Exercice 10 - Mini Application de Synthèse

Objectif

Combiner tous les concepts du Module 1 dans une application complète.

Concepts révisés

- JSX et composants
- Props et state (useState)
- Événements (onClick, onChange)
- Listes avec map() et key
- Rendu conditionnel
- Formulaires contrôlés
- CSS et classes dynamiques

Énoncé : Gestionnaire de tâches (Todo List)

Créer une application de gestion de tâches avec les fonctionnalités suivantes :

Fonctionnalités requises

1. **Afficher la liste des tâches** (map + key)
2. **Ajouter une tâche** (formulaire contrôlé + state)
3. **Marquer une tâche comme complétée** (toggle + classes dynamiques)
4. **Supprimer une tâche** (filter + callback prop)
5. **Filtrer les tâches** (toutes / actives / complétées)
6. **Compteur de tâches restantes** (rendu conditionnel)

Structure recommandée

```
ex10-todo/
  TodoApp.jsx       # Composant principal (state)
  TodoForm.jsx      # Formulaire d'ajout
  TodoList.jsx      # Liste des tâches
  TodoItem.jsx      # Une tâche individuelle
  TodoFilters.jsx   # Boutons de filtre
  TodoApp.css       # Styles
```

Correction

TodoApp.jsx (Composant principal)

```jsx
// src/exercices/ex10-todo/TodoApp.jsx

import { useState } from 'react';
import TodoForm from './TodoForm';
import TodoList from './TodoList';
import TodoFilters from './TodoFilters';
import './TodoApp.css';

function TodoApp() {
  // ═══════════════════════════════════════════════════════════
  // STATE PRINCIPAL
  //
  // - todos : tableau des tâches
  // - filter : filtre actif ('all', 'active', 'completed')
  //
  // Le state est ici car ce composant "orchestre" les autres.
  // C'est le pattern "lifting state up" : le state est dans le
  // plus proche ancêtre commun des composants qui l'utilisent.
  // ═══════════════════════════════════════════════════════════
  const [todos, setTodos] = useState([
    { id: 1, text: "Apprendre React", completed: true },
    { id: 2, text: "Créer une todo app", completed: false },
    { id: 3, text: "Pratiquer les hooks", completed: false },
  ]);

  const [filter, setFilter] = useState('all');

  // ═══════════════════════════════════════════════════════════
  // HANDLERS
  // Ces fonctions modifient le state et seront passées en props
  // aux composants enfants qui en ont besoin.
  // ═══════════════════════════════════════════════════════════

  // Ajouter une tâche
  function addTodo(text) {
    const newTodo = {
      id: Date.now(),  // ID unique basé sur le timestamp
      text: text.trim(),
      completed: false
    };
    setTodos(prev => [...prev, newTodo]);
  }

  // Basculer le statut d'une tâche
  function toggleTodo(id) {
    setTodos(prev =>
      prev.map(todo =>
        todo.id === id
          ? { ...todo, completed: !todo.completed }
          : todo
      )
    );
  }
```

```
  // Supprimer une tâche
  function deleteTodo(id) {
    setTodos(prev => prev.filter(todo => todo.id !== id));
  }

  // ============================================================
  // CALCULS DÉRIVÉS
  // Ces valeurs sont calculées à partir du state à chaque rendu
  // ============================================================

  // Filtrer les todos selon le filtre actif
  const filteredTodos = todos.filter(todo => {
    if (filter === 'active') return !todo.completed;
    if (filter === 'completed') return todo.completed;
    return true;  // 'all'
  });

  // Compteurs
  const totalCount = todos.length;
  const activeCount = todos.filter(t => !t.completed).length;
  const completedCount = todos.filter(t => t.completed).length;

  // ============================================================
  // RENDU
  // Chaque composant enfant reçoit les props dont il a besoin
  // ============================================================
  return (
    <div className="todo-app">
      <h1>Mes Tâches</h1>

      {/* Formulaire d'ajout */}
      <TodoForm onAddTodo={addTodo} />

      {/* Filtres */}
      <TodoFilters
        currentFilter={filter}
        onFilterChange={setFilter}
        counts={{ total: totalCount, active: activeCount, completed:
completedCount }}
      />

      {/* Liste des tâches */}
      {filteredTodos.length === 0 ? (
        <p className="empty-message">
          {filter === 'all'
            ? "Aucune tâche pour le moment"
            : `Aucune tâche ${filter === 'active' ? 'active' : 'complétée'}`
          }
        </p>
      ) : (
        <TodoList
          todos={filteredTodos}
          onToggle={toggleTodo}
```

```
            onDelete={deleteTodo}
          />
        )}

        {/* Compteur */}
        <footer className="todo-footer">
          {activeCount === 0 ? (
            <p>Toutes les tâches sont complétées !</p>
          ) : (
            <p>{activeCount} tâche{activeCount > 1 ? 's' : ''} restante{activeCount
> 1 ? 's' : ''}</p>
          )}
        </footer>
      </div>
    );
}

export default TodoApp;
```

TodoForm.jsx

```
// src/exercices/ex10-todo/TodoForm.jsx

import { useState, useRef } from 'react';

function TodoForm({ onAddTodo }) {
  // ═══════════════════════════════════════════════════════════
  // STATE LOCAL
  // Le state du formulaire est local à ce composant
  // Il n'a pas besoin d'être "remonté" car seul ce composant l'utilise
  // ═══════════════════════════════════════════════════════════
  const [text, setText] = useState('');
  const inputRef = useRef(null);

  function handleSubmit(e) {
    e.preventDefault();

    // Validation
    if (!text.trim()) {
      inputRef.current.focus();
      return;
    }

    // Appeler la fonction du parent pour ajouter la tâche
    onAddTodo(text);

    // Réinitialiser le champ
    setText('');

    // Garder le focus pour enchaîner les ajouts
    inputRef.current.focus();
```

```
  }

  return (
    <form onSubmit={handleSubmit} className="todo-form">
      <input
        ref={inputRef}
        type="text"
        value={text}
        onChange={(e) => setText(e.target.value)}
        placeholder="Ajouter une tâche..."
        className="todo-input"
      />
      <button
        type="submit"
        disabled={!text.trim()}
        className="todo-add-btn"
      >
        Ajouter
      </button>
    </form>
  );
}

export default TodoForm;
```

TodoList.jsx

```
// src/exercices/ex10-todo/TodoList.jsx

import TodoItem from './TodoItem';

function TodoList({ todos, onToggle, onDelete }) {
  // ═══════════════════════════════════════════════════════════
  // Ce composant ne fait que "passer" les props aux TodoItem
  // C'est un pattern courant : composant "wrapper" de liste
  // ═══════════════════════════════════════════════════════════
  return (
    <ul className="todo-list">
      {todos.map(todo => (
        <TodoItem
          key={todo.id}
          todo={todo}
          onToggle={onToggle}
          onDelete={onDelete}
        />
      ))}
    </ul>
  );
}

export default TodoList;
```

TodoItem.jsx

```jsx
// src/exercices/ex10-todo/TodoItem.jsx

function TodoItem({ todo, onToggle, onDelete }) {
  // ═══════════════════════════════════════════════════════════
  // CLASSES DYNAMIQUES
  // La classe 'completed' est ajoutée si la tâche est complétée
  // ═══════════════════════════════════════════════════════════
  const itemClass = `todo-item ${todo.completed ? 'completed' : ''}`;

  return (
    <li className={itemClass}>
      {/* Checkbox pour toggle */}
      <input
        type="checkbox"
        checked={todo.completed}
        onChange={() => onToggle(todo.id)}
        className="todo-checkbox"
      />

      {/* Texte de la tâche */}
      <span className="todo-text">{todo.text}</span>

      {/* Bouton supprimer */}
      <button
        onClick={() => onDelete(todo.id)}
        className="todo-delete-btn"
        aria-label="Supprimer"
      >
        ×
      </button>
    </li>
  );
}

export default TodoItem;
```

TodoFilters.jsx

```jsx
// src/exercices/ex10-todo/TodoFilters.jsx

function TodoFilters({ currentFilter, onFilterChange, counts }) {
  const filters = [
    { key: 'all', label: `Toutes (${counts.total})` },
    { key: 'active', label: `Actives (${counts.active})` },
    { key: 'completed', label: `Complétées (${counts.completed})` },
  ];
```

```
  return (
    <div className="todo-filters">
      {filters.map(filter => (
        <button
          key={filter.key}
          onClick={() => onFilterChange(filter.key)}
          className={`filter-btn ${currentFilter === filter.key ? 'active' : ''}`}
        >
          {filter.label}
        </button>
      ))}
    </div>
  );
}

export default TodoFilters;
```

TodoApp.css

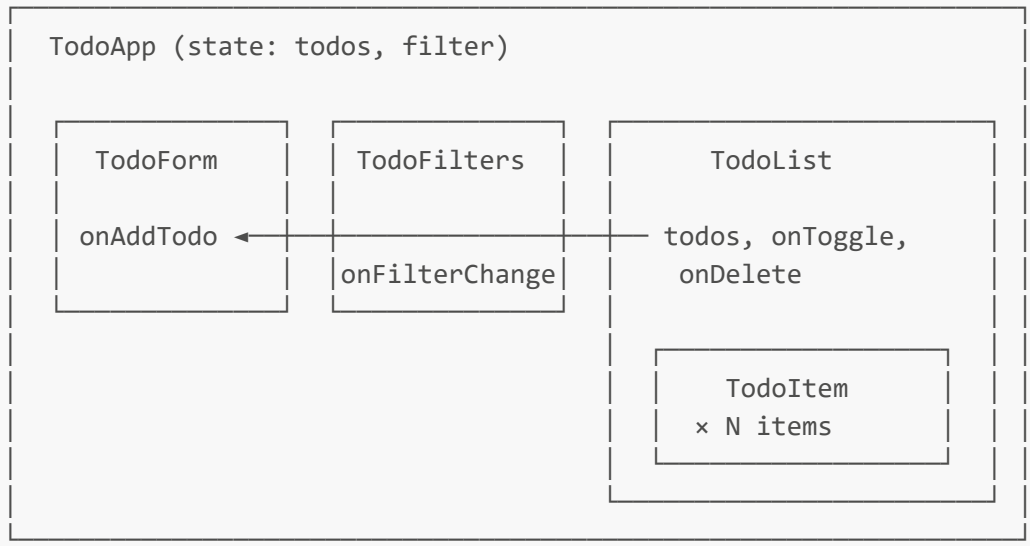
```
/* src/exercices/ex10-todo/TodoApp.css */

.todo-app {
  max-width: 500px;
  margin: 2rem auto;
  padding: 1.5rem;
  background: white;
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

.todo-app h1 {
  margin: 0 0 1.5rem 0;
  text-align: center;
  color: #1f2937;
}

/* Formulaire */
.todo-form {
  display: flex;
  gap: 0.5rem;
  margin-bottom: 1rem;
}

.todo-input {
  flex: 1;
  padding: 0.75rem;
  font-size: 1rem;
  border: 1px solid #d1d5db;
  border-radius: 4px;
}
```

```css
.todo-input:focus {
  outline: none;
  border-color: #2563eb;
}

.todo-add-btn {
  padding: 0.75rem 1.5rem;
  font-size: 1rem;
  color: white;
  background: #2563eb;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.todo-add-btn:disabled {
  background: #9ca3af;
  cursor: not-allowed;
}

/* Filtres */
.todo-filters {
  display: flex;
  gap: 0.5rem;
  margin-bottom: 1rem;
}

.filter-btn {
  flex: 1;
  padding: 0.5rem;
  font-size: 0.875rem;
  border: 1px solid #d1d5db;
  border-radius: 4px;
  background: white;
  cursor: pointer;
  transition: all 0.2s;
}

.filter-btn:hover {
  background: #f3f4f6;
}

.filter-btn.active {
  background: #2563eb;
  color: white;
  border-color: #2563eb;
}

/* Liste */
.todo-list {
  list-style: none;
  padding: 0;
  margin: 0;
}
```

```css
.todo-item {
  display: flex;
  align-items: center;
  gap: 0.75rem;
  padding: 0.75rem;
  border-bottom: 1px solid #e5e7eb;
  transition: background 0.2s;
}

.todo-item:hover {
  background: #f9fafb;
}

.todo-item.completed .todo-text {
  text-decoration: line-through;
  color: #9ca3af;
}

.todo-checkbox {
  width: 1.25rem;
  height: 1.25rem;
  cursor: pointer;
}

.todo-text {
  flex: 1;
}

.todo-delete-btn {
  padding: 0.25rem 0.5rem;
  font-size: 1.25rem;
  color: #9ca3af;
  background: none;
  border: none;
  cursor: pointer;
  border-radius: 4px;
}

.todo-delete-btn:hover {
  color: #dc2626;
  background: #fee2e2;
}

/* Footer */
.todo-footer {
  margin-top: 1rem;
  padding-top: 1rem;
  border-top: 1px solid #e5e7eb;
  text-align: center;
  color: #6b7280;
  font-size: 0.875rem;
}
```

```css
.empty-message {
  text-align: center;
  color: #9ca3af;
  padding: 2rem;
}
```

Schéma d'architecture




```
Flux des données :
  ↓ Props descendent (todos, handlers)
  ↑ Événements remontent (onAddTodo, onToggle, onDelete)
```

Checklist finale Module 1

Après cet exercice, vérifier que vous savez :

- ☐ Créer un composant React
- ☐ Utiliser JSX avec des expressions {}
- ☐ Passer et recevoir des props
- ☐ Gérer le state avec useState
- ☐ Gérer les événements (onClick, onChange, onSubmit)
- ☐ Afficher des listes avec map() et key
- ☐ Faire du rendu conditionnel (ternaire, &&)
- ☐ Appliquer des classes CSS dynamiques
- ☐ Créer un formulaire contrôlé
- ☐ Structurer une application en plusieurs composants