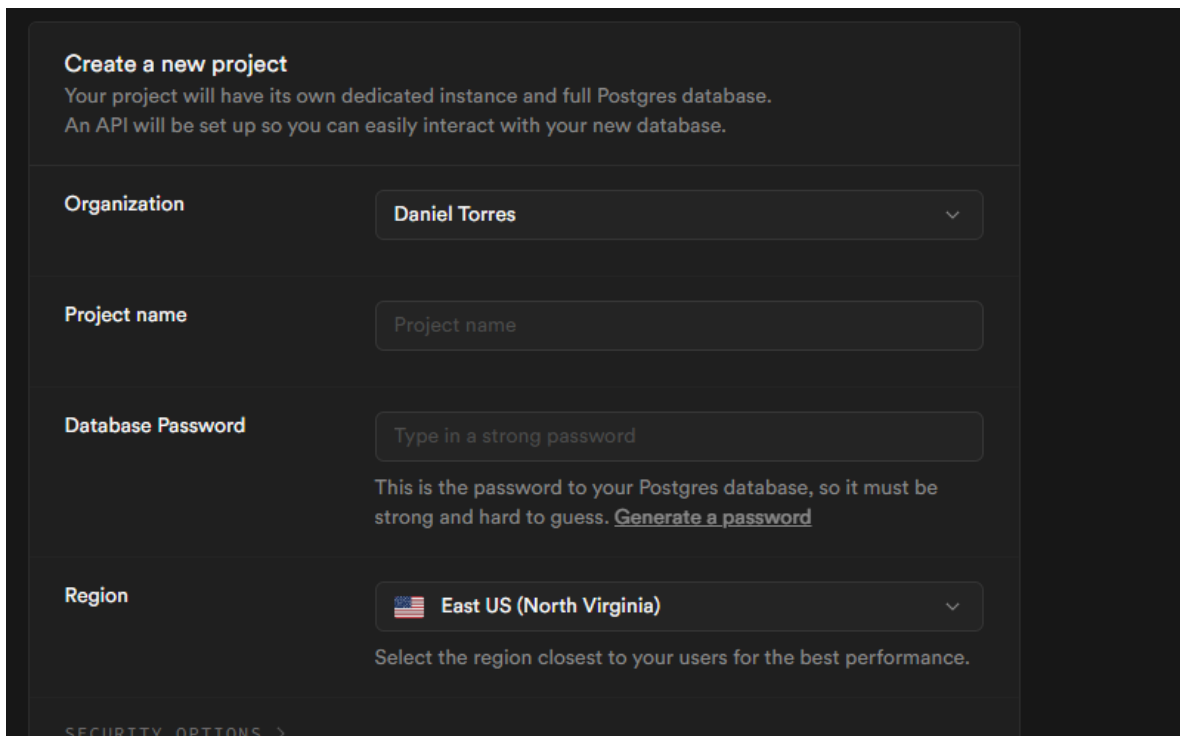




En el ejercicio de hoy realizaremos lo visto en las clases pasadas esta vez trabajando con dos tablas, así que lo primero que vamos a hacer es ingresar a supabase y crear un nuevo proyecto.

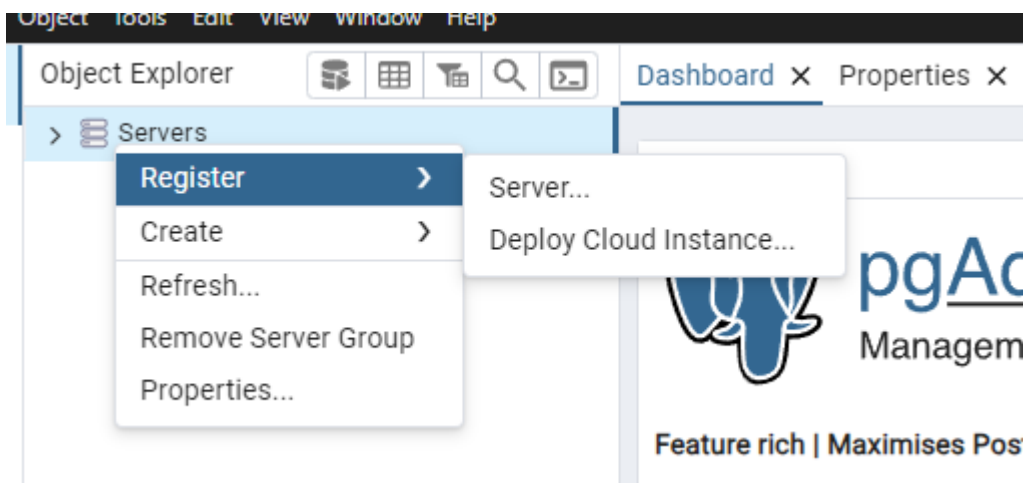


The screenshot shows the 'Create a new project' form in Supabase. It includes the following fields and options:

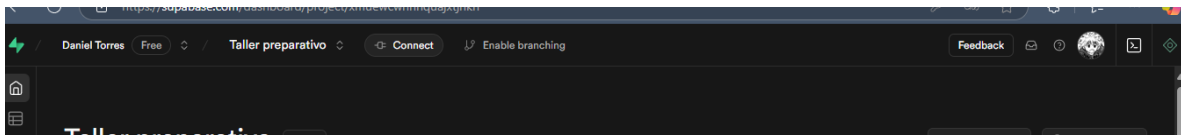
- Organization:** A dropdown menu with 'Daniel Torres' selected.
- Project name:** A text input field with the placeholder 'Project name'.
- Database Password:** A text input field with the placeholder 'Type in a strong password'. Below it, a note states: 'This is the password to your Postgres database, so it must be strong and hard to guess. [Generate a password](#)'.
- Region:** A dropdown menu with 'East US (North Virginia)' selected. Below it, a note states: 'Select the region closest to your users for the best performance.'

At the bottom left, there is a link for 'SECURITY OPTIONS >'.

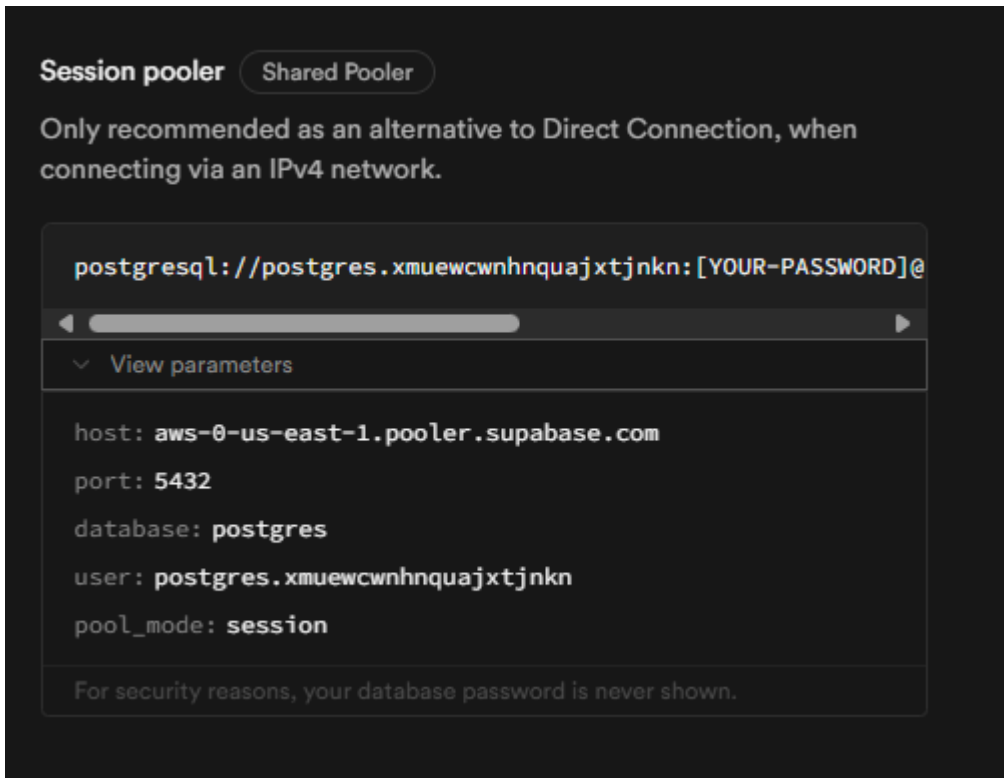
Acá le otorgaremos el nombre a nuestro proyecto, luego de esto lo conectaremos con nuestro pgAdmin4, una vez entremos a nuestro pgAdmin 4 nos iremos a la parte izquierda donde dice servers en donde le daremos click derecho y le daremos a registrar server como se ve en la imagen.



Luego seguiremos en la conexión en pgadmin donde nos iremos al apartado de connect, en donde estarán los datos para continuar con nuestra conexión.



Ahí encontraremos los siguientes datos.



Estos datos son los que nos ayudaran a nuestra conexión, y los ingresaremos acá.

Register - Server

General

Connection

Parameters

SSH Tunnel

Advanced

Tags

Host name/address

aws-0-us-east-1.pooler.supabase.com

Port

5432

Maintenance database

postgres

Username

postgres.xmuewcwnhnquajxtjkn

Kerberos authentication?

☐

Password

.....

Save password?

☐

Role

Service

i

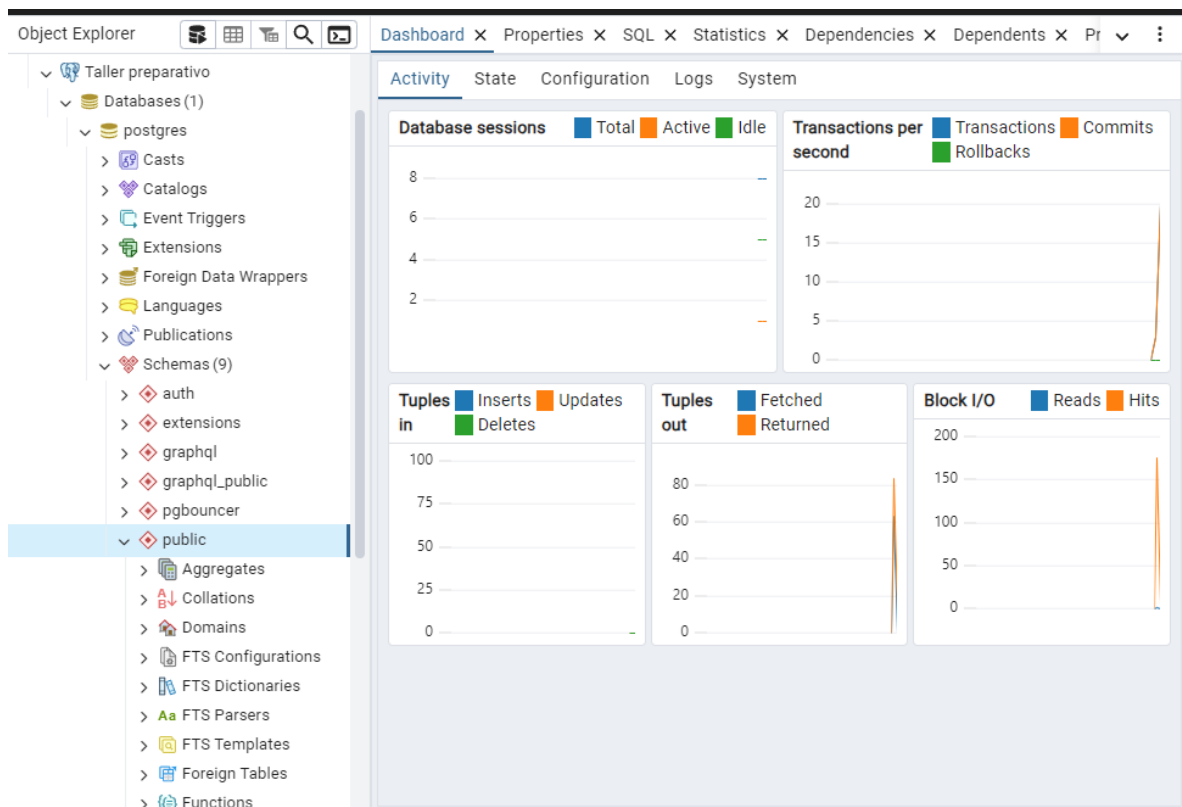
?

Close

Reset

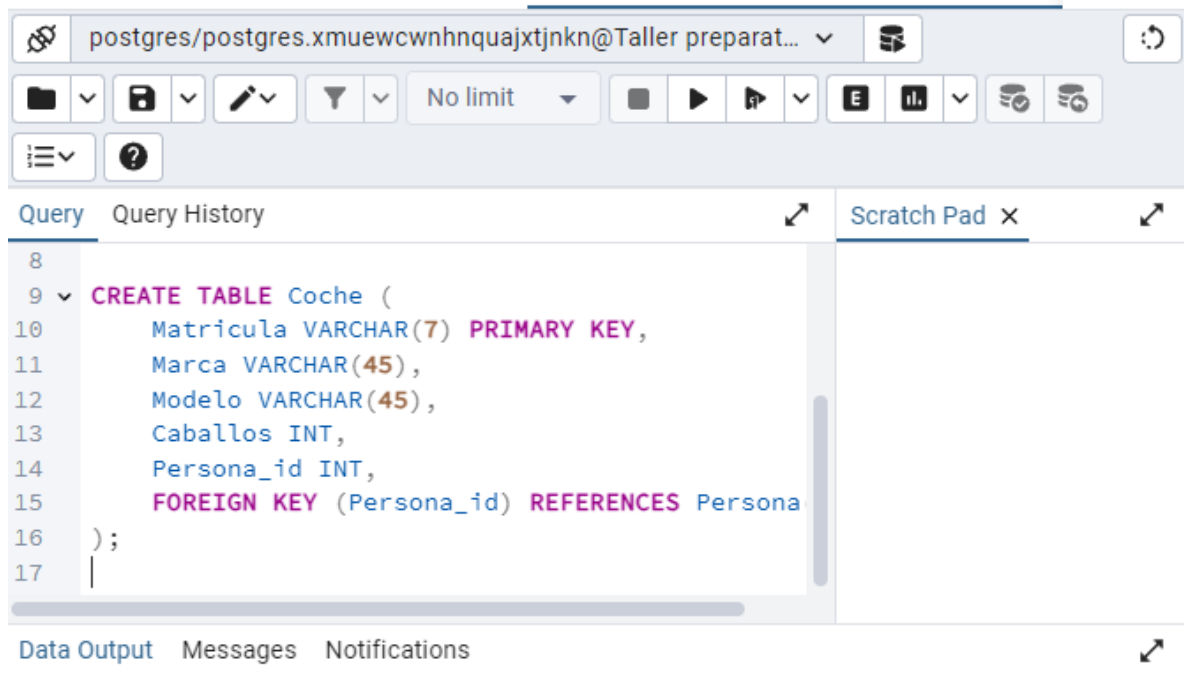
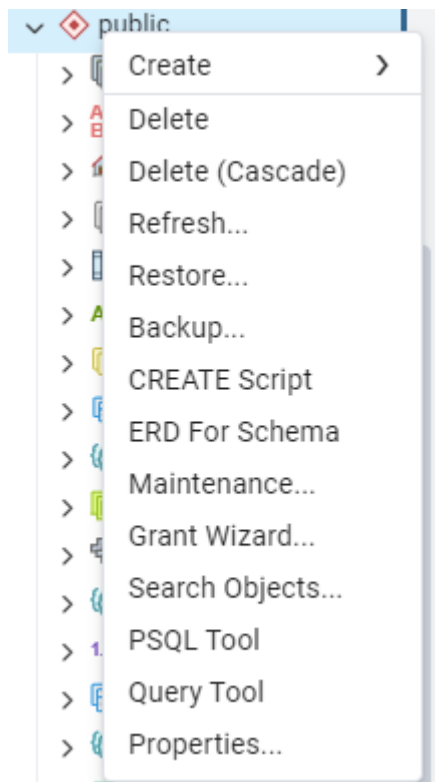
Save

Y al darle al botón save veremos que ya tenemos nuestra conexión de manera correcta.

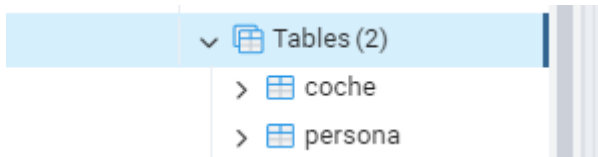


Ahora vamos a crear las tablas que se nos piden en el ejercicio.

Para ello nos vamos a public y le damos click derecho en donde le daremos click a Query Tools, en donde se nos abrirá una pestaña donde crearemos las tablas.



Y así ya estarían creadas nuestras tablas como podemos ver a continuación.



Ahora vamos a ingresar los registros correspondientes, en este caso se nos pidió que hiciéramos 100, para ello seguiremos el mismo paso que al crear las tablas, nos vamos a schemas, click derecho y query tool y ahí ingresaremos nuestros datos.

```
1  -- Insertar 100 coches ficticios relacionados con
2  INSERT INTO Coche (Matricula, Marca, Modelo, Caba
3  SELECT
4      'MAT' || LPAD(s::text, 4, '0'),
5      'Marca_' || (s % 10),
6      'Modelo_' || (s % 15),
7      100 + (s % 100),
8      (RANDOM() * 99 + 1)::INT
9  FROM generate_series(1, 100) s;
```

```
1  -- Insertar 100 personas ficticias
2  INSERT INTO Persona (Nombre, Apellido1, Apellido2,
3  SELECT
4      'Nombre' || s,
5      'Apellido1_' || s,
6      'Apellido2_' || s,
7      LPAD(s::text, 9, '0')
8  FROM generate_series(1, 100) s;
```

Una vez hecho esto nos dirigiremos a nuestro visual studia y a nuestro postman en donde seguiremos con el ejercicio.

En visual actauklizaremos nuestra conexión en db.js con los datos de nuestra actual base de datos.

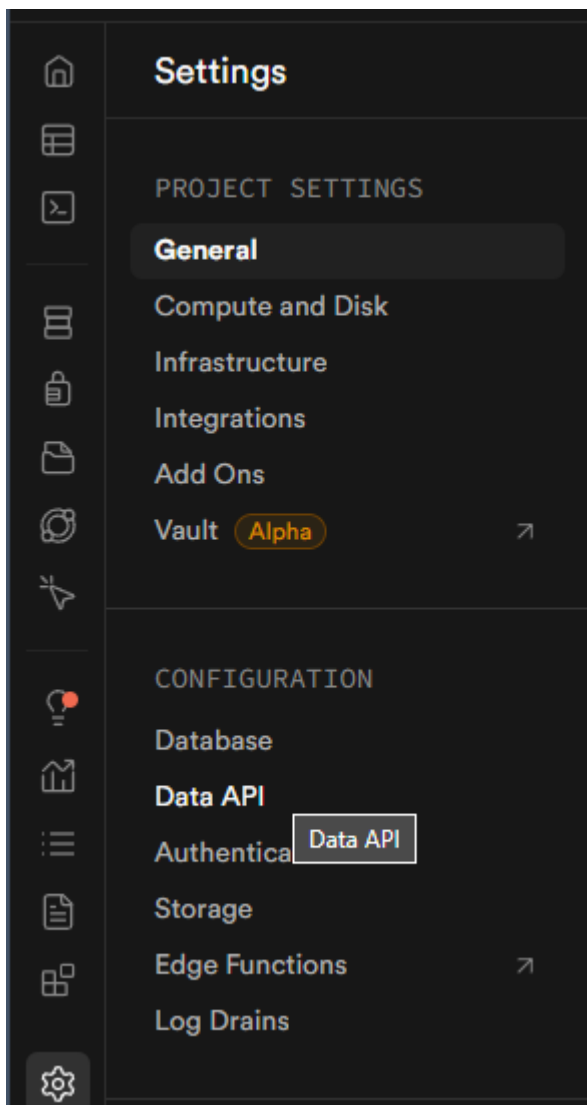
```

1  const { Client } = require('pg');
2
3  const client = new Client({
4    host: 'aws-0-us-east-1.pooler.supabase.com',
5    port: 5432,
6    user: 'postgres.xmuewcwnhnquajxtjnkn',
7    password: 'Vang16htY3HKYcHb',
8    database: 'postgres',
9    ssl: { rejectUnauthorized: false },
10  });
11
12  client.connect()
13    .then(() => console.log('Conectado a PostgreSQL con pg'))
14    .catch(err => console.error('Error al conectar', err));
15
16  module.exports = client;

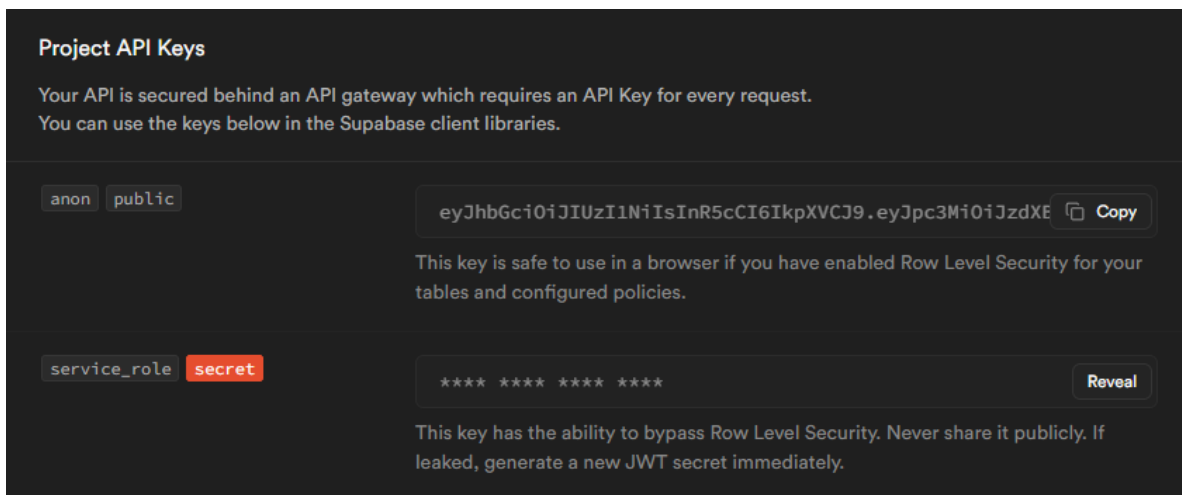
```

Para empezar a configurar las apis, nos iremos a supabase e iremos a la parte de configuración, en donde encontraremos el apartado de Data Api, allí encontraremos la información para conectar las apis a nuestro proyecto ya creado.





Estos son los datos que nos van a ayudar en la conexión.



Especialmente el anon public.

Cambiamos los datos requeridos en esta ´parte del código.

```
// Supabase config  
const supabaseUrl = 'https://xmuewcwnhnquaajxtjknk.supabase.co/rest/v1/persona';  
const supabaseKey = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImhtdWV3Y3duaG5xdWFqeHRqbmtuIiwiaWF0IjoiMTYxMjEwNDAyLTIwMjQifQ.';
```

Y ya podríamos empezar con las consultas en el postman, que son el guardar, obtener, eliminar y actualizar.

Funciones que ya tenemos dentro de nuestro código así.

```

app.get('/api/obtener', async (req, res) => {
  try {
    const response = await axios.get(supabaseUrl, {
      headers: {
        apikey: supabaseKey,
        Authorization: `Bearer ${supabaseKey}`
      }
    });

    res.status(200).json({
      success: true,
      message: 'Datos obtenidos correctamente',
      data: response.data
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Error al recuperar los datos',
      details: error.message
    });
  }
});

```

### Función Obtener

```

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

  try {
    const response = await axios.delete(`${supabaseUrl}?cedula=eq.${cedula}`, {
      headers: {
        apikey: supabaseKey,
        Authorization: `Bearer ${supabaseKey}`,
        Prefer: 'return=representation'
      }
    });

    if (response.data.length === 0) {
      res.status(404).json({
        success: false,
        message: `No existe el registro con cédula ${cedula}`
      });
    } else {
      res.status(200).json({
        success: true,
        message: 'Registro eliminado exitosamente',
        deleted: response.data
      });
    }
  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Error al eliminar el registro',
      details: error.message
    });
  }
});

```

### Función Eliminar

```

app.post('/api/guardar', (req, res) => {
  const { cedula, nombre, edad, profesion } = req.body;
  const query = 'INSERT INTO persona (cedula, nombre, edad, profesion) VALUES ($1, $2, $3, $4)';
  const values = [cedula, nombre, edad, profesion];

  connection.query(query, values, (error, result) => {
    if (error) {
      res.status(500).json({ message: 'LA API FALLO', error });
    } else {
      res.status(201).json({ cedula, nombre, edad, profesion });
    }
  });
});

```

### Función Guardar

```

app.put('/api/actualizar/:cedula', async (req, res) => {
  const { cedula } = req.params;
  const { nombre, edad, profesion } = req.body;

  console.log("Body recibido:", req.body); // <-- VERIFICAR EL BODY

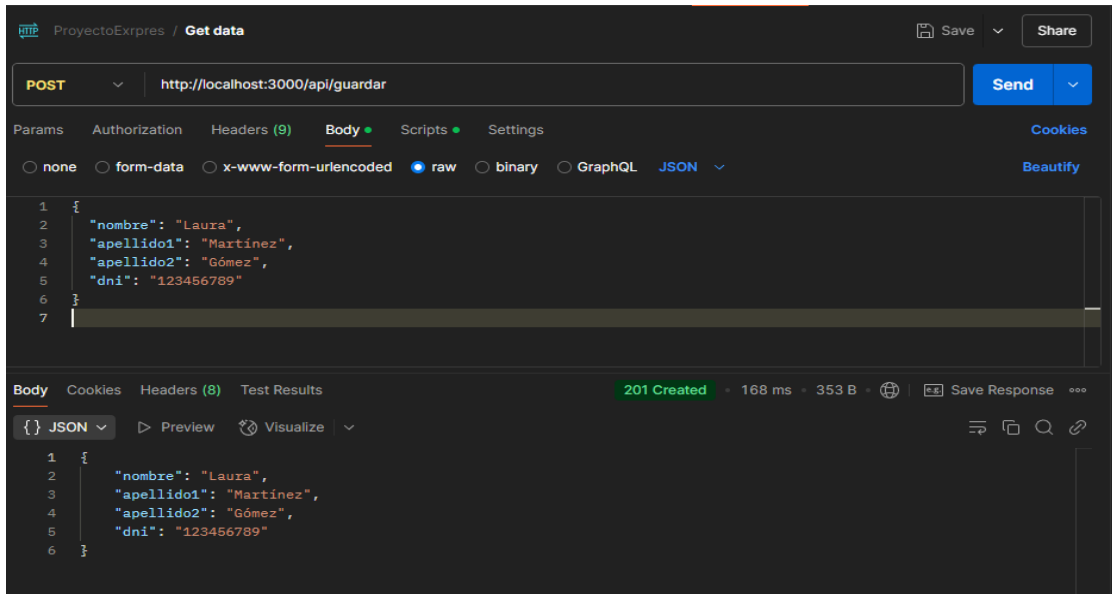
  try {
    const response = await axios.patch(`${supabaseUrl}>cedula=eq.${cedula}`,
      { nombre, edad, profesion },
      {
        headers: {
          apikey: supabaseKey,
          Authorization: `Bearer ${supabaseKey}`,
          Prefer: 'return=representation'
        }
      }
    );

    if (response.data.length === 0) {
      res.status(404).json({
        success: false,
        message: 'No existe ningún registro con la cédula ${cedula}'
      });
    } else {
      res.status(200).json({
        success: true,
        message: 'Registro actualizado exitosamente',
        updated: response.data
      });
    }
  } catch (error) {

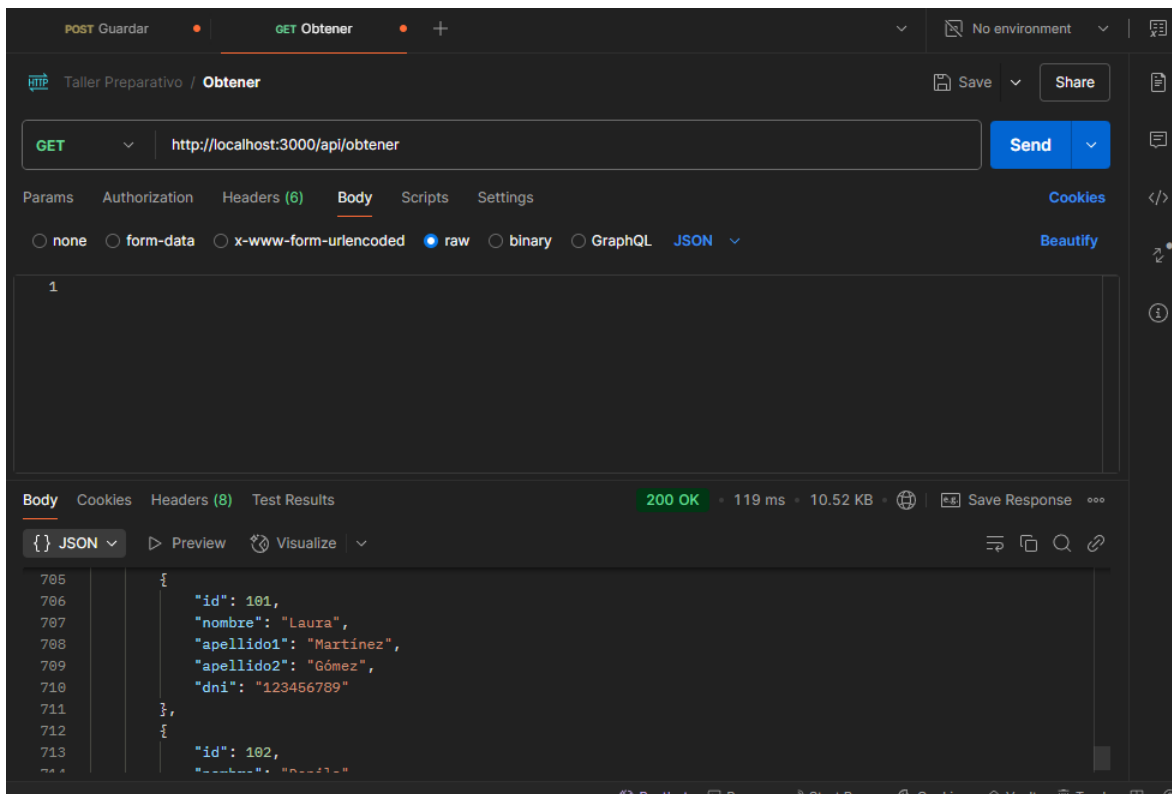
```

## Función Actualizar

Empezaremos con la función POST, que es la que nos va a guardar nuevos datos dentro de nuestras tablas, así que la cambiamos en nuestro postman y agregamos `http://localhost:3000/api/guardar` para que nos referencie en donde lo vamos a guardar a su vez agregamos un header de tipo content-type que tendrá valor `application/json`, y así se ve.



Ahora vamos con la función GET para mostrar todos nuestros registros para ello no necesitamos headers, solo la ruta <http://localhost:3000/api/obtener>, y así se ve.



Como podemos ver en la parte de abajo se muestran algunos de los datos que ya insertamos.

Ahora vamos con el método DELETE, acá nos encontramos con un pequeño obstáculo el cual es que nuestro dni esta asociado como una llave foránea lo cual no nos va a dejar eliminarlo, para ello, nos iremos a pgadmin, e ingresaremos un comando para desligarlo y que así nos deje hacer el DELETE.



Hecho esto, así se debe ver cuando la consulta se ha hecho de manera correcta.

The screenshot shows a Postman interface for a DELETE request. The URL is `http://localhost:3000/api/eliminar/000000091`. The response is a 200 OK status with a response time of 140 ms and a body size of 440 B. The response body is a JSON object:

```
{
  "success": true,
  "message": "Registro eliminado exitosamente",
  "deleted": {
    "id": 91,
    "nombre": "Nombre91",
    "apellido1": "Apellido1_91",
    "apellido2": "Apellido2_91",
    "dni": "000000091"
  }
}
```

Y por ultimo nos vamos a ir a la función de actualizar, que así se ve al haberla hecho correctamente.

The screenshot shows a Postman interface for a PUT request. The URL is `http://localhost:3000/api/actualizar/1`. The request body is a JSON object:

```
{
  "nombre": "Juan",
  "apellido1": "Pico",
  "apellido2": "Sánchez",
  "marca": "Toyota",
  "modelo": "Corolla",
  "caballos": 150
}
```

The response is a 200 OK status with a response time of 263 ms and a body size of 525 B. The response body is a JSON object:

```
{
  "success": true,
  "message": "Persona y coche actualizados correctamente",
  "persona": {
    "id": 1,
    "nombre": "Juan",
    "apellido1": "Pico",
    "apellido2": "Sánchez",
    "dni": "1"
  }
}
```

