

Atelier 03 TodoList

Objectif général :

Maîtriser les concepts fondamentaux de React (state, props, composants, événements, useEffect) et construire une application interactive complète avec persistance via localStorage.

Étape 1 : Préparer l'environnement React

Objectif pédagogique :

Savoir créer un projet React fonctionnel et structurer les fichiers.

Instructions :

1. Dans le dossier Ressources de cet atelier, copier le dossier **TodoListApp** dans le dossier **C:\ReactProjects**
2. Ouvrir le dossier **C:\ReactProjects\TodoListApp** avec votre éditeur de code
3. Démarrer docker et vérifier que les containers ne sont pas en exécution
4. Avec l'invite de commande, entrez dans le dossier **C:\ReactProjects\TodoListApp>** et lancer la commande suivante :

docker-compose up -d --build

5. Créez un projet React avec Vite ou Create React App.

1. Accéder au TodoListApp_container avec la commande suivante :

docker exec -it TodoListApp_container sh

2. **npm create vite@latest . --template**

Deux traits avant
template

```
/usr/src/app # npm create vite@latest . --template
> npx
> create-vite .
|
> Select a framework:
  React
|
> Select a variant:
  JavaScript
|
> Use rollup-vite (Experimental)?:
  No
|
  Install with npm and start now?
    • Yes / o No
```

3. Si le serveur Vite lancé, arrête-le (CTR + C)

4. Quitter le container

```
/usr/src/app # exit  
C:\ReactProjects\TodoListApp>
```

5. Modifier le fichier vite.config.js

```
import { defineConfig } from 'vite'  
import react from '@vitejs/plugin-react'  
  
// https://vitejs.dev/config/  
export default defineConfig({  
  plugins: [react()],  
  server: {  
    host: true,           // équivaut à --host  
    watch: {  
      usePolling: true, // <== active le mode "polling"  
      interval: 1000,   // <== vérifie les changements toutes les 1s  
    },  
  },  
})
```

6. Puis arrêter et démarrer le container, directement avec le docker ou bien exécuter ces deux commandes

```
docker-compose down  
docker-compose up -d --build
```

7. Entre dans le container

```
docker exec -it TodoListApp_container sh
```

8. Lancer cette commande :

```
npm run dev -- --host
```

9. Lancer l'application sur l'url <http://localhost:5173/>

6. Créez les dossiers **src/components**.

Étape 1 : Créer la structure des tâches

Objectif :

Comprendre le **state local** avec useState et gérer une **liste d'objets**.

Instructions :

1. Créez un composant **components/TodoList.jsx**.
2. Déclarez un state **tasks** initialisé avec un tableau de quelques tâches d'exemple.

```
{ id: 1, text: "Apprendre React", completed: false },
{ id: 2, text: "Faire les courses", completed: true }
```

- ✓ Affichez les tâches dans un en utilisant **map**.
 - ✓ Afficher la liste des tâches, si non afficher « pas des tâches »
 - ✓ Devant la tâche complète, ajouter le symbole (✓)
- Apprendre React
 - Faire les courses (✓)

Correction :

```
export default function TodoList() {
  const tasks = [
    { id: 1, text: "Apprendre React", completed: false },
    { id: 2, text: "Faire les courses", completed: true }

  ];
  return (
    (tasks.length > 0) ? (
      <ul>
        {tasks.map((task) => (
          <li key={task.id}>
            {task.text} {task.completed ? "(✓)" : ""}
          </li>
        ))}
      </ul>
    ) : (
      <p>No tasks available</p>
    )
  );
}
```

Etape 2 . Créez un composant principal App.jsx (si n'est pas créé).

1. Ce composant Ajouter un titre de H1 : « My ToDo list »
2. Appel le composant <TodoList />
3. Optionnel, Utiliser ce code

```
style={{ textDecoration: task.completed ? 'line-through' : 'none' }}
```

Correction / Exemple :

```
// src/App.jsx
import TodoList from './components/TodoList';

function App() {
  return (
    <div>
      <h1>My ToDo List</h1>
      <TodoList />
    </div>
  );
}
export default App;
```

Étape 3 : Ajouter une nouvelle tâche

Objectif:

Utiliser **state**, **événements** et **inputs contrôlés**.

Instructions :

1. Ajoutez un champ <input> et un bouton "Ajouter".
2. Créez un state **newTask** pour l'input.
3. Implémentez **handleAddTask** qui ajoute un nouvel objet tâche dans tasks.

Correction :

```
//src/components/TodoList.jsx
import { useState } from "react";

export default function TodoList() {
  const [tasks, setTask] = useState([
    { id: 1, text: "Apprendre React", completed: false },
    { id: 2, text: "Faire les courses", completed: true }
  ]);

  const [newTask, setNewTask] = useState("");
  function handleAddTask() {
    if (newTask.trim() === "") return;
    const newTaskObject = {
      id: tasks.length + 1,
      text: newTask,
      completed: false
    };
    setTask([...tasks, newTaskObject]);
    setNewTask("");
    console.log("Tâche ajoutée :", newTaskObject);
  }

  return (
    <>
      <input type="text"
        value={newTask}
        placeholder="Ajouter une tâche"
        name="textTask"
        onChange={e => setNewTask(e.target.value)} />
      <input type="button"
        onClick={handleAddTask}
        value="Ajouter" />

      {tasks.length > 0 ? (
        <ul>
          {tasks.map((task) => (
            <li key={task.id}>
              {task.text} {task.completed ? "(√)" : ""}
            </li>
          )))
        </ul>
      ) : (
        <p>No tasks available</p>
      )}
    </>
  );
}
```

Étape 4 : Marquer une tâche comme complétée

Objectif:

Comprendre la **modification d'état d'une liste** et **gestion d'événements dynamiques**.

Instructions :

1. Ajoutez un bouton ou **checkbox** à chaque tâche.
2. Implémentez **handleToggleCompleted** qui inverse completed de la tâche sélectionnée.

Correction / Exemple :

```
//src/components/TodoList.jsx
import { useState } from "react";

export default function TodoList() {
  const [tasks, setTask] = useState([
    { id: 1, text: "Apprendre React", completed: false },
    { id: 2, text: "Faire les courses", completed: true }
  ]);
  const [newTask, setNewTask] = useState("");
  function handleAddTask() {
    if (newTask.trim() === "") return;
    const newTaskObject = {
      id: tasks.length + 1,
      text: newTask,
      completed: false
    };
    setTask([...tasks, newTaskObject]);
    setNewTask("");
    console.log("Tâche ajoutée :", newTaskObject);
  }
  // Toggle task completion status
  function handleToggleTaskCompletion(taskId) {
    const updatedTasks = tasks.map(task => {
      if (task.id === taskId) {
        return { ...task, completed: !task.completed };
      }
      return task;
    });
    setTask(updatedTasks);
  }

  return (
    <>
      <input type="text"
        value={newTask}
        placeholder="Ajouter une tâche"
        name="textTask"
        onChange={e => setNewTask(e.target.value)} />
      <input type="button"
        onClick={handleAddTask}
        value="Ajouter" />

      {tasks.length > 0 ? (
        <ul>
          {tasks.map((task) => (
            <li key={task.id}>
              <input
                type="checkbox"
                checked={task.completed} />
            {task.text}
          </li>
        ))
      ) : (
        <p>Aucune tâche ajoutée.</p>
      )}
    </>
  );
}
```

```

        onChange={() => handleToggleTaskCompletion(task.id)} />
        {task.text} {task.completed ? "(√)" : ""}
      </li>
    )})
  ) : (
    <p>No tasks available</p>
  )
}
</>
);}

```

Étape 5 : Supprimer une tâche

Objectif :

Apprendre à **filtrer une liste** dans le state.

Instructions :

1. Ajoutez un bouton "Supprimer" pour chaque tâche.
2. Implémentez handleDeleteTask qui supprime la tâche correspondante.

Ma To-Do List



Correction / Exemple :

```

//src/components/TodoList.jsx
import React, { useState } from 'react';

function TodoList() {
  const [tasks, setTasks] = useState([
    { id: 1, text: "Apprendre React", completed: false },
    { id: 2, text: "Faire les courses", completed: true }
  ]);
  const [newTask, setNewTask] = useState('');
  const handleAddTask = () => {
    if (!newTask) return;
    setTasks([...tasks, { id: Date.now(), text: newTask, completed: false }]);
    setNewTask('');
  };

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

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

return (
  <div>
    <input

```

```

        type="text"
        value={newTask}
        onChange={e => setNewTask(e.target.value)}
        placeholder="Nouvelle tâche"
      />
      <button onClick={handleAddTask}>Ajouter</button>

      <ul>
        {tasks.map(task => (
          <li key={task.id}>
            <button onClick={() => handleDeleteTask(task.id)}>X</button>
            <input
              type="checkbox"
              checked={task.completed}
              onChange={() => handleToggleCompleted(task.id)}
            />
            {task.text} {task.completed ? "(√)" : ""}
          </li>
        )));
      </ul>
    </div>
  );
}

export default TodoList;

```

Étape 6 : Filtrer les tâches

Objectif pédagogique :

Comprendre l'utilisation de **state pour le filtrage** et la communication entre composants.

Instructions :

1. Créez un state « **filter** » avec valeurs "all" | "completed" | "pending".
2. Affichez uniquement les tâches correspondant au filtre choisi.

Ma To-Do List

Toutes	Complétées	Non complétées
Nouvelle tâche		Ajouter

- Écriture et maintenance du code (✓)
- Revue de code (Code Review)
- Tests et débogage (✓)
- Veille technologique et amélioration continue
- Corriger un bug signalé sur Jira / Trello / Gitlab (✓)
- Code Review sur une Pull Request d'un collègue
- Déployer une nouvelle version sur un serveur

Correction / Exemple :

```
//src/components/TodoList.jsx
import React, { useState } from 'react';

function TodoList() {
  const [tasks, setTasks] = useState([
    { id: 1, text: "Apprendre React", completed: false },
    { id: 2, text: "Faire les courses", completed: true }
  ]);

  const [filter, setFilter] = useState('all');
  const filteredTasks = tasks.filter(task => {
    if (filter === 'all') return true;
    if (filter === 'completed') return task.completed;
    return !task.completed;
  });

  const [newTask, setNewTask] = useState('');
  const handleAddTask = () => {
    if (!newTask) return;
    setTasks([...tasks, { id: Date.now(), text: newTask, completed: false }]);
    setNewTask('');
  };

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

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

  return (
    <div>

      <div>
        <button onClick={() => setFilter('all')}>Toutes</button>
        <button onClick={() => setFilter('completed')}>Complétées</button>
        <button onClick={() => setFilter('pending')}>Non complétées</button>
      </div>

      <input
        type="text"
        value={newTask}
        onChange={e => setNewTask(e.target.value)}
        placeholder="Nouvelle tâche"
      />
      <button onClick={handleAddTask}>Ajouter</button>

      <ul>
        {filteredTasks.map(task => (
          <li key={task.id}>
            <button onClick={() => handleDeleteTask(task.id)}>X</button>
            <input
              type="checkbox"
              checked={task.completed}
              onChange={() => handleToggleCompleted(task.id)}
            />

            {task.text} {task.completed ? "(√)" : ""}
          </li>
        )));
      </ul>
    </div>
  );
}
```

```
}
```

```
export default TodoList;
```

Étape 7 : Compteur dynamique

Objectif:

Comprendre la **calcul dynamique à partir du state**.

Instructions :

1. Ajoutez un compteur qui affiche le nombre de tâches non complétées.

Ma To-Do List



- Écriture et maintenance du code (✓)
- Revue de code (Code Review)
- Tests et débogage (✓)
- Veille technologique et amélioration continue
- Corriger un bug signalé sur Jira / Trello / Gitlab (✓)
- Code Review sur une Pull Request d'un collègue
- Déployer une nouvelle version sur un serveur

Tâches restantes : 4

Correction / Exemple :

```
<p>Tâches restantes : {tasks.filter(task => !task.completed).length}</p>
```

Étape 8 : Sauvegarde avec localStorage

Objectif pédagogique :

Apprendre **useEffect** pour synchroniser le state avec **localStorage**.

Instructions :

1. Au chargement, lire les tâches depuis localStorage.
2. À chaque modification de tasks, mettre à jour localStorage.

Correction / Exemple :

```
//src/components/TodoList.jsx
```

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

```
function TodoList() {
  const [tasks, setTasks] = useState([
    { id: 1, text: "Apprendre React", completed: false },
    { id: 2, text: "Faire les courses", completed: true }
  ]);

  const [filter, setFilter] = useState('all');
  const filteredTasks = tasks.filter(task => {
    if (filter === 'all') return true;
    if (filter === 'completed') return task.completed;
    return !task.completed;
  });

  const [newTask, setNewTask] = useState('');
  const handleAddTask = () => {
    if (!newTask) return;
    setTasks([...tasks, { id: Date.now(), text: newTask, completed: false }]);
    setNewTask('');
  };

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

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

  useEffect(() => {
    const savedTasks = JSON.parse(localStorage.getItem('tasks')) || [];
    setTasks(savedTasks);
  }, []);

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

  return (
    <div>

      <div>
        <button onClick={() => setFilter('all')}>Toutes</button>
        <button onClick={() => setFilter('completed')}>Complétées</button>
        <button onClick={() => setFilter('pending')}>Non complétées</button>
      </div>

      <input
        type="text"
        value={newTask}
        onChange={e => setNewTask(e.target.value)}
        placeholder="Nouvelle tâche"
      />
      <button onClick={handleAddTask}>Ajouter</button>

      <ul>
        {filteredTasks.map(task => (
          <li key={task.id}>
            <button onClick={() => handleDeleteTask(task.id)}>X</button>
            <input
              type="checkbox"
              checked={task.completed}
              onChange={() => handleToggleCompleted(task.id)}
            />
            {task.text} {task.completed ? "(√)" : ""}
          </li>
        ))}
      </ul>
    </div>
  );
}
```

```

        ))}
    </ul>
    <p>Tâches restantes : {tasks.filter(task => !task.completed).length}</p>
  </div>
}
}

export default TodoList;

```

Fin de l'atelier

Étape 9 : Refactoriser avec des composants enfants

Objectif:

Comprendre **props** et la communication parent-enfant.

Instructions :

1. Créez un composant Task.jsx pour chaque tâche.
2. Passez les props task, onToggle, onDelete.

Correction / Exemple :

```
//src/components/Task.jsx
export default function Task({ task, onToggle, onDelete }) {
  return (
    <li>
      <input type="checkbox" checked={task.completed} onChange={() => onToggle(task.id)} />
      {task.text}
      <button onClick={() => onDelete(task.id)}>Supprimer</button>
    </li>
  );
}
```

// TodoList.jsx

```
//src/components/TodoList.jsx
import React, { useState, useEffect } from 'react';
import Task from './Task';

function TodoList() {
  const [tasks, setTasks] = useState([
    { id: 1, text: "Apprendre React", completed: false },
    { id: 2, text: "Faire les courses", completed: true }
  ]);

  const [filter, setFilter] = useState('all');
  const filteredTasks = tasks.filter(task => {
    if (filter === 'all') return true;
    if (filter === 'completed') return task.completed;
    return !task.completed;
  });

  const [newTask, setNewTask] = useState('');
  const handleAddTask = () => {
    if (!newTask) return;
    setTasks([...tasks, { id: Date.now(), text: newTask, completed: false }]);
  };
}

export default TodoList;
```

```

        setNewTask('');
    };

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

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

    useEffect(() => {
        const savedTasks = JSON.parse(localStorage.getItem('tasks')) || [];
        setTasks(savedTasks);
    }, []);

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

    return (
        <div>
            <div>
                <button onClick={() => setFilter('all')}>Toutes</button>
                <button onClick={() => setFilter('completed')}>Complétées</button>
                <button onClick={() => setFilter('pending')}>Non complétées</button>
            </div>

            <input
                type="text"
                value={newTask}
                onChange={e => setNewTask(e.target.value)}
                placeholder="Nouvelle tâche"
            />
            <button onClick={handleAddTask}>Ajouter</button>

            <ul>
                {filteredTasks.map(task => (
                    <Task key={task.id} task={task} onToggle={handleToggleCompleted}
onDelete={handleDeleteTask} />
                )));
            </ul>

            <p>Tâches restantes : {tasks.filter(task => !task.completed).length}</p>
        </div>
    );
}

export default TodoList;

```

Étape 10 : Bonus : édition de tâche

Objectif pédagogique :

Mettre en pratique **state local** et **props** pour modifier des données existantes.

Instructions :

1. Ajoutez un bouton "Modifier" pour chaque tâche.
2. Permettez de changer le texte et sauvegarder la modification dans le state tasks.

Correction / Exemple :

```
const handleEditTask = (id, newText) => {
  setTasks(tasks.map(task => task.id === id ? { ...task, text: newText } : task));
};
```

Code final

```
// app/src/components/TodoList.jsx
import React, { useState, useEffect } from 'react';
import Task from './Task';

function TodoList() {
  const [tasks, setTasks] = useState([]);
  const [filter, setFilter] = useState('all');
  const [newTask, setNewTask] = useState('');

  // Charger depuis localStorage au démarrage
  useEffect(() => {
    const savedTasks = JSON.parse(localStorage.getItem('tasks')) || [];
    setTasks(savedTasks);
  }, []);

  // Sauvegarder dans localStorage à chaque changement
  useEffect(() => {
    localStorage.setItem('tasks', JSON.stringify(tasks));
  }, [tasks]);

  const handleAddTask = () => {
    if (!newTask.trim()) return;
    setTasks([...tasks, { id: Date.now(), text: newTask, completed: false }]);
    setNewTask('');
  };

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

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

  const handleEditTask = (id, newText) => {
    setTasks(tasks.map(task =>
      task.id === id ? { ...task, text: newText } : task
    ));
  };

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

  return (
    <div>
      <div>
        <button onClick={() => setFilter('all')}>Toutes</button>
```

```

        <button onClick={() => setFilter('completed')}>Complétées</button>
        <button onClick={() => setFilter('pending')}>Non complétées</button>
    </div>

    <input
        type="text"
        value={newTask}
        onChange={e => setNewTask(e.target.value)}
        placeholder="Nouvelle tâche"
    />
    <button onClick={handleAddTask}>Ajouter</button>

    <ul>
        {filteredTasks.map(task => (
            <Task
                key={task.id}
                task={task}
                onToggle={handleToggleCompleted}
                onDelete={handleDeleteTask}
                onEdit={handleEditTask}
            />
        ))}
    </ul>

    <p>Tâches restantes : {tasks.filter(task => !task.completed).length}</p>
</div>
);
}

export default TodoList;

```

```

// app/src/components/Task.jsx
import { useState } from "react";

export default function Task({ task, onToggle, onDelete, onEdit }) {
    const [isEditing, setIsEditing] = useState(false);
    const [newText, setNewText] = useState(task.text);

    const handleSave = () => {
        if (!newText.trim()) return;
        onEdit(task.id, newText);
        setIsEditing(false);
    };

    return (
        <li>
            <input
                type="checkbox"
                checked={task.completed}
                onChange={() => onToggle(task.id)}
            />

            {isEditing ? (
                <input
                    type="text"
                    value={newText}
                    onChange={(e) => setNewText(e.target.value)}
                    autoFocus
                />
            ) : (
                <span
                    style={{
                        textDecoration: task.completed ? "line-through" : "none",
                        marginRight: "10px"
                    }}
                >
                    {task.text}
                </span>
            )}
        </li>
    );
}

```

```
        </span>
    )}
    {isEditing ? (
        <button onClick={handleSave}>💾</button>
    ) : (
        <button onClick={() => setIsEditing(true)}>✍</button>
    )}
    <button onClick={() => onDelete(task.id)}>☒</button>
</li>
);
}
```