Protractor

A Testing Tool for AngularJS

Presentation slides are available at
http://angelstam.github.io/SQAT-LAB3/

By
Johan Angelstam & Fredrik Rosenqvist
Harbin Institute of Technology (HIT)
04-2014

Contents

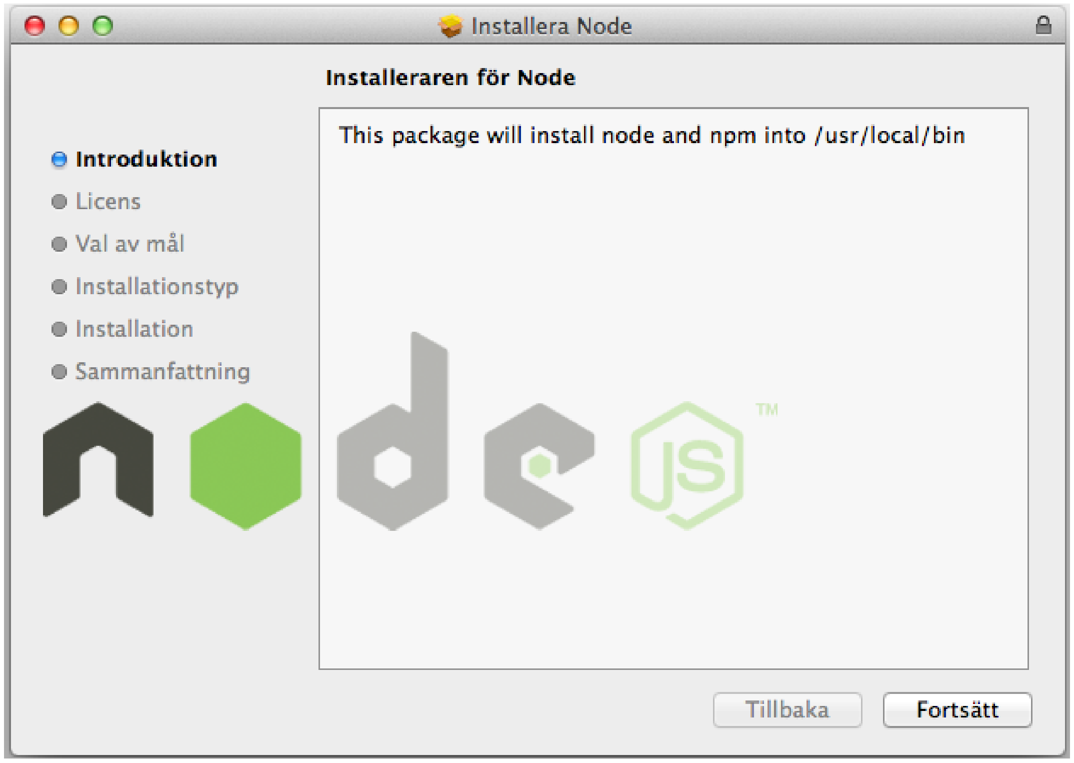
1. What is Protractor?

Protractor is a behavior-driven end to end testing framework mainly developed for testing web sites and web applications based on AngularJS. AngularJS is an open source framework for web development that was released by Google in 2010. The framework structures the web application into an MVC pattern with a designated view and controller. The AngularJS framework is making it very easy to communicate between the controller and the view of the web application, which is something that Protractor uses while testing [1].

Protractor, which also is an open source testing tool from Google, is based upon the Jasmine testing framework, but is specifically designed to be easily applicable on AngularJS web applications. Protractor, just like Jasmine is cross-platform compatible and aims to be easy to read and understand. The testing framework had its first release in June 2013 (release 0.2.0), which makes it very new and still under development. The latest release was version 0.21.0 which came out in the beginning of April and shows that there is still some work needed before their first full release (version 1.0) will be made [2].

2. Protractor installation

To install Protractor, start by downloading and installing Node.js [3]. After the installation of Node.js the package manager `npm` will be available for installing Protractor.



This package will install node and npm into /usr/local/bin

Now open a terminal or a command prompt and enter the following command:

```
npm install -g protractor
```

This will download and install Protractor in your `PATH`.

Once the installation of Protractor is complete the final step of the installation process, is to download and start the Selenium WebDriver server used by Protractor to control the browser interaction.

`webdriver-manager update`	Download/update WebDriver
`webdriver-manager start`	Start the WebDriver server

To run your tests using Protractor, you need to create a configuration file. The following is an example of a configuration that runs the tests using the Firefox browser.

```
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  capabilities: { 'browserName': 'firefox' },
  specs: ['*Test.js']
}
```

The configuration instructs Protractor to use the Selenium WebDriver server at the default localhost location and to run all tests in files ending with “Test.js”.

3. Building tests with Protractor

Protractor is aiming to make it easy for the tester to write the end to end tests. The goal is that it should be as easy to read and understand the test as it is to read and understand lines of text. The Protractor tests are built based on two main statements; the “describe-statement” and the “it-statement”. These will be described in more detail below.

3.1. Describe-statement

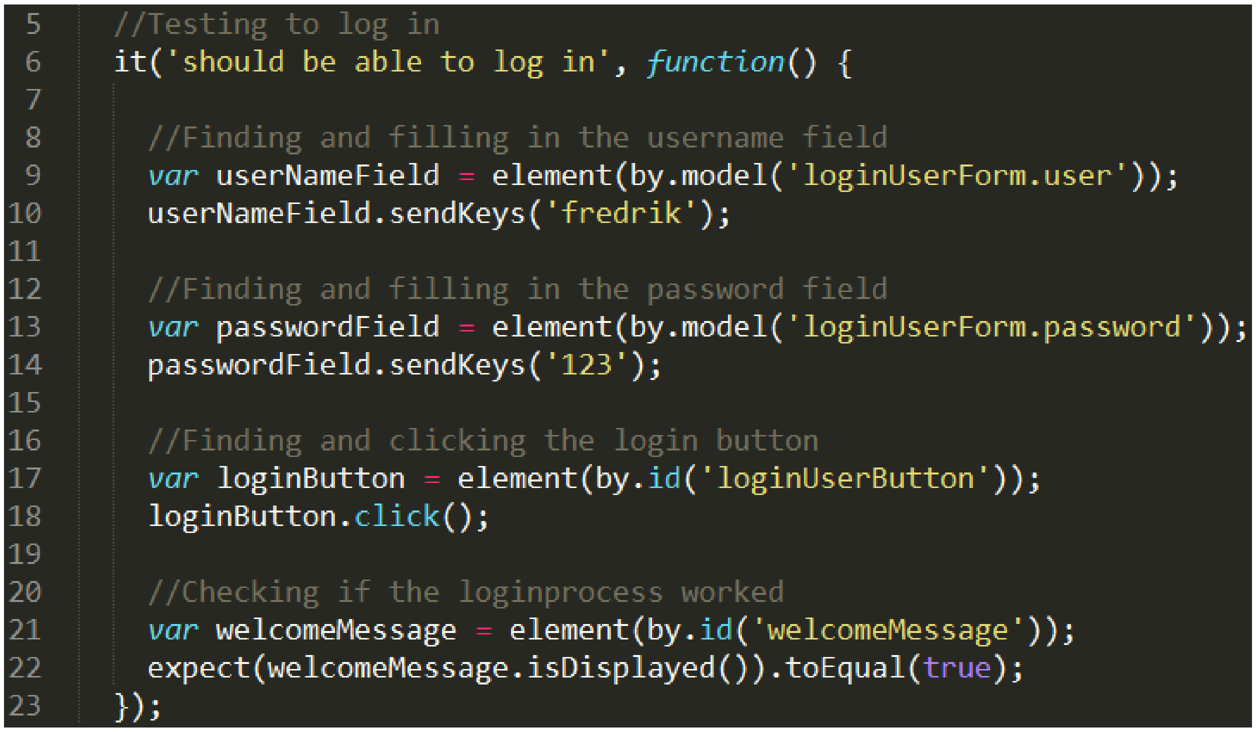
A Protractor test suit begins with a “describe-statement” containing a string and a function. The string is the name of the specific test suit and usually describes what is being tested. It is good practice to think of this string as part of a sentence beginning with “describe”. For example if the test is supposed to test basic functionality of a website, the name could be “basic test on a website”. When putting this into the describe statement it will be very easy to get a quick understanding of what the test suit is for, since it connects well with the describe statement itself.

```
describe('basic test on website', function() {
    browser.get('http://sqat3.local/');

    ...

});
```

The function in the describe statement is what is containing all test case code. It usually starts with a statement that retrieves the website so be tested, as can be seen above [2].

3.2. It-statement

Each “describe-statement” will contain a number of “it-statements”. These statements are what describe each specific test that is to be performed. Just like the “describe-statement”, the “it-statements” contains a test-name string and a function containing the test code. Just like the name string within the “describe-statement”, it is good practice that the name string within the “it-statement” should be written as part of a sentence beginning with “it”. If the test is supposed to test the login functionality the specified website, the name could be “should be able to log in”. This will also help to make the tests easy to understand in the future.



```
//Testing to log in
it('should be able to log in', function() {

  //Finding and filling in the username field
  var userNameField = element(by.model('loginUserForm.user'));
  userNameField.sendKeys('fredrik');

  //Finding and filling in the password field
  var passwordField = element(by.model('loginUserForm.password'));
  passwordField.sendKeys('123');

  //Finding and clicking the login button
  var loginButton = element(by.id('loginUserButton'));
  loginButton.click();

  //Checking if the loginprocess worked
  var welcomeMessage = element(by.id('welcomeMessage'));
  expect(welcomeMessage.isDisplayed()).toEqual(true);
});
```

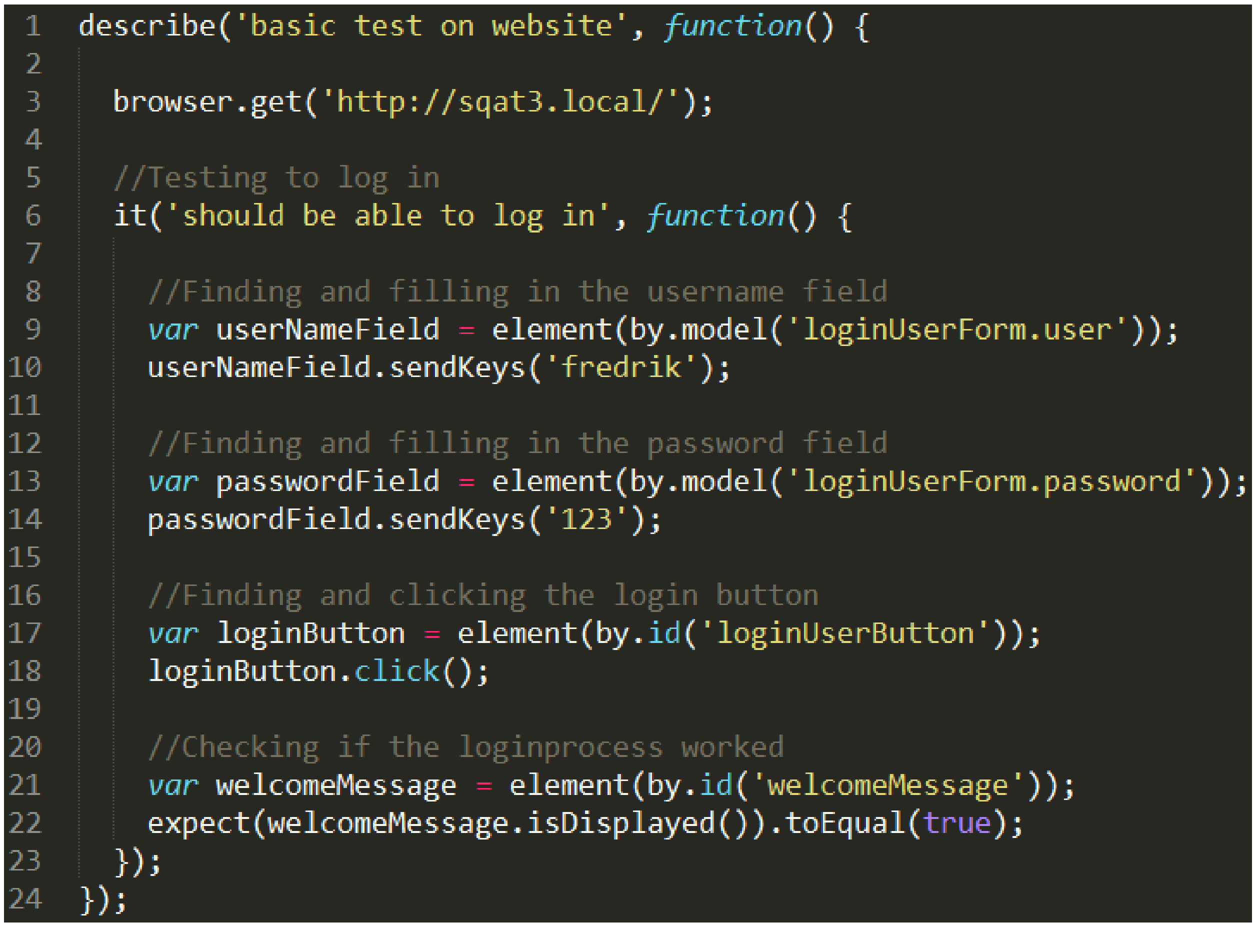
The function connected to the “it-statement” contains the code for the specific test. As can be seen in the code example above, the test is using element statement to locate certain items within the html code for the tested website. In this case it is locating the login fields and the login button. The “sendKeys” statement is used to insert information into the found field, and the “click-statement” is used to click the login button”.

At the end of each “it-statement” there should be an “expect-statement”. This is where the actual confirmation is performed to check if the website has produced the expected result after having gone through the procedures above. In the example above, the test checks if an item called “welcomeMessage” is displayed on the screen.

There is no limit for how many “it-statements” that can be included in each “describe-statements”; however it is good practice to create test suits focusing on specific parts of the website functionality. For example could a specific test suit be created to test the website security, while another test suit is performs test on some other site functionality. This will make it easier to late on choose what types of tests to perform on the website, since you can choose to run specific test suits to fulfill your testing needs.

3.3. The whole test

When the whole test is written for testing the login functionality of the website, the test could look something like this:



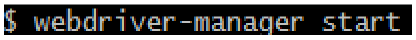
```
describe('basic test on website', function() {

  browser.get('http://sqat3.local/');

  //Testing to log in
  it('should be able to log in', function() {

    //Finding and filling in the username field
    var userNameField = element(by.model('loginUserForm.user'));
    userNameField.sendKeys('fredrik');

    //Finding and filling in the password field
    var passwordField = element(by.model('loginUserForm.password'));
    passwordField.sendKeys('123');

    //Finding and clicking the login button
    var loginButton = element(by.id('loginUserButton'));
    loginButton.click();

    //Checking if the loginprocess worked
    var welcomeMessage = element(by.id('welcomeMessage'));
    expect(welcomeMessage.isDisplayed()).toEqual(true);
  });
});
```

4. Running Protractor tests and understanding the results

When you have finished writing your tests and want to run them you have to do the following.

4.1. Start the selenium server

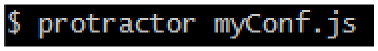
The first thing to do is to start the selenium server that is used to run the tests. This is done by running the following command in your command prompt:



```
$ webdriver-manager start
```

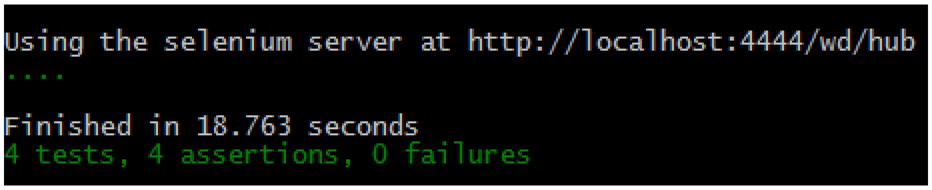
4.2. Run the test configuration file

When the selenium server is running, the next thing to do is to open up a new command prompt and navigate to the directory where the test configurations file is kept. While standing in the specified directory, run the following command to execute the tests.



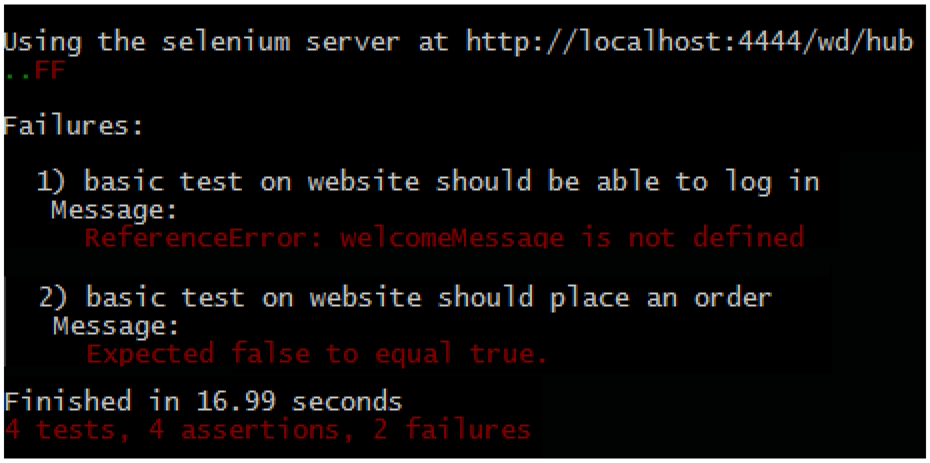
```
$ protractor myConf.js
```

The name of the configurations file for this example was “myConfig” but can be any name. Protractor will now run all the tests specified in your configurations file and in your command prompt you will be able to follow the progress and status of the tests performed. As can be seen below, a number of dots can be found in the command prompt printout. These dots give a primary indication of the status for each performed test. If a green dot is printed, the test performed was successful. A printout of such a situation can be seen below.



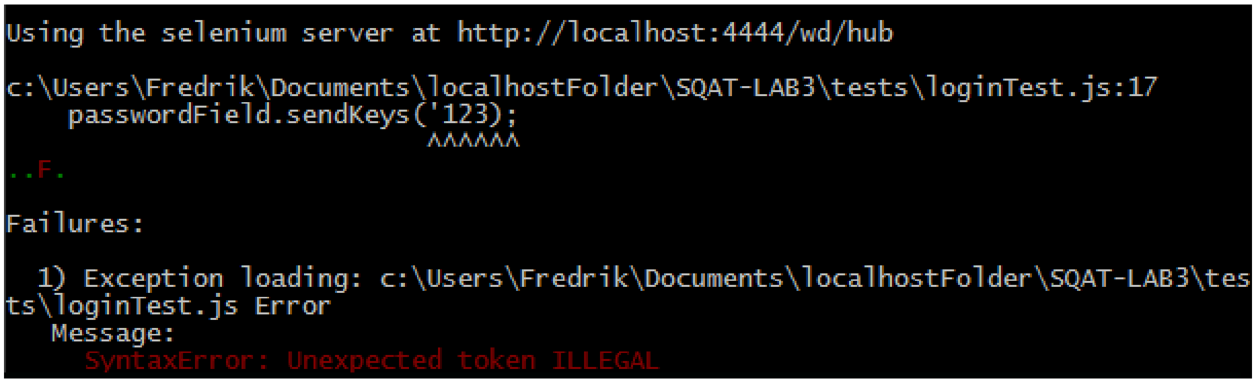
```
Using the selenium server at http://localhost:4444/wd/hub
....

Finished in 18.763 seconds
4 tests, 4 assertions, 0 failures
```

If the status indicator shows an “F” instead of a dot, this means that a failure of some sort occurred during the running of that specific test. These failures can occur due to either if Protractor discovered errors in the actual test code or if the expected result of a test was not equal to the actual result delivered by the system. No matter which of the two cases that occurs, Protractor will give the tester information about what went wrong. Below you find two cases of printouts from test runs where some test cases failed.



```
Using the selenium server at http://localhost:4444/wd/hub
..FF

Failures:

  1) basic test on website should be able to log in
   Message:
     ReferenceError: welcomeMessage is not defined

  2) basic test on website should place an order
   Message:
     Expected false to equal true.

Finished in 16.99 seconds
4 tests, 4 assertions, 2 failures
```

During this test run, one of the tests failed due to not being able to find a specific item that was to be included in one of the tests. Another test in the test run also failed due to the expected result not being equal to the actual result given by system.



```
Using the selenium server at http://localhost:4444/wd/hub

c:\Users\Fredrik\Documents\localhostFolder\SQAT-LAB3\tests\loginTest.js:17
    passwordField.sendKeys('123);
                           ^^^^^^
..F.

Failures:

  1) Exception loading: c:\Users\Fredrik\Documents\localhostFolder\SQAT-LAB3\tes
ts\loginTest.js Error
   Message:
     SyntaxError: Unexpected token ILLEGAL
```

During this test run, one of the tests failed due to errors in the code of one of the test cases. As can be seen above, Protractor finds the errors in the code and marks it up for the tester.

In the entire test run cases seen above four different tests were performed on the system. Even though some of the test failed, Protractor keeps on running and testing the remaining test cases. This means that the testers don’t have to worry about errors occurring that might end the test run. All test cases specified in the test configurations file will always be run.

5. Advantages and Disadvantages

In order to know if Protractor is the right testing tool for your testing purposes, here are some advantages and disadvantaged with the tool.

5.1. Advantages

- Protractor is very easy to use
- Protractor integrates well with AngularJS.
- It is very easy to create test cases in Protractor.
- Test cases are easy to understand when reading them (if written as described earlier in this document).
- There is a lot of documentation online about the basic principles of writing test cases for Protractor, due to the fact that it is built upon the Jasmine framework.
- Protractor is very flexible in the sense that the tool does not have any limits for what can or cannot be tested on a specific website. If website contains the information, Protractor can test it.

5.2. Disadvantages

- Since protractor itself is a very young test framework, more documentation is needed about how to test the angular related items on a website more thoroughly.
- Protractor only supports End to End testing of AngularJS websites (however the Jasmine framework supports unit testing, which can be used as a part of the test suit for your AngularJS application).

6. References

[1] http://angularjs.org/, 2014-04-17
[2] http://jasmine.github.io/2.0/introduction.html, 2014-04-17
[3] http://nodejs.org/, 2014-04-19