

# PG6300-14 Webutvikling & API-design lecture notes 09: Testing

Martin Lehmann

12th April 2015

## 1 Intro

Core functionality is now complete (except security in WebSockets – not part of the curriculum).

Stability time! Testing is a great way of ensuring stability AND expressing user stories (thus removing the need for extensive documentation).

Tests developers know when they made a mistake that messed up the rest of the application. You’ve all done (part of) this before.

A well-written application uses both end-to-end and unit tests on both the client and the server.

Minor side note: **Dev dependencies**

```
$ npm install --save mongoose
```

```
$ npm install --save-dev gulp
```

## 2 End-to-end testing

- Test *everything* from UI to database
- Slow, but very good for catching errors (catch everything)

### 2.1 Protractor

- A tool for running end-to-end tests in Angular JS applications
- Not limited to Angular, but specifically designed for Angular (by the AngularJS team)
- Protractor is a Node.js app

- Uses WebDriver & Selenium to run an actual browser (thus, slow)
- Install as dev dependency via NPM

```
$ npm install --save-dev protractor
```

- Protractor has very large files for an NPM package – don't include it as a production dependency as deployment will be slowed down.

- Set up WebDriver with Selenium "automagically"

```
$ ./node_modules/.bin/webdriver-manager update
```

```
Updating selenium standalone
```

```
downloading https://selenium-release.storage.googleapis.com/2.45/selenium-server-standalone-2.45.0.jar
```

```
Updating chromedriver
```

```
downloading https://chromedriver.storage.googleapis.com/2.14/chromedriver_mac32.zip ...
```

```
chromedriver_2.14.zip downloaded to /Users/theneva/Dropbox/WACT/PG6300 Webutvikling og
```

```
selenium-server-standalone-2.45.0.jar downloaded to /Users/theneva/Dropbox/WACT/PG6300
```

```
$
```

- Protractor is just a test *runner*. Time to add a testing framework!
- Several options
  - QUnit (for JQuery): inflexible, verbose, weak when it comes to asynchronous and promise-based testing
  - Jasmine (for Behaviour-Driven Development): Based on testing in Ruby. Much more concise than QUnit, but weak when it comes to asynchronous and promise-based testing. The choice for most Angular apps (including the Angular team)
  - Mocha: The choice for most Node applications. Flexible, pick & choose tools that fit your application. Well supported with all Angular tools. Will be using this.

## 2.2 Mocha

- Test framework (you write tests with Mocha, just like with JUnit for Java, NUnit for .NET, XCTest for Objective-C and Swift, and so on)
- Requires a few configuration options, but nothing too bad

## 2.3 Basic Protractor test

- Convention to put all tests in *project root/test/*
- Three different test methods:

- End-to-end: *project/test/e2e/*
- Node server: *project/test/node*
- Angular: *project/test/angular*
- ex:

```
$ mkdir -p test/e2e test/angular test/node
```

```
project root
├─ test
│   └─ angular
│       └─ e2e
│           └─ node
```

### 2.3.1 First test time!

- Protractor is great for describing user stories; design tests with users in mind
- E-commerce: Come to the site, find a product, add it to the shopping card, complete the order
- Tests hit many (all?) parts of the application and ensure that common flows are always stable
- Too many tests (or too fine-grained tests) become too slow, take too long to write, and thus changing design becomes bad. Feature tests are good, but use with care

First test: *test/e2e/making-a-post.spec.js*

```
describe('making a post', function() {
  it('logs in and creates a new post', function() {
    // go to homepage
    // click 'login'
    // fill out and submit login form
    // submit a new post on the posts page

    // The new post should be visible as the first post on the page
  });
});
```

- Tests MUST be named \*.spec.js for Protractor to find them – this allows us to have normal .js utility files that are not treated as tests
- "describe" describes a test scenario to give context. Can be nested!
- "it" is an actual test

- Final assertion on its own line ('The new post should be visible as the first post on the page'): Think of the final assertion first. What should ultimately be tested? Could describe the flow backwards so that you only need to think of one prerequisite at a time.

From pseudocode to a (barely) running test (test/e2e/making-a-post.spec.js):

```
describe('making a post', function() {
  it('logs in and creates a new post', function() {
    // go to homepage
    browser.get('http://localhost:3000');
    // click 'login'
    // fill out and submit login form
    // submit a new post on the posts page

    // The new post should be visible as the first post on the page
  });
});
```

Configure Protractor to use Mocha (and tell it where to find your tests)

*project root*/protractor.conf.js

```
exports.config = {
  framework: 'mocha',
  specs: [
    'test/e2e/**/*.spec.js'
  ]
};
```

Install Mocha as a dev dependency:

```
$ npm install --save-dev mocha
```

Run Protractor (the same command as previously, but without 'update'). **NB:** This requires the file retrieved at `http://localhost:3000` to be an actual Angular application. You cannot run this on an empty HTML file!

```
$ ./node_modules/.bin/protractor
```

This...

- Opens a browser window briefly (that's WebDriver/Selenium, required to actually perform the tests)
- Prints information in the console:

```
./node_modules/.bin/protractor
Starting selenium standalone server ...
[launcher] Running 1 instances of WebDriver
Selenium standalone server started at http://10.21.24.41:64200/wd/hub
```

```
. making a post logs in and creates a new post: 457ms
```

```
1 passing (460ms)
```

```
Shutting down selenium standalone server.
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #1 passed
```

Start node inside Protractor to avoid having to keep the server running at all times.

*project root/protractor.conf.js*

```
exports.config = {
  framework: 'mocha',
  specs: [
    'test/e2e/**/*.spec.js'
  ],
  onPrepare: function() {
    require('./server-node/hello-tests-server.js');
  }
};
```

...but this makes you unable to already have a server running. Use a different port set by environment variables!

Edit server.js to pick its port from an environment variable:

```
var express = require('express');
var app = express();
...
var port = process.env.PORT || 3000;
...
app.listen(port, function() {
  console.log('Listening on port ' + port);
});
```

Then set the environment variable inside Protractor's onPrepare:

*project root/protractor.conf.js*

```

exports.config = {
  framework: 'mocha',
  specs: [
    'test/e2e/**/*.spec.js'
  ],
  onPrepare: function() {
    process.env.PORT = 3001;
    require('./server-node/hello-tests-server.js');
  }
};

```

...and finally point the test code to the port of the server started by Protractor (3001):

```

describe('making a post', function() {
  it('logs in and creates a new post', function() {
    // go to homepage
    browser.get('http://localhost:3001');
    // click 'login'
    // fill out and submit login form
    // submit a new post on the posts page

    // the new post should be visible as the first post on the page
  });
});

```

## Protractor **Locators**

Interact with the DOM elements on the page! For more information, see <https://github.com/angular/protractor/blob/master/docs/locators.md>.

First, get a hold of the DOM element:

```

// find an element using css selector
by.css('someclass');

// find an element using id
by.id('someid');

// find an element by ng-model (e.g. username)
by.model('username');

// find an element by binding (e.g., {{currentUser}})
by.binding('currentUser');

```

Pass the locator to `element(locator)`, and you can interact with it:

```
// click a button or link
element(by.css( '.mybutton ')).click();

// fill out a text input
element(by.css( '.username-input ')).sendKeys( 'theneva ');
```

**NB:** These are all asynchronous events!

### 3 Unit testing

- Test isolated components (single units, also known as functions and classes)
- Used in both Node and Angular, but completely separately
- Every test is ignorant of the other tests
- These are the tests you write before coding
- Great for test-driven development (TDD)