Web Application Testing with Protractor

Anisha Narang

Red Hat Inc. | @anisha_narang

DevConf.US, Boston 2018

Agenda

- Getting started with Protractor
- Jasmine
- Global variables and Element locators
- Understanding Promises and Control Flow
- Page Object Pattern
- Protractor Browser Logs
- Generating HTML reports
- Demo

What is Protractor?

An end-to-end test framework for Angular and AngularJS applications

Protractor is built on top of WebDriverJS

Protractor supports Angular-specific locator strategies

Getting started

Installation:

- Use npm to install Protractor globally

```
$npm install -g protractor

$protractor --version
> Version 5.1.1

$webdriver-manager update
$webdriver-manager start
```

- The webdriver-manager will help you to get an instance of a Selenium Server running at *http://localhost:4444/wd/hub*

Writing your first test

- Configuration file
- Spec file

Config file:

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

Spec file:

```
// spec.js
describe('Angular Sample application::', function() {
  it('The resulting text should match "Hello Anisha!"', function() {
    browser.get('https://angularjs.org/');
    element(by.model('yourName')).sendKeys('Anisha');

    expect(element(by.binding('yourName')).getText()).toEqual('Hello Anisha!');
  });
});
```

Running the test(s):

```
protractor conf.js
```

Understanding the essentials

Jasmine

Jasmine is a behavior-driven development framework for testing JavaScript code.

```
describe("Sample Application::", function() {
  it("should match the text", function() {
    var ele = element(by.id('text-intro'));

    expect(ele.getText()).toEqual("hello world");
  });
});
```

Global variables

browser: A wrapper around an instance of WebDriver, used for navigation and page-wide information.

---

element: A helper function for finding and interacting with DOM elements on the page you are testing.

---

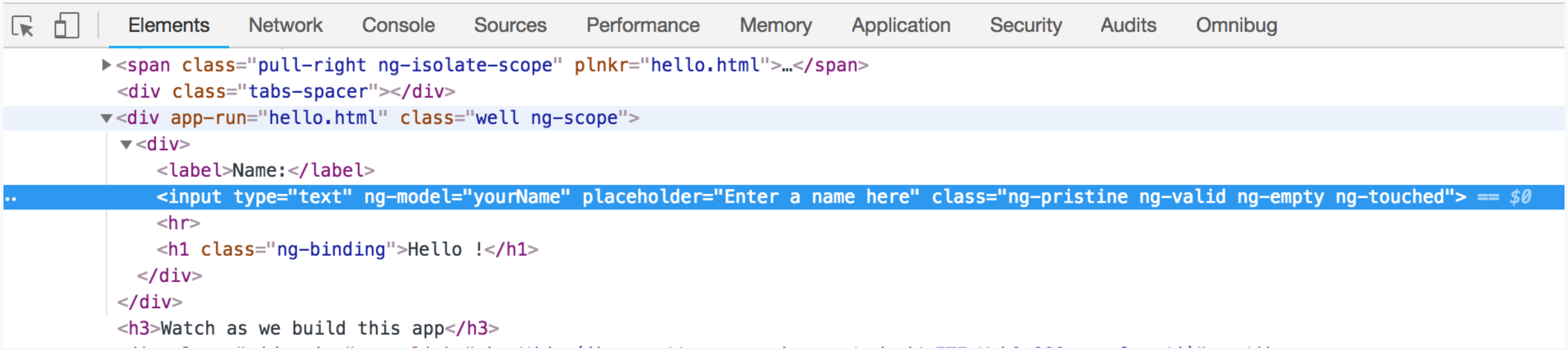
by: A collection of element locator strategies.

Element locators

- **by.css('.myclass')** : Find an element using a css selector
- **by.id('myid')** : Find an element with a given id.
- **by.model('name')**: Find an element with a certain ng-model(Angular specific).
- **by.binding('bindingname')**: Find an element bound to the given variable(Angular specific).

Locators are passed to the `element` function: `element(by.css('some-css'));`

```
element(by.model('yourName'));
```



```
▶<span class="pull-right ng-isolate-scope" plnkr="hello.html">…</span>
 <div class="tabs-spacer"></div>
▼<div app-run="hello.html" class="well ng-scope">
  ▼<div>
     <label>Name:</label>
     <input type="text" ng-model="yourName" placeholder="Enter a name here" class="ng-pristine ng-valid ng-empty ng-touched"> == $0
     <hr>
     <h1 class="ng-binding">Hello !</h1>
   </div>
 </div>
 <h3>Watch as we build this app</h3>
```

Actions

The element() function returns an ElementFinder object.

```
var ele = element(locator);

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

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

Since all actions are asynchrounous, all action methods return a promise.

```
//log the text of an element

var ele = element(locator);
ele.getText().then(function(text) {
  console.log(text);
});
```

Understanding Promises and Control Flow

- 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.

We dont need to write promises, the control flow mechanism of Protractor allows us to write promises in a synchronous way.

Promises

Promises represent the eventual result of an operation.

Promises are objects. The Promise object is used for asynchronous computations.

```
Example code for understanding promises:
```

```
var promise = $http.get("/api/docs/<id>");
promise.success(function(title) {
   console.log("Title of the published doc: " + title);
});
promise.error(function(response, status) {
   console.log("The request failed with response " + response + " and status code " + status);
});
```

Control Flow

Control Flow coordinates the scheduling and execution of commands operating on a queue.

Without Control flow:

```
browser.get(“http://www.google.com”).
then(function() {
  return element(by.id('q'));
}).
then(function(ele) {
  return ele.sendKeys('webdriver');
}).
then(function() {
  return element(by.name('btnG')).click();
})......
```

With Control flow:

```
browser.get(“http://www.google.com”);
element(by.id('q')).sendKeys('webdriver');
element(by.name('btnG')).click();
......
```

Page Object Pattern

A Page Object simply models the page elements as objects within the test code.

Reduces the amount of duplicate code.

If the UI changes, the fix needs to be done at only one place.

Example for Without Page Object:

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

With Page Object Pattern:

```
#PageObjectFile.js:

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.getGreetingText = function() {
    return greeting.getText();
  };
};
module.exports = new AngularHomepage();
```

```
#Test.spec:

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

    angularHomepage.setName('Anisha');

    expect(angularHomepage.getGreetingText()).toEqual('Hello
Anisha!');
  });
});
```

Protractor Browser Logs Assertion

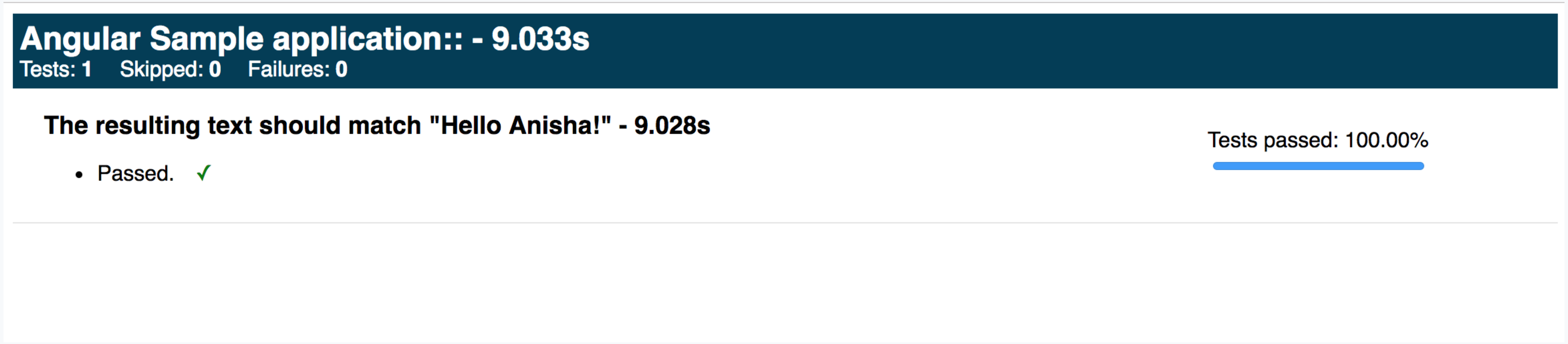
Allows asserting the browser console logs in each test for warnings and errors.

```
npm install protractor-browser-logs
```

```
var browserLogs = require('protractor-browser-logs');

var verifyBrowserConsoleLogs = function() {
  var logs = browserLogs(browser);

  beforeEach(function() {
    logs.reset();
  });

  afterEach(function() {
    return logs.verify();
  });
};
```

Generating reports

```
$ npm install jasmine-spec-reporter
$ npm install protractor-jasmine2-html-reporter
```



Angular Sample application:: - 9.033s

Tests: 1 Skipped: 0 Failures: 0

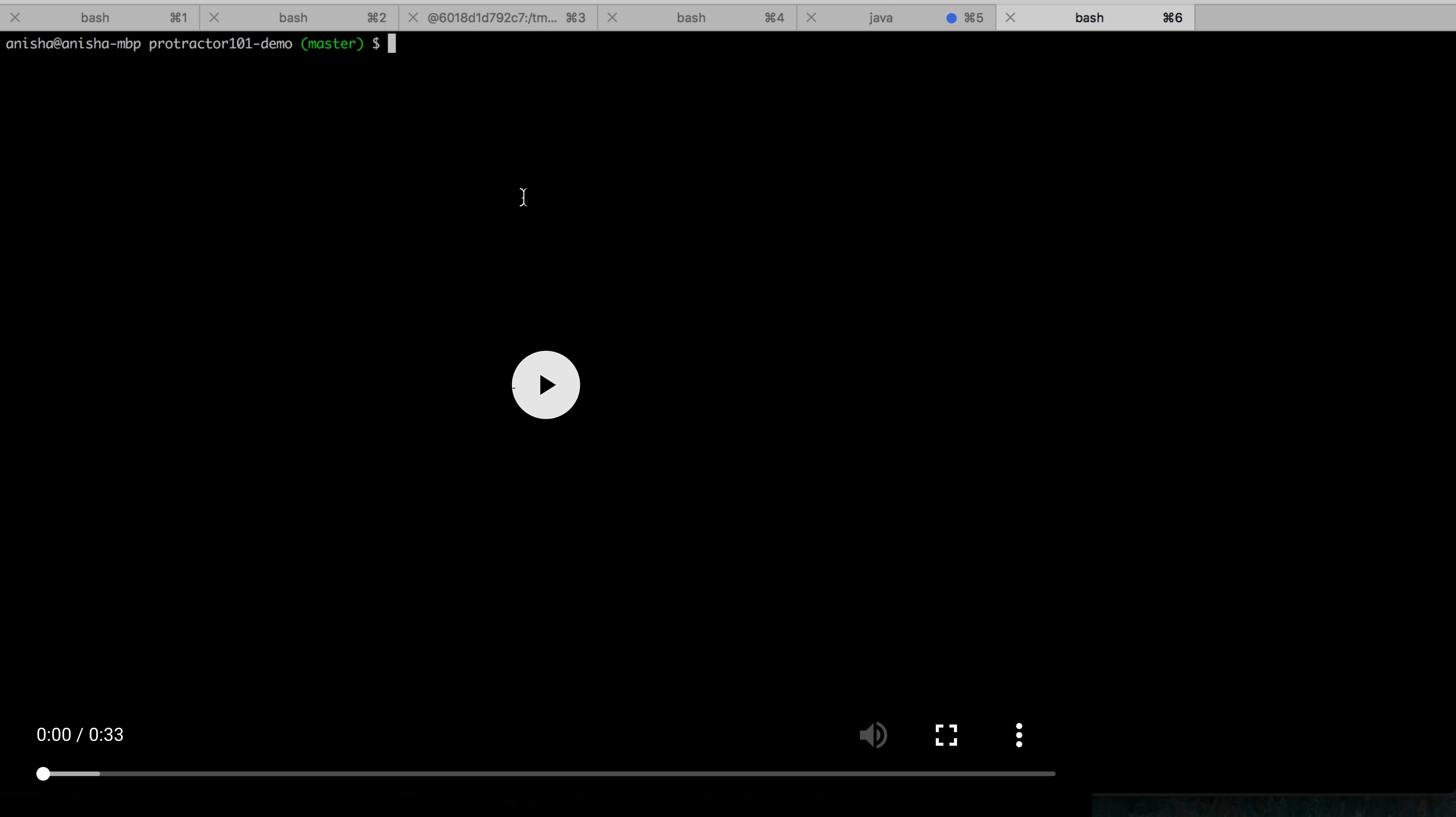
The resulting text should match "Hello Anisha!" - 9.028s

- Passed. ✓

Tests passed: 100.00%

Demo

Code available here: https://goo.gl/qJBiFH



```
anisha@anisha-mbp protractor101-demo (master) $
```

Any questions?

Thank You :-)

@anisha_narang

Slides available at:

http://anarang.github.io/protractor/index.html