# Node + Express+jwt+postgres

Texto

Descripción generada automáticamente

Dar npm init -y esto agrega un package json

Texto

Descripción generada automáticamente

Agregar type module

Captura de pantalla de un celular

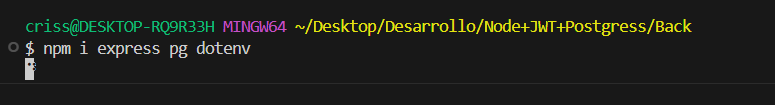
Descripción generada automáticamente

Agregar nodemon y start

Captura de pantalla de un celular

Descripción generada automáticamente

# Instalaciones express variables de entorno



Y nodemon

Texto

Descripción generada automáticamente

Agregar configuraciones en el index.js

// Importar los módulos necesarios

import 'dotenv/config'

import express from 'express';

const app = express();

// Definir una ruta básica

app.get('/', (req, res) => {

  res.send('Hola Mundo');

});

// Definir el puerto en el que el servidor va a escuchar

const port = process.env.PORT || 3000;

// Iniciar el servidor y escuchar en el puerto definido

app.listen(port, () => {

  console.log(`Servidor corriendo en el puerto ${port}`);

});

Npm run dev

Texto

Descripción generada automáticamente

# Crear carpetas para entorno de trabajo

Captura de pantalla de computadora

Descripción generada automáticamente

# Configurar la BD

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

Abrir el pgadmin

Crear db llamada jwt\_db

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ahora en query tool

Captura de pantalla de computadora

Descripción generada automáticamente

Agregar script

drop table if exists USERS;

-- Eliminar la tabla 'users' si existe

DROP TABLE IF EXISTS users;

-- Crear la tabla 'users'

CREATE TABLE users (

UID SERIAL PRIMARY KEY,

EMAIL VARCHAR(50) NOT NULL UNIQUE,

PASSWORD VARCHAR(60) NOT NULL,

USERNAME VARCHAR(50) NOT NULL

);

-- Insertar un registro en la tabla 'users'

INSERT INTO users (EMAIL, PASSWORD, USERNAME) VALUES

('test@test.com', '1234', 'test');

-- Seleccionar todos los registros de la tabla 'users'

SELECT \* FROM users;

# Conexión a postgress

En la pagina de postgres

Texto

Descripción generada automáticamente

Código connection.databse.js

import 'dotenv/config';

import pg from 'pg';

const { Pool } = pg;

const connectionString = process.env.DATABASE\_URL

export const db = new Pool({

    allowExitOnIdle: true,

    connectionString

})

    try {

        await db.query('SELECT NOW()');

        console.log('DATABASE CONNECTED');

    } catch (error) {

        console.error(error);

    }

Código env

DATABASE\_URL="postgresql://postgres:1234@localhost:5432/jwt\_db"

Prueba

node database/connection.database.js

Texto

Descripción generada automáticamente

# Creando el modelo

Se agrega user.models.js

Captura de pantalla de computadora

Descripción generada automáticamente

El codigo

import { db } from '../database/connection.database.js';

const create = async (email, password, username) => {

    const query = {

        text: `INSERT INTO users (email, password, username)

               VALUES ($1, $2, $3)

               RETURNING email, username, uid`,

        values: [email, password, username]

    };

    const { rows } = await db.query(query);

    return rows[0]; // Retorna la primera fila que contiene los datos insertados

};

const findOneByEmail = async (email) => {

    const query = {

        text: `SELECT \*

               FROM users

               WHERE email = $1`,

        values: [email]

    };

    const { rows } = await db.query(query);

    return rows[0]; // Retorna la primera fila que contiene el usuario encontrado

};

export const UserModel = {

    create,

    findOneByEmail

};

# Creando el contralor

Texto

Descripción generada automáticamente

import { UserModel } from "../models/user.model.js";

const register = async(req, res) => {

    try {

        console.log(req.body)

        const { username, email, password } = req.body

        return res.json({ ok: true, msg: "user ok" })

    } catch (error) {

        console.log(error)

        return res.status(500).json({

            ok: false,

            msg: 'Error Server'

        })

    }

}

const login = async(req, res) => {

    try {

    } catch (error) {

        console.log(error)

        return res.status(500).json({

            ok: false,

            msg: 'Error Server'

        })

    }

}

export const UserController =

 { register, login };

# Creando las RUTAS

Texto

Descripción generada automáticamente

import { Router} from "express";

import { UserController } from "../controllers/user.controller.js"

const router = Router()

router.post('/register', UserController.register);

router.post('/login', UserController.login);

export default router;

# Probando los cambios

Modificar el index.js

// Importar los módulos necesarios

import 'dotenv/config'

import express from 'express';

import userRouter from './routes/user.route.js'

const app = express();

app.use('/api/v1/users',userRouter)

// Definir una ruta básica

// app.get('/', (req, res) => {

//   res.send('Hola Mundo');

// });

// Definir el puerto en el que el servidor va a escuchar

const port = process.env.PORT || 3000;

// Iniciar el servidor y escuchar en el puerto definido

app.listen(port, () => {

  console.log(`Servidor corriendo en el puerto ${port}`);

});

Levantar

Imagen que contiene Logotipo

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Error

ndefined

TypeError: Cannot destructure property 'username' of 'req.body' as it is undefined.

at register (file:///C:/Users/criss/Desktop/Desarrollo/Node+JWT+Postgress/Back/controllers/user.controller.js:7:17)

at Layer.handle [as handle\_request] (C:\Users\criss\Desktop\Desarrollo\Node+JWT+Postgress\Back\node\_modules\express\lib\router\layer.js:95:5)

at next (C:\Users\criss\Desktop\Desarrollo\Node+JWT+Postgress\Back\node\_modules\express\lib\router\route.js:149:13)

at Route.dispatch (C:\Users\criss\Desktop\Desarrollo\Node+JWT+Postgress\Back\node\_modules\express\lib\router\route.js:119:3)

at Layer.handle [as handle\_request] (C:\Users\criss\Desktop\Desarrollo\Node+JWT+Postgress\Back\node\_modules\express\lib\router\layer.js:95:5)

agregar estas modificaciones

app.use(express.json());

app.use(express.urlencoded({extended:true}));e

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Instalar thunder client

Captura de pantalla de computadora

Descripción generada automáticamente

New request

Captura de pantalla de un celular

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Prueba agregando

{

"username": "test",

"email":"test@test.com",

"password":"1234"

}

Captura de pantalla de computadora

Descripción generada automáticamente

# Ahora en el request realizar todas las validaciones

En el controller agregar lo siguiente

Captura de pantalla de computadora

Descripción generada automáticamente

        // validaciones

        if(!username || !email || !password){

            return res.status(400).json({ok: false, msg: "falla en requerir los campos de user, email, password"})

        }

Realizando pruebas

Interfaz de usuario gráfica

Descripción generada automáticamente

Se agrega nuevo código

import { UserModel } from "../models/user.model.js";

// /api/v1/users/register  ruta para el registro

const register = async(req, res) => {

    try {

        console.log(req.body)

        const { username, email, password } = req.body

        // validaciones

        if(!username || !email || !password){

            return res.status(400).json({ok: false, msg: "falla en requerir los campos de user, email, password"})

        }

        const user = await UserModel.findOneByEmail(email)

        if(user){

            return res.status(409).json({ ok: false, msg: "Email existe"})

        }

        return res.json({ ok: true, msg: "user ok" })

    } catch (error) {

        console.log(error)

        return res.status(500).json({

            ok: false,

            msg: 'Error Server'

        })

    }

}

// /api/v1/users/login   ruta para el login

const login = async(req, res) => {

    try {

    } catch (error) {

        console.log(error)

        return res.status(500).json({

            ok: false,

            msg: 'Error Server'

        })

    }

}

export const UserController =

 { register, login };

# Instalar bcryptjs

npm i bcryptjs

Texto

Descripción generada automáticamente

Pruebas con el código

import bcryptjs from 'bcryptjs';

import { UserModel } from "../models/user.model.js";

// /api/v1/users/register  ruta para el registro

const register = async (req, res) => {

    try {

        console.log(req.body);

        const { username, email, password } = req.body;

        // Validaciones

        if (!username || !email || !password) {

            return res.status(400).json({ ok: false, msg: "Falla en requerir los campos de usuario, email, password" });

        }

        const user = await UserModel.findOneByEmail(email);

        if (user) {

            return res.status(409).json({ ok: false, msg: "El email ya existe" });

        }

        const salt = await bcryptjs.genSalt(10);

        const hashedPassword = await bcryptjs.hash(password, salt);

        // Crear usuario

        const newUser = await UserModel.create(email, hashedPassword, username);

        return res.json({ ok: true, msg: "Usuario registrado con éxito" });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

};

// /api/v1/users/login   ruta para el login

const login = async (req, res) => {

    try {

        // Implementa la lógica de inicio de sesión aquí

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

export const UserController = { register, login };

explicación del código se agregaron el salt que permite agregar saltos aleatoreo en la clave

        const salt = await bcryptjs.genSalt(10);

        const hashedPassword = await bcryptjs.hash(password, salt);

        // Crear usuario

        const newUser = await UserModel.create(email, hashedPassword, username);

pruebas de código

Captura de pantalla de un celular

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

  return res.status(201).json({ ok: true, msg: newUser });

se agrega la contraseña hasheada

Captura de pantalla de computadora

Descripción generada automáticamente

Las contraseñas son distintas

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

# Crear Token “” commit en rama

Instalar json web token

npm i jsonwebtoken

Texto

Descripción generada automáticamente

Agregar la palabra secreat en env

JWT\_SECRET="palabrasecreta"

Texto

Descripción generada automáticamente

Agregando el código

import bcryptjs from 'bcryptjs';

import { UserModel } from "../models/user.model.js";

import jwt from "jsonwebtoken";

// /api/v1/users/register  ruta para el registro

const register = async (req, res) => {

    try {

        console.log(req.body);

        const { username, email, password } = req.body;

        // Validaciones

        if (!username || !email || !password) {

            return res.status(400).json({ ok: false, msg: "Falla en requerir los campos de usuario, email, password" });

        }

        const user = await UserModel.findOneByEmail(email);

        if (user) {

            return res.status(409).json({ ok: false, msg: "El email ya existe" });

        }

        // Hash de la contraseña

        const salt = await bcryptjs.genSalt(10);

        const hashedPassword = await bcryptjs.hash(password, salt);

        // Crear usuario

        const newUser = await UserModel.create(email, hashedPassword, username);

        // Generar token JWT

        const token = jwt.sign(

            { email: newUser.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(201).json({ ok: true, msg: "Usuario registrado con éxito", newUser, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

};

// /api/v1/users/login   ruta para el login

const login = async (req, res) => {

    try {

        // Implementa la lógica de inicio de sesión aquí

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

export const UserController = { register, login };

si intento registrar el mismo usuario

Captura de pantalla de computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Podemos revisar el token en la pagina de JWT decode

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InBydWViYXNAZ21haWwuY29tIiwiaWF0IjoxNzE4MzI4MzMyLCJleHAiOjE3MTgzMzE5MzJ9.Z-XukVb3hDaWjFAILGFpNqd6VnbfYfur3hYFHk2Wf90

Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente

# Revisando el login

const login = async (req, res) => {

    try {

        // Implementa la lógica de inicio de sesión aquí

        const { username, email, password } = req.body;

        //

        if(!email || !password){

            return res.status(400).json({

                error: "Requiere los campos: email, password"

            })

        }

        const user = await UserModel.findOneByEmail(email)

        if(!user){

            return res.status(404).json({

                error: "usuario no encontrado"

            })

        }

        const isMatch = await bcryptjs.compare(password, user.password)

        if(!isMatch){

            return res.status(404).json({

                error: "credenciales invalidas"

            })

        }

         // Generar token JWT

         const token = jwt.sign(

            { email: user.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(200 ).json({ ok: true, msg: "Usuario registrado con éxito", newUser, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

export const UserController = { register, login };

código completo

import bcryptjs from 'bcryptjs';

import { UserModel } from "../models/user.model.js";

import jwt from "jsonwebtoken";

// /api/v1/users/register  ruta para el registro

const register = async (req, res) => {

    try {

        console.log(req.body);

        const { username, email, password } = req.body;

        // Validaciones

        if (!username || !email || !password) {

            return res.status(400).json({ ok: false, msg: "Falla en requerir los campos de usuario, email, password" });

        }

        const user = await UserModel.findOneByEmail(email);

        if (user) {

            return res.status(409).json({ ok: false, msg: "El email ya existe" });

        }

        // Hash de la contraseña

        const salt = await bcryptjs.genSalt(10);

        const hashedPassword = await bcryptjs.hash(password, salt);

        // Crear usuario

        const newUser = await UserModel.create(email, hashedPassword, username);

        // Generar token JWT

        const token = jwt.sign(

            { email: newUser.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(201).json({ ok: true, msg: "Usuario registrado con éxito", newUser, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

};

// /api/v1/users/login   ruta para el login

const login = async (req, res) => {

    try {

        // Implementa la lógica de inicio de sesión aquí

        const { username, email, password } = req.body;

        //

        if(!email || !password){

            return res.status(400).json({

                error: "Requiere los campos: email, password"

            })

        }

        const user = await UserModel.findOneByEmail(email)

        if(!user){

            return res.status(404).json({

                error: "usuario no encontrado"

            })

        }

        const isMatch = await bcryptjs.compare(password, user.password)

        if(!isMatch){

            return res.status(404).json({

                error: "credenciales invalidas"

            })

        }

         // Generar token JWT

         const token = jwt.sign(

            { email: user.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(200 ).json({ ok: true, msg: "Usuario login con éxito", user, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

export const UserController = { register, login };

Captura de pantalla de computadora

Descripción generada automáticamente

Validando contraseña

# Ruta protegida

Agregar en routes

Captura de pantalla de computadora

Descripción generada automáticamente

import { Router} from "express";

import { UserController } from "../controllers/user.controller.js"

import { verifyToken } from "../middlewares/jwt.middleware.js";

const router = Router()

router.post('/register', UserController.register);

router.post('/login', UserController.login);

router.get('/profile', verifyToken, UserController.profile);

export default router;

Crear un archivo middleware

Captura de pantalla de un celular

Descripción generada automáticamente

jwt.middleware.js

import jwt from 'jsonwebtoken';

export const verifyToken = (req, res, netx) => {

    let token = req.headers.authorization

    if(!token){

        return res.status(401).json({

            error: "Token not provided"

                })

    }

    netx()

}

User controller

Captura de pantalla de computadora

Descripción generada automáticamente

import bcryptjs from 'bcryptjs';

import { UserModel } from "../models/user.model.js";

import jwt from "jsonwebtoken";

// /api/v1/users/register  ruta para el registro

const register = async (req, res) => {

    try {

        console.log(req.body);

        const { username, email, password } = req.body;

        // Validaciones

        if (!username || !email || !password) {

            return res.status(400).json({ ok: false, msg: "Falla en requerir los campos de usuario, email, password" });

        }

        const user = await UserModel.findOneByEmail(email);

        if (user) {

            return res.status(409).json({ ok: false, msg: "El email ya existe" });

        }

        // Hash de la contraseña

        const salt = await bcryptjs.genSalt(10);

        const hashedPassword = await bcryptjs.hash(password, salt);

        // Crear usuario

        const newUser = await UserModel.create(email, hashedPassword, username);

        // Generar token JWT

        const token = jwt.sign(

            { email: newUser.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(201).json({ ok: true, msg: "Usuario registrado con éxito", newUser, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

};

// /api/v1/users/login   ruta para el login

const login = async (req, res) => {

    try {

        // Implementa la lógica de inicio de sesión aquí

        const { username, email, password } = req.body;

        //

        if(!email || !password){

            return res.status(400).json({

                error: "Requiere los campos: email, password"

            })

        }

        const user = await UserModel.findOneByEmail(email)

        if(!user){

            return res.status(404).json({

                error: "Correo invalido"

            })

        }

        const isMatch = await bcryptjs.compare(password, user.password)

        if(!isMatch){

            return res.status(404).json({

                error: "Contraseña invalida"

            })

        }

         // Generar token JWT

         const token = jwt.sign(

            { email: user.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(200 ).json({ ok: true, msg: "Usuario login con éxito", user, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

// Configurar la ruta protegida

const profile = async(req,rest)=>{

    try {

       // const user = await UserModel.findOneByEmail(??)

       return res.json({ ok: true})

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

export const UserController = { register, login,profile };

pruebas

Captura de pantalla de computadora

Descripción generada automáticamente

Imagen que contiene botella, tabla, teclado, computadora

Descripción generada automáticamente

Envio de token escribiendo cualquier cosa \*\*\*\*buscar el error en el código

import bcryptjs from 'bcryptjs';

import { UserModel } from "../models/user.model.js";

import jwt from "jsonwebtoken";

// /api/v1/users/register  ruta para el registro

const register = async (req, res) => {

    try {

        console.log(req.body);

        const { username, email, password } = req.body;

        // Validaciones

        if (!username || !email || !password) {

            return res.status(400).json({ ok: false, msg: "Falla en requerir los campos de usuario, email, password" });

        }

        const user = await UserModel.findOneByEmail(email);

        if (user) {

            return res.status(409).json({ ok: false, msg: "El email ya existe" });

        }

        // Hash de la contraseña

        const salt = await bcryptjs.genSalt(10);

        const hashedPassword = await bcryptjs.hash(password, salt);

        // Crear usuario

        const newUser = await UserModel.create(email, hashedPassword, username);

        // Generar token JWT

        const token = jwt.sign(

            { email: newUser.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(201).json({ ok: true, msg: "Usuario registrado con éxito", newUser, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

};

// /api/v1/users/login   ruta para el login

const login = async (req, res) => {

    try {

        // Implementa la lógica de inicio de sesión aquí

        const { username, email, password } = req.body;

        //

        if(!email || !password){

            return res.status(400).json({

                error: "Requiere los campos: email, password"

            })

        }

        const user = await UserModel.findOneByEmail(email)

        if(!user){

            return res.status(404).json({

                error: "Correo invalido"

            })

        }

        const isMatch = await bcryptjs.compare(password, user.password)

        if(!isMatch){

            return res.status(404).json({

                error: "Contraseña invalida"

            })

        }

         // Generar token JWT

         const token = jwt.sign(

            { email: user.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(200 ).json({ ok: true, msg: "Usuario login con éxito", user, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

// Configurar la ruta protegida

const profile = async(req,res)=>{

    try {

       // const user = await UserModel.findOneByEmail(??)

       return res.json({ ok: true})

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

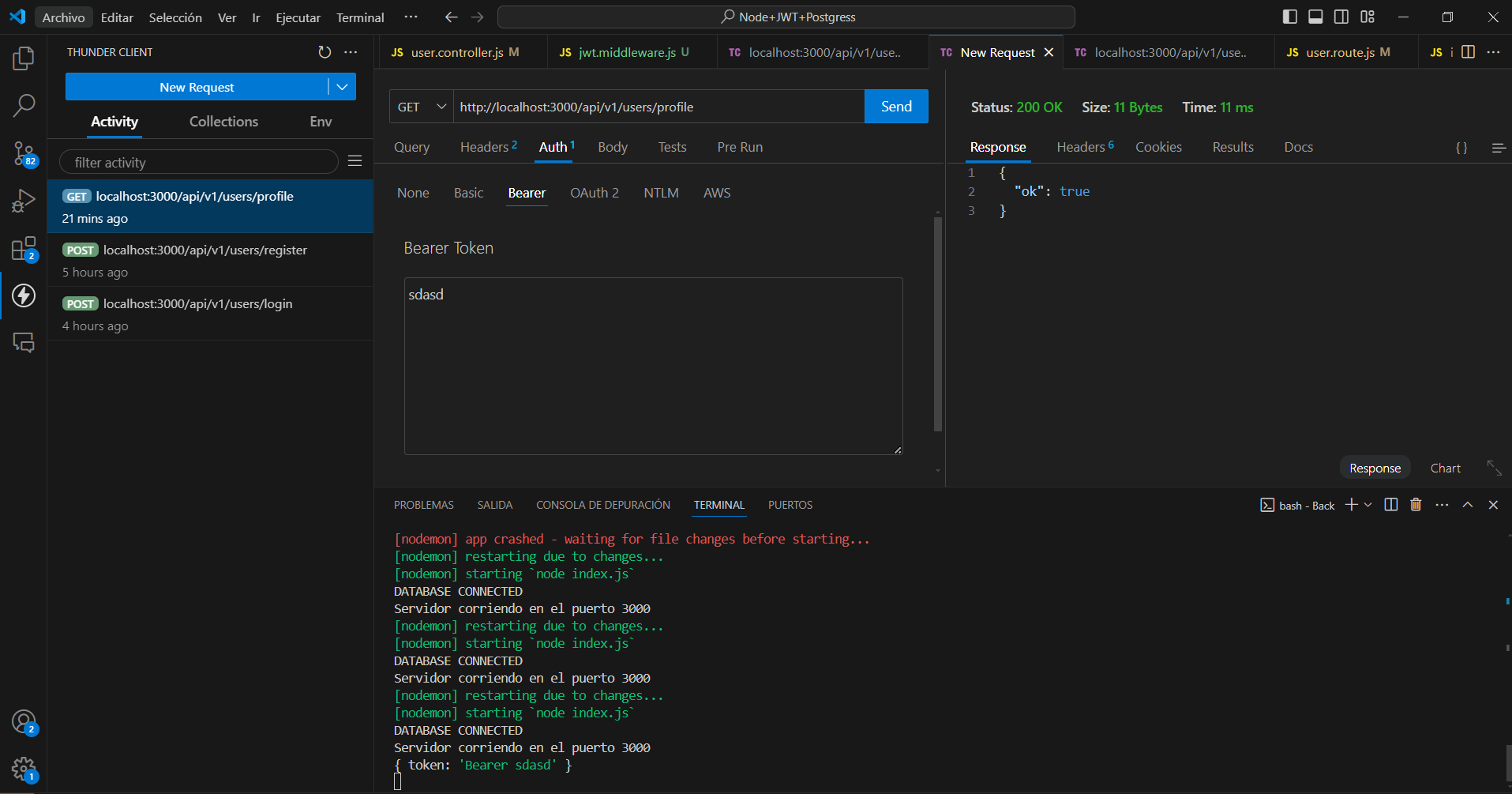
export const UserController = { register, login,profile };

ahora corrigiendo entra al true

Captura de pantalla de computadora

Descripción generada automáticamente

Ahora vamos a separar el token viene por defecto con bearer en el middleware



Antes en el token viene un bearer investigar sobre esto

Captura de pantalla de computadora

Descripción generada automáticamente

import jwt from 'jsonwebtoken';

export const verifyToken = (req, res, netx) => {

    let token = req.headers.authorization

    if(!token){

        return res.status(401).json({

            error: "Token not provided"

                })

    }

    console.log({token})

    token = token.split(" ")[1]

    try {

        // jwt.verify(token)

        netx()

    } catch (error) {

    }

}

Ahora cambiando el consola log

Captura de pantalla de computadora

Descripción generada automáticamente

Solo envia el elemento seleccionado

Texto

Descripción generada automáticamente

Proceso de prueba

import jwt from 'jsonwebtoken';

export const verifyToken = (req, res, netx) => {

    let token = req.headers.authorization

    if(!token){

        return res.status(401).json({

            error: "Token not provided"

                })

    }

    token = token.split(" ")[1]

    console.log({token})

    try {

        const {email} = jwt.verify(token, process.env.JWT\_SECRET)

        console.log(email)

        netx()

    } catch (error) {

        console.log(error)

        return res.status(400).json({ error: "Token Invalido"})

    }

}

Se crea un login

Captura de pantalla de computadora

Descripción generada automáticamente

Si se pega un token expirado da error

Captura de pantalla de computadora

Descripción generada automáticamente

Inyectar el email en la verificación del token

Agregar

req.email = email

Captura de pantalla de computadora

Descripción generada automáticamente

Y en user controller

Captura de pantalla de computadora

Descripción generada automáticamente

Agregando msg: user

Captura de pantalla de computadora

Descripción generada automáticamente

Ahora devuelve todo

Captura de pantalla de computadora

Descripción generada automáticamente

Código

middleware

import jwt from 'jsonwebtoken';

export const verifyToken = (req, res, netx) => {

    let token = req.headers.authorization

    if(!token){

        return res.status(401).json({

            error: "Token not provided"

                })

    }

    token = token.split(" ")[1]

    console.log({token})

    try {

        const {email} = jwt.verify(token, process.env.JWT\_SECRET)

        console.log(email)

        req.email = email

        netx()

    } catch (error) {

        console.log(error)

        return res.status(400).json({ error: "Token Invalido"})

    }

}

User\_controller

import bcryptjs from 'bcryptjs';

import { UserModel } from "../models/user.model.js";

import jwt from "jsonwebtoken";

// /api/v1/users/register  ruta para el registro

const register = async (req, res) => {

    try {

        console.log(req.body);

        const { username, email, password } = req.body;

        // Validaciones

        if (!username || !email || !password) {

            return res.status(400).json({ ok: false, msg: "Falla en requerir los campos de usuario, email, password" });

        }

        const user = await UserModel.findOneByEmail(email);

        if (user) {

            return res.status(409).json({ ok: false, msg: "El email ya existe" });

        }

        // Hash de la contraseña

        const salt = await bcryptjs.genSalt(10);

        const hashedPassword = await bcryptjs.hash(password, salt);

        // Crear usuario

        const newUser = await UserModel.create(email, hashedPassword, username);

        // Generar token JWT

        const token = jwt.sign(

            { email: newUser.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(201).json({ ok: true, msg: "Usuario registrado con éxito", newUser, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

};

// /api/v1/users/login   ruta para el login

const login = async (req, res) => {

    try {

        // Implementa la lógica de inicio de sesión aquí

        const { username, email, password } = req.body;

        //

        if(!email || !password){

            return res.status(400).json({

                error: "Requiere los campos: email, password"

            })

        }

        const user = await UserModel.findOneByEmail(email)

        if(!user){

            return res.status(404).json({

                error: "Correo invalido"

            })

        }

        const isMatch = await bcryptjs.compare(password, user.password)

        if(!isMatch){

            return res.status(404).json({

                error: "Contraseña invalida"

            })

        }

         // Generar token JWT

         const token = jwt.sign(

            { email: user.email },

            process.env.JWT\_SECRET,

            { expiresIn: "1h" }

        );

        return res.status(200 ).json({ ok: true, msg: "Usuario login con éxito", user, token });

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

// Configurar la ruta protegida

const profile = async(req,res)=>{

    try {

        const user = await UserModel.findOneByEmail(req.email)

       return res.json({ ok: true, msg: user})

    } catch (error) {

        console.log(error);

        return res.status(500).json({

            ok: false,

            msg: 'Error del servidor'

        });

    }

}

export const UserController = { register, login,profile };

routes

import { Router} from "express";

import { UserController } from "../controllers/user.controller.js"

import { verifyToken } from "../middlewares/jwt.middleware.js";

const router = Router()

router.post('/register', UserController.register);

router.post('/login', UserController.login);

router.get('/profile', verifyToken, UserController.profile);

export default router;