Parcial 2

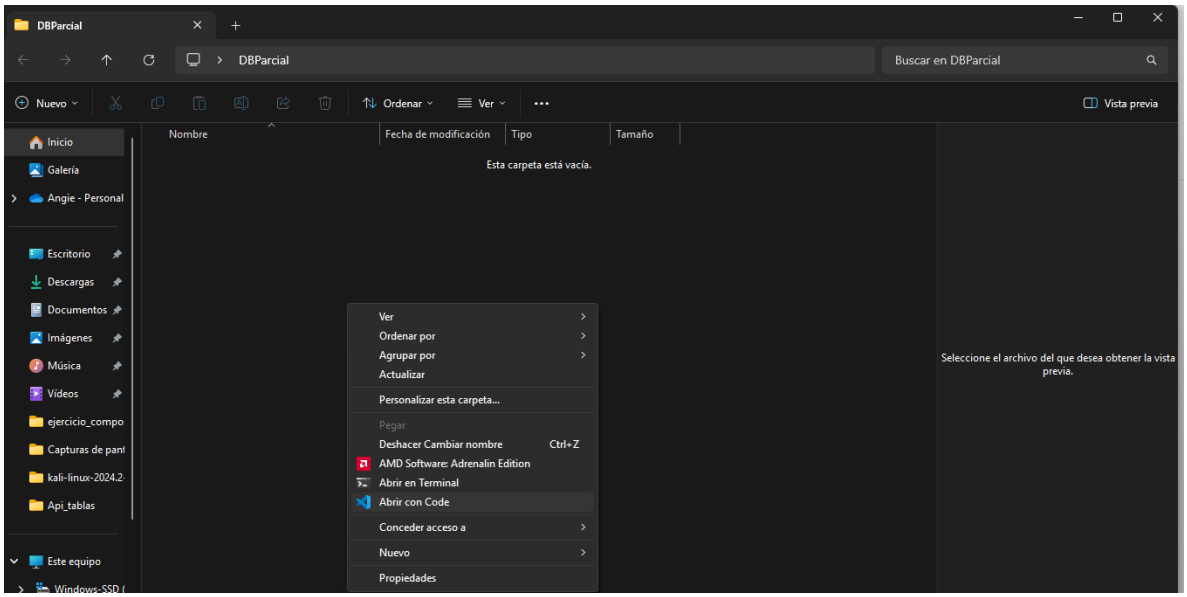
Angie Lorena Jiménez Porras

Ing. Sistemas, UNIMINUTO

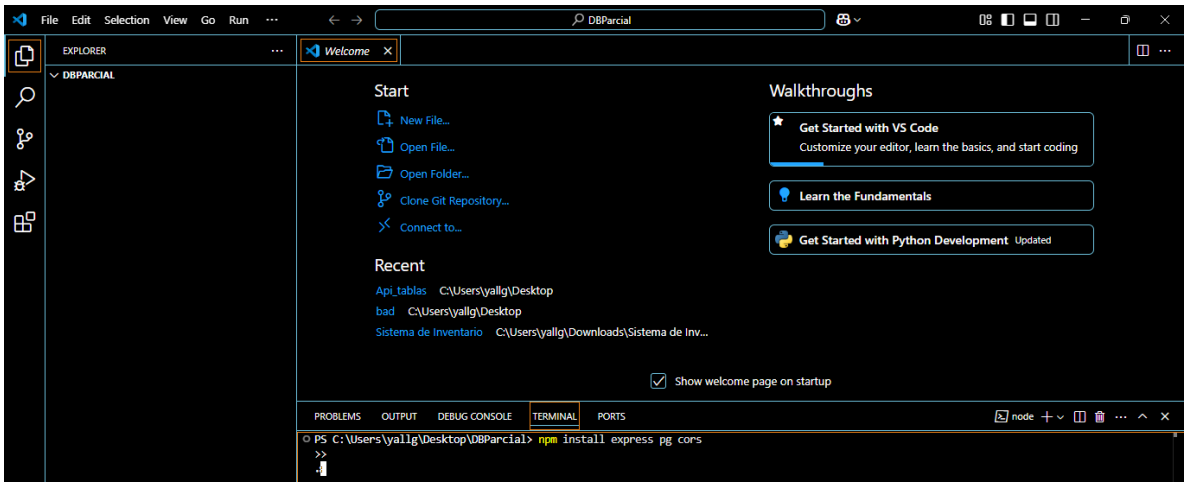
William Alexander Matallana Porras

Bases de datos masivas

25/04/2025

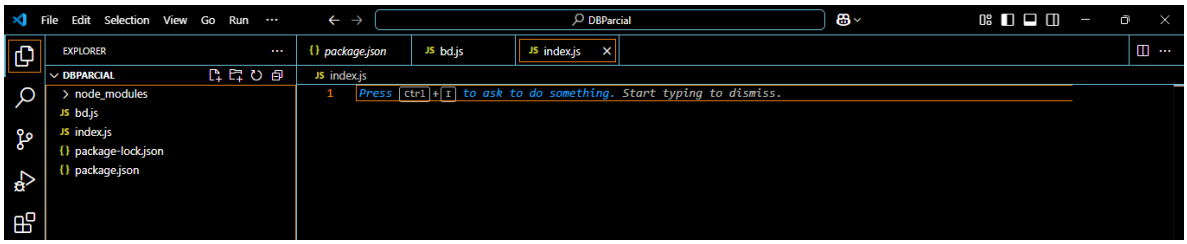


En una carpeta vacía abrimos con visual studio code

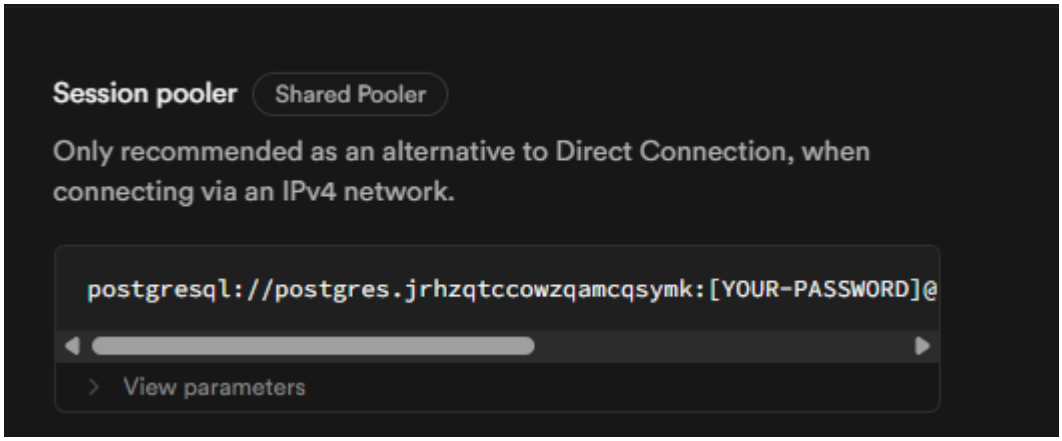


Una vez abierta la carpeta instalamos las extensiones que vayamos a usar.

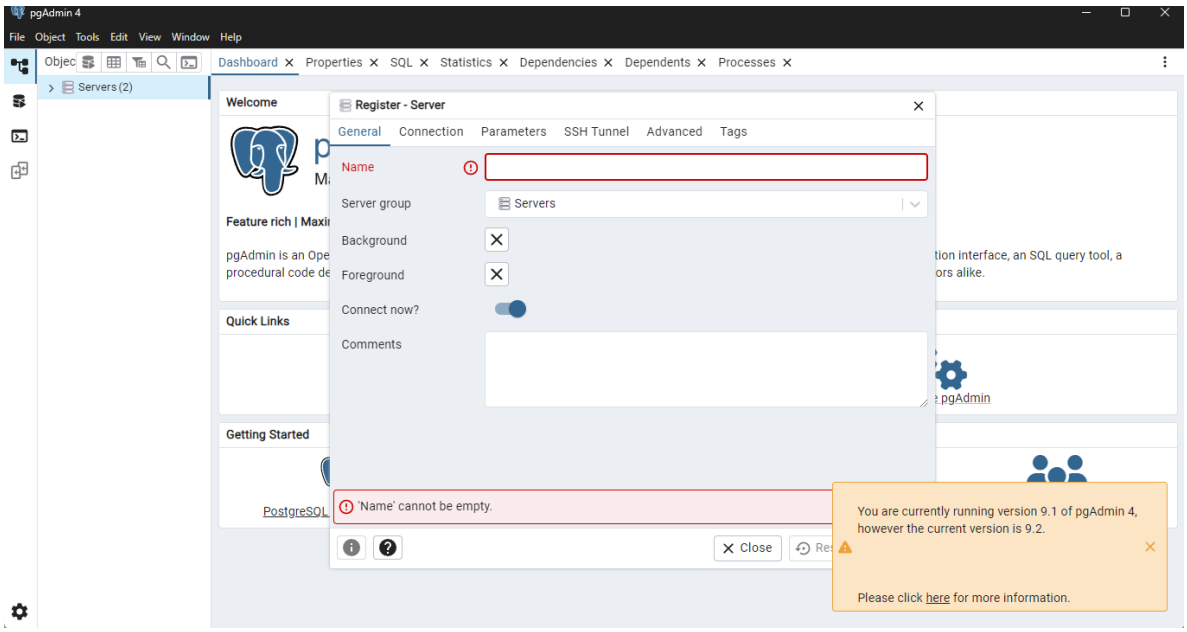
Ahora vamos a tener 2 archivos tipo json



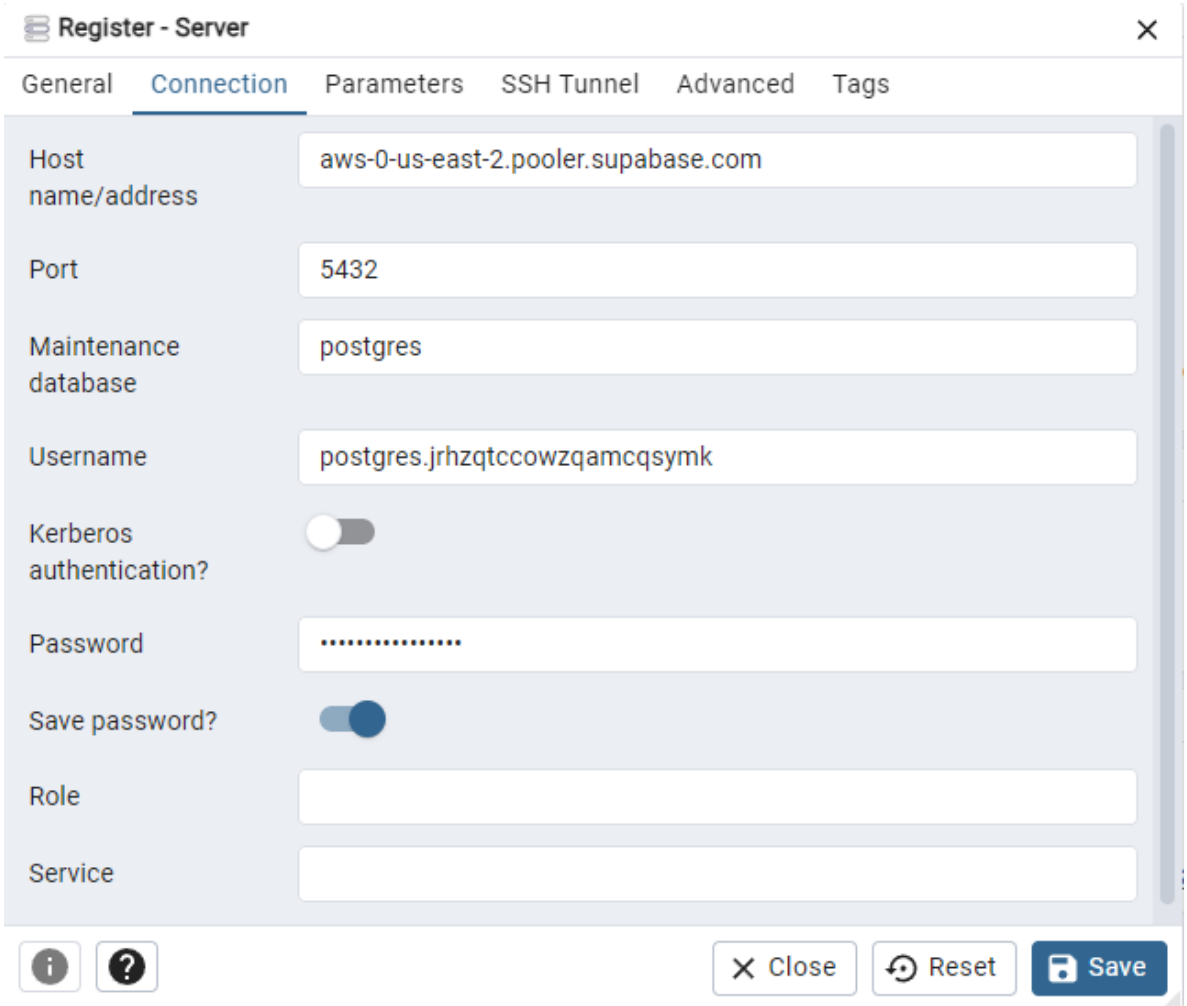
Creamos 2 archvios .js que es bd.js e index.js: esto va a contener nuestra conexión a supabase



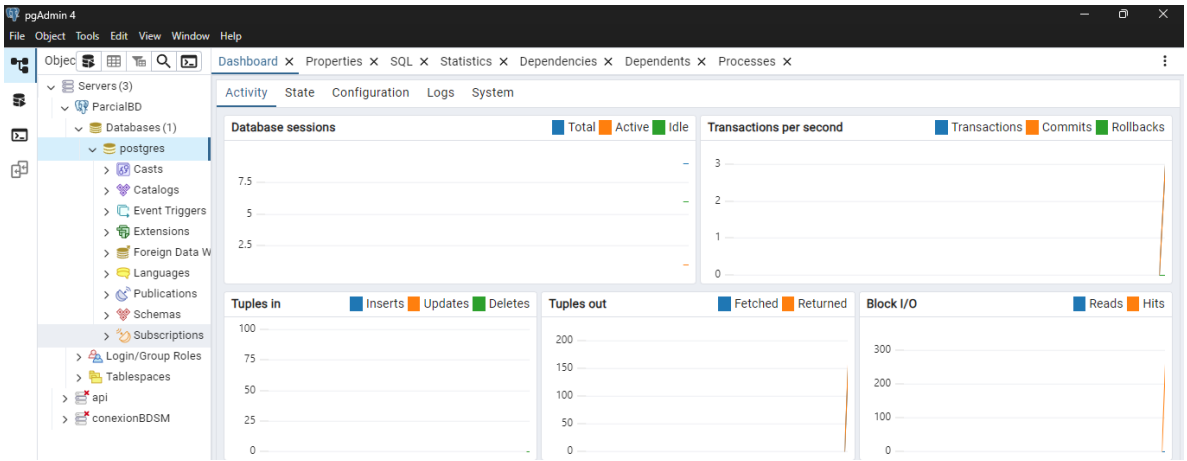
Para tener la conexión con SupaBase vamos al Connect de un nuevo proyecto y copiaremos lo que nos arroja junto con la contraseña generada



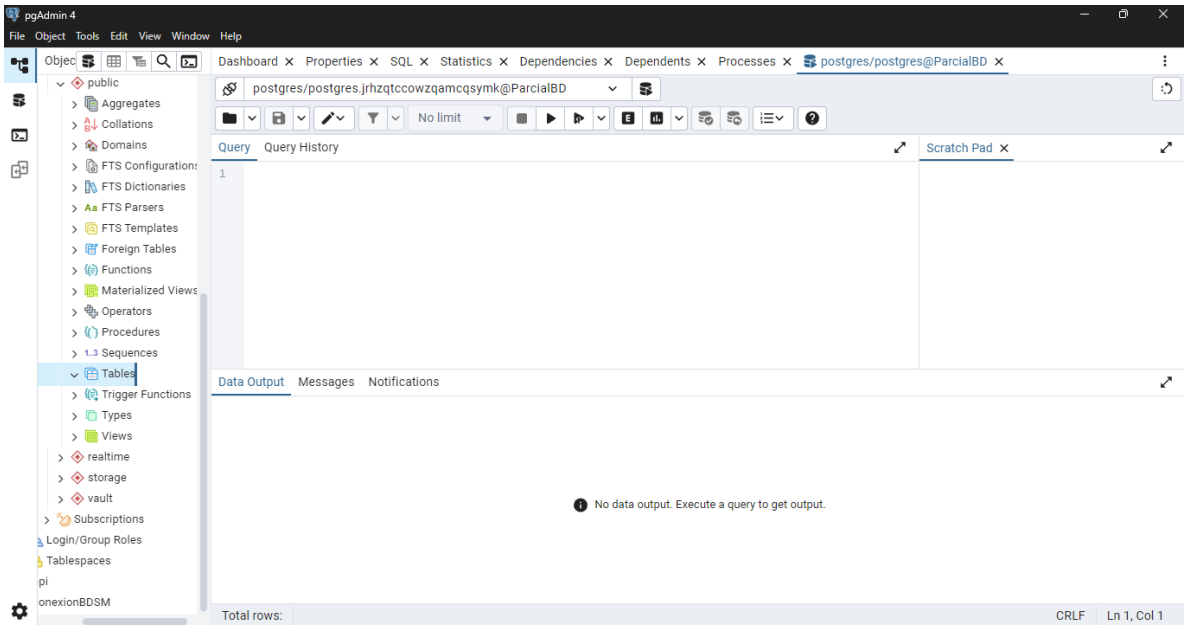
Vamos a usar pg admin que será el cliente que usaremos para conectar a nuestro proyecto de supabase



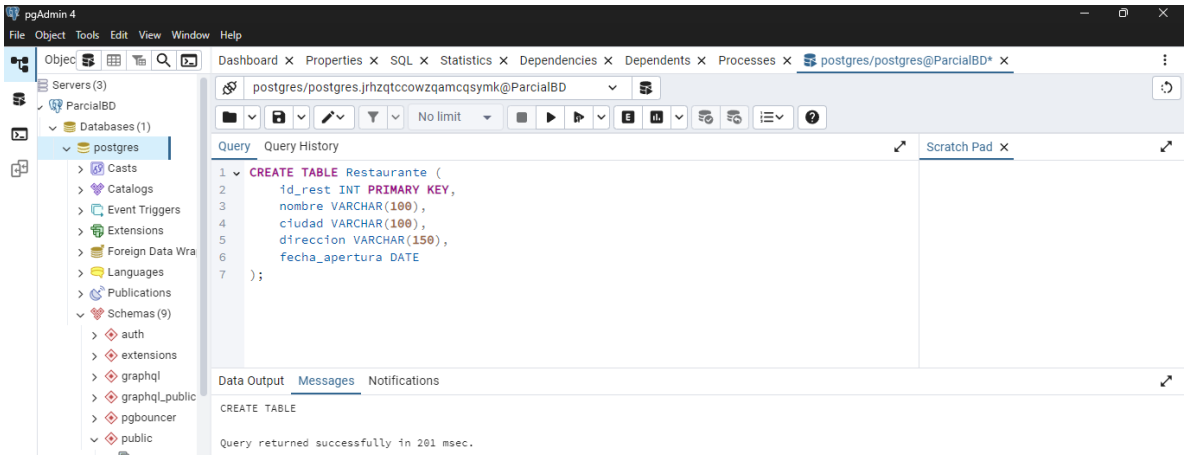
Con los datos proporcionados en SupaBase hacemos la conexión



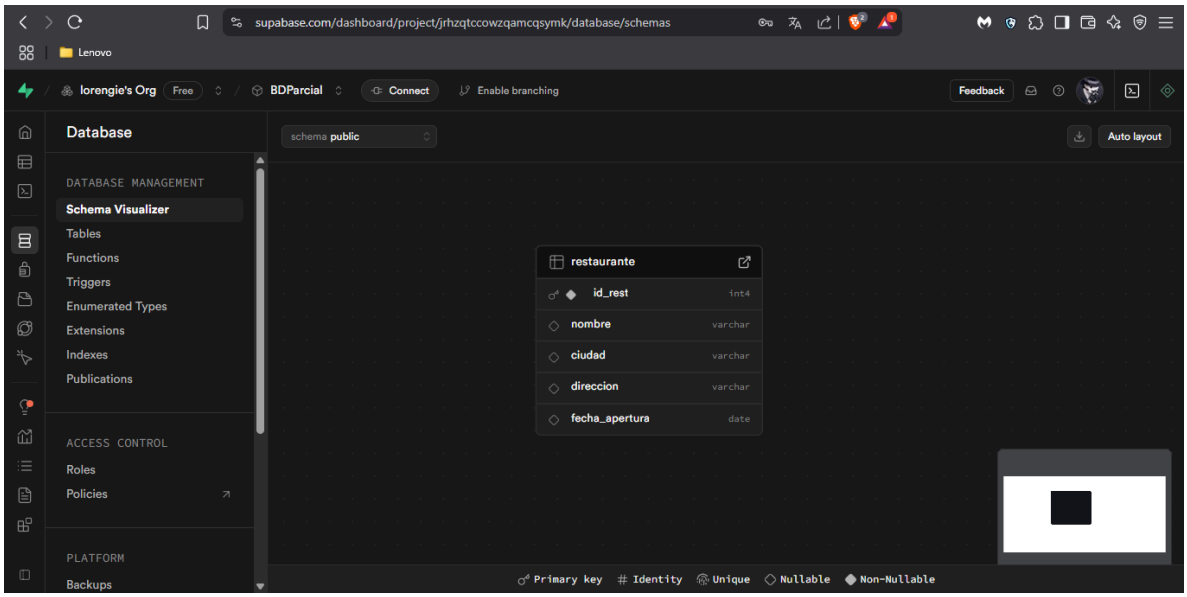
Una vez carga la conexión tendremos una base de datos llamada postres, aquí van a aparecer todas las tablas que vamos a crear



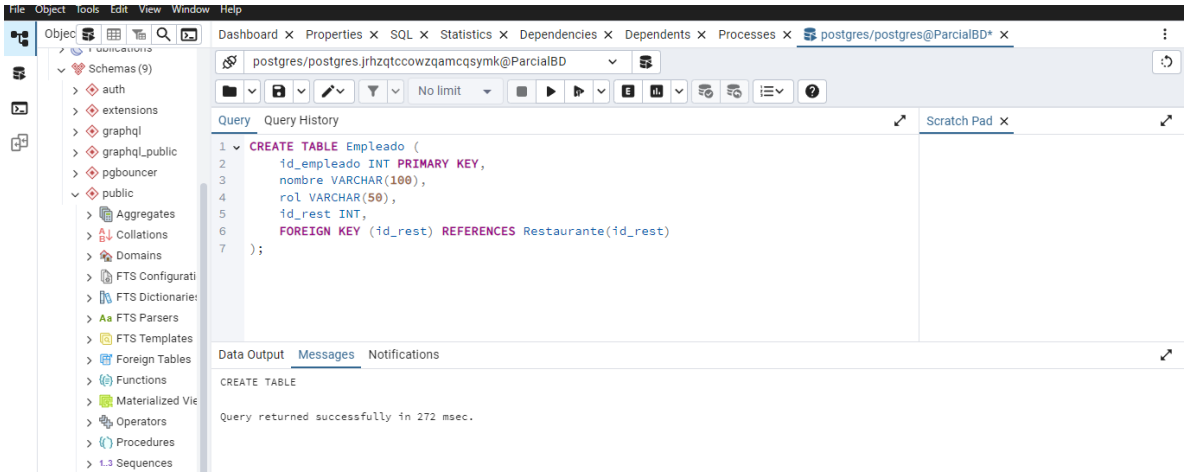
Ahora usaremos la herramienta de Query tools para generar nuestras tablas según los requerimientos dados



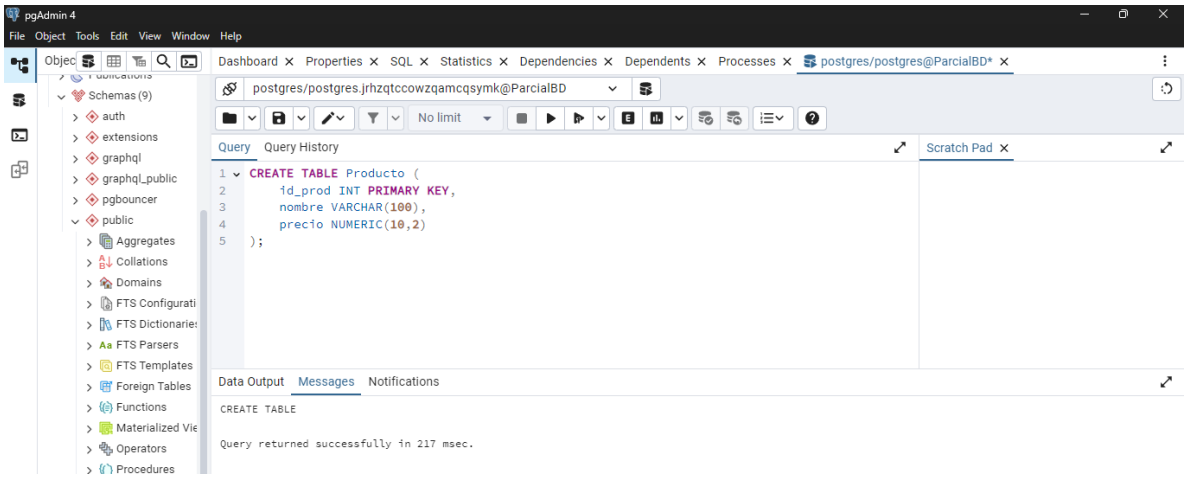
Iniciamos creando la tabla del restaurante según las llaves y campos requeridos



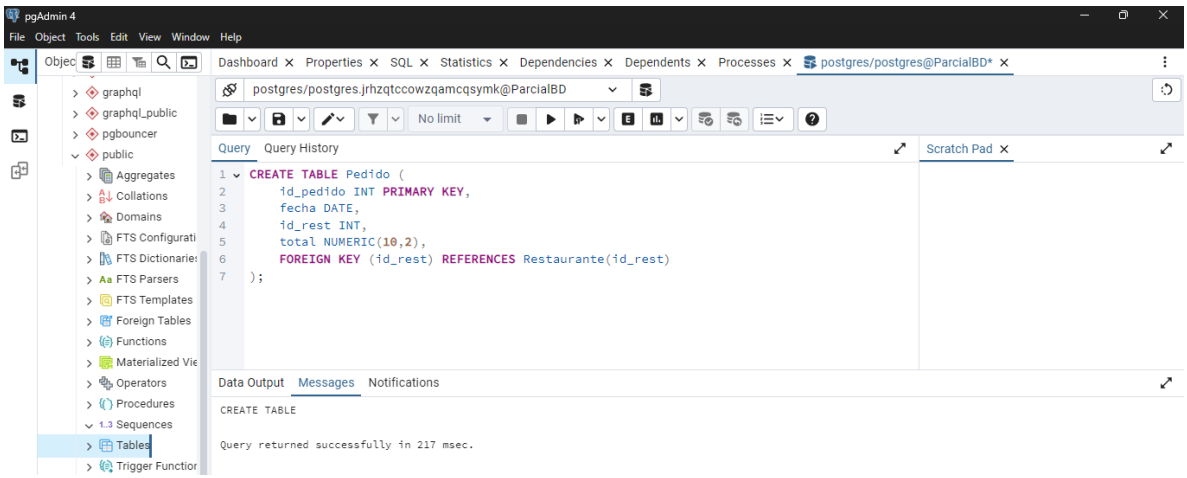
Con supabase revisamos que se haya generado la tabla esto con el de verificar que la conexión haya quedado bien y para verificar que todo haya quedado sincronizado



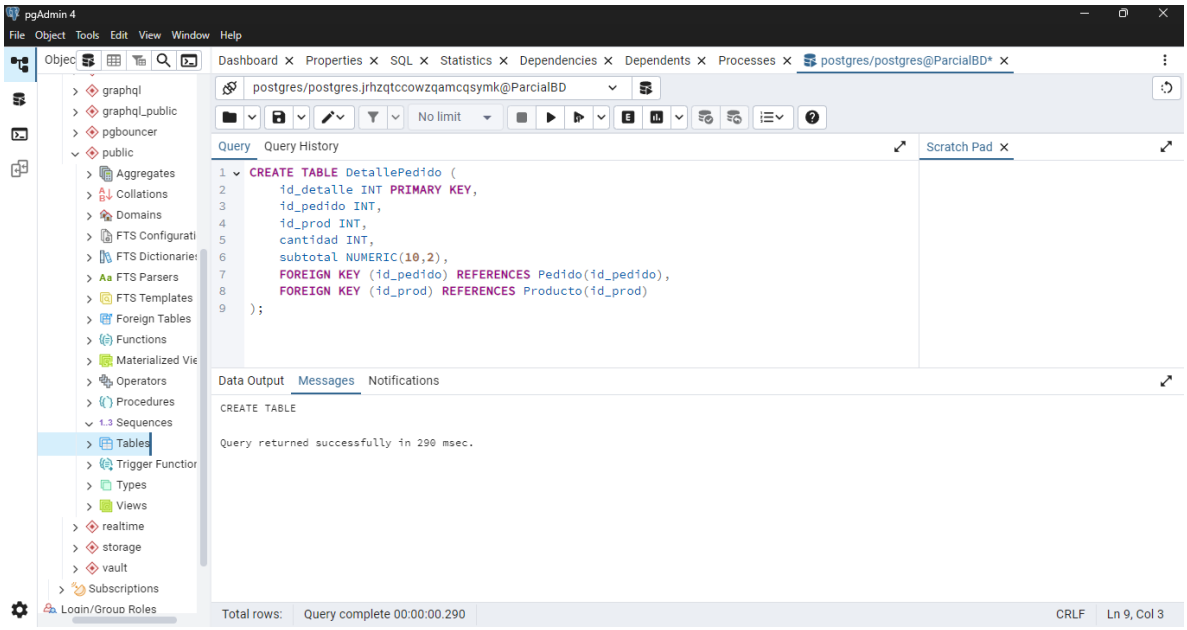
Como vimos que toda la conexión quedo bien hecha y responde procedemos a crear la tabla de empleado



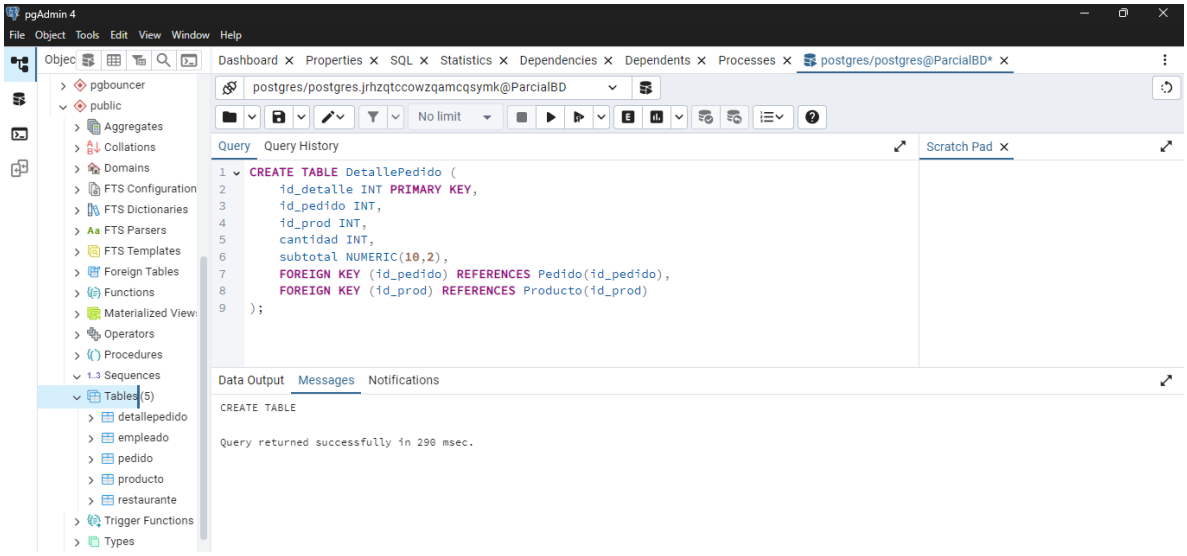
Continuamos generando la tabla de Producto según los campos requeridos



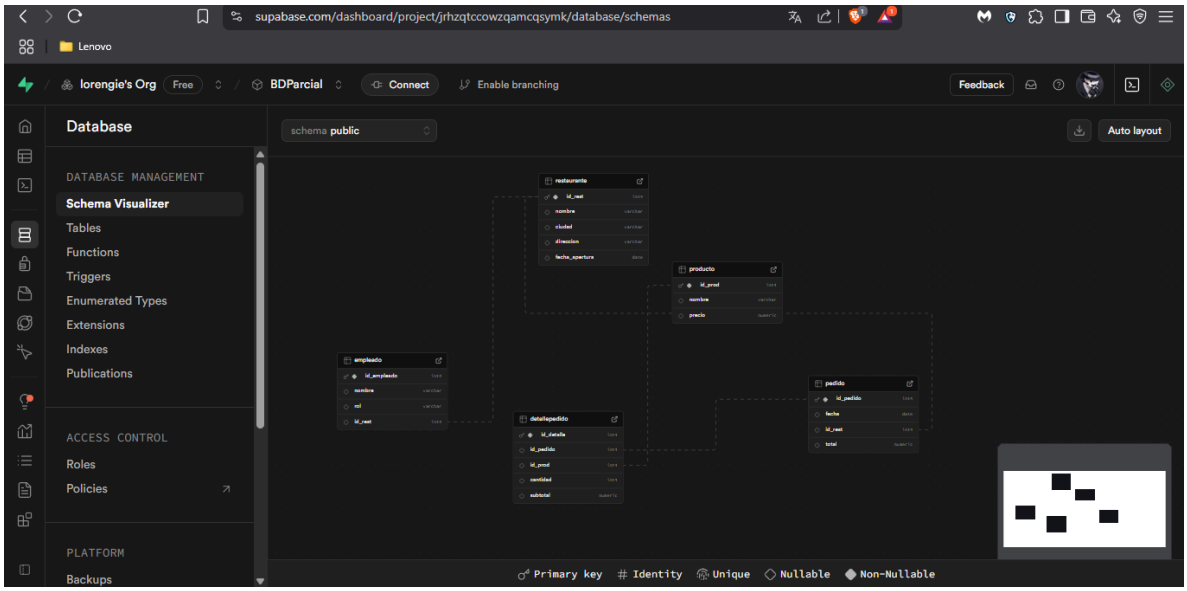
Creamos la tabla de Pedido



Finalmente creamos la tabla de DetallePedido según las llaves foráneas, llave primaria y campos



Mediante pgadmin verificamos que todas las tablas se hayan creado de manera correcta



Revisamos el esquema en supabase. Con esto finalizamos la creación de los modelos de la base de datos usando SupaBase y PgAdmin4

Creación de las API

Iniciamos creando la conexión con supabase en el db.js

En el index vamos a poner la conexión, el puerto y los datos que generó supabase

API CRUD

Para hacer el crud CREAR: de restaurantes se inicia con un app.post para usarlo posteriormente en postman, va a requerir los datos de los campos de la tabla que hayamos configurado

Crud LEER: usamos .get, el cual va a hacer una consulta según los datos de la tabla y va a seleccionar todos los datos, los visualizaremos con el GET de postman

```
app.get('/api/restaurantes/:id', async (req, res) => {
  const id = req.params.id;
  try {
    const result = await pool.query('SELECT * FROM restaurante WHERE id_restaurante = $1', [id]);

    if (result.rows.length === 0) {
      return res.status(404).json({ message: 'Restaurante no encontrado' });
    }

    res.json(result.rows[0]);
  } catch (error) {
    console.error('Error al obtener restaurante:', error.message);
    res.status(500).json({ error: 'Error al obtener restaurante' });
  }
});
```

LEER según el id del restaurante, usamos :id para en postaman daremos el id del que queremos consultar

```
app.put('/api/restaurantes/:id', (req, res) => {
  const id = req.params.id;
  const data = req.body;
  res.json({ message: `Restaurante ${id} actualizado`, data });
});
app.delete('/api/restaurantes/:id', (req, res) => {
  const id = req.params.id;
  res.json({ message: `Restaurante ${id} eliminado` });
});
```

ACTUALIZAR: usamos .put y el id de lo que vayamos a utilizar

BORRAR: usamos .delete para eliminar según el id que pondremos en postman

CRUD empleados

```
app.post('/api/empleados', (req, res) => {
  const data = req.body;
  res.json({ message: 'Empleado creado', data });
});

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

```
66
67  app.get('/api/empleados/:id', async (req, res) => {
68    const id = req.params.id;
69    try {
70      const result = await pool.query('SELECT * FROM empleado WHERE id_empleado = $1', [id]);
71
72      if (result.rows.length === 0) {
73        return res.status(404).json({ message: 'Empleado no encontrado' });
74      }
75
76      res.json(result.rows[0]);
77    } catch (error) {
78      console.error('Error al obtener empleado:', error.message);
79      res.status(500).json({ error: 'Error al obtener empleado' });
80    }
81  });
```

```
83  app.put('/api/empleados/:id', (req, res) => {
84    const id = req.params.id;
85    const data = req.body;
86    res.json({ message: `Empleado ${id} actualizado`, data });
87  });
88
89  app.delete('/api/empleados/:id', (req, res) => {
90    const id = req.params.id;
91    res.json({ message: `Empleado ${id} eliminado` });
92  });
93
```

CRUD Productos

```
app.post('/api/productos', (req, res) => {
  const data = req.body;
  res.json({ message: 'Producto creado', data });
});

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

```
app.get('/api/productos/:id', async (req, res) => {
  const id = req.params.id;
  try {
    const result = await pool.query('SELECT * FROM producto WHERE id_producto = $1', [id]);

    if (result.rows.length === 0) {
      return res.status(404).json({ message: 'Producto no encontrado' });
    }

    res.json(result.rows[0]);
  } catch (error) {
    console.error('Error al obtener producto:', error.message);
    res.status(500).json({ error: 'Error al obtener producto' });
  }
});
```

```

app.put('/api/productos/:id', (req, res) => {
  const id = req.params.id;
  const data = req.body;
  res.json({ message: `Producto ${id} actualizado`, data });
});

app.delete('/api/productos/:id', (req, res) => {
  const id = req.params.id;
  res.json({ message: `Producto ${id} eliminado` });
});
```

CRUD pedido

```
138  app.post('/api/pedidos', (req, res) => {
139    const data = req.body;
140    res.json({ message: 'Pedido creado', data });
141  });
142
143  app.get('/api/pedidos', async (req, res) => {
144    try {
145      const result = await pool.query('SELECT * FROM pedido');
146      res.json(result.rows);
147    } catch (error) {
148      console.error('Error al obtener pedidos:', error.message);
149      res.status(500).json({ error: 'Error al obtener pedidos' });
150    }
151  });
```

```
153  app.get('/api/pedidos/:id', async (req, res) => {
154    const id = req.params.id;
155    try {
156      const result = await pool.query('SELECT * FROM pedido WHERE id_pedido = $1', [id]);
157
158      if (result.rows.length === 0) {
159        return res.status(404).json({ message: 'Pedido no encontrado' });
160      }
161
162      res.json(result.rows[0]);
163    } catch (error) {
164      console.error('Error al obtener pedido:', error.message);
165      res.status(500).json({ error: 'Error al obtener pedido' });
166    }
167  });
```

```
169
170  app.put('/api/pedidos/:id', (req, res) => {
171    const id = req.params.id;
172    const data = req.body;
173    res.json({ message: `Pedido ${id} actualizado`, data });
174  });
175
176  app.delete('/api/pedidos/:id', (req, res) => {
177    const id = req.params.id;
178    res.json({ message: `Pedido ${id} eliminado` });
179  });
```

CRUD detallepedido

```
app.post('/api/detallepedidos', (req, res) => {
  const data = req.body;
  res.json({ message: 'Detalle de pedido creado', data });
});

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

```
app.get('/api/detallepedidos/:id', async (req, res) => {
  const id = req.params.id;
  try {
    const result = await pool.query('SELECT * FROM detallepedido WHERE id_detalle = $1', [id]);

    if (result.rows.length === 0) {
      return res.status(404).json({ message: 'DetallePedido no encontrado' });
    }

    res.json(result.rows[0]);
  } catch (error) {
    console.error('Error al obtener detallepedido:', error.message);
    res.status(500).json({ error: 'Error al obtener detallepedido' });
  }
});
```

```

app.put('/api/detallepedidos/:id', (req, res) => {
  const id = req.params.id;
  const data = req.body;
  res.json({ message: `DetallePedido ${id} actualizado`, data });
});

app.delete('/api/detallepedidos/:id', (req, res) => {
  const id = req.params.id;
  res.json({ message: `DetallePedido ${id} eliminado` });
});
```

Consultas nativas

1Productos de un pedido específico

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

Este api pedirá el id dentro de un pedido para mostrar la información; con req.params se extraerá el id desde la url; hace una unión entre la tabla de producto y detallepedido usando un join donde solo se muestre el id del pedido que tenga ese id haciendo un filtro de búsqueda según el id del pedido. Con res.json se devolverá un array tipo json con los detalles de la consulta

2Productos más vendidos

```
app.get('/consultas/productos-mas-vendidos/:cantidad_min', async (req, res) => {
  const { cantidad_min } = req.params;
  const result = await pool.query(`
    SELECT p.nombre, SUM(dp.cantidad) AS total_vendido
    FROM DetallePedido dp
    JOIN Producto p ON dp.id_prod = p.id_prod
    GROUP BY p.nombre
    HAVING SUM(dp.cantidad) > $1
  `, [cantidad_min]);
  res.json(result.rows);
});
```

Para hacer este api se hace una búsqueda según un filtro de ventas que solo mostrará los productos con esa mínima cantidad de ventas donde: join junta detalle pedido con producto, agrupa según el nombre, con sum se sumá las unidades vendidas, having filtra solo los que sumen el total lo

mínimo que se pasó como parámetro para la consulta. Finalmente mostrara en un formato json los resultados

3Total de ventas por restaurante

```
252  app.get('/consultas/ventas-por-restaurante', async (req, res) => {
253    const result = await pool.query(`
254      SELECT r.nombre, SUM(p.total) AS total_ventas
255      FROM Pedido p
256      JOIN Restaurante r ON p.id_rest = r.id_rest
257      GROUP BY r.nombre
258    `);
259    res.json(result.rows);
260  });
```

No se reciben parámetros específicos, se hace mediante la unión de pedido con restaurante usando con join y agrupando según el nombre

4Pedidos por fecha

```
263  app.get('/consultas/pedidos-por-fecha/:fecha', async (req, res) => {
264    const { fecha } = req.params;
265    const result = await pool.query(`SELECT * FROM Pedido WHERE fecha = $1`, [fecha]);
266    res.json(result.rows);
267  });
268
```

Se recibe un dato de tipo fecha para hacer la consulta, se mostrará solo los que tengan la fecha que se pasó y coincida con el valor, en caso de no tener ninguno de esa fecha se devolverá un json en blanco

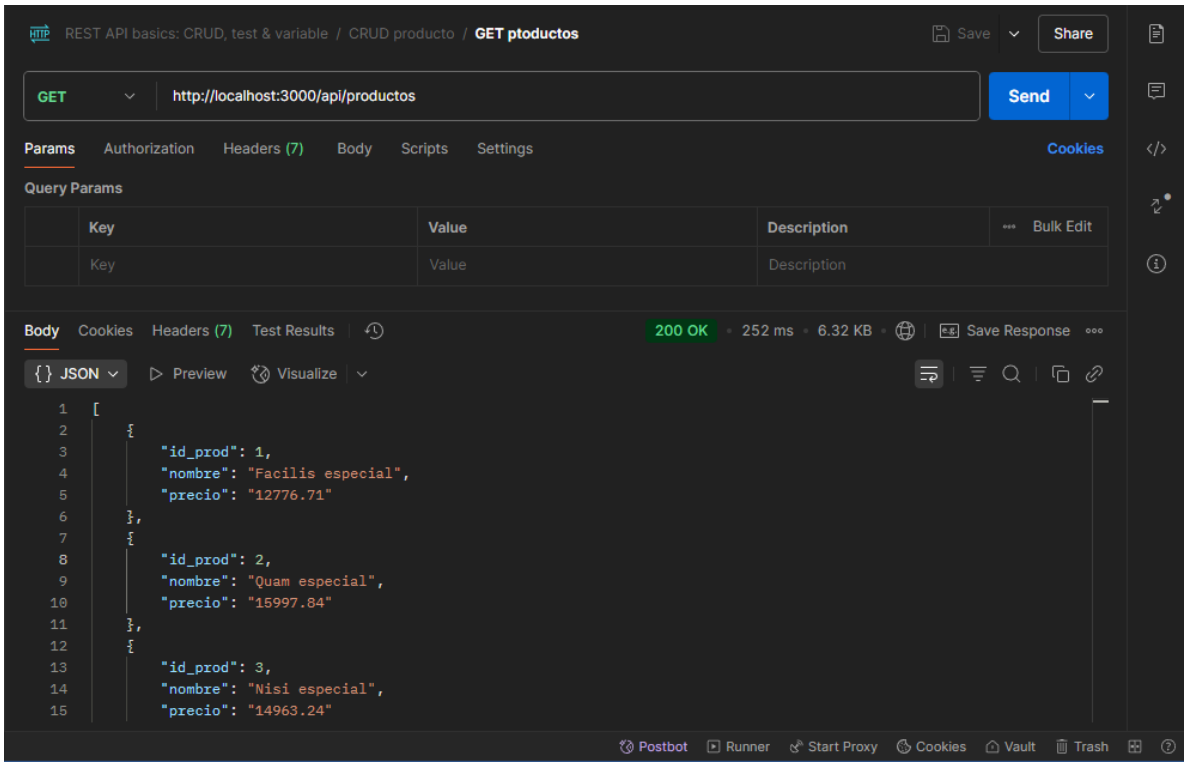
5Empleados por rol en un restaurante

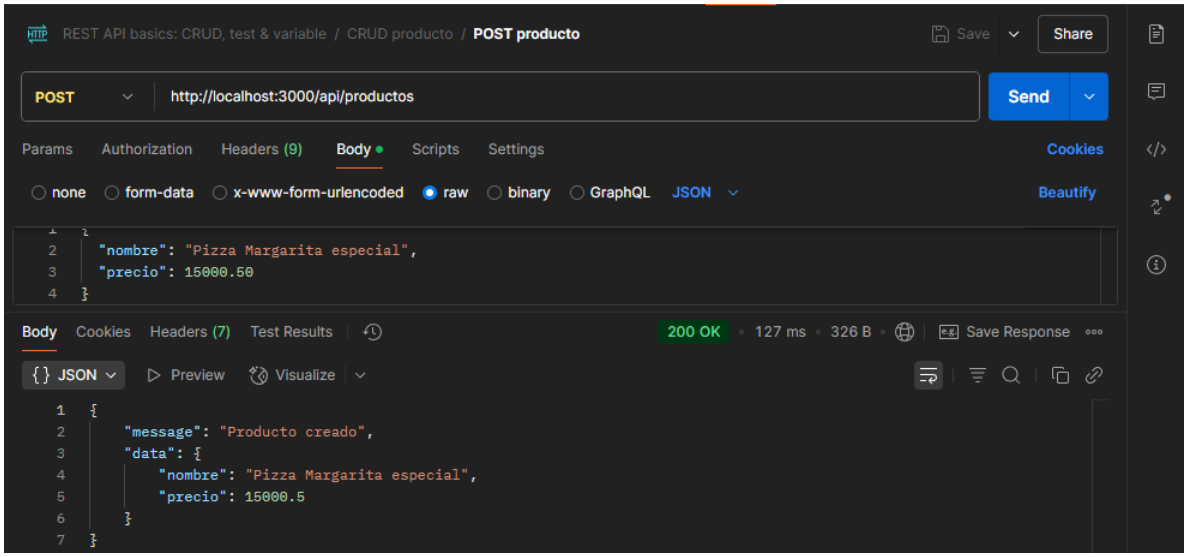
```
270  app.get('/consultas/empleados-por-rol/:id_rest/:rol', async (req, res) => {
271    const { id_rest, rol } = req.params;
272    const result = await pool.query(`
273      SELECT nombre FROM Empleado
274      WHERE id_rest = $1 AND rol = $2
275    `, [id_rest, rol]);
276    res.json(result.rows);
277  });
```

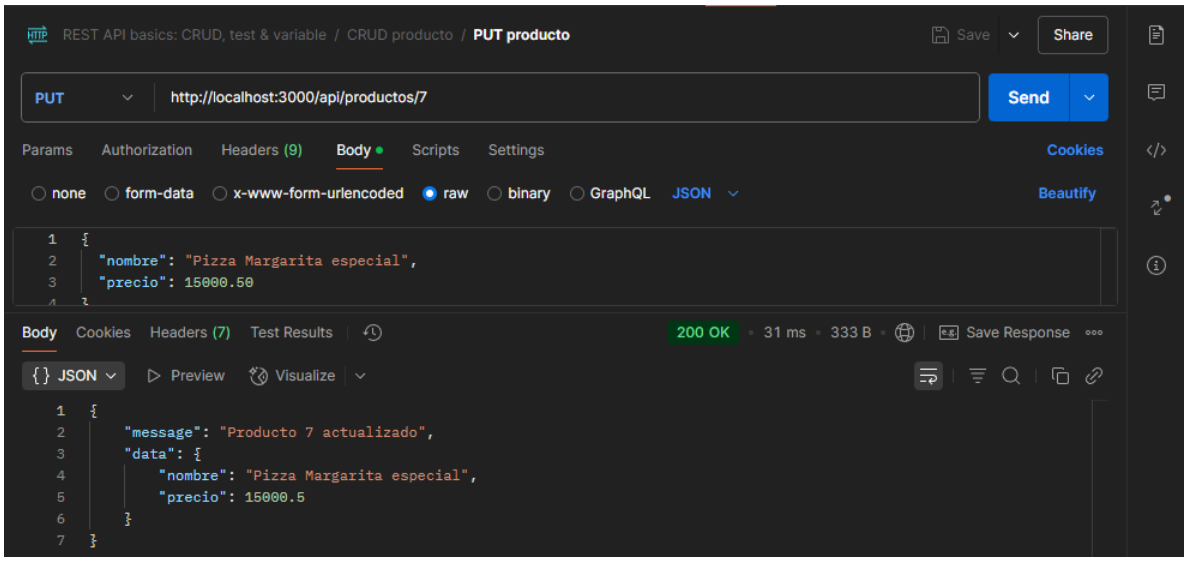
Esta api pedirá el id del empelado y el rol dado, hará una consulta en la tabla empleado donde el id y el rol sean los que se pasó

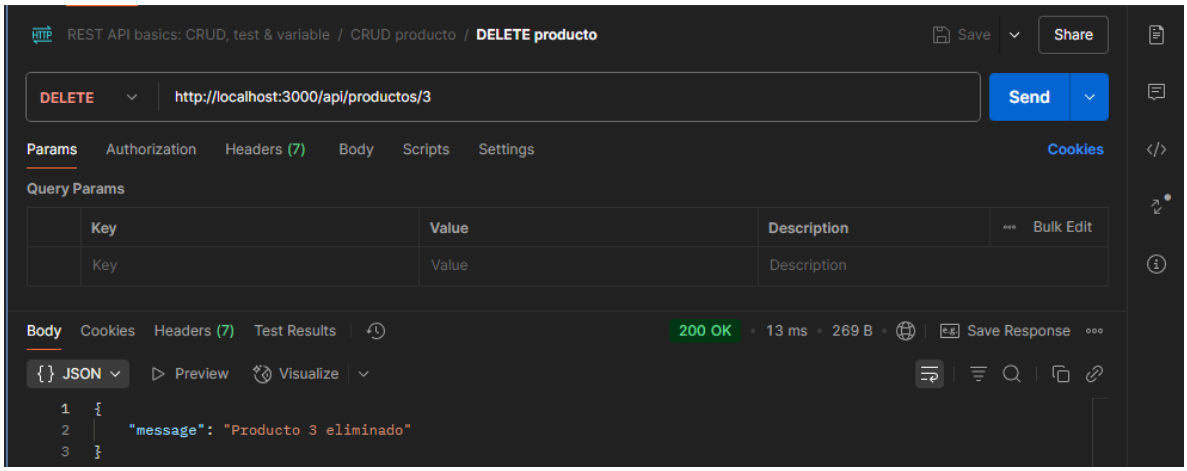
Prueba con las API creadas usando Postman

CRUD Productos

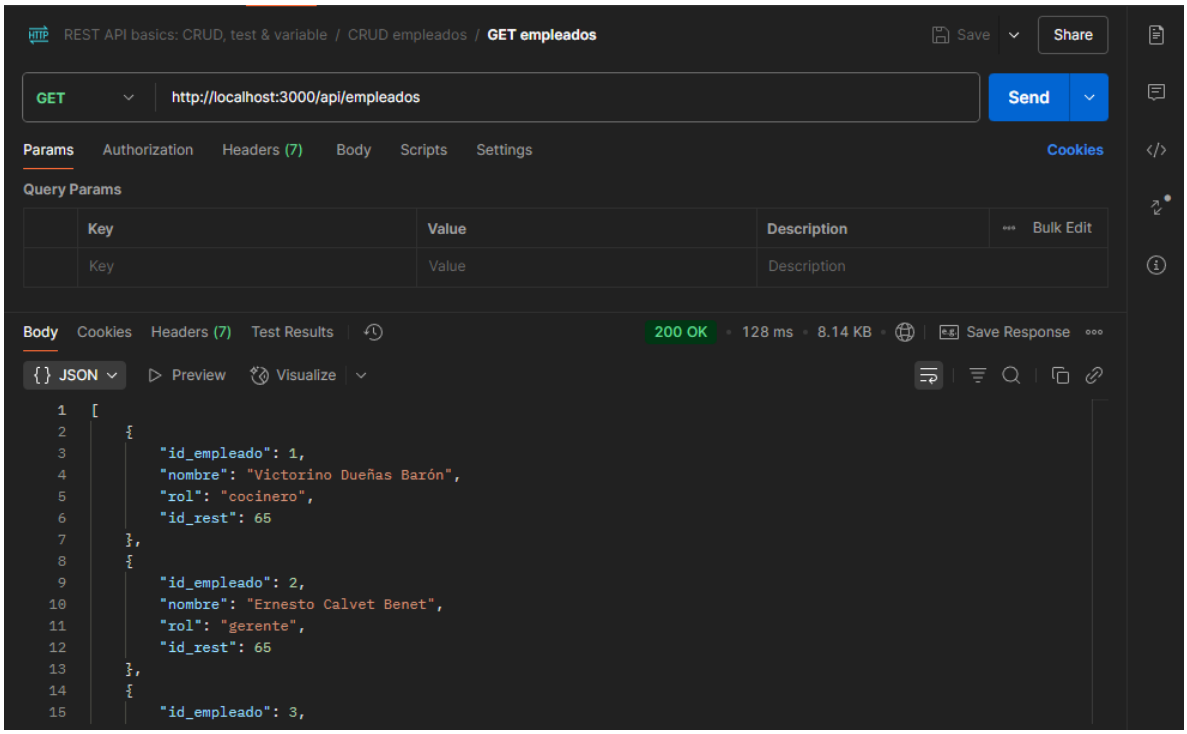


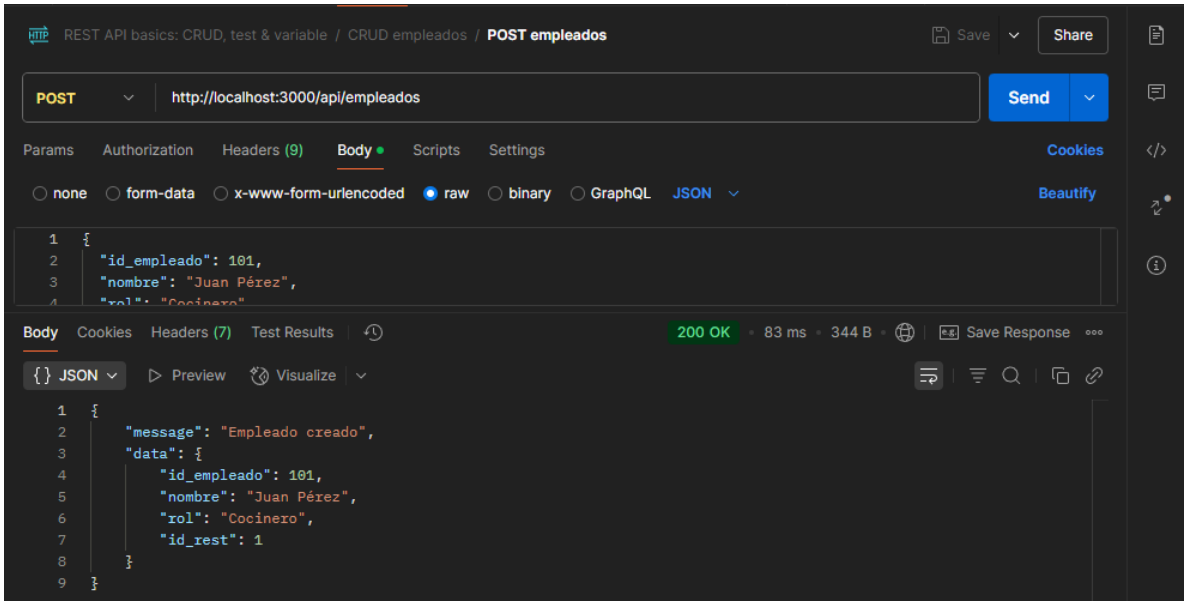


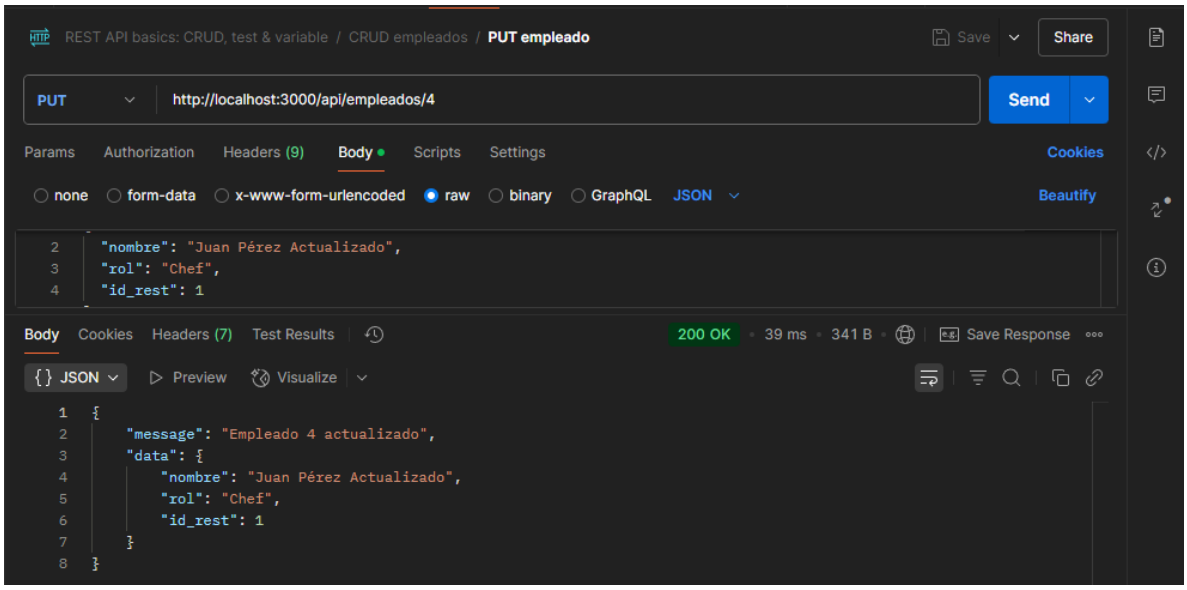


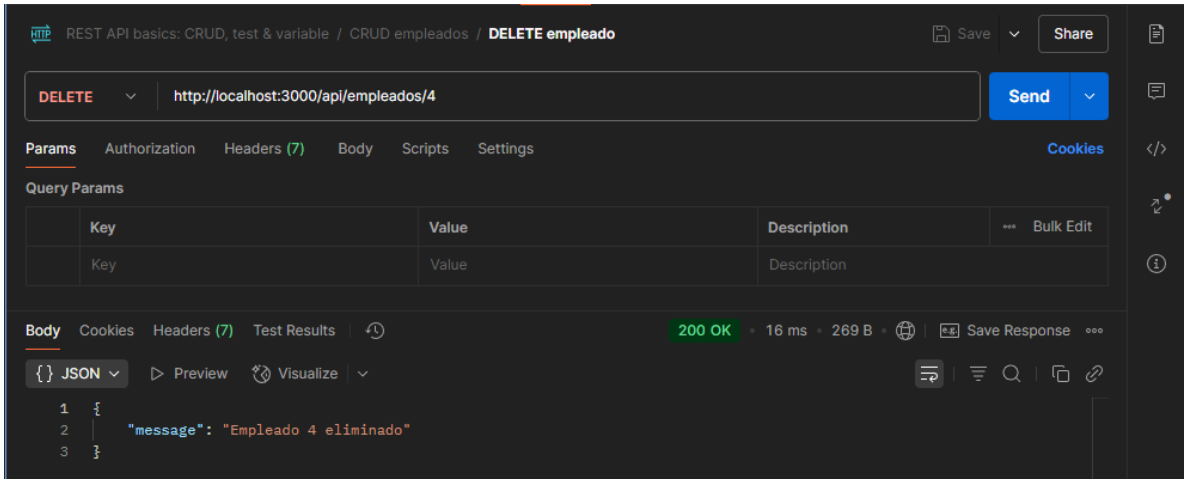


CRUD empleados

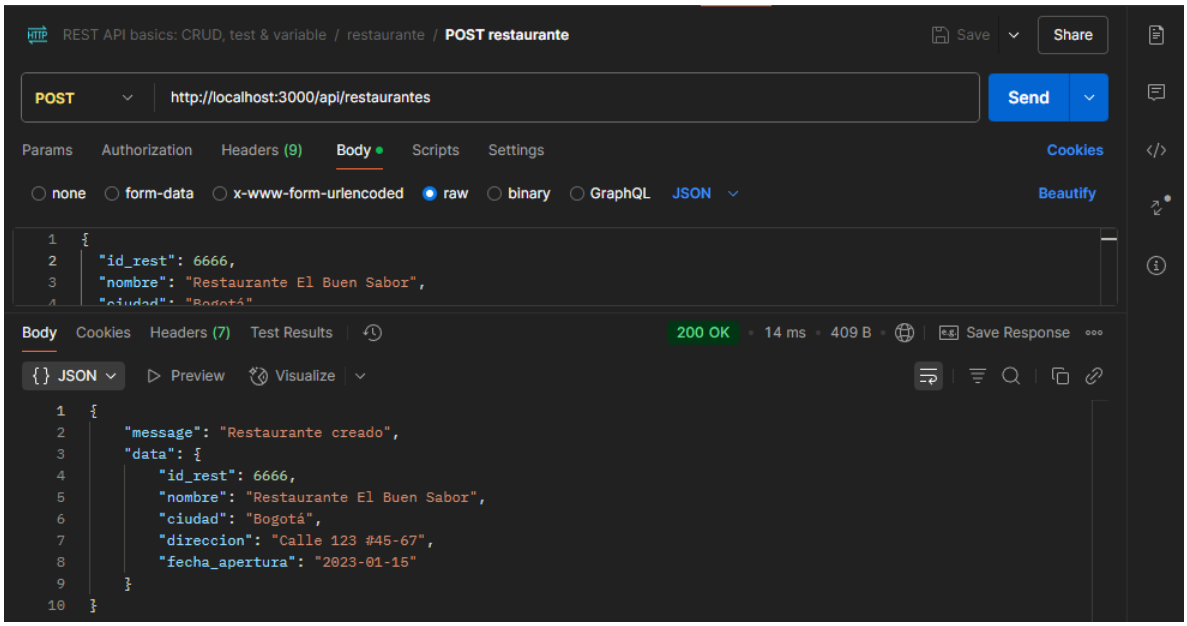

REST API basics: CRUD, test & variable / CRUD empleados / GET empleados
Save
Share
GET
http://localhost:3000/api/empleados
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
200 OK
128 ms
8.14 KB
Save Response
JSON
Preview
Visualize
[
{
"id_empleado": 1,
"nombre": "Victorino Dueñas Barón",
"rol": "cocinero",
"id_rest": 65
},
{
"id_empleado": 2,
"nombre": "Ernesto Calvet Benet",
"rol": "gerente",
"id_rest": 65
},
{
"id_empleado": 3,

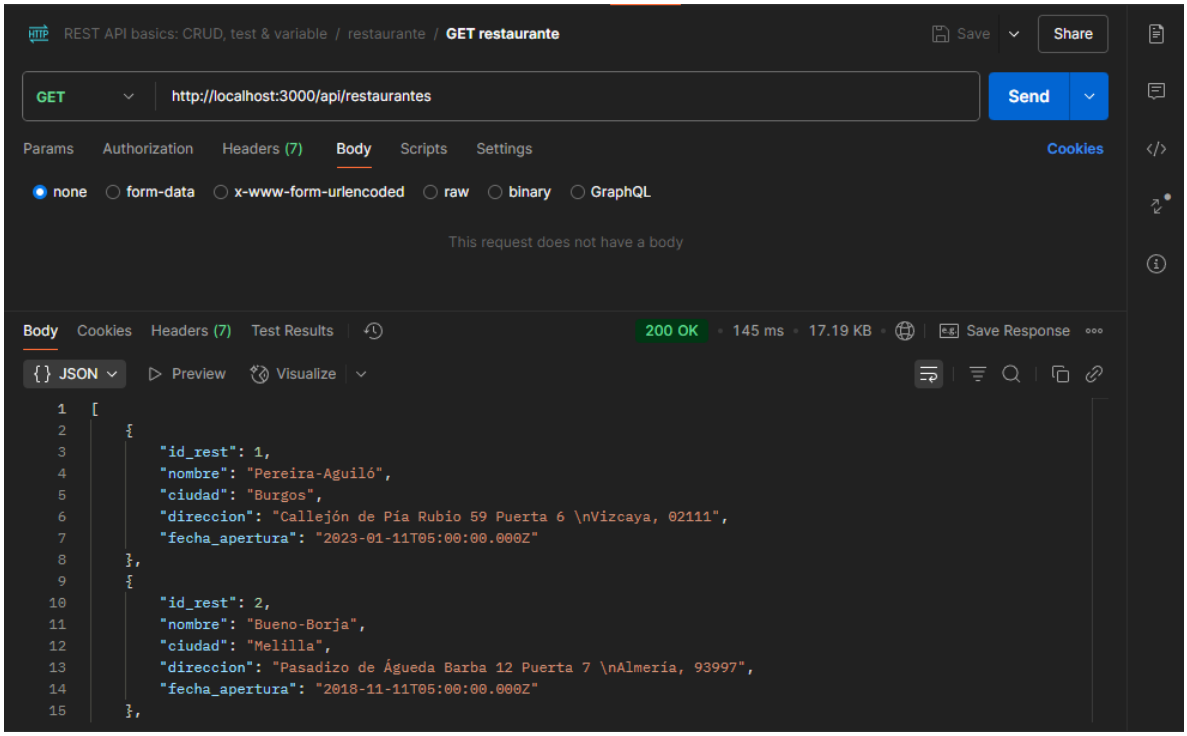

REST API basics: CRUD, test & variable / CRUD empleados / POST empleados
Save
Share
POST
http://localhost:3000/api/empleados
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id_empleado": 101,
"nombre": "Juan Pérez",
Body
Cookies
Headers (7)
Test Results
200 OK
83 ms
344 B
Save Response
JSON
Preview
Visualize
{
"message": "Empleado creado",
"data": {
"id_empleado": 101,
"nombre": "Juan Pérez",
"rol": "Cocinero",
"id_rest": 1
}
}

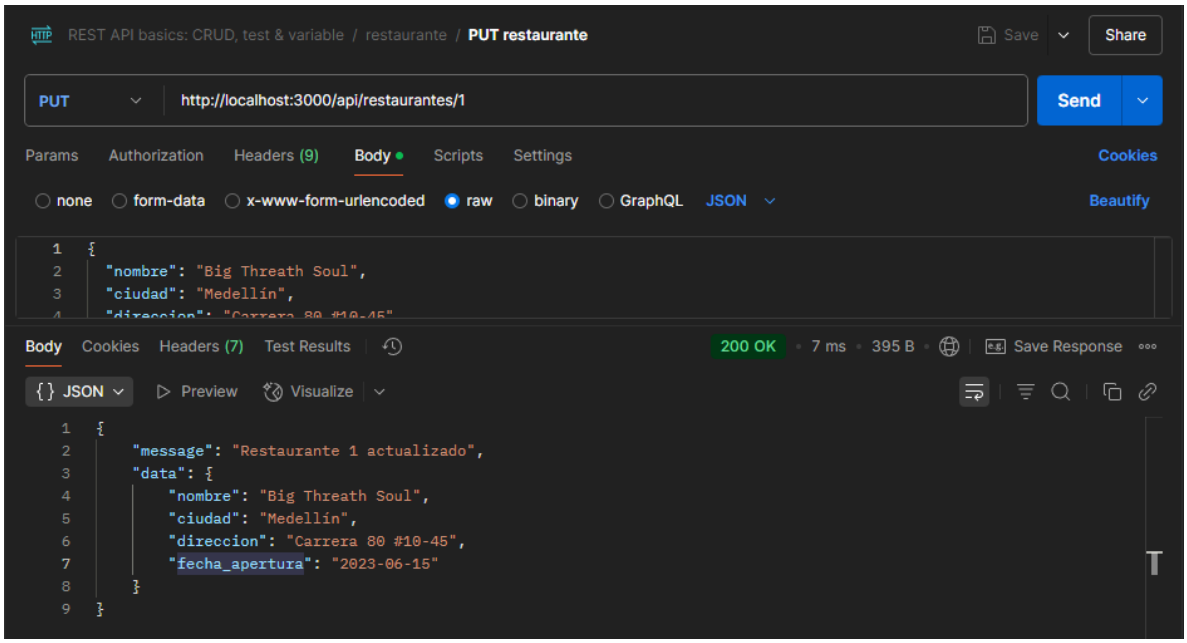


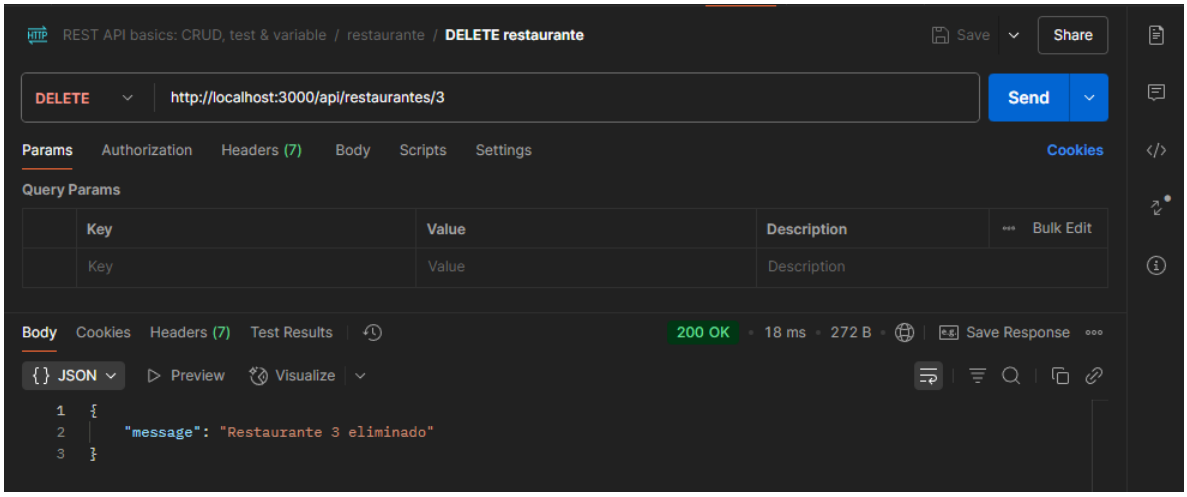


CRUD restaurantes

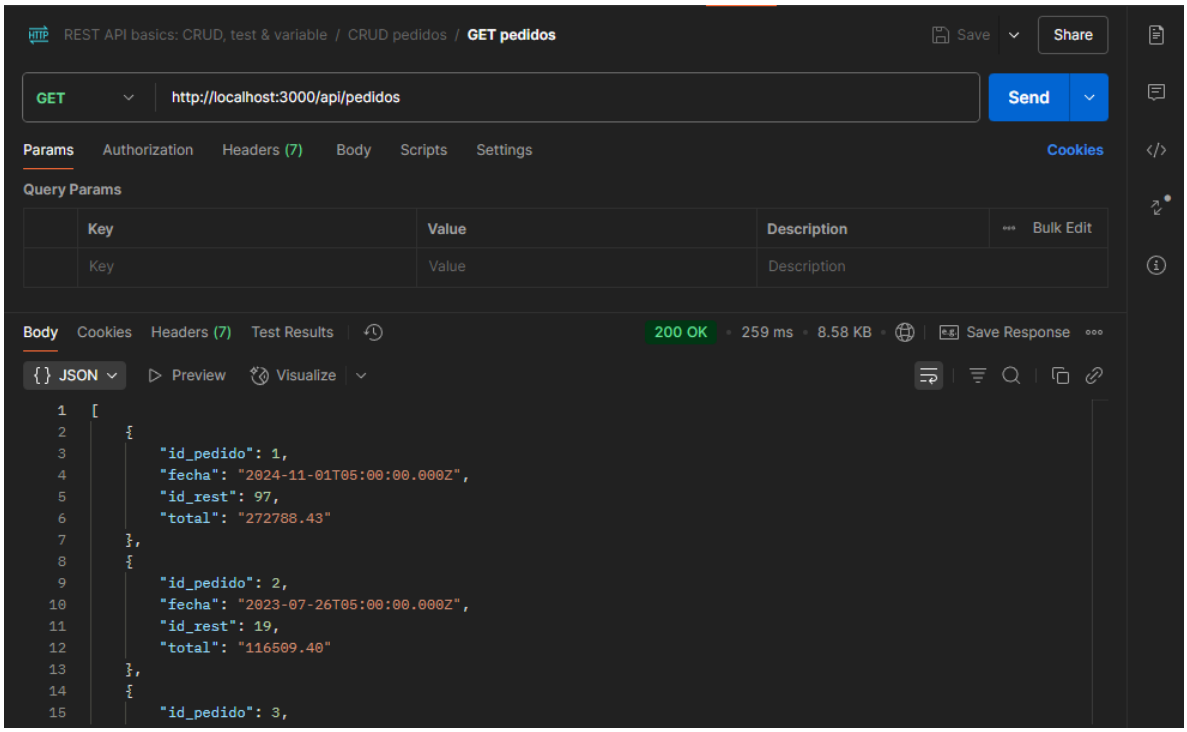

REST API basics: CRUD, test & variable / restaurante / POST restaurante
Save
Share
POST
http://localhost:3000/api/restaurantes
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id_rest": 6666,
"nombre": "Restaurante El Buen Sabor",
Body
Cookies
Headers (7)
Test Results
200 OK
14 ms
409 B
Save Response
JSON
Preview
Visualize
{
"message": "Restaurante creado",
"data": {
"id_rest": 6666,
"nombre": "Restaurante El Buen Sabor",
"ciudad": "Bogotá",
"direccion": "Calle 123 #45-67",
"fecha_apertura": "2023-01-15"
}
}

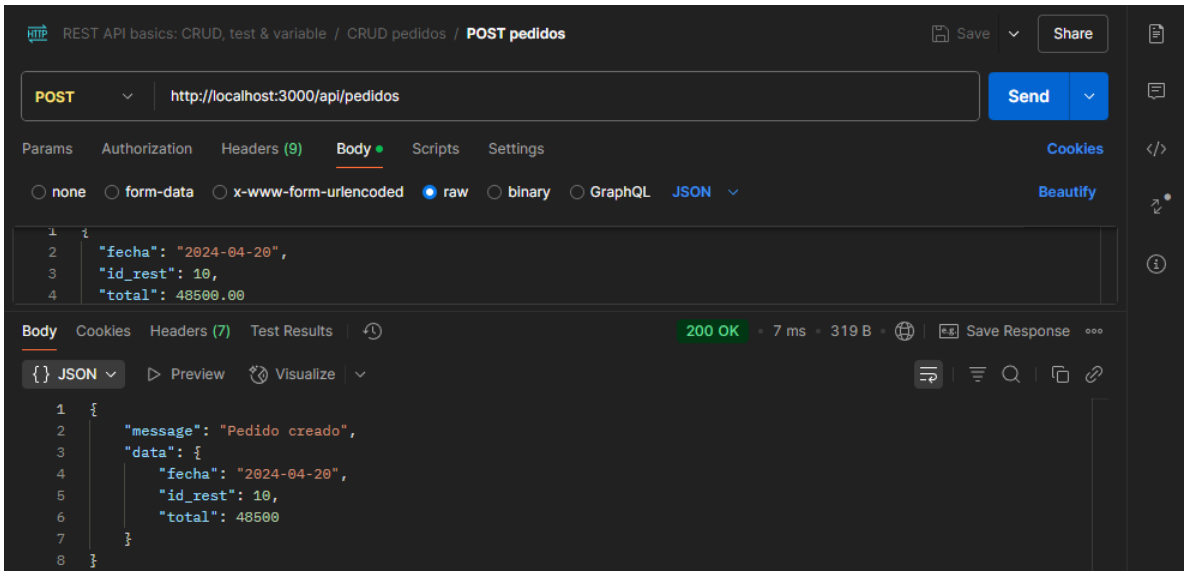

REST API basics: CRUD, test & variable / restaurante / GET restaurante
Save
Share
GET
http://localhost:3000/api/restaurantes
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body
Body
Cookies
Headers (7)
Test Results
200 OK
145 ms
17.19 KB
Save Response
JSON
Preview
Visualize
[
{
"id_rest": 1,
"nombre": "Pereira-Aguiló",
"ciudad": "Burgos",
"direccion": "Callejón de Pía Rubio 59 Puerta 6 \nVizcaya, 02111",
"fecha_apertura": "2023-01-11T05:00:00.000Z"
},
{
"id_rest": 2,
"nombre": "Bueno-Borja",
"ciudad": "Melilla",
"direccion": "Pasadizo de Águeda Barba 12 Puerta 7 \nAlmeria, 93997",
"fecha_apertura": "2018-11-11T05:00:00.000Z"
},

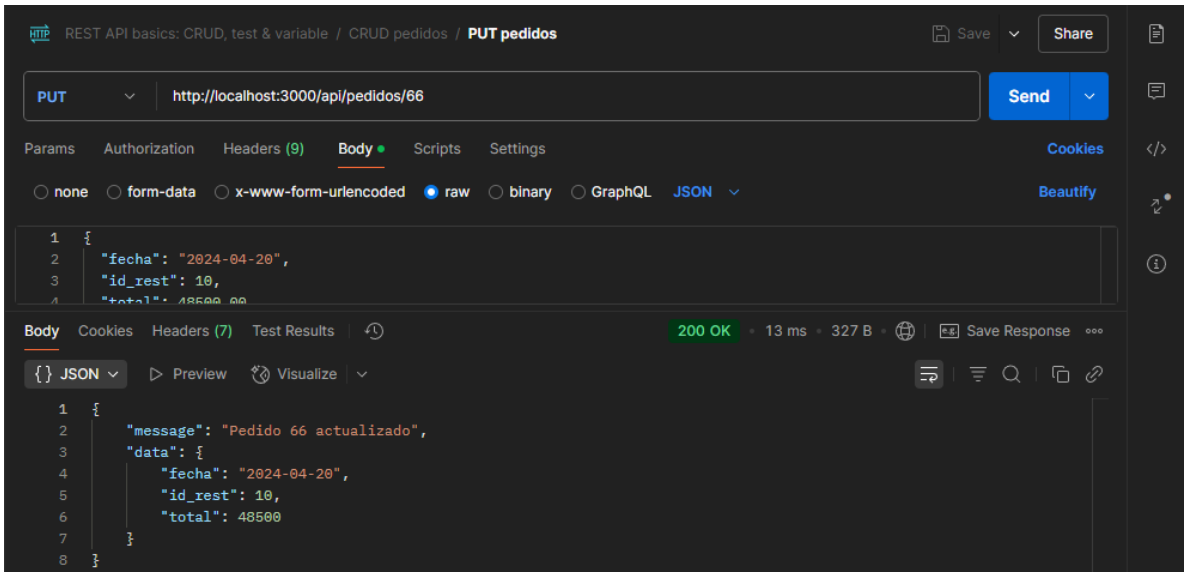


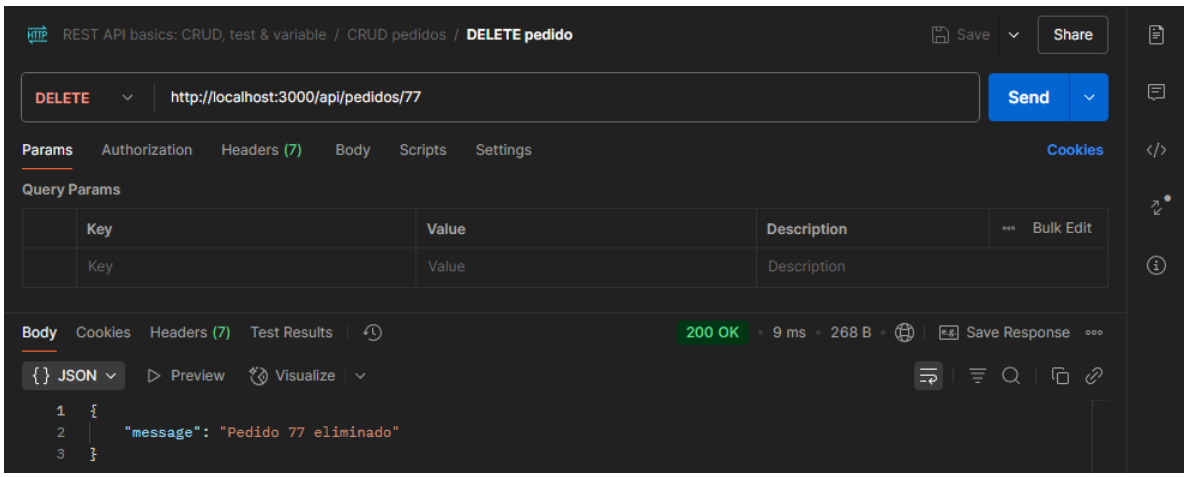


CRUD pedidos

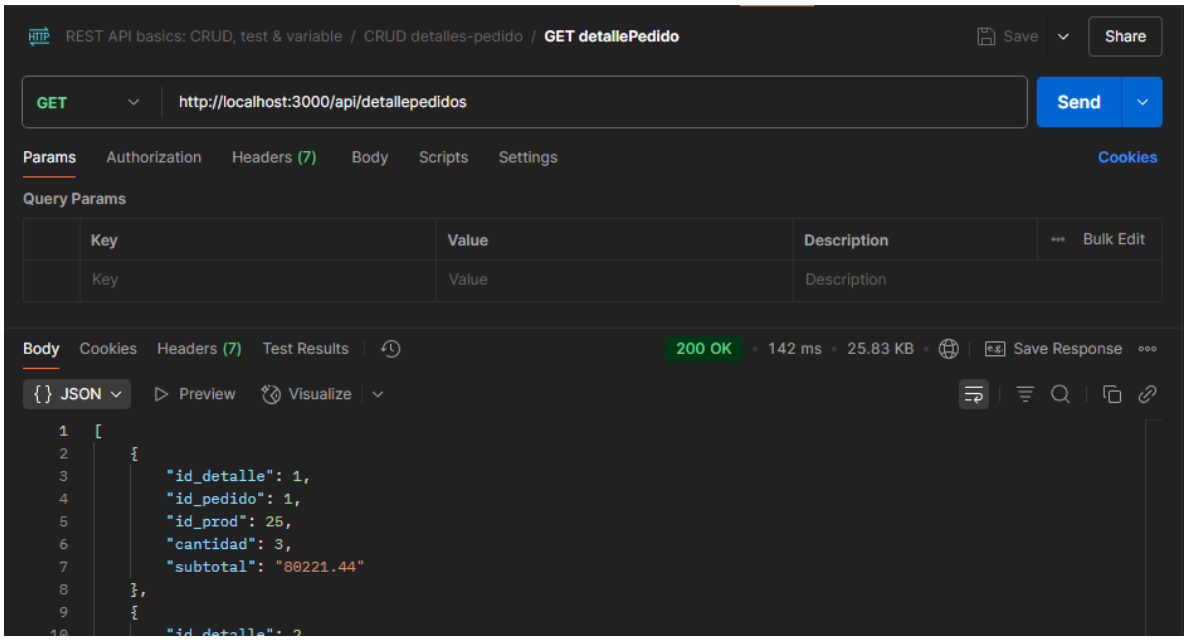

REST API basics: CRUD, test & variable / CRUD pedidos / GET pedidos
Save
Share
GET
http://localhost:3000/api/pedidos
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
259 ms
8.58 KB
Save Response
JSON
Preview
Visualize
[
{
"id_pedido": 1,
"fecha": "2024-11-01T05:00:00.000Z",
"id_rest": 97,
"total": "272788.43"
},
{
"id_pedido": 2,
"fecha": "2023-07-26T05:00:00.000Z",
"id_rest": 19,
"total": "116509.40"
},
{
"id_pedido": 3,

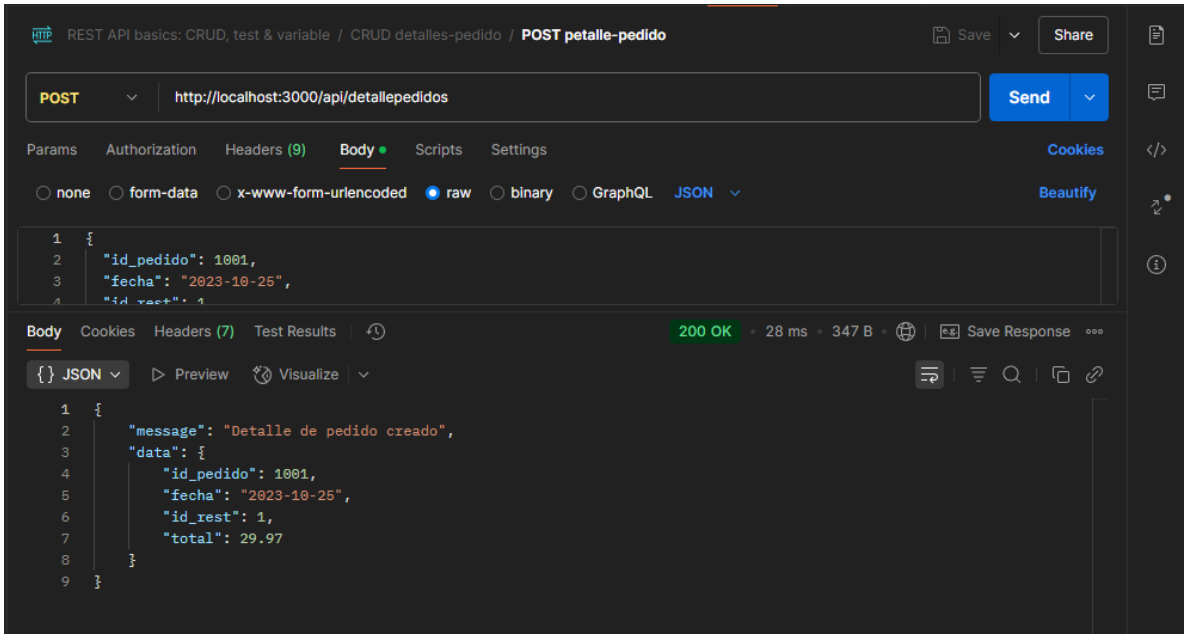

REST API basics: CRUD, test & variable / CRUD pedidos / POST pedidos
Save
Share
POST
http://localhost:3000/api/pedidos
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"fecha": "2024-04-20",
"id_rest": 10,
"total": 48500.00
Body
Cookies
Headers (7)
Test Results
200 OK
7 ms
319 B
Save Response
JSON
Preview
Visualize
{
"message": "Pedido creado",
"data": {
"fecha": "2024-04-20",
"id_rest": 10,
"total": 48500
}
}

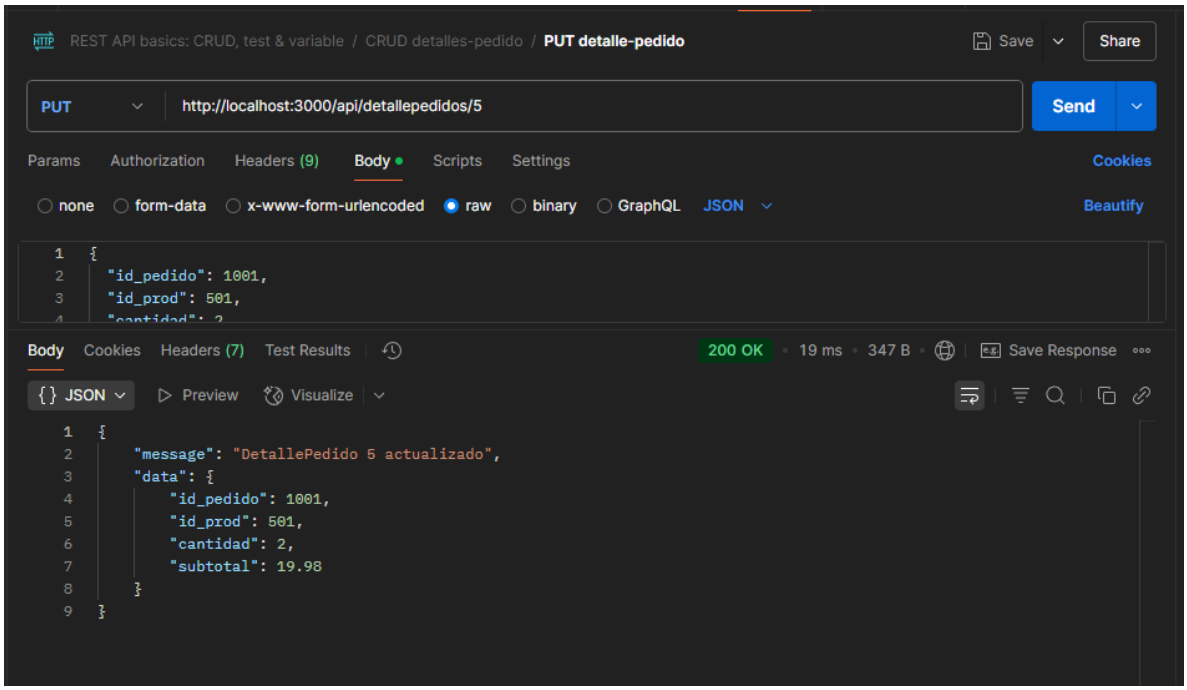


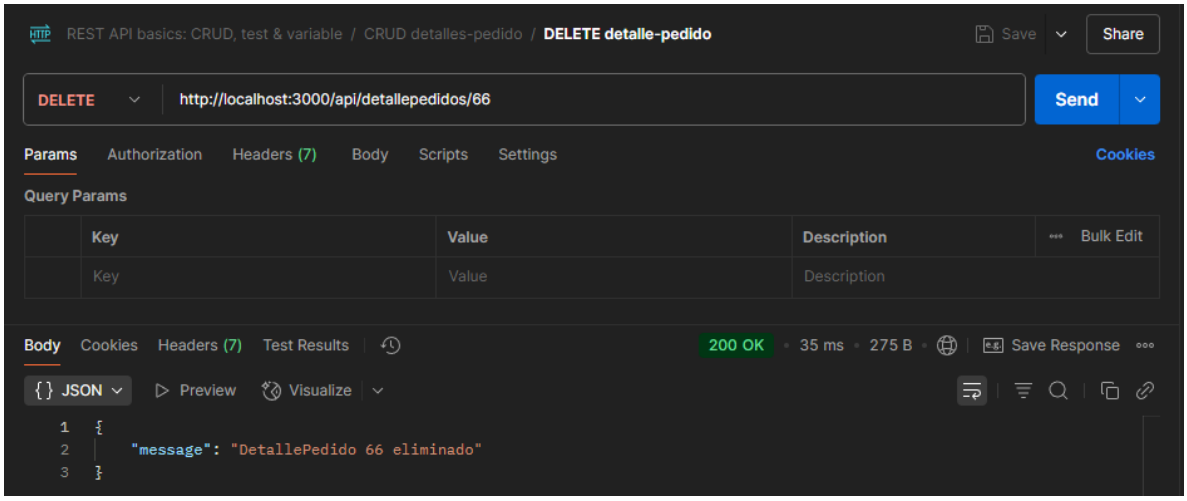


CRUD detallepedido


REST API basics: CRUD, test & variable / CRUD detalles-pedido / GET detallePedido
Save
Share
GET
http://localhost:3000/api/detallepedidos
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
200 OK
142 ms
25.83 KB
Save Response
JSON
Preview
Visualize
[
{
"id_detalle": 1,
"id_pedido": 1,
"id_prod": 25,
"cantidad": 3,
"subtotal": "80221.44"
},
{

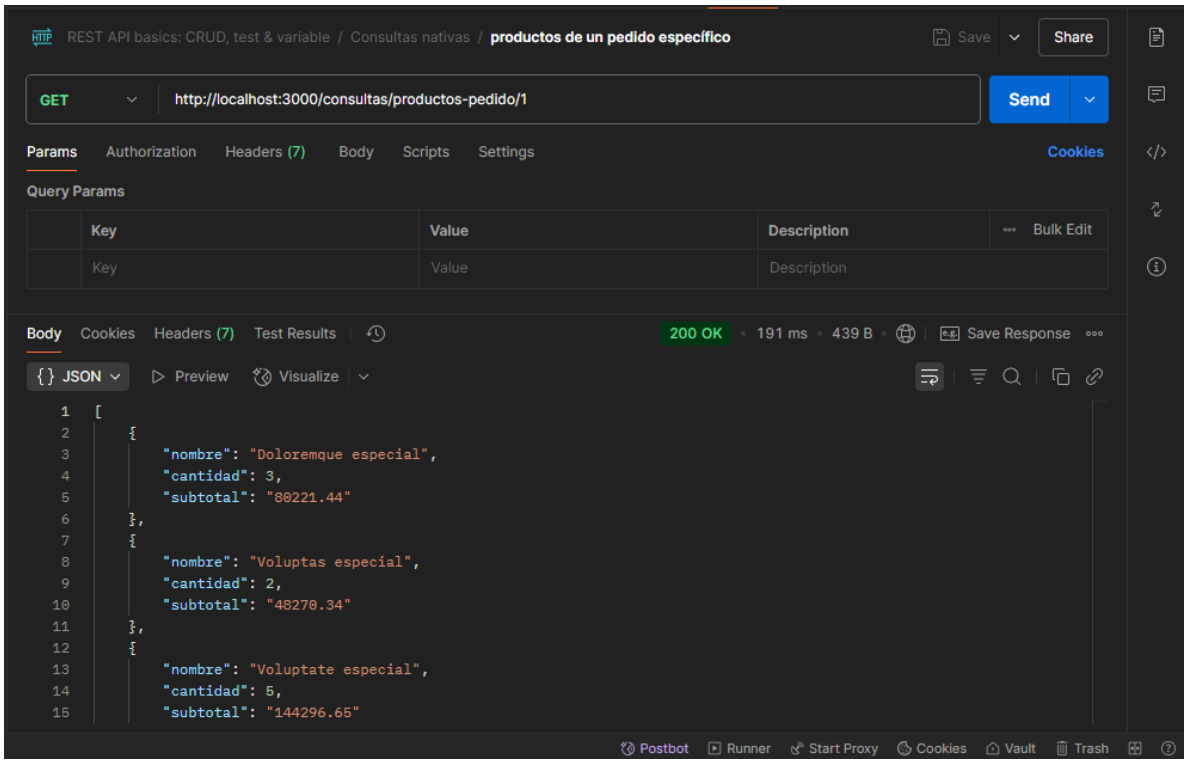

REST API basics: CRUD, test & variable / CRUD detalles-pedido / POST petalle-pedido
Save
Share
POST
http://localhost:3000/api/detallepedidos
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id_pedido": 1001,
"fecha": "2023-10-25",
Body
Cookies
Headers (7)
Test Results
200 OK
28 ms
347 B
Save Response
JSON
Preview
Visualize
{
"message": "Detalle de pedido creado",
"data": {
"id_pedido": 1001,
"fecha": "2023-10-25",
"id_rest": 1,
"total": 29.97
}
}



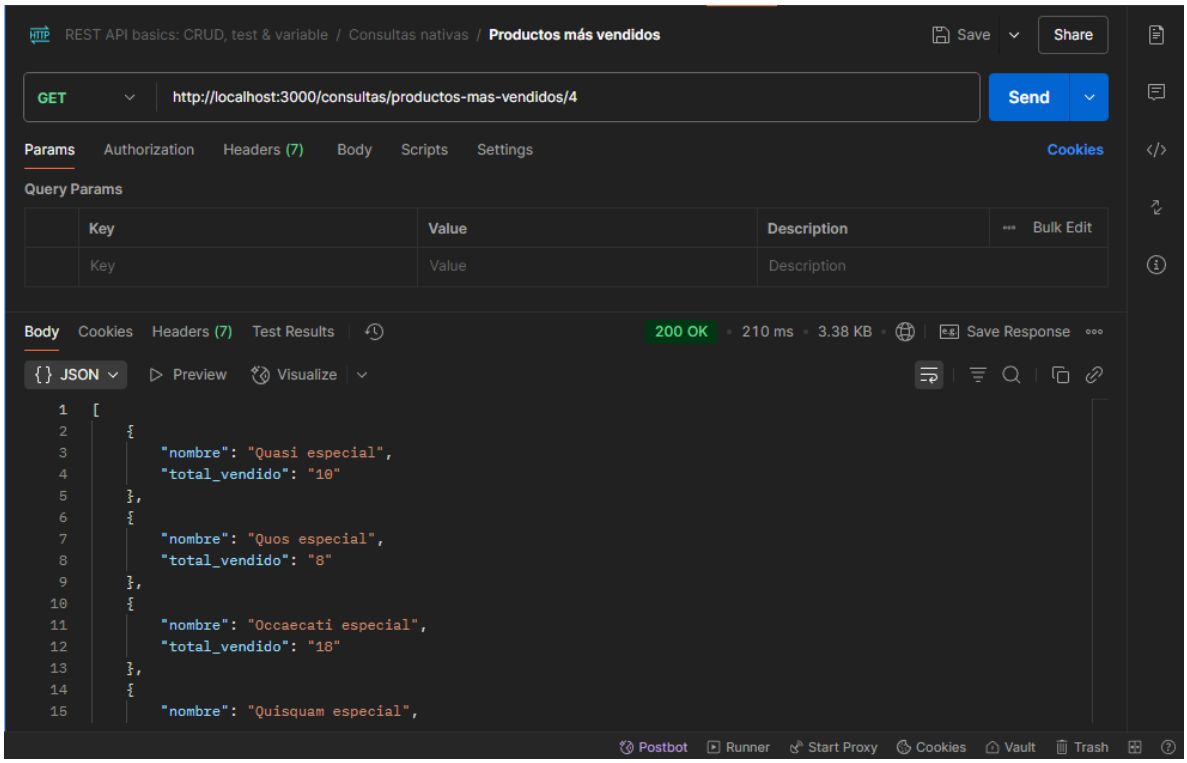


CONSULTAS NATIVAS

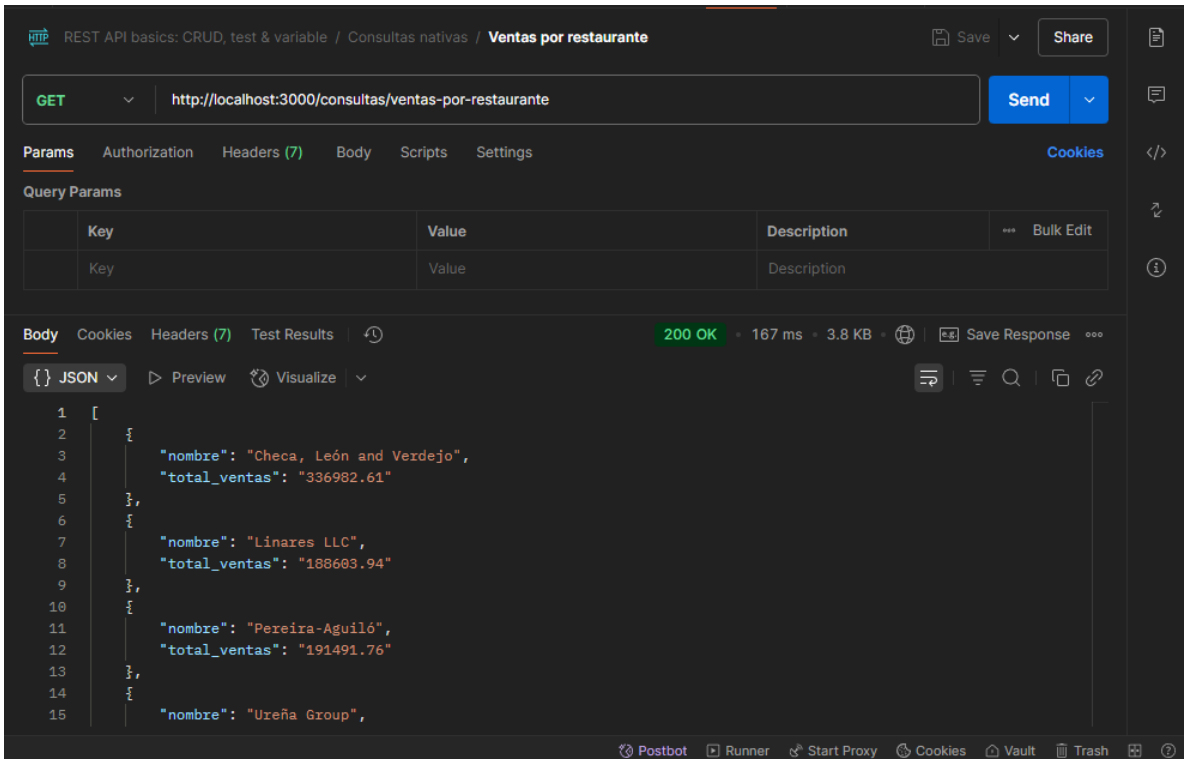
1Productos de un pedido específico



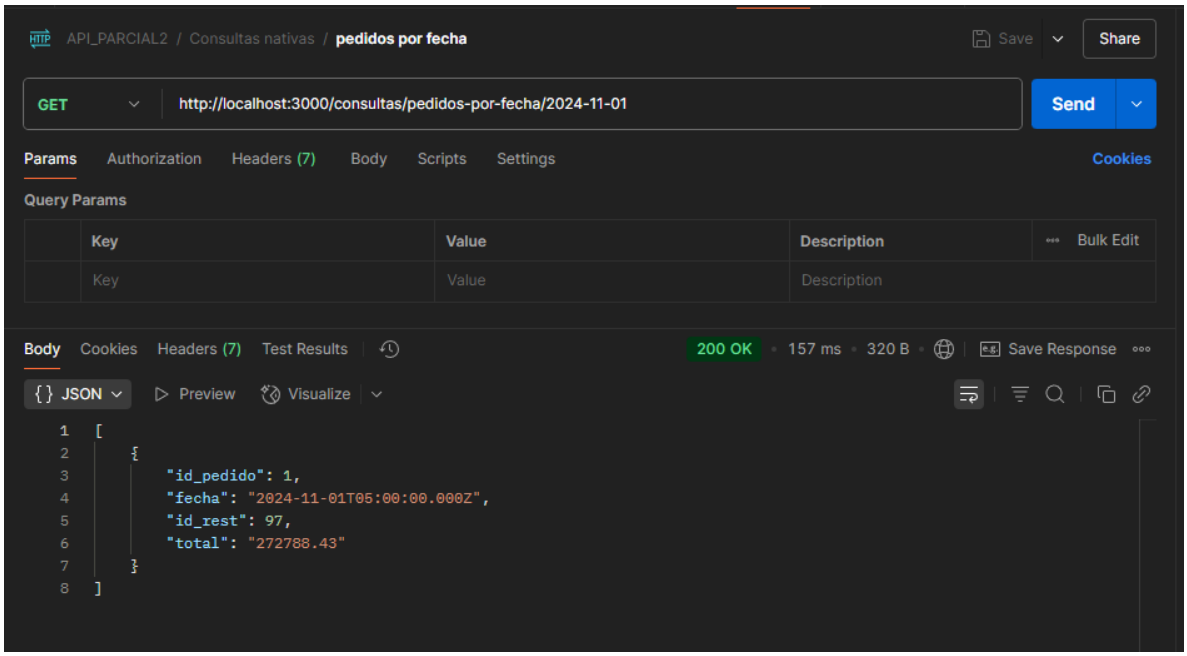
2Productos más vendidos



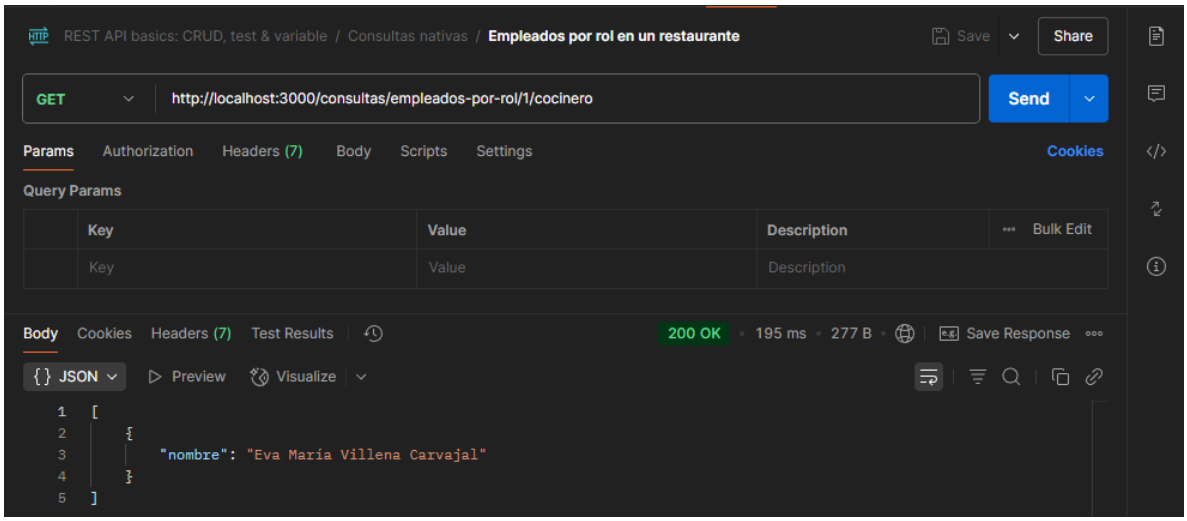
3Total de ventas por restaurante



4Pedidos por fecha



5Empleados por rol en un restaurante


REST API basics: CRUD, test & variable / Consultas nativas / Empleados por rol en un restaurante
Save
Share
GET
http://localhost:3000/consultas/empleados-por-rol/1/cocinero
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
200 OK
195 ms
277 B
Save Response
JSON
Preview
Visualize
1 [
2 {
3 "nombre": "Eva María Villena Carvajal"
4 }
5]