Projet Guidé : Création d'une Application React de Gestion des Tâches

Dans ce projet guidé, nous allons développer une **application de gestion des tâches** en React. Ce projet vous permettra de comprendre les concepts clés de React tels que les composants, les props, l'état (state), et les hooks (notamment `useState` et `useEffect`). Nous aborderons également les bonnes pratiques pour structurer votre code.

Objectifs du Projet

1. **Comprendre les bases de React :** création de composants, gestion des props et du state.
2. **Utiliser des hooks :** `useState` pour gérer l'état local et `useEffect` pour des effets secondaires simples.
3. **Appliquer des styles :** intégrer du CSS pour rendre l'application plus attrayante.
4. **Mettre en place un système CRUD basique :** Ajouter, Lire, Mettre à jour et Supprimer des tâches.

Aperçu de l'Application

Notre application permettra de :

1. Afficher une liste de tâches.
2. Ajouter une nouvelle tâche.
3. Marquer une tâche comme terminée.
4. Supprimer une tâche.

Étape 1 : Configuration du Projet

Prérequis

- Avoir **Node.js** installé sur votre machine.
- Installer **npm** ou **yarn** pour gérer les dépendances.

Initialisation du Projet

1. Créez un nouveau projet React :

   Accédez au dossier :

   ```
   npx create-react-app gestion-taches
   ```

   ```
   cd gestion-taches
   ```

2. Lancez l'application :

   ```
   npm start
   ```

Vous devriez voir l'application de base React dans votre navigateur.

Étape 2 : Structure du Projet

Organisez votre projet comme suit :

```
src/
|-- components/
|   |-- TaskList.js
|   |-- TaskItem.js
|   |-- AddTask.js
|-- App.js
|-- App.css
```

Étape 3 : Création des Composants

3.1. `TaskItem.js`

Ce composant représentera une tâche individuelle.

```
import React from 'react';

function TaskItem({ task, toggleComplete, deleteTask }) {
  return (
    <div className="task-item">
      <input
```

```
        type="checkbox"
        checked={task.completed}
        onChange={() => toggleComplete(task.id)}
      />
      <span style={{ textDecoration: task.completed ? 'line
-through' : 'none' }}>
        {task.text}
      </span>
      <button onClick={() => deleteTask(task.id)}>Supprimer
</button>
    </div>
  );
}

export default TaskItem;
```

3.2. `TaskList.js`

Ce composant affichera la liste des tâches.

```
import React from 'react';
import TaskItem from './TaskItem';

function TaskList({ tasks, toggleComplete, deleteTask }) {
  return (
    <div>
      {tasks.map((task) => (
        <TaskItem
          key={task.id}
          task={task}
          toggleComplete={toggleComplete}
          deleteTask={deleteTask}
        />
      ))}
    </div>
  );
}
```

```
export default TaskList;
```

3.3. `AddTask.js`

Ce composant gérera l'ajout de nouvelles tâches.

```
import React, { useState } from 'react';

function AddTask({ addTask }) {
  const [taskText, setTaskText] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (taskText.trim()) {
      addTask(taskText);
      setTaskText('');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Nouvelle tâche"
        value={taskText}
        onChange={(e) => setTaskText(e.target.value)}
      />
      <button type="submit">Ajouter</button>
    </form>
  );
}

export default AddTask;
```

Étape 4 : Assemblage dans `App.js`

Voici le code de votre composant principal :

```
import React, { useState } from 'react';
import TaskList from './components/TaskList';
import AddTask from './components/AddTask';
import './App.css';

function App() {
  const [tasks, setTasks] = useState([]);

  const addTask = (text) => {
    setTasks([...tasks, { id: Date.now(), text, completed:
false }]);
  };

  const toggleComplete = (id) => {
    setTasks(
      tasks.map((task) =>
        task.id === id ? { ...task, completed: !task.comple
ted } : task
      )
    );
  };

  const deleteTask = (id) => {
    setTasks(tasks.filter((task) => task.id !== id));
  };

  return (
    <div className="App">
      <h1>Gestion des Tâches</h1>
      <AddTask addTask={addTask} />
      <TaskList
        tasks={tasks}
        toggleComplete={toggleComplete}
        deleteTask={deleteTask}
      />
    </div>
  );
}
```

```
export default App;
```

Étape 5 : Ajout de Styles

Ajoutez les styles dans `App.css` :

```
.App {
  text-align: center;
  margin: 20px;
}

.task-item {
  display: flex;
  align-items: center;
  justify-content: space-between;
  margin: 10px 0;
}

.task-item span {
  flex-grow: 1;
  margin-left: 10px;
}

button {
  margin-left: 10px;
  padding: 5px 10px;
  border: none;
  background-color: #ff6b6b;
  color: white;
  cursor: pointer;
  border-radius: 5px;
}

button:hover {
  background-color: #ff4c4c;
}
```

Résultat

En lançant votre application, vous pourrez :

- Ajouter des tâches.
- Les marquer comme terminées.
- Les supprimer.

Étape 6 : Prochaines Améliorations

1. **Stockage Local :** Utiliser `localStorage` pour persister les données.
2. **Filtrage :** Ajouter des filtres pour afficher toutes les tâches, uniquement les tâches terminées ou celles en cours.
3. **Animations :** Ajouter des transitions pour rendre l'interface plus dynamique.

Étape 6 : Prochaines Améliorations en Détail

6.1. Stockage Local avec `localStorage`

L'objectif est de persister les tâches entre les rechargements de la page en utilisant `localStorage`.

Modifications dans `App.js`

Ajoutez l'utilisation du hook `useEffect` pour lire et sauvegarder les tâches dans le stockage local.

```
jsx
Copy code
import React, { useState, useEffect } from 'react';
import TaskList from './components/TaskList';
import AddTask from './components/AddTask';
import './App.css';

function App() {
```

```
  const [tasks, setTasks] = useState(() => {
    const savedTasks = localStorage.getItem('tasks');
    return savedTasks ? JSON.parse(savedTasks) : [];
  });

  useEffect(() => {
    localStorage.setItem('tasks', JSON.stringify(tasks));
  }, [tasks]);

  const addTask = (text) => {
    setTasks([...tasks, { id: Date.now(), text, completed:
false }]);
  };

  const toggleComplete = (id) => {
    setTasks(
      tasks.map((task) =>
        task.id === id ? { ...task, completed: !task.comple
ted } : task
      )
    );
  };

  const deleteTask = (id) => {
    setTasks(tasks.filter((task) => task.id !== id));
  };

  return (
    <div className="App">
      <h1>Gestion des Tâches</h1>
      <AddTask addTask={addTask} />
      <TaskList
        tasks={tasks}
        toggleComplete={toggleComplete}
        deleteTask={deleteTask}
      />
    </div>
  );
```

```
}

export default App;
```

Explication :

- Le `useState` initialise les tâches à partir du `localStorage` si elles existent.
- Le `useEffect` surveille les changements dans `tasks` et les sauvegarde dans `localStorage`.

6.2. Ajout des Filtres

Ajoutons des filtres pour afficher :

1. Toutes les tâches.
2. Uniquement les tâches terminées.
3. Uniquement les tâches en cours.

Création du Composant `Filter.js`

Ajoutez un nouveau composant pour gérer les filtres.

```
jsx
Copy code
import React from 'react';

function Filter({ filter, setFilter }) {
  return (
    <div className="filter">
      <button
        className={filter === 'all' ? 'active' : ''}
        onClick={() => setFilter('all')}
      >
        Toutes
      </button>
      <button
        className={filter === 'completed' ? 'active' : ''}
        onClick={() => setFilter('completed')}
      >
```

```
        Terminées
      </button>
      <button
        className={filter === 'active' ? 'active' : ''}
        onClick={() => setFilter('active')}
      >
        En cours
      </button>
    </div>
  );
}

export default Filter;
```

Ajoutez les styles pour les boutons actifs dans `App.css` :

```
css
Copy code
.filter button {
  margin: 5px;
  padding: 5px 15px;
  border: none;
  cursor: pointer;
  border-radius: 5px;
}

.filter button.active {
  background-color: #4caf50;
  color: white;
}

.filter button:not(.active):hover {
  background-color: #ddd;
}
```

Intégration dans `App.js`

Ajoutez le filtre dans l'état global et utilisez-le pour afficher les tâches.

```
jsx
Copy code
import React, { useState, useEffect } from 'react';
import TaskList from './components/TaskList';
import AddTask from './components/AddTask';
import Filter from './components/Filter';
import './App.css';

function App() {
  const [tasks, setTasks] = useState(() => {
    const savedTasks = localStorage.getItem('tasks');
    return savedTasks ? JSON.parse(savedTasks) : [];
  });
  const [filter, setFilter] = useState('all');

  useEffect(() => {
    localStorage.setItem('tasks', JSON.stringify(tasks));
  }, [tasks]);

  const addTask = (text) => {
    setTasks([...tasks, { id: Date.now(), text, completed:
false }]);
  };

  const toggleComplete = (id) => {
    setTasks(
      tasks.map((task) =>
        task.id === id ? { ...task, completed: !task.comple
ted } : task
      )
    );
  };

  const deleteTask = (id) => {
    setTasks(tasks.filter((task) => task.id !== id));
  };
```

```
  const filteredTasks = tasks.filter((task) => {
    if (filter === 'completed') return task.completed;
    if (filter === 'active') return !task.completed;
    return true; // 'all'
  });

  return (
    <div className="App">
      <h1>Gestion des Tâches</h1>
      <AddTask addTask={addTask} />
      <Filter filter={filter} setFilter={setFilter} />
      <TaskList
        tasks={filteredTasks}
        toggleComplete={toggleComplete}
        deleteTask={deleteTask}
      />
    </div>
  );
}

export default App;
```

Explication :

- Le composant `Filter` permet de changer le filtre actif.
- Les tâches affichées sont filtrées selon l'état `filter`.

6.3. Ajout d'Animations

Utilisons la bibliothèque **React Transition Group** pour animer l'ajout et la suppression des tâches.

Installation

Installez la bibliothèque avec :

```
bash
Copy code
```

```
npm install react-transition-group
```

Mise à Jour de `TaskList.js`

Modifiez le composant pour inclure des animations.

```
jsx
Copy code
import React from 'react';
import { CSSTransition, TransitionGroup } from 'react-trans
ition-group';
import TaskItem from './TaskItem';
import './TaskList.css';

function TaskList({ tasks, toggleComplete, deleteTask }) {
  return (
    <TransitionGroup>
      {tasks.map((task) => (
        <CSSTransition key={task.id} timeout={300} classNam
es="task">
          <TaskItem
            task={task}
            toggleComplete={toggleComplete}
            deleteTask={deleteTask}
          />
        </CSSTransition>
      ))}
    </TransitionGroup>
  );
}

export default TaskList;
```

Ajout des Styles d'Animation

Créez un fichier `TaskList.css` pour les animations.

```
css
Copy code
.task-enter {
  opacity: 0;
  transform: translateY(-20px);
}
.task-enter-active {
  opacity: 1;
  transform: translateY(0);
  transition: opacity 300ms, transform 300ms;
}
.task-exit {
  opacity: 1;
  transform: translateY(0);
}
.task-exit-active {
  opacity: 0;
  transform: translateY(-20px);
  transition: opacity 300ms, transform 300ms;
}
```

Étape 7 : Résultat Final

Vous disposez maintenant :

1. **D'un système persistant de tâches avec `localStorage`.**
2. **De filtres pour gérer les tâches affichées.**
3. **D'animations fluides pour ajouter/supprimer des tâches.**