Look Ma’, No Hands!

UI Automation for Developers

Who am I?

Bret Shawn
Staff Software Engineer,
GE Digital

github/bshawn
bret.shawn@gmail.com

What are we talking about?

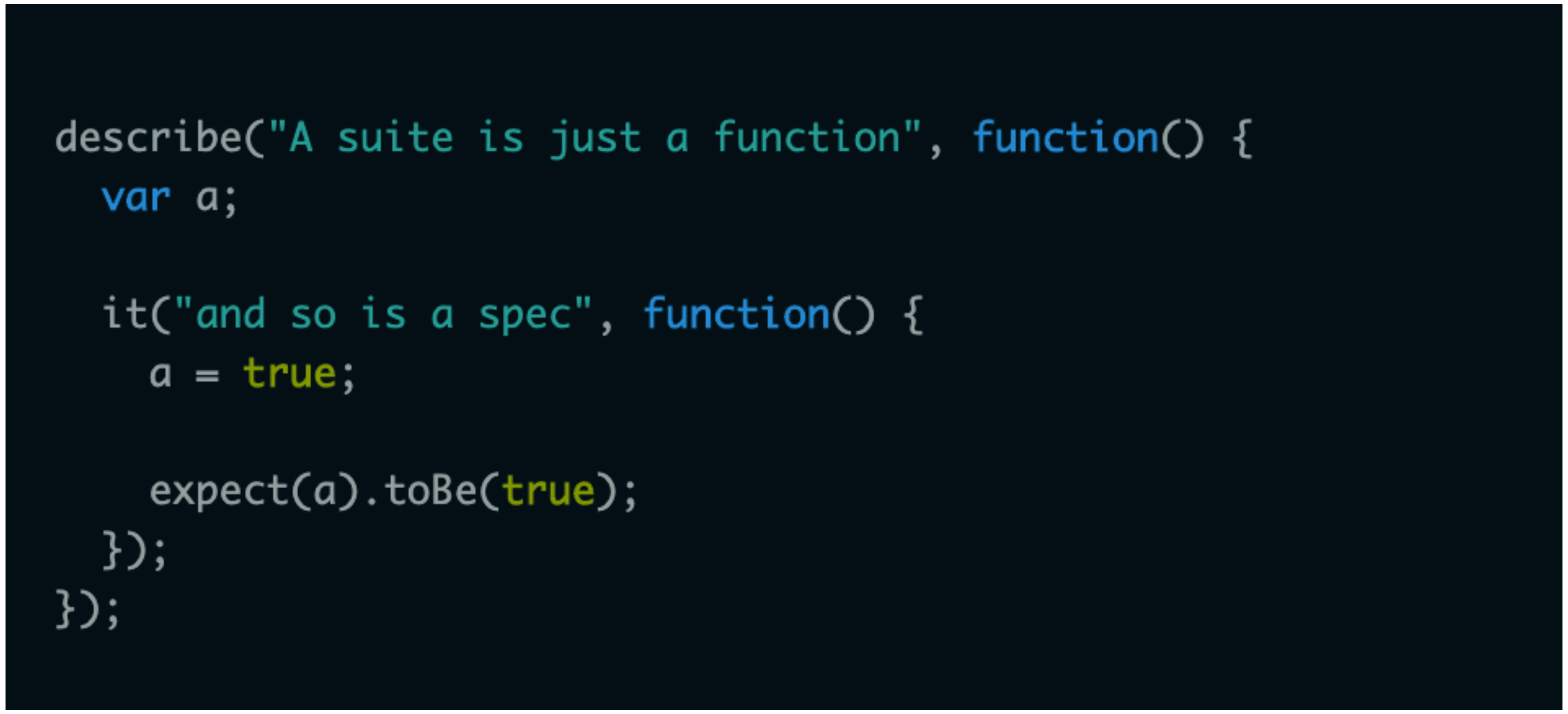
- UI Automation Testing is the task of automating the actions a user will take in the user interface of your application, so that we can make assertions about the outcomes of those actions.
- Actions we can automate:
    - Button clicks
    - Mouse movements (drag and drop)
    - Text entry
    - Etc…. Basically anything a user can do with the browser…
- Jasmine
- Protractor

Introduction to Jasmine

Jasmine

- Jasmine is a behavior-driven development framework for testing JavaScript code
- Jasmine Spec files have the following structure:
    - A Suite describes your test
    - A Spec performs an action and makes an assertion
- Suites and Specs are really just JavaScript function calls which accept a string and a callback

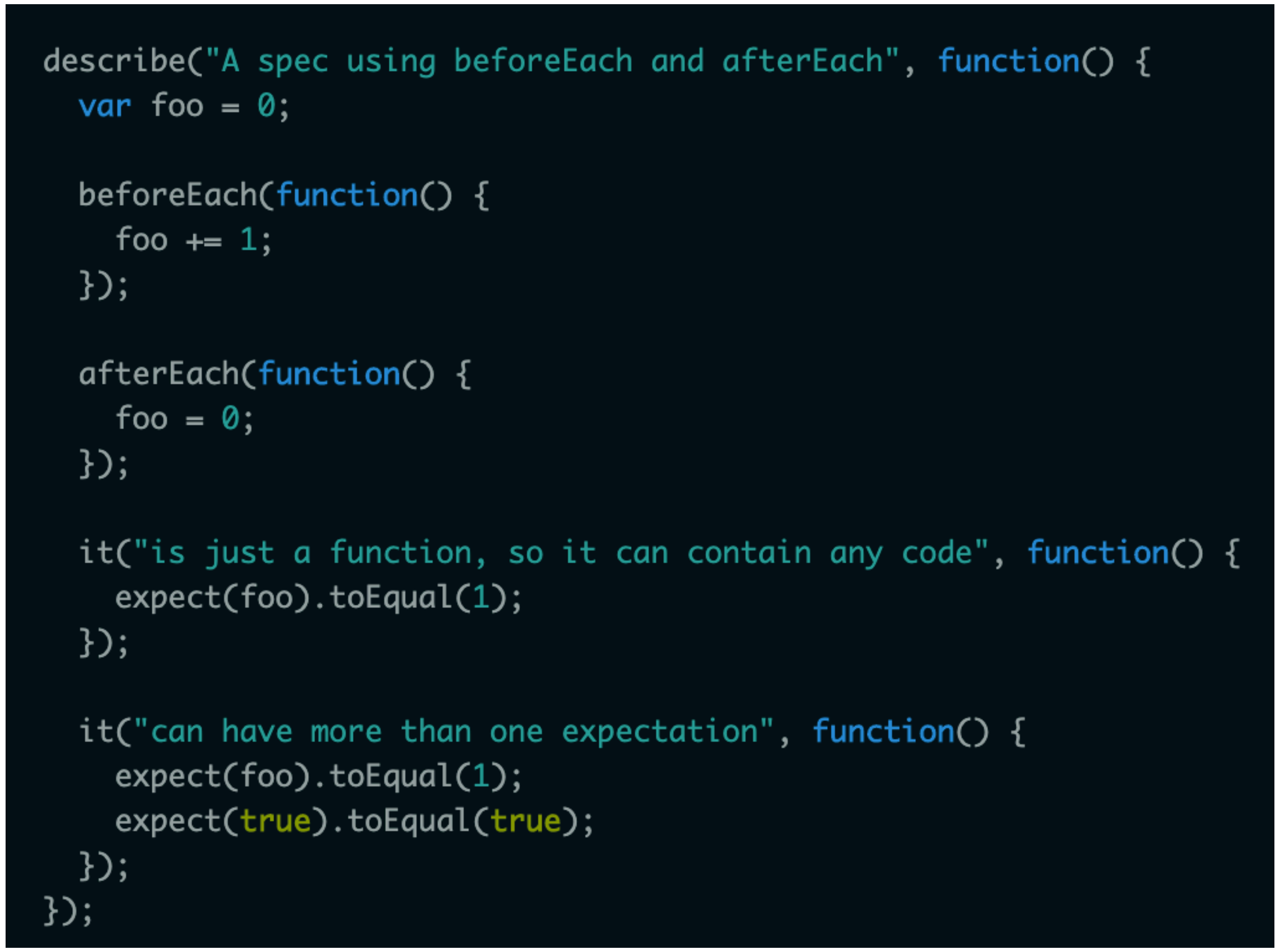
Test Structure



```
describe("A suite is just a function", function() {
  var a;

  it("and so is a spec", function() {
    a = true;

    expect(a).toBe(true);
  });
});
```

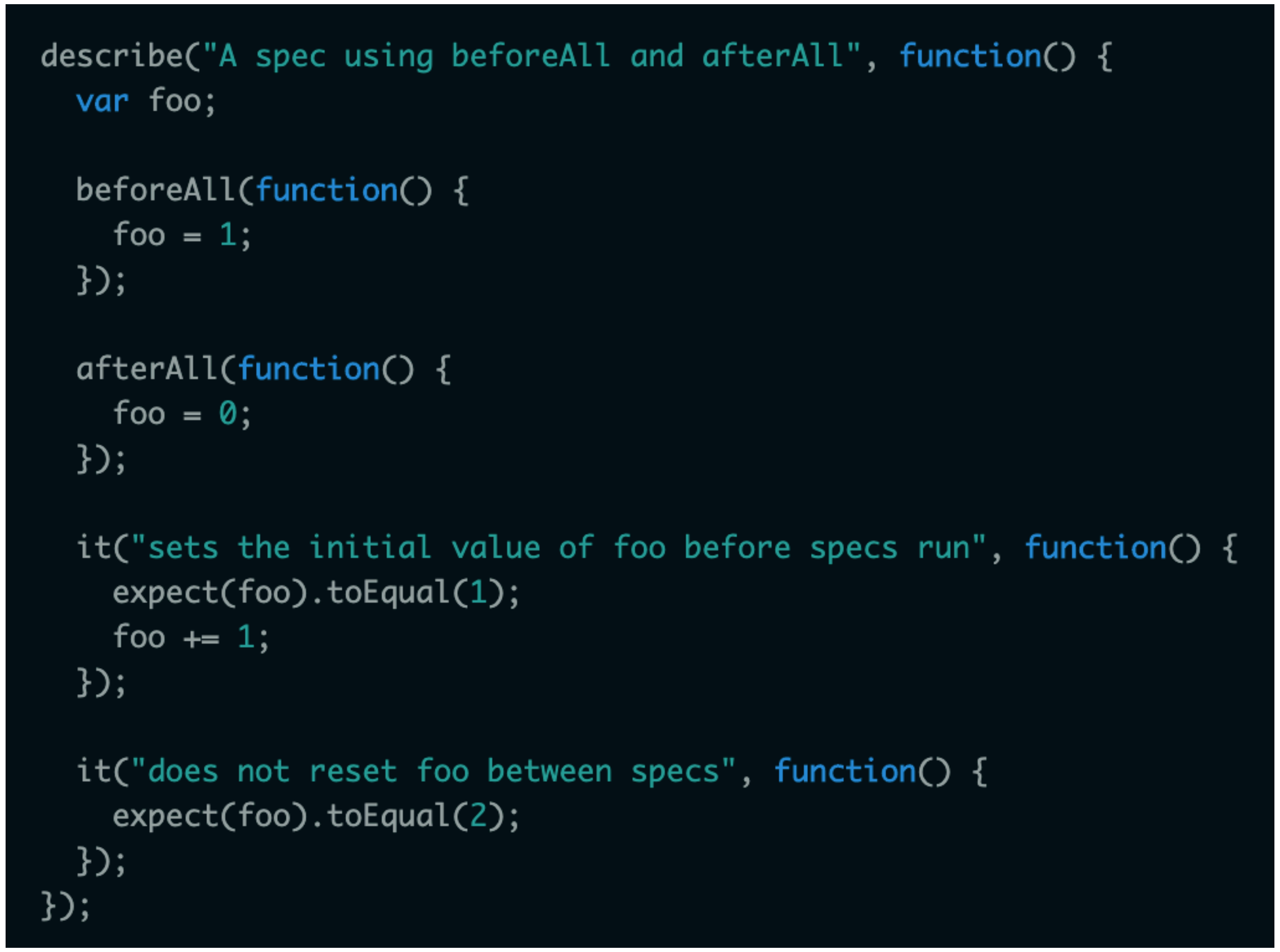
- Suites are defined inside the “describe” function call
- Specs are defined inside the “it” call
- Notice the hierarchical structure formed by the callbacks
- The “expect” call takes the actual value as an argument, chained with a Matcher function which takes an expected value

Test Setup and Teardown



```
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);
  });
});
```

Test Setup and Teardown

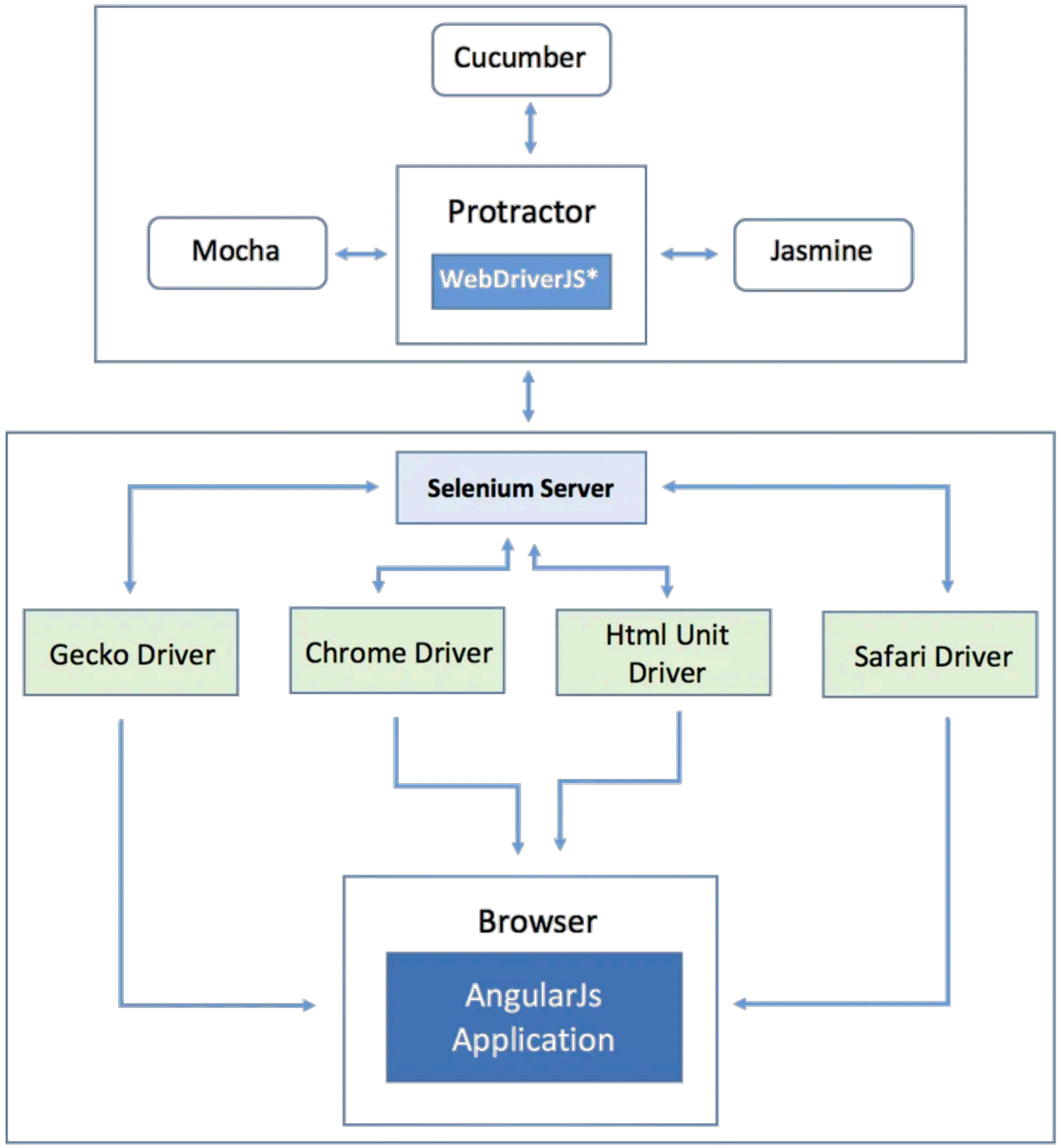


```
describe("A spec using beforeAll and afterAll", function() {
  var foo;

  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);
  });
});
```

Introduction to Protractor

What is Protractor?

- End-to-end test framework for Angular / AngularJS apps
- Written in 2013 by Julie Ralph - Senior Software Engineer in Test at Google
- Built on WebDriverJS / Selenium (web browser automation)
- https://github.com/angular/protractor
- Works for any web app, not just Angular…



Protractor Architecture

WebDriverJS

- This is how we “talk” with the browser to simulate user actions
- Asynchronous!!! (Promise-based, unless testing Angular)
- Angular testers can take advantage of Control Flow to avoid dealing with async

Protractor Globals

- element - Accepts a locator argument and provides the ability to interact with an element on the page. Knows how to find an element, ***but does contact the browser until an action method is called***.
- by - Creates a locator for finding elements on the page
  - by.css('div.mydiv')
  - by.xpath('//div[@class="mydiv"]')
  - by.model('todo') (Angular)
  - by.repeater('todo in todos') (Angular)
- browser - Provides the ability to interact directly with the browser
  - browser.get(url)
  - browser.sleep(5000)
- protractor - Wraps WebDriver and provides static utility functions

Usage

- element(by.css(‘#username’))

```
<input type=“text” id=“username” />
```

- element(by.model(‘username’))

```
<input ng-model=“username” />
```

- Interacting with element

```
element(by.css(‘.search-box’)).sendKeys(‘find this’);
```

Running tests

- conf.js
- spec.js
- command line interface

```
protractor conf.js
```

Demo

ExpectedConditions

- const EC = protractor.ExpectedConditions;
- Useful when testing non-Angular apps
- Returns a function that evaluates to a Promise
- When passed to the browser.wait() function, the wait promise will not resolve until the condition passes or the timeout expires
- Example:

```
const loginLink = element(by.css(".nav-link[href='#login']"));

const loginIsClickable = EC.elementToBeClickable(loginLink, timeout);
```

- Other expected conditions: alertIsPresent, textToBePresentInElement, textToBePresentInElementValue, titleContains, titleIs, urlContains, urlIs, **presenceOf**, stalenessOf, **visibilityOf**, invisibilityOf, elementToBeSelected
- Multiple ECs can be chained together using `.and`, `.or`, or `.not`

```
const loginIsPresent = EC.presenceOf(loginLink, timeout)

Const loginIsClickable = EC.elementToBeClickable(loginLink, timeout)

EC.and(loginIsPresent, loginIsClickable, timeout);
```

Organizing the Code

- Page Object pattern
  - Move elements and locators to a Page Object file
  - Separates the test from the UI, so maintainability is easier and code is more reusable
  - For small UI changes, only the Page Object will change
  - Share Page Object libraries between teams

Demo