To write unit test cases for the CustomRtf StencilJS component using TinyMCE, you can follow these steps. Since TinyMCE involves complex behavior with DOM manipulation and events, you'll need to mock TinyMCE and test the component's interaction with it.

**1. Set Up Testing Environment**

Ensure your StencilJS project is set up with Jest for unit testing. Jest usually comes pre-configured in Stencil projects.

**2. Mock TinyMCE**

Create a mock for TinyMCE so that you don't actually initialize the editor in your tests. The mock should simulate TinyMCE's methods and events. you need to include the \_\_mocks\_\_/tinymce.js file in your project if you're mocking TinyMCE for your tests. This file contains the mock implementation of TinyMCE, which ensures that your unit tests run without actually initializing the TinyMCE editor, making your tests faster and more isolated.

In your **\_\_mocks\_\_/tinymce.js** file:

const tinymce = {

init: jest.fn(),

remove: jest.fn(),

triggerSave: jest.fn(),

get: jest.fn().mockReturnValue({

setContent: jest.fn(),

getContent: jest.fn().mockReturnValue(''),

on: jest.fn(),

mode: {

set: jest.fn(),

},

}),

};

export default tinymce;

**3. Write the Unit Tests**

You can now write the unit tests using Jest. Below are example test cases for different scenarios:

**Example Test Cases**

import { newSpecPage } from '@stencil/core/testing';

import { CustomRtf } from './custom-rtf'

**Explanation of Test Cases:**

1. **Initialization of TinyMCE**: Checks that TinyMCE is initialized when the component loads, and verifies the initial content.
2. **Prop Change Handling**: Tests that when the initialvalue prop changes, the editor's content is updated.
3. **Handling Content Change Events**: Verifies that content changes in TinyMCE are detected and emitted by the contentChanged event.
4. **Cleanup on Component Unload**: Ensures that TinyMCE is properly removed when the component is unloaded.
5. **Emitting valueChange Event**: Tests that the valueChange event is emitted when the editor's content changes.
6. **Read-Only Mode Handling**: Verifies that the editor is set to read-only mode when the disabled prop is true.

**4. Run Your Tests**

Run your tests using the following command:

npm test

**Mock importance:**

Here's why it's important:

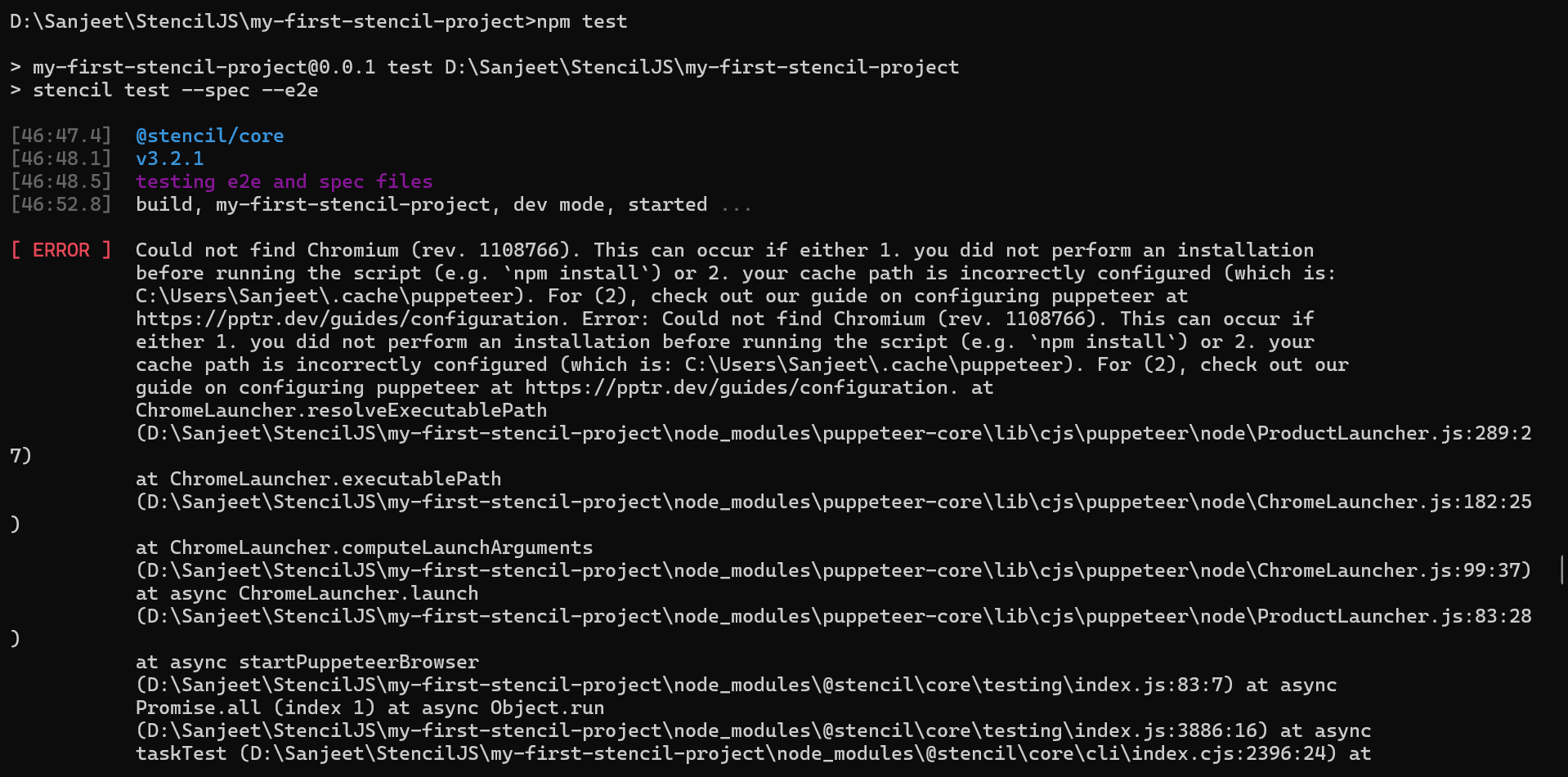
1. **Isolation of Tests**: By mocking TinyMCE, you isolate your tests from external dependencies and prevent them from relying on the actual TinyMCE library. This is crucial because the real TinyMCE library interacts with the DOM and can introduce flakiness in your tests.
2. **Faster Tests**: Mocking removes the overhead of initializing the full TinyMCE editor, which can be slow. Your tests will run faster since they only simulate the behavior of TinyMCE rather than loading the entire library.
3. **Control Over Behavior**: The mock file allows you to control the behavior of TinyMCE during tests. You can define how methods like init, getContent, and setContent should behave, which is useful for testing different scenarios in your component.

**Where to Place the Mock File**

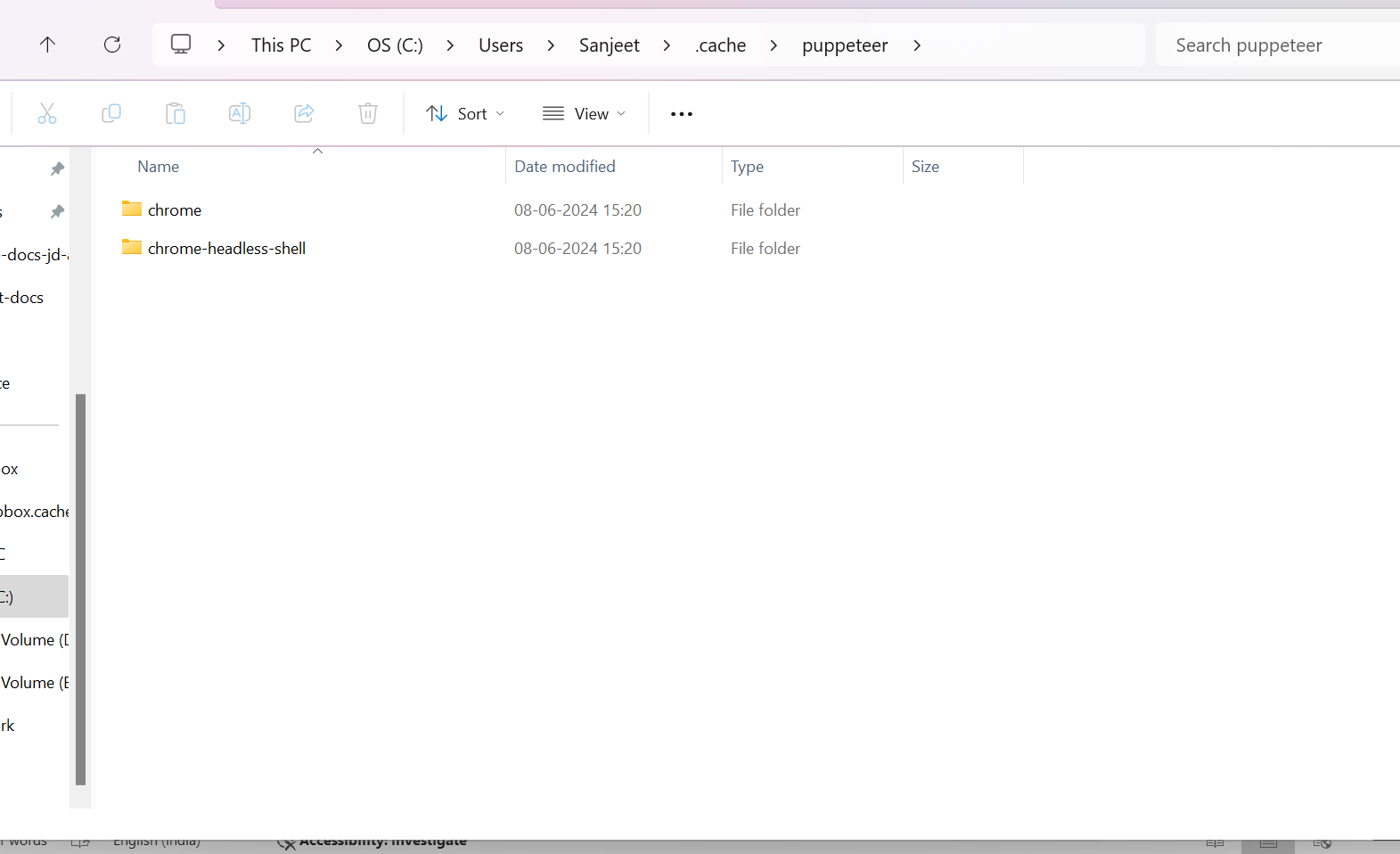
* Place the tinymce.js mock file under the \_\_mocks\_\_ directory in the root of your project (or within the same directory as your tests, depending on your project's structure).

**How Jest Uses the Mock**

* Jest automatically uses the mock when you import tinymce in your test files. This behavior is due to the convention of placing the mock file inside a directory named \_\_mocks\_\_.
* The mock will ensure that TinyMCE's behavior is simulated according to your test requirements, without actually invoking the real TinyMCE library.



C



First, find the path to your Chromium or Chrome installation. Then, set the PUPPETEER\_EXECUTABLE\_PATH environment variable to point to that path:

set PUPPETEER\_EXECUTABLE\_PATH="C:\path\to\your\chrome.exe"

C:\Users\Sanjeet\.cache\puppeteer\chrome\win64-121.0.6167.85\chrome-win64\chrome.exe

**6. Use a Different Chromium Version**

If the specific revision is problematic, you can try using a different version of Puppeteer or forcing Puppeteer to use a different Chromium revision by specifying it in your package.json:

"puppeteer": {

"chromium\_revision": "LATEST\_REVISION"

}

Replace LATEST\_REVISION with a known good revision.

**7. Running in CI Environment**

If you're running tests in a Continuous Integration (CI) environment, ensure that the environment is configured to allow Chromium to be downloaded or use a pre-installed version of Chromium.

**8. Using Playwright Instead of Puppeteer**

If you're having consistent issues with Puppeteer, consider switching to Playwright, which tends to be more reliable with headless browsers:

npm install @playwright/test

Playwright handles browser installations more robustly and supports multiple browsers.

**Summary**

The error indicates that the required version of Chromium wasn't found. Reinstalling dependencies, forcing a fresh download of Chromium, or pointing to a local Chromium installation should resolve the issue.