

*END TO END TESTS
WITH PROTRACTOR*

Hugh McCamphill

Hugh McCamphill

- Test Lead with ShopKeep
- Organising Belfast Selenium Meetup since 2013
- Co-organizer of Belfast Software Testing Clinic
- @hughleo01

SETUP

Open <https://github.com/hughleo/protractor-workshop>
and follow instructions

At the end you should have running:

- Angular application (which we will run tests against)
- And a backing API (for the Angular app to talk to)

You should also have been able to run a test with:

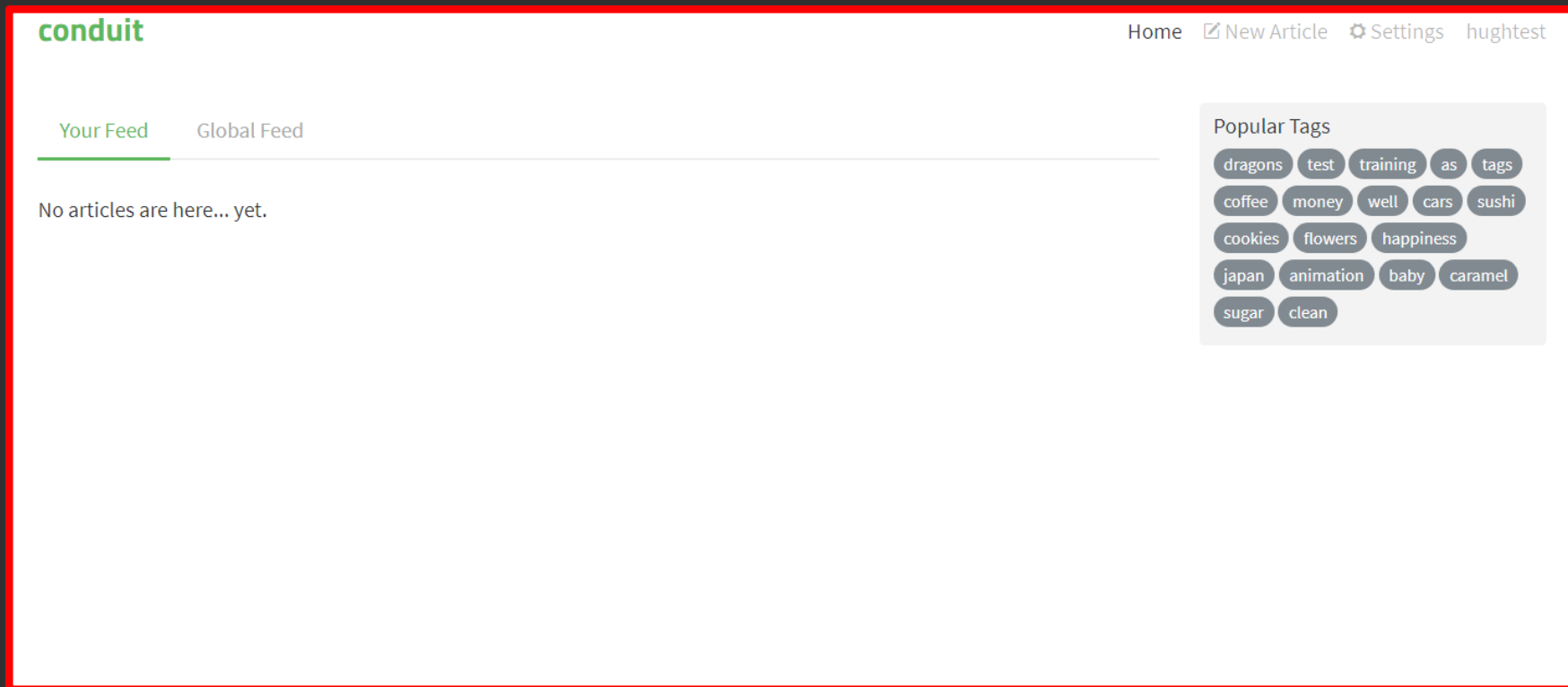
`protractor protractor.conf.js` (terminal / command line)

AGENDA

- Getting Setup
- Intro
- Configuration
- TypeScript
- Promises and Control Flow
- Page Objects and DSL
- Data Builder Pattern
- Wrapper methods
- Async / Await
- Close

Intro

The application you'll be working with

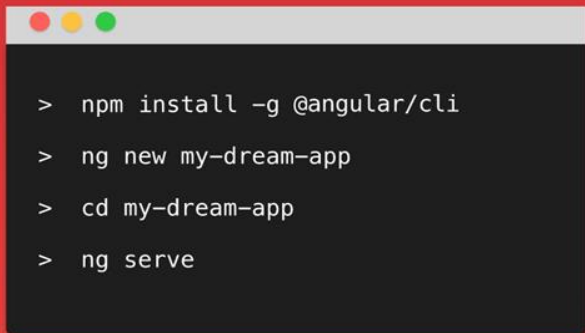




Spring Boot



Angular 4

A terminal window with a dark background and a light gray title bar containing three colored window control buttons (red, yellow, green). The terminal displays four lines of commands, each preceded by a prompt character '>'.

```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

Angular CLI

A command line interface for Angular

GET STARTED

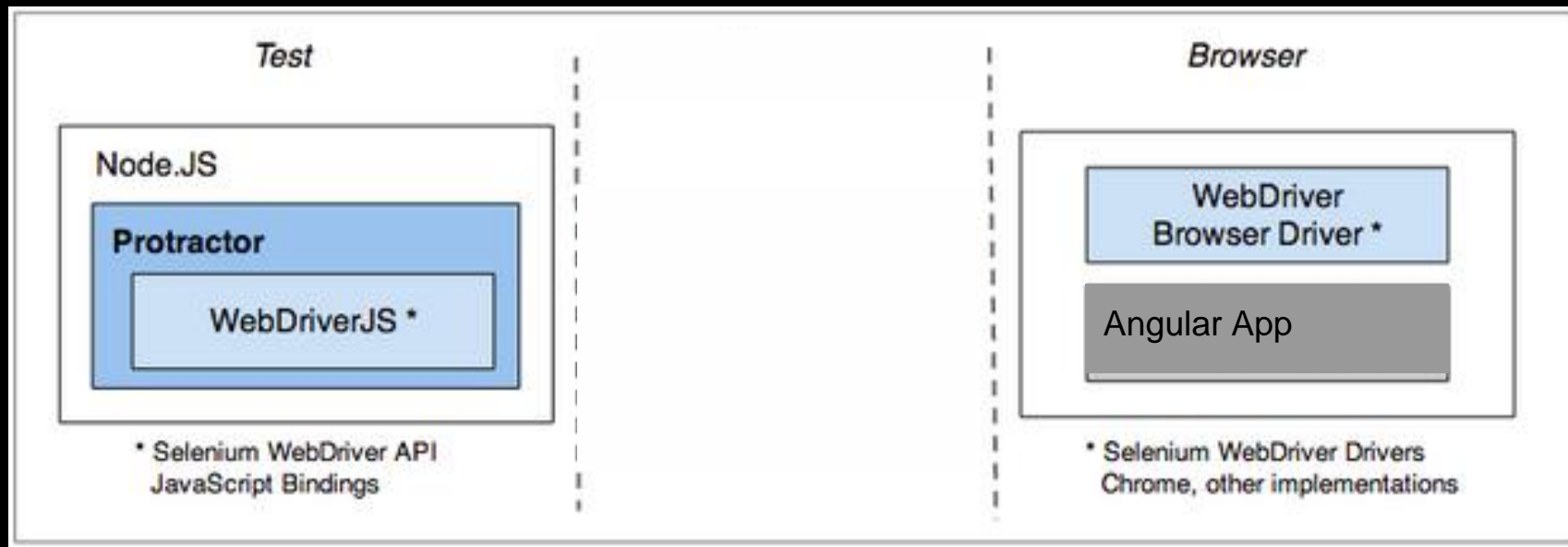
ng new

The Angular CLI makes it easy to create an application that already works, right out of the box. It already follows our best practices!

ng generate

Generate components, routes, services and pipes with a simple command. The CLI will also create simple test shells for all of these.

PROTRACTOR OVERVIEW



Webdriver Manager

[circleci](#)[passing](#)[gitter](#)[join chat](#)

A selenium server and browser driver manager for your end to end tests. This is the same tool as `webdriver-manager` from the [Protractor](#) repository.

Note: Version 9 and lower please reference [pose/webdriver-manager](#). If there are features that existed in version 9 and lower, please open up an issue with the missing feature or a create a pull request.

Getting Started

```
npm install -g webdriver-manager
```

DEMO

Running `should` navigate to create new article page in
`e2e/specs/basic.e2e-spec.ts`

Exercise: Run first test

Ensure you have a working email / password in params.

In terminal / command line:

>protractor protractor.conf.js

```
const { browser } = require('protractor');
const { SpecReporter } = require('jasmine-spec-reporter');

exports.config = {
  allScriptsTimeout: 11000,
  specs: [
    './e2e/**/*.e2e-spec.ts'
  ],
  capabilities: {
    'browserName': 'chrome'
  },
  directConnect: true,
  baseUrl: 'http://localhost:4200',
  framework: 'jasmine',
  jasmineNodeOpts: {
    showColors: true,
    defaultTimeoutInterval: 30000,
    print: function() {}
  },
  params: {
    user: {
      email: 'hughleo@email.com',
      password: 'password'
    }
  },
  onPrepare() {
    require('ts-node').register({
      project: 'e2e/tsconfig.e2e.json'
    });
    jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace: true } }));
  }
};
```

TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.

TYPESCRIPT

- Encapsulation through classes and modules
- Support for constructors, properties, functions
- Lambda style function support
- Provides static typing
- Intellisense and syntax checking

Jump To: [ajax.js](#) [boot.js](#) [custom_equality.js](#) [custom_matcher.js](#) [introduction.js](#) [node.js](#) [python_egg.py](#) [ruby_gem.rb](#) [upgrading.js](#)

introduction.js

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests. This guide is running against Jasmine version 2.0.0.

Standalone Distribution

The [releases page](#) has links to download the standalone distribution, which contains everything you need to start running Jasmine. After downloading a particular version and unzipping, opening `SpecRunner.html` will run the included specs. You'll note that both the source files and their respective specs are linked in the `<head>` of the `SpecRunner.html`. To start using Jasmine, replace the source/spec files with your own.

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

Exercise

Add a method to EditorPage page object

- `e2e/page-objects/basic/editor-page.po.ts`
- Adding elements for title, description, body and button
- Add method to enter these fields and click Publish

Exercise

Add test for adding article

- Go to `e2e/specs/basic.e2e-spec.ts` and finish the test `should add article`
- Use the page objects already created to complete the test

Demo : Page Objects

- Constructor for sync on page
- Return type for navigation

Promises (and adding expectations)

WebDriverJS is an asynchronous API, where every command returns a promise. This can make even the simplest WebDriver scripts very verbose. Consider the canonical "Google search" test:

```
let driver = new Builder().forBrowser('firefox').build();
driver.get('http://www.google.com/ncr')
  .then(_ => driver.findElement(By.name('q')))
  .then(q => q.sendKeys('webdriver'))
  .then(_ => driver.findElement(By.name('btnG')))
  .then(btnG => btnG.click())
  .then(_ => driver.wait(until.titleIs('webdriver - Google Search'), 1000))
  .then(_ => driver.quit(), e => {
    console.error(e);
    driver.quit();
  });
```

WebDriverJS uses a promise manager that tracks command execution, allowing tests to be written as if they were using a synchronous API:

```
let driver = new Builder().forBrowser('firefox').build();
driver.get('http://www.google.com/ncr');
driver.findElement(By.name('q')).sendKeys('webdriver');
driver.findElement(By.name('btnG')).click();
driver.wait(until.titleIs('webdriver - Google Search'), 1000);
driver.quit();
```

Demo: Promise returned from method

*Demoing in e2e/specs/promise.e2e-spec.ts
with 'should navigate to create new article'*

Exercise

Resolve promise when returning from a method

- Go to `e2e/specs/basic.e2e-spec.ts`
- Add expectation that article was added in `'should add article'`

The WebDriver Control Flow

The [WebDriverJS API](#) is based on [promises](#), which are managed by a [control flow](#) and adapted for [Jasmine](#). A short summary about how Protractor interacts with the control flow is presented below.

Disabling the Control Flow

In the future, the control flow is being removed (see [SeleniumHQ's github issue](#) for details). To disable the control flow in your tests, you can use the `SELENIUM_PROMISE_MANAGER: false` [config option](#).

Instead of the control flow, you can synchronize your commands with promise chaining or the upcoming ES7 feature `async / await`. See [/spec/ts/](#) for examples of tests with the control flow disabled.

Because `async / await` uses native promises, it will make the Control Flow unreliable. As such, if you're writing a library or plugin which needs to work whether or not the Control Flow is enabled, you'll need to handle synchronization using promise chaining.

Promises and the 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. For example, consider this test:

```
const { browser } = require('protractor');
const { SpecReporter } = require('jasmine-spec-reporter');

exports.config = {
  allScriptsTimeout: 11000,
  specs: [
    './e2e/**/*.e2e-spec.ts'
  ],
  SELENIUM_PROMISE_MANAGER: false,
  capabilities: {
    'browserName': 'chrome'
  },
  directConnect: true,
  baseUrl: 'http://localhost:4200',
  framework: 'jasmine',
  jasmineNodeOpts: {
    showColors: true,
    defaultTimeoutInterval: 30000,
    print: function() {}
  },
  params: {
    user: {
      email: 'hughleo@email.com',
      password: 'password'
    }
  },
  onPrepare() {
    require('ts-node').register({
      project: 'e2e/tsconfig.e2e.json'
    });
    jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace: true } }));
  }
};
```

Demo: Async / Await

Exercise: Run async tests

- Go to `e2e/specs/example.aysnc.e2e-spec.ts`
- Run 'should add article' test
- Note the order in which the operations happen

Exercise

Update page object to do operations in order

- Go to `e2e/page-objects/async/editor-page.po.ts`
- Add `async / await` operations
- Update return type to be a promise
- Run previous test again

Demo: Wrapping Methods

Exercise: Add a clear and send keys method and use it for editor page

- Go to `e2e/wrappers/element-wrapper.ts`
- Add the `clearAndSendKeys` method
- Add calling it in `updateArticleContent` in `e2e/page-objects/async/editor-page.po.ts`

Demo: domain models and builders

Exercise: Use domain object

Instead of passing information into article method one variable at a time, pass it in as an object

Exercises

- Add a test to
 - Add comments to an article
 - Add then delete an article
 - Add then edit an article

Recap

- Protractor
- Webdriver Manager
- TypeScript
- Promises and Control Flow
- Async / Await
- Page Objects and DSL
- Data Builder Pattern
- Wrapper methods

How to's

Waiting with sleeps

- If the sleep you have added is too short, then your test may fail
- If the sleep you have added is too long, then you are waiting unnecessarily, and this is a big deal when you start to scale your number of tests

Prefer explicit waits

```
browser.wait(protractor.ExpectedConditions.urlContains(url), 10000);
```

Ignoring Synchronization

If for example you had a login page that wasn't an angular page

```
signIn(user: User): HomePage {  
  
    browser.waitForAngularEnabled(false);  
  
    this.emailField.sendKeys(user.email);  
    this.passwordField.sendKeys(user.password);  
    this.signInField.click();  
  
    browser.waitForAngularEnabled(true);  
    return new HomePage();  
}
```


Using Generics for deciding navigation

```
clickYourFeed<T>(c: { new(): T; }): T {  
    this.yourFeed.click();  
    return new c();  
}
```

```
homePage = new HomePage();  
// if I was logged in  
homePage.clickYourFeed(YourFeed);  
// if I was not logged in  
homePage.clickYourFeed(SignIn);
```

Close out

Set `directConnect` to `false` to run tests through a grid / cloud service

```
capabilities: {
  'browserName': 'chrome'
},
directConnect: true,
baseUrl: 'http://localhost:4200',
framework: 'jasmine',
jasmineNodeOpts: {
  showColors: true,
  defaultTimeoutInterval: 30000,
  print: function() {}
},
params: {
  user: {
    email: 'hughleo@email.com',
    password: 'password'
  }
},
onPrepare() {
  require('ts-node').register({
    project: 'e2e/tsconfig.e2e.json'
  });
  jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace: true } }));
}
};
```

```
const { browser } = require('protractor');
const { SpecReporter } = require('jasmine-spec-reporter');
```

Amending baseUrl

```
],
capabilities: {
  'browserName': 'chrome'
},
directConnect: true,
baseUrl: 'http://localhost:4200',
framework: 'jasmine',
jasmineNodeOpts: {
  showColors: true,
  defaultTimeoutInterval: 30000,
  print: function() {}
},
params: {
  user: {
    email: 'hughleo@email.com',
    password: 'password'
  }
},
onPrepare() {
  require('ts-node').register({
    project: 'e2e/tsconfig.e2e.json'
  });
  jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace: true } }));
}
};
```