UI Testing with

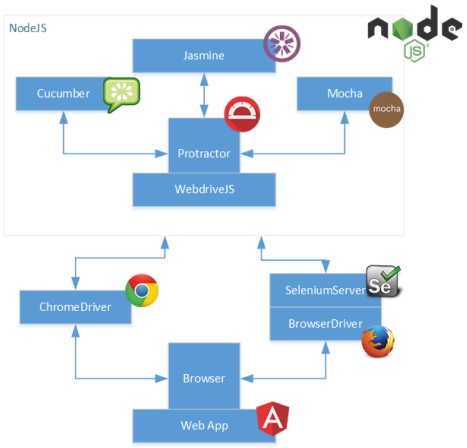
Testing your form

Two options:

1. Integration test
   - Using Karma
   - Use Angular testing utilities
   - Manually trigger events and manipulate input
2. UI test
   - Using Protractor
   - Use the actual *live* application
   - Test like an end user

We'll be using UI test to test our form.

Protractor overview



Setup

- Install Protractor
  `npm install --save-dev protractor`
- Make sure your development server is running
- Run Selenium if needed:
  `node node_modules/.bin/webdriver-manager start`
- Run Protractor:
  `node node_modules/.bin/protractor protractor.conf.js`
- This is already done for us in the ng cli generated project

Configuration

- Configure in `protractor.conf.js` file

```
module.exports.config = {     // Export the config object as a node module

  directConnect: true,        // Use ChromeDriver directly instead of Selenium

  capabilities: {             // Set the capabilities (browser) for webdriver.js
    'browserName': 'chrome'
  },

  framework: 'jasmine',       // Configure test framework to use

  specs: [                    // Point to the test files here
    'test/helpers/**/*.js',
    'test/ui/**/*.js' ],

  jasmineNodeOpts: {          // Add jasmine/node/cucumber specific settings
    defaultTimeoutInterval: 30000
  },

  useAllAngular2AppRoots: true  // Signal protractor that we're using angular 2
};
```

Usage

They modeled Protractor after jasmine

```
// Import stuff from protractor
import { browser, element, by } from 'protractor';

describe('Home page', () => {
    beforeEach(() => {
        // Tell the browser to navigate to a URL
        browser.get('http://localhost:4200');
    });

    it('should display the page title', () => {
        // Use `element` and `by` to select elements on a page
        const title = element(by.css('app-root h2')).getText();
        expect(title).toBe('Welcome to the app'));
    });
});
```

Using Protractor with page objects

```
// homePage.uitest.ts
import { HomePage } from './page-objects/home-page';

describe('Home page (using Page Object)', () => {

    beforeEach(() => {
        const homePage = new HomePage();
        homePage.get();
    });

    it('should display the page title', () => {
        expect(homePage.title()).toBe('Welcome to the app');
    });
});
```

```
// pageObjects/home-page.ts
import { browser, element, by } from 'protractor';

export class HomePage {
    get() { browser.get(''); }
    title() {
        return element(by.css('app-root h2')).getText();
    }
}
```

Page Objects

- Use page objects to **isolate all interactions** with the page
- When the HTML of your page evolves, your tests remain untouched
- PageObject methods make specs easier to read:

```
homePage.firstItem();
```

vs

```
element.all(by.css('app-root li')).first();
```

Available locators

- `by.css('.class element')`
- `by.id('id')`
- `by.name`

More locators: http://www.protractortest.org/#/locators

Protractor is async

```
<h2>Superheroes</h2>
```

```
let title = element(by.css('h2')).getText();

expect(title).toBe('Superheroes') // works!

expect(title === 'Superheroes').toBe(true); // breaks!
```

- All commands must be sent over the wire to the browser
- Each Protractor method in a spec is just:
  - Queuing up a command
  - Returning a promise
- Jasmine's `expect`, `beforeEach` and `it` have been adapted
- Control flow requires dealing with promises

Dealing with promises

Sometimes you'll need to manually deal with promisses.

```
it('should display the right number of heroes on the page', () => {
    const myHeroes = [{ id: 'superman' }, { id: 'spiderman' }];
    const promises = myHeroes.map((myHero) => {
        element(by.id(myHero.id)).isPresent();
    });
    protractor.promise.all(promises).then((isPresentValues) => {
        isPresentValues.forEach((presentValue) => {
            expect(presentValue).toBeTruthy();
        });
    });
});
```