

PARCIAL 2

Javier Adrian Pedraza Garcia

Ingeniería de sistemas, Corporación Universitaria Minuto de Dios

60747: Bases de Datos Masivas

Prof. William Alexander Matallana Parra

2025

PARCIAL SEGUNDO CORTE

```
MINGW64:/c/Users/janpe/Desktop/parcialsegundocorte
create mode 100644 README.md

janpe@LAPTOP-612TSCVB MINGW64 ~/Desktop/parcialsegundocorte (master)
$ git branch -M main

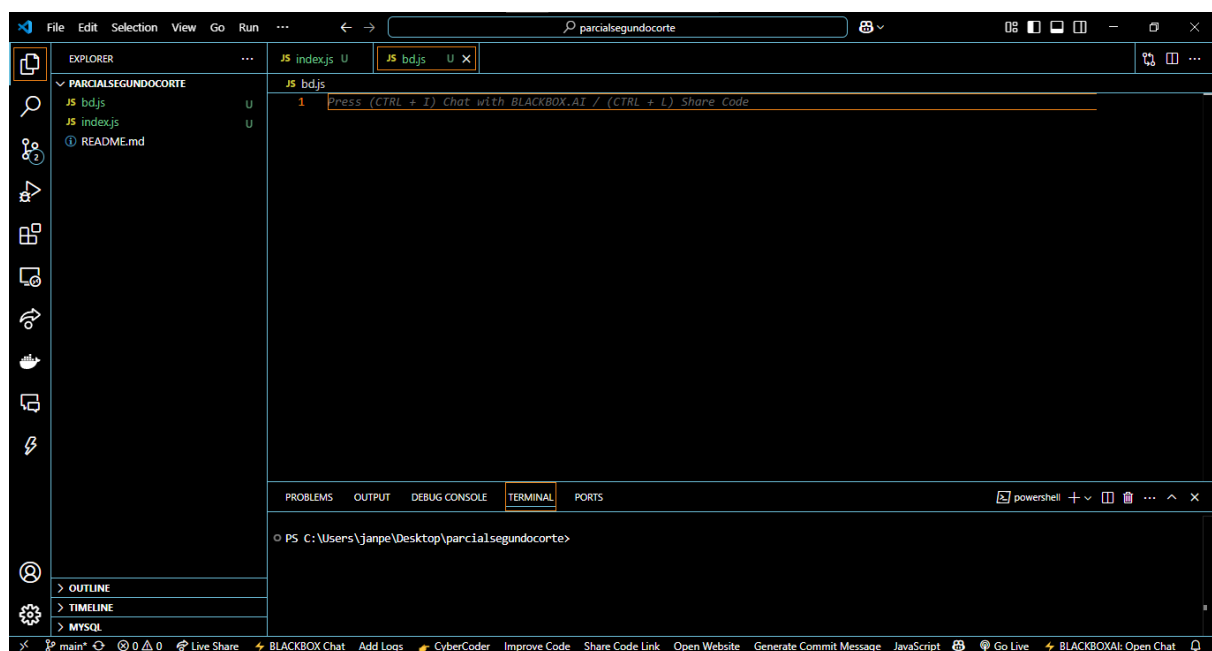
janpe@LAPTOP-612TSCVB MINGW64 ~/Desktop/parcialsegundocorte (main)
$ git remote add origin https://github.com/GAdrianPedrazaJ/PARCIAL2JavierAdrianPedrazaGarcia.git

janpe@LAPTOP-612TSCVB MINGW64 ~/Desktop/parcialsegundocorte (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 256 bytes | 256.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/GAdrianPedrazaJ/PARCIAL2JavierAdrianPedrazaGarcia.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

janpe@LAPTOP-612TSCVB MINGW64 ~/Desktop/parcialsegundocorte (main)
$

janpe@LAPTOP-612TSCVB MINGW64 ~/Desktop/parcialsegundocorte (main)
$
```

NOTA: Conexión de la carpeta creada para el proyecto con el repositorio alojado el GIT



NOTA: Creación de la carpeta del proyecto y la creación de los archivos .js bd e index mediante el comando “npm -y bd.js” y npm -y index.js”

```
1 CREATE TABLE restaurante (  
2     id_rest SERIAL PRIMARY KEY,  
3     nombre VARCHAR(100),  
4     ciudad VARCHAR(100),  
5     direccion VARCHAR(150),  
6     fecha_apertura DATE  
7 );  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

Results Chart ✓ Source Primary Database Role postgres Run CTRL ↵

NOTA: Creación de la tabla restaurante que contiene: id del restaurante que es llave primaria y de tipo SERIAL para ser auto-incrementable, nombre de tipo VARCHAR, ciudad de tipo VARCHAR, dirección de tipo Varchar y fecha de apertura que es de tipo DATE para guardar fecha.

```
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 );  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

Results Chart ✓ Source Primary Database Role postgres Run CTRL ↵

NOTA: Creación de la tabla empleado con los campos: id del empleado que es de tipo SERIAL y es llave primaria, nombre que es de tipo VARCHAR, rol que es de tipo VARCHAR y id del restaurante que es llave foránea que hace referencia a la tabla del restaurante ya que ese empleado estará trabajando en ese restaurante.

```
1 CREATE TABLE producto (  
2     id_prod SERIAL PRIMARY KEY,  
3     nombre VARCHAR(100),  
4     precio NUMERIC(10,2)  
5 );  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

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

NOTA: Creación de la tabla producto que tiene: id del producto que es de tipo SERIAL para ser auto incrementable y también es llave primaria, nombre que es de tipo VARCHAR y el precio del producto que es de tipo NUMERIC para aceptar decimales.

```
1 CREATE TABLE pedido (  
2     id_pedido SERIAL PRIMARY KEY,  
3     fecha DATE,  
4     id_rest INT REFERENCES restaurante(id_rest),  
5     total NUMERIC(10,2)  
6 );  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

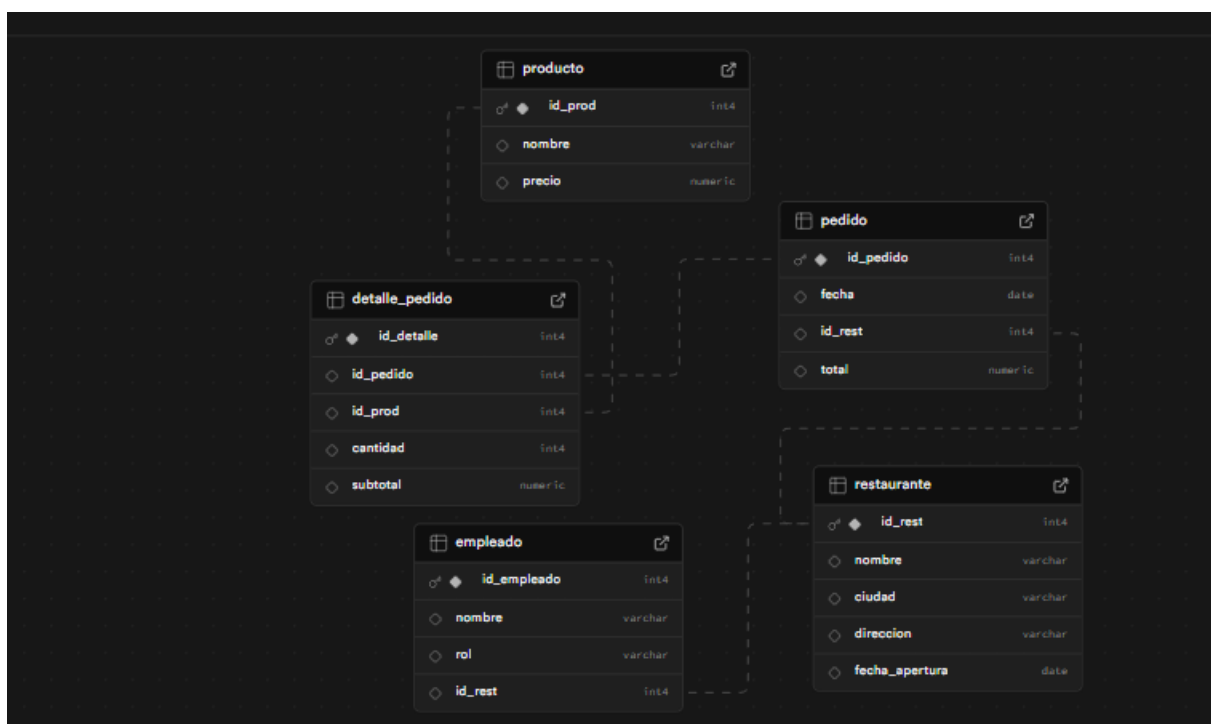
Results Chart ✓ ⋮ Source Primary Database Role postgres Run CTRL ↵

NOTA: Creación de la tabla pedido que tiene: id del pedido que es de tipo SERIAL para que sea auto incrementable y es llave primaria, fecha que es de tipo DATE para guardar la fecha en la que se realizó el pedido, id del restaurante que es llave foránea con referencia al restaurante por que el pedido se está haciendo a ese restaurante y total que es de tipo NUMERIC para que acepte decimales.

```
1 CREATE TABLE detalle_pedido (  
2   id_detalle SERIAL PRIMARY KEY,  
3   id_pedido INT REFERENCES pedido(id_pedido),  
4   id_prod INT REFERENCES producto(id_prod),  
5   cantidad INT,  
6   subtotal NUMERIC(10,2)  
7 );  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

Results Chart ✓ Source Primary Database Role postgres Run CTRL ↵

NOTA: Creación de la tabla detalle_pedido que tiene: id del detalle que es de tipo SERIAL para que sea auto incrementable y es llave primaria, id del pedido que es llave foránea hacia la tabla pedidos ya que aquí se guardarán los detalles de los pedidos hechos, id del producto que es llave foránea haciendo referencia a la tabla de productos ya que en esa tabla están guardados todos los productos, cantidad que es de tipo INT y el sub total que es de tipo NUMERIC para que acepte decimales.



NOTA: Diagrama del modelo entidad relación sacado de SUPABASE.

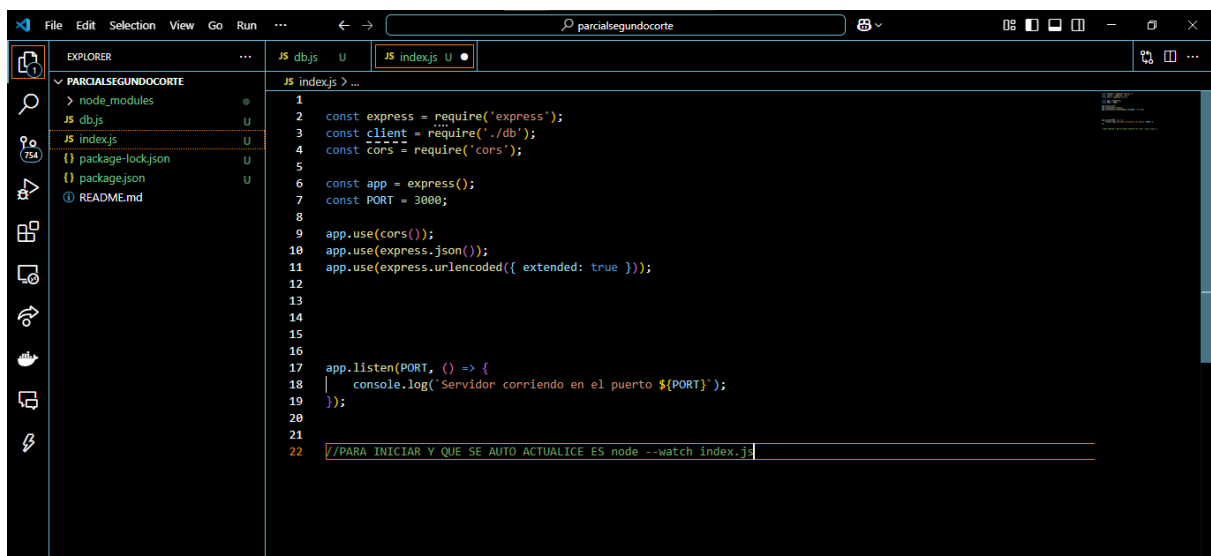


The screenshot shows the Visual Studio Code editor with a project named 'parcialsegundocorte'. The Explorer sidebar on the left shows the file structure: 'node_modules', 'db.js', 'index.js', 'package-lock.json', 'package.json', and 'README.md'. The main editor window displays the content of 'db.js'. The code defines a 'Client' object using the 'pg' library and connects it to a Supabase database. The connection details are as follows:

```
1 //Trabajo realizado por javier durian pedraza garcia
2
3
4 const { Client } = require('pg');
5
6 // Datos de conexión con Supabase
7 const client = new Client({
8   host: 'aws-0-us-east-2.pooler.supabase.com',
9   port: 5432,
10  user: 'postgres.pvexvntdyknjpjqdzpjn',
11  password: 'Janpedr@1110',
12  database: 'postgres',
13  ssl: {
14    rejectUnauthorized: false
15  }
16 });
17
18
19 client.connect((error) => {
20   if (error) {
21     console.log('Error conectando con la base de datos:', error);
22     return;
23   } else {
24     console.log('Conectado con la base de datos');
25   }
26 });
27
28 module.exports = client;
29
```

The bottom of the editor shows the 'TERMINAL' tab with the output: 'Conectado con la base de datos'.

NOTA: Código de la conexión con la base de datos donde se solicita el host, el puerto, el usuario, la contraseña y el nombre de la base de datos. Que todos estos datos son dados en la plataforma de SUPABASE



The screenshot shows the Visual Studio Code editor with the same project. The Explorer sidebar shows the file structure. The main editor window displays the content of 'index.js'. The code sets up an Express.js server, connects to the database using the 'db.js' module, and starts the server on a specified port. The code is as follows:

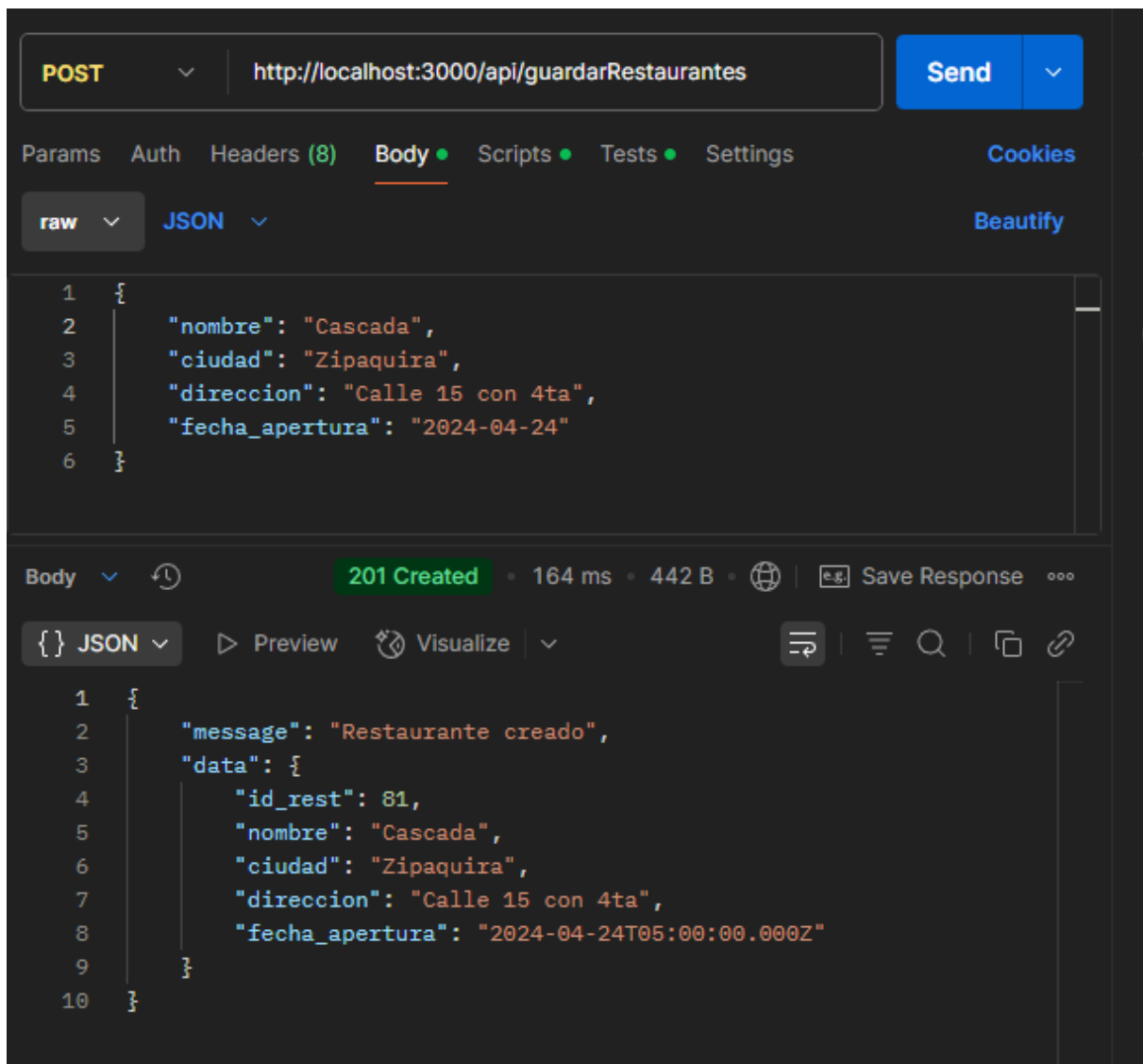
```
1
2 const express = require('express');
3 const client = require('./db');
4 const cors = require('cors');
5
6 const app = express();
7 const PORT = 3000;
8
9 app.use(cors());
10 app.use(express.json());
11 app.use(express.urlencoded({ extended: true }));
12
13
14
15
16
17 app.listen(PORT, () => {
18   console.log('Servidor corriendo en el puerto ${PORT}');
19 });
20
21
22 //PARA INICIAR Y QUE SE AUTO ACTUALICE ES node --watch index.js

```

NOTA: Código inicial del index que solicita el express, la base de datos y el cors.

```
//API PARA CRUD DE RESTAURANTE
app.post('/api/guardarrestaurantes', 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) RE

  try {
    const result = await client.query(query, [nombre, ciudad, direccion, fecha_apertura]);
    res.status(201).json({ message: "Restaurante creado", data: result.rows[0] });
  } catch (error) {
    res.status(500).json({ message: 'Error creando restaurante', error: error.message });
  }
});
```



NOTA: API para la inserción de datos de la tabla restaurante que nos solicita que en el body dentro del postman ingresemos el nombre, ciudad, dirección y la fecha de apertura que se guardaran en \$1, &2, \$3, \$4 que están en la consulta de INSERT INTO. Además, la prueba que se realizó dentro de POSTMAN para verificar que si funciona

```
// Obtener todos los restaurantes
app.get('/api/obtenerrestaurantes', async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM restaurante');
    res.status(200).json({ success: true, data: result.rows });
  } catch (error) {
    res.status(500).json({ success: false, message: 'Error al obtener restaurantes', error: error.message });
  }
});
```

PARCIAL2 / RESTAURANTE / **Obtener Restaurantes** Save Share

GET ▼ http://localhost:3000/api/obtenerRestaurantes Send ▼

Params Auth Headers (6) Body Scripts ● Tests ● Settings Cookies

Query Params

	Key	Value	Descript...	...	Bulk Edit
	Key	Value	Description		

Body ▼ 🕒 200 OK • 139 ms • 11.01 KB • 🌐 📄 Save Response ...

{} **JSON** ▼ ▶ Preview 🔄 Visualize ▼ 🔍 📄 🔗

```

1  {
2    "success": true,
3    "data": [
4      {
5        "id_rest": 1,
6        "nombre": "La Cazuela",
7        "ciudad": "Madrid",
8        "direccion": "Calle Mayor 45",
9        "fecha_apertura": "2015-03-10T05:00:00.000Z"
10     },
11     {
12       "id_rest": 2,
13       "nombre": "El Fogón",
14       "ciudad": "Barcelona",
15       "direccion": "Avenida Diagonal 88"

```

NOTA: API para realizar una consulta de SELECT * FROM que llamara todas las consultas que hay en la tabla restaurante


```
// Actualizar restaurante
app.put('/api/actualizarrestaurantes/: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 {
    const result = await client.query(query, [nombre, ciudad, direccion, fecha_apertura, id]);
    res.status(200).json({ message: "Restaurante actualizado", data: result.rowCount });
  } catch (error) {
    res.status(500).json({ message: "Error actualizando restaurante", error: error.message });
  }
});
```

```
1 SELECT * FROM restaurante;
```

id_rest	nombre	ciudad	direccion	fecha_apertura
1	La Cazuela	Madrid	Calle Mayor 45	2015-03-10
2	El Fogón	Barcelona	Avenida Diagonal 88	2017-05-22
3	Sabor Criollo	Valencia	Plaza Central 12	2018-10-01
4	Tierra y Mar	Sevilla	Calle Betis 33	2020-02-14
5	Buen Provecho	Bilbao	Gran Vía 15	2019-07-30
6	Sabores del Mundo	Granada	Calle Recogidas 20	2021-04-05
7	Gusto Italiano	Zaragoza	Paseo Independencia 7	2016-12-12
8	Delicias Express	Murcia	Avenida Libertad 14	2022-01-19
9	Mar y Tierra	Alicante	Calle del Sol 11	2014-06-18
10	El Rincón Gourmet	Málaga	Boulevard 9	2023-09-03

PARCIAL2 / RESTAURANTE / Actualizar Restaurantes

Save

Share

PUT

http://localhost:3000/api/actualizarRestaurantes/1

Send

Params Auth Headers (8) Body Scripts Tests Settings

Cookies

raw

JSON

Beautify

```
1 {
2   "nombre": "La Cazuela",
3   "ciudad": "Madrid",
4   "direccion": "Calle menor 45",
5   "fecha_apertura": "20158-03-10"
6 }
```

Body

200 OK

114 ms

313 B

Save Response

{ } JSON

Preview

Visualize

```
1 {
2   "message": "Restaurante actualizado",
3   "data": 1
4 }
```

```
1 SELECT * FROM restaurante;
```

id_rest	nombre	ciudad	direccion	fecha_apertura
73	El Rincón del Chef	Yopal	Carrera 12 #13-14	2018-02-12
74	Sazón Casero	Florencia	Calle 7 #8-9	2021-01-19
75	La Esquina	San Andrés	Avenida 3 #4-5	2017-09-23
76	Delicias del Sur	Leticia	Calle 1 #2-3	2016-12-31
77	El Buen Sabor	Arauca	Carrera 14 #15-16	2022-05-27
78	La Parrilla Criolla	Mocoa	Calle 9 #10-11	2015-10-13
79	Sabores de Mi Tierra	Mitú	Carrera 3 #4-5	2014-06-09
80	El Fogón del Pueblo	Puerto Carreño	Calle 6 #7-8	2013-04-18
81	Cascada	Zipaquirá	Calle 15 con 4ta	2024-04-24
1	La Cazuela	Madrid	Calle menor 45	20158-03-10

NOTA: API para actualizar los registros de la tabla restaurante donde nos solicitan que en el body pongamos el nombre, ciudad, dirección y la fecha de apertura para ser modificadas mediante la consulta UPDATE que esta en la API y que en la URL se ponga el id del restaurante

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

PARCIAL2 / RESTAURANTE / Eliminar Restaurantes

DELETE Send

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body 110 ms • 461 B • Save Response

{ JSON Preview Visualize

```
1 {
2   "message": "Error eliminando restaurante",
3   "error": "update or delete on table \"restaurante\" violates
4     foreign key constraint \"empleado_id_rest_fkey\" on table
      \"empleado\""
```

DELETE Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body Cookies Headers (8) Test Results (1/1) 200 OK • 181 ms • 311 B • Save Response

{ JSON Preview Visualize

```
1 {
2   "message": "Restaurante eliminado",
3   "data": 1
4 }
```

NOTA: API para eliminar un registro de la tabla restaurante utilizando la id del restaurante que se debe poner en la URL utilizando la consulta DELETE que está en la misma API. En la primera captura de la prueba no se logra eliminar la consulta ya que en las tablas no estaba habilitado el ON DELETE CASCADE y al haber empleado relacionado no dejaba realizar la eliminación.

```
app.post('/api/empleados', async (req, res) => {
  const { nombre, rol, id_rest } = req.body;
  const query = 'INSERT INTO empleado (nombre, rol, id_rest) VALUES ($1, $2, $3) RETURNING *';

  try {
    const result = await client.query(query, [nombre, rol, id_rest]);
    res.status(201).json({ message: "Empleado creado", data: result.rows[0] });
  } catch (error) {
    res.status(500).json({ message: 'Error creando empleado', error: error.message });
  }
});
```

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

- URL:** `http://localhost:3000/api/guardarEmpleados`
- Method:** `POST`
- Body (raw):**

```
{
  "nombre": "Juan Pérez",
  "rol": "Cocinero",
  "id_rest": 1
}
```
- Status:** `201 Created` (135 ms, 381 B)
- Response (JSON):**

```
{
  "message": "Empleado creado",
  "data": {
    "id_empleado": 111,
    "nombre": "Juan Pérez",
    "rol": "Cocinero",
    "id_rest": 1
  }
}
```

NOTA: API para realizar el guardado o insertar un registro en la tabla de empleados donde la consulta nos pide que en el body ingresemos el nombre, el rol que va a desempeñar y la id del restaurante en el que trabajara este.

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

PARCIAL2 / EMPLEADO / Obtener Empleados

GET http://localhost:3000/api/obtenerEmpleados

Send

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Descrip...	Bulk Edit
Key	Value	Description	

Body 200 OK • 105 ms • 8.25 KB • Save Response

{ } JSON Preview Visualize

```
4   "id_empleado": 1,
5   "nombre": "Juan Pérez",
6   "rol": "Cocinero",
7   "id_rest": 1
8 },
9 {
10  "id_empleado": 2,
11  "nombre": "Ana Torres",
12  "rol": "Mesero",
13  "id_rest": 1
14 },
15 {
16  "id_empleado": 3,
17  "nombre": "Luis Gómez",
18  "rol": "Administrador",
19  "id_rest": 2
20 },
21 }
```

NOTA: API para visualizar los registros de la tabla empleados utilizando la consulta de SLECT * FROM y esto será devuelto en un .json

```

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

```

PARCIAL2 / EMPLEADO / Eliminar Empleados Save Share

DELETE ▼ **http://localhost:3000/api/eliminarEmpleados/1** Send ▼

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Descript...	...	Bulk Edit
	Key	Value	Description		

Body ▼ 🔄 **200 OK** • 103 ms • 308 B • 🌐 📄 Save Response ⋮

{} **JSON** ▼ ▶ Preview 🔄 Visualize ▼ 🔍 📄 🔗

```

1  {
2    "message": "Empleado eliminado",
3    "data": 1
4  }

```

Console Postbot Runner Vault

NOTA: API para eliminar un registro de la tabla empleado que nos solicita que en la URL se ponga el id del empleado que se va a eliminar y este proceso se realizara mediante la consulta DELETE que esta en el API y dentro del POSTMAN nos devolverá un .JSON que nos dirá si se eliminó o no el registro

```
app.put('/api/actualizareempleados/: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  
  `;  
};
```

The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/api/actualizarEmpleados/10`. The request body is a JSON object: `{ "nombre": "Juan Pérez", "rol": "Mesero", "id_rest": 1 }`. The response is a 200 OK status with a response time of 118 ms and a size of 310 B. The response body is a JSON object: `{ "message": "Empleado actualizado", "data": 1 }`.

PUT `http://localhost:3000/api/actualizarEmpleados/10` Send

Params Auth Headers (8) **Body** Scripts Tests Settings Cookies

raw JSON Beautify

```
1 {  
2   "nombre": "Juan Pérez",  
3   "rol": "Mesero",  
4   "id_rest": 1  
5 }  
6
```

Body 200 OK • 118 ms • 310 B • Save Response

{ } JSON Preview Visualize

```
1 {  
2   "message": "Empleado actualizado",  
3   "data": 1  
4 }
```

NOTA: API para actualizar un registro de la tabla empleado donde el API nos pide que en la URL pongamos el id del empleado que vamos a actualizar los datos y el el body no solicita el nombre del empleado, el cargo o rol del empleado y el id del restaurante en el que trabaja o trabajara

```
app.post('/api/guardarproductos', async (req, res) => {
  const { nombre, precio } = req.body;
  const query = 'INSERT INTO producto (nombre, precio) VALUES ($1, $2) RETURNING *';

  try {
    const result = await client.query(query, [nombre, precio]);
    res.status(201).json({ message: "Producto creado", data: result.rows[0] });
  } catch (error) {
    res.status(500).json({ message: 'Error creando producto', error: error.message });
  }
});
```

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

- Method:** POST
- URL:** http://localhost:3000/api/guardarProductos
- Body (JSON):**

```
{
  "nombre": "Pizza Hawaiana",
  "precio": 25000
}
```
- Response:**
 - Status:** 201 Created
 - Time:** 138 ms
 - Size:** 370 B
 - Body (JSON):**

```
{
  "message": "Producto creado",
  "data": {
    "id_prod": 131,
    "nombre": "Pizza Hawaiana",
    "precio": "25000.00"
  }
}
```

NOTA: API para realizar la inserción de datos en la tabla productos que nos solicita que en el body se coloque el nombre del producto, y el precio de dicho producto para que se pueda realizar correctamente la consulta de INSERT.


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

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

- Request Method:** GET
- Request URL:** http://localhost:3000/api/obtenerProductos
- Response Status:** 200 OK
- Response Time:** 104 ms
- Response Size:** 7.69 KB
- Response Body (JSON):**

```
{
  "data": [
    {
      "id_prod": 1,
      "nombre": "Hamburguesa Clásica",
      "precio": "8.50"
    },
    {
      "id_prod": 2,
      "nombre": "Pizza Margarita",
      "precio": "9.75"
    },
    {
      "id_prod": 3,
      "nombre": "Ensalada César",
      "precio": "7.20"
    },
    {
      "id_prod": 4,

```

NOTA: API para visualizar todos los productos que hay en la tabla de productos mediante la consulta SELECT * FROM que hay en el cuerpo del API

```

app.put('/api/actualizarProductos/: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 {
    const result = await client.query(query, [nombre, precio, id]);
    res.status(200).json({ message: "Producto actualizado", data: result.rowCount });
  } catch (error) {
    res.status(500).json({ message: "Error actualizando producto", error: error.message });
  }
});

```

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

- URL:** `http://localhost:3000/api/actualizarProductos/2`
- Method:** `PUT`
- Body (JSON):**

```

{
  "nombre": "Pizza Margarita",
  "precio": 10.25
}

```
- Response:**
 - Status:** `200 OK`
 - Body (JSON):**

```

{
  "message": "Producto actualizado",
  "data": 1
}

```

NOTA: API para actualizar un registro de la tabla productos donde nos solicita que en la URL pongamos la id del producto que se va a modificar y en el cuerpo pongamos en nombre y el precio del producto para que se pueda ejecutar correctamente la consulta UPDATE que esta en el cuerpo de la API.

```

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

```

[Gua](#) • [GET Obte](#) • [PUT Actu.](#) • [DEL Elimii](#) • > + ▾ No environment ▾

[HTTP](#) PARCIAL2 / PRODUCTO / **Eliminar Producto** [Save](#) ▾ [Share](#)

DELETE ▾ <http://localhost:3000/api/eliminarProductos/4> [Send](#) ▾

[Params](#) [Auth](#) [Headers \(6\)](#) [Body](#) [Scripts](#) • [Tests](#) • [Settings](#) [Cookies](#)

Query Params

	Key	Value	Descript...	...	Bulk Edit
	Key	Value	Description		

Body ▾ [500 Internal Server Error](#) • 130 ms • 467 B • [Save Response](#) ...

[JSON](#) ▾ [Preview](#) [Visualize](#) ▾

```

1  {
2    "message": "Error eliminando producto",
3    "error": "update or delete on table \"producto\" violates
4    foreign key constraint \"detalle_pedido_id_prod_fkey\" on
      table \"detalle_pedido\"
  }

```

HTTP PARCIAL2 / PRODUCTO / Eliminar Producto Save Share

DELETE http://localhost:3000/api/eliminarProductos/100 Send

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Descript...	Bulk Edit
	Key	Value	Description	

Body 200 OK • 117 ms • 308 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "message": "Producto eliminado",
3   "data": 1
4 }
```

NOTA: API para eliminar un registro de la producto donde nos solicitan que en la URL se ponga la id del producto a eliminar para que se ejecute la consulta de tipo DELETE que se encuentra en el cuerpo de la API

```
//API PARA CRUD DE PEDIDOS
app.post('/api/guardarPedidos', async (req, res) => {
  const { fecha, id_rest, total } = req.body;
  const query = 'INSERT INTO pedido (fecha, id_rest, total) VALUES ($1, $2, $3) RETURNING *';

  try {
    const result = await client.query(query, [fecha, id_rest, total]);
    res.status(201).json({ message: "Pedido creado", data: result.rows[0] });
  } catch (error) {
    res.status(500).json({ message: 'Error creando pedido', error: error.message });
  }
});
```

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

- URL:** `http://localhost:3000/api/guardarPedidos`
- Method:** `POST`
- Body (raw):**

```
{
  "fecha": "2024-04-24",
  "id_rest": 1,
  "total": 75000
}
```
- Status:** `201 Created` (132 ms, 390 B)
- Response Body (JSON):**

```
{
  "message": "Pedido creado",
  "data": {
    "id_pedido": 51,
    "fecha": "2024-04-24T05:00:00.000Z",
    "id_rest": 1,
    "total": "75000.00"
  }
}
```

NOTA: API para guardar el registro de un nuevo pedido que se realice donde en el body nos solicita que pongamos la fecha, la id del restaurante y el total a pagar

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

Actu. • DEL Elimir • POST Gua • **GET Obte** • > + v No environment v

PARCIAL2 / PEDIDOS / **Obtener Pedido** Save Share

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

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Descript...	...	Bulk Edit
	Key	Value	Description		

Body 200 OK • 101 ms • 4.28 KB • Save Response

JSON Preview Visualize

```

1  {
2    "data": [
3      {
4        "id_pedido": 1,
5        "fecha": "2024-01-10T05:00:00.000Z",
6        "id_rest": 1,
7        "total": "25.50"
8      },
9      {
10       "id_pedido": 2,
11       "fecha": "2024-01-11T05:00:00.000Z",
12       "id_rest": 2,
13       "total": "35.60"
14     },
15     {
16       "id_pedido": 3.
```

NOTA: API para visualizar los registros que se encuentran en la tabla de pedidos utilizando la consulta `SELECT * FROM` que se encuentra en el cuerpo de la API

```

app.put('/api/actualizarPedidos/: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 {
    const result = await client.query(query, [fecha, id_rest, total, id]);
    res.status(200).json({ message: "Pedido actualizado", data: result.rowCount });
  } catch (error) {
    res.status(500).json({ message: 'Error actualizando pedido', error: error.message });
  }
});

```

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

- Method:** PUT
- URL:** http://localhost:3000/api/actualizarPedidos/1
- Body (JSON):**

```

{
  "fecha": "2024-04-24",
  "id_rest": 1,
  "total": 75000
}

```
- Status:** 200 OK
- Response (JSON):**

```

{
  "message": "Pedido actualizado",
  "data": 1
}

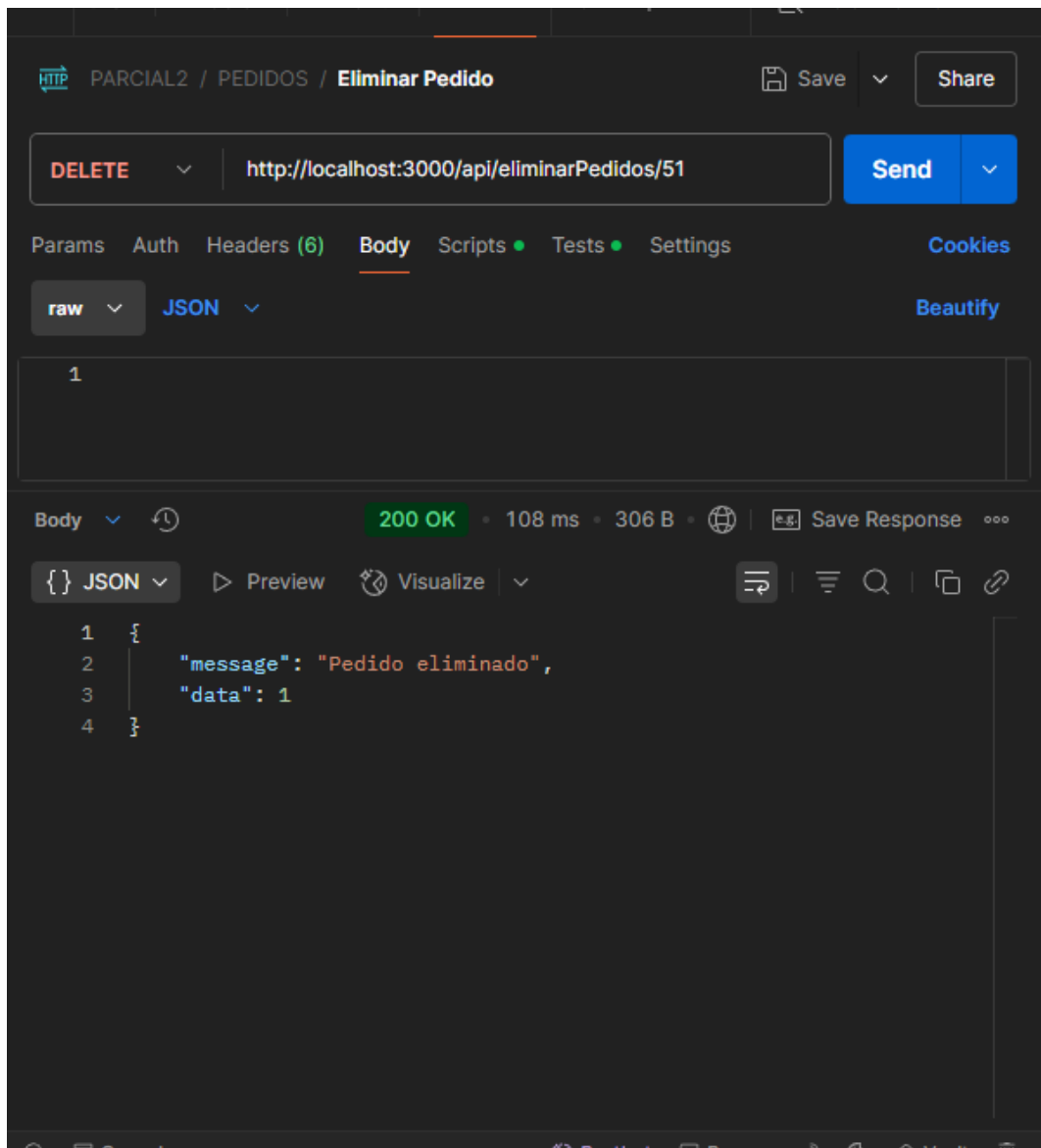
```

NOTA: API para actualizar un registro de la tabla pedido donde nos solicita que en la URL se ponga el id del pedido que se va a modificar y en el BODY solicita la fecha, la id del restaurante y el total a pagar para ejecutar la consulta de tipo UPDATE que se encuentra en el cuerpo de la API

```

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

```



NOTA: API para eliminar registros de la tabla pedido que solicita que en la URL se ponga la id del producto que se va a eliminar para que se pueda ejecutar la consulta de tipo DELETE que se encuentra en el cuerpo del API


```
//API DETALLES PEDIDO
app.post('/api/guardarDetallesPedidos', async (req, res) => {
  const { id_pedido, id_prod, cantidad, subtotal } = req.body;
  const query = 'INSERT INTO detalle_pedido (id_pedido, id_prod, cantidad, subtotal) VALUES ($1, $2, $3, $4) RE
  try {
    const result = await client.query(query, [id_pedido, id_prod, cantidad, subtotal]);
    res.status(201).json({ message: "DetallePedido creado", data: result.rows[0] });
  } catch (error) {
    res.status(500).json({ message: 'Error creando detalle de pedido', error: error.message });
  }
});
```

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

- Method:** POST
- URL:** http://localhost:3000/api/guardarDetallesPedidos
- Body (raw):**

```
{
  "id_pedido": 50,
  "id_prod": 2,
  "cantidad": 3,
  "subtotal": 45000
}
```
- Status:** 201 Created (848 ms, 394 B)
- Response (JSON):**

```
{
  "message": "DetallePedido creado",
  "data": {
    "id_detalle": 81,
    "id_pedido": 50,
    "id_prod": 2,
    "cantidad": 3,
    "subtotal": "45000.00"
  }
}
```

NOTA: API para insertar un registro en la tabla detalle_pedido donde se solicita que en el cuerpo pongamos el id del pedido, id del producto, la cantidad del producto y el subtotal a pagar

```
app.get('/api/obtenerDetallesPedidos', async (req, res) => {
  try {
    const result = await client.query('SELECT * FROM detalle_pedido');
    res.status(200).json({ data: result.rows });
  } catch (error) {
    res.status(500).json({ message: 'Error al obtener detalles de pedidos', error: error.message });
  }
});
```

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

- Method:** GET
- URL:** http://localhost:3000/api/obtenerDetallesPedidos
- Status:** 200 OK
- Time:** 114 ms
- Size:** 4.09 KB
- Response Type:** JSON

The response body is a JSON array of objects, each representing a record from the `detalle_pedido` table. The first two records are shown in the image:

```
{
  "data": [
    {
      "id_detalle": 1,
      "id_pedido": 1,
      "id_prod": 1,
      "cantidad": 2,
      "subtotal": "17.00"
    },
    {
      "id_detalle": 2,
      "id_pedido": 1,
      "id_prod": 16,
      "cantidad": 1,
      "subtotal": "2.90"
    }
  ]
}
```

NOTA: API para visualizar los registros de la tabla `detalle_pedido` donde se realiza una consulta de tipo `SELECT * FROM` para mostrar los registros en un `.json`

```

app.put('/api/actualizarDetallesPedidos/:id', async (req, res) => {
  const { id } = req.params;
  const { id_pedido, id_prod, cantidad, subtotal } = req.body;
  const query = 'UPDATE detalle_pedido SET id_pedido = $1, id_prod = $2, cantidad = $3, subtotal = $4 WHERE id_'

  try {
    const result = await client.query(query, [id_pedido, id_prod, cantidad, subtotal, id]);
    res.status(200).json({ message: "DetallePedido actualizado", data: result.rowCount });
  } catch (error) {
    res.status(500).json({ message: 'Error actualizando detalle de pedido', error: error.message });
  }
});

```

The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/api/actualizarDetallesPedidos/1`. The request body is a JSON object with the following fields:

```

{
  "id_pedido": 10,
  "id_prod": 24,
  "cantidad": 5,
  "subtotal": 45000
}

```

The response is a 200 OK status with a response time of 133 ms and a size of 315 B. The response body is a JSON object with the following fields:

```

{
  "message": "DetallePedido actualizado",
  "data": 1
}

```

NOTA: API para realizar la actualización de un registro de la tabla `detalle_pedido` donde se solicita que en la URL se ponga la id del registro que se va a modificar y en el cuerpo se solicita la id del pedido, id del producto, la cantidad de productos y el subtotal a pagar

```
app.delete('/api/eliminarDetalleProducto/:id', async (req, res) => {
  const { id } = req.params;
  try {
    const result = await client.query('DELETE FROM detalle_pedido WHERE id_detalle = $1', [id]);
    res.status(200).json({ message: "DetallePedido eliminado", data: result.rowCount });
  } catch (error) {
    res.status(500).json({ message: 'Error eliminando detalle de pedido', error: error.message });
  }
});
```

PARCIAL2 / DEATALLE... / **Eliminar Detalles de Pedido** Save Share

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

Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Descript...	...	Bulk Edit
	Key	Value	Description		

Body ▼ ↺ 200 OK • 132 ms • 313 B • 🌐 Save Response ...

{} **JSON** ▼ ▶ Preview 🔍 Visualize ▼ 🔧 🔍 📄 🔗

```
1 {
2   "message": "DetallePedido eliminado",
3   "data": 1
4 }
```

Console Postbot Runner Vault

NOTA: API para la eliminacion de registros de la tabla detalle_pedido donde se solicita que en la URL se ponga la id del registro a eliminar para que se pueda ejecutar la consulta DELETE que se encuentra en el cuerpo de la API

```

app.get('/api/productosPorPedido/:id_pedido', async (req, res) => {
  const { id_pedido } = req.params;
  const query = `
    SELECT p.nombre, p.precio, dp.cantidad, dp.subtotal
    FROM detalle_pedido dp
    JOIN producto p ON dp.id_prod = p.id_prod
    WHERE dp.id_pedido = $1
  `;

  try {
    const result = await client.query(query, [id_pedido]);
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ message: 'Error obteniendo productos del pedido', error: error.message });
  }
});

```

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

- URL:** `http://localhost:3000/api/productosPorPedido/2`
- Method:** GET
- Status:** 200 OK
- Response Time:** 110 ms
- Response Size:** 497 B
- Response Body (JSON):**

```

[
  {
    "nombre": "Pizza Margarita",
    "precio": "10.25",
    "cantidad": 1,
    "subtotal": "9.75"
  },
  {
    "nombre": "Ensalada César",
    "precio": "7.20",
    "cantidad": 2,
    "subtotal": "14.40"
  },
  {
    "nombre": "Café Expreso",
    "precio": "1.80",
    "cantidad": 1,
    "subtotal": "1.80"
  }
]

```

NOTA: API para filtrar los productos que hay en un pedido que se hace utilizando la referencia que tiene el detalle del pedido con el id del pedido y el id del producto mediante una consulta de tipo JOIN para que en el archivo .JSON se muestre el nombre del producto, el precio del producto, la cantidad de productos y el subtotal a pagar esto por el id del pedido. En la URL se solicita el id del pedido para mostrar los productos

```

app.get('/api/productosMasVendidos', async (req, res) => {
  const { limite } = req.query;
  let query = `
    SELECT p.nombre, SUM(dp.cantidad) AS total_vendido
    FROM detalle_pedido dp
    JOIN producto p ON dp.id_prod = p.id_prod
    GROUP BY p.nombre
    ORDER BY total_vendido DESC
  `;
  if (limite && !isNaN(limite)) {
    query += ` LIMIT ${parseInt(limite)} `;
  }
  try {
    const result = await client.query(query);
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ message: 'Error obteniendo productos más vendidos', error: error.message });
  }
});

```

GET <http://localhost:3000/api/productosMasVendidos> Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results (1/1) 200 OK • 139 ms • 1.16 KB Save Response

{ } JSON Preview Visualize

```

1 [
2   {
3     "nombre": "Taco Mexicano",
4     "total_vendido": 7
5   },
6   {
7     "nombre": "Pescado Frito",
8     "total_vendido": 6
9   },
10  {
11    "nombre": "Ensalada César",
12    "total_vendido": 6
13  },
14  {
15    "nombre": "Filete de Ternera",
16    "total_vendido": 6
17  },
18 ]

```

Postbot Runner Start Proxy Cookies Vault Trash

NOTA: API para visualizar los productos mas vendidos donde se mostraran todos los productos y la cantidad de ventas que han tenido utilizando la relación de las tablas detalle_pedido y producto y estas se ordenan de forma descendiente poniendo el mayor numero de ventas en la parte superior y el producto con menor cantidad en la parti inferior. En la URL no se solicita ningún dato en específico

```

app.get('/api/ventasPorRestaurantes', async (req, res) => {
  const 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
    ORDER BY total_ventas DESC
  `;

  try {
    const result = await client.query(query);
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ message: 'Error obteniendo ventas por restaurante', error: error.message });
  }
});

```

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

- Method:** GET
- URL:** http://localhost:3000/api/ventasPorRestaurantes
- Status:** 200 OK
- Response Time:** 97 ms
- Response Size:** 2.46 KB
- Response Format:** JSON

The response body contains a JSON array of restaurant sales data, ordered by total sales in descending order:

```

[
  {
    "restaurante": "La Cazuela",
    "total_ventas": "75026.50"
  },
  {
    "restaurante": "Gusto Italiano",
    "total_ventas": "98.10"
  },
  {
    "restaurante": "El Rincón Gourmet",
    "total_ventas": "89.50"
  },
  {
    "restaurante": "Tierra y Mar",
    "total_ventas": "82.10"
  }
]

```

NOTA: API para visualizar la cantidad de ventas que tiene el restaurante por ganancia esto por medio de la tabla pedido y utilizando su relación con la tabla restaurante para mostrar la cantidad de ganancias que ha tenido cada uno esto se muestra de forma descendiente de el restaurante que mas ganancias tubo al restaurante que menos ganancias tubo. En la URL no se solicita ningún dato en especial

```

app.get('/api/pedidosPorFecha/:fecha', async (req, res) => {
  const { fecha } = req.params;
  const query = `
    SELECT pedido.id_pedido, pedido.fecha, pedido.total, restaurante.nombre AS nombre_restaurante
    FROM pedido
    JOIN restaurante ON pedido.id_rest = restaurante.id_rest
    WHERE pedido.fecha = $1
  `;

  try {
    const result = await client.query(query, [fecha]);
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ message: 'Error obteniendo pedidos por fecha', error: error.message });
  }
});

```

PARCIAL21 / CONSULTAS NATIVAS / Pedidos Por Fecha

GET http://localhost:3000/api/pedidosPorFecha/2024-01-15

Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results (1/1) 200 OK • 129 ms • 377 B Save Response

JSON Preview Visualize

```

1 [
2   {
3     "id_pedido": 6,
4     "fecha": "2024-01-15T05:00:00.000Z",
5     "total": "34.70",
6     "nombre_restaurante": "Sabores del Mundo"
7   }
8 ]

```

NOTA: API para buscar los pedidos realizaron en cierta fecha esto mediante el WHERE fecha y el dato que toma en la URL que seria la fecha que se seleccione, en el body se mostrara el id del pedido, la fecha, el restaurante este ultimo utilizando la relación de la tabla pedidos con la tabla restaurante para llamar el nombre del restaurante con el id que esta en la tabla. Como mencione al inicio en la URL se solicita la fecha del pedido

```

app.get('/api/empleadosPorRol/:id_rest/:rol', async (req, res) => {
  const { id_rest, rol } = req.params;
  const query = `
    SELECT empleado.id_empleado, empleado.nombre, empleado.rol, restaurante.nombre AS nombre_restaurante
    FROM empleado
    JOIN restaurante ON empleado.id_rest = restaurante.id_rest
    WHERE empleado.id_rest = $1 AND empleado.rol = $2
  `;

  try {
    const result = await client.query(query, [id_rest, rol]);
    res.status(200).json({ success: true, data: result.rows });
  } catch (error) {
    res.status(500).json({ success: false, message: 'Error obteniendo empleados por rol', error: error.message });
  }
});

```


GET http://localhost:3000/api/empleadosPorRol/2/Cocinero Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results (1/1) 200 OK · 442 ms · 386 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "success": true,
3   "data": [
4     {
5       "id_empleado": 4,
6       "nombre": "Carla Méndez",
7       "rol": "Cocinero",
8       "nombre_restaurante": "El Fogón"
9     }
10  ]
11 }
```

```
1 SELECT * FROM empleado
2 ;
```

Results Chart Export Source Primary Database Role postgres Run CTRL ↵

id_empleado	nombre	rol	id_rest
2	Ana Torres	Mesero	1
3	Luis Gómez	Administrador	2
4	Carla Méndez	Cocinero	2
5	Jorge Ruiz	Mesero	3
6	Sandra López	Cajero	3
7	Pedro Álvarez	Cocinero	4
8	Marta Salas	Mesero	4
9	David Ríos	Administrador	5
11	Andrés Villa	Cocinero	6
12	Clara Nieto	Cajero	6

NOTA: API para filtrar los empleados de un restaurante con el rol que tiene esto haciendo una consulta de SELECT de la tabla empleados donde se utiliza la relación de la tabla de empleado y la tabla restaurante para llamar el nombre del restaurante, y mostrar en el body el id del empleado, nombre del empleado, rol del empleado y el nombre del restaurante. En la URL se solicita la id del restaurante y el rol.

