Node.js

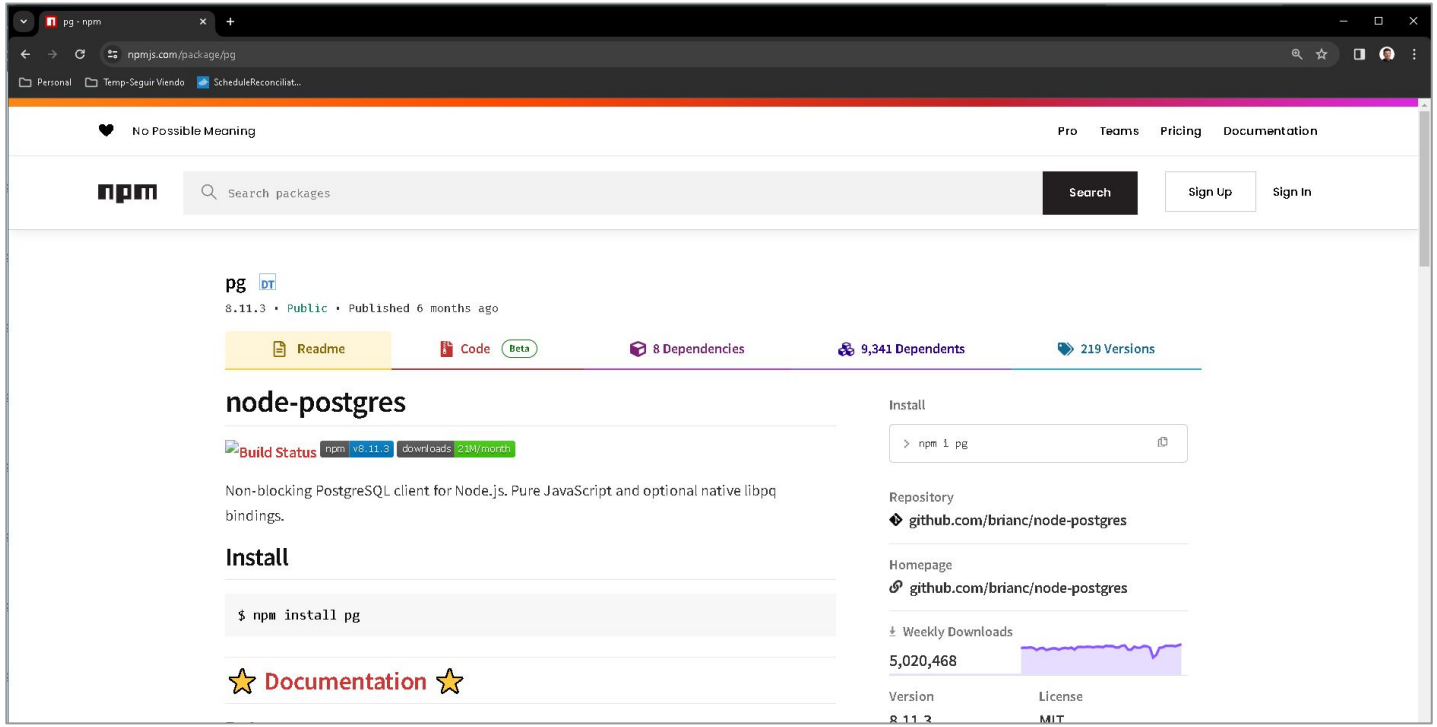
Acceso a datos - pg

https://www.npmjs.com/package/pg

Crear el proyecto

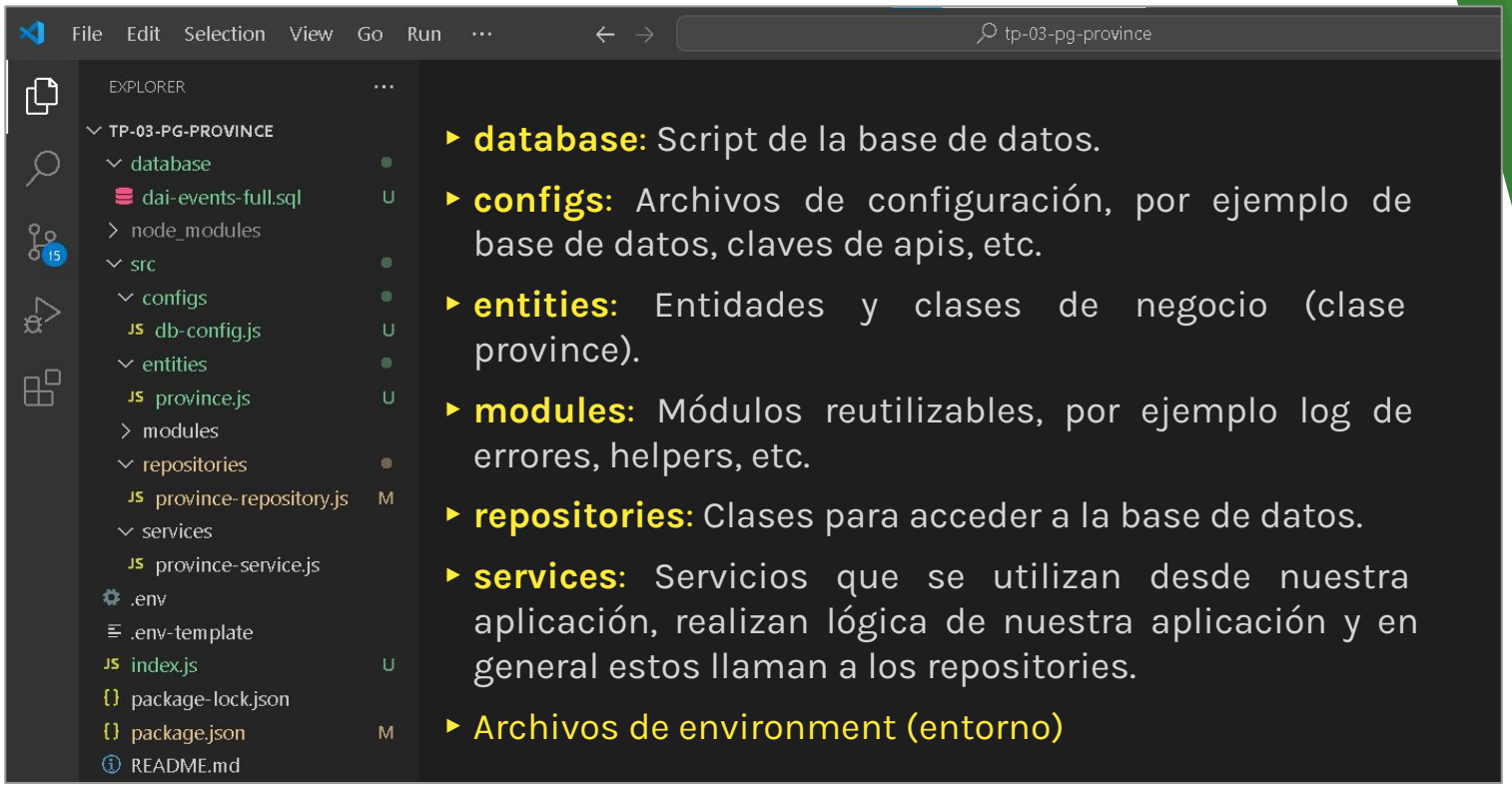
- Creamos un proyecto en node y agregamos la dependencia pg (https://www.npmjs.com/package/pg)

```
npm i pg;
```



Estructura del proyecto (carpetas)

- Acá se puede ver la estructura de carpetas y cuales son los archivos que vamos a crear.



- **database**: Script de la base de datos.
- **configs**: Archivos de configuración, por ejemplo de base de datos, claves de apis, etc.
- **entities**: Entidades y clases de negocio (clase province).
- **modules**: Módulos reutilizables, por ejemplo log de errores, helpers, etc.
- **repositories**: Clases para acceder a la base de datos.
- **services**: Servicios que se utilizan desde nuestra aplicación, realizan lógica de nuestra aplicación y en general estos llaman a los repositories.
- Archivos de environment (entorno)

Archivo province.js

- Este archivo es el modelo que se corresponde con cada uno de los registros de la tabla **provinces**.

```
class Province {
    id;
    name;
    full_name;
    latitude;
    longitude;
    display_order;
}

export default Province;
```

Archivo dbconfig.js

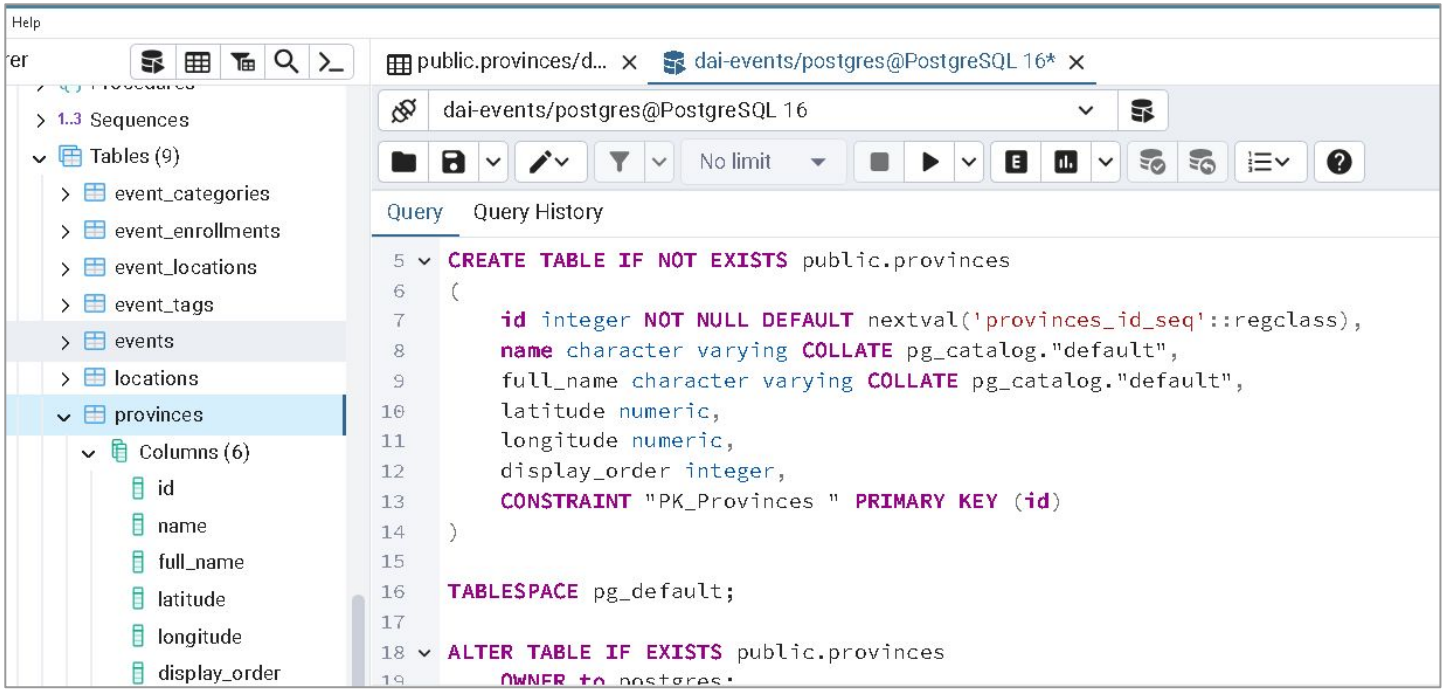
- Este archivo es el responsable de tener la configuración necesaria para acceder a la base de datos. Es el equivalente al **ConnectionString** de .NET Core.

```
const config = {
    host        : "localhost",
    database    : "dai-events",
    user        : "postgres",
    password    : "root",
    port        : 5432
}

export default config;
```

Ejecutando un SELECT - Diagrama

- Vamos a tomar como ejemplo la table **provinces** de la base de datos **dai-events**.



Ejecutando un SELECT - Diagrama

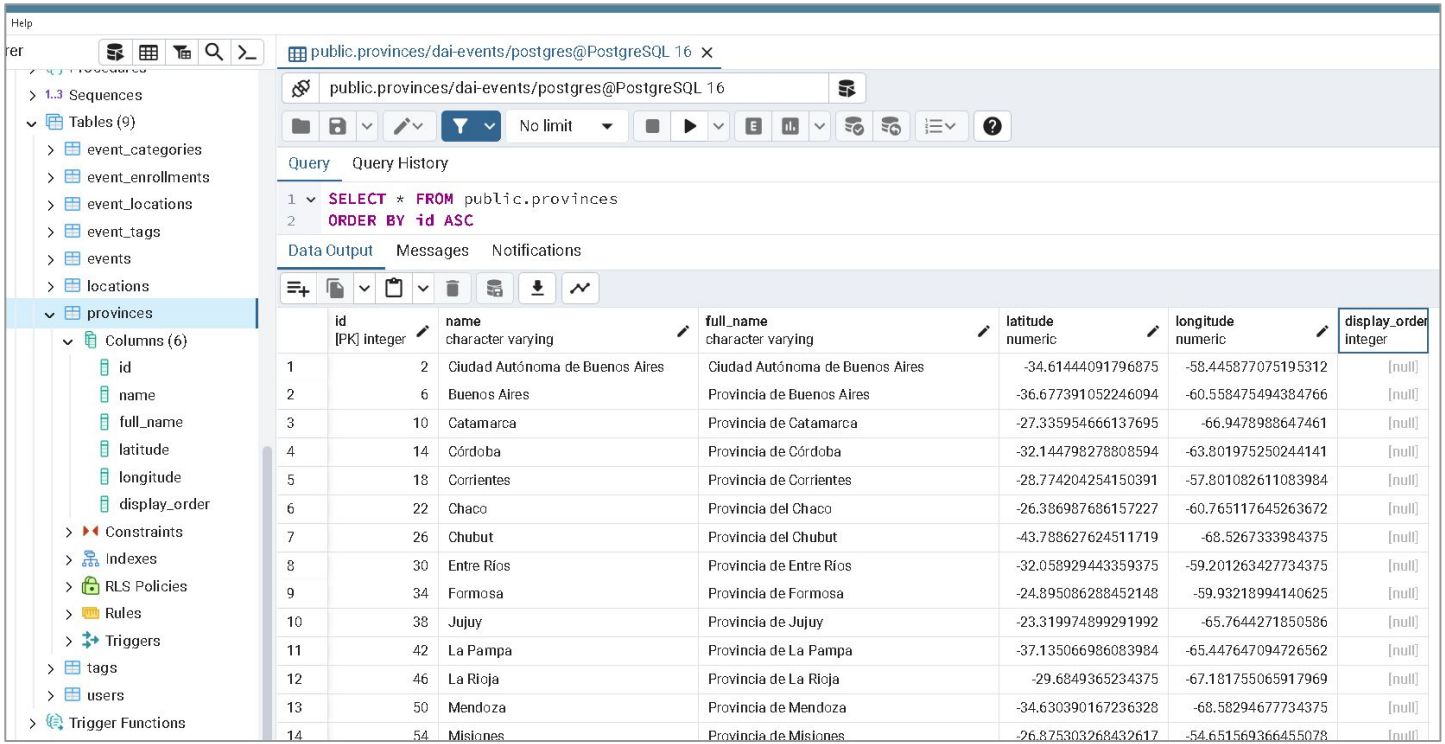
- Vamos a tomar como ejemplo la table **provinces** de la base de datos **dai-events**.

```
CREATE TABLE public.provinces (
    id integer NOT NULL,
    name character varying,
    full_name character varying,
    latitude numeric,
    longitude numeric,
    display_order integer
);
```

Ejecutando un SELECT - Ejemplo

- Chequeamos los datos que tiene la tabla, mediante la consulta:

```
SELECT * FROM public.provinces ORDER BY id DESC
```



	id [PK] integer	name character varying	full_name character varying	latitude numeric	longitude numeric	display_order integer
1	2	Ciudad Autónoma de Buenos Aires	Ciudad Autónoma de Buenos Aires	-34.61444091796875	-58.445877075195312	[null]
2	6	Buenos Aires	Provincia de Buenos Aires	-36.677391052246094	-60.558475494384766	[null]
3	10	Catamarca	Provincia de Catamarca	-27.335954666137695	-66.9478988647461	[null]
4	14	Córdoba	Provincia de Córdoba	-32.144798278808594	-63.801975250244141	[null]
5	18	Corrientes	Provincia de Corrientes	-28.774204254150391	-57.801082611083984	[null]
6	22	Chaco	Provincia del Chaco	-26.386987686157227	-60.765117645263672	[null]
7	26	Chubut	Provincia del Chubut	-43.788627624511719	-68.5267333984375	[null]
8	30	Entre Ríos	Provincia de Entre Ríos	-32.058929443359375	-59.201263427734375	[null]
9	34	Formosa	Provincia de Formosa	-24.895086288452148	-59.93218994140625	[null]
10	38	Jujuy	Provincia de Jujuy	-23.319974899291992	-65.7644271850586	[null]
11	42	La Pampa	Provincia de La Pampa	-37.135066986083984	-65.447647094726562	[null]
12	46	La Rioja	Provincia de La Rioja	-29.6849365234375	-67.181755065917969	[null]
13	50	Mendoza	Provincia de Mendoza	-34.630390167236328	-68.58294677734375	[null]
14	54	Misiones	Provincia de Misiones	-26.875303268432617	-54.651569366455078	[null]

Ejecutando un SELECT - Codigo

- Para obtener los registros de la tabla `provinces`, es necesario crear un **Client** y conectarse usando el método **connect()** enviando por parámetro la configuración (config). Luego realizar la consulta mediante el método **query()**, del objeto **client**.

```
import config from './src/configs/db-config.js'
import pkg from 'pg'
const { Client }  = pkg;
```

```
// https://node-postgres.com/apis/client
const client = new Client(config);
await client.connect();

let sql = `SELECT * from provinces`; // `... limit 5`
let result = await client.query(sql);
await client.end();
// 'rows' es un array. rows[0] el 1ºer registro.
console.log(result.rows);
```

Ejecutando un SELECT - Respuesta

- La respuesta sería la siguiente.

```
[
  {
    id: 2,
    name: 'Ciudad Autónoma de Buenos Aires',
    full_name: 'Ciudad Autónoma de Buenos Aires',
    latitude: '-34.61444091796875',
    longitude: '-58.445877075195312',
    display_order: null
  },
  ...
  {
    id: 94,
    name: 'Tierra del Fuego, Antártida e Islas del Atlántico Sur',
    full_name: 'Provincia de Tierra del F...',
    latitude: '-82.521133422851562',
    longitude: '-50.74285888671875',
    display_order: null
  }
]
```

Ejecutando - SELECT con parámetros

- Para obtener los registros de la tabla provinces, es necesario crear un **Client** y conectarse usando el método **connect()** enviando por parámetro la configuración (config). Luego realizar la consulta mediante el método **query()**, del objeto **client**.

```
const client = new Client(config);
await client.connect();

const sql = 'SELECT * from provinces WHERE id=$1';
// Array con los valores.
const values = [id];
const result = await client.query(sql, values);
await client.end();

// En 'result.rows.length' tenemos un 1.
// En 'rows[0]' tenemos el resultado de la consulta (un objeto)
console.log('Resultados: length', result.rows.length);
console.log('result.rows[0]', result.rows[0]);
```

Ejecutando - INSERT con parámetros

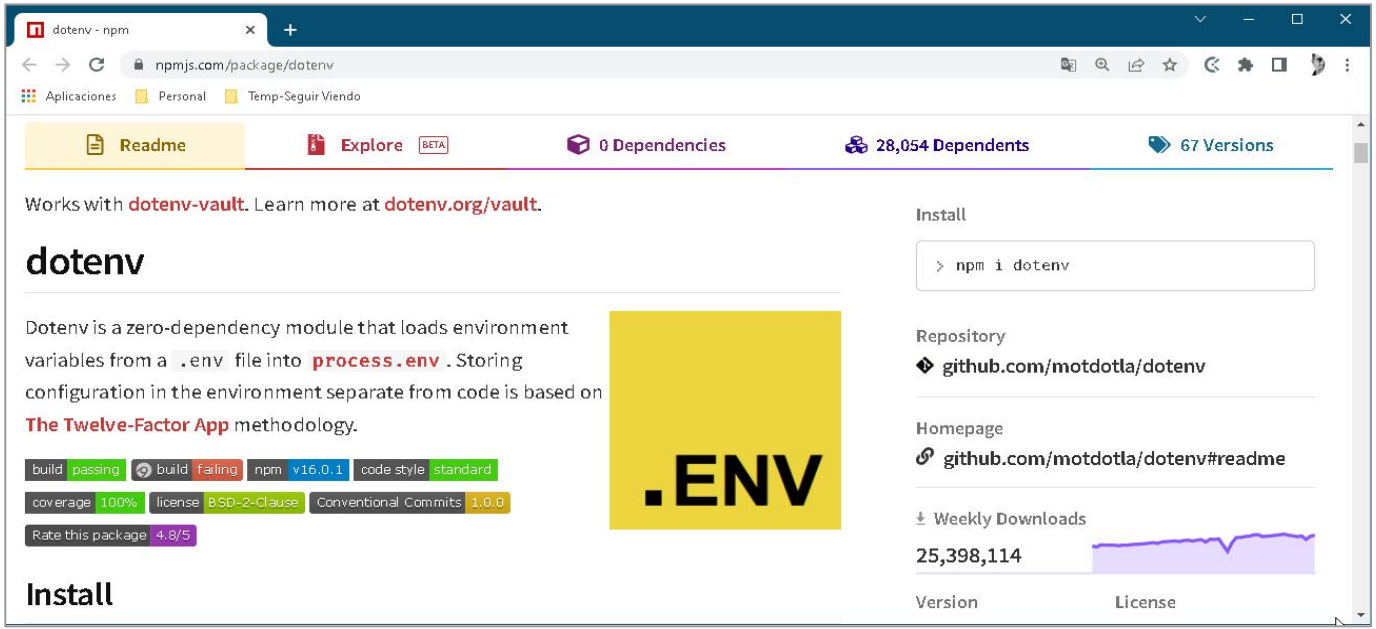
- Ejemplo de inserción de un registro en la tabla provinces.

```
const client = new Client(config);
await client.connect();
const sql = `
    INSERT INTO provinces
        (name, full_name, latitude, longitude, display_order)
    VALUES
        ($1, $2, $3, $4, $5)`;
// Array con los valores.
const values =
    ['Jujuy', 'Provincia de Jujuy', -23.319974, -65.764427, 3];
const result = await client.query(sql, values);
await client.end();

// En 'rowCount' tenemos el número de registros
//  procesados por el último comando. En este caso 1.
console.log('rowCount: ', result.rowCount);
```

Haciendo la configuración más flexible.

- Se puede hacer que la configuración de la conexión a la base de datos sea más flexible, poniendo los datos que varían fuera del código. Para ello utilizaremos el módulo dotenv que se encuentra en (https://www.npmjs.com/package/dotenv)



¿Preguntas?

Preguntas de revisión

- ¿Para qué se utiliza el paquete pg?
- ¿En que propiedad del objeto result se obtienen los resultados de una consulta del tipo SELECT?
- ¿En que propiedad del objeto result se obtienen los resultados de los registros afectados por el último comando, por ejemplo un INSERT, UPDATE o DELETE?
- ¿Para qué se deben utilizar los parámetros $1, $2, etc. en vez de concatenar las instrucciones SQL?