Rapport individuel
Projets React

SI5

Anis KHALILI

2020-2021

Identifiant GitHub

Mon identifiant GitHub : **Anis-KHALILI**

Tâches effectuées

Coté FrontEnd(React) j'étais engagé à faire les tâches suivantes :

- Développer le composant **List** .
- Développer le composant **Authentification (BONUS)**
  - → Validation des champs (email , password)
  - → Vérification des champs pour s'authentifier.
- Participation à l'intégration et développement du composant racine **RegionsStats** qui permet de gérer les données du backend et l'envoyer au composant **Map** et **List** pour les afficher.
- Faire les changements nécessaires coté **App.js** pour bien intégrer les composant.
  - → Gérer le routage entre les différents composants.
  - → Ajouter un **AdminGuardedRoute** pour vérifier si l'utilisateur est authentifié ou pas pour accéder au contenu du site web.

Stratégie employée pour la gestion des versions avec Git

Pour bien gérer les modifications sur le projet par les différents membres de l'équipe on a choisi cette stratégie :

- Au début du projet on a mis toutes les tâches à réaliser et les bonus dans les To Do du projet sur le git sous forme de tickets.
- Après, chaque membre choisi un ou plusieurs tickets :
  - → Convertir le ticket en issue en précisant la description et mettre le bon label.
- Créer une branche pour travailler sur cette issue.
- Quand l'issue est terminée :
  - → On crée une Pull Request.
  - → On assigne un autre membre en review.
  - → On écrit dans le Channel Slack "Pull Request prête pour review par le membre assigné : lien vers la Pull Request".

- Le membre assigné fait le review du code.
  - → Si tout est bon, je merge la branche dans main et je ferme l'issue, sinon le membre assigné me demande de corriger des conflits.
  - → Je corrige les conflits et je redemande une review au membre assigné etc. jusqu'à ce que le merge ne soit effectuée.

Rq: pendant le développement, on fait merge main dans la branche où je travaille pour garantir que j'ai toujours la version à jours du projet.

Mes branches coté frontEnd :

- → 25#-authentification.
- → 23#-list-component-frontEnd.

Solutions choisies

Pour afficher les données pour le **NewCases** et le **mentalHealth** (anxiété, dépression, pbsommeil) sur la carte et sur une liste, on a pensé à utiliser un composant racine **RegionsStats** qui fait appel au serveur du backend pour collecter les données. Après il envoit ces données après un traitement (mettre ces données dans le bon modèle, prendre en compte les données manquantes, choisir les données selon un filtre de date par jours...) sous forme de deux objets **newCasesNow** et **mentalHealthNow.**

Le composant fils **Map** ou **List** recevoir ces objets pour faire l'affichage selon le jour choisi.

Cette façon de centraliser la récupération des données et les traiter en fonction du filtre est efficace surtout qu'on a 2 composants Map et list qui affichent les mêmes données mais de façon différente. Aussi, cette alternative permet d'intégrer n'importe quelle autre composant qui utilise les mêmes données.

Pour l'authentification j'étais besoin de faire la validation des champs pour le mail pour qu'il soit du format xxx@xxx.xx . Aussi le password ne doit pas être vide. En plus pour ces champs-là doivent être vérifier :

Email : admin@polytech.com password : admin

J'ai choisi la bibliothèque de **Formik** comme alternative pour faire ces contraintes.

Cette bibliothèque nous permet d'ajouter des formulaires qui gèrent automatiquement les erreurs de validation des champs avec "**validationSchema",** aussi la vérification avec **"handleSubmit".**

Difficultés rencontrées

Au début, l'affichage du tableau du composant **List** est basé sur la récupération de l'objet **newCasesNow** , et pour chaque région récupérée de newCasesNow j'ajoute une ligne dans le tableau.

Avec cette méthode j'ai constaté que parfois il y a des régions qui se répètent (l'Object **newCasesNow** parfois retourne des donnés redondantes) en plus les noms des régions ne sont pas indiqués dans l'Api utilisée pour les **mentalhealth** .

Donc pour corriger cette redondance et ajouter les noms des régions, j'ai créé un fichier Régions qui comporte les 13 régions de la France avec leurs noms et leurs id. Et pour afficher les lignes, j'ai fait parcourir les régions et pour chaque région je cherche les données correspondantes dans **newCasesNow**, par exemple :

{newCasesNow.find(r => region.id + "" === r.regionId)?.newCases || "?"}

Avec cette manière je garanti un bon affichage sans redondance de données et un affichage des noms des régions.

Autre difficulté c'était au niveau de l'authentification, en fait après avoir mis en place le composant authentification j'ai remarqué qu'on peut accéder aux autres pages du site comme le **/home** ou **région/id** sans restriction (sans vérification si l'utilisateur est login vraiment ou pas). Donc cette faille détruit le principe d'authentification.

Pour résoudre ce problème j'ai ajouté une méthode dans le **App.js "AdminGuardedRoute"** qui retourne un composant : soit redirection vers la page d'authentification soit vers le composant demandé.

Cette vérification est gérée par un attribut appelé **"user"** dans le **cookies**. Donc si l'authentification est réussite le cookies "user" change de valeur en **"login"**. Si non il reste **undefined**.

De cette manière la fonction **AdminGuardedRoute** vérifie la valeur du cookies et renvoi le bon composant.

Temps de développement / tâche

Chaque jeudi et le weekend, j'essaye d'avancer le maximum sur les différentes tâches donc c'est un peu difficile de mesurer exactement combien de temps j'ai passé par tâche mais à peu près :

→ Le composant List + une partie du composant RegionsStats :

La première semaine.

→ Le composant authentification (comptant les modifications et la gestion des routages des différents composants aussi l'ajout de l'adminGuard avec la vérification de login avec le cookies):

La deuxième semaine.

Rq: Jai pas compté le temps passé sur le backend : les 2 semaine d'avant.

Code :

```
REGIONS.map(region =>
<tr key={region.id}
    className={+userRegionId === +region.id ? "user-region-list" : ""}
    onClick={() => history.push("/graph/" + region.id)}
    style={{cursor: "pointer"}}>
    <td>
        {region.name} {+userRegionId === +region.id && "(votre région)"}
    </td>
    <td>
        {newCasesNow.find(r => region.id + "" === r.regionId)?.newCases || "?"}
    </td>
    <td>
        {mentalHealthNow.find(r => region.id + "" === r.regionId)?.anxiete || "?"}
    </td>
    <td>
        {mentalHealthNow.find(r => region.id + "" === r.regionId)?.depression || "?"}
    </td>
    <td>
        {mentalHealthNow.find(r => region.id + "" === r.regionId)?.pbsommeil || "?"}
    </td>
```

Le composant List, je pense que c'est une partie qui a été bien développée car ici j'ai essayé de garantir que les données à afficher sont bien placées et il n'y a pas de redondance d'affichage pour une telle région. Pour chaque région on cherche les données correspondantes.

```
    handleSubmit: (props) => {

        if(props.email === "admin@polytech.com" && props.password === "admin" ) {
            alert('SUCCESS Login');
            window.location.href = "http://localhost:3000/home";
            Cookies.set("user","login");
        } else {
            alert('Email ou mot de passe erroné. Veuillez réessayer. ')
        }},
})(Authentication);
```

Dans le composant authentification j'ai utilisé un seul email et password mais c'était mieux de gérer des vrais utilisateurs. Par exemple je stocke les utilisateurs dans une BD et je fais la vérification des utilisateurs coté backend (le backend envoi au front si cet utilisateur est autorisé ou pas).