Quiz Bases De Datos

Michael Stiven Castillo Castillo Id 825947

Corporación Universitaria Minuto de Dios

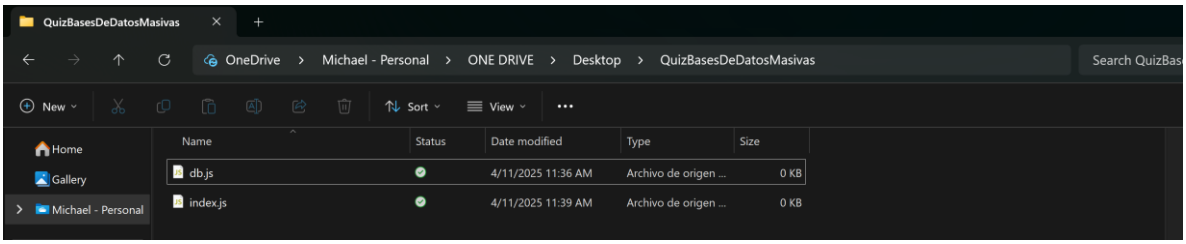
Ingeniería de Sistemas,

Bases de datos masivas

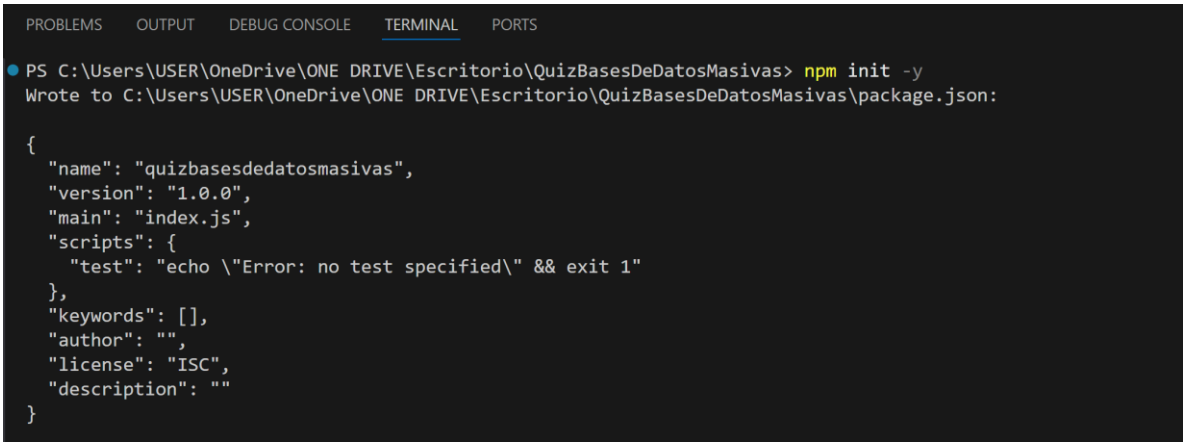
Alexander Matallana Porras

11 de abril del 2025

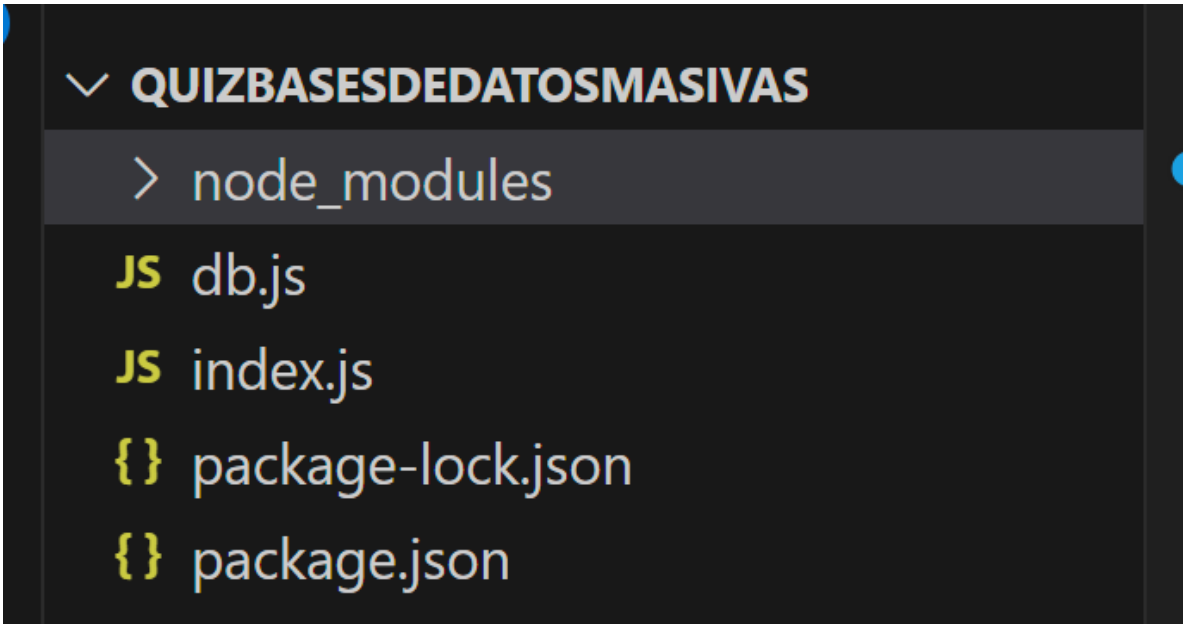
Creamos una carpeta donde se va a guardar todos los datos
creamos dos archivos llamados db.js y index.js



Usamos el comando npm init -y en la terminal de visual studio code lo que genera un package.json



Despues agregamos el siguiente comando npm install dotenv el cual general los módulos

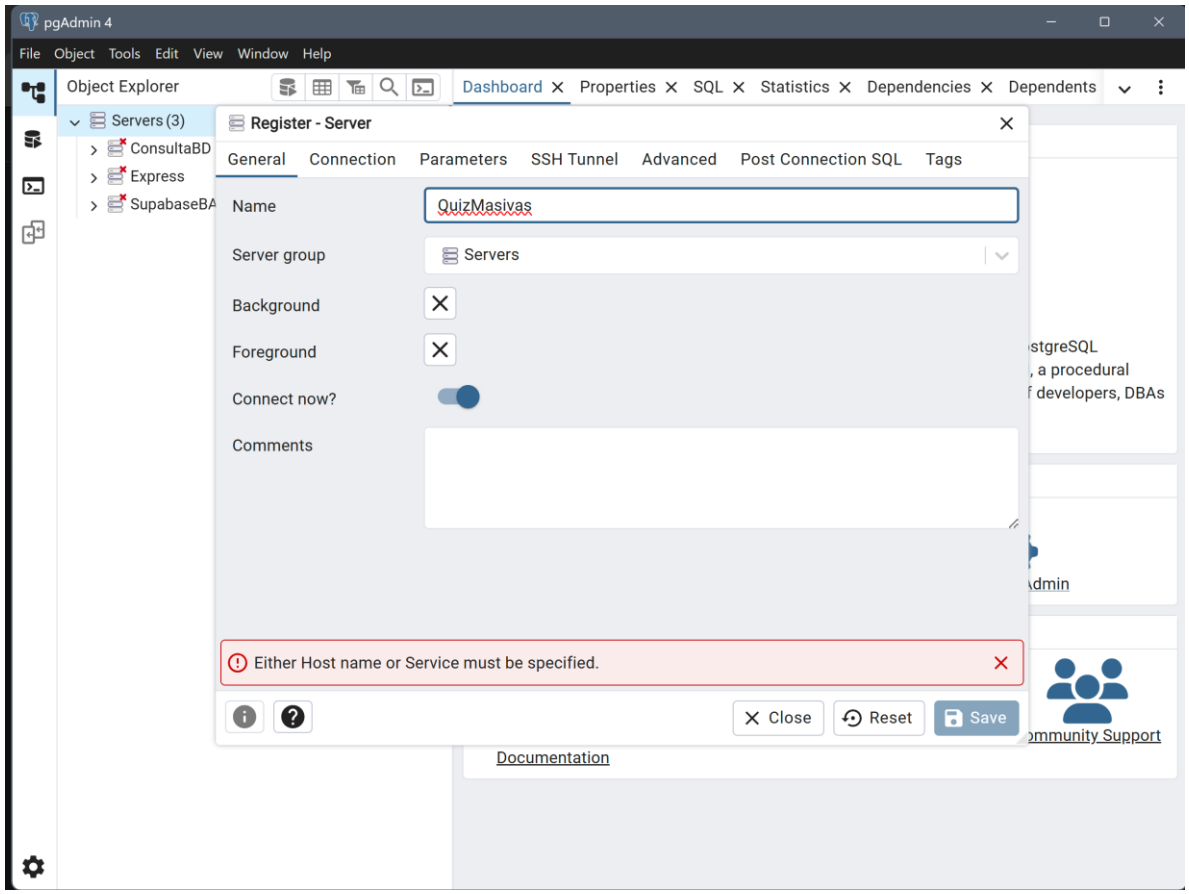


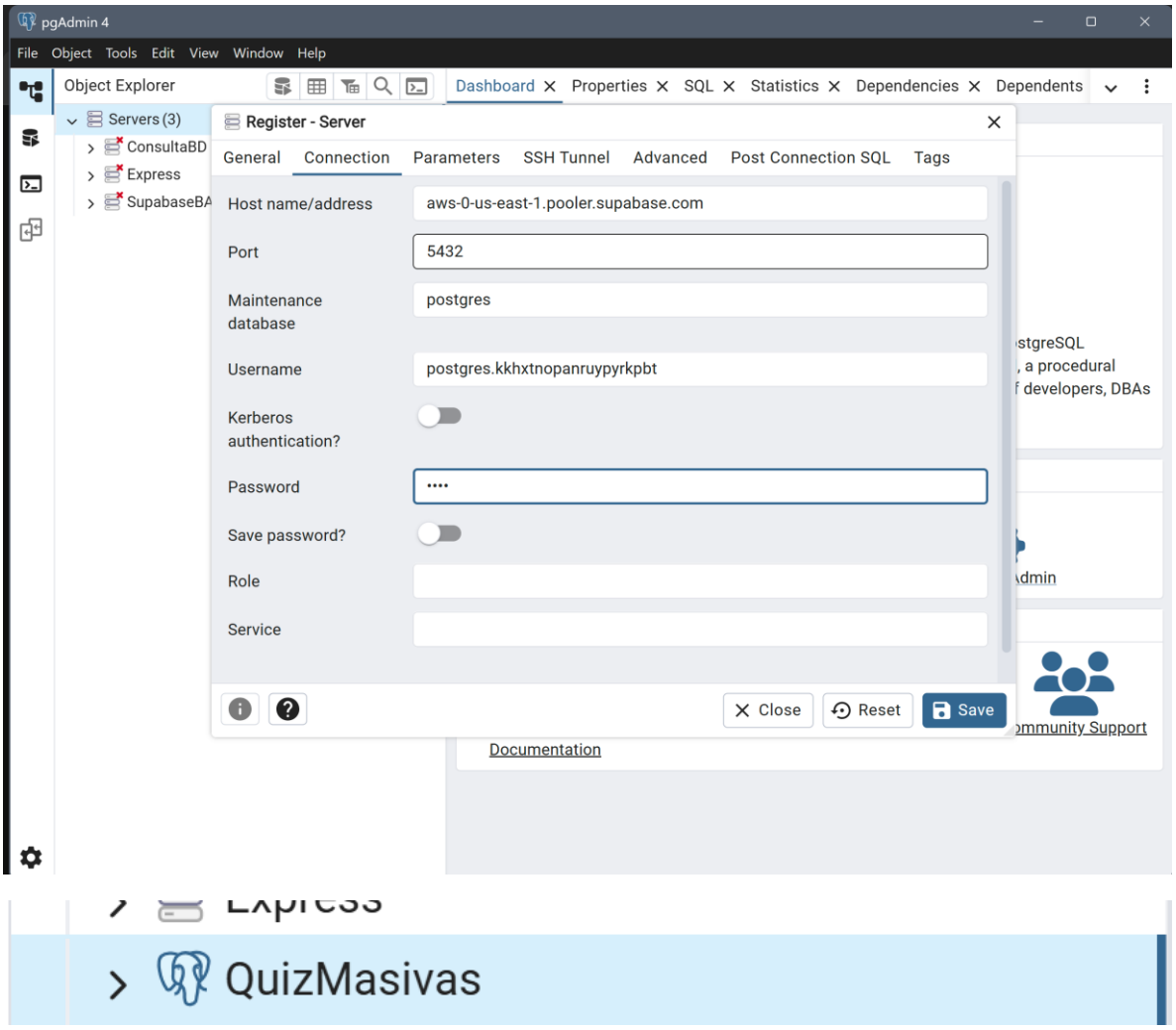
```
PS C:\Users\USER\OneDrive\ONE DRIVE\Escritorio\QuizBasesDeDatosMasivas> npm install express cors pg dotenv
>>

added 82 packages, and audited 84 packages in 15s

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

En db.js accemos la conexión con supabase

```
JS db.js
JS db.js > ...
1  const { Pool } = require('pg');
2
3  const pool = new Pool({
4    user: 'postgres.kkhxtnopanruypyrkpbt',
5    host: 'aws-0-us-east-1.pooler.supabase.com',
6    database: 'postgres',
7    password: '1234',
8    port: 5432,
9    ssl: {
10     rejectUnauthorized: false
11   }
12 });
13
14 pool.connect((err, client, release) => {
15   if (err) {
16     console.error('❌ Error al conectar con Supabase:', err.stack);
17   } else {
18     console.log('🟢 Conectado exitosamente a Supabase');
19     release();
20   }
21 });
22
23 module.exports = pool;
24
```

Creamos la conexión con pgadmin4 con supabase


pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Servers (3)
ConsultaBD
Express
SupabaseBA
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
Register - Server
General
Connection
Parameters
SSH Tunnel
Advanced
Post Connection SQL
Tags
Name
QuizMasivas
Server group
Servers
Background
Foreground
Connect now?
Comments
Either Host name or Service must be specified.
Close
Reset
Save
Documentation
Community Support



Ahora creamos las api para cada tabla las cuales son personas y coches se realiza en el index.js

```
// Obtener todas las personas
app.get('/api/personas', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM persona');
    res.status(200).json(result.rows);
  } catch (err) {
    res.status(500).json({ error: '❌ Error al obtener personas', details: err.message });
  }
});

// Crear una persona
app.post('/api/personas', async (req, res) => {
  const { nombre, apellido1, apellido2, dni } = req.body;
  try {
    await pool.query(
      'INSERT INTO persona (nombre, apellido1, apellido2, dni) VALUES ($1, $2, $3, $4)',
      [nombre, apellido1, apellido2, dni]
    );
    res.status(201).json({ message: '✅ Persona creada' });
  } catch (err) {
    res.status(500).json({ error: '❌ Error al crear persona', details: err.message });
  }
});
```

JS db.js JS index.js 1 ●

JS index.js > app.delete('/api/personas/:id') callback

```
41  // Actualizar persona
42  app.put('/api/personas/:id', async (req, res) => {
43    const { id } = req.params;
44    const { nombre, apellido1, apellido2, dni } = req.body;
45    try {
46      await pool.query(
47        'UPDATE persona SET nombre=$1, apellido1=$2, apellido2=$3, dni=$4 WHERE id=$5',
48        [nombre, apellido1, apellido2, dni, id]
49      );
50      res.status(200).json({ message: '✏️ Persona actualizada' });
51    } catch (err) {
52      res.status(500).json({ error: '❌ Error al actualizar persona', details: err.message });
53    }
54  });
55
56  // Eliminar persona
57  app.delete('/api/personas/:id', async (req, res) => {
58    const { id } = req.params;
59    try {
60      await pool.query('DELETE FROM persona WHERE id = $1', [id]);
61      res.status(200).json({ message: '🗑️ Persona eliminada' });
62    } catch (err) {
63      res.status(500).json({ error: '❌ Error al eliminar persona', details: err.message });
64    }
65  });
```

```
// Obtener todos los coches
app.get('/api/coches', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM coche');
    res.status(200).json(result.rows);
  } catch (err) {
    res.status(500).json({ error: '❌ Error al obtener coches', details: err.message });
  }
});

// Obtener coches de una persona
app.get('/api/personas/:id/coches', async (req, res) => {
  const { id } = req.params;
  try {
    const result = await pool.query('SELECT * FROM coche WHERE persona_id = $1', [id]);
    res.status(200).json(result.rows);
  } catch (err) {
    res.status(500).json({ error: '❌ Error al obtener coches de la persona', details: err.message });
  }
});
```

index.js > ...

```
89   // Crear un coche
90   app.post('/api/coches', async (req, res) => {
91     const { matricula, marca, modelo, caballos, persona_id } = req.body;
92     try {
93       await pool.query(
94         'INSERT INTO coche (matricula, marca, modelo, caballos, persona_id) VALUES ($1, $2, $3, $4, $5)',
95         [matricula, marca, modelo, caballos, persona_id]
96       );
97       res.status(201).json({ message: '✅ Coche creado' });
98     } catch (err) {
99       res.status(500).json({ error: '❌ Error al crear coche', details: err.message });
100    }
101  });
102
103  // Actualizar coche
104  app.put('/api/coches/:matricula', async (req, res) => {
105    const { matricula } = req.params;
106    const { marca, modelo, caballos, persona_id } = req.body;
107    try {
108      await pool.query(
109        'UPDATE coche SET marca=$1, modelo=$2, caballos=$3, persona_id=$4 WHERE matricula=$5',
110        [marca, modelo, caballos, persona_id, matricula]
111      );
112      res.status(200).json({ message: '🖍️ Coche actualizado' });
113    } catch (err) {
```

```
// Eliminar coche
app.delete('/api/coches/:matricula', async (req, res) => {
  const { matricula } = req.params;
  try {
    await pool.query('DELETE FROM coche WHERE matricula = $1', [matricula]);
    res.status(200).json({ message: '🗑️ Coche eliminado' });
  } catch (err) {
    res.status(500).json({ error: '❌ Error al eliminar coche', details: err.message });
  }
});


// Iniciar el servidor
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`🚀 Servidor corriendo en http://localhost:${PORT}`);
});
```

En supabase accedemos a sql editor y creamos las tablas de personas y coche

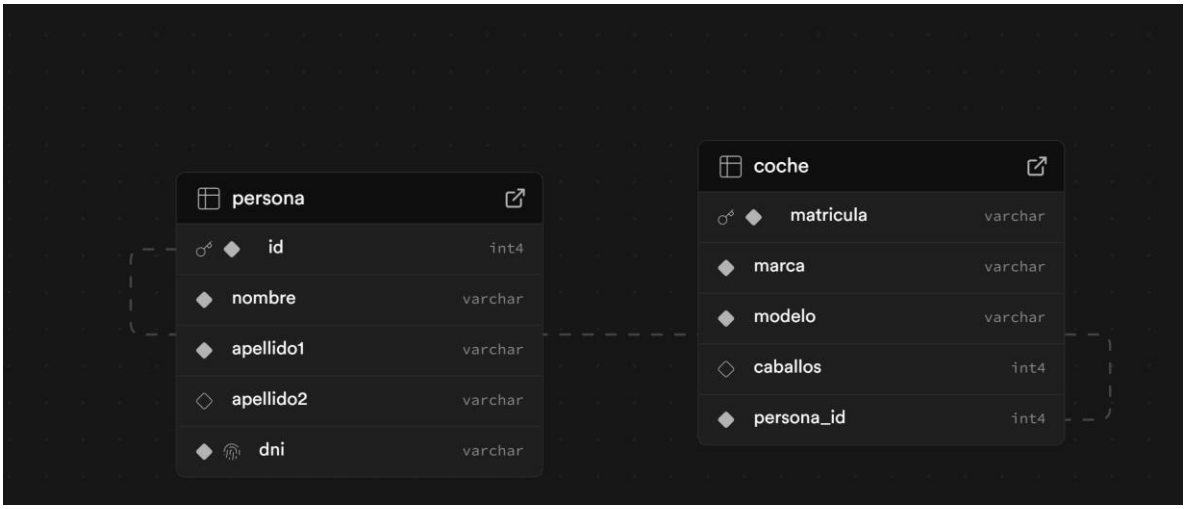
```sql
CREATE TABLE IF NOT EXISTS persona (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(80) NOT NULL,
  apellido1 VARCHAR(80) NOT NULL,
  apellido2 VARCHAR(80),
  dni VARCHAR(9) UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS coche (
  matricula VARCHAR(7) PRIMARY KEY,
  marca VARCHAR(45) NOT NULL,
  modelo VARCHAR(45) NOT NULL,
  caballos INT,
  persona_id INT NOT NULL,
  CONSTRAINT fk_persona FOREIGN KEY (persona_id) REFERENCES persona(id) ON DELETE CASCADE
);

```

Nos dice que se crearon

```sql
CREATE TABLE IF NOT EXISTS persona (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(80) NOT NULL,
  apellido1 VARCHAR(80) NOT NULL,
  apellido2 VARCHAR(80),
  dni VARCHAR(9) UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS coche (
  matricula VARCHAR(7) PRIMARY KEY,
  marca VARCHAR(45) NOT NULL,
  modelo VARCHAR(45) NOT NULL,
  caballos INT,
  persona_id INT NOT NULL,
  CONSTRAINT fk_persona FOREIGN KEY (persona_id) REFERENCES persona(id) ON DELETE CASCADE
);

```

Results Chart Export Source Primary Database Role postgres Run CTRL

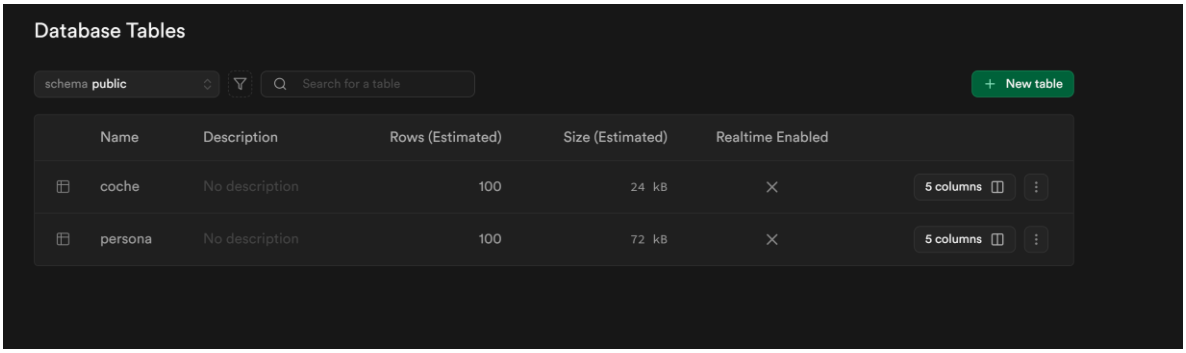
```
Success. No rows returned
```



Ahora le insertamos los 100 registros a cada tabla

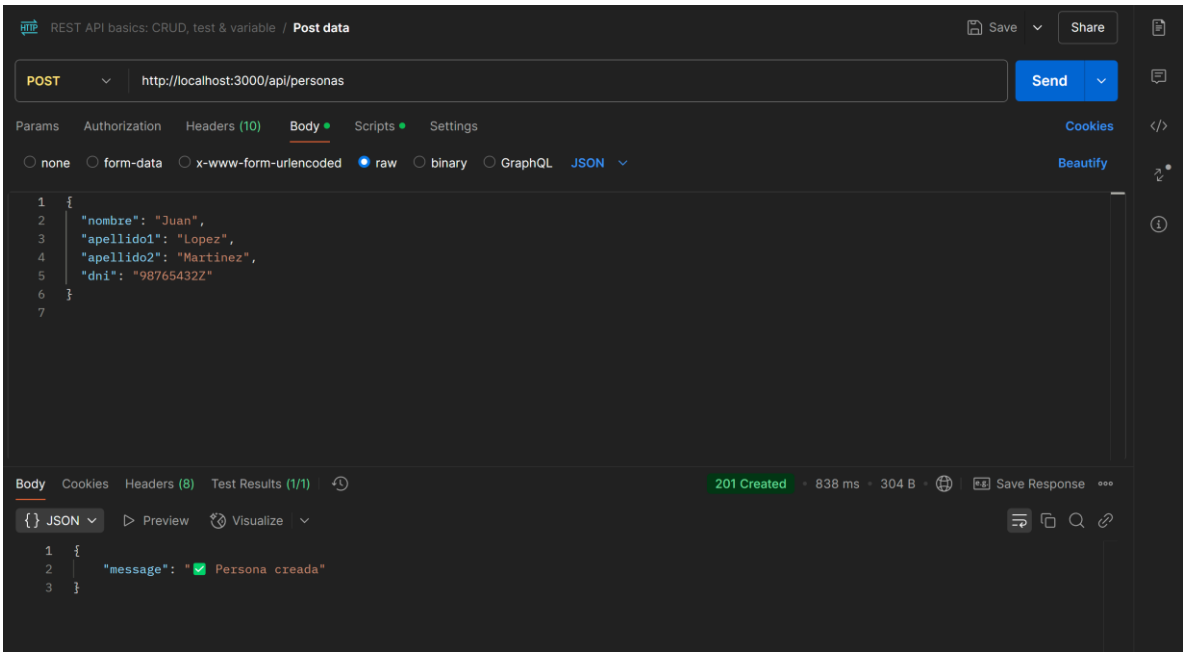
```sql
INSERT INTO persona (nombre, apellido1, apellido2, dni)
SELECT
  'Nombre' || i,
  'Apellido1_' || i,
  'Apellido2_' || i,
  LPAD(i::text, 8, '0') || 'A'
FROM generate_series(1, 100) AS s(i);

INSERT INTO coche (matricula, marca, modelo, caballos, persona_id)
SELECT
  'MAT' || LPAD(i::text, 4, '0'),
  'Marca' || ((i % 10) + 1),
  'Modelo' || ((i % 20) + 1),
  (100 + (i % 200)),
  FLOOR(RANDOM() * 100 + 1)::INT
FROM generate_series(1, 100) AS s(i);

```

Verificamos y efectivamente se crearon los 100 registros

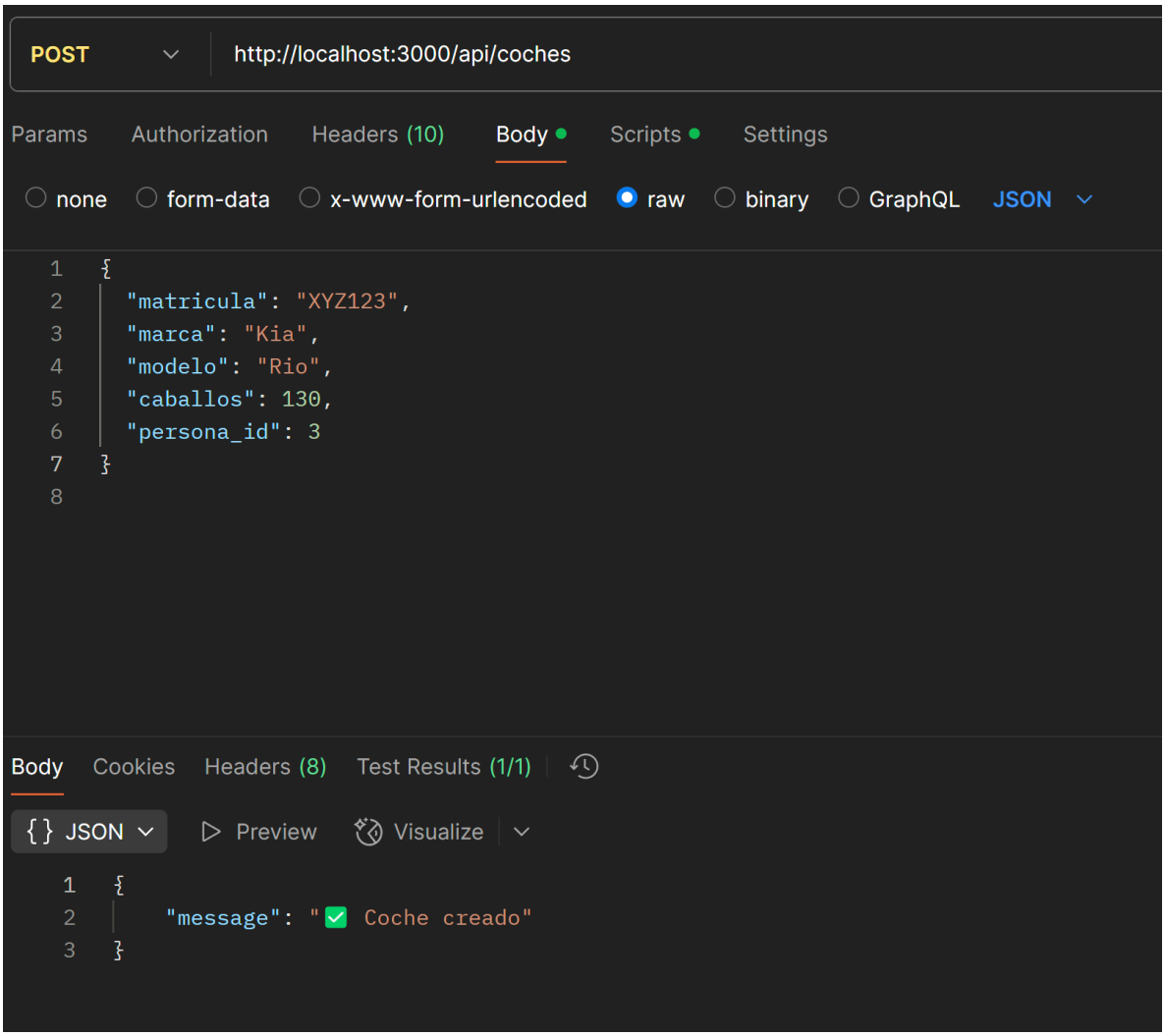


Accemos la consulta de cada api en el postman para validar que haya funcionado correctamente

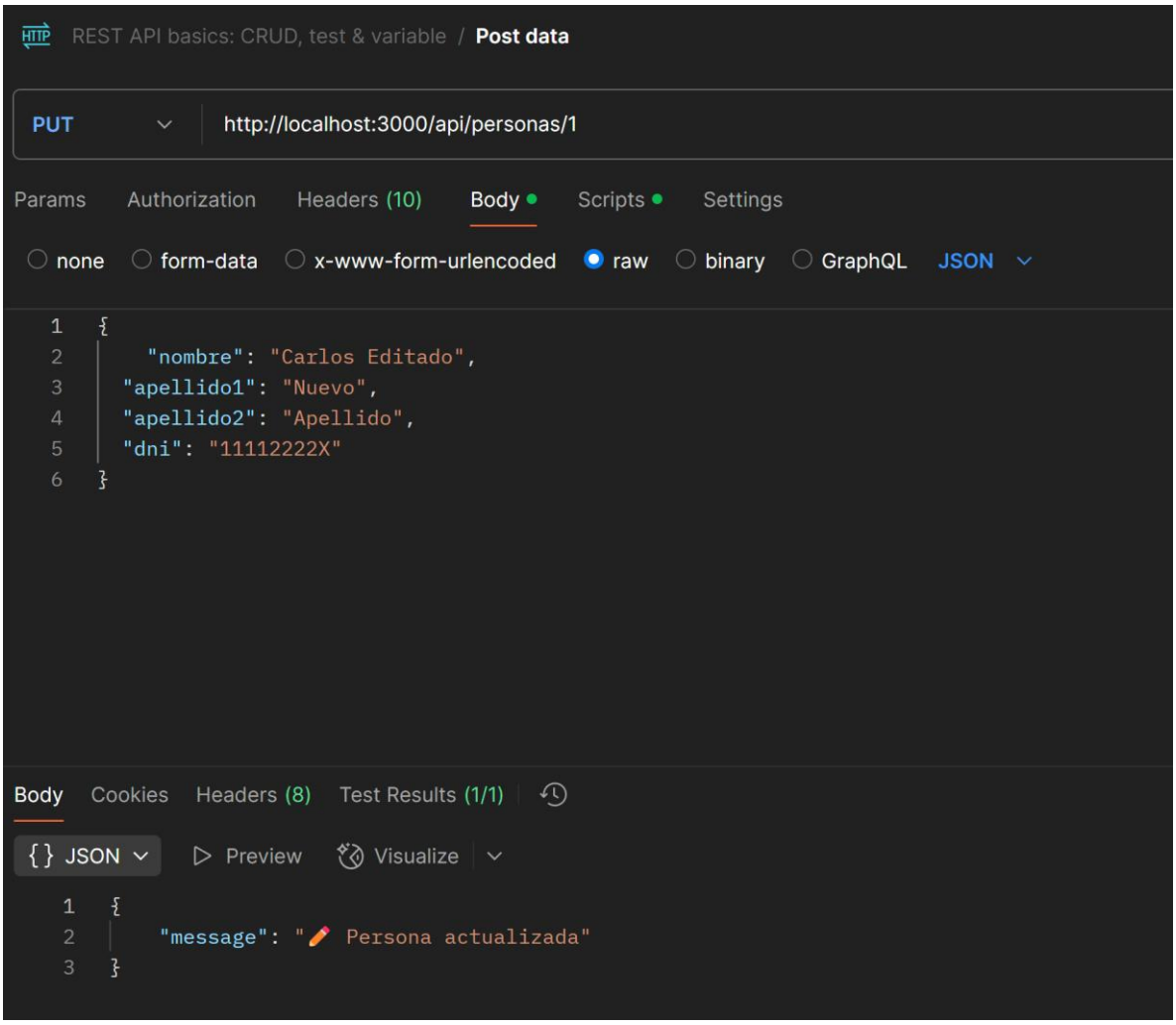
Persona creada



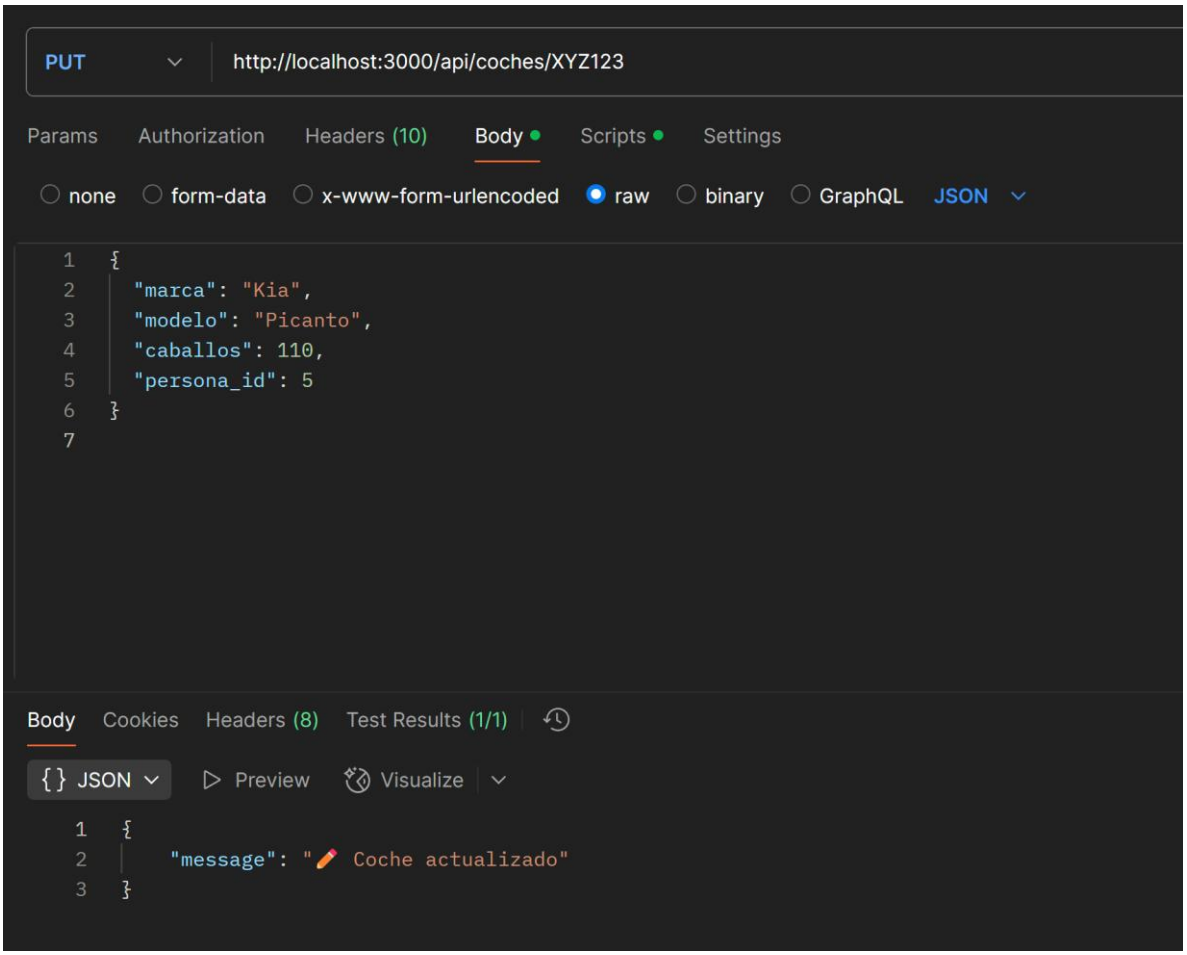
Crear un coche



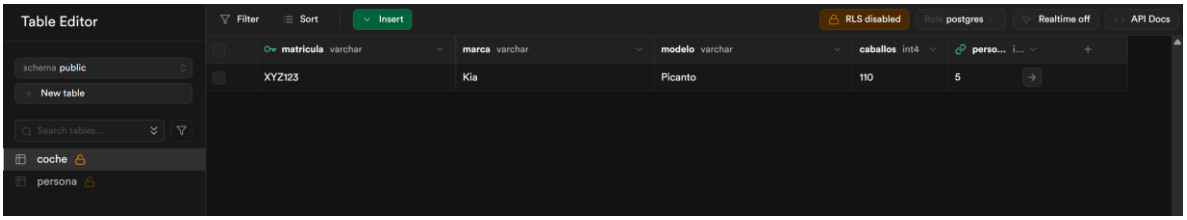
Con el put podemos modificar dependiendo si hemos creado la api en este caso editamos la primera persona

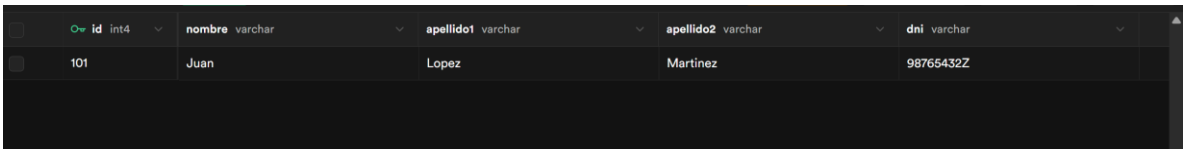


Accemos lo mismo con el coche que ya habíamos creado



Podemos verificar que antes teníamos 100 datos ahora tenemos 101





id int4	nombre varchar	apellido1 varchar	apellido2 varchar	dni varchar
101	Juan	Lopez	Martinez	98765432Z