



# 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.

The screenshot shows the VS Code Explorer sidebar for a project named 'tp-03-pg-province'. The file tree is as follows:

- TP-03-PG-PROVINCE
  - database
    - dai-events-full.sql
  - node\_modules
  - src
    - configs
      - db-config.js
    - entities
      - province.js
    - modules
    - repositories
      - province-repository.js
    - services
      - province-service.js
  - .env
  - .env-template
  - index.js
  - package-lock.json
  - package.json
  - README.md

On the right side of the Explorer, there are descriptions for several key folders:

- **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;
```



pg



# Ejecutando un SELECT - Diagrama



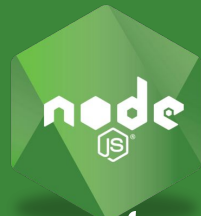
- Vamos a tomar como ejemplo la table **provinces** de la base de datos **dai-events**.

The screenshot shows a database management interface with a left sidebar containing a tree view of database objects. The 'provinces' table is selected, and its columns (id, name, full\_name, latitude, longitude, display\_order) are listed. The main pane displays the SQL script for creating the table.

```
5 CREATE TABLE IF NOT EXISTS public.provinces
6 (
7     id integer NOT NULL DEFAULT nextval('provinces_id_seq'::regclass),
8     name character varying COLLATE pg_catalog."default",
9     full_name character varying COLLATE pg_catalog."default",
10    latitude numeric,
11    longitude numeric,
12    display_order integer,
13    CONSTRAINT "PK_Provinces " PRIMARY KEY (id)
14 )
15
16 TABLESPACE pg_default;
17
18 ALTER TABLE IF EXISTS public.provinces
19     OWNER to postgres;
```



# Ejecutando un SELECT - Diagrama



pg

- 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**

The screenshot shows a PostgreSQL client interface with the query `SELECT * FROM public.provinces ORDER BY id ASC` executed. The results are displayed in a table with 7 columns: `id` (integer), `name` (character varying), `full_name` (character varying), `latitude` (numeric), `longitude` (numeric), and `display_order` (integer). The table contains 14 rows of data, ordered by `id` in ascending order.

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





# Ejecutando un SELECT -Codigo



- Para obtener los registros de la tabla provincias, 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 provincias`; // `... 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](https://www.npmjs.com/package/dotenv) que se encuentra en (<https://www.npmjs.com/package/dotenv>)



The screenshot shows the npm package page for **dotenv**. The page includes a header with navigation links like 'Readme', 'Explore', and 'Dependencies'. The main content area describes **dotenv** as a zero-dependency module for loading environment variables from a `.env` file into `process.env`. It also mentions 'The Twelve-Factor App' methodology. On the right, there's an 'Install' section with a code block showing `> npm i dotenv`, a 'Repository' link to `github.com/motdotla/dotenv`, and a 'Homepage' link to `github.com/motdotla/dotenv#readme`. At the bottom, there's a 'Weekly Downloads' graph showing 25,398,114 downloads and a 'Version' section.

dotenv - npm

npmjs.com/package/dotenv

Aplicaciones Personal Temp-Seguir Viendo

Readme Explore BETA 0 Dependencies 28,054 Dependents 67 Versions

Works with **dotenv-vault**. Learn more at [dotenv.org/vault](https://dotenv.org/vault).

## dotenv

Dotenv is a zero-dependency module that loads environment variables from a `.env` file into `process.env`. Storing configuration in the environment separate from code is based on **The Twelve-Factor App** methodology.

build passing build failing npm v16.0.1 code style standard  
coverage 100% license BSD-2-Clause Conventional Commits 1.0.0  
Rate this package 4.8/5

### Install

Install

```
> npm i dotenv
```

Repository

[github.com/motdotla/dotenv](https://github.com/motdotla/dotenv)

Homepage

[github.com/motdotla/dotenv#readme](https://github.com/motdotla/dotenv#readme)

Weekly Downloads

25,398,114

Version License



## ¿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?

