COMP3006 – Full-Stack Development

Workshop 5 – Client-Side Testing

Autumn 2020

In this week's lecture you learned about testing in client-side web applications. In this workshop you will build some client-side unit tests that will test functions, objects and user interface components.

You will also extend your work on the mini project by writing some unit tests for the client-side software you have developed so far.

Before the Workshop

Exercise 1

In the first exercise your task is to unit test a function. The function should be called **sayHello**, and it should return a string. The string should be made of of two substrings – one is **"Hello"** and the other is stored in a variable called **name**, which is passed to the string as an argument.

Therefore, if the value of the **name** variable is "David", the function should return **"Hello David"**.

Download the template on the DLE and modify it to include:

1. A test **suite** containing a unit test that asserts that the function returns the correct value.
2. The **sayHello** function, so that the test runs and passes.

Exercise 2

Your next task is to take the function developed in Exercise 1 and use it to display the message in the page. You should get the name to include in the message from a text input. Include a button in the page with a **click** event handler that takes the text from the input, generates the message, and inserts it into the page.

Extend the template by adding a test to the suite so that:

1. A value is assigned to the text box.
2. The button is clicked.
3. The message is inserted into the page.

You will need to use the **clickElement** function, which is included in the template for you.

You also need to implement the function so that the test passes, and should include a **teardown** method to remove the values from the form and the message element.

During the workshop

Your task during the workshop is to implement a simple calculator and provide unit tests to ensure that it works correctly.

Exercise 3

Begin by adding the following HTML to the template:

```
<div id="calculator">
    <p id="result">0.00</p>
    <input id="number1" /><input id="number2" />
    <button id="btnAdd">Add</button>
    <button id="btnMul">Mul</button>
</div>
```

Then modify the CSS with the following lines:

```
#calculator { width: 150px; background-color: #DFDFDF; padding: 5px; }
#calculator input { width: 50px; float: left; margin: 5px; }
```

Create a suite for the calculator tests. You're going to need a **suiteSetup** method to setup the test data you will use in the tests. Start the suite as follows:

```
suite("Calculator suite", function() {

    suiteSetup(function() {
        self.x = [1, 2, 3, 4, 5];
        self.y = [2, 4, 6, 8, 10];
    });

});
```

Then add a test method, which:

1. Defines test data (adding together the data you defined in your **suiteSetup** method.
2. Loops over the test data and asserts that the **sum** function produces the correct answer for each pair of inputs.

Once you've written the test, add the **sum** function to the application and your test should pass.

Exercise 4

Repeat the steps taken in the previous exercise to ensure that the **product** function operates correctly. The function should multiply values rather than adding them.

You can still use **self.x** and **self.y** as your test data.

Exercise 5

The final part is to test the user interface. Begin by adding a test that checks the following attributes of the **result** paragraph:

- Is the **background** #ffffff?
- Is the **font-size** 20px?

Then implement the CSS needed to make the tests pass.

The next step is to check that the buttons work correctly. Beginning with the **sum** button, add a test method that:

1. Adds values to the two inputs.
2. Clicks the button.
3. Confirms that the correct result is shown in the page.

Again, use **clickElement** here.

To get the test to pass, implement the event handler that will populate the result with the correct value.

When the **sum** button is working and tested, repeat the process for the **multiply** button.

Finally, add a **teardown** method that will remove the values from the inputs and results paragraph when each test has run.

This is similar to the **suiteSetup** method above – look at the lecture slides to check the syntax.

Mini Project

This week you will extend the mini project by adding some unit tests to test the JavaScript code you've so far developed.

Exercise 6

Add some unit tests to confirm that a new message is added to the message history properly. Think carefully about what possibilities are – choose appropriate test cases to make sure you've fully tested the feature.

Are there any other tests you could include? What are they?

Extension task

Exercise 7

Extend the calculator exercise by adding in other common mathematical operations – you could include subtraction, division, raising to a power and rooting numbers, for example. The process will be the same as for the operators included earlier, but the process of writing the tests is good practice. Try to think of a more extensive range of tests that you could include.