Identifiant Git Hub : Ayoub555555

Taches effectuées : voir les issues

- Afficher le Tableau des départements lors du clique sur un département front
- Faire un loader qui se lance à l'exécution de l'application qui charge toutes les données dans des objets pour éviter les temps de chargement back front
- Nav Bar front
- Router front
- Tableau back front
- Ecrire dans le wiki les problemes rencontres et les solutions apportées documentation
- Authentification basic back front
- Formulaire de contact front
- Tester les api avec post man et avoir une collection complète back
- Ecrire les différentes fonctions CRUD de la BDD back
- Finaliser la structure de données back

A propos du Git :

Nos branches :

-Main : production -Unreviewed-Works : branche de développement Nous avons opté pour une stratégie des plus standard et des plus efficaces. Nous avons choisis de créer une branche par fonctionnalité principale de notre projet. Au terme d'un travail potable et fonctionnel un merge (ou rebase) se fait sur la Branche Unreviewed-Woks. Une fois par semaine une pull request serait faîte depuis unreviewed-works vers main. le code reviewer se charge de revoir le code le linter et donner d'éventuelles améliorations possibles. Malheureusement seule une pull request fût faite car à chaque fin de semaine nous avons encore du code non potable dans unreviewed-works malgré des avancements significatifs. Lors d'un merge vers unreviewed-works chacun doit s'assurer d'une gestion des conflits qui ne laisse pas d'erreur qui font planter l'application. Nous aurions put faire une branche par composant mais nous aurions eu des soucis pour s'y retrouver dans les commits car parfois un composant a besoin d'autres. Nous aurions pût aussi faire une approche par page mais vu que dans nos pages

Solutions choisies

Afficher le Tableau des départements lors du clique sur un département front

C'est une partie de l'issue Nav Bar Front, elle s'est fait avec un history.push avec comme state le nom du département. ceci m'a permit de comprendre le code de mes camarades. une alternative aurait été de mettre une liste cliquable des departements

Faire un loader qui se lance à l'exécution de l'application qui charge toutes les données dans des objets pour éviter les temps de chargement back front

Etant à l'initiative de cette idée j'ai pas eu le temps de l'implémenter j'ai donc géré la promise dans mon composant, j'ai ajouté un load spinner mais que j'ai laissé en commentaire. l'idée est de charger directement un objet et non pas lancer la promise dans le composant afin de rendre le render du composant plus propre. L'alternative aurait été de faire ce que je n'ai pas fait.

Nav Bar front

Au lieu d'une nav bar nous avons décider sur chacun des composant de mettre un bouton de retour afin de ne pas passer par la précédent du navigateur. j'ai fait un composant qui en gère deux et j'ai fait en sorte que l'authentification renvoi à la carte de la France afin de ne pas avoir de bouton de connexion qui ne servirait à rien dans notre cas. le bouton est générique mais pas vraiment car il n'est générique que pour les régions. Une alternative aurait été de mettre une nav tout en haut de l'application afin de pouvoir aller vers le composant qui nous intérressé.

Router front

A l'image de ce qu'on a vu en cours on a toutes les routes vers nos composants. une alternative aurait été d'utiliser une autre librairie que react-router-dom pour gérer cette partie.

Tableau

Cette fonctionnalité est la table que l'on doit afficher afin de l'étailler un peu j'ai ajouté une pagination et un sorting en plus de la recupération des données depuis la BDD. Décomposé en plusieurs composants (row, row data, tabheader et departementDataLoader). dont chacun fait une partie du travail ce qui a causé pas mal de problèmes pour passer les hooks entre les différents composants. Pour la pagination et le sorting je le fait directement dans le composant principal departement-data-tab pour éviter de perdre beaucoup de temps pour passer les

state entre les composants. Un important composant de css a été implémenté en SASS afin de faire correspondre avec le thème dans departement-data-tab.css En alternative j'aurais pût essayer de trouver un paquet de npm qui permet d'implémenter une table comme celle là. Un composant générique ce qui m'aurait fait gagner du temps mais qui n'aurait pas permit de démontrer mes compétences

Ecrire dans le wiki les problemes rencontres et les solutions apportées documentation

Ce travail est le rapport

Authentification basic back front

Une authentification avec le useLocalStorage afin de permettre l'accès à l'application (**login : user, password: user**) Une alternative aurait été de faire un boutton de connexion avec une mattrice de droits sur les composants

Formulaire de contact front

Un composant qui n'est pas terminé car il ne gere pas les erreurs et qui a pris beaucoup de temps de devellopement car j'ai dut creer une route sécurisée sur la back afin d'avoir le mot de passe de ma boite mail afin d'envoyer le mail qui s'ecrit dans le field commentaire. Le css était très dur à gérer car je n'arrivais pas à agrandir les deux derniers inputs. de plus, j'ai eu du mal à cacher les erreurs et ne les afficher que si le bouton submit a été cliqué (chose que j'ai fait facilement dans l'authentification), j'aurais pût avec plus de temps le faire. Je n'ai jamais pu tester si le mail s'envoi mais les erreurs sur le formulaire sont bien gérée. Une alternative aurait été de ne pas essayé d'envoyer le mail et de juste d'enregistrer le message quelquepart ou de faire une F.A.Q qui se garde un historique

Tester les api avec post man et avoir une collection complète back

Sans avoir réussit à la compléter à temps je vous met quelques unes : https://filebin.net/ci1l3cw8tuz9836d une alternative aurait été d'utiliser un logiciel qui depuis le code renvoi les api qu'il contient

Ecrire les différentes fonctions CRUD de la BDD back

Au début nous pensions que nous nous repartirions le travail en front/back mais suite aux directives nous nous sommes repartis le travail, chacun a créée la route dont il avait besoin. Pour ma part c'était dans le composant qui gère le document

Département, nous nous démarquons avec une architecture de travail très propre en séparant routes controller et service, fastidieux mais bien plus beau à voir. Une alternative aurait été une architecture type MVC avec des objets. la notre ressemble plus à du SOA chose que personnellement je ne connaissait pas.

Finaliser la structure de données back

Concernant le schema de la base de données nous avons eu pas mal de soucis car nous voulions que le schema "parent" connaisse l'"enfant". Ce qui n'aurait posé aucun probleme avec des inners join mais dans ce genre de BDD (type document) une architecture en clés étrangères permet à l'enfant de connaitre le parent et ainsi de mieux correspondre aux données CSV qui eux même sont des matrices (plus proche d'une table que d'un document). Une alternative aurait été de passer par un lookup (particularité de mangoose) afin d'accéder à une autre collection.

Difficultés rencontrées

le setState dans useEffect ce qui provoque des render infinis

la solution est de bien lire la doc, les éviter au maximum ou en écrire plusieurs ou encore de créer une fonction constructor (cf departement-data-tab) l'impact est sur le temps de développement et surtout sur le moral car les tutoriels présents donnent des bouts de codes qui ne fonctionnent pas.

syntaxe pas très intuitive parfois. la solution est de vraiment passer beaucoup de temps à repérer toute particularité de la doc

un problème innhérant à React parcequ'il utilise tout les types de données imaginables et couplé à l'anonymicité d'à peu près n'importe quoi donne un code qui enchaînes les [}) sans fin... l'impact est qu'il faut compter et revoir chacun de ces symboles à chaque fois ce qui est peu intuitif. La solution est d'utiliser une extension qui les met en lumière.

hiérarchie des routes express

avec le app.use qui parfois ne nous apparaissent pas ou qui sont fait par d'autres développeur on a parfois beaucoup de mal à retrouver la syntaxe des routes ce qui donne lieu à des longues périodes de recherche dans le code. je n'ai pas trouver de solution à se problème appart de lire le code ligne par ligne.

gestion des erreurs liée aux routes react

Parfois pour les erreurs (surtout trop de re render) il est très dur de trouver la/les lignes en cause car le stackTrace ne les donne pas ce qui conduit à des tests fastidieux sur chaque partie susceptible de provoquer l'erreur. la solution est de mettre des consoles logs et des try catch à tout va. une solution serait d'avoir un meilleur débogueur avec des espions pour mieux voir la valeur des variables.

Typage

vu que les variables ne sont pas typées parfois dans les boucles on a du mal à identifier le type d'une variable notamment la variable qui fait office de compteur. l'impact est de devoir aller revérifier la valeur du compteur. une solution aurait été un meilleur recensement des types des variable comme tool tip dans l'IDE plutôt que any à chaque fois.

Temps de développement / tâche

Jour inconnu :

-revision de l'archi + es lint sur tout le code + explication de comment utiliser les query/param : 30 min -15 min pour régler un soucis d'evenement on click sur région -15 min localisation bug -15 min revérification BDD car données manquantes après CSV -15 min association des région à leur numero dans la BDD -20 min etat avancement + petits problèmes de code -2 heures à essayer de faire marcher setState dans useState alors que je faisais appel à une fonction asynchrone pour recuperer mes données depuis le back. j'ai réglé en mettant directement le code qui get dans le composant sans qu'il soit asynchrone avec un useState blocant ([] en second parametre);

- 1 heure pour essayer de faire marcher un premier composant de pagination car on ne peut voir toute la page d'un coup : bug d'affichage le haut n'apparait pas.
- 1h pour changer de composant de pagination et utiliser react boostrap pagination. 1 heure avec esl lint pour le back.

1/02/2021 :

- 3 heures choisir les données bien toutes les visualiser et les comprendre(g), avoir un objectif (g), modeliser les shémas des données (g)

8/02/2021 :

- 4 heures sur installer react(p) + faire l'archi du back(g) et la modifier pour la rendre meilleure (meilleure requete avec foreign keys de jour dans region et de region dans dep)v (p) + commencer les controllers du back(jour + region) qui recup toutes données avec condition(jour/jour+numregion/jour+libregion) (p)+ schémas du back(g) (bien les expliquer et pourquoi definir des objets marche pas => definir foreign key en type primitifs) + issues

11/02/2021 :

- 2 heures sur modelisation front(g) + issues(p) + methodologie des branches(p)+ methodologie de travail par fonctionnalités(g) + lerna(p) + directives/discussion à l'équipe concernant les branches et qui fait quoi(g) + revision archi systeme de fichiers (p)

15/02/2021 :

- 1 heure pour repartition des taches. 5 min Router . 1 heure 30 pour ecrire le debut des différents composants de table

18/02/2021 :

- 1 h 30 pour regrouper toutes les branches dans la branche unreviewed-changes + régler les petits bugs + tests puis merge dans main. 1/2h d'avancement sur le composant tableau pour qu'il soit affiché.
- 1h pour faire le theming du composant en fonction du theme de l'application

20/02/2021 :

-6 heures dont 3 heures pour faire le departement back (une erreur de syntaxe m'a pris 1h pour être trouvée). 1 heure de lecture doc MangoDB et architecture microservices avec Node, 1h d'avancement sur le composant Tableau au front. 1 heure de code review

23/02/2021 :

- 15 minute pour faire un loader séparé du composant + push tout les commit en veillant à les separer +rebase
- 15 minutes pour les composants principaux du contact form

25/02/2021 :

- 1h30 heures pour faire le composant contact avec la libraire des profs. 1 h de discussion optimisation du composant Map.
- 30 min de discution sur la démo
- 2h30 pour faire un semblant d'authentification et une sécurité sur le mot de passe de l'envoi d'email
- 30 min pour régler les problemes liés à ce nouveau code
- 3 heures pour comprendre les useEffect et me rendre compte qu'il fallait faire une sorte de constructeur dans departement-data-tab + écriture du code
- 3 heures pour finir le pagination (coder le on Click et passer la bonne valeur)
- 1h30 pour commencer les principaux composants de l'authentification
- 1 heure pour régler certains bug d'affichage du composant contact + un peu de css

27/02/2021 :

- 1 heure pour gérer des problèmes de commit
- 2 h pour rendre le tableau triable
- 1/2 h sur le composant authentification pour la form
- 1/4 h pour rediriger du composant qui affiche les départements en carte vers le tableau
- 1 h pour rendre le composant d'authentification fonctionnel

28/02/2021 :

- 3 heures de travail collectif sur les commit
- 1 heures pour mettre authentification à la première page et mettre des boutons de redirection
- jusqu'à la fin de la journées des petites modifications

environ 52 heures de travail sur le projet.

Commenter une fonction ou un composant (max 100 lignes) que vous avez écrit et qui vous semble élégant ou optimal

Dans departement-data-tab.js const useConstructor = (callBack = () => { }) => { const hasBeenCalled = useRef(false); if(hasBeenCalled.current) return; callBack(); hasBeenCalled.current=true; }

```
export const DepartementDataTab = (props) => {
```

```
  let location = useLocation();
  const [currentPart, setCurrentPart] = useState(0);
  var [parts,setParts] = useState(new Map());
  const [data,setData] = useState({});
  const [sortType,setSortType] = useState(new Map());
  const [forcerender,SetForceRender] = useState(true);

useConstructor(() => {
    let data;
    DepartementDataLoader(location.state).then(res => {
        data=res.data.data_tab;

        var nextNum=0;
        var newParts = new Map();
        var partCount=0;

        for(let i=1;i<=data.length;i++){
            if(i%10===0){
                newParts.set(partCount,data.slice(nextNum,i))
                partCount++;
                nextNum=i;
            }
        }
        setData(data);
        setParts(newParts);
        let temp= new Map();
        for(let i =0 ; i<=6 ; i++){
            temp.set(i,'asc');
        }
        setSortType(temp);
    });
});
```
. . .suite du code

Ce bout de code est pour moi le plus optimal. Souvent avec les hooks il est très difficile de simuler le comportement d'un constructeur (comme avec les classes) sinon il faut passer par plusieurs useEffect qui départagent le code et qui sont souvent redondants ou encore initialiser dans le use State ce qui est vraiment moche et surtout s'il faut faire des calculs sur les données. Sans ce composant mon code aurait été bien plus dur à fournir (environ 3 heures pour essayer de le faire marcher avec les useEffect sans résultat probant. De plus cela permet des renders inutiles qui sont souvent monnaie courrante dans les hooks. Il démontre aussi assez l'utilisation de plusieurs types différents de hooks.

Commenter une fonction ou un composant (max 100 lignes) que vous avez écrit et qui mériterait une optimisation ou amélioration

Dans departement-data-tab.js :

const handleSorting = (changingKey,index,newSortBy) =>{ let oldParts=parts; setSortType(newSortBy); let tableau = parts.get(currentPart); tableau.sort(compareForSpecificColumn(changingKey, newSortBy.get(index))); oldParts.set(currentPart, tableau); setParts(oldParts); if(forcerender === true){ SetForceRender(false); } else{ SetForceRender(true); } }

Cette fonction passe à un composant enfant :

Dans le composant enfant : tab Header

export const TabHeader = (props) => { const [sortBy,setSortBy] = useState(new Map());

```
useEffect(() => {
    setSortBy(props.sortBy);
}, [props.sortBy])

const handleOnClick = (value, type) => {
    let twoValueArray = [];
    let newMap = sortBy;
    let index = 0;
    let changingKey='';

    if(value === 'Jour'){
        index=0;
        changingKey='jour';
    }

    if(value === 'Cas positifs'){
        index=1;
        changingKey='nbtest_positif';
    }

    if(value === 'Cas au total'){
        index=2;
        changingKey='nbtest';
    }

    if(value === 'Taux de positivité'){
        index=3;
        changingKey='tx_pos';
    }

    if(value === "Taux d'incidence"){
        index=4;
        changingKey="tx_inc";
    }

    if(value === 'Capacité analytique'){
        index=5;
        changingKey='tx_an';
    }

    if(newMap.get(index)==='asc')
    newMap.set(index,'desc');
    else newMap.set(index,'asc');
```

```
    setSortBy(newMap);
    props.onSort(changingKey, index, newMap);
}
if(sortBy !== undefined){
    return(
        <tr>
            <th onClick={() => handleOnClick('Jour', sortBy.get(0))}>Jour
{sortBy.get(0) ==='asc' ? <ArrowDropUpIcon /> : <ArrowDropDownIcon />}</th>
            <th onClick={() => handleOnClick('Cas positifs', sortBy.get(1))}>Cas
positifs {sortBy.get(1) ==='asc' ? <ArrowDropUpIcon /> : <ArrowDropDownIcon
/>}</th>
            <th onClick={() => handleOnClick('Cas au total', sortBy.get(2))}>Cas
au total {sortBy.get(2) ==='asc' ? <ArrowDropUpIcon /> : <ArrowDropDownIcon
/>}</th>
            <th onClick={() => handleOnClick('Taux de positivité',
sortBy.get(3))}>Taux de positivité {sortBy.get(4) ==='asc' ? <ArrowDropUpIcon /> :
<ArrowDropDownIcon />}</th>
            <th onClick={() => handleOnClick("Taux d'incidence",
sortBy.get(4))}>Taux d'incidence {sortBy.get(5) ==='asc' ? <ArrowDropUpIcon /> :
<ArrowDropDownIcon />}</th>
            <th onClick={() => handleOnClick('Capacité analytique',
sortBy.get(5))}>Capacité analytique {sortBy.get(6) ==='asc' ? <ArrowDropUpIcon />
: <ArrowDropDownIcon />}</th>
        </tr>
    )
}
else return (<div>is Loading</div>)
```

Ce qui est bien avec ce code c'est qu'il démontre la complexité de React pour faire remonter des états la fonction aurait pût être améliorée avec un objet de type handleOnClick et le return avec une map sur les values du dictionnaire. une utilisation des use Mémo aurait put être envisagée si le composant prenait du temps à charger.