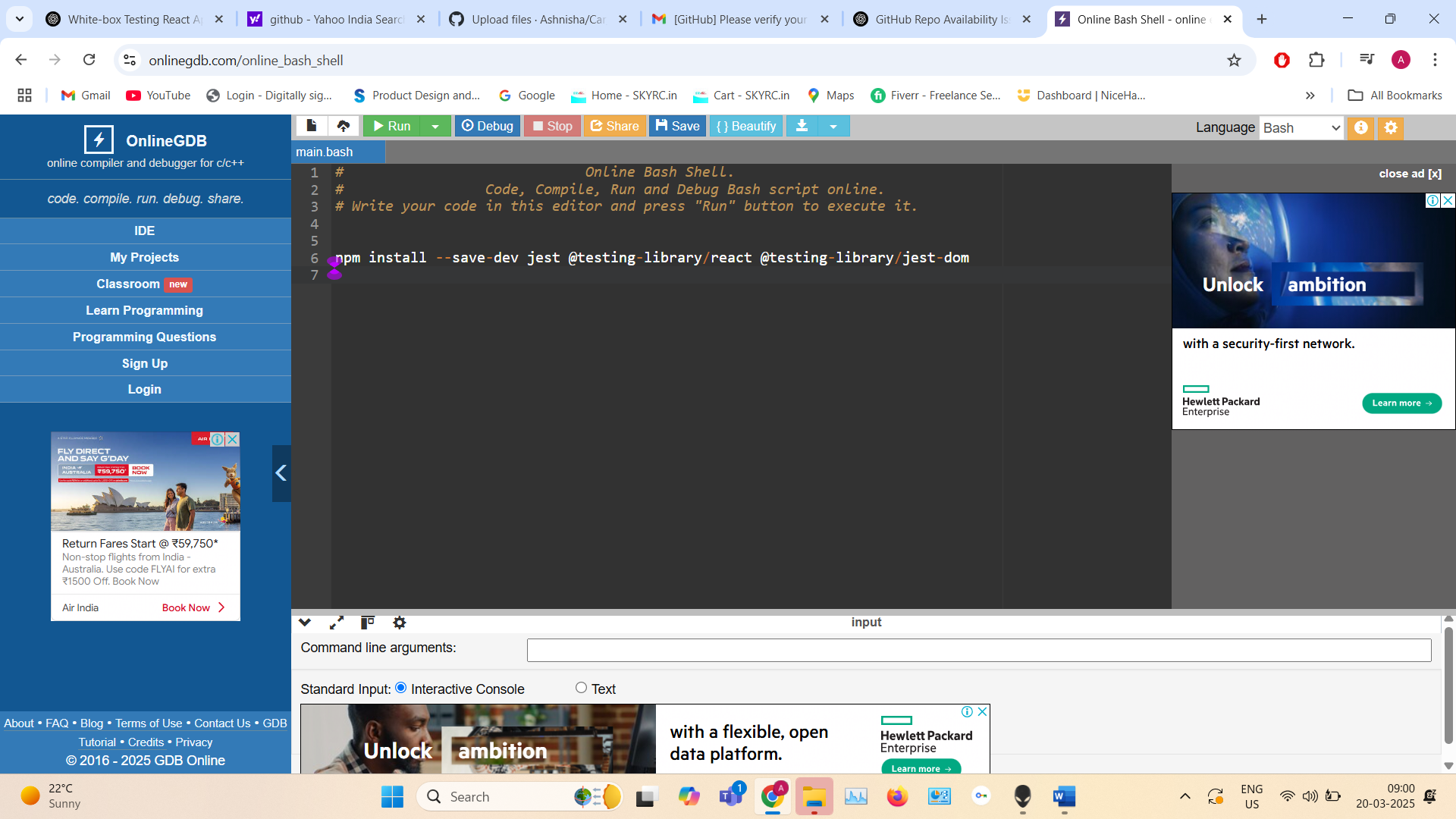
**Check-Points for White Box testing the react – app using Jest Framework and RTL(React testing library) .**

**✅ Step 1: Environment Setup**

1. **Install Jest and RTL:**  
   Ensure your project has Jest and React Testing Library installed using npm or yarn:

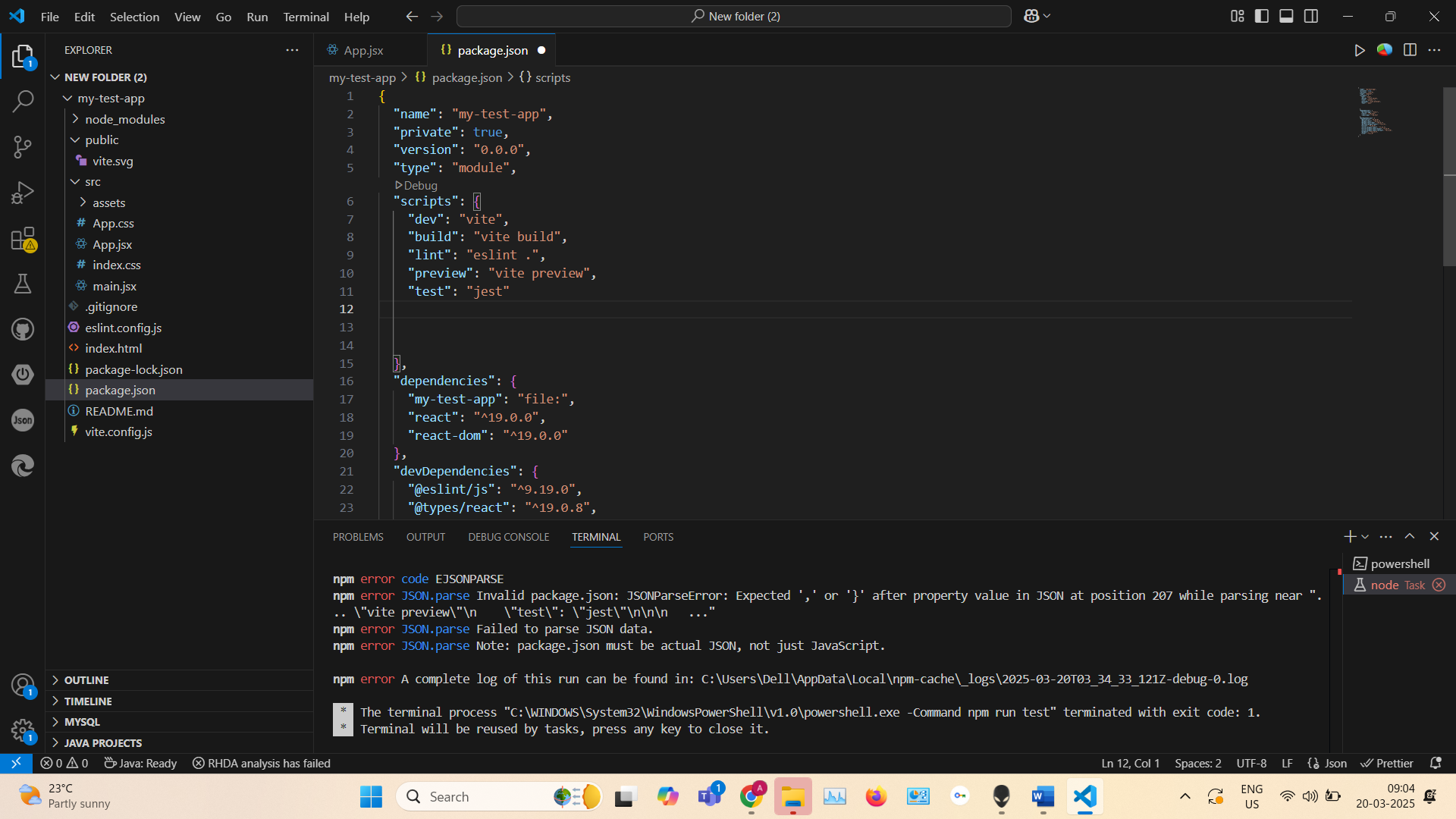
Bash =>



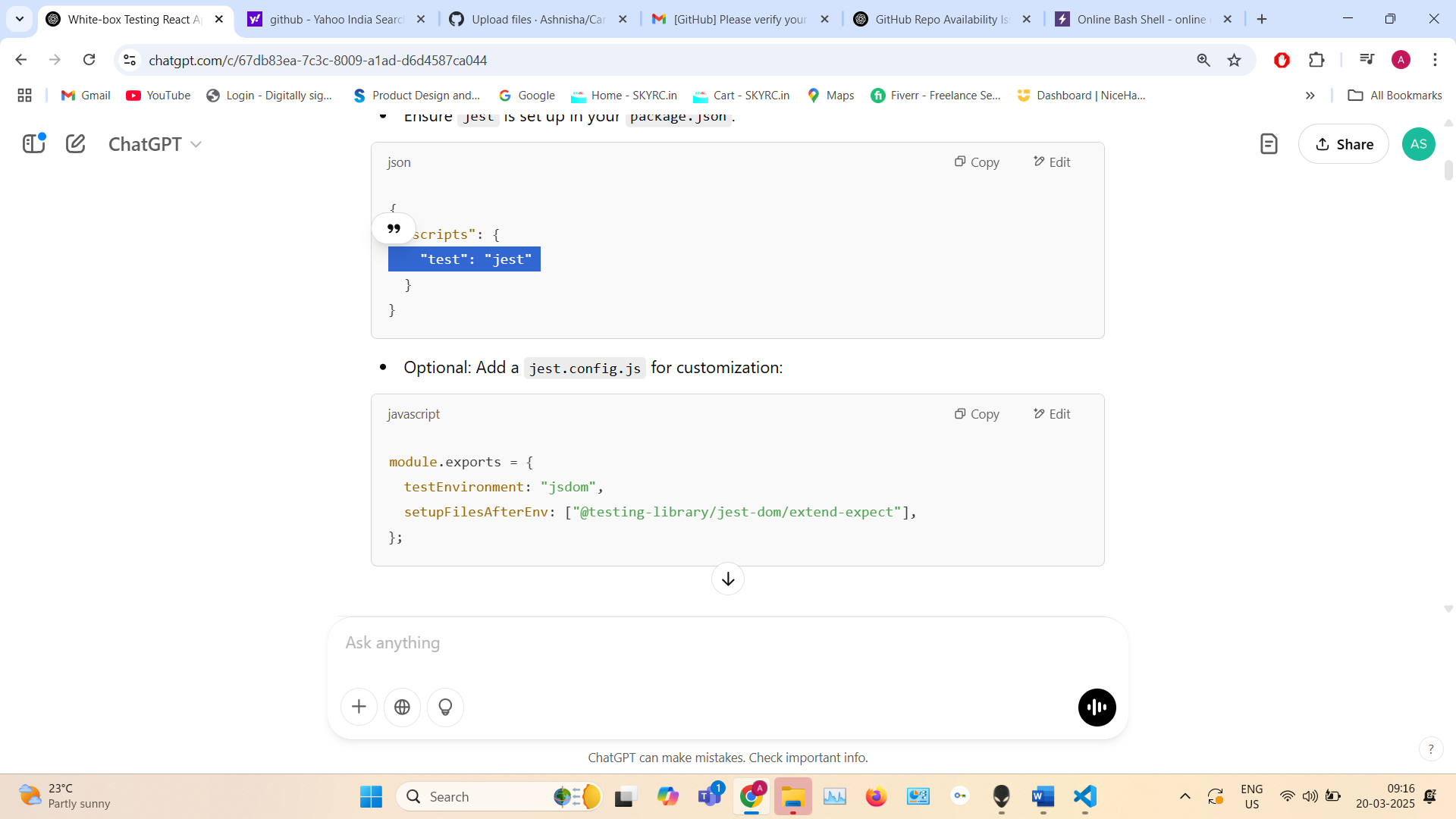
**npm install --save-dev jest @testing-library/react @testing-library/jest-dom**

1. **Configure Jest:**
   * **Ensure jest is set up in your package.json:**

**Json -**



* + **Optional: Add a jest.config.js for customization:**



module.exports = {

testEnvironment: "jsdom",

setupFilesAfterEnv: ["@testing-library/jest-dom/extend-expect"],

};

**✅ Step 2: Test File Structure**

* Will Follow a common convention by creating a \_\_tests\_\_ folder or adding .test.js or .spec.js files alongside components:

src/

├── components/

│ ├── Button.js

│ ├── Button.test.js

│ └── Header.test.js

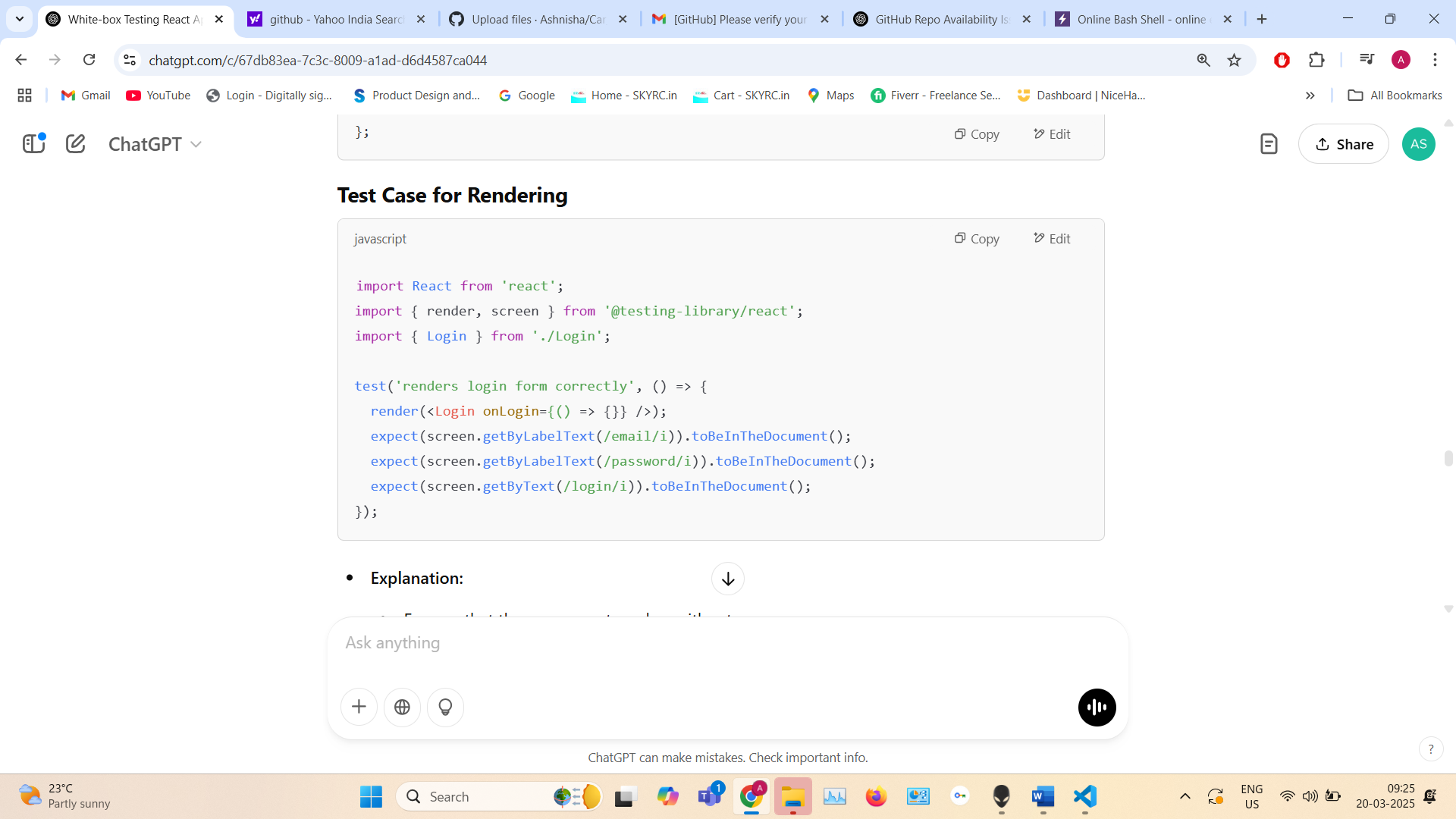
└── \_\_tests\_\_/

├── App.test.js

**✅ Step 3: Write Unit Test Cases**

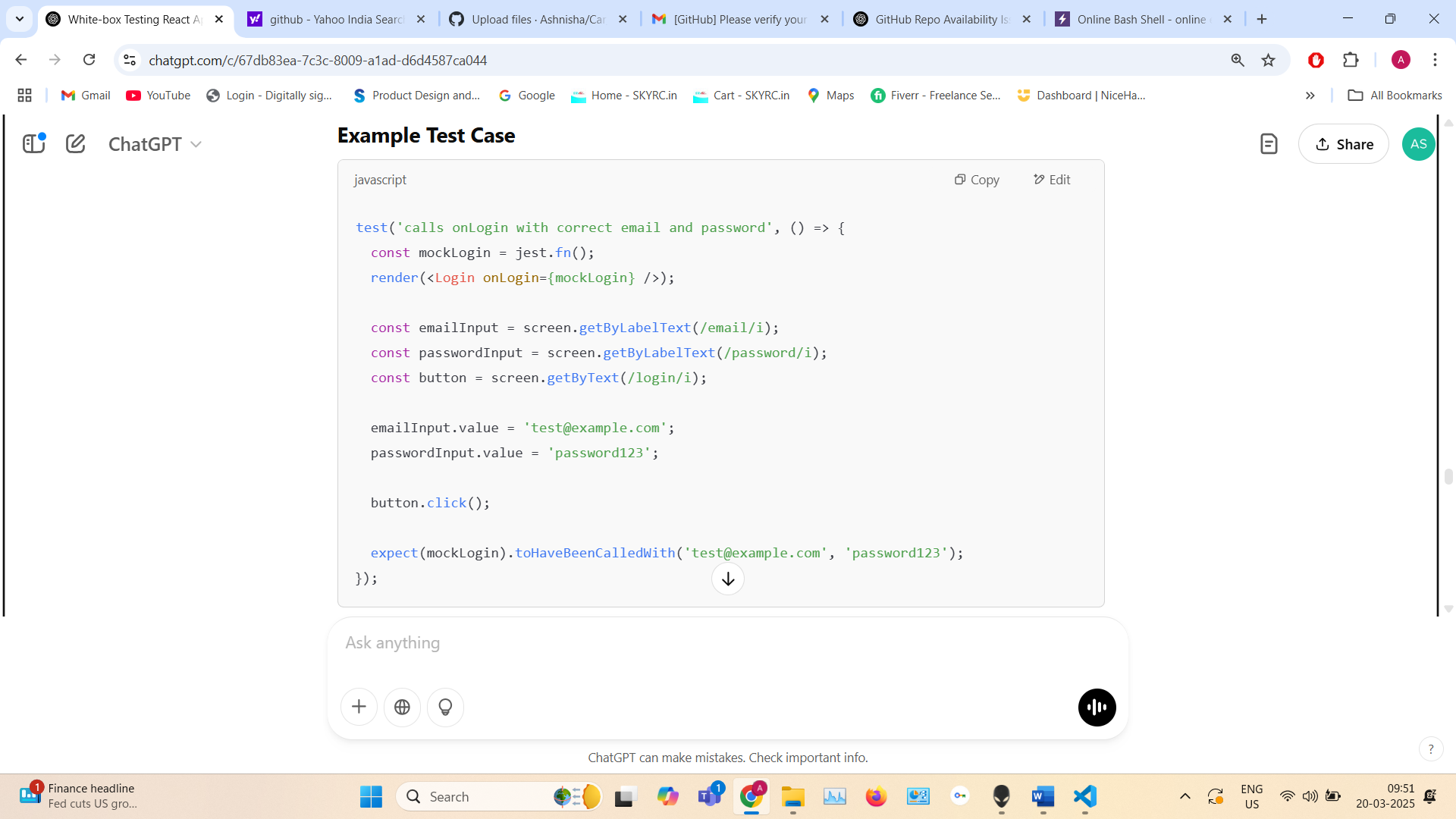
Here are common unit test scenarios for React components using Jest and RTL:

* **Rendering Components**
  + Ensure the component renders without crashing.
  + **Example for Test Case for Rendering =>**



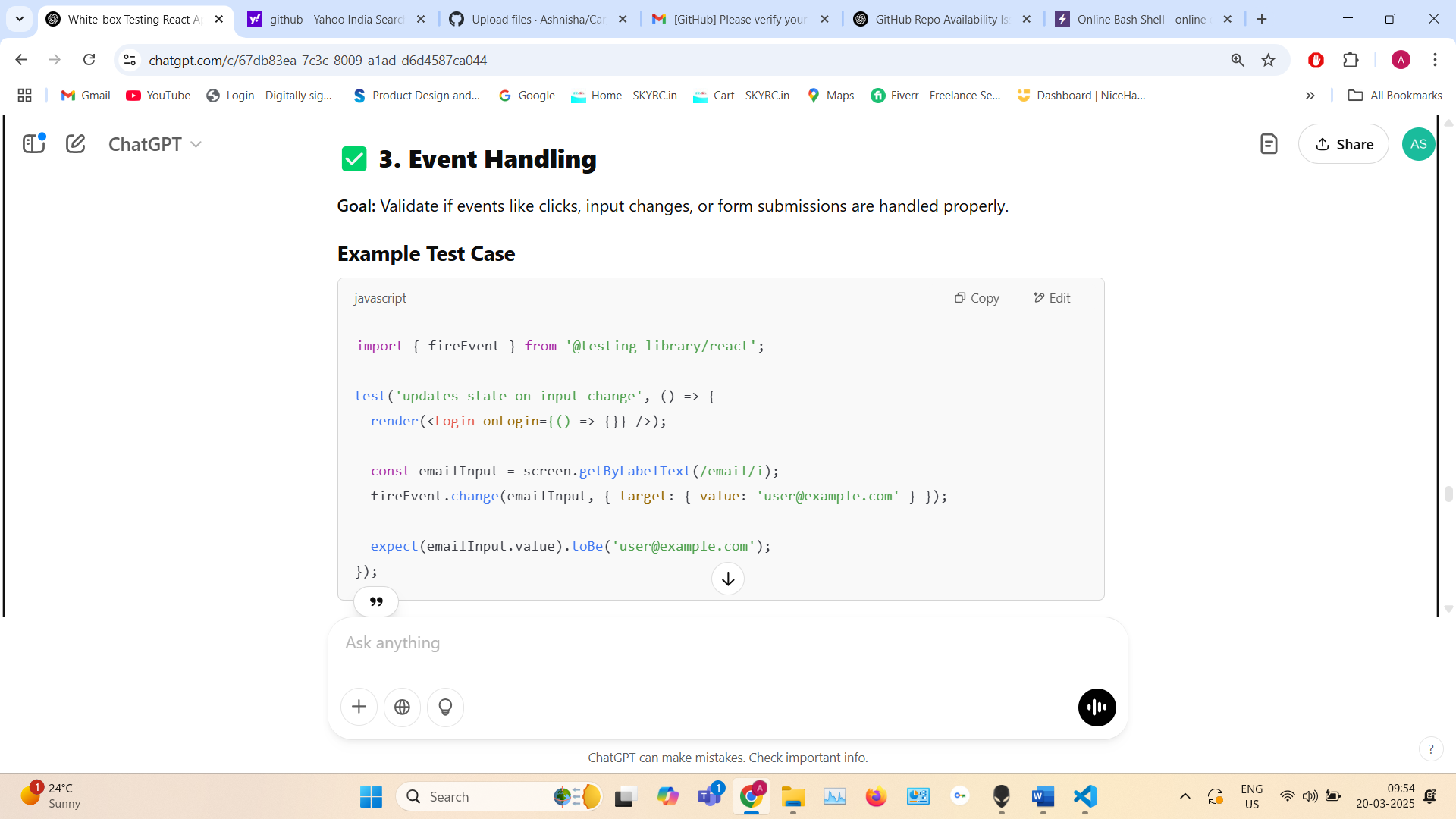
**Explanation:**

* Ensures that the component renders without errors.
* Verifies all form elements (email, password, and button) are visible.
* **Props Validation**
  + Test if the component correctly renders based on different props.

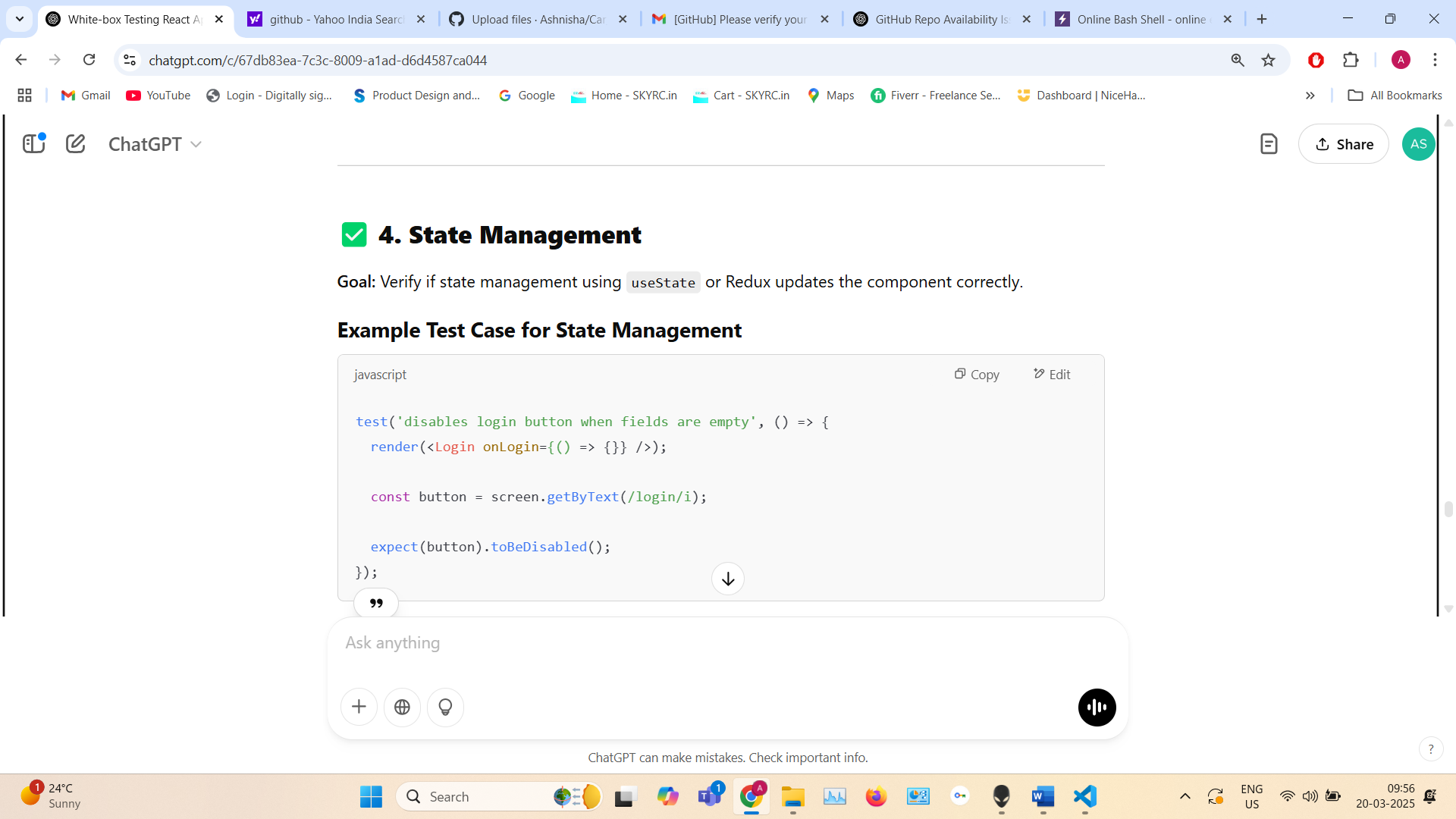


**Explanation:**

* Simulates entering email and password.
* Checks if the onLogin prop is called with correct values.
* **Event Handling**
  + Check if click, input, or submit events work as expected.

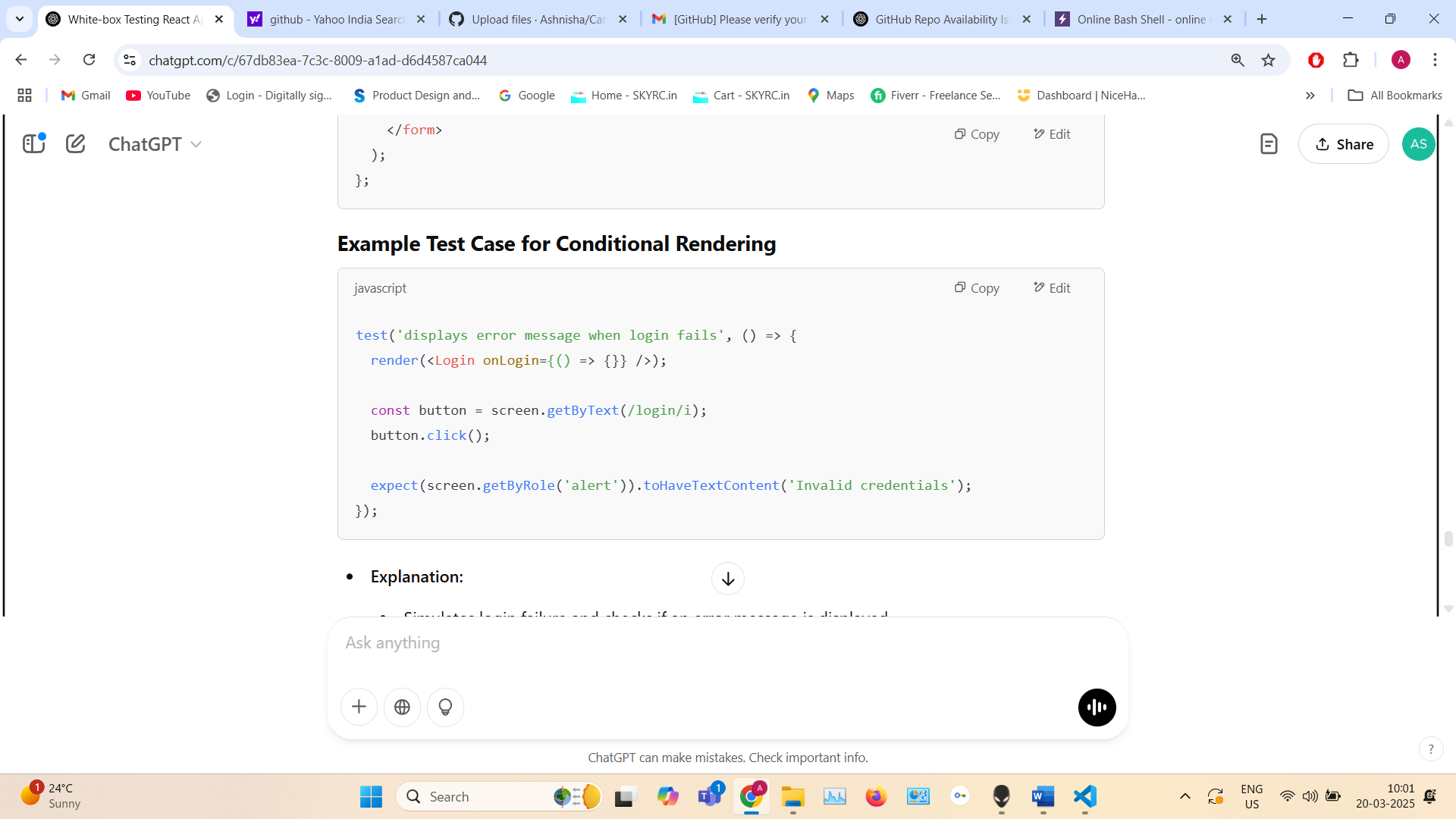


**Explanation:**

* Simulates an input change.
* Validates state update using fireEvent.change.
* **State Management**
  + Validate component behavior on state change using useState or Redux.
* 

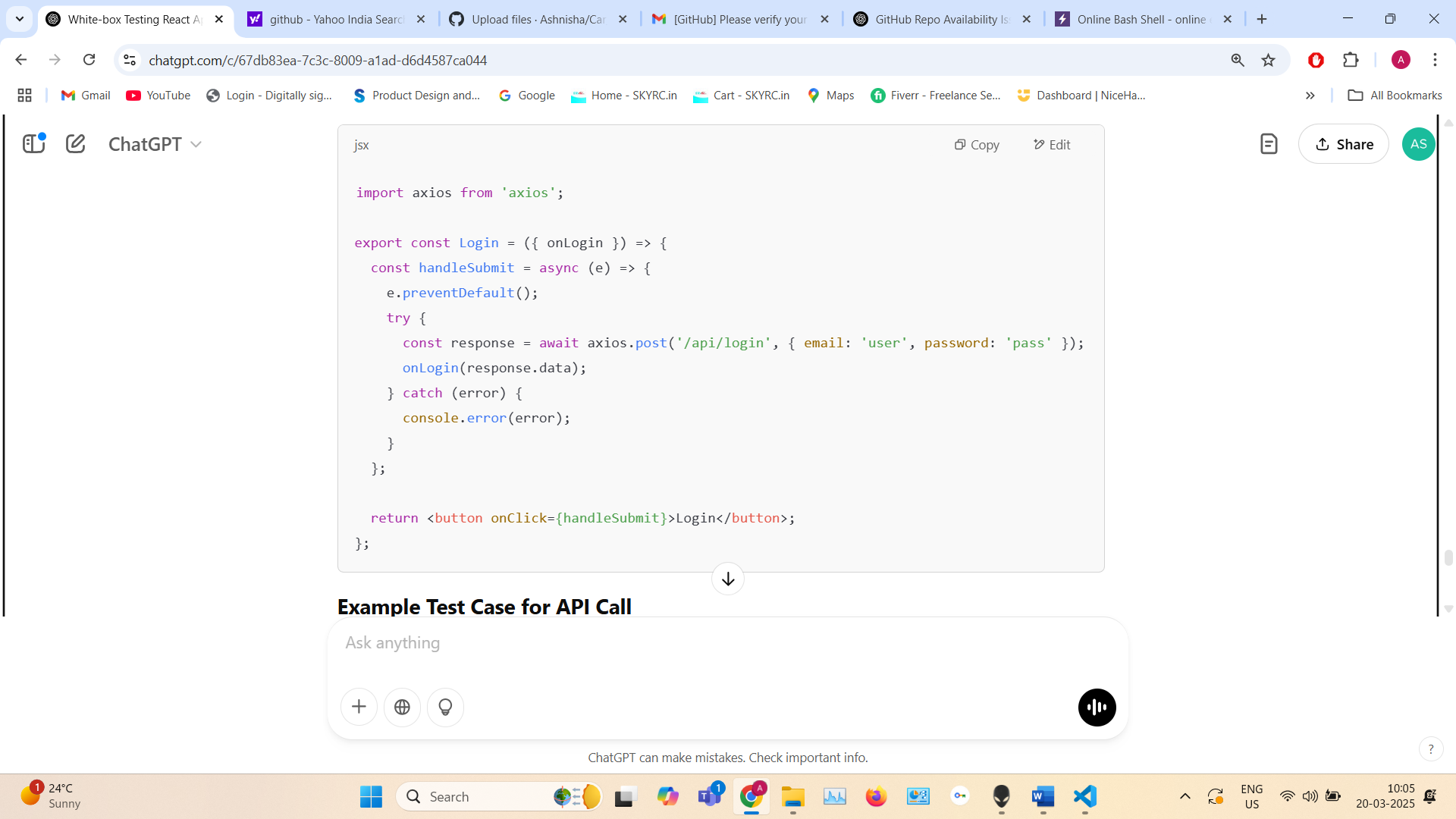
**Explanation:**

* Checks if the button remains disabled if fields are empty.
* You can add conditions to disable buttons using state values.
* **Conditional Rendering**
  + Verify correct UI for different conditional states.

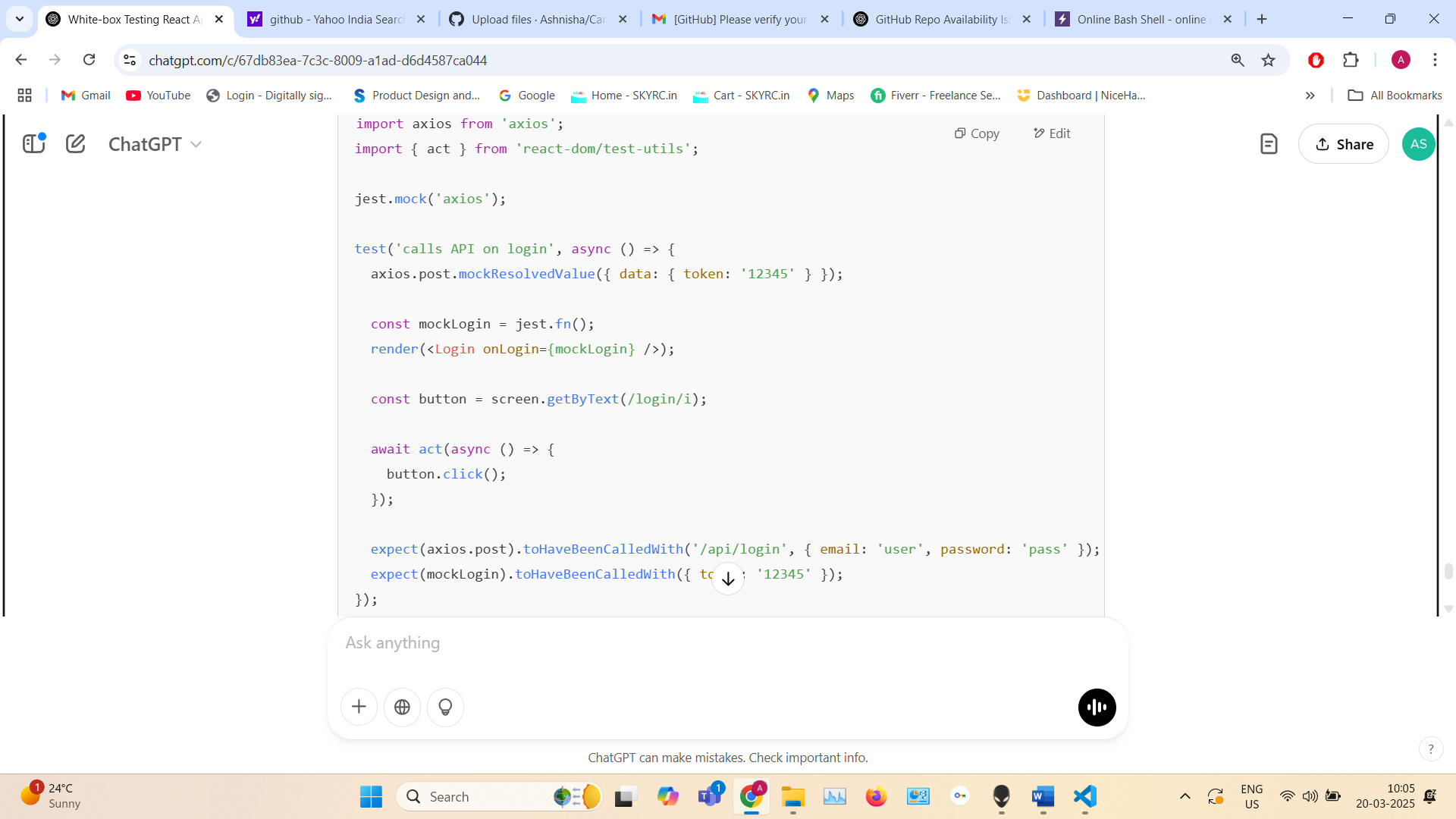


* **Explanation:**
  + Simulates login failure and checks if an error message is displayed.
* **API Calls or Side Effects**
  + Mock APIs using jest.fn() or jest.mock().

**Login component with API call react code =>**



**Unit Test case for it ->**



**Explanation:**

* Mocks API using jest.mock().
* Validates the API is called with correct parameters.
* Checks if the response is handled correctly using onLogin().

**✅ Final Requirements for White-Box Testing in a React App Using Jest and React Testing Library**

To set up your environment and perform white-box testing, you will need:

1. **Node.js** (Includes npm)
   * npm will be installed automatically with Node.js
2. **VS Code**
   * Install **Visual Studio Code** for development and testing.
3. **React App**
   * Create a React app using:

npx create-react-app my-app

cd my-app

1. **Install Testing Tools**  
   From the VS Code terminal, run the following:

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

1. **Optional Installations (If Needed)**
   * **Babel** (For ES6+ support):

npm install --save-dev babel-jest @babel/core @babel/preset-env @babel/preset-react

* + **Axios** (If testing API calls):

npm install axios

1. **Run Tests**
   * After writing your test cases, run them using:

npm test