Rapport de projet Client Side Programming
Théo Fafet
Identifiant Github Théo-Fafet

Pour ce projet inter-matières nous avons réalisé une application permettant de visualiser des données relatives à la propagation du Covid-19.

Cette application consiste en un front-end développé sur ReactJS, et d'un back-end utilisant ExpressJS et une base de données MongoDB.

Ayant eu plusieurs problèmes avec React sur mon système, que je détaillerais un peu plus loin, je me suis plutôt concentré sur le back-end tout en aidant les autres membres de l'équipe sur les différents problèmes qu'ils ont pu rencontrer. J'ai notamment aidé Anthony pour résoudre des problèmes liés à la carte interactive.

Le choix de ExpressJS pour la gestion des routes à été motivé par plusieurs facteurs. D'une part, il s'agit d'une bibliothèque que toute l'équipe avait déjà eu l'occasion d'utiliser auparavant. D'autre part, mettre en place une API Rest basique est relativement simple et rapide. Notre projet étant d'une taille assez petite nous n'avions pas besoin de routes très diverses.

Le choix de MongoDB a quant-à lui été imposé par le cours Server Side. Nous avons choisi d'utiliser de ne pas l'héberger nous-même mais d'utiliser un serveur hébergé par mongodb.net. Cela nous a permis d'éviter des complications de mise en place supplémentaires. Il nous a donc simplement fallu créer le serveur, les différentes tables SQL et uploader les données.

L'API de MongoDB permet de récupérer les données qui nous intéressent en utilisant des contraintes sur les noms des champs ou les valeurs. J'ai donc décidé de créer une route unique permettant de récupérer toutes les données importantes en utilisant un système de filtrage. La route accepte donc une liste de paramètres indiquant des clés et valeurs dans l'URL. Ces données sont alors simplement transformées par une fonction du serveur afin de correspondre au format compatible avec MongoDB. De cette manière l'ajout de données dans la base, ou l'ajout de fonctionnalités côté client ne nécessitent en principe aucun développement supplémentaire côté serveur. La fonction en question se trouve en annexe.

Travaillant exclusivement sur Linux, je me suis confronté à un problème totalement bloquant en travaillant sur la partie client. L'implémentation actuelle de react-script permet de surveiller les modifications sur les différents fichiers du projet, et donc de mettre à jour le serveur de développement en temps réel. Cependant l'implémentation ne fait aucune distinction entre les fichiers du projet et ceux contenus dans node-modules. Il arrive donc régulièrement que le nombre de fichiers à surveiller dépasse de manière écrasante la limite du noyau Linux. C'est exactement le problème auquel j'ai dû faire face. Après de nombreuses heures de recherche et tentatives infructueuses, une solution potentielle que j'ai trouvé est d'augmenter la limite. Au début, cette solution a fonctionné, alors que je commençais à travailler sur le mode sombre de l'application. Cependant il arriva très vite en ajoutant une nouvelle bibliothèque JavaScript que

la limite de fichiers soit à nouveau dépassée. Lorsque les cartes et graphiques interactifs ont été ajoutés je n'ai jamais réussi à relancer le serveur de développement.

Afin de participer au développement du client, j'ai suivi de près les avancées des autres membres de l'équipe, regardé de la documentation, suggéré des solutions aux problèmes rencontrés. J'ai par exemple aidé Anthony sur la carte. Il a par exemple été bloqué un certain temps sur le chargement des données relatives aux régions. L'un des problèmes rencontrés a été une incohérence dans les identifications des régions. Les données qui nous ont été fournies au début du projet utilisant des entiers comme identifiants, alors que la bibliothèque un acronyme à trois lettres. Une solution temporaire que nous avons trouvé est d'ajouter une route dans le serveur pour effectuer la traduction. Une première implémentation de la solution était une fonction sur le serveur qui renvoyait la traduction d'un identifiant, mais cette solution générait de nombreuses requêtes identiques entre le client et le serveur. Une meilleure solution, toujours pas optimale cela-dit, est de retourner directement un tableau de traduction au client, qui n'a alors besoin de réaliser qu'une requête. La solution idéale serait d' harmoniser les systèmes en modifiant nos données.

Annexes

Commande pour augmenter la limite de watch :

```
echo fs.notify.max_user_watches=32768 | sudo tee -a /etc/sysctl.conf
```

Code de la fonction de filtres pour MongoDB :

```
const compareKeywords = [
    { key: ">=", value: "$gte" },
    { key: "<=", value: "$lte" },
    { key: ">", value: "$gt" },
    { key: "<", value: "$lt" },
    { key: "!", value: "$ne" }
];

const compareFilter = (key, value) => {
    for (let i = 0; i < compareKeywords.length; i++) {
        const keyword = compareKeywords[i];
        if (value.startsWith(keyword.key)) {
            const filter = {}
            filter[keyword.value] = parseInt(value.replace(keyword.key, ""))
            return filter
        }
    }
    return value
}

const filtering = (filters) => {
    const properties = Object.getOwnPropertyNames(filters)
    const results = {};
    for (let i = 0; i < properties.length; i++) {
        const key = properties[i]
        const value = filters[key];
        if (value == parseInt(value)) {
            results[key] = parseInt(value)
        } else {
            if (compareKeywords.map(kw => kw.key).find(k => value.startsWith(k)) != undefined) {
                results[key] = compareFilter(key, value)
            } else {
                results[key] = value
            }
        }
    }
    return results
}
```

Code pour envoyer la table de traduction des identifiants des régions ::

```
const { Router } = require('express');
const fs = require("fs");

const regions = JSON.parse(fs.readFileSync("./region2020.csv", "utf-8"));

const router = new Router();

router.get('/', (req, res) => {
    res.status(200).json(regions);
});

module.exports = router;
```