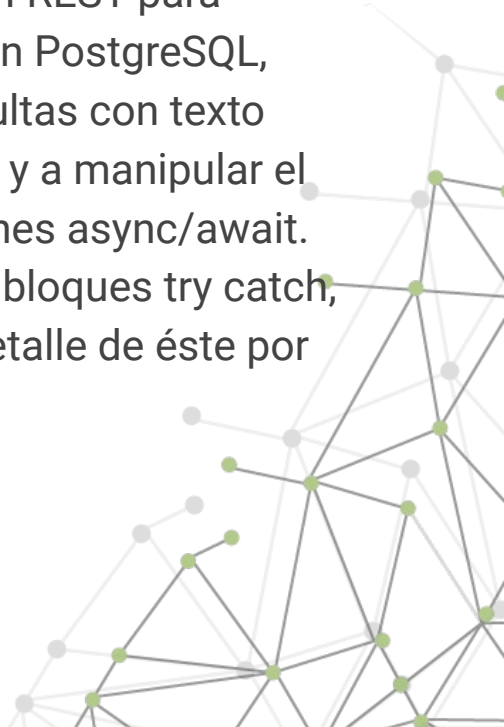
Implementación y gestión de una base de datos

Instalación y configuración de la librería pg

{desafío} latam_

¿Qué aprenderemos en este módulo?

En el módulo de Acceso a bases de datos con Node, realizarás una API REST para disponibilizar diferentes rutas para la gestión de datos almacenados en PostgreSQL, utilizando el paquete pg de NPM. Además, aprenderás a realizar consultas con texto parametrizado utilizando un JSON como argumento del método query y a manipular el resultado de cada consulta SQL controlando su asincronía con funciones async/await. Finalmente, se ejecutarán operaciones transaccionales SQL dentro de bloques try catch, capturando los posibles errores que puedan surgir e imprimiendo el detalle de éste por consola.



{desafío} latam_

Implementar la conexión a una base de datos PostgreSQL en una aplicación Node utilizando buenas prácticas.

Programar instrucciones para la obtención y manipulación de información desde una base de datos utilizando buenas prácticas, acorde al entorno Node.js.

{desafío} latam_

- Unidad 1:
  Implementación y gestión de una base de datos
- Unidad 2:
  Transacciones y API REST
- Unidad 3:
  Trabajo práctico

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Reconoce las dependencias y paquetes requeridos para la implementación de una conexión a una base de datos PostgreSQL en el entorno Node.*
- *Distingue las características, ventajas y desventajas de una conexión de tipo pooling en comparación a una simple.*
- *Utiliza sentencias de conexión y desconexión a una base de datos PostgreSQL utilizando el entorno Node.js.*

¿Tienes algún conocimiento sobre los contenidos que veremos hoy?

{desafío} latam_

/* Librería pg para Node */

Librería pg para Node

La librería pg es utilizada en Node y contiene una colección de métodos, los cuales permiten desarrollar aplicaciones que utilicen las principales características incluidas en la base de datos relacional PostgreSQL.

/* Características principales */

Características principales

- Permite realizar conexiones nativas con la base de datos PostgreSQL.
- Permite la conexión de un cliente para realizar consultas estáticas.
- Permite agrupar conexiones de distintos clientes, configurando la cantidad aceptada y tiempos de inactividad antes de ser desconectados.
- Acepta SSL en las conexiones.
- Permite el control y captura de errores.
- Acepta eventos que puedan monitorear el estado de la base de datos.
- Cuenta con soporte para las características principales de Node, cómo: callbacks, promises, async/await, cursors entre otras.

/* Instalación, configuración y conexión */

Conexión con un objeto de tipo cliente

- Para realizar esta conexión crearemos una base de datos llamada users con los campos o columnas id, nombre y apellido.

```
create database users;

create table users (id serial, primary key, name varchar(50) not null, email varchar(255) not null);

insert into users(name, email) values ('Nombre 1', 'email1@email.com');
```

Conexión con un objeto de tipo cliente

- Ahora, nos dirigimos al editor de código y realizamos la instalación del paquete pg.

npm install pg

- Luego definimos el objeto de tipo cliente para la conexión a la base de datos.

```
const { Client } = require("pg");

const conection = async () => {
  const client = new Client({
    host: "localhost",
    port: 5432,
    database: "tu_base_de_datos",
    user: "tu_usuario",
    password: "tu_password",
  });
  await client.connect();
  const result = await client.query("SELECT * FROM users");
  console.log(result);
  await client.end();
};
conection();
```

Conexión con un objeto de tipo cliente

Result

- Al obtener una conexión satisfactoria, veremos como resultado algunos datos del servidor.

```
PS D:\Node JS\connectObjectClient> node .\index.js
Result {
  command: 'SELECT',
  rowCount: 3,
  oid: null,
  rows: [
    { id: 1, name: 'Nombre 1', email: 'email1@email.com' },
    { id: 2, name: 'Nombre 2', email: 'email2@email.com' },
    { id: 3, name: 'Nombre 3', email: 'email3@email.com' }
  ],
```

Conexión con un objeto de tipo cliente

Result

- Recibimos un objeto Result, donde una de sus keys es el rows.
- La key rows es quien contiene la información de nuestra base de datos.
- Para obtener esto, solo debemos agregar `console.log(result.rows)`

```
[
  { id: 1, name: 'Nombre 1', email: 'email1@email.com' },
  { id: 2, name: 'Nombre 2', email: 'email2@email.com' },
  { id: 3, name: 'Nombre 3', email: 'email3@email.com' }
]
```

Código final conexión de tipo Cliente

- Una conexión de tipo cliente se crea y se conecta a la base de datos una sola vez.
- Se puede usar para realizar consultas y operaciones SQL, pero una vez que se ha terminado de usar, debe cerrarse para liberar recursos.

Cierre de conexión

```
const { Client } = require("pg");

const conection = async () => {
  const client = new Client({
    host: "localhost",
    port: 5432,
    database: "tu_base_de_datos",
    user: "tu_usuario",
    password: "tu_password",
  });
  await client.connect();
  const result = await client.query("SELECT * FROM users");
  console.log(result.rows);
  await client.end();
};
conection();
```

Instalación, configuración y conexión con Pool

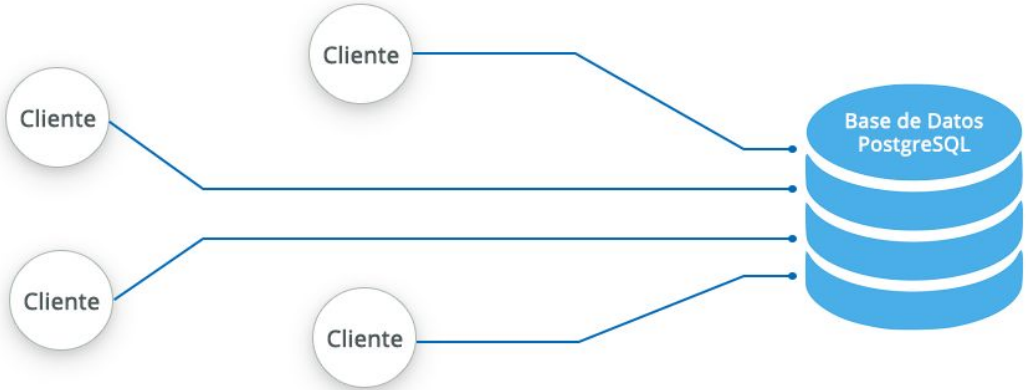
La instalación del paquete pg no es diferente a los demás, solo debes ocupar el siguiente comando. Una vez descargado el paquete, debemos importar una clase dentro de la misma dependencia llamada Pool, el cual recibirá un objeto de configuración y será el gestor de nuestra base de datos.

```
npm install pg
```

```
const { Pool } = require('pg');
```

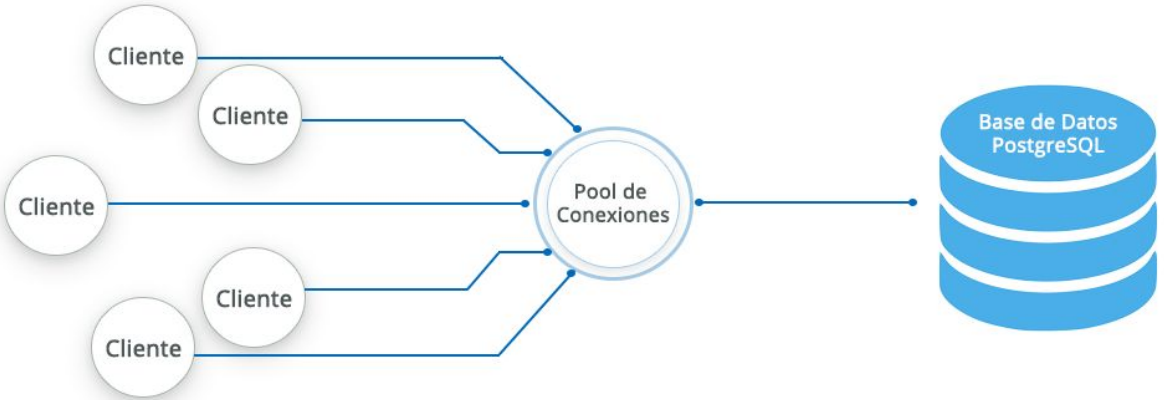
Instalación, configuración y conexión

Diagramas que representan las conexiones realizadas con la clase Client



Instalación, configuración y conexión

Diagramas que representan las conexiones realizadas con la clase Client



/* Configuración */

Configuración

La clase Pool recibe en su constructor un objeto de configuración, el cual debe especificar las credenciales de acceso, el servidor donde está alojada la base de datos, el nombre de la base de datos, entre otras posibles propiedades.

/* String de conexión */

String de conexión

Un string de conexión es básicamente una cadena de texto, que agrupa los parámetros con la información de las credenciales necesarias para generar una conexión con la base de datos.

```
'postgresql://dbuser:dbpassword@dbserver:dbport/database'
```

String de conexión

- dbuser: Es el nombre de usuario configurado en la base de datos que nos permitirá la conexión con el servidor.
- dbpassword: Contraseña asignada al usuario.
- dbserver: Corresponde al Nombre o IP del servidor PostgreSQL.
- dbport: Es el puerto del servidor donde se ejecuta PostgreSQL, por defecto se utiliza el puerto 5432.
- database: Corresponde al nombre de la base de datos.

Hasta aquí, ¿podrías decir con tus palabras que has aprendido?

/* Creación de la base de datos */

Creación de la base de datos

Procedamos con la creación de la base de datos, pero antes definamos la temática a abordar, la cual en esta ocasión y para toda esta lectura será la gestión de una base de datos para la tienda de ropa B&H. Para la creación de la base de datos abre tu terminal psql y escribe el siguiente comando para crear una base de datos llamada JEANS:

```
CREATE DATABASE JEANS
```

Creación de la base de datos

Con la base de datos creada, ahora debemos cambiar los valores de nuestra configuración, los cuales serían los siguientes:

- Usuario: postgres
- Password: postgres
- Server: localhost
- Port: 5432
- Database: jeans

Resultado: `'postgresql://postgres:postgres@localhost:5432/jeans'`

Creación de la base de datos

Esta URI debemos definirla como valor de una propiedad llamada “connectionString” dentro de un objeto, el cual será la configuración de nuestra conexión con la clase Pool.

```
const { Pool } = require('pg');

const pool = new Pool({
  connectionString:
'postgresql://postgres:postgres@localhost:5432/jeans'
});
```

Creación de la base de datos

Esta URI debemos definirla como valor de una propiedad llamada “connectionString” dentro de un objeto, el cual será la configuración de nuestra conexión con la clase Pool.

```
const { Pool } = require('pg');

const pool = new Pool({
  connectionString:
'postgresql://postgres:postgres@localhost:5432/jeans'
});
```

/* Conexión por objeto de configuración */

Conexión por objeto de configuración

Otra forma de definir la configuración de nuestra conexión a PostgreSQL es a través de un objeto, compuesto por diferentes propiedades que representarán cada uno de los parámetros que definimos anteriormente en la URI.

```
const config = {
   user: 'postgres',
   host: 'localhost',
   database: 'jeans',
   password: 'postgres',
   port: 5432,
}
```

Conexión por objeto de configuración

Ahora este objeto debemos pasarlo al constructor de la clase Pool, con el objetivo de cederle a la instancia los atributos correspondientes a la configuración y pueda proceder con la conexión a PostgreSQL.

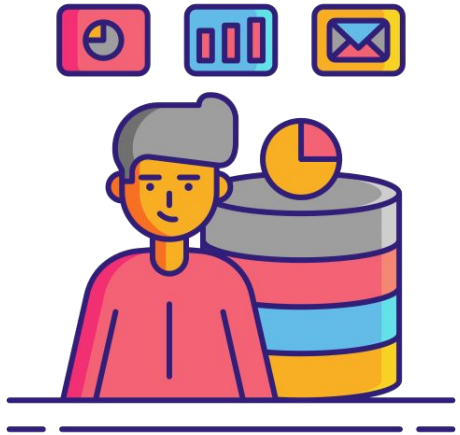
```
const { Pool } = require('pg');

const config = {
   user: 'postgres',
   host: 'localhost',
   database: 'jeans',
   password: 'postgres',
   port: 5432,
}

const pool = new Pool(config );
```

Conexión por objeto de configuración

Considera que deberás especificar tu usuario y contraseña correspondiente, en caso de tener recién instalado PostgreSQL estas credenciales deberán ser “postgres” como usuario y contraseña, no obstante si lo deseas puedes crear un usuario y una contraseña personal para proceder con los ejercicios y puedas probar correctamente una conexión.



/* Primera consulta y su respuesta */

Primera consulta y su respuesta

Consulta

Para probar una conexión a nuestra base de datos, debemos simplemente ocupar funciones async/await y usar el método query de nuestra instancia pool, internamente escribimos una sentencia SQL que nos retorne la fecha actual producida por PostgreSQL.

```
const getDate = async () => {
    const result = await pool.query("SELECT NOW()")
    console.log(result)
}

getDate()
```

Primera consulta y su respuesta

Respuesta

```
$ node index.js
Result {
  command: 'SELECT',
  rowCount: 1,
  oid: null,
  rows: [ { now: 2020-10-17T00:57:14.622Z } ],
  fields: [
    Field {
      name: 'now',
      tableID: 0,
      columnID: 0,
      dataTypeID: 1184,
      dataTypeSize: 8,
      dataTypeModifier: -1,
      format: 'text'
    }
  ],
  _parsers: [ [Function: parseDate] ],
  _types: TypeOverrides {
    _types: {
      getTypeParser: [Function: getTypeParser],
      setTypeParser: [Function: setTypeParser],
      arrayParser: [Object],
      builtins: [Object]
    },
    text: {},
    binary: {}
  },
  RowCtor: null,
  rowAsArray: false
}
```

/* Propiedades del objeto result y la asincronía en una consulta */

Propiedades del objeto result y la asincronía en una consulta

Consideraciones previas

Utiliza el siguiente comando en tu terminal psql para crear la tabla ropa con las columnas: id con la instrucción SERIAL para que sea un entero autoincremental, nombre, color y talla.

```
CREATE TABLE ropa (
  id SERIAL PRIMARY KEY,
  nombre varchar(50) NOT NULL,
  color varchar(10) NOT NULL,
  talla varchar(5) NOT NULL
);
```

Propiedades del objeto result y la asincronía en una consulta

Queries con texto plano

Texto plano: Consultas con formato string que indican exactamente todos los datos que se usarán para realizar la query a la base de datos. Por ejemplo:

```
"SELECT * FROM ropa WHERE id=25"
```

Propiedades del objeto result y la asincronía en una consulta

Texto plano con parámetros

Texto plano con parámetros: Consultas en formato string, que incluyen parámetros u ocultan parte de la data que se está describiendo en la consulta, por medio de parámetros definidos programáticamente. Por ejemplo:

```
"SELECT * FROM ropa WHERE id=$1"
```

Propiedades del objeto result y la asincronía en una consulta

El objeto result

Dentro del objeto result tenemos varias propiedades interesantes, pero las 2 más destacables son las propiedades “rows” y “fields”, las cuales representarán en formato de arreglo las filas y los campos que fueron agregados en la tabla a la que le estemos realizando la consulta.

```
pool.query("<consulta> RETURNING *")
```

Según lo aprendido hoy

¿Cómo puedes optimizar la conexión a una base de datos utilizando la librería pg para Node.js?

{desafío} latam_

Próxima sesión...

- *Utiliza sentencias para la captura y procesamiento de errores de conexión y estados de base de datos utilizando el entorno Node.js*
- *Utiliza sentencias para la consulta de datos mediante una conexión simple o un pool de conexiones para dar solución a un requerimiento.*

{desafío}
latam_

Academia de
talentos digitales