INFORMATIQUE
Sciences
Université Paris Cité

Programmation Web
TP n° 5b : Services Web et Base de Données

Le but de cette seconde partie est de réimplémenter le service de dictionnaire de la partie précédente en untilisant, cette fois, une vraie base de données pour le stockage des mots.

1 Accès à la base

Commencez par créer deux copies des fichiers de la première partie : vous devrez réorganiser et modifier ces copies.

1. La création d'un pool de connection à la base, la connection d'un client au pool, etc, vous sont expliquées dans le chapitre `3_applications.pdf` du cours disponible sur moodle.

   Un pool ne peut se connecter qu'à une base de données déjà existante : il faudra donc créer la base du dictionnaire, par exemple à l'aide de `psql` (c.f. le fichier `README.TXT` sur moodle).

   Cette base, *e.g.* `dictionary`, peut-être minimaliste : une seule table `words` à une seule colonne `word` de type chaîne à au plus 255 caractères. Après la commande `psql -U` *votre_nom* `-W postgres`, il suffira d'entrer les commandes :

   ```
   CREATE DATABASE dictionary;
   \c dictionary
   CREATE TABLE words (word VARCHAR(255));
   \q
   ```

2. Toutes les méthodes du dictionnaire devront être définies comme `async`. Dans le corps de ces méthodes, toutes les demandes d'opérations asynchrones (connection au pool, requête à la base via un client, etc.) devront être précédées de `await` pour garantir la séquentialité des instructions. Il faudra aussi ajouter au dictionnaire une méthode `.connect()` permettant de connecter un client à la base :

```
function Dico_db () {
  const pg = require('pg');
  const pool = ...;
  // client global, initialise par connect()
  let client;

  this.connect = async function() {
    client = await pool.connect();
    // a completer, par exemple en commencant par vider
    // la table words par un "TRUNCATE ... ".
  }

  this.search = async function() {
    // ... async client.query(...), etc.
  }

  // ...
}
```

3. Le code du serveur devra être transféré dans une unique fonction `async`, appelée en toute fin de son fichier. Les invocations des méthodes du dictionnaire devront être précédées de `async`, toujours pour garantir la séquentalité du code.

```
const dico = require('./Dico_db');
async function run() {
  // connection
  await dico.connect();
  // remplissage
  await dico.insert("...");
  // ...
  // initialisation et lancement du serveur
  // ...
}
run();
```

4. Enfin, il y a deux manières d'invoquer les méthodes du dictionnaire dans les gestionnaires de routes :

   (a) En construisant des chaînes de promesses dans ces gestionnaires, définis comme des fonctions ordinaires :

```
dico.search(word).then((data) => {
  /* traitement des donnees recues */
});
```

   (b) En définissant les gestionnaires comme des fonctions `async` et en utilisant `await` :

```
let data = await dico.search(word);
// traitement des donnees recues
```

   Choisissez la forme qui vous convient le mieux.