

Parcial II

Cristian Camilo Tique Tapiero

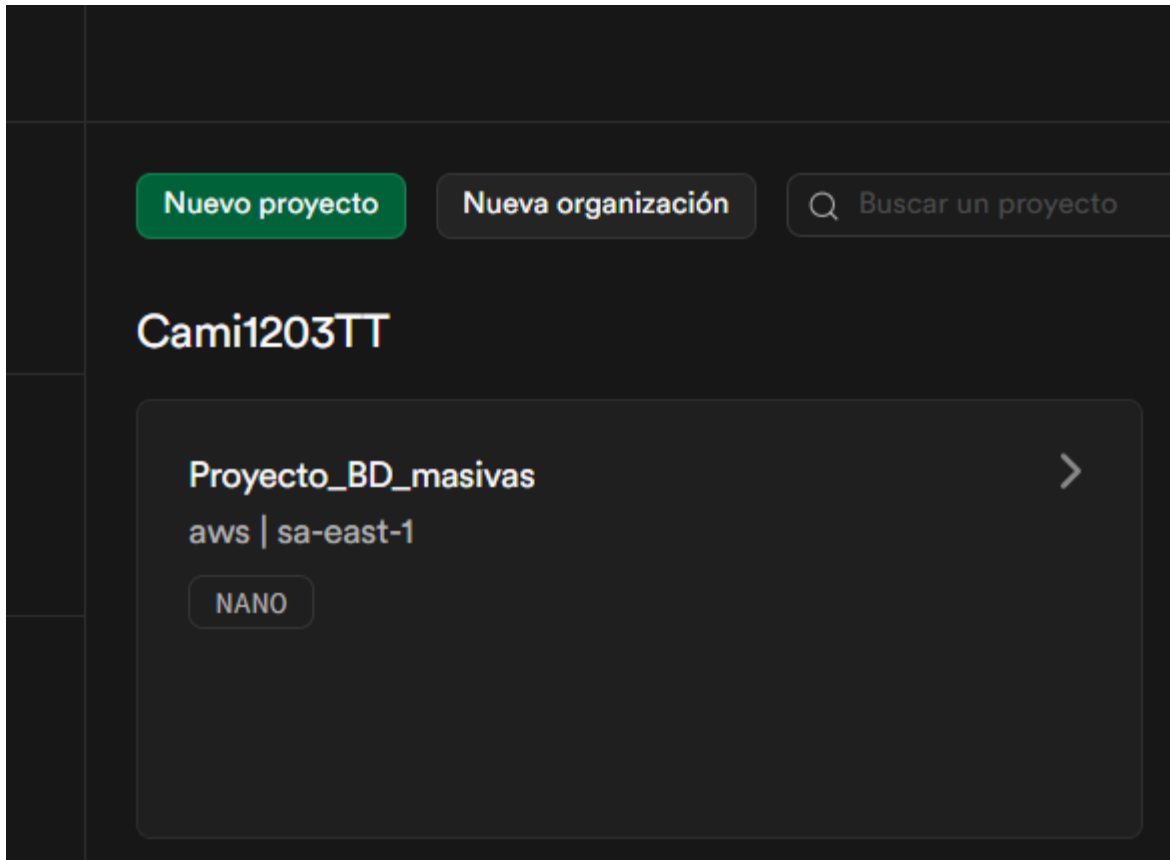
Ingeniería de sistemas, Corporación Universitaria Minuto de Dios

60747: Bases de Datos Masivas

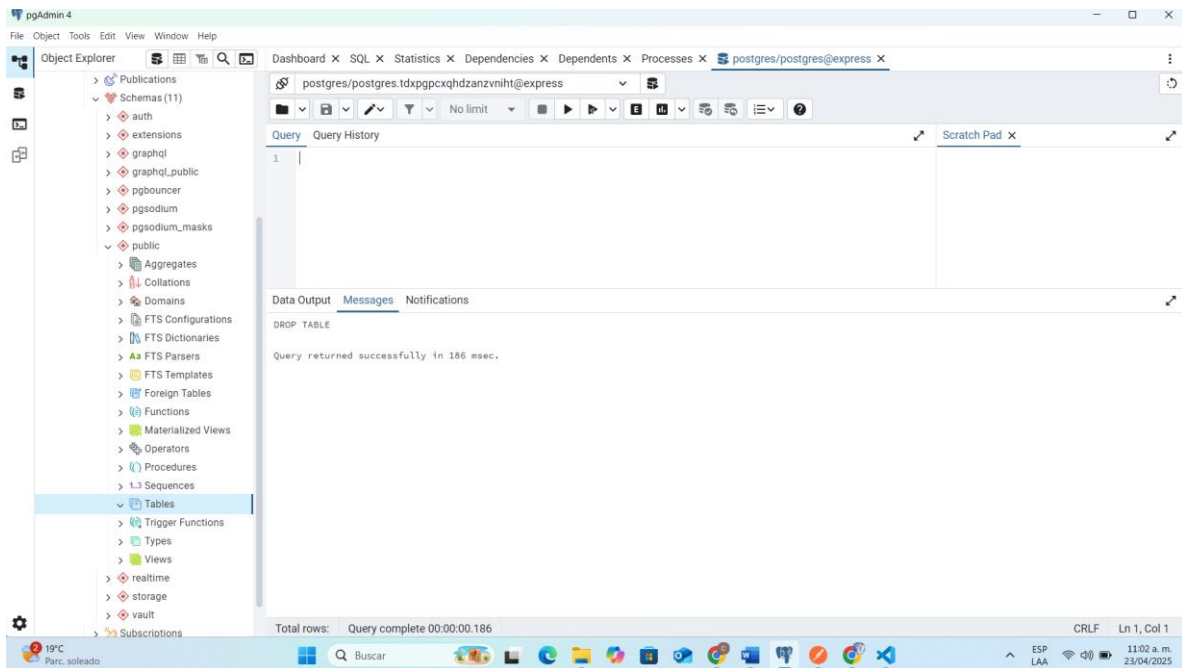
Prof. William Alexander Matallana Parra

25 de abril de 2025

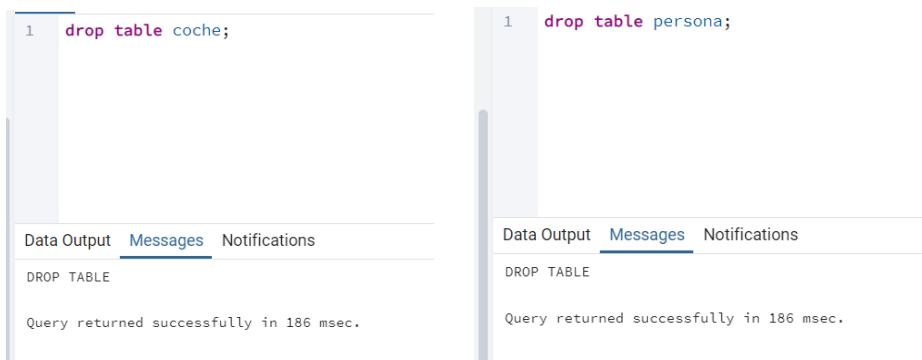
Hago la conexión de pgadmin 4 con supabase. En mi caso, utilizare el mismo proyecto que se trabajo en la clase anterior, debido a que supabase solo permite dos proyectos en el modo gratuito



- Conexión con pgadmin lista:



Como en la anterior clase se había creado una tabla persona, proceso a eliminarla



El siguiente paso, será crear las tablas desde la consola de pgadmin4 para no hacerlo desde Supabase:

Para abrir la consola, se expande las siguientes opciones: schemas y public, en la opción public se le da clic derecho y se toma la opción que dice Query tool

- Create >
- Delete
- Delete (Cascade)
- Refresh...
- Restore...
- Backup...
- CREATE Script
- ERD For Schema
- Maintenance...
- Grant Wizard...
- Search Objects...
- PSQL Tool
- Query Tool
- Properties...

Tabla Restaurante

```
CREATE TABLE Restaurante (  
    id_rest INT PRIMARY KEY,  
    nombre VARCHAR(100),  
    ciudad VARCHAR(100),  
    direccion VARCHAR(150),  
    fecha_apertura DATE  
);
```

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);

Data OutputMessagesNotifications

CREATE TABLE

Query returned successfully in 505 msec.

✓ Query returned successfully in 505 msec. ✕

Reviso que la tabla aparezca en las tablas:



Tabla Empleado

```
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)  
);
```

```
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)  
);
```

Verifico que la tabla esta:

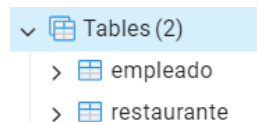


Tabla Producto

```
CREATE TABLE Producto (
```

```
id_prod INT PRIMARY KEY,  
nombre VARCHAR(100),  
precio NUMERIC(10,2)  
);
```

```
1 CREATE TABLE Producto (  
2     id_prod INT PRIMARY KEY,  
3     nombre VARCHAR(100),  
4     precio NUMERIC(10,2)  
5 );  
6
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 236 msec.

Tables (3)

- empleado
- producto
- restaurante

Tabla Pedido

```
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)  
);
```

```


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  );
8

```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 200 msec.

✓  Tables (4)





- >  empleado
- >  pedido
- >  producto
- >  restaurante

Tabla DetallePedido

```

CREATE TABLE DetallePedido (
    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)
);







```

```
1  CREATE TABLE DetallePedido (  
2      id_detalle INT PRIMARY KEY,  
3      id_pedido INT,  
4      id_prod INT,  
5      cantidad INT,  
6      subtotal NUMERIC(10,2),  
7      FOREIGN KEY (id_pedido) REFERENCES Pedido(id_pedido),  
8      FOREIGN KEY (id_prod) REFERENCES Producto(id_prod)  
9  );  
10
```

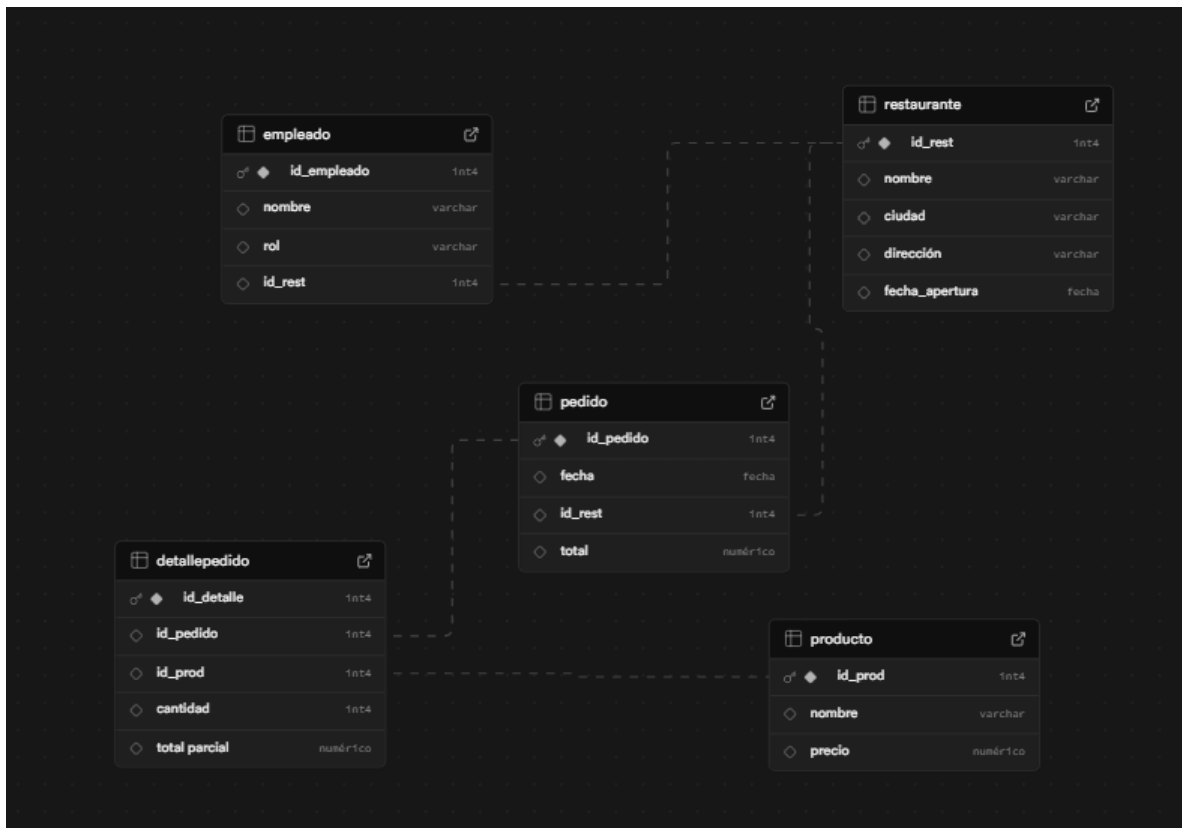
Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 296 msec.

- ▼  Tables (5)
- >  detallepedido
 - >  empleado
 - >  pedido
 - >  producto
 - >  restaurante

Si vamos a Supabase, ingresamos al proyecto, y de las opciones de lado izquierdo, seleccionamos “bases de datos” y después la opción “Visualizador de esquemas”. Allí podremos ver las tablas



La imagen anterior es el modelo que me da Supabase

Ahora, voy a insertar registros para cada tabla, con el fin de darle utilidad y funcionamiento las consultas nativas que se harán próximamente. (la inserción de registros de hizo desde pgadmin4, por su consola)

Tabla restaurante

Query	Query History
<pre> 1 ✓ INSERT INTO Restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES 2 (1, 'El Buen Sabor', 'Bogotá', 'Calle 123 #45-67', '2021-01-10'), 3 (2, 'Sazón del Valle', 'Cali', 'Carrera 30 #12-34', '2020-06-15'), 4 (3, 'Comida Criolla', 'Medellín', 'Av. 80 #45-12', '2019-09-01'), 5 (4, 'Mar y Tierra', 'Cartagena', 'Calle del Mar #7-89', '2022-03-22'), 6 (5, 'Delicias Andinas', 'Pasto', 'Calle 10 #3-21', '2021-11-11'), 7 (6, 'La Arepa Viajera', 'Bucaramanga', 'Calle 60 #23-40', '2020-02-02'), 8 (7, 'Tamal Express', 'Ibagué', 'Calle 19 #5-17', '2022-08-18'), 9 (8, 'Ajiaco Real', 'Bogotá', 'Cra 7 #84-12', '2023-01-01'), 10 (9, 'Sabor del Caribe', 'Santa Marta', 'Cra 4 #18-90', '2018-12-01'), 11 (10, 'Gastro Fusión', 'Pereira', 'Calle 14 #16-10', '2021-07-07'), 12 (11, 'La Cocina de Mama', 'Manizales', 'Calle 22 #5-66', '2020-05-05'), 13 (12, 'Chorizo Capital', 'Bogotá', 'Cra 10 #26-40', '2019-03-03'), </pre>	
Data Output	Messages
INSERT 0 20	
Query returned successfully in 172 msec.	

Tabla empleado

<pre> 1 ✓ INSERT INTO Empleado (id_empleado, nombre, rol, id_rest) VALUES 2 (1, 'Carlos Pérez', 'Chef', 1), 3 (2, 'Laura Gómez', 'Mesero', 1), 4 (3, 'Jorge Ramírez', 'Cajero', 2), 5 (4, 'Ana Torres', 'Chef', 2), 6 (5, 'Luis Morales', 'Chef', 3), 7 (6, 'María López', 'Mesero', 3), 8 (7, 'Pedro Díaz', 'Cajero', 4), 9 (8, 'Sara Castillo', 'Mesero', 4), 10 (9, 'David Ruiz', 'Chef', 5), 11 (10, 'Lucía Vargas', 'Mesero', 5), 12 (11, 'Tomás Herrera', 'Chef', 6), 13 (12, 'Verónica Mendoza', 'Cajero', 6), </pre>	
Data Output	Messages
INSERT 0 20	
Query returned successfully in 267 msec.	

Tabla producto

Query	Query History
1	▼ INSERT INTO Producto (id_prod, nombre, precio) VALUES
2	(1, 'Hamburguesa', 12000),
3	(2, 'Perro Caliente', 10000),
4	(3, 'Salchipapas', 8000),
5	(4, 'Arepa Rellena', 9000),
6	(5, 'Tamal', 7000),
7	(6, 'Empanada', 2000),
8	(7, 'Ajiaco', 14000),
9	(8, 'Cazuela de Mariscos', 18000),
10	(9, 'Pollo Broaster', 13000),
11	(10, 'Sopa del Día', 6000),
12	(11, 'Churrasco', 20000),
13	(12, 'Posta Negra', 19000),
14	(13, 'Posta Blanca', 7500),

Data Output	Messages	Notifications
INSERT 0 20		
Query returned successfully in 216 msec.		

Tabla pedido

Query	Query History
9	(8, '2024-04-06', 8, 30000),
10	(9, '2024-04-07', 9, 24000),
11	(10, '2024-04-08', 10, 36000),
12	(11, '2024-04-01', 1, 11000),
13	(12, '2024-04-02', 1, 20000),
14	(13, '2024-04-04', 2, 12000),
15	(14, '2024-04-06', 3, 14500),
16	(15, '2024-04-07', 4, 8000),
17	(16, '2024-04-08', 5, 9000),
18	(17, '2024-04-09', 6, 27000),
19	(18, '2024-04-09', 7, 17000),
20	(19, '2024-04-10', 8, 25000),
21	(20, '2024-04-10', 9, 19000);
22	

Data Output	Messages	Notifications
INSERT 0 20		
Query returned successfully in 196 msec.		

Tabla detalle pedido


```

PS C:\Users\trcri\OneDrive\Escritorio\Parcial 2> npm init -y
Wrote to C:\Users\trcri\OneDrive\Escritorio\Parcial 2\package.json:

{
  "name": "parcial-2",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}

PS C:\Users\trcri\OneDrive\Escritorio\Parcial 2>

```

- Si todo ha sido exitoso, debería de aparecer un archivo llamado package.json dentro de la carpeta

```

▼ PARCIAL 2 [🔍] [📁] [🔄] [📄]
  > node_modules
    {} package-lock.json
    {} package.json

```

- Ahora se tiene que descargar las dependencias de postgres, para que sea posible trabajar sobre el proyecto. Para ello ejecutamos el siguiente comando “npm install express pg cors”. Al ejecutar deberá de aparecer que está cargando

```

PS C:\Users\trcri\OneDrive\Escritorio\Parcial 2> npm install express pg cors

```

Y después de cargar, si todo ha salido bien, deberá de aparecer el siguiente mensaje:

```

PS C:\Users\trcri\OneDrive\Escritorio\Parcial 2> npm install express pg cors

added 82 packages, and audited 83 packages in 4s

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

found 0 vulnerabilities
PS C:\Users\trcri\OneDrive\Escritorio\Parcial 2>

```

Esto confirma que fue exitoso el proceso

- Ahora, dentro de la carpeta vamos a agregar dos archivos nuevos con extensión .js

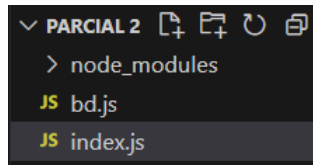
- El primero, se llamará bd.js (donde estará la conexión con supabase)

```

▼ PARCIAL 2 [🔍] [📁] [🔄] [📄]
  > node_modules
    JS bd.js

```

- El segundo, llevara como nombre index.js (donde se creará las APIS del CRUD)



6. Ahora, tenemos que agregarle líneas de código para que se pueda realizar la conexión y la construcción de las APIS

6.1. Archivo bd.js

Para este archivo se agrega el siguiente código, donde, en la primera parte, se hace la conexión con Supabase (estos datos se obtienen desde el proyecto de SUPABASE, como, en mi caso, estoy trabajando sobre el mismo proyecto del ejercicio anterior, la conexión será la misma, para este nuevo proyecto)

```
JS bd.js  X  JS index.js
JS bd.js > ...
3 // Datos de conexión con Supabase
4 const client = new Client({
5   host: 'aws-0-sa-east-1.pooler.supabase.com',
6   port: 5432,
7   user: 'postgres.tdxpgpcxqhdzanzvniht',
8   password: 'k7uARMxtf0qyJ9m3',
9   database: 'postgres',
10  ssl: {
11    rejectUnauthorized: false
12  }
13 });
14 // mensajes los cuales me arrojaran en caso de que haya un error, o, por lo contrario, se hizo la conexión exitosamente
15 client.connect((error) => {
16   if (error) {
17     console.log('Error conectando con la base de datos:', error);
18     return;
19   } else {
20     console.log('Conectado con la base de datos de Supabase');
21   }
22 });
```

Este archivo se encarga de establecer la conexión entre mi proyecto en Node.js y la base de datos PostgreSQL que tengo alojada en Supabase. Para lograrlo, utilizo el paquete pg para conectarme a bases de datos PostgreSQL desde Node.js.

```
const client = new Client({
  host: 'aws-0-sa-east-1.pooler.supabase.com',
  port: 5432,
  user: 'postgres.tdxpgpcxqhdzanzvniht',
  password: 'k7uARMxtf0qyJ9m3',
  database: 'postgres',
  ssl: {
    rejectUnauthorized: false
  }
});
```

De la imagen anterior: lo que estoy haciendo es crear un nuevo cliente “client” configurándolo con los datos de conexión que me da Supabase:

- ✓ host: es la dirección del servidor donde está la base de datos.
- ✓ port: es el puerto por donde se conecta (el 5432 es el estándar de PostgreSQL).
- ✓ user: es el usuario de la base de datos.
- ✓ password: es la clave de ese usuario.
- ✓ database: en este caso, la base de datos por defecto de Supabase se llama postgres.
- ✓ ssl: se usa para hacer la conexión segura. Puse rejectUnauthorized: false para evitar problemas de certificados.

```
client.connect((error) => {  
  if (error) {  
    console.log('Error conectando con la base de datos:', error);  
    return;  
  } else {  
    console.log('Conectado con la base de datos de Supabase');  
  }  
});  
(alias) const export=: Client  
import export=  
module.exports = client;
```

De la imagen anterior: intento conectar el cliente con la base de datos. Si ocurre algún error, se muestra en la consola con un mensaje de, si la conexión es exitosa, muestra un mensaje confirmando que se estableció correctamente.

Por otro lado, exporto el objeto client para poder usar esta conexión en otros archivos del proyecto, como en los controladores o rutas.

6.2 index.js

Dentro de este archivo se crea la relación con a base de datos y las APIS. En dicho ejercicio, se hace la construcción del CRUD (crear, consultar, actualizar y eliminar) por cada tabla. Siendo las respectivas consultas para cada una.


```
const express = require('express');
const cors = require('cors');
const client = require('./db');

const app = express();
const PORT = 3000;

app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

express es el framework que permite crear rutas HTTP.

cors permite que la API sea accedida desde cualquier cliente.

client es la conexión a la base de datos PostgreSQL que tengo en Supabase (configurada en el archivo db.js).

Restaurante

```
try {
  await client.query(
    'INSERT INTO Restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES ($1, $2, $3, $4, $5)',
    [id_rest, nombre, ciudad, direccion, fecha_apertura]
  );
  res.status(201).json({ success: true, message: 'Restaurante creado' });
} catch (error) {
  res.status(500).json({ success: false, error: error.message });
}
});

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

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

app.delete('/api/restaurante/:id', async (req, res) => {
  const { id } = req.params;
```

- /api/restaurante (POST): Inserta un nuevo restaurante en la base de datos.
- /api/restaurante (GET): Obtiene todos los restaurantes.
- /api/restaurante/:id (PUT): Actualiza un restaurante por ID.

- /api/restaurante/:id (DELETE): Elimina un restaurante por ID.

Empleado

```
app.post('/api/empleado', async (req, res) => {
  const { id_empleado, nombre, rol, id_rest } = req.body;
  try {
    await client.query(
      'INSERT INTO Empleado (id_empleado, nombre, rol, id_rest) VALUES ($1, $2, $3, $4)',
      [id_empleado, nombre, rol, id_rest]
    );
    res.status(201).json({ success: true, message: 'Empleado creado' });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

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

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

app.delete('/api/empleado/:id', async (req, res) => {
```

- /api/empleado (POST): Crea un nuevo empleado asociado a un restaurante.
- /api/empleado (GET): Devuelve todos los empleados.
- /api/empleado/:id (PUT): Modifica un empleado por su ID.
- /api/empleado/:id (DELETE): Borra un empleado por ID.

Producto

```

app.post('/api/producto', async (req, res) => {
  try {
    await client.query(
      'INSERT INTO Producto (id_prod, nombre, precio) VALUES ($1, $2, $3)',
      [id_prod, nombre, precio]
    );
    res.status(201).json({ success: true, message: 'Producto creado' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

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

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

app.delete('/api/producto/:id', async (req, res) => {
  const { id } = req.params;

```

- /api/producto (POST): Registra un nuevo producto.
- /api/producto (GET): Lista todos los productos disponibles.
- /api/producto/:id (PUT): Actualiza los datos de un producto.
- /api/producto/:id (DELETE): Elimina un producto de la base de datos.

Pedido

```

app.post('/api/pedido', async (req, res) => {
  try {
    await client.query(
      'INSERT INTO Pedido (id_pedido, fecha, id_rest, total) VALUES ($1, $2, $3, $4)',
      [id_pedido, fecha, id_rest, total]
    );
    res.status(201).json({ success: true, message: 'Pedido creado' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

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

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

app.delete('/api/pedido/:id', async (req, res) => {
  const { id } = req.params;

```

- /api/pedido (POST): Crea un nuevo pedido.
- /api/pedido (GET): Muestra todos los pedidos registrados.
- /api/pedido/:id (PUT): Edita un pedido según su ID.
- /api/pedido/:id (DELETE): Elimina un pedido por su ID.

DetallePedido

```

app.post('/api/detalle_pedido', async (req, res) => {
  const { id_detalle, id_pedido, id_prod, cantidad, subtotal } = req.body;
  try {
    await client.query(
      'INSERT INTO DetallePedido (id_detalle, id_pedido, id_prod, cantidad, subtotal) VALUES ($1, $2, $3, $4, $5)',
      [id_detalle, id_pedido, id_prod, cantidad, subtotal]
    );
    res.status(201).json({ success: true, message: 'Detalle de pedido creado' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

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

app.put('/api/detalle_pedido/:id', async (req, res) => {
  const { id } = req.params;
  const { id_pedido, id_prod, cantidad, subtotal } = req.body;
  try {
    await client.query(
      'UPDATE DetallePedido SET id_pedido=$1, id_prod=$2, cantidad=$3, subtotal=$4 WHERE id_detalle=$5',
      [id_pedido, id_prod, cantidad, subtotal, id]
    );
    res.json({ success: true, message: 'Detalle de pedido actualizado' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

app.delete('/api/detalle_pedido/:id', async (req, res) => {

```

- /api/detalle_pedido (POST): Agrega productos a un pedido.
 - /api/detalle_pedido (GET): Muestra todos los detalles de todos los pedidos.
 - /api/detalle_pedido/:id (PUT): Modifica un detalle específico.
 - /api/detalle_pedido/:id (DELETE): Elimina un detalle de un pedido.
7. El siguiente paso es construir las consultas desde Postman. Donde se crear una carpeta para cada tabla, y dentro de cada tabla se crea las cuatro peticiones (POST, GET, PUT, DELETE) de acuerdo a la consulta que se requiera.

Lo primero es crear la conexión de cada tabla:

Tabla Restaurante:

▼	Restaurante
	POST Crear
	GET Consultar
	PUT Actualizar
	DEL Eliminar

Crear

POST

http://localhost:3000/api/restaurante

Params

Authorization

Headers (9)

Body

Scripts

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

```
1 {
2   "id_rest": 21,
3   "nombre": "Hamburguesas Deli",
4   "ciudad": "Medellin",
5   "direccion": "Calle 123 #45-67",
6   "fecha_apertura": "2023-05-02"
7 }
8
```

Body

Cookies

Headers (8)

Test Results

201 Created • 228 ms • 319 B •

{ } JSON

Preview

Visualize

```
1 {
2   "success": true,
3   "message": "Restaurante creado"
4 }
```

Me creo con éxito el registro 21

Consultar

GET http://localhost:3000/api/restaurante

Params Authorization Headers (7) Body Scripts Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results 200 OK • 186 ms • 3.04 KB Save F

{ } JSON Preview Visualize

```
129     "id_rest": 19,
130     "nombre": "Panadería El Punto",
131     "ciudad": "Bogotá",
132     "direccion": "Cra 11 #65-90",
133     "fecha_apertura": "2018-08-08T05:00:00.000Z"
134   },
135   {
136     "id_rest": 20,
137     "nombre": "La Esquina del Pollo",
138     "ciudad": "Villavicencio",
139     "direccion": "Calle 25 #6-44",
140     "fecha_apertura": "2022-12-24T05:00:00.000Z"
141   },
142   {
143     "id_rest": 21,
144     "nombre": "Hamburguesas Deli",
145     "ciudad": "Medellin",
146     "direccion": "Calle 123 #45-67",
147     "fecha_apertura": "2023-05-02T05:00:00.000Z"
148   }
149 ]
```

Postbot Runner Start Proxy Cookies

Me muestra todos los registros de la tabla. De ultimo registro está el 21.

Actualizar

Voy a actualizar el 21 y le cambiare la ciudad por Bogotá:

PUT ⌵ http://localhost:3000/api/restaurante/21

Params Authorization Headers (9) **Body** ● Scripts Tests Settings

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

```
1 {  
2   "id_rest": 21,  
3   "nombre": "Hamburguesas Deli",  
4   "ciudad": "Bogota",  
5   "direccion": "Calle 123 #45-67",  
6   "fecha_apertura": "2023-05-02T05:00:00.000Z"  
7 }  
8
```

Body Cookies Headers (8) Test Results ↺ **200 OK** • 1

{} JSON ⌵ ▶ Preview 🔗 Visualize ⌵

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

Si hago nuevamente la consulta, el cambio debió hacerse efectuado.

GET

http://localhost:3000/api/restaurante

ParamsAuthorizationHeaders (7)BodyScriptsTestsSettings

Query Params

Key	Value
Key	Value

BodyCookiesHeaders (8)Test Results

{ } JSON

PreviewVisualize

```
129      "id_rest": 19,
130      "nombre": "Panadería El Punto",
131      "ciudad": "Bogotá",
132      "direccion": "Cra 11 #65-90",
133      "fecha_apertura": "2018-08-08T05:00:00.000Z"
134    },
135
136      "id_rest": 20,
137      "nombre": "La Esquina del Pollo",
138      "ciudad": "Villavicencio",
139      "direccion": "Calle 25 #6-44",
140      "fecha_apertura": "2022-12-24T05:00:00.000Z"
141    },
142
143      "id_rest": 21,
144      "nombre": "Hamburguesas Deli",
145      "ciudad": "Bogota",
146      "direccion": "Calle 123 #45-67",
147      "fecha_apertura": "2023-05-02T05:00:00.000Z"
148    }
149  ]
```

Eliminar

En la url pongo el id del registro que quiero eliminar (en este caso el 21) después, consulto para asegurar la eliminación del registro.

DELETE
▼
http://localhost:3000/api/restaurante/21

Params
Authorization
Headers (7)
Body
Scripts
Tests

Query Params

	Key	Value
	Key	Value

Body
Cookies
Headers (8)
Test Results
↺

{}
JSON
▼
▶
Preview
🔗
Visualize
▼

```

1  {
2    "success": true,
3    "message": "Restaurante eliminado"
4  }

```

Consulta:

GET
▼
http://localhost:3000/api/restaurante

Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings

Query Params

	Key	Value
	Key	Value

Body
Cookies
Headers (8)
Test Results
↺

{}
JSON
▼
▶
Preview
🔗
Visualize
▼

```

122  "id_rest": 18,
123  "nombre": "Sándwich y Más",
124  "ciudad": "Armenia",
125  "direccion": "Cra 14 #21-30",
126  "fecha_apertura": "2020-04-04T05:00:00.000Z"
127  },
128  {
129    "id_rest": 19,
130    "nombre": "Panadería El Punto",
131    "ciudad": "Bogotá",
132    "direccion": "Cra 11 #65-90",
133    "fecha_apertura": "2018-08-08T05:00:00.000Z"
134  },
135  {
136    "id_rest": 20,
137    "nombre": "La Esquina del Pollo",
138    "ciudad": "Villavicencio",
139    "direccion": "Calle 25 #6-44",
140    "fecha_apertura": "2022-12-24T05:00:00.000Z"
141  }
142  ]

```

200

Estas pruebas, tal cual, como el primer ejemplo, serán similares:

Tabla Empleado

Crear

POST

http://localhost:3000/api/empleado

ParamsAuthorizationHeaders (9)BodyScriptsTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

{

2

"id_empleado": 22,

3

"nombre": "Camilo Tique",

4

"rol": "Camarero",

5

"id_rest": 2

6

}

7

BodyCookiesHeaders (8)Test Results201 Created

{}

JSON

Preview

Visualize

1

{

2

"success": true,

3

"message": "Empleado creado"

4

}

Consultar

GET

http://localhost:3000/api/empleado

ParamsAuthorizationHeaders (7)BodyScriptsTestsSettings

Query Params

	Key	Value
	Key	Value

BodyCookiesHeaders (8)Test Results

{}

JSON

Preview

Visualize

114

"id_rest": 10

115

},

116

{

117

"id_empleado": 20,

118

"nombre": "Héctor Salazar",

119

"rol": "Mesero",

120

"id_rest": 10

121

},

122

{

123

"id_empleado": 21,

124

"nombre": "Juan Perez",

125

"rol": "Chef",

126

"id_rest": 1

127

},

128

{

129

"id_empleado": 22,

130

"nombre": "Camilo Tique",

131

"rol": "Camarero",

132

"id_rest": 2

133

}

134

]

Actualizar

PUT

▼

http://localhost:3000/api/empleado/21

ParamsAuthorizationHeaders (9)Body●ScriptsTestsSettings

☐ none☐ form-data☐ x-www-form-urlencoded☒ raw☐ binary☐ GraphQLJSON▼

```
1  {
2    "id_empleado": 22,
3    "nombre": "Cristian Tique",
4    "rol": "Chef",
5    "id_rest": 1
6  }
```

BodyCookiesHeaders (8)Test Results↻

200 OK

{ }JSON▼▶ Preview🔗 Visualize▼

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

```
127  },
128  {
129    "id_empleado": 21,
130    "nombre": "Cristian Tique",
131    "rol": "Chef",
132    "id_rest": 1
133  }
134 ]
```

Eliminar

DELETE



http://localhost:3000/api/empleado/21

Params

Authorization

Headers (7)

Body

Scripts

Tests

Query Params

	Key	Value
	Key	Value

Body

Cookies

Headers (8)

Test Results



{ } JSON



Preview



Visualize



```
1  {
2    "success": true,
3    "message": "Empleado eliminado"
4  }
```

```
102 |     "id_rest": 9
103 |   },
104 |   {
105 |     "id_empleado": 18,
106 |     "nombre": "Iván Rojas",
107 |     "rol": "Chef",
108 |     "id_rest": 9
109 |   },
110 |   {
111 |     "id_empleado": 19,
112 |     "nombre": "Natalia Franco",
113 |     "rol": "Cajero",
114 |     "id_rest": 10
115 |   },
116 |   {
117 |     "id_empleado": 20,
118 |     "nombre": "Héctor Salazar",
119 |     "rol": "Mesero",
120 |     "id_rest": 10
121 |   }
122 | }
```

Postbot Runner Start Proxy Cookies Vault Trash

Tabla producto

Crear

POST

http://localhost:3000/api/producto

Params

Authorization

Headers (9)

Body

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

{

2

"id_prod": 21,

3

"nombre": "Combo X3 Hamburguesa",

4

"precio": 54000

5

}

6

Body

Cookies

Headers (8)

Test Results

201 Created

• 227 ms • 316 B

{}

JSON

Preview

Visualize

1

{

2

"success": true,

3

"message": "Producto creado"

4

}

Consultar

GET

http://localhost:3000/api/producto

Send

Params

Authorization

Headers (7)

Body

Scripts

Tests

Settings

Cookies

Query Params

	Key	Value	Description		Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (8)

Test Results

200 OK

• 136 ms • 1.44 KB •

Save Response

{}

JSON

Preview

Visualize

96

},

97

{

98

"id_prod": 20,

99

"nombre": "Caf ",

100

"precio": "2500.00"

101

},

102

{

103

"id_prod": 21,

104

"nombre": "Combo X3 Hamburguesa",

105

"precio": "54000.00"

106

}

107

}

Postman

Runner

Start Proxy

Cookies

Vault

Trash

Actualizar

PUT

▼

http://localhost:3000/api/producto/1

ParamsAuthorizationHeaders (9)Body ●ScriptsTestsSettings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

J

1

{

2

"id_prod": 21,

3

"nombre": "Combo X4 Hamburguesa",

4

"precio": "76400.00"

5

}

6

7

8

BodyCookiesHeaders (8)Test Results

{ } JSON ▼

▶ Preview

🔗 Visualize ▼

1

{

2

"success": true,

3

"message": "Producto actualizado"

4

}

{

"id_prod": 21,

"nombre": "Combo X4 Hamburguesa",

"precio": "76400.00"

}

Eliminar

DELETE



http://localhost:3000/api/producto/21

Params

Authorization

Headers (7)

Body

Scripts

Tests

Settings

Query Params

	Key	Value
	Key	Value

Body

Cookies

Headers (8)

Test Results



{ } JSON



Preview



Visualize



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

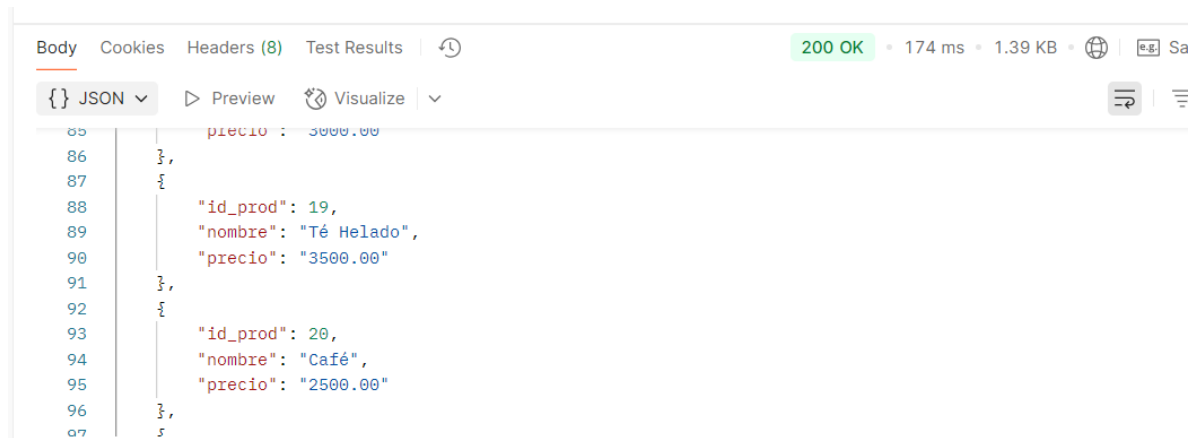
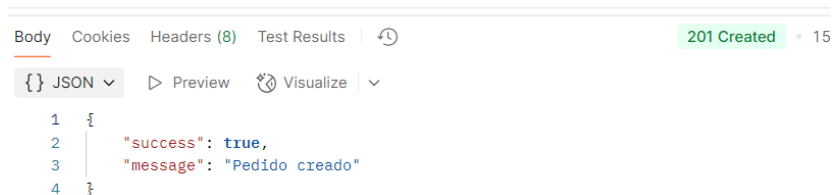
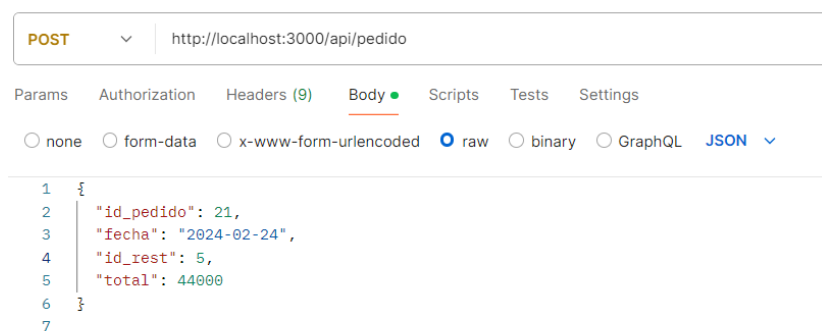



Table pedido

Crear



Consultar

GET

▼

http://localhost:3000/api/pedido

Send ▼

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (8) Test Results ↻

200 OK 161 ms 1.96 KB 🌐 📄 Save Response ⋮

{ } JSON ▼

▶ Preview 🔗 Visualize ▼

🔧 📄 🔍 📋 🔗

```
117     "id_pedido": 20,
118     "fecha": "2024-04-10T05:00:00.000Z",
119     "id_rest": 9,
120     "total": "19000.00"
121   },
122   {
123     "id_pedido": 21,
124     "fecha": "2024-02-24T05:00:00.000Z",
125     "id_rest": 5,
126     "total": "44000.00"
127   }
128 ]
```

Actualizar

PUT

▼

http://localhost:3000/api/pedido/21

ParamsAuthorizationHeaders (9)Body●ScriptsTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

▼

1

{

2

"id_pedido": 21,

3

"fecha": "2024-06-25T05:00:00.000Z",

4

"id_rest": 3,

5

"total": "20000.00"

6

}

7

BodyCookiesHeaders (8)Test Results↺

200 OK

{}

JSON

▼

▶ Preview

🔗 Visualize

▼

1

{

2

"success": true,

3

"message": "Pedido actualizado"

4

}

1

{

2

"id_pedido": 21,

3

"fecha": "2024-06-25T05:00:00.000Z",

4

"id_rest": 3,

5

"total": "20000.00"

6

}

7

}

PostbotRunnerStart Proxy

Eliminar

DELETE

▼

http://localhost:3000/api/pedido/21

ParamsAuthorizationHeaders (7)BodyScriptsTestsSettings

Query Params

	Key	Value	Description
	Key	Value	Description

BodyCookiesHeaders (8)Test Results

200 OK • 349 ms

{ } JSON ▼▶ Preview🔗 Visualize ▼

1 {

2 "success": true,

3 "message": "Pedido eliminado"

4 }

110 {

111 "id_pedido": 20,

112 "fecha": "2024-04-10T05:00:00.000Z",

113 "id_rest": 9,

114 "total": "19000.00"

115 },

116 {

⌘ Back⌘ Forward⌘ Stop⌘ Cookies⌘ Vault⌘ Trash

Tabla detalle pedido

Crear

POST

▼

http://localhost:3000/api/detalle_pedido

Params

Authorization

Headers (9)

Body ●

Scripts

Tests

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

```
1  {
2    "id_detalle": 21,
3    "id_pedido": 3,
4    "id_prod": 1,
5    "cantidad": 2,
6    "subtotal": 36000
7  }
8
```

Body

Cookies

Headers (8)

Test Results

↺

201 Created

{ } JSON ▼

▶ Preview

🔍 Visualize ▼

```
1  {
2    "success": true,
3    "message": "Detalle de pedido creado"
4  }
```

Consultar

GET

http://localhost:3000/api/detalle_pedido

Params

Authorization

Headers (7)

Body

Scripts

Tests

Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body

Cookies

Headers (8)

Test Results

🔄

200 OK

• 135 ms

• 1.88 KB

🌐

📄 Sav

{ } JSON

▶ Preview

🔗 Visualize

▼

⌵

⌵

138

139

140

141

142

143

144

145

146

147

148

149

]

"id_prod": 6,

"cantidad": 5,

"subtotal": "10000.00"

}

,

{

"id_detalle": 21,

"id_pedido": 3,

"id_prod": 1,

"cantidad": 2,

"subtotal": "36000.00"

}

🔗 Postbot

🏃 Runner

🌐 Start Proxy

🍪 Cookies

🔗

Actualizar

PUT http://localhost:3000/api/detalle_pedido/21

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

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

```
1 {
2   "id_detalle": 21,
3   "id_pedido": 4,
4   "id_prod": 2,
5   "cantidad": 10,
6   "subtotal": "137000.00"
7 }
8
9
10
```

Body Cookies Headers (8) Test Results 200 OK • 131 ms • 325 B

{ } JSON ▾ ▶ Preview Visualize ▾

```
1 {
2   "success": true,
3   "message": "Detalle de pedido actualizado"
4 }
```

```
142 {
143   "id_detalle": 21,
144   "id_pedido": 4,
145   "id_prod": 2,
146   "cantidad": 10,
147   "subtotal": "137000.00"
148 }
149 ]
```

Eliminar

DELETE http://localhost:3000/api/detalle_pedido/21 Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results 200 OK • 222 ms • 323 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "success": true,
3   "message": "Detalle de pedido eliminado"
4 }
```

```
126   "subtotal": "7000.00"
127 },
128 {
129   "id_detalle": 20,
130   "id_pedido": 11,
131   "id_prod": 6,
132   "cantidad": 5,
133   "subtotal": "10000.00"
134 },
135 {
```

Lo anterior fue de acuerdo a las pruebas por cada tabla, donde se realizo el CRUD en cada una de ellas, dando todas las pruebas con éxito.

Allí mismo en el postman; hay una carpeta adicional, donde se halla para consultas nativas:

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


```

//Obtener todos los productos de un pedido específico
app.get('/api/productos-pedido/:id_pedido', async (req, res) => {
  const { id_pedido } = req.params;
  try {
    const result = await client.query(`
      SELECT p.id_prod, p.nombre, p.precio, 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);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

Lo anterior, me permite consultar todos los productos asociados a un pedido determinado.

Se utiliza el parámetro `id_pedido`, que se recibe desde la URL, para buscar en la tabla `DetallePedido` todos los registros que correspondan a ese pedido.

Para obtener más información del producto (nombre, precio), se hace un JOIN con la tabla `Producto` usando la clave foránea `id_prod`.

El este resultado incluye el id del producto, su nombre, precio, la cantidad solicitada en ese pedido y el subtotal de cada uno (cantidad × precio).

siendo útil para ver en detalle qué fue lo que un cliente compró en un pedido específico.

- Prueba en postman

GET http://localhost:3000/api/productos-pedido/3

Params Authorization Headers (7) Body Scripts Tests Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (8) Test Results

{ } JSON Preview Visualize

```
1 [
2   {
3     "id_prod": 7,
4     "nombre": "Ajiaco",
5     "precio": "14000.00",
6     "cantidad": 1,
7     "subtotal": "14000.00"
8   },
9   {
10    "id_prod": 6,
11    "nombre": "Empanada",
12    "precio": "2000.00",
13    "cantidad": 3,
14    "subtotal": "6000.00"
15  }
16 ]
```

- Otro ejemplo:

GET http://localhost:3000/api/productos-pedido/2

Params Authorization Headers (7) Body Scripts Tests Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (8) Test Results

{ } JSON Preview Visualize

```
1 [
2   {
3     "id_prod": 4,
4     "nombre": "Arepa Rellena",
5     "precio": "9000.00",
6     "cantidad": 2,
7     "subtotal": "18000.00"
8   },
9   {
10    "id_prod": 10,
11    "nombre": "Gaseosa",
12    "precio": "3000.00",
13    "cantidad": 3,
14    "subtotal": "9000.00"
15  }
16 ]
```

- **Obtener los productos más vendidos (más de X unidades)**

```
//Obtener los productos más vendidos (más de X unidades)
app.get('/api/productos-mas-vendidos/:cantidad', async (req, res) => {
  const { cantidad } = req.params;
  try {
    const result = await client.query(`
      SELECT p.id_prod, p.nombre, SUM(dp.cantidad) AS total_vendido
      FROM DetallePedido dp
      JOIN Producto p ON dp.id_prod = p.id_prod
      GROUP BY p.id_prod, p.nombre
      HAVING SUM(dp.cantidad) > $1
      ORDER BY total_vendido DESC
    `, [cantidad]);
    res.json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

La consulta me permite revisar cuáles son los productos que han tenido mayor demanda. Se recibe un parámetro llamado cantidad desde la URL, que define el mínimo de unidades vendidas que debe tener un producto para aparecer en los resultados.

La consulta suma (SUM) la cantidad de veces que cada producto se ha vendido, agrupando los resultados por id_prod y nombre.

Se utiliza HAVING para filtrar solamente los productos cuya suma total supera la cantidad indicada.

Por último, los resultados se ordenan en forma descendente (ORDER BY) para que aparezcan primero los productos más vendidos.

- Pruebas en postman

GET

▼

http://localhost:3000/api/productos-mas-vendidos/5

Params

Authorization

Headers (7)

Body

Scripts

Tests

Settings

Query Params

	Key	Value	Descri
	Key	Value	Descri

Body

Cookies

Headers (8)

Test Results

↺

200 OK

• 250 ms

{ } JSON ▼

▶ Preview

🔍 Visualize

▼

```
1  {
2    {
3      "id_prod": 2,
4      "nombre": "Perro Caliente",
5      "total_vendido": "10"
6    },
7    {
8      "id_prod": 6,
9      "nombre": "Empanada",
10     "total_vendido": "8"
11   },
12   {
13     "id_prod": 18,
14     "nombre": "Gaseosa",
15     "total_vendido": "6"
16   },
17   {
18     "id_prod": 17,
19     "nombre": "Jugo Natural",
20     "total_vendido": "6"
21   }
}
```

Otro ejemplo:

GET ▼ http://localhost:3000/api/productos-mas-vendidos/7

Params Authorization Headers (7) Body Scripts Tests Settings

Query Params

	Key	Value	Desc
	Key	Value	Desc

Body Cookies Headers (8) Test Results ↺ 200 OK • 146 n

{ } JSON ▼ ▶ Preview 🔗 Visualize ▼

```
1  [  
2    {  
3      "id_prod": 2,  
4      "nombre": "Perro Caliente",  
5      "total_vendido": "10"  
6    },  
7    {  
8      "id_prod": 6,  
9      "nombre": "Empanada",  
10     "total_vendido": "8"  
11   }  
12 ]
```

Captu
Guard

- Obtener el total de ventas por restaurante

```

//Obtener el total de ventas por restaurante
app.get('/api/ventas-por-restaurante', async (req, res) => {
  try {
    const result = await client.query(`
      SELECT r.id_rest, r.nombre, SUM(p.total) AS total_ventas
      FROM Pedido p
      JOIN Restaurante r ON p.id_rest = r.id_rest
      GROUP BY r.id_rest, r.nombre
      ORDER BY total_ventas DESC
    `);
    res.json(result.rows);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

```

Aquí, puedo obtener el monto total de ventas acumulado por cada restaurante. Se hace un JOIN entre las tablas Pedido y Restaurante para poder asociar cada pedido con su respectivo restaurante; luego, se utiliza SUM(p.total) para calcular la suma de todos los totales de pedidos correspondientes a cada restaurante. Se agrupa (GROUP BY) por ID y nombre del restaurante para que los resultados no se mezclen entre sí. Al final, se ordena (ORDER BY) de mayor a menor venta para identificar rápidamente cuáles restaurantes están generando más ingresos.

- pruebas

GET
http://localhost:3000/api/ventas-por-restaurante

Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings

Query Params

Key	Value	Description
-----	-------	-------------

Body
Cookies
Headers (8)
Test Results
200 OK • 189 ms • 932 B

JSON
Preview
Visualize

```

2      {
3        "id_rest": 3,
4        "nombre": "Comida Criolla",
5        "total_ventas": "55500.00"
6      },
7      {
8        "id_rest": 8,
9        "nombre": "Ajiaco Real",
10       "total_ventas": "55000.00"
11     },
12     {
13       "id_rest": 6,
14       "nombre": "La Arepa Viajera",
15       "total_ventas": "49000.00"
16     },
17     {
18       "id_rest": 9,
19       "nombre": "Sabor del Caribe",
20       "total_ventas": "43000.00"
21     },
22     {
23       "id_rest": 2,
24       "nombre": "Sazón del Valle",
25       "total_ventas": "30000.00"

```

Postbot
Runner
Start Proxy

• Obtener los pedidos realizados en una fecha específica

Esta consulta permite listar todos los pedidos que se realizaron en un día determinado; recibe un parámetro por la URL, que debe tener el formato compatible con la base de datos; por ejemplo, YYYY-MM-DD.

La consulta filtra directamente en la tabla Pedido, buscando aquellos registros cuyo campo fecha coincida exactamente con el valor recibido.

Devuelve todos los campos de cada pedido, como id, id de restaurante, total, entre otros.

- pruebas

GET

▼

http://localhost:3000/api/pedidos-por-fecha/2024-04-03

ParamsAuthorizationHeaders (7)BodyScriptsTestsSettings

Query Params

	Key	Value	Description
--	-----	-------	-------------

BodyCookiesHeaders (8)Test Results

200 OK • 143 ms • 433

{ } JSON ▼▶ Preview🔗 Visualize ▼

```
1  [
2    {
3      "id_pedido": 4,
4      "fecha": "2024-04-03T05:00:00.000Z",
5      "id_rest": 4,
6      "total": "12000.00"
7    },
8    {
9      "id_pedido": 5,
10     "fecha": "2024-04-03T05:00:00.000Z",
11     "id_rest": 5,
12     "total": "19500.00"
13   }
14 ]
```

- **Obtener los empleados por rol en un restaurante**

Esta consulta sirve para traer todos los empleados que desempeñan un determinado rol en un restaurante específico.

Recibe dos parámetros desde la URL, pero a través de query params: el rol (por ejemplo, 'chef', 'Cajero', etc.) y el id_rest, que corresponde al restaurante al que pertenecen los empleados. Después, se realiza una consulta que filtra la tabla Empleado utilizando WHERE, exigiendo que el rol y el id del restaurante coincidan con los valores proporcionados.

- pruebas

GET

▼

http://localhost:3000/api/empleados-por-rol?rol=Chef&id_rest=1

Params ● Authorization Headers (7) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	rol	Chef

Body Cookies Headers (8) Test Results ↻

200

{}

JSON ▼

▶ Preview

🔄 Visualize ▼

```
1  [
2    {
3      "id_empleado": 1,
4      "nombre": "Carlos Pérez",
5      "rol": "Chef",
6      "id_rest": 1
7    }
8  ]
```

Otro ejemplo:

GET http://localhost:3000/api/empleados-por-rol?rol=Cajero&id_rest=10

Params Authorization Headers (7) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description
<input checked="" type="checkbox"/>	rol	Cajero	

Body Cookies Headers (8) Test Results 200 OK • 200 ms • 341 B

{ } JSON Preview Visualize

```
1  [
2    {
3      "id_empleado": 19,
4      "nombre": "Natalia Franco",
5      "rol": "Cajero",
6      "id_rest": 10
7    }
8  ]
```

Siendo esto la estructura y pruebas de las 5 consultas nativas solicitadas.