3. Create Your First CodeceptJS Test

Now it is time to create your first CodeceptJS end to end test, to do so we just run `npx codeceptjs gt .`

As filename we type in "editor", then press enter.

CodeceptJS is smart, it proposes "Editor" as feature name, which is fine for us, just press enter.

3.1 Scenario Description

Open the freshly created test and start by replacing "test something" with "Can be opened and the text input and bold formatting is working." as scenario description.

3.2 Test Steps

3.2.1 Add the First Test Step

Next we want to add the first test steps to the our scenario, all available WebDriver helper step actions can be found under https://codecept.io/helpers/WebDriver.

The first step is to navigate to the index page of our application under test.

To do so we add `I.amOnPage("/")` as first step in the scenario.

3.2.2 CodeceptJS "Built in Assertion"

Then we add `I.click("WYSIWYG Editor")` in the next line. This demonstrates an important concept of CodeceptJS. By clicking on a specific link text (here: "WYSIWYG Editor") we add something which I call a "built in assertion". The link must be available before CodeceptJS can click it, but there is not need to type out "wait for link to be available" explicitely. If the link would not be there, the test would automatically fail at this step.

3.2.3 Add More Test Steps

Because the content of the WYSISWYG editor itself is injected with an iFrame to the page, we need to switch to that iFrame to be able to interact with the contents of

the editor.

Almost every test step in end to end test automation for the web is based on locators. To find a proper locator I always open the Google Chrome developer tools and inspect the available elements.

In our use case the iframe has a nice id attribute, which is `#mce_0_ifr` .

To switch the context to the iFrame in the test, we write `I.switchTo("#mce_0_ifr")` as the next test step.

Next is to fill the text area with some text, we inspect the page again with Google Chrome dev tools and find `#tinymce` as possible locator.

Then we add the test step `I.fillField("#tinymce", "My text bold")` .

To select the last word of the text the test just filled in, we add `I.doubleClick("#tinymce")` as next test step.

To be able to click the "bold" button of the editor, we need to switch back from the iFrame context to the parent page, this is done by adding `I.switchTo()` to the next line.

The next steps is to find a locator for the bold icon. We use the dev tools again and find `#mceu_3` . Then to click the "bold" icon, we add `I.click("#mceu_3")` as next test step.

Now we want to get rid of the previous made text selection, so we switch back into the iFrame context by adding `I.switchTo("#mce_0_ifr")` and clicking once on the textarea by adding the step `I.click("#tinymce")` .

Last but not least, the test should save a screenshot of the current state of the editor to the previously configured "output" folder of CodeceptJS: `I.saveScreenshot("editor_test.png")` .

The complete `editor_test.js` should now look like

```
Feature("Editor");

Scenario(
  "Can be opened and the text input and bold formatting is workin
g",
```

```
  I => {
    I.amOnPage("/");
    I.click("WYSIWYG Editor");
    I.switchTo("#mce_0_ifr");
    I.fillField("#tinymce", "My text bold");
    I.doubleClick("#tinymce");
    I.switchTo();
    I.click("#mceu_3");
    I.switchTo("#mce_0_ifr");
    I.click("#tinymce");
    I.saveScreenshot("editor_test.png");
  }
);
```

You made it - your first automated e2e test with CodeceptJS is done, congrats for your progress, I really appreciate it!