

- Instalar typeScript

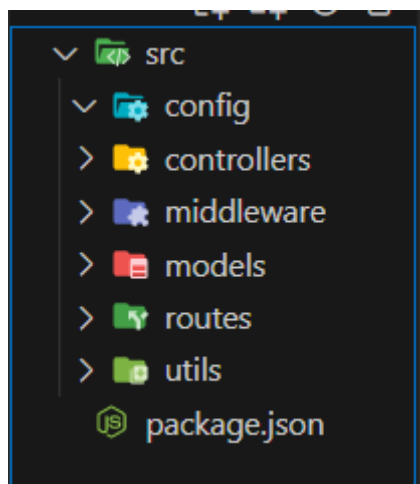
```
C:\Users\52418\Documents\utng\Materias\desarrollo web integral\Unidad 3\BackEnd-app> tsc -v
Version 5.1.3
```

- Dentro de la carpeta ejecutar el siguiente comando para creación de archivo
npm init -y

```
PS C:\Users\52418\Documents\utng\Materias\desarrollo web integral\Unid
3\BackEnd-app> npm init -y
Wrote to C:\Users\52418\Documents\utng\Materias\desarrollo web integra
Unidad 3\BackEnd-app\package.json:

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

- Crear la estructura de carpetas de nuestra aplicación



- Manejar [TSC](#)
- Configurar con [Visual Studio Code](#)

Crear un nuevo **tsconfig.ts** con el comando **tsc -init**

```
PS C:\Users\52418\Documents\utng\Materias\desarrollo web integral\Unidad 3\BackEnd-app> 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>

- Abrir archivo tsconfig.json y modificar las propiedades

```
// "maxNodeModuleJsDepth": 1,

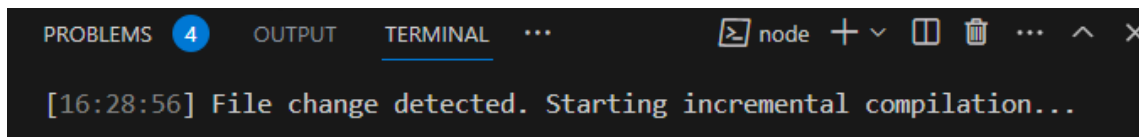
/* Emit */
// "declaration": true,
// "declarationMap": true,
// "emitDeclarationOnly": true,
"sourceMap": true,
// "inlineSourceMap": true,
// "outFile": "./",
"outDir": "./build",
// "removeComments": true,
// "noEmit": true,
// "importHelpers": true,
// "importsNotUsedAsValues": "remove",
// "downlevelIteration": true,
// "sourceRoot": "",
```

- Modificar el archivo package.json

```
Debug
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "build": "tsc -w"
},
"keywords": [],
"author": "",
"license": "ISC"
```

- Ejecutar el comando

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

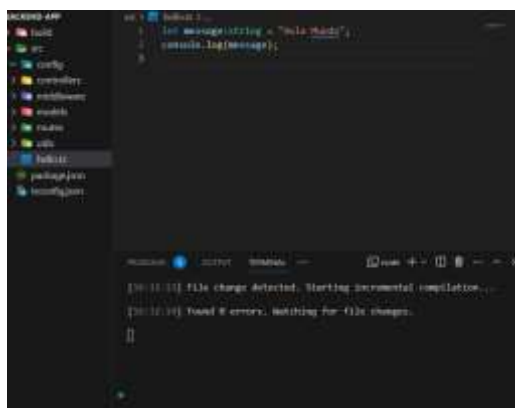
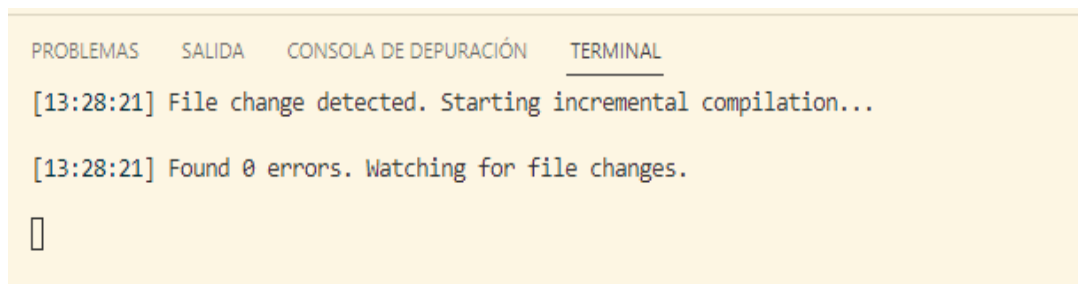


- Crear un archivo `src/hello.ts`

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



- Verificar que no existan errores



- Instalar dependencias

```
npm install -g promise-mysql express nodemon morgan cors crypto-js jsonwebtoken validator
```

```
PS C:\Users\52418\Documents\utng\Materias\desarrollo web i
ntegral\Unidad 3\BackEnd-app> npm install -g promise-mysql
express nodemon morgan cors crypto-js jsonwebtoken valida
tor

added 131 packages in 9s

11 packages are looking for funding
  run `npm fund` for details
PS C:\Users\52418\Documents\utng\Materias\desarrollo web i
ntegral\Unidad 3\BackEnd-app>
PS C:\Users\52418\Documents\utng\Materias\desarrollo web i
ntegral\Unidad 3\BackEnd-app> █
```

- Crear archivo config|dbmysql

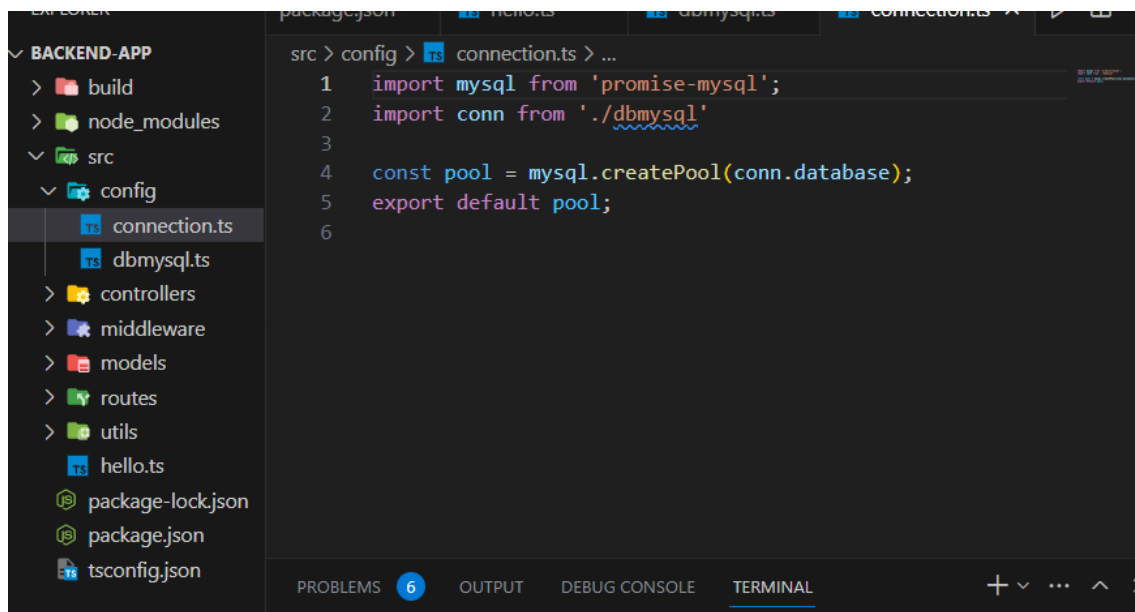
```
export default {
  database: {
    host: 'localhost',
    port: 3306,
    user: 'admin',
    password: 'admin',
    database: 'web_integral'
  }
}
```

```
src > config > TS dbmysql.ts > default > database > database
1  export default {
2    database: {
3      host: 'localhost',
4      port: 3306,
5      user: 'admin',
6      password: 'admin',
7      database: 'web_integral'
8    }
9  }
10
```

Connection config|connection.ts

```
import mysql from 'promise-mysql';  
  
import keys from '../config/keys';  
  
const pool = mysql.createPool(keys.database);  
  
export default pool;
```

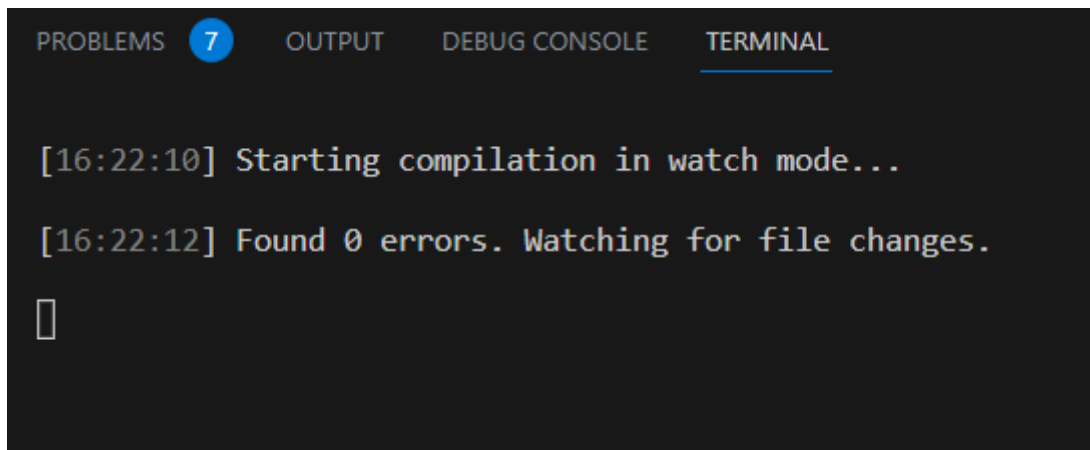
```
PS C:\Users\52418\Documents\utng\Materias\desarrollo web i  
ntegral\Unidad 3\BackEnd-app> npm i promise-mysql  
  
added 16 packages, and audited 17 packages in 4s  
  
found 0 vulnerabilities  
PS C:\Users\52418\Documents\utng\Materias\desarrollo web i  
ntegral\Unidad 3\BackEnd-app> █
```



Crear la table usuarios

```
1 create database webIntegral;  
2  
3 use webIntegral;  
4  
5 CREATE TABLE Usuarios (  
6     id INT AUTO_INCREMENT PRIMARY KEY,  
7     username VARCHAR(30) NOT NULL,  
8     password VARCHAR(20) NOT NULL,  
9     name varchar(50) not null  
10 )  
11  
12 select * from Usuarios;
```

- Ejecutar el comando



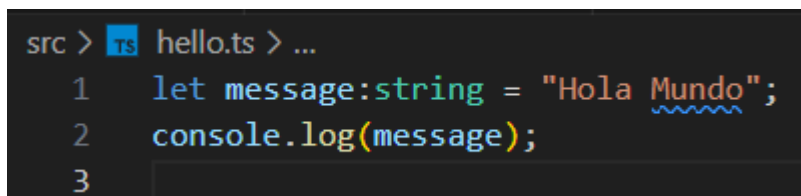
```
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL


[16:22:10] Starting compilation in watch mode...

[16:22:12] Found 0 errors. Watching for file changes.

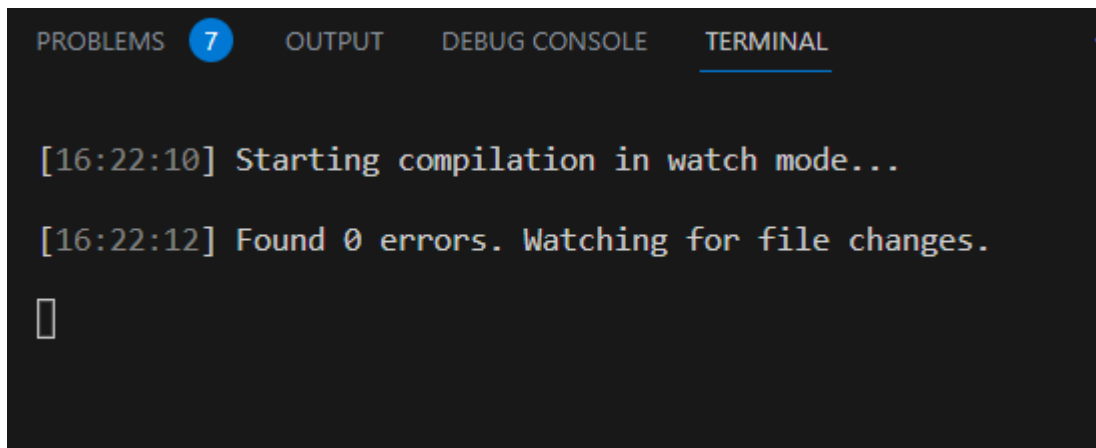
█
```

- Crear un archivo `src/hello.ts`



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

- Verificar que no existan errores



```
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL

[16:22:10] Starting compilation in watch mode...

[16:22:12] Found 0 errors. Watching for file changes.

█
```

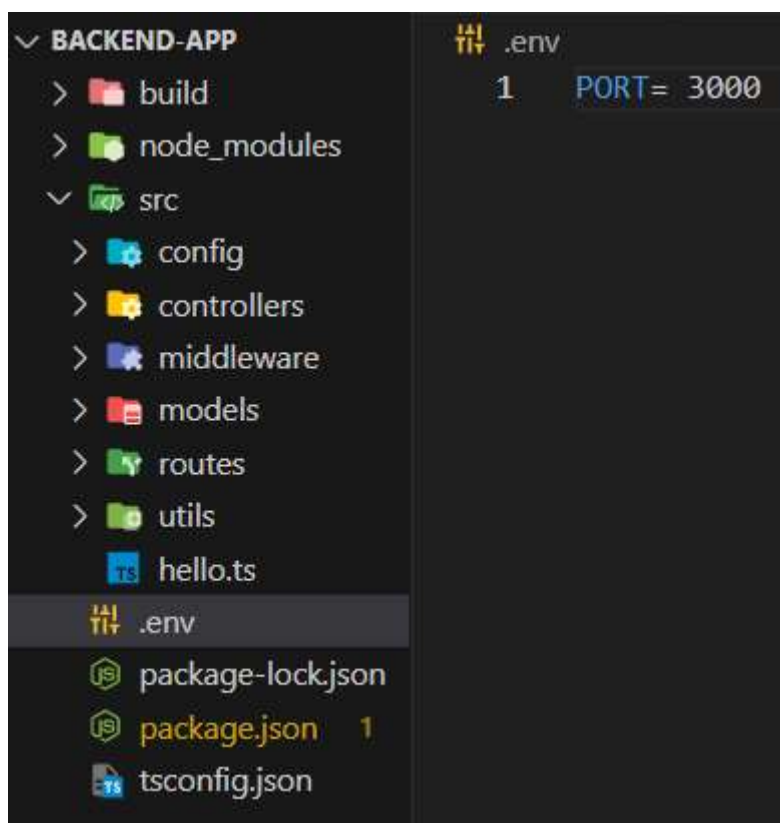
- Modificar el archivo `package.json` para agregar script `start`

```

package.json > {} scripts
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "start": "node build/app.js",
9     "build": "tsc -w"
10  },
11  "keywords": [],
12  "author": "",
13  "license": "ISC",
14  "dependencies": {
15    "promise-mysql": "^5.2.0"
16  }
17 }
18

```

- Crear el archivo .env

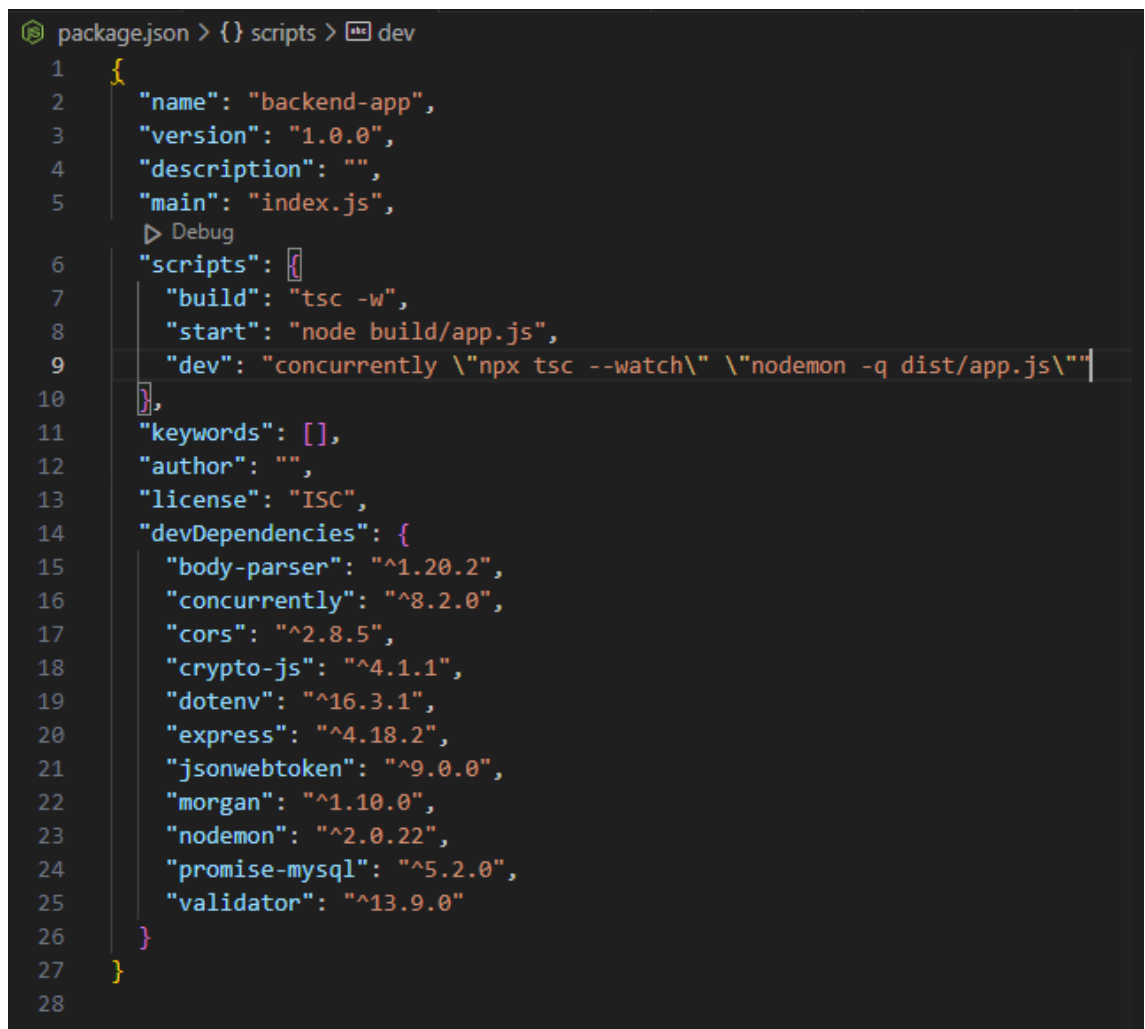


- Crear archivo src/app.ts, es nuestro punto de inicio de nuestra aplicación



```
src > app.ts > ...
1  import express from 'express';
2  const app = express();
3
4  /* DotEnv */
5  const dotenv = require('dotenv');
6  dotenv.config();
7  const port = process.env.PORT;
8
9  app.get('/', (req, res) => {
10     res.send('Hello World!');
11 });
12
13 app.listen(port, () => {
14     return console.log(`Express is listening at http://local
15 });
16
```

- Modificar el archivo package.json



```
package.json > {} scripts > dev
1  {
2    "name": "backend-app",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "build": "tsc -w",
8      "start": "node build/app.js",
9      "dev": "concurrently \"npx tsc --watch\" \"nodemon -q dist/app.js\"",
10    },
11    "keywords": [],
12    "author": "",
13    "license": "ISC",
14    "devDependencies": {
15      "body-parser": "^1.20.2",
16      "concurrently": "^8.2.0",
17      "cors": "^2.8.5",
18      "crypto-js": "^4.1.1",
19      "dotenv": "^16.3.1",
20      "express": "^4.18.2",
21      "jsonwebtoken": "^9.0.0",
22      "morgan": "^1.10.0",
23      "nodemon": "^2.0.22",
24      "promise-mysql": "^5.2.0",
25      "validator": "^13.9.0"
26    }
27  }
28
```

pap

- Ejecutar el comando

[npm run dev](#)


```

17:00:09 - Starting compilation in watch mode...
[0]
[0]
[0] 17:00:14 - Found 0 errors. Watching for file changes.
[1] Express is listening at http://localhost:3000

```

- Modificar archivo [app.ts](#) con el uso de clases

```

src > app.ts > Server > routes
22   this.app.listen(this.app.get("port"), () => {
23       console.log("Server on port", this.app.get("port"));
24   });
25   }
26
27
28   //Configuración de módulos
29   config(): void {
30       // configuración del puerto para el servidor
31       this.app.set("port", 3000);
32
33
34       // muestra las peticiones en consola
35       this.app.use(morgan("dev"));
36
37       // puertos de conexión de la API
38       this.app.use(cors());
39
40       // solo se permiten peticiones en formato JSON
41       this.app.use(bodyParser.json());
42       this.app.use(bodyParser.urlencoded({ extended: false, }));
43   };
44   }
45
46   //Configura las rutas
47   routes() {
48       this.app.use("/", (req, res) => {
49           res.send("Invocando autenticacion")
50       });
51   };
52   }
53
54   const server = new Server();

```

PROBLEMS 27 OUTPUT DEBUG CONSOLE TERMINAL

[1] Server on port 3000

Definiendo rutas

Referencia

- Ubicar carpeta de rutas [src/routes](#)
- Crear archivo [src/routes/authRoutes.ts](#)

Definir la clase para rutas de autenticación

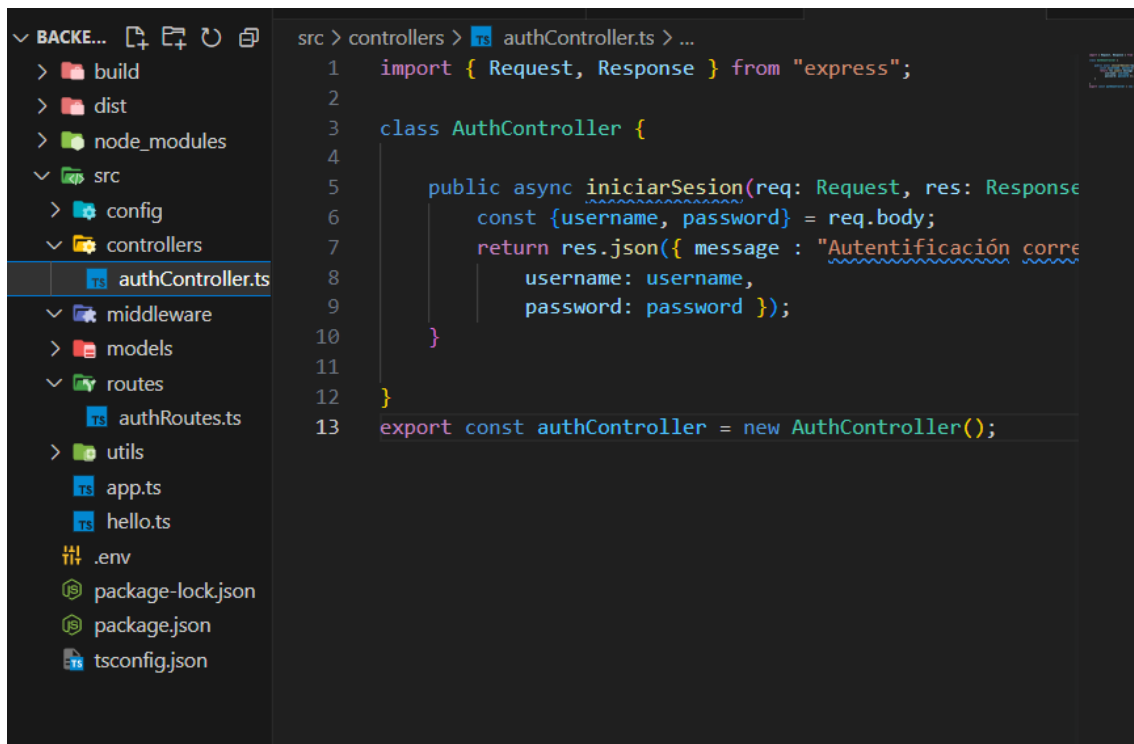
Definir método de configuración e inicializar la clase

```
1  import { Router,RouterOptions } from "express";
2
3  class AuthRoutes{
4      public router:Router
5
6      constructor(){
7          this.router = Router()
8          this.config()
9      }
10     config(){
11         this.router.get('/',(req,res)=>{
12             res.send('Inboca Autenticacion')
13         })
14     }
15 }
16
17 const authRoutes = new AuthRoutes()
18 export default authRoutes.router;
19
```

Definiendo controlador

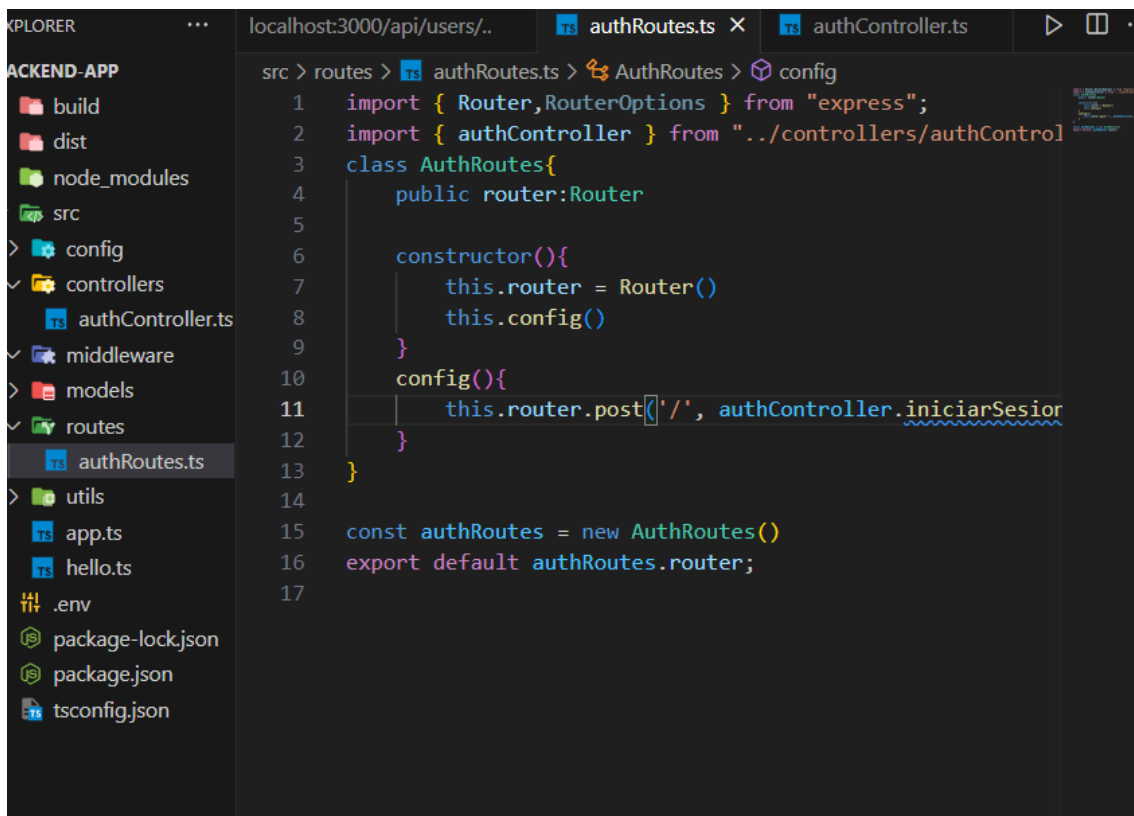
Ubicar carpeta src|controllers

Crear archivo src|controllers|authController.ts



```
src > controllers > authController.ts > ...
1  import { Request, Response } from "express";
2
3  class AuthController {
4
5      public async iniciarSesion(req: Request, res: Response) {
6          const {username, password} = req.body;
7          return res.json({ message : "Autenticación correcta",
8                          username: username,
9                          password: password });
10     }
11 }
12
13 export const authController = new AuthController();
```

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



```
src > routes > authRoutes.ts > AuthRoutes > config
1  import { Router, RouterOptions } from "express";
2  import { authController } from "../controllers/authController";
3  class AuthRoutes{
4      public router:Router
5
6      constructor(){
7          this.router = Router()
8          this.config()
9      }
10     config(){
11         this.router.post('/', authController.iniciarSesion);
12     }
13 }
14
15 const authRoutes = new AuthRoutes()
16 export default authRoutes.router;
17
```

Verificar envío de datos nuevamente con Postman

POST ⌵http://localhost:3000/

Send

Query

Headers ²

Auth

Body ¹

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

1 {

2 "username": "juan",

3 "password": "12345"

4

5 }

Status: 200 OK

Size: 76 Bytes

Time: 56 ms

Response

Headers ⁷

Cookies

Results

Docs

{}

≡

1 {

2 "message": "Autenticación correcta",

3 "username": "juan",

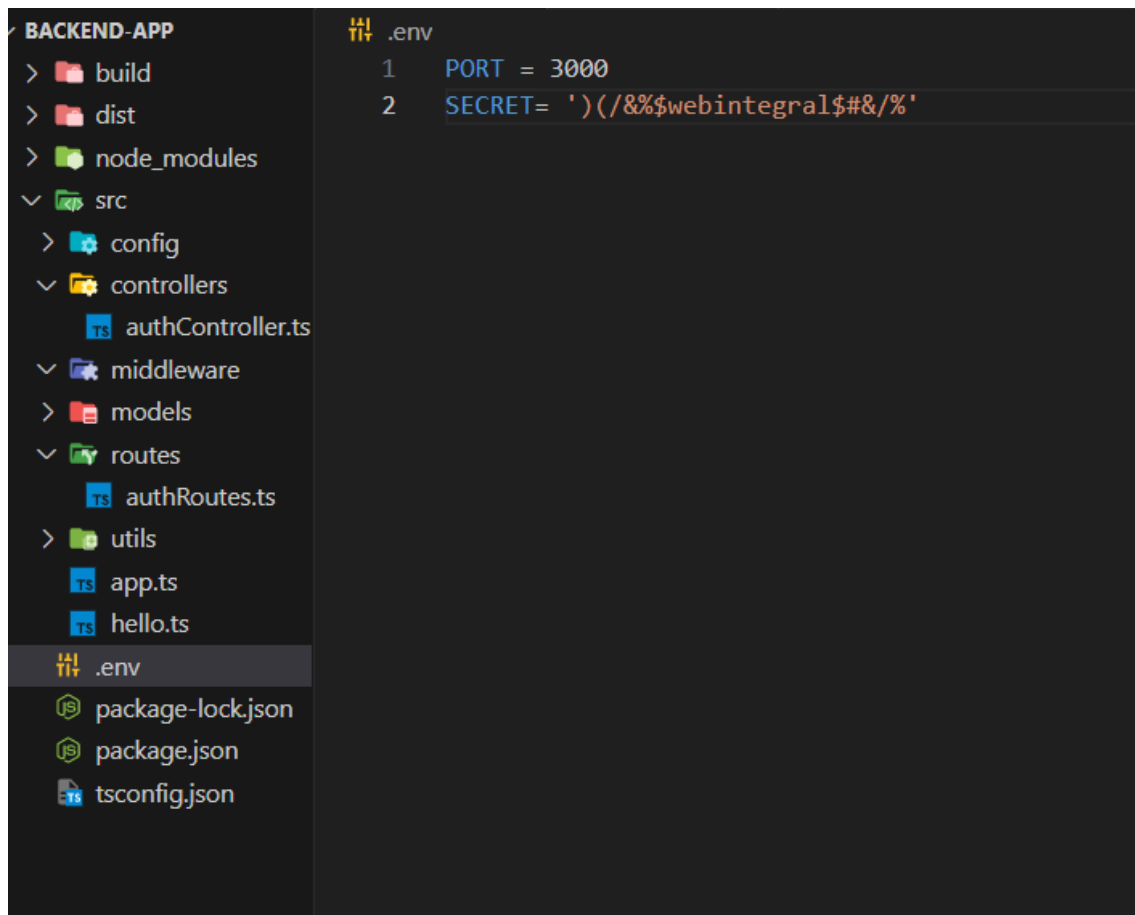
4 "password": "12345"

5 }

Copy

Configurar base de datos

Definir clave secreta en archivo .env



Crear una base de datos en MySQL llamada web_integral

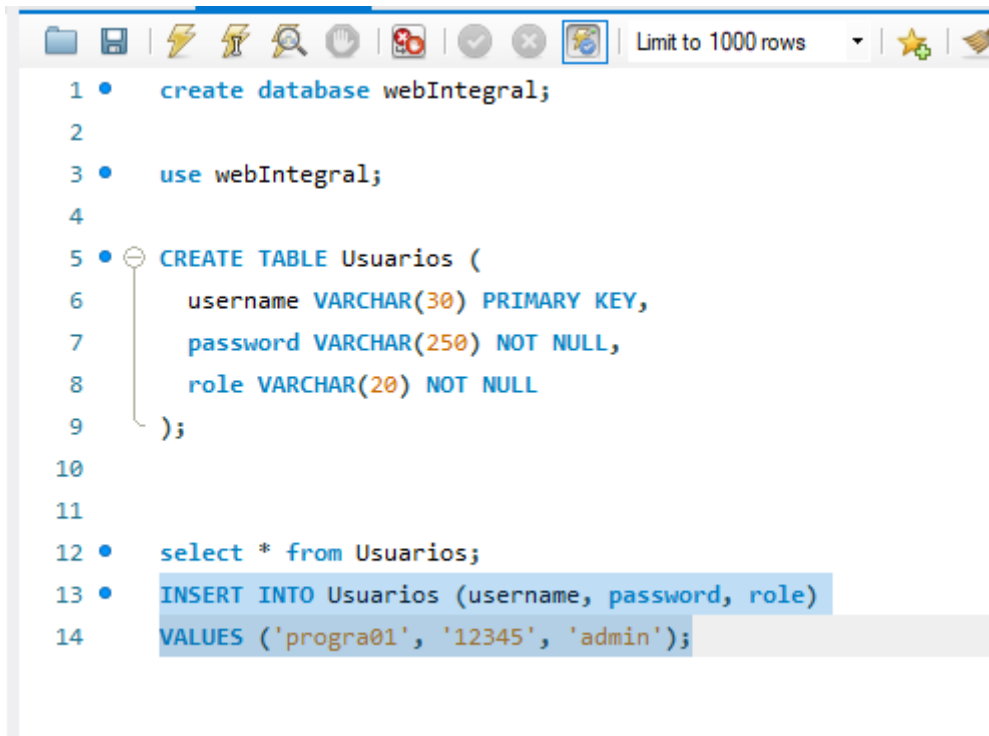
Crear una tabla llamada tbl_usuario

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



```
1 • create database webIntegral;
2
3 • use webIntegral;
4
5 • CREATE TABLE Usuarios (
6     username VARCHAR(30) PRIMARY KEY,
7     password VARCHAR(250) NOT NULL,
8     role VARCHAR(20) NOT NULL
9 );
10
11
12 • select * from Usuarios;
13 • INSERT INTO Usuarios (username, password, role)
14 VALUES ('progra01', '12345', 'admin');
```

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

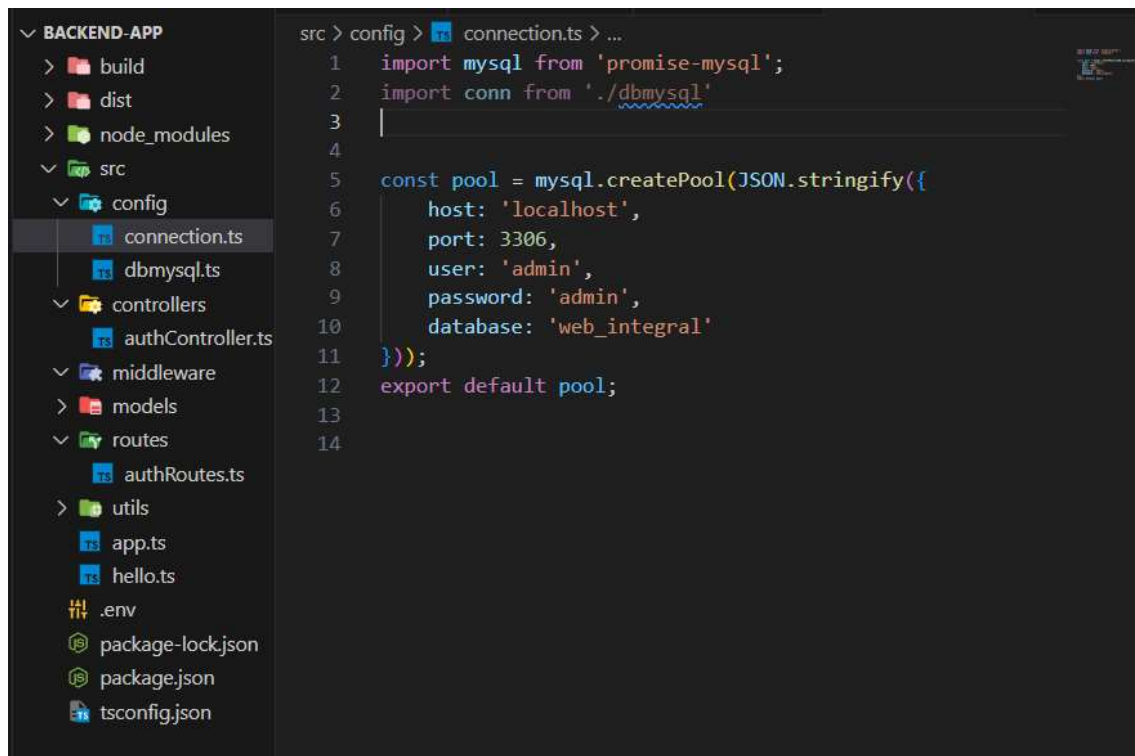
`ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'root';`

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;
```

```

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

```

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}` });
    }
}

```

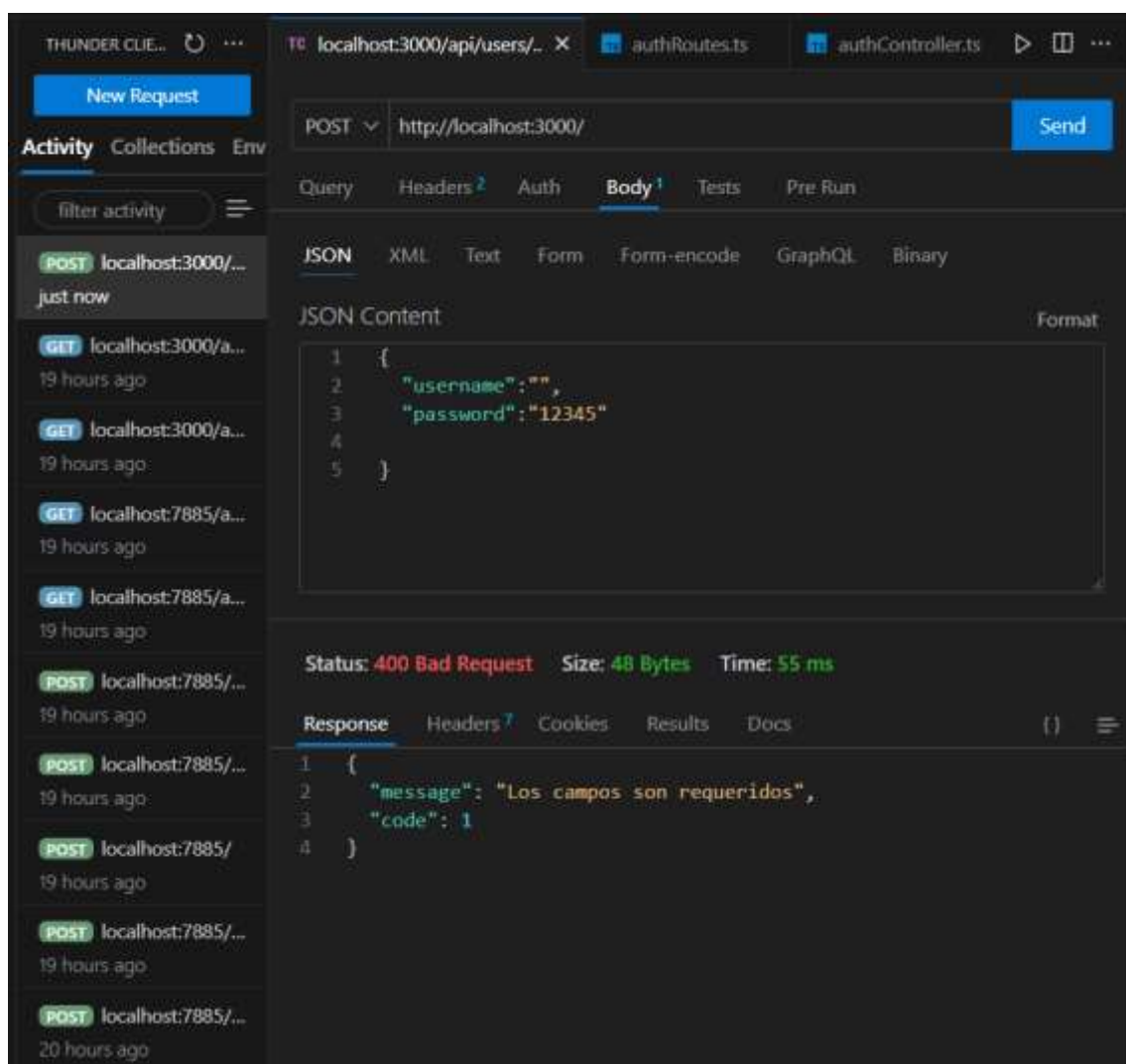
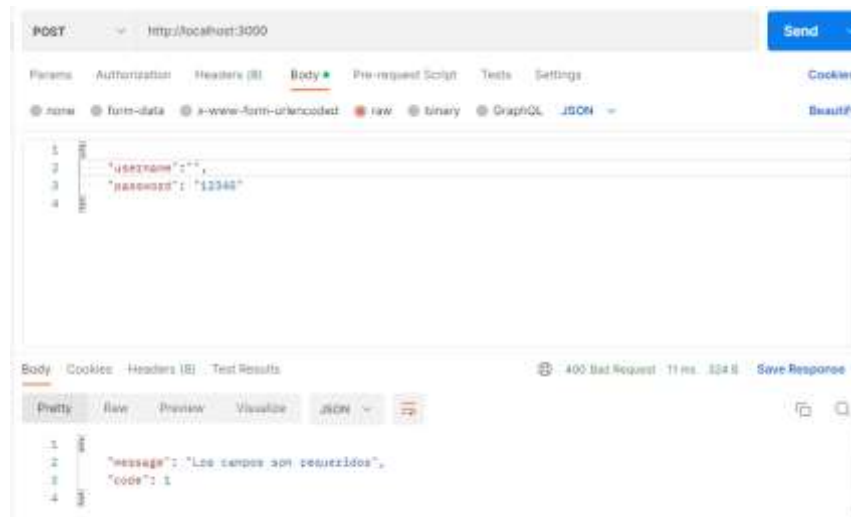


```

src > controllers >  authController.ts > ...
1  import { Request, Response } from "express";
2  import validator from 'validator';
3  import model from '../models/authModelo';
4  class AuthController {
5
6      /**
7       * Método para validar Inicio de sesión
8       * @param req Petición
9       * @param res respuesta
10      * @returns
11      */
12      public async iniciarSesion(req: Request, res: Response) {
13
14          try {
15              const {username, password }= req.body;
16
17              // verificar que los datos no esten vacios
18              if (validator.isEmpty(username.trim()) ||
19                  validator.isEmpty(password.trim())) {
20                  return res
21                      .status(400)
22                      .json({ message: "Los campos son requeridos"
23                  })
24
25                  return res.json({ message : "Autenticación correcta"
26
27              } catch (error: any) {
28                  return res.status(500).json({ message : ` ${err
29              }
30          }
31
32      }
33  }
34  export const authController = new AuthController();
35

```

Verificar con Postman



Definiendo el acceso a los datos de datos

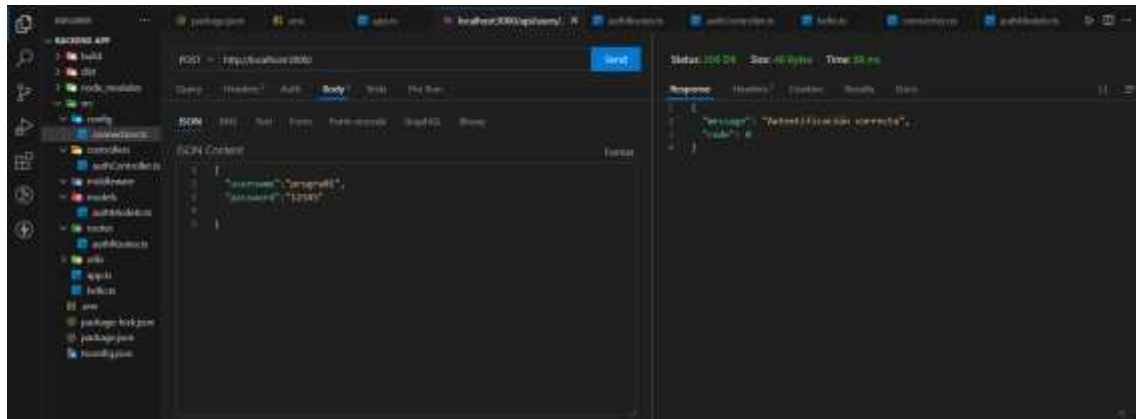
```
import validator from 'validator';
import model from '../models/authModelo';
```

```
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 });

```



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();

```

Realizar las pruebas con Post Eliminar



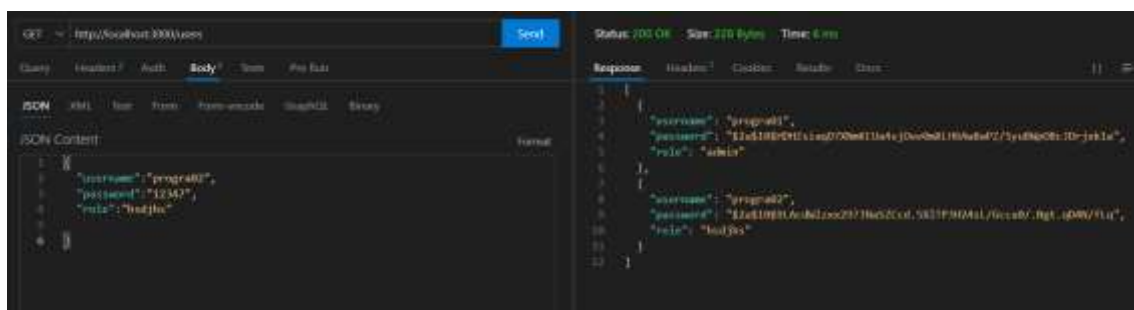
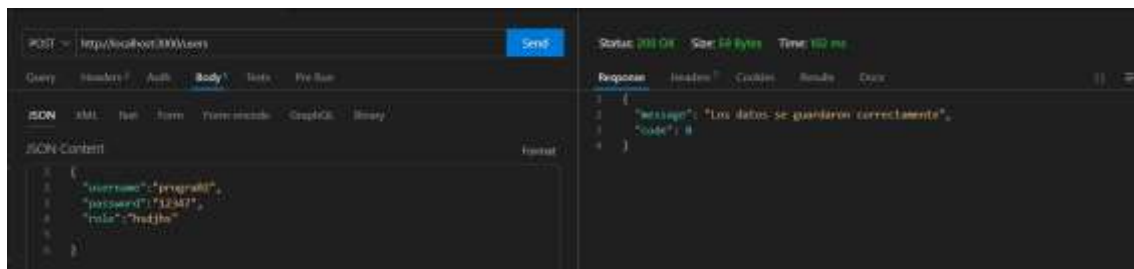
Select



Actualizar



Insertar



Se termino en clase.

Últimos pasos jwt

JWT en AuthController

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

```
import { Request, Response } from "express";
import validator from "validator";
import model from "../models/authModelo";
import jwt from 'jsonwebtoken';
import db from '../config/database'
```

Definir la clave secreta para el JWT

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

Decoded _____

[illegible]