Test Cafe - How to Write a Test

What is TestCafe?

TestCafe is a node.js tool used to automate end-to-end web testing. It works on all popular environments, includes support for many browsers, and has been very easy to use in our experience.

Refer to their homepage for further reading

> A node.js tool to automate end-to-end web testing | TestCafe
> https://devexpress.github.io/testcafe/

Getting Started

Basics

After you've installed our package using `npm i beer-library` you may want to write some tests to ensure your additions to the library are working properly. In that case, this tutorial is meant for you. First, some terminonology:

npm - package manager that is used to install packages as dependencies into your project

fixture - term used to describe where your tests will take place, think of a hanging rack fixture in a retail store, defining a fixture will tell TestCafe which url or filepath to go to in order to perfrom its tests

Installation

After installing our library, run `npm i --save-dev testcafe` to install testcafe onto your system, you may choose to install it globally by running `npm i -g testcafe`

Note: If you are a beginner, I recommend you install it globally

Next, add a *test* script into your *package.json* file, under the *scripts* object. This will be helpful when you want to run your tests, as you will only need to use the

command `npm run test` to start the magic that is TestCafe

```
// package.json
"scripts": { "test": "testcafe chrome tests/" }
```

Live Server

Unless you are going to be hosting your project on the internet, with the help of GithubSites or anything like that, local development will be your best friend here. You will need the ability to spawn a live development server on your LAN, on any port, but must be accessible via `localhost:<port-number>`. Some great resources to do so can be found here:

- https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer
- https://docs.python.org/2/library/simplehttpserver.html
- https://docs.python.org/3/library/http.server.html

Writing your First Test

Where will are your tests go?

In the root directory of your project, create a new directory names tests, we will place all of our code to run the tests in this directory. Navigate to the tests directory and create two files, `myFirstTest.js` and `testingPage.html`. myFirstTest will hold all the code used to run the tests, whereas testingPage will be an HTML page used to interact with the components you want to test.

myFirstTest.js

Every testing file written for TestCafe will need to specify a Fixture.page() combo. This just means that you will title your test suite for this particular file and specify where the tests will take place.

```
// myFirstTest.js

fixture `Getting Started`
        .page `localhost:5500/test/testingPage.html`;
```

Because we will be selecting nodes on the DOM in our html page testingPage.html, we need to import a framework to help us. Thankfully, testcafe provides just the tool, Selector

```
// myFirstTest.js

import {Selector} from "testcafe";
...
```

In order to write a test, we must use a test Method that takes in two parameters, a string that will name our test, and an async function that will perform all the necessary operations to perform the tests.

```
// myFirstTest.js
...
test("My First Test", async t => {
    const p = Selector(() => document.querySelector("#my-p-tag");

    await t
        .expect(p).notEql(undefined)
        .expect(link.hasAttribute("id")).eql(true)
});
```

Our test name is "My First Test", it Selects a tag that matches `<p id="my-p-tag">Some text</p>` , expects it to not be undefined, and expects it to have an id attribute.

At this point, you can run `npm run test` and see your first test pass!

```
richard@richard-ubuntu:~/School/Team11$ npm run test

> team11@1.0.0 test /home/richard/School/Team11
> testcafe chrome test

 Running tests in:
 - Chrome 75.0.3770 / Linux 0.0.0

 Getting Started
 ✓ My First Test


 1 passed (0s)
richard@richard-ubuntu:~/School/Team11$
```

Let's take a look at one of the tests we wrote for our notable `<beer-brand>` button

```
test("Disable button should not let the button do anything", async t => {
    const disableBeer = Selector("#disabled-beer");
    const box = await Selector("#number");
    await t
        .expect(box.value).eql("0")
        .click(disableBeer).click(disableBeer).click(disableBeer)
        .expect(box.value).eql("0");
});
```

Here, we have a `<beer-brand>` on our HTML page that includes the *disabled* attribute, which renders the button unusable until the attribute is removed. Therefore, performing a click on the element should have no effect on any other element on the page.

testingPage.html

In this page, you can add all your components and regular HTML native tags, `<p>, <a>, <ul>,...` and they can be tested

Here is the testingPage.html that I used for this tutorial

```
<!-- testingPage.html -->
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Testing Tutorial</title>
                <script src="">
</head>
<body>
    <p id="my-p-tag">My own p tag</p>
                <beer-brand>

</body>
</html>
```