Welcome

Copyright Notice

Protractor

End to End Application Testing framework

Prakash Badhe

prakash.badhe@vishwasoft.in

About Me : Prakash Badhe

- ❑Enterprise Technologies Consultant-Trainer for 16+ years
- ❑Passion for technologies and frameworks
- ❑Proficient in Java, JavaScript Frameworks and UI libraries
- ❑Proficient with Html5,Ajax,XML standards
- ❑Working with EXTJS, GWT, Angular, Ember, JQuery…
- ❑Supports agile technical practices in agile development environment.

About you..

- Job Role
- Skill-Sets
- Objectives
- Prior experience with JavaScript, Testing.
- Exposure to UI programming with JavaScript
- Exposure to UI testing with Selenium
- Experience in Web applications development
- Exposure to build management, Jenkins.

Agenda

- Java Script introduction
- Object Oriented Java Script
- JavaScript testing with Jasmine
- Unit testing with Karma and Jasmine
- Protractor introduction
- Web application testing with Protractor
- End to End testing with Protractor
- Protractor Page object pattern for test script
- Jenkins Continuous Integration
- Protractor and Jenkins Integration

Setup

- Windows 7/10 64 bit, 4 GB RAM, 500GB HDD
- Admin rights
- Internet connection
- NODE JS
- Google Chrome, Firefox browsers
- JDK 1.8
- VS Code Editor
- GIT repository server and client.
- Jenkins CI Server

JavaScript Review

- The JavaScript has been defined to add dynamism to html pages in the form of user interactions, event listeners, validation, server interactions etc.
- Java Script code is included within html/dynamic web pages by using <script> tags.
- The JavaScript code in the html page is interpreted at runtime by the JavaScript interpreter engine in the browser and executed line by line.
- The java script makes most part of the application client centric with minimum server interaction.

JavaScript Applications

- It's widely used in tasks ranging from the validation of form data to the creation of complex user interfaces inside the browser, dynamically.
- JavaScript supports dynamic manipulation (update/delete/create) of the visual graphics contents of the web page.
- JavaScript combined with CSS is used to create animations and graphic effects in web pages.
- JavaScript supports dynamic event handling for graphic components like button, check box etc.

Java script applications..

- JavaScript supports user interaction in the form of taking inputs from the user, display messages, dialogs and submitting data to server programs.
- JavaScript also supports pulling data from the server and process it on the client side asynchronously(New addition with Ajax).
- By utilizing the Java script the page running in browser is self contained with minimum server interactions.

JavaScript standard

- JavaScript is the open source standard scripting language and most of recent versions of web browsers support java script.
- NetScape created JavaScript.
- Earlier MS IE browsers had their own scripting language VbScript which was not supported on other browsers.
- Nothing to do with java.
- A common standard ‘ECMA 3/4/5.0/ES6’ is defined for java script.

Java script compatibility

- Browsers add the extensions to JavaScript which most times is not supported by other browsers.
- Some of the java script behavior is being implemented differently in different browsers and some is not supported.
- This makes the web page dependent on the particular browser for execution.
- Programming techniques/frameworks to make web page browser independent.

JavaScript Libraries

- JQuery
- Angular
- Ember
- DurandalJS
- Knockout
- Backbone
- Bootstrap

JavaScript on Server

- NODE JS (with V8 engine-same as Google Chrome)
- NODE script for server side applications
- Node Package Manager – NPM
- NODE JS Eco system of tools

JavaScript Testing

- Testing involves defining test functions to invoke the target functions with simulated data and verifying the results during the test execution.
- The testing frameworks
- QUnit
- YUI Test
- Jasmine
- Mocha

BDD Unit Testing

- The behavior Driven Development testing specifies test specifications in defined format to specify the test conditions.
- The Jasmine test framework supports BDD testing for unit and end to end testing.
- For testing the AngularJS supports Jasmine like scenario test runner.
- For unit testing the AngularJS supports unit tests with Jasmine test runner.

End to End Testing

- The testing of external behaviour of web application such as user login, info retrieval etc.
- Web application UI testing
- Automated acceptance testing
- Involves input simulations and actions
- Testing of web page as user will interact
- Test script to be customized for different behaviour

UI testing tool

- The angular has built an Angular Scenario Runner which simulates user interactions that will help to verify the behaviour of the Angular application.
- The unit and end to end testing is supported by built-in implementation angular mocks which supports scenario based testing.

E2E Testing Tools

- **Selenium**
- **Angular Ng-scenario**
- **Nightwatch.js**
- **Casperjs**
- **Protractor from Google**

Test Runner

- QUnit Test Runner
- Jasmine Test Runner
- Angular Scenario Test Runner
- Karma Test runner
- Protractor

Jasmine

- Open source unit testing framework for JavaScript
- Test Suites has a hierarchical structure
- Tests as specifications
- Support test result as expectations and verifications
- Matchers, both built-in and custom
- Spies, a test double pattern

Intro to BDD

- Behavior Driven Development
- In a BDD style test you describe your code and tell the test what it should be doing. Then you expect your code to do something.
- Jasmine supports BDD style test cases.

Jasmine BDD style

- //describe your code
- describe ('presentation.js' , function(){
- //what it should do
- it ('should be informative', function(){
-   //expect something
-   expect(presentation.inform()).toBeTruthy();)
- }; });

Jasmine Test Specs

- Spec files are where the test cases live.
- Typically a single spec will be written for each .js file in your app.
- **Describe** blocks can be nested.
- As a rule of thumb nested **describe** blocks describe three or more **expect** statements in an **it** block.

Jasmine Test Suite

- describe("A specification suite", function() {
- …//test specs
- });
- Group specifications together using nested describe function blocks.
- Useful for delineating context-specific specifications.

Jasmine test case

- describe("A specification suite", function() {
- it(“contains spec with an expectation", function() {
- expect(view.tagName).toBe(‘tr’);
-   });
- });
- The test cases/ specifications are expressed with the **it function.**
- The description is the details.
- Expectations are expressed with the **expect function.**

Test setup

Jasmine setup using beforeEach

```
describe("PintailConsulting.ToDoListView", function() {
var view;
beforeEach(function(){
  view = new PintailConsulting.ToDoListView();
});
it(“sets the tagName to ‘div’", function() {
   expect(view.tagName).toBe(‘div’);
});
});
```

Test tear down

```
describe("PintailConsulting.ToDoListView", function() {
var view;
beforeEach(function(){
  view = new PintailConsulting.ToDoListView();
});
afterEach(function(){
view = null;
});
it(“sets the tagName to ‘div’", function() {
expect(view.tagName).toBe(‘div’);
});});
```

Matcher

- Each matcher implements a boolean comparison between the actual value and the expected value.
- It is responsible for reporting to Jasmine if the expectation is true or false.
- Jasmine will then pass or fail the spec.
- Any matcher can evaluate to a negative assertion by chaining the call to expect with a not before calling the matcher.

Test Matchers

- toBe('expected') //exact compare (===)
- toEqual('expected') //more general compare, can compare objects
- toBeDefined( ) //checks if var is not undefined
- toBeUndefined( ) //checks for undefined
- toBeNull( ) //checks if a variable is null
- toMatch(/regex/) //matches against regex
- toBeTruthy( ) //checks if variable is true
- toBeFalsy( ) //checks if variable is falsy
- toBeLessThan(number) //checks if value is less than number

Expectation

- expect(function(){
-     fn();
-   }).toThrow(e) //fn() should throw an error if result is not matching
- expect(5).not.toEqual(3);

Matching the expectation

- Jasmine has a rich set of matchers included.
- There is also the ability to write custom matchers for when a project's domain calls for specific assertions that are not included.

Set of Expectations

- expect(true).toBe(true) expect(true).not.toBe(true)
- expect(a).toEqual(bar)
- expect(message).toMatch(/bar/)
- expect(message).toMatch('bar')
- expect(a.foo).toBeDefined()
- expect(a.foo).toBeUndefined()
- expect(a.foo).toBeNull()
- expect(a.foo).toBeTruthy() expect(a.foo).toBeFalsy()
- expect(message).toContain('hello')

Expect more

- expect(pi).toBeGreaterThan(3)
- expect(pi).toBeLessThan(4)
- expect(pi).toBeCloseTo(3.1415, 0.1)
- expect(func).toThrow()

Custom matchers

```
beforeEach(function() {
this.addMatchers({
toBeLessThan: function(expected) {
var actual = this.actual;
var notText = this.isNot ? " not" : "";
this.message = function () {
return "Expected " + actual + notText +
" to be less than " + expected;
}
return actual < expected;
}
});
});
```

Jasmine Spies

- Test double pattern.
- Interception-based test double mechanism
- Spies record invocations and invocation parameters,
- allowing to inspect the spy after exercising the SUT.
- Very similar to mock objects.

Spy Usage

- ***Spying and verifying invocation***
- var spy = spyOn(dependency, “render”);
- systemUnderTest.display();
- expect(spy).toHaveBeenCalled();
- ***Spying, verifying invocation and argument(s)***
- var spy = spyOn(dependency, “render”);
- systemUnderTest.display(“Hello”);
- expect(spy).toHaveBeenCalledWith(“Hello”);

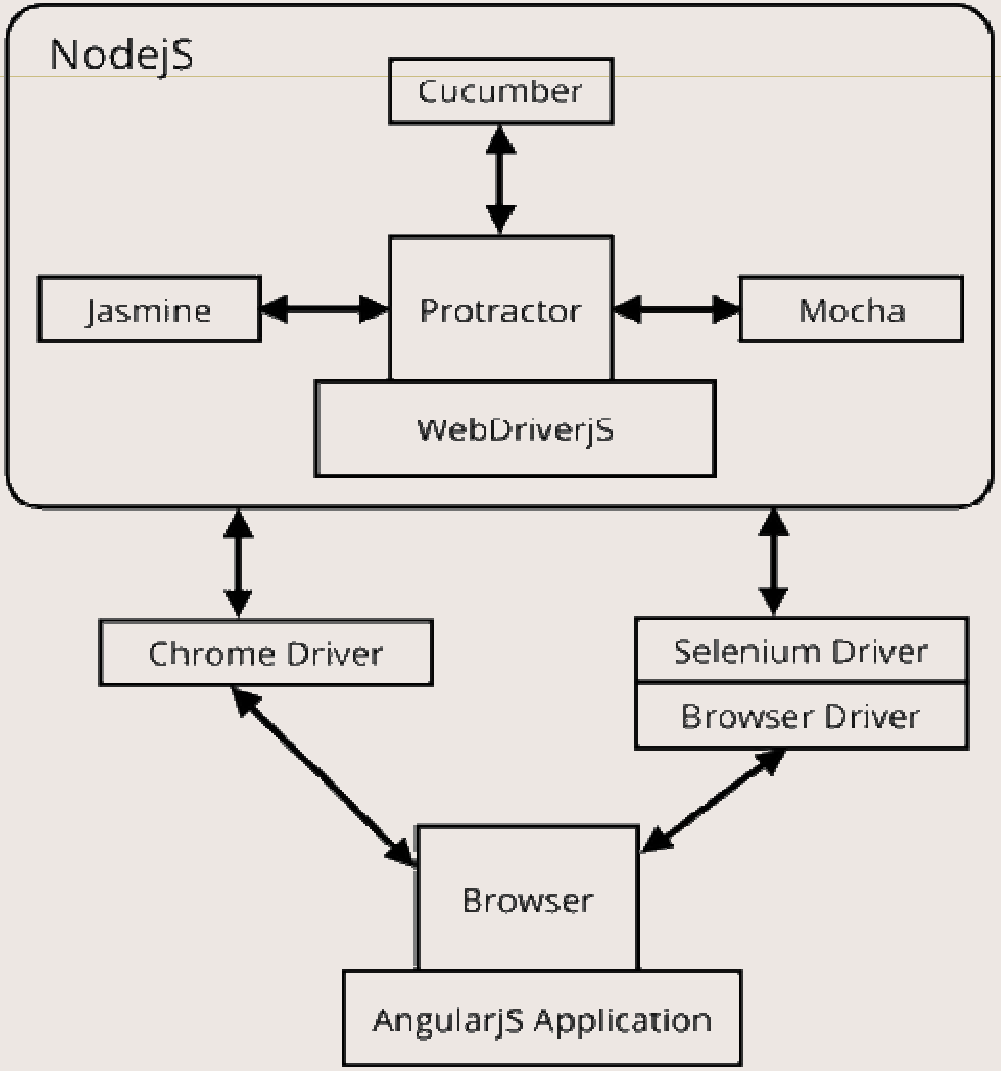
Jasmine Spy usage

- spyOn(foo, 'setBar') spyOn(foo, 'setBar').andReturn(123)
- spyOn(foo, 'getBar').andCallFake(function() { return 1001; }) foo.setBar(123)
- expect(foo.setBar).toHaveBeenCalled()
- expect(foo.setBar).toHaveBeenCalledWith(123)
- expect(foo.setBar.calls.length).toEqual(2)
- expect(foo.setBar.calls[0].args[0]).toEqual(123)

Protractor

- Protractor is an end-to-end testing framework for web applications and works as a solution integrator
- Combining powerful tools and technologies such as NodeJS, Selenium, webDriver, Jasmine, Cucumber and Mocha with Protractor.
- It runs the test specs on real browsers and headless browsers.

Protractor architecture



Protractor test execution

- Selenium Server to manage browsers
- Selenium WebDriver to invoke browser APIs
- Protractor node app to run tests
- Jasmine, etc as test framework

UI testing

- Protractor runs on top of the Selenium, and thus provides all the benefits and advantages from Selenium.
- In addition, it provides customizable features to test web applications UI.
- It is also possible to use some drivers which implement WebDriver's wire protocol like ChromeDriver and GhostDriver, as Protractor runs on top of the Selenium.
- With ChromeDriver it supports to run tests without the Selenium Server.(directConnect mode)

Jasmine Integration

- The protractor framework integrated with Jasmine supports create and organize tests and user expectations.
- Jasmine is compatible with Protractor due to which all resources that are extracted from browsers can be used to make tests as promises.
- Those promises are resolved internally by using the “expect” command from Jasmine. That way the promises work smoothly while creating tests.
- The promise is an asynchronous execution process

Browser support

- Chrome
- Safari
- Mozulla fireFox
- Internet explorer
- Opera

Configuration

protractor.conf.js

```
exports.config = {
  onPrepare: function () { … },
  capabilities: {'browserName’:'firefox’},
  specs: ['../tests/*.spec.js'],
  baseUrl: 'http://localhost:8080/’,
  jasmineNodeOpts: { … },
  // and many more options
}
```

- To run
- protractor protractor.conf.js

Test spec

```
describe('HomePage', function () {
    it('should be the default page',
      function () {
        browser.get(browser.baseUrl);
        expect(browser.getCurrentUrl())
         .toEqual(browser.baseUrl
          + 'projectsinfo');
    });
});
```

Protractor webDriver API

Debugging the test

Add to script:

```
browser.debugger();
```

Launch in debug mode:

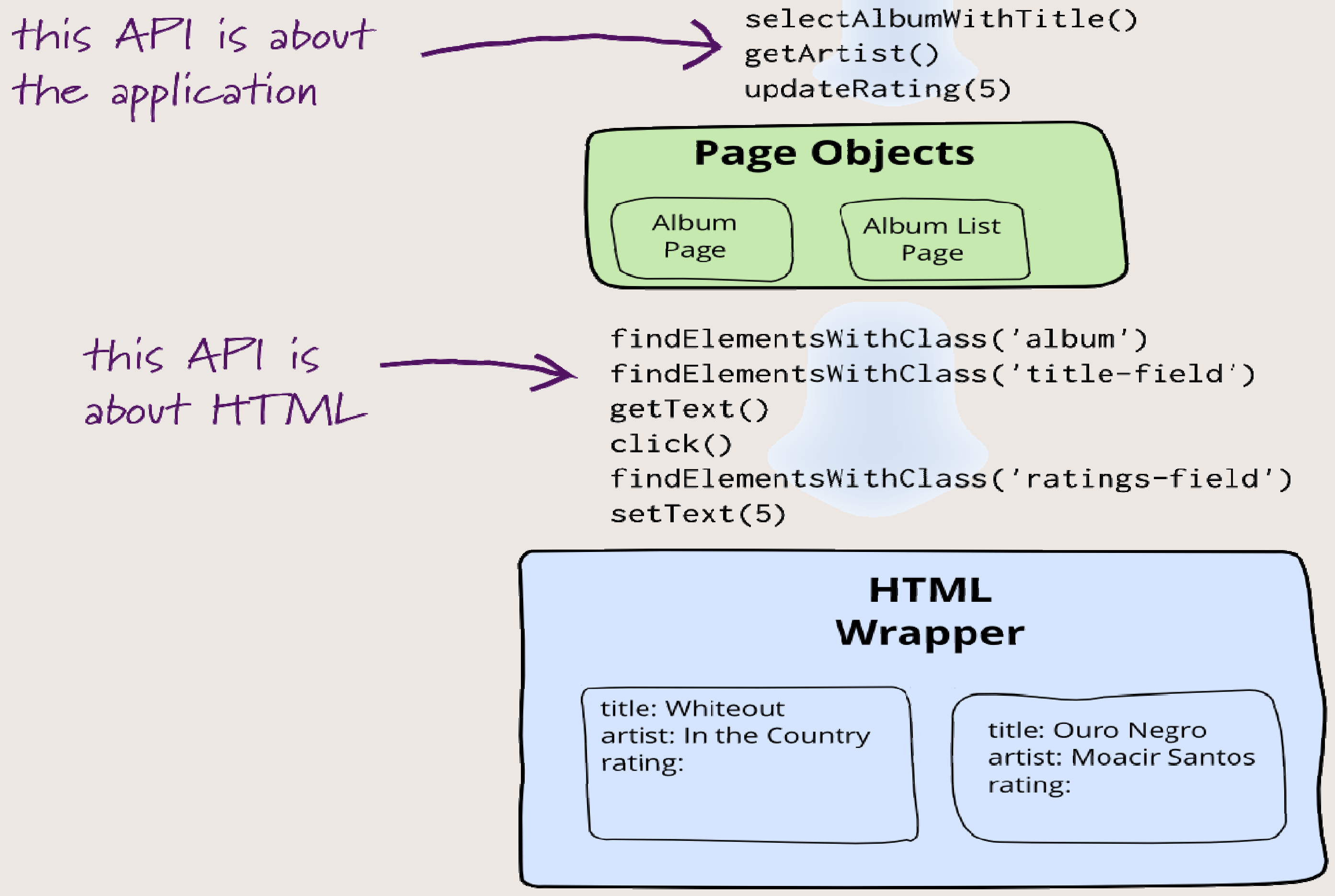
```
protractor debug protractor.conf.js
```

Test Reports

Page Object Pattern

- When you write tests against a web page, you need to refer to elements within that web page in order to click links and determine what's displayed.
- However, the tests that manipulate the HTML elements directly are brittle to changes in the UI.
- A page object wraps an HTML page, or fragment, with an application-specific API, allowing to manipulate page elements without digging around in the HTML.

Page Object


this API is about the application
selectAlbumWithTitle()
getArtist()
updateRating(5)
Page Objects
Album Page
Album List Page
this API is about HTML
findElementsWithClass('album')
findElementsWithClass('title-field')
getText()
click()
findElementsWithClass('ratings-field')
setText(5)
HTML Wrapper
title: Whiteout
artist: In the Country
rating:
title: Ouro Negro
artist: Moacir Santos
rating:

Page Object Usage

- Page objects are commonly used for testing, but should not make assertions themselves.
- Their responsibility is to provide access to the state of the underlying page.
- It's up to test clients to carry out the assertion logic.
- Page objects are most commonly used in testing, but can also be used to provide a scripting interface on top of an application

Thank You!