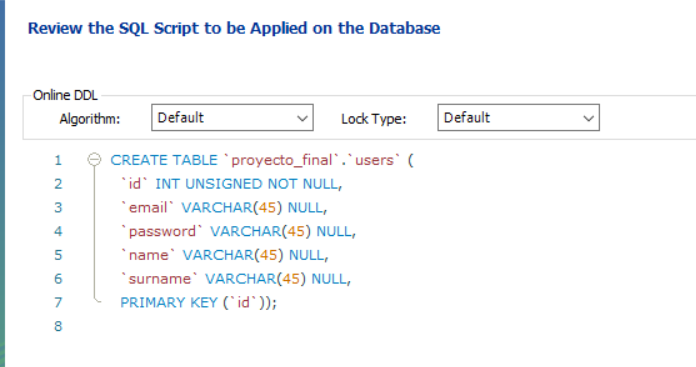
# Registro y login:

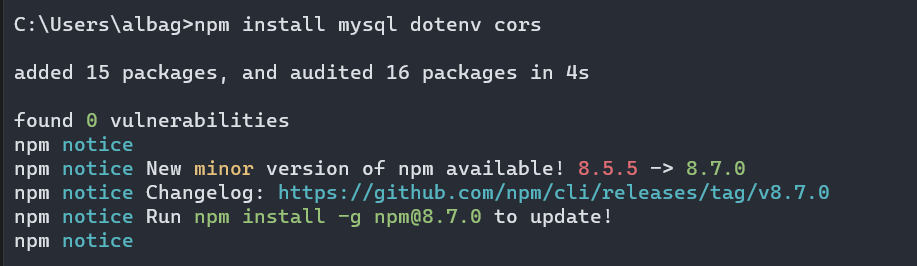
* Nodejs y librerías: moment, jwt-simple, bcrypt, dotenv, mysql, cors.
* Express v4
* MySQL v5.7
* Creamos una base de datos para el proyecto en MySQL , solo crearemos la tabla “users” con datos simples para poder autentificar y logear.



**Conectamos nuestra aplicación a la base de datos:**  
- Instalamos en node cors, mysql y dotenv.  
- Creamos en nuestro proyecto un archivo .env, dentro colocamos los siguientes datos: host, user, password, port, name “ Todos de nuestra base de datos antes creada en MySQL ”.  
- Creamos db.js , aqui estara la configuración de nuestra base de datos.  
- Requerimos en nuestro app.js a db.js, cors y dotenv.  
- Por ultimo usamos cors en app.js.

**Instalación de mysql, cors y dotenv**

npm install mysql dotenv cors



**Archivo .env**

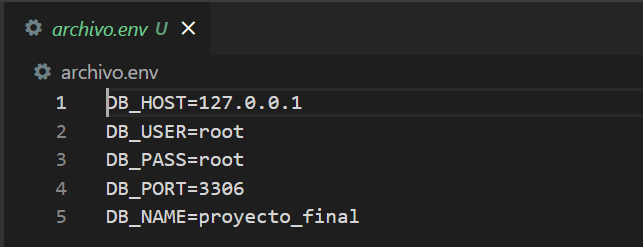
**DB\_HOST=127.0.0.1**

**DB\_USER=root**

**DB\_PASS=root**

**DB\_PORT=3306**

**DB\_NAME=proyecto\_final**

****

**Configuración db.js (Base de datos)**

const mysql = require( 'mysql');

const pool = mysql.createPool({

    host: process.env.DB\_HOST,

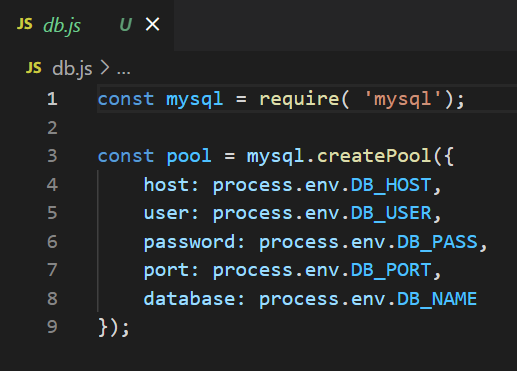
    user: process.env.DB\_USER,

    password: process.env.DB\_PASS,

    port: process.env.DB\_PORT,

    database: process.env.DB\_NAME

});



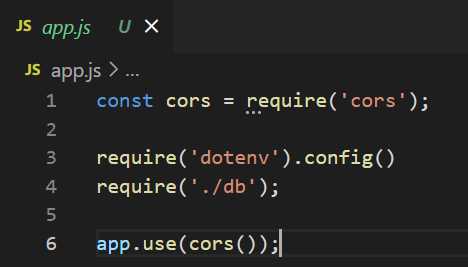
**Require de cors, dotenv y db.js, uso de cors.**

const cors = require('cors');

require('dotenv').config()

require('./db');

app.use(cors());



* Creamos una carpeta ***models*** y dentro el modelo ***users.js,*** donde estarán todas las funciones para las peticiones a la base de datos de usuarios.
* Hacemos la primera función en ***users.js*** que nos servirá para comprobar la conexión con la base de datos.

**Función para pedir todos los usuarios a la base de datos**

const getAll = () => {

    return new Promise((resolve, reject) => {

        db.query('SELECT \* FROM users', (err, rows) => {

            if (err) reject(err)

            resolve(rows);

        });

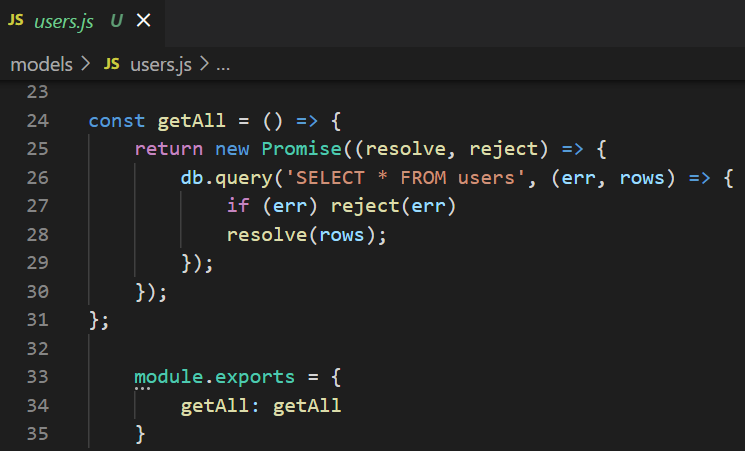
    });

};

    module.exports = {

        getAll: getAll

    }



* Creamos un manejador de rutas para poder mostrar un JSON y usar nuestro modelo de usuario en ***routes/users.js.***

const express = require('express');

const router = express.Router();

const Users= require('../models/users');

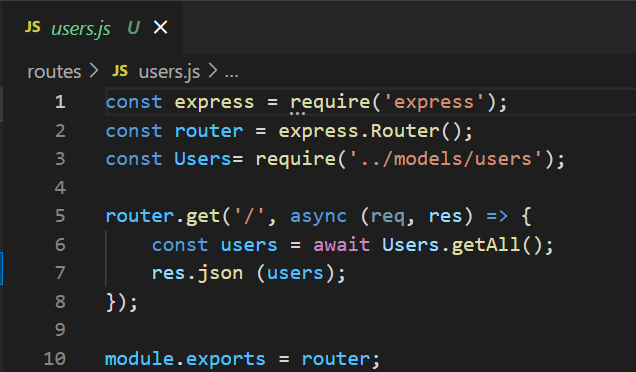
router.get('/', async (req, res) => {

    const users = await Users.getAll();

    res.json (users);

});

module.exports = router;



* Hacemos otras dos funciones en nuestro ***models/users.js*** que nos servirán para registrar un usuario y para el login.

**Funciones insert y getByEmail**

/\*  Registro de usuarios \*/

const insert = ({ email, password, name, surname }) => {

    return new Promise((resolve, reject) => {

        db.query('INSERT INTO users (email, password, name, surnane) VALUES (?, ?, ?, ? )', [email, password, name, surname], (err, result) => {

            if (err) reject(err)

            if (result) {

                resolve(result)

            };

        });

    });

};

/\* Obtener usuarios por su email \*/

const getByEmail = (pEmail) => {

    return new Promise((resolve, reject) => {

        db.query('SELECT FROM users wWHERE email = ?', [pEmail], (err, rows) => {

            if (err) reject(err)

            resolve(rows[0])

        });

    });

};

const getAll = () => {

    return new Promise((resolve, reject) => {

        db.query('SELECT \* FROM users', (err, rows) => {

            if (err) reject(err)

            resolve(rows);

        });

    });

};

module.exports = {

    getAll: getAll,

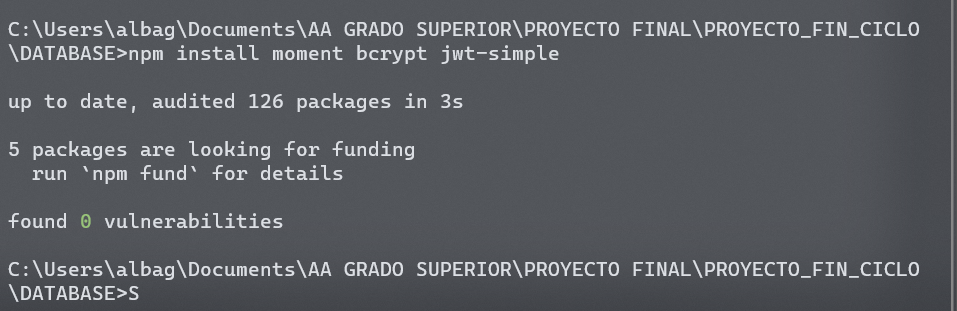
    insert: insert,

    getByEmail: getByEmail

}

* Usaremos ***users.js*** que ya debe estar en la carpeta ***routes.***
* Instalamos bcrypt, jwt-simple y moment en nuestro proyecto.

npm install moment bcrypt jwt-simple



* Creamos el manejador de rutas ***‘/register’***y requerimos las librerías instaladas en***routes/users.js.***bcrypt nos encripta el password, así que recordar la password para poder verificarla luego en el login.

**Manejador de ruta ‘/register’**

const express = require('express');

const router = express.Router();

const Users= require('../models/users');

const bcrypt = require ('bcrypt');

const jwt = require('jwt-simple') ;

const moment = require( 'moment ');

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

    const users = await Users.getAll();

    res.json (users);

});

router.post ('/register', async (req, res) => {

    console.log (req.body);

    req.body.password = bcrypt.hashSync(req.body.password, 10);

    const result = await Users.insert(req.body);

    res.json( result);

});

module.exports = router;

Creamos la función para generar un Token en routes/users.js, el cual nos permitirá comprobar cuando el usuario esté logueado con ayuda del navegador.

- Agregar al .env el TOKEN\_KEY=”Token-Auth”

**Función createToken**

const createToken = (user)=> {

    let payload = {

    userId: user.id,

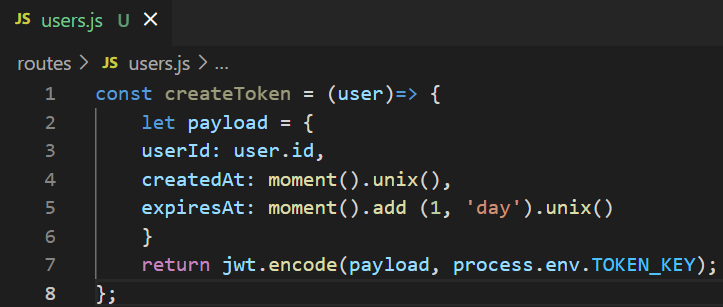
    createdAt: moment().unix(),

    expiresAt: moment().add (1, 'day').unix()

    }

    return jwt.encode(payload, process.env.TOKEN\_KEY);

};



**Archivo .env**

DB\_HOST=127.0.0.1

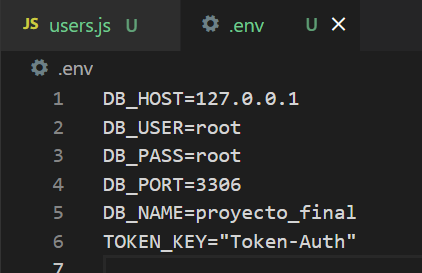
DB\_USER=root

DB\_PASS=root

DB\_PORT=3306

DB\_NAME=proyecto\_final

TOKEN\_KEY="Token-Auth"



* Procedemos a crear el manejador para la ruta***‘/login’*** .

router.post('/login', async (req, res) => {

    const user = await Users.getByEmail(req.body.email)

    if (user === undefined) {

        res.json({

            error: 'Error, email or password not found'

        })

    } else {

        const equals = bcrypt.compareSync(req.body.password, user.password);

        if (!equals) {

            res.json({

                error: 'Error, email or pas sword not found'

            });

        } else {

            res.json({

                succesfull: createToken(user),

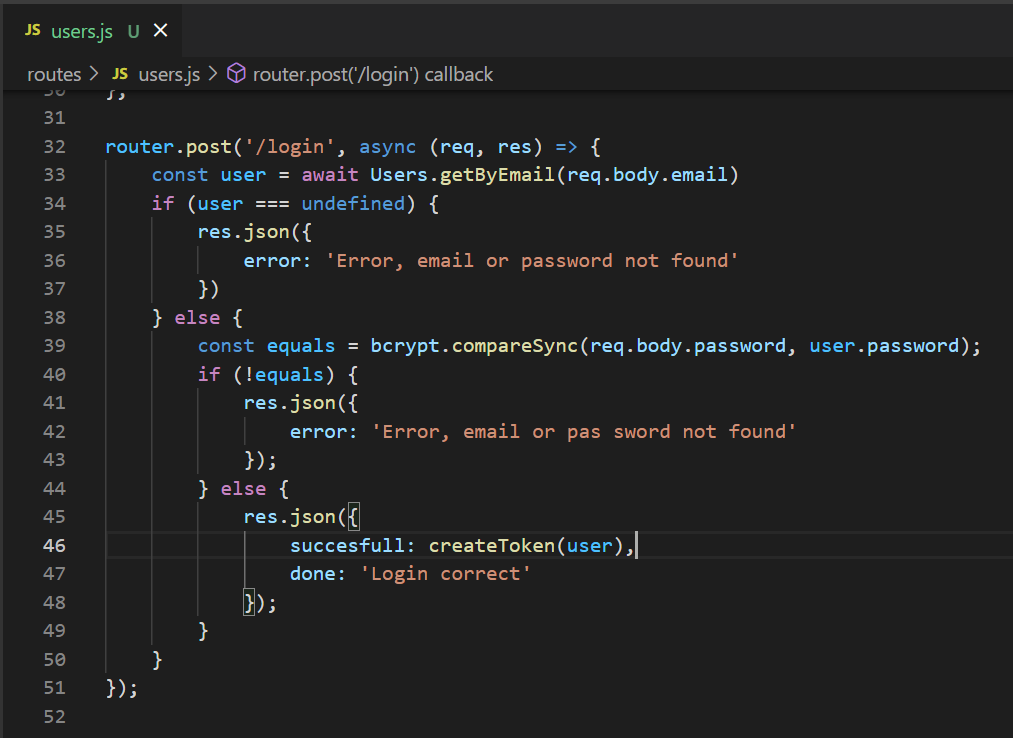
                done: 'Login correct'

            });

        }

    }

});



Ya tenemos tanto el login que nos genera un Token y el registro que encripta nuestra password, para finalizar con nodeJS y nuestro backend, nos falta crear un middleware, este se encargará de verificar cada petición que le hagan a ‘/users’ y esperar el Token como header, si el Token existe obtendremos el id del usuario y de esa manera su información para que en cada ruta de nuestra aplicación tengamos activo al Usuario logueado**.**

* Creamos nuestro *middleware.js*en la carpeta *routes*.

**middleware.js**

const jwt = require('jwt-simple');

const moment = require('moment');

const checkToken = (req, res, next) => {

    if (!req.headers['user\_token'])

        return res.json({

            error: "You must include the header"

        });

    const token = req.headers['user\_token'];

    let payload = null

    try {

        payload = jwt.decode(token, process.env.TOKEN\_KEY)

    } catch (err) {

        return res.json({

            error: 'Invalid token'

        });

    }

    if (moment().unix() > payload.expiresAt) {

        return res.json({ error: 'Expired token');

    };

    req.userId = payload.userId;

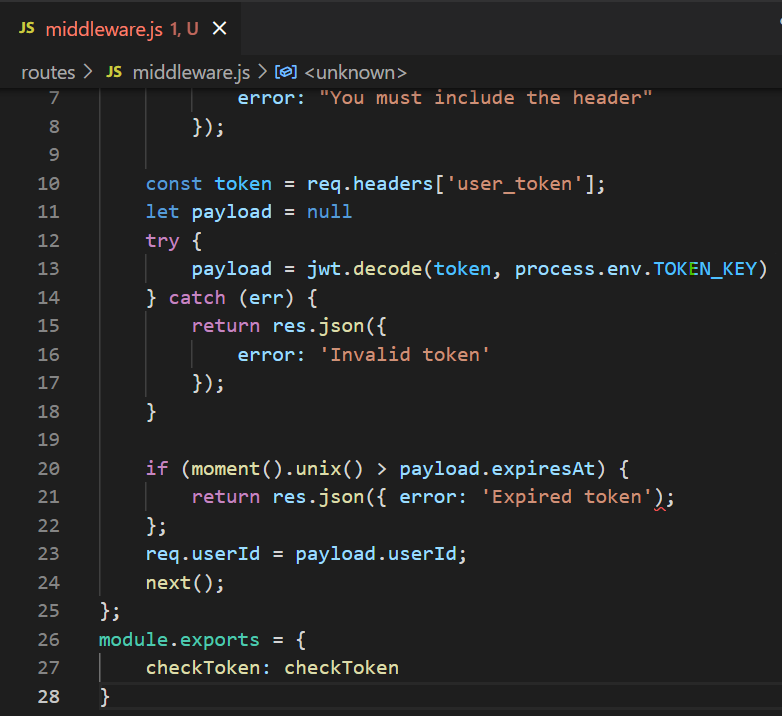
    next();

};

module.exports = {

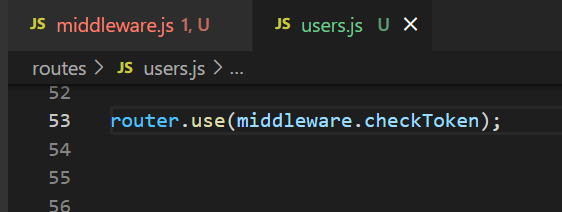
    checkToken: checkToken

}



* Usamos el *middleware* en *routes/users.js* después de nuestro manejadores de rutas ‘*/login’*y *‘/register’*, si lo usaramos antes, nos pediría el Token en el header de la petición que aun no tendríamos generado ya que este Token se crea en el *login*.

router.use(middleware.checkToken);



* Creamos una nueva función en ***models/users.js*** para obtener un usuario a partir de su ***id***.

**Función getById**

/\* Obtener usuarios por su ID \*/

const getById = (pId) => {

    return new Promise((resolve, reject) => {

        db.query('SELECT FROM users WHERE id = ?', [pId], (err, rows) => {

            if (err) reject(err)

            resolve(rows[0])

        });

    });

};

module.exports = {

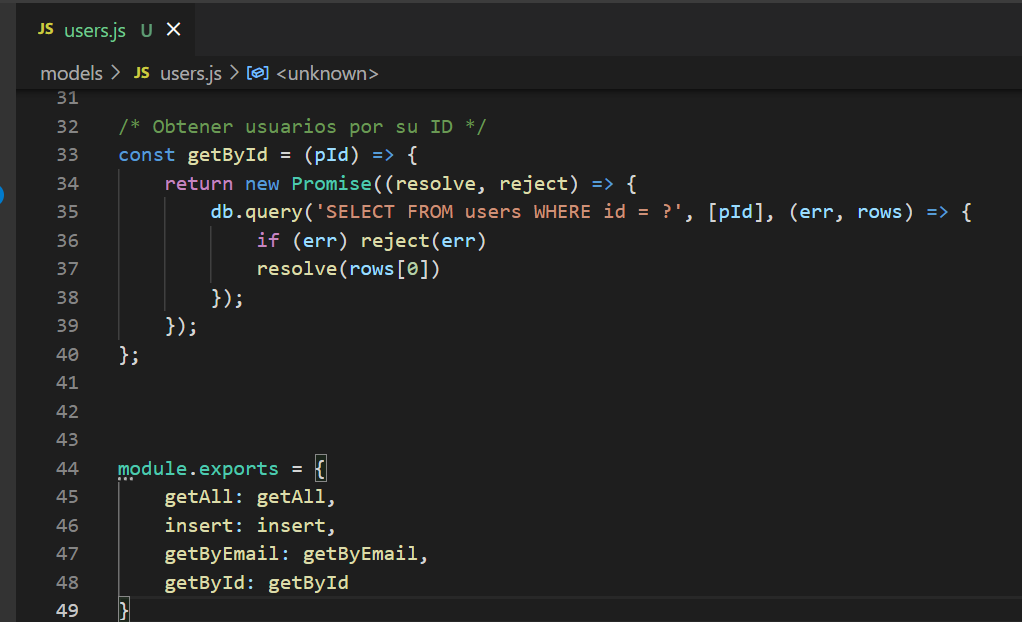
    getAll: getAll,

    insert: insert,

    getByEmail: getByEmail,

    getById: getById

}

****

* Para finalizar nuestro login en el backend necesitaremos un manejador de ruta que utilice el Token que nos enviaran en el header de la petición y nos devuelva el usuario que se ha logueado, que posteriormente usaremos en nuestro frontend con Angular o cualquier otro framework.
* Creamos un nuevo manejador de ruta ***‘/mainUser’*** en ***routes/users.js***, este recibirá el id del usuario en el header gracias a nuestro middleware.

**Manejador /mainUser**

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

    Users.getById(req.userId)

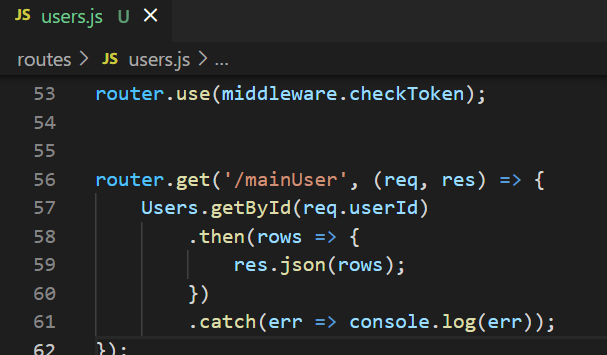
        .then(rows => {

            res.json(rows);

        })

        .catch(err => console.log(err));

});

****

**SEGUNDA OPCION:**

Empecemos con el código

En nuestro archivo **server/server.js** consta de varias partes las cuales, las explico de forma breve

Exportamos nuestra dependencias al inicio del archivo. Estas dependencias serán las que nos ayuden al funcionamiento correcto del server.

require('./config/config');

const express = require('express')

const app = express()

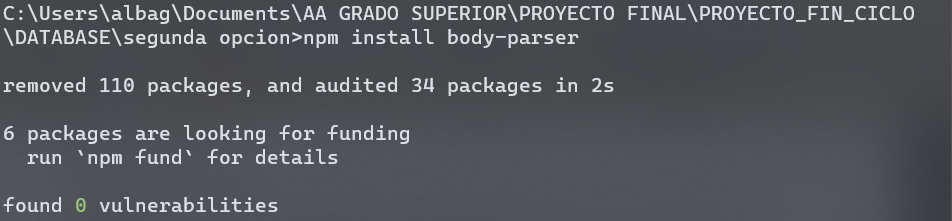
const mongoose = require('mongoose');

const bodyParser = require('body-parser')

const path = require('path');

El siguiente código es básicamente un middleware que nos ayuda a “parsear” los datos que recibimos a través del protocolo http.

Instalamos el siguiente paquete con este comando “**npm install body-parser**”



Fichero **server/server.js**

// parse application/x-www-form-urlencoded

app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json

app.use(bodyParser.json())

Los controladores deben de tener su propio archivo y exportarlos como un “middleware” al archivo server.js

// Configuracion global de rutas

app.use(require('./routes/index'));

Las siguientes líneas nos indica una ruta raíz, es decir, al momento que vamos a localhost:3000 nodejs nos servirá un archivo html sencillo.

let renderHTML = path.resolve(\_\_dirname, '../public/index.html');

app.get('/', function (req, res) {

    res.sendFile(renderHTML);

})

La conexión a nuestra db no debe de faltar

mongoose.connect(process.env.URLDB, {

    useNewUrlParser: true,

    useCreateIndex: true,

    useUnifiedTopology: true

}, (err) => {

    if (err) throw err;

    console.log("Base de datos online");

});

Process.env.URLDB y process.env.PORT son una variable de entorno que veremos más adelante.

Y por último, el código para lanzar nuestro servidor

app.listen(process.env.PORT, ()=> {

    console.log("Escuchando en puerto 3000");

})

**Instalación de mongo db:**

Todo proyecto web, requiere de configuraciones básicas disponibles desde cualquier lado de nuestro proyecto, y la mayoría de entornos de desarrollo contienen estas variables de configuración, como en laravel, tenemos un archivo llamado .env, y en nodejs no es la excepción; a estas configuraciones se les conoce como variables de entorno, y es acá donde definimos por ejemplo, nuestra cadena de conexión, un seed de encriptación de contraseña; entre otros.

Dentro de la carpeta **/config/config.js** definimos:

* Puerto de acceso desde nuestro localhost:

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

* Entorno de desarrollo (dev, local, prod, test):

process.env.NODE\_ENV = process.env.NODE\_ENV || 'dev';

* Cadena de conexión a nuestra bd:

let urlDB = "";

if (process.env.NODE\_ENV === 'dev') {

    urlDB = "mongodb://localhost:27017/mediumNodeLogin";

} else {

    urlDB = "here write the mongo connection with mongo atlas and      other type of connection mode"

};

process.env.URLDB = urlDB;

* Caducidad del token:

process.env.CADUCIDAD\_TOKEN = '48h';

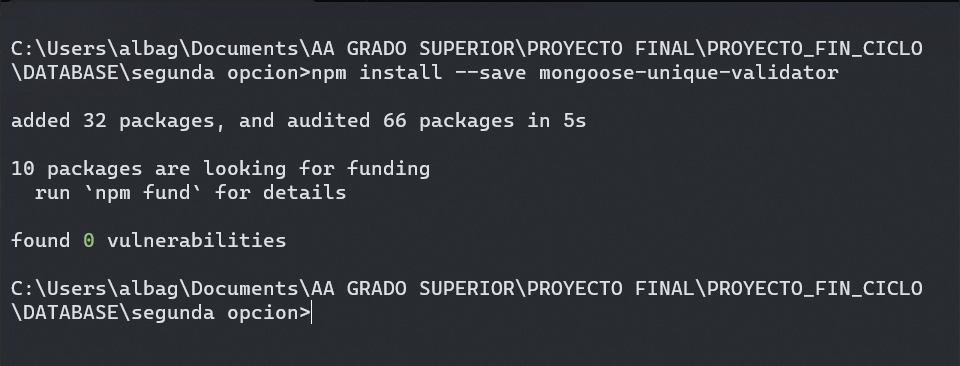
* Seeds de autenticación.

process.env.SEED\_AUTENTICACION = process.env.SEED\_AUTENTICACION ||  'este-es-el-seed-desarrollo';

**Creando modelo usuario:**

Los modelos los podemos nombrar con cualquier nombre, en este caso, modelo Usuario; que facilmente podría llamarse modelo User, o como se les sea más conveniente.

Empezamos importando los archivos js que requerimos para hacer un modelo con mongoose



const mongoose = require('mongoose');

var uniqueValidator = require('mongoose-unique-validator');

La importación de uniqueValidator lo que hace es facilitarnos de escribir por nosotros mismos la validación de que un campo sea único, en este caso, el campo email, email es de suma importancia que sea único en nuestro modelo Usuario.

Continuamos definiendo un Object que nos ayude a validar que tipo de rol será válido para usuario. Por ejemplo, queremos asegurarnos de que solo existan dos tipos de roles, ADMIN Y USER y que así, evitamos que nos envíen un role que no esta permitido en nuestra db, por ejemplo el role CONSERJE.

Además, definimos un Schema de mongoose.

let rolesValidos = {

    values: ["ADMIN", "USER"],

    message: '{VALUE} no es un role válido'

}

let Schema = mongoose.Schema;

Luego continuamos definiendo un nuevo Schema; recuerda que acá es donde definimos los campos de nuestro document de mongodb; es decir que si queremos que exista el campo por poner un ejemplo de estado de si está o no activo el usuario; debe de ir dentro del modelo Usuario

let usuarioSchema = new Schema({

    nombre: {

        type: String,

        required: [true, 'El nombre es necesario'],

    },

    email: {

        type: String,

        unique: true,

        required: [true, "El correo es necesario"],

    },

    password: {

        type: String,

        required: [true, "Le contraseña es obligatoria"],

    },

    role: {

        type: String,

        default: 'USER',

        required: [true],

        enum: rolesValidos,

    },

    });

Ahora, el modelo Usuarios también se encarga de traernos la data de cada usuario registrado al momento de hacer login; en este caso, nos devolvería toda la información del modelo, incluyendo la contraseña, aunque la contraseña estará encriptada con un hash jwt ¿queremos que cualquier usuario tenga acceso a la cadena encriptada de nuestros usuarios?

La respuesta es un rotundo no, por lo cual, al momento de hacer login, tenemos que asegurarnos que este campo no nos lo devuelva, motivo por el cual, tendremos que eliminarlo de la response de la petición http

// elimina la key password del objeto que retorna al momento de crear un usuario

usuarioSchema.methods.toJSON = function() {

    let user = this;

    let userObject = user.toObject();

    delete userObject.password;

    return userObject;

 }

Y por último, agregamos el plugin de validación única y exportamos el modelo recién creado.

usuarioSchema.plugin(uniqueValidator, {

    message: '{PATH} debe de ser único'

})

module.exports = mongoose.model('Usuario', usuarioSchema)

**Rutas de nuestra app:**

Como se observó en la estructura de carpetas, dentro de la carpeta routes contamos con 3 archivos, el index, que es el archivo principal donde importamos el resto de rutas; seguido del archivo login.js y register.js que contiene el código pertinente para hacer login y register.

Nuestro archivo **/routes/index.js**contiene el siguiente código:

const express = require('express')

const app = express()

app.use(require('./login'));

app.use(require('./register'));

module.exports = app;

Importamos express y app para hacer uso del middleware .use y al final, exportamos app con module.exports

**Ruta de login:**

Empezamos importando a nuestro archivo /routes/login.js las dependencias que se requieren para el funcionamiento; en este caso ya requerimos del modelo recién creado; ya que es a través de este modelo que vamos a poder guardar, editar, buscar, y listas los usuarios registrados.

const express = require('express');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const Usuario = require('./../models/usuario');

const app = express();

Es de suma importancia que nuestros modelos sean asignados con la primera letra en mayúscula, esto para que podamos diferenciar el uso del modelo en el código.

Las rutas se crean con la siguiente sintaxis:

app.post('/login', function (req, res) { // here write code  
})

donde app es la referencia a express y .post es el verbo http que requerimos para está ruta, es decir, que si deseamos un get, un update o delete, .post debe de ser reemplazado por cualquier otro verbo requerido.

dentro recibe dos parámetros (en este caso) el primero ‘/login’ que es el nombre que le asignamos a la ruta, y como segundo parámetro, recibe una función donde se recibe como parametro req y res, **req**es la encargada de recibir la data proveniente del frontend, y **res**es quien se encarga de devolvernos un status code y/o header que establezcamos.

Dentro de la ruta, debemos de obtener la información proveniente del frontend.

let body = req.body;

y guardamos la información que recibimos.

Usuario.findOne({ email: body.email }, (erro, usuarioDB) => {

    if (erro) {

        return res.status(500).json({

            ok: false,

            err: erro

        })

    }

    // Verifica que exista un usuario con el mail escrita por el usuario.

    if (!usuarioDB) {

        return res.status(400).json({

            ok: false,

            err: {

                message: "Usuario o contraseña incorrectos"

            }

        })

    }

    // Valida que la contraseña escrita por el usuario, sea la almacenada en la db

    if (!bcrypt.compareSync(body.password, usuarioDB.password)) {

        return res.status(400).json({

            ok: false,

            err: {

                message: "Usuario o contraseña incorrectos"

            }

        });

    }

    // Genera el token de autenticación

    let token = jwt.sign({

        usuario: usuarioDB,

    }, process.env.SEED\_AUTENTICACION, {

        expiresIn: process.env.CADUCIDAD\_TOKEN

    })

    res.json({

        ok: true,

        usuario: usuarioDB,

        token,

    })

})

y al finalizar, exportamos nuevamente app

module.exports = app;

**Ruta register:**

La ruta register es similar a la ruta login, a diferencia que acá es donde se tiene que encriptar la contraseña, y es donde se hace uso de **bcrypt.**

const express = require('express');

const bcrypt = require('bcrypt');

const Usuario = require('./../models/usuario');

const app = express();

app.post('/register', function (req, res) {

  let body = req.body;

  let { nombre, email, password, role } = body;

  let usuario = new Usuario({

    nombre,

    email,

    password: bcrypt.hashSync(password, 10),

    role

  });

usuario.save((err, usuarioDB) => {

    if (err) {

      return res.status(400).json({

         ok: false,

         err,

      });

    }

    res.json({

          ok: true,

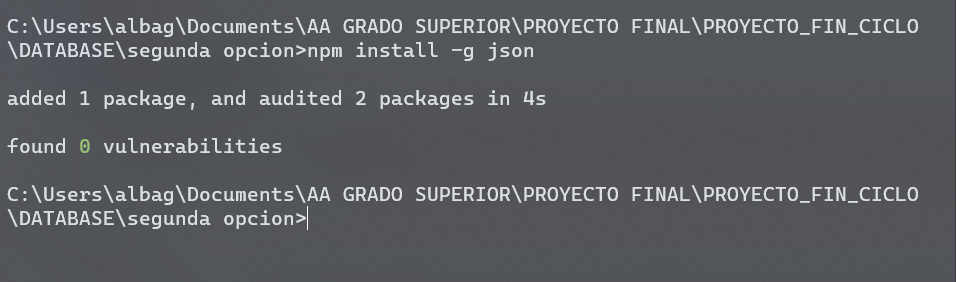
          usuario: usuarioDB

       });

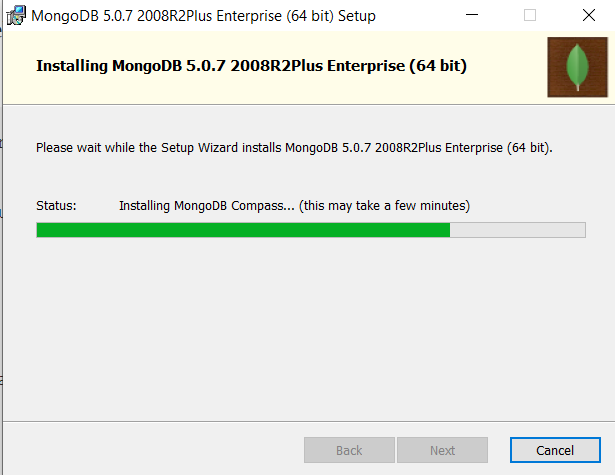
    })

});

module.exports = app;

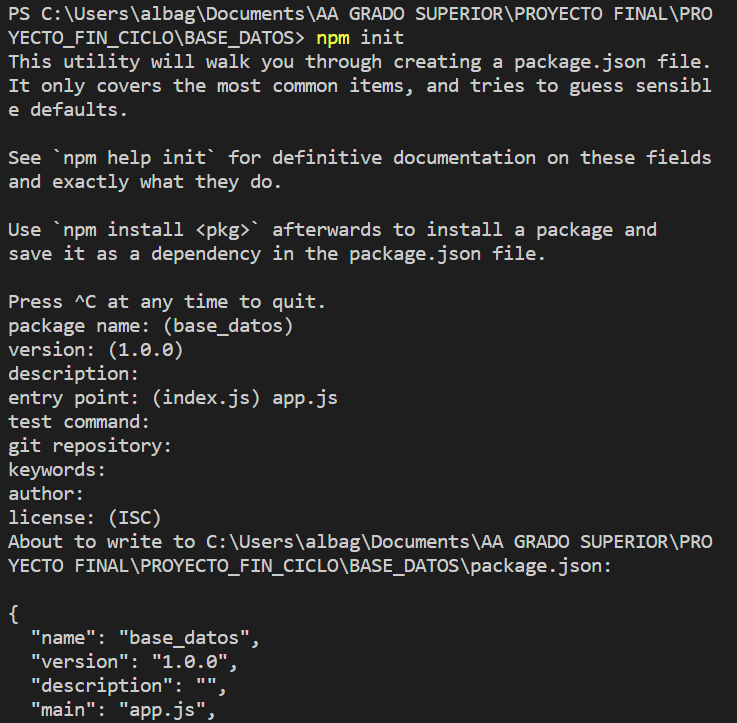
****

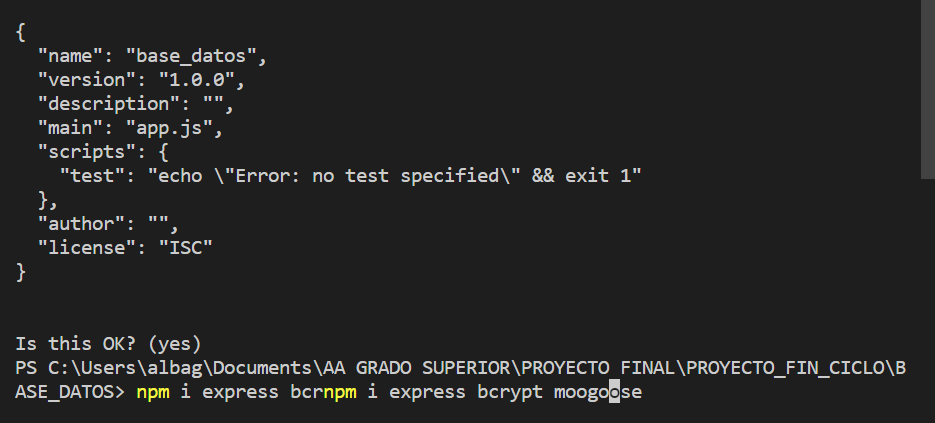
**Instalación de mongo db:**

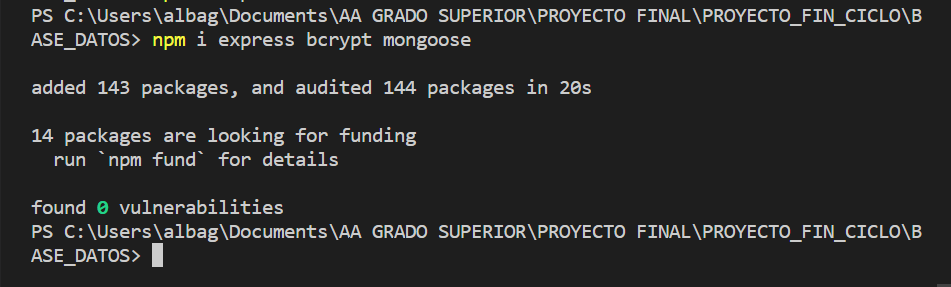


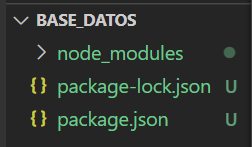


Tercera opción:









App.js

const express = require('express');

const path = require('path');

const bodyParser = require('body-parser');

const app = express();

const bcrypt = require('bcrypt');

const mongoose = require('mongoose');

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: false }));

app.use(express.static(path.join(\_dirname, 'public')));

const mongo\_uri = 'mongodb://dev:dev@localhost/proyecto\_final';

mongoose.connect(mongo\_uri, function (err) {

    if (err) {

        throw err;

    } else {

        console.log(`Succefully connected to ${mongo\_uri}`)

    }

});

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

});

app.listen(3000, () => {

    console.log('server started');

})

module.exports = app;

index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Login</title>

</head>

<body>

    hola

</body>

</html>

