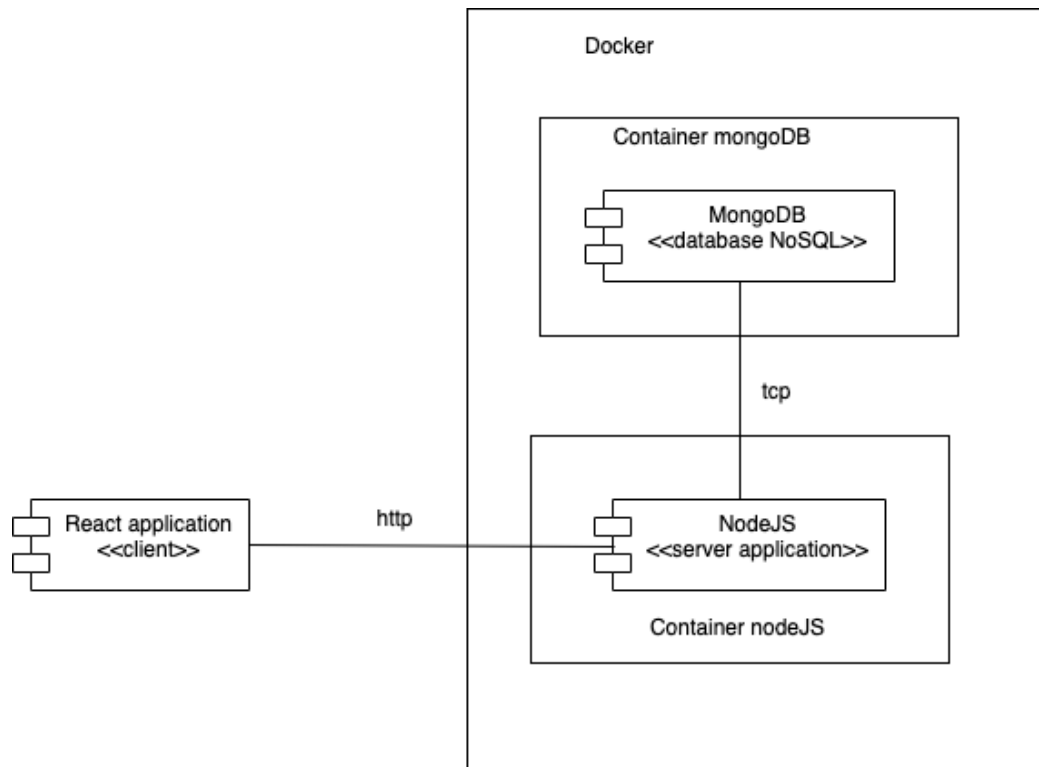


RAPPORT CLIENT-SIDE

Projet React - SI5

28/02/2021

Virgile-Giovanni FANTAUZZI / Identifiant Github : Kirabium



MES TACHES EFFECTUEES :

Back-end :

- *Création du repo : 10 min*
- *Mise en place du back-end : 4h*
 - *Pull les images dockers*
 - *Création du dockerfile*
 - *Création du docker-compose.yml*
 - *Mise en place du network/bridge*
 - *Mise en place d'un mock et des routes CRUD*
 - *Mise en place de Express, de l'architecture controller, ...*
- *Feed de la DB : 8h (voir problèmes rencontrés)*
 - *Transformation des CSV en JSON*
 - *Hébergement des JSON*
 - *Création des modèles et interfaces avec mongoose*
 - *CRUD sur les 6 modèles*
 - *Route pour init la DB*
- *Des fixes pour améliorer les temps de réponses/fonctionnement de certaines routes (pour la pagination notamment) : 30 min*

Front-end :

- *Mise en place de la liste d'incidences : 4h*
 - *Mise en place du composant et de la table*
 - *Mise en place du routing*
 - *Mise en place de la pagination côté front*
 - *Mise en place de la communication avec le back-end*
 - *Mise en place du filtre par day/week, par région/département/ pays et par date*
- *Mise en en place de la liste des données hospitalières : 3h*
 - *Mise en place du composant et de la table*
 - *Mise en place du routing*
 - *Mise en place de la pagination côté front*
 - *Mise en place de la communication avec le back-end*
 - *Mise en place du filtre par sexe et par date*
- *Mise en place du formulaire de contact : 1h*
 - *Mise en place du composant et du formulaire*
 - *Mise en place du routing*
 - *Mise en place de l'envoi du mail*

UTILISATION DE GITHUB

On a utilisé la branche « dev » principalement pour le développement. Avec des pull request sur main de façon assez régulière. Lorsque je travaillais sur l'implémentation des listes et des tables, je travaillais sur des branches dédiées « develop-list » et « develop-list-filter » que j'ai ensuite merge sur dev et puis sur main.

SOLUTIONS CHOISIES

Pour le back-end j'ai choisi de déployer notre serveur et notre mongoDB sur des images docker afin de permettre une installation et une utilisation rapide pour tous le monde une fois que la configuration des dockerfile/docker-compose est bien faite.

Les alternatives auraient été soit :

- Installer mongoDB en local et d'exécuter le serveur en local avec un npm run dev, c'est plus long à mettre en place qu'un simple docker-compose build && docker-compose run et ça nécessite que tous les utilisateurs fassent la bonne configuration (qui est dépendante de l'OS de notre machine).
- Héberger nos solutions sur Heroku par exemple. C'est une solution tout à fait viable et qui permettrait d'éviter l'installation de docker. Mais nous n'avons pas eu le temps de le faire, nous avons préféré continuer l'implémentation de nos features.

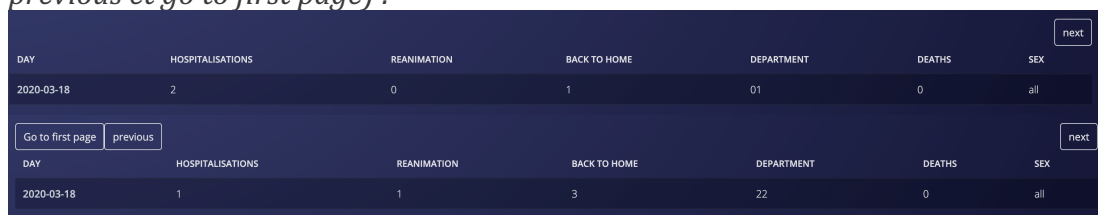
Afin de populate la mongodb, j'ai hébergé les fichiers json sur github, je les récupère ensuite avec une requête get pour les insérer dans la database via des routes init. J'ai choisi d'héberger les fichiers json car c'est la seule façon qui me permettait de compiler via docker (voir problèmes rencontrés pour plus de détails). Pour la route init, j'ai trouvé ça mieux de le faire via une route POST afin de choisir si oui ou non on charge les données dans la database lorsqu'on lance le serveur.

L'alternative aurait pu être de vérifier au lancement du serveur si la DB est déjà populate ou non, et si non, lancer le « remplissage ». Je ne l'ai pas choisi car je ne voulais pas être forcé de la populate

Pour le front-end :

- Mise en place de la liste des incidences :
 - o Mise en place de la pagination côté front

Pour la pagination j'ai choisi de mettre en place un système de boutons (next, previous et go to first page) :



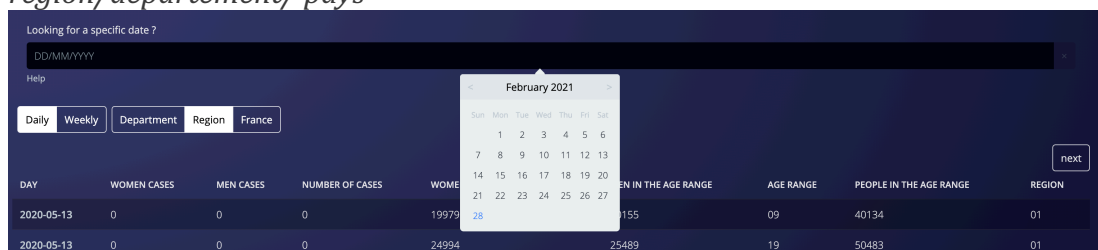
The screenshot shows a table with two rows of data. The first row has values: DAY: 2020-03-18, HOSPITALISATIONS: 2, REANIMATION: 0, BACK TO HOME: 1, DEPARTMENT: 01, DEATHS: 0, SEX: all. The second row has values: DAY: 2020-03-18, HOSPITALISATIONS: 1, REANIMATION: 1, BACK TO HOME: 3, DEPARTMENT: 22, DEATHS: 0, SEX: all. Navigation buttons include 'Go to first page', 'previous', 'next', and 'next'.

DAY	HOSPITALISATIONS	REANIMATION	BACK TO HOME	DEPARTMENT	DEATHS	SEX
2020-03-18	2	0	1	01	0	all
2020-03-18	1	1	3	22	0	all

Car c'est la façon la plus familière avec laquelle je navigue entre des pages. Une alternative aurait été d'afficher la liste de nombre des pages et de cliquer dessus pour naviguer.

- Mise en place du filtre par day/week, par région/département/ pays et par date

Le filtre par date me paraissait être le filtre indispensable pour exploiter un minimum les données de la DB, je l'ai implémenté avec un DatePicker, on peut écrire la date à la main et/ou la pick dans un petit calendrier et les données associées apparaissent. A noter que le filtre par date se cumule au filtre par jour/semaine, et région/département/ pays



The screenshot shows a date picker for February 2021. Below it is a table with filters: Daily, Weekly, Department, Region, France. The table has columns: DAY, WOMEN CASES, MEN CASES, NUMBER OF CASES, WOMEN IN THE AGE RANGE, AGE RANGE, PEOPLE IN THE AGE RANGE, REGION. The first row shows data for 2020-05-13 with values: 0, 0, 0, 19979, 155, 09, 40134, 01. The second row shows data for 2020-05-13 with values: 0, 0, 0, 24994, 25489, 19, 50483, 01.

DAY	WOMEN CASES	MEN CASES	NUMBER OF CASES	WOMEN IN THE AGE RANGE	AGE RANGE	PEOPLE IN THE AGE RANGE	REGION
2020-05-13	0	0	0	19979	155	40134	01
2020-05-13	0	0	0	24994	25489	50483	01

Une alternative aurait été de laisser seulement le choix de la date à l'écriture mais c'est selon moi, moins pratique. Mais cela aurait eu l'avantage de ne pas dépendre d'une librairie externe et donc d'alléger (un peu) le projet.

Pour les filtres par jour/semaine, et région/département/ pays, j'ai choisi d'implémenter des groupe buttons, car je trouve ça bien visuel et agréable à utiliser. Une alternative aurait été d'utiliser 2 dropdown button afin de choisir ses options.

- Mise en place de la liste des données hospitalières :
 - o Mise en place de la pagination côté front

Même chose que ci-dessus

- o Mise en place du filtre par sexe et par date

Même choses que ci-dessus, afin de rester cohérent j'ai gardé l'utilisation des button groups, l'alternative est la même que ci-dessus.

Par rapport aux filtres choisis l'alternative aurait été d'implémenter des filtres non pas sur le sexe, mais sur le département (ou les deux) par exemple. C'est une alternative tout à fait viable et cohérente, mais afin d'avoir le client le plus léger et rapide possible, pour chaque filtre/tri ou opération, nous passons par des routes back-end pour délocaliser les calculs. Nous n'avons pas eu le temps d'implémenter toutes les routes pour tous les types de filtres.

- Mise en place du formulaire de contact :

- Mise en place de l'envoi du mail

Pour le formulaire de contact, à l'envoi, la page redirige vers un link mail :to avec les données écrites en paramètre afin que l'utilisateur termine l'envoi sur son fournisseur de mail. J'ai trouvé cette solution similaire à celles que j'utilise sur mobile et elle me convient donc tout à fait.

Une alternative aurait été de passer par un service externe tel que Emailjs (je me suis même inscrit dessus) qui envoie directement le mail. Le problème était qu'il fallait forcément utiliser des templates et faire leur implémentation à eux (qui ne me convient pas, je veux faire le formulaire que je veux). Cela reste une option viable mais je ne l'ai pas choisi pour le produit final

DIFFICULTES RENCONTREES

Pour commencer, la première difficulté que j'ai rencontrée est que je n'ai jamais fait de React avant cette matière, je suis donc assez long dans la courbe d'apprentissage et j'ai dû me familiariser rapidement pour pouvoir produire quelque chose. (Merci les cours et la doc)

La deuxième difficulté est l'accumulation des projets, des rapports et de tas de choses à rendre dans plusieurs matières qui limitent le temps qu'on passe sur chacun des projets.

La plus grosse difficulté que j'ai rencontré dans le code est côté back-end, les csv transformés en json étaient beaucoup trop lourds. Impossible de les importer dans le projet, impossible de les inclure dans l'image docker (mémoire saturée). Ma solution a été de les ajouter à la racine du répo github, de les récupérer avec Axios pour ensuite populate la database via des routes init. Même avec cette solution, l'initialisation prend beaucoup de temps, et si la connexion est mauvaise ça peut provoquer un timeout

En conséquence du montant de données, il était beaucoup trop long de les récupérer côté front-end aussi, nous avons donc mis en place une pagination pour accéder aux données dans les tables.

En petite difficultés, j'ai tourné en rond un petit moment pour garder l'état des tri sélectionnés et des boutons highlightés lorsque je navigue entre les pages d'une table, après plusieurs essais j'ai réussi à me débrouiller avec le state du composant.

CODE

Code que j'aime :

```
render() {  
  return (  
    <div className={s.root}>  
      {this.displayFilters()}  
      {this.displayChangePage()}  
      {this.state.page && this.getTable()}  
    </div>  
  );  
}
```

Je trouve ce code élégant, je ne dis pas forcément que le code est bon, mais récupérer mes composants depuis une fonction et en fonction de l'état de mon composant principal me plaît. Cela évite les gros pavés des render habituels et propose un render plus compacte.

Code que j'aime moins :

```
displayChangePage = () => {  
  return <Row>  
    <Col className={s.prev}>  
      {this.state.filteredList &&  
        <Button outline color="secondary" onClick={async () => {  
          await this.setState( state: {filteredList: null});  
          this.getPage( uri: "http://localhost:2023/hospitalDay/1/" + this.state.sexe)  
        }}>Back</Button>  
      }  
      {this.isFirstPage() &&  
        <Button outline color="secondary"  
          onClick={() => this.getPage( uri: "http://localhost:2023/hospitalDay/1/" + this.state.sexe)}>Go  
          to first page</Button>  
      }  
      {this.isFirstPage() &&  
        <Button outline color="secondary"  
          onClick={() => this.getPage(this.state.page.prevPage)}>previous</Button>  
      }  
    </Col>  
    <Col className={s.next}>  
      {this.isLastPage() &&  
        <Button outline color="secondary" onClick={() => this.getPage(this.state.page.nextPage)}>next</Button>  
      }  
    </Col>  
  </Row>  
};
```

Il y a du code qui se répète un peu, je sais que c'est améliorable, je dois pouvoir wrap mon button dans un composant custom et jouer avec les props afin de spécifier les quelques différences de ceux-ci, mais je ne suis pas arrivé à l'articuler correctement (et rapidement). Si j'y passe plus de temps, je pourrai faire quelque chose de plus compact et maintenable via les composants et les props.

Merci de m'avoir lu