**Angular Testing R&D**

# Continuous integration (CI)

The underlying idea is to commit small chunks of code at a time and test them gradually as they come, to avoid putting together a lot of code at the same time and having to face hell debugging. In our case, we wish to maintain CI for unit testing, the code may compile, but we also want the unit tests to pass without someone having to manually execute them every single time.

Some of the frameworks for CI (can slow down some tests):

* Travis CI (paid service, unless open source project): it is more of a monitoring service that continuously look for new commits on your GitHub repo and run tests on them. Setup is done with a *.travis.yml* file at the root of the project.
* Semaphore: works with GitHub
* Jenkins: can download extension to work with TFS

<https://github.com/jenkinsci/tfs-plugin/blob/master/README.md>

<https://medium.com/aubergine-solutions/continuous-integration-for-angular-with-angular-cli-jenkins-phantomjs-34a870fa096f> (guide)

* Circle CI (paid service, unless open source project): works with GitHub and Bitbucket. Whenever a new commit comes in, it builds the project and run tests on it. The team is given a report of the bugs. Automated deployment available

For a comparison of CI solutions, see:

<https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software>

Extra notes:

* Angular CLI is capable of code coverage reports, which shows any parts of our code base that may not be properly tested by unit tests.
* The CLI takes care of Jasmine and karma configuration for you.
* You can fine-tune many options by editing the karma.conf.js and the test.ts files in the src/ folder.

# Karma

* A tool that simulates a web server and executes source code against test code
* The Angular CLI command, ng test, launches Karma test runner
* Has support for Travis, Semaphore and Jenkins for CI
* When the browsers connect, Karma serves a 'client.html' page; when this page runs in the browser it connects back to the server via websockets
* This page includes the test framework adapter, the code to be tested, and the test code

# Jasmine

* An open-source JavaScript testing framework
* Angular CLI uses Jasmine framework
* We write all of our unit tests in this framework
* Expectations in Jasmine are adapted to understand promises (wait for promises to resolve before executing the expectations)
* Supports spies, which can be used to mocks services or individual functions
* Supports asynchronous testing
* Built-in matchers include:
  + 1. toBe (checks if two things are the same object)
    2. toBeTruthy
    3. toBeFalsy
    4. toContain
    5. toBeDefined
    6. toBeUndefined
    7. toBeNull
    8. toBeNaN
    9. toBeGreaterThan
    10. toBeLessThan
    11. toBeCloseTo (accepts two parameters and checks if a number is close to the first parameter, given a certain amount of decimal precision as indicated by the second parameter)
    12. ToEqual (checks the values of 2 things)

# Protractor

* A node.js program that is an e2e framework
* By default, uses Jasmine testing framework
* Is a wrapper around WebDriverJS, which uses a promise manager allowing us to write code in a synchronous fashion
* Adapts Jasmine so that each spec automatically waits until the control flow is empty before exiting, making e2e possible as e2e often involves executing an action and waiting for promises to resolve.
* A work flow is comprised of beforeEach(), afterEach() and the individual tests, it() in our case. It is a queue of pending promises that helps organize the execution.

# Unit Tests

* Tests a unit in isolation. In our case, this is an individual Angular component most of the times but it can also be a function.
* Involves mocking a lot, as we are mainly interested in one unit’s functionality at a time. If we wish to test a component with many dependencies, we use mocks and spies on those dependencies.
* The simplest form of tests, easy to write and understand

# e2e

* Stands for end-to-end testing
* Can be implemented using Protractor framework
* We gradually move away from mocking in e2e, as we are interested in testing the actual functionality of the units together
* Usually slower to run, as it tries to test whole application
* Applicable when for example, testing that the router navigates to a specific page and that the view is re-rendered accordingly
* E2E tests are great for high-level validation of the entire system. But they can't give you the comprehensive test coverage that you'd expect from unit tests.
* E2E tests are difficult to write and perform poorly compared to unit tests. They break easily, often due to changes or misbehavior far removed from the site of breakage.
* E2E tests can't easily reveal how your components behave when things go wrong, such as missing or bad data, lost connectivity, and remote service failures.
* E2E tests for apps that update a database, send an invoice, or charge a credit card require special tricks and back-doors to prevent accidental corruption of remote resources. It can even be hard to navigate to the component you want to test.
* Because of these many obstacles, you should test DOM interaction with unit testing techniques as much as possible.

# Integration Tests

* Tests the interaction of many units together as a group. In our case, the interaction between the Angular components and with the backend logic.
* Should be done after unit tests passed
* Big Bang approach: all or most of the units are combined together and tested in one go
* Top Down approach: where top-level units are tested first and lower level units are tested step by step after that
* Bottom Up approach: bottom level units are tested first and upper-level units step by step after that.
* *Sandwich/Hybrid* is an approach to Integration Testing which is a combination of Top Down and Bottom Up approaches.
* **component integration testing:**Testing performed to expose defects in the interfaces and  
  interaction between integrated components

Main Sources:

<https://angular.io/guide/testing>

<https://hackernoon.com/testing-your-frontend-code-part-i-introduction-7e307eac4446>

<http://softwaretestingfundamentals.com/integration-testing/>

<https://en.wikipedia.org/wiki/Jasmine_(JavaScript_testing_framework)>

https://jasmine.github.io/2.9/introduction