Créer un projet avec Node.js et Postgres

Configuration

Initialiser un projet Node.js :

Exécutez la commande suivante pour initialiser un projet Node.js avec les paramètres par défaut :

npm init --y

Créer un fichier .env pour stocker les variables d'environnement de la base de données :

Utilisez cette commande pour créer un fichier .env qui contiendra les informations sensibles :

touch .env

Créer un dossier avec un fichier de configuration pour la base de données :

Pour organiser les configurations de la base de données, créez un dossier dédié et un fichier de configuration :

mkdir database
touch database/db.js

Créer le fichier principal de Node.js :

Créez un fichier main.js qui servira de point d'entrée de votre application :

touch main.js

Installer dotenv pour lire les variables d'environnement :

Installez le package dotenv pour gérer les variables d'environnement de votre projet :

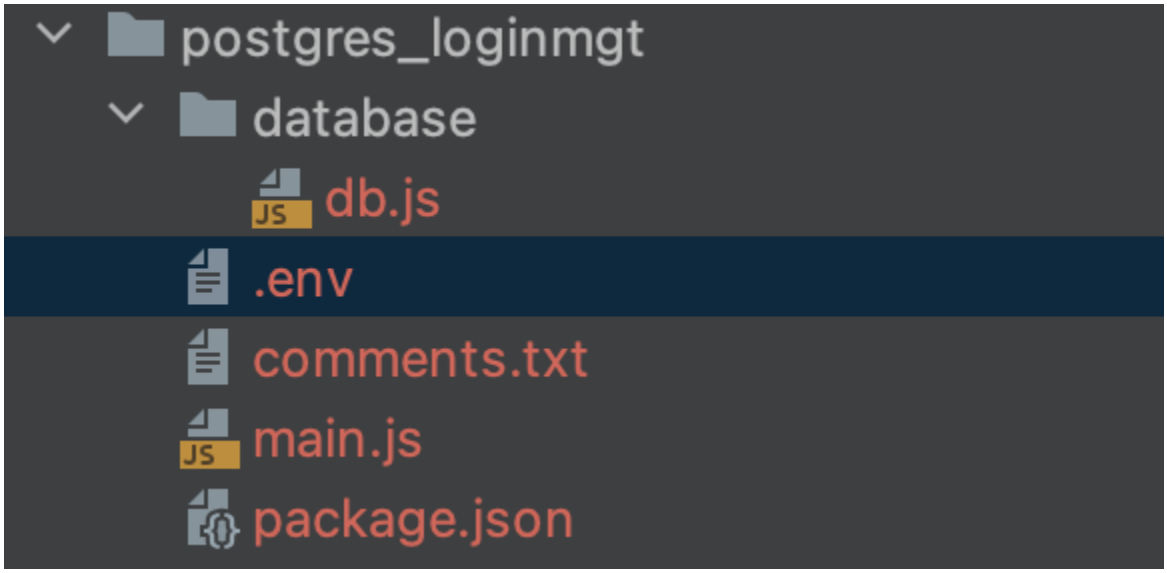
npm install dotenv

Installer postgres-node :

Enfin, installez le package pg pour intégrer Postgres à votre projet Node.js :

npm install pg pg-format

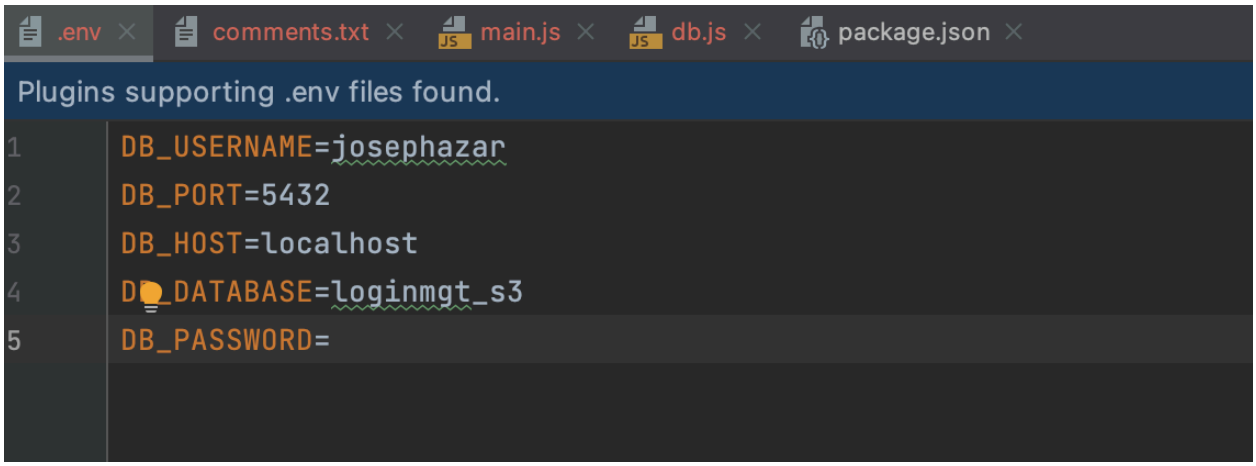
https://node-postgres.com/



.env

Dans le fichier .env mettez les valeurs de connexion à la base de données :

```
DB_USERNAME=username
DB_PORT=5432
DB_HOST=localhost
DB_DATABASE=database
DB_PASSWORD=
```



database/db.js: Exportation d'un objet pool de base de données

```
const {Pool} = require("pg");
require('dotenv').config();

const credentials = {
   user: process.env.DB_USERNAME,
   host: process.env.DB_HOST,
   database: process.env.DB_DATABASE,
   password: process.env.DB_PASSWORD,
   port: process.env.DB_PORT,
};
const pool =  new Pool(credentials)
module.exports = pool;
```

Le fichier database/db.js dans votre projet Node.js est essentiel pour gérer la connexion à votre base de données PostgreSQL. Voici pourquoi il est important :

Importation des Dépendances :

const {Pool} = require("pg"); : Importe la classe Pool du module pg, utilisée pour interagir avec la base de données PostgreSQL. Pool est efficace pour gérer plusieurs connexions à la base de données.

Variables d'Environnement :

require('dotenv').config(); : Charge les variables d'environnement du fichier .env dans process.env. Cette méthode est cruciale pour sécuriser les informations sensibles comme les identifiants de la base de données, en évitant de les intégrer directement dans le code.

Identifiants de la Base de Données :

L'objet credentials contient les détails de connexion à la base de données : nom d'utilisateur, hôte, base de données spécifique, mot de passe, et port. Cela permet une connexion sécurisée à la base de données.

Création de l'Instance de Pool :

const pool = new Pool(credentials) : Crée une instance de Pool avec les identifiants fournis, facilitant la gestion des connexions à la base de données.

Exportation du Pool :

module.exports = pool; : Exporte l'instance de pool pour qu'elle soit utilisée dans d'autres parties de l'application, permettant ainsi des requêtes à la base de données.

https://node-postgres.com/features/pooling

Exemple 1 : Obtenir un utilisateur par son identifiant

main.js

```
const pool = require("./database/db");

no usages
async function getUser(id){
    const client = await pool.connect()
    const res = await client.query('SELECT * FROM UTILISATEURS WHERE user_id = $1', [id])
    console.log(res.rows[0])
    client.release()
}

getUser( id: 1);
```

```
(base) josephazar@Josephs-MBP-2 postgres_loginmgt % node main.js
{
  first_name: 'Joseph',
  last_name: 'Azar',
  date_created: 2023-11-08T08:27:29.477Z,
  group_id: 1,
  password: 'joseph123',
  user_id: 1
}
```

Bookmarks

ructure

Exemple 2 : recherche par nom complet

main.js

```
12  async function getUserByFullNameLike(nom, prenom) {
13      const client = await pool.connect();
14      try {
15          const query = 'SELECT * FROM UTILISATEURS WHERE last_name LIKE $1 AND first_name LIKE $2';
16          const res = await client.query(query, [`%${nom}%`, `%${prenom}%`]);
17          console.log(res.rows);  // Logs all matching users
18          return res.rows;  // Returns the array of users
19      } catch (error) {
20          console.error(error);
21          throw error;  // Propagates the error
22      } finally {
23          client.release();  // Ensure the client is released in case of success or error
24      }
25  }
26
27  getUserByFullNameLike( nom: 'Perr', prenom: 'Milvyne').then(users => {
28      console.log('Found Users:', users);
29  }).catch(error => {
30      console.error('Error:', error);
31  });
```

```
(base) josephazar@Josephs-MBP-2 postgres_loginmgt % node main.js
[
  {
    first_name: 'Milvyne',
    last_name: 'Perrolet',
    date_created: 2023-11-08T08:27:29.479Z,
    group_id: 2,
    password: 'milvyne123',
    user_id: 2
  }
]
```

Exemple 3 : Insertion de données

main.js

```
const pool = require("./database/db");
const format = require('pg-format');  // npm install pg-format if not already installed

no usages
async function insertMultipleLogs() {
    const client = await pool.connect();
    try {
        // Prepare your data
        const dateTimeNow = new Date().toISOString();  // current date and time in ISO format
        const data = [
            [1, dateTimeNow, "Connexion réussie"],
            [2, dateTimeNow, "Connexion réussie"]
        ];
        // Use pg-format to safely format the SQL query for multiple inserts
        const query = format('INSERT INTO JOURNAUX_UTILISATEURS (user_id, date_time, event) VALUES %L', data);
        // Execute the query
        await client.query(query);
        console.log('Multiple logs inserted successfully');
    } catch (error) {
        console.error('Error during multiple inserts:', error);
        throw error;
    } finally {
        client.release();
    }
}
```

```
/*
getUserByFullNameLike('Perr', 'Milvyne').then(users => {
    console.log('Found Users:', users);
}).catch(error => {
    console.error('Error:', error);
});
*/

insertMultipleLogs().then(() => {
    console.log('Insert operation completed.');
}).catch(error => {
    console.error('Insert operation failed:', error);
});
```

```
(base) josephazar@Josephs-MBP-2 postgres_loginmgt % node main.js
Multiple logs inserted successfully
Insert operation completed.
```

```
1  SELECT * FROM JOURNAUX_UTILISATEURS;
```

Data Output | Messages | Notifications

	id [PK] integer	user_id integer	date_time timestamp without time zone	event character varying (255)
4	4	1	2022-05-12 09:00:00	Connexion réussie
5	5	1	2022-06-15 11:00:00	Changement de mot de passe
6	6	1	2022-12-25 18:30:00	Connexion réussie
7	7	2	2022-03-22 14:20:00	Changement de mot de passe
8	8	2	2022-07-08 10:45:00	Tentative de connexion échouée
9	9	2	2022-08-30 16:00:00	Changement de mot de passe
10	10	2	2022-09-15 10:00:00	Connexion réussie
11	11	3	2022-11-05 20:30:00	Tentative de connexion échouée
2	12	1	2023-11-12 22:15:32.138	Connexion réussie
3	13	2	2023-11-12 22:15:32.138	Connexion réussie

Exemple 4: Mettre à jour la table GROUPE

```
1 usage
async function updateGroupToAdmin(groupId) {
    const client = await pool.connect();
    try {
        // Prepare your SQL query
        const query = 'UPDATE groupes SET groupe = $1 WHERE id = $2';

        // Execute the query
        await client.query(query, ['Admin', groupId]);
        console.log(`Group with id ${groupId} updated to 'Admin'`);
    } catch (error) {
        console.error('Error during update:', error);
        throw error;
    } finally {
        client.release();
    }
}

// Call the function to update the group with id 1
updateGroupToAdmin( groupId: 1).then(() => {
    console.log('Update operation completed.');
}).catch(error => {
    console.error('Update operation failed:', error);
});
```

```
(base) josephazar@Josephs-MBP-2 postgres_loginmgt % node main.js
Group with id 1 updated to 'Admin'
Update operation completed.
```

Exemple 5 : Transaction

Ceci est un exemple pour montrer comment fonctionnent les transactions. Nous allons insérer un groupe puis le supprimer en une seule transaction. Référence : https://node-postgres.com/features/transactions

main.js

```
async function createAndDeleteGroup() {
    const client = await pool.connect();
    try {
        // Start the transaction
        await client.query('BEGIN');
        // Step 1: Insert a new group
        const insertGroupText = 'INSERT INTO GROUPES(groupe) VALUES($1) RETURNING id';
        const insertGroupResult = await client.query(insertGroupText, ['Prestataire']);
        const newGroupId = insertGroupResult.rows[0].id;
        console.log(`New group created with id: ${newGroupId}`);
        // Step 2: Delete the newly created group
        const deleteGroupText = 'DELETE FROM GROUPES WHERE id = $1';
        await client.query(deleteGroupText, [newGroupId]);
        console.log(`Newly created group with id ${newGroupId} has been deleted`);
        // Commit the transaction
        await client.query('COMMIT');
        console.log('Transaction completed successfully');
    } catch (error) {
        // Roll back the transaction in case of an error
        await client.query('ROLLBACK');
        console.error('Transaction failed:', error);
        throw error;
    } finally {
        // Release the client back to the pool
        client.release();
    }
}
// Call the function
createAndDeleteGroup().then(() => {
    console.log('Function execution completed.');
}).catch(error => {
    console.error('Function execution failed:', error);
});
```

```
(base) josephazar@Josephs-MBP-2 postgres_loginmgt % node main.js
New group created with id: 5
Newly created group with id 5 has been deleted
Transaction completed successfully
Function execution completed.
```

Exemple 6 : Utilisation de curseurs pour réduire la charge mémoire

Référence : https://node-postgres.com/apis/cursor

Un curseur peut être utilisé pour lire efficacement de grands ensembles de résultats sans charger l'intégralité de l'ensemble de résultats en mémoire à l'avance. Il est utile de simuler une lecture de données de style « streaming ».

Installer pg-cursor:

npm install pg-cursor

Chargez tous les utilisateurs et groupes et imprimez uniquement les utilisateurs avec le groupe "Visiteurs".

```
const pool = require("./database/db");
const format = require('pg-format');
const Cursor = require('pg-cursor');


async function fetchAndPrintUsers() {
   const client = await pool.connect();
   try {
       const queryText = 'SELECT * FROM utilisateurs FULL OUTER JOIN groupes
ON groupes.id = utilisateurs.group_id';
       const cursor = client.query(new Cursor(queryText));
       const readNextBatch = () => {
           cursor.read(100, (err, rows) => {
               if (err) {
                   console.error('Error reading rows:', err);
                   return finish();
               }
               // Process the rows
               rows.forEach(row => {
                   if (row.groupe === 'Visiteurs') {
                       console.log(row);  // Print the user if groupe is
'Externe'
s initially created empty and will create new clients lazily as they are
needed. Every field of the config object is entirely optional. The config
passed to the pool is also passed to every client instance within the pool
when the pool creates that client.s initially created empty and will create
```

```
new clients lazily as they are needed. Every field of the config object is
entirely optional. The config passed to the pool is also passed to every
client instance within the pool when the pool creates that client.
}
                });
                // Check if there are more rows to read
                if (rows.length === 0) {
                    finish();
                } else {
                    readNextBatch();  // Read the next batch of rows
                }
            });
        };
        const finish = () => {
            cursor.close(() => {
                client.release();
                console.log('Processing completed.');
            });
        };
        readNextBatch();  // Start reading the first batch of rows
    } catch (error) {
        console.error('Error during query execution:', error);
        client.release();
    }
}

fetchAndPrintUsers().catch(error => {
    console.error('Failed to fetch and print users:', error);
});
```

```
(base) josephazar@Josephs-MBP-2 postgres_loginmgt % node main.js
{
  first_name: 'Karine',
  last_name: 'Deschinkel',
  date_created: 2023-11-08T08:27:29.481Z,
  group_id: 3,
  password: 'karine123',
  user_id: 4,
  id: 3,
  groupe: 'Visiteurs'
}
Processing completed.
```