AngularJS Testing
Testing

- The most popular tool for running automated tests for AngularJS applications is Karma
  - runs unit tests and “end-to-end” tests in real browsers and PhantomJS
  - can use many testing frameworks, but Jasmine seems to be supported best

- Create test directory with subdirectories `unit` and `e2e`
  - in unit directory, create subdirectories `controllers` and `services`
To install **Karma**

- `npm install -g karma`
- `cd` to top directory of project
- `karma init` (answer questions; creates `karma.config.js`)
  - in Windows, this didn't work from Cygwin but did from a Command Prompt
- edit `karma.config.js`
  - add to `files` array passed to `config.set` the paths to `angular.min.js`, `angular-mocks.js` and the JavaScript files being tested

*maybe because node and npm for Windows don't work in Cygwin*
Browser Support

- To use Google **Chrome**
  - karma-chrome-launcher is installed automatically when karma is installed
  - on Windows, set `CHROME_BIN` environment variable to point to `chrome.exe`

- To use **Firefox**
  - karma-firefox-launcher is installed automatically when karma is installed
  - on Windows, set `FIREFOX_BIN` environment variable to point to `firefox.exe`

- To use **IE** (Windows-only)
  - `npm install -g karma-ie-launcher`

- To use **Safari** (Mac-only)
  - `npm install -g karma-safari-launcher`

- To use **PhantomJS**
  - karma-phantomjs-launcher is installed automatically when karma is installed; includes PhantomJS
  - set `PHANTOMJS_BIN` environment variable to point to `phantomjs` executable (not needed on Windows)
  - executable is in `npm root -g`/karma-phantomjs-launcher/node_modules/phantomjs/bin
We will walk through writing tests for a simple application

The application has
- one service
- three controllers
- one custom filter
- one custom directive
- routing

Pressing the “Increment” button increments the number
- the controller calls a service method to increment it - keeping things simple!

A filter formats the numbers 1, 2 and 3 as words

A directive applies different CSS classes to the number based on whether it is even or odd

Routes cause different footers to be displayed based on whether the number is even or odd
<!DOCTYPE html>
<html ng-app="KarmaDemo">
  <head>
    <link rel="stylesheet" href="karma-demo.css"/>
    <script src="lib/angular.min.js"></script>
    <script src="scripts/karma-demo.js"></script>
  </head>
  <body>
    <h1>Karma Demo</h1>
    <div ng-controller="DemoCtrl">
      <div>
        Number:
        <span even-odd-class="foo,bar">{{number | asWord}}</span>
      </div>
      <button ng-click="increment()">Increment</button>
    </div>
    <div ng-view="footer"></div>
  </body>
</html>
(function () {
    'use strict';

    // Create a module just to use for testing module dependencies.
    angular.module('OtherModule', []);

    var app = angular.module('KarmaDemo', ['OtherModule']);

    app.factory('demoSvc', function () {
        var svc = {};
        svc.increment = function (number) {
            return number + 1;
        };
        return svc;
    });

    app.controller('DemoCtrl', function ($scope, $location, demoSvc) {
        $scope.number = 1;
        $scope.increment = function () {
            $scope.number = demoSvc.increment($scope.number);
            var isEven = $scope.number % 2 === 0;
            $location.path('/' + (isEven ? 'even' : 'odd'));
        };
    });

    // These empty controllers are only used to test that routes
    // associated the correct controller with a view.
    app.controller('EvenFooterCtrl', function () {});
    app.controller('OddFooterCtrl', function () {});
});
app.directive('evenOddClass', function () {
  var evenClass, oddClass;

  function evaluate(number, element) {
    if (number % 2 === 0) {
      element.removeClass(oddClass);
      element.addClass(evenClass);
    } else {
      element.removeClass(evenClass);
      element.addClass(oddClass);
    }
  }

  return {
    restrict: 'A',
    link: function (scope, element, attrs) {
      var classNames = attrs.evenOddClass.split(',');
      evenClass = classNames[0];
      oddClass = classNames[1];
      scope.$watch('number', function (newValue, oldValue) {
        evaluate(newValue, element);
      });
      evaluate(scope.number, element);
    }
  }
});

app.filter('asWord', function () {
  return function (number) {
    return number === 1 ? 'one' : number === 2 ? 'two' : number === 3 ? 'three' : number;
  }
});
app.config(function ($routeProvider) {
$routeProvider
  .when('/even', {
    'controller': 'EvenFooterCtrl',
    'templateUrl': 'views/evenFooter.html',
    'view': 'footer'
  })
  .when('/odd', {
    'controller': 'OddFooterCtrl',
    'templateUrl': 'views/oddFooter.html',
    'view': 'footer'
  })
  .otherwise({
    redirectTo: '/odd'
  });
});
describe('KarmaDemo module', function () {
  var module = angular.module('KarmaDemo');

  it('should exist', function () {
    expect(module).not.toBeNull();
  });

  it('should have one dependency', function () {
    expect(module.requires.length).toBe(1);
    expect(module.requires).toContain('OtherModule');
  });
});
describe('demoSvc service', function () {
  var svc;

  beforeEach(module('KarmaDemo'));

  it('should have demoSvc service', inject(function ($injector) {
    svc = $injector.get('demoSvc');
    expect(svc).not.toBeNull();
  }));

  it('should increment properly', function () {
    expect(svc.increment(2)).toBe(3);
  });
});

The method angular.mock.module is added to window as module.
describe('DemoCtrl controller', function () {
  var scope;

  beforeEach(function () {
    module('KarmaDemo');
    inject(function ($rootScope, $controller) {
      scope = $rootScope.$new(); // create a scope for DemoCtrl to use
      $controller('DemoCtrl', {$scope: scope}); // give scope to DemoCtrl
    });
  });

  it('should have number in scope', function () {
    expect(scope.number).not.toBeNull();
    expect(scope.number).toBe(1);
  });

  it('should increment properly', function () {
    scope.increment();
    expect(scope.number).toBe(2);
    scope.increment();
    expect(scope.number).toBe(3);
  });
});
Filter Tests

describe('asWord filter', function () {
  var filter;

  beforeEach(module('KarmaDemo'));

  it('should have asWord filter', inject(function ($filter) {
    filter = $filter('asWord');
    expect(filter).not.toBeNull();
  }));

  it('should translate 2', function () {
    expect(filter(2)).toBe('two');
  });
});
describe('evenOddClass directive', function () {
  var directive;

  beforeEach(module('KarmaDemo'));

  // TODO: Is there a way to test that a directive with a given name
  // TODO: exists in a module?

  it('should add the correct CSS class to an element', inject(function ($compile, $rootScope) {

    $rootScope.number = 1;
    // The directive should add the CSS class "bar"
    // to the element because number is odd.
    var element = $compile('<span even-odd-class="foo,bar"></span>')($rootScope);
    expect(element.hasClass('bar')).toBe(true);

    $rootScope.number = 2;
    // The directive should add the CSS class "foo"
    // to the element because number is even.
    element = $compile('<span even-odd-class="foo,bar"></span>')($rootScope);
    expect(element.hasClass('foo')).toBe(true);
  }));
});
describe('footer routes', function () {
    it('should change path', function () {
        module('KarmaDemo'); // must do before inject

        inject(function ($route) {
            var route = $route.routes['/even'];
            expect(route.controller).toBe('EvenFooterCtrl');
            expect(route.templateUrl).toBe('views/evenFooter.html');
            expect(route.view).toBe('footer');

            route = $route.routes['/odd'];
            expect(route.controller).toBe('OddFooterCtrl');
            expect(route.templateUrl).toBe('views/oddFooter.html');
            expect(route.view).toBe('footer');
        });
    });
});
module.exports = function (config) {
    config.set({
        basePath: '', // used to resolve relative file paths
        frameworks: ['jasmine'],
        files: [
            'lib/angular.min.js',
            'lib/angular-mocks.js',
            'scripts/*.js',
            'test/unit/*Spec.js'
        ],
        exclude: [],

        // options: 'dots', 'progress', 'junit', 'growl', 'coverage'
        reporters: ['progress'],

        port: 9876, // web server port
        colors: true, // in output of reporters and logs
        logLevel: config.LOG_INFO,
        autoWatch: true, // watch files and execute tests when any change
        // options: Chrome, ChromeCanary, Firefox, Opera,
        // Safari (Mac-only), IE (Windows-only), PhantomJS
        browsers: ['Chrome'],

        // if browser doesn’t capture output in given timeout(ms), kill it
        captureTimeout: 60000,

        // if true, it capture browsers, runs tests and exits
        singleRun: false
    });
};
Running Unit Tests

- For a single run
  - edit `karma.conf.js` and verify that in the object passed to `config.set`
    auto-watch is false and singleRun is true

- For repeated runs every time a watched file is modified
  - edit `karma.conf.js` and verify that in the object passed to `config.set`
    auto-watch is true and singleRun is false

- To run unit tests
  - `karma start`
  - starts its own server and configured browsers
  - runs tests
  - exits if configured for a single run
Mock Dependency Injection

- In unit test code, mock implementations of services can be used in place of real implementations.
- One way to do this:
  - create an object that has all the methods of the service.
  - register that object under the name of the service to be replaced.
    - `app.value(svcName, mockSvcObject);`
- Other ways?
End-to-End Tests

- **Old approach - Angular Scenario Runner**
  - used with Karma
  - `npm install -g karma-ng-scenario`
  - must start a server that serves all static content
    - can use Grunt with the connect plugin
    - must do a local `npm install` of grunt, grunt-contrib-connect and grunt-karma (takes a LONG time!)
  - being phased out in favor of Protractor

- **New approach - Protractor**
  - [https://github.com/angular/protractor](https://github.com/angular/protractor)
  - based on Selenium WebDriverJS
  - designed for AngularJS, but not restricted to that
  - can use Jasmine or Mocha test frameworks
  - a test runner; doesn’t require using Karma
  - can use Selenium Grid to test in multiple browsers simultaneously and run multiple instances of the same browser for load testing
Protractor ...

To install

- `npm install -g protractor`
- `cd` to top project directory
- Determine directory where global node modules are installed by running `npm root -g`
- Install standalone Selenium server
  - `./that-dir/protractor/bin/install_selenium_standalone`
  - On Windows, `node /that-dir/protractor/bin/install_selenium_standalone`
- Copy `/that-dir/protractor/referenceConf.js` to `protractor.conf.js`

Edit `protractor.conf.js`

- Modify `specs` property to include directory where tests will be stored (ex. `test/e2e`)
- Modify `capabilities.browserName` property to the name of the browser to be used
  - Common choices are `android`, `chrome`, `firefox`, `internet explorer`, `iPhone`, `iPad` and `safari`
- Modify `baseUrl` property to be URL where web app is served

For a detailed description of options, see https://code.google.com/p.selenium/wiki/DesiredCapabilities

not needed if using an existing Selenium server
To run tests

- cd to top project directory
- start Web server that servers static files and processes Ajax calls
- `protractor protractor.conf.js`
  - starts Selenium server, runs tests, reports results, shuts down Selenium server, and exits
  - `console.log` output in tests will go to stdout
Working With Page Elements

- Get Protractor instance and load a page

```javascript
var ptor;

beforeEach(function () {
  ptor = protractor.getInstance();
  ptor.get('some-url');
});
```

- Get element objects by calling `ptor.findElement` or `ptor.findElements` with a “locator”
  - `.findElement` returns the first matching element
  - `.findElements` returns an array of all matching elements.
  - locators are `protractor.By.x('some-string')` where `x` is one of `className, css, id, linkText, partialLinkText, name, tagName` or `xpath`
  - AngularJS-specific values for `x` include `binding, select, selectedOption, input` and `repeater`
  - throws if no matching elements are found, causing test to fail
Element Methods

- Can call many methods on these “element” objects
  - `clear()`, `click()`, `getAttribute(name)`, `getCssValue(propertyName)`, `getLocation()`, `getSize()`, `getTagName()`, `getText()`, `isDisplayed()`, `isEnabled()`, `isSelected()`, `sendKeys(text)`

- Typically want to call `clear()` on input elements
  before calling `sendKeys(text)`

- Methods that return data such as `getText` actually return a promise

- When a promise is passed to `expect`, it waits until the promise is resolved or rejected to evaluate it
Protractor Sugar

- Directly using Protractor methods can be verbose
- Here are some utility methods to make Protractor tests less verbose
- This is used in the example test on the next slide

```javascript
module.exports = function (ptor) {
  var obj = {};

  obj.click = function (id) {
    this.getElementById(id).click();
  };

  obj.getElementById = function (id) {
    return ptor.findElement(protractor.By.id(id));
  };

  obj.getText = function (id) {
    return this.getElementById(id).getText();
  };

  obj.setInputValue = function (id, value) {
    var input = this.getElementById(id);
    input.clear();
    input.sendKeys(value);
  };

  return obj;
};
```
var protractorSugar = require('./protractorSugar');

describe('login', function () {
  var ps, ptor;

  beforeEach(function () {
    ptor = protractor.getInstance();
    ps = protractorSugar(ptor);
    ptor.get('/');
  });

  it('should fail on bad username/password', function () {
    ps.setInputValue('userId', 'foo');
    ps.setInputValue('password', 'bar');
    ps.click('login');
    expect(ps.getText('loginError')).toBe('The user id or password is incorrect.');
  }, 5000); // five second timeout

  it('should get password hint', function () {
    ps.setInputValue('userId', 'gumby');
    ps.click('passwordHintLink');
    expect(ps.getText('passwordHint')).toBe('my horse');
  }, 5000); // five second timeout
});