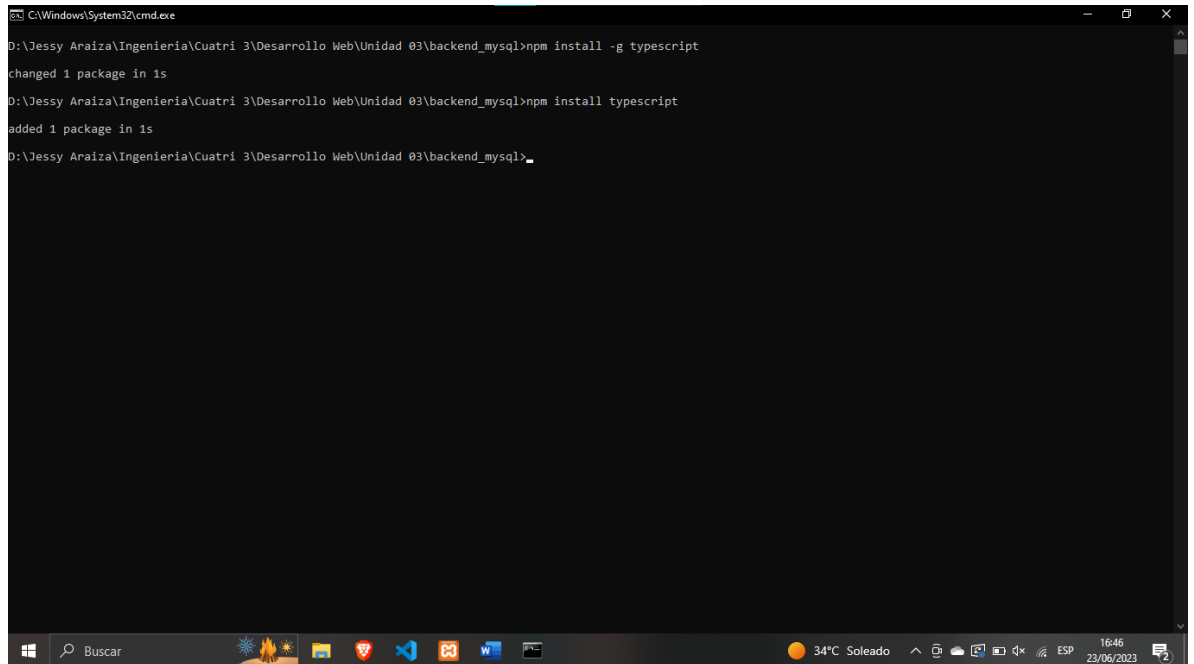


# BackEnd con Express

Para mayor referencia véase [TypeScriptLand](#)

## Instrucciones

- Instalar [TypeScript](#)  
`npm install -g typescript`

A screenshot of a Windows Command Prompt window. The title bar reads 'C:\Windows\System32\cmd.exe'. The command prompt shows the following sequence of commands and output: 1. Command: 'D:\Jessy Araiza\Ingenieria\Cuatri 3\Desarrollo Web\Unidad 03\backend\_mysql>npm install -g typescript'. Output: 'changed 1 package in 1s'. 2. Command: 'D:\Jessy Araiza\Ingenieria\Cuatri 3\Desarrollo Web\Unidad 03\backend\_mysql>npm install typescript'. Output: 'added 1 package in 1s'. 3. The prompt ends with 'D:\Jessy Araiza\Ingenieria\Cuatri 3\Desarrollo Web\Unidad 03\backend\_mysql>'. The Windows taskbar is visible at the bottom, showing the search bar, task view, and several application icons. The system tray on the right shows the temperature as 34°C, the location as Soleado, and the time as 16:46 on 23/06/2023.

- Checar la versión de TypeScript  
`tsc -v`

## Creación de Backend con Node Express

```
C:\WINDOWS\system32\cmd.exe
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>tsnc -v
Version 5.1.3
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>_
```

- Ubicarse en una carpeta de trabajo para la creación del BackeEnd con NodeJs y Express
- Crear una carpeta backend\_mysql

`mkdir backend_mysql`

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19045.3086]
(c) Microsoft Corporation. Todos los derechos reservados.
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03>mkdir backend_mysql
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03>cd backend_mysql
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>_
```

- Cambiarse a carpeta backend\_mysql recién creada,

`cd backend`

## Creación de Backend con Node Express

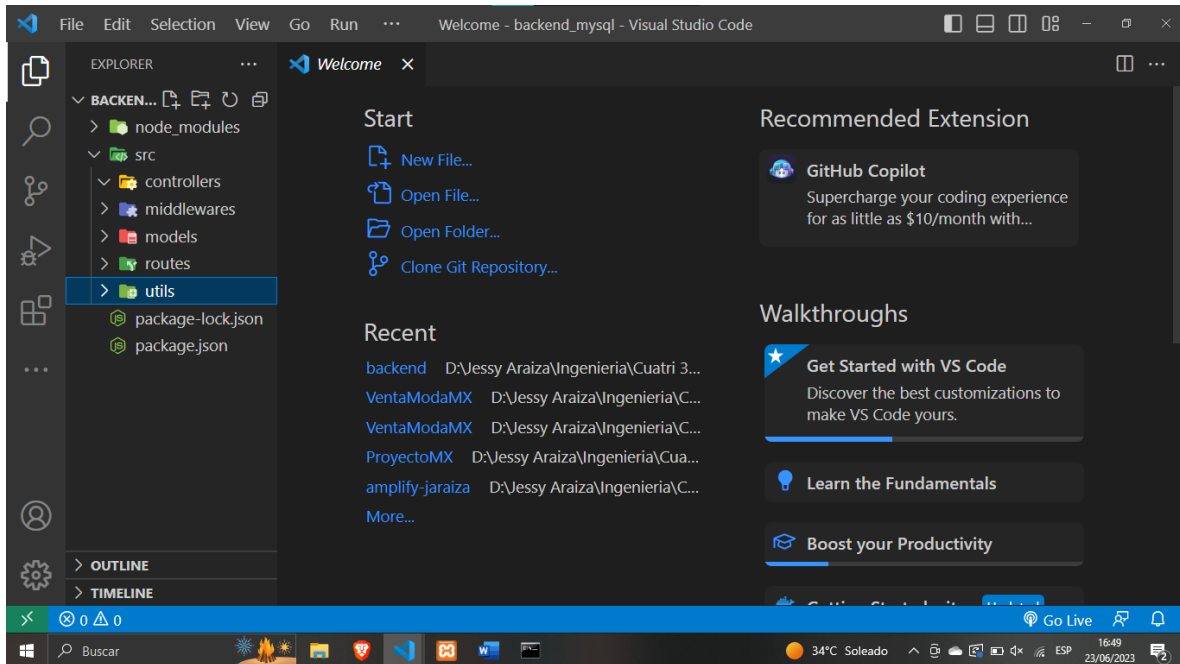
```
C:\WINDOWS\system32\cmd.exe
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>tsc -v
Version 5.1.3
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>
```

- Dentro de la carpeta ejecutar el siguiente comando para creación de archivo package.json

`npm init -y`

```
C:\WINDOWS\system32\cmd.exe
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>tsc -v
Version 5.1.3
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>npm init -y
Wrote to D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql\package.json:
{
  "dependencies": {
    "typescript": "^5.1.3"
  },
  "name": "backend_mysql",
  "version": "1.0.0",
  "main": "index.js",
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>
```

- Crear la estructura de carpetas de nuestra aplicación



- Para una mayor referencia sobre manejo [TSC](#)
- Para una mayor referencia sobre configuración con [Visual Studio Code](#)
- Crear un nuevo **tsconfig.ts**

**tsc --init**

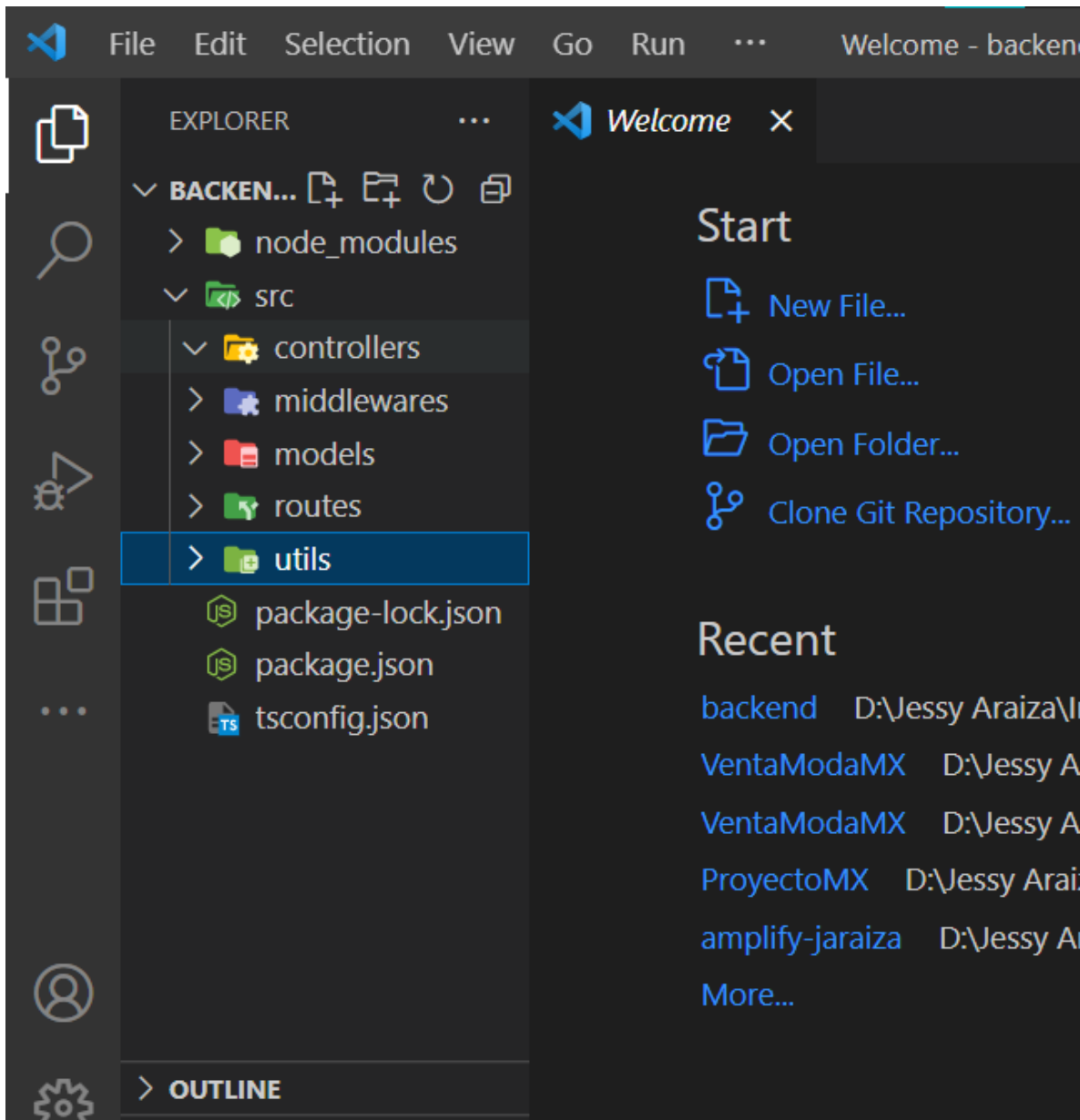
```
D:\Jessy Araiza\Ingenieria\Cuatrí 3\Desarrollo Web\Unidad 03\backend_mysql>
D:\Jessy Araiza\Ingenieria\Cuatrí 3\Desarrollo Web\Unidad 03\backend_mysql>tsc --init

Created a new tsconfig.json with:

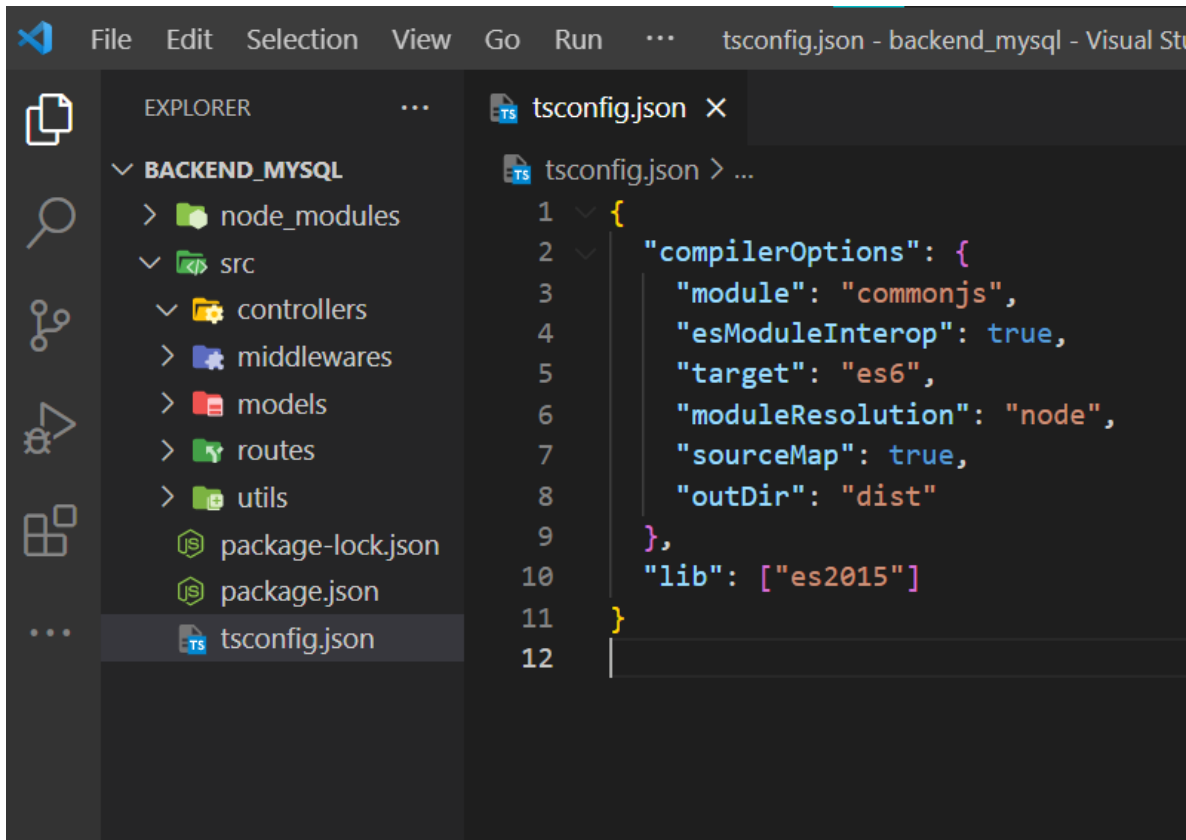
  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true

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

D:\Jessy Araiza\Ingenieria\Cuatrí 3\Desarrollo Web\Unidad 03\backend_mysql>
```



- Reemplazar el contenido del archivo por las siguientes líneas de configuración



## Instalar dependencias

- Escribir el siguiente comando

```
npm install promise-mysql express nodemon morgan cors crypto-js jsonwebtoken validator concurrently
dotenv body-parser validator morgan -D
```

```
You can learn more at https://aka.ms/tsconfig
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>npm install promise-mysql express nodemon morgan cors crypto-js jsonwebtoken validator concu
rrently dotenv body-parser validator morgan -D
added 158 packages, and audited 160 packages in 4m
19 packages are looking for funding
  run `npm fund` for details
3 moderate severity vulnerabilities
To address all issues, run:
  npm audit fix
Run `npm audit` for details.
D:\Jessy Araiza\Ingenieria\Cuatrì 3\Desarrollo Web\Unidad 03\backend_mysql>
```

- Modificar el archivo package.json para agregar el script build

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "build": "tsc -w"
```

```

20 },
21
22   ▶ Debug
23   "scripts": {
24     "test": "echo \"Error: no test specified\" && exit 1",
25     "build": "tsc -w"
26   },
27   "keywords": [],
28   "author": "",

```

- Ejecutar el comando

**npm run build** o a través Ctrl + shift + B

```

C:\WINDOWS\system32\cmd.exe
[16:57:24] Starting compilation in watch mode...

error TS18003: No inputs were found in config file 'D:/Jessey Araiza/Ingenieria/Cuatrimerio 3/Desarrollo Web/Unidad 03/backend_mysql/tsconfig.json'. Specified 'include' paths
were '["**/*"]' and 'exclude' paths were '["dist"]'.

[16:57:24] Found 1 error. Watching for file changes.

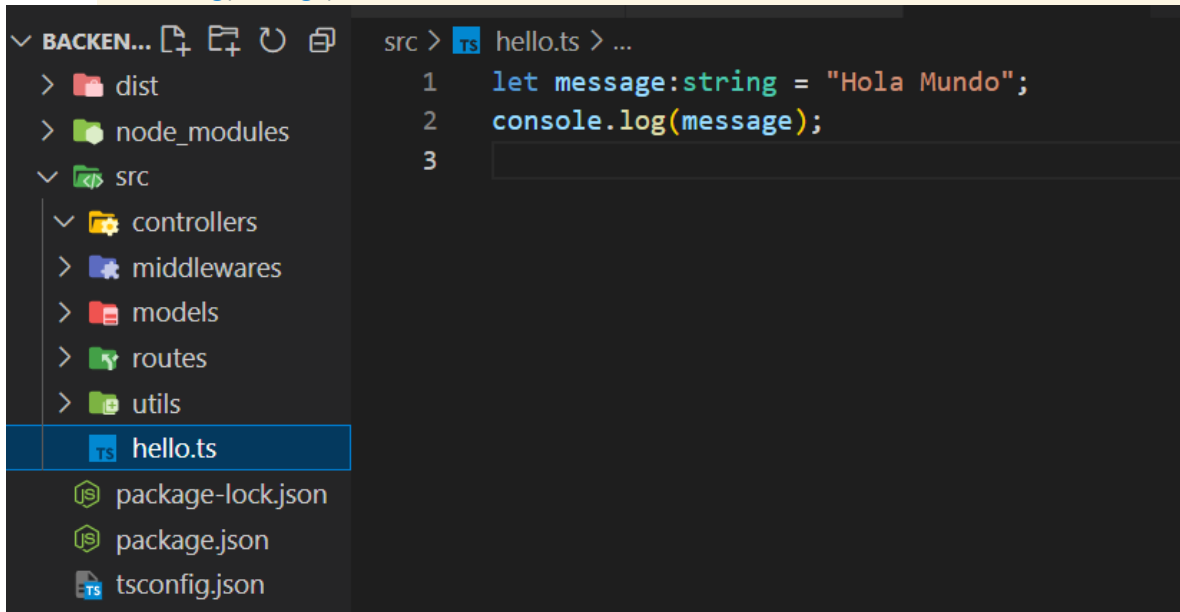
```

- Crear un archivo src/hello.ts

```

let message:string = "Hola Mundo";
console.log(message);

```



```

src > TS hello.ts > ...
1  let message:string = "Hola Mundo";
2  console.log(message);
3

```

- Verificar que no existan errores

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  
[13:28:21] File change detected. Starting incremental compilation...  
[13:28:21] Found 0 errors. Watching for file changes.  
█
```

```
C:\WINDOWS\system32\cmd.exe  
[16:58:05] File change detected. Starting incremental compilation...  
[16:58:11] Found 0 errors. Watching for file changes.
```

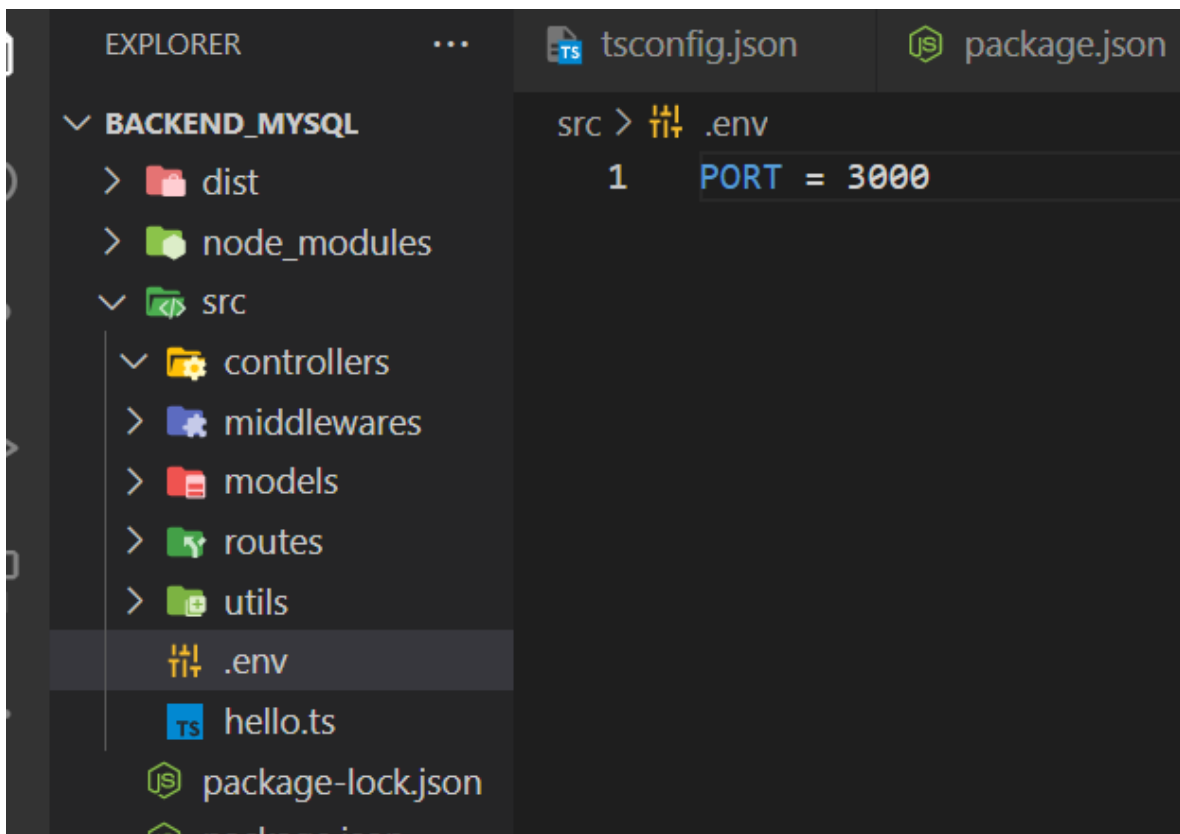
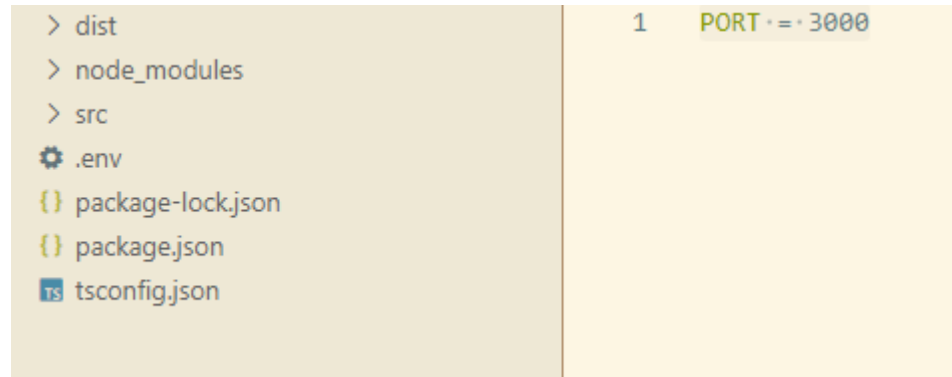
- Modificar el archivo package.json para agregar script [start](#)

```
"scripts": {  
  "build": "tsc -w",  
  "start": "node build/app.js",  
  "test": "echo \"Error: no test specified\" && exit 1"  
},
```

```
▶ Debug  
✓  
"scripts": {  
  "build": "tsc -w",  
  "start": "node build/app.js",  
  "test": "echo \"Error: no test specified\" && exit 1"  
},  
"keywords": [],
```

- Crear el archivo .env





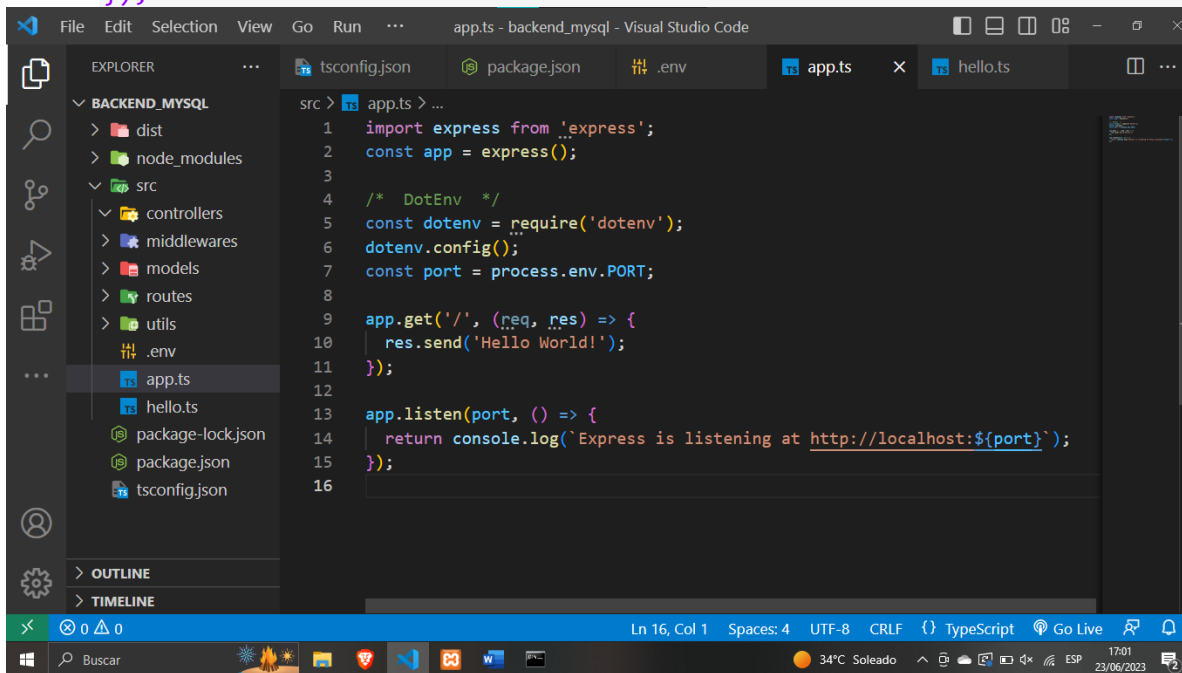
- Crear archivo src/app.ts, es nuestro punto de inicio de nuestra aplicación

```
import express from 'express';
const app = express();

/* DotEnv */
const dotenv = require('dotenv');
dotenv.config();
const port = process.env.PORT;

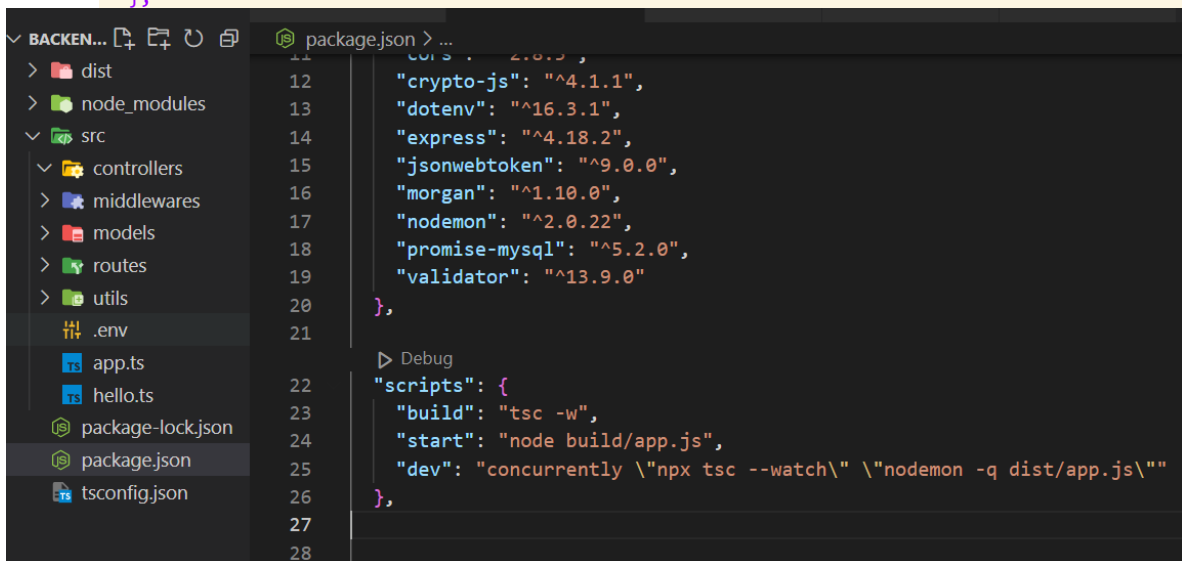
app.get('/', (req, res) => {
  res.send('Hello World!');
});
```

```
app.listen(port, () => {
  return console.log(`Express is listening at
http://localhost:${port}`);
});
```



- Modificar el archivo package.json

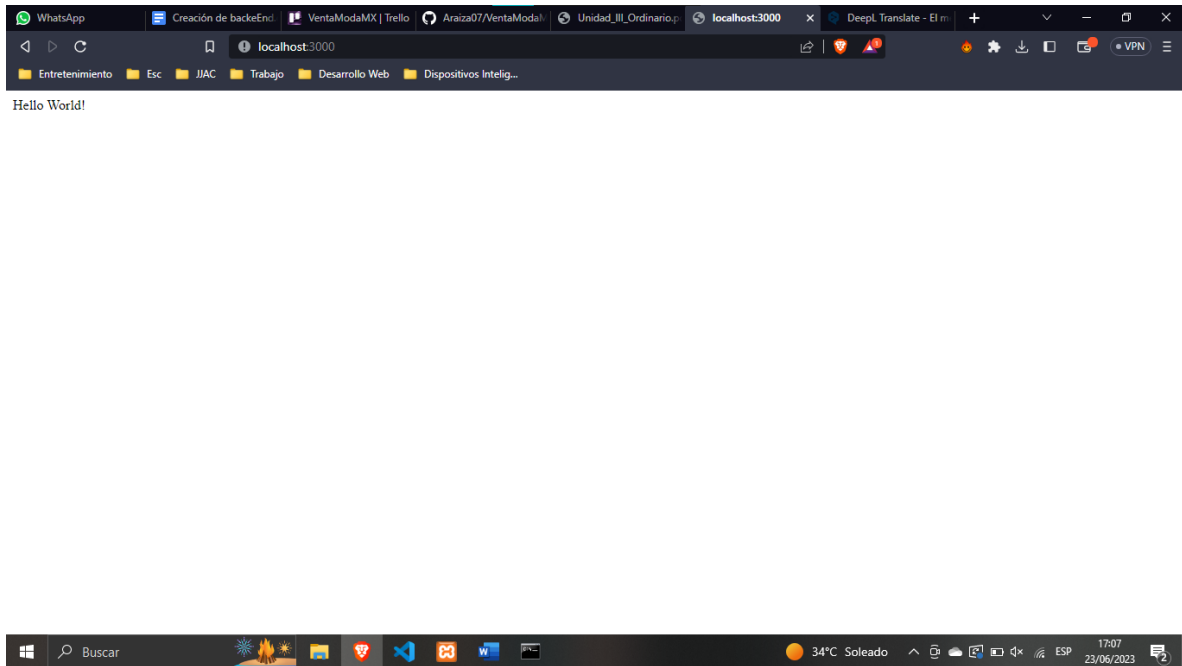
```
"scripts": {
  "build": "tsc -w",
  "start": "node build/app.js",
  "dev": "concurrently \"npx tsc --watch\" \"nodemon -q dist/app.js\""
},
```



- Ejecutar el comando

npm run dev

## Creación de Backend con Node Express



## Redefiniendo archivo app.ts

Referencia [AQUÍ](#)

- Modificar archivo `app.ts` con el uso de clases

```
1  class Server {
2
3  }
4
5  const server = new Server();
```

Definir los métodos de la clase

```
1  import express,{Application} from 'express';
2  /*
3   * Clase de inicio de nuestra aplicación NodeJsExpress
4   * Autor: Gabriel Barrón Rodríguez
5   * Fecha: 23 Junio 2023
6   */
7  class Server {
8      private app: Application;
9
10     //Inicializa clase
11     constructor() {
12
13     }
14
15     //Configuración de módulos
16     config(): void {
17
18     }
19
20     //Configura las rutas
21     routes() {
22
23     }
24 }
25
26 const server = new Server();
```

Definir lo qué tiene que inicializar el constructor

```
constructor() {
    this.app = express();
    this.config();
    this.routes();
    this.app.listen(this.app.get("port"), () => {
        console.log("Server on port", this.app.get("port"));
    });
}
```

- Definiendo método config de clase Server

```
import morgan from 'morgan';
import cors from 'cors';
import bodyParser from 'body-parser';

config(): void {
  // configuración del puerto para el servidor
  this.app.set("port", 3000);

  // muestra las peticiones en consola
  this.app.use(morgan("dev"));

  // puertos de conexión de la API
  this.app.use(cors());

  // solo se permiten peticiones en formato JSON

  this.app.use(bodyParser.json());

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

Referencia [CORS](#)

### Intercambio de recursos de origen cruzado (CORS)

El intercambio de recursos de origen cruzado ( CORS ) es un mecanismo basado en el encabezado HTTP que permite que un servidor indique cualquier origen (dominio, esquema o puerto) que no sea el suyo desde el cual un navegador debería permitir la carga de recursos. CORS también se basa en un mecanismo mediante el cual los navegadores realizan una solicitud de "verificación previa" al servidor que aloja el recurso de origen cruzado, para verificar que el servidor permite la solicitud real. En esa verificación previa, el navegador envía encabezados que indican el método HTTP y los encabezados que se utilizarán en la solicitud real.

Definiendo método routes

```
import authRoutes from './routes/authRoutes';

//Configura las rutas
routes() {
  this.app.use("/", authRoutes);
}
```

## Archivo final app.ts

```
import express, { Application } from 'express';
import cors from 'cors';
import morgan from 'morgan';
import bodyParser from 'body-parser';

import authRoutes from './routes/authRoutes';
/*
 * Clase de inicio de nuestra aplicación NodeJsExpress
 * Autor: Gabriel Barrón Rodríguez
 * Fecha: 23 Junio 2023
 */
class Server {
  private app: Application;

  //Inicializa clase
  constructor() {
    this.app = express();
    this.config();
    this.routes();
    this.app.listen(this.app.get("port"), () => {
      console.log("Server on port", this.app.get("port"));
    });
  }

  //Configuración de módulos
  config(): void {
    // configuración del puerto para el servidor
    this.app.set("port", 3000);

    // muestra las peticiones en consola
    this.app.use(morgan("dev"));

    // puertos de conexión de La API
    this.app.use(cors());

    // solo se permiten peticiones en formato JSON
    this.app.use(bodyParser.json());
    this.app.use(bodyParser.urlencoded({ extended: false, }));
  }

  //Configura Las rutas
  routes() {
    this.app.use("/", authRoutes);
  }
}

const server = new Server();
```

```

src > app.ts > Server > routes
1  import express, { Application } from 'express';
2  import cors from 'cors';
3  import morgan from 'morgan';
4  import bodyParser from 'body-parser';
5
6  //import authRoutes from './routes/authRoutes';
7  /*
8   * Clase de inicio de nuestra aplicación NodeJsExpress
9   * Autor: Jessie Javier Araiza Cervantes
10  * Fecha: 23 Junio 2023
11  */
12  class Server {
13      private app: Application;
14
15      //Inicializa clase
16      constructor() {
17          this.app = express();
18          this.config();
19          this.routes();
20          this.app.listen(this.app.get("port"), () => {
21              console.log("Server on port", this.app.get("port"));

```

## Definiendo rutas

### Referencia

- Ubicar carpeta de rutas `src|routes`
- Crear archivo `src|routes|authRoutes.ts`

Definir la clase para rutas de autenticación

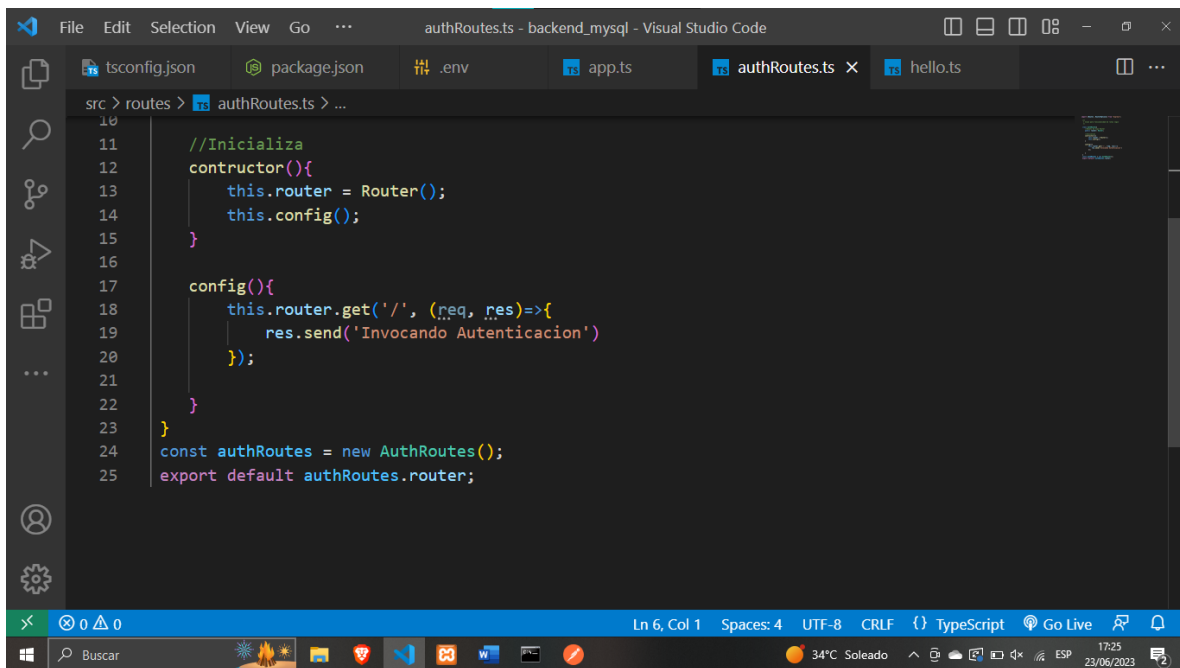
```

1  import { Router, RouterOptions } from "express";
2
3  /*
4   * Clase para funcionalidad de rutas Login
5   */
6  class AuthRoutes {
7      //Objeto de tipo Router
8      public router: Router;
9
10     //Inicializa
11     constructor() {
12     }
13
14     config() {
15     }
16 }
17 const authRoutes = new AuthRoutes();
18 export default authRoutes.router;

```

Definir método de configuración e inicializar la clase

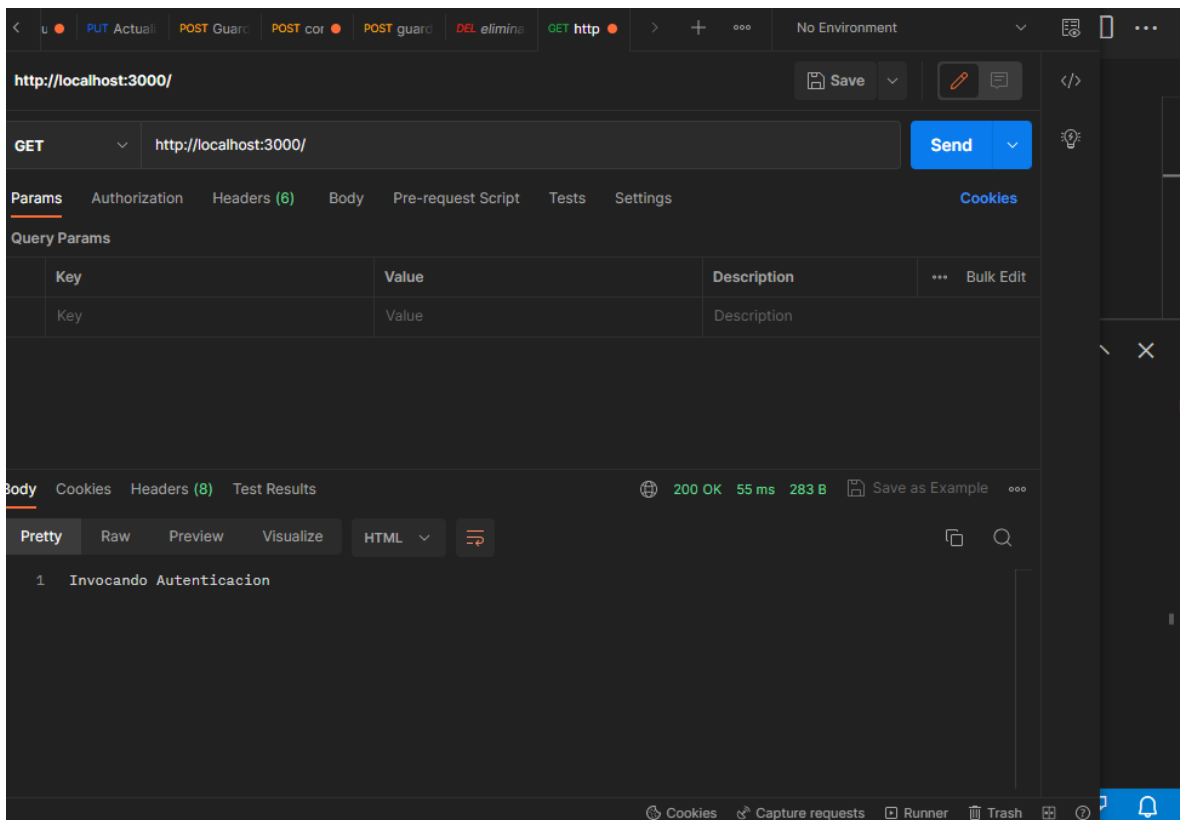
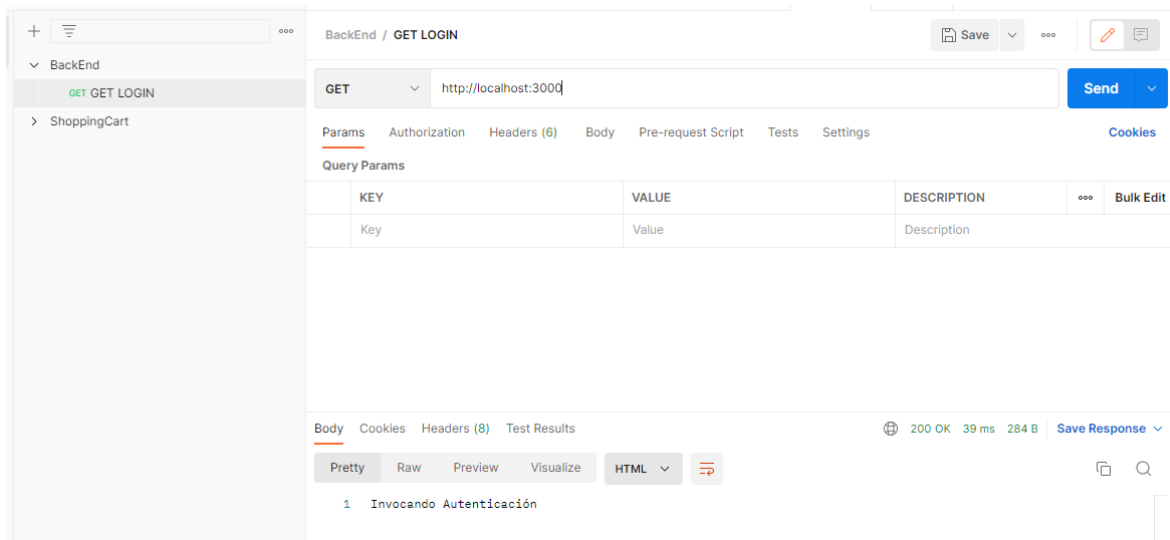
```
class AuthRoutes {  
  //Objeto de tipo Router  
  public router: Router;  
  
  //Inicializa  
  constructor() {  
    this.router = Router();  
    this.config();  
  }  
  
  config() {  
    this.router.get('/', (req, res) => {  
      res.send('Invocando Autenticación')  
    });  
  }  
}  
  
const authRoutes = new AuthRoutes();  
export default authRoutes.router;
```



Verificar path



## Creación de Backend con Node Express



## Definiendo controlador

Ubicar carpeta src|controllers

Crear archivo src|controllers|authController.ts

```
import { Request, Response } from "express";
```

```
class AuthController {

  public async iniciarSesion(req: Request, res: Response) {
    const {username, password} = req.body;
    return res.json({ message : "Autenticación correcta",
      username: username,
      password: password });
  }

}

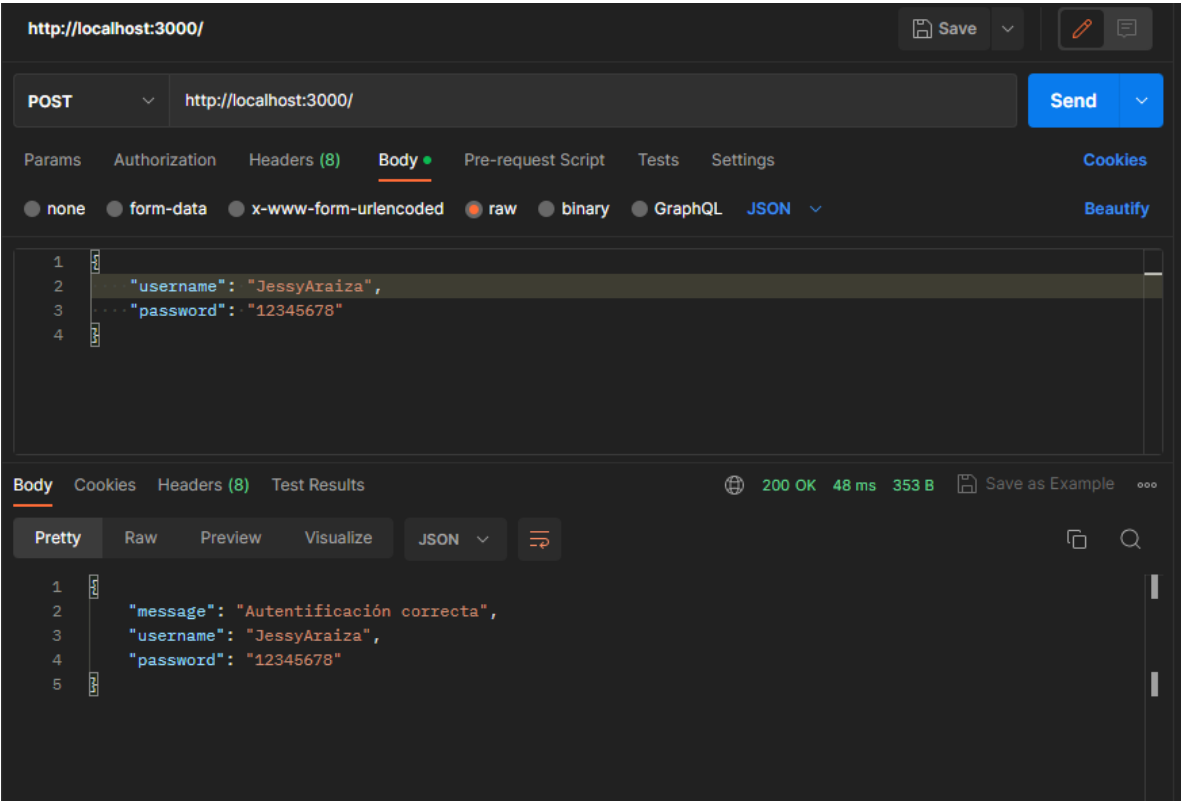
export const authController = new AuthController();
```

Modificar el método de configuración de archivo src/routes/authRoutes a través del método POST

```
config() {
  this.router.post('/', authController.iniciarSesion);
}
```

Verificar envío de datos nuevamente con Postman

The screenshot shows the Postman interface for a POST request to `http://localhost:3000`. The request body is a JSON object: `{ "username": "juanito", "password": "12345" }`. The response is a 200 OK status with a response time of 42 ms and a body size of 346 B. The response body is displayed in the 'Body' tab, showing the JSON response: `{ "message": "Autenticación correcta", "username": "juanito", "password": "12345" }`.



## Configurar base de datos

Definir clave secreta en archivo .env

```
PORT = 3000

SECRET= ')(/ &%$webintegral$#&/%'
```

Crear una base de datos en MySql llamada web\_integral

Crear una tabla llamada tbl\_usuario

Atributo	Tipo de datos	Restricción
Username	Varchar(30)	Primary Key
Password	Varchar(250)	Not Null
role	Varchar(20)	Not Null

Insertar el usuario de acuerdo a los siguientes datos

username	progra01
password	12345
role	admin

Inserta al menos un registro

```
insert into tbl_usuario(username, password, role) values('progra01','123456','admin');
```

The screenshot shows a MySQL database management interface. The top navigation bar includes tabs for 'Examinar', 'Estructura', 'SQL', 'Buscar', 'Insertar', 'Exportar', 'Importar', 'Privilegios', 'Operaciones', and 'Disparado'. The main content area displays a message: 'Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,0003 segundos.)'. Below this, a SQL query is shown: 'SELECT \* FROM `usuario`'. A toolbar contains buttons for 'Perfilando', 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'Actualizar'. A filter section shows 'Mostrar todo', 'Número de filas: 25', and 'Filtrar filas: Buscar en esta tabla'. An 'Opciones extra' button is also present. The table view shows columns 'id', 'username', 'password', and 'role'. A single row is displayed with values: '1', 'progra01', '123456', and 'admin'. Below the table, there are buttons for 'Editar', 'Copiar', and 'Borrar'. A summary bar at the bottom indicates 'Seleccionar todo' and 'Para los elementos que están marcados: Editar, Copiar, Borrar, Exportar'. Another filter section is visible at the bottom, identical to the one above.

id	username	password	role
1	progra01	123456	admin

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'root';
```

Crear archivo para conexión a base de datos MySQL [src|config|connection.ts](#)

[Referencia Promise-MySQL](#)

```
import mysql from 'promise-mysql';

const pool = mysql.createPool(JSON.stringify({
  host: 'localhost',
  port: 3306,
  user: 'admin',
  password: 'admin',
  database: 'web_integral'
}));
export default pool;
```

## Modelos

Crear archivo [src|models|authModelo.ts](#)

```
import pool from '../utils/connection';

class AuthModelo {

  /*
  *Método para buscar un usuario por username
  */
  public async getUserByUsername(username: string) {
    const result = await pool.then(async (connection) => {
      return await connection.query(
        " SELECT * FROM tbl_usuario WHERE username = ? ",
        [username]);
    });
    return result;
  }
}

const model = new AuthModelo();
export default model;
```

## Controlador

Modificar archivo `src\controllers\authController`

Extraer los datos del cuerpo de la solicitud, verificar código de estado [HTTP](#)

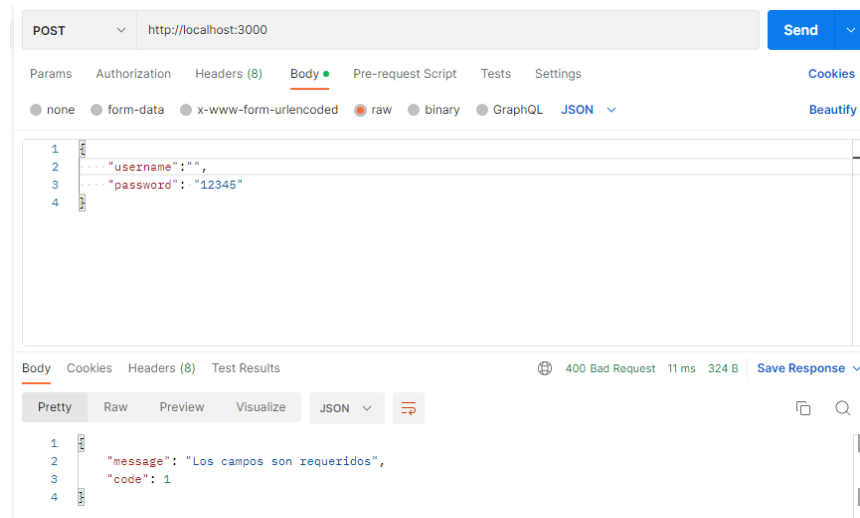
```
/**
 * Método para validad Inicio de sesión
 * @param req Petición
 * @param res respuesta
 * @returns
 */
public async iniciarSesion(req: Request, res: Response) {

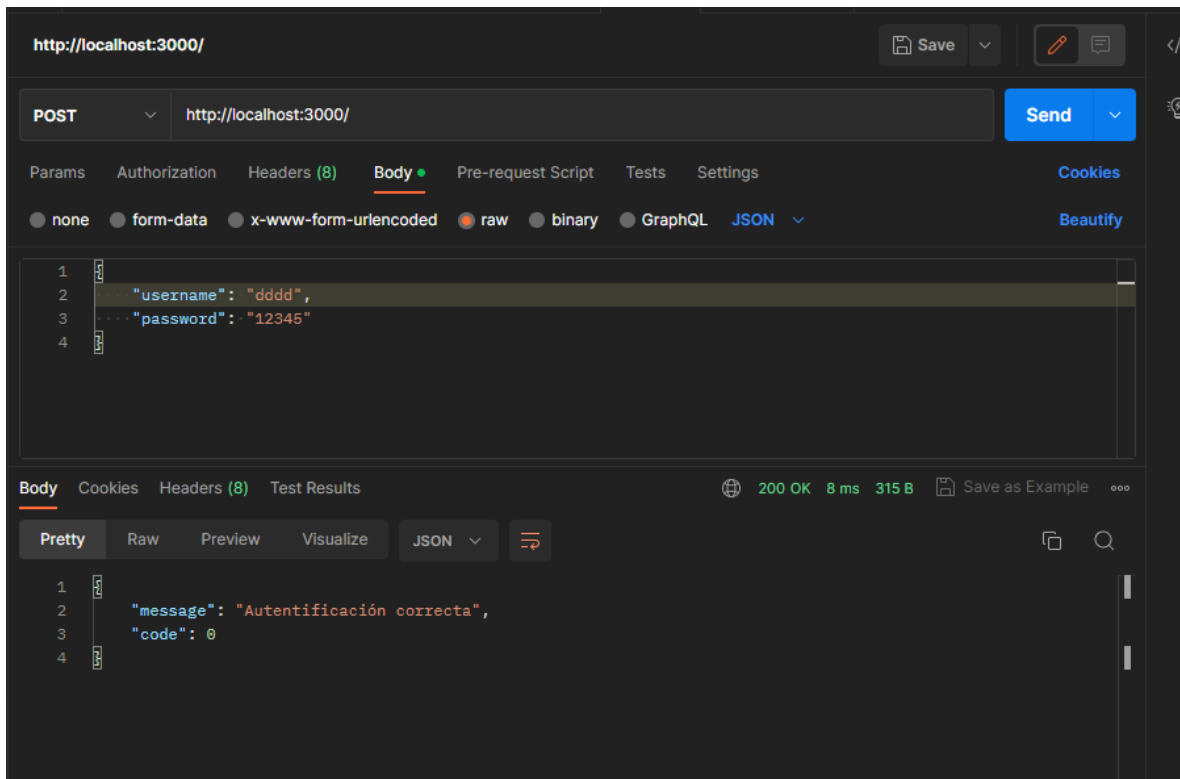
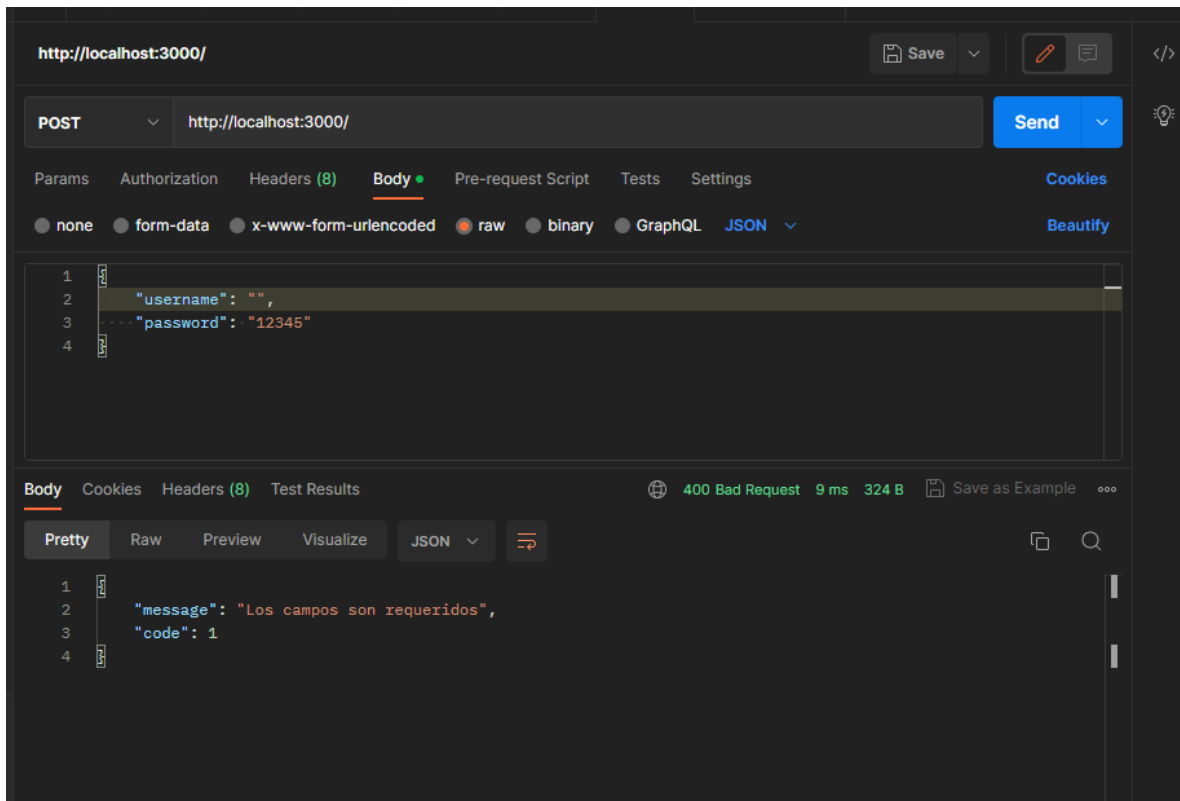
  try {
    const {username, password }= req.body;

    // 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 });
    }

    return res.json({ message : "Autenticación correcta", code: 0 });
  } catch (error: any) {
    return res.status(500).json({ message : `${error.message}` });
  }
}
```

Verificar con Postman





Definiendo el acceso a los datos de datos

```
import validator from 'validator';
```

```
import model from '../models/authModelo';
```

```
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 });  
}  
  
return res.json({ message: "Autenticación correcta", code: 0 });
```



## Creación de Backend con Node Express

The screenshot shows a Postman interface for a POST request to `http://localhost:3000/`. The request body is a JSON object: `{ "username": "progra01", "password": "123456" }`. The response status is `200 OK` with a response time of `10 ms` and a size of `315 B`. The response body is a JSON object: `{ "message": "Autenticación correcta", "code": 0 }`.

```
POST http://localhost:3000/

{
  "username": "progra01",
  "password": "123456"
}
```

200 OK 10 ms 315 B

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

The screenshot shows a Postman interface for a POST request to `http://localhost:3000/`. The request body is a JSON object: `{ "username": "progra00", "password": "123456" }`. The response status is `404 Not Found` with a response time of `13 ms` and a size of `337 B`. The response body is a JSON object: `{ "message": "El usuario y/o contraseña es incorrecto", "code": 1 }`.

```
POST http://localhost:3000/

{
  "username": "progra00",
  "password": "123456"
}
```

404 Not Found 13 ms 337 B

```
{
  "message": "El usuario y/o contraseña es incorrecto",
  "code": 1
}
```

## Path Usuarios

```
import { Router } from "express";
import { usuarioController } from "../controllers/usuarioController";

class UsuarioRoutes {

  public router: Router;

  constructor() {
    this.router = Router();
    this.config();
  }

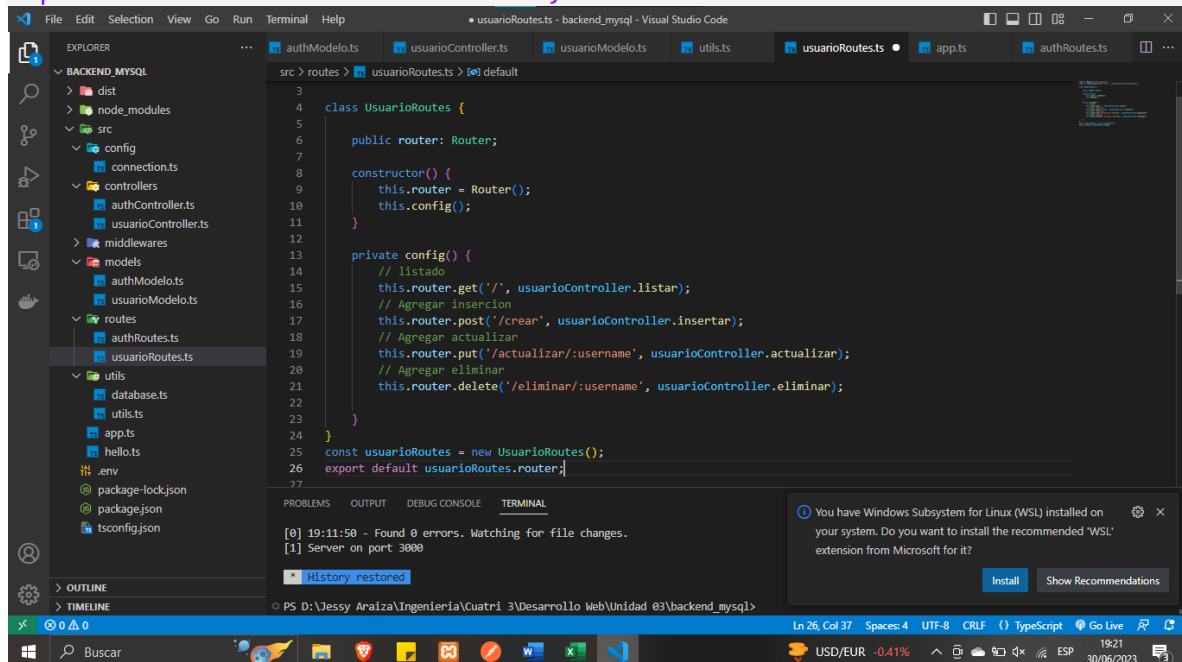
  private config() {
    // listado
    this.router.get('/', usuarioController.listar);

    // Agregar insercion

    // Agregar actualizar

    // Agregar eliminar
  }
}

const usuarioRoutes = new UsuarioRoutes();
export default usuarioRoutes.router;
```

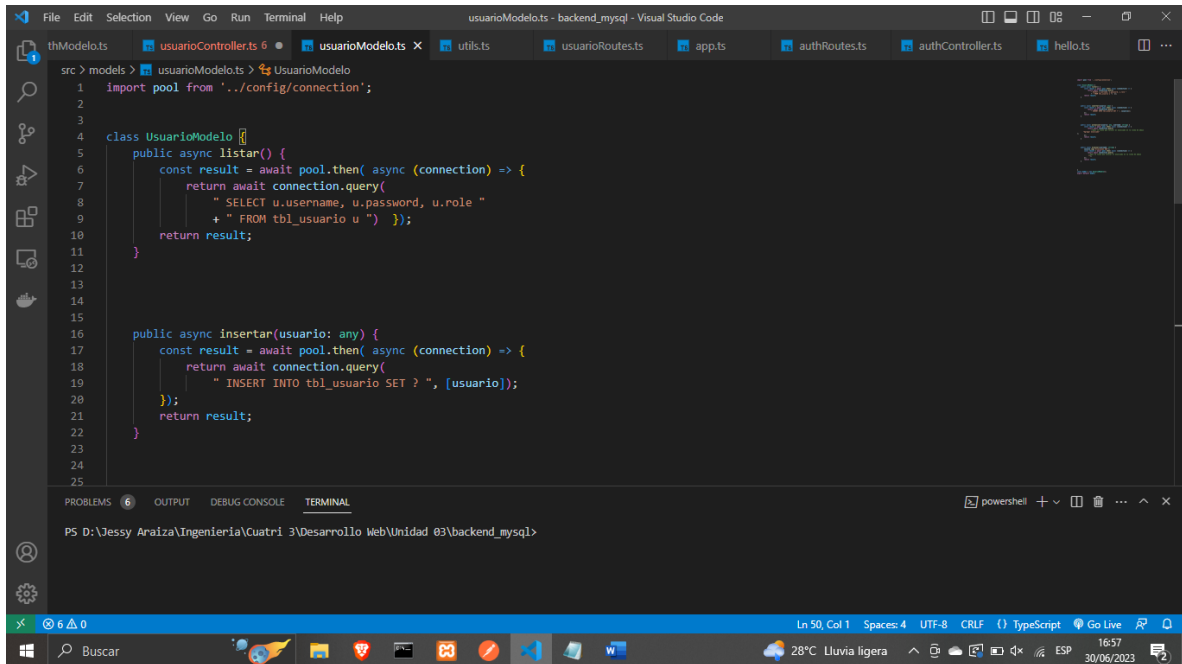


Crear el modelo usuarioModelo.ts en carpeta models

```
import pool from '../utils/connection';
```

```
class UsuarioModelo {  
  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;  
  }  
  
  public async insertar(usuario: any) {  
    const result = await pool.then( async (connection) => {  
      return await connection.query(  
        " INSERT INTO tbl_usuario SET ? ", [usuario]);  
    });  
    return result;  
  }  
  
  public async actualizar(usuario: any, username: string) {  
    const result = await pool.then( async (connection) => {  
      return await connection.query(  
        //Agregar enunciado  
      );  
    });  
    return result;  
  }  
  
  public async eliminar(username: string) {  
    console.log('Eliminando DAO');  
    const result = await pool.then( async (connection) => {  
      return await connection.query(  
        //Agregar enunciado  
      );  
    });  
    return result;  
  }  
}
```

```
}  
  
const model = new UsuarioModelo();  
  
export default model;
```



## Encriptamiento de texto

Crear archivo `src|utils|utils.ts`

Instalar bcrypt

```
npm i bcryptjs -D
```

Modificar archivo `utils.ts` para encriptar un texto

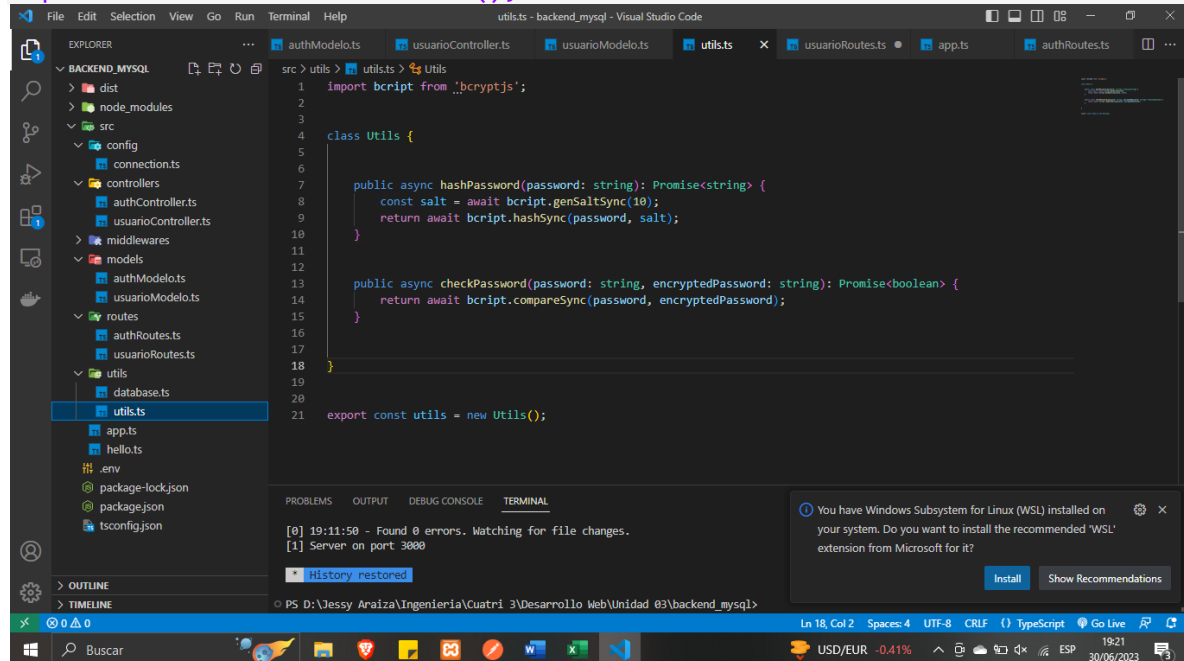
```
import bcrypt from 'bcryptjs';

class Utils {

  public async hashPassword(password: string): Promise<string> {
    const salt = await bcrypt.genSaltSync(10);
    return await bcrypt.hashSync(password, salt);
  }

  public async checkPassword(password: string, encryptedPassword: string):
  Promise<boolean> {
    return await bcrypt.compareSync(password, encryptedPassword);
  }
}

export const utils = new Utils();
```



## Controlador usuarioController

```
import { Request, Response } from "express";
import model from '../models/usuarioModelo';
import validator from 'validator';
import { utils } from '../utils/utils';

class UsuarioController {
  /**
   * @description Lista Los usuarios disponibles
   * @param req
   * @param res
   * @returns Promise<Response<any, Record<string, any>> | undefined>
   */
  public async listar(req: Request, res: Response) {
    try {
      const result = await model.listar();
      res.json(result);
    } catch (error: any) {
      return res.status(500).json({ message : `${error.message}` });
    }
  }

  /**
   * @description Inserción de usuarios a la bd
   * @param req
   * @param res
   * @returns Promise<Response<any, Record<string, any>> | undefined>
   */
  public async insertar(req: Request, res: Response) {
    try {
      // se obtienen los datos del body
      var usuario = req.body;
      console.log(usuario);

      // validar que los datos no sean nulos o indefinidos
      if (!usuario.username
        || !usuario.password
        || !usuario.role) {
        return res.status(404).json({ message: "Todos los datos son requeridos", code: 1});
      }

      // encriptar nuestra contraseña
      var encryptedText = await utils.hashPassword(usuario.password);
      usuario.password = encryptedText;
      console.log("Contraseña encriptada " + typeof usuario.password);

      const newUser = {
```

```

        username: usuario.username,
        password: usuario.password,
        role: usuario.role
    }

    console.log(newUser);

    // inserción de los datos
    //Agregar enunciado

    if (result.affectedRows > 0) {
        return res.json({message: "Los datos se guardaron
correctamente", code: 0});
    } else {
        return res.status(404).json({ message: result.message, code:
1});
    }

    } catch (error: any) {
        return res.status(500).json({ message : `${error.message}` });
    }
}

public async actualizar(req: Request, res: Response) {
    try {
        // se obtienen los datos del body
        var usuario = req.body;

        // validar que los datos no sean nulos o indefinidos
        if (!usuario.username
            || !usuario.password) {
            return res.status(404).json({ message: "Todos los datos
son requeridos", code: 1});
        }

        // se verifica que los datos no se encuentren vacios
        if ( validator.isEmpty(usuario.username) ||
validator.isEmpty(usuario.password)){
            return res.status(404).json({ message: "Todos los datos
son requeridos", code: 1});
        }

        let encryptedText = await utils.hashPassword(usuario.password);
        usuario.password = encryptedText;
    }
}

```

```

        const newUser = {
            password: usuario.password
        }

        // actualización de los datos
        //Agregar enunciado

        if (result.affectedRows > 0) {
            return res.json({message: "Los datos se actualizaron
correctamente", code: 0});
        } else {
            return res.status(404).json({ message: result.message, code:
1});
        }

    } catch (error: any) {
        return res.status(500).json({ message : `${error.message}` });
    }
}

public async eliminar(req: Request, res: Response) {
    try {
        // se obtienen los datos del

        const username = req.params.username;
        console.log(username);

        // validar que los datos no sean nulos o indefinidos
        if (!username) {
            return res.status(404).json({ message: "Todos los datos
son requeridos", code: 1});
        }

        // se verifica que los datos no se encuentren vacios
        if (validator.isEmpty(username)) {
            return res.status(404).json({ message: "Todos los datos
son requeridos", code: 1});
        }

        // actualización de los datos
        //Agregar enunciado

```



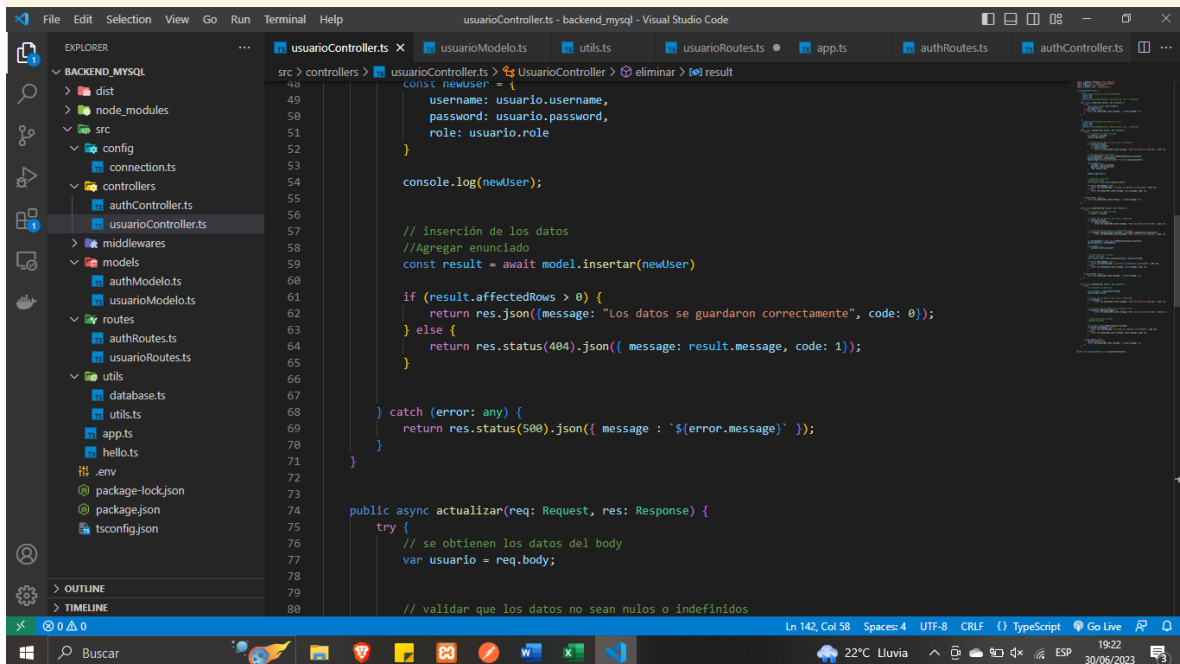
```

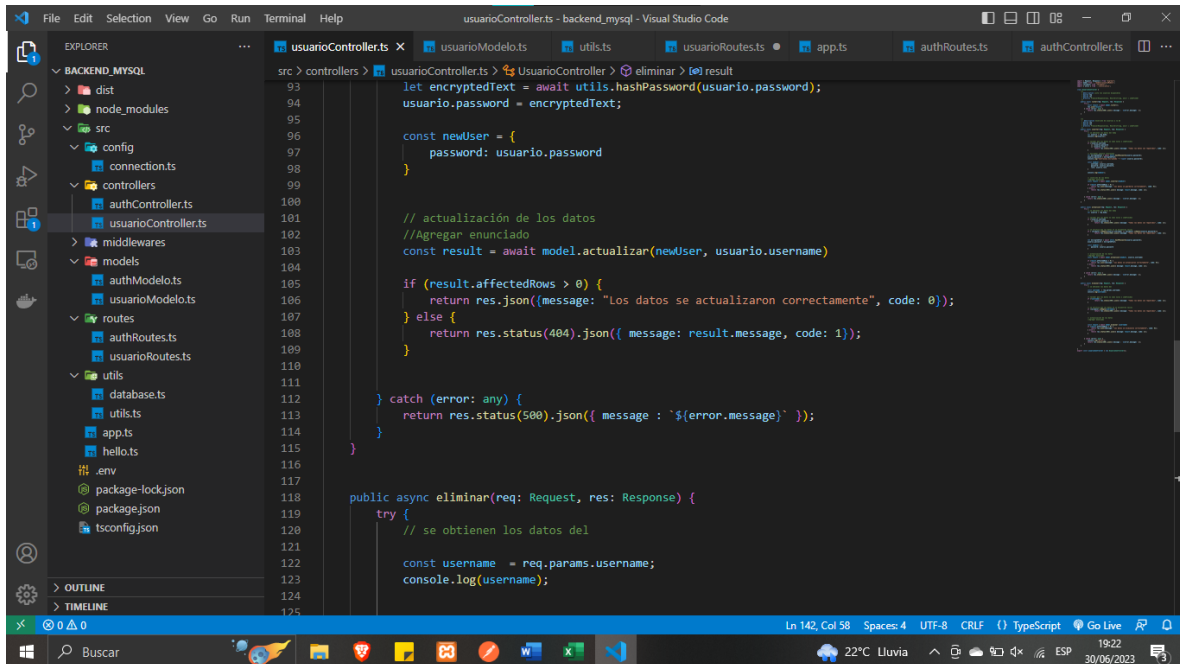
        if (result.affectedRows > 0) {
            return res.json({message: "Los datos se eliminaron
correctamente", code: 0});
        } else {
            return res.status(404).json({ message: result.message, code:
1});
        }

    } catch (error: any) {
        console.log("Error");
        return res.status(500).json({ message : `${error.message}` });
    }
}

export const usuarioController = new UsuarioController();

```





```
src > controllers > UsuarioController > eliminar > result
let encryptedText = await utils.hashPassword(usuario.password);
usuario.password = encryptedText;

const newUser = {
  password: usuario.password
}

// actualización de los datos
//Agregar enunciado
const result = await model.actualizar(newUser, usuario.username)

if (result.affectedRows > 0) {
  return res.json({message: "Los datos se actualizaron correctamente", code: 0});
} else {
  return res.status(404).json({ message: result.message, code: 1});
}

} catch (error: any) {
  return res.status(500).json({ message: `${error.message}` });
}

}

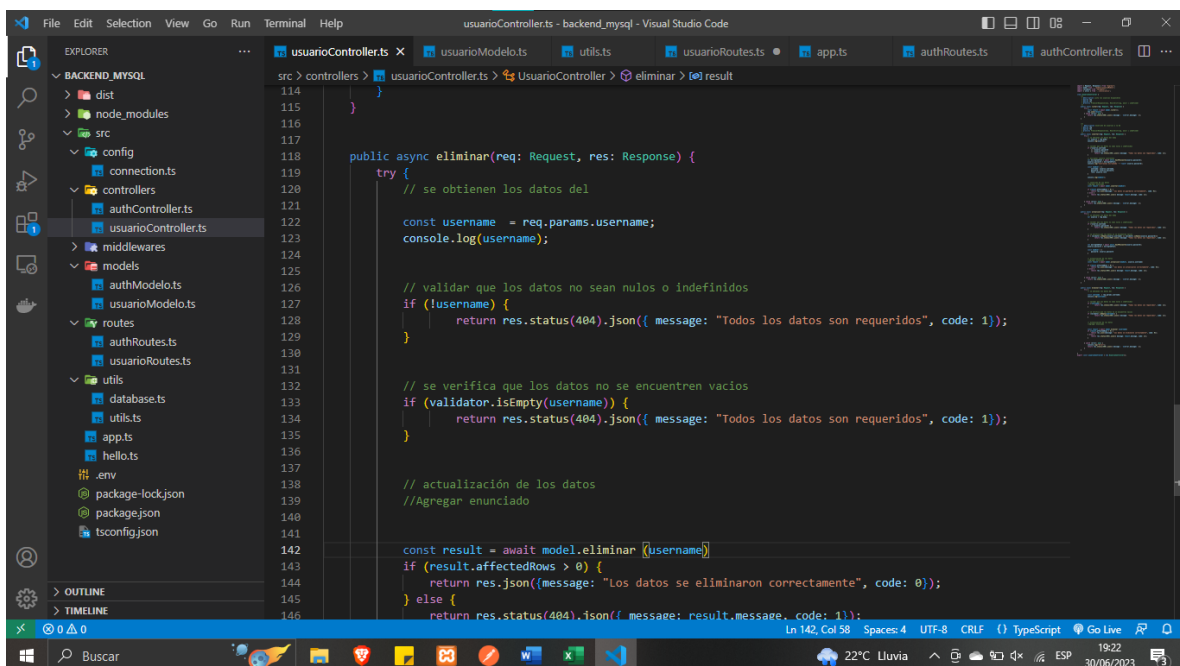
public async eliminar(req: Request, res: Response) {
  try {
    // se obtienen los datos del

    const username = req.params.username;
    console.log(username);

    // actualización de los datos
    //Agregar enunciado

    const result = await model.eliminar (username)

    if (result.affectedRows > 0) {
      return res.json({message: "Los datos se eliminaron correctamente", code: 0});
    } else {
      return res.status(404).json({ message: result.message, code: 1});
    }
  } catch (error: any) {
    return res.status(500).json({ message: `${error.message}` });
  }
}
```



```
src > controllers > UsuarioController > eliminar > result
}

}

public async eliminar(req: Request, res: Response) {
  try {
    // se obtienen los datos del

    const username = req.params.username;
    console.log(username);

    // validar que los datos no sean nulos o indefinidos
    if (!username) {
      return res.status(404).json({ message: "Todos los datos son requeridos", code: 1});
    }

    // se verifica que los datos no se encuentren vacios
    if (validator.isEmpty(username)) {
      return res.status(404).json({ message: "Todos los datos son requeridos", code: 1});
    }

    // actualización de los datos
    //Agregar enunciado

    const result = await model.eliminar (username)

    if (result.affectedRows > 0) {
      return res.json({message: "Los datos se eliminaron correctamente", code: 0});
    } else {
      return res.status(404).json({ message: result.message, code: 1});
    }
  } catch (error: any) {
    return res.status(500).json({ message: `${error.message}` });
  }
}
```

## Realizar las pruebas con Postman

### JSON Web Token

JWT (JSON Web Token) es un estándar que está dentro del documento RFC 7519. En el mismo se define un mecanismo para poder propagar entre dos partes, y de forma segura, la identidad de un determinado usuario, además con una serie de claims o privilegios.

Estos privilegios están codificados en objetos de tipo JSON, que se incrustan dentro del payload o cuerpo de un mensaje que va firmado digitalmente.

### Token JWT

En la práctica, se trata de una cadena de texto que tiene tres partes codificadas en Base64, cada una de ellas separadas por un punto, como la que vemos en la imagen siguiente:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.ikFGEvw-Du0f30vBaA742D_wqPA5BBHXgUY6wwqab1w
```

Podemos utilizar un [debugger online](#) para decodificar ese token y visualizar su contenido. Si accedemos al mismo y pegamos dentro el texto completo, se nos mostrará lo que contiene:

Podemos ver el contenido del token sin necesidad de saber la clave con la cual se ha generado, aunque no podemos validarlo sin la misma.

Como hemos dicho, un token tres partes:

- **Header:** encabezado dónde se indica, al menos, el algoritmo y el tipo de token, que en el caso del ejemplo anterior era el algoritmo HS256 y un token JWT.
- **Payload:** donde aparecen los datos de usuario y privilegios, así como toda la información que queramos añadir, todos los datos que creamos convenientes.
- **Signature:** una firma que nos permite verificar si el token es válido, y aquí es donde radica el quid de la cuestión, ya que si estamos tratando de hacer una comunicación segura entre partes y hemos visto que podemos coger cualquier token y ver su contenido con una herramienta sencilla, ¿dónde reside entonces la potencia de todo esto?

### Firma de un token JWT

La firma se construye de tal forma que vamos a poder verificar que el remitente es quien dice ser, y que el mensaje no se ha modificado por el camino.

Se construye como el HMACSHA256, que son las siglas de Hash-Based Message Authentication Code (Código de Autenticación de Mensajes), cifrado además con el algoritmo SHA de 256 bits. Se aplica esa función a:

- **Codificación en Base64 de header.**
- **Codificación en Base64 de payload.**
- **Un secreto**, establecido por la aplicación.

De esta forma, si alguien modifica el token por el camino, por ejemplo, inyectando alguna credencial o algún dato malicioso, entonces podríamos verificar que la comprobación de la firma no es correcta, por lo que no podemos confiar en el token recibido y deberíamos denegar la solicitud de recursos que nos haya realizado, ya sea para obtener datos o modificarlos.

Si lo que estamos requiriendo es que el usuario esté autenticado, deberíamos denegar esa petición, por lo que siempre que trabajemos con JWT deberíamos verificar con esa firma si el token es válido o no lo es.

### Token JWT seguro

Aunque el algoritmo nos permita verificar la firma, y, por tanto, confiar o no, tanto el encabezado como el cuerpo si llevan muchos datos van en abierto, ya que Base64 no es un cifrado, es simplemente una codificación que es muy fácilmente decodificable.

JWT nos invita siempre a que la comunicación entre partes se realice con HTTPS para encriptar el tráfico, de forma que, si alguien lo interceptara, el propio tráfico a través de HTTP sobre esos sockets SSL, cifrara toda la comunicación, la del token y todo lo demás, y así añadir esa posibilidad de seguridad.

De hecho, **siempre deberíamos utilizar HTTPS y un servidor con certificado** para hacer el despliegue de nuestras aplicaciones, no solamente con este con este tipo de token JWT.

### Ciclo de vida de un Token JWT

Vamos a ver ahora el ciclo de vida de un token JWT, si lo queremos utilizar en el marco de un proceso de autenticación.

Como hemos visto, JWT no es un estándar de autenticación, sino que simplemente un estándar que nos permite hacer una comunicación entre dos partes de identidad de usuario. Con este proceso, además, podríamos implementar la autenticación de usuario de una manera que fuera relativamente segura.



Comenzaríamos desde el cliente, haciendo una petición POST para enviar el usuario y contraseña, y realizar el proceso de login.

Se comprobaría que ese usuario y su contraseña son correctos, y de serlos, generar el token JWT para devolverlo al usuario.

A partir de ahí la aplicación cliente, con ese token, haría peticiones solicitando recursos, siempre con ese token JWT dentro de un encabezado, que sería `Authorization: Bearer XXXXXXXX`, siendo Bearer el tipo de prefijo seguido de todo el contenido del token.

En el servidor se comprobaría el token mediante la firma, para verificar que el token es seguro, y, por tanto, podemos confiar en el usuario.

Dentro del cuerpo del token, además, tenemos los datos de quién es el usuario que ha realizado esa petición, porque podemos contener en el payload todos los datos de usuario que queramos.

Tras verificar que el token es correcto y saber quién es el que ha hecho la petición, podemos aplicar entonces el mecanismo de control de acceso, saber si puede acceder o no, y si es así, responder con el recurso protegido, de manera que lo podría recibir de una forma correcta.

De esta forma podríamos **implementar el proceso de autenticación**, y hacerlo, además, con estos JSON Web Token.


## JWT en AuthController

Modificar el archivo `AuthController.ts` agregando los imports de JWT, la clave secreta y el encriptamiento

```
import jwt from 'jsonwebtoken';  
import db from '../utils/database';
```

```
import {utils} from '../utils/utils';
```

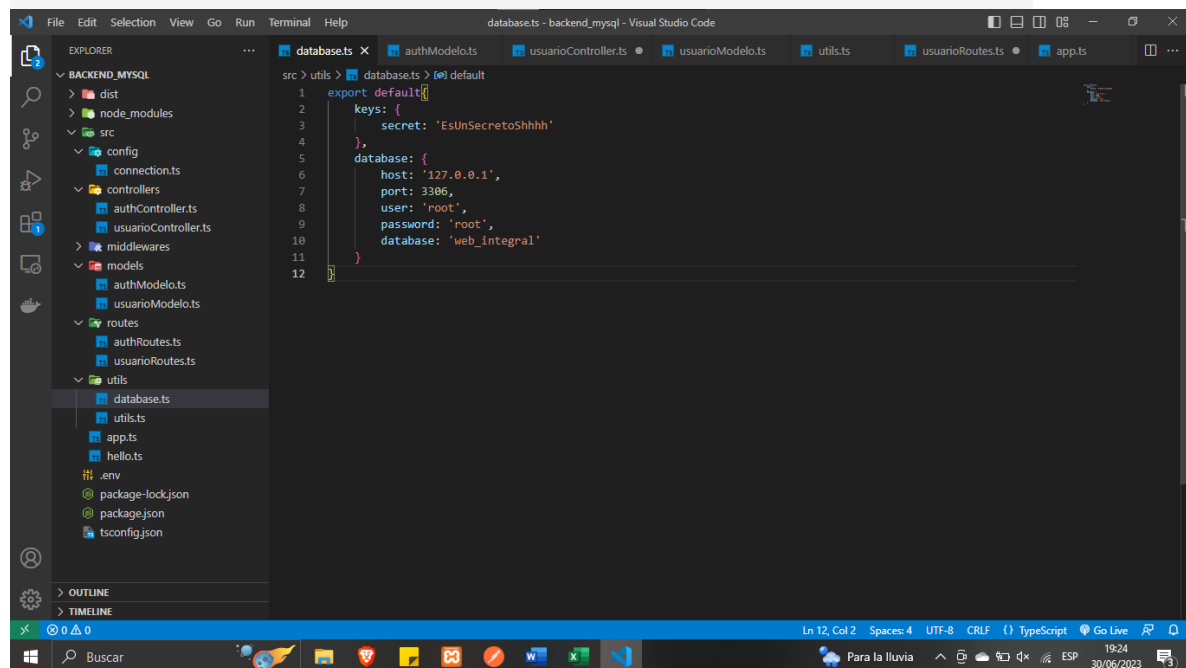
Definir la clave secreta para el JWT



```

TS authController.ts 3    TS utils.ts    TS database.ts X
src > utils > TS database.ts > [🔍] default > 🔑 keys > 🔑 secret
You, 4 minutes ago | 1 author (You)
1  export default {
2      keys: {
3          secret: ')(/%%$webintegral$#&/%'
4      },
5      database: {
6          host: '127.0.0.1',
7          port: 3306,
8          user: 'root',
9          password: 'root',
10         database: 'web_integral'
11     }
12 }

```



Verificar qué los métodos de encriptamiento se encuentren en src|utils

```

1  import bcrypt from 'bcryptjs';
2
3  You, 5 minutes ago | 1 author (You)
4  class Utils {
5
6      public async hashPassword(password: string): Promise<string> {
7          const salt = await bcrypt.genSaltSync(10);
8          return await bcrypt.hashSync(password, salt);
9      }
10
11     public async checkPassword(password: string, encryptedPassword: string): Promise<boolean> {
12         return await bcrypt.compareSync(password, encryptedPassword);
13     }
14 }
15
16 export const utils = new Utils();

```

You, 5 minutes ago • Version inicial

Agregar la instrucción para verificar al usuario y la contraseña resultado de consulta a la base de datos

```

const lstUsers = await dao.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);
    return res.json({ message: 'Autenticación Correcta', code: 0 });
} catch (error: any) {
    return res.status(500).json({ message: `${error.message}` });
}

export const authController = new AuthController();

```

Visual Studio Code interface showing the `authController.ts` file. The code defines an `AuthController` class with a `login` method. The `login` method queries the database for a user by username. If no user is found, it returns a 404 status with a message. If a user is found, it logs the user's username and password (highlighted with a red box) and returns a 200 status with a success message. The file is named `authController.ts` and is part of a project named `backend_mysql`.

Una vez que se encuentra el usuario y la contraseña verifica con el módulo **bcrypt** si coincide la contraseña

## Creación de Backend con Node Express

```

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});
}
return res.json({message: 'Autenticación Correcta', code:0});
} catch (error: any) {
  return res.status(500).json({ message : `${error.message}` });
}

```

## Realiza la prueba con Postman

Web Integral / POST Login

Save

POST

http://localhost:3000

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {  
2   "username": "progra01",  
3   "password": "12345"  
4 }
```

Body

Cookies

Headers (8)

Test Results

Status: 200 OK Time: 389 ms Size: 602 B Save as Example

Pretty

Raw

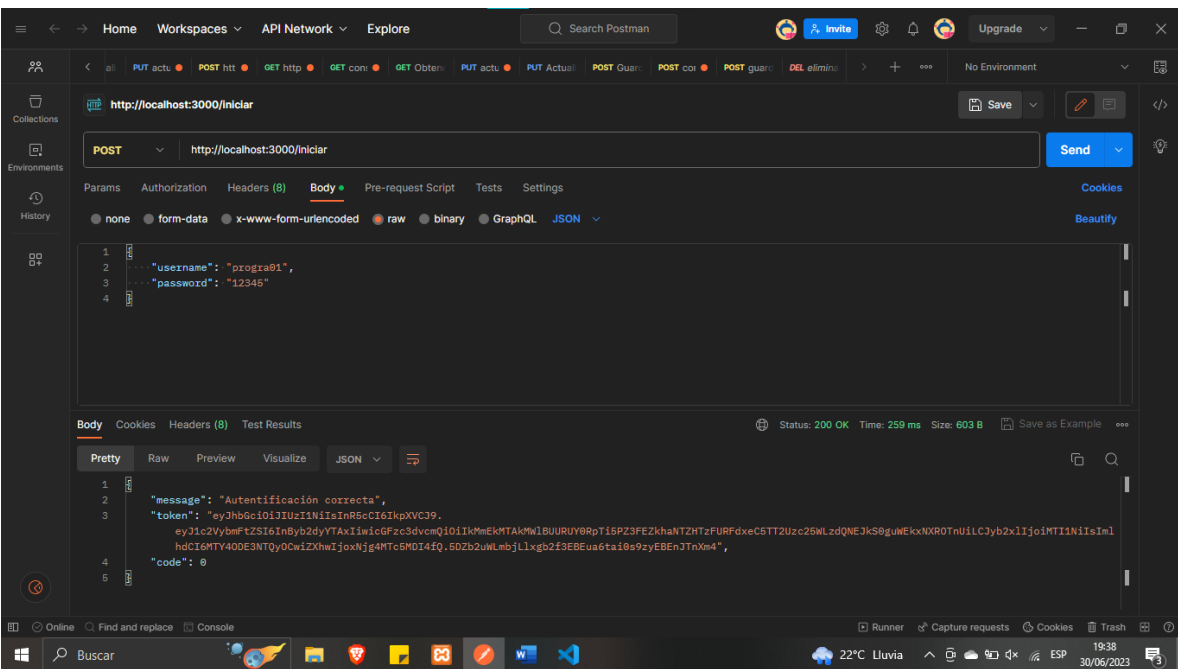
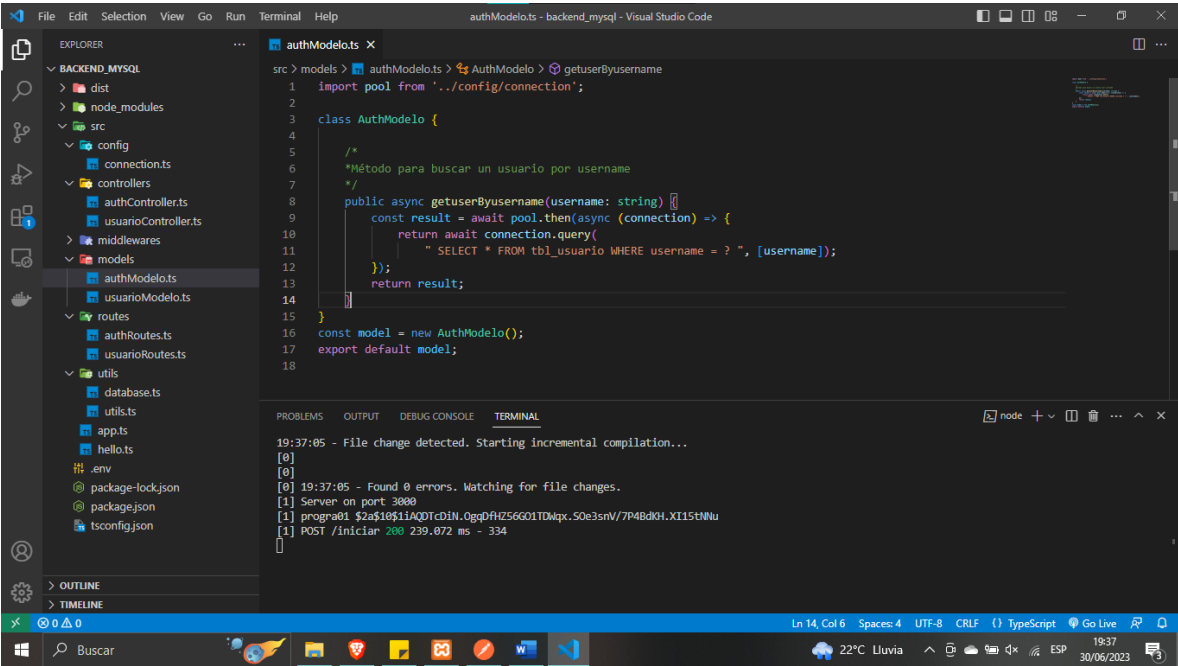
Preview

Visualize

JSON



```
1 {  
2   "message": "Autenticación Correcta",  
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
    eyJ1c2VybmFtZSI6ImByb2dyYTxiIiwicGFzc3dvcmQiOiIkMmEkMTAkyTNVndhdWdXZGVQWhSd1QyNjR3ZWdiSmtdseXhtDnZ4Vz  
    Jld3hlZXVtbi9lYWRtaW4iLCJyb2x1IjoieWRTaw4iLCJpYXQiojE2ODc4ODU1NjUsImV4cCI6MTY4Nzg0TE2NX0.  
    24Ku0_I5xv6NoRhxiuZ-izS8j5U1Sp1hdoD_t3EY6V0",  
4   "code": 0  
5 }
```

## Creación de Backend con Node Express



Ingresa al debugger de JWT



 Debugger Libraries Introduction Ask Crafted by  by Okta

### Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InByb2dyYTAxiiwicGFzc3dvc  
mQiOiIkMmEkMTAKYTN4VndhZWdXZ0VGQWwhSd1Qy  
NjR3ZWdISmtFd1RyTTdseXh0TnZ4VzJUd3hlZXV  
tNi9IWk8iLCJyb2x1IjoieWRTaW4iLCJpYXQiOj  
E2ODc4ODQ5NTQ5ImV4cCI6MTY4Nzg4ODU1NH0.I  
iZCugB3t8kvd01_06BaeF3vPT8F8XEgmbmdA5XU  
Ips
```

### Decoded



EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{  
  "username": "progra01",  
  "password":  
"$2a$10$a3xVwaegWgEFAhRwT264wegHJkEwTrM71yxCNvxW2Twxeeu  
m6/HZ0",  
  "role": "admin",  
  "iat": 1687884954,  
  "exp": 168788554  
}
```

 Depurador Bibliotecas Introducción Preguntar Hecho a mano por  by Okta

### Codificado

PEGUE UN TOKEN AQUÍ

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InByb2dyYTAxiiwicGFzc3dvc  
mQiOiIkMmEkMTAKMWlBUURUY0RpTi5PZ3FEZkha  
NTZHTzFURFdxZC5TT2Uzc25WLzdQNEJkS0guWEk  
xNXROTnUjLCJyb2x1IjoieWRTaW4iLCJpYXQiOj  
Y4ODE3NTQyOCwiZXhwIjojNjg4MTc5MDI4fQ.5D  
Zb2uWLMbjLlXgb2f3EBEua6tai0s9zyEBEnJTNX  
m4
```

### Decodificado

EDITAR LA CARGA ÚTIL Y EL SECRETO

CABEZERO: TIPO DE ALGORITMO Y TOKEN

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

CARGA DE PAGO: DATOS

```
{  
  "username": "progra01",  
  "password":  
"$2a$10$11AQDTcDiN.OgqDFHZ56001TDWqx.S0e3snV/7P4BdKH.XI  
15tNnu",  
  "role": "1256",  
  "iat": 1688175428,  
  "exp": 1688179028  
}
```

FIRMA VERIFICAR

22°C Lluvia 19:38 30/06/2023