JavaScript Unit Testing

Myth or Fact?

After writing your code, you'll never touch it again

Unit Testing Framework

- Test Runner / Reporter
  - Allows developers to describe and organize test cases
  - Run test cases, collect results
  - Analysis and Test Coverage
- Matchers
  - Assertions
  - Utility predicates for writing boolean expressions
- Test Doubles
  - Developer-managed replacement for external dependencies
  - Mocks, Stubs, Spies, ...

Reference(s)

- An Overview of JavaScript Testing in 2017

How Would You Test ______

- Access to remote resources when they do not exist
- Failed verification of user credentials
- Search data in local database when the query returns no results
- Add new data to your DB without actually altering the DB
- Handle timeout logic
- User login from a different timezone
- … many more ...

Mock Objects

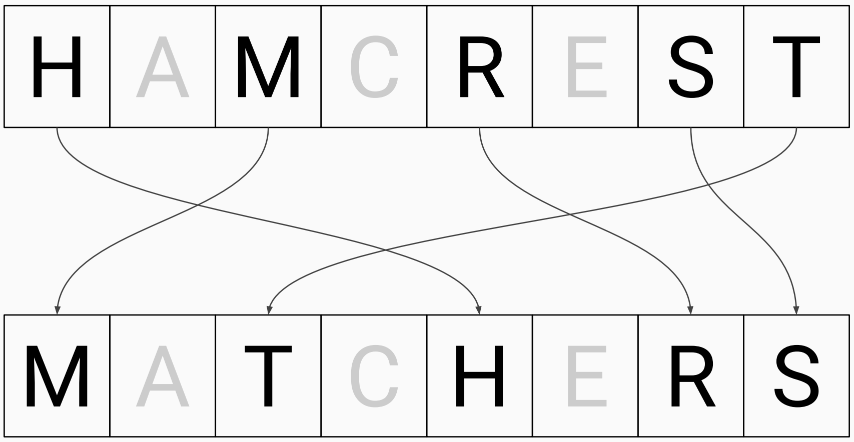
- Replacement objects for external depencencies
- Developers have full control how these replacement objects behave
  - User authentication succeeded or failed
  - HTTP status 200, 404, 401, …
  - Database queries with controlled results
  - AJAX requests with customized responses
  - Fake timer
  - Fake location provider
  - … and many more ...

Java Tools vs. JS Tools

- JUnit4 / JUnit5
  - Runner
  - Matchers
- Hamcrest
  - Matchers
- jMock, JMockit, EasyMock, Mockito,
- Others....

- Chai
- Istanbul
- Jasmine
- Jest
- Karma
- Mocha
- Sinon

Anagram



JavaScript Overview

	Test Runner	Assertions / Matchers	Test Doubles	Coverage
Chai		✓		
Istanbul				✓
Jasmine	✓	✓	✓	
Jest	✓	✓	✓	✓
Karma (browser)	✓			
Mocha	✓			
Sinon			✓	

Which One?

What to Test?

- Data presentation
- User interactions
- Server communications
- Storage (local/remote)
- Application State

How to write tests?

Test Structures

- Test-Driven Development (TDD)
    - Easily understood by programmers
    - `suite(), test()`
    - `assertEquals (27, x);`
- Behavior-Driven Development (BDD)
    - Easily understood by non-programmers
    - `describe(), it()`
    - `x.should_be(27);`

TDD Example: JUnit Test Suites and Annotations

```
public class TestSuiteSample {
  @Before
  public void setup() { // runs before each test case
    calc = new Calculator();
  }

  @Test public void addTwoIntegers() {
    assertEquals (14, calc.add(6, 8));
  }

  @Test public void addTwoFloats() {
    assertEquals (14.0, calc.add(6.0, 8.0), 1E-3);
  }

  @After public void cleanup() {
    // runs after each test case
  }
}
```

- *Java annotations (@Before, @Test, @After) for organizing / describing test cases*
- *Assertion methods to verify the object under test meets the requirements*

Jasmine

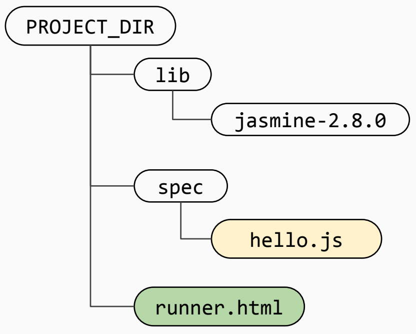
https://jasmine.github.io/2.8/introduction.html

Jasmine

- Behavior Driven Development
- Latest version: Jasmine 2.8
    - 2010 - 2012: v1.0 - v1.3
    - 2013: Dec v2.0
    - 2014: Nov v2.1
    - 2015: Feb v2.2, Apr v2.3, Dec v2.4
    - 2016: Aug v2.5
    - 2017: Apr v2.6, Jul v2.7, Aug v2.8 (lastest and last release of version 2.x)
- https://jasmine.github.io/2.0/introduction.html

Using Jasmine

1. Download and unzip Jasmine **Standalone** from https://github.com/jasmine/jasmine/releases
2. Copy the lib directory to your project directory



```
<!-- runner.html -->
<html>
<head>
  <link rel=”shortcut icon”
        href=”lib/jasmine-2.8.0/jasmine_favicon.png”>
  <link rel=”stylesheet”
        href=”lib/jasmine-2.8.0/jasmine.css”>
  <script src=”lib/jasmine-2.8.0/jasmine.js”></script>
  <script src=”lib/jasmine-2.8.0/jasmine-html.js”></script>
  <script src=”lib/jasmine-2.8.0/boot.js”></script>

  <script src=”spec/hello.js”></script>
</head>
<body>
</body>
</html>
```

Jasmine BDD

```
describe(“Description of your test suite”, function() {

  it(“Description of your test case”, function() {
    // function body of test case
  });

});
```

```
describe(“Description of your test suite”, () => {

  it(“Description of your test case”, () => {
    // function body of test case
  });

});
```

JUnit TDD vs. Jasmine BDD

```
// in TestSuiteSample.java
public class TestSuiteSample {
  private Calculator calc;

  @Before
  public void setup() {
    calc = new Calculator();
  }

  @Test public void addTwoIntegers() {
    assertEquals (14, calc.add(6, 8));
  }

  @Test public void addTwoFloats() {
    assertEquals (14.0, calc.add(6.0, 8.0), 1E-3);
  }

  @After public void cleanup() {
    // runs after each test case
  }
}
```

```
// In calctest.js
describe(“Sample Test Suite”, () => {
  var calc;

  beforeEach(() => {
    calc = new Calculator();
  }

  it(“adds two integers”, () => {
    expect (calc.add(6, 8)).toEqual(14);
  }

  it(“adds two floats”, () => {
    expect(calc.add(6.0, 8.0))
      .toBeCloseTo(14.0, 3);
  }

  afterEach( () => {
    // runs after each test case
  }
});
```

Jasmine “Hello World”

```
// in spec/hello.js
describe(“Hello Suite”,
  function() {

    it(“says hello”,
      function() {
        expect (“Hello World”).toContain(“or”);
      }
    );

  }
);
```

Jasmine “Hello World”

```
// in spec/hello.js
describe(“Hello Suite”,
  () => {

    it(“says hello”,
      () => {
        expect (“Hello World”).toContain(“or”);
      }
    );

  }
);
```

Jasmine Demo: Hello World

Git sha1: 561b96

Jasmine Matchers

```
expect(___).to____();
expect(___).toBe____();
expect(___).toHave____();
```

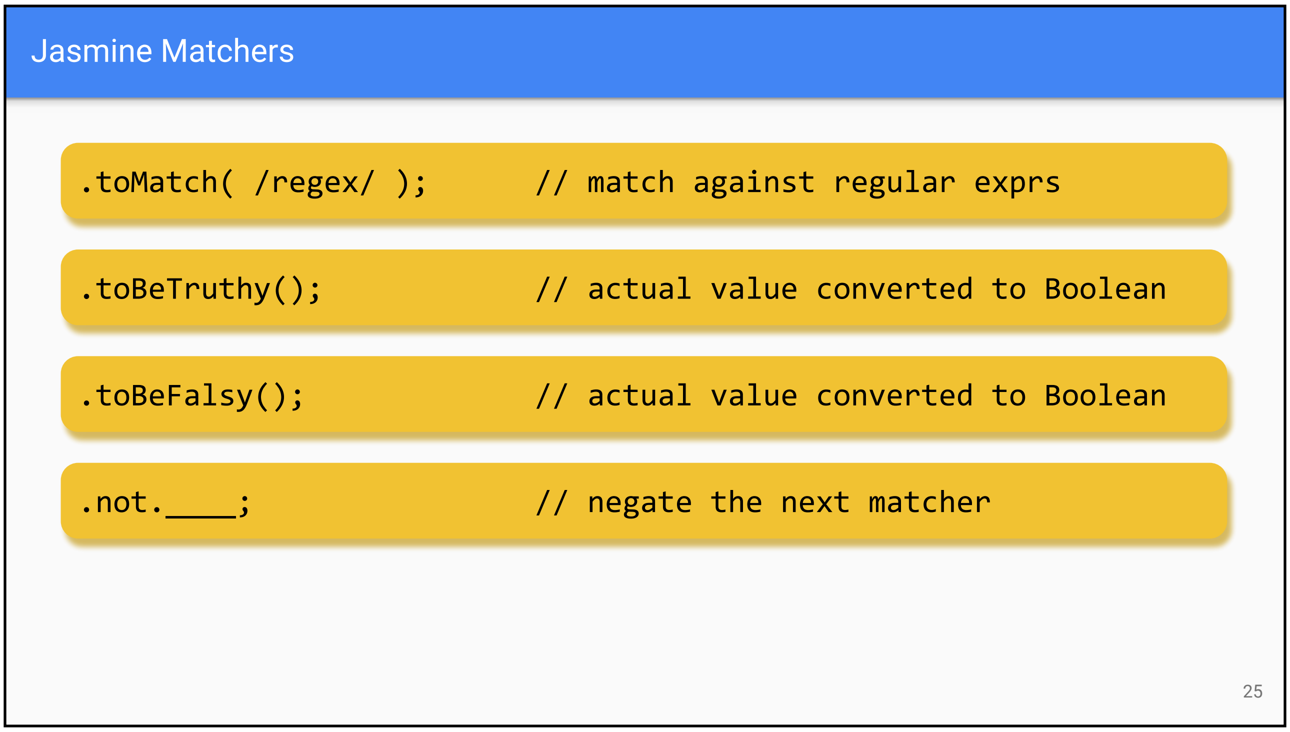
Jasmine Evaluator and Matchers

```
expect(actualvalue)               // evaluate
  .matcher1(expectedvalue1)       // compare
  .not.matcher2(expectedvalue2)   // compare
  .______
  .matcherN(expectedvalueN);      // compare
```

- *Matchers can be chained together*
- *Use .not to negate the matcher logic*

Jasmine Matchers

```
.toBe( expectedValue );     // exact compare using ===
.toEqual( expectedValue );  // can compare object equality
.toBeNull();                // check nullity
.toBeDefined();             // is variable defined?
.toBeUndefined();           // is variable undefined?
```



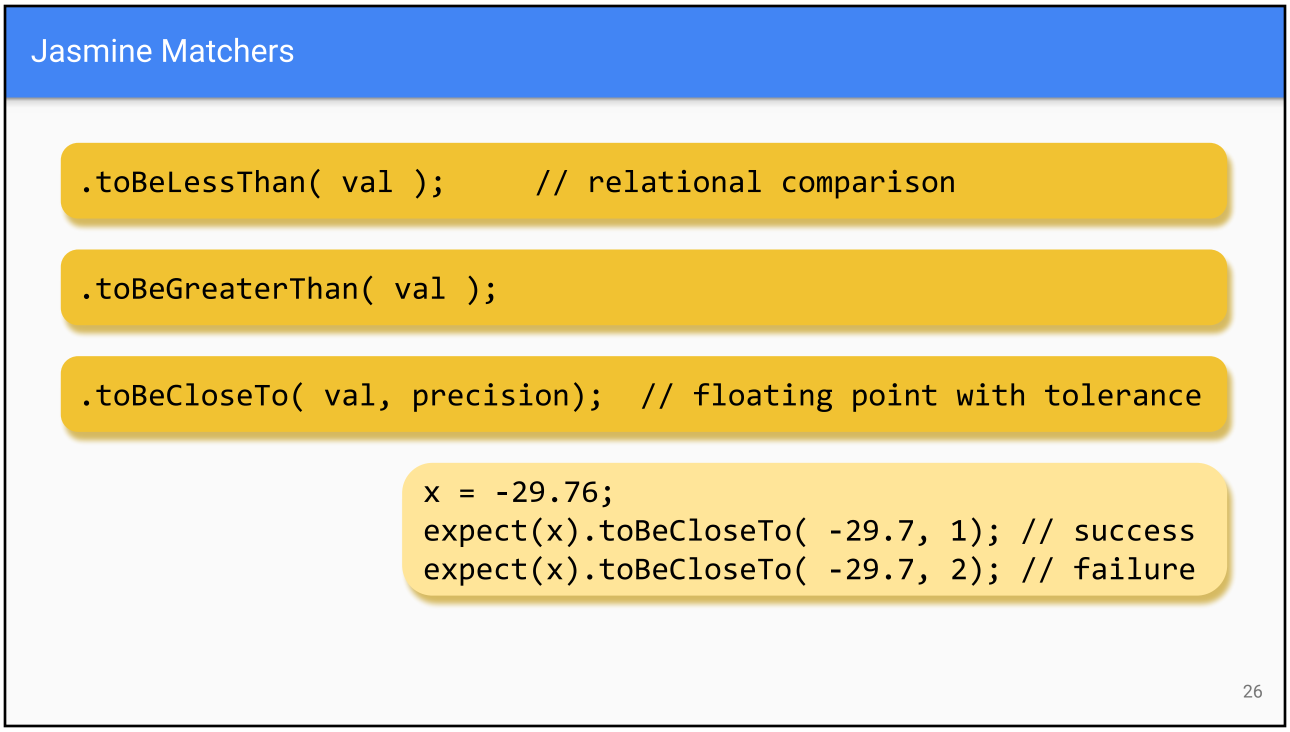
Jasmine Matchers

```
.toMatch( /regex/ );       // match against regular exprs
```

```
.toBeTruthy();             // actual value converted to Boolean
```

```
.toBeFalsy();              // actual value converted to Boolean
```

```
.not.____;                 // negate the next matcher
```



Jasmine Matchers

```
.toBeLessThan( val );      // relational comparison
```

```
.toBeGreaterThan( val );
```

```
.toBeCloseTo( val, precision);  // floating point with tolerance
```

```
x = -29.76;
expect(x).toBeCloseTo( -29.7, 1); // success
expect(x).toBeCloseTo( -29.7, 2); // failure
```

What to test?

(1) Data Presentation

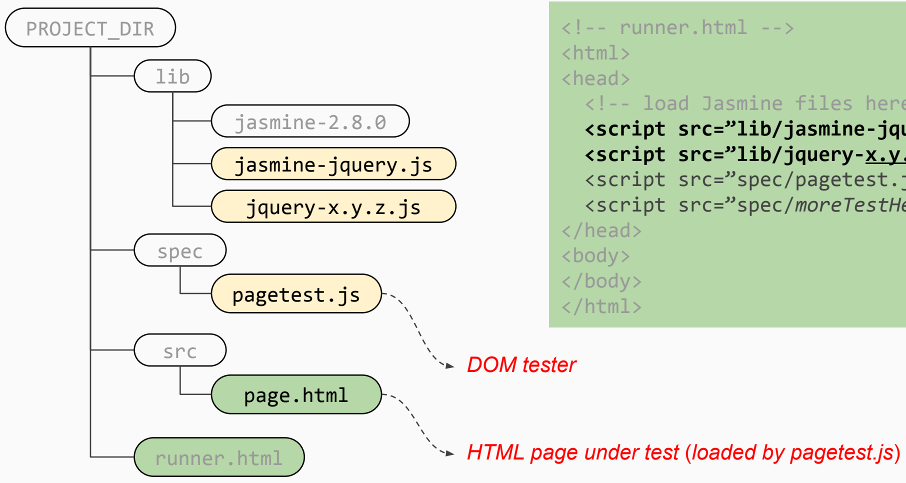
Jasmine jQuery

https://github.com/velesin/jasmine-jquery

Matchers for testing DOM

Using Jasmine jQuery

Download `jasmine-jquery.js` from https://github.com/velesin/jasmine-jquery

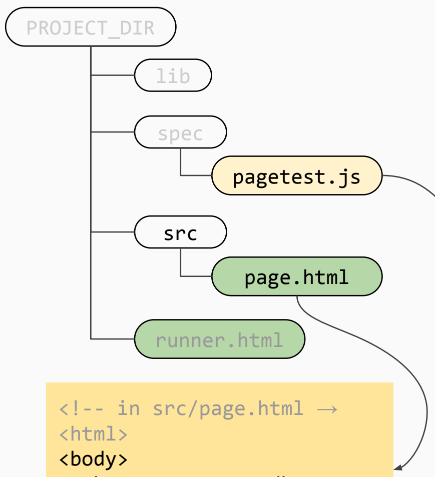


```
<!-- runner.html -->
<html>
<head>
  <!-- load Jasmine files here -->
  <script src="lib/jasmine-jquery.js"></script>
  <script src="lib/jquery-x.y.z.js"></script>
  <script src="spec/pagetest.js"></script>
  <script src="spec/moreTestHere.js"></script>
</head>
<body>
</body>
</html>
```

Jasmine jQuery Fixtures

- Jasmine jQuery needs a DOM object to test against
- HTML files under test ("target HTML") are placed **separately** from `runner.html`.
  - Default path for target files: `spec/javascripts/fixtures`, but can be configured to your own settings:
    `jasmine.getFixtures().fixturePath = "path/to/dir/of/html/files/";`
  - Use `loadFixtures()` to load HTML files into Jasmine workflow
  - Use jQuery `$()` to access elements in the target HTML files

Writing Jasmine jQuery Tests



```
<!-- in src/page.html →
<html>
<body>
  <h1>A test page</h1>
</body>
</html>
```

```
// in spec/pagetest.js
describe(“HTML Test Sample”, () => {
  beforeAll( () => {
    jasmine.getFixtures().fixturesPath = “src/”;
  });

  beforeEach(() => {
    loadFixtures(“page.html”);
  });

  it (“shows level1 heading”, () => {
    const el = $(‘h1’);
    expect(el).toExist();
    expect(el).toBeVisible();
    expect(el).toContainText(“age”);
    expect(document).not.toContainElement(‘form’);
  });
});
```

Jasmine jQuery DOM Matchers

```
.toBeChecked();             // element has checked attribute?
.toBeDisabled();            // is element disabled?
.toBeHidden();              // is element hidden?
.toBeVisible();             // is element visible?
.toExist();                 // does element exist?
```

Complete reference at: https://github.com/velesin/jasmine-jquery

Jasmine jQuery DOM Matchers

```
.toBeSelected();              // is element selected?
```

```
.toContain(string);
```

```
.toContainElement(css-selector);  // inspect DOM trees
```

```
.toContainText(string);      // has a child text node
                             // with specific content?
```

```
.toContainHtml(string);
```

Complete reference at: https://github.com/velesin/jasmine-jquery

Jasmine jQuery DOM Matchers

```
.toHaveClass(className);
```

```
.toHaveCss(cssObject);
```

```
.toHaveId(id);
```

```
.toHaveProp(propName, propValue);
```

Complete reference at: https://github.com/velesin/jasmine-jquery

Practical Use of DOM Matchers

- Are the buttons labelled with the right text?
- Are the checkboxes checked or unchecked?
- Are the warning messages rendered using the right style?
- Are the elements assigned the right classes? Attributes?
- Are the texts rendered in bigger font?
- Is the error message displayed as a child of a particular element?

Demo:
User Interactions & DOM Matchers

Git sha1: a07ce8, 0dcee8

What to test?

(2) User Interactions

Mocked User Interactions

User Interactions

- Verifying that your app responds correctly upon user inputs
  - Does the dialog show up when the user selected option X?
  - Is the data stored into the right place when the user hits “Save”?
  - Is the counter updated correctly when the user deletes selected items?
  - ...
- And we want to *automate these interactions* programmatically in code

Simulate User Interactions Programmatically with jQuery

```
<!-- HTML -->
<input id=”okBtn” type=”button” value=”Go”>
```

```
/* jQuery / JavaScript */
$(“#okBtn”).click();
```

```
<!-- HTML -->
<input id=”city” type=”text”>
```

```
/* jQuery / JavaScript */
$(“#city”).val(“New York”);
```

```
<!-- HTML -->
<select id=”lakes”>
  <option value=”Hu”>Huron</option>
  <option value=”On”>Ontario</option>
  <option value=”Mi”>Michigan</option>
  <option value=”Er”>Erie</option>
  <option value=”Su”>Superior</option>
</select>
```

```
/* jQuery / JavaScript */
$(“#lakes”).val(“Mi”);
```

Simulate User Interactions Programmatically with jQuery

```
<!-- HTML -->
<input type=”radio” name=”lakes” id=”Hu” value=”Huron” />
<input type=”radio” name=”lakes” id=”On” value=”Ontario” />
<input type=”radio” name=”lakes” id=”Mi” value=”Michigan” />
<input type=”radio” name=”lakes” id=”Er” value=”Erie” />
<input type=”radio” name=”lakes” id=”Su” value=”Superior” />
```

```
/* jQuery / JavaScript */
$(‘#Mi’).prop(“checked”, true);
```

Writing Jasmine jQuery Tests

```
<!-- in src/page.html -->
<html>
<body>
  <h1>A test page</h1>
  <input id=”btn” type=”button” value=”Go”>
  <span id=”msg”>
     This is a short text
  </span>
  <script>
   var msgEl = $(‘#msg’);
   var btnEl = $(‘#btn’);

   msgEl.hide();
   btnEl.click( () => {
     msgEl.show();
   });
  </script>
</body>
</html>
```

```
// in spec/pagetest.js
describe(“HTML Test Sample”, () => {
  beforeAll( () => {
    jasmine.getFixtures().fixturesPath = “src/”;
  });

  beforeEach(() => {
    loadFixtures(“page.html”);
  });

  it (“shows hidden message on click”, () => {
    const mspan = $(‘#msg’);
    expect(mspan).toBeHidden();

    $(‘#btn’).trigger(‘click’);

    expect(mspan).toBeVisible();
  });
});
```

Demo: Trigger Events

Git sha1: 0e433e

(Function) Spies

Spies: Test Double Functions

spy:

1. A person who secretly *collects and reports information on the activities*, movements, and plans of an enemy or competitor

Matchers vs. Spies

- Compare **actual** values against expected values
- Wide range of data types to compare
  - Number, string, boolean
  - Arrays, Objects (equality vs identity)
  - DOM elements
  - Functions
  - ...
- These **actual** values are usually output of a **function**

- Instead of evaluating function output, spies **monitor function activities**
  - Are functions invoked at all?
  - Are they invoked with the right args?
  - Do they trigger exceptions?
  - How many times they are invoked?
  - What are the input arguments on the most recent call?
- Peek into function call graph

Why Function Spies?

- Your common “ritual” in using a program *debugger*
  - Setup breakpoints in *strategic spots* throughout your program
  - Watch how variables change over time
  - Trace flow of execution: **Step over**, **Step Into**
- How do you come up with these strategic spots?
  - Conditional statements, loops
  - **Function calls**
- *Skipped functions ⇒ something is wrong*

Jasmine: Setup (Function) Spies

*Use **`window`** as “objName” when spying on global JavaScript function*

`spyOn(objName, ‘funcName’)`	Create a spy for an **existing** function in a given object and **replace** the function with the spy
`spyOn(___).and.callThrough()`	Create a spy but also call the original function
`spyOn(___).and.returnValue(___)`	Create a spy and replace the original function with a spy that always returns the provided value
`spyOn(___).and.callFake(fakeFn)`	Create a spy and replace the original function with the providefd fake function
`obj.fSpy = jasmine.createSpy()`	Create a bare spy that is **not linked to any existing functions**.

Jasmine: Spy Matchers

`expect(objName.funcName).toHaveBeenCalled()`	Verify if a spied function was invoked
`expect(_____).toHaveBeenCalledWith(args)`	Verify if a spied function was invoked with particular arguments

`objName.funcName.calls`	Property that records call history to a spy
`objName.funcName.calls.count()`	Number of calls
`objName.funcName.calls.argsFor(index)`	Arguments passed to a particular invocation (0 ⇒ first invocation)

Spy Example

```
<!-- in src/page.html -->
<html>
<body>
  <input id=”card” type=”text” name=”cardnum”>
  <input id=”btn” type=”submit” value=”Pay”>
  <script>
   function payWith (x) {
     /* code here */
   }

   $(‘#btn’).click( () => {
     const cc = $(‘input[name]’).val();

     // remove non-digit characters
     const ccnum = cc.replace(/[^\d]+/g, ‘’);
     payWith(ccnum);
   });
  </script>
</body>
</html>
```

```
// in spec/pagetest.js
describe(“HTML Test Sample”, () => {

  beforeEach(() => {
    loadFixtures(“page.html”);
  });

  it (“calls payWith() with numeric string”,
    () => {
      $(‘input[name]’).val(“KY#478DD12”);
      spyOn(window, ‘payWith’);
      $(‘#btn’).trigger(‘click’);
      expect(window.payWith)
        .toHaveBeenCalledWith(“47812”);
  });

});
```

Demo: Function Spies

Git sha1: 94a13d

Testing Asynchronous Code

Async Code: Callbacks

```
var tabRef;
describe("Firebase test", () => {
  beforeEach(() => {
    tabRef = firebase.database()
                  .ref("____");
  });

  it('contains data', function() {
    tabRef.on("value", snapshot => {
      expect(snapshot).not.toBeNull();
    });
    /* finished too soon */
  });
});
</script>
```

Does not work, it() finished too soon

```
var tabRef;
describe("Firebase test", () => {
  beforeEach(() => {
    tabRef = firebase.database()
                  .ref("____");
  });

  it('contains data', function(done) {
    tabRef.on("value", snapshot => {
      expect(snapshot).not.toBeNull();
      done();
    });
  });
});
</script>
```

53

Async Code: Promises

```
var tabRef;
describe("Firebase test", () => {
  beforeAll(() => {
    tabRef = firebase.database().ref();
  });

  beforeEach(() => {
    tabRef.child("/config").set({num: 50});
  });

  it("Sample test", () => {
    /* test code that depends on /config */
  });
});
```

```
var tabRef;
describe("Firebase test", () => {
  beforeAll(() => {
    tabRef = firebase.database().ref();
  });

  beforeEach((done) => {
    tabRef.child("/config").set({num: 50});
      .then(() => {
         done();
       });
  });

  it("Sample test", () => {
    /* test code that depends on /config */
  });
});
```

Does not work, it() finished too soon

54

Jasmine AJAX

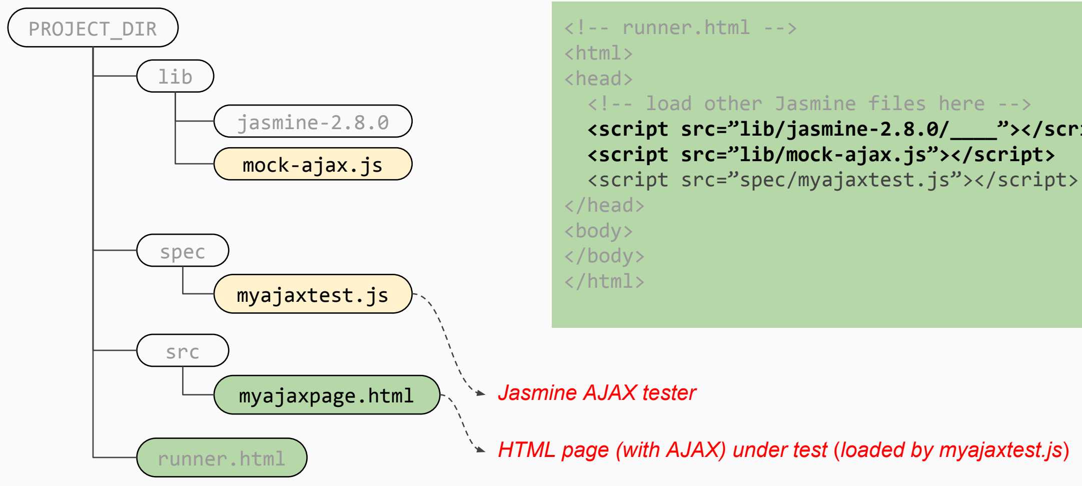
https://jasmine.github.io/2.8/ajax.html

Jasmine AJAX

- Jasmine plugin for testing AJAX calls
- Mocked implementation of the `XMLHttpRequest` class
- Reference: https://jasmine.github.io/2.8/ajax.html
- Script to include: mock-ajax.js
    - Call jasmine.Ajax.install() in beforeEach(), call jasmine.Ajax.uninstall() in afterEach()
- Inside each `it()` function
    - Call `jasmine.Ajax.requests.mostRecent()` to access the most recent AJAX request
    - Stub out the **desired** HTTP response of the request

Using Jasmine AJAX

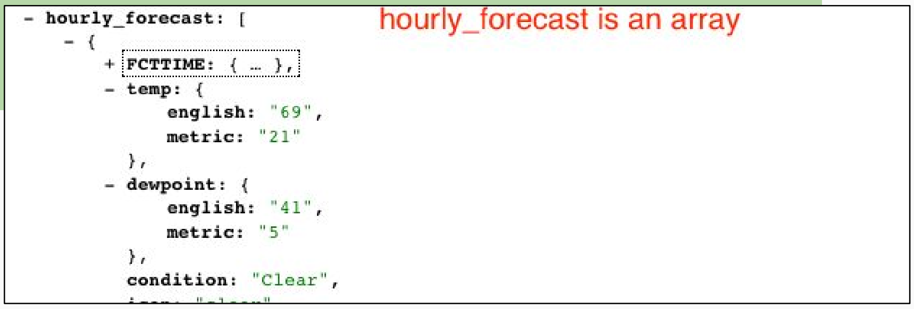
Download `mock-ajax.js` from https://github.com/jasmine/jasmine-ajax/tree/master/lib



```
<!-- runner.html -->
<html>
<head>
  <!-- load other Jasmine files here -->
  <script src="lib/jasmine-2.8.0/____"></script>
  <script src="lib/mock-ajax.js"></script>
  <script src="spec/myajaxtest.js"></script>
</head>
<body>
</body>
</html>
```

AJAX Test Example

```
<!-- myajaxpage.html -->
<button type="button" onclick="fetchWeather()">Go!</button>
<script>
function fetchWeather() {
  const url = "http://api.wunderground.com/api/${APIKEY}/hourly/q/MI/Allendale.json";
  $.get(url, (json, status, xhr) => {
    if (status == "success") {
      $('#temperature').text (json.hourly_forecast[0].temp.english);
      $('#condition').text(json.hourly_forecast[0].condition);
    }
  });
}
</script>
```



hourly_forecast is an array

```
- hourly_forecast: [
   - {
      + FCTTIME: { … },
      - temp: {
            english: "69",
            metric: "21"
        },
      - dewpoint: {
            english: "41",
            metric: "5"
        },
        condition: "Clear",
```

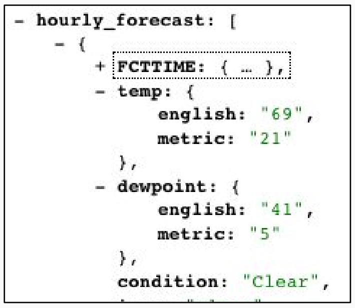
AJAX Test Case

```
// in myajaxtest.js
describe (“AJAX Test Sample”, () => {
  beforeEach(() => {
    loadFixture(“myajaxpage.html”);   // must loadFixture prior to
                                      // installation of AJAX mocks
    jasmine.Ajax.install();
  });

  afterEach( () => {
    jasmine.Ajax.uninstall();
  });

  it(“spies AJAX calls”, () => {
    $(‘button’).click();   // emulate user clicks the button

    request = jasmine.Ajax.requests.mostRecent();
    expect(request.url).toContain(“MI/Allendale”);
  });
});
```

Jasmine AJAX: Stubbing HTTP Responses

```
const req = jasmine.Ajax.request.mostRecent();

req.respondWith(
  {
    status: _____, /* Numeric HTTP status code*/
    responseText: “HTTP response text goes here“
  }
);
```

AJAX Test Case



```
- hourly_forecast: [
    - {
        + FCTTIME: { … },
        - temp: {
              english: "69",
              metric: "21"
          },
        - dewpoint: {
              english: "41",
              metric: "5"
          },
          condition: "Clear",
```

```
// in myajaxtest.js
describe (“AJAX Test Sample”, () => {
  beforeEach(() => { /* same as before */ });
  afterEach( () => { /* same as before */ });

  it(“stubs AJAX reponses”, () => {
    $(‘button’).click();
    request = jasmine.Ajax.request.mostRecent();
    const wunderResponse = {
      hourly_forecast:
        [{ temp: {english: “78”} }, condition: “Sunny” }]
    };
    request.respondWith({
      status: 200, responseText: JSON.stringify(wunderResponse)
    });

    expect($(‘#temperature’)).toContainText(“78”);
    expect($(‘#condition’)).toContainText(“Sunny”);
  });
});
```

Demo: Firebase Auth Spies/Stubs

Git sha1: 27fc98

Mocha

http://mochajs.org

Mocha (+Chai, +Sinon)

- Latest version: 3.0.0
- Test runner
- Supports both BDD (default) and TDD
- Missing components
  - Assertion library, but Mocha can be used with any assertion library of your preference
  - Test doubles library
- Additional Libraries
  - Assertion library: **Chai**
  - Test doubles: **Sinon**

Jasmine vs. Mocha

```
// Jasmine
describe(“Sample Test Suite”, () => {
  beforeAll(() => { /* one time init */ });

  afterAll( () => { /* one time cleanup */ });

  beforeEach(() => { /* per test case init */ });

  afterEach(() => {
    /* per test case cleanup */
  });

  it(“verifies the first feature”, () => {
     // test code here
  }

  it(“verifies the second feature”, () => {
     // test code here
  }
});
```

```
// Mocha
describe(“Sample Test Suite”, () => {
  before(() => { /* one time init */ });

  after( () => { /* one time cleanup */ });

  beforeEach(() => { /* per test case init */ });

  afterEach(() => {
    /* per test case cleanup */
  });

  it(“verifies the first feature”, () => {
     // test code here
  }

  it(“verifies the second feature”, () => {
     // test code here
  }
});
```

Chai Assertion Library

- Supports three different interfaces
  - Should
  - **Expect (will be used in code examples)**
  - Assert
- Online documentation at http://chaijs.com

Jasmine Expect vs. Chai Expect

Jasmine Expect	Chai Expect
expect(*var*).toBe (___)	expect(*var*).to.equal(___)
__.toEqual(*{object}*)	__.to.deep.equal(*{object}*)
__.toBeNull()	__.to.be.a(‘null’)
__.toBeDefined()	__.to.not.be.undefined __.not.to.be.undefined
__.toBeUndefined()	__.to.be.undefined
__.toMatch(*/regex/*)	__.to.match(*/regex/*)
__.toBeLessThan(*val*)	__.to.be.below(*val*)
__.toBeGreaterThan(*val*)	__.to.be.above(*val*)

Sinon.JS

- Latest Version 4.1.2
- Supported Test Doubles
  - Spies
  - Stubs
  - Mocks
  - Fake AJAX
  - Fake Server
  - Fake Timers

Jasmine Spies vs. Sinon Spies/Stubs

Jasmine Spies	Sinon Spies/Stubs
`spyOn(objName, ‘funcName’)`	`const aspy = sinon.spy(objName, ‘funcName’);`
`spyOn(obj,’func’).and.returnValue(___)`	`sinon.stub(obj, ‘func’).returns(____)`
`spyOn(obj,’func’).and.callFake(___)`	`sinon.stub(obj,’func’).callFake(___)`
`obj.fSpy = jasmine.createSpy();`	`obj.fSpy = sinon.spy();`

Spy Matchers: Jasmine vs. Sinon-Chai

```
// Jasmine Spies
spyOn(objName, ‘funcName’);

expect(objName.funcName).toHaveBeenCalled();
expect(objName.funcName).toHaveBeenCalledWith(“OXY”, 182);
```

expect() argument is the spied function!

```
// Sinon Spies
const aspy = sinon.spy(objName, ‘funcName’);

expect(aspy).to.have.been.called;
expect(aspy).to.have.been.calledWith(“OXY”, 182));
```

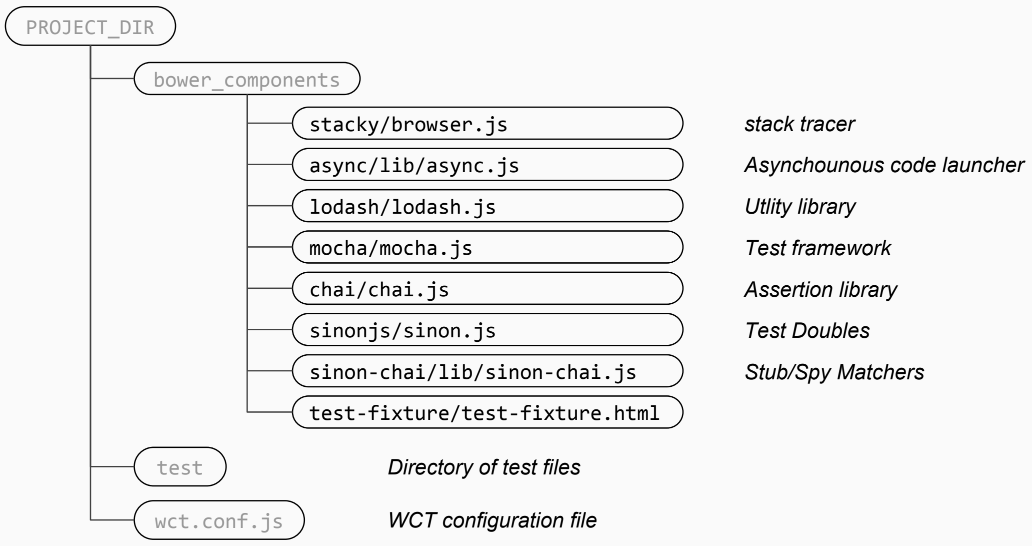
expect() argument is the spy itself

Web Component Tester (WCT)

Testing Polymer Custom Elements

- Web Component Tester (wct 6.4.1), automatically installed by polymer-cli
- Included libraries
  - Mocha
  - Chai
  - Sinon
  - Test-Fixture
  - Async
  - Lodash
- All test-related files under stored under the `test` subdirectory

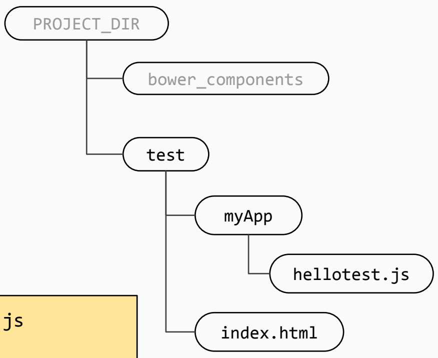
Using Web Component Tester



Web Component Tester: Hello World

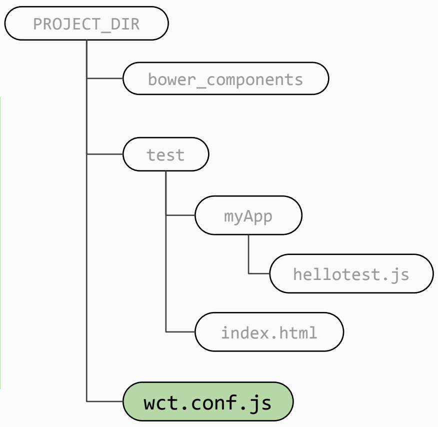
```
<!-- PROJDIR/test/index.html-->
<!doctype html>
<html lang="en">
 <head>
   <script
     src="../bower_components/web-component-tester/browser.js">
   </script>
 </head>
 <body>
   <script>
     WCT.loadSuites([
          'myApp/hellotest.js',
     //   ‘more/testfile.html’
     ]);
   </script>
 </body>
</html>
```

```
// PROJDIR/test/myApp/hellotest.js
describe(“WCT Sample”, () => {

  it(“says hello”, () => {
    expect(“Hello World”).to.match(/^He/);
  });

});
```



Web Component Tester Configuration

```
/* PROJDIR/wct.conf.js */
module.exports = {
  "expanded": true,     /* show details of test description */
  "plugins": {
    "local": {
      "browsers": ["chrome", “firefox”]
    }
  }
}
```



https://github.com/Polymer/web-component-tester/blob/master/runner/config.ts

Running The Tests

1. From terminal

```
> cd /path/to/your/project/folder
> polymer test                     # run the test on all browsers in your computer
> polymer test --local chrome      # test only on selected browsers
```

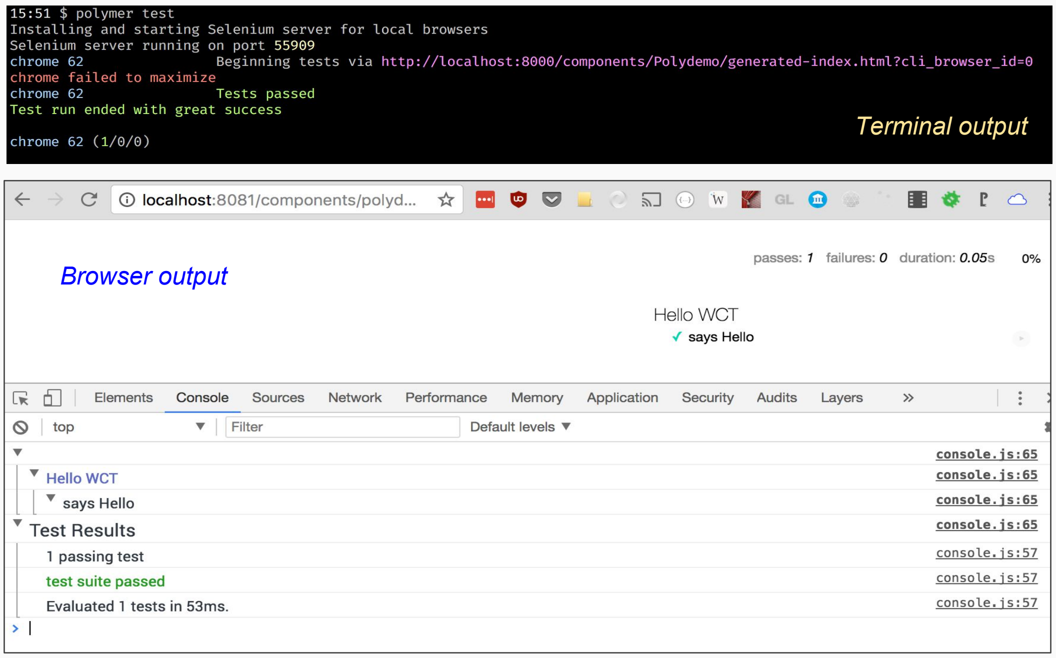
2. From Browser
   a. Run the local server

```
> cd /path/to/your/project/folder
> polymer serve       # output will show port number and path (below)
```

   b. Open `localhost:NNNN/components/`*`projectname`*`/test/index.html`

Demo: WCT Hello World

Git sha1: 024eb



```
15:51 $ polymer test
Installing and starting Selenium server for local browsers
Selenium server running on port 55909
chrome 62                Beginning tests via http://localhost:8000/components/Polydemo/generated-index.html?cli_browser_id=0
chrome failed to maximize
chrome 62                Tests passed
Test run ended with great success

chrome 62 (1/0/0)
```

Terminal output

localhost:8081/components/polyd...

Browser output

passes: *1* failures: *0* duration: *0.05*s 0%

Hello WCT
✓ says Hello

```
▼                                                console.js:65
  ▼ Hello WCT                                    console.js:65
    ▼ says Hello                                 console.js:65
▼ Test Results                                   console.js:65
    1 passing test                               console.js:57
    test suite passed                            console.js:57
    Evaluated 1 tests in 53ms.                   console.js:57
```

Test Fixtures

- Counterpart of Jasmine `loadFixture()` function
- DOM state of the element under fixture is reset between test runs
- Elements under test are inserted under <test-fixture> and <template>

```
<test-fixture>
  <template>
    <custom-elem-here></custom-elem-here>
  </template>
</test-fixture>
```

Using <test-fixture>

```
// in mytest.html
<head>
  <link rel="import" href="src/path/to/custom_element.html">
</head>
<body>
  <test-fixture id="mypage">
    <template>
      <your-custom-el></your-custom-el>
    </template>
  </test-fixture>
  <script src="mytest.js"></script>
</body>
```

```
// in mytest.js
describe ("Test Sample", () => {
  var elem;
  beforeEach(() => {
    elem = fixture("mypage");
  });

  it("validates your-custom-el", () => {
    const z1 = elem.shadowRoot.getElementById('obj');
    expect(z).to._____;
    const z2 = elem.$.obj;
    expect(z2).to._____;
  });
});
```

Accessing Shadow DOM Elements

```
var top = fixture(“id-of-your-test-fixture”);

// (1) Using DOM APIs, must call getElement____() from shadowRoot

var el1    = top.shadowRoot.getElementById(‘goBtn’);
var elems1 = top.shadowRoot.getElementsByTagName(‘span’);
var elems2 = top.shadowRoot.getElementsByClassName(‘warning’);
var el2    = top.shadowRoot.querySelector(‘input[type=submit]’);
Var elems3 = top.shadowRoot.querySelectorAll(‘input[type=text]’);

// (2) Using Polymer $
var el3 = top.$.goBtn;
```

Chai-DOM

- DOM Matcher that works with Chai
- Counterpart of Jasmine jQuery plugin
- Installation: `bower install --save-dev chai-dom`
- Chai-DOM is **not** automatically installed by Polymer-CLI
  - Must be manually loaded into WCT workspace by setting the array `WCT.environmentScripts`

Loading chai-dom.js into WCT workspace

```
<head>
<script src=”../../../webcomponentjs/webcomponents-lite.js”></script>
<script>
WCT.environmentScripts: [ // Must be set PRIOR TO loading wct/browser.js
  ‘stacky/browser.js’,
  ‘async/lib/async.js’,
  ‘lodash/lodash.js’,
  ‘mocha/mocha.js’,
  ‘chai/chai.js’,
  ‘sinonjs/sinon.js’,
  ‘sinon-chai/lib/sinon-chai.js’,
  ‘chai-dom/chai-dom.js’
];
</script>
<script src=”../../../web-component-tester/browser.js”></script>
<link rel=”import” href=”../../src/path/to/custom-element.html”>
</head>
<body>
  <test-fixture id=”testEl”>
    <template><custom-element></custom-element></template>
  </test-fixture>
</body>
```

Jasmine jQuery vs. Chai-DOM

Jasmine jQuery	Chai DOM
expect(*var*).toBeChecked ()	expect(*var*).to.have.attr(“checked”, true)
__.toBeDisabled()	__.to.have.attr(“disabled”)
__.toBeHidden()	__.to.have.attr(“hidden”)
__.toBeVisible()	__.to.be.displayed
__.toExist()	__.to.exist
__.toBeSelected()	__.to.have.attr(“selected”, true)
__.toContain(*string*)	__.to.contain.text(*string*)
__.toContainElement(*css-sel*)	__.to.contain(*css-sel*)

Jasmine jQuery vs. Chai-DOM

Jasmine jQuery	Chai DOM
`expect(`*var*`).toHaveClass(`*clsname*`)`	`expect(`*var*`).to.have.class(`*clsname*`)`
`__.toHaveCss(`*cssObject*`)`	*No equivalent matcher*
`__.toHaveId(`*reqId*`)`	`__.to.have.id(`*reqId*`)`

Sinon Fake Timers

```
describe("____", () => {
  var mockedClock;

  beforeEach( () => {
    mockedClock = sinon.useFakeTimers();
  });

  afterEach( () => {
    mockedClock.restore();
  });

  it("_______", () => {
    expect(____).to_______;  // current clock is    0ms
    mockedClock.tick(3000);  // current clock is 3000ms
    expect(____).to_______;
    mockedClock.tick(1200);  // current clock is 4200ms
    expect(____).to_______;
  });
});
```

Demo: Mocked Interaction & Fake Timers

Git sha1: cb44b