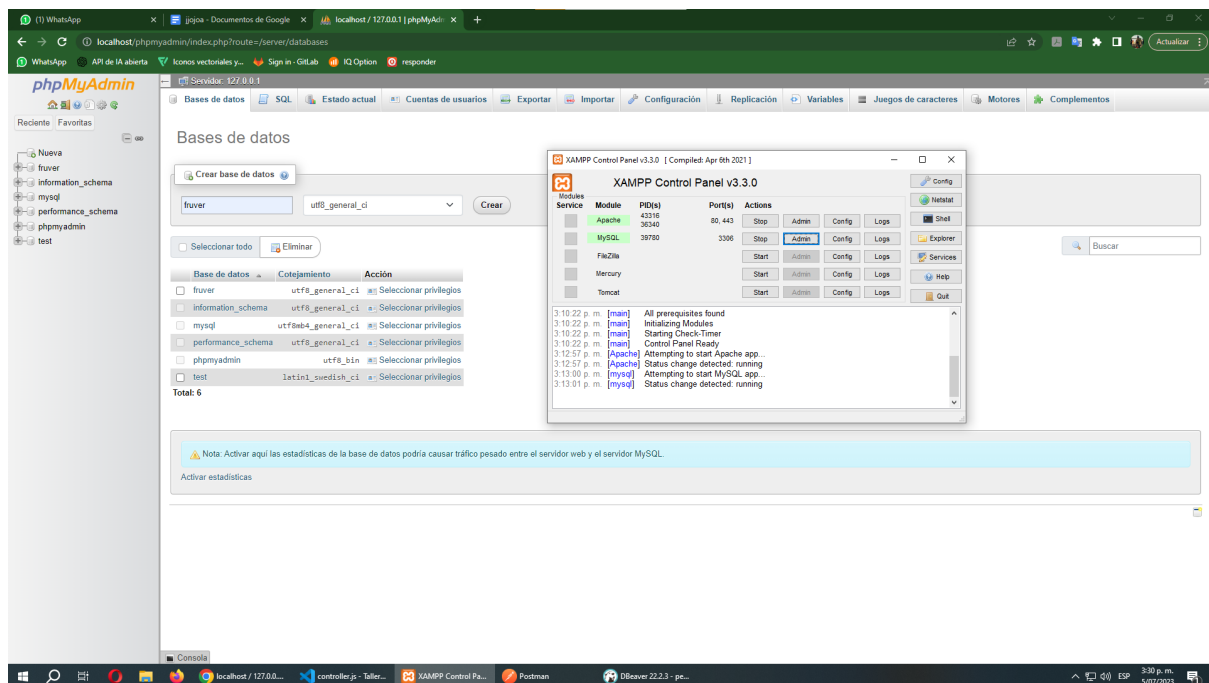
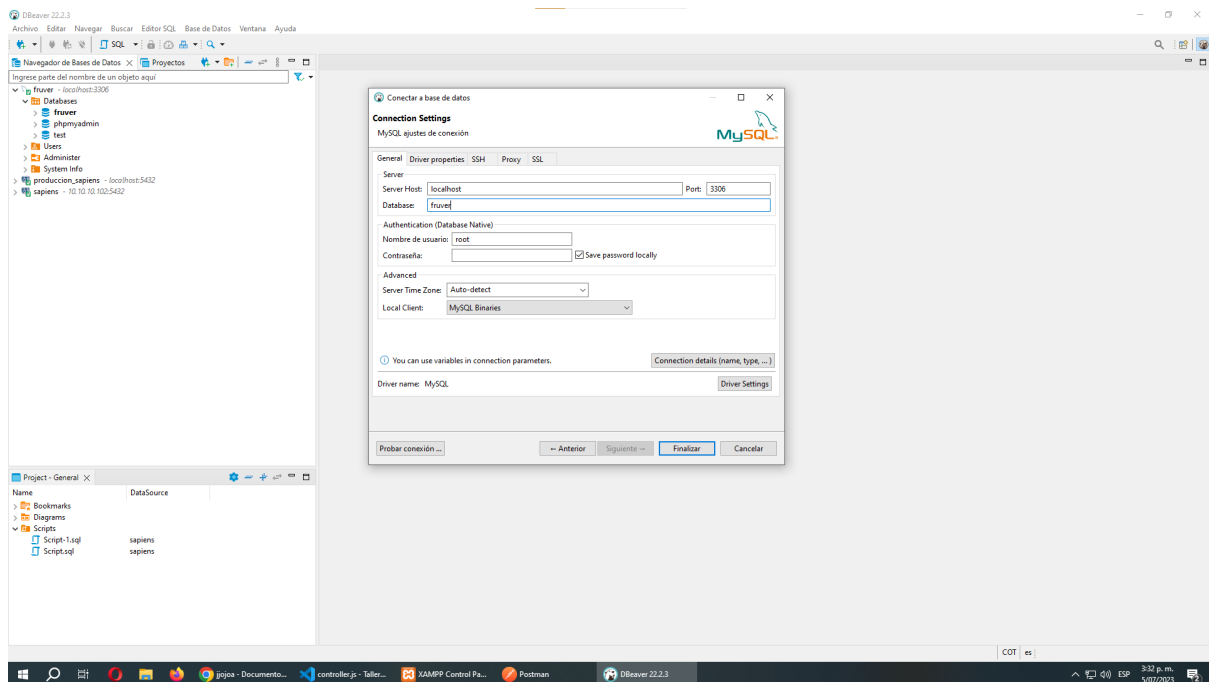


Para el desarrollo de esta actividad necesitaremos XAMPP y Dbeaver para la administración de la base de datos.

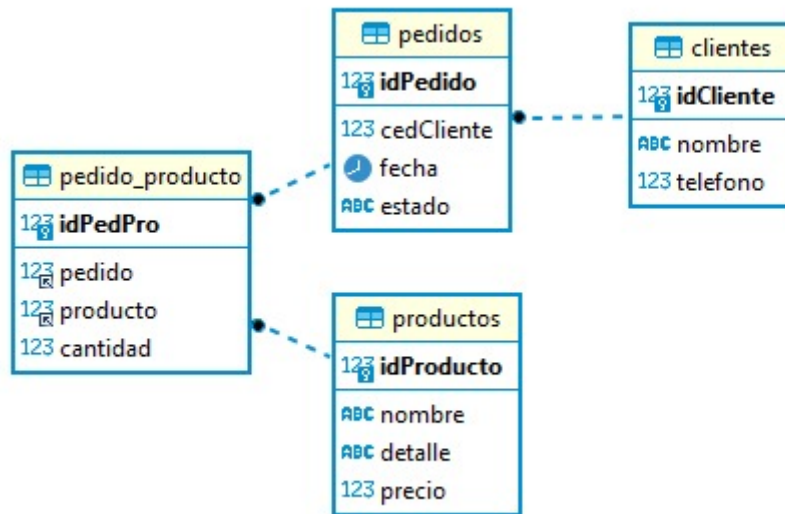
Iniciamos el servicio mysql en xampp y presionamos en admin para crear la base de datos fruver



En Dbeaver creamos la conexión con la base de datos fruver



Creamos las respectivas tablas en la base de datos y sus reacciones



DBavear 22.2.3 - pedido_producto

Archivo Editar Navegar Buscar Editor SQL Base de Datos Ventana Ayuda

Ingrese parte del nombre de un objeto aquí

Database: fruver Tables: productos clientes pedidos pedido_producto

Table Name: pedido_producto Engine: InnoDB Auto Increment: 1 Charset: utf8 Collation: utf8_general_ci

Column Name	#	Data Type	Not Null	Auto Increment	Key	Default	Extra	Expression	Comment
idPedPro	1	int(11)	[v]	[v]	PRI				
pedido	2	int(11)	[v]	[v]	MUL		auto_increment		
producto	3	int(11)	[v]	[v]	MUL				
cantidad	4	int(11)	[v]	[v]					

Columns: idPedPro, pedido, producto, cantidad

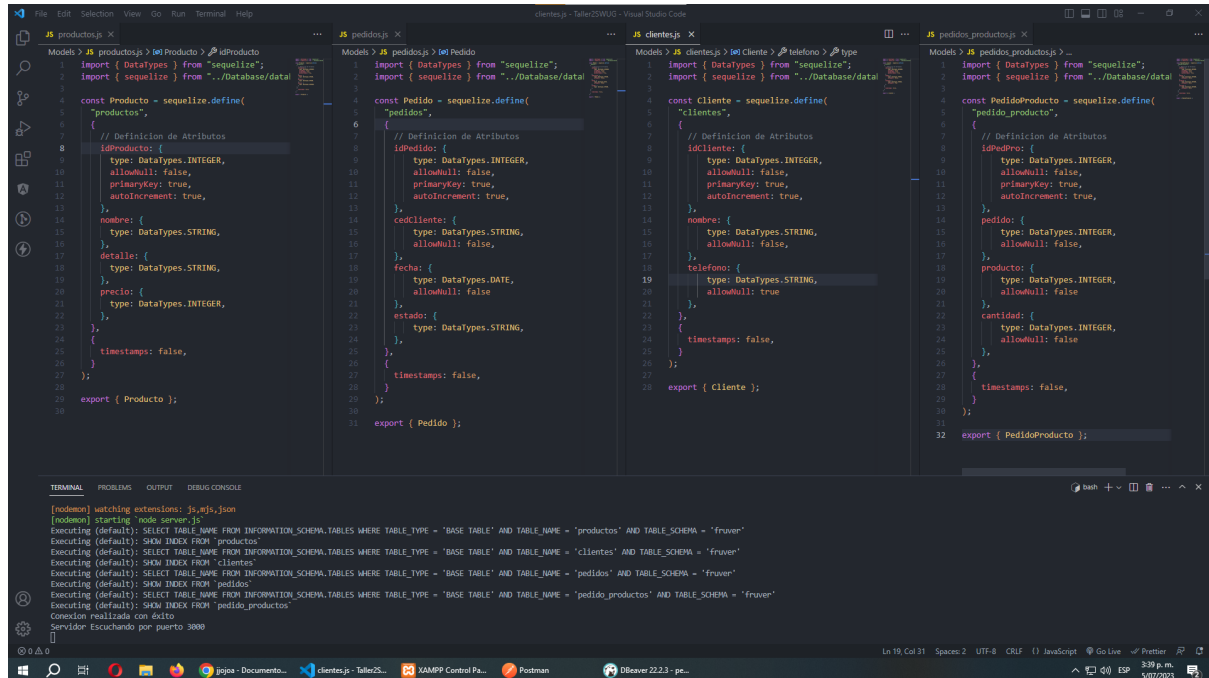
Constraints: Foreign Keys, Triggers, Indexes, Partitions, Statistics, Virtual

CTOT es

3:34 p.m. 5/07/2023

Desarrollar una aplicación backend implementada en node js, usando la estructura vista en clase.

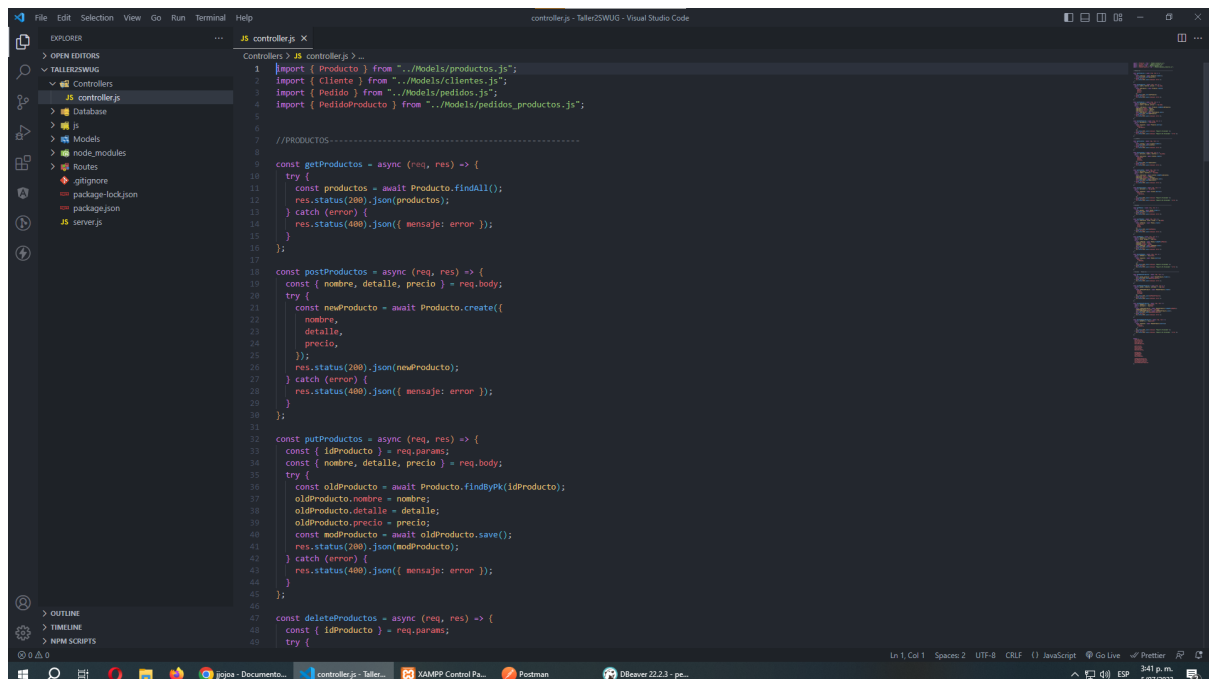
Se define cada tabla en un modelo usando sequelize, un archivo js para cada tabla.



```
Modelo > JS productos.js > idProducto
1 import { DataTypes } from 'sequelize';
2 import { sequelize } from '../Database/data';
3
4 const Producto = sequelize.define(
5   'productos',
6   {
7     // Definición de Atributos
8     idProducto: {
9       type: DataTypes.INTEGER,
10      allowNull: false,
11      primaryKey: true,
12      autoIncrement: true,
13    },
14     nombre: {
15       type: DataTypes.STRING,
16     },
17     detalle: {
18       type: DataTypes.STRING,
19     },
20     precio: {
21       type: DataTypes.INTEGER,
22     },
23     timestamps: false,
24   },
25   {
26     timestamps: false,
27   });
28 export { Producto };
29
Modelo > JS pedidos.js > idPedido
1 import { DataTypes } from 'sequelize';
2 import { sequelize } from '../Database/data';
3
4 const Pedido = sequelize.define(
5   'pedidos',
6   {
7     // Definición de Atributos
8     idPedido: {
9       type: DataTypes.INTEGER,
10      allowNull: false,
11      primaryKey: true,
12      autoIncrement: true,
13    },
14     nombre: {
15       type: DataTypes.STRING,
16     },
17     fecha: {
18       type: DataTypes.DATE,
19     },
20     estado: {
21       type: DataTypes.STRING,
22     },
23     timestamps: false,
24   },
25   {
26     timestamps: false,
27   });
28 export { Pedido };
29
Modelo > JS pedidos_productos.js > idPedidoPro
1 import { DataTypes } from 'sequelize';
2 import { sequelize } from '../Database/data';
3
4 const PedidoProducto = sequelize.define(
5   'pedido_producto',
6   {
7     // Definición de Atributos
8     idPedidoPro: {
9       type: DataTypes.INTEGER,
10      allowNull: false,
11      primaryKey: true,
12      autoIncrement: true,
13    },
14     nombre: {
15       type: DataTypes.STRING,
16     },
17     producto: {
18       type: DataTypes.INTEGER,
19     },
20     cantidad: {
21       type: DataTypes.INTEGER,
22     },
23     timestamps: false,
24   },
25   {
26     timestamps: false,
27   });
28 export { PedidoProducto };
29
```

```
Terminal
[Node] watching extensions: js, json
[Node] starting 'node server.js'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'productos'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'pedidos'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'pedido_producto'
Conexion realizada con éxito
Servidor escuchando por puerto 3000
```

En el archivo controller.js se definen las funciones que se van a utilizar para la gestión de los datos en la base de datos



```
Controller.js
1 import { Producto } from '../Models/productos.js';
2 import { Cliente } from '../Models/clientes.js';
3 import { Pedido } from '../Models/pedidos.js';
4 import { PedidoProducto } from '../Models/pedidos_productos.js';
5
6 //PRODUCTOS-----
7
8 const getProductos = async (req, res) => {
9   try {
10     const productos = await Producto.findAll();
11     res.status(200).json(productos);
12   } catch (error) {
13     res.status(400).json({ mensaje: error });
14   }
15 };
16
17 const postProductos = async (req, res) => {
18   const { nombre, detalle, precio } = req.body;
19   try {
20     const newProducto = await Producto.create({
21       nombre,
22       detalle,
23       precio,
24     });
25     res.status(200).json(newProducto);
26   } catch (error) {
27     res.status(400).json({ mensaje: error });
28   }
29 };
30
31 const putProductos = async (req, res) => {
32   const { idProducto } = req.params;
33   const { nombre, detalle, precio } = req.body;
34   try {
35     const oldProducto = await Producto.findById(idProducto);
36     oldProducto.nombre = nombre;
37     oldProducto.detalle = detalle;
38     oldProducto.precio = precio;
39     const modProducto = await oldProducto.save();
40     res.status(200).json(modProducto);
41   } catch (error) {
42     res.status(400).json({ mensaje: error });
43   }
44 };
45
46 const deleteProductos = async (req, res) => {
47   const { idProducto } = req.params;
48   try {
49     await Producto.destroy({
50       where: { idProducto },
51     });
52     res.status(200).json({ mensaje: 'Producto eliminado' });
53   } catch (error) {
54     res.status(400).json({ mensaje: error });
55   }
56 };
57
```

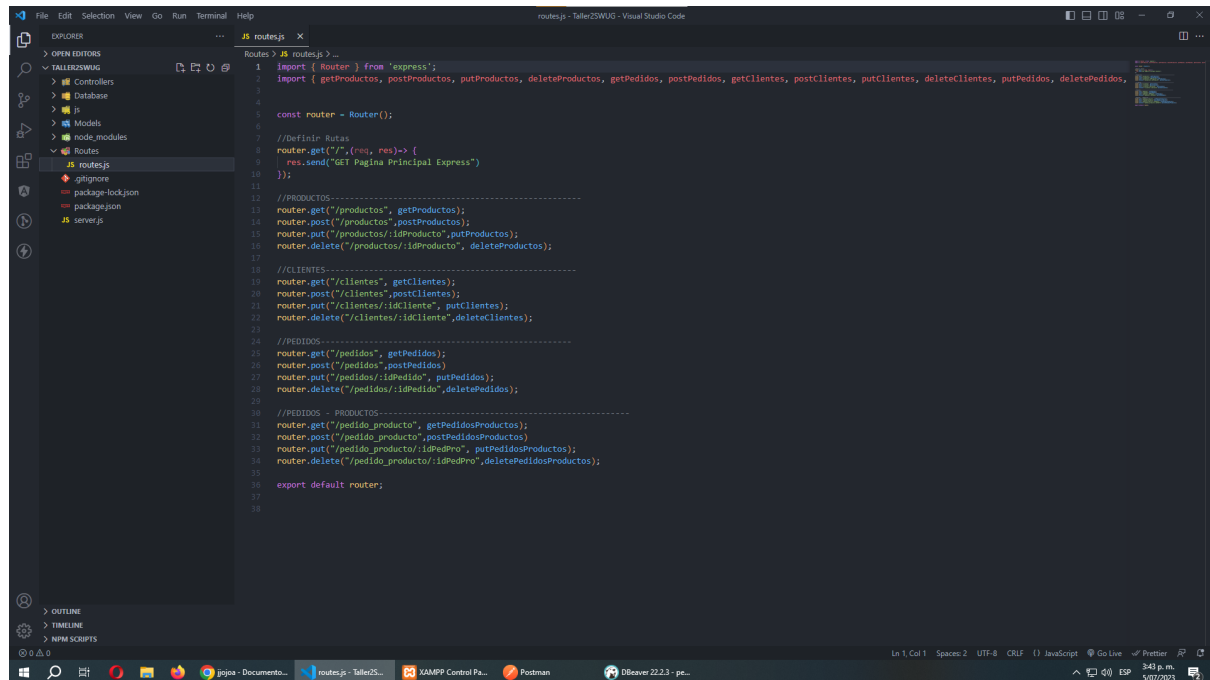
```
Terminal
[Node] watching extensions: js, json
[Node] starting 'node server.js'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'productos'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'pedidos'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'pedido_producto'
Conexion realizada con éxito
Servidor escuchando por puerto 3000
```

```
193 const { idPedido } = req.params;
194 const { cantidad } = req.body;
195 try {
196   const oldPedidoProducto = await PedidoProducto.findById(idPedido);
197   oldPedidoProducto.cantidad = cantidad;
198   const modPedidoProducto = await oldPedidoProducto.save();
199   res.status(200).json(modPedidoProducto);
200 } catch (error) {
201   res.status(400).json({ mensaje: error });
202 }
203 };
204
205 const deletePedidosProductos = async (req, res) => {
206   const { idPedido } = req.params;
207   try {
208     const respuesta = await PedidoProducto.destroy({
209       where: {
210         idPedido,
211       },
212     });
213     res.status(200).json({ mensaje: "Registro Eliminado" });
214   } catch (error) {
215     res.status(400).json({ mensaje: "Registro No Eliminado" + error });
216   }
217 };
218
219
220 export {
221   getProductos,
222   postProductos,
223   putProductos,
224   deleteProductos,
225
226   getClientes,
227   postClientes,
228   putClientes,
229   deleteClientes,
230
231   getPedidos,
232   postPedidos,
233   putPedidos,
234   deletePedidos,
235
236   getPedidosProductos,
237   postPedidosProductos,
238   putPedidosProductos,
239   deletePedidosProductos
240 };
241
```

En el archivo database.js se configura la conexión a la base de datos

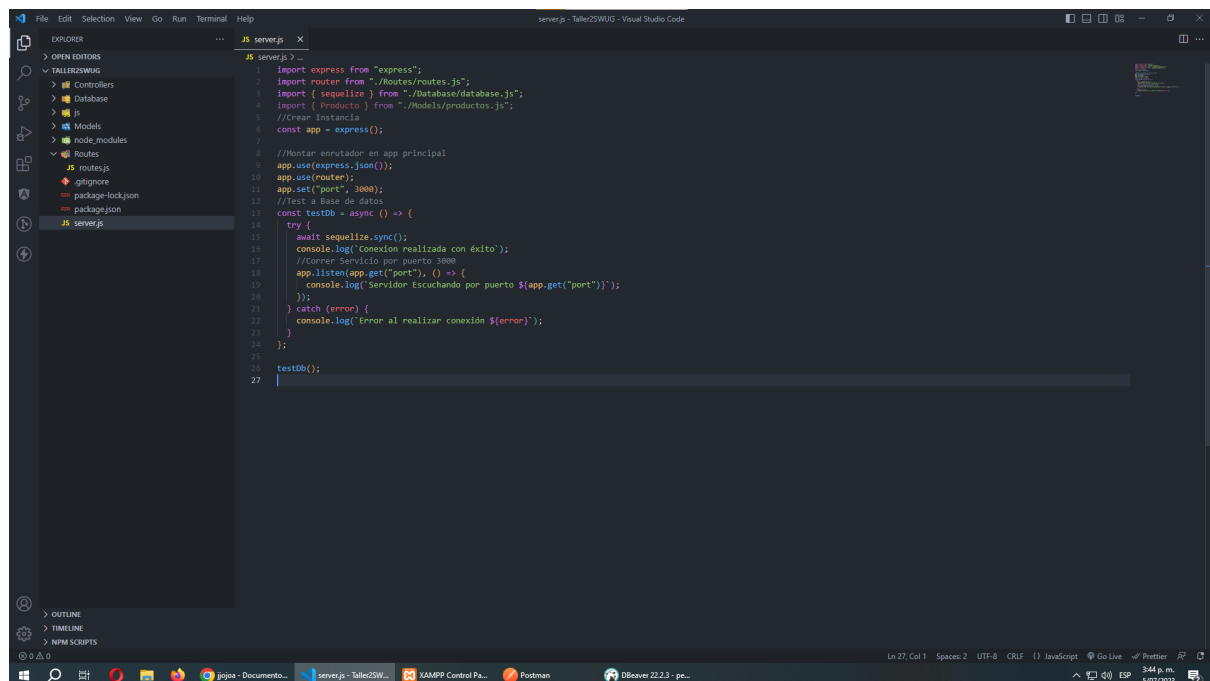
```
1 import Sequelize from "sequelize";
2
3 const sequelize = new Sequelize("fruver", "root", "", {
4   host: "localhost",
5   dialect: "mysql",
6 });
7
8
9 export {
10   sequelize
11 }
12
```

En el archivo routes.js se define las rutas y con la respectiva función que se importa del controlador.



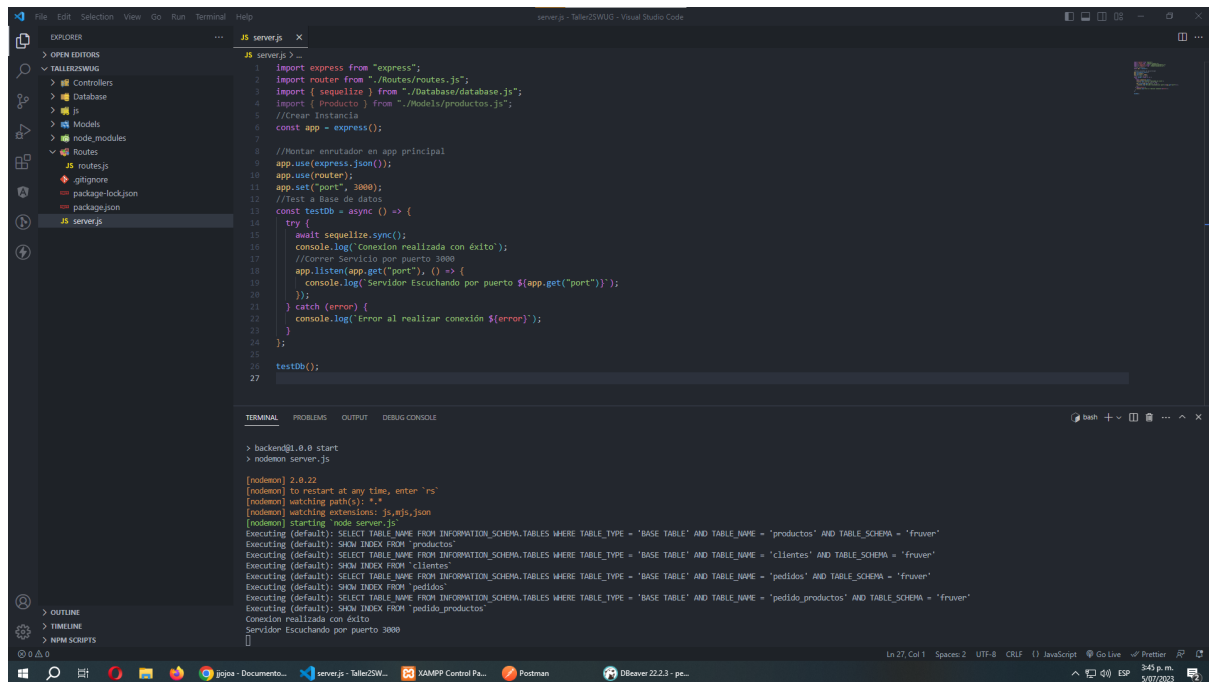
```
1 import { Router } from 'express';
2 import { getProductos, postProductos, putProductos, deleteProductos, getPedidos, postPedidos, getClientes, postClientes, putClientes, deleteClientes, putPedidos, deletePedidos,
3
4
5 const router = Router();
6
7 //Definir Rutas
8 router.get("/", (req, res) => {
9   res.send("GET Pagina Principal Express")
10 });
11
12 //PRODUCTOS-----
13 router.get("/productos", getProductos);
14 router.post("/productos", postProductos);
15 router.put("/productos/:idProducto", putProductos);
16 router.delete("/productos/:idProducto", deleteProductos);
17
18 //CLIENTES-----
19 router.get("/clientes", getClientes);
20 router.post("/clientes", postClientes);
21 router.put("/clientes/:idCliente", putClientes);
22 router.delete("/clientes/:idCliente", deleteClientes);
23
24 //PEDIDOS-----
25 router.get("/pedidos", getPedidos);
26 router.post("/pedidos", postPedidos);
27 router.put("/pedidos/:idPedido", putPedidos);
28 router.delete("/pedidos/:idPedido", deletePedidos);
29
30 //PEDIDOS - PRODUCTOS-----
31 router.get("/pedido_producto", getPedidosProductos);
32 router.post("/pedido_producto", postPedidosProductos);
33 router.put("/pedido_producto/:idPedido", putPedidosProductos);
34 router.delete("/pedido_producto/:idPedido", deletePedidosProductos);
35
36 export default router;
37
38
```

En el archivo server.js se configura el servidor usando el entorno de trabajo Express.



```
1 import express from "express";
2 import router from "../Routes/routes.js";
3 import { sequelize } from "../Database/database.js";
4 import { producto } from "../Models/productos.js";
5 //Crear Instancia
6 const app = express();
7
8 //Montar enrutador en app principal
9 app.use(express.json());
10 app.use(router);
11 app.set("port", 3000);
12 //Test a base de datos
13 const testDb = async () => {
14   try {
15     await sequelize.sync();
16     console.log("Conexion realizada con éxito");
17     //Correr Servicio por puerto 3000
18     app.listen(app.get("port"), () => {
19       console.log("Servidor Escuchando por puerto $(app.get('port'))");
20     });
21   } catch (error) {
22     console.log("Error al realizar conexión $(error)");
23   }
24 };
25
26 testDb();
27
```

Se lanza el servicio backend con npm start



The screenshot shows the Visual Studio Code editor with a project named 'serverjs - Taller25WIG'. The Explorer sidebar on the left shows the file structure, including 'server.js'. The main editor displays the content of 'server.js', which is a Node.js application using Express.js and Sequelize. The code includes imports for express, router, sequelize, and a product model, followed by app configuration, database connection, and a test database function. The bottom panel shows the terminal with the command 'npm start' and the output of the application, which includes logs for database connection, server listening on port 3000, and test database operations.

```
server.js
1 import express from "express";
2 import router from "../Routes/routes.js";
3 import { sequelize } from "../Database/database.js";
4 import { Producto } from "../Models/productos.js";
5 //Crear Instancia
6 const app = express();
7
8 //Puntar servidor en app principal
9 app.use(express.json());
10 app.use(router);
11 app.set("port", 3000);
12 //test a base de datos
13 const testDb = async () => {
14   try {
15     await sequelize.sync();
16     console.log("Conexion realizada con éxito");
17     //Correr Servicio por puerto 3000
18     app.listen(app.get("port"), () => {
19       console.log(`Servidor Escuchando por puerto ${app.get("port")}`);
20     });
21   } catch (error) {
22     console.log("Error al realizar conexión $(error)");
23   }
24 };
25
26 testDb();
27
```

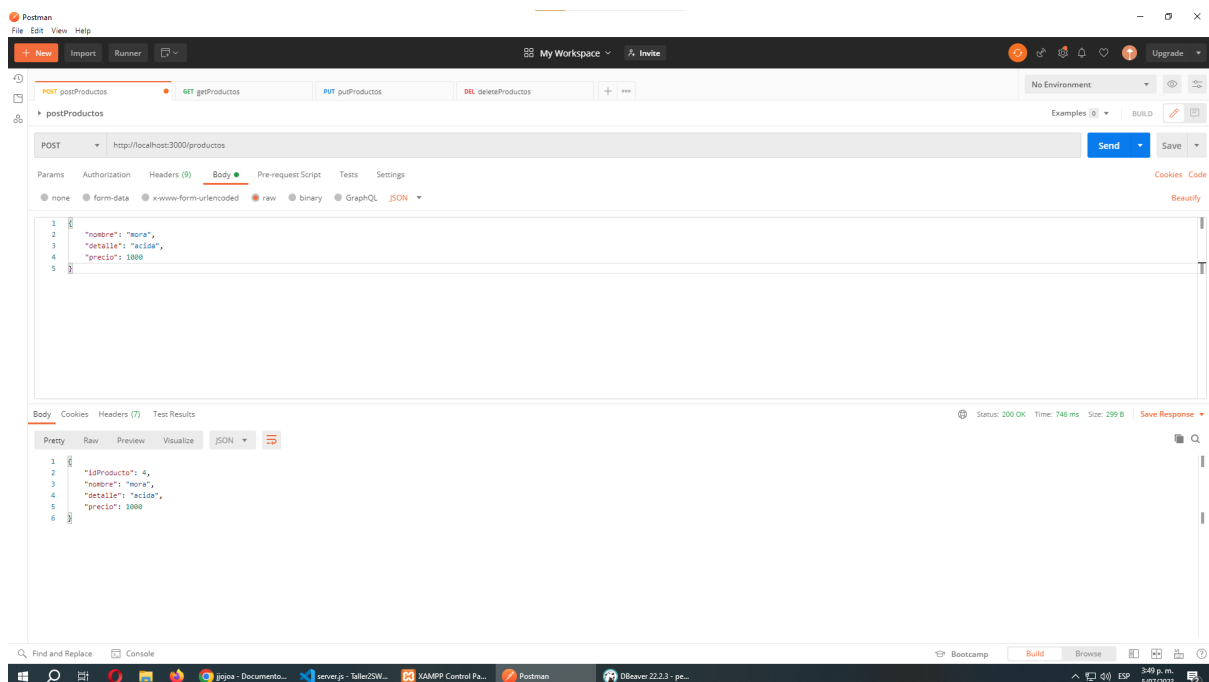
```
> backend@1.0.0 start
> nodemon server.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node server.js'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'productos' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'productos'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'clientes' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'clientes'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'pedidos' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'pedidos'
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'pedido_productos' AND TABLE_SCHEMA = 'fruver'
Executing (default): SHOW INDEX FROM 'pedido_productos'
Conexion realizada con éxito
Servidor Escuchando por puerto 3000
```

Usando la plataforma Postman se realizarán las solicitudes http para probar las rutas que ya definimos previamente para la gestión de los datos

TABLA PRODUCTOS

post



get

Postman interface showing a GET request to `http://localhost:3000/productos`. The response is a JSON array of two product objects.

Request:

- Method: GET
- URL: `http://localhost:3000/productos`

Response:

```
1 {
2   {
3     "idProducto": 3,
4     "nombre": "sandia",
5     "detalle": "Dulce",
6     "precio": 2000
7   },
8   {
9     "idProducto": 4,
10    "nombre": "naranja",
11    "detalle": "acida",
12    "precio": 1800
13  }
14 }
```

Status: 200 OK, Time: 68 ms, Size: 369 B

put

Postman interface showing a PUT request to `http://localhost:3000/productos/4`. The request body is a JSON object for a product with id 4. The response is the same JSON object.

Request:

- Method: PUT
- URL: `http://localhost:3000/productos/4`
- Body:

```
1 {
2   "nombre": "naranja",
3   "detalle": "extra dulce",
4   "precio": 1800
5 }
```

Response:

```
1 {
2   "idProducto": 4,
3   "nombre": "naranja",
4   "detalle": "extra dulce",
5   "precio": 1800
6 }
```

Status: 200 OK, Time: 69 ms, Size: 305 B

del

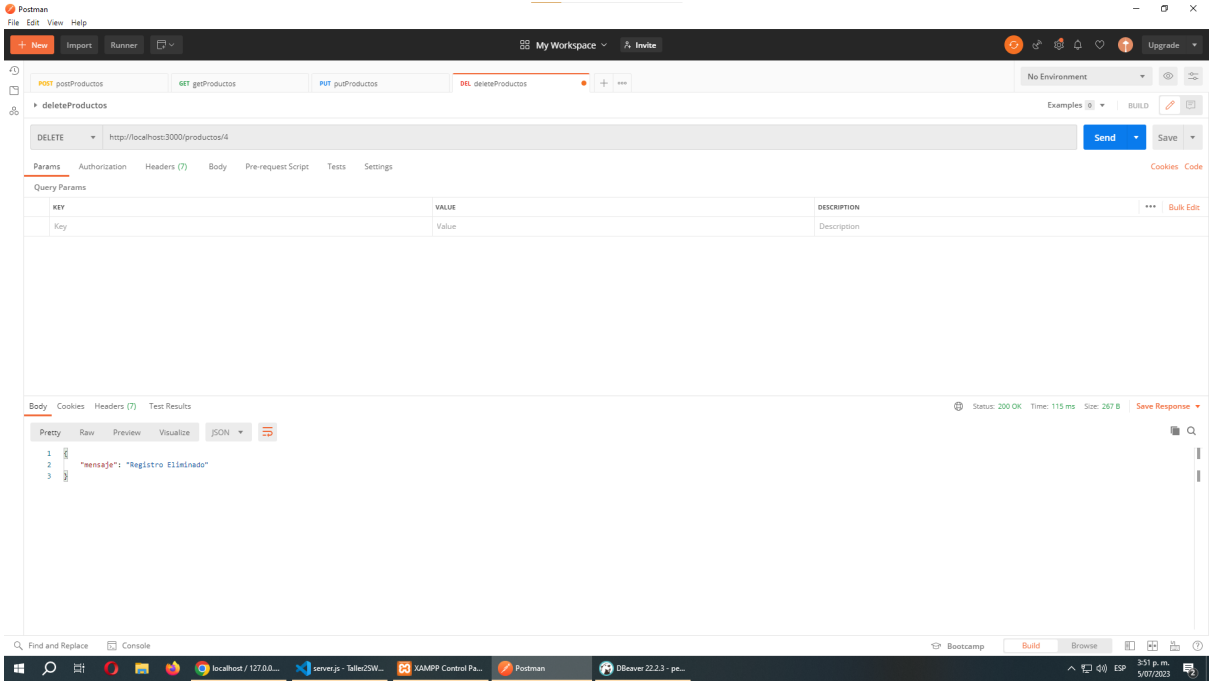
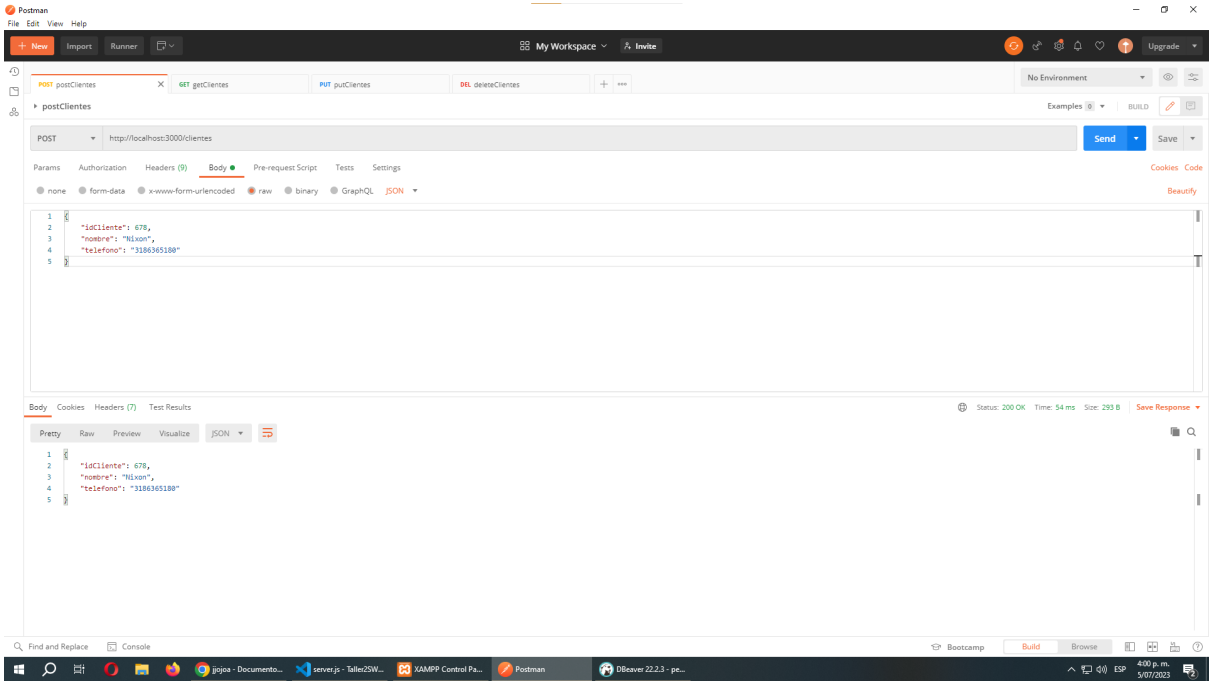


TABLA CLIENTES

post



get

Postman interface showing a GET request to `http://localhost:3000/clientes`. The response is a JSON array of 3 client objects.

```
GET http://localhost:3000/clientes
```

Params: Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 13 ms Size: 424 B Save Response

```
1 {
2   {
3     "idCliente": 123,
4     "nombre": "Juan",
5     "telefono": "3177851113"
6   },
7   {
8     "idCliente": 321,
9     "nombre": "Evelin Alexandra",
10    "telefono": "3122429531"
11  },
12  {
13    "idCliente": 678,
14    "nombre": "Nelson",
15    "telefono": "3186365180"
16  }
17 }
```

put

Postman interface showing a PUT request to `http://localhost:3000/clientes/678`. The request body is a JSON object for a client with id 678. The response is the same JSON object.

```
PUT http://localhost:3000/clientes/678
```

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body: none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "nombre": "Nelson Darío",
3   "telefono": "3122429525"
4 }
```

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 77 ms Size: 299 B Save Response

```
1 {
2   "idCliente": 678,
3   "nombre": "Nelson Darío",
4   "telefono": "3122429525"
5 }
```

del

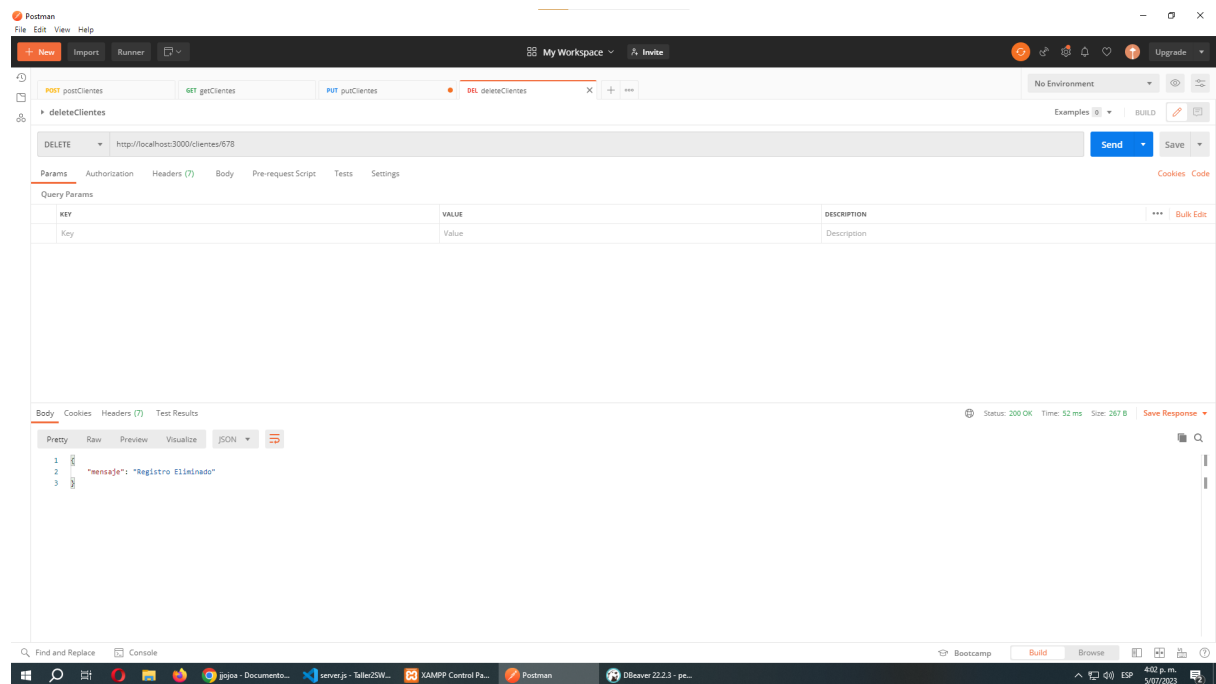
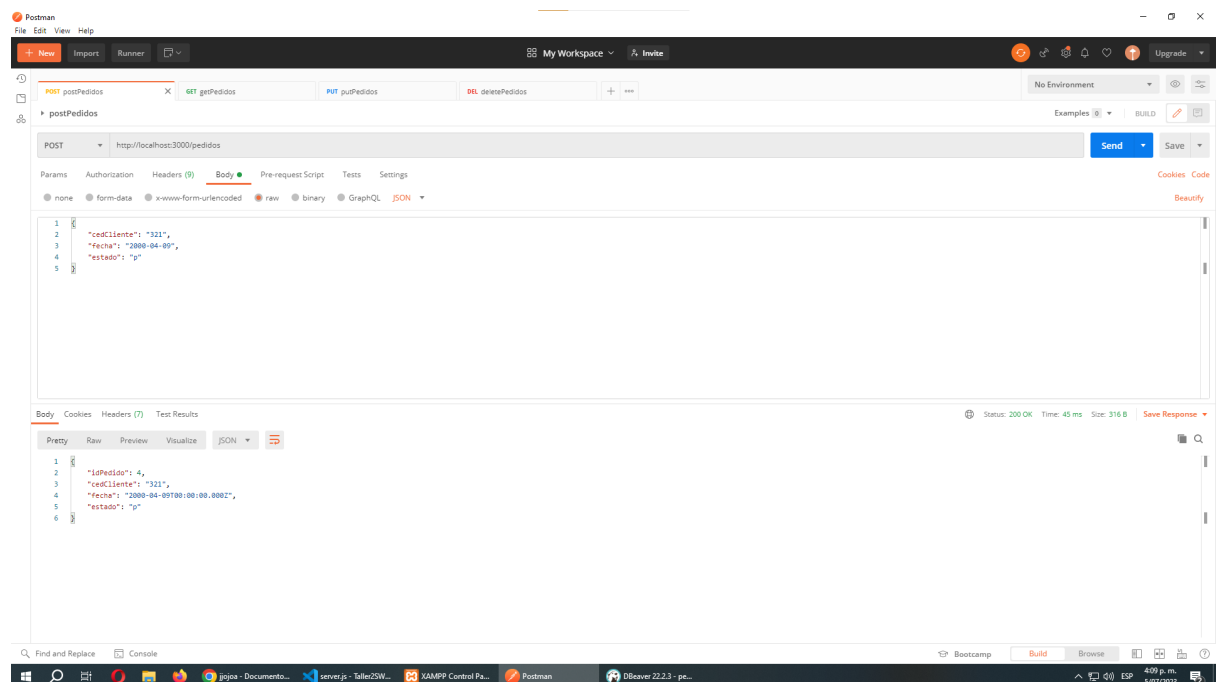


TABLA PEDIDOS

Para las pruebas de gestión de la tabla pedidos se necesita tener una cédula ya registrada por la referencia que se hace hacia la tabla clientes, de lo contrario ocurrirá un error.

post



get

Postman interface showing a GET request to `http://localhost:3000/pedidos`. The response is a JSON array of two objects, each representing a 'pedido'.

```
1 {
2   "idPedido": 2,
3   "cedCliente": 123,
4   "fecha": "2020-07-12",
5   "estado": "p"
6 },
7 {
8   "idPedido": 4,
9   "cedCliente": 321,
10  "fecha": "2008-04-09",
11  "estado": "p"
12 }
13 ]
14 }
```

put

Postman interface showing a PUT request to `http://localhost:3000/pedidos/4`. The request body is a JSON object. The response is a JSON object.

```
1 {
2   "fecha": "2008-04-09",
3   "estado": "f"
4 }
```

```
1 {
2   "idPedido": 4,
3   "cedCliente": 321,
4   "fecha": "2008-04-09T00:00:00Z",
5   "estado": "f"
6 }
```

del

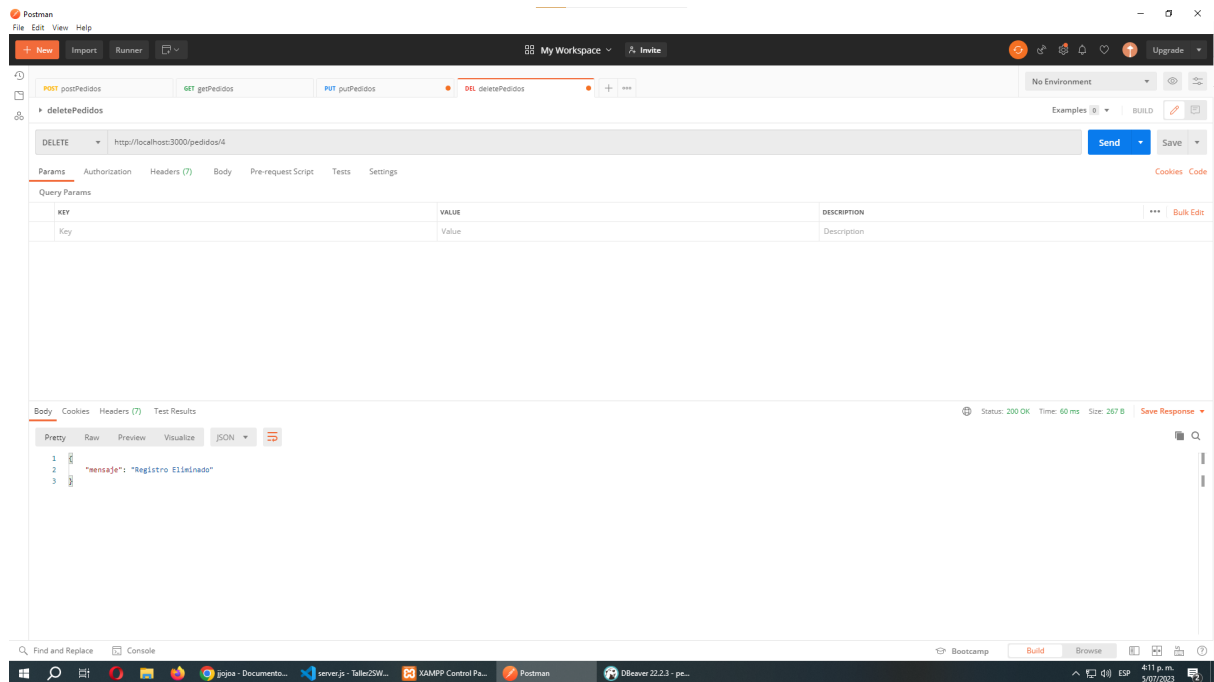
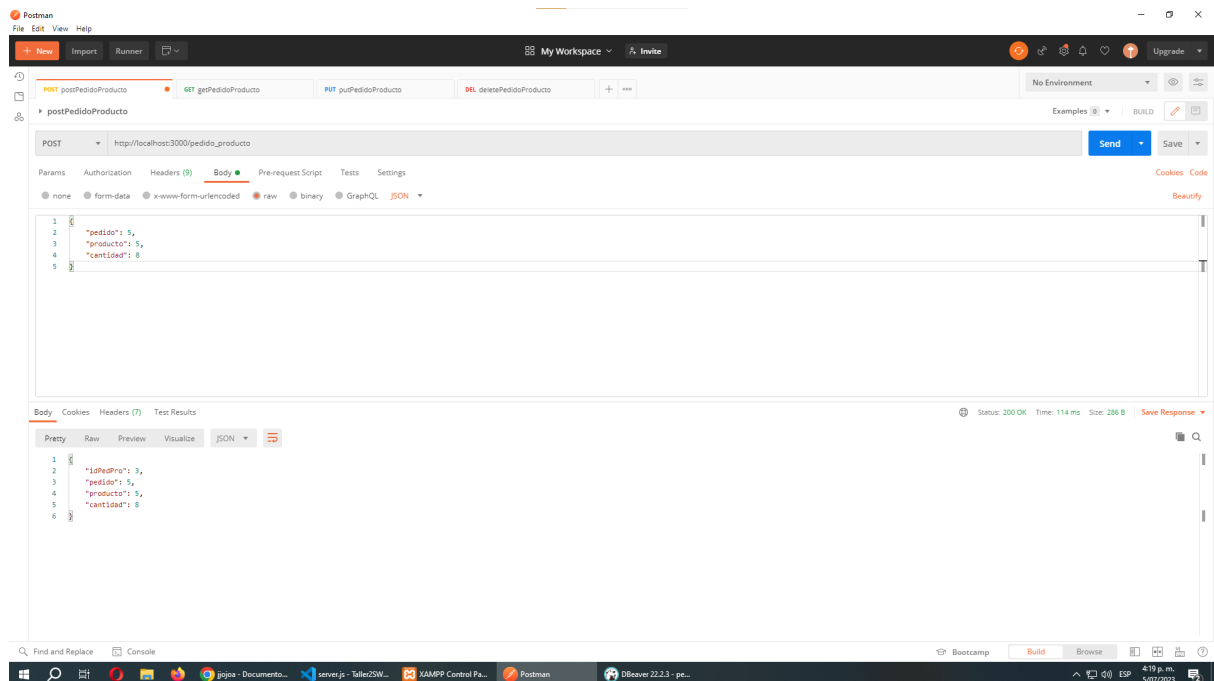


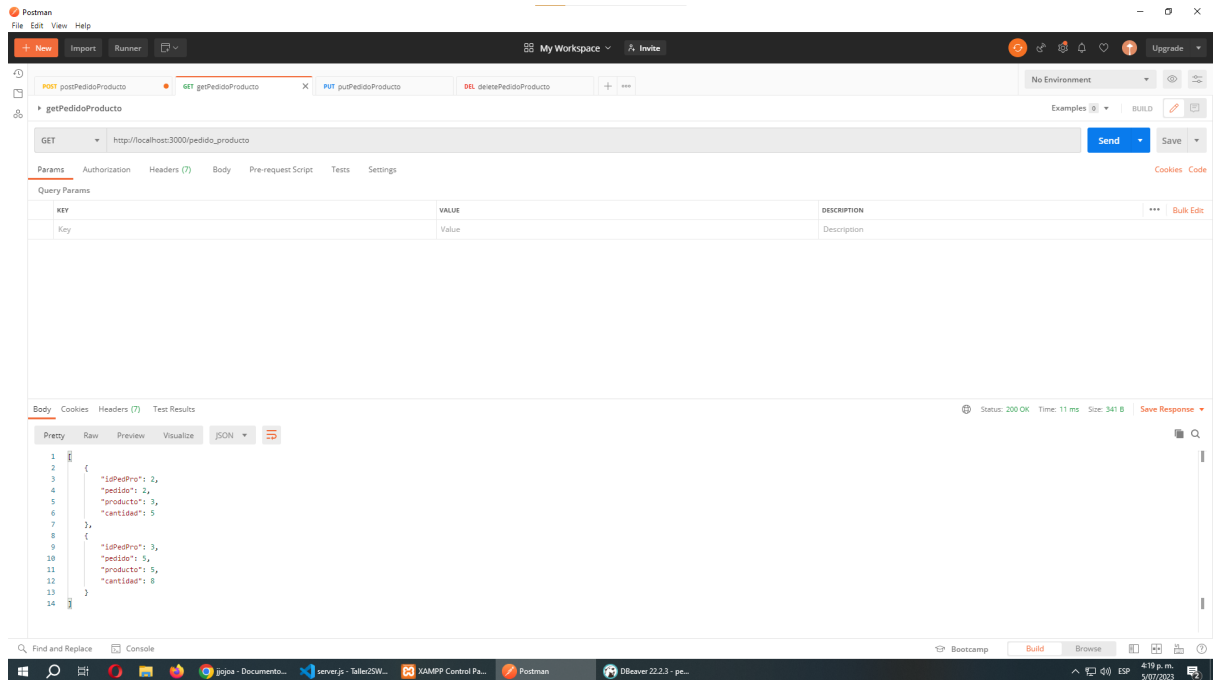
TABLA PEDIDO_PRODUCTO

Para las pruebas de gestión de la tabla `pedido_producto` se necesita tener el id de un pedido y el id de un producto ya registrados por la referencia que se hace hacia la tabla `pedido` y `productos`, de lo contrario ocurrirá un error.

post



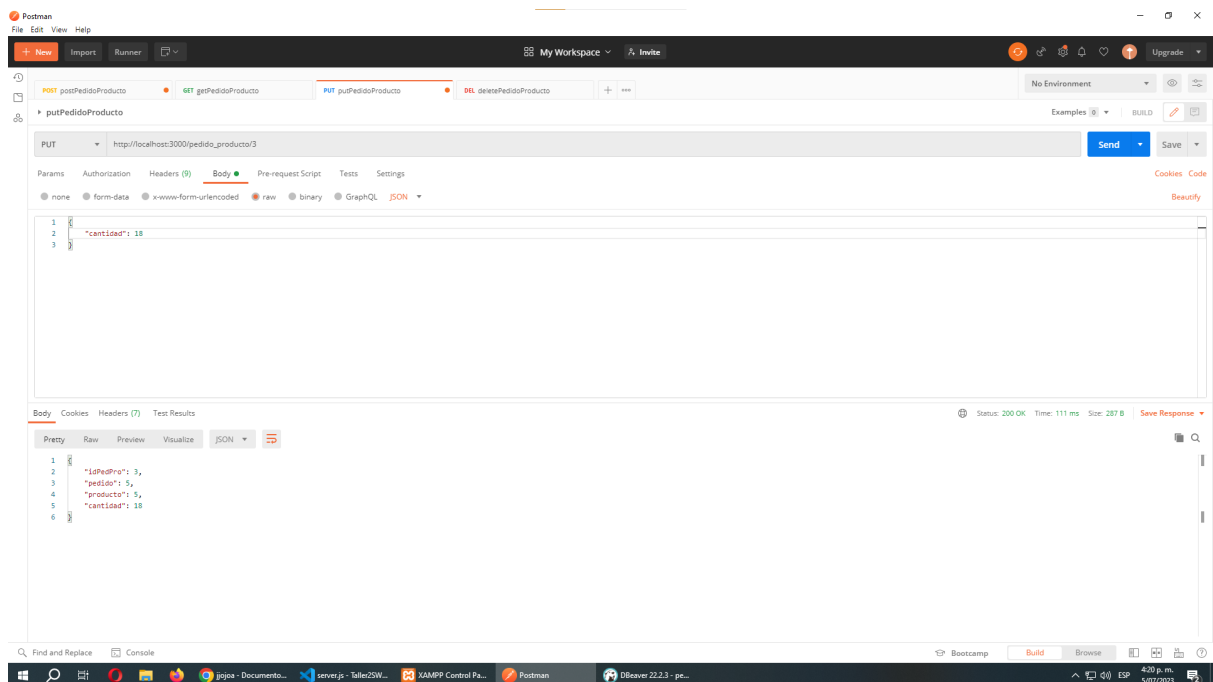
get



Postman interface showing a GET request to `http://localhost:3000/pedido_producto`. The response is a JSON array with 2 objects.

```
1 [
2   {
3     "idPedidoPro": 2,
4     "pedido": 2,
5     "producto": 2,
6     "cantidad": 5
7   },
8   {
9     "idPedidoPro": 3,
10    "pedido": 5,
11    "producto": 5,
12    "cantidad": 8
13  }
14 ]
```

put

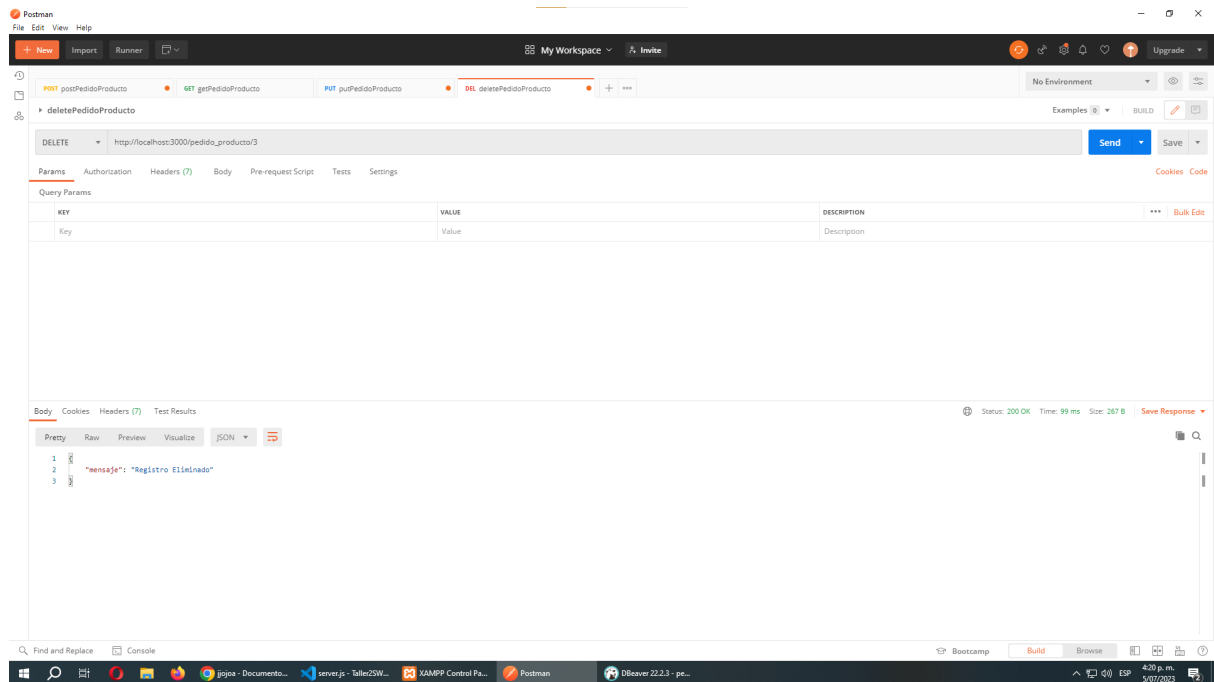


Postman interface showing a PUT request to `http://localhost:3000/pedido_producto/3`. The request body is a JSON object with `"cantidad": 18`. The response is a JSON object with `"idPedidoPro": 3, "pedido": 5, "producto": 5, "cantidad": 18`.

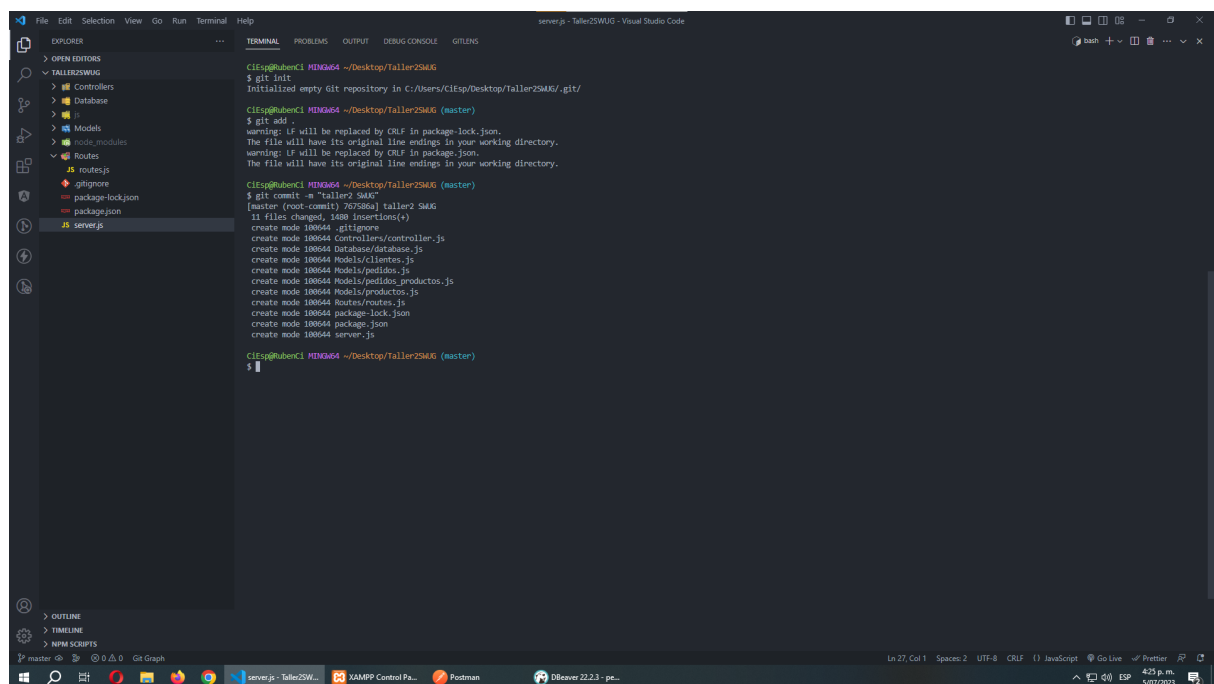
```
1 {
2   "cantidad": 18
3 }

1 {
2   "idPedidoPro": 3,
3   "pedido": 5,
4   "producto": 5,
5   "cantidad": 18
6 }
```

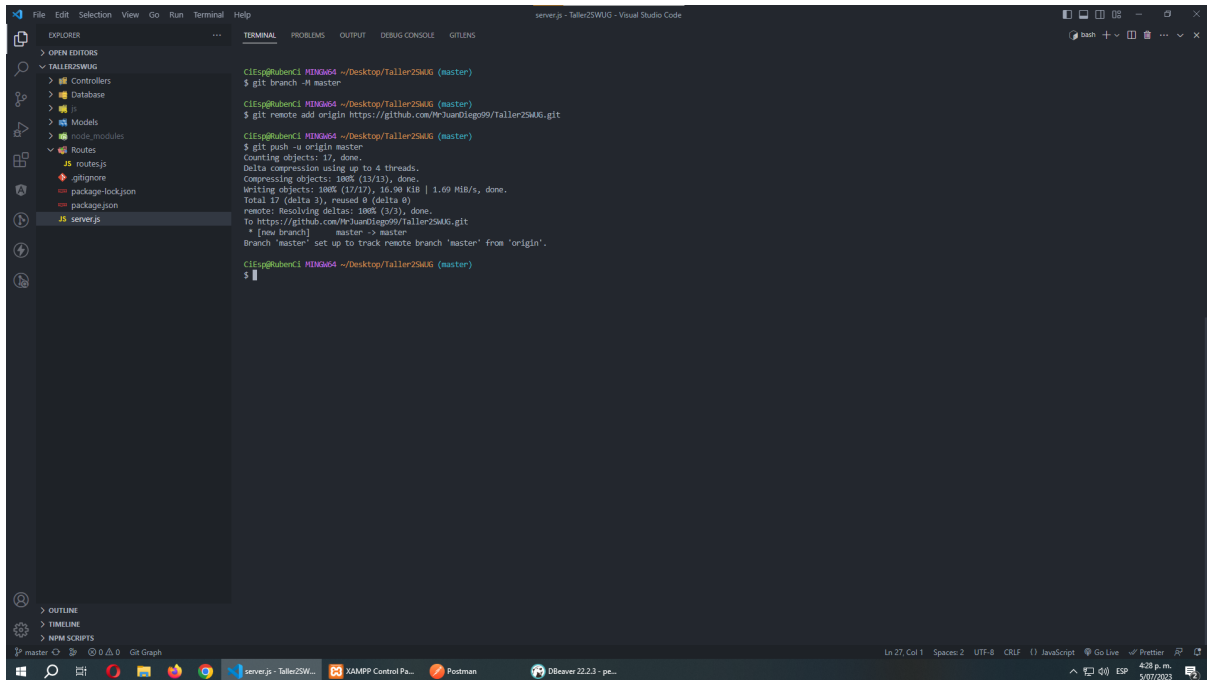
del



Inicializamos un repositorio local git para subir nuestro proyecto



Creamos un repositorio remoto en github y lo sincronizamos con nuestro proyecto



```
CLisp@huberC1 HDK664 ~/Desktop/Taller2SM4G (master)
$ git branch -M master

CLisp@huberC1 HDK664 ~/Desktop/Taller2SM4G (master)
$ git remote add origin https://github.com/PwJuanDiego09/Taller2SM4G.git

CLisp@huberC1 HDK664 ~/Desktop/Taller2SM4G (master)
$ git push -u origin master
Counting objects: 17, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (13/13), done.
Writing objects: 100% (17/17), 16.90 KiB | 1.69 MiB/s, done.
Total 17 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/PwJuanDiego09/Taller2SM4G.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

CLisp@huberC1 HDK664 ~/Desktop/Taller2SM4G (master)
$
```

The Explorer view on the left shows the project structure:

- OPEN EDITORS
- Taller2SM4G
 - Controllers
 - Database
 - lib
 - Models
 - node_modules
 - Routes
 - routes.js
 - utils
 - gitignore
 - package-lock.json
 - package.json
 - server.js

The status bar at the bottom indicates the current file is `server.js` in the `server.js - Taller2SM4G` workspace, with a file size of 27 lines and 1 column. The system clock shows 4:28 p.m. on 5/07/2023.