Filière : IAGI 2 **Matière : Front-END**
Semestre : **7** **Date : 24 / 04/ 2025**

Atelier 6 : React (Exercice 2)
Gestion d'une Liste de Tâches Avancée

Cet exercice vise à créer une application de gestion de tâches avancée avec les fonctionnalités suivantes :

- Ajout, suppression et filtrage des tâches.
- Marquer une tâche comme terminée.
- Persistance des données dans le localStorage.
- Utilisation de hooks (useState, useEffect, useContext) et création de composants réutilisables.

Etape 1 : Structure du projet

Créez un projet React nommé **"ENSAMTaskManager"** avec la structure suivante :

```
src/
├── components/
|   ├── Task.js          # Composant pour afficher une tâche
|   ├── TaskList.js      # Composant pour afficher la liste des tâches
|   ├── TaskForm.js      # Composant pour ajouter une tâche
|   ├── Filter.js        # Composant pour filtrer les tâches
|   └── Header.js        # Composant pour l'en-tête
├── context/
|   └── TaskContext.js   # Contexte pour gérer l'état global
├── App.js
├── App.css
└── index.js
```

Étape 2 : création des composants

Composant Task.js : Affiche une tâche avec un bouton pour la supprimer et une case à cocher pour la marquer comme terminée.

```
import React from 'react';
const Task = ({ task, onDelete, onToggle }) => {
  return (
    <div className={`task ${task.completed ? 'completed' : ''}`}>
      <input type="checkbox" checked={task.completed} onChange={() => onToggle(task.
id)} />
      <span>{task.text}</span>
      <button onClick={() => onDelete(task.id)}>Supprimer</button>
    </div>
  );
};
export default Task;
```

Composant TaskList.js : Affiche la liste des tâches en utilisant le composant Task.

```
import React from 'react';
import Task from './Task';
const TaskList = ({ tasks, onDelete, onToggle }) => {
  return (
    <div className="task-list">
      {tasks.map(task => (
        <Task key={task.id} task={task} onDelete={onDelete} onToggle={onToggle} />
      ))}
    </div>
  );
};
export default TaskList;
```

Composant TaskForm.js : Permet d'ajouter une nouvelle tâche.

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

const TaskForm = ({ onAdd }) => {
  const [text, setText] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!text.trim()) return;
    onAdd(text);
    setText('');
  };

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

export default TaskForm;
```

Composant Filter.js : Permet de filtrer les tâches (toutes, actives, terminées).

```
import React from 'react';
const Filter = ({ onFilter }) => {
  return (
    <div className="filter">
      <button onClick={() => onFilter('all')}>Toutes</button>
      <button onClick={() => onFilter('active')}>Actives</button>
      <button onClick={() => onFilter('completed')}>Terminées</button>
    </div>
  );
};
export default Filter;
```

Étape 3 : Création du Contexte (TaskContext.js)

Utiliser **useContext** pour gérer l'état global des tâches.

```
import React, { createContext, useState, useEffect } from 'react';
export const TaskContext = createContext();
export const TaskProvider = ({ children }) => {
  const [tasks, setTasks] = useState([]);
  const [filter, setFilter] = useState('all');

  useEffect(() => {
    const storedTasks = JSON.parse(localStorage.getItem('tasks')) || [];
    setTasks(storedTasks);
  }, []);
  useEffect(() => {
    localStorage.setItem('tasks', JSON.stringify(tasks));
  }, [tasks]);
  const addTask = (text) => {
    setTasks([...tasks, { id: Date.now(), text, completed: false }]);
  };
  const deleteTask = (id) => {
    setTasks(tasks.filter(task => task.id !== id));
  };
  const toggleTask = (id) => {
    setTasks(tasks.map(task =>
      task.id === id ? { ...task, completed: !task.completed } : task
    ));
  };
  const filteredTasks = tasks.filter(task => {
    if (filter === 'active') return !task.completed;
    if (filter === 'completed') return task.completed;
    return true;
  });

  return (
    <TaskContext.Provider value={{ tasks: filteredTasks, addTask, deleteTask, toggle
Task, setFilter }}>
      {children}
    </TaskContext.Provider>
  );
};
```

Étape 4 : Intégration dans App.js

```
import React from 'react';
import { TaskProvider } from './context/TaskContext';
import Header from './components/Header';
import TaskForm from './components/TaskForm';
import TaskList from './components/TaskList';
import Filter from './components/Filter';
import './App.css';
const App = () => {
return (
<TaskProvider>
        <div className="app"> <Header />
        <TaskForm />
                <Filter />
                <TaskList />
        </div>
</TaskProvider> );
};
export default App;
```

Consignes

- Utilisez **useState** pour gérer l'état local des composants.
- Utilisez **useEffect** pour persister les données dans le localStorage.
- Utilisez **useContext** pour partager l'état global entre les composants.
- Créez des composants réutilisables et bien structurés.

Travail à faire :

- Ajoutez une fonctionnalité pour éditer une tâche existante.
- Implémentez des animations pour les transitions entre les états.