Testing Todo App Full

Created	@October 16, 2025 1:47 PM
Tags	

📚 Table of Contents

🧠 Overview

This project shows how to:

- Fetch data from an API using **Axios**
- Post (add) new items to the server
- Display items in a list
- Test API calls and UI updates using **React Testing Library** and **Jest**

We'll use JSONPlaceholder — a free online REST API for testing and prototyping.

⚙️ Project Setup

1️⃣ Create React App (or use Vite)

```
npx create-react-app todo-testing-example
cd todo-testing-example
```

or for Vite:

```
npm create vite@latest todo-testing-example -- --template react
cd todo-testing-example
npm install
```

2️⃣ Install Dependencies

```
npm install axios @testing-library/react @testing-library/jest-dom jest
```

> Note: jest and @testing-library/jest-dom are typically preconfigured with CRA.

> For Vite, you can use vitest instead of jest.

📁 Folder Structure

```
src/
 ├── api/
 │   └── api.js
 ├── components/
 │   └── TodoList.jsx
 ├── __tests__/
 │   └── TodoList.test.jsx
 ├── index.js
 └── App.js
```

1️⃣ Axios API Setup

File: src/api/api.js

```
import axios from "axios";

const api = axios.create({
  baseURL: "https://jsonplaceholder.typicode.com",
});
```

```
export default api;
```

- axios.create() sets a reusable base instance for all API calls.
- This avoids repeating the same base URL everywhere.

2️⃣ TodoList Component

File: src/components/TodoList.jsx

This component:

- Fetches todos from the API (GET /todos)
- Displays them in a list
- Allows the user to add a new todo (POST /todos)

```
import React, { useEffect, useState } from "react";
import api from "../api/api";

const TodoList = () => {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState("");
  const [loading, setLoading] = useState(true);

  // Fetch todos from server
  useEffect(() => {
    const fetchTodos = async () => {
      try {
        const res = await api.get("/todos?_limit=3");
        setTodos(res.data);
      } catch (error) {
        console.error("Error fetching todos:", error);
      } finally {
        setLoading(false);
      }
    };
    fetchTodos();
  }, []);
```

```
// Add new todo
const addTodo = async () ⇒ {
  if (!newTodo.trim()) return;

  try {
    const res = await api.post("/todos", {
      title: newTodo,
      completed: false,
    });
    setTodos((prev) ⇒ [res.data, ...prev]);
    setNewTodo("");
  } catch (error) {
    console.error("Error adding todo:", error);
  }
};

if (loading) return <p>Loading...</p>;

return (
  <div>
    <input
      type="text"
      placeholder="Enter todo"
      value={newTodo}
      onChange={(e) ⇒ setNewTodo(e.target.value)}
      data-testid="todo-input"
    />
    <button onClick={addTodo} data-testid="add-btn">Add</button>

    <ul>
      {todos.map((todo) ⇒ (
        <li key={todo.id} data-testid="todo-item">
          {todo.title}
        </li>
      ))}
    </ul>
  </div>
  );
};

export default TodoList;
```

3️⃣ Unit Testing with Jest & React Testing Library

File: src/__tests__/TodoList.test.jsx

✅ Test Goals

- Verify initial data fetch (GET)
- Verify adding a new todo (POST)
- Ensure UI updates correctly after async calls

```
import { render, screen, waitFor, fireEvent } from "@testing-library/react";
import TodoList from "../components/TodoList";
import api from "../api/api";
import "@testing-library/jest-dom";

jest.mock("../api/api");

describe("TodoList Component", () => {
  it("fetches and displays todos", async () => {
    const mockTodos = [
      { id: 1, title: "Existing Todo 1" },
      { id: 2, title: "Existing Todo 2" },
    ];

    api.get.mockResolvedValueOnce({ data: mockTodos });

    render(<TodoList />);

    // Shows loading initially
    expect(screen.getByText(/loading/i)).toBeInTheDocument();

    // Wait for todos to appear
    await waitFor(() => {
      expect(screen.getAllByTestId("todo-item")).toHaveLength(2);
    });

    expect(screen.getByText("Existing Todo 1")).toBeInTheDocument();
  });

  it("adds a new todo when Add button is clicked", async () ⇒ {
    const mockInitial = [{ id: 1, title: "Initial Todo" }];
    const mockNewTodo = { id: 99, title: "New Todo Item" };

    api.get.mockResolvedValueOnce({ data: mockInitial });
    api.post.mockResolvedValueOnce({ data: mockNewTodo });

    render(<TodoList />);

    await waitFor(() ⇒ {
      expect(screen.getByText("Initial Todo")).toBeInTheDocument();
    });

    // Type new todo
    fireEvent.change(screen.getByTestId("todo-input"), {
      target: { value: "New Todo Item" },
    });

    fireEvent.click(screen.getByTestId("add-btn"));

    // POST request called with correct data
    expect(api.post).toHaveBeenCalledWith("/todos", {
      title: "New Todo Item",
      completed: false,
    });

    // Wait for new item to render
    await waitFor(() ⇒ {
      expect(screen.getByText("New Todo Item")).toBeInTheDocument();
    });
  });
});
```