

## Taller Api\_Parcial2

Andrés Felipe Rojas Santamaría

parcial bases de datos

Universidad uniminuto

Ing. William Alexander Matallana Porras

Sede zipaquirá

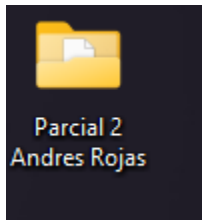
Sexto semestre

24 de abril de 2025

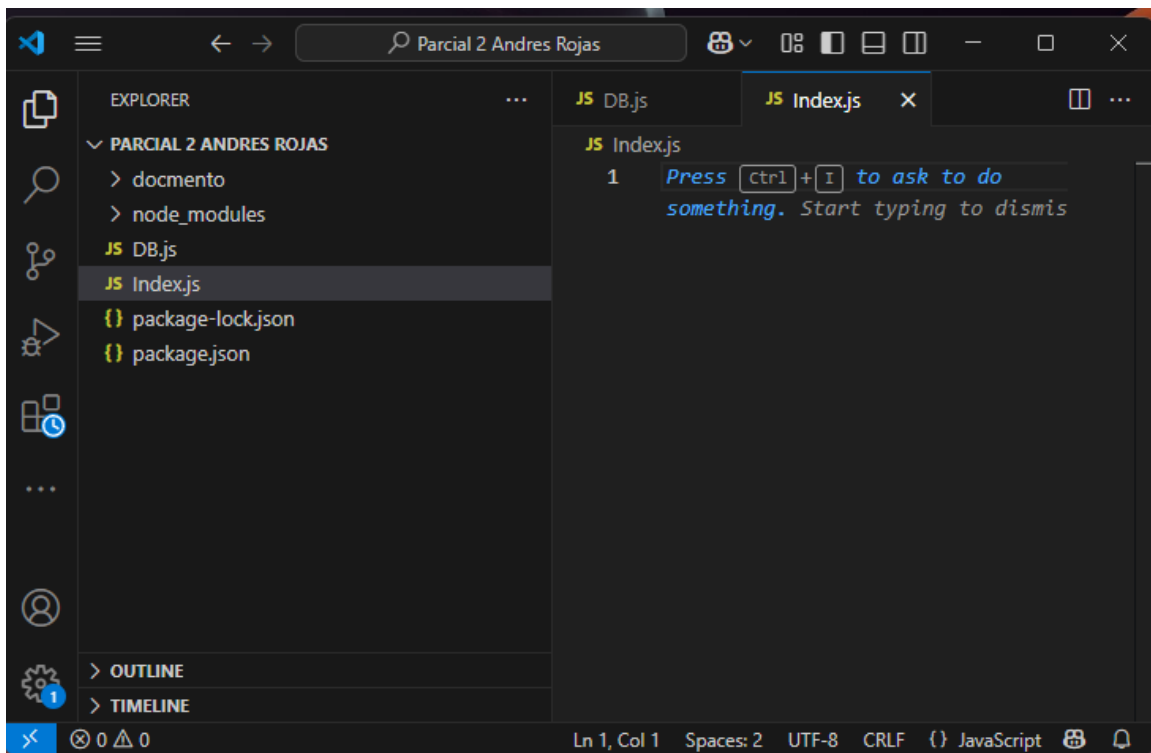
## Introducción

En este trabajo se muestra el paso a paso para crear una API que permite manejar la información de un sistema con datos como restaurantes, empleados, productos y pedidos. Para hacerlo, se usaron herramientas como Supabase para crear la base de datos, pgAdmin4 para conectarse a ella, Visual Studio Code para programar, y Postman para probar que todo funcionara bien. A lo largo del proceso se organizaron los archivos del proyecto y se programaron funciones que permiten ver, agregar, cambiar o eliminar información de cada tabla. Todo esto con el fin de poner en práctica lo aprendido en clase y entender mejor cómo funciona el desarrollo de aplicaciones.

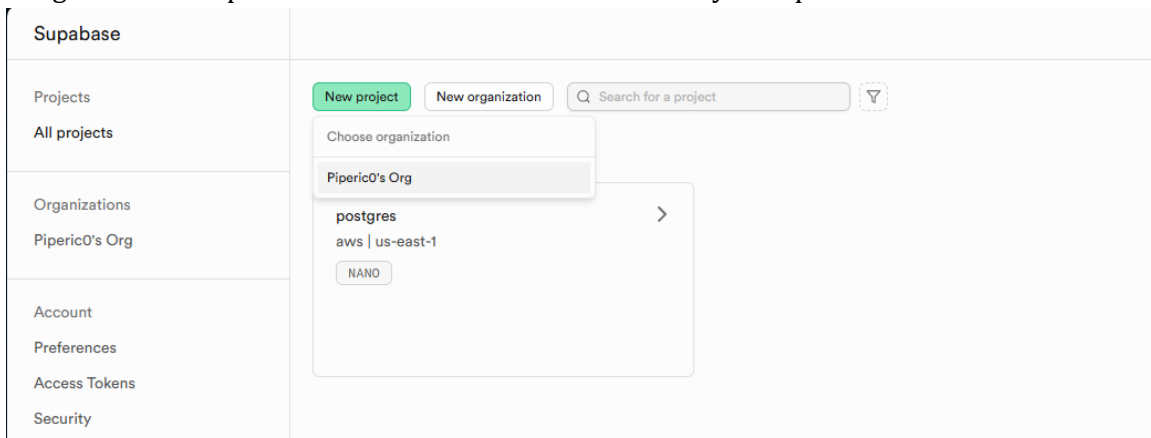
Comenzamos creando la carpeta donde se va a guardar la información de visual



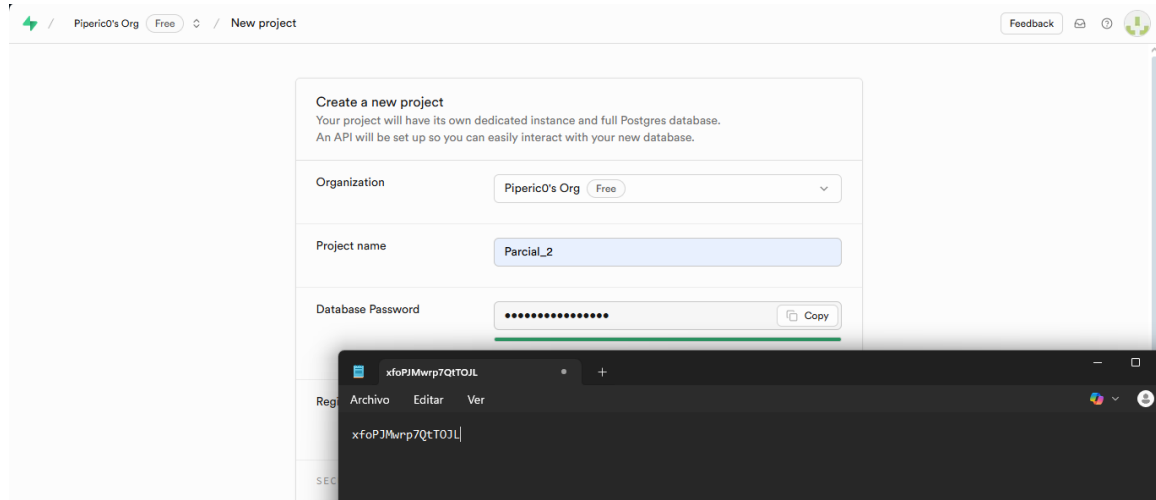
Abrimos la carpeta en visual y creamos el DB.js para la conexión de la base de datos y el Index.js para el tema de Api en mi caso estoy utilizando un proyecto anterior entonces modifiko todo encima de ese proyecto



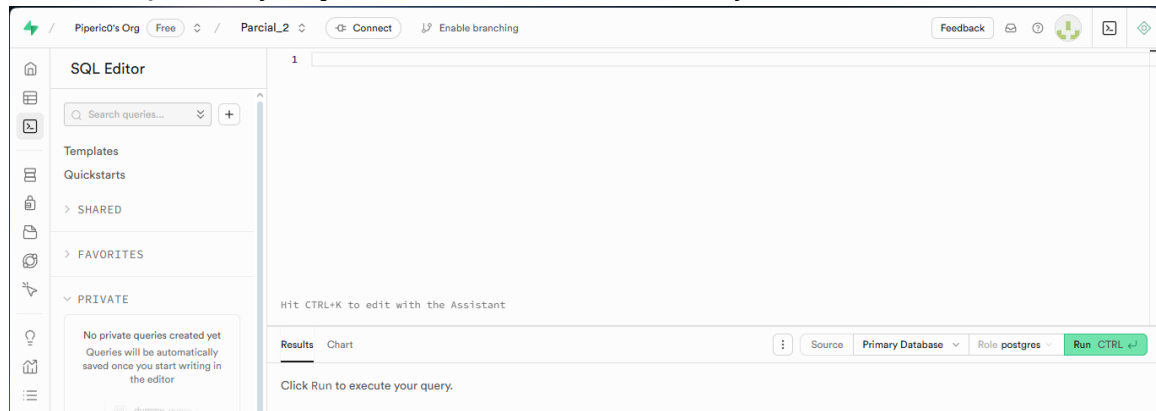
Luego vamos a supabase donde crearemos un nuevo Proyecto que va a ser la base de datos



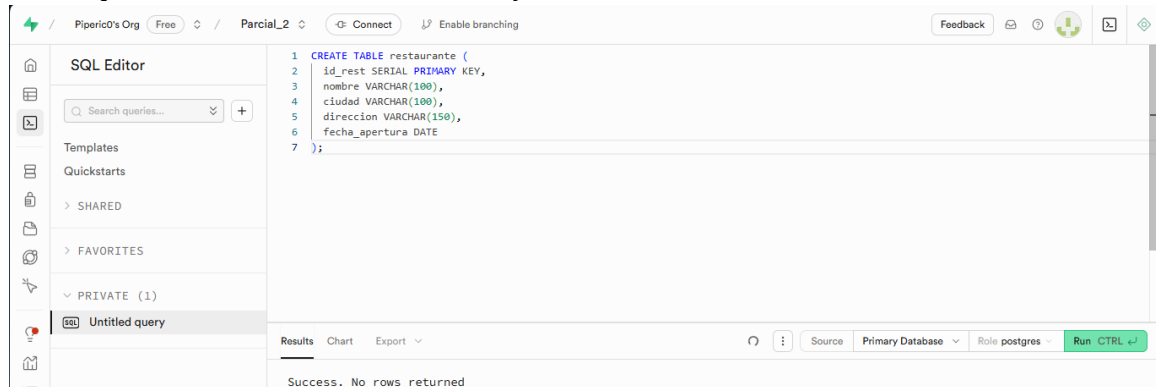
Colocamos la contraseña y en mi caso la guardo en block de notas para no perderla



Vamos a SQL editor y empezamos a crear la base de datos y las tablas



Acá empiezo con la tabla de restaurante y así sucesivamente las demás tablas



```

1 CREATE TABLE empleado (
2   id_empleado SERIAL PRIMARY KEY,
3   nombre VARCHAR(100),
4   rol VARCHAR(50),
5   id_rest INT REFERENCES restaurante(id_rest)
6 );

```

Pipero's Org Free / Parcial\_2 Connect Enable branching Feedback

**SQL Editor**

Search queries... +

Templates

Quickstarts

> SHARED

> FAVORITES

> PRIVATE (1)

Restaurant Table

```

1 CREATE TABLE producto (
2   id_prod SERIAL PRIMARY KEY,
3   nombre VARCHAR(100),
4   precio NUMERIC(10, 2)
5 );
6
7 CREATE TABLE pedido (
8   id_pedido SERIAL PRIMARY KEY,
9   fecha DATE,
10  id_rest INT REFERENCES restaurante(id_rest),
11  total NUMERIC(10, 2)
12 );
13
14 CREATE TABLE detallepedido (
15  id_detalle SERIAL PRIMARY KEY,
16  id_pedido INT REFERENCES pedido(id_pedido),
17  id_prod INT REFERENCES producto(id_prod),
18

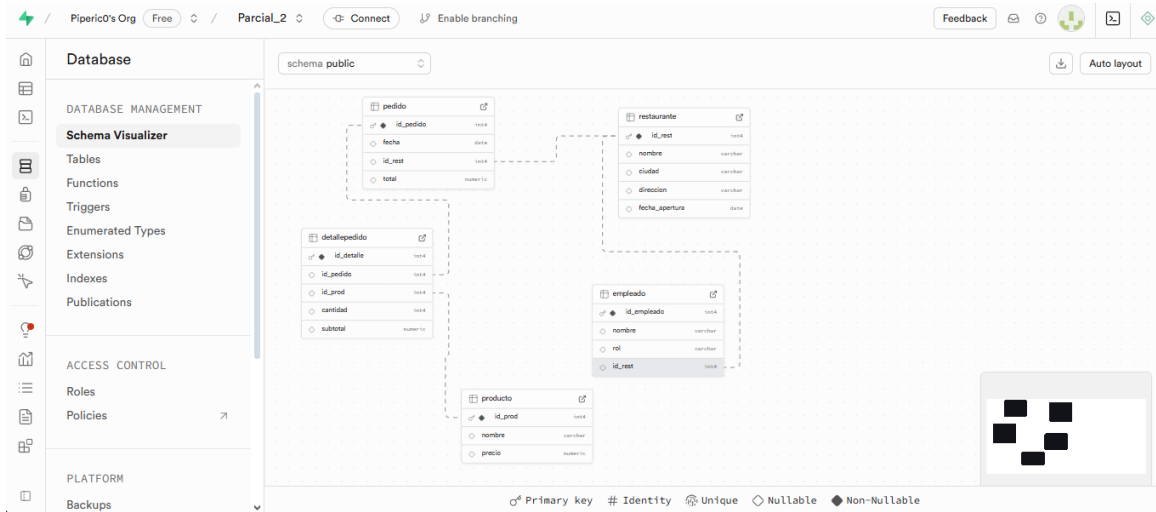
```

Results Chart Export

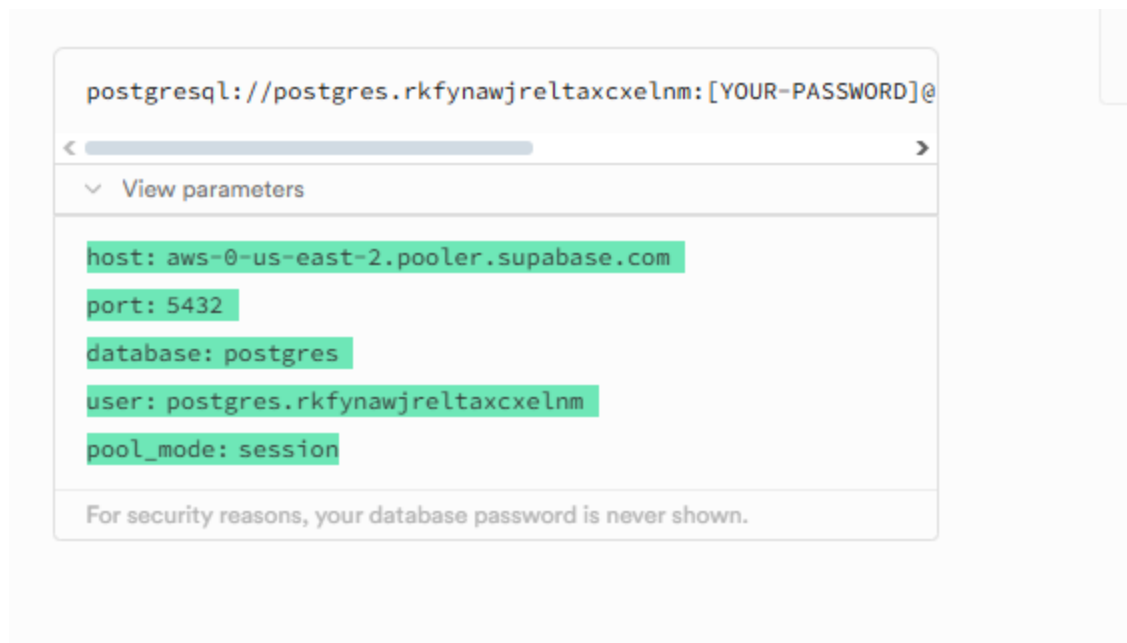
Success. No rows returned

Source Primary Database Role postgres Run CTRL

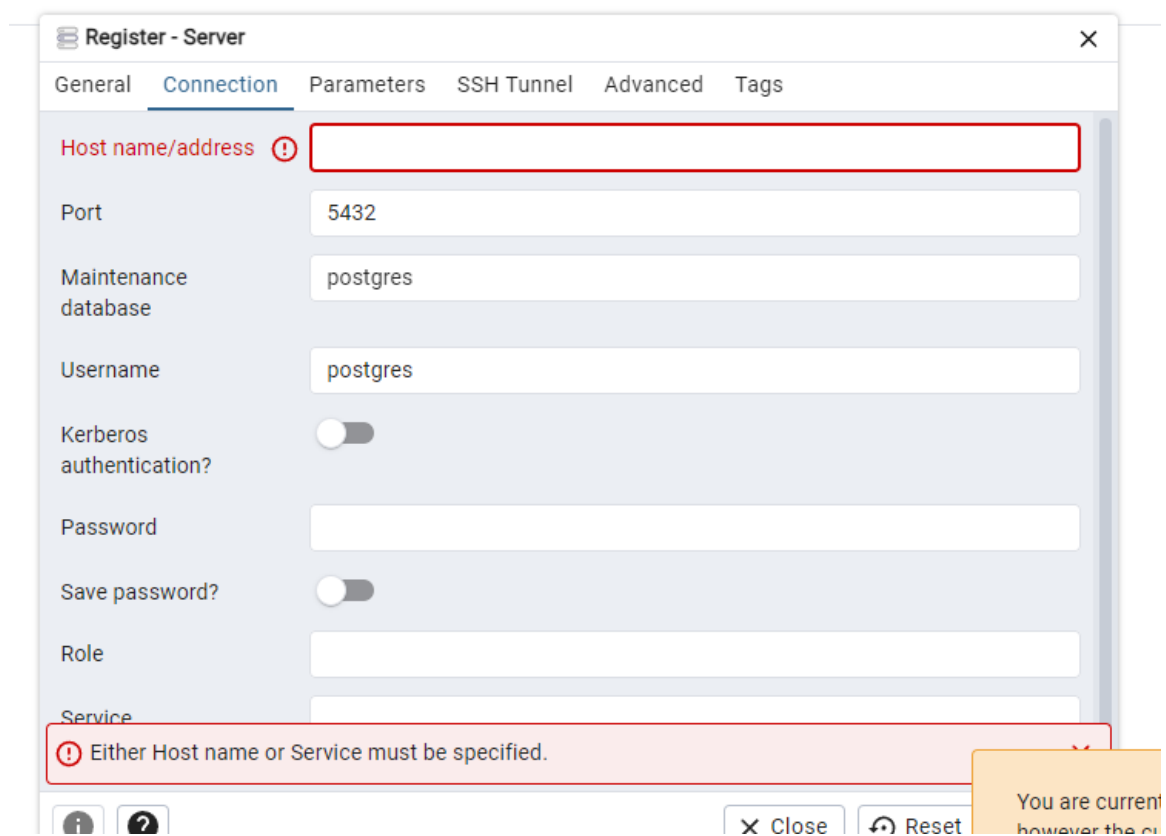
Luego verifico que se hayan creado bien las tablas



Vamos a la parte de connection para ver las url de conexión



Comenzamos a hacer la conexión con pgadmin4 con los datos que nos da supabase



Register - Server

General

Connection

Parameters

SSH Tunnel

Advanced

Tags

Host name/address

aws-0-us-east-2.pooler.supabase.com

Port

5432

Maintenance database

postgres

Username

postgres.rkfynawjreltaxcxelnm

Kerberos authentication?

Saving...

Password

.....

Save password?

Role

Service

i

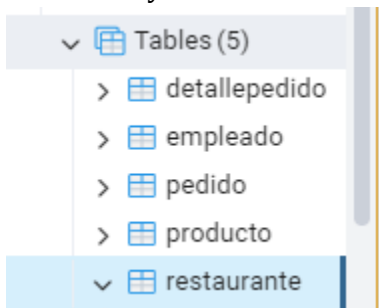
?

Close

Reset

Save

Revisamos y efectivamente nos conectó y aparecen las tablas que tenemos en la DB



Acá le pedí a chat 50 registros para cada tabla a chat y los inserte cada uno en cada tabla

```

1 INSERT INTO restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES
2 (1, 'Restaurante 1', 'Cali', 'Calle 47 # 42-13', '2021-02-09'),
3 (2, 'Restaurante 2', 'Cali', 'Calle 13 # 8-21', '2019-10-13'),
4 (3, 'Restaurante 3', 'Bogotá', 'Calle 71 # 35-13', '2019-09-23'),
5 (4, 'Restaurante 4', 'Barranquilla', 'Calle 68 # 24-2', '2022-01-29'),
6 (5, 'Restaurante 5', 'Cali', 'Calle 73 # 17-11', '2023-12-31'),
7 (6, 'Restaurante 6', 'Barranquilla', 'Calle 53 # 25-29', '2022-11-22'),
8 (7, 'Restaurante 7', 'Medellín', 'Calle 47 # 9-24', '2022-11-08'),
9 (8, 'Restaurante 8', 'Medellín', 'Calle 30 # 22-4', '2021-10-31'),
10 (9, 'Restaurante 9', 'Cali', 'Calle 8 # 23-17', '2021-09-28'),
11 (10, 'Restaurante 10', 'Medellín', 'Calle 89 # 41-10', '2022-09-09'),
12 (11, 'Restaurante 11', 'Medellín', 'Calle 81 # 42-26', '2023-05-16'),
13 (12, 'Restaurante 12', 'Bogotá', 'Calle 72 # 33-12', '2023-08-31'),
14 (13, 'Restaurante 13', 'Cartagena', 'Calle 29 # 7-18', '2019-07-04'),
15 (14, 'Restaurante 14', 'Cartagena', 'Calle 10 # 36-1', '2023-03-29'),
16 (15, 'Restaurante 15', 'Cartagena', 'Calle 56 # 30-5', '2022-05-21'),
17 (16, 'Restaurante 16', 'Bogotá', 'Calle 82 # 28-10', '2020-03-28'),
18 (17, 'Restaurante 17', 'Cali', 'Calle 1 # 13-25', '2019-03-04'),

```

```

1 INSERT INTO empleado (id_empleado, nombre, rol, id_rest) VALUES
2 (1, 'Empleado 1', 'Repartidor', 43),
3 (2, 'Empleado 2', 'Administrador', 33),
4 (3, 'Empleado 3', 'Mesero', 50),
5 (4, 'Empleado 4', 'Cocinero', 31),
6 (5, 'Empleado 5', 'Mesero', 2),
7 (6, 'Empleado 6', 'Cocinero', 24),
8 (7, 'Empleado 7', 'Administrador', 40),
9 (8, 'Empleado 8', 'Cajero', 12),
10 (9, 'Empleado 9', 'Cajero', 25),
11 (10, 'Empleado 10', 'Repartidor', 11),
12 (11, 'Empleado 11', 'Cocinero', 25),
13 (12, 'Empleado 12', 'Cocinero', 7),
14 (13, 'Empleado 13', 'Cocinero', 1),
15 (14, 'Empleado 14', 'Repartidor', 35),
16 (15, 'Empleado 15', 'Cocinero', 38),
17 (16, 'Empleado 16', 'Cajero', 47),
18 (17, 'Empleado 17', 'Mesero', 38),

```

```

1 INSERT INTO pedido (id_pedido, fecha, id_rest, total) VALUES
2 (1, '2024-04-05', 44, 20966),
3 (2, '2024-04-06', 44, 75719),
4 (3, '2024-04-17', 26, 83753),
5 (4, '2024-04-09', 8, 62993),
6 (5, '2024-04-08', 32, 67324),
7 (6, '2024-04-01', 42, 61503),
8 (7, '2024-04-11', 44, 74151),
9 (8, '2024-04-01', 3, 45578),
10 (9, '2024-04-02', 12, 91209),
11 (10, '2024-04-20', 40, 92499),
12 (11, '2024-04-06', 43, 76278),
13 (12, '2024-04-12', 49, 51030),
14 (13, '2024-04-01', 1, 29412),
15 (14, '2024-04-19', 34, 81195),
16 (15, '2024-04-08', 34, 59914),
17 (16, '2024-04-09', 1, 26534),
18 (17, '2024-04-01', 43, 57169),

```

Results Chart Export



Source

Primary Database

Role postgres

Run CTRL

Success. No rows returned



Results Chart Export ✓ ⋮ Source Primary Database Role postgres Run CTRL ↵

 / Pipercio's Org Free / **Parcial\_2** Connect Enable branching Feedback 📄 ? 📥 🔗

Search queries... +

> SHARED

PRIVATE (1)

3	
---	--

☐ ☐

Results Chart Export ⌵ ⋮ Source Primary Database ⌵ Role postgres ⌵ Run CTRL ↵

Success. No rows returned

Instalación de paquetes para mi visual

```
PS C:\Users\Felipe Rojas\Desktop\Parcial 2 Andres Rojas> npm install express cors pg

added 14 packages, and audited 96 packages in 3s

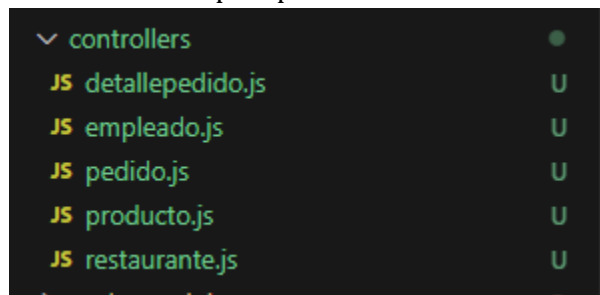
17 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Felipe Rojas\Desktop\Parcial 2 Andres Rojas> npm init -y
Wrote to C:\Users\Felipe Rojas\Desktop\Parcial 2 Andres Rojas\package.json:

{
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.5.0",
    "express": "^5.1.0",
    "mysql2": "^3.14.0",
    "pg": "^8.15.5",
    "postgres": "^3.4.5"
  },
  "name": "parcial-2-andres-rojas",
  "version": "1.0.0",
  "main": "DB.js",
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  }
}

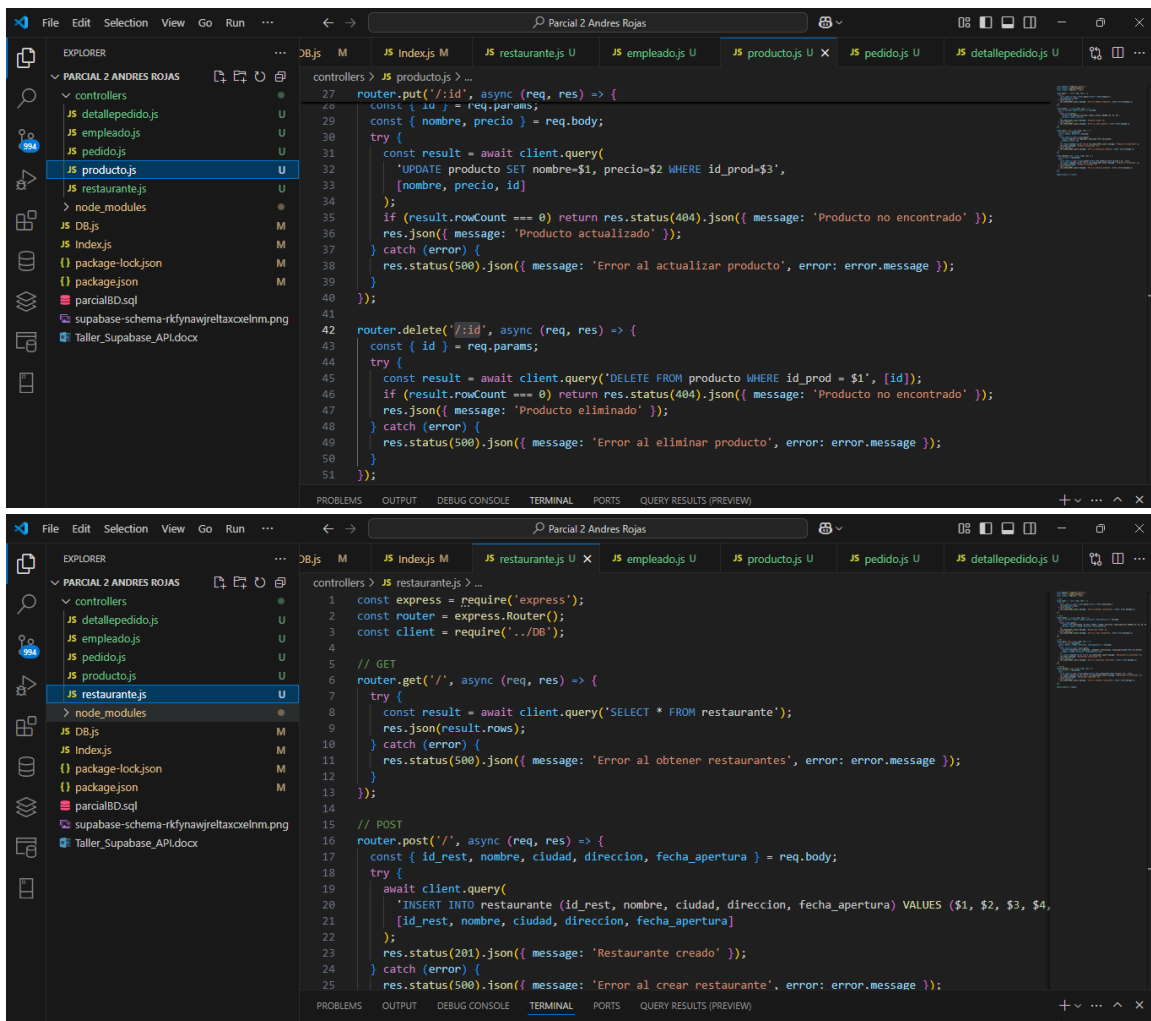
PS C:\Users\Felipe Rojas\Desktop\Parcial 2 Andres Rojas> node --watch index.js
Servidor corriendo en puerto 5000
```

Acá creo una carpeta para los controladores de cada tabla para que cada una sea individual

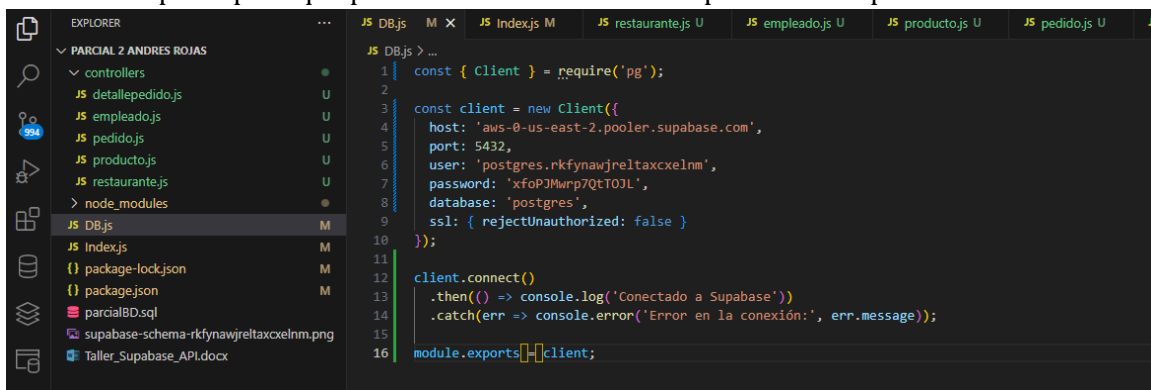


Ahora para la parte de la Api de cada una hare un Crud para cada una de las tablas, en cada código, implemento un conjunto de rutas en Express para gestionar los detalles de los pedidos en una base de datos PostgreSQL. Primero, defino un controlador para manejar las solicitudes GET, POST, PUT y DELETE. En el caso de GET, obtengo todos los registros de la tabla detallepedido. Para POST, inserto un nuevo detalle de pedido con los datos proporcionados en el cuerpo de la solicitud. En PUT, actualizo un detalle específico utilizando su id\_detalle, y en DELETE elimino un detalle de pedido por su id\_detalle. Utilizo async/await para trabajar con las consultas a la base de datos y manejo los errores con bloques try/catch, enviando respuestas directas según el resultado de cada operación y pues eso en cada uno de los controladores de cada tabla

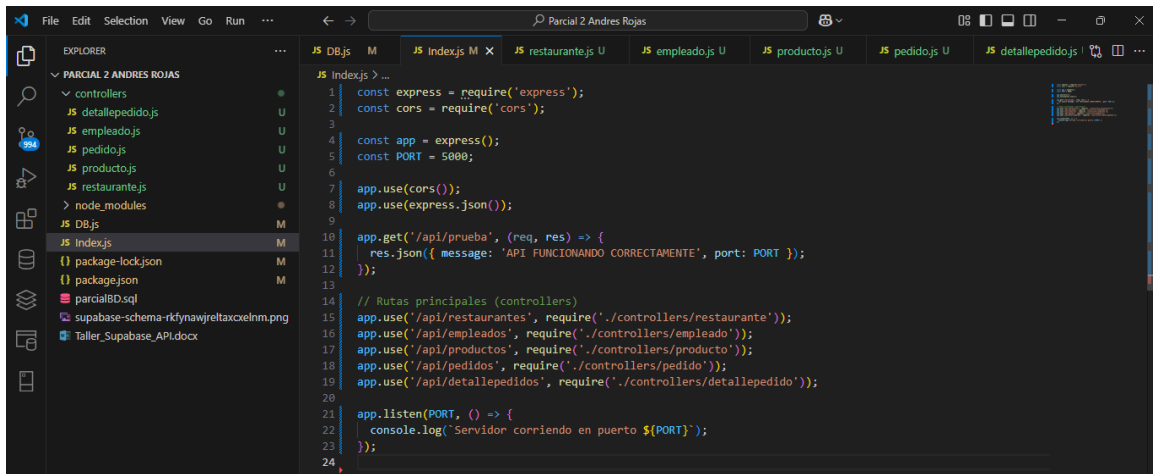




y pues ya en este código de DB.j se establece la conexión con la base de datos y se imprime un mensaje de éxito si la conexión es exitosa o un mensaje de error en caso contrario. El cliente se exporta para que pueda ser utilizado en otras partes de la aplicación.

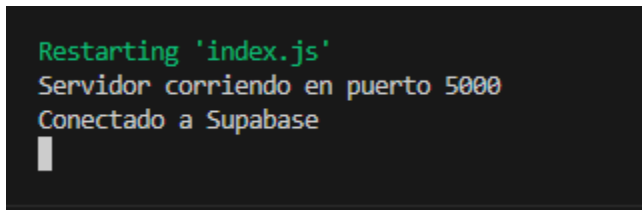


Para la parte de el index.js este código lo uso para crear un servidor con Express que maneja peticiones, permite conexiones externas con CORS y organiza las rutas de mi aplicación para gestionar restaurantes, empleados, productos y pedidos



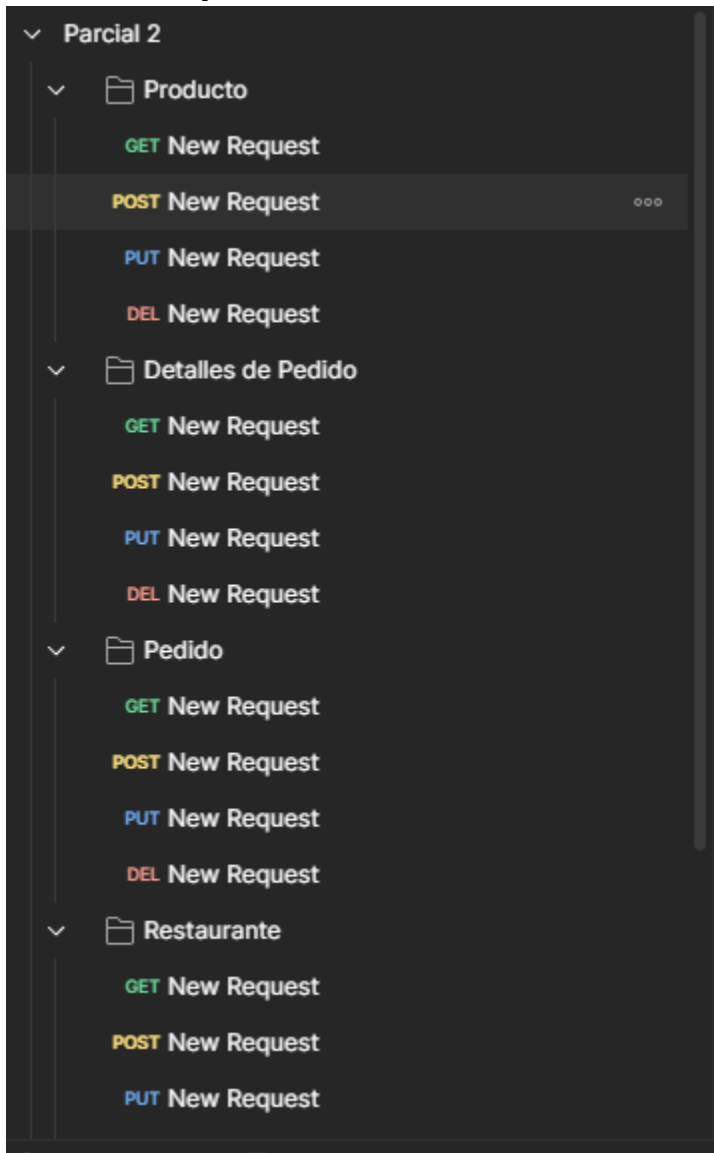
```
1 const express = require('express');
2 const cors = require('cors');
3
4 const app = express();
5 const PORT = 5000;
6
7 app.use(cors());
8 app.use(express.json());
9
10 app.get('/api/prueba', (req, res) => {
11   res.json({ message: 'API FUNCIONANDO CORRECTAMENTE', port: PORT });
12 });
13
14 // Rutas principales (controllers)
15 app.use('/api/restaurantes', require('./controllers/restaurante'));
16 app.use('/api/empleados', require('./controllers/empleado'));
17 app.use('/api/productos', require('./controllers/producto'));
18 app.use('/api/pedidos', require('./controllers/pedido'));
19 app.use('/api/detallepedidos', require('./controllers/detallepedido'));
20
21 app.listen(PORT, () => {
22   console.log('Servidor corriendo en puerto ${PORT}');
23 });
24
```

Verificamos que corra el servidor



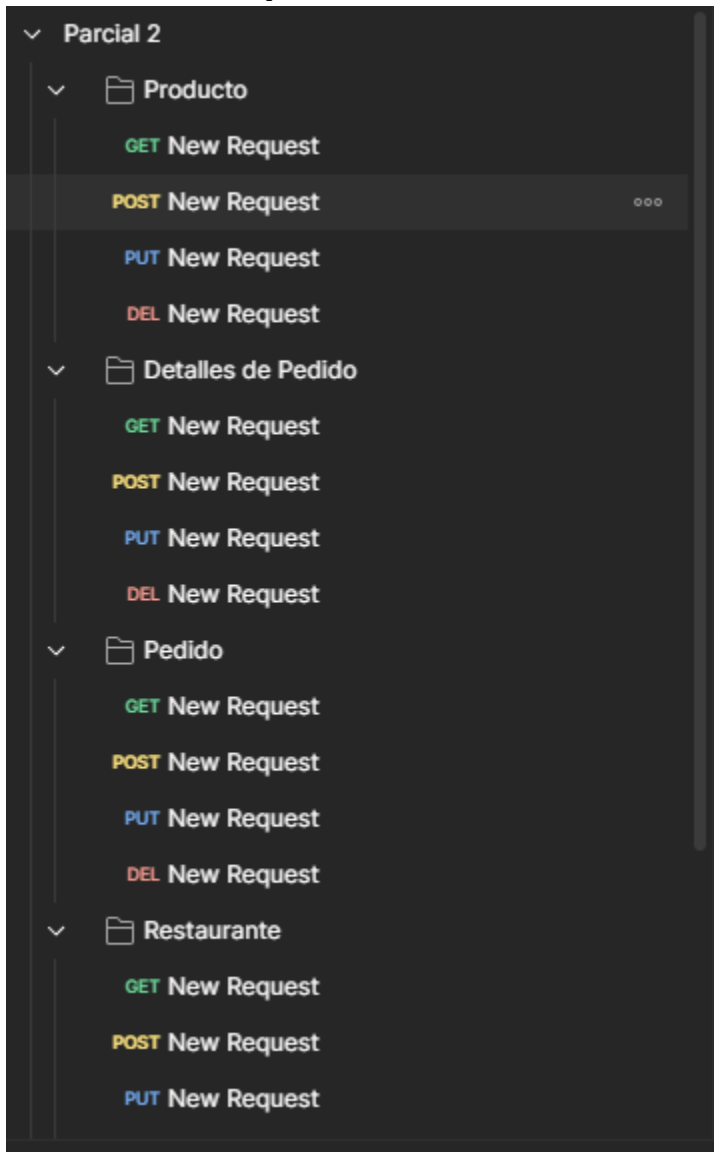
```
Restarting 'index.js'
Servidor corriendo en puerto 5000
Conectado a Supabase
```

Ahora empecé en postman a crear una colección y dentro de ella creo carpetas para cada tabla con su respectivo crud así en cada una de las tablas

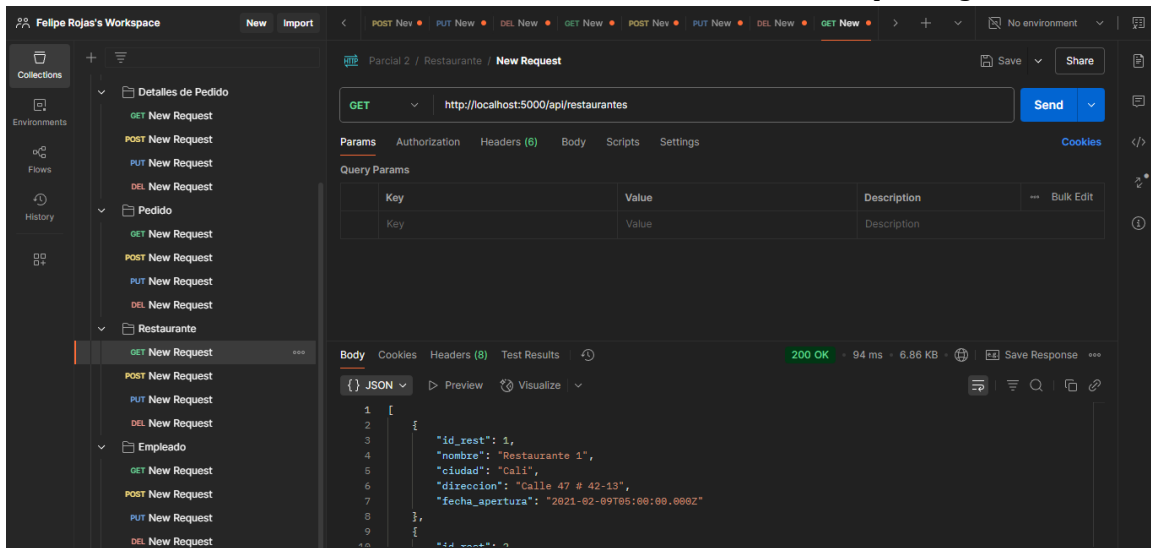


Ahora sacamos la url de cada una de las tablas con los controladores que tengo en visual  
Ahora empecé en postman a crear una colección y dentro de ella creo carpetas para

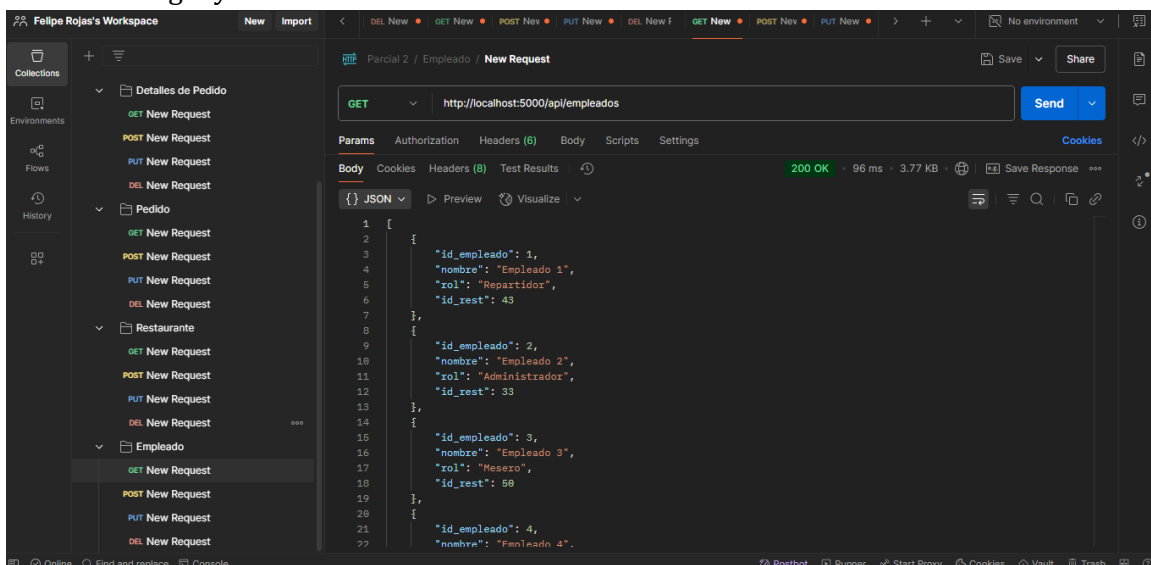
cada tabla con su respectivo crud así en cada una de las tablas



Ahora sacamos la url de cada una de las tablas con los controladores que tengo en visual

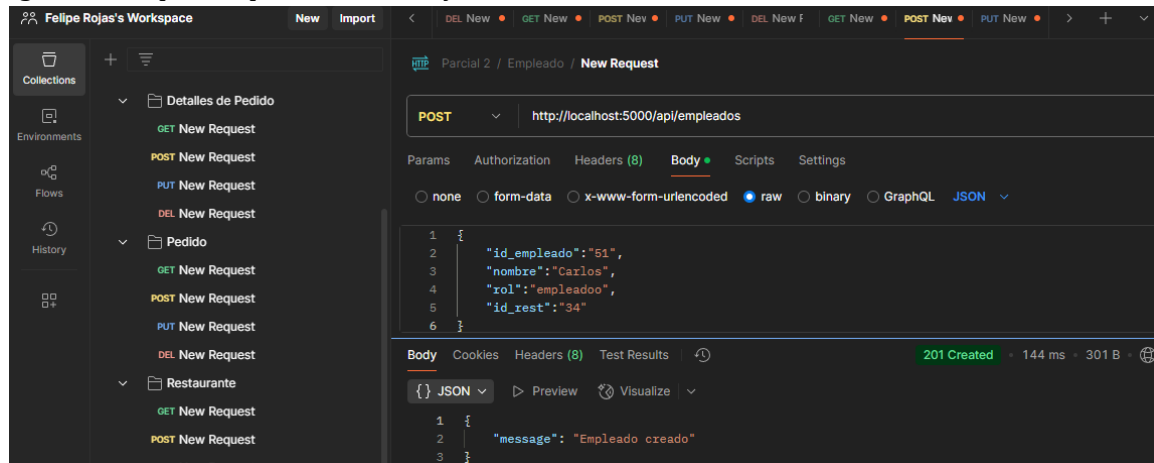


Probamos el get y me corrió en todas las tablas de manera correcta

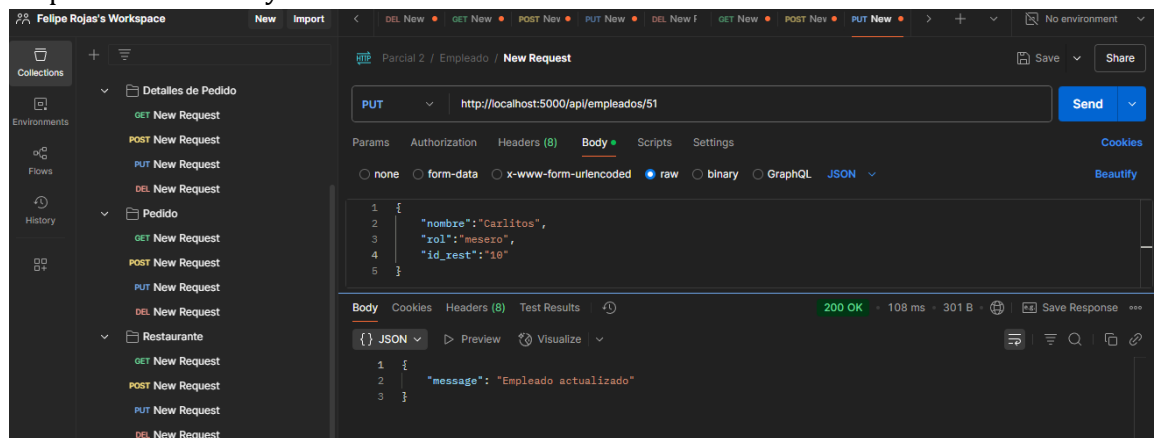




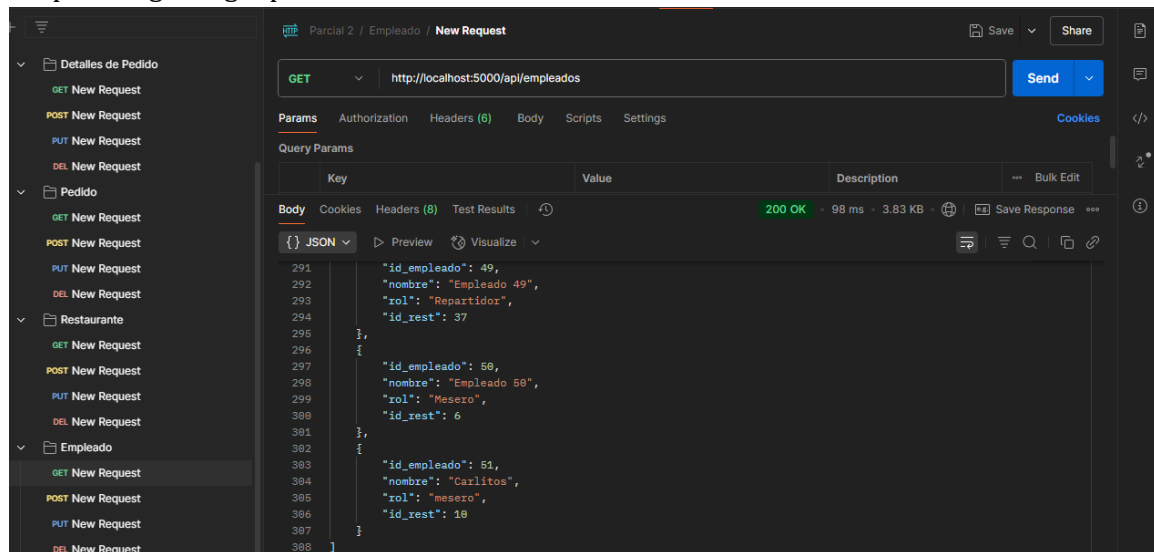
Igual con el post lo probe en todos y bien



En put modificamos y actualizamos



después hago un get para verificar los cambios



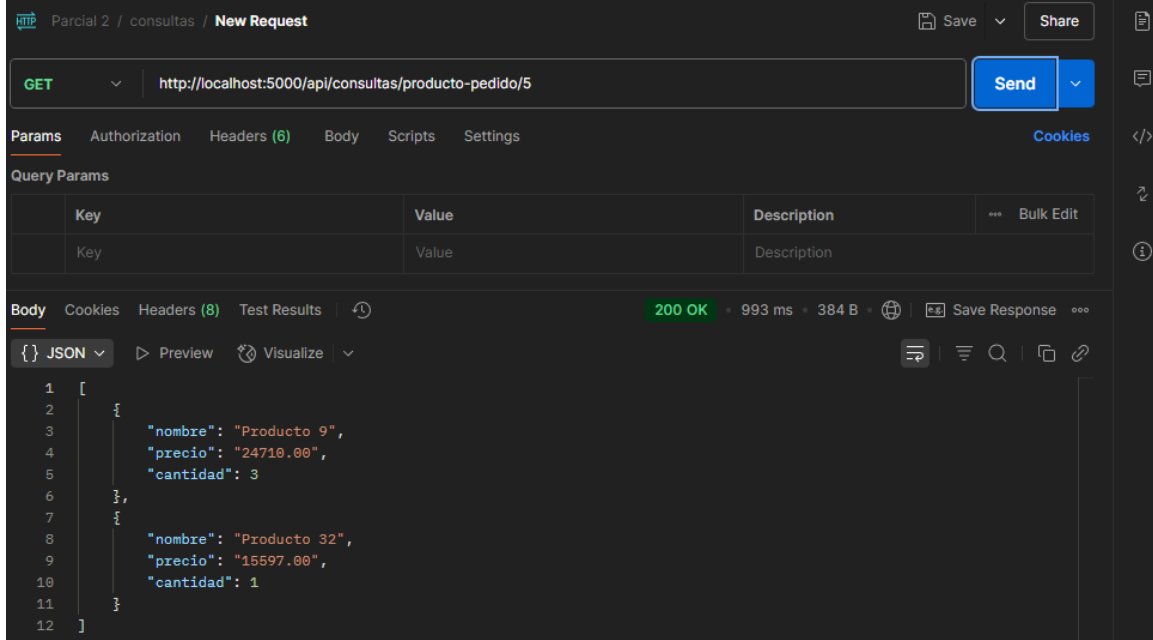
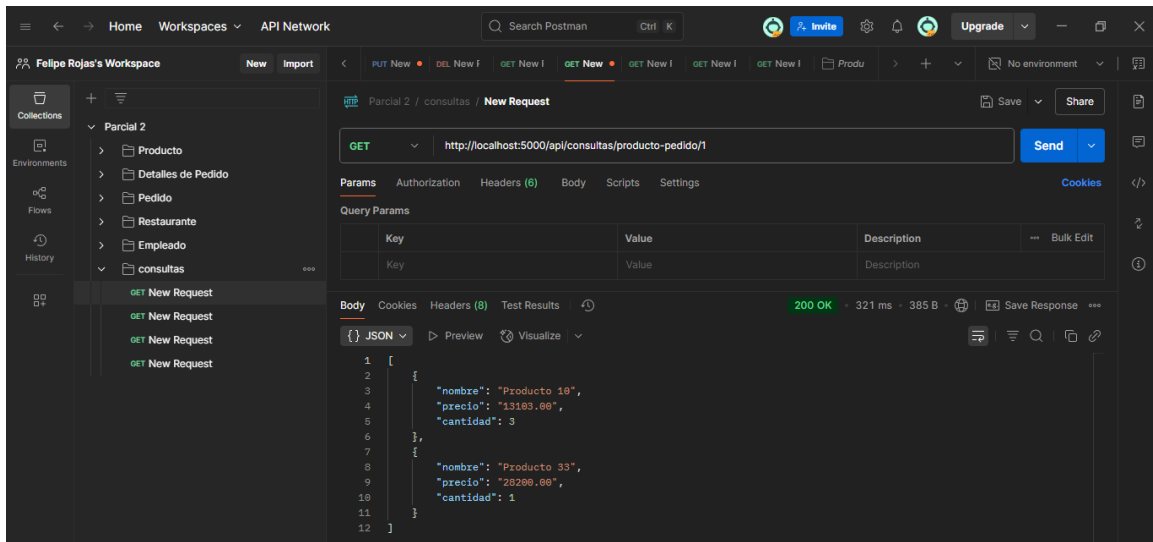
## Consultas

- **Obtener todos los productos de un pedido específico**

GET /producto-pedido/:id

obtengo los productos asociados a un pedido específico. Busco el nombre, precio y la cantidad de cada producto en ese pedido usando su ID.

```
5 router.get('/producto-pedido/:id', async (req, res) => {
6   const { id } = req.params;
7   try {
8     const result = await client.query(
9       `SELECT p.nombre, p.precio, dp.cantidad
10        FROM detallepedido dp
11        JOIN producto p ON dp.id_prod = p.id_prod
12        WHERE dp.id_pedido = $1`, [id]);
13     res.json(result.rows);
14   } catch (err) {
15     res.status(500).json({ error: err.message });
16   }
17 });
```

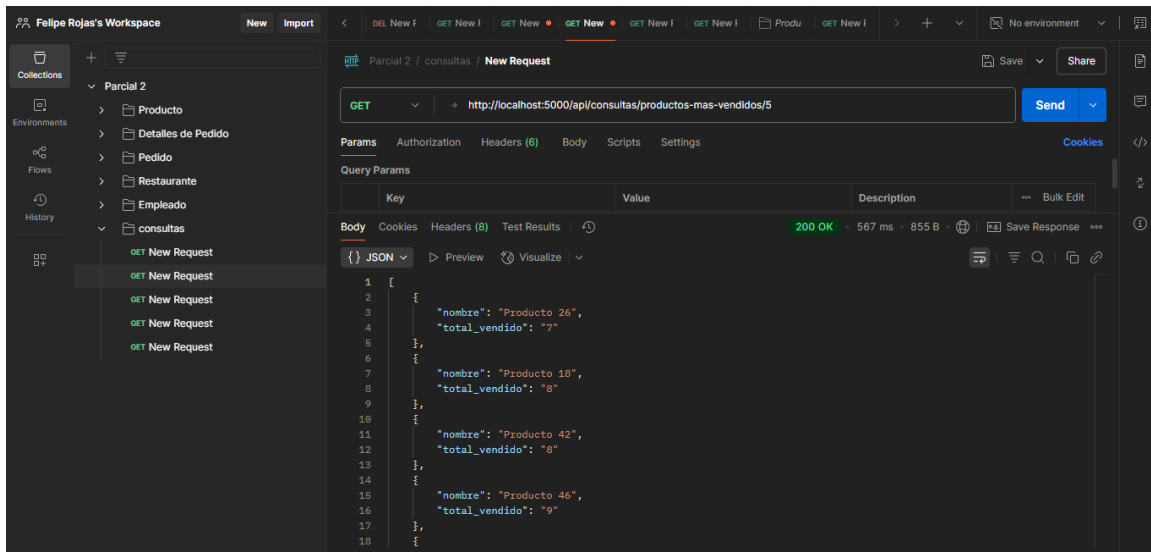


## • Obtener los productos más vendidos (más de X unidades) }

GET /productos-mas-vendidos/:min

pido que muestre los productos que se han vendido más de una cantidad mínima. Agrupo los productos y sumo cuántas veces se han pedido para filtrar solo los más vendidos.

```
router.get('/productos-mas-vendidos/:min', async (req, res) => {
  const { min } = req.params;
  try {
    const result = await client.query(
      `SELECT p.nombre, SUM(dp.cantidad) AS total_vendido
      FROM detallepedido dp
      JOIN producto p ON dp.id_prod = p.id_prod
      GROUP BY p.nombre
      HAVING SUM(dp.cantidad) > $1`, [min]);
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```



## • Obtener el total de ventas por restaurante

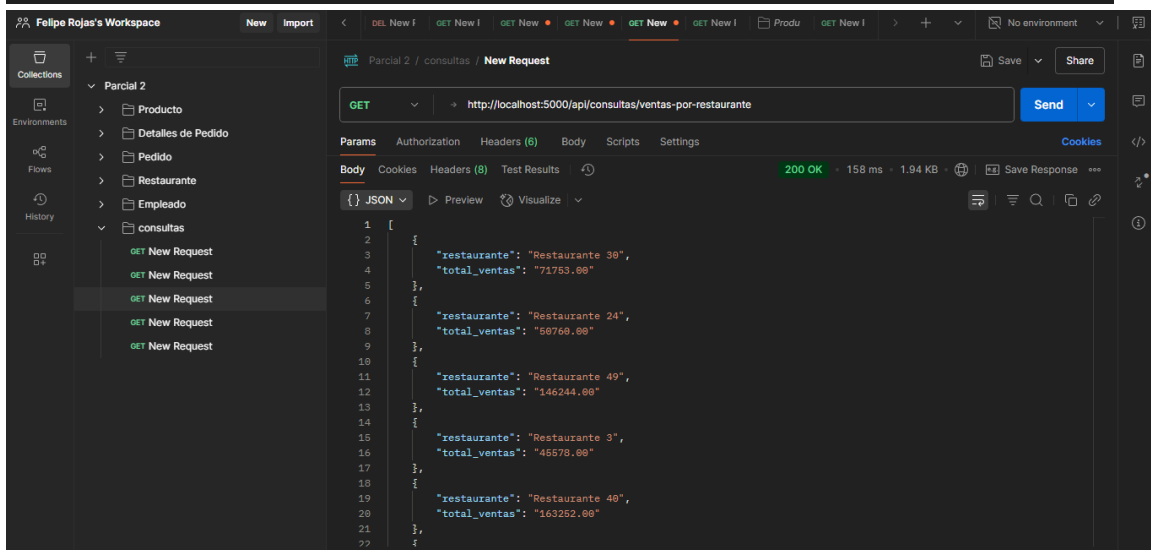
GET /ventas-por-restaurante

hago un calculo de las ventas totales de cada restaurante. Sumo el total de todos los pedidos y los agrupo por restaurante para ver cuál ha vendido más.

```

router.get('/ventas-por-restaurant', async (req, res) => {
  try {
    const result = await client.query(
      `SELECT r.nombre AS restaurante, SUM(p.total) AS total_ventas
      FROM pedido p
      JOIN restaurante r ON p.id_rest = r.id_rest
      GROUP BY r.nombre`;
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

```

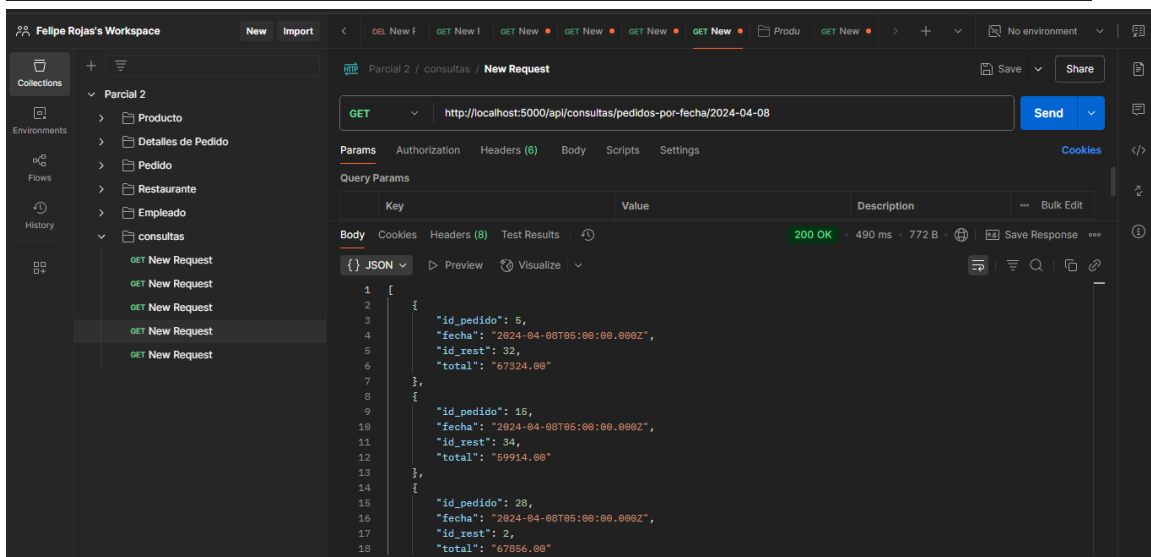


## • Obtener los pedidos realizados en una fecha específica

GET /pedidos-por-fecha/:fecha

De la base de datos busco todos los pedidos realizados en una fecha específica. Filtro los registros para que solo aparezcan los que coinciden con la fecha dada.

```
router.get('/pedidos-por-fecha/:fecha', async (req, res) => {
  const { fecha } = req.params;
  try {
    const result = await client.query(
      `SELECT * FROM pedido
      WHERE fecha::DATE = $1`, [fecha]);
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```



## • Obtener los empleados por rol en un restaurante

GET /empleados-por-rol/:restaurantId/:rol

De la base de datos obtengo los empleados de un restaurante según su rol los filtro por el ID del restaurante y el rol (por ejemplo, cocinero o mesero).

```

router.get('/empleados-por-rol/:restauranteId/:rol', async (req, res) => {
  const { restauranteId, rol } = req.params;
  try {
    const result = await client.query(
      `SELECT nombre, rol
      FROM empleado
      WHERE id_rest = $1 AND rol = $2`, [restauranteId, rol]);
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

module.exports = router;

```

Felipe Rojas's Workspace

Parcial 2 / consultas / New Request

GET http://localhost:5000/api/consultas/empleados-por-rol/1/Cocinero

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value	Description	Bulk Edit

Body Cookies Headers (8) Test Results

200 OK · 640 ms · 310 B · Save Response

JSON Preview Visualize

```

1 [
2   {
3     "nombre": "Empleado 13",
4     "rol": "Cocinero"
5   }
6 ]

```

Felipe Rojas's Workspace

Parcial 2 / consultas / New Request

GET http://localhost:5000/api/consultas/empleados-por-rol/1/Repartidor

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value	Description	Bulk Edit

Body Cookies Headers (8) Test Results

200 OK · 509 ms · 312 B · Save Response

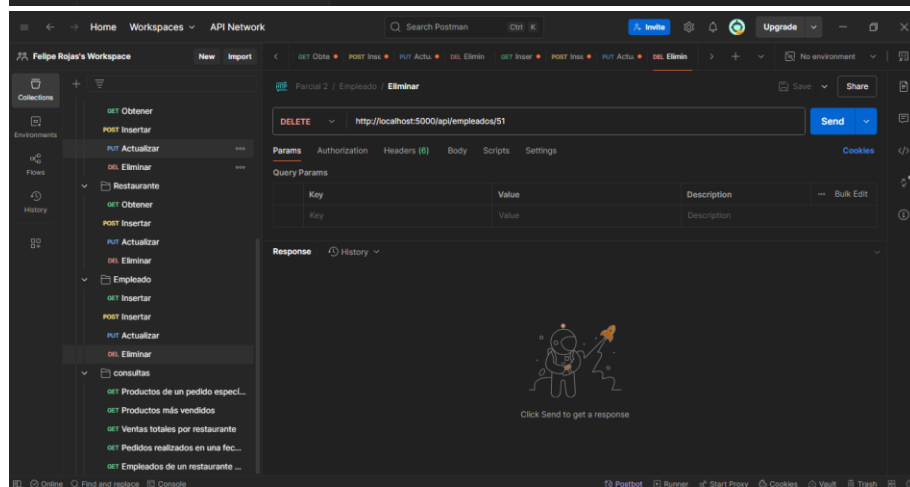
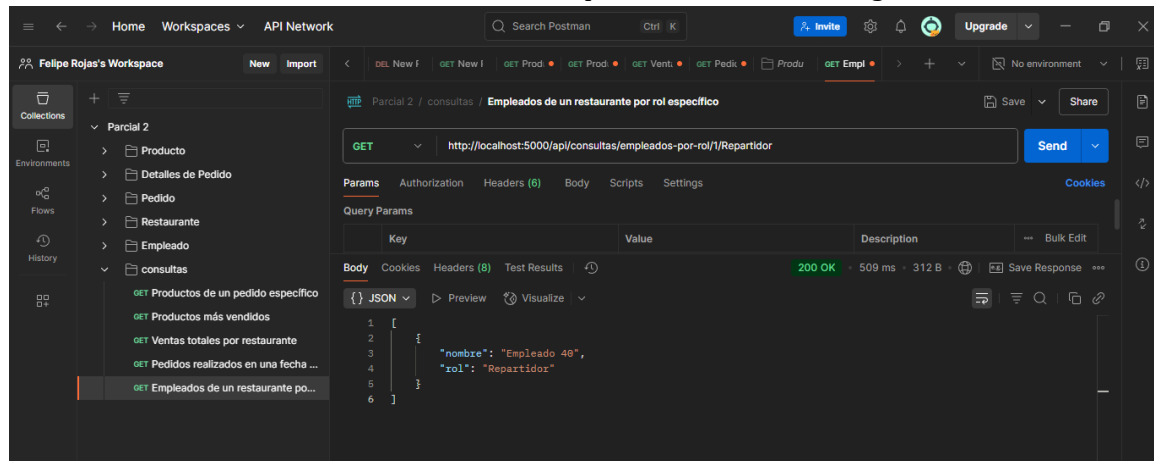
JSON Preview Visualize

```

1 [
2   {
3     "nombre": "Empleado 40",
4     "rol": "Repartidor"
5   }
6 ]

```

Por ultimo cambio el nombre de las consultas para tener todo mas organizado



## Conclusiones

Este proyecto permitió poner en práctica lo aprendido sobre cómo crear y manejar una API. Se logró conectar correctamente el sistema con la base de datos y probar que todas las funciones (ver, agregar, modificar y eliminar datos) funcionaran bien. También se aprendió a organizar el código por partes, para que todo fuera más fácil de entender y mantener. El trabajo fue una buena forma de ver cómo se aplica en la vida real lo que se estudia en clase,



y cómo diferentes herramientas se pueden usar juntas para crear un sistema completo y funcional.