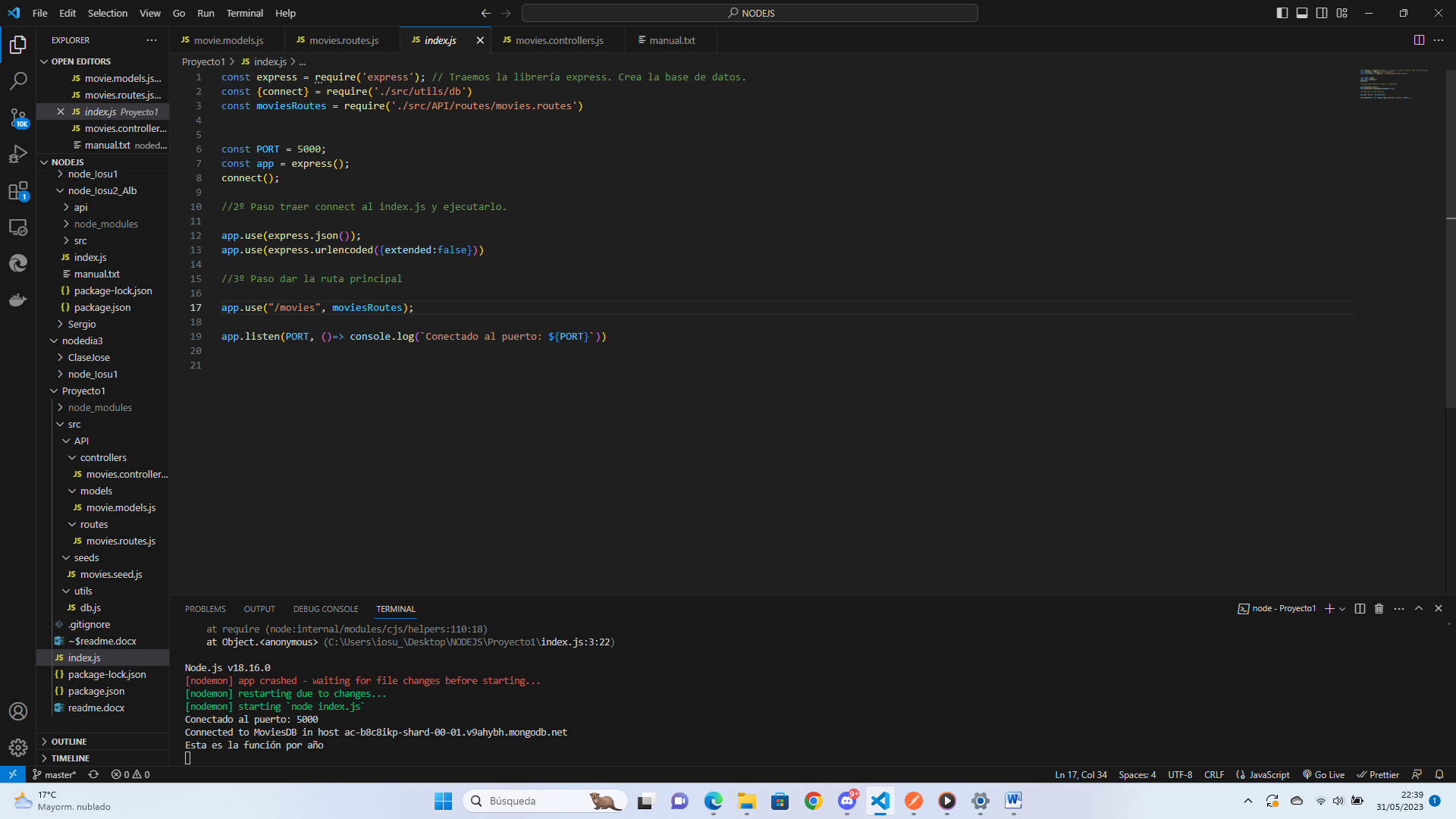
5º Generar dos funciones asíncronas:

1. En la primera se comprueba la longitud del array de las películas que se muestran. Si la longitud del array es mayor a 0 (si hay películas dentro), realiza el método drop y vacía el array.
2. En la segunda se cogen las películas y se pasan por el schema importado para meterlas dentro del array con el formato adecuado.

**Al final de cada función se pone un console.log para comprobar el funcionamiento de ambas funciones.**

## 4º Crear un endpoint get que devuelva todas las películas.

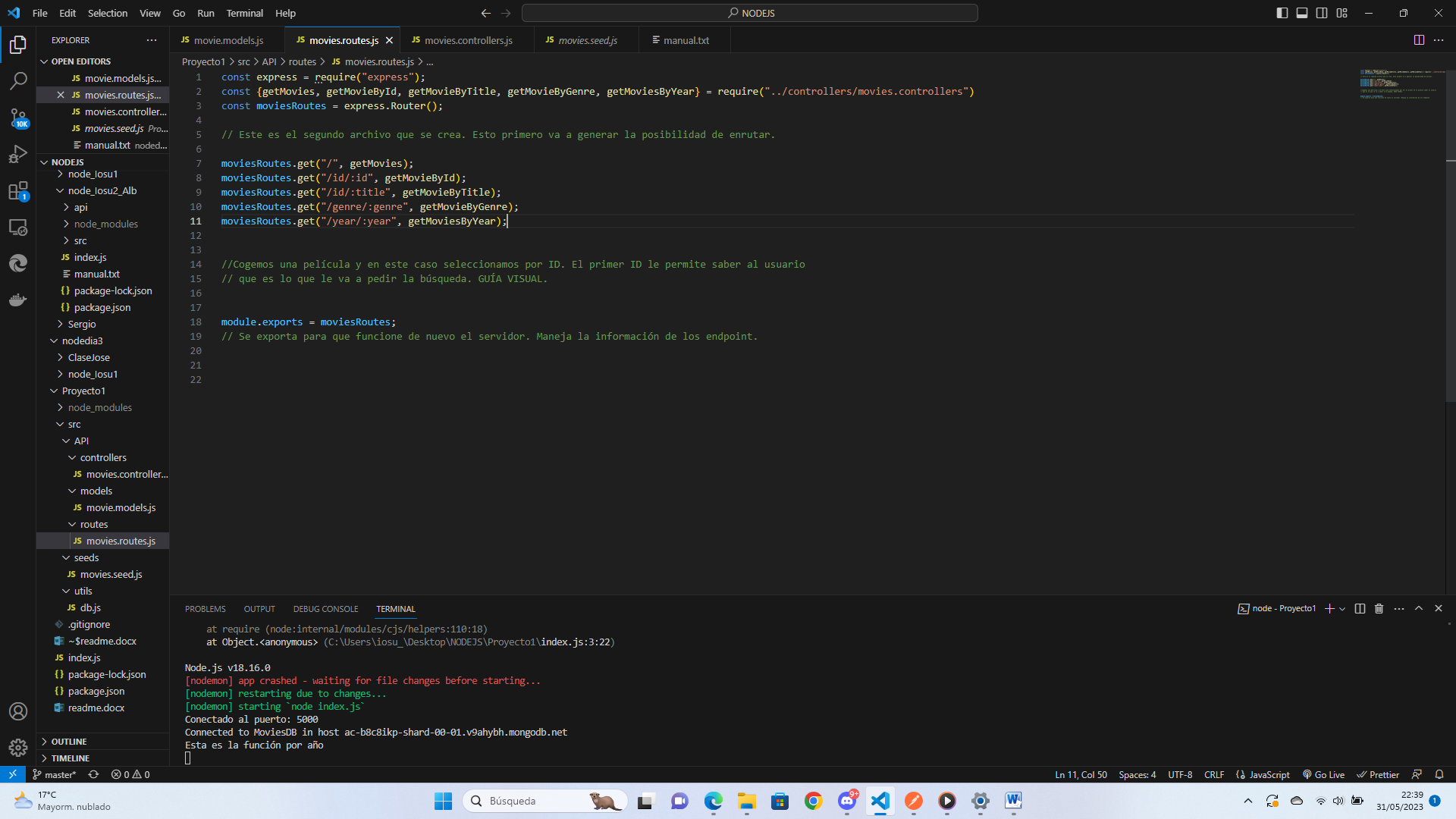
***Antes demostrar el endpoint get es interesante ver cómo funcionan.***



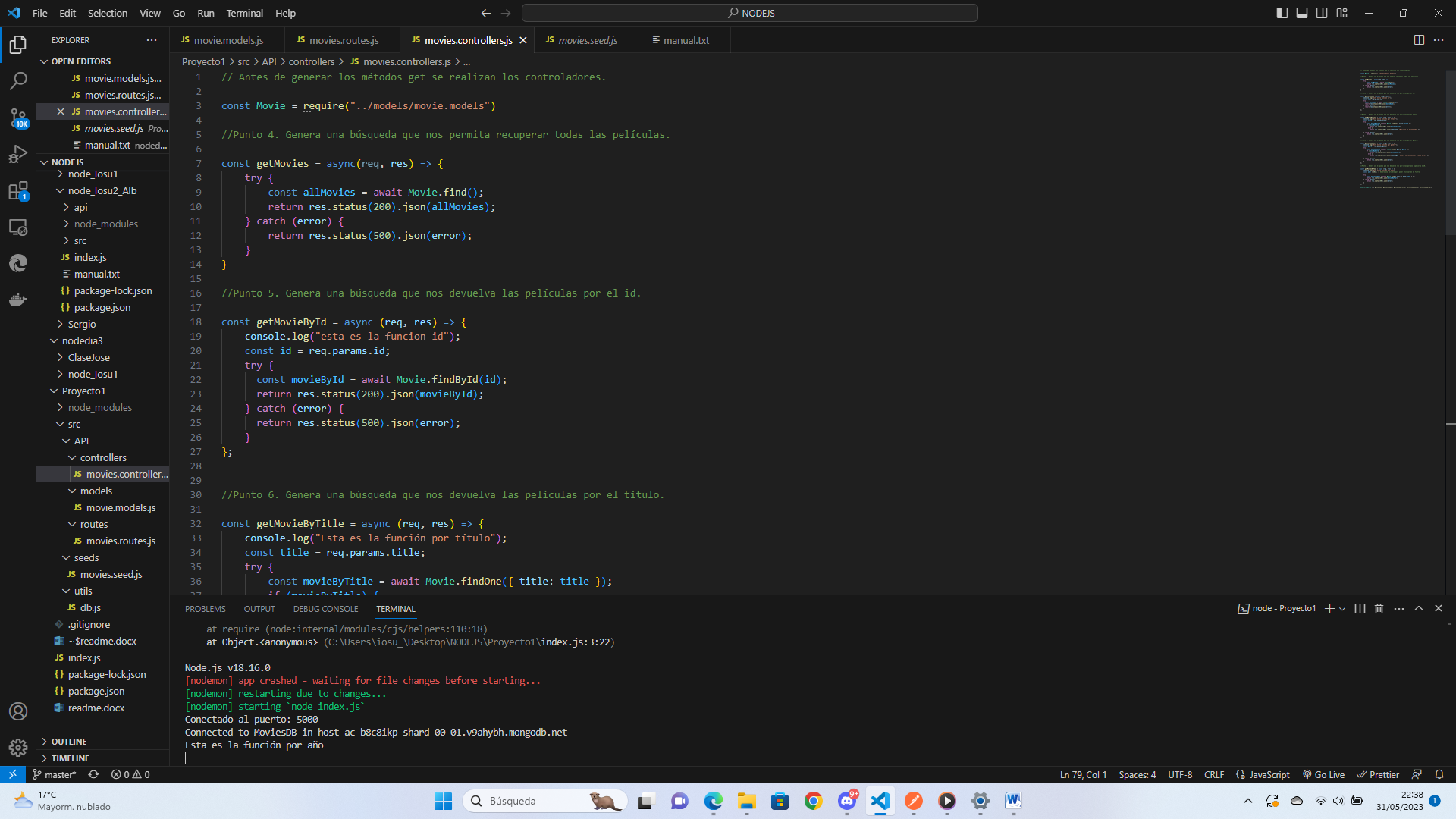
**1º** Usar app.use para que se genere la posibilidad de usar las rutas. La dirección que quedará será:

**Localhost + PUERTO + / + Nombre deseado.**

**2º** Importar las rutas desde el archivo oportuno. En el proyecto se localiza dentro de API, en la carpeta routes y por último dentro del archivo movies.routes.js.



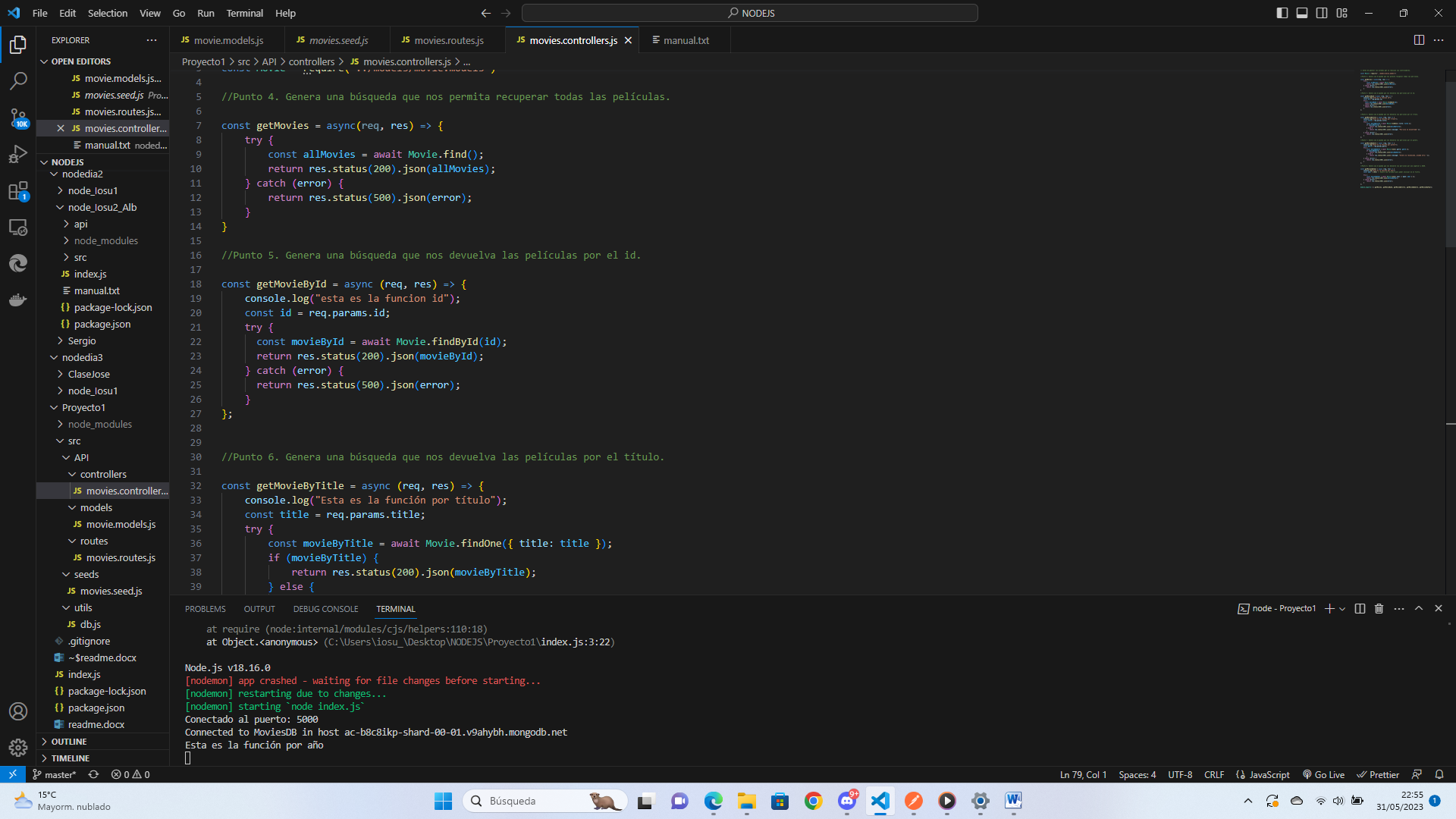
**3º Archivo de rutas completo.** En cada una de estas rutas se encuentra asociada una función que se exporta con module.exports para ser usadas por el index.js. Estas fórmulas han sido previamente creadas en el archivo llamado movies.controllers.



**4º Finalizando la explicación.** Vemos en el archivo controllers de movies una función llamada getMovies. En este caso y en el resto también, tendremos funciones asíncronas. Esta función asíncrona realiza una llamada a la base de datos y utiliza el esquema que se ha importado y se representa con “Movie” para buscar los elementos que coinciden con él.

La segunda línea de la función en la parte del try nos devuelve el status OK y un JSON de todas las películas. Este JSON puede ser visualizado por el usuario si usa el POSTMAN o el navegador correspondiente.

## 5º . Crear un endpoint \*\*get\*\* que devuelva una película según su \*\*\_id\*\*



*PUNTOS DE INTERÉS DE LA FUNCIÓN*

* **Lectura de la declaración: const getMovieById** **= async (req, res) =>** {: Aquí se declara una función llamada `getMovieById` que será usada como controlador de una ruta específica. *Toma dos argumentos, `req` y `res`, que representan la solicitud y la respuesta HTTP, respectivamente*.
* **Control de la ejecución de la orden:** **console.log("esta es la funcion id"):** Esta línea imprime un mensaje en la consola cuando se ejecuta la función `getMovieById`. *Puede ser útil para depurar y verificar el flujo de ejecución*.
* **Requiriendo el parámetro deseado: const id = req.params.id:** Aquí se extrae el parámetro "id" de la URL de la solicitud utilizando `req.params.id`. Esta parte asume que estás utilizando una ruta que tiene un segmento de URL variable llamado "id".
* **Buscando la película en cuestión: const movieById = await Movie.findById(id):** Aquí se realiza una búsqueda en la base de datos utilizando el método `findById` proporcionado por el modelo `Movie`. El `await` se utiliza para esperar a que se complete la consulta antes de continuar con la siguiente línea de código. Se asume que tienes un modelo llamado `Movie` definido previamente.
* **Devolver el resultado en un JSON: return res.status(200).json(movieById):** Si se encuentra la película correspondiente al ID proporcionado, se devuelve una respuesta HTTP con un estado de 200 (éxito) y los datos de la película en formato JSON.

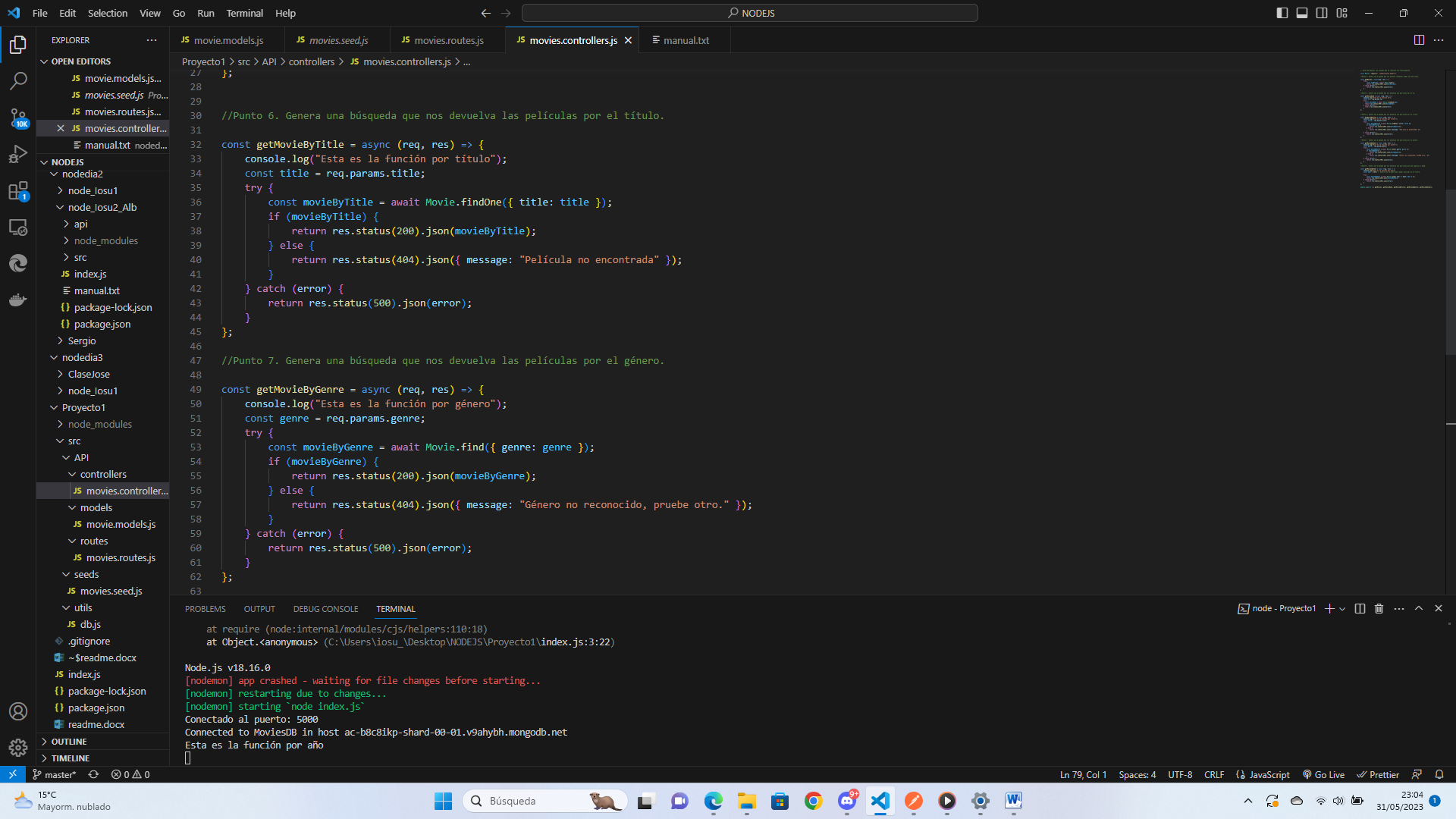
**OJO: NO OLVIDAR EL MODULE.EXPORTS AL FINAL DEL DOCUMENTO CONTROLLERS PARA PODER LLEVARSE DEL DOCUMENTO TODAS LAS FUNCIONES PREPARADAS.**

**module.exports = {getMovies, getMovieById};**

* **`**

## 6º. Crear un endpoint \*\*get\*\* que devuelva un valor por su titulo y otro por su género

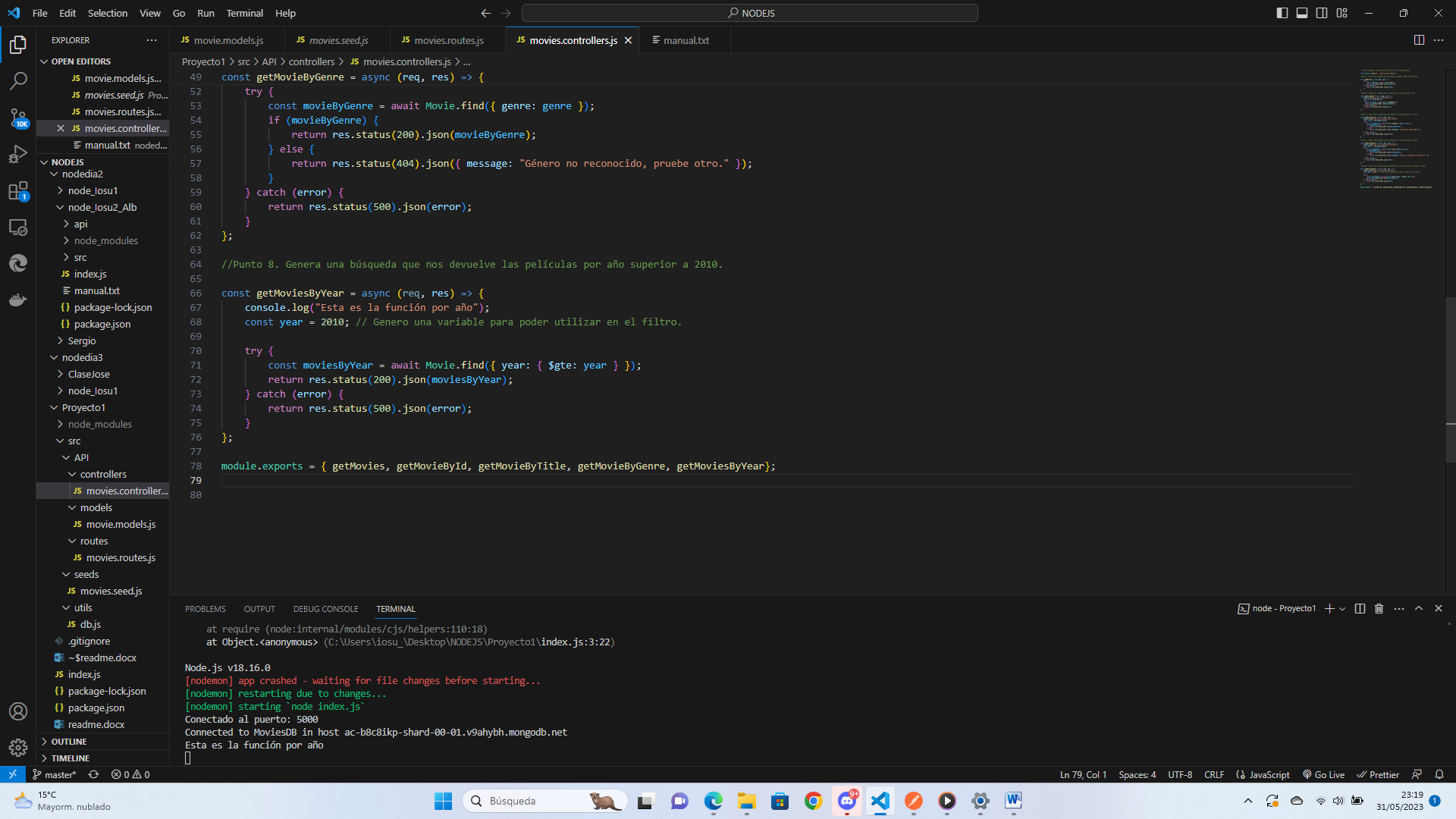
**CONTROLADOR GET PARA BUSCAR POR TÍTULO**



Debido a la similitud de ambos puntos se explican en un solo epígrafe su planteamiento.

* **const getMovieByTitle = async (req, res) => { // const: getMovieByGenre = async (req, res) =>{** Se declaran unas funciones que serán usadas como controladores de unas rutas específicas. En ambos casos tomamos dos argumentos, req y res.
* **console.log("Esta es la función por título // género"):** Esta línea imprime un mensaje en la consola cuando se ejecuta la función getMovieByTitle o la función getMovieByGender.
* **const title = req.params.title // const genre = req.params.genre**: Aquí se extrae el parámetro "title" de la URL de la solicitud utilizando `req.params.title`. Asume que estas utilizando title o genre en una parte del URL.
* **const movieByTitle = await Movie.findOne({ title: title }) // const movieByGenre= await Movie.find({ genre: genre }):** Aquí se realiza una búsqueda en la base de datos utilizando el método `findOne` proporcionado por el modelo `Movie`. Se busca una película cuyo título coincida con el título proporcionado en la solicitud. En el caso de “genre” se usa find() porque se asume que puede haber una película de un género o varias. Por eso find() es el método elegido.
* **if (movieByTitle) { ... } else { ... } // if (movieByGenre) { ... } else { ... }:** Si se encuentra una película con el título proporcionado, se devuelve una respuesta HTTP con un estado de 200 (éxito) y los datos de la película en formato JSON. De lo contrario, si no se encuentra ninguna película, se devuelve una respuesta con un estado de 404 (no encontrado) y un mensaje indicando que la película no fue encontrada.
* **return res.status(500).json(error):** Si se produce un error durante la búsqueda de la película, se devuelve una respuesta HTTP con un estado de 500 (error interno del servidor) y los detalles del error en formato JSON.

## 7º. Crear un endpoint \*\*get\*\* que devuelva las películas que se han estrenado a partir de 2010.



La función getMoviesByYear es exactamente igual a las anteriores. Sólo cambian dos cosas:

* Se declara una variable const year que se utiliza como base para alimentar el find a posteriori.
* Dentro del find() usamos el operador $gte que nos permite buscar que sean mayores o iguales que 2010. Es decir, películas estrenadas a partir de 2010.