Protractor \ E2E tests

E2E TESTING FOR ANGULAR JS

What is Protractor

- Protractor is an end-to-end test framework for AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

Test Like a User
Protractor is built on top of WebDriverJS, which uses native events and browser-specific drivers to interact with your application as a user would.

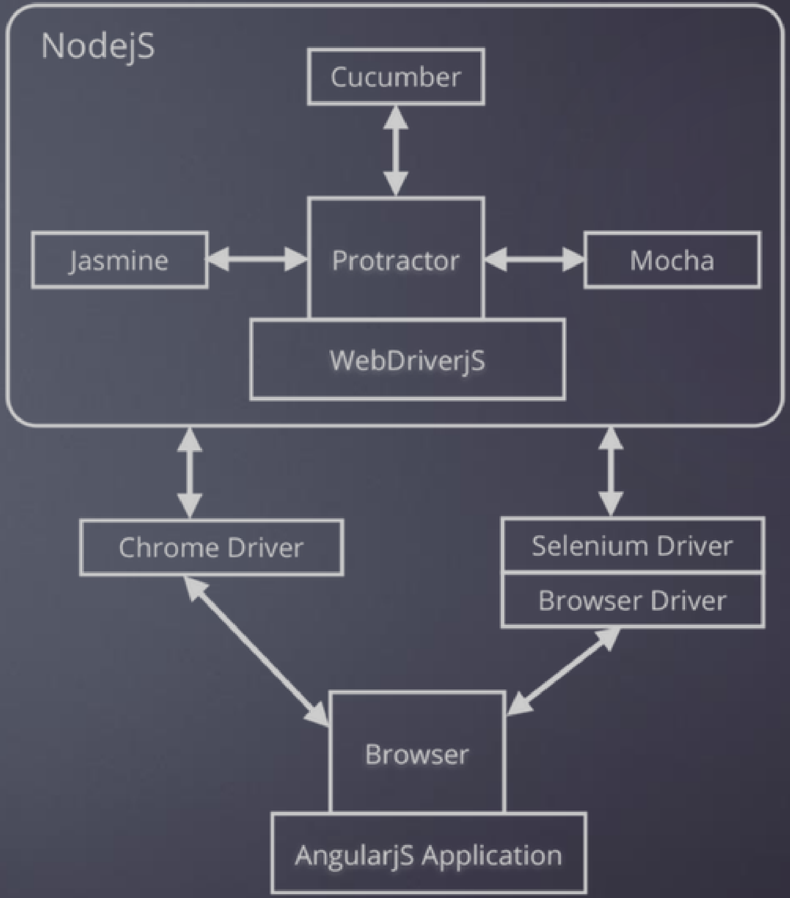
For AngularJS Apps
Protractor supports Angular-specific locator strategies, which allows you to test Angular-specific elements without any setup effort on your part.

Automatic Waiting
You no longer need to add waits and sleeps to your test. Protractor can automatically execute the next step in your test the moment the webpage finishes pending tasks, so you don't have to worry about waiting for your test and webpage to sync.

Protractor + BDD frameworks

BDD Frameworks

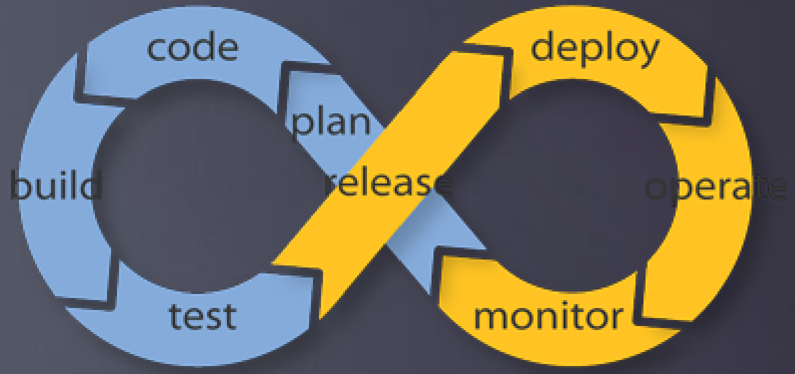
- Protractor supports four behavior driven development (BDD) test frameworks: Jasmine 1.3, Jasmine 2.0, Mocha, and Cucumber. These frameworks are based on JavaScript and Node.js and provide the syntax, scaffolding, and reporting tools you will use to write and manage your tests.



Protractor and Continous Integration

Support for many CI envirnoments

- Protractor suport for Continous Integration envirnoment is strictly connected with BDD frameworks abilities.
- Jasmine is a default BDD framework in protractor, it supports Jenkins, TeamCity, Hudson and many more CI envirnoments.
- CI reporters can be easly attached to any test suite using Protractor config file



Getting started with Protractor

To get Protractor working:

- Download Protractor using npm (**npm install –g protractor@1.8.0**)
- Download the Chrome Driver & Selenium Server *(already in repo)*
- Create the folder where tests & config file will be stored …
- Prepare config file for protractor *(basic config already in repo)*
- Write simple test *(already in repo)*
- Run Protractor

Protractor config file

The minimal config should contains following sections:

- **exports.config = {}** – as a configuration frame
- **seleniumServerJar** or **seleniumAddress** pointing to the Selenium Server
- **chromeDriver** pointing to the chromedriver (in case we are using Chrome)
- **suites** or **specs** pointing to the path with tests
- **framework** – if other than Jasmine 1.3

Protractor config file

Additional important sections:

- **capabilities** section – defining browser details
- **baseUrl** the default test URL
- **onPrepare** function – here you can atttach reporter to the CI system
- **jasmineNodeOpts / cucumberOpts / mochaOpts** – options of individual frameworks

Jasmine tests syntax

Following code represents basic Jasmine syntax :

```
describe ('My first test suite', function(){
    it('Will do nothing', function(){


    });
    it('Will also do nothing', function(){


    });
});
```

Jasmine assertions (matchers)

Basic schema of assertion: `expect(A).matcher(B);`

Following matchers are avaliable by default:

The '**toMatch**' matcher is for regular expressions

The '**toBeDefined**' matcher compares against `undefined`

The `**toBeUndefined**` matcher compares against `undefined`

The '**toBeNull**' matcher compares against null

The '**toBeTruthy**' matcher is for boolean casting testing

The '**toBeFalsy**' matcher is for boolean casting testing

The '**toContain**' matcher is for finding an item in an Array

The '**toBeLessThan**' matcher is for mathematical comparisons

The '**toBeGreaterThan**' matcher is for mathematical comparisons

The '**toBeCloseTo**' matcher is for precision math comparison

The '**toThrow**' matcher is for testing if a function throws an exception

The '**toThrowError**' matcher is for testing a specific thrown exception

- All matchers can be negated by adding .not before matcher
- User can define own matchers in the tests code

Web elements locators

Protractor supports it's own implementation of web elements locators, which is fully compilant with webdrivers implementation and extended by Angular-specific locators:

Locators in protractor are defined as follows:

```
element/all (by.locator(<locator string>))
```

Avaliable locators:

```
- css
- id
- xpath
- tagName
- Binding (ng-binding)
- Repeater (ng-repeat)
- model (ng-model)
- Name
- exactBinding
- buttonText
```

Locators tricks & tips, good practices for using locators, examples

Locators can be combined:

```
element(by.id(<someId>)).
element(by.css(<someCss>)).
element(by.tagName(<someTag>)).
all(by.repeater(<someRepeater>))

etc.
```

Locators tricks & tips, good practices for using locators, examples

CSS Locators can be used as other basic locators :

```
element(by.css(„.class”)) = css locator
element(by.css(„button”)) = tag locator
element(by.css(„#ideee”)) = id locator
```

Locators tricks & tips, good practices for using locators, examples

CSS Locators can be combined:

```
element(by.css(„.class button #id”))
```

Locators tricks & tips, good practices for using locators, examples

CSS Locators can be used for advanced search:

```
element(by.css(`buton[class*=„foo”]`)) -> find buton with class name
containing „foo” string

element(by.css(`.someCss:not(.ng-hide)`)) -> find only visible elements
with css = .someCss

element(by.css(`a:contains(„Change password”)`)) -> find only links
elements with text= „Change password”
```

Locators tricks & tips, good practices for using locators, examples

CSS Locators can be used for horizontal search:

```
element(by.css(`.ng-binding.ng-show.dimmed`)) -> find element with
given 3 classes at the same level
```

Actions on objects

Protractor supports a range of default actions on located web elements:

```
- click()
- sendKeys()
- getAttribute()
- isPresent()
- isDisplayed()
- clear()
- getText()
- Many many more:(protractor API)
```

!!! All of above actions return **promisses** and need to be **solved** by using **then** function

Writing tests in Protractor(exercise 1)

Write your first test using sample structure given in repo

Pre-requirements:
- Protractor correctly installed
- We are using yesterday's application
- Node server started on localhost:8080 (hosting our web app)
(lesson-6-path/node server.js)
- Test must be runned from the Lesson 6 folder directly using CMD

Requirements:

- Locate input box of tested application
- Give a unique name of your new ToDo task
- Add the task
- Check if the task is displayed on the list (by name)

Writing tests in Protractor(exercise 2)

Write your first test using sample structure given in repo

```
Pre-requirements:
- Protractor correctly installed
- We are using yesterday's application
- Node server started on localhost:8080 (hosting our web app)
(lesson-6-path/node server.js)
- Test must be runned from the Lesson 6 folder directly using CMD

Requirements:

- Locate input box of tested application
- Give a unique name of your new ToDo task
- Add the task
- Check if the task is displayed on the list (by name)
- Click on added element
- Check if is marked as done
- Verify if All and ToDo counters have correct values
```

E2E tools tribute – page objects

Let's analyse the code of our POP library and see how the informations from previous part of course can be used in real live

Layer 1:
POP library

Layer 2:
Defining page objects

Layer 3:
Writing tests

HTTP backend mocks 4 Protractor

HttpBackend workflow is quite simple:

On browser.get() a mock module is injected to your angularjs application

On when*or when you call manually backend.sync(), fixtures is synchronised with your angularjs app.

HTTP backend mocks 4 Protractor

```
var HttpBackend = require('httpbackend');
var backend = null;

describe('Test Http backend methods', function() {

    beforeEach(function() {
        backend = new HttpBackend(browser);
    });

    afterEach(function() {
        backend.clear();
    });

    it('Test whenGET with string response', function() {
        backend.whenGET(/result/).respond('raoul');

        browser.get('http://127.0.0.1:8080');

        var result = element(by.binding('result'));
        expect(result.getText()).toEqual('raoul');
    });
});
```

Thank You

DZIĘKUJE ZA UWAGĘ