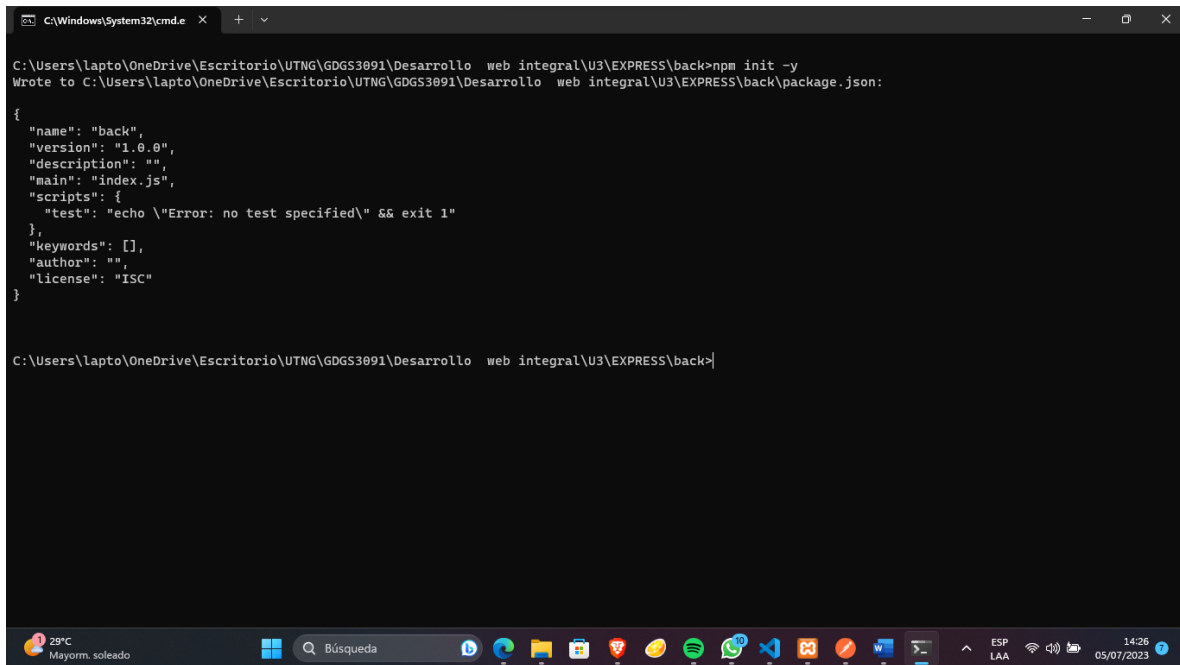


## Iniciando nuestra aplicación en node js



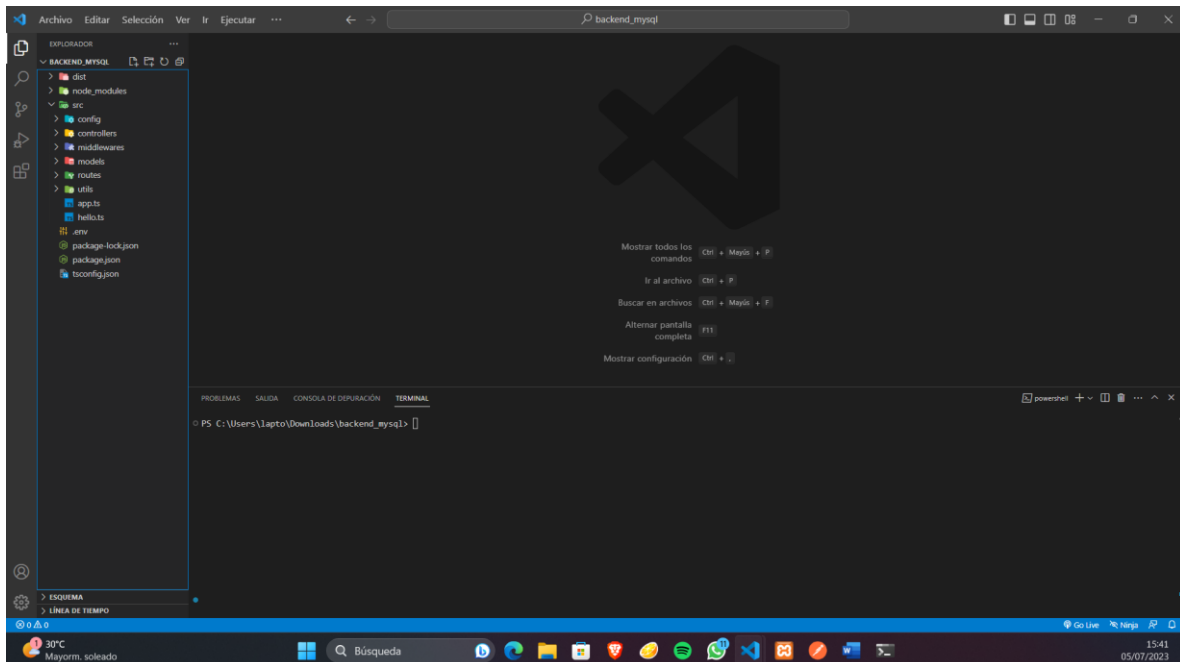
A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The command prompt shows the following text:

```
C:\Users\lapto\OneDrive\Escritorio\UTNG\GDGS3091\Desarrollo web integral\U3\EXPRESS\back>npm init -y
Wrote to C:\Users\lapto\OneDrive\Escritorio\UTNG\GDGS3091\Desarrollo web integral\U3\EXPRESS\back\package.json:

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

The prompt then shows the current directory: `C:\Users\lapto\OneDrive\Escritorio\UTNG\GDGS3091\Desarrollo web integral\U3\EXPRESS\back>`

## Implementando nuestros directorios para el proyecto



## Creación del tsconfig.json

```
C:\Users\lapto\OneDrive\Escritorio\UTNG\GDGS3091\Desarrollo web integral\U3\EXPRESS\back>tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true

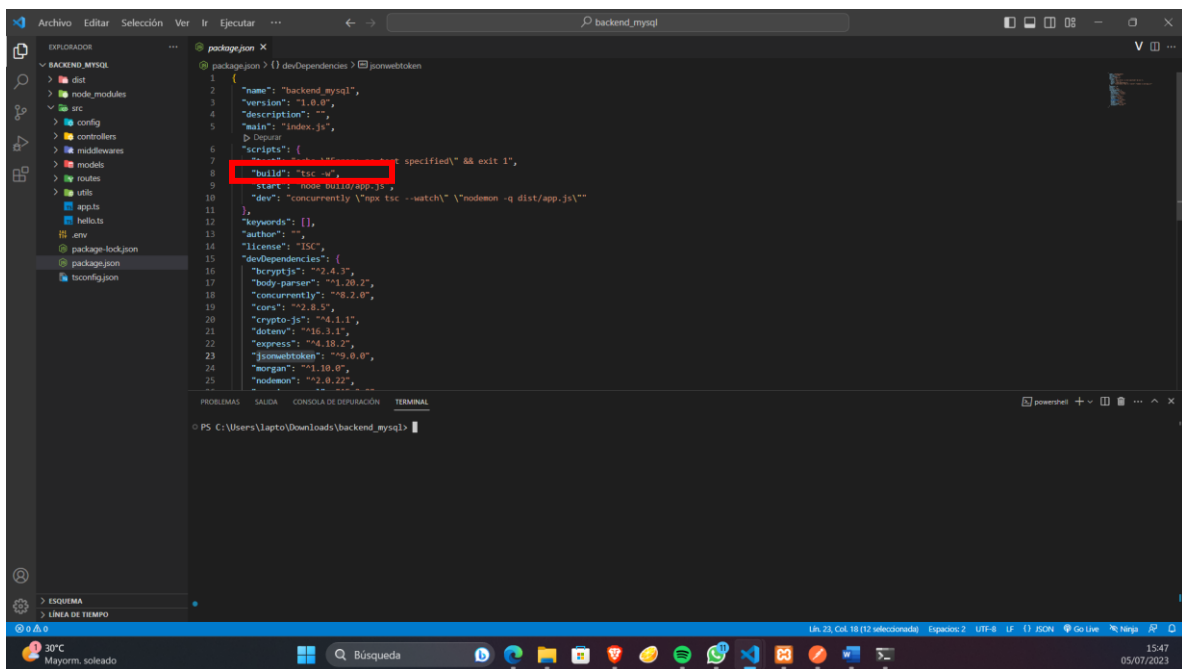
You can learn more at https://aka.ms/tsconfig

C:\Users\lapto\OneDrive\Escritorio\UTNG\GDGS3091\Desarrollo web integral\U3\EXPRESS\back>A
```

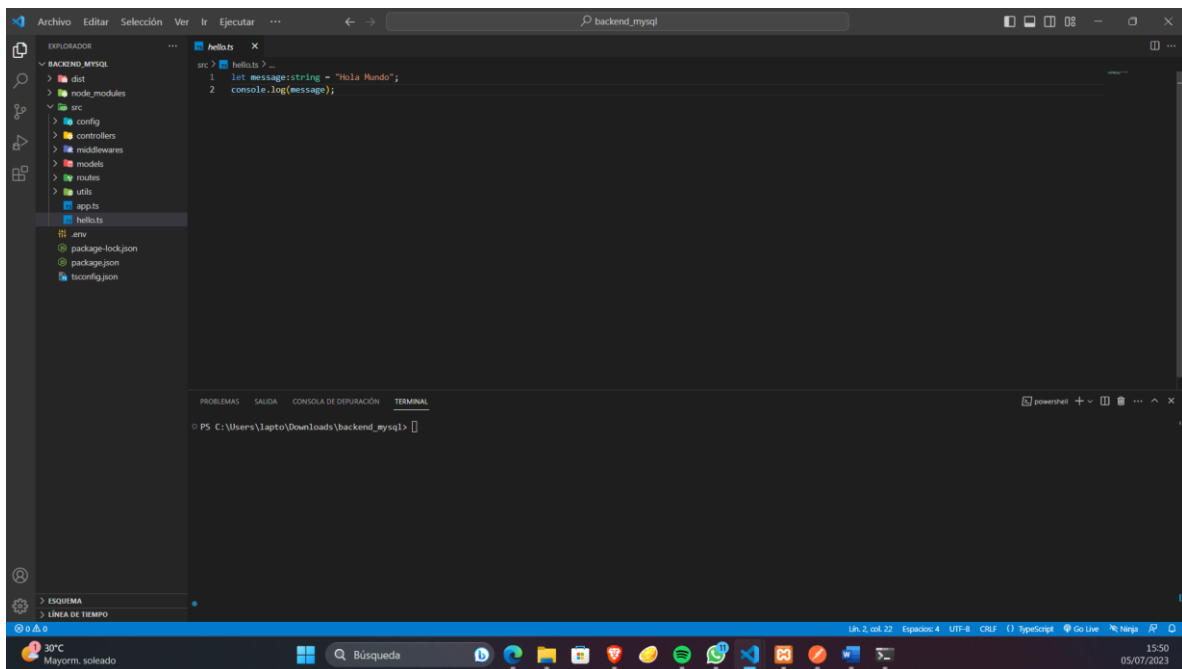
Poniendo la ruta para que al momento de mandarlo a produccion lo pase a javascript

```
tsconfig.json
{
  "compilerOptions": {
    "target": "es2016",
    "module": "commonjs",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "outDir": "build",
    "removeComments": true,
    "noEmit": true,
    "importHelpers": true,
    "importsNotUsedAsValues": "remove",
    "downlevelIteration": true,
    "sourceMap": true,
    "mapRoot": ""
  },
  "include": ["src"]
}
```

Poniendo el comando para observar los cambios



**Mandando un saludo, se creó la carpeta build**



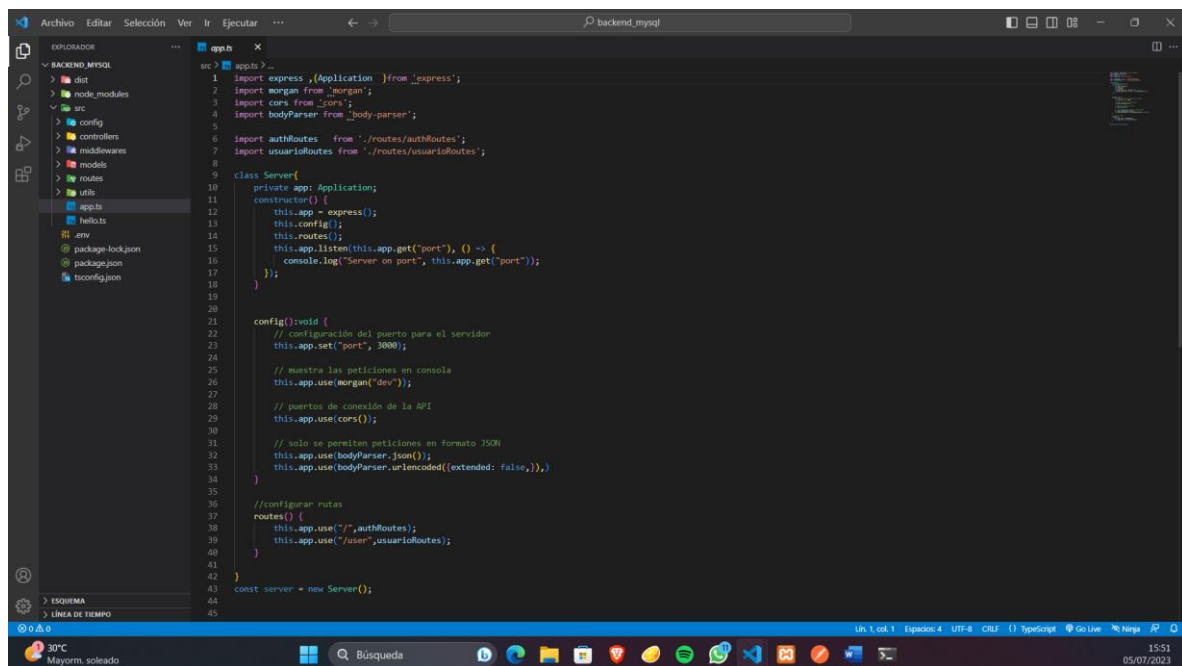
**Dependencias instaladas**

```

11  },
12  "keywords": [],
13  "author": "",
14  "license": "ISC",
15  "devDependencies": {
16    "bcryptjs": "^2.4.3",
17    "body-parser": "^1.20.2",
18    "concurrently": "^8.2.0",
19    "cors": "^2.8.5",
20    "crypto-js": "^4.1.1",
21    "dotenv": "^16.3.1",
22    "express": "^4.18.2",
23    "jsonwebtoken": "^9.0.0",
24    "morgan": "^1.10.0",
25    "nodemon": "^2.0.22",
26    "promise-mysql": "^5.2.0",
27    "validator": "^13.9.0"
28  }
29 }
30

```

## Configuramos nuestro index.ts

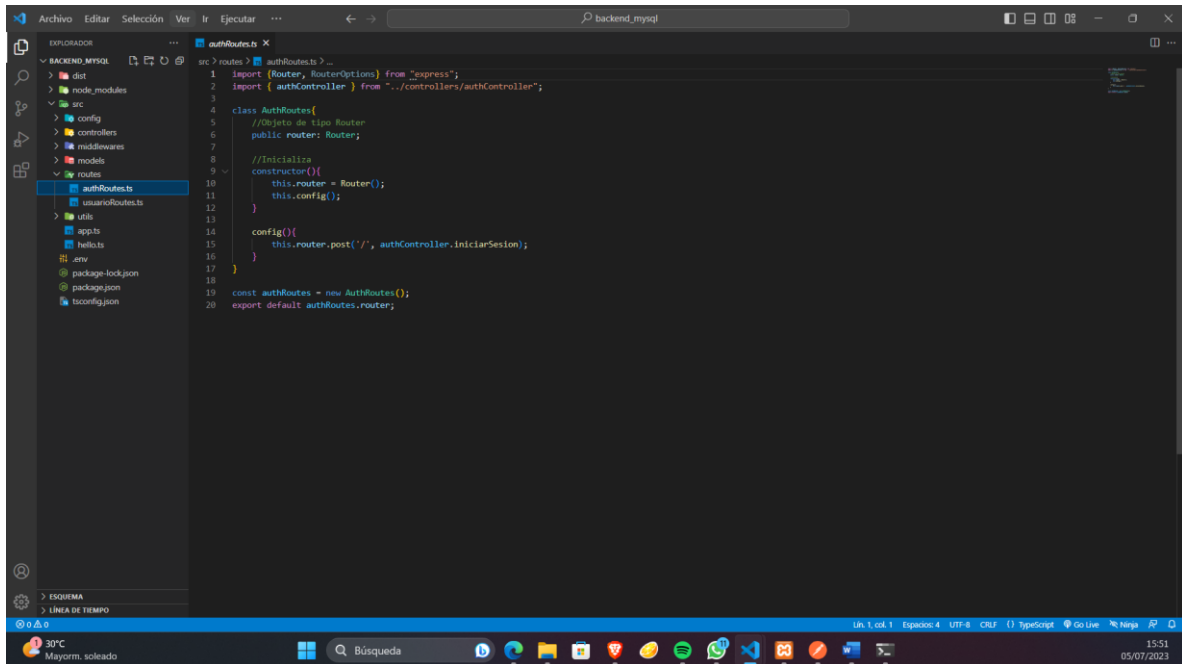


```

src > index.ts
1  import express, { Application } from 'express';
2  import morgan from 'morgan';
3  import cors from 'cors';
4  import bodyParser from 'body-parser';
5
6  import authRoutes from './routes/authRoutes';
7  import usuarioRoutes from './routes/usuarioRoutes';
8
9  class Server {
10    private app: Application;
11    constructor() {
12      this.app = express();
13      this.config();
14      this.routes();
15      this.app.listen(this.app.get('port'), () => {
16        console.log("Server on port", this.app.get('port'));
17      });
18    }
19
20
21    config(): void {
22      // configuración del puerto para el servidor
23      this.app.set('port', 3000);
24
25      // muestra las peticiones en consola
26      this.app.use(morgan("dev"));
27
28      // puertos de conexión de la API
29      this.app.use(cors());
30
31      // solo se permiten peticiones en formato JSON
32      this.app.use(bodyParser.json());
33      this.app.use(bodyParser.urlencoded({ extended: false }));
34    }
35
36    // configurar rutas
37    routes() {
38      this.app.use("/", authRoutes);
39      this.app.use("/user", usuarioRoutes);
40    }
41
42  }
43
44  const server = new Server();
45

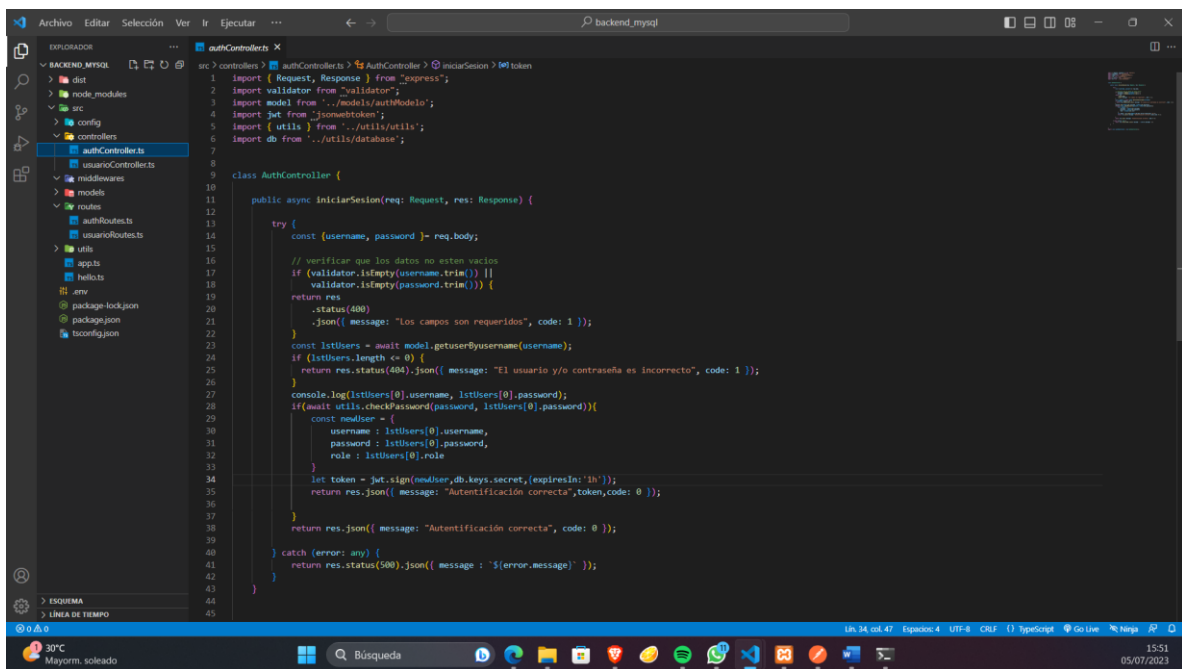
```

## Creamos nuestro archivo de rutas para definir las



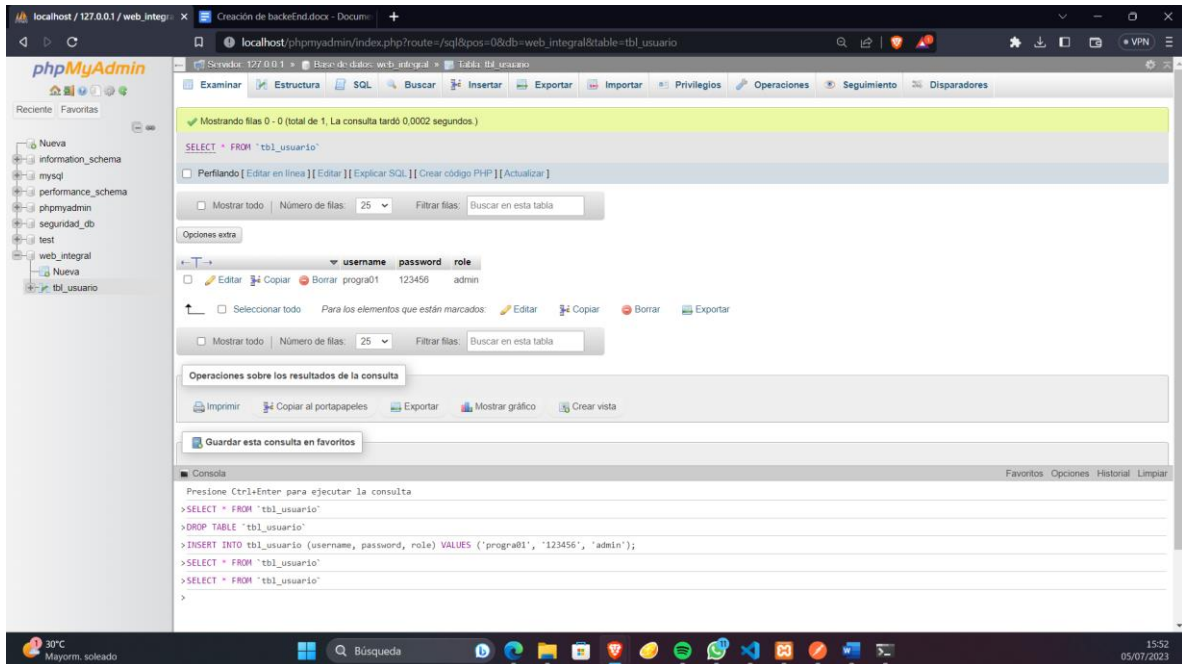
```
1 import { Router, RouterOptions } from "express";
2 import { AuthController } from "../controllers/authController";
3
4 class AuthRoutes {
5   // Objeto de tipo Router
6   public router: Router;
7
8   // Inicializa
9   constructor() {
10     this.router = Router();
11     this.config();
12   }
13
14   config() {
15     this.router.post('/', authController.iniciarSession);
16   }
17 }
18
19 const authRoutes = new AuthRoutes();
20 export default authRoutes.router;
```

## Creamos el método iniciar sesión

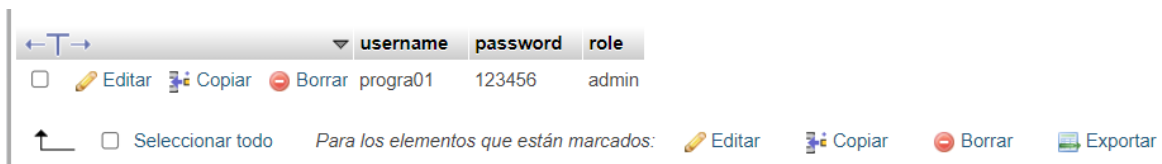


```
1 import { Request, Response } from "express";
2 import validator from "validator";
3 import model from "../models/authModel";
4 import jwt from "jsonwebtoken";
5 import { utils } from "../utils/utils";
6 import db from "../utils/database";
7
8 class AuthController {
9
10   public async iniciarSession(req: Request, res: Response) {
11     try {
12       const { username, password } = req.body;
13
14       // verificar que los datos no esten vacios
15       if (validator.isEmpty(username.trim()) || validator.isEmpty(password.trim())) {
16         return res.status(400).json({ message: "Los campos son requeridos", code: 1 });
17       }
18       const listUsers = await model.getUserByUsername(username);
19       if (listUsers.length <= 0) {
20         return res.status(404).json({ message: "El usuario y/o contraseña es incorrecto", code: 1 });
21       }
22       console.log(listUsers[0].username, listUsers[0].password);
23       if (await utils.checkPassword(password, listUsers[0].password)) {
24         const newUser = {
25           username: listUsers[0].username,
26           password: listUsers[0].password,
27           role: listUsers[0].role
28         };
29         let token = jwt.sign(newUser, db.keys.secret, { expiresIn: '1h' });
30         return res.json({ message: "Autenticación correcta", token, code: 0 });
31       }
32       return res.json({ message: "Autenticación correcta", code: 0 });
33     } catch (error: any) {
34       return res.status(500).json({ message: `${error.message}` });
35     }
36   }
37 }
38
39 export default AuthController;
```

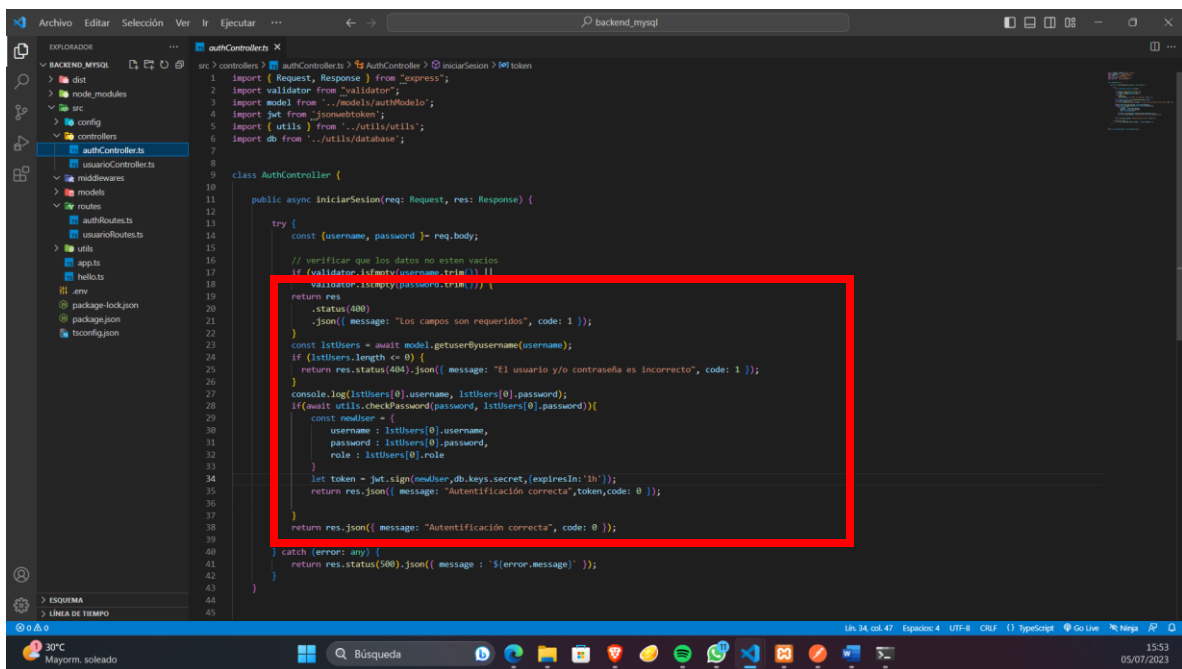
## Creamos la tabla



## Insertamos el usuario

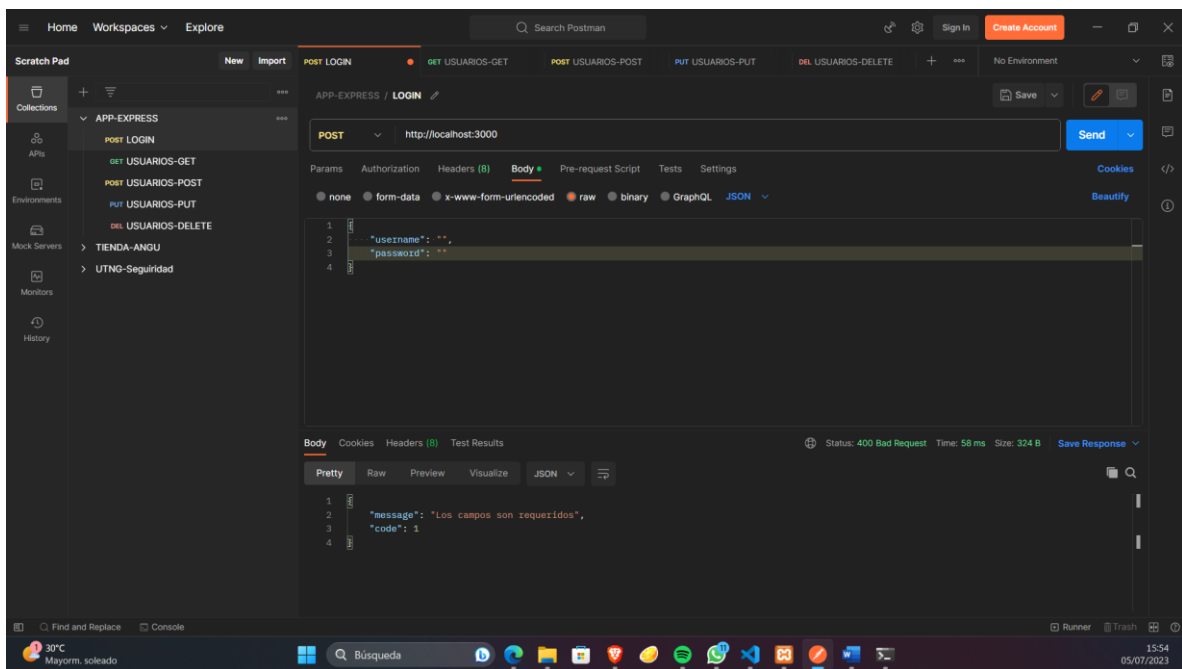


## Modificando el auth-controller para validación de entradas



```
1 import { Request, Response } from "express";
2 import validator from "validator";
3 import model from "../models/authModel";
4 import jwt from "jsonwebtoken";
5 import { utils } from "../utils/utils";
6 import db from "../utils/database";
7
8 class AuthController {
9
10   public async IniciarSesion(req: Request, res: Response) {
11     try {
12       const { username, password } = req.body;
13
14       // verificar que los datos no esten vacios
15       if (!validator.isEmail(username.trim())) {
16         return res
17           .status(400)
18           .json({ message: "Los campos son requeridos", code: 1 });
19       }
20       const listUsers = await model.getUserByUsername(username);
21       if (listUsers.length <= 0) {
22         return res.status(404).json({ message: "El usuario y/o contraseña es incorrecto", code: 1 });
23       }
24       console.log(listUsers[0].username, listUsers[0].password);
25       if (await utils.checkPassword(password, listUsers[0].password)) {
26         const newUser = {
27           username: listUsers[0].username,
28           password: listUsers[0].password,
29           role: listUsers[0].role
30         };
31         let token = jwt.sign(newUser, db.keys.secret, { expiresIn: '1h' });
32         return res.json({ message: "Autenticación correcta", token, code: 0 });
33       }
34       return res.json({ message: "Autenticación correcta", code: 0 });
35     } catch (error: any) {
36       return res.status(500).json({ message: `${error.message}` });
37     }
38   }
39 }
```

## Validación de entrada



POST LOGIN

POST http://localhost:3000

Body

```
1 {
2   "username": "",
3   "password": ""
4 }
```

Status: 400 Bad Request Time: 58 ms Size: 324 B Save Response

Body

```
1 {
2   "message": "Los campos son requeridos",
3   "code": 1
4 }
```

## Agregando Autenticación y verificando si existe

```

AuthController > iniciarSession > token
public async iniciarSession(req: Request, res: Response) {
  try {
    const {username, password} = req.body;

    // verificar que los datos no esten vacios
    if (validator.isEmpty(username.trim()) ||
        validator.isEmpty(password.trim())) {
      return res
        .status(400)
        .json({ message: "Los campos son requeridos", code: 1 });
    }

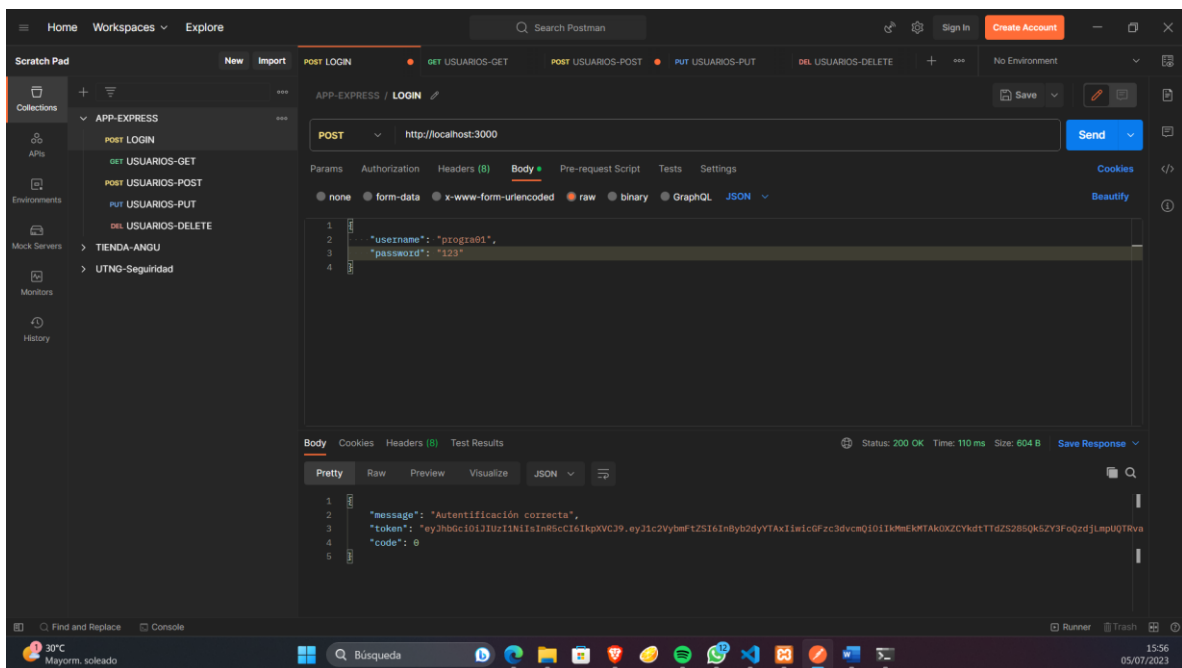
    const lstUsers = await model.getUserByusername(username);
    if (lstUsers.length <= 0) {
      return res.status(404).json({ message: "El usuario y/o contraseña es incorrecto", code: 1 });
    }

    console.log(lstUsers[0].username, lstUsers[0].password);
    if (await utils.checkPassword(password, lstUsers[0].password)){
      const newUser = {
        username : lstUsers[0].username,
        password : lstUsers[0].password,
        role : lstUsers[0].role
      }

      let token = jwt.sign(newUser,db.keys.secret,{expiresIn:'1h'});
      return res.json({ message: "Autenticación correcta",token,code: 0 });
    }
  }
}

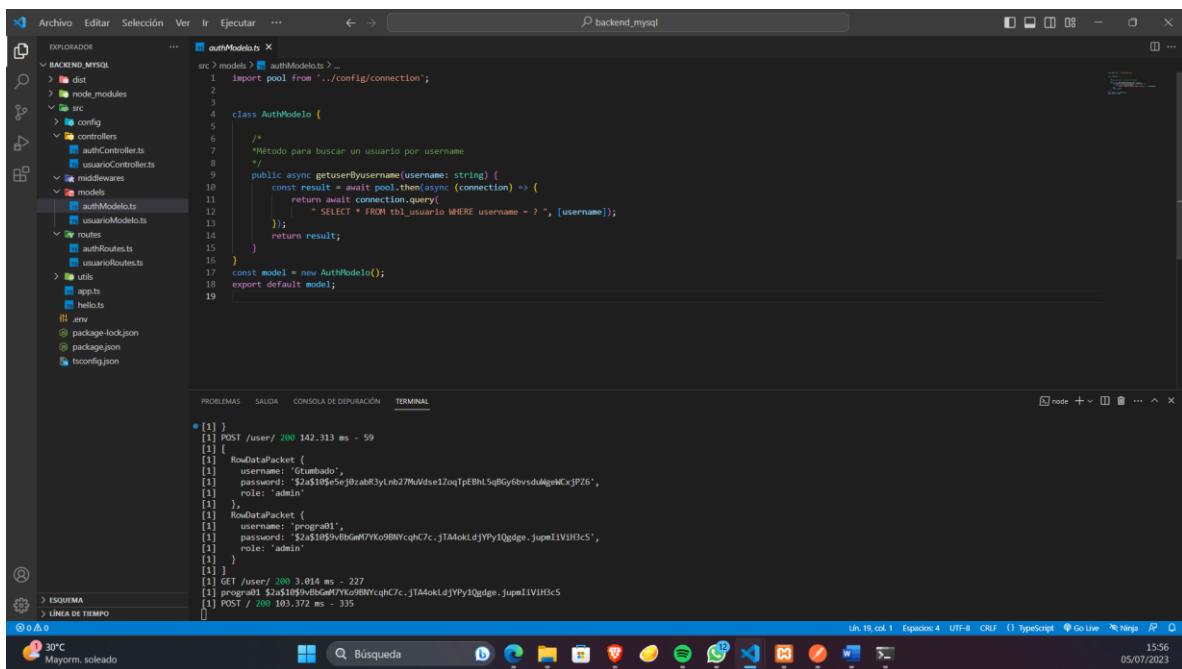
```

## Verificación de inicio de sesión

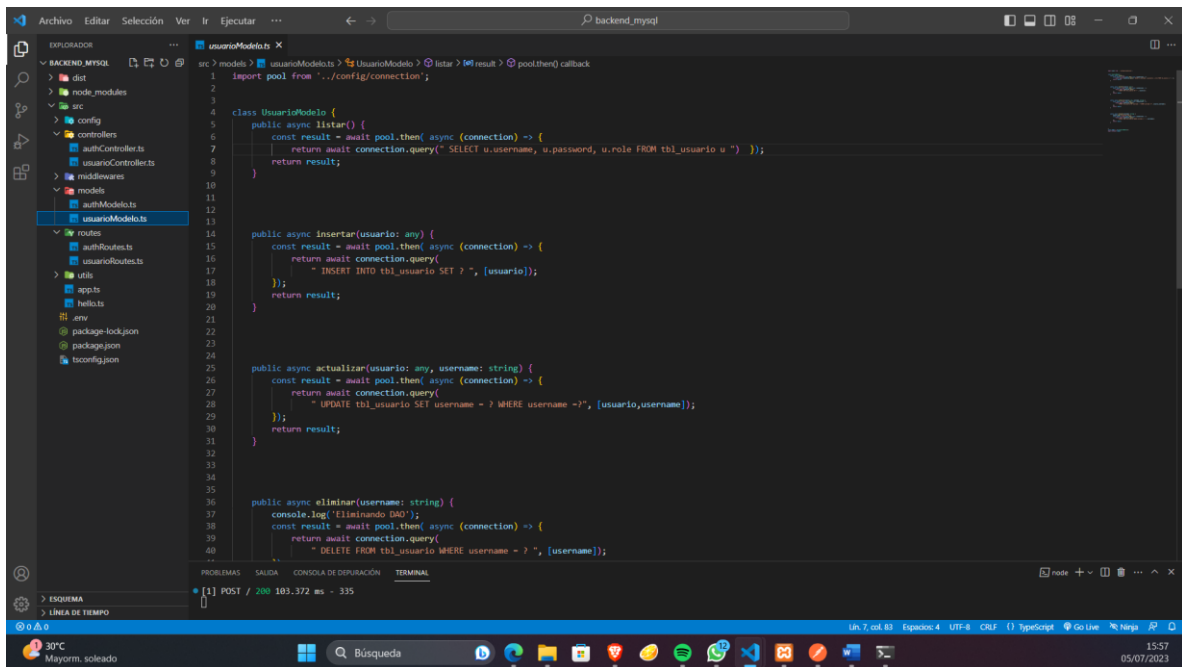


## Modificando la consulta para que valide el username y el password





## Creamos el user modelo



## Agregando lógica en usuarios

### Listar

```
public async listar() {  
    const result = await pool.then( async (connection) => {  
        return await connection.query(" SELECT u.username, u.password, u.role FROM tbl_usuario u ");  
    });  
    return result;  
}
```

### Insertar

```
public async insertar(usuario: any) {  
    const result = await pool.then( async (connection) => {  
        return await connection.query(  
            " INSERT INTO tbl_usuario SET ? ", [usuario]);  
        });  
    return result;  
}
```

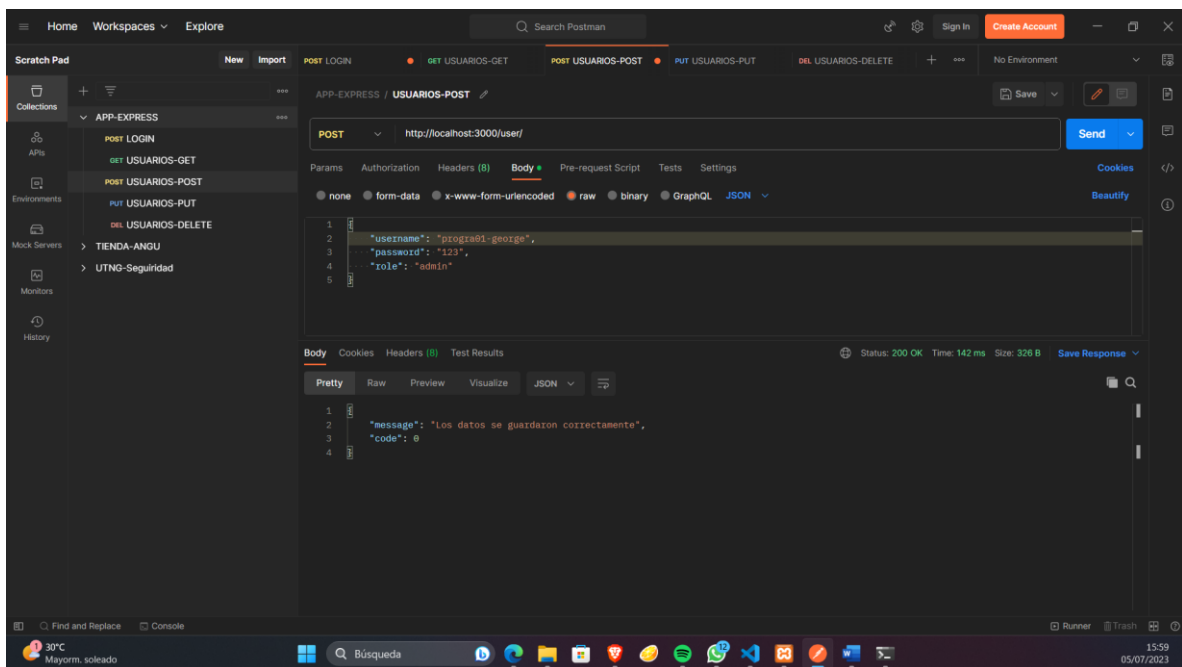
### Actualizar

```
public async actualizar(usuario: any, username: string) {  
    const result = await pool.then( async (connection) => {  
        return await connection.query(  
            " UPDATE tbl_usuario SET username = ? WHERE username =?", [usuario,username]);  
        });  
    return result;  
}
```

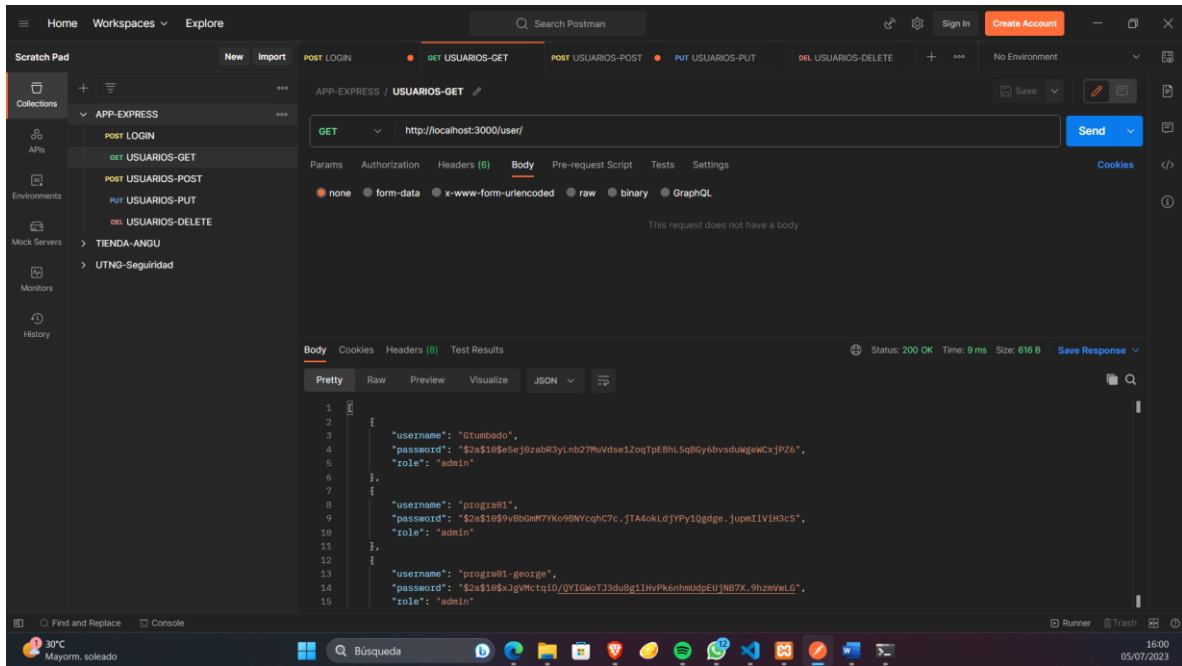
### Eliminar

```
public async eliminar(username: string) {  
  console.log('Eliminando DAO');  
  const result = await pool.then( async (connection) => {  
    return await connection.query(  
      " DELETE FROM tbl_usuario WHERE username = ? ", [username]);  
    });  
  return result;  
}
```

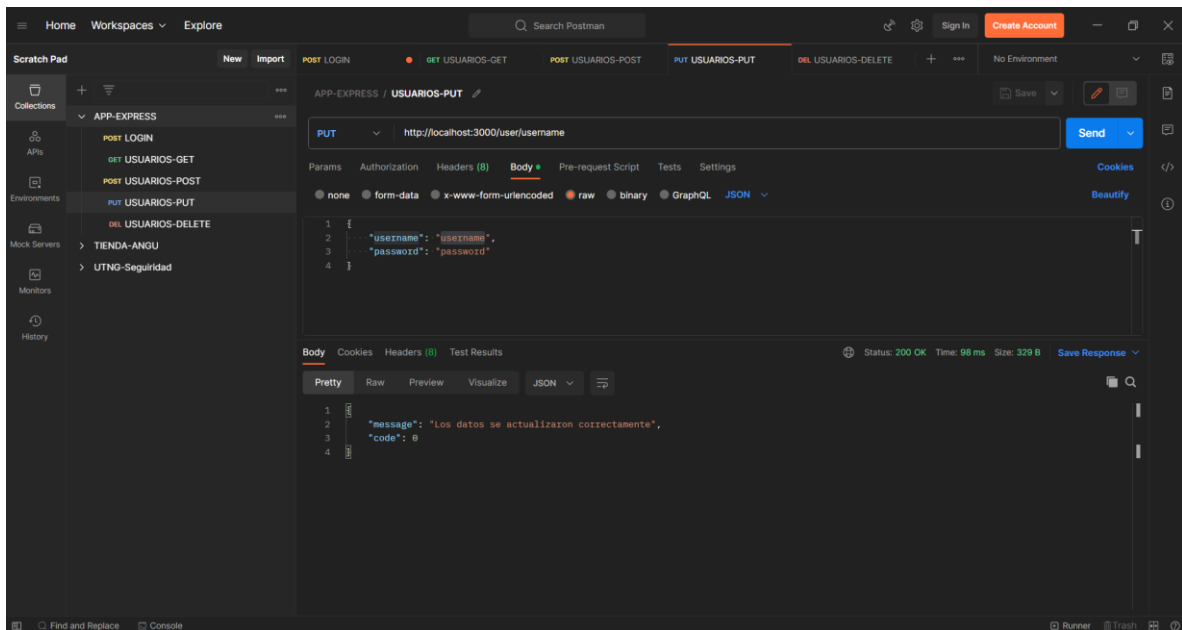
## Verificando en guardar



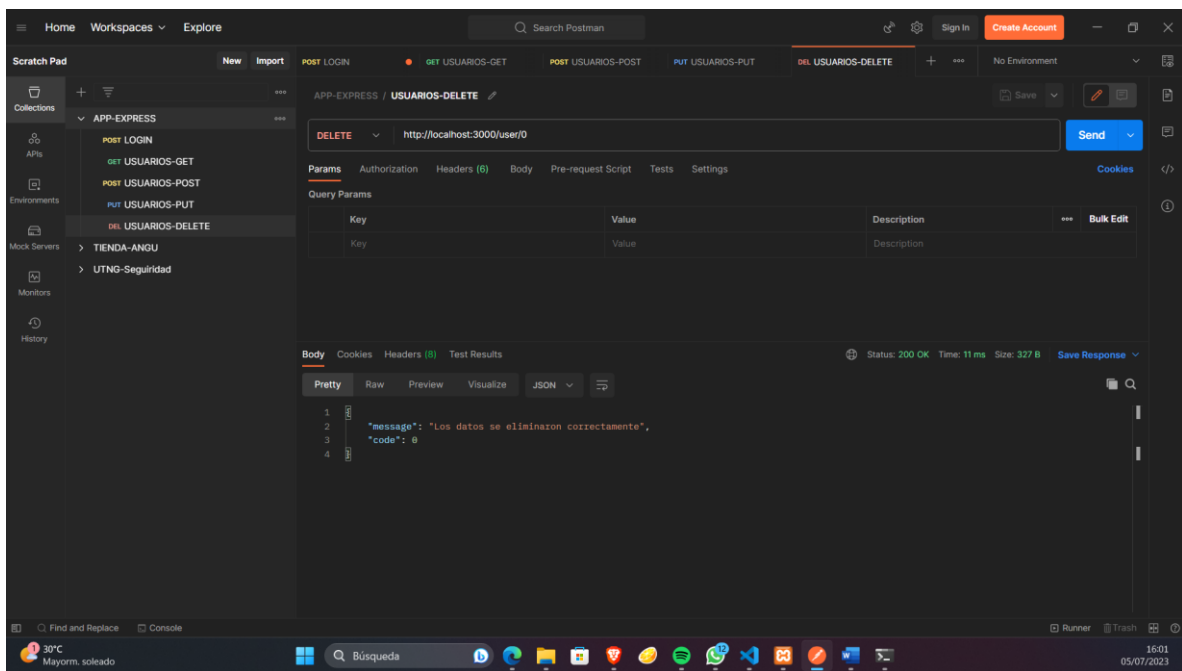
## Verificando Listar



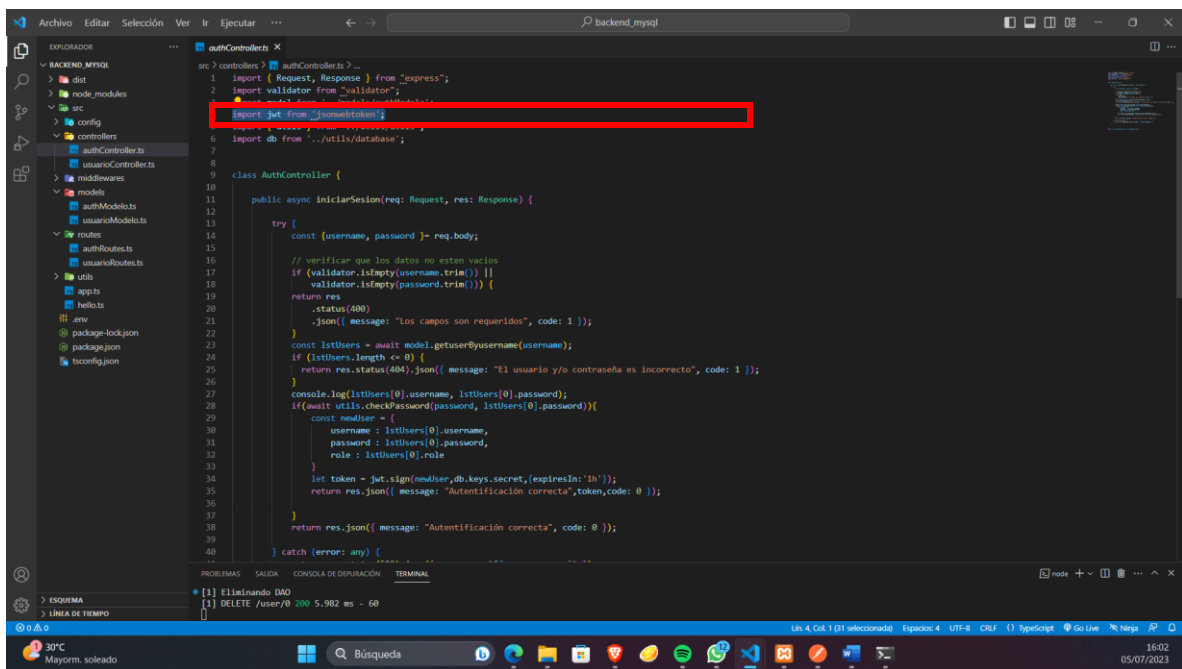
## Verificando actualizar



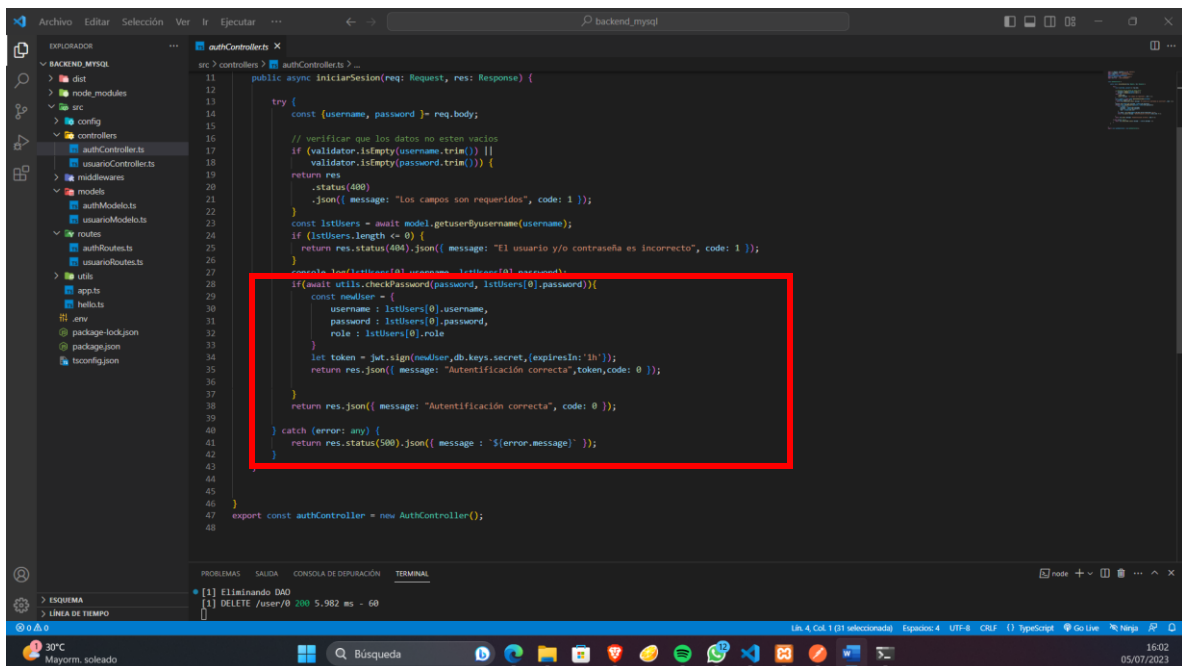
## Verificando eliminar



Json web token



Enviamos el token



Probando con Post man

