

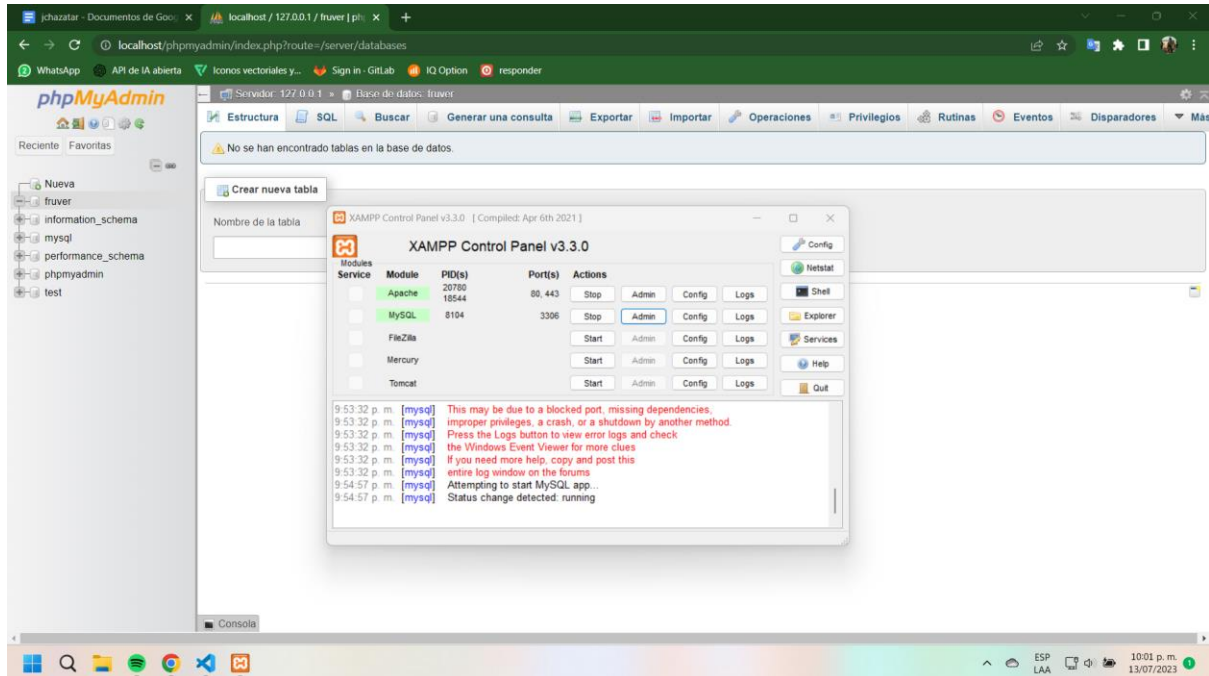
BACKEND FRUVER CHAZATAR

PRESENTADO POR  
JAIME FERNANDO CHAZATAR CARANGUAY

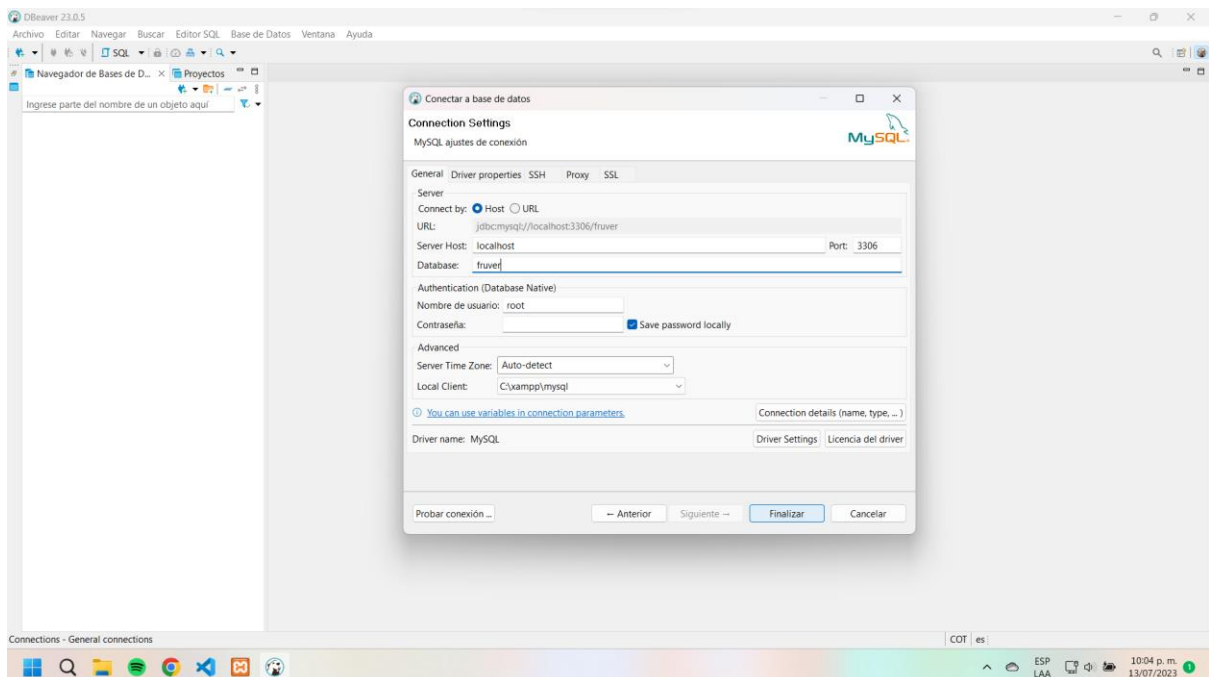
PRESENTADO A  
MG. VICENTE AUX REVELO

DIPLOMADO DE NUEVAS TECNOLOGÍAS PARA EL DESARROLLO DE SOFTWARE  
UNIVERSIDAD DE NARIÑO  
JULIO DE 2023

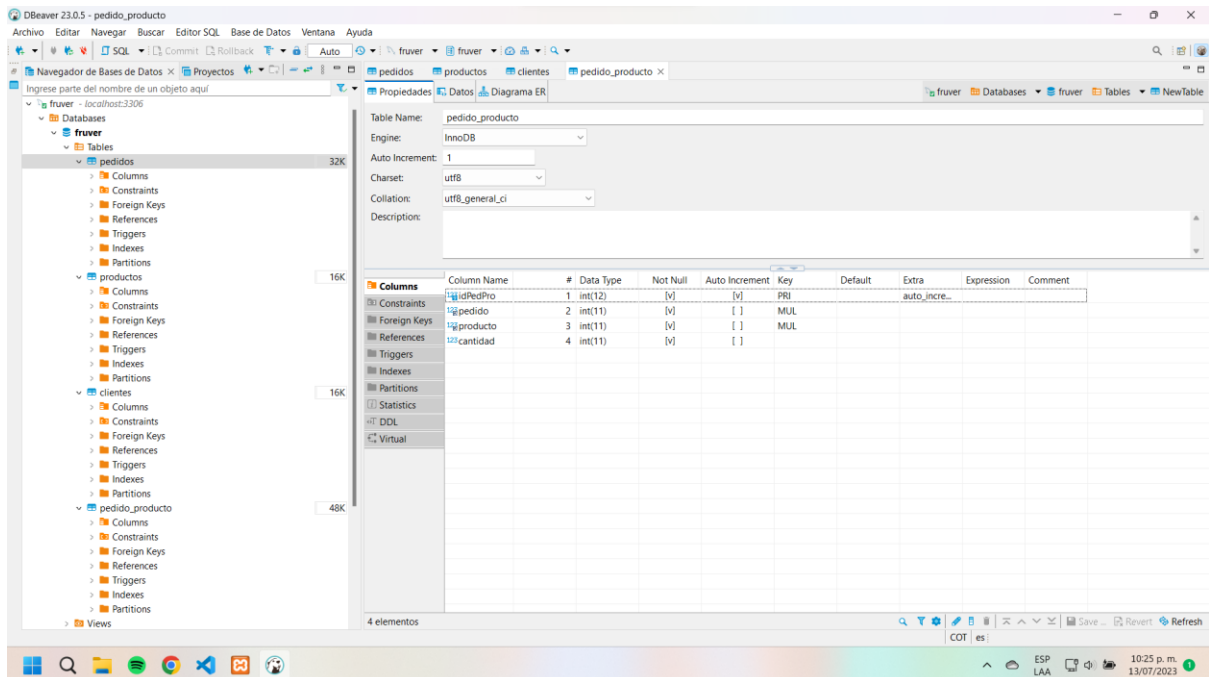
1. Para el desarrollo de esta actividad necesitaremos XAMPP y Dbeaver para la administración de la base de datos.
2. En Xampp Iniciamos el servicio **mysql** y presionamos en **admin** para crear la base de datos fruver



3. En **Dbeaver** realizamos la conexión con la base de datos **fruver** que es con la que vamos a trabajar nuestro proyecto.

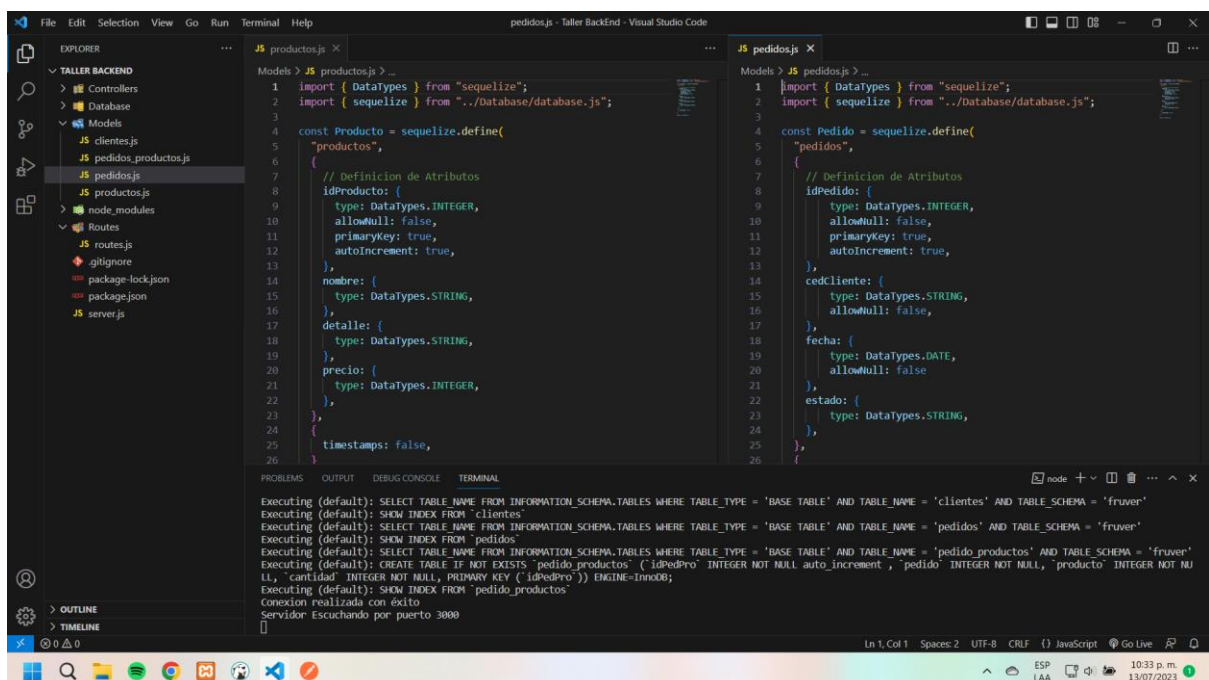


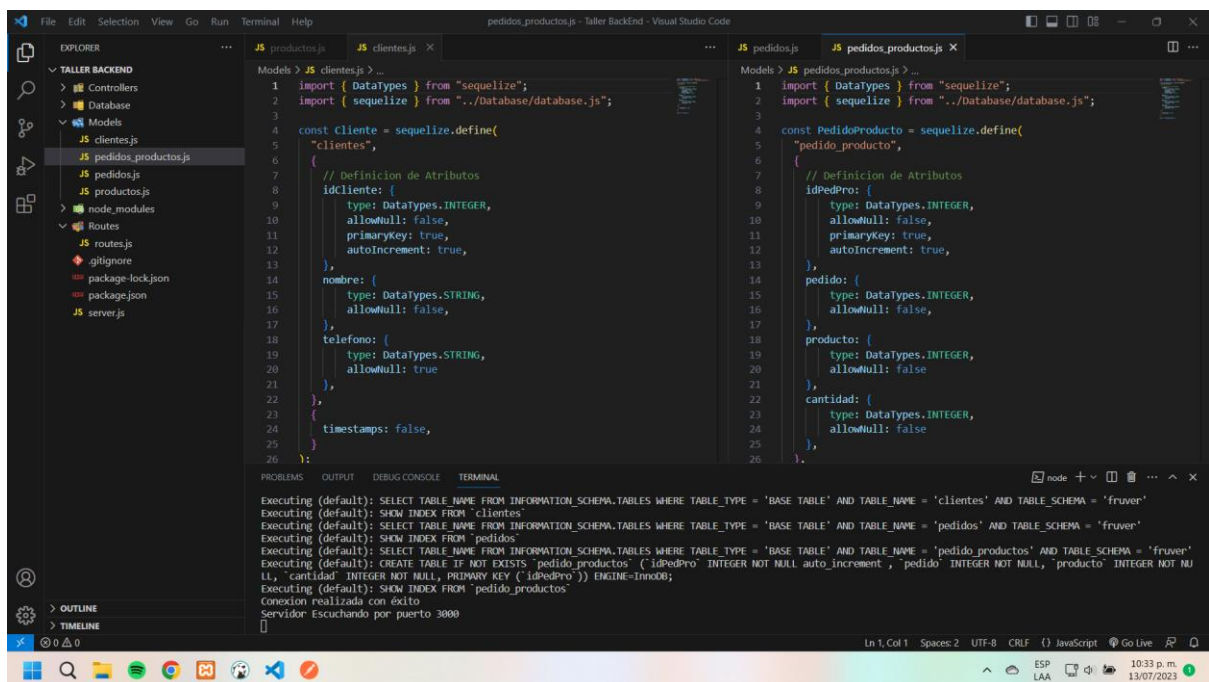
4. Creamos las respectivas tablas pedidos, productos, clientes y pedido\_producto en la base de datos y sus relaciones



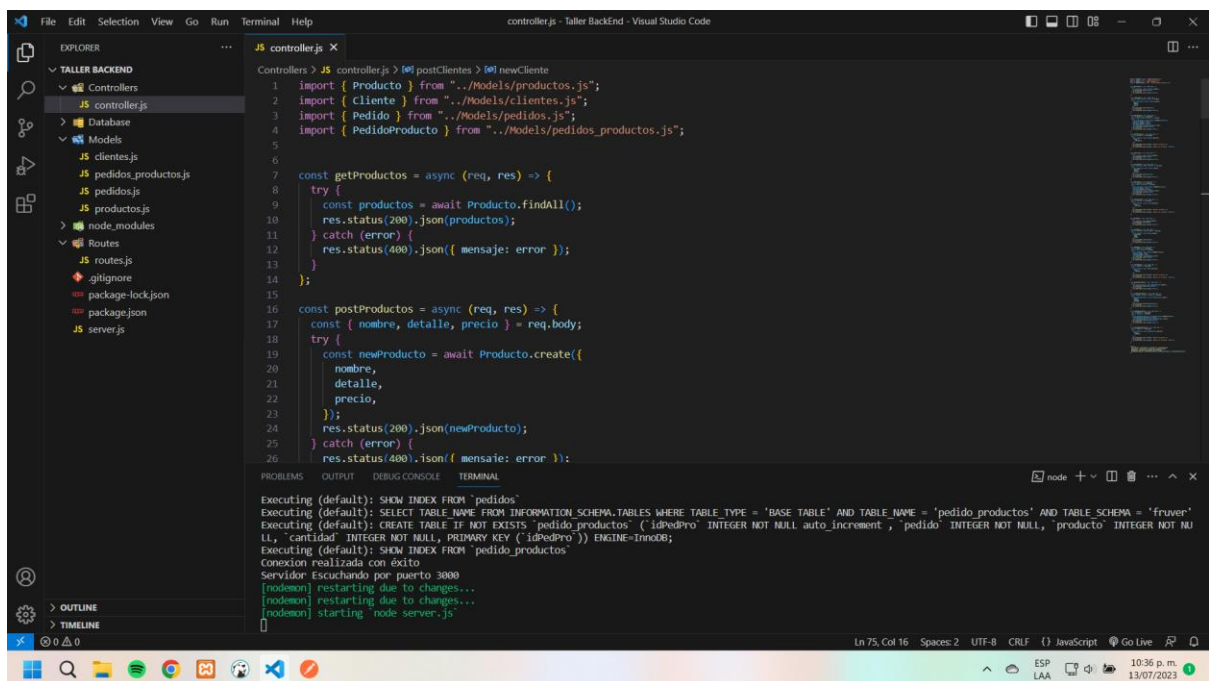
5. Procedemos ahora a desarrollar una aplicación backend implementada en node js, usando los códigos estructurales vistos en nuestro módulo.

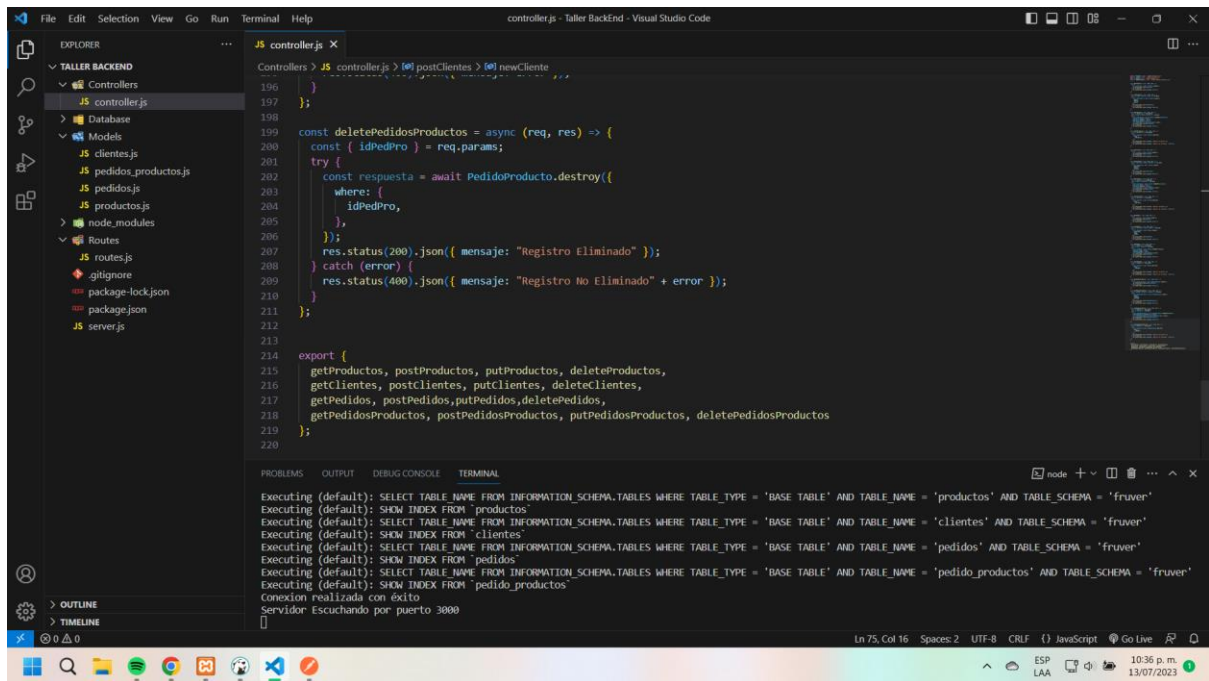
Se define cada tabla en un modelo usando sequelize, se crea un archivo js para cada tabla de la base de datos.





6. En el archivo `controller.js` se definen cada una de las funciones que utilizaremos en la base de datos para su gestion.





```
const deletePedidosProductos = async (req, res) => {
  const { idPedPro } = req.params;
  try {
    const respuesta = await PedidoProducto.destroy({
      where: {
        idPedPro,
      },
    });
    res.status(200).json({ mensaje: "Registro Eliminado" });
  } catch (error) {
    res.status(400).json({ mensaje: "Registro No Eliminado" + error });
  }
};

export {
  getProductos, postProductos, putProductos, deleteProductos,
  getClientes, postClientes, putClientes, deleteClientes,
  getPedidos, postPedidos, putPedidos, deletePedidos,
  getPedidosProductos, postPedidosProductos, putPedidosProductos, deletePedidosProductos
};
```

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'

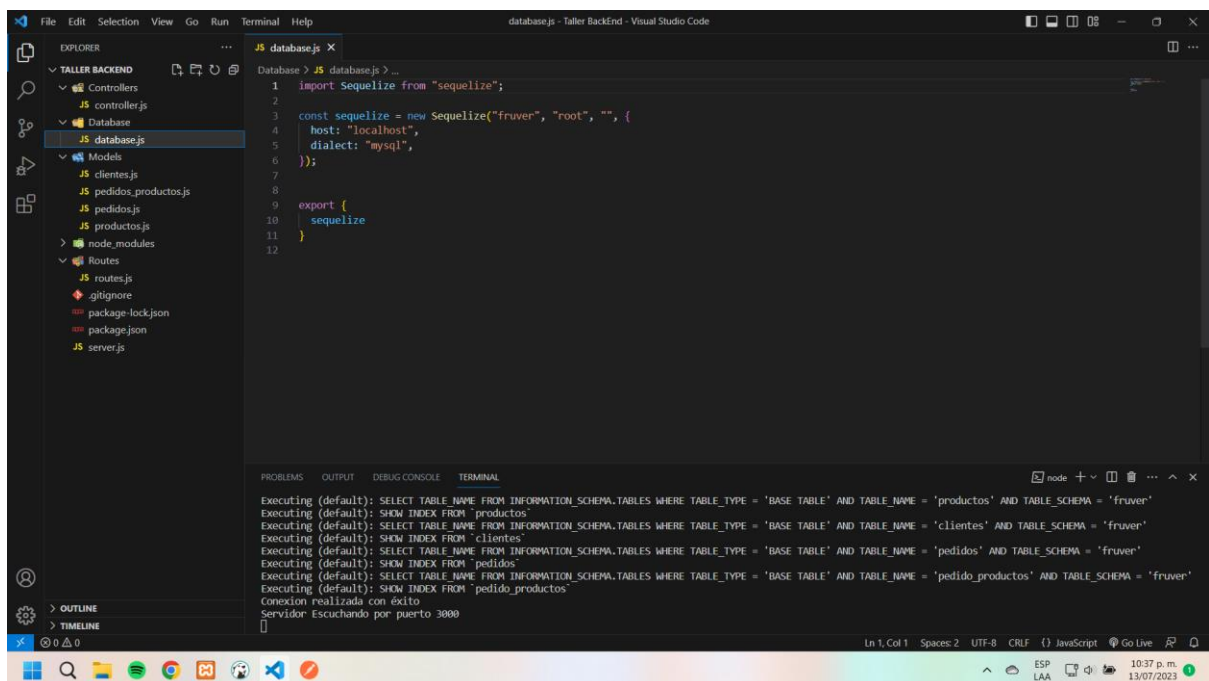
Executing (default): SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE = 'BASE TABLE' AND TABLE\_NAME = 'pedido\_productos' AND TABLE\_SCHEMA = 'fruver'

Executing (default): SHOW INDEX FROM 'pedido\_productos'

Conexion realizada con éxito

Servidor Escuchando por puerto 3000

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



```
import Sequelize from "sequelize";

const sequelize = new Sequelize("fruver", "root", "", {
  host: "localhost",
  dialect: "mysql",
});

export {
  sequelize
};
```

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'

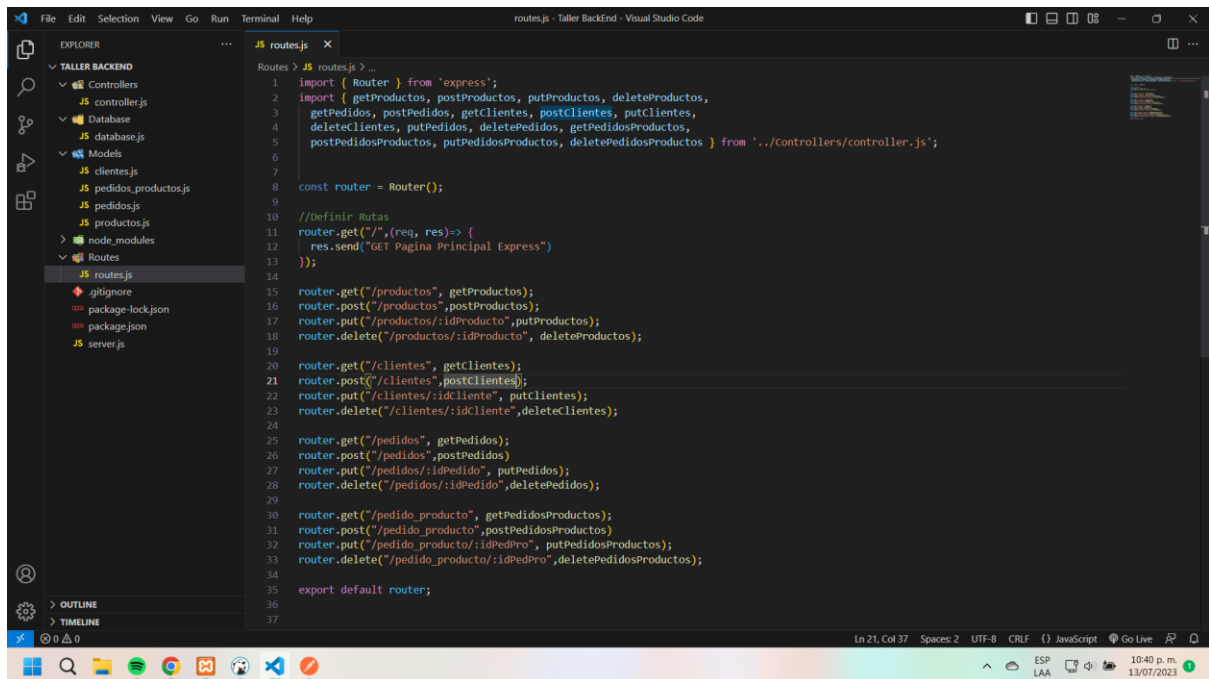
Executing (default): SHOW INDEX FROM 'pedido\_productos'

Conexion realizada con éxito

Servidor Escuchando por puerto 3000

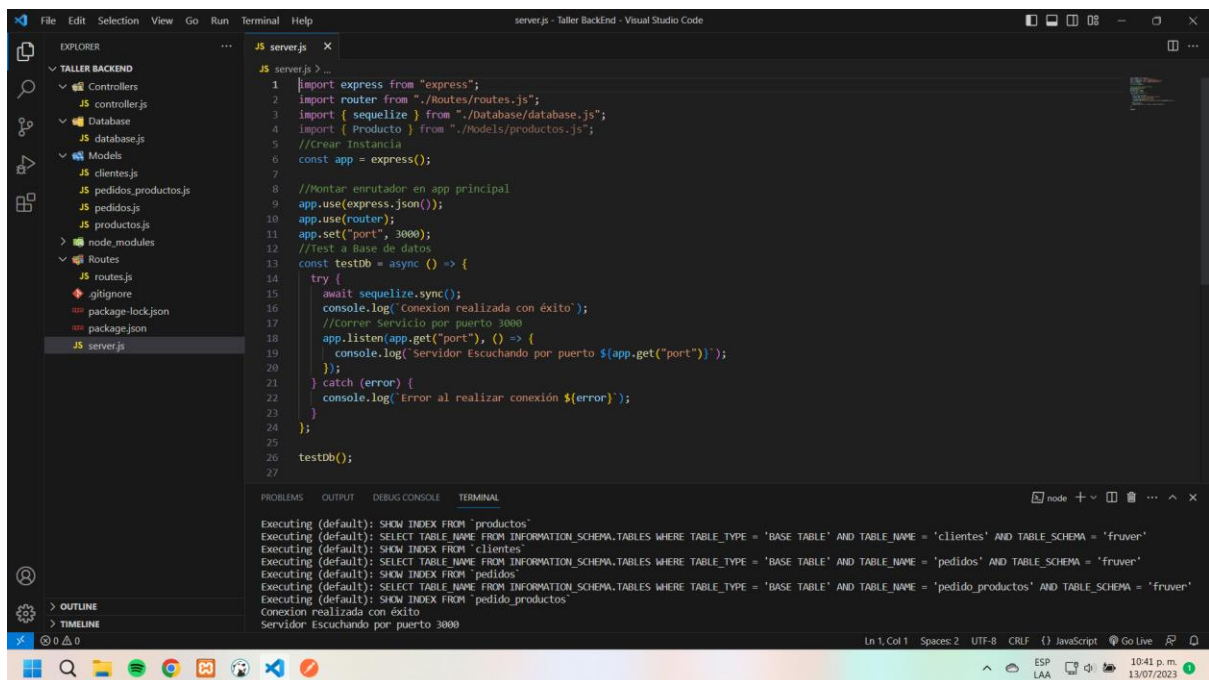


8. Una vez creados los anteriores archivos se procede a la creación del archivo routes.js donde se definen las rutas con su respectiva función importada desde el controlador.



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

9. 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();
```

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"

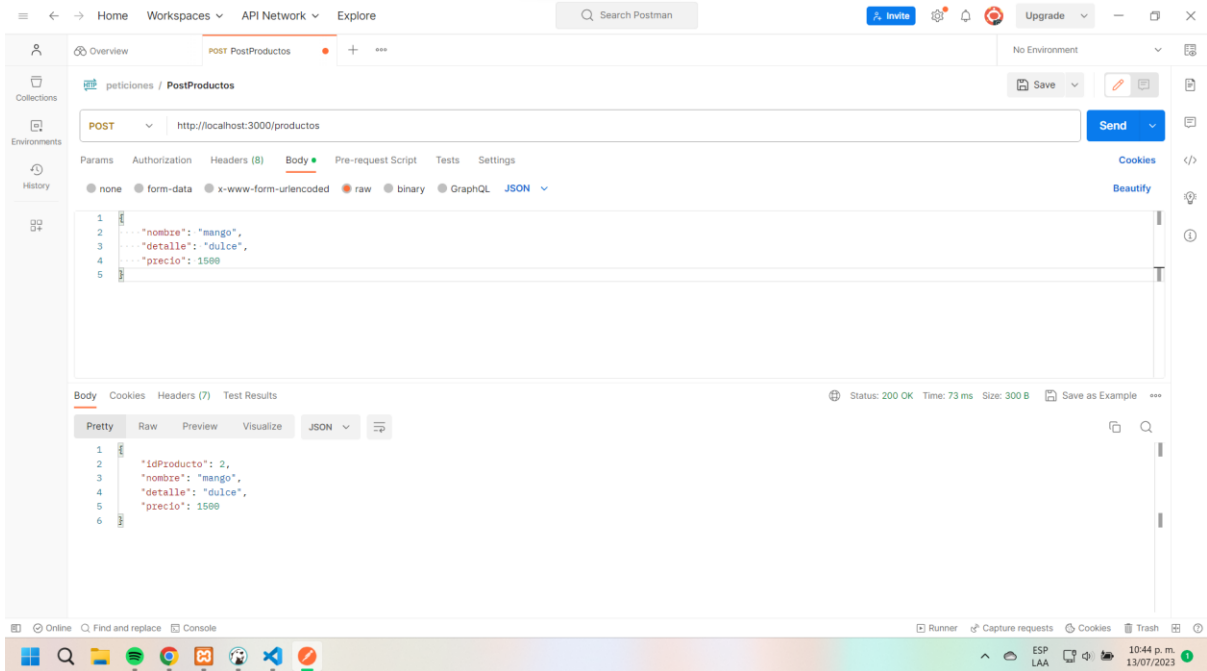
Conexion realizada con éxito

Servidor Escuchando por puerto 3000

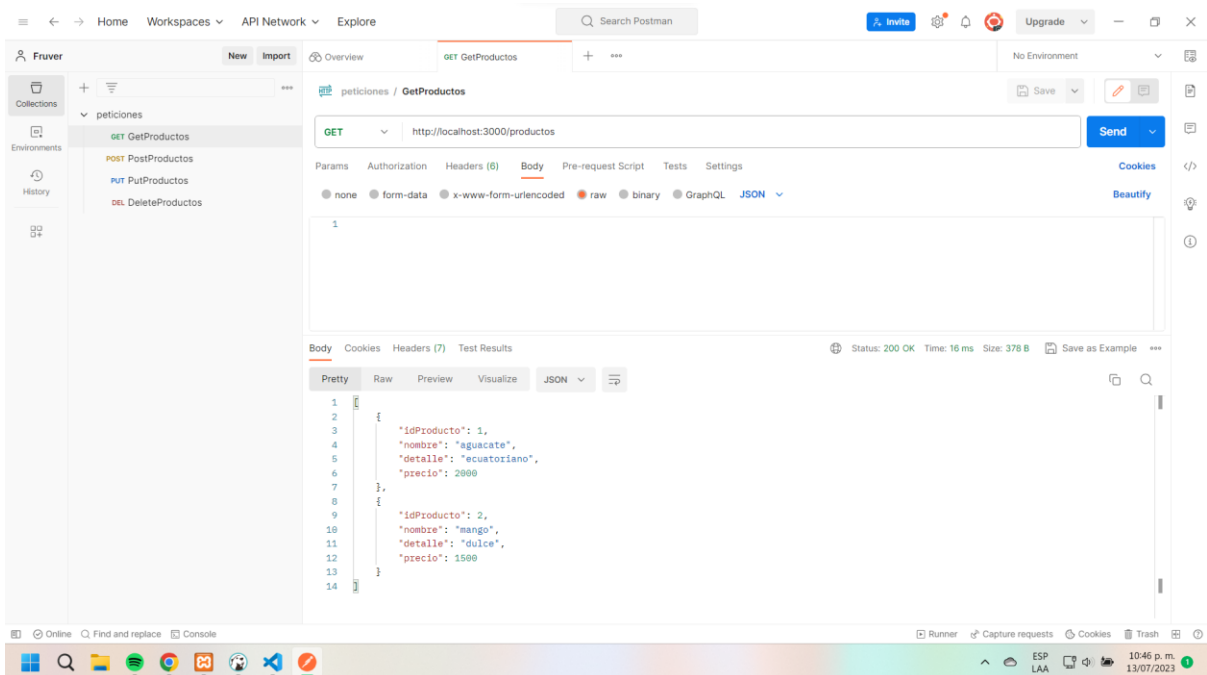
10. Usando la plataforma Postman se realizarán las solicitudes http para probar las rutas que ya definimos previamente para la gestión de los datos

## TABLA PRODUCTOS

post



get



# put

The screenshot shows the Postman interface with a PUT request configured. The request is named "PUT PutProductos" and is located in a collection named "peticiones". The URL is "http://localhost:3000/productos/1". The request body is a JSON object with the following structure:

```
1 {
2   "nombre": "aguacate",
3   "detalle": "Colombiano",
4   "precio": 2000
5 }
```

The response is displayed in the "Body" tab, showing a JSON object with the following structure:

```
1 {
2   "idProducto": 1,
3   "nombre": "aguacate",
4   "detalle": "Colombiano",
5   "precio": 2000
6 }
```

The status of the response is "200 OK", the time is "160 ms", and the size is "308 B".

# delete

The screenshot shows the Postman interface with a DELETE request configured. The request is named "DEL DeleteProductos" and is located in a collection named "peticiones". The URL is "http://localhost:3000/productos/2". The request body is a JSON object with the following structure:

```
1 {
2   "mensaje": "Registro Eliminado"
3 }
```

The response is displayed in the "Body" tab, showing a JSON object with the following structure:

```
1 {
2   "mensaje": "Registro Eliminado"
3 }
```

The status of the response is "200 OK", the time is "95 ms", and the size is "267 B".



# TABLA CLIENTES

## post

Postman interface showing a POST request to `http://localhost:3000/clientes`. The request body is a JSON object with the following data:

```
{  "idCliente": 1234,  "nombre": "Juan Diego",  "telefono": "317781113"}
```

The response status is 200 OK. The response body is a JSON object with the following data:

```
{  "idCliente": 1234,  "nombre": "Juan Diego",  "telefono": "317781113"}
```

## get

Postman interface showing a GET request to `http://localhost:3000/clientes`. The response status is 200 OK. The response body is a JSON array containing two objects with client information:

```
[  {    "idCliente": 1234,    "nombre": "Juan Diego",    "telefono": "317781113"  },  {    "idCliente": 4321,    "nombre": "Jaime",    "telefono": "3186365188"  }]
```

## put

Postman interface showing a PUT request to `http://localhost:3000/clientes/4321`. The request body is a JSON object with the following fields:

```
1 {
2   "nombre": "Evelin Guerrero",
3   "telefono": "3186365188"
4 }
```

The response is a JSON object with the following fields:

```
1 {
2   "idCliente": 4321,
3   "nombre": "Evelin Guerrero",
4   "telefono": "3186365188"
5 }
```

## delete

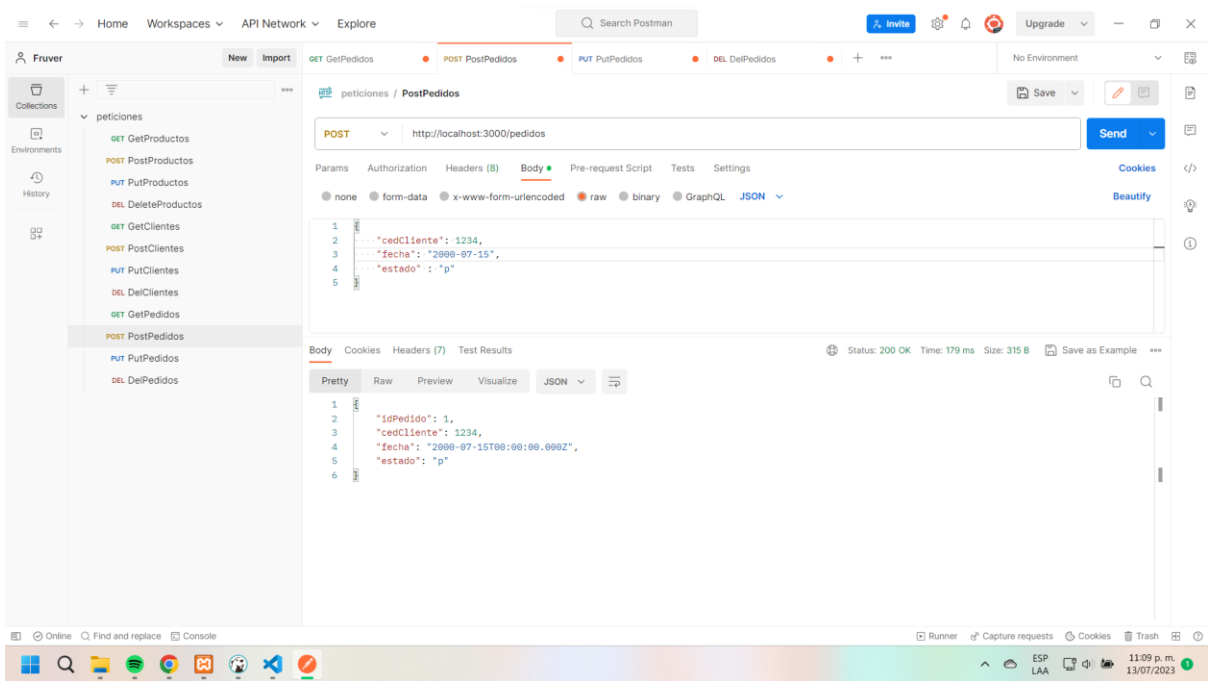
Postman interface showing a DELETE request to `http://localhost:3000/clientes/4321`. The response is a JSON object with the following field:

```
1 {
2   "mensaje": "Registro Eliminado"
3 }
```

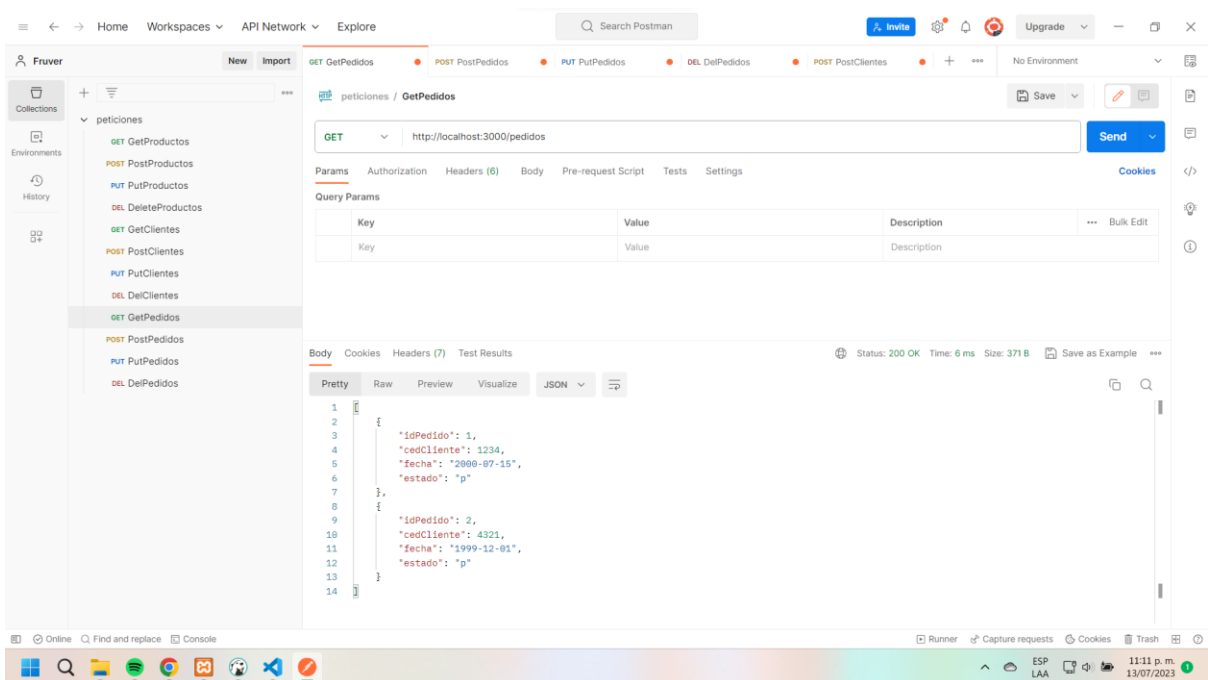
## TABLA PEDIDOS

*Nota: 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



## put

peticiones / PutPedidos

PUT http://localhost:3000/pedidos/1

Params Authorization Headers (8) Body Pre-request Script Tests Settings

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

```
1 {
2   "fecha": "1999-12-01",
3   "cedCliente": "A"
4 }
```

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 142 ms Size: 315 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "IdPedido": 1,
3   "cedCliente": 1234,
4   "fecha": "1999-12-01T00:00:00.000Z",
5   "estado": "A"
6 }
```

## delete

peticiones / DelPedidos

DELETE http://localhost:3000/pedidos/2

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 66 ms Size: 267 B Save as Example

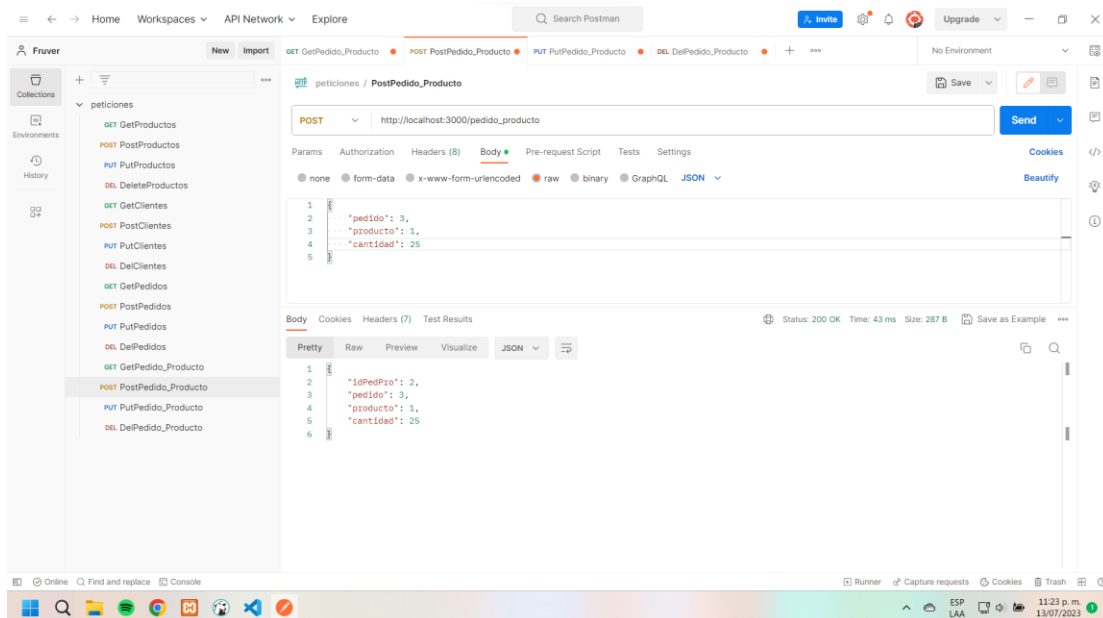
Pretty Raw Preview Visualize JSON

```
1 {
2   "mensaje": "Registro Eliminado"
3 }
```

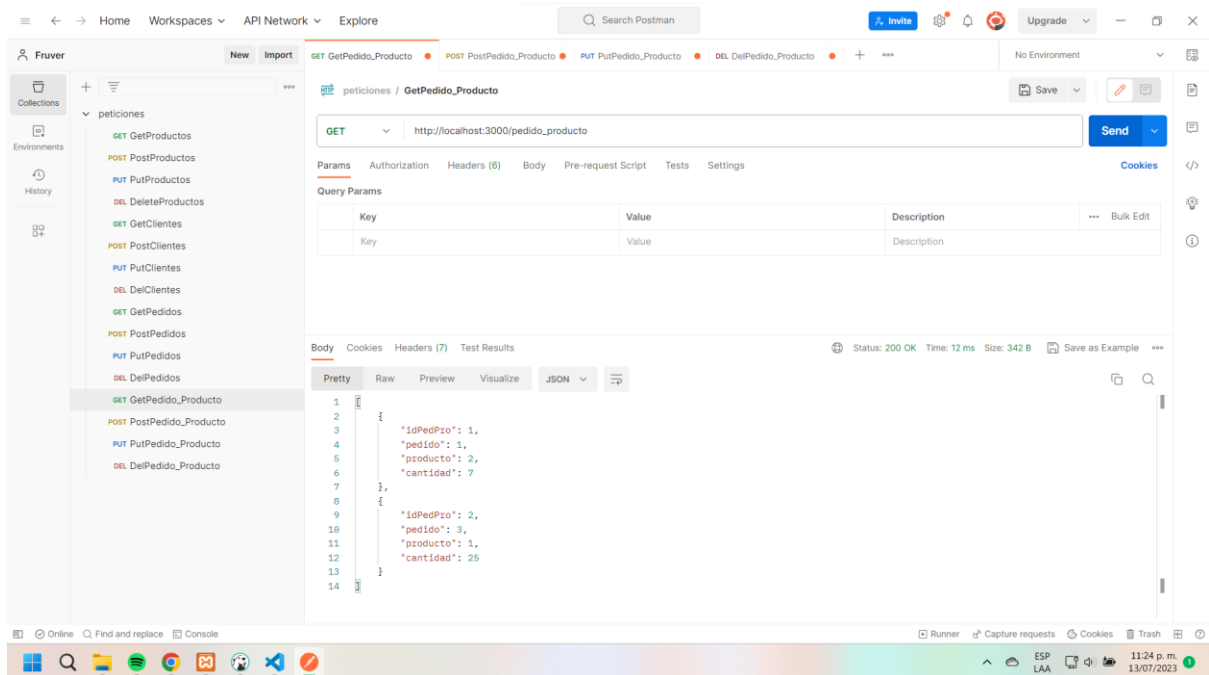
## TABLA PEDIDO\_PRODUCTO

*Nota: 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



## put

The screenshot shows the Postman application interface. On the left, a sidebar lists collections under 'peticiones', with 'PutPedido\_Producto' selected. The main panel displays a PUT request to 'http://localhost:3000/pedido\_producto/2'. The 'Body' tab is active, showing a JSON payload: 

```
{  "cantidad": 5}
```

. The response status is '200 OK' with a time of '160 ms' and size of '286 B'. The response body is displayed in the 'Body' tab as: 

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

## delete

The screenshot shows the Postman application interface. On the left, a sidebar lists collections under 'peticiones', with 'DelPedido\_Producto' selected. The main panel displays a DELETE request to 'http://localhost:3000/pedido\_producto/2'. The response status is '200 OK' with a time of '151 ms' and size of '267 B'. The response body is displayed in the 'Body' tab as: 

```
{  "mensaje": "Registro Eliminado"}
```