TESTING ANGULAR

Ori Calvo, 2017

oric@trainologic.com

https://trainologic.com

Types of Testing

- Unit
- Integration
- System
- Stress
- Performance
- Load
- More …

Unit Testing

- Unit as the smallest testable part of the application
- Created by programmers
- Run by programmers
- Usually inside a class boundary
- Does not cross process/network boundaries

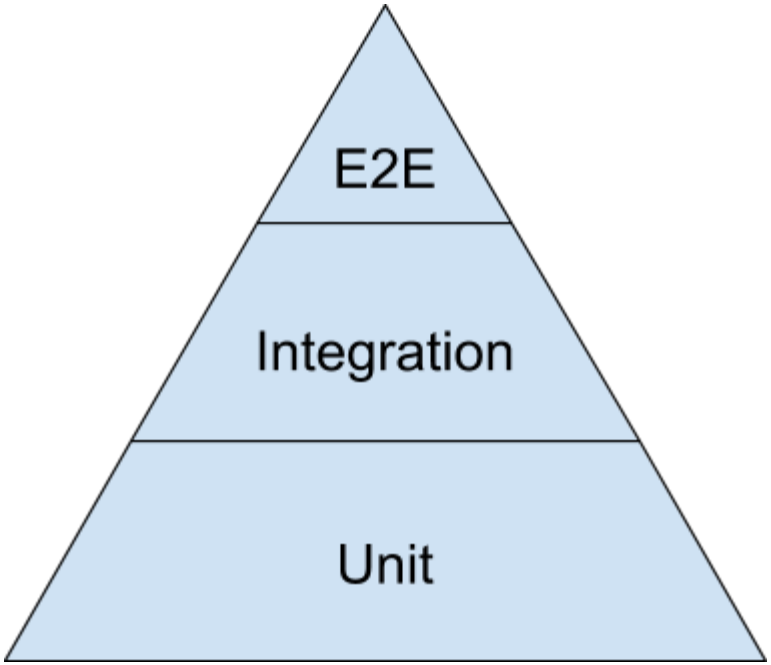
F.I.R.S.T Principles

- Fast
- Isolated/Independent
- Repeatable
- Self Validating
- Thorough and Timely

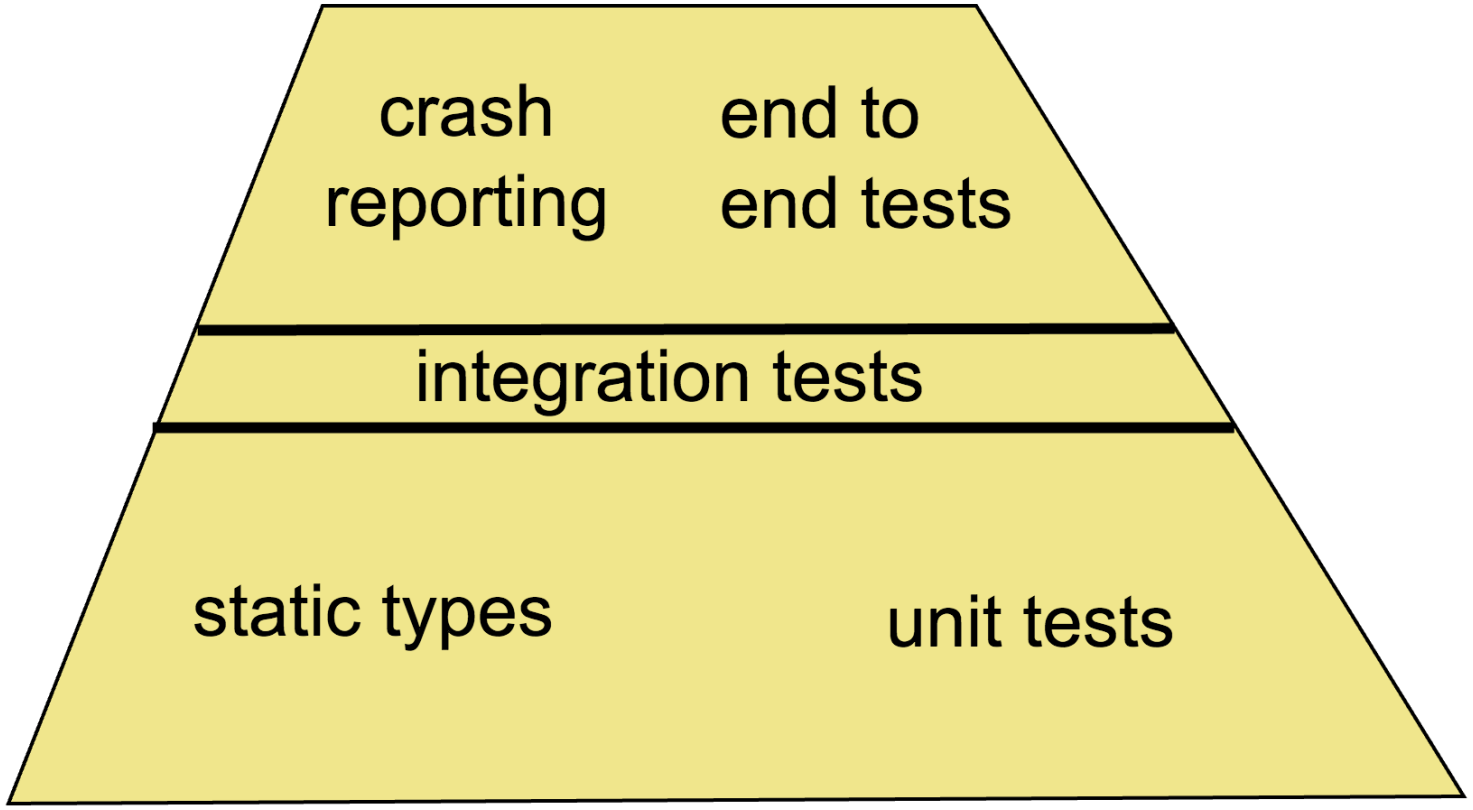
End to End Testing

- The entire application is tested in a real world scenario
- Testing whether a flow of the application is performing as designed from start to finish
- Ensure the right information is passed between various system components and systems

Testing Pyramid



Testing Trapezoid



Technology Stack

- Jasmine
- Angular testing utilities
- Karma
- Protractor
- WebDriver

Jasmine

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

Running under browser

- npm install jasmine
- Add reference to the following scripts/css

```
<script src="node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
<script src="node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
<script src="node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>

<link rel="stylesheet" href="node_modules/jasmine-core/lib/jasmine-
core/jasmine.css">
```

- Add reference to your spec scripts

Running under NodeJS

- node_modules/.bin/jasmine init
  - Creates spec/support/jasmine.json
  - Ensure spec_dir is correct
- node_modules/.bin/jasmine
  - Executes all tests according to spec_files

My First Spec


```
class Contact {
    constructor(name) {
        if(!name) {
            throw new Error("Ooops");
        }
    }
}
```

```
describe("contact", function() {
    it("does not allow instantiation without name", function () {
        expect(() => {
            new Contact()
        }).toThrow();
    });
});
```

Ingredients

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

Suite

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

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

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

         expect(comp.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("CounterNewComponent", function () {
     it("does not allow adding a counter without a name", function () {
         var comp = new CounterNewComponent();

         comp.name = "";
         comp.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();
     });
});
```

afterAll/beforeAll

- Same as prveious slide
- But this time the setup is executed before/after all specs

```
describe("contact", function() {
    beforeAll(function() {
        console.log("before");
    });

    afterAll(function() {
        console.log("after");
    });
});
```

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
- Appending “f” disable others

```
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 comp = new CounterNewComponent(store);

     comp.name = "New Counter";

     spyOn(store, "add").and.callThrough();
     comp.add();

     expect(store.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 () {
     const me = this;

     const 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);
         })
         .catch(function (err) {
             expect(false).toBeTruthy();
         })
         .finally(function () {
             done();
         });

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

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

Testing Angular Entities

- For each entity you may use different approach for testing
- Services are usually easier to test
- Components require HTML introspection
- Pipes are stateless and therefore much easier to test

Isolated Testing

- Examine an instance of a class all by itself
- No Angular
- No dependency injection
- The tester use the new keyword
- Supplying some test doubles for the ctor's parameters
- Probes the test instance API
- Most suited for services & pipes

Testing a Service


```
export class ContactService {
 constructor(private httpClient: HttpClient) { }

 getAll(): Promise<Contact[]> {
  return this.httpClient.get<Contact[]>("assets/contacts.json").toPromise();
 }
}
```

```
it("should return all contacts when executing getAll", async done => {
 httpClientMock = {
  get: function () {
   return Observable.of([1, 2, 3]);
  }
 };

 const service = new ContactService(httpClientMock);

 const contacts = await service.getAll();
 expect(contacts.length).toBe(3);

 done();
});
```

Testing a Component

- Isolated testing
  - Test component without rendering
- Shallow testing
  - Render template without rendering children
- Integration testing
  - Render the whole component sub tree

Component Shallow Testing


```
it('should render with a title', async done => {
  await TestBed.configureTestingModule({
    declarations: [AppComponent],
    schemas: [NO_ERRORS_SCHEMA],
  }).compileComponents();

  const fixture = TestBed.createComponent(AppComponent);

  const h1 = fixture.nativeElement.querySelector("h1");
  expect(h1).toBeTruthy();
  expect(h1.textContent).toBe("My App");

  done();
});
```

Component Integration Testing


```
<app-clock [format]=“clockFormat"></app-clock>
```

```
it(‘should push format into clock', async done => {
  await TestBed.configureTestingModule({
    declarations: [AppComponent, ClockComponent],
  }).compileComponents();

  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  const clock = fixture.debugElement.query(x => x.name == "app-clock").componentInstance;

  app.clockFormat = "HH";
  fixture.detectChanges();

  expect(clock.format).toBe("HH");

  done();
});
```

Component with Async Service


```
it('should load all contacts on initialization', async done => {
  await TestBed.configureTestingModule({
    declarations: [AppComponent],
    schemas: [NO_ERRORS_SCHEMA],
    providers: [ContactService],
  }).compileComponents();

  const contacts = [{id:1, name: "Ori"}];
  const service = TestBed.get(ContactService);
  const spy = spyOn(service, 'getAll')
    .and.returnValue(Promise.resolve(contacts));

  const fixture = TestBed.createComponent(AppComponent);
  const comp = fixture.componentInstance;
  fixture.detectChanges(); // let ngOnInit run

  await spy.calls.mostRecent().returnValue;

  expect(comp.contacts).toBe(contacts);

  done();
});
```

```
export class AppComponent {
  contacts: Contact[];

  constructor(private contactService: ContactService) {
  }

  async ngOnInit() {
    this.contacts = await this.contactService.getAll();
  }
}
```

async

- Angular utility
- Allows for easier syntax by using Zones
- Monitor all asynchronous activities and notify jasmine on completeness

```
beforeEach(async(() => {
  TestBed.configureTestingModule({
    declarations: [ BannerComponent ],
  })
    .compileComponents();  // compile template and css
}));
```

Page Object

- A common pattern amongst testers
- Wraps HTML inside a class
- Allowing you to manipulate page element without digging into HTML

```
class Page {
  saveBtn: DebugElement;
  cancelBtn: DebugElement;

  constructor(fixture) {
    const buttons    =
fixture.debugElement.queryAll(By.css('button'));
    this.saveBtn     = buttons[0];
    this.cancelBtn   = buttons[1];
  }

  save() {
    this.saveBtn.triggerEventHandler("click");
  }
}
```

End to End Testing

- Selenium RC
- Selenium 2 & WebDriver API
- selenium-webdriver binding to NodeJS
- Protractor

Selenium Remote Control (RC)

- The original selenium solution for automating tests
- Injects some JavaScript into the page
- The injected code interacts with Selenium Server
- The test app interact with the same Selenium Server
- Thus, the test app can communicate with the web page

Web Driver Protocol

- W3C standard (recommendation phase)
- A remote control interface for controlling user agents
- A platform and language neutral wire protocol
- Allows for discovering and manipulation of DOM elements
- Primary intent is to support automated tests
- https://www.w3.org/TR/webdriver/

Supported Browsers

- Chrome
- IE 7+
- Firefox
- Safari
- Opera
- PhantomJS
- Android
- iOS

selenium-webdriver

- A NodeJS binding to WebDriver API
- Must install a driver for each type of browser and put it inside PATH
  - chromdriver.exe
  - IEDriverServer.exe
  - Others …
- Then just use plain JavaScript inside NodeJS application

Using selenium-webdriver


```
const {Builder, By, Key, until} = require('selenium-webdriver');

let driver = new Builder()
    .forBrowser('chrome')
    .withCapabilities({
        browserName: "chrome",
        chromeOptions: {
            args: ['disable-infobars']
        }
    })
    .build();

driver.get('http://www.google.com');
driver.findElement(By.name('q')).sendKeys('webdriver', Key.RETURN);
driver.wait(until.titleContains('webdriver'), 1000);
driver.quit();
```

Selenium Server

- Most browser drivers do not accept remote connections
- Selenium server acts as a proxy between our test app and browser's driver
- Thus, allowing the browser to execute on a remote machine
- No need to use it for local execution
- A Java based application

Running with Selenium Server

- java -jar selenium-server-standalone-2.45.0.jar

```
var driver = new webdriver.Builder()
    .forBrowser('firefox')
    .usingServer('http://localhost:4444/wd/hub')
    .build();
```

Common API

- findElement
- By.name
- By.css
- getText
- click
- submit
- switchTo().window('windowName')
- navigate().forward

Control Flow

- WebDriver uses a special promise manager that schedule all promises in a serial fashion
- You write cleaner code
- No need to handle then and catch
- Might be surprising during debugging
- Can be disabled using SELENIUM_PROMISE_MANAGER

Protractor

- An abstraction on top of WebDriver
- Offers simpler API
- Offers some tools
  - Webdriver-manager
- Has integration with Angular
  - Mostly Angular 1

Getting Started

- Everything is already setup by angular/cli
- You may ensure browser drivers are up to date using webdriver-manager update
- npm run e2e
  - Launches WebDriver instance
  - Executes all specs under e2e folder

Protractor API

- browser
- element
- by
- ExpectedConditions

Page Object (again …)


```
export class AppPage {
  buttonInc: ElementFinder;
  counter: ElementFinder;

  constructor() {
    this.buttonInc = element(by.css("app-root button.inc"));
    this.counter = element(by.css("app-root span.counter"));
  }

  navigateTo() {
    return browser.get('/');
  }

  inc() {
    this.buttonInc.click();
  }
}
```

Testing the Page


```
describe('AppComponent', () => {
  let page: AppPage;

  beforeEach(() => {
    page = new AppPage();
  });

  it('inc the counter when clicked', () => {
    page.navigateTo();
    page.inc();

    expect(page.counter.getText()).toEqual('1');
  });
});
```

No Control Flow

- SET SELENIUM_PROMISE_MANAGER=0
- num run e2e

```
it('inc the counter when clicked', async () => {
  await page.navigateTo();
  await page.inc();

  expect(await page.counter.getText()).toEqual('1');
});
```

- Debugging is now much nicer ☺

Summary

- The big question is
  - Unit or E2E testing ?
- Probably both
- You need to find the balance
- Writing E2E is much easier today
- Still, investigating is hard and time consuming