Marquer comme terminé

TP 2

1. récupérer l'archive contenant le code de démarrage et l'intégrer au code du TP précédent, en modifiant les fichiers qui le nécessitent
2. analyser le code résultant
   - le composant `App` fait appel au composant `TodoList`
   - le composant `TodoList` import des données de Todo, contenues dans le fichier `Helpers/todoData.js`.

```
const todoData = [
    {
        id: 1,
        content: "faire ses devoirs",
        done: true
    },
    {
        id: 2,
        content: "ranger sa chambre",
        done: false
    },
    {
        id: 3,
        content: "essuyer la vaisselle",
        done: false
    }
];

export default todoData
```

3. le composant `TodoList` utilise ces données pour construire une liste plate constituée de composants `TodoItem` pour chaque item de la liste\ :

```
        <FlatList
            style={{ paddingLeft: 10 }}
            data={todoData}
            renderItem={({item}) => <TodoItem item={item} />} />
```

4. Une `Flatlist` a besoin de données et d'une propriété `renderItem`, qui indique quelle fonction sera affichée pour rendre chaque item. Notez bien la syntaxe utilisée dans les paramètres\ : `({item})`, ce qui permet dans `TodoItem` d'obtenir l'objet désiré dans `props.item`, et non `(item)`, ce qui obligerait à utiliser `props.item.item`.
5. Normalement, une `Flatlist` a besoin d'un attribut indiquant comment calculer les clés des items, par exemple `keyExtractor={(item, index) => index}` ou `keyExtractor={(item) => item.id}. Dans le cas présent, c'est inutile car nos item ont un champ`id`.
6. le composant `TodoItem` est celui qui a été développé au TP précédent.
7. démarrez l'application\ : `npm start`
8. ajoutez un texte dans le composant `TodoList` permettant d'afficher le nombre de `TodoItem` réalisés. Pour cela, il faut un compteur comme vu en cours et passer au composant fils les fonctions nécessaire à sa modification. Pour initialiser le compteur, on peut utiliser `todoData.filter((item)=>item.done).length`.
9. Faites en sorte qu'un appui sur l'icône de poubelle détruise l'item correspondant.

Attention\ : la fonction de modification d'état est *asynchrone*. Vous ne pouvez donc pas enchaîner deux modifications d'état avec la deuxième utilisant l'état modifié par la première. Le code ci-dessous ne fonctionnera donc pas\ :

```
  const deleteTodo = (id) => {
    setTodos(todos.filter(item => item.id != id))
    setCount(todos.filter(item => item.done).length)
  }
```

Sinon, ici nous allons simplement calculer la nouvelle liste des todos dans une variable temporaire, qui servira à appeler les deux changements d'état:

```
  const deleteTodo = (id) => {
    const newTodos = todos.filter(item => item.id != id)
    setTodos(newTodos)
    setCount(newTodos.filter(item => item.done).length)
  }
```

?

11. ajoutez un `TextInput` et un bouton `new` pour ajouter un item à la liste. Le TextInput se paramètre comme suit\ :

```
        <TextInput
          onChangeText={setNewTodoText}
          placeholder='saisir ici un nouvel item'
          onSubmitEditing={addNewTodo}
          value={newTodoText}
    />
```

L'attribut `onSubmitEditing` indique la fonction à appeler quand on valide l'input à l'aide de la touche *Entrée*. Lorsqu'on valide la saisie, il faut réinitialiser la valeur du `TextInput`, c'est pour cette raison que sa valeur est gérée par une variable d'état et que l'attribut `onChangeText` modifie l'état.

Pour que la `Flatlist` reste cohérente, les identifiants des items doivent rester différents. Pour cela, le nouvel item créé pourra avoir comme identifiant le maximum des identifiants existants plus 1. La fonction `Math.max(...tableau)` retourne le maximum d'un tableau et pour construire le tableau des identifiants on utilise la fonction `map`, qui applique sa fonction paramètre à chaque élément d'un tableau\ :

```
Math.max(...todos.map(item => item.id)) + 1
```

Il faudra passer à `setTodos` un tableau constitué des anciens `todos` et d'un nouveau *todo*, ce qui peut se faire à l'aide de la syntaxe suivante (`...todos` signifie *le contenu du tableau `todos`*)\ :

```
setTodos([...todos, { id: monId, content: monContenu, done: false }])
```

- ajoutez des boutons pour filtrer l'affichage\ : n'afficher que les *todos* en cours, non résolus, ou *tout afficher*.

Modifié le: lundi 9 septembre 2024, 11:25

◄ CM 2

Choisir un élément

Aller à...

Démarrage TP 2 ►

?