

Parcial II

1. Se crea la carpeta donde se almacenará el proyecto y nos posicionamos sobre la misma

```
/dev/fd/15.18. command not found. Computer
● hannahgear@Hannahs-MacBook-Pro ~ % cd Desktop
● hannahgear@Hannahs-MacBook-Pro Desktop % mkdir parcialii
● hannahgear@Hannahs-MacBook-Pro Desktop % cd parcialii
○ hannahgear@Hannahs-MacBook-Pro parcialii % █
```

2. Se genera el inicio del proyecto y la instalación de las dependencias requeridas para hacer el mismo

```
● hannahgear@Hannahs-MacBook-Pro parcialii % npm init -y
Wrote to /Users/hannahgear/Desktop/parcialii/package.json:

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

● hannahgear@Hannahs-MacBook-Pro parcialii % npm install express mysql2 cors dotenv
added 80 packages, and audited 81 packages in 7s
16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

3. Primero vamos a crear las tablas en supabase, entonces

Create a new table under public

Enable Row Level Security (RLS) Recommended
Restrict access to your table by enabling RLS and writing Postgres policies.

i Policies are required to query data

You need to create an access policy before you can query data from this table. Without a policy, querying this table will return an empty array of results. You can create policies after saving this table.

[Documentation](#)

Enable Realtime
Broadcast changes on this table to authorized subscribers

Columns

[About data types](#) [Import data from CSV](#)

Name <small>?</small>	Type	Default Value <small>?</small>	Primary
id_rest	# int8	NULL	<input checked="" type="checkbox"/>  
nombre	T varchar	NULL	 
ciudad	T varchar	NULL	 
direccion	T varchar	NULL	 
fecha_apertura	date	NULL	 

Add column

[Cancel](#) [Save](#)

Create a new table under public

Policies are required to query data

You need to create an access policy before you can query data from this table. Without a policy, querying this table will return an [empty array](#) of results. You can create policies after saving this table.

[Documentation](#)

Enable Realtime
Broadcast changes on this table to authorized subscribers

Columns

[About data types](#) | [Import data from CSV](#)

Name ?	Type	Default Value ?	Primary
id_empleado	# int8	NULL	<input checked="" type="checkbox"/>  X
nombre	T varchar	NULL	 X
rol	T varchar	NULL	 X
id_rest	int8	NULL	<input type="checkbox"/>  X

[Add column](#)

Foreign keys

ADD Foreign key relation to:  public.restaurant
`id_rest → public.restaurant.id_rest`

[Edit](#) [Remove](#)

[Cancel](#) [Save](#)

Create a new table under public

Name

Description

Enable Row Level Security (RLS) Recommended
Restrict access to your table by enabling RLS and writing Postgres policies.

i Policies are required to query data
You need to create an access policy before you can query data from this table. Without a policy, querying this table will return an empty array of results. You can create policies after saving this table.

Documentation

Enable Realtime
Broadcast changes on this table to authorized subscribers

Columns

About data types Import data from CSV

Name ?	Type	Default Value ?	Primary
<input type="text" value="id_prod"/>	# int8	<input type="text" value="NULL"/>	<input checked="" type="checkbox"/> 1 X
<input type="text" value="nombre"/>	T varchar	<input type="text" value="NULL"/>	<input type="checkbox"/> X
<input type="text" value="precio"/>	# numeric	<input type="text" value="NULL"/>	<input type="checkbox"/> X

Cancel Save

Create a new table under public

Restrict access to your table by enabling RLS and writing Postgres policies.



Policies are required to query data

You need to create an access policy before you can query data from this table. Without a policy, querying this table will return an empty array of results. You can create policies after saving this table.

[Documentation](#)

Enable Realtime

Broadcast changes on this table to authorized subscribers

Columns

[About data types](#)

[Import data from CSV](#)

Name ?	Type	Default Value ?	Primary
id_pedido	# int8	NULL	<input checked="" type="checkbox"/>
fecha	date	NULL	<input type="checkbox"/>
id_rest	int8	NULL	<input type="checkbox"/>
total	# numeric	NULL	<input type="checkbox"/>

Add column

Foreign keys

[ADD](#)

Foreign key relation to: public.restaurant

Edit

Remove

Cancel

Save

Columns

About data types Import data from CSV

Name	Type	Default Value	Primary
id_detalle	# int8	NULL	<input checked="" type="checkbox"/> 1
id_pedido	int8	NULL	<input type="checkbox"/>
id_prod	int8	NULL	<input type="checkbox"/>
cantidad	# int8	NULL	<input type="checkbox"/>
subtotal	# numeric	NULL	<input type="checkbox"/>

Add column

Foreign keys

ADD Foreign key relation to: public.pedido

`id_pedido → public.pedido.id_pedido`

ADD Foreign key relation to: public.producto

`id_prod → public.producto.id_prod`

Add foreign key relation

4. Luego creamos los archivos que vamos a necesitar para generar la conexión del proyecto con supabase y node js

A screenshot of a code editor showing two files: '.env' and 'supabaseClient.js'. The '.env' file contains environment variables:

```
SUPABASE_URL=https://mwomnkpevlugdfptukhnq.supabase.co
SUPABASE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3Mi01JzdXBhYmFzZSIsInJlZii6Im13b21ua3Bldmx1cWRmdHVraG5xIiwiem9sZSI6ImFab24iLCJpYXQiOjE3NDU0MjM5MTcsImV4cCI6M
PORT=3000
```

The 'supabaseClient.js' file imports 'dotenv' and creates a Supabase client using the environment variables:

```
const { createClient } = require('@supabase/supabase-js');
require('dotenv').config();
const supabase = createClient(process.env.SUPABASE_URL, process.env.SUPABASE_KEY);
module.exports = supabase;
```

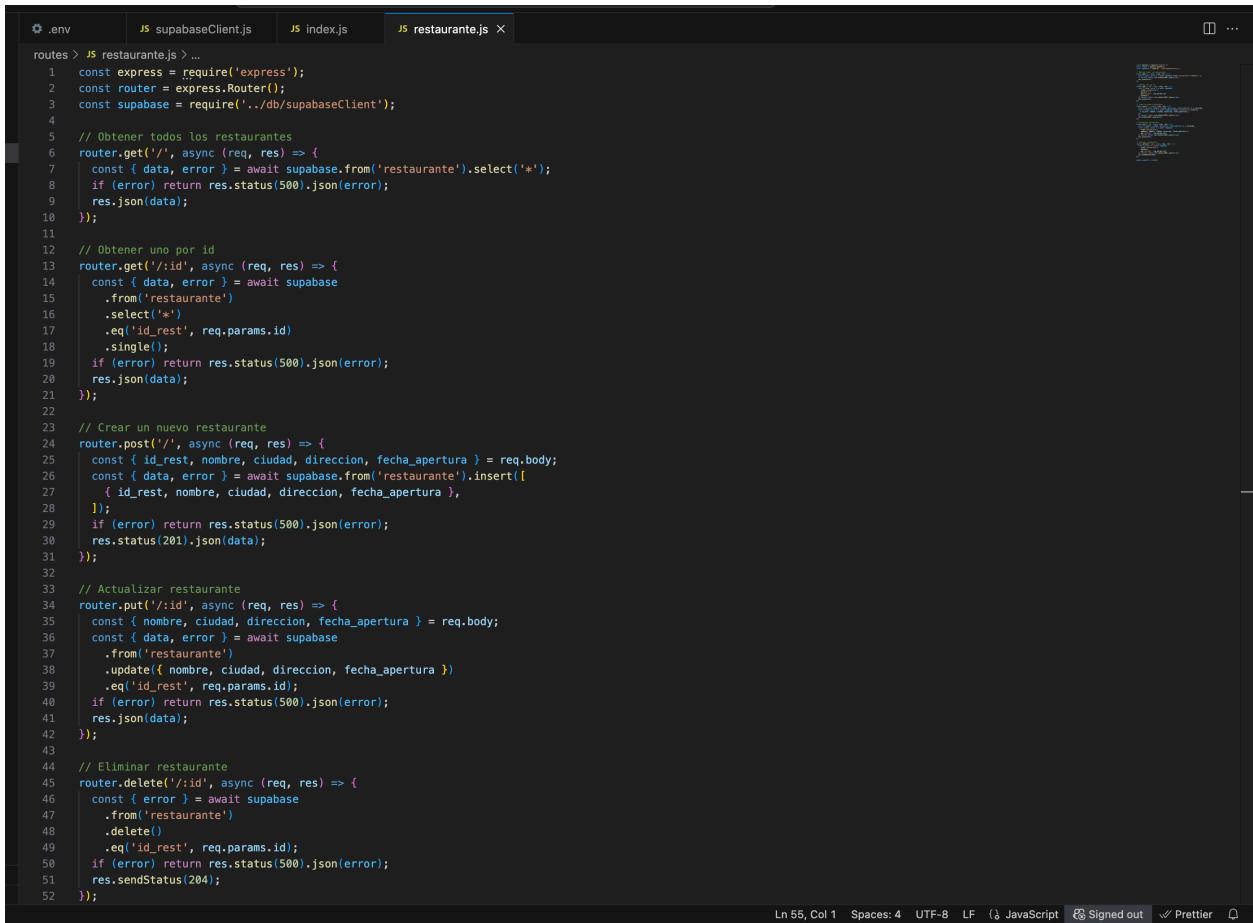
A screenshot of a code editor showing the 'supabaseClient.js' file. The code is identical to the one shown in the previous screenshot.

5. Creamos el file del servidor de Express

A screenshot of a code editor showing the 'index.js' file. The code sets up an Express server, requiring 'express', 'dotenv', and connecting to the Supabase database:

```
const express = require('express');
require('dotenv').config();
const app = express();
const port = process.env.PORT || 3000;
app.use(express.json());
app.use('/restaurante', require('./routes/restaurante'));
app.listen(port, () => {
  console.log(`Servidor corriendo en http://localhost:${port}`);
});
```

6. Creamos el primer CRUD para generar la conexión a postman, el primero es la tabla de restaurantes, teniendo en cuenta que siempre se genera el uso de las constantes que almacenan variables de requerimiento y respuesta, obteniendo los datos solicitados y generando la respuesta programada.



```
routes.get('/', async (req, res) => {
  const express = require('express');
  const router = express.Router();
  const supabase = require('../db/supabaseClient');

  // Obtener todos los restaurantes
  router.get('/', async (req, res) => {
    const { data, error } = await supabase.from('restaurante').select('*');
    if (error) return res.status(500).json(error);
    res.json(data);
  });

  // Obtener uno por id
  router.get('/:id', async (req, res) => {
    const { data, error } = await supabase
      .from('restaurante')
      .select('*')
      .eq('id_rest', req.params.id)
      .single();
    if (error) return res.status(500).json(error);
    res.json(data);
  });

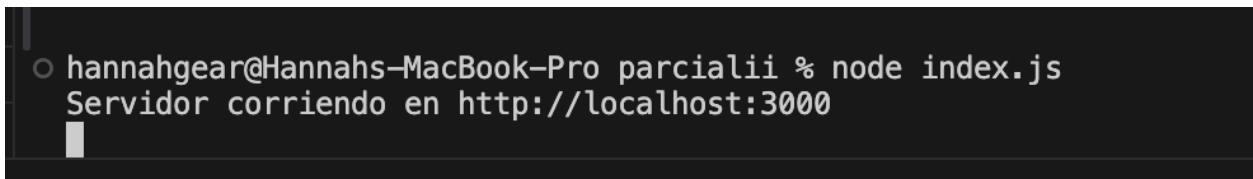
  // Crear un nuevo restaurante
  router.post('/', async (req, res) => {
    const { id_rest, nombre, ciudad, direccion, fecha_apertura } = req.body;
    const { data, error } = await supabase.from('restaurante').insert([
      { id_rest, nombre, ciudad, direccion, fecha_apertura },
    ]);
    if (error) return res.status(500).json(error);
    res.status(201).json(data);
  });

  // Actualizar restaurante
  router.put('/:id', async (req, res) => {
    const { nombre, ciudad, direccion, fecha_apertura } = req.body;
    const { data, error } = await supabase
      .from('restaurante')
      .update({ nombre, ciudad, direccion, fecha_apertura })
      .eq('id_rest', req.params.id);
    if (error) return res.status(500).json(error);
    res.json(data);
  });

  // Eliminar restaurante
  router.delete('/:id', async (req, res) => {
    const { error } = await supabase
      .from('restaurante')
      .delete()
      .eq('id_rest', req.params.id);
    if (error) return res.status(500).json(error);
    res.sendStatus(204);
  });
});

Ln 55, Col 1 Spaces: 4 UTF-8 LF ⚙️ JavaScript 🗑️ Signed out ✅ Prettier
```

7. Ejecutamos el servidor desde node js y verificamos en el navegador que el localhost esté funcionando correctamente.



```
hannahgear@Hannahs-MacBook-Pro parcialii % node index.js
Servidor corriendo en http://localhost:3000
```

8. Posteriormente, verificamos la conexión de la API con postman

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'Laura's Workspace' with a collection named 'Parcial BDM' containing items like 'Restaurante', 'Empleado', 'Pedido', 'Producto', and 'Detalle_pedido'. A 'New Request' button is highlighted. The main workspace shows a 'GET New Request' tab with the URL 'http://localhost:3000/Restaurante'. The 'Headers' tab is selected, showing 'Content-Type: application/json'. The 'Body' tab is selected, showing an empty JSON object: {} . The 'Test Results' tab shows a successful response: 200 OK, 780 ms, 235 B. The 'Preview' section shows an empty array: [] .

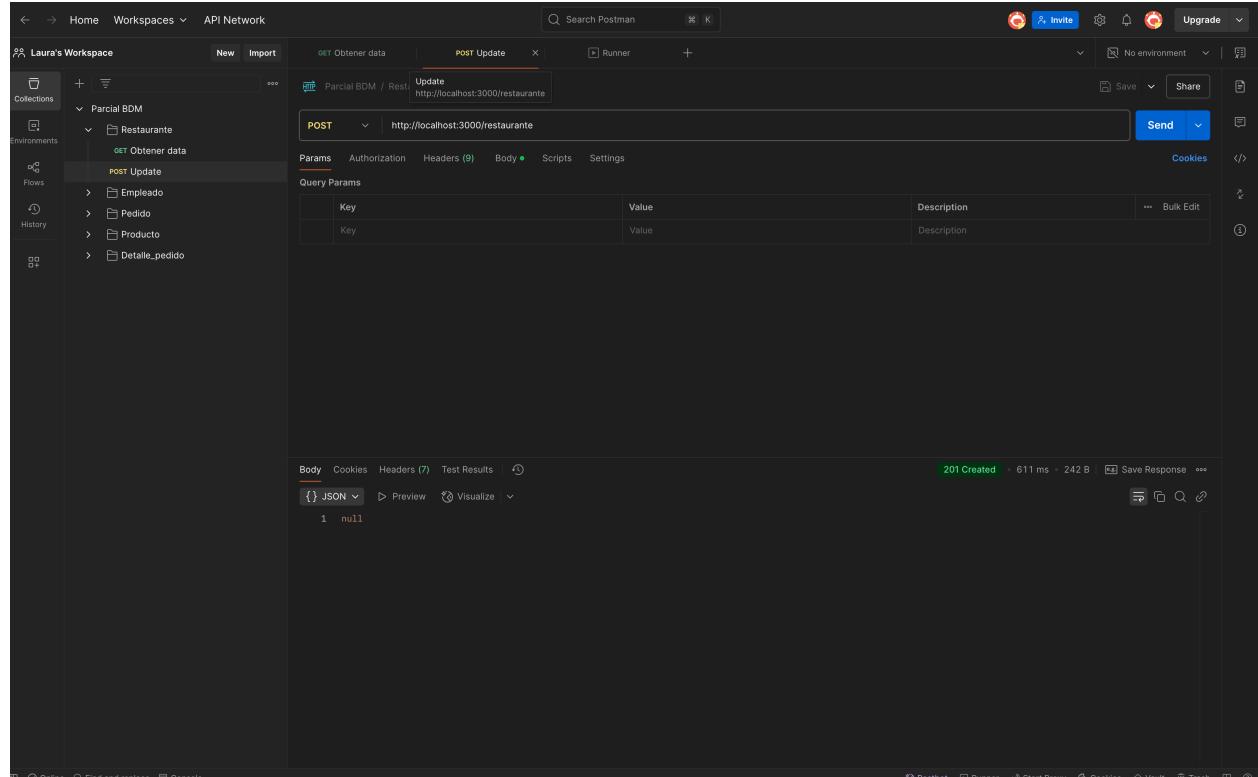
Get que nos permite obtener la información de la tabla restaurante

The screenshot shows the Postman interface with a dark theme. The sidebar is identical to the previous screenshot. The main workspace shows a 'GET Obtener data' tab with the URL 'http://localhost:3000/restaurant'. The 'Headers' tab is selected, showing 'Content-Type: application/json'. The 'Body' tab is selected, showing an empty JSON object: {} . The 'Test Results' tab shows a successful response: 200 OK, 388 ms, 455 B. The 'Preview' section shows two restaurant objects in JSON format:

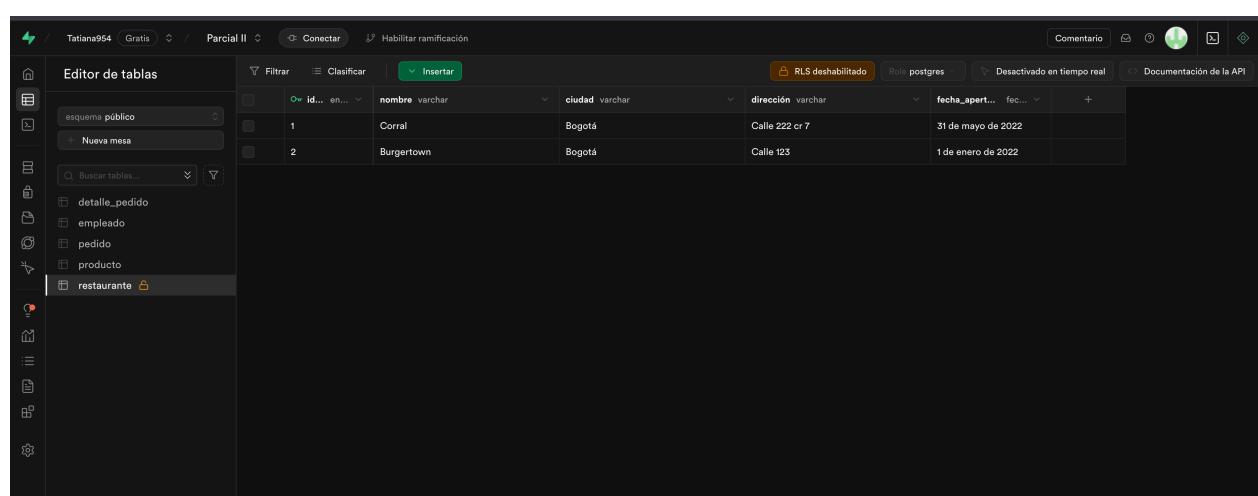
```
[{"id_rest": 1, "nombre": "Corral", "ciudad": "Bogota", "direccion": "Calle 222 cr 7", "fecha_apertura": "2022-05-31"}, {"id_rest": 2, "nombre": "Burgertown", "ciudad": "Bogota", "direccion": "Calle 123", "fecha_apertura": "2022-01-01"}]
```

Y post funciona como un insert para la tabla en supabase, para generar la inserción de los datos se ubica la url del servidor, y luego nos ubicamos en Body, y luego en raw, verificamos que reciba información tipo JSON.

Y ponemos el json que se vaya a utilizar para insertar los datos en la tabla de restaurantes.



The screenshot shows the Postman interface. On the left, the sidebar displays 'Laura's Workspace' with a collection named 'Parcial BDM' containing several items: 'Restaurante', 'GET Obtener data', 'post Update', 'Empleado', 'Pedido', 'Producto', and 'Detalle_pedido'. The main workspace shows a POST request titled 'Update' to 'http://localhost:3000/restaurante'. The 'Body' tab is selected, showing a JSON object with a single key 'Key' and a value 'null'. The response pane at the bottom shows a 201 Created status with a timestamp of 611 ms and a size of 242 B. The response body is empty, showing only 'null'.



The screenshot shows the Supabase Studio interface. On the left, the sidebar lists tables: 'restaurante' (selected), 'detalle_pedido', 'empleado', 'pedido', 'producto', and 'esquema público'. The main area is the 'Editor de tablas' for the 'restaurante' table. It shows two rows of data:

	nombre	ciudad	dirección	fecha_apert...
1	Corral	Bogotá	Calle 222 cr 7	31 de mayo de 2022
2	Burgertown	Bogotá	Calle 123	1 de enero de 2022

Update que tiene la funcionalidad de actualizar los valores de alguna tabla

Postman screenshot showing a collection named "Parcial BDM" with various API endpoints for Restaurant (Restaurante) and Employee (Empleado). The "Obtener data" endpoint is selected, showing a GET request to http://localhost:3000/restaurante. The response body is a JSON array containing two restaurant objects:

```
[{"id_rest": 1, "nombre": "Corral", "ciudad": "Bogota", "direccion": "Calle 222 cr 7", "fecha_apertura": "2022-05-31"}, {"id_rest": 2, "nombre": "Burgertown", "ciudad": "Bogot\u00e1", "direccion": "Calle 123", "fecha_apertura": "2022-01-01"}]
```

Postman screenshot showing the same collection "Parcial BDM". The "Update restaurante" endpoint is selected, showing a PUT request to http://localhost:3000/restaurante/2. The request body is a JSON object:

```
{"nombre": "Burgertown", "ciudad": "Bogot\u00e1", "direccion": "Calle 99 14-24", "fecha_apertura": "2024-08-31"}
```

The response body is null.

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Laura's Workspace' with collections like 'Parcial BDM' and its sub-endpoints: 'Obtener data', 'Insert restaurante', 'Update restaurante', 'Obtener empleados', 'Insert empleado', 'Update empleado', 'Obtener pedidos', 'Insert pedido', 'Obtener productos', 'Insert producto', 'Obtener detalle_pedido', and 'Insert detalle_pedido'. The 'Obtener data' endpoint under 'Restaurante' is currently selected.

In the main workspace, a GET request is defined for the URL `http://localhost:3000/restaurante`. The 'Params' tab is active, showing no parameters. The 'Body' tab is selected, showing the JSON response:

```

1 [ 
2   {
3     "id_rest": 1,
4     "nombre": "Corral",
5     "ciudad": "Bogota",
6     "direccion": "Calle 222 cr 7",
7     "fecha_apertura": "2022-05-31"
8   },
9   {
10    "id_rest": 2,
11    "nombre": "Burgertown",
12    "ciudad": "Bogota",
13    "direccion": "Calle 123",
14    "fecha_apertura": "2024-08-31"
15  }
16 ]

```

The status bar at the bottom indicates a successful response: `200 OK`, `364 ms`, `455 B`.

Y por último, para completar el CRUD usamos Delete para eliminar registros en las tablas

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'Laura's Workspace' with various collections like 'Parcial BDM', 'Restaurante', 'Empleado', 'Pedido', 'Producto', and 'Detalle_pedido'. Under 'Restaurante', there are methods: GET Obtener data, POST Insert restaurante, PUT Update restaurante, and DEL Delete restaurante. The main panel shows a DELETE request to 'http://localhost:3000/restaurante/2'. The 'Test Results' section indicates a successful 204 No Content response with a duration of 397 ms and a size of 175 B.

This screenshot shows a GET request to 'http://localhost:3000/restaurante'. The response is a 200 OK with a duration of 487 ms and a size of 346 B. The JSON body contains one record:

```

1 [
2   {
3     "id_rest": 1,
4     "nombre": "Corral",
5     "ciudad": "Bogota",
6     "direccion": "Calle 222 cr 7",
7     "fecha_apertura": "2022-05-31"
8   }
9 ]

```

9. Luego seguimos con la configuración de las otras tablas

- Empleado:

```

routes > JS empleado.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const supabase = require('../db/supabaseClient');
4
5  const router: Router = {
6    router.get('/', async (req, res) => {
7      const { data, error } = await supabase.from('empleado').select('*');
8      if (error) return res.status(500).json(error);
9      res.json(data);
10    });
11
12    // Obtener un empleado por ID
13    router.get('/id', async (req, res) => {
14      const { data, error } = await supabase.from('empleado').select('*').eq('id_empleado', req.params.id).single();
15      if (error) return res.status(500).json(error);
16      res.json(data);
17    });
18
19    // Crear empleado
20    router.post('/', async (req, res) => {
21      const { id_empleado, nombre, rol, id_rest } = req.body;
22      const { data, error } = await supabase.from('empleado').insert([{ id_empleado, nombre, rol, id_rest }]);
23      if (error) return res.status(500).json(error);
24      res.status(201).json(data);
25    });
26
27    // Actualizar empleado
28    router.put('/id', async (req, res) => {
29      const { nombre, rol, id_rest } = req.body;
30      const { data, error } = await supabase.from('empleado').update({ nombre, rol, id_rest }).eq('id_empleado', req.params.id);
31      if (error) return res.status(500).json(error);
32      res.json(data);
33    });
34  };

```

SELECT:

The screenshot shows the Postman interface with a collection named "Parcial BDM". A GET request is selected for the "Empleado" endpoint. The request URL is `http://localhost:3000/empleado`. The request body is a JSON object:

```

1  [
2    {
3      "id_empleado": 2,
4      "nombre": "Omar Velez",
5      "rol": "Cajero",
6      "id_rest": 1
7    },
8    {
9      "id_empleado": 2,
10     "nombre": "Carlos Perez",
11     "rol": "Cocinero",
12     "id_rest": 1
13   }
14 ]

```

The response status is 200 OK, and the response body is a JSON array containing two objects.

INSERT:

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays a collection named "Parcial BDM" containing various API endpoints for "Restaurante", "Empleado", "Pedido", "Producto", and "Detalle_pedido". The main workspace shows a POST request titled "Update empleado" with the URL `http://localhost:3000/empleado`. The "Body" tab is selected, showing the following JSON payload:

```
1 {
2   "id_empleado": 2,
3   "nombre": "Carlos Perez",
4   "rol": "Cocinero",
5   "id_rest": 1
6 }
```

The "Test Results" tab shows a successful response with status code 201 Created, execution time of 300 ms, and a response size of 242 B. The response body is null.

UPDATE:

The screenshot shows the Postman interface with a dark theme. The sidebar displays a collection named "Parcial BDM" containing various API endpoints for "Restaurante", "Empleado", "Pedido", "Producto", and "Detalle_pedido". The main workspace shows a GET request titled "Obtener empleados" with the URL `http://localhost:3000/empleado`. The "Headers" tab is selected, showing the following query parameters:

Key	Value	Description
Key	Value	Description

The "Test Results" tab shows a successful response with status code 200 OK, execution time of 478 ms, and a response size of 375 B. The response body is a JSON array containing two employee objects:

```
1 [
2   {
3     "id_empleado": 1,
4     "nombre": "Onax Velez",
5     "rol": "Cajero",
6     "id_rest": 1
7   },
8   {
9     "id_empleado": 2,
10    "nombre": "Carlos Perez",
11    "rol": "Cocinero",
12    "id_rest": 1
13  }
14 ]
```

The screenshot shows the Postman interface with a PUT request to `http://localhost:3000/empleado/1`. The request body is a JSON object:

```

1 {
2   "nombre": "Omar Vélez",
3   "rol": "Gerente",
4   "id_rest": 1
5 }
6

```

The response status is 200 OK, with a duration of 354 ms and a size of 237 B. The response body is null.

The screenshot shows the Postman interface with a GET request to `http://localhost:3000/empleados`. The query parameters section is visible, showing a table with columns: Key, Value, and Description. The response status is 200 OK, with a duration of 569 ms and a size of 377 B. The response body is a JSON array of employee objects:

```

1 [
2   {
3     "id_empleado": 2,
4     "nombre": "Carlos Perez",
5     "rol": "Cocinero",
6     "id_rest": 1
7   },
8   {
9     "id_empleado": 1,
10    "nombre": "Omar Vélez",
11    "rol": "Gerente",
12    "id_rest": 1
13  }
14 ]

```

DELETE:

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'Laura's Workspace' with a collection named 'Parcial BDM'. Under this collection, there are several requests: 'GET Obtener restaurante', 'POST Insert restaurante', 'PUT Update restaurante', 'DEL Delete restaurante', 'GET Obtener empleados', 'POST Insert empleado', 'PUT Update empleado', and 'DEL Delete empleado'. The 'DELETE empleado' request is selected. The main panel shows the request details: method 'DELETE', URL 'http://localhost:3000/empleado/2', and a table for 'Query Params' with one entry: 'Key' (Value) and 'Description'. Below the table, tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results' are visible. The 'Test Results' tab shows a single digit '1'. At the bottom, the response status is '204 No Content', with a duration of '447 ms' and a size of '175 B'. Buttons for 'Save Response' and other actions are at the bottom right.

This screenshot shows the same Postman session after executing the 'GET Obtener empleados' request. The main panel now displays the results of the 'GET' operation. The 'Body' tab is active, showing a JSON array of employee objects. The array contains one element, which is a JSON object with fields: 'id_empleado' (value: 1), 'nombre' (value: 'Omar Velez'), 'rol' (value: 'Gerente'), and 'id_rest' (value: 1). The 'Headers (7)' tab shows the response headers, and the 'Test Results' tab shows a status of '200 OK' with a duration of '710 ms' and a size of '305 B'. The bottom navigation bar includes links for 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and other workspace options.

- **Producto:**

```
routes > JS producto.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const supabase = require('../db/supabaseClient');
4
5  // Obtener todos los productos
6  router.get('/', async (req, res) => {
7    const { data, error } = await supabase.from('producto').select('*');
8    if (error) return res.status(500).json(error);
9    res.json(data);
10 });
11
12 // Obtener un producto por ID
13 router.get('/:id', async (req, res) => {
14   const { data, error } = await supabase.from('producto').select('*').eq('id_prod', req.params.id).single();
15   if (error) return res.status(500).json(error);
16   res.json(data);
17 });
18
19 // Crear producto
20 router.post('/', async (req, res) => {
21   const { id_prod, nombre, precio } = req.body;
22   const { data, error } = await supabase.from('producto').insert([{ id_prod, nombre, precio }]);
23   if (error) return res.status(500).json(error);
24   res.status(201).json(data);
25 });
26
27 // Actualizar producto
28 router.put('/:id', async (req, res) => {
29   const { nombre, precio } = req.body;
30   const { data, error } = await supabase.from('producto').update({ nombre, precio }).eq('id_prod', req.params.id);
31   if (error) return res.status(500).json(error);
32   res.json(data);
33 });
34
35 // Eliminar producto
36 router.delete('/:id', async (req, res) => {
37   const { error } = await supabase.from('producto').delete().eq('id_prod', req.params.id);
38   if (error) return res.status(500).json(error);
39   res.sendStatus(204);
40 });
41
42 module.exports = router;
43
```

In 43 Col 1 Spaces: 4 (175%) 11F (-) JavaScript 22 StandardOut ↻ Pd

SELECT:

The screenshot shows the Postman interface with a collection named "Parcial BDM". A GET request is made to `http://localhost:3000/producto`. The response status is 200 OK, and the response body is a JSON array with one element:

```
[{"id_prod": 1, "nombre": "Hamburguesa", "precio": 40000}]
```

INSERT:

The screenshot shows the Postman interface with the same collection. A POST request is made to `http://localhost:3000/producto`. The response status is 201 Created, and the response body is null.

UPDATE:

The screenshot shows the Postman application interface with two requests displayed.

Request 1: GET Obtener productos

- Method:** GET
- URL:** http://localhost:3000/producto
- Headers:** (7)
- Body:** ({} JSON) - Preview shows a JSON array of products:

```
1 [  
2   {  
3     "id_prod": 1,  
4     "nombre": "Hamburguesa",  
5     "precio": 40000  
6   },  
7   {  
8     "id_prod": 2,  
9     "nombre": "Hamburguesa Doble",  
10    "precio": 15000  
11   }  
12 ]
```
- Test Results:** 200 OK, 387 ms, 347 B

Request 2: PUT Update producto

- Method:** PUT
- URL:** http://localhost:3000/producto/1
- Headers:** (9)
- Body:** (raw) - Raw JSON body:

```
1 {  
2   "nombre": "Hamburguesa Callejera",  
3   "precio": 28000  
4 }
```
- Test Results:** 200 OK, 413 ms, 237 B

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'Laura's Workspace' with a collection named 'Parcial BDM' containing various API endpoints like 'Obtener productos', 'Insertar producto', etc. The main workspace shows a 'GET Obtener productos' request to 'http://localhost:3000/producto'. The response status is '200 OK' with a response time of 611 ms and a size of 357 B. The response body is a JSON array of products:

```
[{"id_prod": 2, "nombre": "Hamburguesa Doble", "precio": 15000}, {"id_prod": 1, "nombre": "Hamburguesa Callejera", "precio": 28000}]
```

DELETE:

The screenshot shows the Postman interface with a dark theme. The sidebar remains the same, displaying 'Laura's Workspace' and the 'Parcial BDM' collection. The main workspace shows a 'DELETE Delete producto' request to 'http://localhost:3000/producto/2'. The response status is '204 No Content' with a response time of 569 ms and a size of 175 B. The response body is empty.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (Parcial BDM), 'Environments', 'Flows', and 'History'. The main area displays a collection named 'Parcial BDM' with several endpoints under 'Producto':

- GET Obtener productos
- POST Insert producto
- PUT Update producto
- DELETE Delete producto

Under 'Detalle_pedido':

- GET Obtener detalle
- POST Insert detalle pedido
- PUT Update detalle

The 'Obtener productos' endpoint is currently selected. The request details show a GET request to `http://localhost:3000/producto`. The 'Body' tab shows the response body as JSON:

```
1 [  
2   {  
3     "id_prod": 1,  
4     "nombre": "Hamburguesa Callejera",  
5     "precio": 20000  
6   }  
7 ]
```

The response status is 200 OK with a duration of 419 ms and a size of 298 B. The response body is also displayed in the preview pane.

- Pedido:

```
routes > JS pedido.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const supabase = require('../db/supabaseClient');
4
5  // Obtener todos los pedidos
6  router.get('/', async (req, res) => {
7    const { data, error } = await supabase.from('pedido').select('*');
8    if (error) return res.status(500).json(error);
9    res.json(data);
10 });
11
12 // Obtener un pedido por ID
13 router.get('/:id', async (req, res) => {
14   const { data, error } = await supabase.from('pedido').select('*').eq('id_pedido', req.params.id).single();
15   if (error) return res.status(500).json(error);
16   res.json(data);
17 });
18
19 // Crear pedido
20 router.post('/', async (req, res) => {
21   const { id_pedido, fecha, id_rest, total } = req.body;
22   const { data, error } = await supabase.from('pedido').insert([{ id_pedido, fecha, id_rest, total }]);
23   if (error) return res.status(500).json(error);
24   res.status(201).json(data);
25 });
26
27 // Actualizar pedido
28 router.put('/:id', async (req, res) => {
29   const { fecha, id_rest, total } = req.body;
30   const { data, error } = await supabase.from('pedido').update({ fecha, id_rest, total }).eq('id_pedido', req.params.id);
31   if (error) return res.status(500).json(error);
32   res.json(data);
33 });
34
35 // Eliminar pedido
36 router.delete('/:id', async (req, res) => {
37   const { error } = await supabase.from('pedido').delete().eq('id_pedido', req.params.id);
38   if (error) return res.status(500).json(error);
39   res.sendStatus(204);
40 });
41
42 module.exports = router;
43
```

SELECT:

Parcial BDM / Pedido / Obtener pedidos

GET http://localhost:3000/pedido

Key	Value	Description
Key		Description

```
{
  "id_pedido": 1,
  "fecha": "2025-04-23",
  "id_rest": 1,
  "total": 40000
}
```

INSERT:

Parcial BDM / Pedido / Update pedido

POST http://localhost:3000/pedido

Body	Cookies	Headers	Test Results
{ } JSON			201 Created

```
{
  "id_pedido": 2,
  "fecha": "2025-04-25",
  "id_rest": 1,
  "total": 30000
}
```

Body Cookies Headers (7) Test Results

{} JSON ▾ ▶ Preview ⚡ Visualize ▾

1 null

UPDATE:

Home Workspaces API Network

Laura's Workspace

Parcial BDM / Pedido / Obtener pedidos

GET http://localhost:3000/pedido

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (7) Test Results

{ } JSON Preview Visualize

200 OK 470 ms 363 B Save Response

```
[{"id_pedido": 1, "fecha": "2025-04-23", "id_rest": 1, "total": 40000}, {"id_pedido": 2, "fecha": "2025-04-25", "id_rest": 1, "total": 30000}]
```

Postbot Runner Start Proxy Cookies Vault Trash

Home Workspaces API Network

Laura's Workspace

Parcial BDM / Pedido / Update pedido

PUT http://localhost:3000/pedido/2

Params Authorization Headers (9) Body Scripts Settings

Body

{ } none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{"fecha": "2025-04-26", "id_rest": 1, "total": 40000}
```

Body Cookies Headers (7) Test Results

{ } JSON Preview Visualize

200 OK 547 ms 237 B Save Response

```
null
```

Postbot Runner Start Proxy Cookies Vault Trash

GET Obtener pedidos

Parcial BDM / Pedido / Obtener pedidos

```

[{"id_pedido": 1, "fecha": "2025-04-25", "id_rest": 1, "total": 40000}, {"id_pedido": 2, "fecha": "2025-04-26", "id_rest": 1, "total": 40000}]
  
```

DELETE:

DELETE Delete pedido

Parcial BDM / Pedido / Delete pedido

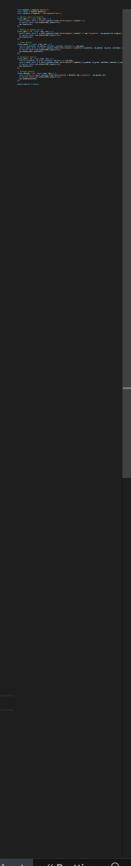
```
1
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Laura's Workspace' containing collections like 'Parcial BDM', 'Restaurante', 'Empleado', and 'Pedido'. Under 'Pedido', there are several endpoints: 'GET Obtener pedidos', 'POST Insert pedido', 'PUT Update pedido', 'DELETE Delete pedido', 'GET Detalle_pedido', 'GET Obtener detalle', 'POST Insert detalle pedido', and 'PUT Update detalle'. The main area shows a 'GET Obtener pedidos' request. The 'Body' tab displays a JSON response:

```
1 [  
2   {  
3     "id_pedido": 2,  
4     "fecha": "2025-04-20",  
5     "id_rest": 1,  
6     "total": 40000  
7   }  
8 ]
```

- Detalle pedido:

```
routes > JS detallePedido.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const supabase = require('../db/supabaseClient');
4
5  // Obtener todos los detalles
6  router.get('/', async (req, res) => {
7    const { data, error } = await supabase.from('detallepedido').select('*');
8    if (error) return res.status(500).json(error);
9    res.json(data);
10 });
11
12 // Obtener un detalle por ID
13 router.get('/:id', async (req, res) => {
14   const { data, error } = await supabase.from('detallepedido').select('*').eq('id_detalle', req.params.id).single();
15   if (error) return res.status(500).json(error);
16   res.json(data);
17 });
18
19 // Crear detalle
20 router.post('/', async (req, res) => {
21   const { id_detalle, id_pedido, id_prod, cantidad, subtotal } = req.body;
22   const { data, error } = await supabase.from('detallepedido').insert([{ id_detalle, id_pedido, id_prod, cantidad, subtotal }]);
23   if (error) return res.status(500).json(error);
24   res.status(201).json(data);
25 });
26
27 // Actualizar detalle
28 router.put('/:id', async (req, res) => {
29   const { id_pedido, id_prod, cantidad, subtotal } = req.body;
30   const { data, error } = await supabase.from('detallepedido').update({ id_pedido, id_prod, cantidad, subtotal }).eq('id_detalle', req.params.id);
31   if (error) return res.status(500).json(error);
32   res.json(data);
33 });
34
35 // Eliminar detalle
36 router.delete('/:id', async (req, res) => {
37   const { error } = await supabase.from('detallepedido').delete().eq('id_detalle', req.params.id);
38   if (error) return res.status(500).json(error);
39   res.sendStatus(204);
40 });
41
42 module.exports = router;
43 |
```



Ln 43, Col 1 Spaces: 4 UTF-8 LF ⓘ JavaScript ⚙️ Signed out ✓ Prettier

SELECT:

The screenshot shows the Postman interface with a dark theme. On the left, there's a sidebar with 'Laura's Workspace' containing collections like 'Parcial BDM', 'Restaurante', 'Empleado', 'Pedido', and 'Producto'. Under 'Producto', there are three items: 'GET Obtener productos', 'POST Update producto', and 'GET Obtener detalle'. The main area shows a 'GET' request to 'http://localhost:3000/detalle'. The 'Body' tab displays a JSON response:

```
[{"id_detalle": 1, "id_pedido": 1, "id_prod": 1, "cantidad": 1, "subtotal": 40000}]
```

INSERT:

The screenshot shows the Postman interface with a dark theme. The sidebar is identical to the previous one. The main area shows a 'POST' request to 'http://localhost:3000/detalle'. The 'Body' tab is set to 'raw' and contains the following JSON payload:

```
{"id_detalle": 2, "id_pedido": 2, "id_prod": 1, "cantidad": 2, "subtotal": 80000}
```

The response status is '201 Created' with a timestamp of '475 ms' and a size of '242 B'. The response body is null.

UPDATE:

The screenshot shows the Postman interface with a successful GET request. The URL is `http://localhost:3000/detalle`. The response body is a JSON array containing two objects, each representing a product detail:

```
1 [  
2   {  
3     "id_detalle": 1,  
4     "id_pedido": 1,  
5     "id_prod": 1,  
6     "cantidad": 1,  
7     "subtotal": 40000  
8   },  
9   {  
10    "id_detalle": 2,  
11    "id_pedido": 2,  
12    "id_prod": 1,  
13    "cantidad": 2,  
14    "subtotal": 80000  
15  }  
16 ]
```

Response status: 200 OK | 399 ms | 383 B

The screenshot shows the Postman interface with a successful PUT request. The URL is `http://localhost:3000/detalle/1`. The request body is a JSON object:

```
1 {  
2   "id_pedido": 1,  
3   "id_prod": 1,  
4   "cantidad": 3,  
5   "subtotal": 84000  
6 }
```

Response status: 200 OK | 510 ms | 237 B

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'Laura's Workspace' with a collection named 'Parcial BDM' containing several items: 'Restaurante', 'Empleado', 'Pedido', 'Producto', and 'Detalle_pedido'. Under 'Detalle_pedido', there are three requests: 'GET Obtener detalle', 'POST Insertar detalle pedido', and 'PUT Update detalle'. The main workspace shows a 'GET Obtener detalle' request for 'Parcial BDM / Detalle_pedido / Obtener detalle'. The request URL is `http://localhost:3000/detalle`. The 'Body' tab shows a JSON response with two objects, each representing a detail item:

```
[{"id": 1, "id_detalle": 1, "id_pedido": 1, "id_prod": 1, "cantidad": 1, "subtotal": 80000}, {"id": 2, "id_detalle": 2, "id_pedido": 2, "id_prod": 1, "cantidad": 2, "subtotal": 160000}],
```

DELETE:

The screenshot shows the Postman interface with a dark theme. The sidebar is identical to the previous one, displaying 'Laura's Workspace' with the 'Parcial BDM' collection. The main workspace shows a 'DELETE Obtener detalle' request for 'Parcial BDM / Detalle_pedido / Delete detalle'. The request URL is `http://localhost:3000/detalle/2`. The 'Body' tab shows a raw response with the number '1'.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Laura's Workspace' containing collections like 'Parcial BDM' (which includes 'Restaurante', 'Empleado', 'Pedido', 'Producto', and 'Detalle_pedido'), and other operations like 'Insert detalle pedido', 'Update detalle', and 'Delete detalle'. The main workspace shows a 'GET Obtener detalle' request under 'Parcial BDM / Detalle_pedido / Obtener detalle'. The request method is 'GET', the URL is 'http://localhost:3000/detalle', and the response status is '200 OK'. The response body is shown as a JSON object with one item: { }.

10. Para generar las consultas primero nos apoyamos en el uso de Chat GPT para la inserción de varios datos en las tablas.

- Restaurante:

Entendiendo la estructura json para inserción de datos, ayúdame por favor con 50 datos para una tabla restaurante con los campos id_rest que es autoincremental, nombre varchar, ciudad varchar, dirección varchar, fecha_apertura (formato yyyy-mm-dd)

Aquí tienes 50 registros de ejemplo en formato JSON listos para insertar restaurante id_rest es autoincremental, JSON (lo generará la base de datos automáticamente al insertar).

json

Copiar Editar

```
[  
 {  
   "nombre": "El Rincón Paisa",  
   "ciudad": "Medellín",  
   "direccion": "Calle 10 #30-45",  
   "fecha_apertura": "2015-04-12"  
,  
 {  
   "nombre": "Sabor Caribeño",  
   "ciudad": "Barranquilla",  
   "direccion": "Cra 54 #72-11",  
   "fecha_apertura": "2018-08-19"  
,  
 {  
   "nombre": "Bogotá Gourmet",  
   "ciudad": "Bogotá",  
   "direccion": "Av. Chile #21-34",  
   "fecha_apertura": "2020-01-15"  
,  
 {  
   "nombre": "Aroma Express"  
 }]  
 ↓
```

|Ask anything



Buscar



Razón



Crear imagen

...



ChatGPT puede tener errores. Revisa la información importante.

- Empleado:

```
json Copiar Editar

[
  { "id_empleado": 1, "nombre": "Ana Torres", "rol": "Cocinero", "id_rest": 1 },
  { "id_empleado": 2, "nombre": "Carlos Muñoz", "rol": "Cajero", "id_rest": 1 },
  { "id_empleado": 3, "nombre": "Julián Ríos", "rol": "Mesero", "id_rest": 2 },
  { "id_empleado": 4, "nombre": "Laura Méndez", "rol": "Cajero", "id_rest": 3 },
  { "id_empleado": 5, "nombre": "Diana Salas", "rol": "Cocinero", "id_rest": 4 },
  { "id_empleado": 6, "nombre": "Esteban Díaz", "rol": "Mesero", "id_rest": 5 },
  { "id_empleado": 7, "nombre": "Juanita Gómez", "rol": "Cajero", "id_rest": 6 },
  { "id_empleado": 8, "nombre": "Carlos Rivas", "rol": "Cocinero", "id_rest": 7 },
  { "id_empleado": 9, "nombre": "Tatiana López", "rol": "Cocinero", "id_rest": 8 },
  { "id_empleado": 10, "nombre": "Andrés Pérez", "rol": "Mesero", "id_rest": 9 }
]
```

- Producto:

```
json Copiar Editar

[
  { "id_prod": 1, "nombre": "Hamburguesa sencilla", "precio": 12000 },
  { "id_prod": 2, "nombre": "Papas grandes", "precio": 9000 },
  { "id_prod": 3, "nombre": "Gaseosa", "precio": 5000 },
  { "id_prod": 4, "nombre": "Pizza personal", "precio": 16000 },
  { "id_prod": 5, "nombre": "Taco", "precio": 8000 },
  { "id_prod": 6, "nombre": "Combo Sencillo", "precio": 20000 },
  { "id_prod": 7, "nombre": "Ensalada", "precio": 10000 },
  { "id_prod": 8, "nombre": "Sushi roll", "precio": 18000 },
  { "id_prod": 9, "nombre": "Churros", "precio": 6000 },
  { "id_prod": 10, "nombre": "Jugo natural", "precio": 7000 }
]
```

- Pedido:

```
json Copiar Editar

[
  { "id_pedido": 1, "fecha": "2024-04-25", "id_rest": 1, "total": 24000 },
  { "id_pedido": 2, "fecha": "2024-04-25", "id_rest": 1, "total": 30000 },
  { "id_pedido": 3, "fecha": "2024-04-26", "id_rest": 2, "total": 18000 },
  { "id_pedido": 4, "fecha": "2024-04-26", "id_rest": 3, "total": 25000 },
  { "id_pedido": 5, "fecha": "2024-04-27", "id_rest": 4, "total": 21000 },
  { "id_pedido": 6, "fecha": "2024-04-27", "id_rest": 5, "total": 32000 },
  { "id_pedido": 7, "fecha": "2024-04-28", "id_rest": 6, "total": 29000 },
  { "id_pedido": 8, "fecha": "2024-04-28", "id_rest": 7, "total": 27000 },
  { "id_pedido": 9, "fecha": "2024-04-28", "id_rest": 8, "total": 15000 },
  { "id_pedido": 10, "fecha": "2024-04-28", "id_rest": 9, "total": 31000 }
]
```

- Detalle pedido:

```
json Copiar Editar

[
  { "id_detalle": 1, "id_pedido": 1, "id_prod": 1, "cantidad": 1, "subtotal": 12000 },
  { "id_detalle": 2, "id_pedido": 1, "id_prod": 3, "cantidad": 1, "subtotal": 5000 },
  { "id_detalle": 3, "id_pedido": 2, "id_prod": 6, "cantidad": 1, "subtotal": 20000 },
  { "id_detalle": 4, "id_pedido": 3, "id_prod": 4, "cantidad": 1, "subtotal": 16000 },
  { "id_detalle": 5, "id_pedido": 4, "id_prod": 5, "cantidad": 2, "subtotal": 16000 },
  { "id_detalle": 6, "id_pedido": 5, "id_prod": 1, "cantidad": 1, "subtotal": 12000 },
  { "id_detalle": 7, "id_pedido": 6, "id_prod": 7, "cantidad": 1, "subtotal": 10000 },
  { "id_detalle": 8, "id_pedido": 7, "id_prod": 8, "cantidad": 1, "subtotal": 18000 },
  { "id_detalle": 9, "id_pedido": 8, "id_prod": 10, "cantidad": 1, "subtotal": 7000 },
  { "id_detalle": 10, "id_pedido": 9, "id_prod": 9, "cantidad": 2, "subtotal": 12000 }
]
```