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:
  1.1.1. toBe (checks if two things are the same object)
  1.1.2. toBeTruthy
  1.1.3. toBeFalsy
  1.1.4. toContain
  1.1.5. toBeDefined
  1.1.6. toBeUndefined
  1.1.7. toBeNull
  1.1.8. toBeNaN
  1.1.9. toBeGreaterThan
  1.1.10. toBeLessThan
  1.1.11. 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)
  1.1.12. 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.

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