React & Tailwind CSS 4 Todo CRUD App Mastery Guide

Project Overview

This guide will walk you through building a complete Todo application with React and Tailwind CSS 4, implementing all CRUD (Create, Read, Update, Delete) operations. By the end, you'll have mastered:

1. React fundamentals (components, hooks, state management)
2. Tailwind CSS 4 styling approach
3. CRUD operations with React
4. Form handling and validation
5. Local storage for data persistence
6. Responsive design principles

Setup & Installation

Start by creating a new React project with Vite and installing Tailwind CSS 4:

bash

```bash
# Create a new React project with Vite
npm create vite@latest todo-app -- --template react

# Navigate to the project directory
cd todo-app

# Install dependencies
npm install

# Install Tailwind CSS 4 and its peer dependencies
npm install -D tailwindcss@latest postcss autoprefixer

# Initialize Tailwind CSS
npx tailwindcss init -p
```

Configure Tailwind CSS 4 by updating the `tailwind.config.js` file:

javascript

Copy

```javascript
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Add Tailwind directives to your CSS in `src/index.css`:

css

Copy

```css
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Project Structure

Organize your project with the following structure:

Copy

```
src/
├── components/
│   ├── TodoForm.jsx
│   ├── TodoItem.jsx
│   ├── TodoList.jsx
│   └── TodoFilter.jsx
├── hooks/
│   └── useTodos.js
├── App.jsx
└── main.jsx
```

1. Building the Todo State Management

Create a custom hook for managing todos in `src/hooks/useTodos.js`:

javascript

Copy

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

export default function useTodos() {
  // Get todos from localStorage or start with empty array
  const [todos, setTodos] = useState(() => {
    const savedTodos = localStorage.getItem('todos');
    return savedTodos ? JSON.parse(savedTodos) : [];
  });

  // Store todos in localStorage whenever they change
  useEffect(() => {
    localStorage.setItem('todos', JSON.stringify(todos));
  }, [todos]);

  // Add a new todo
  function addTodo(title) {
    const newTodo = {
      id: Date.now(),
      title,
      completed: false,
      createdAt: new Date().toISOString()
    };

    setTodos(prevTodos => [newTodo, ...prevTodos]);
  }

  // Toggle completion status
  function toggleTodo(id) {
    setTodos(prevTodos =>
      prevTodos.map(todo =>
        todo.id === id ? { ...todo, completed: !todo.completed } : todo
      )
    );
  }

  // Update todo title
  function updateTodo(id, title) {
    setTodos(prevTodos =>
      prevTodos.map(todo =>
        todo.id === id ? { ...todo, title } : todo
      )
    );
  }

  // Delete a todo
  function deleteTodo(id) {
    setTodos(prevTodos => prevTodos.filter(todo => todo.id !== id));
  }

  return {
    todos,
    addTodo,
    toggleTodo,
    updateTodo,
    deleteTodo
  };
}
```

2. Creating the Todo Form Component

Build the form for adding new todos in `src/components/TodoForm.jsx` using Tailwind CSS 4 classes:

jsx

Copy

```jsx
import { useState } from 'react';

export default function TodoForm({ addTodo }) {
  const [title, setTitle] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();

    // Validate input
    if (!title.trim()) {
      setError('Todo cannot be empty');
      return;
    }

    // Add the todo and reset form
    addTodo(title.trim());
    setTitle('');
    setError('');
  };

  return (
    <form
      onSubmit={handleSubmit}
      className="mb-6"
    >
      <div className="flex flex-col gap-2">
        <input
          type="text"
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          placeholder="Add a new todo..."
          className="w-full px-4 py-2 border rounded-lg focus-visible:outline-none focus-visibl
        />
        {error && (
          <p className="text-red-500 text-sm">{error}</p>
        )}
      </div>
      <button
        type="submit"
        className="mt-2 w-full bg-blue-500 hover:bg-blue-600 text-white font-medium py-2 px-4 r
      >
```

```
        Add Todo
      </button>
    </form>
  );
}
```

3. Building the Todo Item Component

Create the component for individual todo items in `src/components/TodoItem.jsx`:

jsx

Copy

```jsx
import { useState } from 'react';

export default function TodoItem({ todo, toggleTodo, updateTodo, deleteTodo }) {
  const [isEditing, setIsEditing] = useState(false);
  const [editValue, setEditValue] = useState(todo.title);

  const handleEditSubmit = () => {
    if (editValue.trim()) {
      updateTodo(todo.id, editValue);
      setIsEditing(false);
    }
  };

  return (
    <li className="border rounded-lg mb-2 overflow-hidden">
      <div className="p-4 flex items-center justify-between bg-white">
        {isEditing ? (
          <div className="flex-grow flex items-center gap-2">
            <input
              type="text"
              value={editValue}
              onChange={(e) => setEditValue(e.target.value)}
              className="flex-grow px-2 py-1 border rounded focus-visible:outline-none focus-vi
              autoFocus
            />
            <button
              onClick={handleEditSubmit}
              className="bg-green-500 hover:bg-green-600 text-white px-3 py-1 rounded"
            >
              Save
            </button>
            <button
              onClick={() => setIsEditing(false)}
              className="bg-gray-300 hover:bg-gray-400 text-gray-800 px-3 py-1 rounded"
            >
              Cancel
            </button>
          </div>
        ) : (
          <>
            <div className="flex items-center flex-grow">
              <input
            type="checkbox"
            checked={todo.completed}
            onChange={() => toggleTodo(todo.id)}
            className="h-5 w-5 text-blue-500 rounded accent-blue-500 focus-visible:ring-0"
          />
          <span
            className={`ml-3 ${
              todo.completed ? 'line-through text-gray-500' : 'text-gray-800'
            }`}
          >
            {todo.title}
          </span>
        </div>
        <div className="flex items-center gap-2">
          <button
            onClick={() => setIsEditing(true)}
            className="text-blue-500 hover:text-blue-700"
          >
            <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
              <path d="M13.586 3.586a2 2 0 112.828 2.828l-.793.793-2.828-2.828.793-.793zM11
            </svg>
          </button>
          <button
            onClick={() => deleteTodo(todo.id)}
            className="text-red-500 hover:text-red-700"
          >
            <svg xmlns="http://www.w3.org/2000/svg" className="h-5 w-5" viewBox="0 0 20 20"
              <path fillRule="evenodd" d="M9 2a1 1 0 00-.894.553L7.382 4H4a1 1 0 000 2v10a2
            </svg>
          </button>
        </div>
      </>
    )}
  </div>
</li>
);
}
```

4. Creating the Todo List Component

Build the component that renders all todos in `src/components/TodoList.jsx`:

jsx

```jsx
import TodoItem from './TodoItem';

export default function TodoList({ todos, toggleTodo, updateTodo, deleteTodo, filter }) {
  // Filter todos based on the selected filter
  const filteredTodos = todos.filter(todo => {
    if (filter === 'all') return true;
    if (filter === 'active') return !todo.completed;
    if (filter === 'completed') return todo.completed;
    return true;
  });

  return (
    <div className="mt-4">
      {filteredTodos.length === 0 ? (
        <p className="text-center text-gray-500 py-4">No todos to display</p>
      ) : (
        <ul className="flex flex-col gap-2">
          {filteredTodos.map(todo => (
            <TodoItem
              key={todo.id}
              todo={todo}
              toggleTodo={toggleTodo}
              updateTodo={updateTodo}
              deleteTodo={deleteTodo}
            />
          ))}
        </ul>
      )}
    </div>
  );
}
```

5. Building the Todo Filter Component

Create a component for filtering todos in `src/components/TodoFilter.jsx`:

jsx

Copy

```jsx
export default function TodoFilter({ filter, setFilter, todos }) {
  const activeCount = todos.filter(todo => !todo.completed).length;
  const completedCount = todos.filter(todo => todo.completed).length;

  return (
    <div className="flex flex-col sm:flex-row sm:items-center justify-between mt-6 mb-4 bg-gray
      <div className="mb-2 sm:mb-0">
        <span className="text-gray-700">
          {activeCount} {activeCount === 1 ? 'item' : 'items'} left
        </span>
      </div>

      <div className="flex gap-2">
        <button
          onClick={() => setFilter('all')}
          className={`px-3 py-1 rounded ${
            filter === 'all'
              ? 'bg-blue-500 text-white'
              : 'bg-white text-gray-700 hover:bg-gray-200'
          }`}
        >
          All
        </button>
        <button
          onClick={() => setFilter('active')}
          className={`px-3 py-1 rounded ${
            filter === 'active'
              ? 'bg-blue-500 text-white'
              : 'bg-white text-gray-700 hover:bg-gray-200'
          }`}
        >
          Active ({activeCount})
        </button>
        <button
          onClick={() => setFilter('completed')}
          className={`px-3 py-1 rounded ${
            filter === 'completed'
              ? 'bg-blue-500 text-white'
              : 'bg-white text-gray-700 hover:bg-gray-200'
          }`}
        >
          Completed ({completedCount})
        </button>
      </div>
    </div>
  );
}
```

6. Putting It All Together in App.jsx

Finally, assemble all components in `src/App.jsx`:

jsx Copy

```jsx
import { useState } from 'react';
import TodoForm from './components/TodoForm';
import TodoList from './components/TodoList';
import TodoFilter from './components/TodoFilter';
import useTodos from './hooks/useTodos';

function App() {
  const { todos, addTodo, toggleTodo, updateTodo, deleteTodo } = useTodos();
  const [filter, setFilter] = useState('all');

  return (
    <div className="min-h-dvh bg-gray-100 py-8">
      <div className="max-w-md mx-auto bg-white rounded-xl shadow-md overflow-hidden p-6">
        <h1 className="text-2xl font-bold text-center text-gray-800 mb-6">
          React Todo App
        </h1>

        <TodoForm addTodo={addTodo} />

        <TodoFilter
          filter={filter}
          setFilter={setFilter}
          todos={todos}
        />

        <TodoList
          todos={todos}
          toggleTodo={toggleTodo}
          updateTodo={updateTodo}
          deleteTodo={deleteTodo}
          filter={filter}
        />

        {todos.length > 0 && (
          <div className="mt-6 text-center">
            <button
              onClick={() => {
                const confirmed = window.confirm('Are you sure you want to clear all completed
                if (confirmed) {
                  todos
                    .filter(todo => todo.completed)
                    .forEach(todo => deleteTodo(todo.id));
                }
              }}
              className="text-sm text-red-500 hover:text-red-700"
            >
              Clear completed
            </button>
          </div>
        )}
      </div>
    </div>
  );
}

export default App;
```

7. Tailwind CSS 4 Specific Features

Tailwind CSS 4 introduces some new features and changes that you can leverage in your Todo app:

1. Logical Properties

jsx Copy

```jsx
// Instead of this (Tailwind CSS 3)
<div className="pl-4 pr-2 pt-2 pb-4">

// Use this in Tailwind CSS 4
<div className="ps-4 pe-2 pt-2 pb-4">
```

2. Modern Focus Selectors

jsx Copy

```jsx
// Instead of this (Tailwind CSS 3)
<input className="focus:ring-2 focus:outline-none" />

// Use this in Tailwind CSS 4
<input className="focus-visible:ring-2 focus-visible:outline-none" />
```

3. Modern Viewport Units

jsx Copy

```jsx
// Instead of this (Tailwind CSS 3)
<div className="min-h-screen">

// Use this in Tailwind CSS 4
<div className="min-h-dvh">
```

4. Using `gap` Instead of `space-y/x`

jsx Copy

```jsx
// Instead of this (Tailwind CSS 3)
<div className="space-y-2">

// Use this in Tailwind CSS 4
<div className="flex flex-col gap-2">
```

5. Simplified Color System

jsx Copy

```jsx
// Add a new color to your theme in tailwind.config.js
theme: {
  extend: {
    colors: {
      'todo-primary': '#3b82f6', // blue-500
      'todo-accent': '#10b981', // emerald-500
    }
  }
}

// Then use it in your components
<button className="bg-todo-primary hover:bg-todo-primary/80">
```

6. CSS Variables for Dynamic Theming

jsx

Copy

```jsx
// Define variables in :root
:root {
  --color-primary: 59 130 246; /* blue-500 */
}

// Configure in tailwind.config.js
theme: {
  extend: {
    colors: {
      primary: 'rgb(var(--color-primary) / <alpha-value>)',
    }
  }
}

// Then use in your components
<button className="bg-primary hover:bg-primary/80">
```

8. Advanced Features to Master React & Tailwind CSS 4

Once you've built the basic app, try implementing these advanced features:

1. **Dark mode with Tailwind CSS 4**

   jsx

   Copy

   ```jsx
   // In tailwind.config.js
   module.exports = {
     darkMode: 'class', // or 'media'
     // ...
   }

   // In your component
   <div className="bg-white dark:bg-gray-800 text-gray-800 dark:text-white">
   ```

2. **Leveraging Tailwind's container queries**

jsx

```jsx
// In your component
<div className="@container">
  <div className="@md:flex @md:gap-4">
    {/* Content adapts based on container size, not viewport */}
  </div>
</div>
```

3. **Implementing a custom plugin**

js

```js
// In tailwind.config.js
const plugin = require('tailwindcss/plugin')

module.exports = {
  plugins: [
    plugin(function({ addComponents }) {
      addComponents({
        '.todo-card': {
          backgroundColor: 'white',
          borderRadius: '0.5rem',
          padding: '1rem',
          boxShadow: '0 1px 3px 0 rgba(0, 0, 0, 0.1)',
          '&:hover': {
            boxShadow: '0 4px 6px -1px rgba(0, 0, 0, 0.1)'
          }
        }
      })
    })
  ]
}
```

4. **Using Tailwind's JIT mode to create dynamic utilities**

jsx

```jsx
<div className="grid-cols-[repeat(auto-fill,minmax(250px,1fr))]">
  {/* Creates a responsive grid with auto-filling columns */}
</div>
```

9. Testing Your Application

Learn to write tests for your React components:

bash

Copy

```bash
# Install testing libraries
npm install -D vitest @testing-library/react @testing-library/jest-dom
```

Create a simple test for the TodoItem component:

jsx

Copy

```jsx
// src/components/TodoItem.test.jsx
import { render, screen, fireEvent } from '@testing-library/react';
import { describe, it, expect, vi } from 'vitest';
import TodoItem from './TodoItem';

describe('TodoItem', () => {
  const mockTodo = {
    id: 1,
    title: 'Test Todo',
    completed: false
  };

  const mockToggle = vi.fn();
  const mockUpdate = vi.fn();
  const mockDelete = vi.fn();

  it('renders the todo item correctly', () => {
    render(
      <TodoItem
        todo={mockTodo}
        toggleTodo={mockToggle}
        updateTodo={mockUpdate}
        deleteTodo={mockDelete}
      />
    );

    expect(screen.getByText('Test Todo')).toBeInTheDocument();
  });

  it('calls toggleTodo when checkbox is clicked', () => {
    render(
      <TodoItem
        todo={mockTodo}
        toggleTodo={mockToggle}
        updateTodo={mockUpdate}
        deleteTodo={mockDelete}
      />
    );

    fireEvent.click(screen.getByRole('checkbox'));
    expect(mockToggle).toHaveBeenCalledWith(1);
  });
});
```

10. Deployment

Finally, learn to build and deploy your React app:

bash

```bash
# Build the production version
npm run build

# Preview the production build locally
npm run preview
```

You can deploy to platforms like:

- Vercel
- Netlify
- GitHub Pages

Mastery Challenges

To truly master React and Tailwind CSS 4, complete these challenges:

1. Implement server-side persistence using a backend API
2. Add user authentication
3. Create a mobile-responsive design using Tailwind CSS 4's container queries
4. Set up proper React performance optimizations (memoization, code splitting)
5. Build a settings page with theme customization using CSS variables
6. Implement animations using Tailwind's new transition utilities

By completing this tutorial and the challenges, you'll have developed a deep understanding of both React.js and Tailwind CSS 4 while building a practical, feature-rich application.