

# REST API

## Nodejs & PostgreSQL



- INSTALAR CODEVISUAL

<https://code.visualstudio.com/>

Extensiones de Visual Studio Code

- |                                |                                      |
|--------------------------------|--------------------------------------|
| 1. jshint Generator            | 11. JavaScript ( ES6 ) code snippets |
| 2. AB HTML Formatter           | 12. jshint                           |
| 3. Angular 8 Snippets          | 13. JSON to TS                       |
| 4. Angular Language Service    | 14. Material Icon Theme              |
| 5. Angular2-inline             | 15. MySQL-inline-decorator           |
| 6. Auto Close Tag              | 16. Prettier - Code formatter        |
| 7. Bootstrap 4, Font Awesome 4 | 17. PWA Tools                        |
| 8. Gist Extension              | 18. Terminal                         |
| 9. HTML CSS Support            | 19. TSLint                           |
| 10. Ionic 4 snippets           | 20. TypeScript Importer              |

- INSTALAR NODE JS

<https://nodejs.org/es/>

- INSTALAR BASE DE DATOS POSTGRESQL

<https://www.postgresql.org/>

Diseño de la base datos para mi agendar de direcciones Web

Users: usuario\_id, name, nick, email, edad

Links: link\_id, detalle, link, estado, usuario\_id

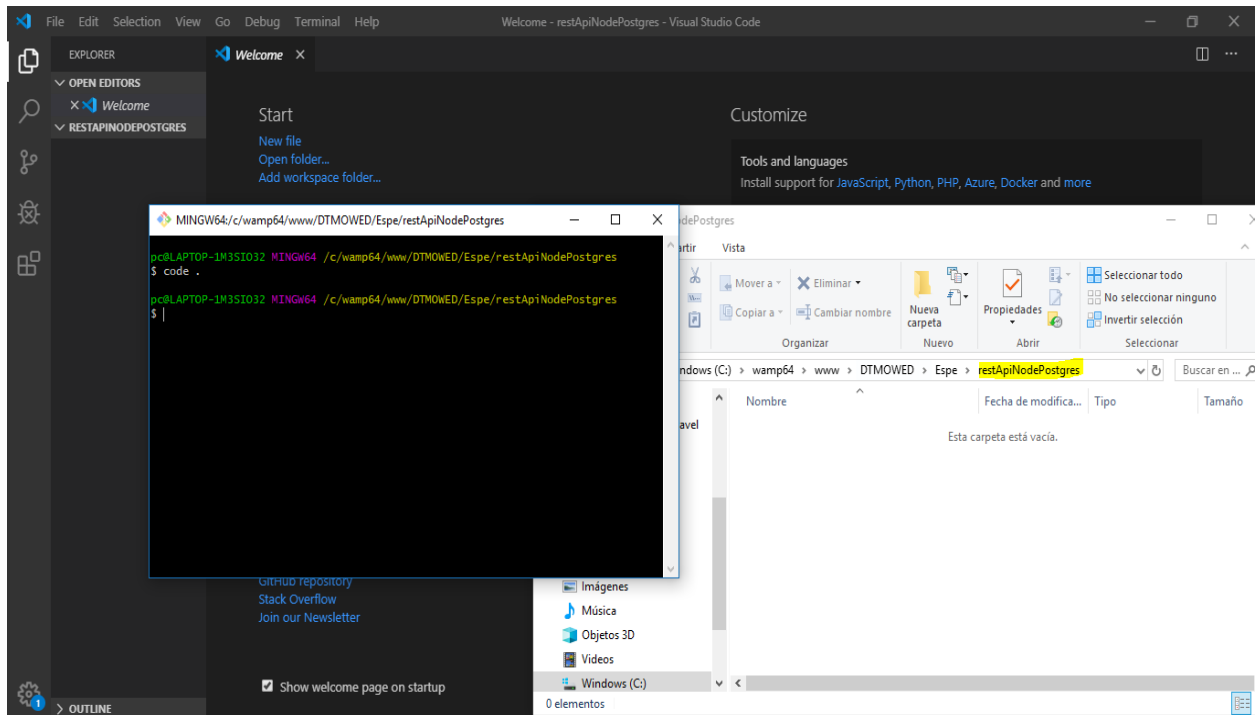
[www.dtmowed.org](http://www.dtmowed.org)

099 338 4378 - 096 919 1290

Autor: Ing. Figueroa Washington  
Guía: APIRESTS Node + PostgreSQL

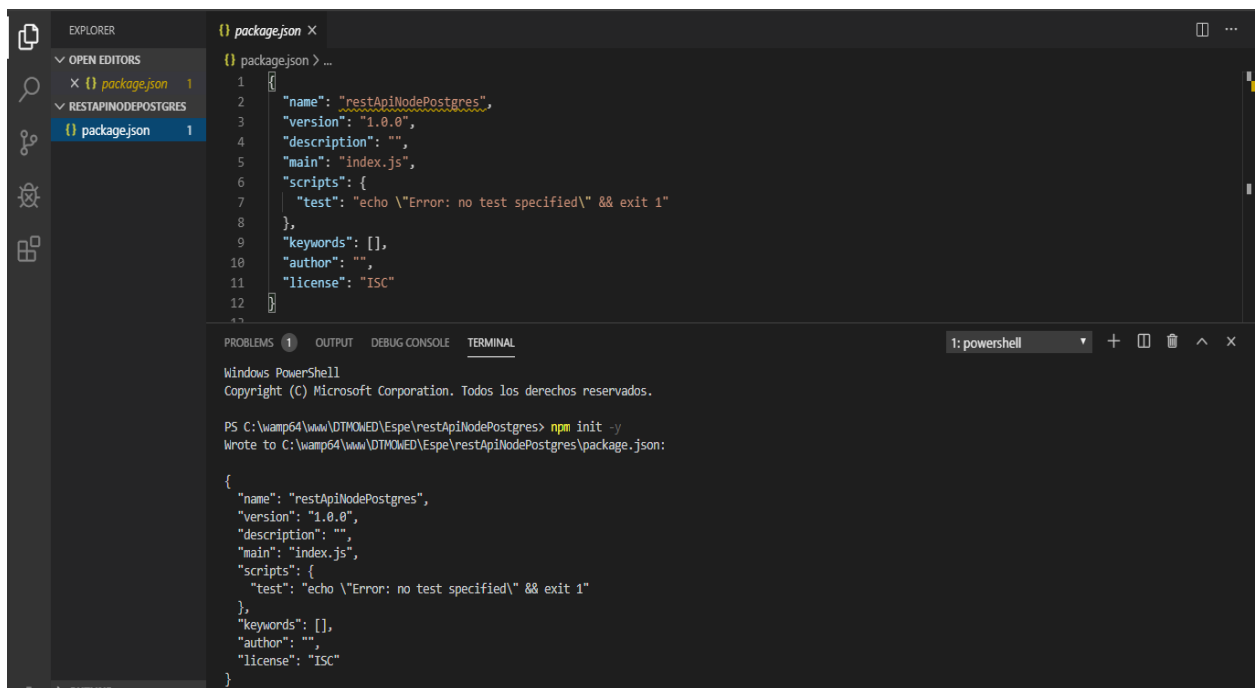
## CODIFICACIÓN PROYECTO

Crear la carpeta `restApiNodePostgres` y abrirla a través de visual code

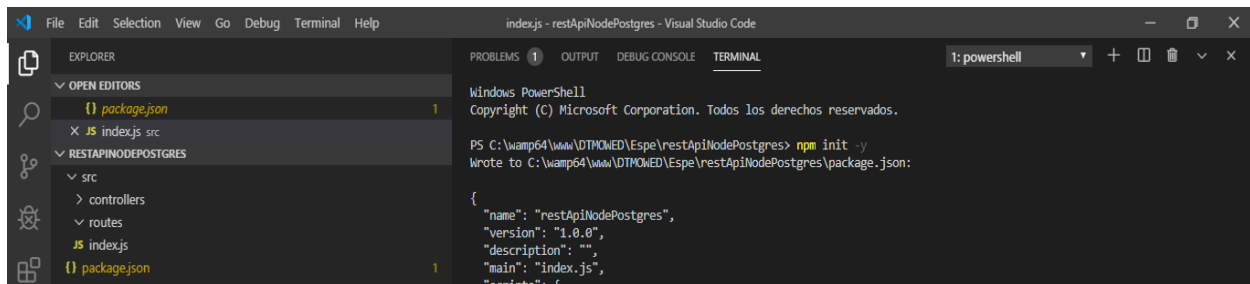


Abrimos la consola y ejecutamos `npm init -y`

Crearé un archivo `package.json` | archivo que describe el proyecto



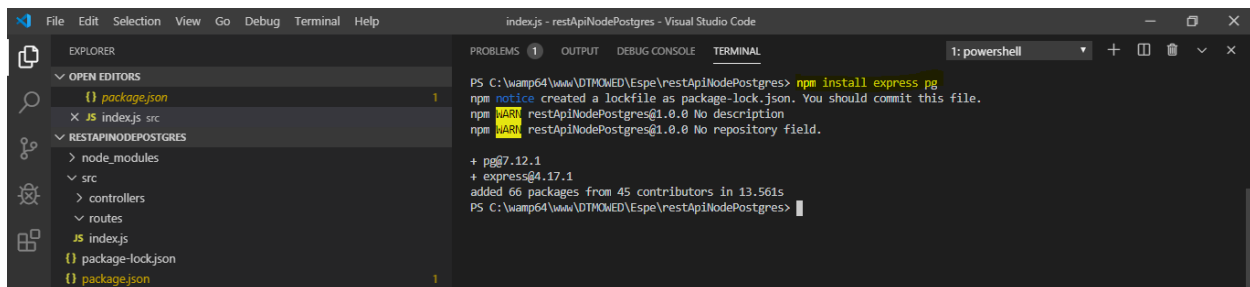
Creamos la carpeta src y dentro de ella las carpetas Router, Controller, y el archivo index.js



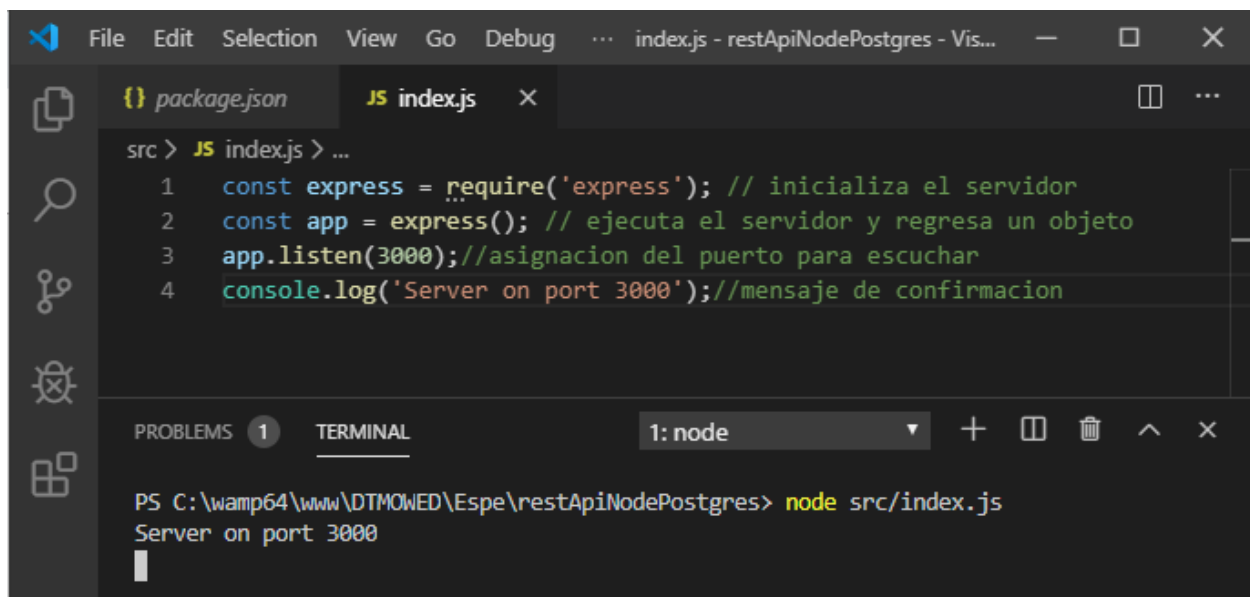
Instalamos los módulos **express y pg**

**Express:** Es un framework de node que permite crear el servidor, definir rutas, recibir datos procesarlos...

**Pg:** Driver de conexión a PostgreSQL



Inicializamos el servidor



Instalación de modulo **nodemon -D** desarrollo, permite actualizar los cambios en el servidor

Implementamos el los scripts **"dev": "nodemon src/index.js"**

Ejecutamos en cmd **npm run dev**

The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'package.json' file is open in the main editor. The 'dependencies' section is visible, showing 'express' and 'pg'. The 'devDependencies' section shows 'nodemon'. The terminal at the bottom shows the command 'npm run dev' being executed, and the output indicates that the server is running on port 3000.

```
{
  "name": "restApiNodePostgres",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon src/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "pg": "^7.12.1"
  },
  "devDependencies": {
    "nodemon": "^1.19.4"
  }
}
```

```
PS C:\wamp64\www\DTMOWED\Espe\restApiNodePostgres> npm run dev

> restApiNodePostgres@1.0.0 dev C:\wamp64\www\DTMOWED\Espe\restApiNodePostgres
> nodemon src/index.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
body-parser deprecated undefined extended: provide extended option src/index.js:6:17
Server on port 3000
```

## Definición de rutas

Creamos el archivo index.js dentro de la carpeta routes y codificamos

The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'index.js' file is open in the main editor. The code defines a Router using 'express' and 'body-parser', and sets up a GET route for '/users'.

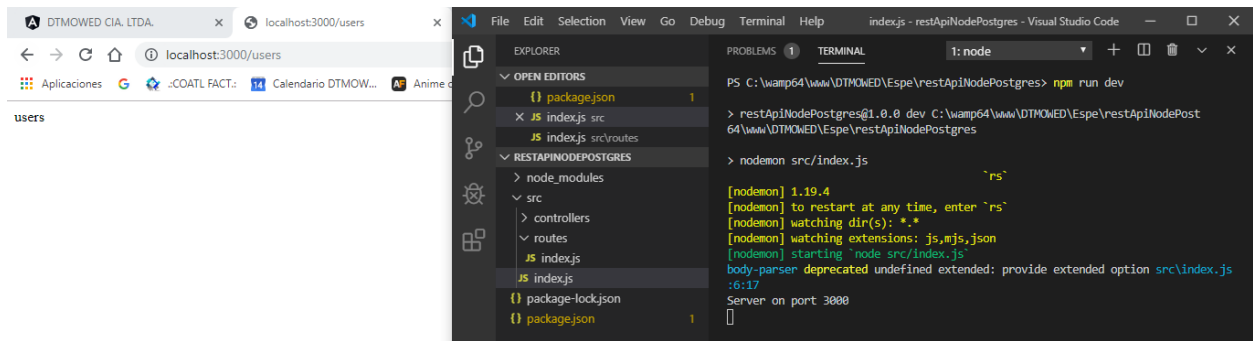
```
src > routes > JS index.js > ...
1  const {Router} = require('express');
2  const router = Router(); // objeto que permite definir las rutas
3
4  router.get('/users', (req, res) => {
5    res.send('users');
6  });
7
8  module.exports = router;
```

Importamos las rutas en el index.js principal e importamos algunas funciones de express

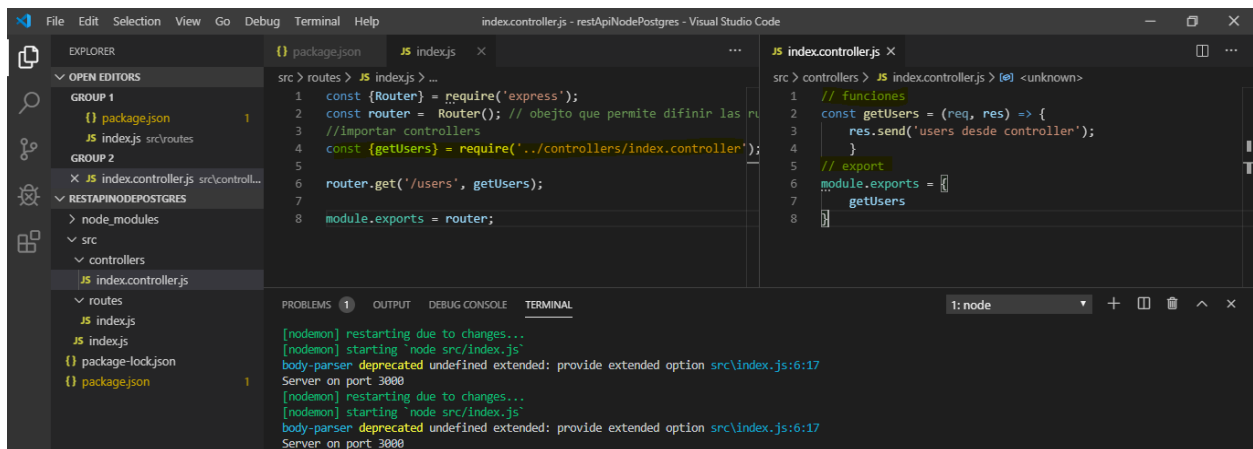
The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'index.js' file is open in the main editor. The code imports 'express' and 'body-parser', sets up the server, and imports the routes from the 'routes/index.js' file.

```
src > JS index.js > ...
1  const express = require('express'); // inicializa el servidor
2  const app = express(); // ejecuta el servidor y regresa un objeto
3  //para recibir formatos json y html
4  //middlewares
5  app.use(express.json()); // json to js
6  app.use(express.urlencoded()); // to objeto
7  // routes
8  app.use(require('./routes/index'));
9
10 app.listen(3000); // asignacion del puerto para escuchar
11 console.log('Server on port 3000'); // mensaje de confirmacion
```

## Miramos por consola

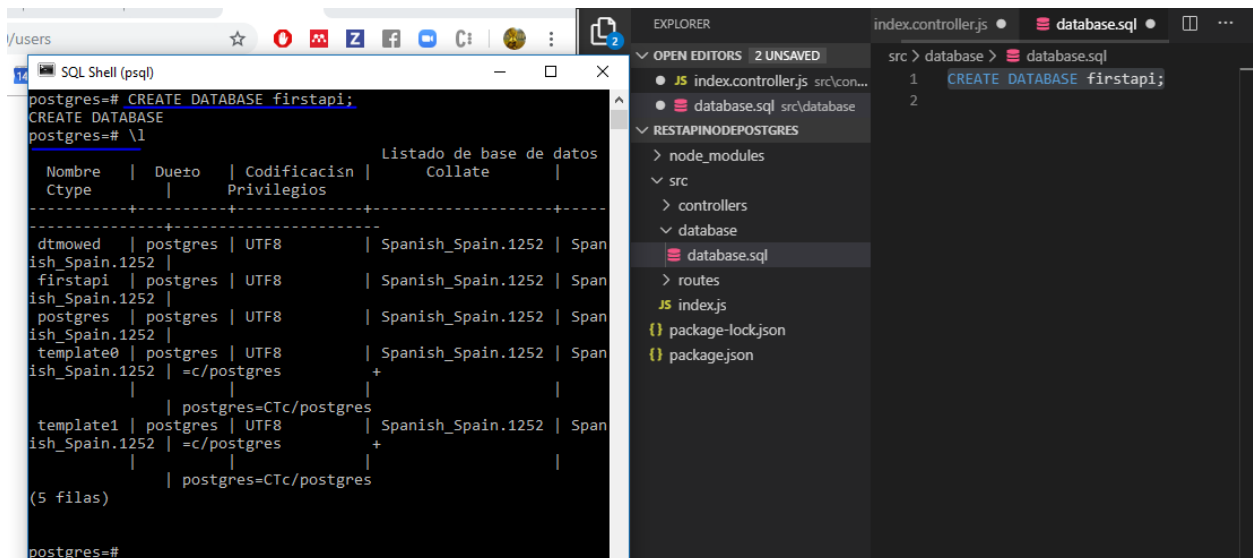


Creamos el archivo `index.controller.js` y creamos las funciones correspondientes y lo importamos en el `index.js` de router



Conexión con PostgreSQL, creamos una nueva carpeta Database y dentro de ella un archivo `database.sql`

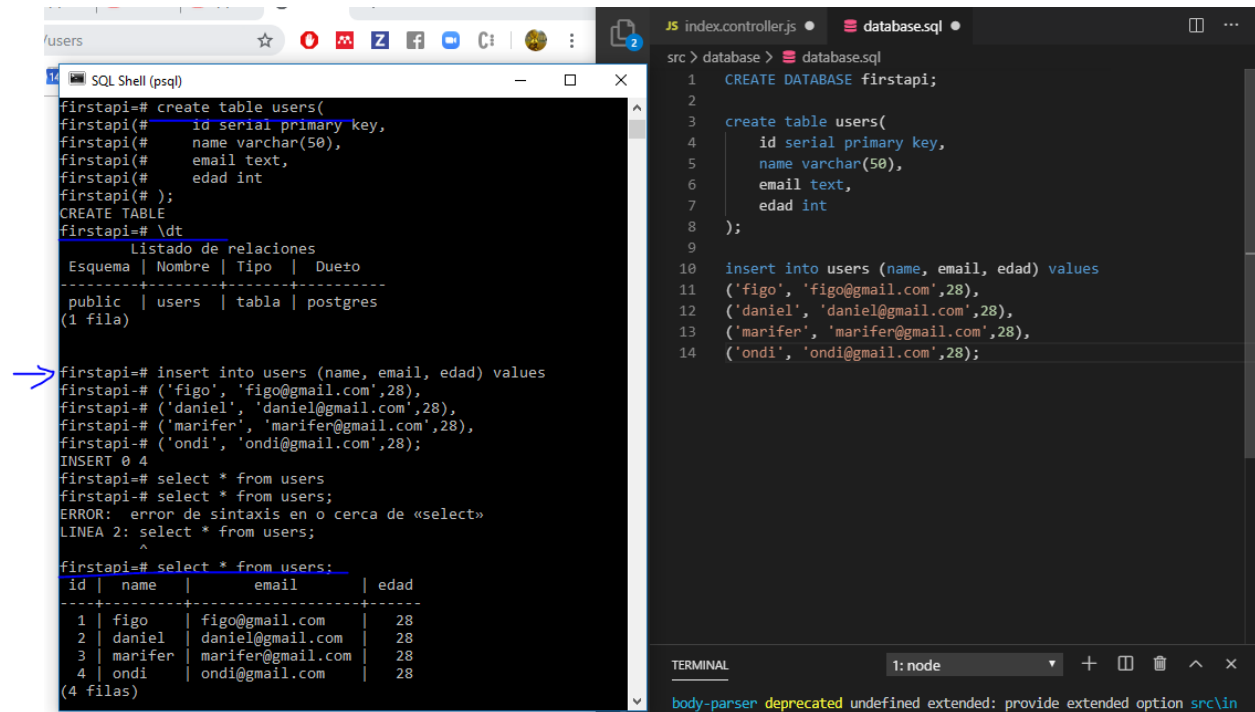
Utilizando el SQL Shell creamos la base de datos



Ingresamos a la base de datos

```
postgres=# \c firstapi
```

Utilizando el SQL Shell creamos la tabla users y agregamos datos



The screenshot shows two windows. The left window is a terminal running SQL Shell (psql) connected to a database named 'firstapi'. It shows the creation of a 'users' table with columns 'id' (serial primary key), 'name' (varchar(50)), 'email' (text), and 'edad' (int). It then shows the insertion of four records: ('figo', 'figo@gmail.com', 28), ('daniel', 'daniel@gmail.com', 28), ('marifer', 'marifer@gmail.com', 28), and ('ondi', 'ondi@gmail.com', 28). Finally, it shows a successful 'select \* from users;' query resulting in a table with 4 rows.

```
firstapi=# create table users(
firstapi=#   id serial primary key,
firstapi=#   name varchar(50),
firstapi=#   email text,
firstapi=#   edad int
firstapi=# );
CREATE TABLE
firstapi=# \dt
Listado de relaciones
Esquema | Nombre | Tipo | Dueño
-----+-----+-----+-----
public  | users  | tabla | postgres
(1 fila)

firstapi=# insert into users (name, email, edad) values
firstapi=# ('figo', 'figo@gmail.com',28),
firstapi=# ('daniel', 'daniel@gmail.com',28),
firstapi=# ('marifer', 'marifer@gmail.com',28),
firstapi=# ('ondi', 'ondi@gmail.com',28);
INSERT 0 4
firstapi=# select * from users
firstapi=# select * from users;
ERROR: error de sintaxis en o cerca de «select»
LINEA 2: select * from users;
        ^
firstapi=# select * from users;
 id | name          | email              | edad
----+-----+-----+----
 1 | figo          | figo@gmail.com    |   28
 2 | daniel        | daniel@gmail.com  |   28
 3 | marifer       | marifer@gmail.com |   28
 4 | ondi          | ondi@gmail.com    |   28
(4 filas)
```

The right window is a code editor showing the same SQL commands in a file named 'database.sql'.

```
src > database > database.sql
1  CREATE DATABASE firstapi;
2
3  create table users(
4      id serial primary key,
5      name varchar(50),
6      email text,
7      edad int
8  );
9
10 insert into users (name, email, edad) values
11 ('figo', 'figo@gmail.com',28),
12 ('daniel', 'daniel@gmail.com',28),
13 ('marifer', 'marifer@gmail.com',28),
14 ('ondi', 'ondi@gmail.com',28);
```

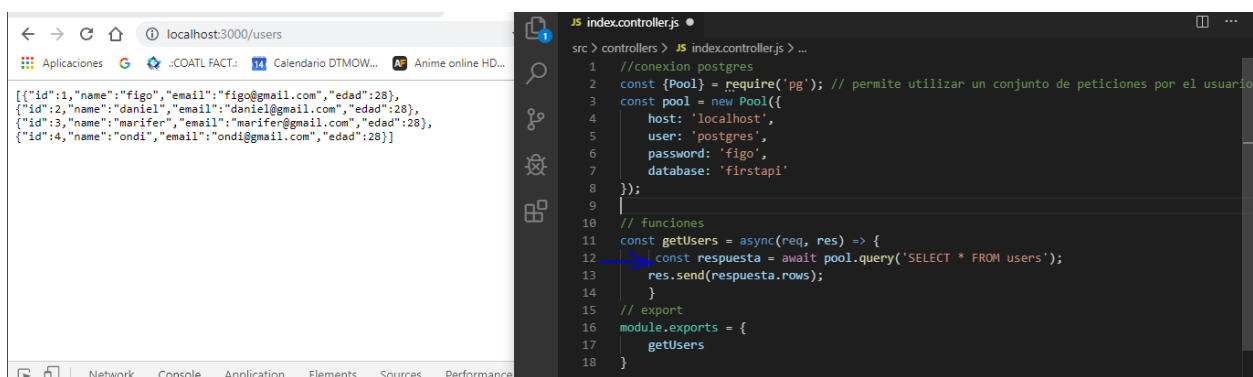
```
CREATE DATABASE firstapi;
create table users(
    id serial primary key,
    name varchar(50),
    email text,
    edad int
);
create table if not exists links (
    id serial primary key,
    link varchar(200),
    detalle varchar(250),
    usersId integer references users(id)
);
insert into users (name, email, edad) values
('figo', 'figo@gmail.com',28),
('daniel', 'daniel@gmail.com',28),
('marifer', 'marifer@gmail.com',28),
('ondi', 'ondi@gmail.com',28);
```

## Conexión PostgreSQL



```
1 //conexion postgres
2 const {Pool} = require('pg'); // permite utilizar un conjunto de peticiones por el usuario
3 const pool = new Pool({
4   host: 'localhost',
5   user: 'postgres',
6   password: 'figo',
7   database: 'firstapi'
8 });
9
10 // funciones
11 const getUsers = async(req, res) => {
12   const respuesta = await pool.query('SELECT * FROM users');
13   res.send(respuesta.rows);
14 }
15 // export
16 module.exports = {
17   getUsers
18 }
```

Generamos nuestra primera consulta y verificamos FUNCIÓN LISTAR **GET**

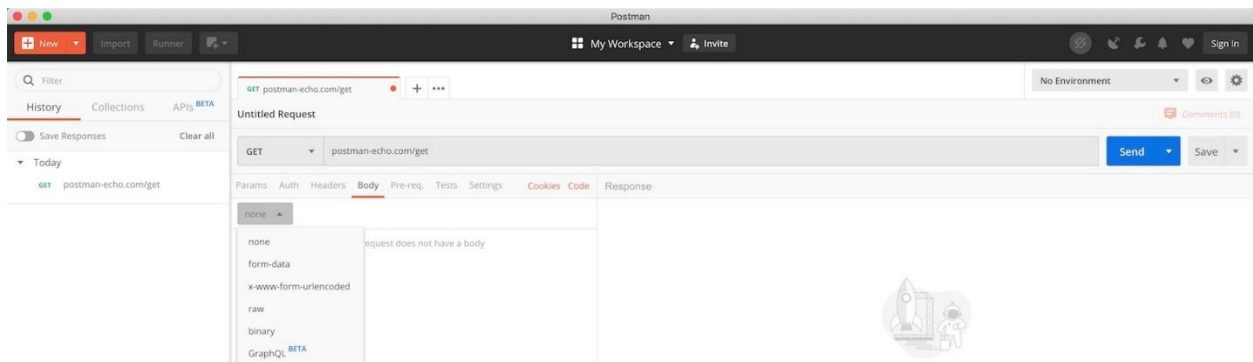
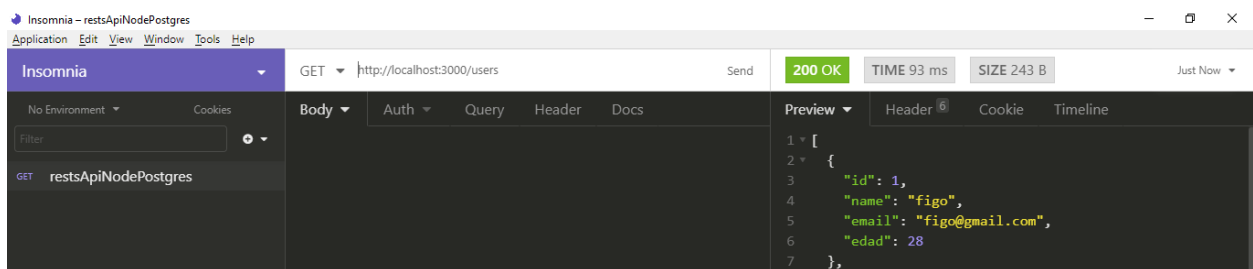


```
1 //conexion postgres
2 const {Pool} = require('pg'); // permite utilizar un conjunto de peticiones por el usuario
3 const pool = new Pool({
4   host: 'localhost',
5   user: 'postgres',
6   password: 'figo',
7   database: 'firstapi'
8 });
9
10 // funciones
11 const getUsers = async(req, res) => {
12   const respuesta = await pool.query('SELECT * FROM users');
13   res.send(respuesta.rows);
14 }
15 // export
16 module.exports = {
17   getUsers
18 }
```

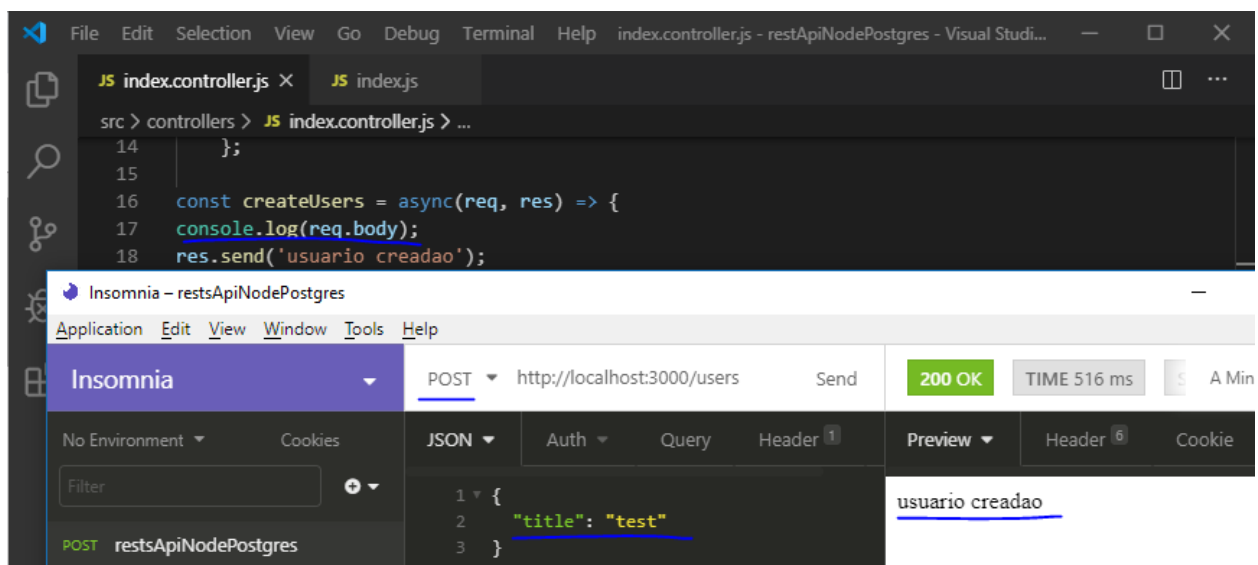
Para trabajar del lado BACK-END utilizaremos la herramienta de INSOMIA – POSTMAN

Permite hacer peticiones a nuestra RESTAPI

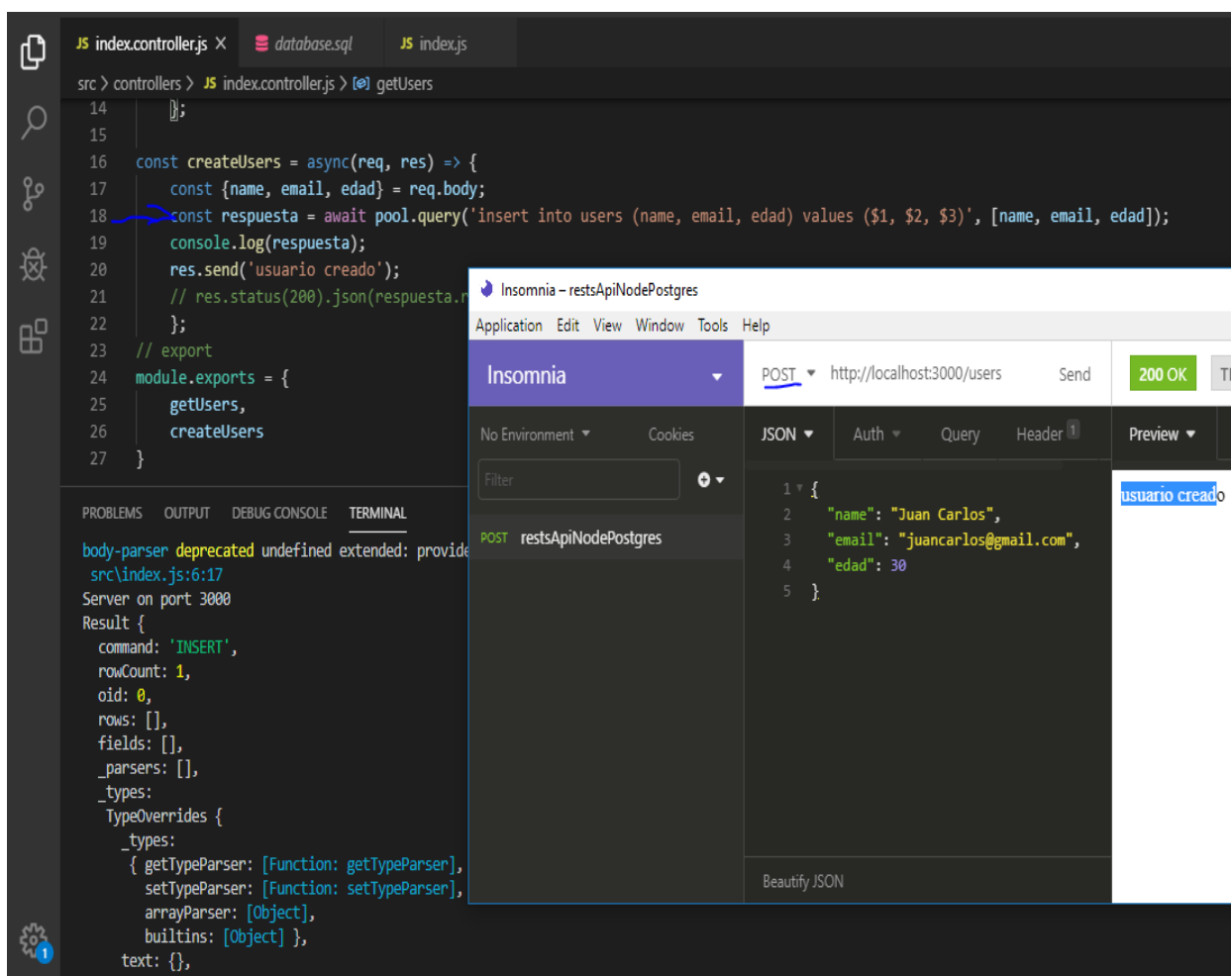
Instalación <https://insomnia.rest/>



Receptamos los valores enviados por post y los imprimimos



## FUNCIÓN DE GUARDAR POST





## Mejorando la respuesta

The screenshot shows the Visual Studio Code editor with the `index.controller.js` file open. The code defines a `createUsers` function that uses `req.body` to create a new user in the database. The response is a JSON object with a `message` and a `user` object. The Insomnia client is also shown, displaying a successful `POST` request to `http://localhost:3000/users` with a `200 OK` status and a response body containing the created user information.

```
const createUsers = async(req, res) => {
  const {name, email, edad} = req.body;
  const respuesta = await pool.query('insert into users (name, email, edad) values ($1, $2, $3)', [name, email, edad]);
  console.log(respuesta);
  res.status(200).json({
    message: 'usuario creado',
    body: {
      user: {name, email, edad}
    }
  });
};

// export
module.exports = {
  getUsers,
  createUsers
};
```

Insomnia - restsApiNodePostgres

Application Edit View Window Tools Help

Insomnia

POST http://localhost:3000/users Send 200 OK TIME 94 ms SIZE 9 Just No

No Environment Cookies

Filter

POST restsApiNodePostgres

JSON Auth Query Header Preview Header Cookie

```
1 {
  2   "name": "Nicol",
  3   "email": "nicol@gmail.com",
  4   "edad": 30
  5 }
```

Beautify JSON

```
1 {
  2   "message": "usuario creado",
  3   "body": {
  4     "user": {
  5       "name": "Nicol",
  6       "email": "nicol@gmail.com",
  7       "edad": 30
  8     }
  9   }
  10 }
```

## FUNCIÓN BUSCAR través de un Id GET

The screenshot shows the Visual Studio Code editor with the `index.js` file open. The code defines a `getUserById` function that uses `req.params.id` to find a user in the database. The response is a JSON object with the user information. The Insomnia client is also shown, displaying a successful `GET` request to `http://localhost:3000/users/4` with a `200 OK` status and a response body containing the user information.

```
const {Router} = require('express');
const router = Router(); // objeto que permite di
//importar controllers
const {getUsers, getUserById, createUsers} = require('./controllers');

router.get('/users', getUsers);
router.get('/users/:id', getUserById);
router.post('/users', createUsers);
module.exports = router;
```

```
const getUserById = async( req, res) => {
  const id = req.params.id;
  const respuesta = await pool.query('select * from users where id = $1', [id]);
  res.json(respuesta.rows);
};

const createUsers = async(req, res) => {
  // ...
};

// export
module.exports = {
  getUsers,
  createUsers,
  getUserById
};
```

Insomnia - restsApiNodePostgres

Application Edit View Window Tools Help

Insomnia

GET http://localhost:3000/users/4 Send 200 OK TIME 78 ms SIZE 9 A Min

No Environment Cookies

Filter

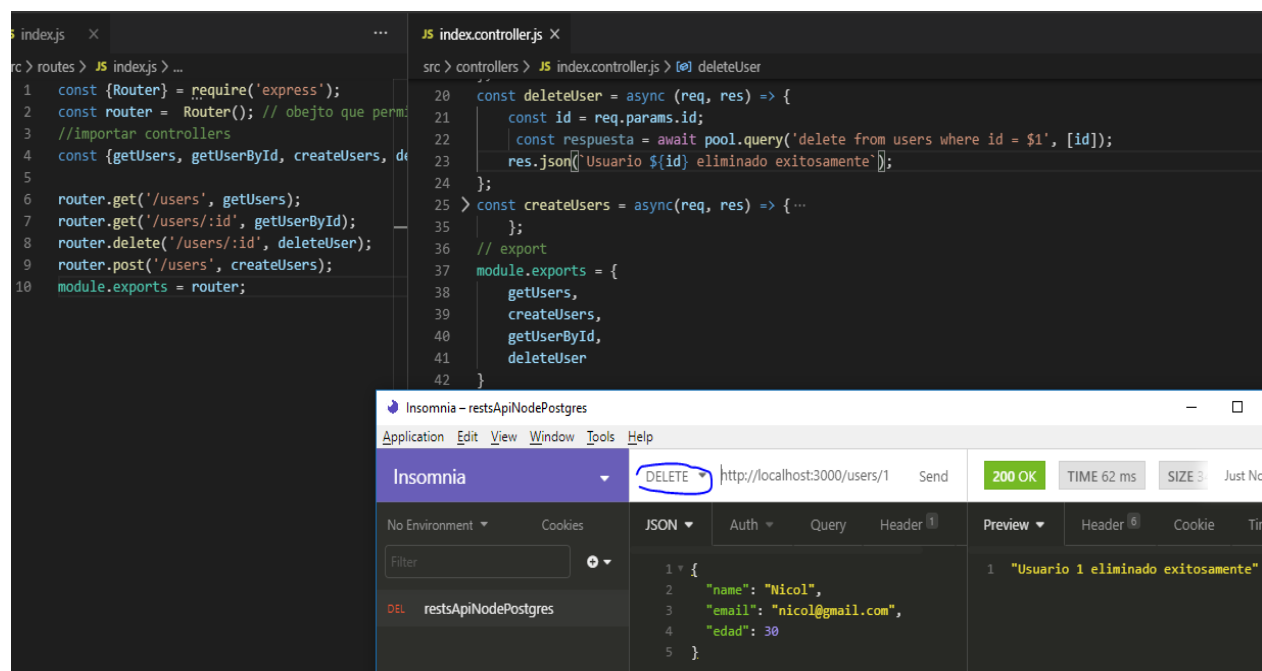
GET restsApiNodePostgres

JSON Auth Query Header Preview Header Cookie

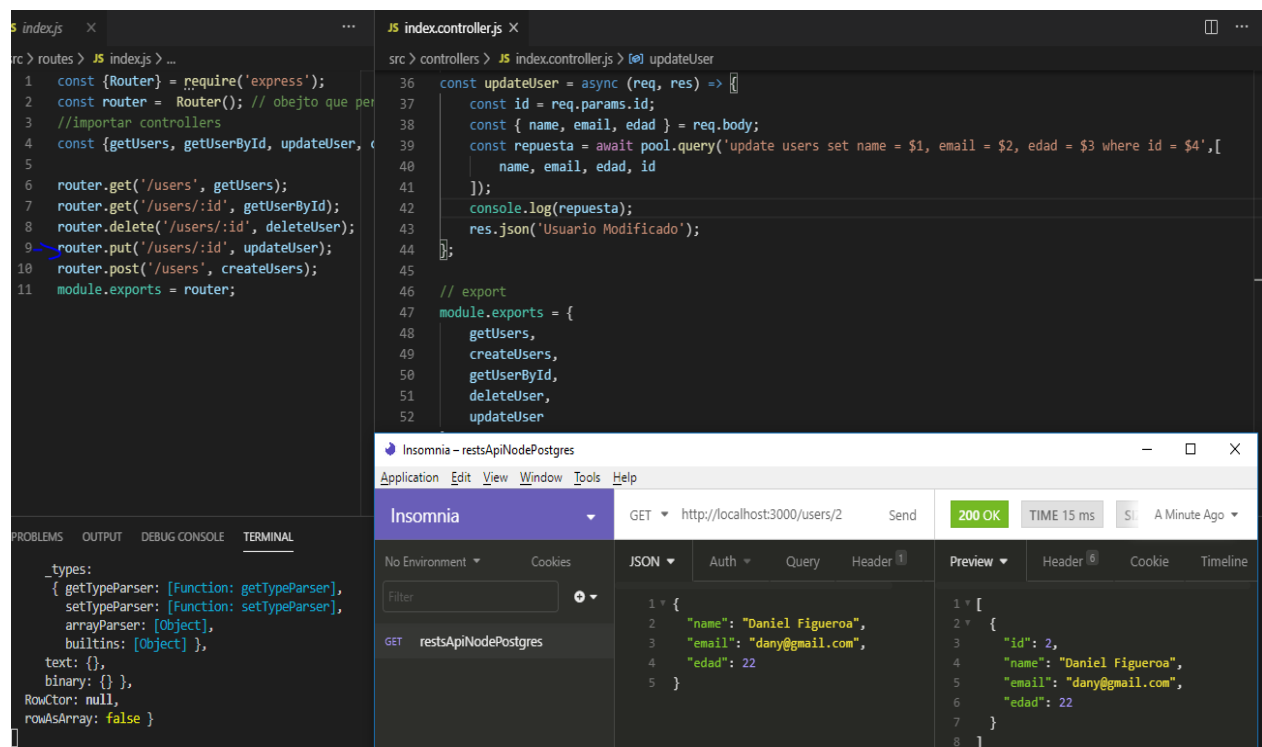
```
1 {
  2   "name": "Nicol",
  3   "email": "nicol@gmail.com",
  4   "edad": 30
  5 }
```

```
1 [
  2   {
  3     "id": 4,
  4     "name": "ondi",
  5     "email": "ondi@gmail.com",
  6     "edad": 28
  7   }
  8 ]
```

## FUNCIÓN ELIMINAR DELETE



## FUNCIÓN MODIFICAR PUT



Trabajo en clase DB Proyecto >> actividades

Proyecto: proyecto\_id, nombre, dirección, fecha\_inicio, fecha\_fin, estado

Actividad: actividad id, evento, descripción, fecha, estado, proyecto\_id

## Consultas con maestro detalle

```

//Create data links
const createLinks = async (req, res) => {
  const {link, detalle, userid} = req.body;
  const respuesta = await
  pool.query('insert into links (link, detalle, userid) values ($1,$2,$3)'
  ,[link, detalle, userid]);
  console.log(respuesta);
  res.json({
    message: 'Link Creado',
    body: {
      user: {link, detalle, userid}
    }
  });
}

//funciones GET de links
const getLinks = async (req, res) => {
  const respuesta = await pool.query('select * from links');
  res.send(respuesta.rows);
}

// funcion buscar por id de usuario
const findIdLink = async (req, res) => {
  const id = req.params.id;
  const respuesta = await pool.query('select * from users,links where users.id
= links.userid and users.id = $1', [id]);
  res.send(respuesta.rows);
}

```

## Archivo de Rutas

```

src > router > JS index.js > ...
1  const {Router} = require('express');
2  const router = Router();
3  // importar controller
4  const { getUsers, createUsers, findUser, deleteUser, updateUsers, createLinks, getLinks, findIdLink } =
5  require('../controller/index.controller');
6  router.get('/users', getUsers);
7  router.get('/users/:id', findUser);
8  router.delete('/users/:id', deleteUser);
9  router.put('/users/:id', updateUsers);
10 router.post('/users', createUsers);
11 //rutas links
12 router.post('/links', createLinks);
13 router.get('/links', getLinks);
14 router.get('/links/:id', findIdLink);
15 module.exports = router;

```