# Testing AngularJS

# Objectives

- Using Jasmine with AngularJS
  - Injecting dependencies
  - Mocking AngularJS services
- Integration testing
  - End 2 End testing with Protractor
  - The page object model

# Using Jasmine with AngularJS

- AngularJS extends the Jasmine test functions
  - ddescribe() – Only run this suite of specifications
  - iit() - Only run this specification
- Part of angular-scenario.js
  - Automatically loaded by Karma



# Testing a controller function

```
describe("Adding numbers in the MathCtrl", function () {
    var ctrl, scope;

    beforeEach(module('myApp'));

    beforeEach(inject(function ($controller, $rootScope) {
        scope = $rootScope.$new();
        ctrl = $controller('MathCtrl', {
            $scope: scope
        });
    }));

    it('should add 1 + 2', inject(function () {
        var sum = scope.add(1, 2);
        expect(sum).toBe(3);
    }));
});
```

# Testing code without its dependencies

- Always test one part of the code at the time
  - Integration testing is used to test the combination
- Create “fake” objects/functions as dependencies
  - Possibly using Jasmine spies
- The module() loads a module and registers dependencies
- The inject() function inserts registered dependencies
- The $controller() function can be used to create a controller and inject its dependencies

# A controller with a service to be tested

```
angular.module('myServices', [])
    .service("theSvc", function () {
        return {
            getData: function () { return 999; }
        };
    });

angular.module('myApp', ['myServices'])
    .controller("TheCtrl", function ($scope, theSvc) {
        $scope.callService = function () {
            return theSvc.getData();
        };
    });
```



# Testing the controller

```
describe("Calling a service in the MathCtrl", function () {
    var ctrl, scope;

    beforeEach(module('myApp'));

    beforeEach(inject(function ($controller, $rootScope) {
        scope = $rootScope.$new();
        var theSvc = jasmine.createSpyObj('svc', ['getData']);
        theSvc.getData.andReturn(1);

        ctrl = $controller('TheCtrl', {
            $scope: scope,
            theSvc: theSvc
        });
    }));

    it('should return 1', inject(function () {
        var sum = scope.callService();
        expect(sum).toBe(1);
    }));
});
```

# Testing the service

```
describe("The theSvc", function () {
    var svc;

    beforeEach(module('myServices'));

    beforeEach(inject(function (theSvc) {
        svc = theSvc;
    }));

    it("should return 999 from getData()", function () {
        var result = svc.getData();

        expect(result).toBe(999);
    });
});
```



# Testing an AJAX service

- The $httpBackend from angular-mocks.js replaces the default
  - Allows you to configure expected requests and their results
- $httpBackend.flush() triggers the result to be pushed back
  - The result is just as asynchronous as the normal call
- $when() sets up a standard response
  - Used for multiple requests
  - Doesn't fail if not actually called
- $expect()
  - Used for a single request
  - Fails the test if the request was not done



# Testing an AJAX service

```
describe("Books service", function () {
    var $httpBackend, service;

    beforeEach(module('booksApp'));

    beforeEach(inject(function (_$httpBackend_, Books) {
        $httpBackend = _$httpBackend_;
        $httpBackend.when("GET", "/api/books").respond([{}]);

        service = Books;
    }));

    it("should return data from query", function () {
        var books = service.query();
        $httpBackend.flush();
        expect(books.length).toBe(1);
    });
});
```

# Testing directives

- Use angular.element() to parse an html string
- Use the $compile() function to turn a static element into a live document
  - Provide a $scope for data-binding
- The $scope.$apply() will trigger the actual binding
  - This will trigger watch functions to fire.



# A simple directive

```
angular.module('myModule', [])
    .directive('innerText', function () {
        return {
            scope: {
                innerText: '@'
            },
            link: function (scope, element) {
                scope.$watch('innerText', function (value) {
                    element.text(value);
                });
            }
        };
    });
```



# Testing the directive

```
describe("The innerText directive", function () {
    var element;

    beforeEach(module('myModule'));

    beforeEach(inject(function ($compile, $rootScope) {
        var $scope = $rootScope.$new();
        element = angular.element(
            '<div inner-text="Some text"/>');
        $compile(element)($scope);
    }));

    it('should display the text', function () {
        element.scope().$apply();
        expect(element.text()).toBe('Some text');
    });
});
```



# Integration testing

- Protractor is an end to end testing framework that knows about AngularJS
  - Build on top of Selenium
- Test your application by driving the browser
- Lets you select elements based on Angular directives
  - More reliable than traditional Selenium style selections
- Understands asynchronous Angular like $http and $resource
  - Will wait for them to complete before continuing
- Use the page object model to make tests more maintainable

# A simple Protractor E2E test

```
describe('homepage', function () {
    beforeEach(function () {
        browser.get('http://localhost:25046/Default.html');
    });

    it('should edit a users first name', function () {
        var row = element(
            by.repeater("p in people")
                .row(5)
                .column("firstName"));
        expect(row.getText()).not.toEqual('Mike');
        row.click();

        var fname = element(
            by.model('currentPerson.firstName'));
        fname.clear();
        fname.sendKeys('Mike');

        expect(row.getText()).toEqual('Mike');
    });
});
```

# An E2E test using the page object model

```
describe('homepage using page object', function () {

    it("Should edit using a page object", function () {
        var page = new DefaultPage();
        page.load();

        var row = page.editRow(5);
        expect(row.getText()).not.toEqual('Mike');

        page.setFirstName("Mike");

        expect(row.getText()).toEqual('Mike');
    });
});
```



# The page object model for the test

```
function DefaultPage() {
    var fname = element(by.model('currentPerson.firstName'));

    this.load = function () {
        browser.get('http://localhost:25046/Default.html');
    };

    this.editRow = function (index) {
        var row = element(
            by.repeater("p in people")
                .row(index).column("firstName"));
        row.click();
        return row;
    };


    this.setFirstName = function (name) {
        fname.clear();
        fname.sendKeys(name);
    };
}
```

# Summary

- Unit test your individual components
  - Angular encourages you to split things up into testable units
  - Jasmine is a powerful unit test framework
  - Use fake objects to manage dependencies
  - Karma makes running unit tests easy
- Create end to end tests to make sure your whole application works
  - Protractor makes running these easy
  - Use the page object model to make them maintainable