

DOCUMENTACION PARCIAL

MIGUEL ANGEL JIMENEZ PORRAS

ID 834889

BASES DE DATOS MASIVAS

NRC:60747

WILLIAM ALEXANDER MATALLANA PORRAS

BASES DE DATOS MASIVAS

5/04/2025

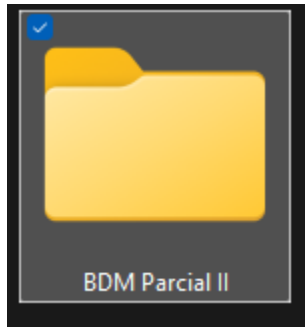
CORPORACIÓN UNIVERSITARIA MINUTO DE DIOS UNIMINUTO

Contenido

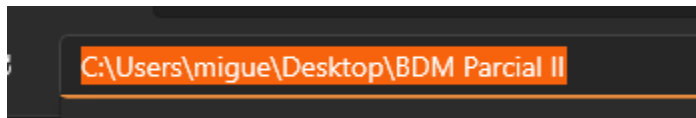
Creación de la carpeta del proyecto.....	3
Creación de base de datos en Supabase	4
Asociación de la base de datos con Pgadmin4.....	5
Creación de las tablas con Pgadmin4.....	8
Creación de tabla Restaurante	9
Creación de tabla Empleado	9
Creación de tabla Producto	10
Creación de tabla Pedido	10
Creación de tabla DetallePedido	11
Conexión de la base de datos en la carpeta	12
Consultas	15
Crud tabla Restaurante.....	15
Table Empleados.....	18
Table Productos.....	21
Table Pedidos.....	24
Table DetallesPedido.....	27
Consultas Nativas.....	30

Creación de la carpeta del proyecto

Para crear la carpeta del proyecto, que va a ser la que contenga todo el contenido, lo primero, será crear la carpeta, en sí, entonces en la ubicación que deseemos, creamos una nueva carpeta, en mi caso fue en Desktop



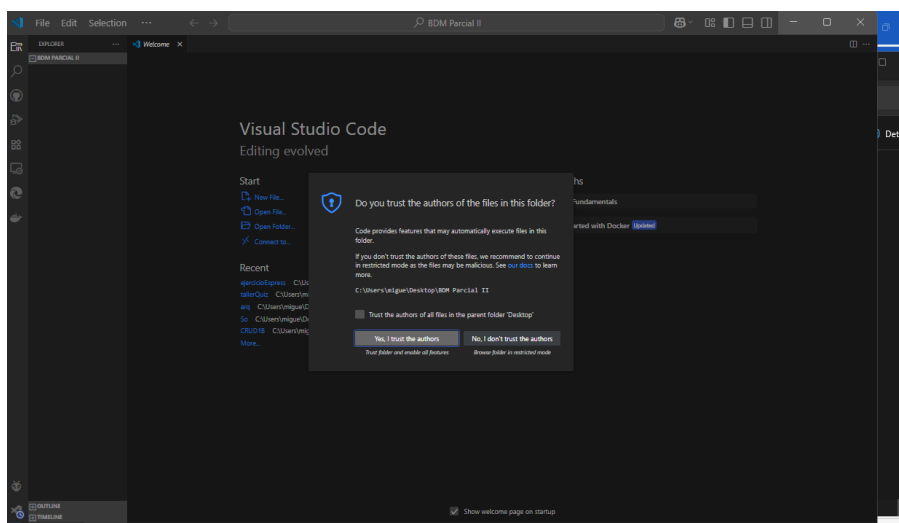
Posteriormente, se copia el path de la carpeta, que nos servirá para que en Visual Studio Code, la peguemos y nos ahorre tiempo buscando la carpeta



Posteriormente, en el entorno de Visual Studio Code, nos dirigimos a la barra de la parte superior izquierda y seleccionamos *open folder*, para empezar a trabajar en el proyecto



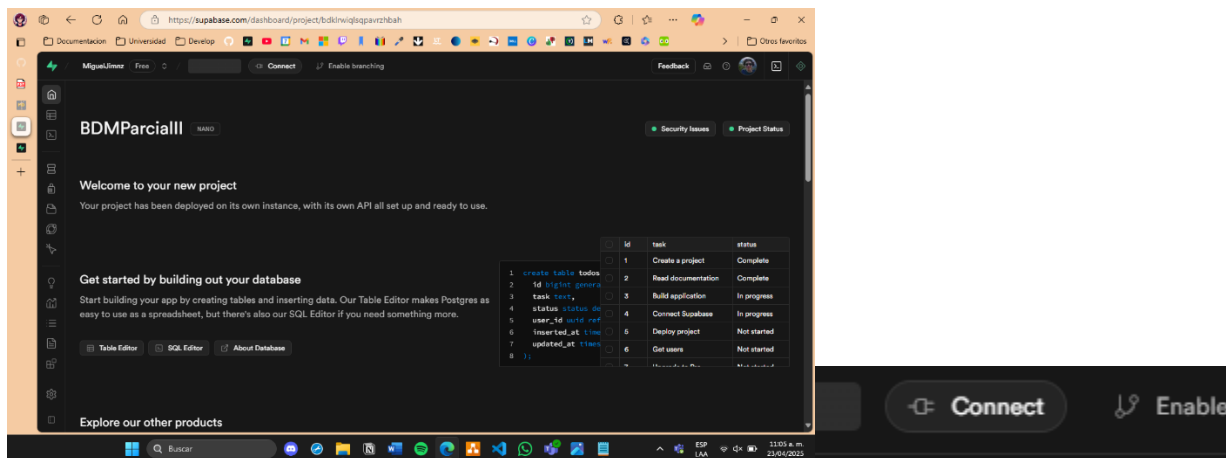
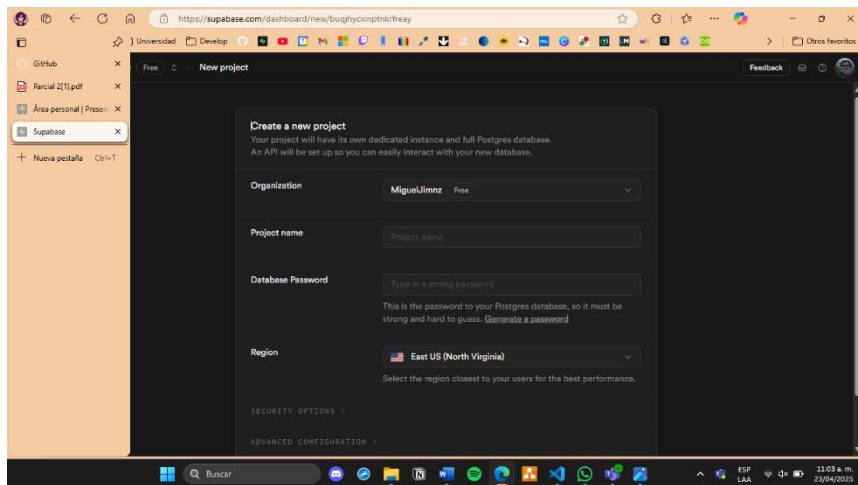
Y en la ventana emergente, seleccionamos *Trust the authors*

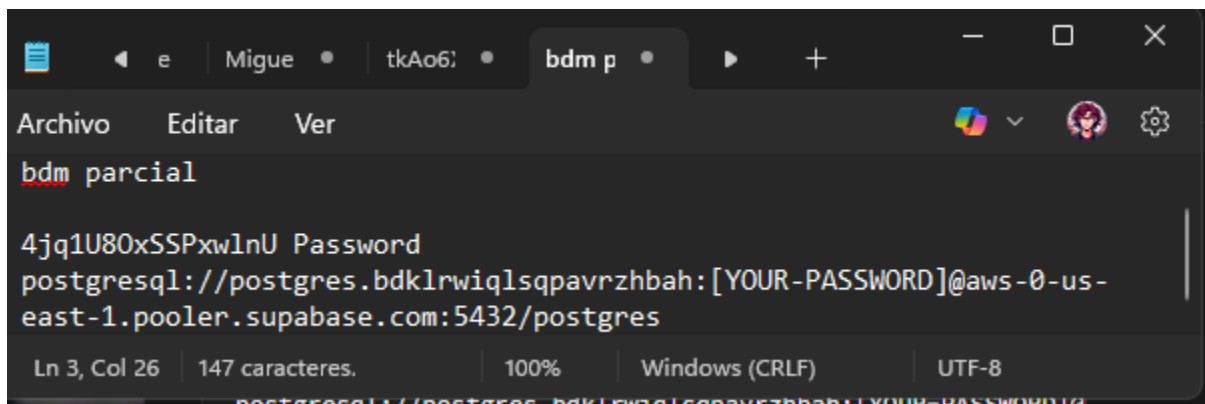
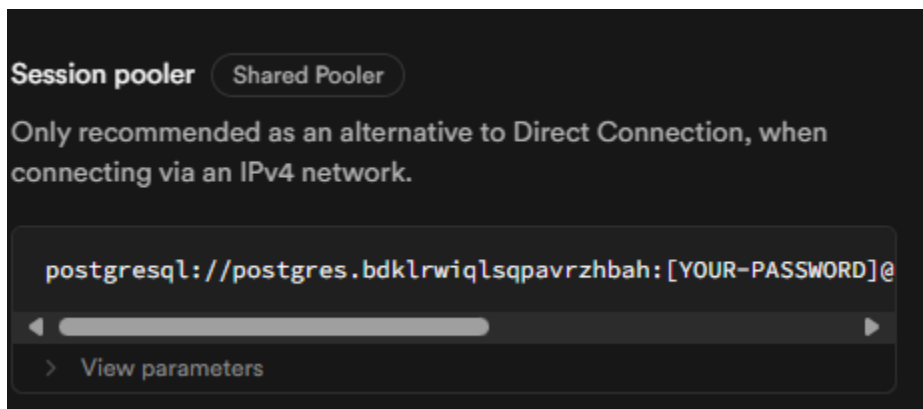


Con esto, ya estaremos en el entorno para trabajar.

Creación de base de datos en Supabase

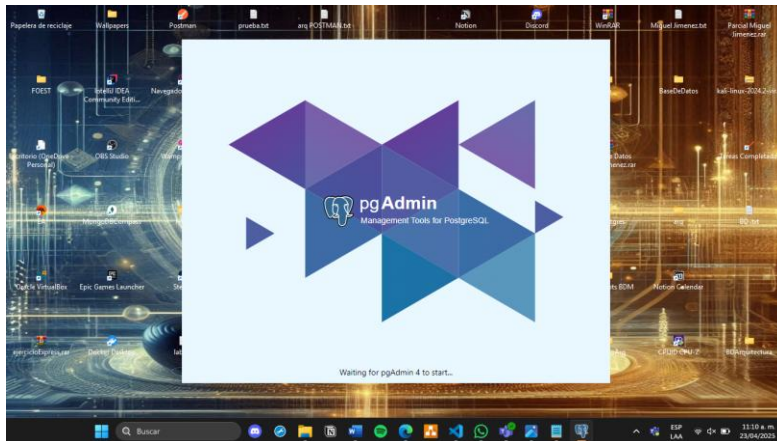
Para crear la base de datos en Supabase, lo primero será crear un nuevo proyecto de base de datos, una vez creado, ir a la sección de *Connect*, en la parte superior, y después en la parte de *Session Pooler*, y los datos registrados los guardamos para posteriormente conectarnos a la base de datos

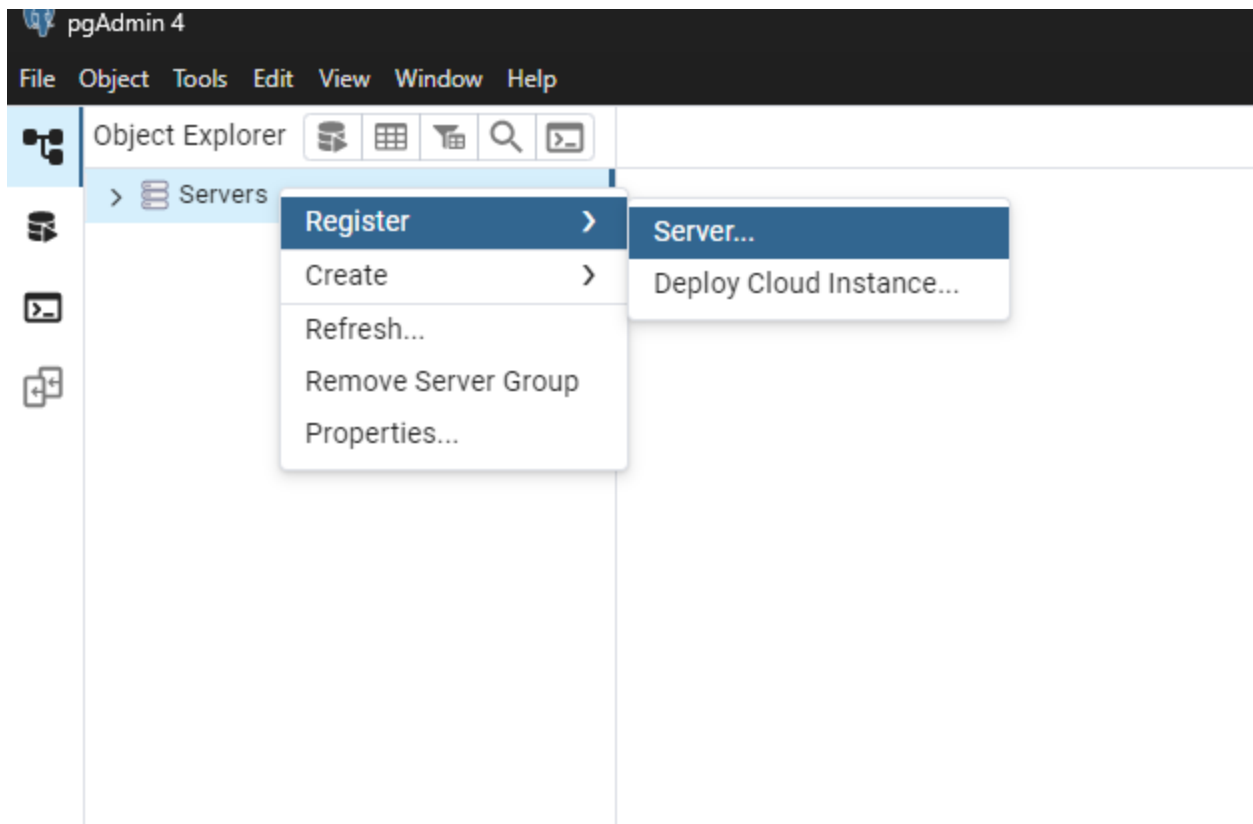




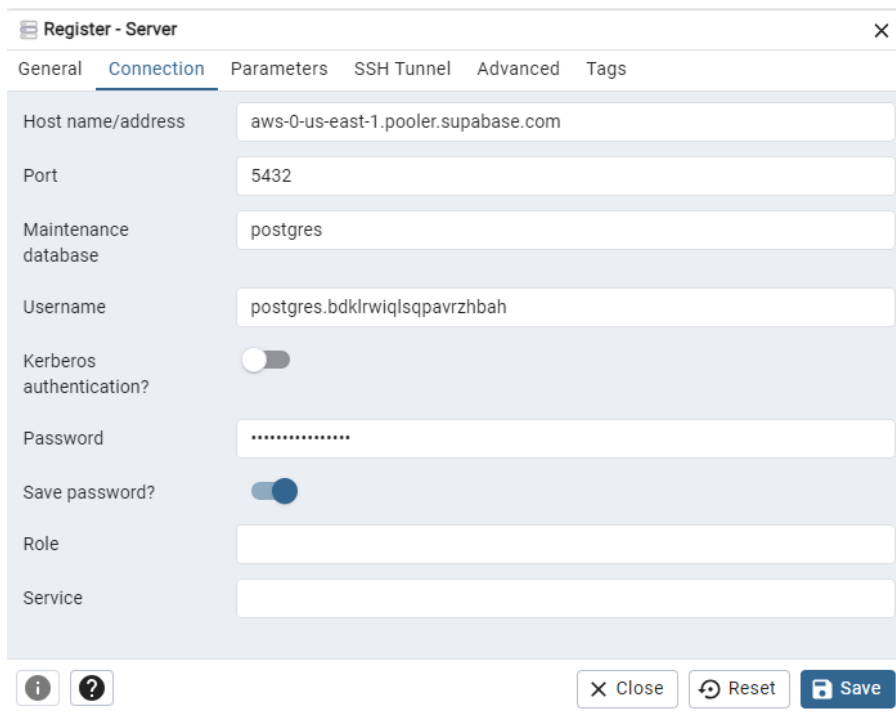
Asociación de la base de datos con Pgadmin4

Para que Pgadmin4, sea nuestra herramienta de trabajo para la base de datos, es necesario que, primero que todo, abriendo el programa y con los datos de conexión, previamente, guardados, en el menú *Servers*, seleccionar con el click derecho -> Register-> Server

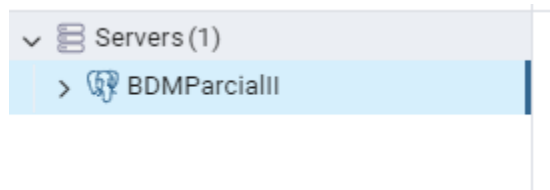




Y en la interfaz de *connection*, establecer los valores, previamente gurdados, el enlace, user, password.

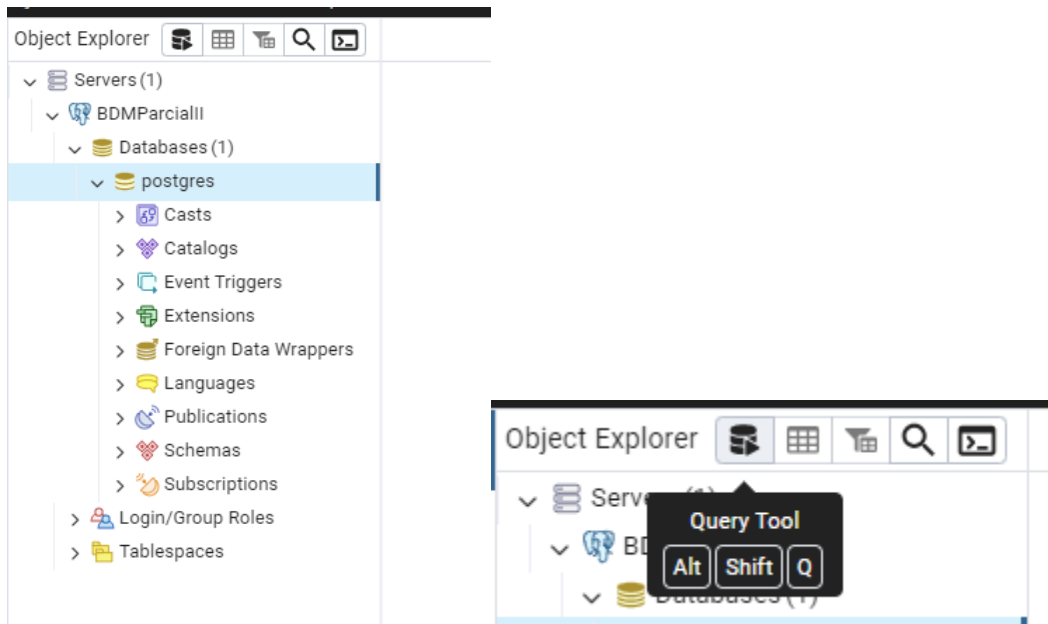


Y si los datos ingresados fueron correctos, nos abrirá la conexión

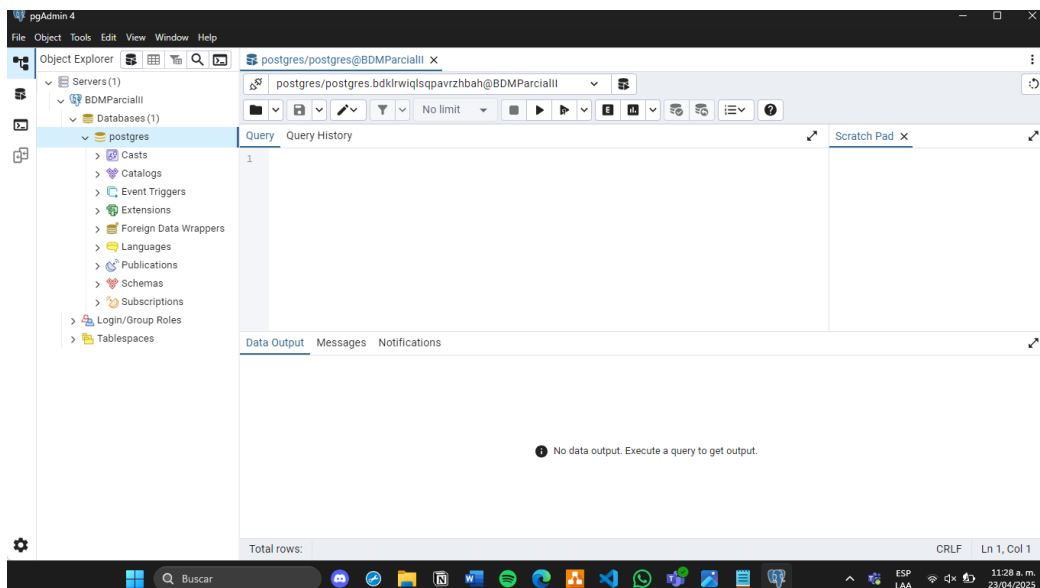


Creación de las tablas con Pgadmin4.

Para crear las tablas del proyecto, lo primero será en el apartado de la conexión, desplegarlo, después *Databases -> postgres*, para que nos habilite el apartado de *query tool*



Y en el apartado de *query tool*, se procede a escribir los comandos de cada tabla.



Creación de tabla Restaurante

The screenshot shows the pgAdmin 4 interface. On the left, the 'Servers' tree is expanded to 'BDMParcialII' > 'postgres'. The 'Query' tab is active, displaying the following SQL code:

```
1 CREATE TABLE Restaurante (  
2     id_rest INT PRIMARY KEY,  
3     nombre VARCHAR(100),  
4     ciudad VARCHAR(100),  
5     direccion VARCHAR(150),  
6     fecha_apertura DATE  
7 );
```

Below the query editor, the 'Messages' tab shows the execution result:

```
CREATE TABLE  
  
Query returned successfully in 2 secs 70 msec.
```

The status bar at the bottom indicates 'Total rows: Query complete 00:00:02.074' and 'Ln 7, Col 3'.

Creación de tabla Empleado

The screenshot shows the pgAdmin 4 interface. On the left, the 'Servers' tree is expanded to 'BDMParcialII' > 'postgres'. The 'Query' tab is active, displaying the following SQL code:

```
1 CREATE TABLE Empleado (  
2     id_empleado INT PRIMARY KEY,  
3     nombre VARCHAR(100),  
4     rol VARCHAR(50),  
5     id_rest INT,  
6     FOREIGN KEY (id_rest) REFERENCES Restaurante(id_rest)  
7 );
```

Below the query editor, the 'Messages' tab shows the execution result:

```
CREATE TABLE  
  
Query returned successfully in 183 msec.
```

Creación de tabla Producto

The screenshot shows a database management interface with a sidebar on the left containing a tree view of database objects. The main area is titled 'Query' and 'Query History'. The SQL query being executed is:

```
1 CREATE TABLE Producto (  
2     id_prod INT PRIMARY KEY,  
3     nombre VARCHAR(100),  
4     precio NUMERIC(10, 2)  
5 );
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing the output of the query.

Creación de tabla Pedido

The screenshot shows the same database management interface as the previous one, but with a different SQL query being executed. The query is:

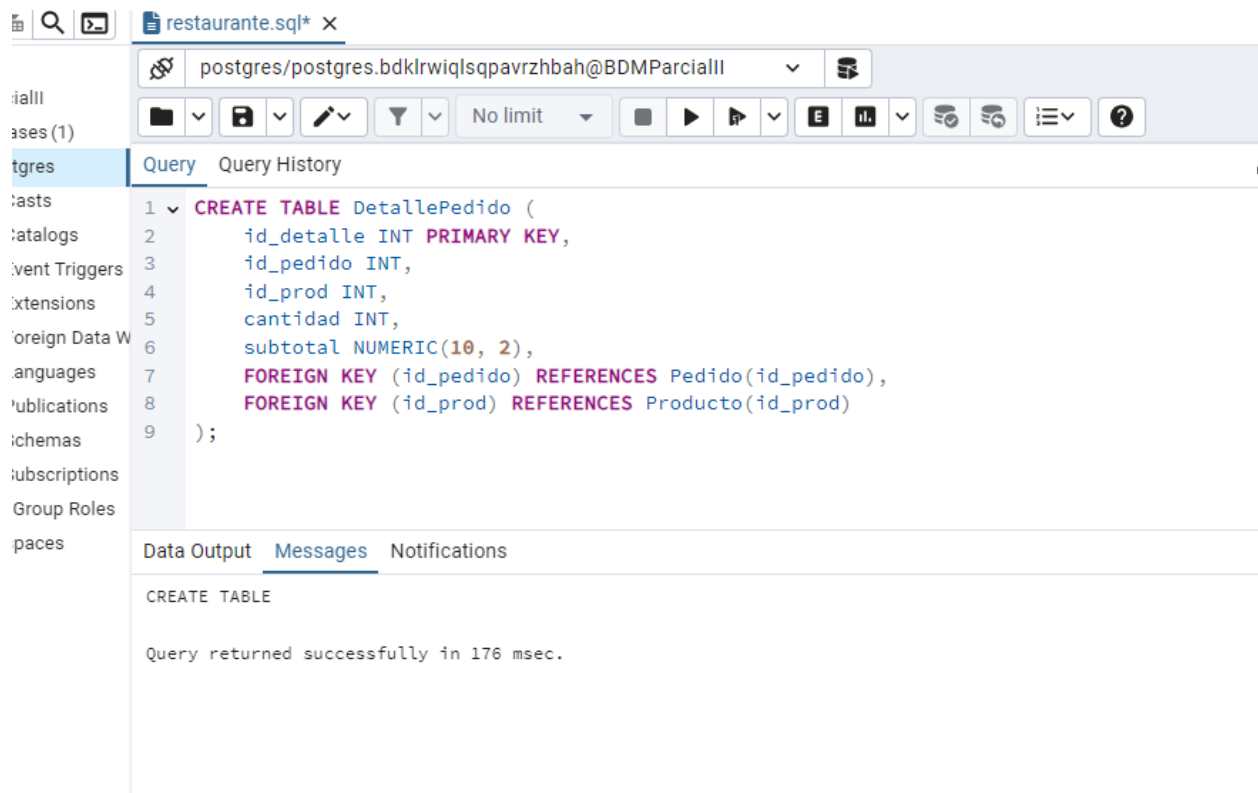
```
1 CREATE TABLE Pedido (  
2     id_pedido INT PRIMARY KEY,  
3     fecha DATE,  
4     id_rest INT,  
5     total NUMERIC(10, 2),  
6     FOREIGN KEY (id_rest) REFERENCES Restaurante(id_rest)  
7 );
```

Below the query editor, the 'Messages' tab is selected, showing the output of the query:

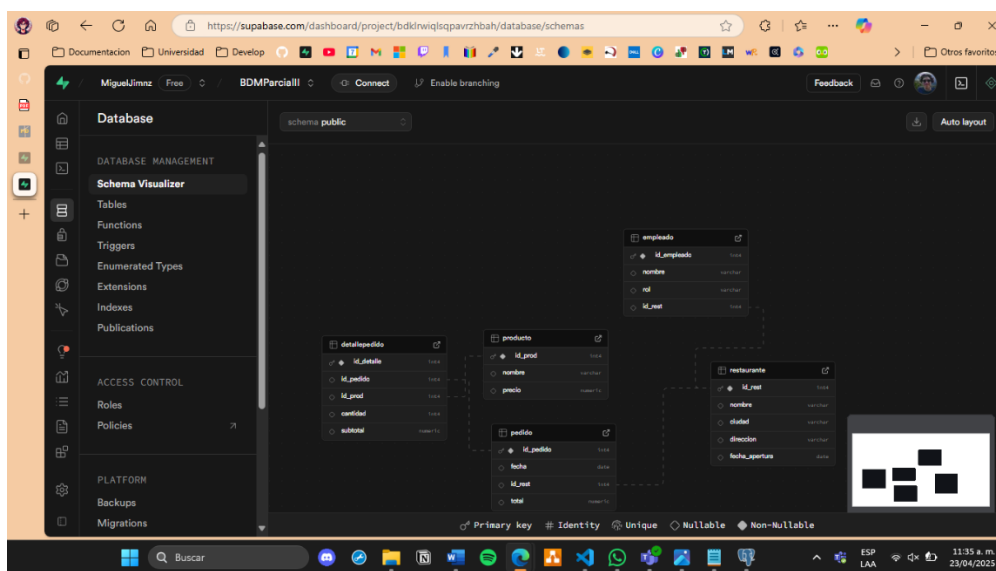
```
CREATE TABLE
```

Query returned successfully in 172 msec.

Creación de tabla DetallePedido



Una vez, con las tablas hechas, solo falta revisar en la conexión de supabase, el modelo de las tablas.



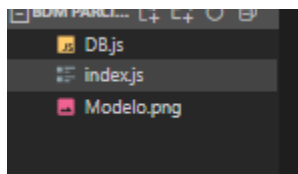
Y con el modelo, lo podemos descargar



Conexión de la base de datos en la carpeta

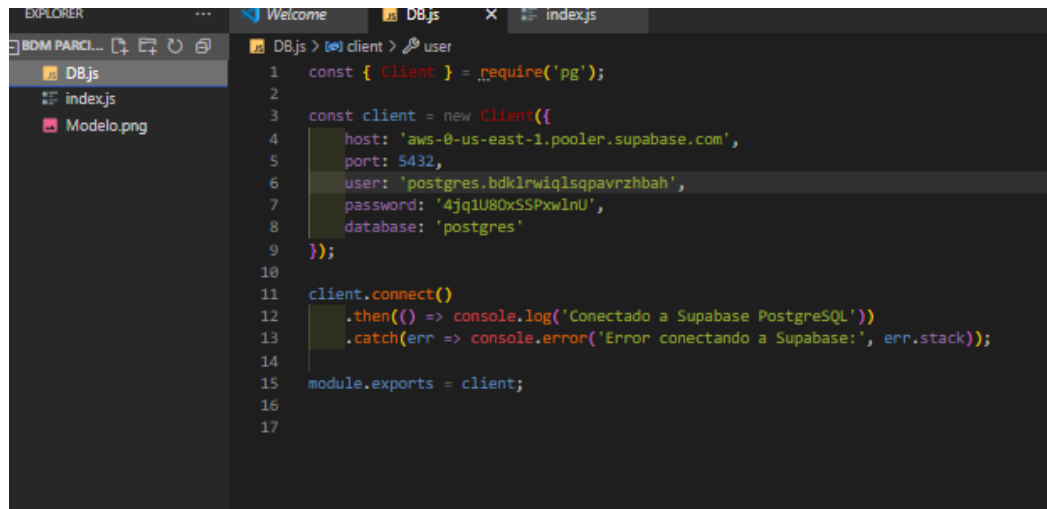
Para ello, lo primero es crear los documentos

1. DB.js
2. Index.js



Una vez creado, se establece la conexión para la base de datos y se verifica su conexión

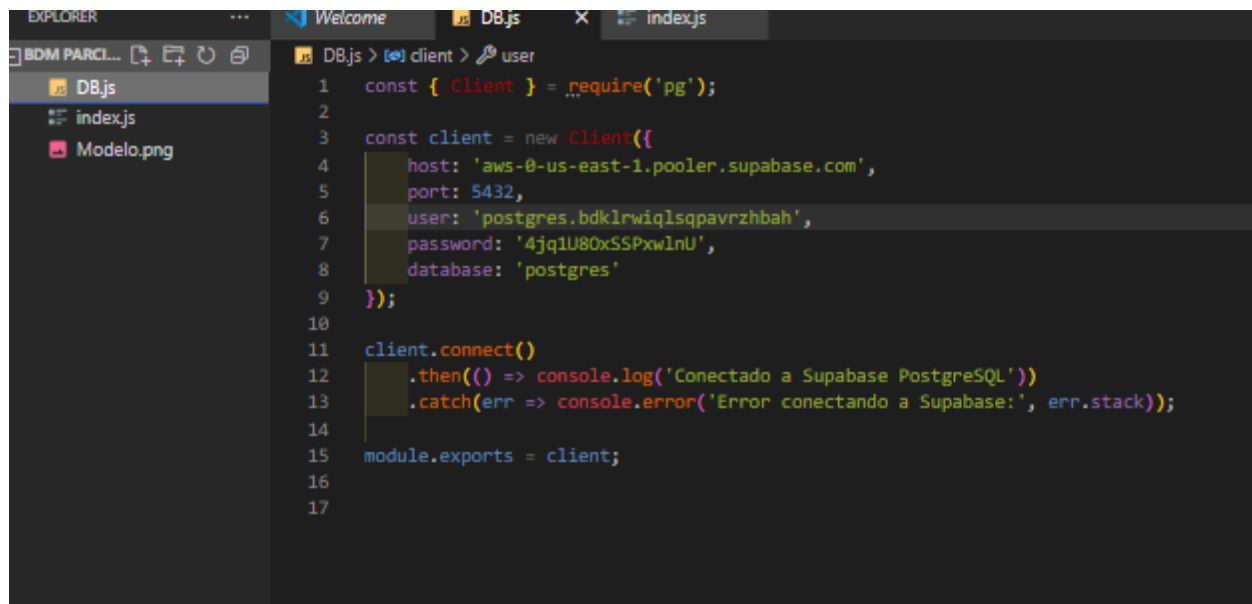
DB.js para la conexión



The screenshot shows the VS Code editor with the Explorer sidebar on the left displaying a project structure with files DB.js, index.js, and Modelo.png. The main editor window shows the content of DB.js, which is a JavaScript file for connecting to a Supabase PostgreSQL database. The code includes the pg module, a Client constructor with connection details (host, port, user, password, database), and a connect method that logs success or error messages. The module is exported as client.

```
1 const { Client } = require('pg');
2
3 const client = new Client({
4   host: 'aws-0-us-east-1.pooler.supabase.com',
5   port: 5432,
6   user: 'postgres.bdklrwiqlsqpavrzhbah',
7   password: '4jq1U80xSSPxwlnU',
8   database: 'postgres'
9 });
10
11 client.connect()
12   .then(() => console.log('Conectado a Supabase PostgreSQL'))
13   .catch(err => console.error('Error conectando a Supabase:', err.stack));
14
15 module.exports = client;
16
17
```

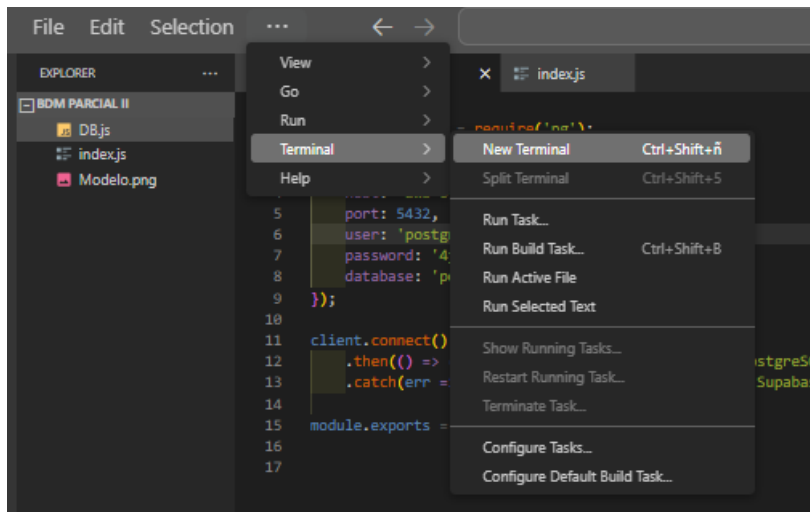
Y index.js para que contenga todas las apis



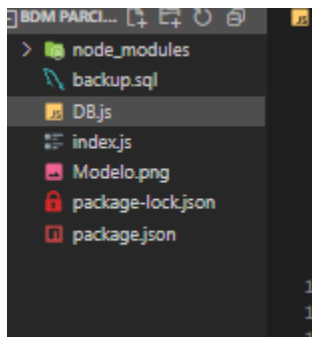
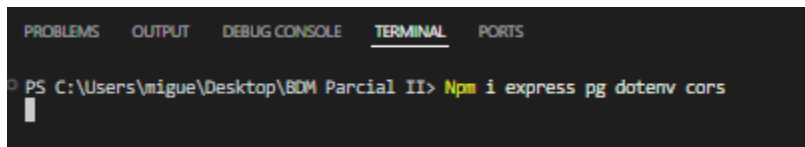
The screenshot shows the VS Code editor with the Explorer sidebar on the left displaying a project structure with files DB.js, index.js, and Modelo.png. The main editor window shows the content of index.js, which is a JavaScript file that exports the client from DB.js. The code is identical to the one in DB.js, including the pg module, Client constructor, connect method, and module export.

```
1 const { Client } = require('pg');
2
3 const client = new Client({
4   host: 'aws-0-us-east-1.pooler.supabase.com',
5   port: 5432,
6   user: 'postgres.bdklrwiqlsqpavrzhbah',
7   password: '4jq1U80xSSPxwlnU',
8   database: 'postgres'
9 });
10
11 client.connect()
12   .then(() => console.log('Conectado a Supabase PostgreSQL'))
13   .catch(err => console.error('Error conectando a Supabase:', err.stack));
14
15 module.exports = client;
16
17
```

Y ahora con new terminal, iniciar todo el proceso para node.js y express



Y se importan las bibliotecas



Y se confirman estas bibliotecas, con los documentos creados

Consultas

Crud tabla Restaurante

Crear Restaurante

```
// CRUD Restaurante
app.post('/api/restaurantes', async (req, res) => {
  const { nombre, ciudad, direccion, fecha_apertura } = req.body;
  const query = 'INSERT INTO restaurante (nombre, ciudad, direccion, fecha_apertura) VALUES ($1, $2, $3, $4)';
  try {
    await client.query(query, [nombre, ciudad, direccion, fecha_apertura]);
    res.status(201).json({ message: 'Restaurante creado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

The screenshot displays a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/api/restaurantes
- Body Type:** raw (selected), with a dropdown menu showing other options like form-data, x-www-form-urlencoded, binary, GraphQL, and JSON.
- Request Body (JSON):**

```
{  "nombre": "Mexican",  "ciudad": "bogota",  "direccion": "Calle 123",  "fecha_apertura": "2032-03-4"}
```
- Response Status:** 201 Created
- Response Body (JSON):**

```
{  "message": "Restaurante creado exitosamente"}
```

Mostar Restaurante

```

app.get('/api/restaurantes', async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM restaurante');
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

GET

Params Authorization Headers (7) **Body** Scripts Settings [Cookies](#)

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☐ JSON [Beautify](#)

1

Body Cookies Headers (8) Test Results [200 OK](#) • 113 ms • 16.94 KB • [Save Response](#)

☒ JSON [Preview](#) [Visualize](#)

```

1  [
2    {
3      "id_rest": 1,
4      "nombre": "Curry-Pearson",
5      "ciudad": "Lake Claire",
6      "direccion": "2592 Johnson Forge Suite 450, Samanthatown, HI 62283",
7      "fecha_apertura": "2018-01-12T05:00:00.000Z"
8    },
9    {
10     "id_rest": 2,
11     "nombre": "Kirby-Williams",
12     "ciudad": "South Shannonbury",

```

Actualizar Restaurante

```

app.put('/api/restaurantes/:id', async (req, res) => {
  const { id } = req.params;
  const { nombre, ciudad, direccion, fecha_apertura } = req.body;
  const query = 'UPDATE restaurante SET nombre = $1, ciudad = $2, direccion = $3, fecha_apertura = $4 WHERE id_rest = $5';
  try {
    await client.query(query, [nombre, ciudad, direccion, fecha_apertura, id]);
    res.status(200).json({ message: 'Restaurante actualizado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```


PUT <http://localhost:3000/api/restaurantes/5> Send

Params Authorization Headers (9) **Body** Scripts Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☒ JSON Beautify

```
1 {
2   "nombre": "Mexican Fiesta",
3   "ciudad": "Medellín",
4   "direccion": "Calle 45 #67-89",
5   "fecha_apertura": "2025-06-01"
6 }
```

Body Cookies Headers (8) Test Results 200 OK • 107 ms • 317 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "message": "Restaurante actualizado exitosamente"
3 }
```

Actualiza con el parámetro del id

Eliminar Restaurante

```
app.delete('/api/restaurantes/:id', async (req, res) => {
  const { id } = req.params;
  try {
    await client.query('DELETE FROM restaurante WHERE id_rest = $1', [id]);
    res.status(200).json({ message: 'Restaurante eliminado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

DELETE <http://localhost:3000/api/restaurantes/89> Send

Params Authorization Headers (7) **Body** Scripts Settings Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body Cookies Headers (8) Test Results 500 Internal Server Error • 154 ms • 420 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "error": "update or delete on table \"restaurante\" violates foreign key constraint \"empleado_id_rest_fkey\"
3   on table \"empleado\""
}
```

Elimina con el parámetro del id, pero hay un error, si se elimina, se dañan las relaciones a la tabla empleado

Table Empleados

Crear Empleado

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/api/empleados`. The request body is a JSON object: `{ "nombre": "Carlos Pérez", "rol": "Cocinero", "id_rest": 1 }`. The response is a 201 Created status with a JSON body: `{ "message": "Empleado creado exitosamente" }`. The response status is 201 Created, with a response time of 105 ms and a body size of 314 B.

```
app.delete('/api/empleados/:id', async (req, res) => {
  const { id } = req.params;
  try {
    await client.query('DELETE FROM empleado WHERE id_empleado = $1', [id]);
    res.status(200).json({ message: 'Empleado eliminado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Mostar Empleado

```
app.get('/api/empleados', async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM empleado');
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Crud Restaurante / Empleado / **obtenerEmpleados** Save Share

GET ▼ Send ▼

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (8) Test Results 🕒 200 OK • 103 ms • 7.51 KB • 🌐 Save Response ...

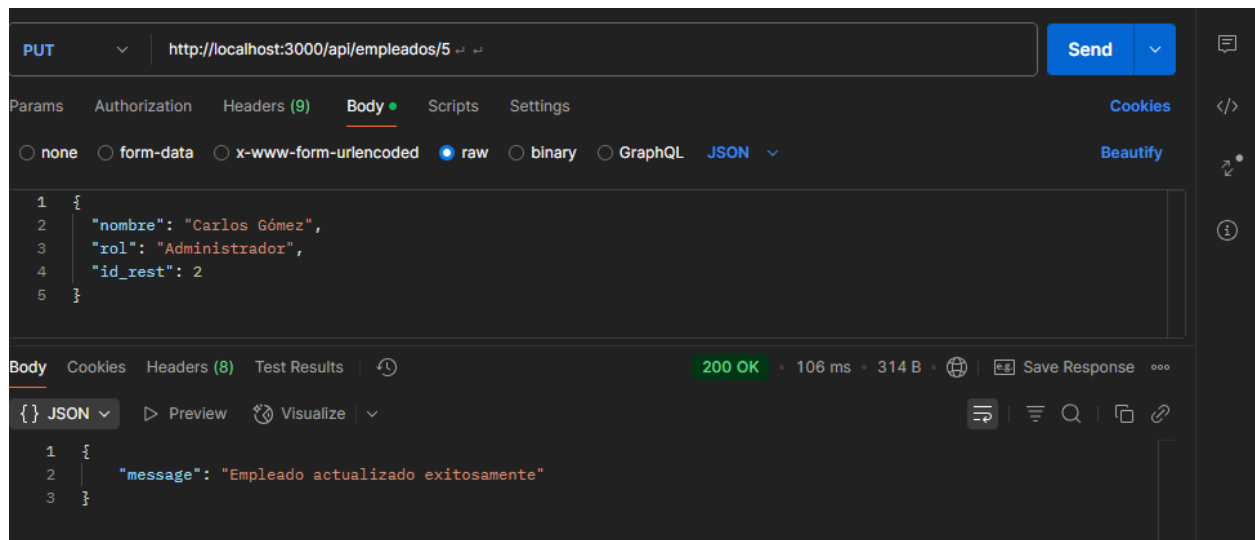
{} JSON ▼ ▶ Preview 🔗 Visualize ▼ ↩ ≡ 🔍 📄 🔗

```
1  [
2    {
3      "id_empleado": 1,
4      "nombre": "Nathan Stevens",
5      "rol": "Cajero",
6      "id_rest": 80
7    },
8    {
9      "id_empleado": 2,
10     "nombre": "Tiffany Hoffman",
11     "rol": "Cajero",
12     "id_rest": 60
13   ]
```

Actualizar Empleado

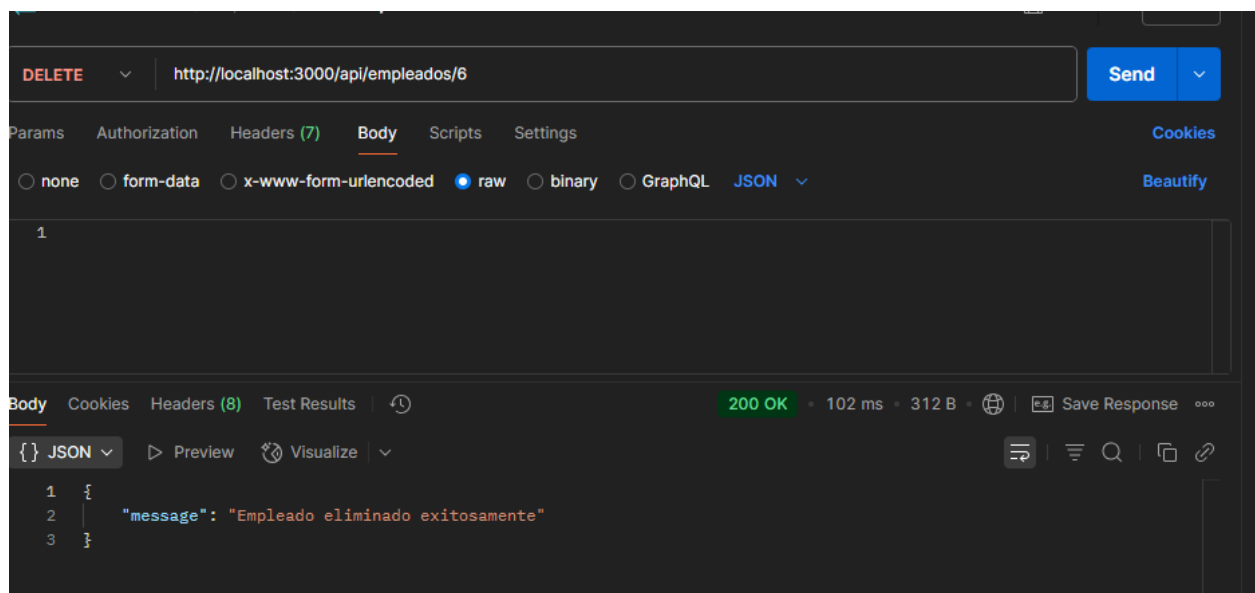
```
app.put('/api/empleados/:id', async (req, res) => {
  const { id } = req.params;
  const { nombre, rol, id_rest } = req.body;
  const query = 'UPDATE empleado SET nombre = $1, rol = $2, id_rest = $3 WHERE id_empleado = $4';
  try {
    await client.query(query, [nombre, rol, id_rest, id]);
    res.status(200).json({ message: 'Empleado actualizado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Actualiza empleado, con el parámetro del id



Eliminar Empleado

```
app.delete('/api/empleados/:id', async (req, res) => {
  const { id } = req.params;
  try {
    await client.query('DELETE FROM empleado WHERE id_empleado = $1', [id]);
    res.status(200).json({ message: 'Empleado eliminado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```



Elimina el empleado, con el parameto del id

Table Productos

Crear Productos

```
app.post('/api/productos', async (req, res) => {
  const { nombre, precio } = req.body;
  const query = 'INSERT INTO producto (nombre, precio) VALUES ($1, $2)';
  try {
    await client.query(query, [nombre, precio]);
    res.status(201).json({ message: 'Producto creado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/api/productos
- Body Type:** raw
- Request Body:**

```
1 {
2   "nombre": "Kebab",
3   "precio": 12.99
4 }
```
- Response Status:** 201 Created
- Response Time:** 110 ms
- Response Size:** 314 B
- Response Body (JSON):**

```
1 {
2   "message": "Producto creado exitosamente"
3 }
```

Mostar Productos

GET http://localhost:3000/api/productos Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (8) Test Results 200 OK 101 ms 6.06 KB Save Response

{} JSON Preview Visualize

```

1  [
2    {
3      "id_prod": 1,
4      "nombre": "Research especial",
5      "precio": "75.67"
6    },
7    {
8      "id_prod": 2,
9      "nombre": "Imagine especial",
10     "precio": "67.19"
11   },
12   ]

```

```

app.get('/api/productos', async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM producto');
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

Actualizar Productos

```

app.put('/api/productos/:id', async (req, res) => {
  const { id } = req.params;
  const { nombre, precio } = req.body;
  const query = 'UPDATE producto SET nombre = $1, precio = $2 WHERE id_prod = $3';
  try {
    await client.query(query, [nombre, precio, id]);
    res.status(200).json({ message: 'Producto actualizado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

PUT ▼ http://localhost:3000/api/productos/7 Send ▼

Params Authorization Headers (9) **Body** • Scripts Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼ Beautify

```
1 {
2   "nombre": "Perro caliente",
3   "precio": 10.5
4 }
```

Body Cookies Headers (8) Test Results ↺ **200 OK** • 117 ms • 314 B • 🌐 📄 Save Response ⋮

{} **JSON** ▼ ▶ Preview 🔗 Visualize ▼ ↺ ☰ 🔍 📄 🔗

```
1 {
2   "message": "Producto actualizado exitosamente"
3 }
```

Modifica el producto, con el parámetro del id

Eliminar Productos

DELETE ▼ http://localhost:3000/api/productos/6 Send ▼

Params Authorization Headers (7) **Body** Scripts Settings Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body Cookies Headers (8) Test Results ↺ **500 Internal Server Error** • 103 ms • 427 B • 🌐 📄 Save Response ⋮

{} **JSON** ▼ ▶ Preview 🔗 Visualize ▼ ↺ ☰ 🔍 📄 🔗

```
1 {
2   "error": "update or delete on table \"producto\" violates foreign key constraint
3     \"detallepedido_id_prod_fkey\" on table \"detallepedido\""
}
```

```

app.delete('/api/productos/:id', async (req, res) => {
  const { id } = req.params;
  try {
    await client.query('DELETE FROM producto WHERE id_prod = $1', [id]);
    res.status(200).json({ message: 'Producto eliminado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

Elimina, con el parámetro del id

Table Pedidos

Obtener Pedidos

```

app.get('/api/pedidos', async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM pedido');
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

Obtener Pedido

GET http://localhost:3000/api/pedidos

Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results 200 OK 109 ms 8.33 KB Save Response

JSON Preview Visualize

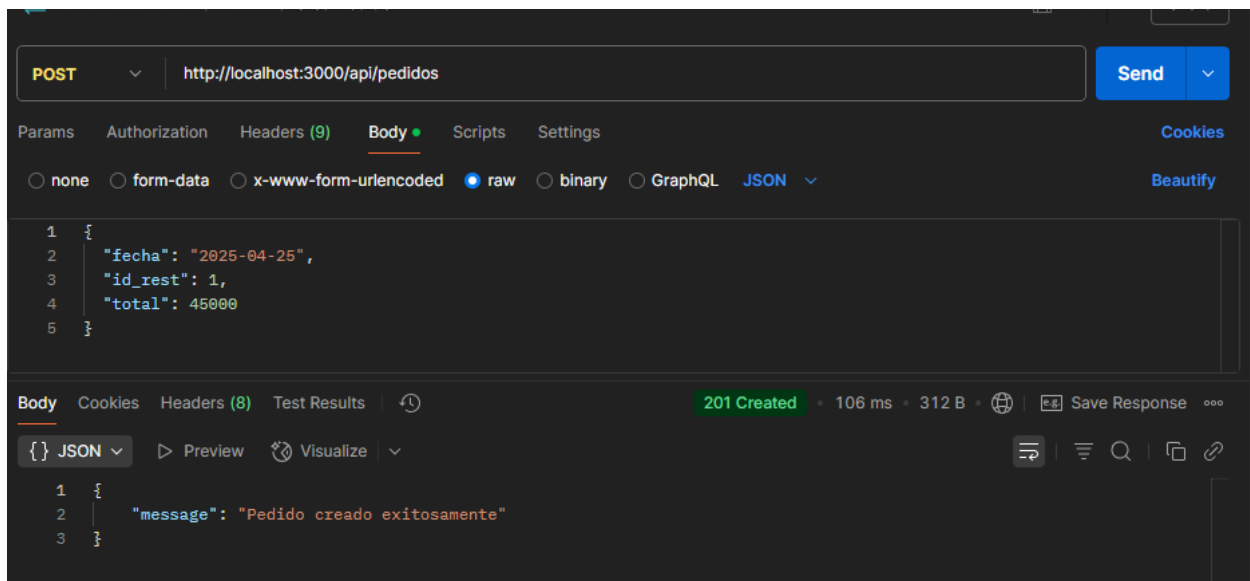
```

1 [
2   {
3     "id_pedido": 1,
4     "id_rest": 64,
5     "fecha": "2023-07-04T05:00:00.000Z",
6     "total": "274.71"
7   },
8   {
9     "id_pedido": 2,
10    "id_rest": 56,
11    "fecha": "2024-01-06T05:00:00.000Z",
12    "total": "191.75"

```


Crear Pedidos

```
app.post('/api/pedidos', async (req, res) => {
  const { fecha, id_rest, total } = req.body;
  const query = 'INSERT INTO pedido (fecha, id_rest, total) VALUES ($1, $2, $3)';
  try {
    await client.query(query, [fecha, id_rest, total]);
    res.status(201).json({ message: 'Pedido creado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```



The screenshot shows a REST client interface with the following details:

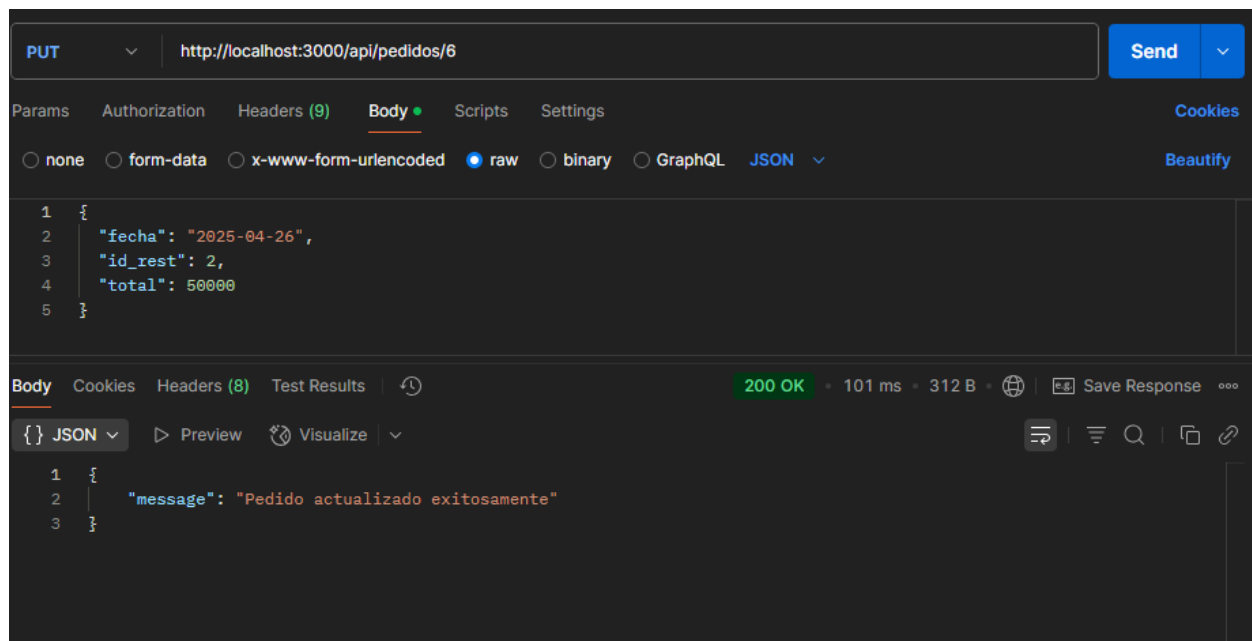
- Method:** POST
- URL:** http://localhost:3000/api/pedidos
- Body Type:** raw
- Request Body (JSON):**

```
{
  "fecha": "2025-04-25",
  "id_rest": 1,
  "total": 45000
}
```
- Response:** 201 Created, 106 ms, 312 B
- Response Body (JSON):**

```
{
  "message": "Pedido creado exitosamente"
}
```

Actualizar

```
app.put('/api/pedidos/:id', async (req, res) => {
  const { id } = req.params;
  const { fecha, id_rest, total } = req.body;
  const query = 'UPDATE pedido SET fecha = $1, id_rest = $2, total = $3 WHERE id_pedido = $4';
  try {
    await client.query(query, [fecha, id_rest, total, id]);
    res.status(200).json({ message: 'Pedido actualizado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```



Actualiza el pedido, con el parámetro del id

Eliminar

```
app.delete('/api/pedidos/:id', async (req, res) => {
  const { id } = req.params;
  try {
    await client.query('DELETE FROM pedido WHERE id_pedido = $1', [id]);
    res.status(200).json({ message: 'Pedido eliminado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

DELETE ▼ http://localhost:3000/api/pedidos/89 Send ▼

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results ↺ 500 Internal Server Error • 102 ms • 427 B 🌐 📄 Save Response ⋮

{ } JSON ▼ ▶ Preview 🔗 Visualize ▼ 🔍 ☰ 🔗

```
1 {
2   "error": "update or delete on table \"pedido\" violates foreign key constraint
3   \"detallepedido_id_pedido_fkey\" on table \"detallepedido\""
```

Elimina el pedido, con el parámetro del id

Table DetallesPedido

Crear DetallePedido

```
app.post('/api/detallepedidos', async (req, res) => {
  const { id_pedido, id_prod, cantidad } = req.body;
  const query = 'INSERT INTO detallepedido (id_pedido, id_prod, cantidad) VALUES ($1, $2, $3)';
  try {
    await client.query(query, [id_pedido, id_prod, cantidad]);
    res.status(201).json({ message: 'DetallePedido creado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

POST http://localhost:3000/api/detallepedidos

Params Authorization Headers (9) Body Scripts Settings Cookies

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

```
1 {
2   "id_pedido": 8,
3   "id_prod": 3,
4   "cantidad": 450
5 }
```

Body Cookies Headers (8) Test Results 201 Created 103 ms 319 B Save Response

{ JSON Preview Visualize

```
1 {
2   "message": "DetallePedido creado exitosamente"
3 }
```

Mostar Detalles

GET http://localhost:3000/api/detallepedidos

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results 200 OK 109 ms 23.32 KB Save Response

{ JSON Preview Visualize

```
1 [
2   {
3     "id_detalle": 1,
4     "id_pedido": 1,
5     "id_prod": 50,
6     "cantidad": 1,
7     "subtotal": "10.87"
8   },
9   {
10    "id_detalle": 2,
11    "id_pedido": 1,
12    "id_prod": 35,
```

```
app.get('/api/detallepedidos', async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM detallepedido');
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Actualizar Detalle

```
app.put('/api/detallepedidos/:id', async (req, res) => {
  const { id } = req.params;
  const { id_pedido, id_prod, cantidad } = req.body;
  const query = 'UPDATE detallepedido SET id_pedido = $1, id_prod = $2, cantidad = $3 WHERE id_detalle = $4';
  try {
    await client.query(query, [id_pedido, id_prod, cantidad, id]);
    res.status(200).json({ message: 'DetallePedido actualizado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/api/detallepedidos/5`. The request body is a JSON object: `{ "id_pedido": 1, "id_prod": 3, "cantidad": 5 }`. The response is a 200 OK status with a JSON body: `{ "message": "DetallePedido actualizado exitosamente" }`.

Request Details:

- Method: PUT
- URL: `http://localhost:3000/api/detallepedidos/5`
- Body Type: raw
- Body Content:


```
1 {
2   "id_pedido": 1,
3   "id_prod": 3,
4   "cantidad": 5
5 }
```

Response Details:

- Status: 200 OK
- Time: 102 ms
- Size: 319 B
- Body Type: JSON
- Body Content:


```
1 {
2   "message": "DetallePedido actualizado exitosamente"
3 }
```

Actualiza detalles, con el parámetro del id

Eliminar Detalle

```
app.delete('/api/detallepedidos/:id', async (req, res) => {
  const { id } = req.params;
  try {
    await client.query('DELETE FROM detallepedido WHERE id_detalle = $1', [id]);
    res.status(200).json({ message: 'DetallePedido eliminado exitosamente' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

DELETE ⌵ http://localhost:3000/api/detallepedidos/67 Send ⌵

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body Cookies Headers (8) Test Results 🔄 200 OK • 103 ms • 317 B • 🌐 💾 Save Response ⋮

{} JSON ▶ Preview 🔗 Visualize ⌵

```
1 {
2   "message": "DetallePedido eliminado exitosamente"
3 }
```

Elimina detalles, con el parámetro del id

Consultas Nativas

Productos de un pedido específico

```
// 1. Productos de un pedido específico
app.get('/api/productos-pedido/:id_pedido', async (req, res) => {
  const { id_pedido } = req.params;
  const result = await client.query(`
    SELECT p.nombre, dp.cantidad, dp.subtotal
    FROM DetallePedido dp
    JOIN Producto p ON dp.id_prod = p.id_prod
    WHERE dp.id_pedido = $1
  `, [id_pedido]);
  res.json(result.rows);
});
```

GET http://localhost:3000/api/productos-pedido/28 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results 200 OK 101 ms 394 B Save Response

{} JSON Preview Visualize

```

1  [
2    {
3      "nombre": "Mission especial",
4      "cantidad": 2,
5      "subtotal": "36.90"
6    },
7    {
8      "nombre": "Brother especial",
9      "cantidad": 4,
10     "subtotal": "267.08"
11   }
12 ]

```

Retorna todos los productos relacionados a un pedido específico

Productos más vendidos

```

// 2. Productos más vendidos
app.get('/api/productos-mas-vendidos/:cantidad_min', async (req, res) => {
  const { cantidad_min } = req.params;
  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
  `, [cantidad_min]);
  res.json(result.rows);
});

```

GET http://localhost:3000/api/productos-mas-vendidos/10 Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (8) Test Results 200 OK 106 ms 1.58 KB Save Response

{} JSON Preview Visualize

```

1  [
2    {
3      "nombre": "Such especial",
4      "total_vendido": "11"
5    },
6    {
7      "nombre": "College especial",
8      "total_vendido": "16"
9    },
10   {
11     "nombre": "What especial",
12     "total_vendido": "36"

```

Muestra los productos, cuya cantidad total vendida supera el valor enviado en el parametro

Ventas Por Restaurante

```

// 3. Total de ventas por restaurante
app.get('/api/ventas-por-restaurante', async (req, res) => {
  const result = await client.query(`
    SELECT r.nombre, 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);
});

```


HTTP Crud Restaurante / ConsultasNativas / **Ventas PorRestaurante** [Share](#)

GET <http://localhost:3000/api/ventas-por-restaurante> **Send**

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results **200 OK** • 104 ms • 3.69 KB • Save Response

JSON Preview Visualize

```
1  [
2    {
3      "nombre": "Wolfe PLC",
4      "total_ventas": "182.80"
5    },
6    {
7      "nombre": "Morrison and Sons",
8      "total_ventas": "667.46"
9    },
10   {
11     "nombre": "Thomas-May",
12     "total_ventas": "447.92"
```

Retorna el total de las ventas, por restaurante, sumando todos los totales de los pedidos

Pedido Por Fecha

Crud Restaurante / ConsultasNativas / PedidosPorFecha

GET

http://localhost:3000/api/pedidos-por-fecha/2024-07-16

Send

Params

Authorization

Headers (7)

Body

Scripts

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (8)

Test Results

200 OK • 99 ms • 350 B • Save Response

{ } JSON

Preview

Visualize

```
1  [  
2    {  
3      "id_pedido": 99,  
4      "id_rest": 93,  
5      "fecha": "2024-07-16T05:00:00.000Z",  
6      "total": "140.67"  
7    }  
8  ]
```

Crud Restaurante / ConsultasNativas / PedidosPorFecha

GET

http://localhost:3000/api/pedidos-por-fecha/2024-07-16

Send

Params

Authorization

Headers (7)

Body

Scripts

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (8)

Test Results

200 OK • 99 ms • 350 B • Save Response

{ } JSON

Preview

Visualize

```
1  [  
2    {  
3      "id_pedido": 99,  
4      "id_rest": 93,  
5      "fecha": "2024-07-16T05:00:00.000Z",  
6      "total": "140.67"  
7    }  
8  ]
```

Devuelve todos los pedidos realizados en una fecha establecida, enviada como parámetro(aaaa-mm-dd)

Empleado Por Rol, En Restaurante

```
// 5. Empleados por rol en un restaurante
app.get('/api/empleados-por-rol', async (req, res) => {
  const { nombre_rol, id_rest } = req.query;

  const query = `
    SELECT *
    FROM empleado
    WHERE rol = $1 AND id_rest = $2
  `;

  try {
    const result = await client.query(query, [nombre_rol, id_rest]);
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

GET http://localhost:3000/api/empleados-por-rol?nombre_rol=Mesero&id_rest=1 Send

Params • Authorization Headers (7) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	⋮ Bulk Edit
<input checked="" type="checkbox"/>	nombre_rol	Mesero		
<input checked="" type="checkbox"/>	id_rest	1		
	Key	Value	Description	

Body Cookies Headers (8) Test Results | 🔄

200 OK • 147 ms • 339 B | 🌐 | 📄 Save Response ⋮

{ } JSON ▾ ▶ Preview 🔍 Visualize ▾

```
1 [
2   {
3     "id_empleado": 81,
4     "nombre": "Melanie Brown",
5     "rol": "Mesero",
6     "id_rest": 1
7   }
8 ]
```

Devuelve los empleados que cumplen un rol específico en un restaurante determinado. Tiene los parámetros de nombre_rol, que es el rol que vamos a llamar, y id_rest, que es el id del restaurante.