Documentation du Projet Trellow

Introduction

Ce document vise à fournir une vue d'ensemble détaillée du projet Trellow, une application développée pour faciliter la gestion de tâches et de workspaces. Ce guide couvrira la structure de l'application, en commençant par `app.js`, le layout global, la sélection de workspaces, et enfin, une exploration des différents composants présents dans le dossier `components`.

Structure de l'Application

app.js

`app.js` sert de point d'entrée pour l'application. Il initialise l'application et configure le routage global. Voici les étapes clés de son fonctionnement :

1. **Initialisation** : Importation des modules nécessaires et configuration de l'environnement de l'application.
2. **Configuration du Routage** : Définition des routes principales de l'application et association de chaque route à son composant respectif.
3. **Rendu** : Rendu du composant racine qui encapsule toute l'application.

Global Layout

Le layout global définit la structure visuelle de base de l'application. Il inclut les éléments suivants :

- **Barre de Navigation** : Permet la navigation entre les différentes sections de l'application.
- **Conteneur Principal** : Zone où le contenu spécifique à chaque route est affiché.
- **Footer** : Contient des informations générales ou des liens utiles.

Sélection de Workspaces

La sélection de workspaces permet à l'utilisateur de choisir parmi les différents espaces de travail disponibles. Le processus est le suivant :

1. **Affichage des Workspaces** : L'utilisateur est présenté avec une liste de workspaces disponibles.
2. **Sélection** : L'utilisateur sélectionne un workspace, ce qui entraîne le chargement des tâches et des informations spécifiques à ce workspace.
3. **Interaction** : L'utilisateur peut créer, modifier ou supprimer des tâches au sein du workspace sélectionné.

Composants dans le Dossier `components`

Le dossier `components` contient les éléments réutilisables et spécifiques de l'application. Voici quelques composants clés :

TaskItem.js

`TaskItem.js` est responsable de l'affichage d'une tâche individuelle. Il gère l'interaction de l'utilisateur avec une tâche spécifique, comme l'expansion pour voir plus de détails ou l'activation de certaines actions (par exemple, marquer comme terminée, éditer ou supprimer).

```
import React from "react";
import { View, Text, TouchableOpacity } from "react-native";

const TaskItem = ({ item, onEdit, onDelete }) => {
  return (
    <TouchableOpacity onPress={() => console.log("Task pressed")}>
      <View style={styles.taskItem}>
        <Text>{item.title}</Text>
        <TouchableOpacity onPress={onEdit}>
          <Text>Edit</Text>
        </TouchableOpacity>
        <TouchableOpacity onPress={onDelete}>
          <Text>Delete</Text>
        </TouchableOpacity>
      </View>
    </TouchableOpacity>
  );
};

const styles = {
  taskItem: {
    margin: 10,
    padding: 20,
    borderRadius: 10,
    backgroundColor: "white",
    // Ajoutez d'autres styles comme nécessaire
  },
};

export default TaskItem;
```

Sidebar.js

`Sidebar.js` fournit une navigation latérale pour accéder rapidement à différentes sections de l'application. Il peut inclure des liens vers les workspaces, les paramètres de l'application, ou d'autres fonctionnalités importantes.

```
import React from "react";
import { View, TouchableOpacity, Text } from "react-native";

const Sidebar = ({ onNavigate }) => {
  return (
    <View style={styles.sidebar}>
      <TouchableOpacity onPress={() => onNavigate("Home")}>
        <Text>Home</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={() => onNavigate("Settings")}>
        <Text>Settings</Text>
      </TouchableOpacity>
      // Ajoutez d'autres liens de navigation comme nécessaire
    </View>
  );
};

const styles = {
  sidebar: {
    // Styles pour la sidebar
  },
};

export default Sidebar;
```

TaskModule.js

`TaskModule.js` est un composant plus complexe qui pourrait gérer la logique d'affichage et d'interaction avec une liste de tâches. Il peut inclure des fonctionnalités pour ajouter une nouvelle tâche, filtrer les tâches existantes, ou trier les tâches selon différents critères.

```
import React, { useState } from "react";
import { View, Button, TextInput } from "react-native";
import TaskItem from "./TaskItem";

const TaskModule = () => {
  const [tasks, setTasks] = useState([]);

  const addTask = (task) => {
    setTasks([...tasks, task]);
  };

  return (
    <View>
      <TextInput
        placeholder="Add a new task"
        onSubmitEditing={(event) => addTask(event.nativeEvent.text)}
      />
      {tasks.map((task, index) => (
        <TaskItem key={index} item={task} />
```

```
        ))}
      </View>
    );
  };

  export default TaskModule;
```

Workspaces.js

`Workspaces.js` pourrait être un composant qui permet aux utilisateurs de créer, sélectionner et gérer différents espaces de travail. Chaque espace de travail pourrait avoir ses propres tâches, membres et paramètres.

```
  import React, { useState } from "react";
  import { View, Button, Text } from "react-native";

  const Workspaces = ({ workspaces, onSelectWorkspace }) => {
    return (
      <View>
        {workspaces.map((workspace) => (
          <Button
            key={workspace.id}
            title={workspace.name}
            onPress={() => onSelectWorkspace(workspace.id)}
          />
        ))}
      </View>
    );
  };

  export default Workspaces;
```

Conclusion

Cette documentation offre un aperçu de la structure et des fonctionnalités clés du projet Trellow. Pour une compréhension plus approfondie, il est recommandé de consulter le code source et les commentaires associés à chaque composant.