PENTASYS

Unser Maßstab ist der Mensch

Protractor Workshop

Munich, April 26th 2016

Julio Herce
Software Engineer bei comSysto

PENTASYS
Unser Maßstab ist der Mensch

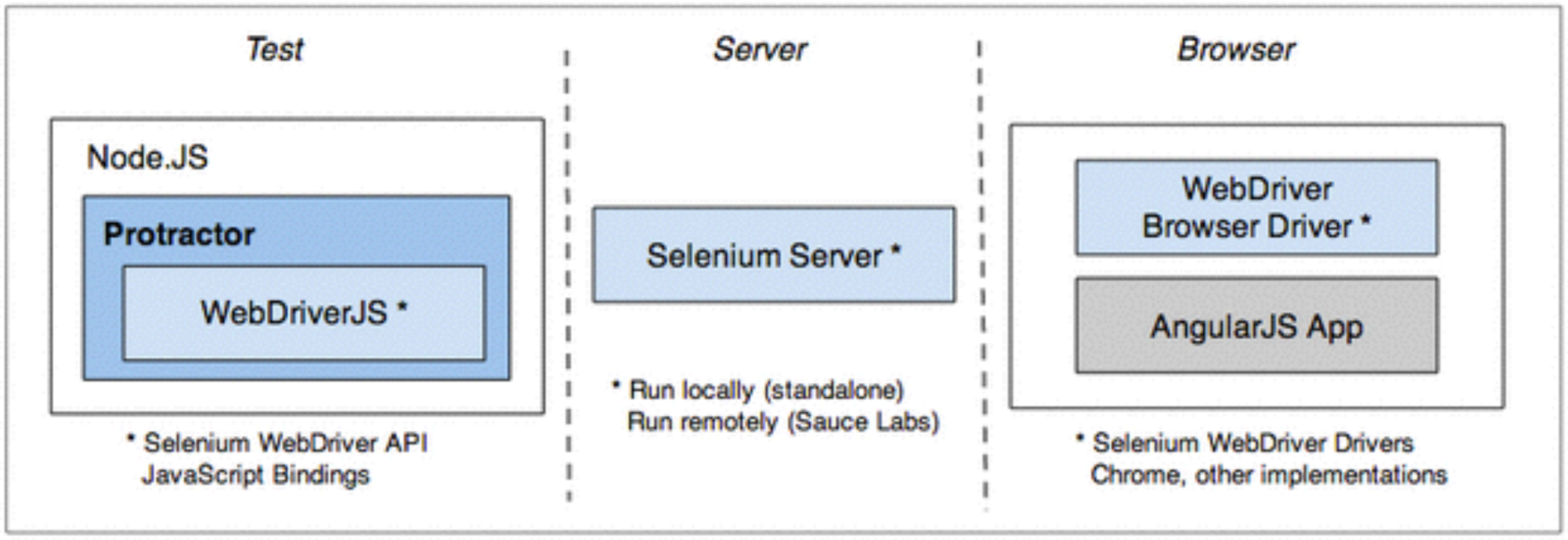
Agenda

- Protractor introduction
- Behaviour driven framework
- How to get started
- Config file
- Locators
- Page Objects
- Control Flow - Promises

Protractor

PENTASYS
Unser Maßstab ist der Mensch

Protractor



PENTASYS
Unser Maßstab ist der Mensch

Jasmine - Behaviour Driven Framework

PENTASYS
Unser Maßstab ist der Mensch

Behaviour Driven Framework

Protractor supports three behaviour driven development (BDD) test frameworks:

- Jasmine

  It comes preinstalled with protractor and it is the default test framework when protractor is installed.

- Cucumber

```
npm install -g cucumber
npm install --save-dev protractor-cucumber-framework
```

- Mocha

```
npm install -g mocha
npm install chai
npm install chai-as-promised
```

PENTASYS
Unser Maßstab ist der Mensch

Behaviour Driven Framework - Jasmine

- Jasmine 2.x is supported and default.

```
framework: 'jasmine2',
```

- Jasmine 1.3 is also available.

```
framework: 'jasmine',
```

PENTASYS
Unser Maßstab ist der Mensch

Behaviour Driven Framework - Jasmine Syntax

```
describe("A suite", function() {
  it("contains spec with an expectation", function() {
    expect(true).toBe(true);
  });
});
```

BeforeEach and AfterEach block

```
describe("A spec using beforeEach and afterEach", function() {
  var foo = 0;

  beforeEach(function() {
    foo += 1;
  });

  afterEach(function() {
    foo = 0;
  });

  it("is just a function, so it can contain any code", function() {
    expect(foo).toEqual(1);
  });

  it("can have more than one expectation", function() {
    expect(foo).toEqual(1);
    expect(true).toEqual(true);
  });
});
```

BeforeAll and AfterAll block

```
describe("A spec using beforeAll and afterAll", function() {
  var foo;

  beforeAll(function() {
    foo = 1;
  });

  afterAll(function() {
    foo = 0;
  });

  it("sets the initial value of foo before specs run", function(
    expect(foo).toEqual(1);
    foo += 1;
  });

  it("does not reset foo between specs", function() {
    expect(foo).toEqual(2);
  });
});
```

Behaviour Driven Framework - Jasmine Syntax

Disabling Suite

```
xdescribe("A spec", function() {
  var foo;

  beforeEach(function() {
    foo = 0;
    foo += 1;
  });

  it("is just a function, so it can contain any code", function() {
    expect(foo).toEqual(1);
  });
});
```

Pending specs

```
describe("Pending specs", function() {




  xit("can be declared 'xit'", function() {
    expect(true).toBe(false);
  });


  it("can be declared with 'it' but without a function");



  it("can be declared by calling 'pending' in the spec body", function() {
    expect(true).toBe(false);
    pending('this is why it is pending');
  });
});
```

How to get started

PENTASYS
Unser Maßstab ist der Mensch

What do we need to get started?

- Protractor

```
npm install -g protractor
```

- Web driver

```
webdriver-manager update
```

downloads

Chromedriver_2.21

Selenium-server-standalone-2.52.0.jar

```
webdriver-manager start
```

starts up

Selenium Server

- Protractor config file

- Spec file

PENTASYS
Unser Maßstab ist der Mensch

What do we need to get started?

spec.js

```
// spec.js
describe('Protractor Demo App', function() {
  it('should have a title', function() {
    browser.get('http://juliemr.github.io/protractor-demo/');

    expect(browser.getTitle()).toEqual('Super Calculator');
  });
});
```

conf.js

```
// conf.js
exports.config = {
  framework: 'jasmine',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js']
}
```

```
protractor conf.js
```

Protractor config file - The backbone from Protractor

PENTASYS
Unser Maßstab ist der Mensch

Protractor config file - Connecting to Browser Driver

There are 4 different built-in possibilities to connect to Browser Drivers.

1. Start Selenium Standalone Server locally
2. Connect to a running selenium server (local or remote)
3. Connect to Sauce Labs / Browser Stack remote Selenium Server
4. Connect directly to chrome or firefox

PENTASYS
Unser Maßstab ist der Mensch

Protractor config file - Connecting to Browser Driver

1. Start Selenium Standalone Server locally

Specify *seleniumServerJar* path

2. Connect to a running selenium server (local or remote)

Specify *seleniumAddress* from running selenium Server

PENTASYS
Unser Maßstab ist der Mensch

Protractor config file - Connecting to Browser Driver

3. Connect to Sauce Labs / Browser Stack remote Selenium Server

Set *sauceUser* and *sauceKey* / Set *browserstackUser* and *browserstackKey*

4. Connect directly to chrome or firefox

Set *directConnect* to *true*

PENTASYS
Unser Maßstab ist der Mensch

Protractor config file - Setting specs and suites

- Setting spec files patterns

```
// Spec patterns are relative to the location of this config.
specs: [
  'spec/*_spec.js'
],
```

- Setting suites

```
suites: {
  smoke: 'spec/smoketests/*.js',
  full: 'spec/*.js'
},
```

```
protractor conf.js --suite="smoke"
```

```
protractor conf.js --suite="full"
```

PENTASYS
Unser Maßstab ist der Mensch

Protractor config file - Capabilities

```
capabilities: {
  browserName: 'chrome',

  // Name of the process executing this capability.  Not used directly by
  // protractor or the browser, but instead pass directly to third parties
  // like BrowserStack and SauceLabs as the name of the job running this test
  name: 'Unnamed Job',

  // User defined name for the capability that will display in the results log
  // Defaults to the browser name
  logName: 'Chrome - English',

  // Number of times to run this set of capabilities (in parallel, unless
  // limited by maxSessions). Default is 1.
  count: 1,

  // If this is set to be true, specs will be sharded by file (i.e. all
  // files to be run by this set of capabilities will run in parallel).
  // Default is false.
  shardTestFiles: false,

  // Maximum number of browser instances that can run in parallel for this
  // set of capabilities. This is only needed if shardTestFiles is true.
  // Default is 1.
  maxInstances: 1,

  // Additional spec files to be run on this capability only.
  specs: ['spec/chromeOnlySpec.js'],

  // Spec files to be excluded on this capability only.
  exclude: ['spec/doNotRunInChromeSpec.js'],

  // Optional: override global seleniumAddress on this capability only.
  seleniumAddress: null

  // Optional: Additional third-party specific capabilities can be
  // specified here.
  // For a list of BrowserStack specific capabilities, visit
  // https://www.browserstack.com/automate/capabilities
},
```

PENTASYS
Unser Maßstab ist der Mensch

Protractor config file - Multi Capabilities

```
// If you would like to run more than one instance of WebDriver on the same
// tests, use multiCapabilities, which takes an array of capabilities.
// If this is specified, capabilities will be ignored.
multiCapabilities: [],

// If you need to resolve multiCapabilities asynchronously (i.e. wait for
// server/proxy, set firefox profile, etc), you can specify a function here
// which will return either `multiCapabilities` or a promise to
// `multiCapabilities`.
// If this returns a promise, it is resolved immediately after
// `beforeLaunch` is run, and before any driver is set up.
// If this is specified, both capabilities and multiCapabilities will be
// ignored.
getMultiCapabilities: null,

// Maximum number of total browser sessions to run. Tests are queued in
// sequence if number of browser sessions is limited by this parameter.
// Use a number less than 1 to denote unlimited. Default is unlimited.
maxSessions: -1,
```

PENTASYS
Unser Maßstab ist der Mensch

Protractor config file - Timeouts

```
jasmineNodeOpts: {
    defaultTimeoutInterval: 30000,
    print: function () {
    }
},

allScriptsTimeout: 11000,

getPageTimeout: 10000,
```

- **defaultTimeoutInterval**: Default time to wait in ms before a test fails.
- **allScriptsTimeout**: The timeout in milliseconds for each script run on the browser. This should be longer than the maximum time your application needs to stabilize between tasks.
- **getPageTimeout**: How long to wait for a page to load.

Locators

PENTASYS
Unser Maßstab ist der Mensch

Locators

- Protractor exports a global function element, which takes a Locator and will return an ElementFinder.

- A locator tells Protractor how to find a certain DOM element.

```
// find an element using a css selector
by.css('.myclass')

// find an element with the given id
by.id('myid')

// find an element with a certain ng-model
by.model('name')

// find an element bound to the given variable
by.binding('bindingname')
```

- ProtractorBy API

PENTASYS
Unser Maßstab ist der Mensch

Locators - Actions

- The element() function returns an ElementFinder object. The ElementFinder knows how to locate the DOM element using the locator you passed in as a parameter, but it has not actually done so yet. It will not contact the browser until an **action** method has been called.

```
var el = element(locator);

// Click on the element
el.click();

// Send keys to the element (usually an input)
el.sendKeys('my text');

// Clear the text in an element (usually an input)
el.clear();

// Get the value of an attribute, for example, get the value of an input
el.getAttribute('value');
```

Locators - Multiple elements, chained locators

To deal with multiple DOM elements, use the `element.all` function. This also takes a locator as its only parameter.

```
element.all(by.css('.selector')).then(function(elements) {
  // elements is an array of ElementFinders.
});
```

Using chained locators to find:

- a sub-element:

```
element(by.css('some-css')).element(by.tagName('tag-within-css'));
```

- to find a list of sub-elements:

```
element(by.css('some-css')).all(by.tagName('tag-within-css'));
```

Locators - Custom Locators

```
// Add custom Locator to find banana element
var findBananaElement = function (elementName, parentElement) {
    parentElement = parentElement || document;
    return parentElement.querySelectorAll('[banana="' + elementName + '"]');
};

by.addLocator('bananaElement', findBananaElement);
```

```
var chiquita = element(by.bananaElement('chiquita'));
```

```
<div class="some-class" banana="chiquita">
    Chiquita is the best banana
</div>
```

Page Objects

PENTASYS
Unser Maßstab ist der Mensch

Page Objects

Without Page Objects

Here's a simple test script (example_spec.js) for 'The Basics' example on the angularjs.org homepage.

```
describe('angularjs homepage', function() {
  it('should greet the named user', function() {
    browser.get('http://www.angularjs.org');
    element(by.model('yourName')).sendKeys('Julie');
    var greeting = element(by.binding('yourName'));
    expect(greeting.getText()).toEqual('Hello Julie!');
  });
});
```

PENTASYS
Unser Maßstab ist der Mensch

Page Objects

With PageObjects

To switch to Page Objects, the first thing you need to do is create a Page Object. A Page Object for 'The Basics' example on the angularjs.org homepage could look like this:

```
var AngularHomepage = function() {
  var nameInput = element(by.model('yourName'));
  var greeting = element(by.binding('yourName'));

  this.get = function() {
    browser.get('http://www.angularjs.org');
  };

  this.setName = function(name) {
    nameInput.sendKeys(name);
  };

  this.getGreeting = function() {
    return greeting.getText();
  };
};
```

```
describe('angularjs homepage', function() {
  it('should greet the named user', function() {
    var angularHomepage = new AngularHomepage();
    angularHomepage.get();

    angularHomepage.setName('Julie');

    expect(angularHomepage.getGreeting()).toEqual('Hello Julie!');
  });
});
```

PENTASYS
Unser Maßstab ist der Mensch

Page Objects

Advantages of using *Page Objects*

- Domain Specific Language Support
- Reduce Code Duplication
- Loose coupling
- Easier maintenance

Control Flow - Promises

Control Flow - Promises

- WebDriverJS (and thus, Protractor) APIs are entirely asynchronous. All functions return promises.

- WebDriverJS maintains a queue of pending promises, called the control flow, to keep execution organized. For example, consider this test:

```
it('should find an element by text input model', function() {
  browser.get('app/index.html#/form');

  var username = element(by.model('username'));
  username.clear();
  username.sendKeys('Jane Doe');

  var name = element(by.binding('username'));

  expect(name.getText()).toEqual('Jane Doe');

  // Point A
});
```

At Point A, none of the tasks have executed yet. The `browser.get` call is at the front of the control flow queue, and the `name.getText()` call is at the back. The value of `name.getText()` at point A is an unresolved promise object.

Control Flow - Promises

Protractor Adaptations

Protractor adapts Jasmine so that each spec automatically waits until the control flow is empty before exiting.

Jasmine expectations are also adapted to understand promises. That's why this line works - the code actually adds an expectation task to the control flow, which will run after the other tasks:

```
expect(name.getText()).toEqual('Jane Doe');
```

PENTASYS
Unser Maßstab ist der Mensch

Fragen?

PENTASYS

Unser Maßstab ist der Mensch

Thank you!

Julio Herce
julio.herce@comsysto.com