Application de gestion de projet

Tout d'abord, ce que j'ai fait jusqu'au 28 février, c'est retravaillé le backend en mettant en place une nouvelle base de données. J'y ai ajouté des routes avec le CRUD pour chaque routeur ainsi que la conception des modèles correspondants.

Ensuite, j'ai poursuivi sur l'objectif principal qui est le frontend en utilisant la bibliothèque "react-hook-form", qui permet de gérer des formulaires dans React de manière performante, ainsi que l'utilisation de Material-UI qui est une bibliothèque d'interface utilisateur afin de créer une application web de gestion de projet.

L'arborescence du frontend se compose d'une partie "component" avec les formulaires et une autre des "pages" contenant des "data-grid" où chacun correspond à une interface de "types/index.ts" qui est le fichier d'export d'interface. J'ai utilisé un framework de css se nommant tailwind que j'utilise dans mon index.css et qui est importé dans "main.tsx" pour le rendu visuel de l'application. Dans les "pages" j'ai créé un fichier "Home.tsx" dans lequel je crée un menu de navigation pour les routes de chaque modèle que j'appelle dans la fonction "HomeScreen()", lors du clique sur l'un des modèles, j'appelle la fonction que j'ai associé à celle-ci et qui a été importé dans "Home.tsx" au préalable. Par la suite j'ai créé les différents fichiers correspondant aux modèles de données auxquelles est associé un formulaire comme dit précédemment donc on va s'intéresser aux différents fichiers de "component" et "pages".

"BackUsers.tsx" affiche les données des utilisateurs présent dans la base de données et "UserForm.tsx" ne sera pas utilisé pour cette démonstration car on utilise les données présentes dans le backend donc lister les users selon le modèle sera la seule chose montré dans cette partie.

- Ligne 10: déclare une constante d'état local avec "useState"
- Ligne 12 à 32: utilisation de "useEffect" pour monter au lancement du projet la récupération des données des utilisateurs avec la fonction "fetchUsers()" et qui va être envoyé dans la constante d'état.
- Ligne 34 à 37 : déclare une constante "columns" de tableau qui récupère des champs du modèle de "fetchUsers()" pouvant être utilisé dans "datagrid".
- ligne 80 à 89 : utilisation d'un "datagrid" récupérant les lignes des utilisateurs grâce à la constante d'état et récupère les colonnes défini dans "columns".

"BackProjects.tsx" affiche les données des projets présent dans la base de données ainsi que supprimer des projets existant et "ProjectForm.tsx" peut créer et modifier ces projets (lorsqu'on veut remplir le formulaire pour les rôles actuellement je met l'id des utilisateurs pour que ca fonctionne et je n'ai pas réussi à implémenter l'appartenance à d'autre types de modèle de données).

BackProjects.tsx :

- ligne 7 à 22 : création d'une fonction delete en récupérant l'id du modèle et le supprimant par la suite
  ligne 24 à 60: même chose que pour "BackUsers.tsx" à part qu'il faut faire avec le modèle de projects
- ligne 61 à 85 : créer 2 boutons éditer et supprimer dans une colonne appelant la fonction pour modifier avec un formulaire et l'autre celle pour supprimer
- ligne 86 à 91: gère l'état de la fenêtre du formulaire lors de la validation
- ligne 92 à 95: gère l'état de la fenêtre du formulaire lors de l'annulation
- ligne 101 à 106 : bouton pour ouvrir la fenêtre du formulaire
- ligne 108 à 117 : même chose que pour "BackUsers.tsx" à part qu'il faut faire avec projects
- ligne 118 à 123 : récupération de l'interface de "ProjectForm.tsx" lors de l'appelle de la fonction de celui-ci

ProjectForm.tsx:

- ligne 5 à 10: création de l'interface du formulaire
- ligne 12 à 19: création de l'interface du modèle
- ligne 22 à 33: défini les champs par requis lors du remplissage du formulaire
- ligne 35: mise en place de la fonction pour le formulaire
  ligne 36 à 45: défini le formulaire initial par défaut
- ligne 46 à 55: reset la valeur de "initialdata" pour pouvoir définir si l'état c'est modifier ou ajouter et ne pas reprendre la dernière valeur entrée dans le formulaire
  ligne 57 à 97 : en fonction de l'état de "modified" il va chercher la route de modification ou de création
- ligne 100 à 155: affichage du formulaire permettant de créer ou modifier un projet

"BackStories.tsx" affiche les données des projets présent dans la base de données ainsi que supprimer des projets existant et "StoryForm.tsx" peut créer et modifier ces stories. "BackStories.tsx" fait la même chose que "BackUsers.tsx" et "StoryForm.tsx" pareil que "ProjectForm.tsx" mais avec un code plus simplifié et factorisé.

"BackTasks.tsx" et "BackSprints.tsx" affichent les données des projets présents dans la base de données. "TaskForm.tsx" et "SprintForm.tsx" ne permettent actuellement pas la création ou la modification, car le champ "status" est un booléen alors que j'ai besoin d'y intégrer plus de deux états.

Pour conclure, ce projet fonctionne correctement dans l'ensemble, bien que je ne l'aie pas encore entièrement finalisé et que je n'aie pas pu établir les liens entre chaque modèle.