

Parcial Bases de Datos Api

Edwin Camilo Rodriguez Arredondo

ID:841424

Bases de Datos Masiva

6° Semestre Ingeniería de Sistemas

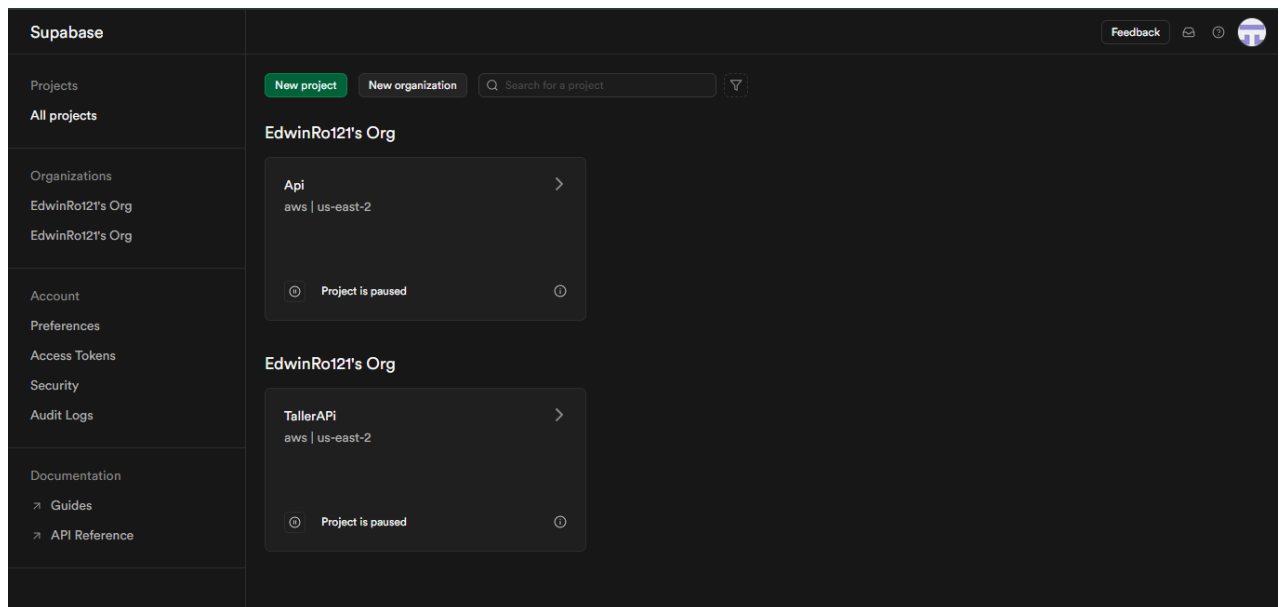
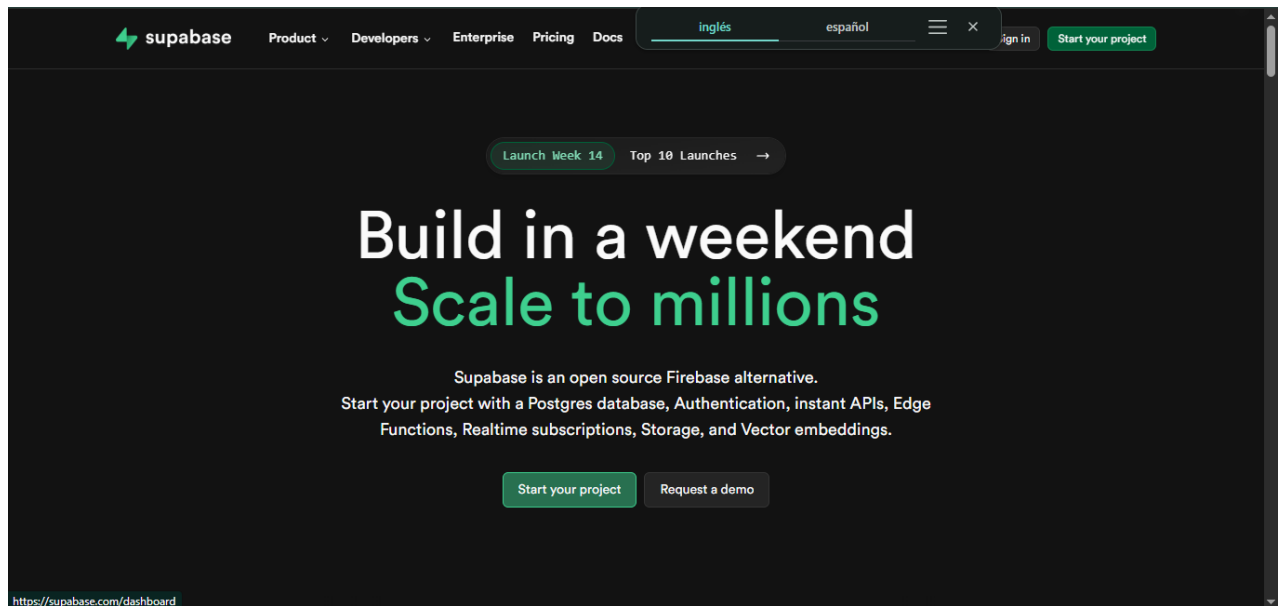
Ingeniería de Sistemas

Universidad UNIMINUTO

23 de abr. de 25

Zipaquirá Cundinamarca

- Se ingresa a supabase y se crar un nuevo proyecto



- Se asigna el nombre y contraseña

Your project will have its own dedicated instance and full Postgres database.
An API will be set up so you can easily interact with your new database.

Organization: EdwinRo121's Org Free

Project name: ParcialBD

Database Password: [masked] [Copy]

Not bad, but your password must be harder to guess. [Generate a password](#)

Region: East US (Ohio)

Select the region closest to your users for the best performance.

SECURITY OPTIONS >

ADVANCED CONFIGURATION >

Cancel Create new project

- Se crean las tablas con sus respectivos parámetros y sus respectivas relaciones

```
1 CREATE TABLE Restaurante (id_rest int PRIMARY KEY,nombre varchar(100),ciudad varchar(100),direccion varchar(150),fecha_apertura DATE);
```

Results Chart Export

Success. No rows returned

```
1 create table empleado (id_empleado int primary key,nombre varchar(100),rol varchar(50),id_rest int,foreign key (id_rest) references restaurante(id_rest));
```

Results Chart Export

Success. No rows returned

```
1 create table producto (id_prod int primary key,nombre varchar(100),precio numeric(10, 2));
```

Results Chart Export

Success. No rows returned

```
1 create table pedido (id_pedido int primary key, fecha date, id_rest int, total numeric(10, 2), foreign key (id_rest) references restaurante (id_rest));
```

Results Chart Export

Source Primary Database Role postgres Run CTRL ↵

Success. No rows returned

```
1 create table detalle_pedido (id_detalle int primary key, id_pedido int, id_prod int, cantidad int, subtotal numeric(10, 2), foreign key (id_pedido) references pedido(id_pedido), foreign key (id_prod) references producto(id_prod));
```

Results Chart Export

Source Primary Database Role postgres Run CTRL ↵

Success. No rows returned

- Se miran los parámetros de la conexión

Session pooler Shared Pooler

Only recommended as an alternative to Direct Connection, when connecting via an IPv4 network.

```
postgresql://postgres.ntijxivkafeylaqisqg:[YOUR-PASSWORD]@
```

View parameters

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

port: 5432

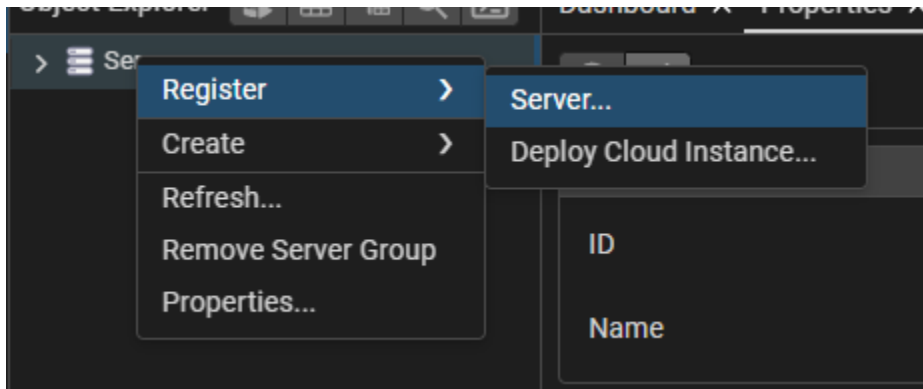
database: postgres

user: postgres.ntijxivkafeylaqisqg

pool_mode: session

For security reasons, your database password is never shown.

- Se crea un nuevo servidor en pgadmin4

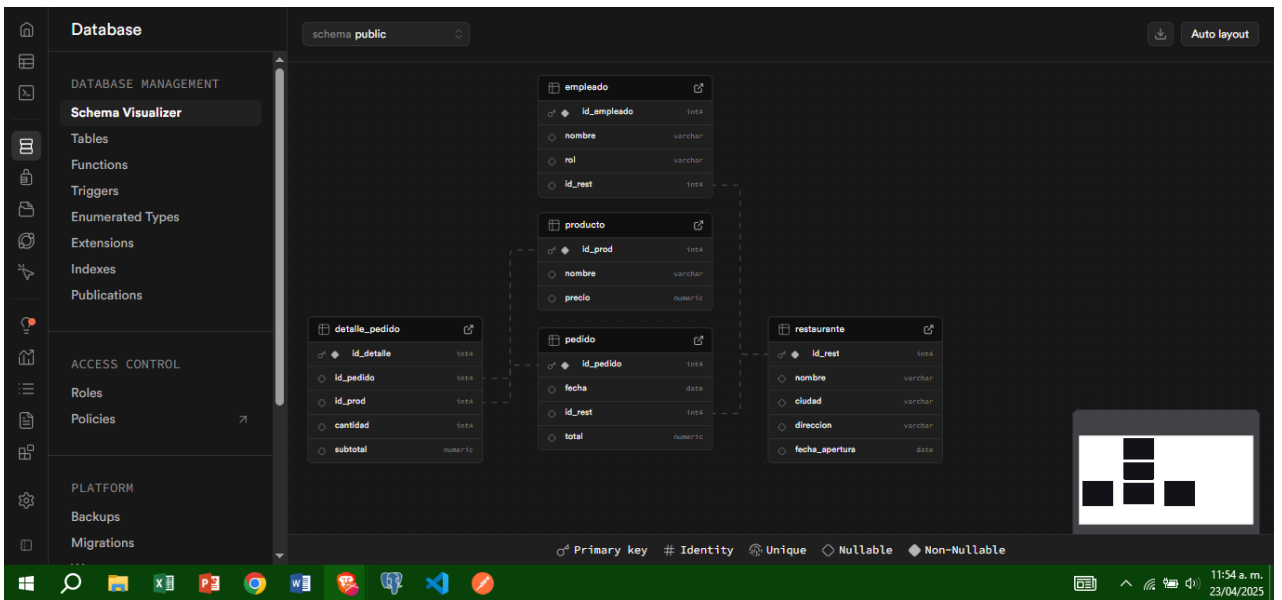
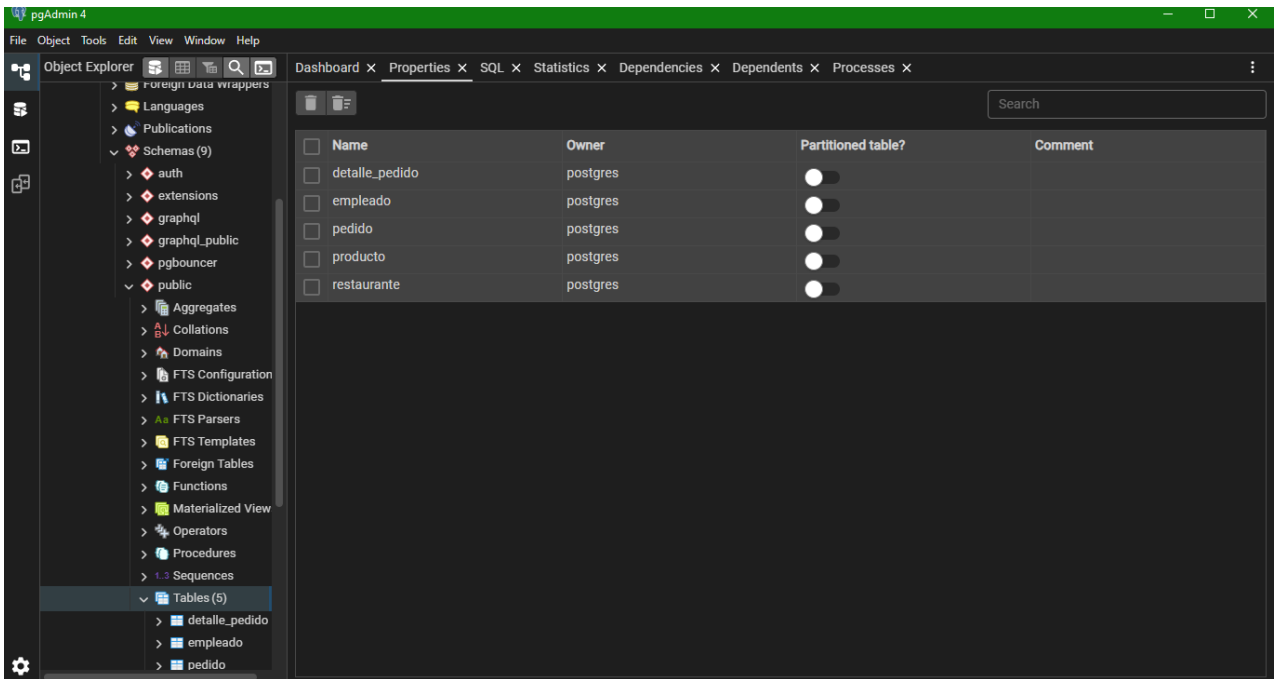


- Se insertan los parámetros de conexión de supabase

A screenshot of the 'Register - Server' dialog box in pgAdmin 4. The 'Connection' tab is selected. The form contains the following fields and controls:

- Host name/address: aws-0-us-east-2.pooler.supabase.com
- Port: 5432
- Maintenance database: postgres
- Username: postgres.ntijxivkafeylaqisqg
- Kerberos authentication?: ☐
- Password:
- Save password?: ☐
- Role:
- Service:

At the bottom, there is a red error message: 'Name' cannot be empty. The bottom bar contains buttons for 'Close', 'Reset', and 'Save'.



- Se insertan los 50 registros de restaurante

```
1 insert into restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) values
2 (1, 'Don Julio', 'Barranquilla', 'Calle 1 #11-1', '2020-01-16'),
3 (2, 'Rico Más Pan', 'Neiva', 'Calle 2 #12-2', '2020-01-31'),
4 (3, 'La Pizzería', 'Cali', 'Calle 3 #13-3', '2020-02-15'),
5 (4, 'Sushi Ya', 'Cartagena', 'Calle 4 #14-4', '2020-03-01'),
6 (5, 'El Corralito', 'Medellín', 'Calle 5 #15-5', '2020-03-16'),
7 (6, 'Arepá Loca', 'Barranquilla', 'Calle 6 #16-6', '2020-03-31'),
8 (7, 'La Hamburguesería', 'Pereira', 'Calle 7 #17-7', '2020-04-15'),
9 (8, 'Taco Town', 'Ibagué', 'Calle 8 #18-8', '2020-04-30'),
10 (9, 'Pollo Riko', 'Barranquilla', 'Calle 9 #19-9', '2020-05-15'),
11 (10, 'Las Delicias', 'Neiva', 'Calle 10 #20-10', '2020-05-30'),
12 (11, 'Kokoriko', 'Medellín', 'Calle 11 #21-11', '2020-06-14'),
13 (12, 'El Buen Sabor', 'Ibagué', 'Calle 12 #22-12', '2020-06-29'),
14 (13, 'Pasta Express', 'Neiva', 'Calle 13 #23-13', '2020-07-14'),
15 (14, 'Burger Station', 'Cali', 'Calle 14 #24-14', '2020-07-29'),
16 (15, 'La Parrilla', 'Bucaramanga', 'Calle 15 #25-15', '2020-08-13'),
17 (16, 'Sabores de Casa', 'Cali', 'Calle 16 #26-16', '2020-08-28'),
18 (17, 'Mr. Wings', 'Ibagué', 'Calle 17 #27-17', '2020-09-12'),
```

Results Chart Export ▾

Success. No rows returned

- Se insertan los 50 de empleado

```
1 insert into empleado (id_empleado, nombre, rol, id_rest) values
2 (1, 'Daniela Ruiz', 'Repartidor', 1),
3 (2, 'Carlos Gómez', 'Repartidor', 2),
4 (3, 'Diana Ruiz', 'Cajero', 3),
5 (4, 'Diana Hernández', 'Repartidor', 4),
6 (5, 'Camila Martínez', 'Mesero', 5),
7 (6, 'Paula Ruiz', 'Cajero', 6),
8 (7, 'Juan Ruiz', 'Cajero', 7),
9 (8, 'Miguel Rodríguez', 'Repartidor', 8),
10 (9, 'Camila Hernández', 'Cajero', 9),
11 (10, 'Sebastián Moreno', 'Repartidor', 10),
12 (11, 'Carlos Gómez', 'Mesero', 11),
13 (12, 'Sara Gómez', 'Cocinero', 12),
14 (13, 'Julián Gómez', 'Cocinero', 13),
15 (14, 'Daniela García', 'Repartidor', 14),
16 (15, 'Sebastián Martínez', 'Cocinero', 15),
17 (16, 'Andrés López', 'Administrador', 16),
18 (17, 'Laura Díaz', 'Repartidor', 17),
```

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

Success. No rows returned

- Se insertan los 50 registros de producto

```
1 insert into producto (id_prod, nombre, precio) values
2 (1, 'Hamburguesa Clásica', 28233.64),
3 (2, 'Pizza Hawaiana', 26450.07),
4 (3, 'Pollo a la Broaster', 23866.23),
5 (4, 'Arepa con Queso', 7130.14),
6 (5, 'Empanada de Carne', 10206.69),
7 (6, 'Taco Mexicano', 5513.11),
8 (7, 'Sushi Roll', 20105.84),
9 (8, 'Salchipapa', 19460.06),
10 (9, 'Chorizo Santarrosano', 17498.03),
11 (10, 'Patacón con Todo', 27988.22),
12 (11, 'Sándwich de Jamón', 21085.36),
13 (12, 'Wok de Pollo', 22048.94),
14 (13, 'Chuzo de Res', 22328.36),
15 (14, 'Cazuela de Frijoles', 10437.93),
16 (15, 'Arroz con Pollo', 23060.63),
17 (16, 'Pasta Alfredo', 5109.98),
18 (17, 'Pizza de Pepperoni', 24059.67),
```

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

Success. No rows returned

- Se insertan 50 registros de pedido

```
1 insert into pedido (id_pedido, fecha, id_rest, total) values
2 (1, '2024-01-02', 1, 99678.82),
3 (2, '2024-01-03', 2, 42115.94),
4 (3, '2024-01-04', 3, 78506.14),
5 (4, '2024-01-05', 4, 93812.48),
6 (5, '2024-01-06', 5, 17103.94),
7 (6, '2024-01-07', 6, 94889.93),
8 (7, '2024-01-08', 7, 63589.38),
9 (8, '2024-01-09', 8, 73185.47),
10 (9, '2024-01-10', 9, 88518.34),
11 (10, '2024-01-11', 10, 79408.92),
12 (11, '2024-01-12', 11, 96687.63),
13 (12, '2024-01-13', 12, 10075.58),
14 (13, '2024-01-14', 13, 36913.35),
15 (14, '2024-01-15', 14, 61851.49),
16 (15, '2024-01-16', 15, 70553.22),
17 (16, '2024-01-17', 16, 78597.93),
18 (17, '2024-01-18', 17, 91767.45),
```

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

Success. No rows returned

- Se insertan 50 registros de detalle_pedido

```
1 insert into detalle_pedido (id_detalle, id_pedido, id_prod, cantidad, subtotal) values
2 (1, 1, 50, 3, 34523.07),
3 (2, 2, 32, 4, 118402.52),
4 (3, 3, 10, 1, 19998.98),
5 (4, 4, 44, 1, 29542.45),
6 (5, 5, 38, 5, 91730.25),
7 (6, 6, 30, 4, 108769.48),
8 (7, 7, 42, 2, 36092.38),
9 (8, 8, 50, 4, 60257.64),
10 (9, 9, 27, 1, 7090.82),
11 (10, 10, 27, 1, 5924.34),
12 (11, 11, 23, 1, 7933.21),
13 (12, 12, 29, 1, 9905.9),
14 (13, 13, 39, 3, 82922.1),
15 (14, 14, 37, 3, 61561.23),
16 (15, 15, 12, 2, 49216.96),
17 (16, 16, 24, 1, 13784.02),
18 (17, 17, 27, 4, 69934.64),
```

Results Chart Export

Success. No rows returned

- Se abre el visual studio y se asigna una carpeta después se inicia un repositorio

```
PS C:\Users\USUARIO\Desktop\ParcialBD> npm init -y
Wrote to C:\Users\USUARIO\Desktop\ParcialBD\package.json:

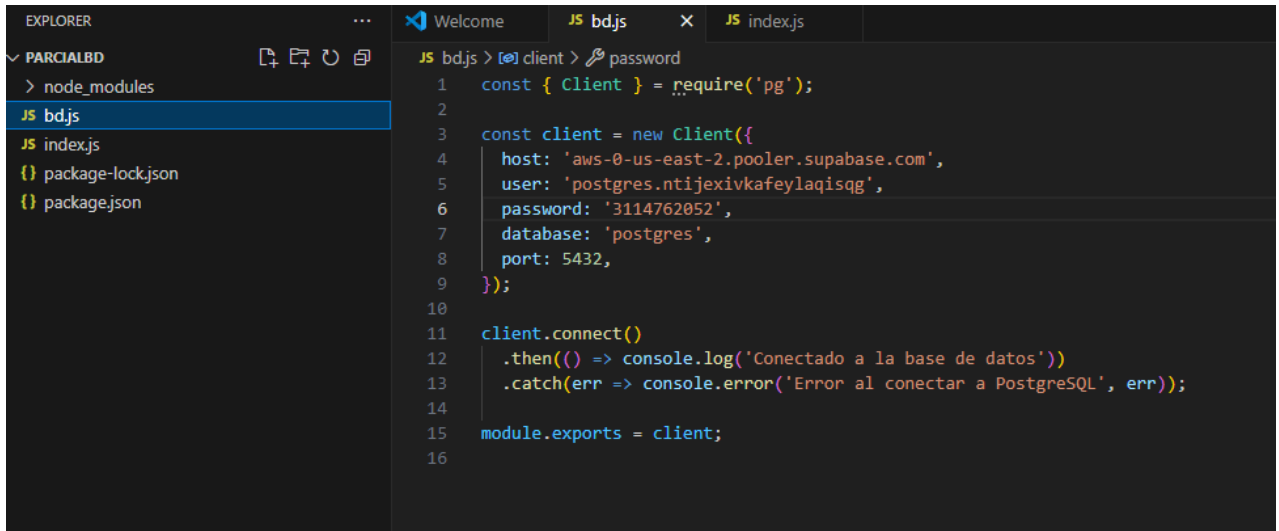
{
  "name": "parcialbd",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

- Se instalan las respectivas las respectivas herramientas que usaremos

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\USUARIO\Desktop\ParcialBD> npm install express cors pg dotenv
```

- Se crea un nuevo archivo y se crea la conexión con supabase



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'PARCIALBD' with a 'node_modules' directory and two JavaScript files: 'bd.js' and 'index.js'. The 'bd.js' file is selected and its content is displayed in the code editor. The code defines a Supabase client and connects it to a PostgreSQL database.

```
1 const { Client } = require('pg');
2
3 const client = new Client({
4   host: 'aws-0-us-east-2.pooler.supabase.com',
5   user: 'postgres.ntijexivkafeylaqisqg',
6   password: '3114762052',
7   database: 'postgres',
8   port: 5432,
9 });
10
11 client.connect()
12   .then(() => console.log('Conectado a la base de datos'))
13   .catch(err => console.error('Error al conectar a PostgreSQL', err));
14
15 module.exports = client;
16
```

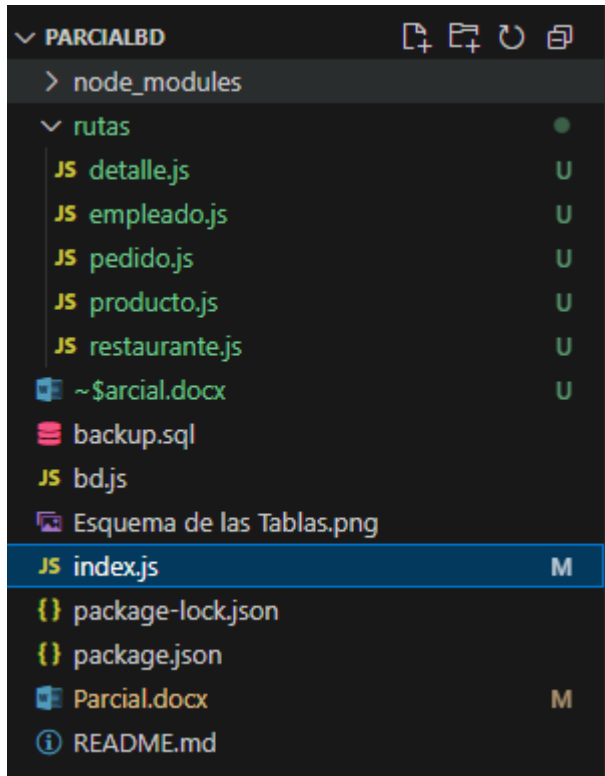
- Se crea otro archivo llamado index.js y se crea las conexiones de cada tabla con sus respectivas apis



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'PARCIALBD' with a 'node_modules' directory and two JavaScript files: 'bd.js' and 'index.js'. The 'index.js' file is selected and its content is displayed in the code editor. The code sets up an Express.js server, configures middleware, and defines API routes for a restaurant management system.

```
1 const express = require('express');
2 const cors = require('cors');
3 const app = express();
4 const PORT = 3000;
5
6 app.use(cors());
7 app.use(express.json());
8 app.use(express.urlencoded({ extended: true }));
9
10 // Conexión a rutas
11 app.use('/api', require('./rutas/restaurante'));
12 app.use('/api', require('./rutas/empleado'));
13 app.use('/api', require('./rutas/producto'));
14 app.use('/api', require('./rutas/pedido'));
15 app.use('/api', require('./rutas/detalle'));
16
17 // Ruta de prueba
18 app.get('/api/prueba', (req, res) => {
19   res.send('API funcionando correctamente 🚀');
20 });
21
22 app.listen(PORT, () => {
23   console.log(`Servidor corriendo en http://localhost:${PORT}`);
24 });
25
```

- Se crea un nuevo archivo para cada tabla



- Se crea el api de insertar

```
// Crear restaurante
router.post('/insertar/restaurante', (req, res) => {
  const { id_rest, nombre, ciudad, direccion, fecha_apertura } = req.body;
  const query = 'INSERT INTO restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES ($1, $2, $3, $4, $5)';
  client.query(query, [id_rest, nombre, ciudad, direccion, fecha_apertura])
    .then(() => res.status(201).json({ mensaje: "Restaurante creado con éxito" }))
    .catch(error => res.status(500).json({ error: error.message }));
});
```

- app.post() crea una ruta POST.
- req.body obtiene los datos enviados por el cliente (en JSON).
- Se arma una consulta SQL con INSERT INTO para guardar el restaurante.
- Se usa client.query() (desde tu conexión a PostgreSQL) para ejecutar la consulta.
- Si todo va bien, responde con 201 (Created).
- Si falla, responde con 500 (Internal Server Error).
- GET para consultar todos los restaurantes.

- Ejecuta `SELECT * FROM restaurante` para obtener todos los registros.
- `result.rows` es un array con todos los resultados.
- Responde con un objeto JSON que contiene la información
- Se crea la api de actualizar

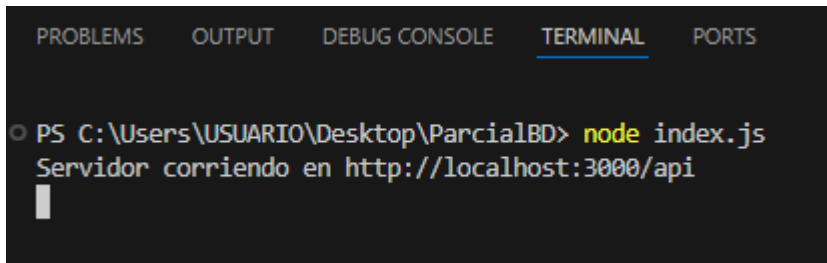
```
// Actualizar restaurante
app.put('/api/actualizar/restaurante/:id', (req, res) => {
  const { id } = req.params;
  const { nombre, ciudad, direccion, fecha_apertura } = req.body;
  const query = 'UPDATE restaurante SET nombre=$1, ciudad=$2, direccion=$3, fecha_apertura=$4 WHERE id_restaurante=$5';
  client.query(query, [nombre, ciudad, direccion, fecha_apertura, id])
    .then(result => {
      if (result.rowCount === 0) {
        res.status(404).json({ mensaje: "Restaurante no encontrado" });
      } else {
        res.status(200).json({ mensaje: "Restaurante actualizado con éxito" });
      }
    })
    .catch(error => {
      res.status(500).json({ error: error.message });
    });
});
```

- Ruta PUT para actualizar un restaurante.
- Usa `:id` como parámetro en la URL `req.params.id`
- Actualiza los campos especificados en el body.
- `rowCount` verifica si se actualizó al menos un registro. Si no, muestra 404.

- se crea la api de eliminar

```
// Eliminar restaurante
app.delete('/api/eliminar/restaurante/:id', (req, res) => {
  const { id } = req.params;
  client.query('DELETE FROM restaurante WHERE id_rest = $1', [id])
    .then(result => {
      if (result.rowCount === 0) {
        res.status(404).json({ mensaje: "Restaurante no encontrado" });
      } else {
        res.status(200).json({ mensaje: "Restaurante eliminado con éxito" });
      }
    })
    .catch(error => {
      res.status(500).json({ error: error.message });
    });
});
```

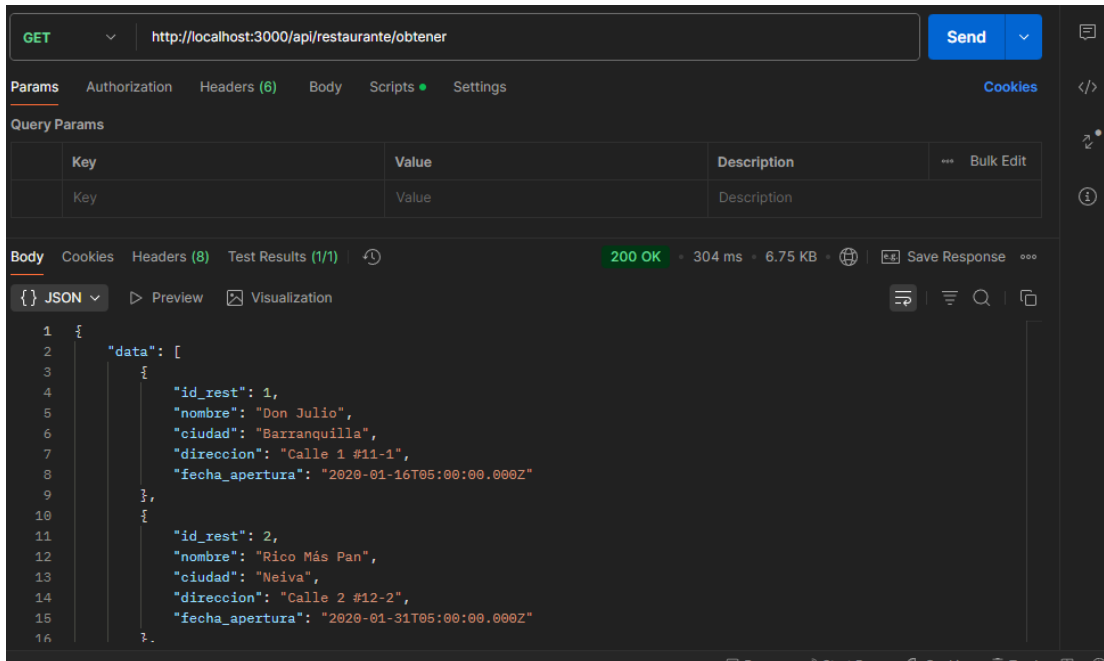
- Ruta DELETE para eliminar un restaurante por ID.
 - Ejecuta DELETE FROM restaurante WHERE id_rest = \$1 (\$1 significa el dato que se le ingrese por ejemplo 4 , Ejecuta DELETE FROM restaurante WHERE id_rest = 4)
 - Si no encuentra el restaurante, rowCount es 0, así que responde con 404
-
- Se corre el servidor



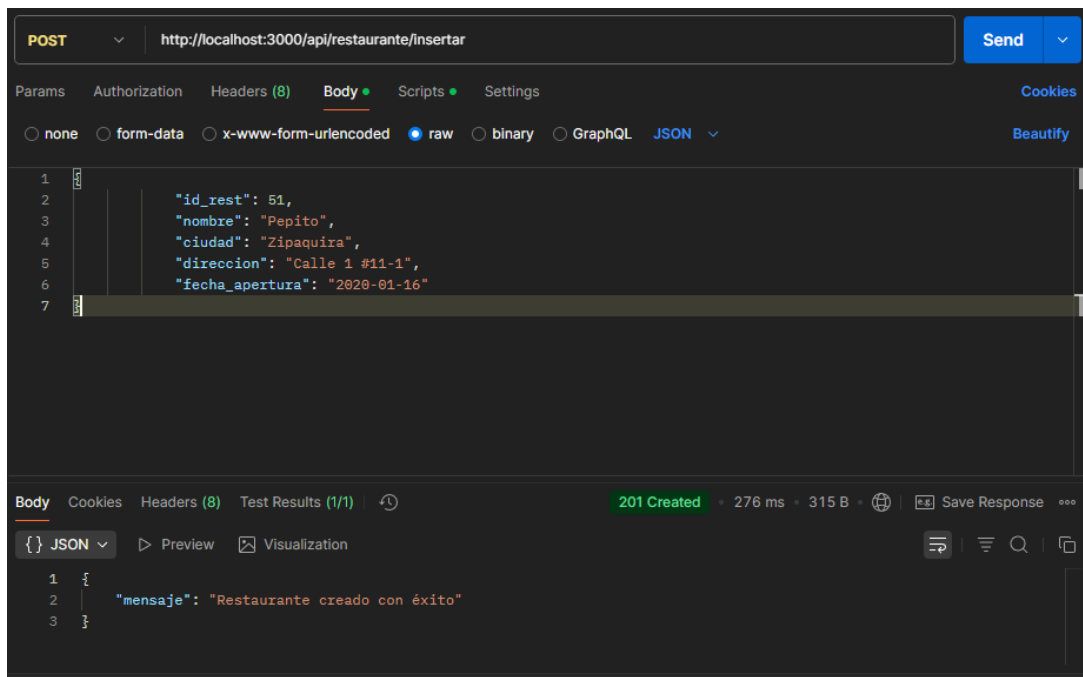
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\USUARIO\Desktop\ParcialBD> node index.js
Servidor corriendo en http://localhost:3000/api
```

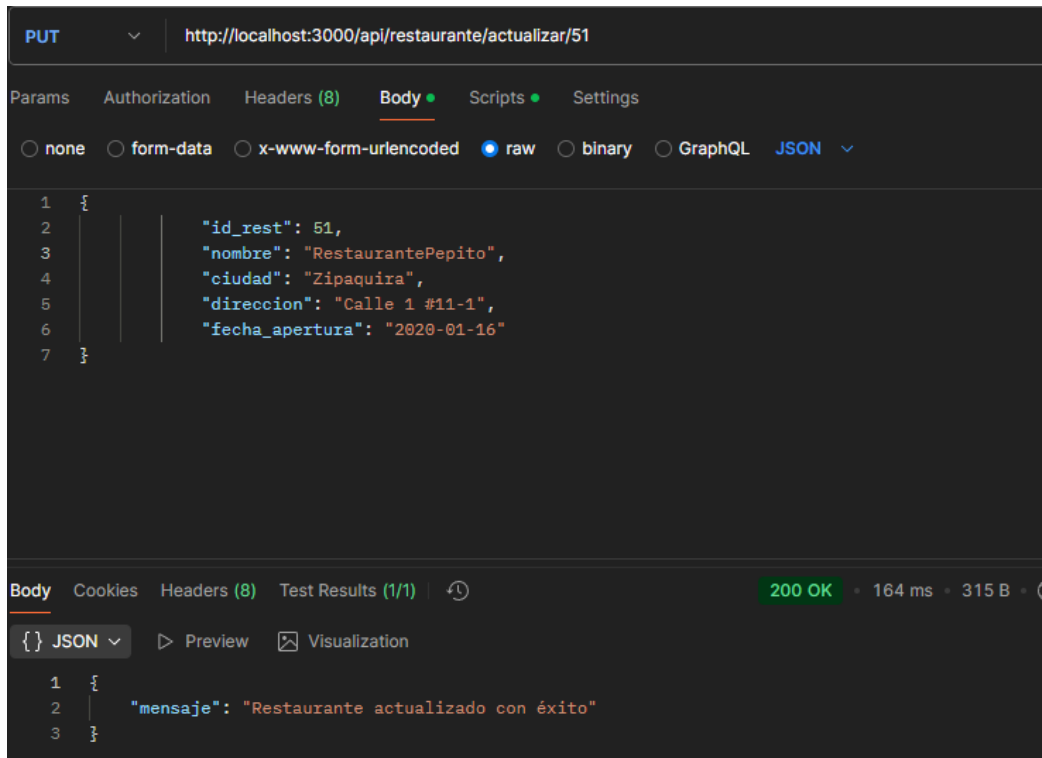
- Probamos las peticiones para comprobar que están funcionando correctamente haremos la prueba con la tabla de restaurante en la aplicación postman
- Se ejecuta en get para traer los registros con la api de obtener



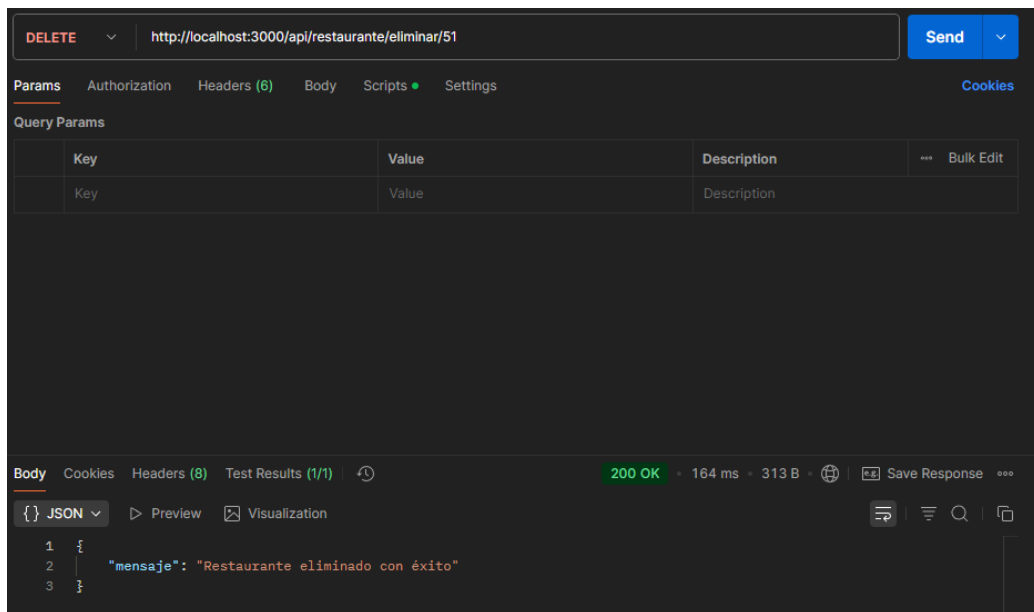
- Se usa el post para insertar un nuevo registro con la api de insertar , para esto toca colocar los parametos en el apartado de body



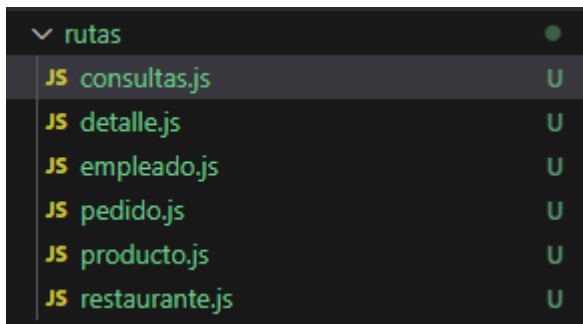
- Se usa put para modificar algún parámetro de un registro , para esto se tiene q escribir el parámetro que se desea cambiar con la api de actualizar



- Se usa del para eliminar un registro, para esto se tiene que ingresar la id del registro que se desea eliminar con la api de eliminar



- Se crea el archivo de consultas



▼ rutas	●
JS consultas.js	U
JS detalle.js	U
JS empleado.js	U
JS pedido.js	U
JS producto.js	U
JS restaurante.js	U

- Se agrega la conexión al archivo



```
// Conexión a rutas
app.use('/api', require('./rutas/restaurante'));
app.use('/api', require('./rutas/empleado'));
app.use('/api', require('./rutas/producto'));
app.use('/api', require('./rutas/pedido'));
app.use('/api', require('./rutas/detalle'));
app.use('/api', require('./rutas/consultas'));
```


- Se crea la api para obtener productos de un pedido específico

```
router.get('/pedido/productos/:id', (req, res) => {  
  const { id } = req.params;  
  const query = `  
    SELECT producto.id_prod, producto.nombre, producto.precio, detallepedido.cantidad, detallepedido.subtot  
    FROM detallepedido  
    INNER JOIN producto ON detallepedido.id_prod = producto.id_prod  
    WHERE detallepedido.id_pedido = $1  
  `;  
  client.query(query, [id])  
    .then(result => res.status(200).json({ data: result.rows }))  
    .catch(error => res.status(500).json({ error: error.message }));  
});
```

- router.get() crea una ruta GET
- req.params.id se pide el parámetro del ID del pedido desde la URL.
- INNER JOIN vincula productos con su detalle en el pedido haciendo una consulta SQL
- Se usa client.query() para ejecutar la consulta en PostgreSQL.
- Si va bien, responde con 200 OK y un JSON con los productos del pedido.
- Si ocurre un error, devuelve 500 Internal Server Error

- Se crea la api de Obtener los productos más vendidos

```
// Obtener los productos más vendidos
router.get('/productos/mas_vendidos/:cantidad', (req, res) => {
  const { cantidad } = req.params;
  const query = `
    SELECT producto.id_prod, producto.nombre, SUM(detallepedido.cantidad) AS total_vendidos
    FROM detallepedido
    INNER JOIN producto ON detallepedido.id_prod = producto.id_prod
    GROUP BY producto.id_prod, producto.nombre
    HAVING SUM(detallepedido.cantidad) > $1
    ORDER BY total_vendidos DESC
  `;
  client.query(query, [cantidad])
    .then(result => res.status(200).json({ data: result.rows }))
    .catch(error => res.status(500).json({ error: error.message }));
});
```

- router.get() se utiliza para usar la ruta de GET.
- req.params.cantidad indica que requiere el parámetro de cantidad ,el cual debe indicar el mínimo de unidades vendidas
- La consulta se agrupa por producto y suma cuántas veces se ha vendido.
- Solo se devuelven productos con ventas mayores a cantidad
- Si la consulta salió correctamente responde con el código 200 con los productos más vendidos, ordenados de mayor a menor.
- Si hay error, se devuelve 500.

- Se crea la api de Obtener el total de ventas por restaurante

```
router.get('/ventas/restaurante/:id', (req, res) => {
  const { id } = req.params;
  const query = `
    SELECT restaurante.id_rest, restaurante.nombre, SUM(pedido.total) AS total_ventas
    FROM restaurante
    INNER JOIN pedido ON restaurante.id_rest = pedido.id_rest
    WHERE restaurante.id_rest = $1
    GROUP BY restaurante.id_rest, restaurante.nombre
  `;
  client.query(query, [id])
    .then(result => res.status(200).json({ data: result.rows }))
    .catch(error => res.status(500).json({ error: error.message }));
});
```

- req.params.id toma el ID del restaurante enviado en la URL.
- Suma (SUM) el total de todos los pedidos asociados a ese restaurante.
- Usa INNER JOIN para conectar restaurante con pedido.
- Agrupa por restaurante.id_rest y nombre.
- client.query() ejecuta la consulta con el ID como parámetro.
- Si todo va bien, responde con 200 OK y el total de ventas.
- Si hay un error, responde con 500 Internal Server Error.

- Obtener los pedidos por fecha específica

```
//Obtener los pedidos por fecha específica
router.get('/pedidos/fecha/:fecha', (req, res) => {
  const { fecha } = req.params;
  const query = 'SELECT * FROM pedido WHERE fecha = $1';
  client.query(query, [fecha])
    .then(result => res.status(200).json({ data: result.rows }))
    .catch(error => res.status(500).json({ error: error.message }));
});
```

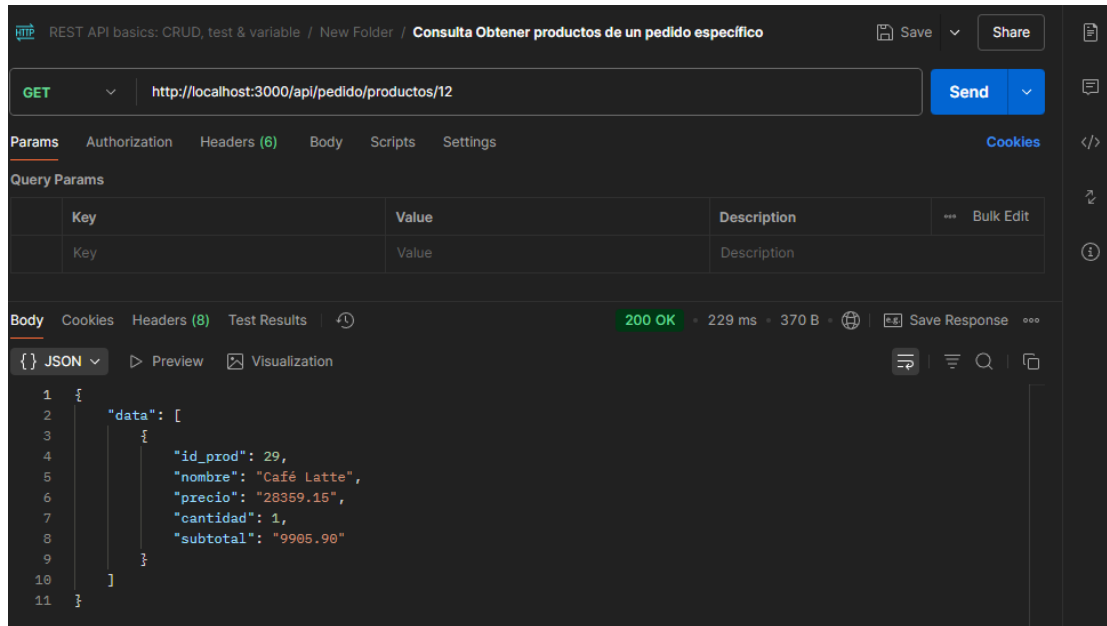
- Ruta GET con fecha como parámetro.
- Se buscan todos los pedidos con la fecha exacta proporcionada.
- Devuelve todos los campos de la tabla pedido.
- Respuesta 200 con los pedidos encontrados.
- Si falla, se devuelve 500.

- Obtener los empleados por rol en un restaurante

```
//Obtener los empleados por rol en un restaurante
router.get('/empleados/:rol/:id_rest', (req, res) => {
  const { rol, id_rest } = req.params;
  const query = 'SELECT * FROM empleado WHERE rol = $1 AND id_rest = $2';
  client.query(query, [rol, id_rest])
    .then(result => res.status(200).json({ data: result.rows }))
    .catch(error => res.status(500).json({ error: error.message }));
});
```

- router.get() define la ruta GET /empleados/:rol/:id_rest.
- req.params.rol y req.params.id_rest capturan el rol y restaurante de la URL.
- La consulta busca empleados que coincidan con ese rol y pertenezcan a ese restaurante.
- client.query() ejecuta la consulta con ambos valores.
- Si encuentra coincidencias, responde con 200 OK y los datos de los empleados.
- Si falla, responde con 500 Internal Server Error.

- Probamos las consultas para comprobar que están funcionando correctamente
- Consulta Obtener productos de un pedido específico



REST API basics: CRUD, test & variable / New Folder / Consulta Obtener productos de un pedido específico

GET <http://localhost:3000/api/pedido/productos/12> Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

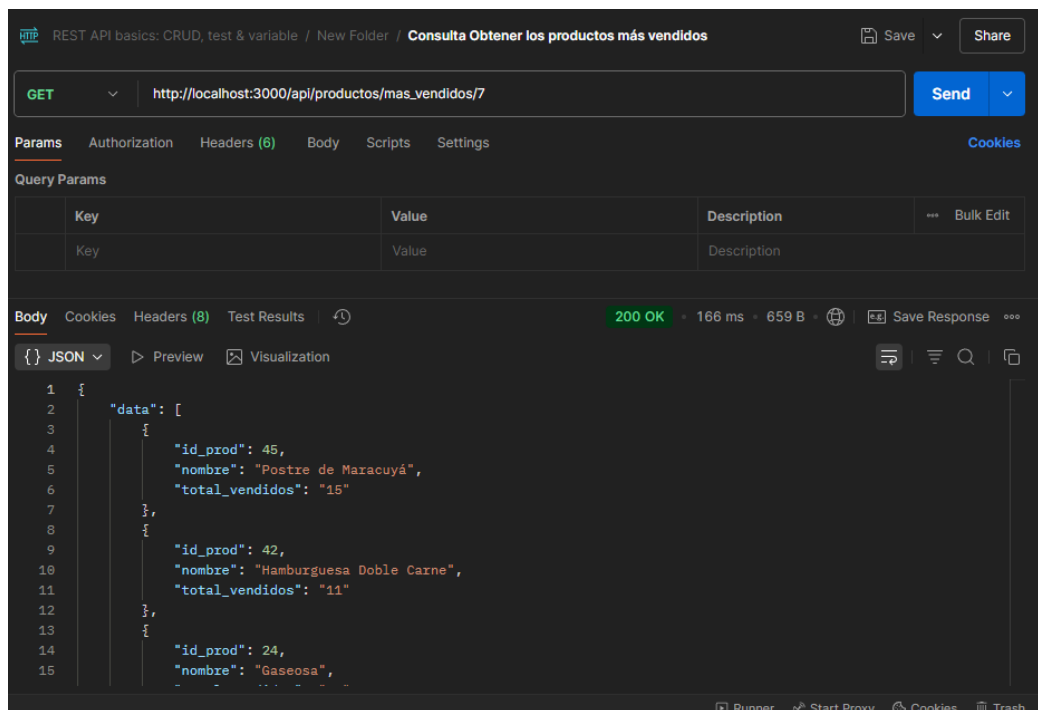
Body Cookies Headers (8) Test Results 200 OK • 229 ms • 370 B Save Response

JSON Preview Visualization

```

1 {
2   "data": [
3     {
4       "id_prod": 29,
5       "nombre": "Café Latte",
6       "precio": "28359.15",
7       "cantidad": 1,
8       "subtotal": "9905.90"
9     }
10  ]
11 }
```

- Consulta obtener los productos mas vendidos



REST API basics: CRUD, test & variable / New Folder / Consulta Obtener los productos más vendidos

GET http://localhost:3000/api/productos/mas_vendidos/7 Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results 200 OK • 166 ms • 659 B Save Response

JSON Preview Visualization

```

1 {
2   "data": [
3     {
4       "id_prod": 45,
5       "nombre": "Postre de Maracuyá",
6       "total_vendidos": "15"
7     },
8     {
9       "id_prod": 42,
10      "nombre": "Hamburguesa Doble Carne",
11      "total_vendidos": "11"
12     },
13     {
14       "id_prod": 24,
15       "nombre": "Gaseosa",
16       "total_vendidos": "10"
17     }
18  ]
19 }
```

Runner Start Proxy Cookies Trash

- Consulta obtener el total de ventas de un restaurante

REST API basics: CRUD, test & variable / New Folder / Consulta Obtener el total de ventas

GET Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results 200 OK • 156 ms • 331 B Save Response

{ } JSON Preview Visualization

```
1 {
2   "data": [
3     {
4       "id_rest": 5,
5       "nombre": null,
6       "total_ventas": "17103.94"
7     }
8   ]
9 }
```

- Consulta Obtener los pedidos por fecha específica

GET Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results 200 OK • 157 ms • 359 B Save Response

{ } JSON Preview Visualization

```
1 {
2   "data": [
3     {
4       "id_pedido": 8,
5       "fecha": "2024-01-09T05:00:00.000Z",
6       "id_rest": 8,
7       "total": "73185.47"
8     }
9   ]
10 }
```

- Consulta Obtener los empleados por rol en un restaurante

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/api/empleados/Cocinero/50`. The response is a 200 OK status with a JSON body containing an array of two employee objects.

Query Params

Key	Value	Description
Key	Value	Description

Body | Cookies | Headers (8) | Test Results | 200 OK • 179 ms • 431 B | Save Response

JSON Preview Visualization

```
1 {
2   "data": [
3     {
4       "id_empleado": 50,
5       "nombre": "Andrés García",
6       "rol": "Cocinero",
7       "id_rest": 50
8     },
9     {
10      "id_empleado": 51,
11      "nombre": "Camilo Rodríguez",
12      "rol": "Cocinero",
13      "id_rest": 50
14    }
15  ]
16 }
```