**Lesson 07 Demo 08**

**Optimizing Test Execution Using GenAI**

**Objective:** To optimize test execution by automating the generation and interpretation of test scripts for improving the functionality and user experience of online library search features

**Tools required:** ChatGPT 4

**Prerequisites:** None

Steps to be followed:

1. Generate test scripts for key functionalities using ChatGPT
2. Interpret test results using ChatGPT

**Note:** Please be advised that ChatGPT, as an artificial intelligence tool, can produce varied outputs even when presented with similar prompts.

**Step 1:** **Generate test scripts for key functionalities using ChatGPT**

* 1. Browse to the **https://chat.openai.com** website and log in to your account  
       
     A screenshot of a computer

     Description automatically generated

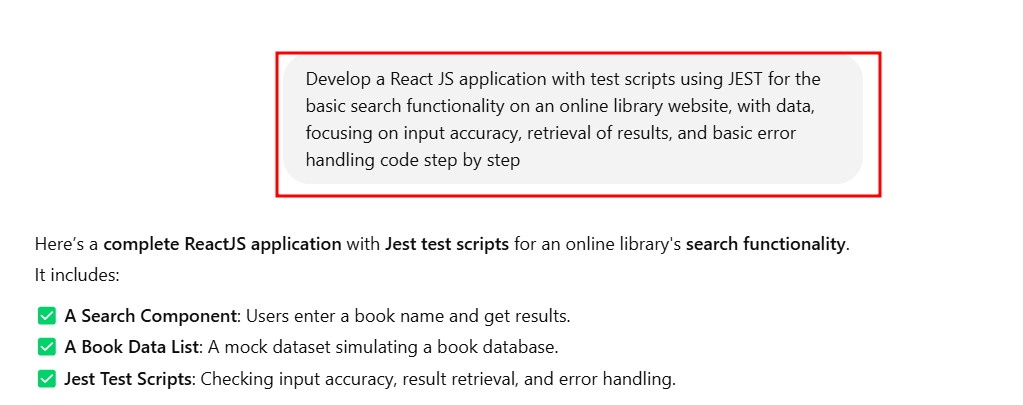
|  |
| --- |
| **Note**: Sign up if you do not have an account |

A screenshot of a login form

Description automatically generated

* 1. Use the following prompt to generate test scripts, as shown in the screenshot below:

**Develop a React JS application with test scripts using JEST for the basic search functionality on an online library website, with data, focusing on input accuracy, retrieval of results, and basic error handling code step by step**

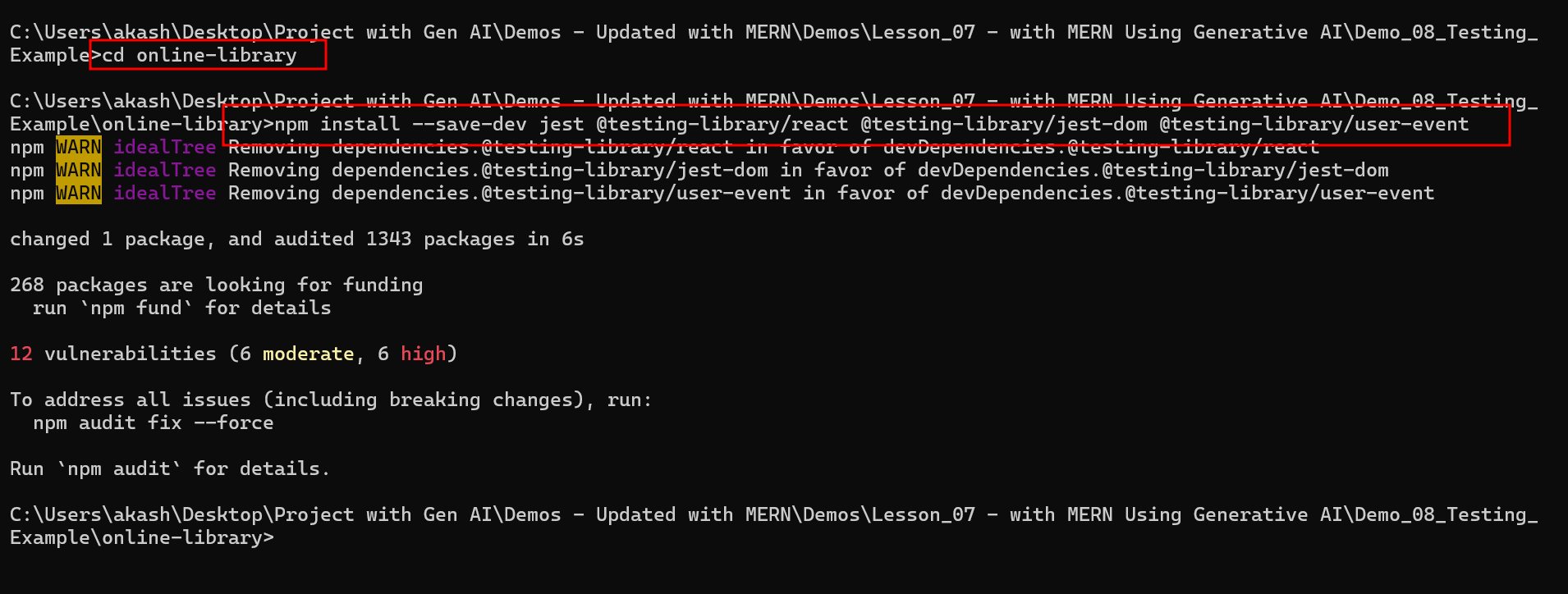


**Setup the React Project**

**npx create-react-app@latest online-library**

**cd online-library**

**npm install --save-dev jest @testing-library/react @testing-library/jest-dom @testing-library/user-event**



Open the project in VS Code

**books.js**

// src/data/books.js

const books = [

{ id: 1, title: "Harry Potter and the Sorcerer's Stone" },

{ id: 2, title: "The Hobbit" },

{ id: 3, title: "The Great Gatsby" },

{ id: 4, title: "Moby Dick" },

];

export default books;

**SearchBar.js**

// src/components/SearchBar.js

import React, { useState } from "react";

import books from "./books";

const SearchBar = () => {

  const [query, setQuery] = useState("");

  const [results, setResults] = useState([]);

  const handleSearch = () => {

    if (query.trim() === "") {

      alert("Please enter a search term.");

      return;

    }

    const filteredBooks = books.filter((book) =>

      book.title.toLowerCase().includes(query.toLowerCase())

    );

    setResults(filteredBooks);

  };

  return (

    <div>

      <input

        type="text"

        placeholder="Search for books..."

        value={query}

        onChange={(e) => setQuery(e.target.value)}

        data-testid="search-input"

      />

      <button onClick={handleSearch} data-testid="search-button">Search</button>

      <ul data-testid="search-results">

        {results.length > 0 ? (

          results.map((book) => <li key={book.id}>{book.title}</li>)

        ) : (

          query && <li>No results found</li>

        )}

      </ul>

    </div>

  );

};

export default SearchBar;

**SearchBar.test.js**

// src/tests/SearchBar.test.js

import React from "react";

import { render, screen, fireEvent } from "@testing-library/react";

import "@testing-library/jest-dom";

import userEvent from "@testing-library/user-event";

import SearchBar from "./SearchBar";

describe("SearchBar Component", () => {

  test("renders input and button correctly", () => {

    render(<SearchBar />);

    expect(screen.getByTestId("search-input")).toBeInTheDocument();

    expect(screen.getByTestId("search-button")).toBeInTheDocument();

  });

  test("updates input value correctly", async () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    await userEvent.type(input, "Harry Potter");

    expect(input).toHaveValue("Harry Potter");

  });

  test("retrieves correct search results", () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "Hobbit" } });

    fireEvent.click(button);

    expect(screen.getByTestId("search-results")).toHaveTextContent("The Hobbit");

  });

  test("shows 'No results found' for unmatched search", () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "Nonexistent Book" } });

    fireEvent.click(button);

    expect(screen.getByTestId("search-results")).toHaveTextContent("No results found");

  });

  test("shows alert when searching with an empty input", () => {

    window.alert = jest.fn();

    render(<SearchBar />);

    const button = screen.getByTestId("search-button");

    fireEvent.click(button);

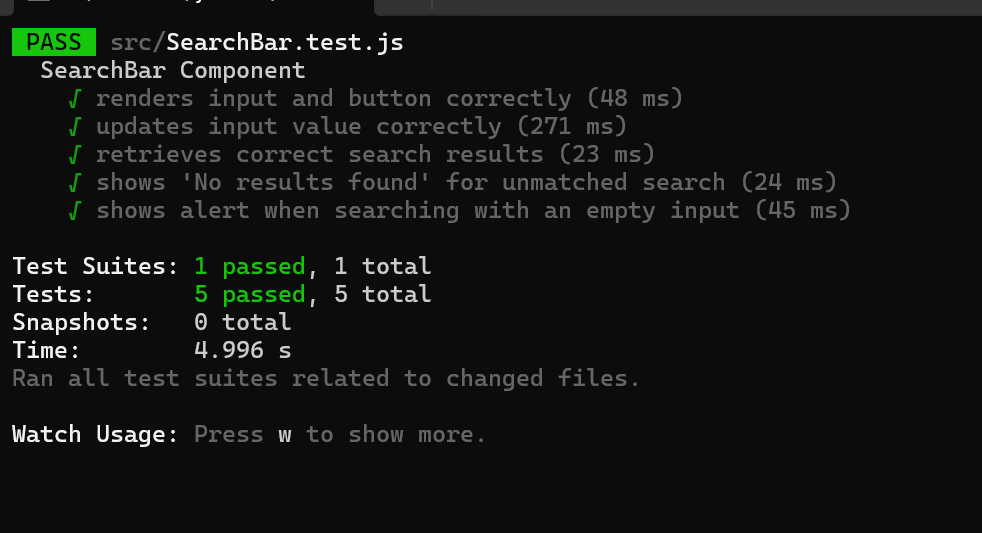
    expect(window.alert).toHaveBeenCalledWith("Please enter a search term.");

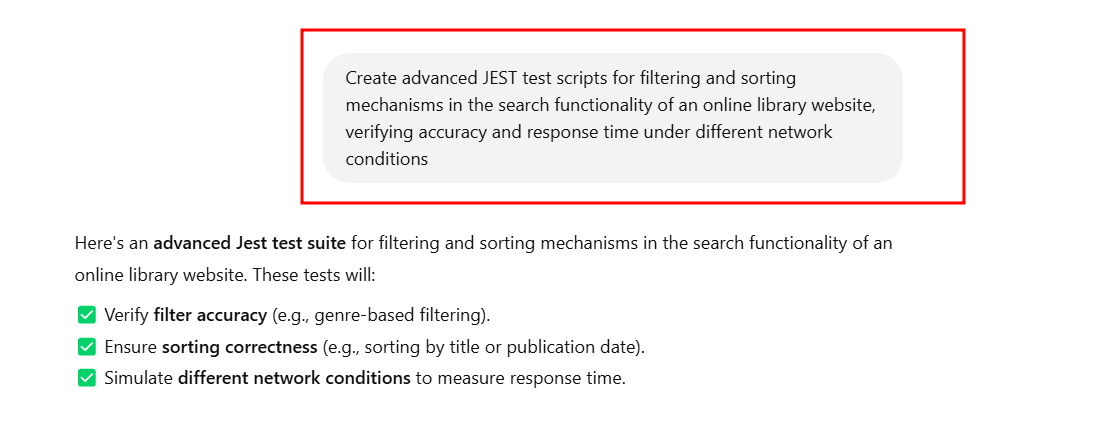
  });

});

**Run the testing file**

**npm test**



* 1. Use the prompt to generate advanced Selenium search scripts with multifunctionality, as shown in the screenshot below:  
     **Create advanced JEST test scripts for filtering and sorting mechanisms in the search functionality of an online library website, verifying accuracy and response time under different network conditions**  
       
     

**Modify existing SearchBar.js file code**

import React, { useState } from "react";

import books from "./books";

const SearchBar = () => {

const [query, setQuery] = useState("");

const [genre, setGenre] = useState("All");

const [sortBy, setSortBy] = useState("title");

const [results, setResults] = useState([]);

const handleSearch = () => {

if (query.trim() === "") {

alert("Please enter a search term.");

return;

}

let filteredBooks = books.filter((book) =>

book.title.toLowerCase().includes(query.toLowerCase())

);

if (genre !== "All") {

filteredBooks = filteredBooks.filter((book) => book.genre === genre);

}

filteredBooks.sort((a, b) =>

sortBy === "title"

? a.title.localeCompare(b.title)

: a.publicationYear - b.publicationYear

);

setResults(filteredBooks);

};

return (

<div>

<input

type="text"

placeholder="Search for books..."

value={query}

onChange={(e) => setQuery(e.target.value)}

data-testid="search-input"

/>

<select

value={genre}

onChange={(e) => setGenre(e.target.value)}

data-testid="genre-filter"

>

<option value="All">All Genres</option>

<option value="Fantasy">Fantasy</option>

<option value="Fiction">Fiction</option>

</select>

<select

value={sortBy}

onChange={(e) => setSortBy(e.target.value)}

data-testid="sort-dropdown"

>

<option value="title">Sort by Title</option>

<option value="year">Sort by Year</option>

</select>

<button onClick={handleSearch} data-testid="search-button">

Search

</button>

<ul data-testid="search-results">

{results.length > 0 ? (

results.map((book) => (

<li key={book.id}>{`${book.title} (${book.publicationYear})`}</li>

))

) : (

query && <li>No results found</li>

)}

</ul>

</div>

);

};

export default SearchBar;

**modified books.js code**

const books = [

{ id: 1, title: "Harry Potter", genre: "Fantasy", publicationYear: 1997 },

{ id: 2, title: "The Hobbit", genre: "Fantasy", publicationYear: 1937 },

{ id: 3, title: "The Great Gatsby", genre: "Fiction", publicationYear: 1925 },

{ id: 4, title: "Moby Dick", genre: "Fiction", publicationYear: 1851 },

];

export default books;

**Advanced Jest Test Script**

This test suite:  
✅ **Filters** by genre.  
✅ **Sorts** by title and year.  
✅ **Measures search response time** under different network conditions.

New **searchBar.test.js** file

import React from "react";

import { render, screen, fireEvent, waitFor } from "@testing-library/react";

import "@testing-library/jest-dom";

import userEvent from "@testing-library/user-event";

import SearchBar from "./SearchBar";

describe("SearchBar Component - Filtering & Sorting", () => {

  test("filters books by genre", () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const genreDropdown = screen.getByTestId("genre-filter");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "Harry" } });

    fireEvent.change(genreDropdown, { target: { value: "Fantasy" } });

    fireEvent.click(button);

    expect(screen.getByTestId("search-results")).toHaveTextContent("Harry Potter");

    expect(screen.getByTestId("search-results")).not.toHaveTextContent("The Great Gatsby");

  });

  test("sorts books by title", () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const sortDropdown = screen.getByTestId("sort-dropdown");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "The" } });

    fireEvent.change(sortDropdown, { target: { value: "title" } });

    fireEvent.click(button);

    const results = screen.getByTestId("search-results");

    const listItems = results.querySelectorAll("li");

    expect(listItems[0]).toHaveTextContent("The Great Gatsby");

    expect(listItems[1]).toHaveTextContent("The Hobbit");

  });

  test("sorts books by publication year", () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const sortDropdown = screen.getByTestId("sort-dropdown");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "The" } });

    fireEvent.change(sortDropdown, { target: { value: "year" } });

    fireEvent.click(button);

    const results = screen.getByTestId("search-results");

    const listItems = results.querySelectorAll("li");

    expect(listItems[0]).toHaveTextContent("The Great Gatsby (1925)");

    expect(listItems[1]).toHaveTextContent("The Hobbit (1937)");

  });

  test("shows 'No results found' for unmatched search", () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "Nonexistent Book" } });

    fireEvent.click(button);

    expect(screen.getByTestId("search-results")).toHaveTextContent("No results found");

  });

});

describe("Search Performance Under Network Conditions", () => {

  test("measures response time under normal conditions", async () => {

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "Harry Potter" } });

    const startTime = performance.now();

    fireEvent.click(button);

    await waitFor(() => screen.getByText("Harry Potter (1997)"));

    const endTime = performance.now();

    const responseTime = endTime - startTime;

    console.log("Normal Network Response Time:", responseTime, "ms");

    expect(responseTime).toBeLessThan(500);

  });

  test("simulates slow network response", async () => {

    jest.useFakeTimers();

    render(<SearchBar />);

    const input = screen.getByTestId("search-input");

    const button = screen.getByTestId("search-button");

    fireEvent.change(input, { target: { value: "Harry Potter" } });

    const startTime = performance.now();

    fireEvent.click(button);

    jest.advanceTimersByTime(2000); // Simulate 2s delay

    await waitFor(() => screen.getByText("Harry Potter (1997)"));

    const endTime = performance.now();

    const responseTime = endTime - startTime;

    console.log("Slow Network Response Time:", responseTime, "ms");

    expect(responseTime).toBeGreaterThanOrEqual(2000);

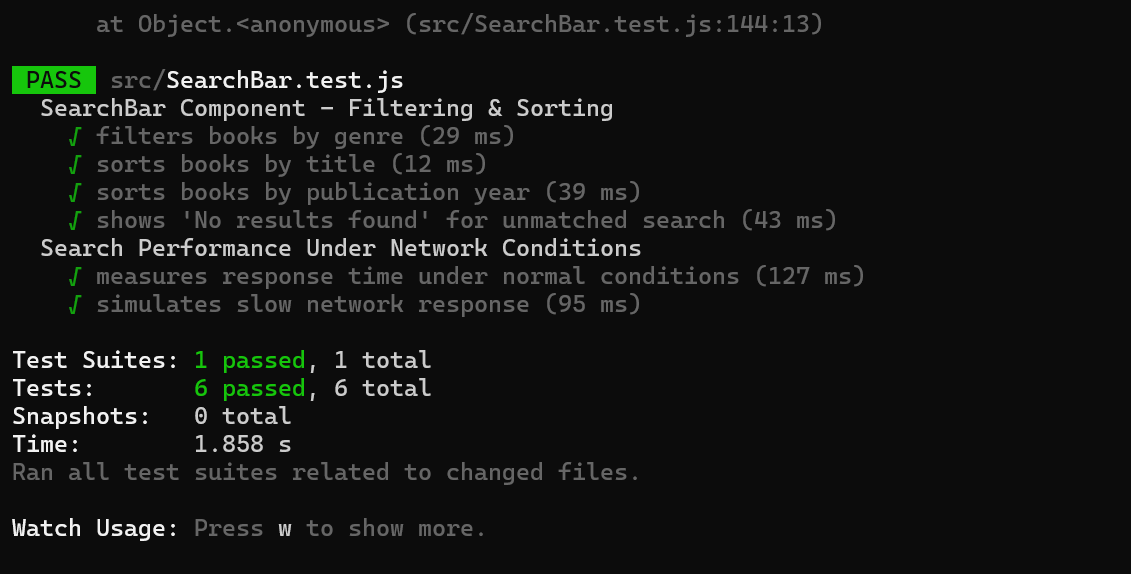
    jest.useRealTimers();

  });

});

After run the application using

**npm test**



By following these steps, you have successfully optimized test execution by using ChatGPT to automate the generation and interpretation of test scripts to enhance both functionality and user experience for online library search features.