Présentation BDD

La base de données Postgres, version 10.6, est déployée sur Heroku au même titre que la partie back effectuée grâce à un server node.js et la partie front (Angular 7).

Par soucis de simplicité, j'ai opté pour une base de données légèrement différente que celle proposée. A la place de la table récursive « place », j'ai intégré une table « arrondissements » issue d'un shapefile téléchargé sur OpenDataParis[1]. Cette table a été créée via un outil PostGIS présent dans le logiciel Qgis. Elle sera filtré par code insee pour n'afficher qu'un seul arrondissement.

La table « entreprises » remonte les champs nécessaires et les renomme pour une meilleure mise en forme. Nous filtrerons cette table également avec le code insee.

Un script node.js d'initialisation de la base de données a été effectué (init_data.js). Il récupère les 500 premières entreprises par arrondissements pour les poster dans la base.

Puisque l'api remonte des données erronées (certaines données sont localisées en dehors de l'arrondissement escompté), un script de correction géographique a été effectué (cleanDb.sql).

Justification choix librairies *template* graphique

Conformément à la demande, la librairie graphique, compatible avec Angular 7, nécessite de répondre à plusieurs besoins :

- Tableau responsive avec pagination, filtres, et classements chronologiques,
- Boutons
- Bouton de type « toggle » pour la légende cartographique
- Tooltip pour la navigation
- Toolbar pour le Header
- Input autocomplete pour la sélection d'un arrondissement

Plusieurs librairies répondent aux besoins, notamment Ng-Bootsrap et Prime-ng, mais mon choix s'est arrêté sur ***Angular Material***. Tout d'abord, car les fonctionnalités du tableau sont fournies nativement (inutile d'ajouter de fonctionnalités *custom* pour répondre au besoin). De plus, l'aspect graphique me semble plus abouti que les autres librairies consultées (style moderne, couleur, animations). Enfin, cet exercice représente une occasion de parfaire mon apprentissage d'une libraire dont la maintenabilité est garantie par Google.

Justification choix back end

Le back-end a été développé en node.js. Il est composé de deux fichiers situés à la racine du projet Angular. Un fichier « server.js » effectue le routing alors qu'un autre fichier « api.js » gère les requêtes SQL vers la base.

Ce choix se justifie d'abord par la contrainte d'un déploiement sur Heroku qui oblige la création d'un server pour distribuer les fichiers issue de la compilation. Une partie du fichier « server.js » était donc déjà créé.

De plus, ce choix garantie l'homogénéité de langage entre la partie manipulation des données, back et front. Ainsi, le travail de connexion pour la manipulation des données a donc servi pour le back.

[1] https://opendata.paris.fr/explore/dataset/arrondissements/information/

Enfin, c'est aussi la simplicité avec laquelle il est possible de créer un service de données avec Node.js qui a justifié ce choix.

Remarques

A des fins de démonstration, nous avons convenu d'une modification de l'énoncé pour afficher des *clusters* de points sur la carte. Je trouvais en effet, que cela répondait mieux au besoin qu'afficher pêle-mêle 200 points sur une carte.

Plutôt qu'un tableau, j'ai choisi un input avec auto complétion pour la sélection d'un arrondissement.

Contrairement à l'idée initiale, le seed / fixture minimaliste a été fait dans le back. Il utilise deux librairies JS (turf.js et chance.js) pour générer de la donnée exploitable.

Fautes de temps, seules quelques éléments ont fait l'objet de tests. J'ai toutefois essayé de faire des tests unitaires sur la plupart des composants (ng test). J'ai également réalisé un scenario de navigation entre les pages home, tableau et carte (ng e2e).

Evolutions pertinentes

Il s'agit ici que d'une démonstration et beaucoup d'autres fonctionnalités/optimisations peuvent être ajoutés.

Dans l'état actuel des choses, il est impossible de filtrer en même temps plusieurs colonnes avec des valeurs différentes. Cette fonctionnalité peut être ajoutée, par exemple, en ajoutant une liste de colonne sur lequel il est possible d'ajouter des filtres différents.

Il est aussi possible d'optimiser l'application pour éviter le rechargement de composants déjà chargé lors de la navigation. Par exemple un passage de la vue carte vers la vue historique. Il est notamment envisageable de développer une solution en Store Architecture[2].

Multiplier le nombre de filtres possible sur la cartographie, styliser la Popup, ou bien ajouter une vue hybride tableau / carto pour éviter les aller/retour sera également appréciable.

[2] https://blog.angular-university.io/angular-ngrx-store-and-effects-crash-course/