JASMINE

What it is?

- Behavior driven development framework
- For testing JavaScript code
- No 3rd party library dependency
- Does not require a DOM
- Offers clean syntax for writing tests

Getting Started

- Download the standalone release for running under the browser from http://jasmine.github.io/
  - Can use Karma for running outside of the browser

```
<!DOCTYPE html>
<html>
<head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>Jasmine Spec Runner v2.0.1</title>
     <link rel="shortcut icon" type="image/png" href="~/Content/jasmine_favicon.png">
     <link rel="stylesheet" type="text/css" href="~/Content/jasmine.css">
     <script src="~/Scripts/Lib/jasmine.js"></script>
     <script src="~/Scripts/Lib/jasmine-html.js"></script>
     <script src="~/Scripts/Lib/jasmine-boot.js"></script>
</head>
<body>
</body>
</html>
```

- Add reference to your spec scripts

Ingredients

- Suite
- Spec
- Expectation
- Matcher
- Setup and Teardown
- Spy

Spec Suite

- Begins with a describe function call
- Has a name
- Contains multiple specs (tests)

```
describe("CounterNewCtrl", function () {
     it("does not allow adding a counter without a name", function () {
          ...
     });
});
```

Spec

- Is defined using the it function
- Takes a title and a function
- Contains one or more expectation
- A spec with all true expectations is considered a passing spec

```
describe("CounterNewCtrl", function () {
    it("does not allow adding a counter without a name", function () {
        var ctrl = new CounterNewCtrl();

        ctrl.name = "";
        ctrl.add();

        expect(ctrl.errors.length).toBeGreaterThan(0);
    });
});
```

Expectations

- Are defined using the expect function
- Takes an actual value and a matcher
- Reports to Jasmine whether to pass or fail the spec
- Negative matcher is achieved using the not function
- No description ☹

```
describe("CounterNewCtrl", function () {
     it("does not allow adding a counter without a name", function () {
         var ctrl = new CounterNewCtrl();

         ctrl.name = "";
         ctrl.add();

         expect(ctrl.errors.length).not.toBe(0);
     });
});
```

Matchers

- Jasmine offers the following matchers

toBe	toBeGreaterThan	toBeNull
toBeDefined	toBeLessThan	
toBeFalsy	toThrow	toBeUndefined
toBeTruthy	toEqual	toMatch(pattern)
toContain(member)	toContain(substring)	

- Consider use **Jasmine-Matchers** library for more matchers

Custom Matcher

- Use addMatchers inside it function

```
it("supports custom matcher", function () {
     jasmine.addMatchers({
         toBeEmptyString: function () {
             return {
                 compare: function (actual, name) {
                     var pass = actual === "";
                     var message = (pass ? name + " is empty" : name + " is not empty");
                     return {
                         pass: pass,
                         message: message,
                     };
                 }
             };
         }
     });

     var name = "Ori";
     expect(name).toBeEmptyString("name");
});
```

Setup and Teardown

- A describe block may contain beforeEach and afterEach functions
- Both are invoked before and after each spec (it)
- The this keyword is the same for all three functions

```
describe("CounterNewCtrl", function () {
     beforeEach(function () {
         this.ctrl = {};
     });

     afterEach(function () {
         this.ctrl = null;
     });

     it("this is the same", function () {
         expect(this.ctrl).toBeDefined();
     });
});
```

More

- describe blocks can be nested
  - beforeEach and afterEach are called according to nesting tree structure
- Appending “x” to a suite or spec disables it
  - xdescribe
  - xit

```
describe("CounterNewCtrl", function () {
     beforeEach(function () {
     });

     describe("validation", function () {
         beforeEach(function () {
         });

         it("nested spec", function () {
             expect(this.ctrl).toBeDefined();
         });
     });

     xit("disabled spec", function () {
         expect(this.ctrl).toBeDefined();
     });
});
```

Spy

- Tracks calls to an object method
- By default does not delegate the call
  - Use spyon.and.callThrough() to force delegation

```
it("calls CounterStore.add when validation pass", function () {
     var store = new CounterStore();
     var ctrl = new CounterNewCtrl(store);

     ctrl.name = "New Counter";

     spyOn(CounterStore, "add").and.callThrough();
     ctrl.add();

     expect(CounterStore.add).toHaveBeenCalledWith("New Counter");
});
```

Spy API

- and.returnValue
- and.callFake
- and.throwError
- and.stub – Disables callThrough behavior
- calls.any
- calls.count
- calls.all

createSpy & createSpyObj

- In some cases there is no a function/object to spy on
- Can create a bare spy

```
it("test", function () {
     var spy = jasmine.createSpy("spy");

     spy();
     spy();

     expect(spy.calls.count()).toEqual(2);
});
```

- Or create a complete mock object

```
it("test", function () {
     var obj = jasmine.createSpyObj("spy", ["func1", "func2"]);

     obj.func1();
     obj.func2();

     expect(obj.func1).toHaveBeenCalled();
     expect(obj.func2).toHaveBeenCalled();
});
```

jasmine.any

- In some cases we don't care about the parameter value being sent to a function but rather its type

```
it("calls obj.func with a string", function () {
     var obj = {
         func: function (str) {
         }
     };

     spyOn(obj, "func");

     obj.func("abc");

     expect(obj.func).toHaveBeenCalledWith(jasmine.any(String));
});
```

- Note: Jasmine compares constructors (not instanceof)

jasmine.objectContaining

- By default toHaveBeenCalledWith verifies all object's fields
- Use jasmine.objectContaining to verify only part of the object

```
it("calls obj.func with an object that contains some fields", function () {
     var obj = {
         func: function (str) {
         }
     };

     spyOn(obj, "func");

     obj.func({
         id: 1,
         name: "Ori",
     });

     expect(obj.func).toHaveBeenCalledWith(jasmine.objectContaining({
         id: 1
     }));
});
```

How would you test the following ?


```
function InactivityMonitor() {
     this.counter = 0;
     this.handle = null;
     this.events = [];
}

InactivityMonitor.prototype.start = function () {
     var me = this;

     var counter = me.counter;

     this.handle = setInterval(function () {
         if (me.counter == counter) {
             me.events.push(new Date());
         }
     }, 60000);
}

InactivityMonitor.prototype.activity = function () {
     ++this.counter;

     clearInterval(this.handle);
     this.handle = null;

     this.start();
}
```

jasmine.clock

- Replaces the native setTimeout/setInterval functions with synchronous implementation
- The registered callbacks are executed only if the clock is ticked forward in time

```
it("queues an event after inactivity of more than 1 minute", function () {
     jasmine.clock().install();

     var monitor = new InactivityMonitor();
     monitor.start();

     jasmine.clock().tick(60000);

     expect(monitor.events.length).toBeGreaterThan(0);
});
```

jasmine-ajax

- A library for faking AJAX response
- Same pattern as the jasmine.clock
- It replaces native XMLHttpRequest with synchronous implementation
- Then, allows you to specify the response manually
- See next slide for
  - jasmine.Ajax.install
  - jasmine.Ajax.requests

jasmine-ajax


```
it("fakes AJAX request", function () {
     jasmine.Ajax.install();

     $.ajax({
         type: "GET",
         url: "/api/counter",
         success: function (counters) {
             expect(counters.length).toBe(2);
         },
         error: function () {
             expect(false).toBeTruthy();
         }
     });

     request = jasmine.Ajax.requests.mostRecent();

     request.response({
         status: 200,
         responseText: '[{"name": "Coffee", "value" :1}, {"name": "Sport", "value": 2}]',
     });

     //  We get here only after success/error callbacks are executed
     jasmine.Ajax.uninstall();
});
```

Promises

- Even when using jasmine-ajax promise behaves in an asynchronous way
- The spec might complete before the promise
- Jasmine offers a done parameter which implies an asynchronous spec
- You need to invoke done() when promise completes and all expectations where set

Promises


```
it("reports spec result only after promise completes", function (done) {
     jasmine.Ajax.install();

     var httpService = new MyApp.HttpService();
     var counterStore = new MyApp.CounterStore(httpService);

     counterStore.getAll()
         .then(function (counters) {
             expect(counters.length).toBe(0);
         })
         .fail(function (err) {
             expect(false).toBeTruthy();
         })
         .fin(function () {
             done();
         });

     request = jasmine.Ajax.requests.mostRecent();
     request.response({
         status: 200,
         responseText: "[]",
     });

     //  We get here before then/fail/fin complete
     jasmine.Ajax.uninstall();
});
```

Summary

- Jasmine is simple and intuitive
- Has suites and specs
- Has a nice expectation vocabulary
- Can be executed inside the browser
- Unit test should follow the F.I.R.S.T principles