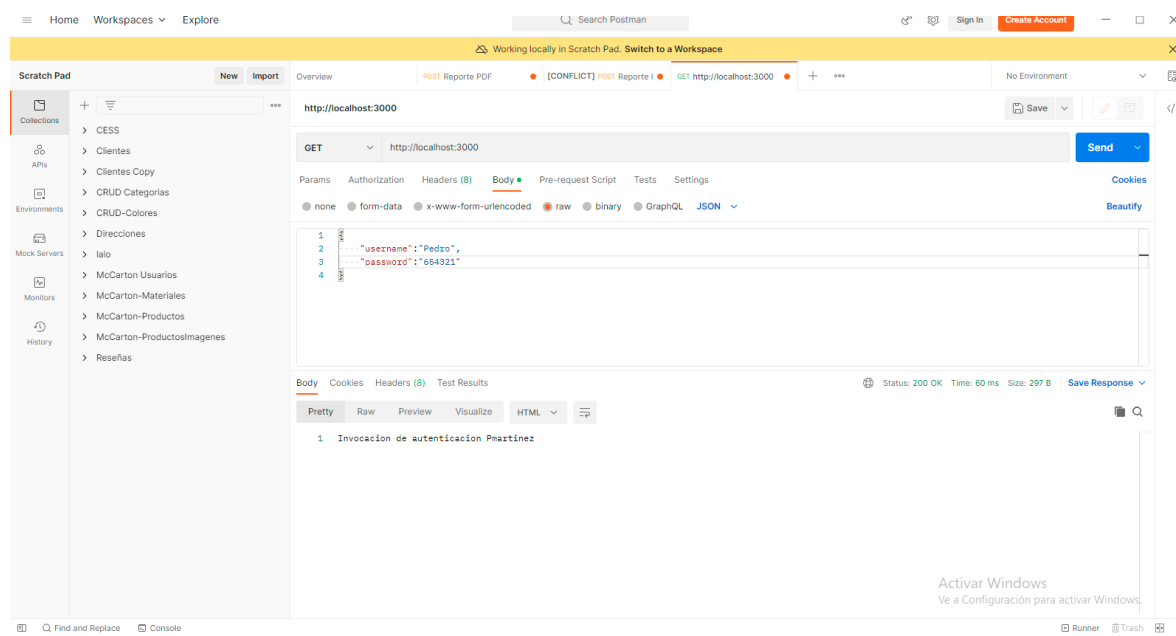
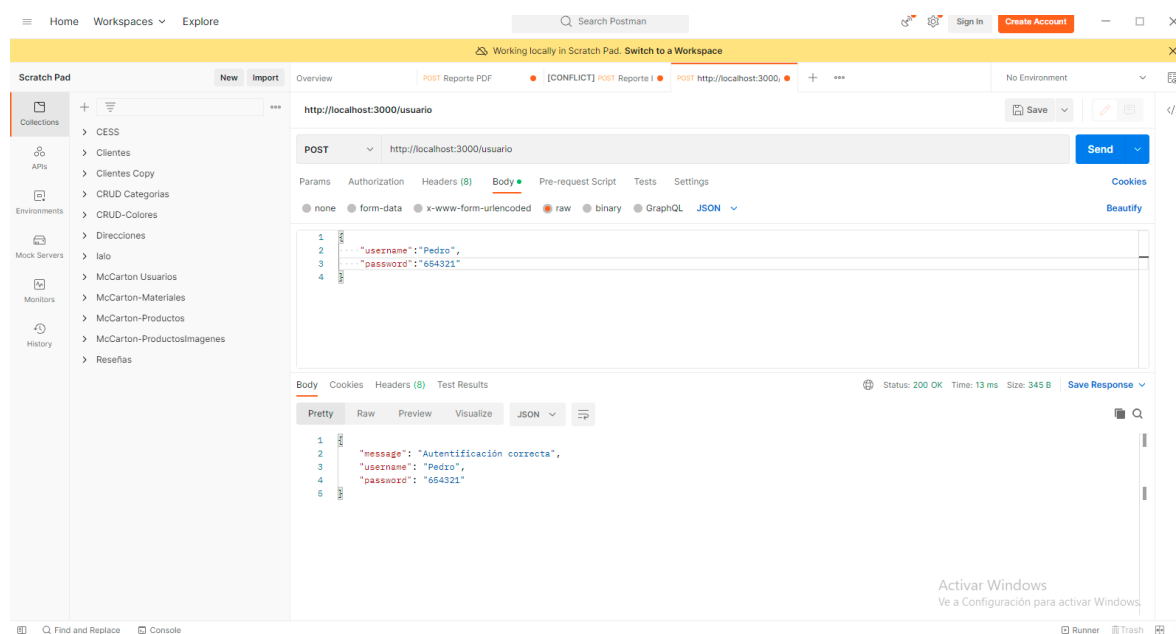


Autenticación en el aplicativo



Autenticación base de datos



HomeWorkspacesExplore

Search Postman

Sign InCreate Account

Working locally in Scratch Pad. Switch to a Workspace

Scratch PadNewImportOverviewPOST Reports PDF[CONFLICT] POST Reporte IPOST http://localhost:3000,***No Environment

Collections+***

APIs

Environments

Mock Servers

Monitors

History

CESS

Clientes

Clientes Copy

CRUD Categorías

CRUD-Colores

Direcciones

lalo

McCarton Usuarios

McCarton-Materiales

McCarton-Productos

McCarton-ProductosImágenes

Reseñas

http://localhost:3000/usuario

SaveEditCode

POSThttp://localhost:3000/usuarioSend

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettingsCookiesBeautify

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

```
1[{"username": "Pedro",
2  "password": "664321"}]
```

BodyCookiesHeaders (8)Test Results

Status: 200 OKTime: 63 msSize: 315 BSave Response

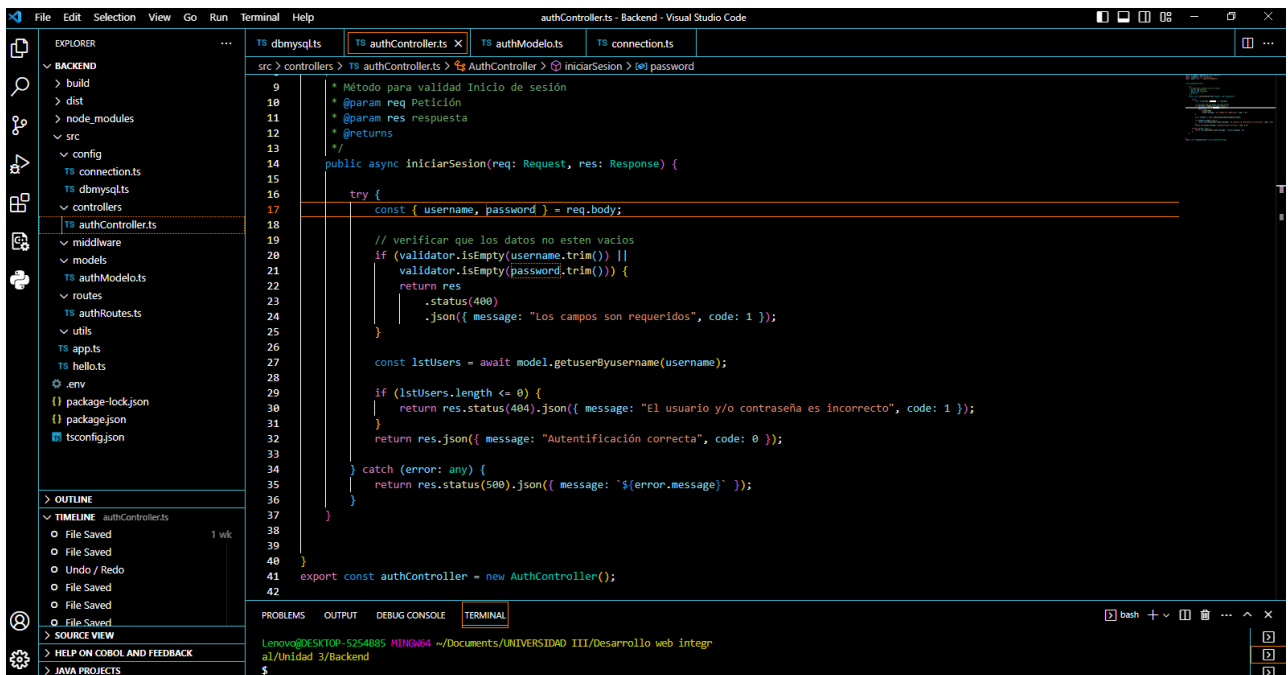
PrettyRawPreviewVisualizeJSON

```
1{"message": "Autenticación correcta",
2  "code": 0}
```

Activar Windows
Ve a Configuración para activar Windows

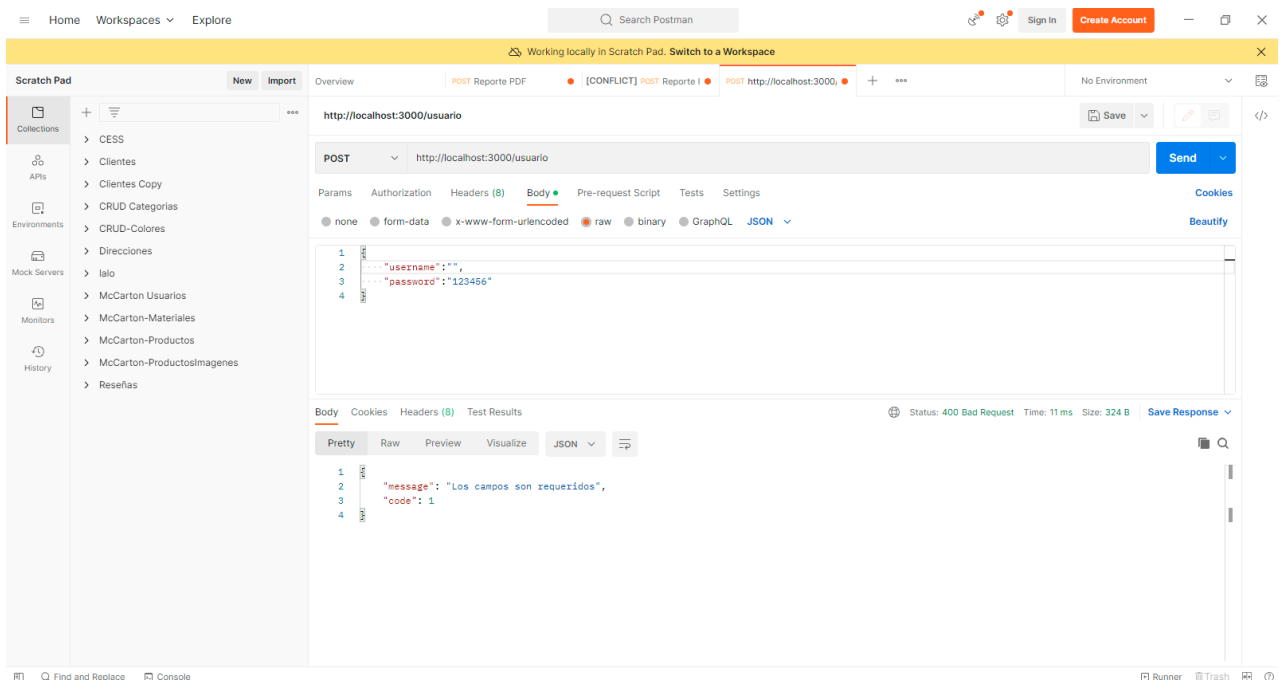
Find and ReplaceConsoleRunnerTrash

Validación de campos

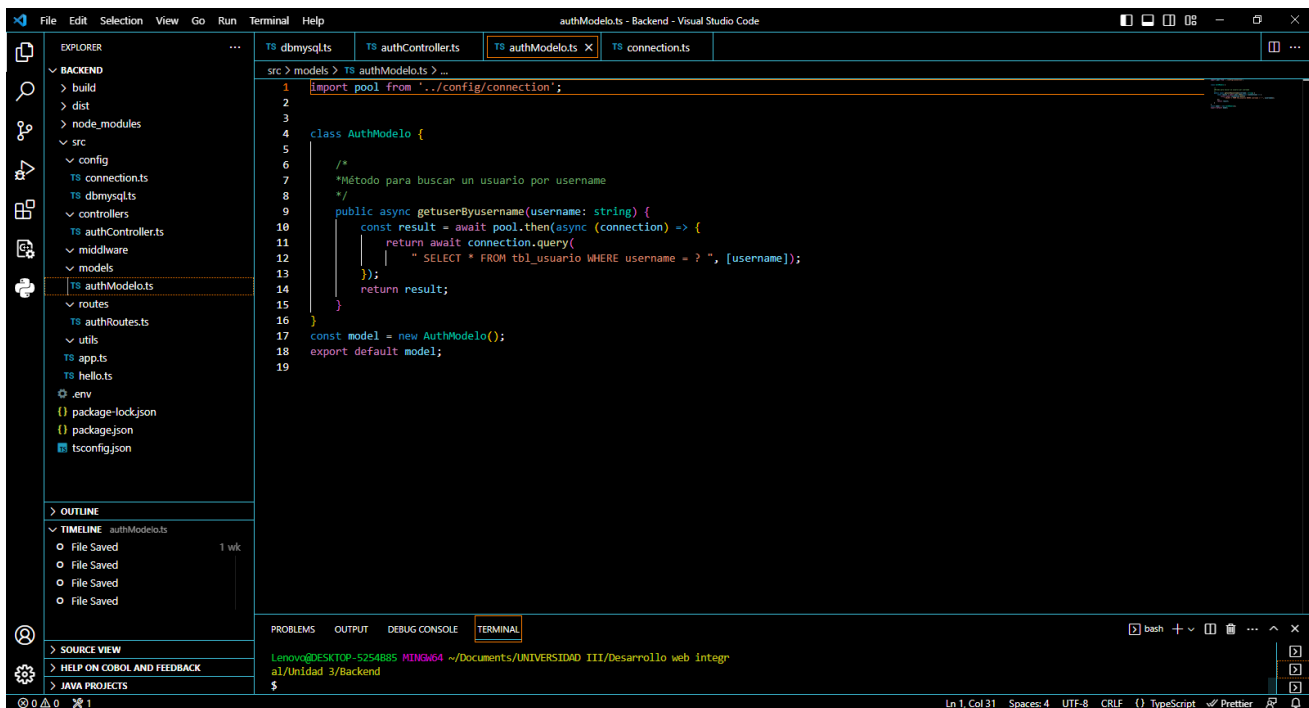


The screenshot shows the Visual Studio Code editor with a project named 'authControllers - Backend'. The Explorer sidebar on the left shows the file structure, including 'controllers' and 'models'. The main editor displays the 'authController.ts' file, which contains the 'iniciarSesion' function. This function performs validation on the 'username' and 'password' fields of the request body. It uses the 'validator' library to check if the fields are empty. If either is empty, it returns a 400 status with a message 'Los campos son requeridos'. If both are present, it checks the 'users' array in the 'model' for a matching username. If no user is found, it returns a 404 status with a message 'El usuario y/o contraseña es incorrecto'. If a user is found, it returns a 200 status with a message 'Autenticación correcta'. The function is wrapped in a try-catch block to handle any unexpected errors, returning a 500 status with the error message. The function is exported as 'authController'.

```
9  * Método para validad Inicio de sesión
10 * @param req Petición
11 * @param res respuesta
12 * @returns
13 */
14 public async iniciarSesion(req: Request, res: Response) {
15
16     try {
17         const { username, password } = req.body;
18
19         // verificar que los datos no esten vacios
20         if (validator.isEmpty(username.trim()) ||
21             validator.isEmpty(password.trim())) {
22             return res
23                 .status(400)
24                 .json({ message: "Los campos son requeridos", code: 1 });
25         }
26
27         const lstUsers = await model.getUserByUsername(username);
28
29         if (lstUsers.length <= 0) {
30             return res.status(404).json({ message: "El usuario y/o contraseña es incorrecto", code: 1 });
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
40
41 export const authController = new AuthController();
42
```



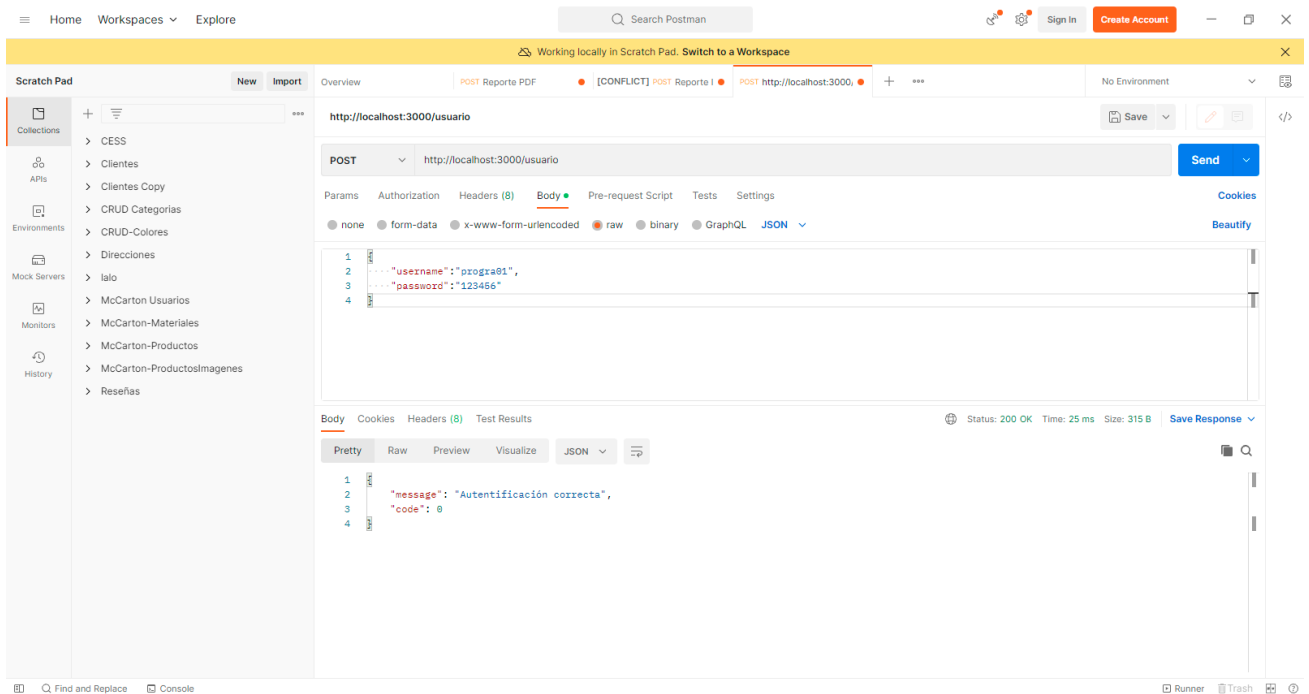
Método para autenticar al usuario



The screenshot shows the Visual Studio Code editor with a project named 'authModel.ts - Backend - Visual Studio Code'. The Explorer sidebar on the left shows the project structure, including folders like 'build', 'dist', 'node_modules', 'src', 'config', 'controllers', 'models', 'routes', 'utils', and 'app.ts'. The 'models' folder is expanded, showing 'authModel.ts' selected. The main editor displays the code for 'AuthModel' in 'authModel.ts'.

```
1 import pool from '../config/connection';
2
3
4 class AuthModel {
5
6   /*
7    *Método para buscar un usuario por username
8    */
9   public async getUserByUsername(username: string) {
10     const result = await pool.then(async (connection) => {
11       return await connection.query(
12         " SELECT * FROM tbl_usuario WHERE username = ? ", [username]);
13     });
14     return result;
15   }
16 }
17 const model = new AuthModel();
18 export default model;
19
```

The bottom status bar shows the file is at 'Ln 1, Col 31' with 'Spaces: 4', 'UTF-8', 'CRLF', and 'TypeScript' encoding. The bottom right corner shows the 'Prettier' formatter is active.

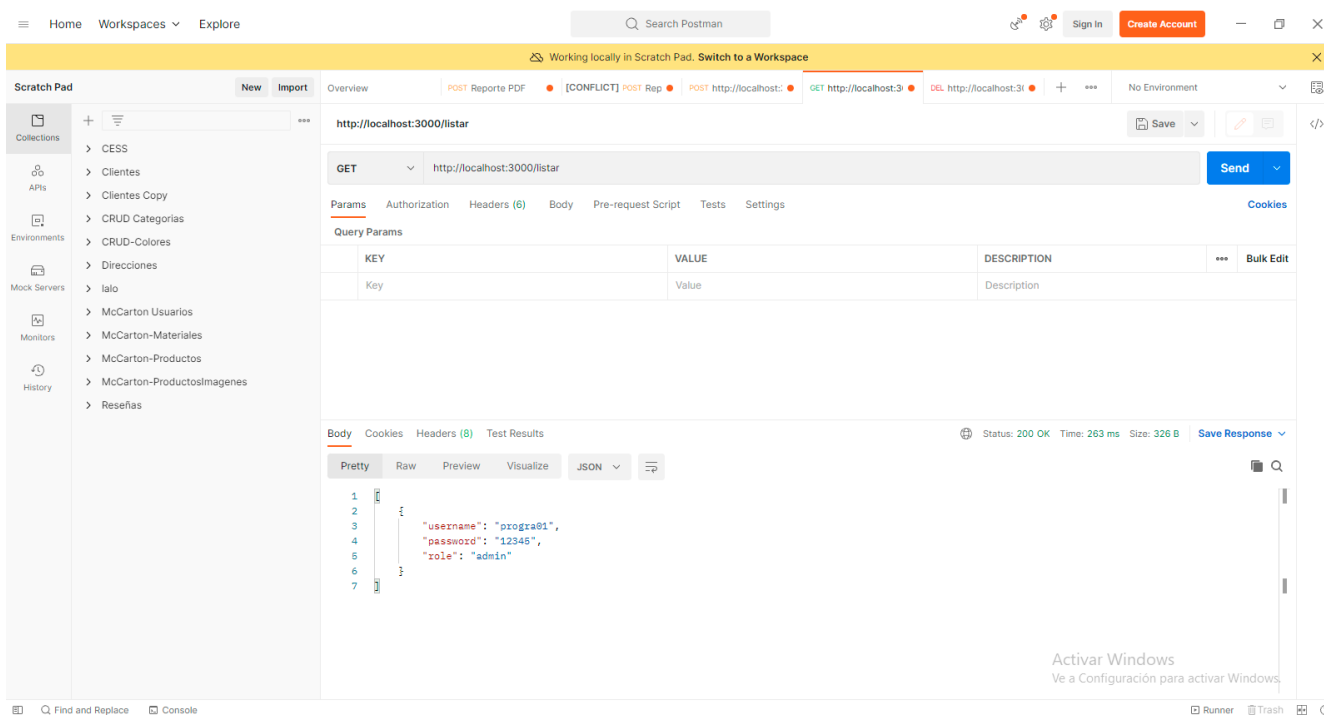


Método para listar a los usuarios

The screenshot shows the Visual Studio Code editor with the 'usuarioModelo.ts' file open. The 'listar' method is highlighted with a red box. The code is as follows:

```
1 import pool from '../config/connection';
2
3
4 class UsuarioModelo {
5   public async listar() {
6     const result = await pool.then(async (connection) => {
7       return await connection.query(
8         " SELECT u.username, u.password, u.role "
9         + " FROM tbl_usuario u " );
10    });
11    return result;
12  }
13
14   public async insertar(usuario: any) {
15     const result = await pool.then(async (connection) => {
16       return await connection.query(
17         " INSERT INTO tbl_usuario SET ? ", [usuario]);
18     });
19     return result;
20   }
21
22   public async actualizar(usuario: any, username: string) {
23     const result = await pool.then(async (connection) => {
24       return await connection.query(
25         " UPDATE tbl_usuario SET ? WHERE username = ?",
26         [usuario, username]);
27     });
28     return result;
29   }
30 }
```

The Explorer sidebar on the left shows the project structure, including 'models', 'controllers', 'routes', and 'utils'. The Terminal at the bottom shows the command prompt with the file saved.



Método para insertar un usuario

The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the `usuarioController.ts` file. The `insertar` method is highlighted with a red box. The `actualizar` and `eliminar` methods are also visible. The terminal at the bottom shows the command prompt and the output of the application.

```
src > models > TS usuarioModelos > UsuarioModelo > actualizar

16 public async insertar(usuario: any) {
17     const result = await pool.then(async (connection) => {
18         return await connection.query(
19             "INSERT INTO tbl_usuario SET ? ", [usuario]);
20     });
21     return result;
22 }

25 public async actualizar(usuario: any, username: string) {
26     const result = await pool.then(async (connection) => {
27         return await connection.query(
28             "UPDATE tbl_usuario SET ? WHERE username = ?",
29             [usuario, username]
30         );
31     });
32     return result;
33 }

38 public async eliminar(username: string) {
39     console.log("Eliminando DAO");
40     const result = await pool.then(async (connection) => {
41         return await connection.query(
42             "DELETE FROM tbl_usuario where username = ?", [username]);
43     });
44 }
```

Terminal output:

```
Lenovo@DESKTOP-52548B5 MINGW64 ~/Documents/UNIVERSIDAD III/Desarrollo web Integr
al/Unidad 3/Backend
$
History restored

Lenovo@DESKTOP-52548B5 MINGW64 ~/Documents/UNIVERSIDAD III/Desarrollo web Integr
al/Unidad 3/Backend
$
```

The screenshot shows the Postman application interface. The left sidebar displays the 'Collections' list, including 'CESS' and 'Environments'. The main workspace shows a POST request to `http://localhost:3000/insertar`. The request body is a JSON object with the following data:

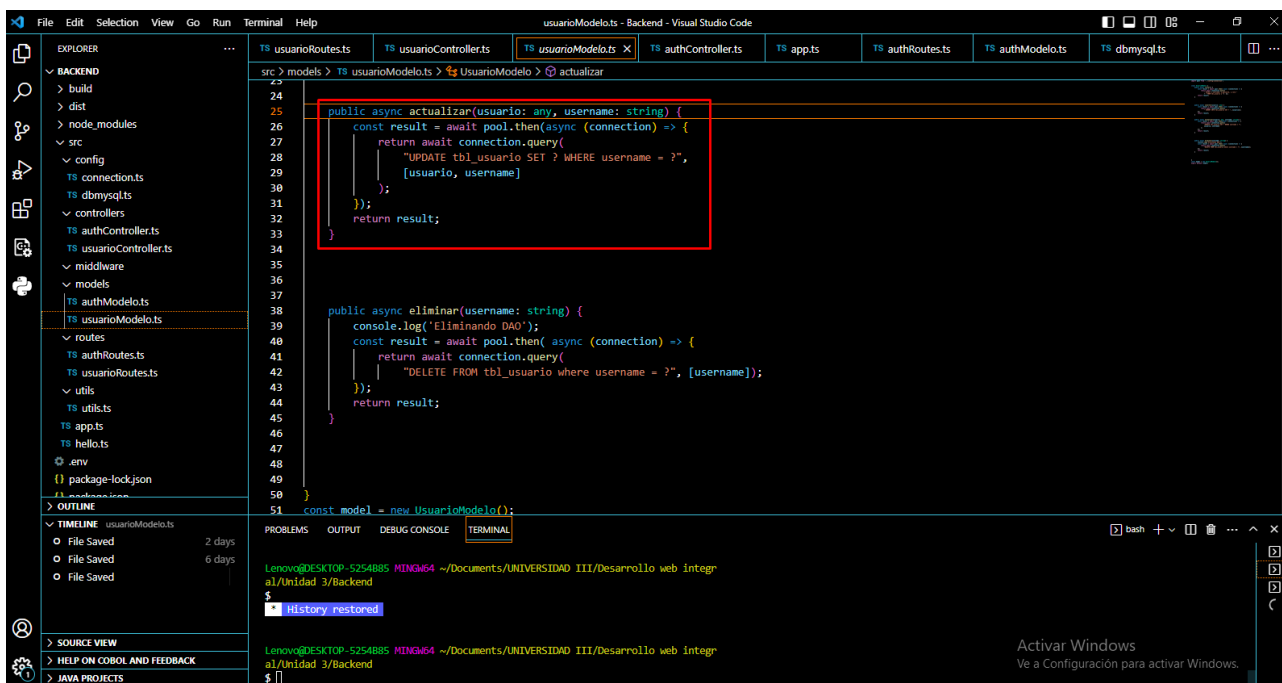
```
{
  "username": "Pedro",
  "password": "12345",
  "role": "editor"
}
```

The response is a JSON object with the following data:

```
{
  "message": "Los datos se guardaron correctamente",
  "code": 0
}
```

The status bar at the bottom indicates a 200 OK response with a time of 188 ms and a size of 326 B.

Método para actualizar a un usuario

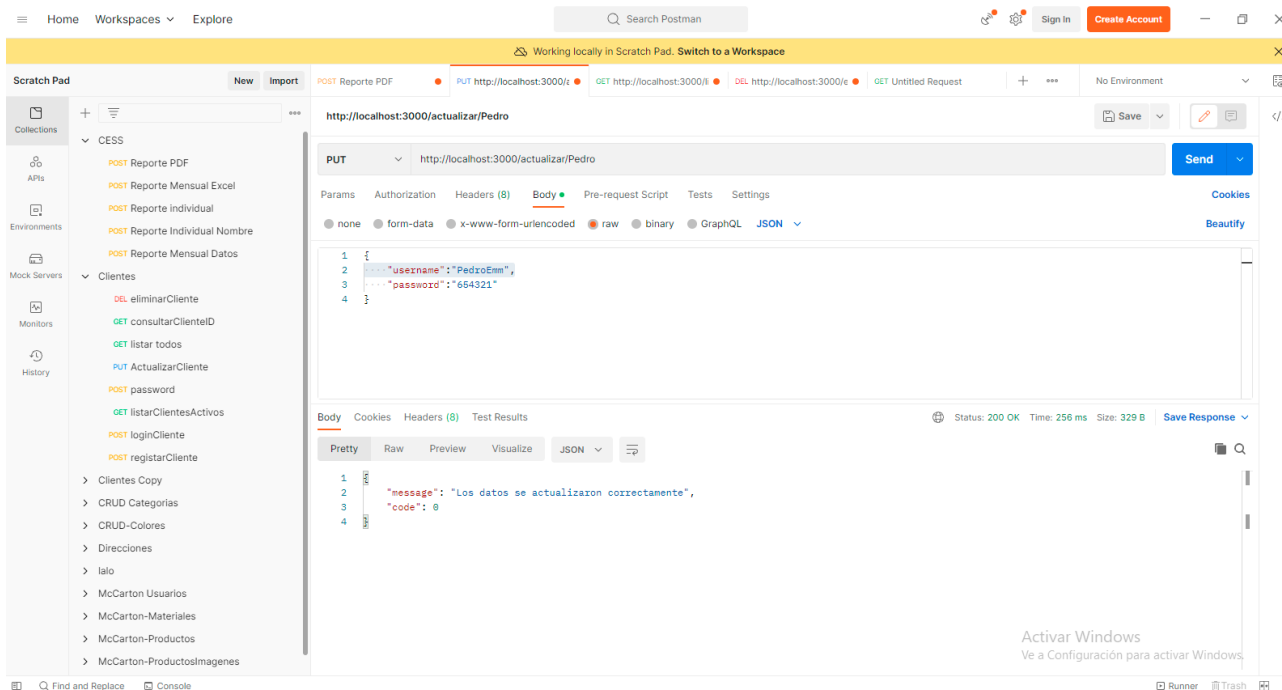


The screenshot shows the Visual Studio Code editor with the 'usuarioModelo.ts' file open. The 'actualizar' method is highlighted with a red box. The code is as follows:

```
24 public async actualizar(usuario: any, username: string) {
25     const result = await pool.then(async (connection) => {
26         return await connection.query(
27             "UPDATE tbl_usuario SET ? WHERE username = ?",
28             [usuario, username]
29         );
30     });
31     return result;
32 }
33
34
35
36
37
38 public async eliminar(username: string) {
39     console.log('Eliminando DAO');
40     const result = await pool.then(async (connection) => {
41         return await connection.query(
42             "DELETE FROM tbl_usuario where username = ?", [username]);
43     });
44     return result;
45 }
46
47
48
49
50 }
51 const model = new UsuarioModelo();
```

The left sidebar shows the Explorer view with the project structure. The bottom status bar shows the terminal output:

```
Lenovo@DESKTOP-5254B85 MINGW64 ~/Documents/UNIVERSIDAD III/Desarrollo web integr
al/Unidad 3/Backend
$ history restored
Lenovo@DESKTOP-5254B85 MINGW64 ~/Documents/UNIVERSIDAD III/Desarrollo web integr
al/Unidad 3/Backend
$
```



The screenshot shows the Postman interface with a PUT request to 'http://localhost:3000/actualizar/Pedro'. The request body is a JSON object:

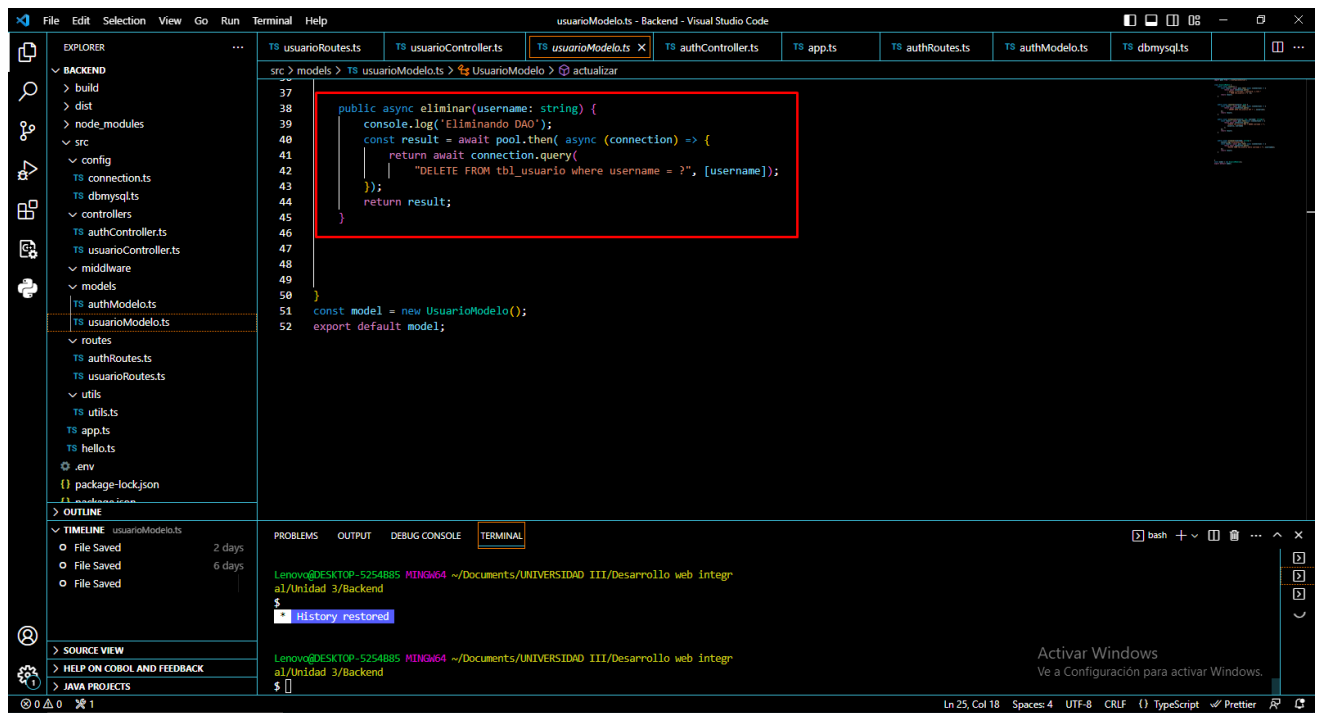
```
{
  "username": "PedroEm",
  "password": "654321"
}
```

The response is a JSON object:

```
{
  "message": "Los datos se actualizaron correctamente",
  "code": 0
}
```

The status bar shows 'Status: 200 OK', 'Time: 256 ms', and 'Size: 329 B'.

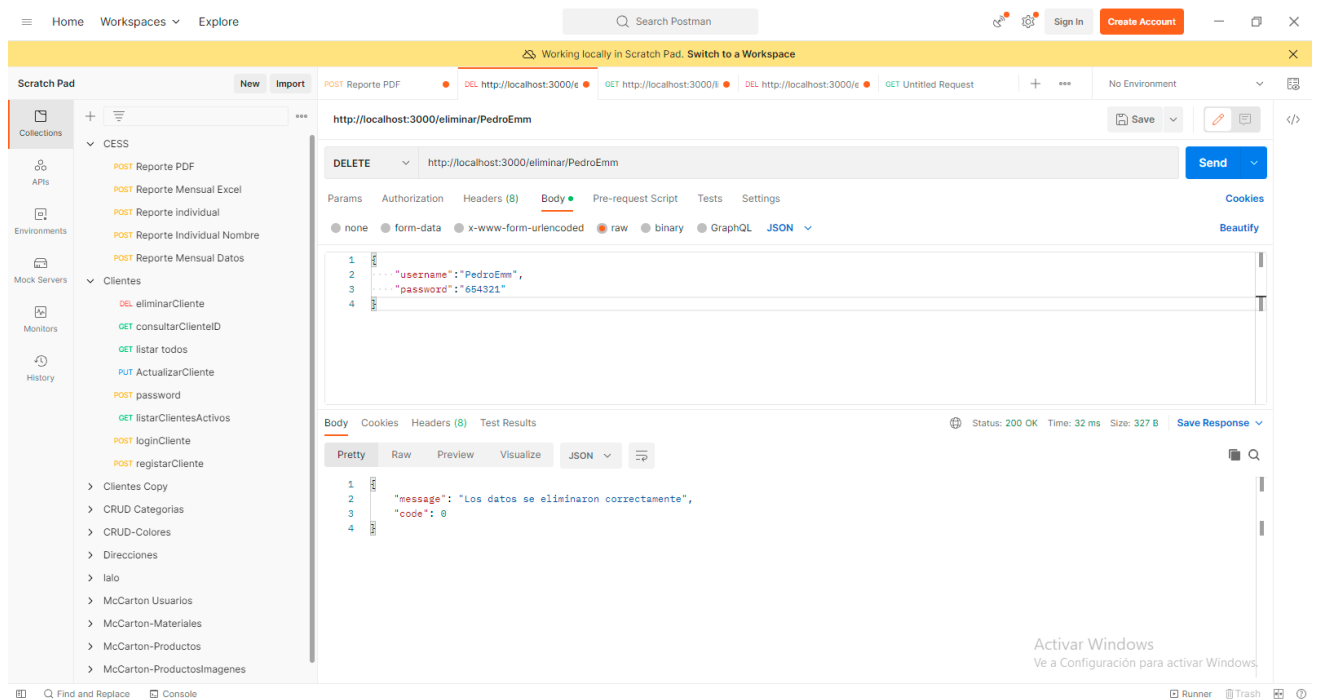
Método para eliminar un usuario



The screenshot shows the Visual Studio Code editor with the file `usuarioModelo.ts` open. The `eliminar` method is implemented as follows:

```
37  
38  
39 public async eliminar(username: string) {  
40     console.log("Eliminando DAO");  
41     const result = await pool.then( async (connection) => {  
42         return await connection.query(  
43             "DELETE FROM tbl_usuario where username = ?", [username]);  
44         });  
45     return result;  
46 }  
47  
48  
49 }  
50  
51 const model = new UsuarioModelo();  
52 export default model;
```

The code is highlighted with a red box. The terminal at the bottom shows the command `History restored` and the status bar indicates the file is saved.



The screenshot shows the Postman application with a DELETE request to `http://localhost:3000/eliminar/PedroEmm`. The request body is a JSON object:

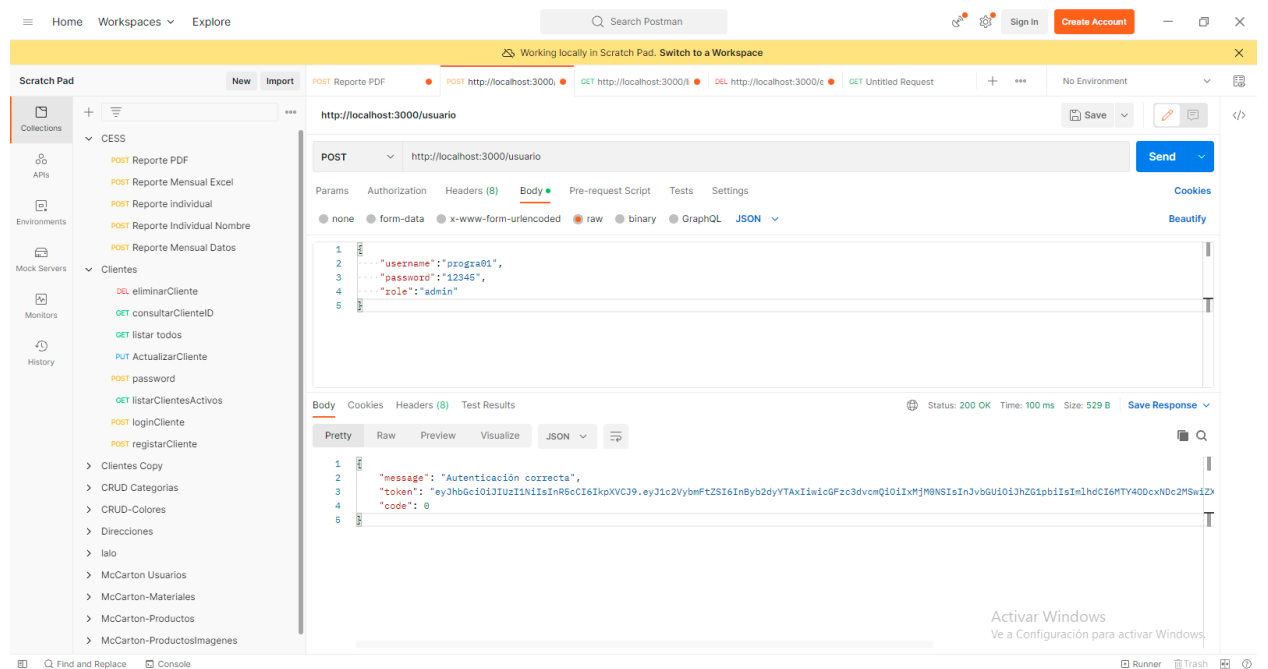
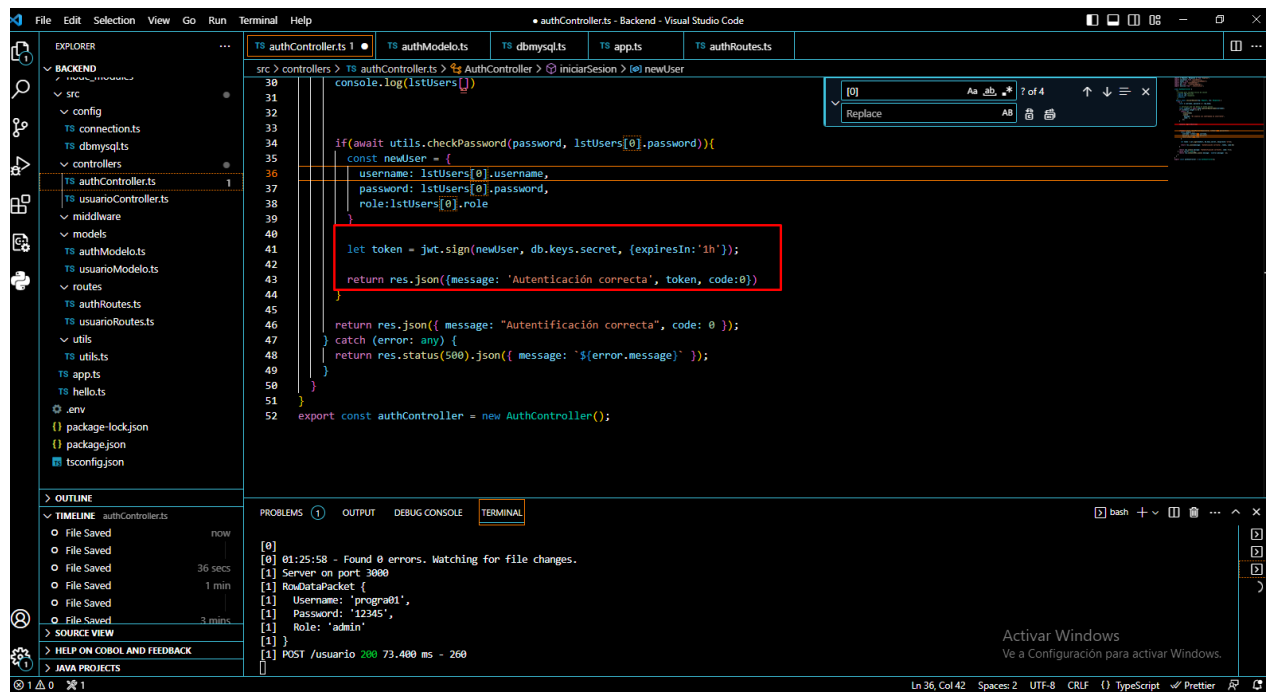
```
1 {  
2   "username": "PedroEmm",  
3   "password": "654321"  
4 }
```

The response status is 200 OK, and the response body is:

```
1 {  
2   "message": "Los datos se eliminaron correctamente",  
3   "code": 0  
4 }
```

The interface includes a sidebar with collections, a top bar with search and account options, and a bottom bar with find and replace, console, and runner tools.

Implementación del JWT



Algorithm	HS256	▼
-----------	-------	---

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InByb2dyYTAxIiwic2FzIjpc3dvc2MqOiIjXmM0NSIsInJvbGUiOiJhZG1pbSI6Im1hZG1MTY0dC3dNdc2MSwiZXhwIjoxNjg4NzE4MzYxLmQKE0Xex06G8AIxwJMW2Bu62FVaIHnuXU9bPn--dcyc

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "username": "progra01",
  "password": "12345",
  "role": "admin",
  "iat": 1688714761,
  "exp": 1688718361
}
```

VERIFY SIGNATURE

```

HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),

```

Activar Windows
Ve a Configuración para activar Windows.