Fork me on GitHub

PRACTICAL

AngularJS

Based on blog posts by

Dinis Cruz

beta version

Practical AngularJS

Real-world examples of using AngularJS in multiple scenarios and platforms (KarmaJS, Firebase, Eclipse, WebStorm, O2 Platform)

Dinis Cruz

This book is for sale at http://leanpub.com/Practical_AngularJS

This version was published on 2014-03-29

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

This work is licensed under a Creative Commons Attribution 3.0 Unported License

Tweet This Book!

Please help Dinis Cruz by spreading the word about this book on Twitter!

The suggested hashtag for this book is #PracticalAngularJS.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#PracticalAngularJS

Contents

Introduction

This book contains the AngularJS related blog posts posted on Dinis Cruz' blog at http://blog.diniscruz.com.

This is the first draft release of this book, so please send your sugesttions, criticisms, ideas or comments to dinis.cruz@owasp.org

Notes about current structure

The first version of this book had the chapter order created by the original 'import from blogger' (i.e. by publish order).

In the current version, the posts are split into the following areas: "Using AngularJS", "KarmaJS", "Firebase", "Misc Tricks", "IDEs", "Troubleshooting", "Appendices", which is a filter based on technology. Note that this can change based on reader feedback, so if you think the order should be different, please send your feedback and ideas.

About the Author

Dinis Cruz is a Developer and Application Security Engineer focused on how to develop secure applications. A key drive is on 'Automating Application Security Knowledge and Workflows' which is the main concept behind the OWASP O2 Platform and Security Innovation's TeamMentor (Dinis is the main developer and architect of both Applications). Current day job is with Security Innovation where Dinis tries to promote openness, quality and sharing as part a core tenet of TeamMentor's application development environment.

After many years (and multiple roles) Dinis is still very active at OWASP, currently leading the O2 Platform project and helping out other projects and initiatives.

After failing to scale his own security knowledge, learned Git, created security vulnerabilities in code published to production servers, delivered training to developers, and building multiple CI (Continuous Integration) environments; Dinis had the epiphany that the key to application security is "Secure Continuous Delivery: Developer's Immediate Connection to What They're Creating". This 'Immediate Connection/Feedback' concept is deep rooted in the development of the O2 Platform/TeamMentor, and is something that will keep Dinis busy for many years.

Table of Contents | Code

Change log:

In 2014, here are the changes made (per version):

- **v0.20 (29 March)**
  - updated Readme.md file to point to existing articles.
  - renamed all content files to *.md (which make sense since they are Markdown files, and they look much better at GitHub) and gave them a name that is related to its current chapter
  - new files: C0-Introduction/Table_of_contents.md
  - created a GitHub-only 'table of contents' which links to all available chapters. Each chapter intro page also has the links for its child articles. Added some navigation links to the end of each article (to allow easy navigation when browsing the content in GitHub)
- **v0.13 (16th March)**
  - added new cover to eBook version
  - lots of formating changes to most chapters/posts (specialy on current source code samples)
  - added leanpub attributes for: frontmatter, mainmatter, backmatter
  - major rename of markdown file names to help to quickly see which file belongs to what chapter (see github repo)
- **v0.12 (11th March)**
  - fixed spellings
  - fixed nested list in this change log (needs 4 spaces on line start)
  - fixed use of sections in 'Appendix Posts Details'
  - created github repository for this book: https://github.com/DinisCruz/Book_Practical_AngularJS
- **v0.11 (11th March)**
  - Create the following main section: "Using AngularJS", "KarmaJS", "Firebase", "Misc Tricks", "IDEs", "Troubleshooting", "Appendices"
  - Changed the order of the chapters (using the new sections created, instead of the original import structure (by month))
  - Book configuration change to add page break for every section (i.e. blog post)
  - Added this change log
  - Created Git repo on local dropbox sync folder
  - Added Appendix "Post's Details"
- **v0.10 (10th March)**
  - First release of book with raw import from blogger posts (no formatting or editing done)

Table of Contents | Code

1 Using AngularJS

This section has the following chapters:

- A really SIMPLE and clean AngularJS+Firebase example
- Using AngularJS in Eclipse, Part 1) The Basics
- Using AngularJS in Eclipse, Part 2) Add Some Control
- Using AngularJS in Eclipse, Part 3) Wire up a Backend
- Using AngularJS in Eclipse, Part 4) Create Components
- AngularJS code editor using UI-Bootstrap and CodeMirror (done without using jQuery)

GitHub Navigation: Table of Contents | Code

1.1 A really SIMPLE and clean AngularJS+Firebase example

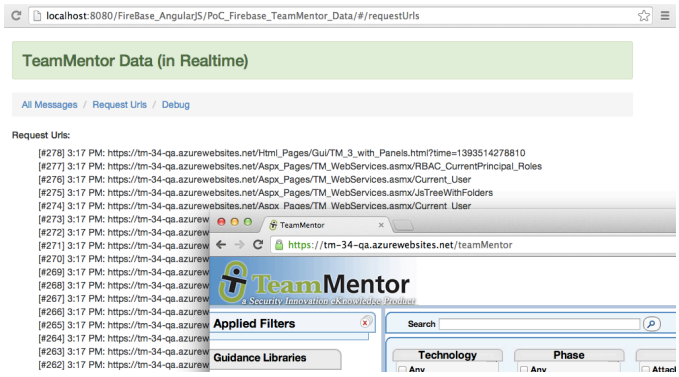
As seen on the First PoC of sending TeamMentor's server-side request URLS to Firebase (and seeing it in realtime in an AngularJS page) I created a ***Simple*** AngularJS website which I'm very happy with (and I mean ***Simple*** with a capital ***S***).

The main reason I really like the solution shown below, is because it represents a number of really nice, clean and ***Simple*** solutions for common (complex) problems that exist while developing in Javascript.

The created application is an:

- AngularJS real-time viewer for HTTP requests,
- … made to an ASP.NET web application (TeamMentor),
- … captured by an custom C# HttpHandler filter,
- … submitted to Firebase using its REST API and
- … pushed back to the AngularJS app using open HTML 5 WebSockets.

The image below shows what the AngularJS+Firebase application looks like, with the urls shown in the background browser being the ones requested when the TeamMentor website is loaded or navigated (note that the latency between *'request made'* _and _*'request listed'* is really small (about ~ 50 milliseconds)):



What I also like about the AngularJS structure that I ended up with, is that it represents a great way to learn AngularJS' architecture and capabilities (and yes I know and agree that for bigger AngularJS apps it is better to organise by feature and group the multiple files under a dedicated folder (for example the login controller, service, factory, view and tests should all go under the same logical folder))

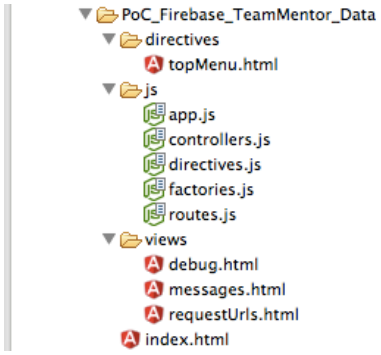
This post contains a tour of the multiple files created (gist here) which where developed/refactored while in Eclipse using the Eclipse Groovy REPL script to sync a Browser with file changes (with recursive folder search via Java's WatchService)

Folder/File Structure:

Inside an Eclipse Web Static project, I created a file structure with:

- _index.html _as the main AngularJS file (i.e. this is ***the*** single-page application file)

- all javascript files were placed in the **_js** _folder (with the file names providing a clue on what they are doing/providing)
- 1 directive created in the ***directives*** folder
- 3 views placed on the ***views*** folder



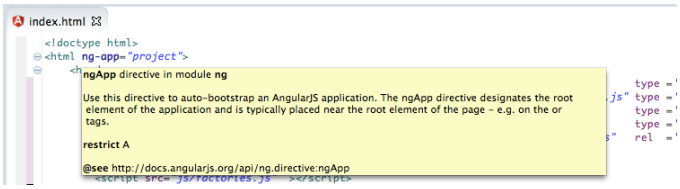
1) Index.html

This is the file loaded directly by the browser, which is made of:

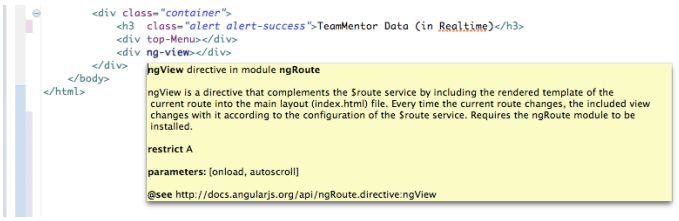
- External/Framework javascript includes: *angular.js, angular-route.js, firebase.js, angularfire.js, bootsrap.min.css*
- AngularJS javascript includes (for the current application): *app.js, factories.js, controllers.js, directives.js, routes.js*
- CSS link to *bootstrap.min.css*
- Html body (containing the AngularJS **_ng-app** _directive) and:
  - div tag with **_container** _css class
  - h3 tag with ***alert-success*** css class (which creates that nice green top banner)
  - the custom ***top-menu* _**directive (*using* _*attribute* instead of *element*)
  - the AngularJS ***ng-view*** directive

Since I'm using the AngularJS Eclipse plugin, hovering the mouse op top of an AngularJS directive provides a nice description of they do.

Here is the _ng-app _directive



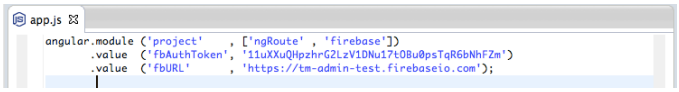
... and here is the ***ng-view*** directive:



2) app.js

This is where the ***project*** module is created (with two dependencies ***ngRoute*** and ***firebase***).

Since I moved the controllers, factories, directives and routes into their own separate js file, there wasn't much to do here, apart from creating global values for the firebase URL and auth token (which will be dependency injected into the controllers and factories)



3) controllers.js

This file contains 3 controllers: ***DebugCtrl***, ***MessagesCtrl*** and _RequestsUrlCtrl _(each will be used on a specific view)

Note that each controller has services injected into them (the AngularJS ***$scope _**and the custom **_-fbDebugMsg***, ***fbRequestUrl***, ***fbaDebugUrl***)

The *DebugCtrl _is currently just adding the injected _fbDebugMsg* and **fbRequestUrl **services into the *$scope* so that we can see them in the view (this is a nice trick to get an inside view of AngularJS objects)

The *MessagesCtrl* is using the Firebase AngularFire API, which is makes it really easy to create the firebase-real-time update view (my only problem with this was that there didn't seem to be an easy way to re-order the new elements (which in the current AngularFire implementation are added at the end of the provided array)

The _RequestsUrlsCtrl _uses the default Firebase Javascript API (i.e not the AngularFire one) which gives us more control on how to handle the data received by Firebase. The **_$scope.urls** _array is used to store the data received from the Firebase **_child_added** _event (one note here to say that the Firebase _child_added _will also provide the entire data-set on first load, which is okish, but I would prefer that the _child_added _only fired for new events)

**4) directives.js **

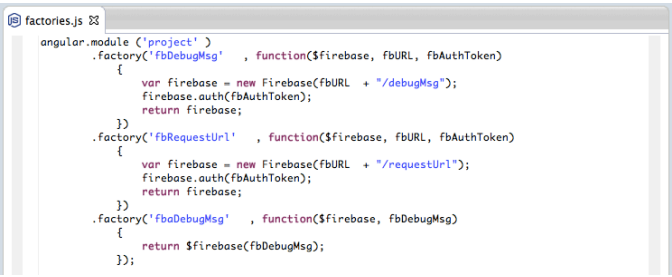
This is a simple directive used by **index.html**, that will display a top menu, created by the content of the ***topMenu.html*** file (directives are AngularJS way to creating/defining new HTML tags/attributes)

5) factories.js

These factories create the Firebase mappings, namely they define the 'area' (or namespace/object) that the data will be read from.

The first two (***fbDebugMsg*** and ***fbRequestUrl***) use the Firebase Javascript API. I needed to do them this way so that I could add the Firebase auth token (that said, I'm sure there is a better way to do this in Angular, since ideally I would have an Angular service that took the two variables that need to be set: the target Firebase area and auth code)

The _fbaDebugMsg _is just a simple service/factory to return an AngularFire API object based on the (dependency injected) ***fbDebugMsg*** service



6) routes.js

The routes configuration is basically defining the 3 available views (each with a different controller mapped to it)

routes.js

```
angular.module ('project')
       .config(function($routeProvider)
        {
            $routeProvider.when   ('/messages'    , { templateUrl : 'views/messages.html'    ,
                                                      controller  : 'MessagesCtrl'          })
                          .when   ('/requestUrls', { templateUrl : 'views/requestUrls.html' ,
                                                      controller  : 'RequestUrlsCtrl'       })
                          .when   ('/debug'       , { templateUrl : 'views/debug.html'       ,
                                                      controller  : 'DebugCtrl'             })
                          .otherwise(               { redirectTo  : '/requestUrls'
        });
```

**7) requestUrls.html ** (view)

Since the *RequestsUrlsCtrl _is responsible for updating the **$scope.urls **array, all we need to do here is to use Angular's _ng-repeat _directive to create a list with all items (the _list-unstyled _class hides the bullet usually shown in HTML _<li>*** tags).

Note that since the *RequestsUrlsCtrl* controller is using the Firebase Javascript API ***child_added*** event, we will see new entries shown in real time (ie. no browser refresh needed), but any changes made to existing items will not be reflected on the UI (unless the entire page is refreshed and the data is reloaded)

requestUrls.html

```
<h5>Request Urls:</h5>
<ul>
    <li ng-repeat="url in urls" class="list-unstyled">
        {{url}}
    </li>
</ul>
```

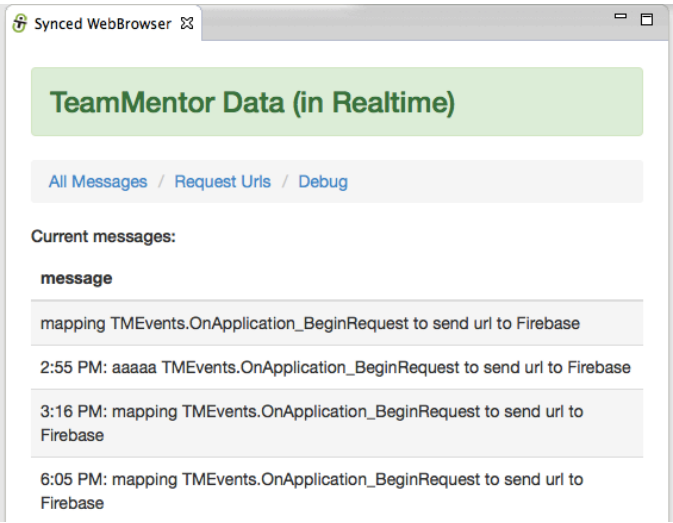
8) messages.html (view)

In this view the ***$scope.messages*** (used in the ***ng-repeat***) is populated by the **MessagesCtrl **controller which is using the AngularFire API. This means that data will be updated in real time (on both add and change events)

messages.html

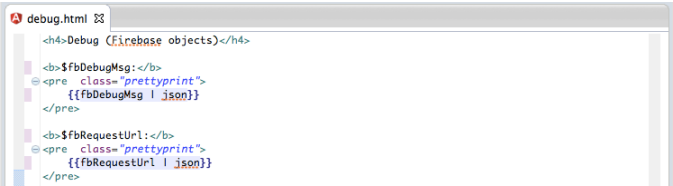
```
<h5>Current messages:</h5>

<table class="table table-striped">
    <thead>
        <tr>
            <th>message</th>
        </tr>
    </thead>
    <tbody>
        <tr ng-repeat="message in messages" >
            <td>{{message}}</td>
        </tr>
    </tbody>
</table>
```

... which look like this:

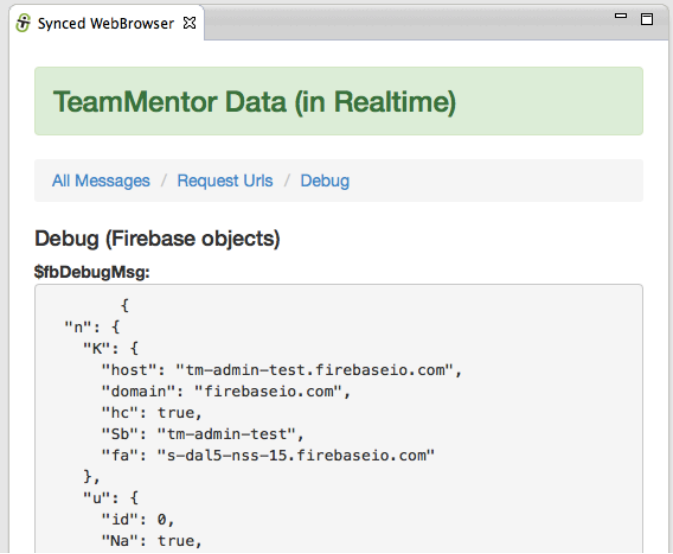


9) debug.html (view)

This view just shows a json representation of the *fbDebugMsg* and **_fbRequestUrl _**

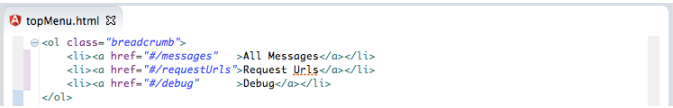


... which looks like this:



10) topMenu.html (directive templateUrl)

Finally this is the html that creates the top menu (this could be improved/refactored by having the url and titles being provided as a ***ng-model*** object)



All code:

For reference here is the entire source code (gist here) of the source code files shown above:

Btw, did you noticed anything different in the formatting of the code samples above?

Is it easier to read?

If you are interested in the topic of code formatting also see On Java code formatting and Formatting code for readability

---

1.2 Using AngularJS in Eclipse, Part 1) The Basics

This is the first of four posts on how to run (inside Eclipse) the examples provided in AngularJS's home page:

- Using AngularJS in Eclipse, Part 1) The Basics
- Using AngularJS in Eclipse, Part 2) Add Some Control
- Using AngularJS in Eclipse, Part 3) Wire up a Backend
- Using AngularJS in Eclipse, Part 4) Create Components

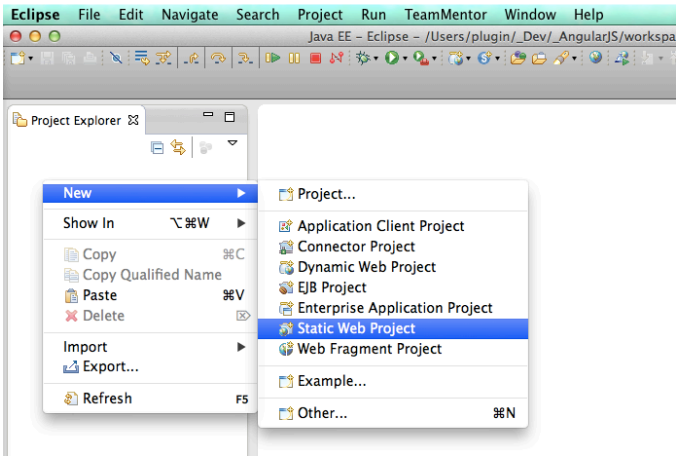
The example covered on this post is the ***The Basics***.

I'm doing this on an OSX laptop and the first step was to download and unzip (eclipse-standard-kepler-SR1-macosx-cocoa.tar.gz (32bit version of Eclipse's Kerpler) into the *~/_Dev/_AngularJS* folder.

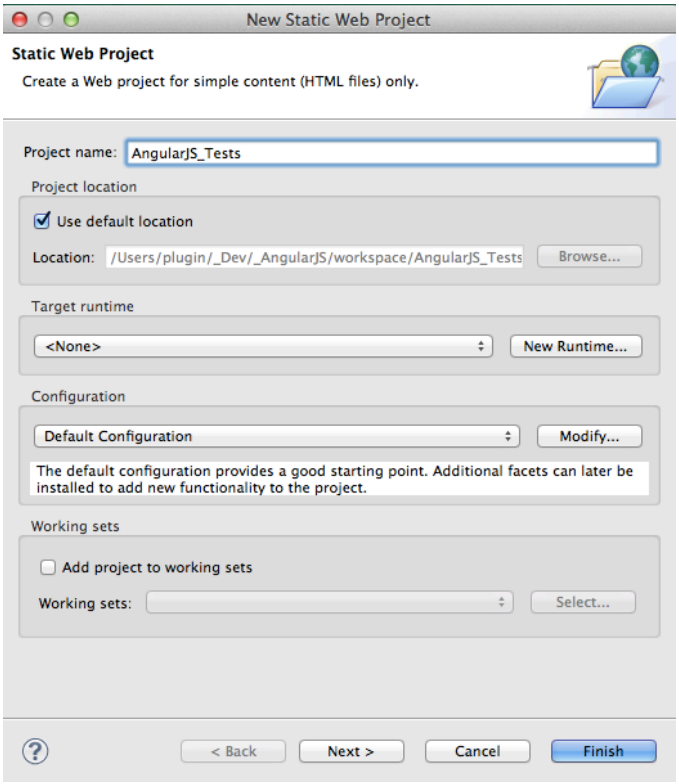
I fired up eclipse, chose the *~/_Dev/_AngularJS/workspace* as the workspace root and installed the Eclipse Groovy REPL Scripting Environment 1.6.0 (update site) and Angular-JS Eclipse Tooling (update site) plugins.

1) Creating an Angular JS Project

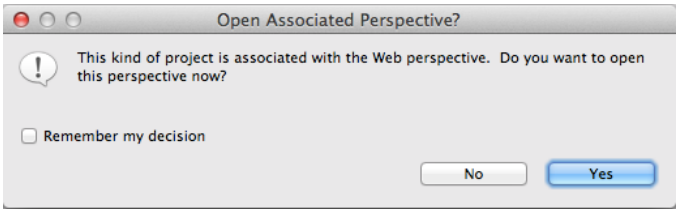
After restarting eclipse, I right-clicked on the ***Project Explorer*** view and chose the ***New*** -> ***Static Web Project*** menu item



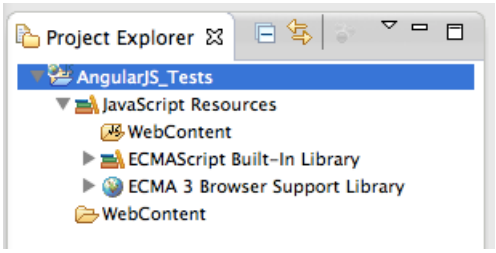
... set ***AngularJS_Tests*** as the project name and clicked ***Finish***



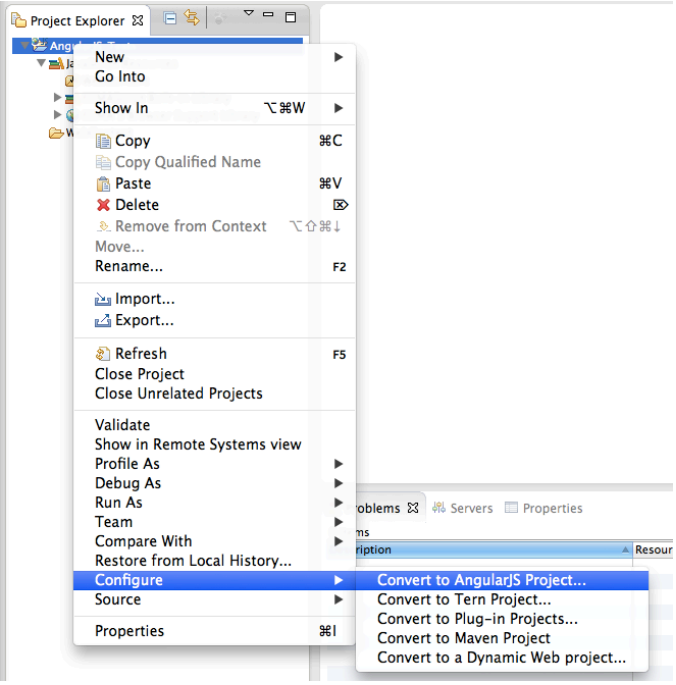
... switched to the ***Web Perspective***



... with the ***Project Explorer*** view now looking like this:

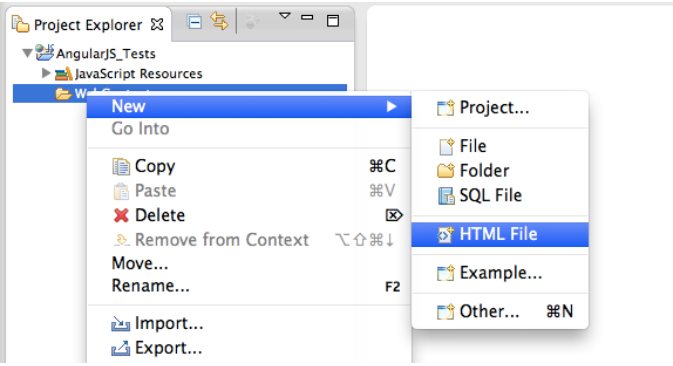


With the final setup being the conversion into an ***AngularJS Project***

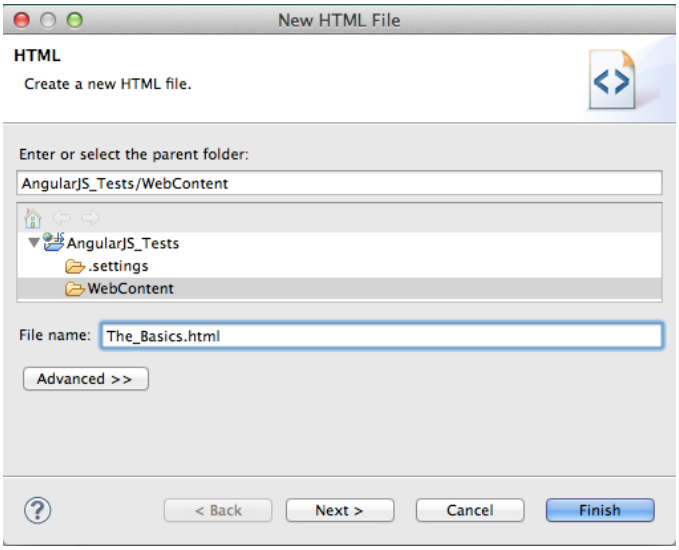


2) Creating the The_Basics.html file

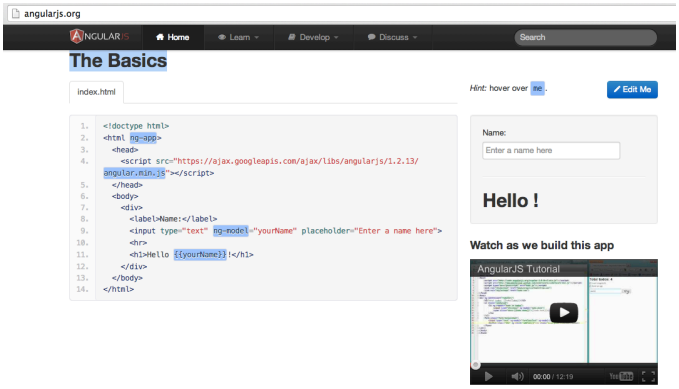
To create the first test file, I right-clicked on the *Web Content* folder, and chose the *New _-> _Html File* menu option:



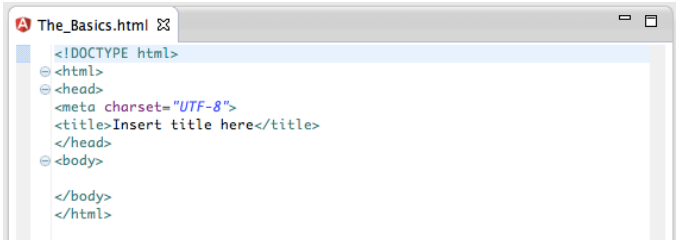
... set the file name to *The_Basics.html* and click on ***Finish***



**NOTE: **The reason this first file is called *The_Basics.html* is because I'm going to be using the examples from AngularJS' home page http://angularjs.org/



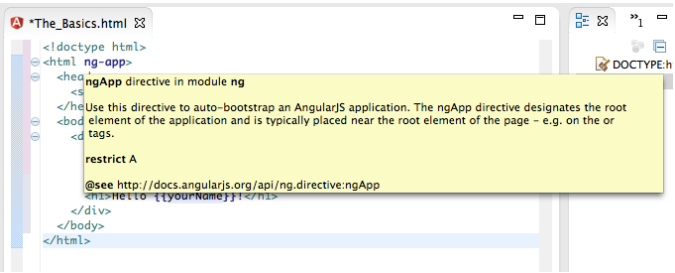
Once the _The_Basics.html _file opens up in eclipse



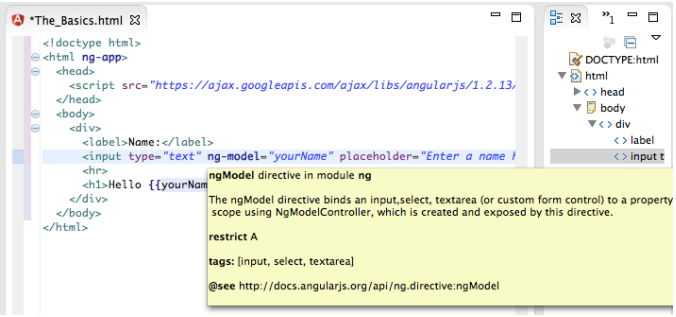
... I change its contents to the code sample from http://angularjs.org/

Note how the AngularJS Eclipse plugin successfully detects the Angular attributes and showed relevant information about it.

Here is ***ng-app***:

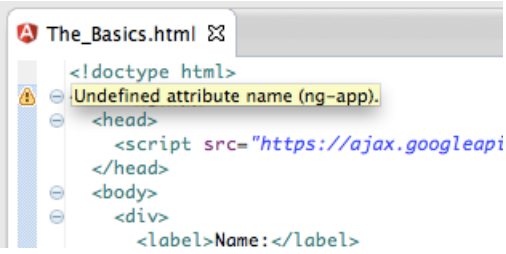


Here is ***ng-model***:



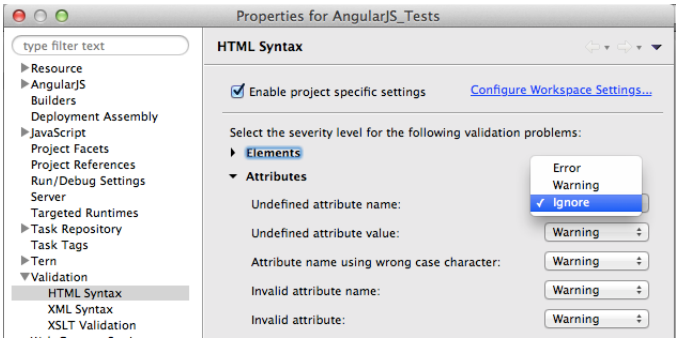
3) Fixing the undefined Attribute name issue

Since I chose to the Eclipse's **Static Web Project**, when we save the _The_Basics.html _file (and if the Eclipse's Html validation settings are the default ones), the following error will show:

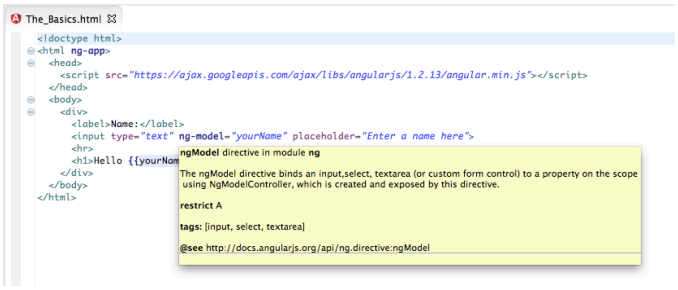


... which basically means that Eclipse is not recognising the AngularJS Html attributes:

To fix this, I went to the ***AngularJS_Test _**project's Properties, opened the _HTML Syntax _page (from the **_Validation*** section) and set to false the **Undefined attribute name **setting (in the ***Attributes'*** options , not the ***Elements***)



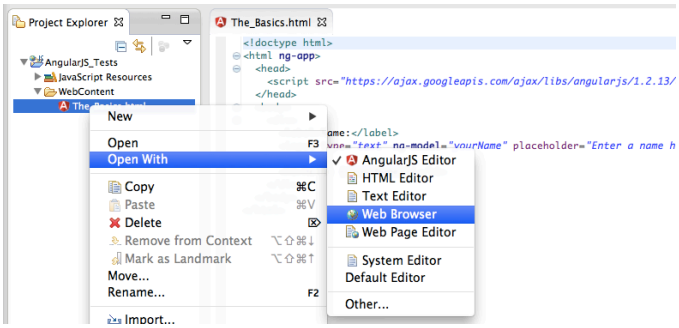
With that config change, there are no problems in this page, and hovering on top of one the AngularJS directives will show the correct tooltip:



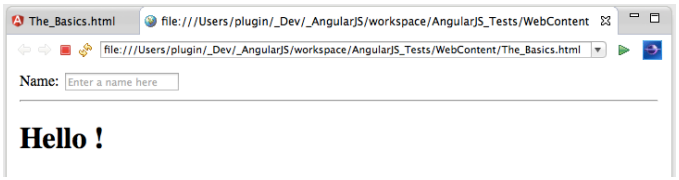
**4) Viewing and previewing the **_The_Basics.html page_

Since at the moment we only have one page, we can view it directly without needing a Web Server.

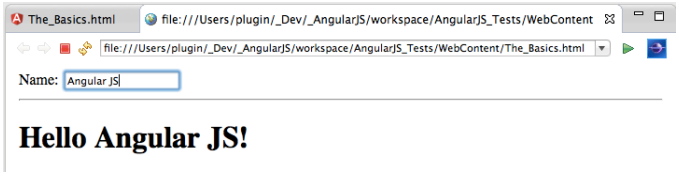
To do that, I clicked on the html file and chose the ***Web Browser*** option from the ***Open With*** menu:



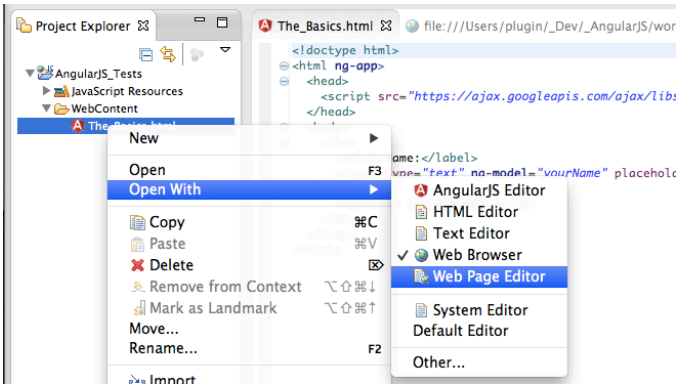
This will open the default Eclipse Browser



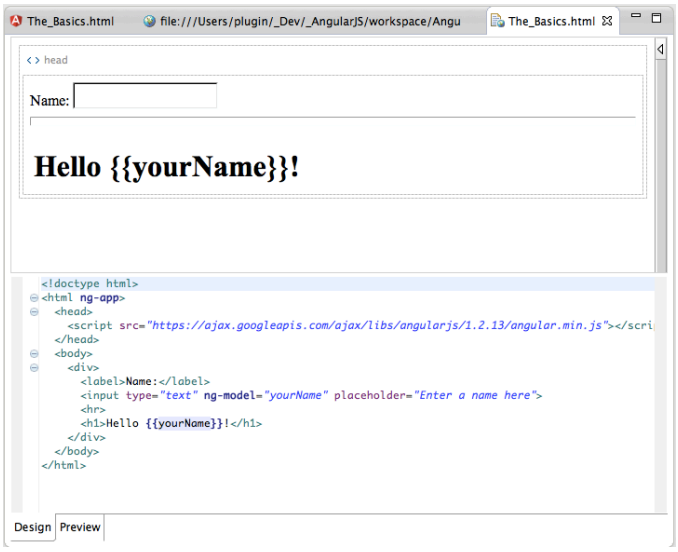
... with the AngularJS test working as expected (in this case any text typed in the ***Name*** TextBox will automatically be shown in the page:



We can also preview some of the changes in real time, by choosing the ***Web Page Editor***:



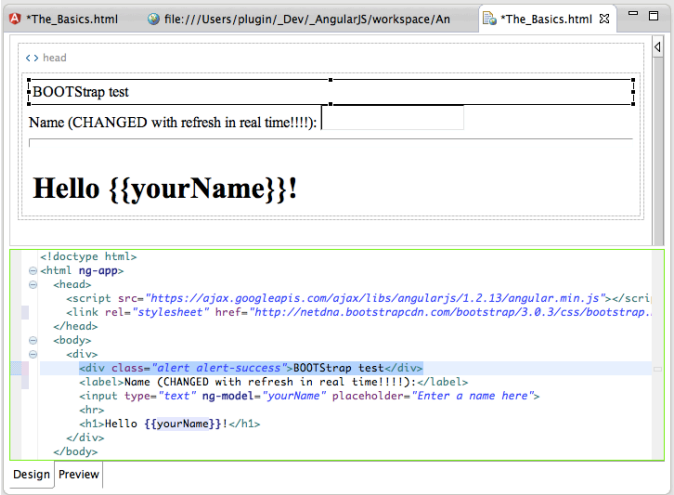
... which will look like this (note the non-processed HTML at the top and the HTML code at the bottom):



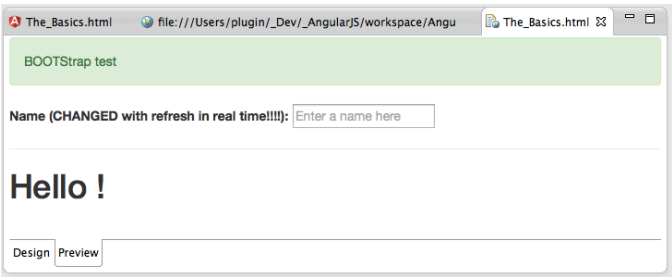
Opening up the ***Preview*** tab (from the default ***Design*** tab) will allow us to test the page currently being edited (note how Angular JS is working):

This means that (when in the ***Design*** tab) we can edit the AngularJS HTML page and see the changes immediately:

NOTE: This version of the **Web Page Editor **doesn't render the CSS while in that **Design** mode, which means that if we add bootstrap to this project:



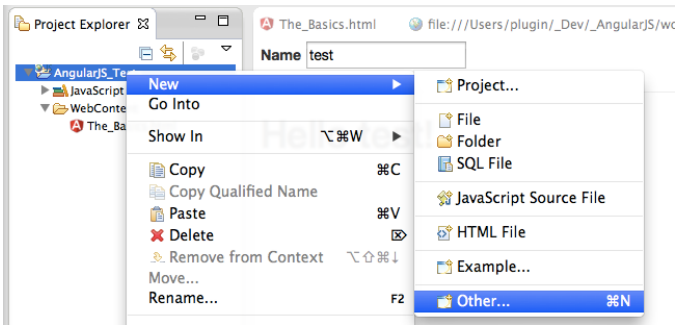
... the CSS will only be visible when in the ***Preview*** tab:



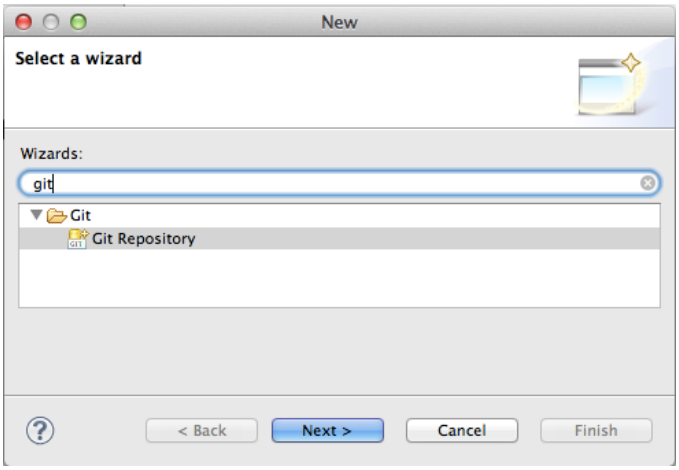
4) Creating a Git Repository for the files created

The best way for me to share these files is via a Git Repository, so the final step of this post is to create one on the files we have already created.

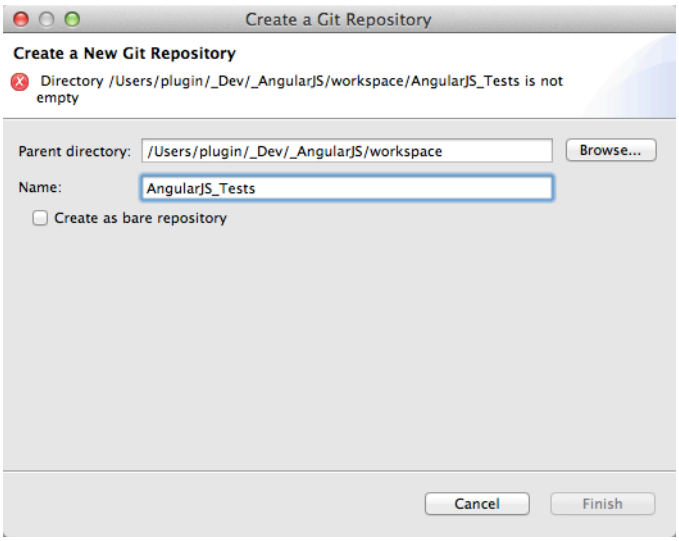
Since we are in eclipse I tried to create an Git Repo for the current project:



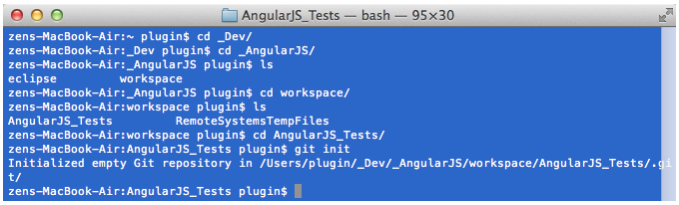
But the _Git Repository _wizard:



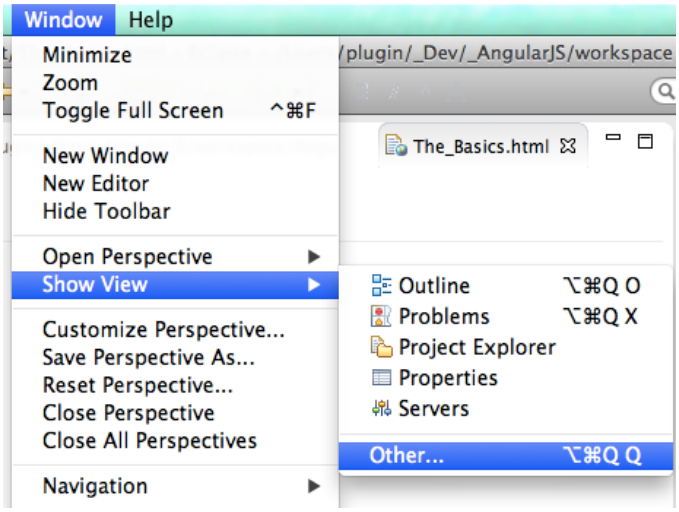
... didn't work, because it expects the target folder to not exist:



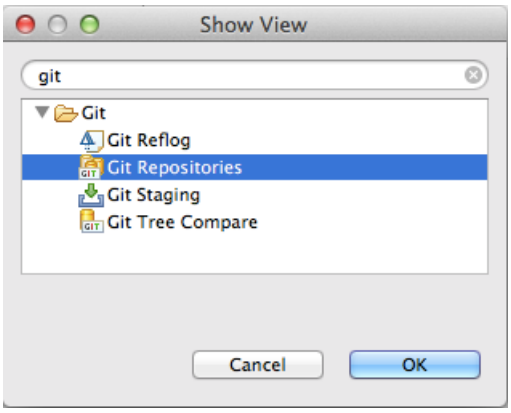
This was easily solved via the command line (by executing *$ git init* on the **_AngularJS_Tests** _folder)



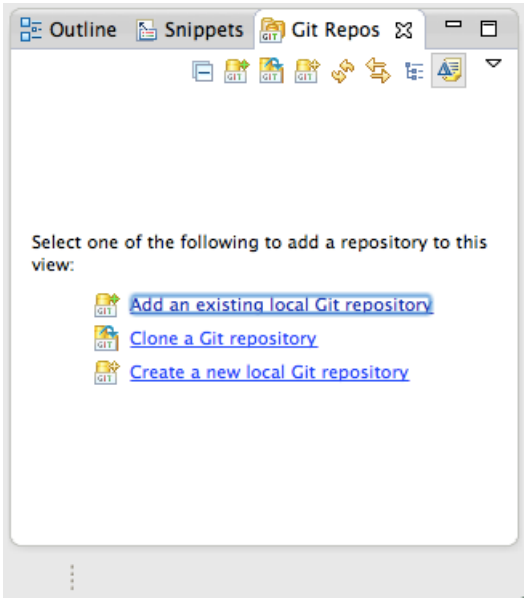
Now that we have a git repository, I was time to open it:



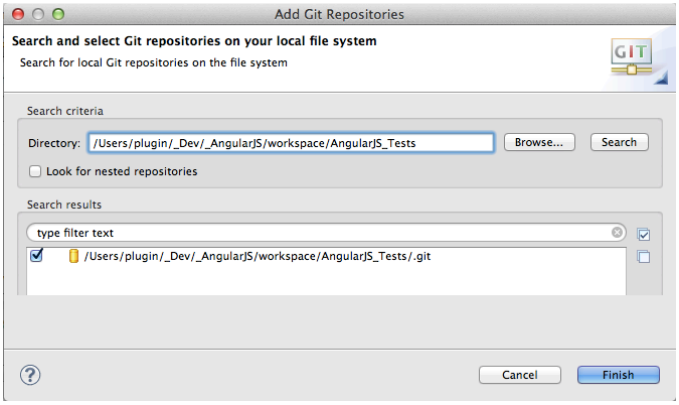
... using the_ **Git Repositories**_ view



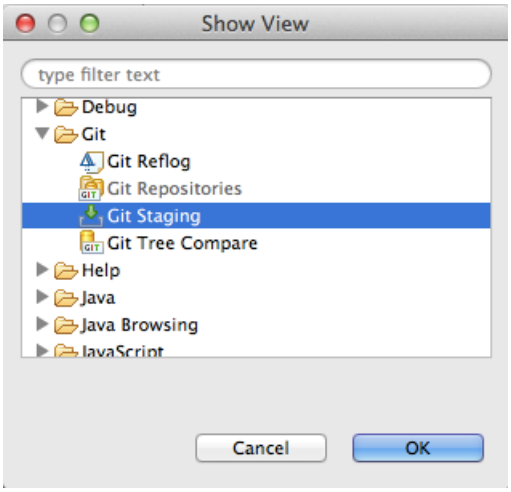
... where we can use the *Add an existing local Git repository* link:



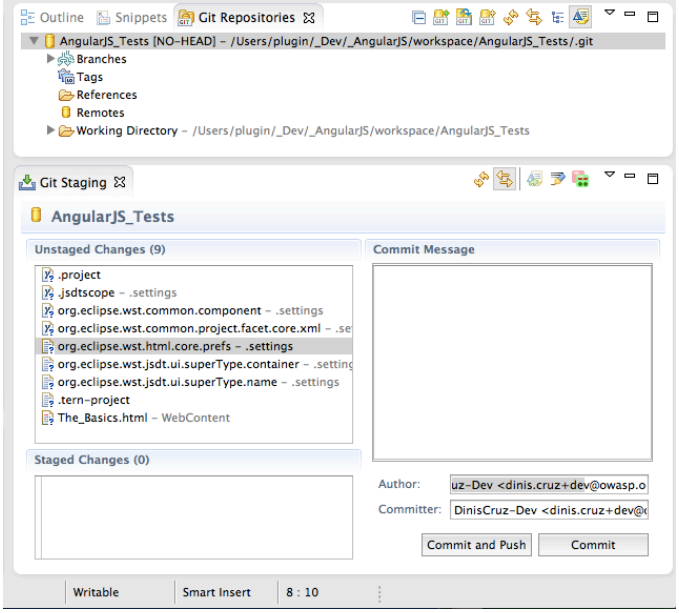
... to open the repository just created:



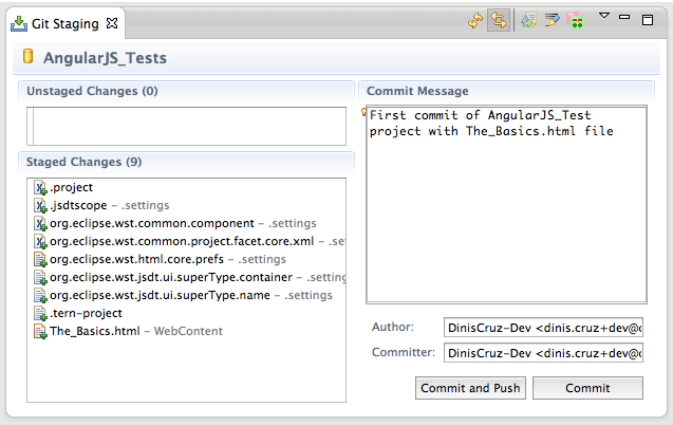
In order to create a git commit, I also opened the ***Git Staging*** view:



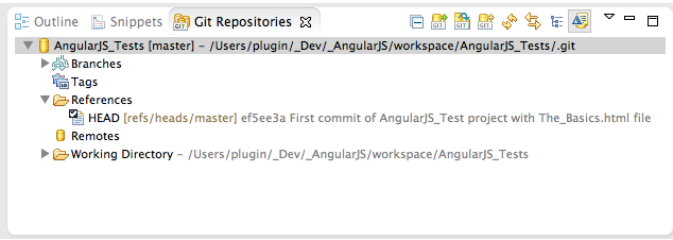
This is what these two git views look like (note that there are not commits/references and the list of new files in the **Unstaged Changes **list)



To commit the files drag-n-drop them from the ***Unstaged Changes*** to the *Staged Changes*, and write a commit message:



After clicking the ***Commit*** button the **_Git Repositories** _view will give a visual representation of the location current HEAD (which is the commit just done)

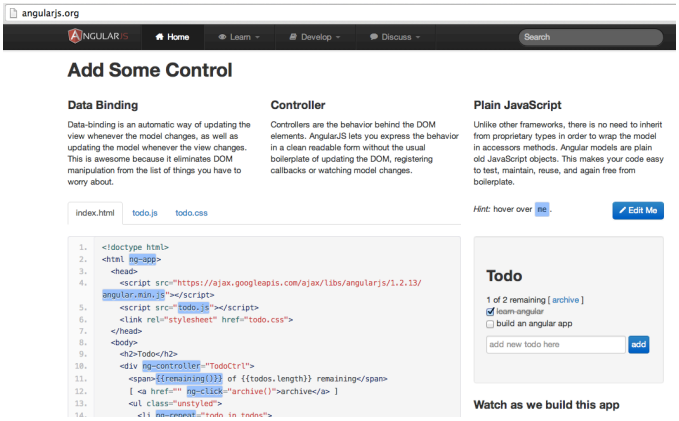


1.3 Using AngularJS in Eclipse, Part 2) Add Some Control

This is the second of four posts on how to run (inside Eclipse) the examples provided in AngularJS's home page:

- Using AngularJS in Eclipse, Part 1) The Basics
- Using AngularJS in Eclipse, Part 2) Add Some Control
- Using AngularJS in Eclipse, Part 3) Wire up a Backend
- Using AngularJS in Eclipse, Part 4) Create Components

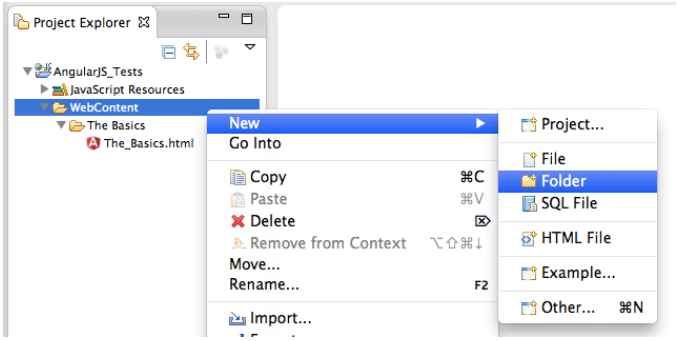
The example covered on this post is the *Add Some Control:*



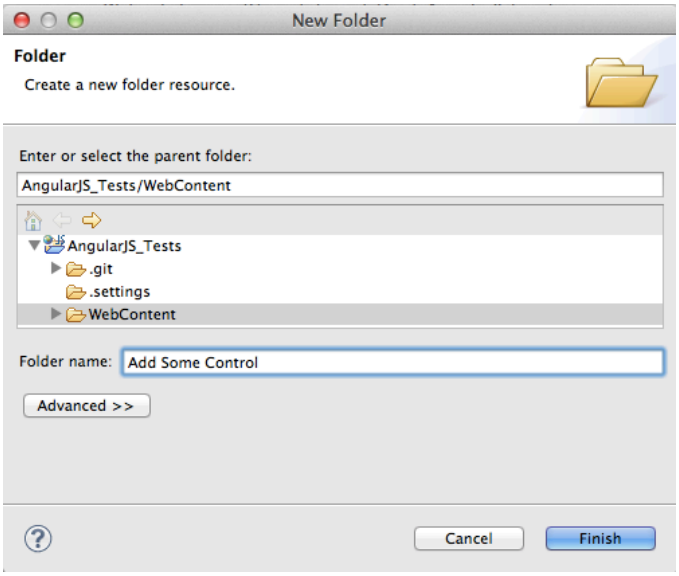
1) Creating the Html, CSS and JS Files

In order to keep the *AngularJS_Tests*** **repository better organised, lets put each sample in its own folder.

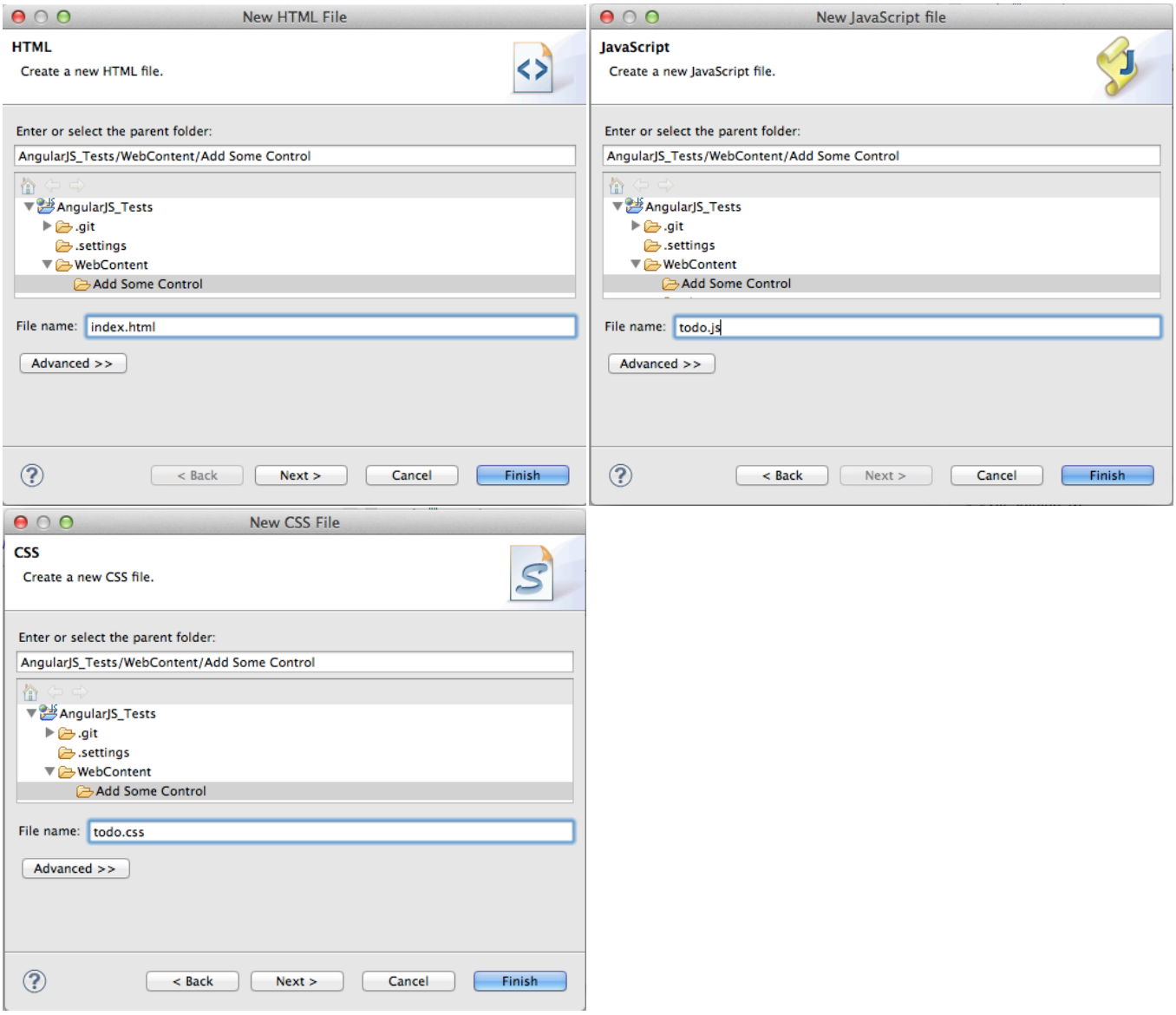
After moving the ***The_Basics.html*** file into its own folder, I created a new folder:



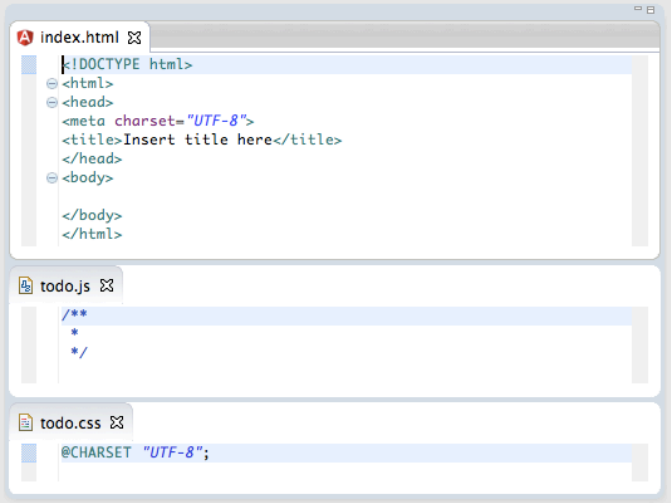
... called *Add Some Control* to the _WebContent _folder of the ***AngularJS_Tests*** project



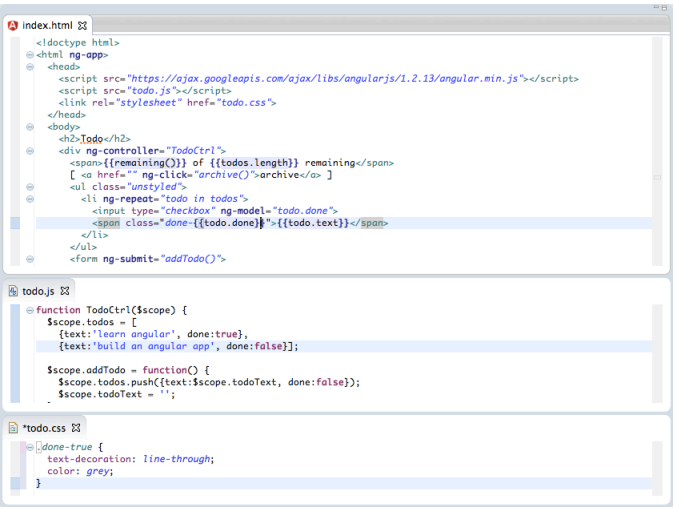
And inside it, I created the **index.html**, ***todo.js*** and ***todo.css*** files (each using the Eclipse default template for the respective file type)



Here is what these 3 files look with the default content:

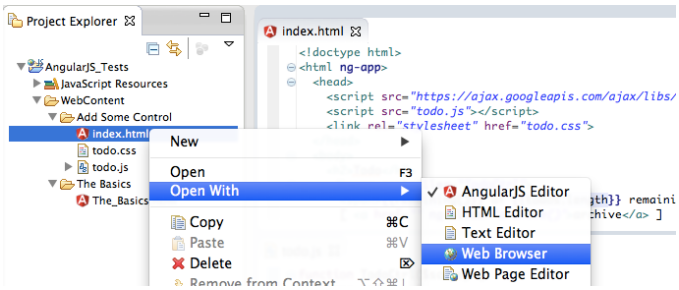


Here is what they look like with the content from the AngularJs.org _**Add Some Control** _sample:



2) Opening up in Web Browser

To see the example in action, right-click on the ***/Add Some Control/index.html*** file, and chose the ***Web Browser*** option form the ***Open With*** menu



... which will look like this:

The objective of this sample is to add new _Todos _to the list shown, using the TextBox provided and the ***add*** button

So in the example shown above, after entering *'This is a new ToDo' _on the TextBox and clicking _add* a new item will be added to the list (see below)

We can also remove **Todos** by using the checkboxes (see above) and clicking on the ***archive*** link (see below for the result)

2) Making some changes to the code to see AngularJS in $scope in action

To have a better understanding of what is going on, let's make some changes to the code sample.

Here is the original code

... which I'm going to modify by adding a new paragraph containing the value of the ***todoText*** variable (note that the _todoText _variable is used on the _input _HTML element, auto-wired to Angular by using the ***ng-model*** attribute).

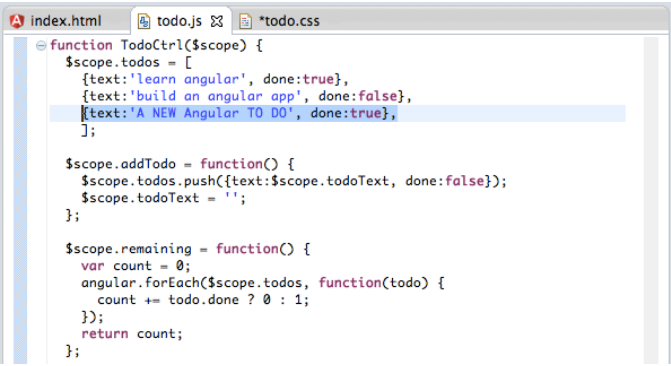
After refreshing the browser, we can see this in action by typing some text on the TextBox and seeing it show in real time after the '**New Todo text:' **text (one of the features of AngularJS is to keep all these variables in sync):

3) Changing default ****Todos

As another test, let's add a new default _todo _to the original version of the **$scope.todos **array (see below).

What is happening is that the_ **$scope.todos**_ variable is populated when the _TodoCtrl _is executed (part of the page build), which is then inserted into the webpage using the _<li ng-repeat="todo in todos"> _HTML code (see screenshot of index.html above).

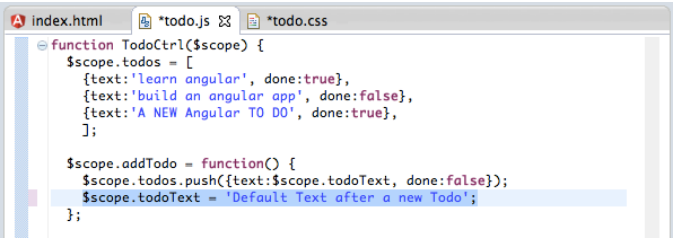
Here is the new ***$scope.todos*** array entry added:



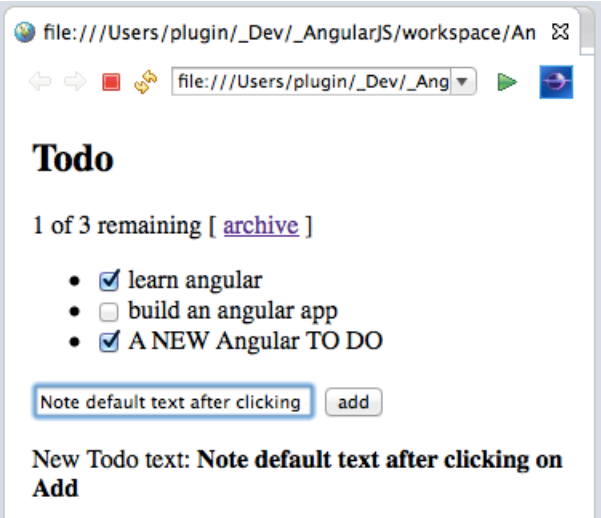
... which can be seen in action by refreshing the browser (note that it is already checked by default, since I set the ***done*** variable to ***true***)

4) Changing default value of TextBox (after adding new Todo)

To show that we can control the ***$scope*** variables from anywhere in the _TodoCtrl _controller, here is an example where I'm changing the **$scope.todoText **to have a specific value after the **_add_** button is clicked (which is wired to the $scope.addTodo using the ***<form ng-submit="addTodo()">*** HTML code)



After refreshing the web browser, we can see that if we add a new ***todo:***



… the TextBox (and paragraph below) will have the value _'Default value' _(vs the original behaviour of being empty)

4) Creating an 'impossible to archive' Todo

An interesting variation is to change the default value of the _$scope.todos _on the _**$scode.archive** _method, which, instead of being empty:

```
$scope.archive = function() {
  var oldTodos = $scope.todos;
  $scope.todos = [];
  angular.forEach(oldTodos, function(todo) {
    if (!todo.done) $scope.todos.push(todo);
  });
};
```

... it contains one entry:

```
$scope.archive = function() {
  var oldTodos = $scope.todos;
  $scope.todos = [{text:'WILL be added on Archive', done:true}];
  angular.forEach(oldTodos, function(todo) {
    if (!todo.done) $scope.todos.push(todo);
  });
};
```

... which means that (after refresh) and clicking on the ***archive*** link

... the _WILL be added on Archive _todo will now exist:

... and even if we try to archive all:

... the _WILL be added on Archive _todo will still be there:

5) Adding a Message label

As a final example, lets modify change the variable shown in the extra paragraph added to index.html to be _message _(which will be wired to ***$scope.message***):

index.html todo.js *todo.css

```
<!doctype html>
<html ng-app>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.13/ang
    <script src="todo.js"></script>
    <link rel="stylesheet" href="todo.css">
  </head>
  <body>
    <h2>Todo</h2>
    <div ng-controller="TodoCtrl">
      <span>{{remaining()}} of {{todos.length}} remaining</span>
      [ <a href="" ng-click="archive()">archive</a> ]
      <ul class="unstyled">
        <li ng-repeat="todo in todos">
          <input type="checkbox" ng-model="todo.done">
          <span class="done-{{todo.done}}">{{todo.text}}</span>
        </li>
      </ul>
      <form ng-submit="addTodo()">
        <input type="text" ng-model="todoText"  size="30"
               placeholder="add new todo here">
        <input class="btn-primary" type="submit" value="add">
        <p>
            Message: <b>{{message}}</b>
        </p>
      </form>
    </div>
  </body>
</html>
```

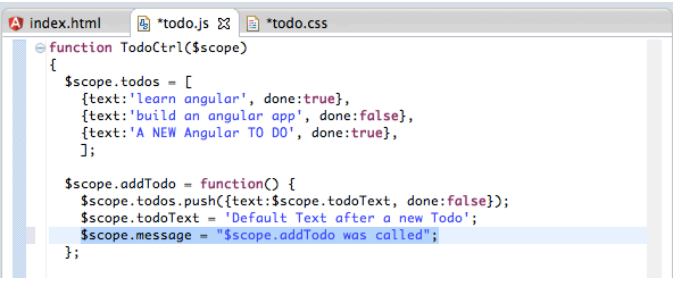
... and lets use that variable to provide visual feedback to the user when the *TodoCtrl* has executed:

index.html todo.js *todo.css

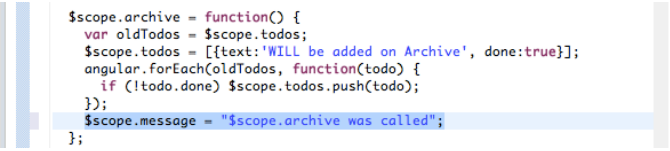
```
function TodoCtrl($scope)
{
  $scope.todos = [
    {text:'learn angular', done:true},
    {text:'build an angular app', done:false},
    {text:'A NEW Angular TO DO', done:true},
    ];

  $scope.addTodo = function() {
    $scope.todos.push({text:$scope.todoText, done:false});
    $scope.todoText = 'Default Text after a new Todo';
  };

  $scope.remaining = function() {
    var count = 0;
    angular.forEach($scope.todos, function(todo) {
      count += todo.done ? 0 : 1;
    });
    return count;
  };

  $scope.archive = function() {
    var oldTodos = $scope.todos;
    $scope.todos = [{text:'WILL be added on Archive', done:true}];
    angular.forEach(oldTodos, function(todo) {
      if (!todo.done) $scope.todos.push(todo);
    });
  };

  $scope.message = "AngularJS controler is all set";
}
```

Now, we should see the message _'AngularJS controller is all set' _just after the page finishes reloading:

To make it more relevant, lets add a _$scope.message _to the _$scope.addTodo _method:



... and the _$scope.archive _method:



Now we will get this message when the page reloads:

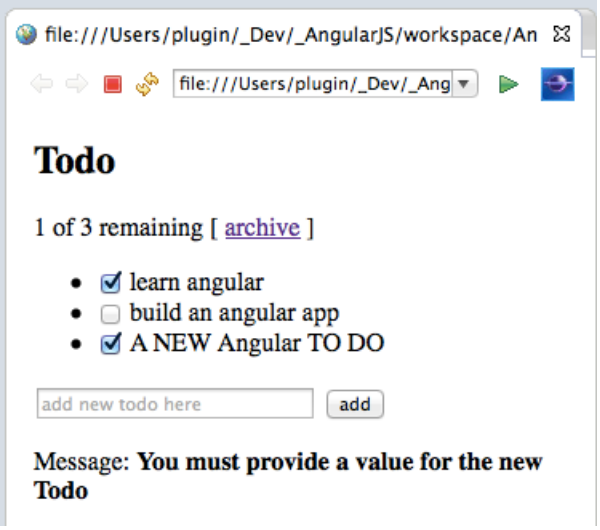
... this message after clicking on the ***add*** button

... this message after clicking on the ***archive link***

Finally let's refactor the *$scope.addTodo* method to so that it doesn't add a new ***Todo*** when no value is provided:

```
$scope.addTodo = function()
{
  if(!$scope.todoText)
  {
      $scope.message = "You must provide a value for the new Todo";

  }
  else
  {
      $scope.todos.push({text:$scope.todoText, done:false});
      $scope.message = "New Todo added:" + $scope.todoText;
      $scope.todoText = '';
  }
};
```

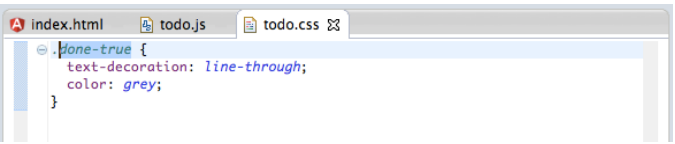
After reloading the page, we can confirm that trying to click on the ***add*** button without entering first any text on the TextBox will show this message:



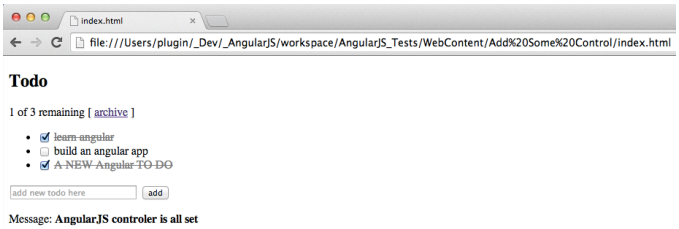
... and if we add a new one (in this case *'not an empty todo'*) we will see this message:

Appendix: CSS not being rendered in Eclipse's browser.

One thing I noticed was that the style changes provided by **todo.css** :



... was not being shown inside eclipse (as you can see on multiple screenshots above), but worked ok in Chrome:



... and stand-alone Safari (note that there is a ***line-through*** applied when an item it checked as done)

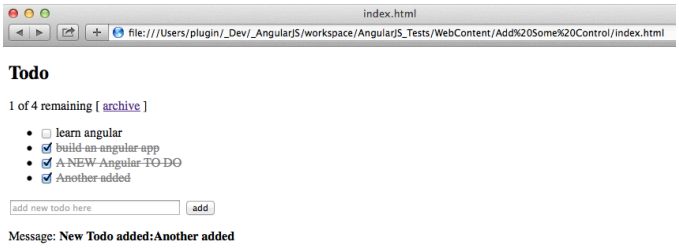


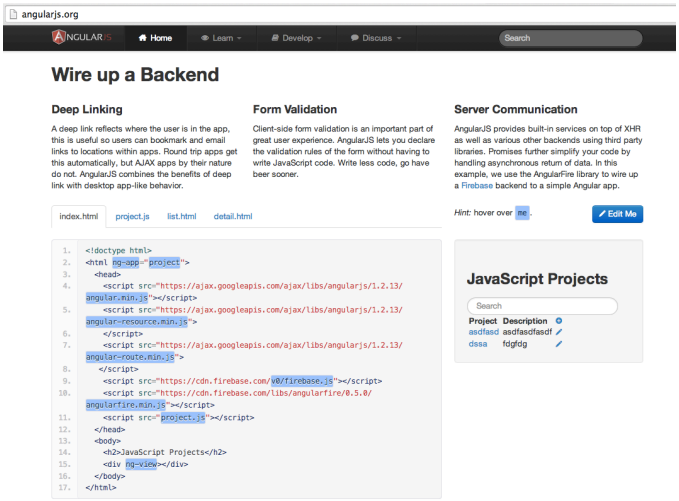
Table of Contents | Code

1.4 Using AngularJS in Eclipse, Part 3) Wire up a Backend

This is the third of four posts on how to run (inside Eclipse) the examples provided in AngularJS's home page:

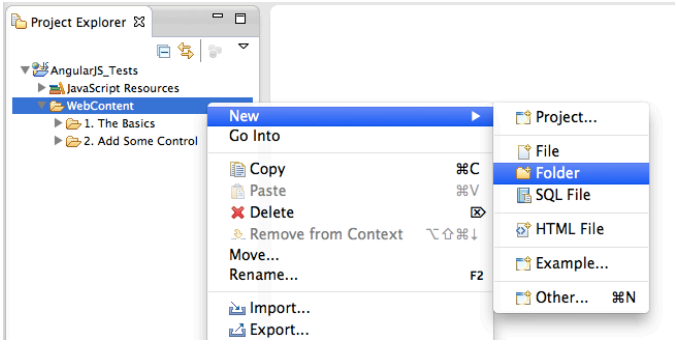
- Using AngularJS in Eclipse, Part 1) The Basics
- Using AngularJS in Eclipse, Part 2) Add Some Control
- Using AngularJS in Eclipse, Part 3) Wire up a Backend
- Using AngularJS in Eclipse, Part 4) Create Components

The example covered on this post is the ***Wire up a Backend***:

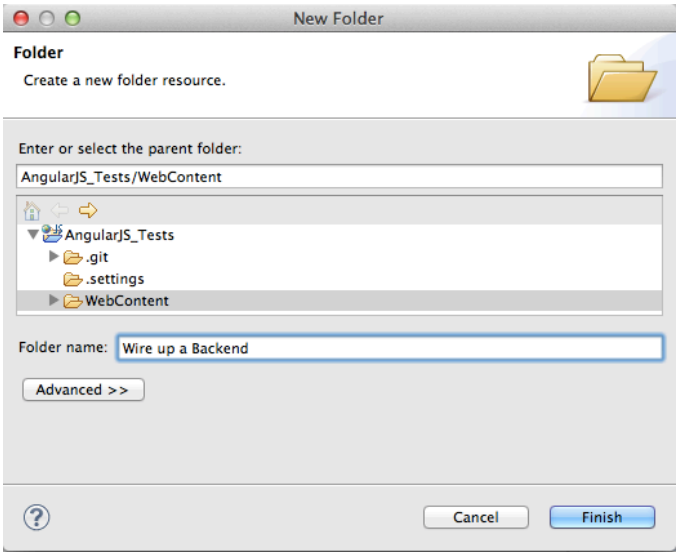


1) Creating the test files

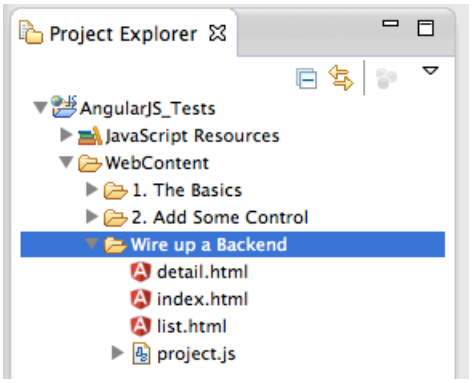
First step is to create a folder:



... called ***Wire a Backend***



... to hold the 4 files required for this example: ***index.html, detail.html, list.html and project.js***



Here is what they look like (with content from http://angularjs.org)

index.html :

project.js :

index.html *project.js list.html detail.html

```
angular.module('project', ['ngRoute', 'firebase'])

.value('fbURL', 'https://angularjs-projects.firebaseio.com/')

.factory('Projects', function($firebase, fbURL) {
  return $firebase(new Firebase(fbURL));
})

.config(function($routeProvider) {
  $routeProvider
    .when('/', {
      controller:'ListCtrl',
      templateUrl:'list.html'
    })
    .when('/edit/:projectId', {
      controller:'EditCtrl',
      templateUrl:'detail.html'
    })
    .when('/new', {
      controller:'CreateCtrl',
      templateUrl:'detail.html'
    })
    .otherwise({
      redirectTo:'/'
    });
})

.controller('ListCtrl', function($scope, Projects) {
  $scope.projects = Projects;
})

.controller('CreateCtrl', function($scope, $location, $timeout, Projects) {
  $scope.save = function() {
    Projects.$add($scope.project, function() {
      $timeout(function() { $location.path('/'); });
```

list.html :

index.html project.js list.html detail.html

```
<input type="text" ng-model="search" class="search-query" placeholder="Search">
<table>
  <thead>
  <tr>
    <th>Project</th>
    <th>Description</th>
    <th><a href="#/new"><i class="icon-plus-sign"></i></a></th>
  </tr>
  </thead>
  <tbody>
  <tr ng-repeat="project in projects | orderByPriority | filter:search | orderBy:'name'">
    <td><a href="{{project.site}}" target="_blank">{{project.name}}</a></td>
    <td>{{project.description}}</td>
    <td>
      <a href="#/edit/{{project.$id}}"><i class="icon-pencil"></i></a>
    </td>
  </tr>
  </tbody>
</table>
```

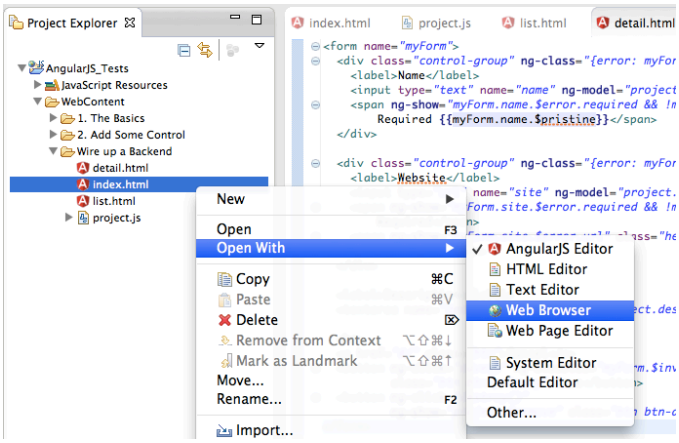
detail.html :

index.html project.js list.html detail.html

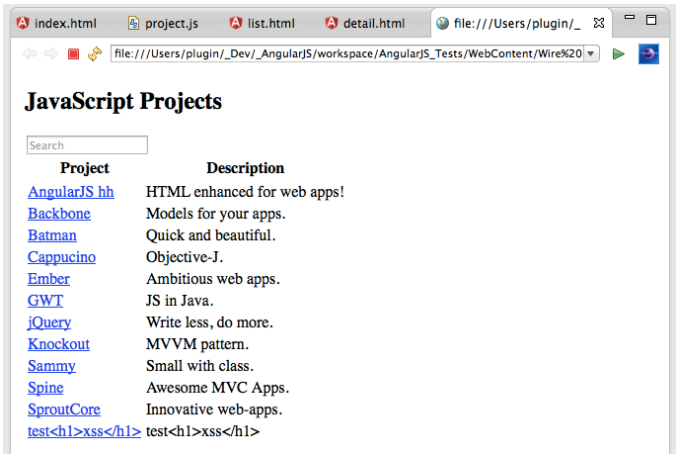
```
<form name="myForm">
  <div class="control-group" ng-class="{error: myForm.name.$invalid && !myForm.name.$pristine}
    <label>Name</label>
    <input type="text" name="name" ng-model="project.name" required>
    <span ng-show="myForm.name.$error.required && !myForm.name.$pristine" class="help-inline">
        Required {{myForm.name.$pristine}}</span>
  </div>

  <div class="control-group" ng-class="{error: myForm.site.$invalid && !myForm.site.$pristine}
    <label>Website</label>
    <input type="url" name="site" ng-model="project.site" required>
    <span ng-show="myForm.site.$error.required && !myForm.name.$pristine" class="help-inline">
        Required</span>
    <span ng-show="myForm.site.$error.url" class="help-inline">
        Not a URL</span>
  </div>

  <label>Description</label>
  <textarea name="description" ng-model="project.description"></textarea>

  <br>
  <a href="#/" class="btn">Cancel</a>
  <button ng-click="save()" ng-disabled="myForm.$invalid"
          class="btn btn-primary">Save</button>
  <button ng-click="destroy()"
          ng-show="project.$remove" class="btn btn-danger">Delete</button>
</form>
```

2) Running in Browser and noticing CSS problems

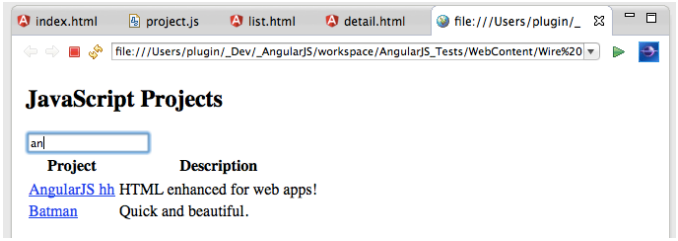
To see this example in action, lets open it in Eclipse's WebBrowser:



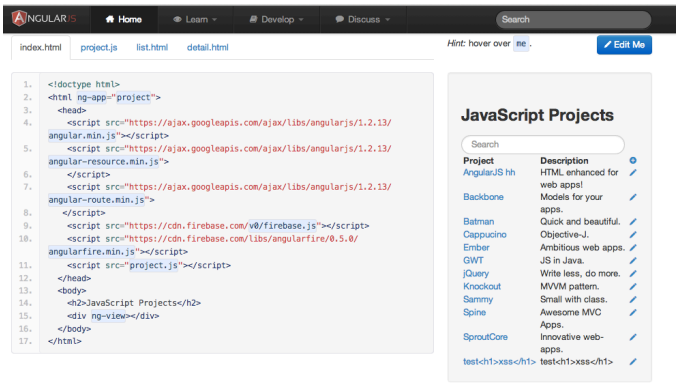
... which looks like this:



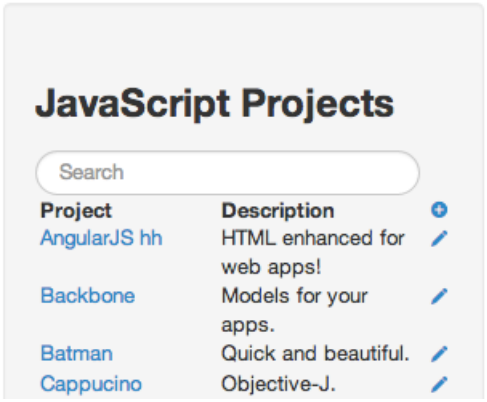
... with a search bar at the top that can be used to filter the loaded data:



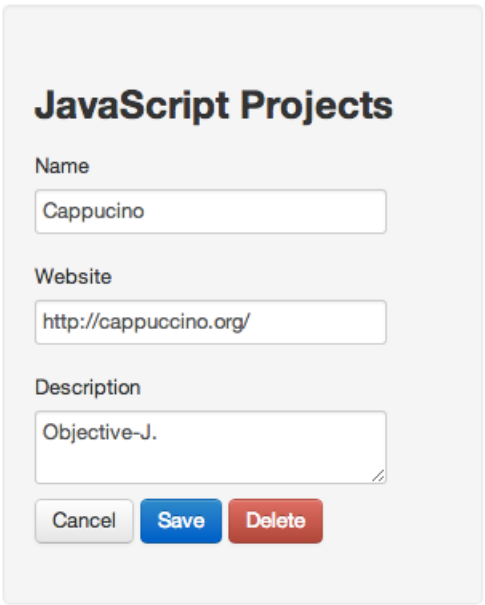
Only problem is that it doesn't look at all like it does on the http://angularjs.org page (see below)



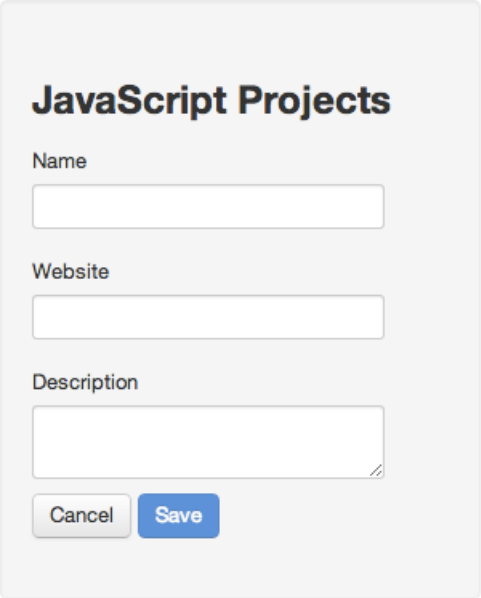
Note how not only the css styles are different, the ***add*** and ***edit*** links (i.e. icons) are missing:



Here is what the ***edit*** page should look like:

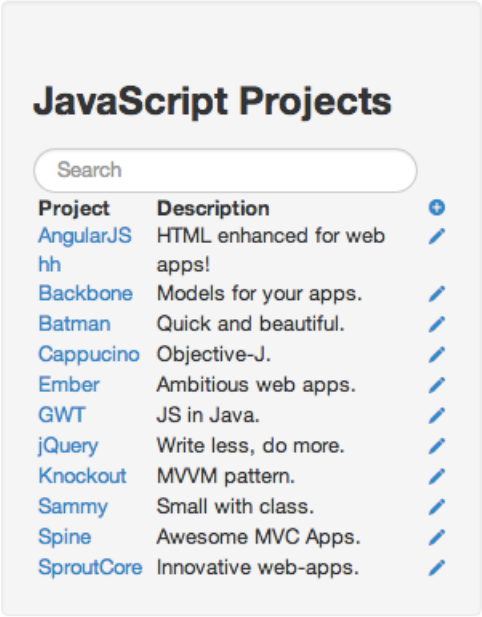


Here is what the _new _page should look like:

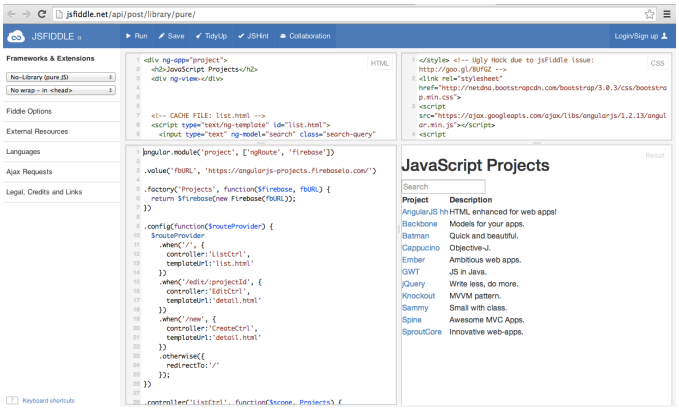


3) Running example on JSFiddle

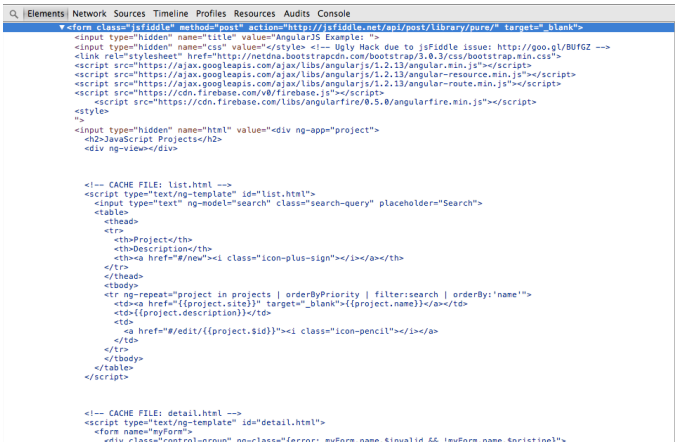
On the http://angularjs.org page there is a button called *Edit Me* (top right) which when clicked will do a cross-site POST submission to http://jsfiddle.net/api/post/library/pure/



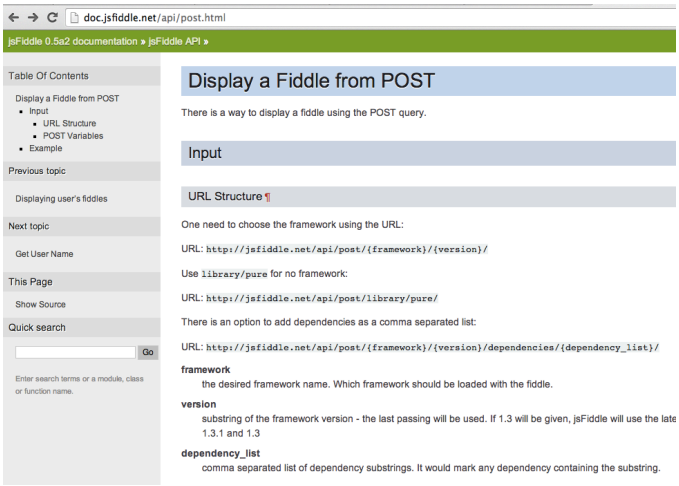
... which looks like this (notice the extra Bootstrap css included that is not present on the code sample provided in the http://angularjs.org page)



For reference here is what the *Edit Me* form looks like in the http://angularjs.org page (with the values passed as hidden parameters to ***jsfiddle***)



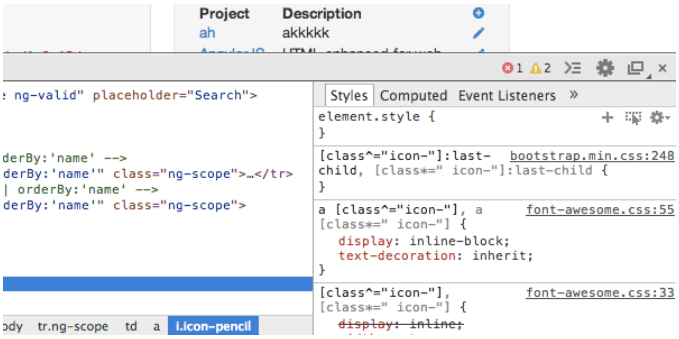
And here is the jsfiddle API description for the method used (http://doc.jsfiddle.net/api/post.html)



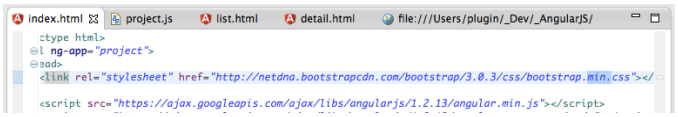
4) Fixing CSS issue

Since I wanted to match the CSS of my local test page to the style used in the example embedded in the http://angularjs.org page, the best place to look is on the source code code of that page :)

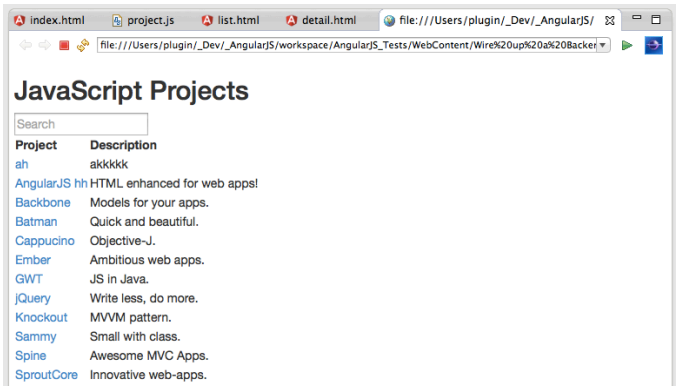
Using Chrome Browser Inspector, on the _Styles _tab, I was able to see that they were using two css libraries: Bootstrap and Font Awesome:



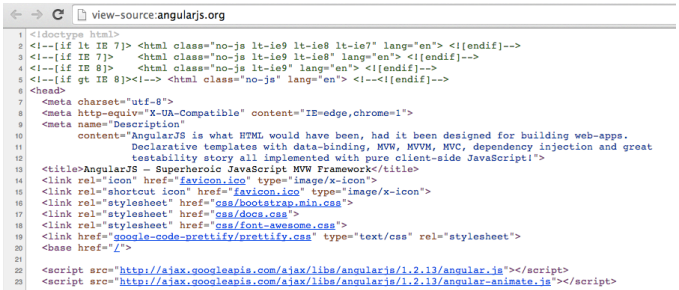
Back in Eclipse, I added the bootstrap css reference



And on refresh, the page looked better (but the ***edit*** and ***add*** icons/links where still missing)

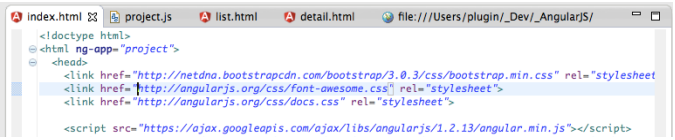


So I went back to the angularjs.org source code to see what they were using:

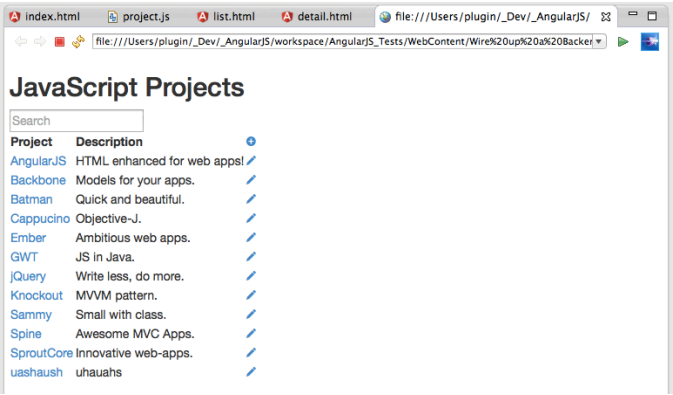


First I tried adding the latest Font Awesome from their CDN:

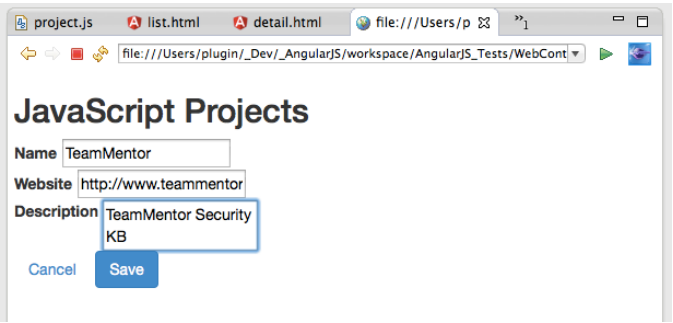
But since that didn't work, I decided to add both **font-awesome.css **and** docs.css **references:



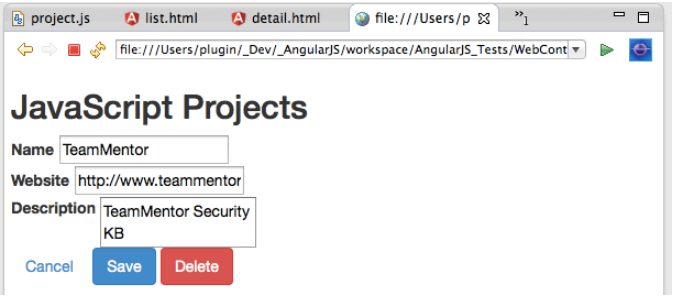
... with much better results (note the icons on the right)



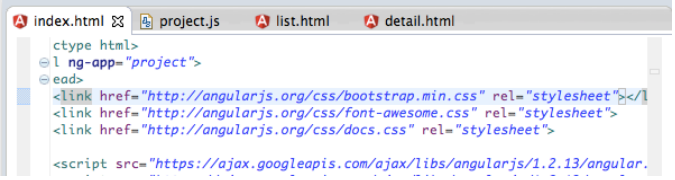
But there was still something wrong, since the ***new*** page looked like this:



... and the ***edit*** page looked like this:

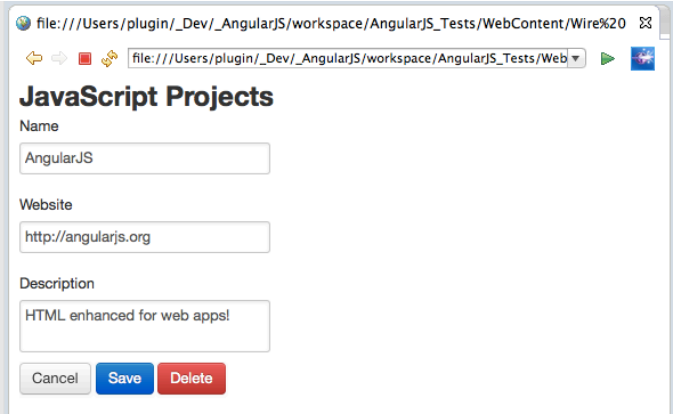


Since it was all good on the angular.org site, my next try was to use the ***bootstrap.min.css*** file from it (vs the CDN)



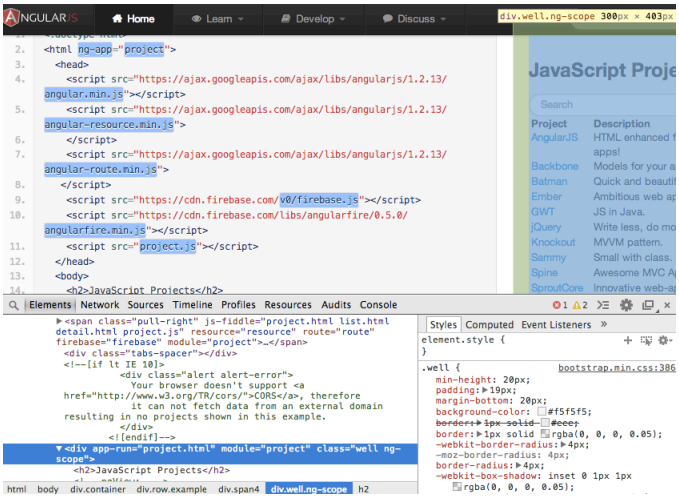
With the local page:

... and the ***edit*** page now looking just like the real thing (note the Search TextBox (above) and the fields (below))

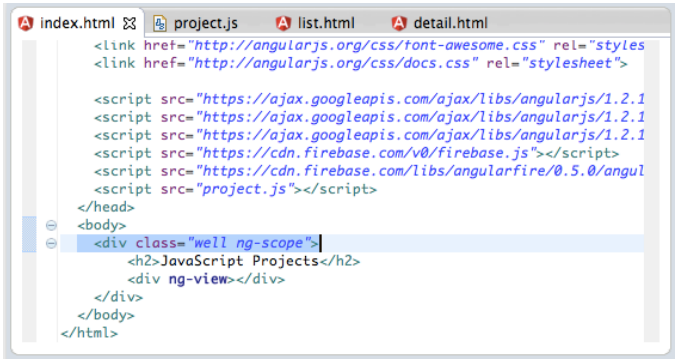


That said, the layout where still a little bit different (namely the background of the local file which was white).

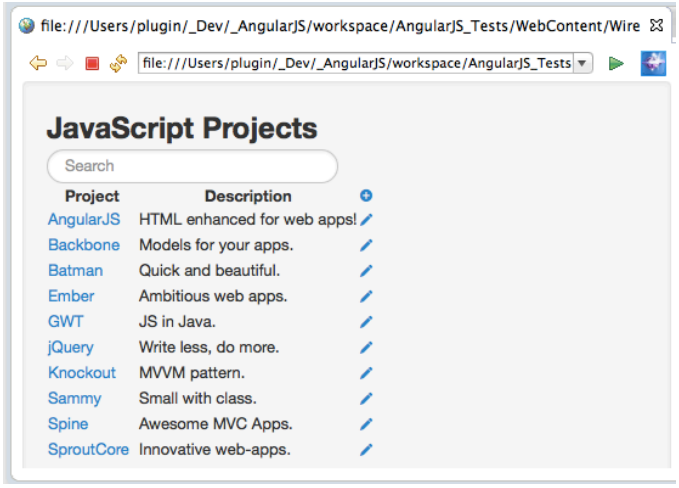
So I had a look at the style of the example in angular.org and noticed that there was a _div _element with the bootstrap classes **well **and** ng-scope**:



Back in the local file, I edited the index.html to also have a similar div:



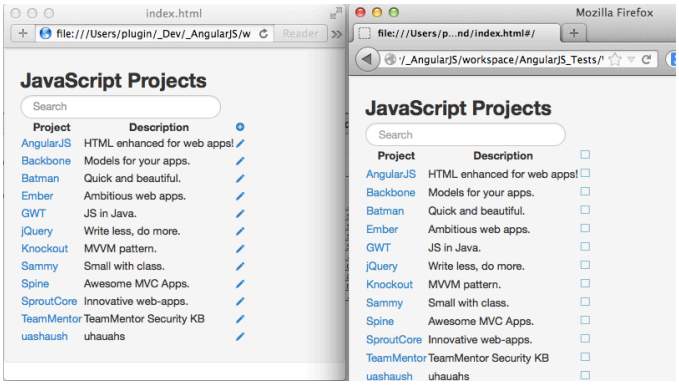
And finally the local form looked just like the original:



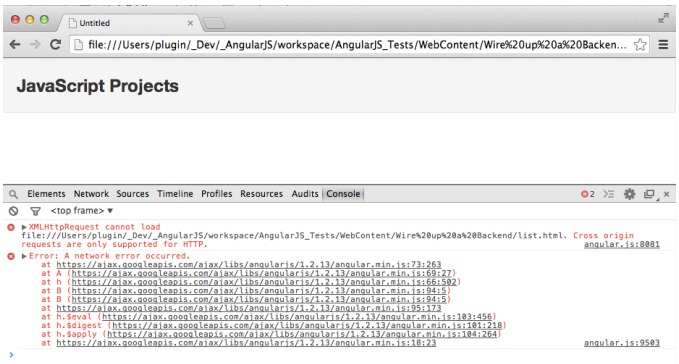
**5) Running local sample on multiple browsers **

At this stage I was curious on what this form would look like on multiple browsers, and found out something interesting.

Here is the page on Safari (left) and Firefox (right), which are loaded ok:

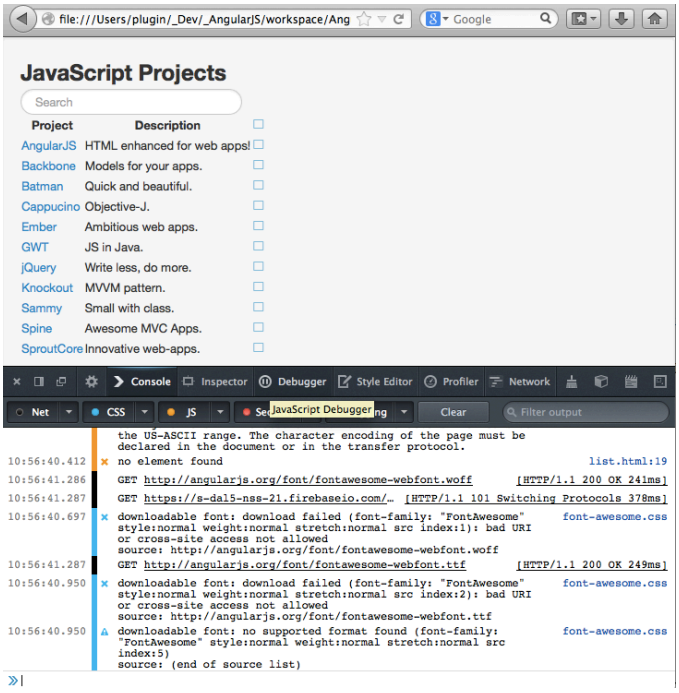


But on Chrome, it doesn't work at all, and the culprit is security!

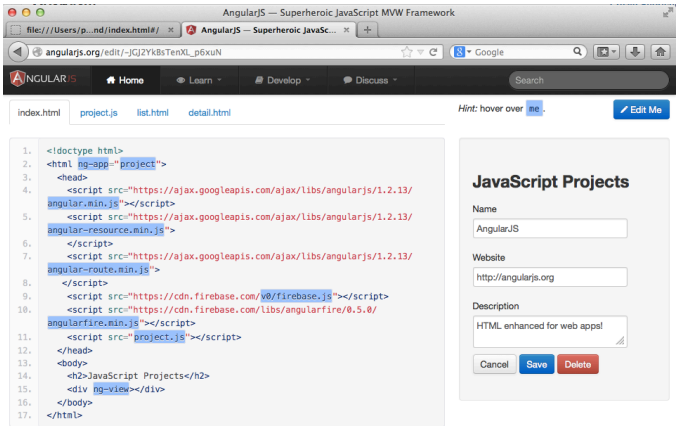


Basically chrome does not allow Cross origin requests from html files loaded from the local disk (which does make sense from a security point of view).

As a final little comment, in case you noticed that on the Firefox screenshot the Angular content was correctly displayed but the icons didn't show up, that much be a similar problem with loading ***Bootstrap*** or ***Font Awesome*** from the local disk



Here is the same example running on Firefox browser, so it works when loaded from http:



Firebase

This example uses the Firebase which is a very powerful API to store and sync data in realtime.

After writing this post, I spend some time researching how it works and can be used.

Here are some posts I've written about Firebase:

- Using Firebase to sync data with a webpage (via Javascript, REST and Firebase Admin panel)
- XSS considerations when developing with Firebase
- Trying out Firebase (Beta) hosting solution and good example of Firebase Security rules
- First PoC of sending TeamMentor's server-side request URLS to Firebase (and seeing it in realtime in an AngularJS page)

- A really SIMPLE and clean AngularJS+Firebase example
- (if you are reading this after Feb 2014) checkout the Firebase section of this blog for newer posts

---

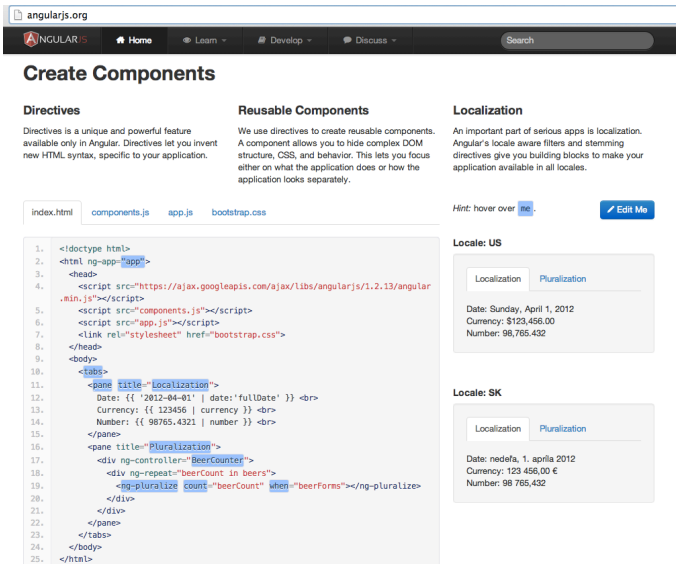
Table of Contents | Code

1.5 Using AngularJS in Eclipse, Part 4) Create Components

This is the last of four posts on how to run (inside Eclipse) the examples provided in AngularJS's home page:

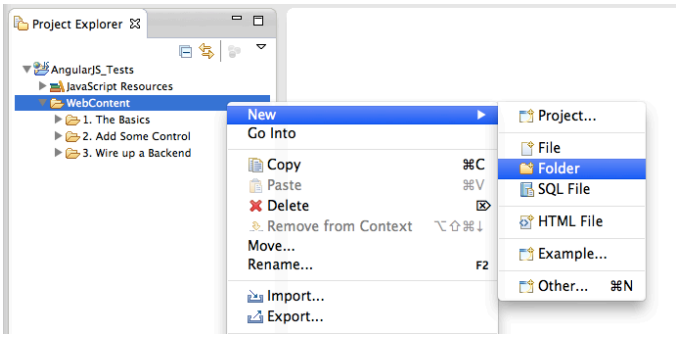
- Using AngularJS in Eclipse, Part 1) The Basics
- Using AngularJS in Eclipse, Part 2) Add Some Control
- Using AngularJS in Eclipse, Part 3) Wire up a Backend
- Using AngularJS in Eclipse, Part 4) Create Components

The example covered on this post is the ***Create Components***:

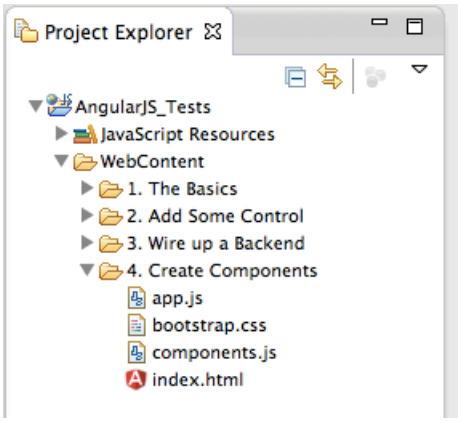


1) Creating the test files

First step is to create a folder called *4. Create Components*:

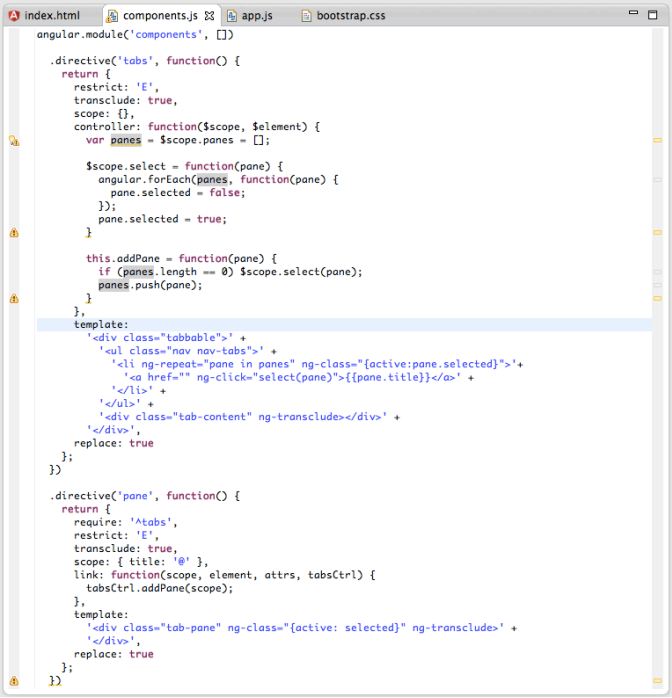


... with the 4 files provided by this example:

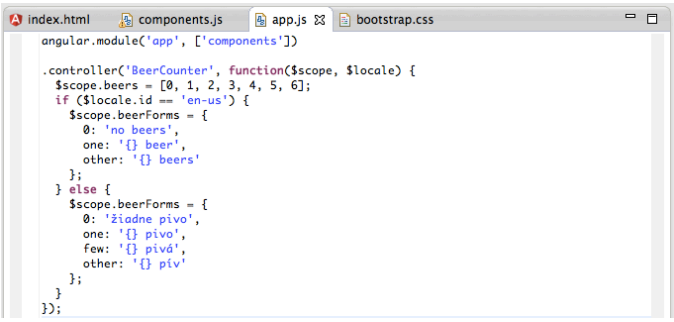


index.html

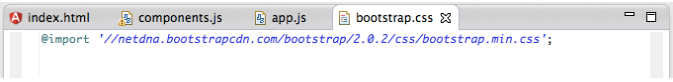
components.js



app.js

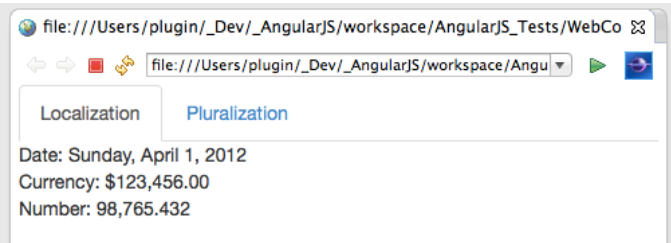


and **bootstrap.css**



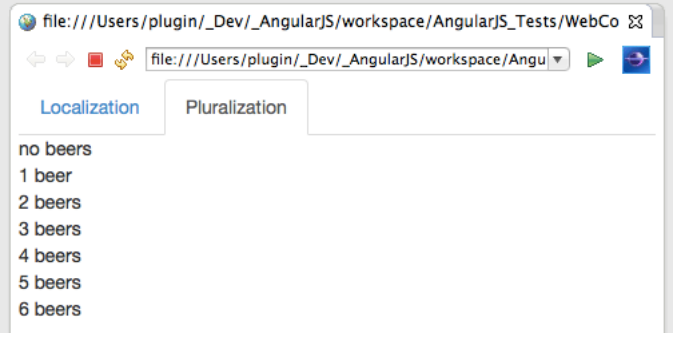
2) Running in browser (and failing to change locale)

Here is what it looks like when loaded locally (in Eclipse):



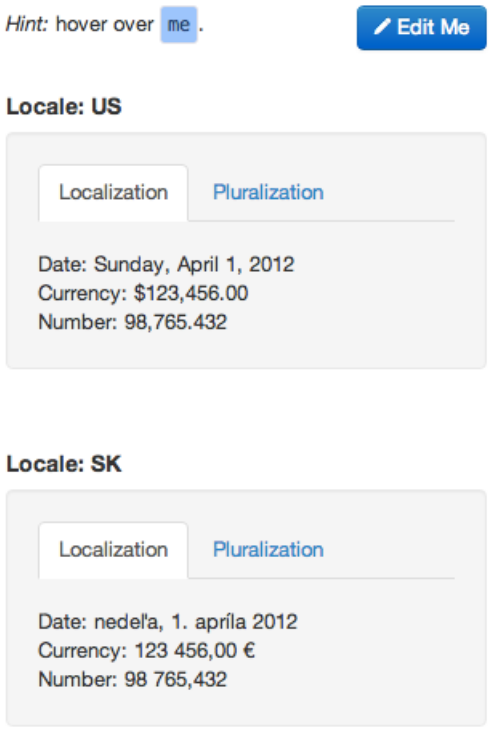
This example shows how we can create components (i.e directives) that are able to handle multiple languages or currencies.

The Localization tab (shown above) is currently set for english (USA) and the Pluralization tab (shown below) is currently set to English.



This demo also has support for SK, but there didn't seem to be an easy way to change the browser's locale (so that we can see the SK values in action)

But on the angular.org page they were able to show both US and SK side by side:



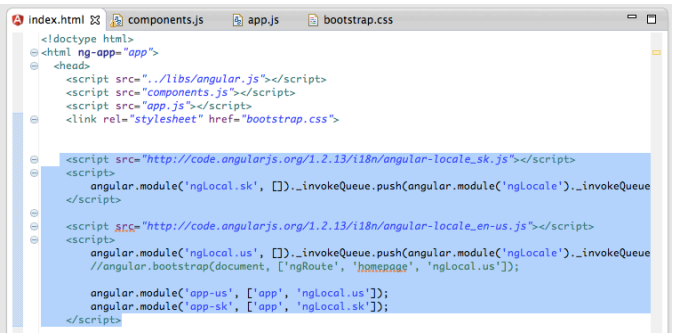
so there had to be a way to do it :)

3) programatically change the locale

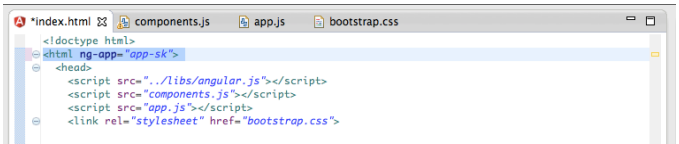
Looking at the code from the angular.org the code changes made below seem to be the ones required.

Basically we need to:

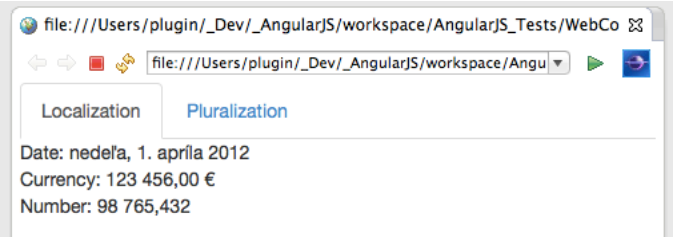
- load up the ***angular-locale_sk.js*** *_file followed by immediately creating a module called* ***_ngLocal.sk***
- load up the ***angular-locale_en-us.js*** *_file followed by immediately creating a module called* ***_ngLocal.us***
- create a module called ***app-us*** that depends/consumes ***ngLocal.us***
- create a module called ***app-sk*** that depends/consumes ***ngLocal.sk***



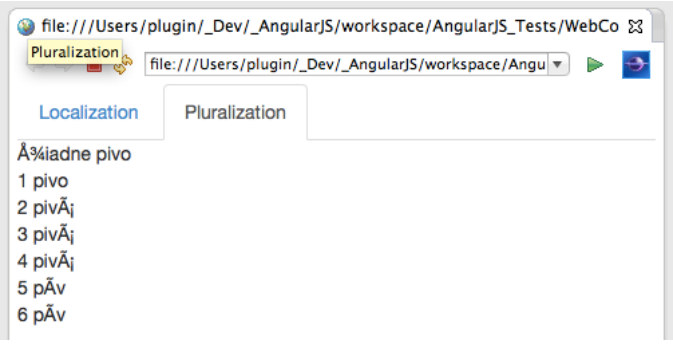
Then we can change the ***ng-app*** to be ***app-sk:***



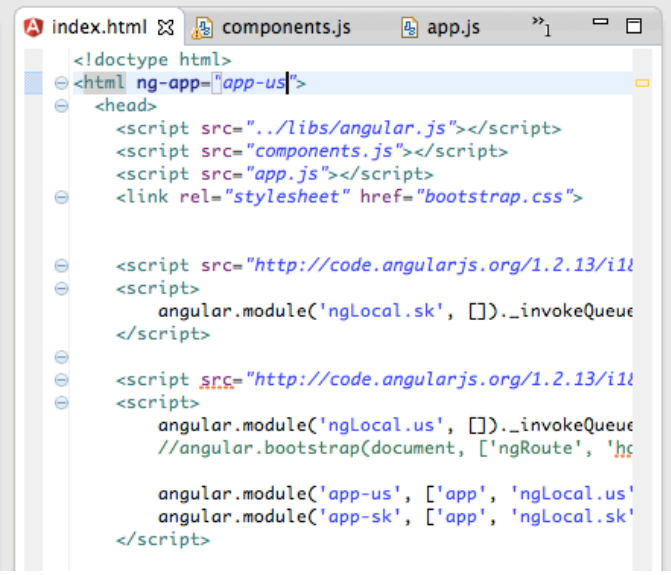
... and now the locale is SK (i.e. Slovak):



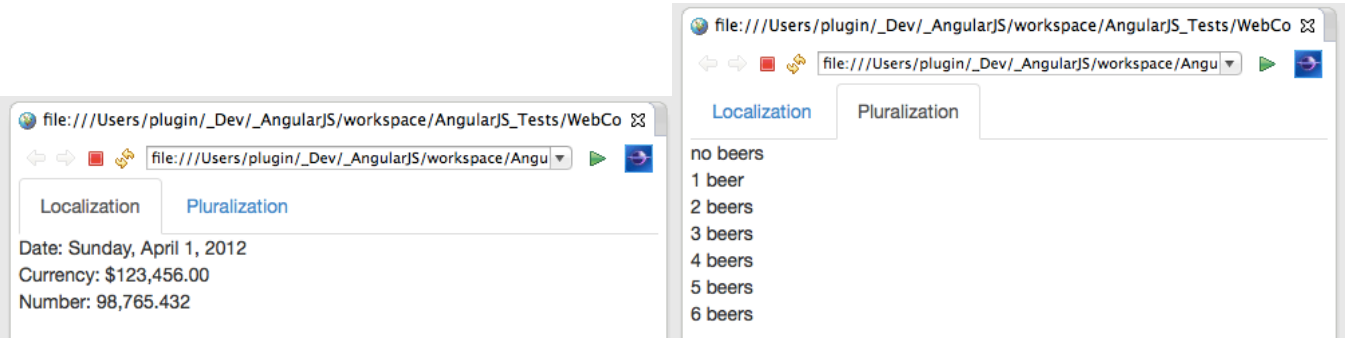
... with the text showing in Slovak (well I hope it does, since I don't speak Slovakian :))



If we wanted to have the site back in english (US), we can change the ***ng-app*** to be ***app-us***



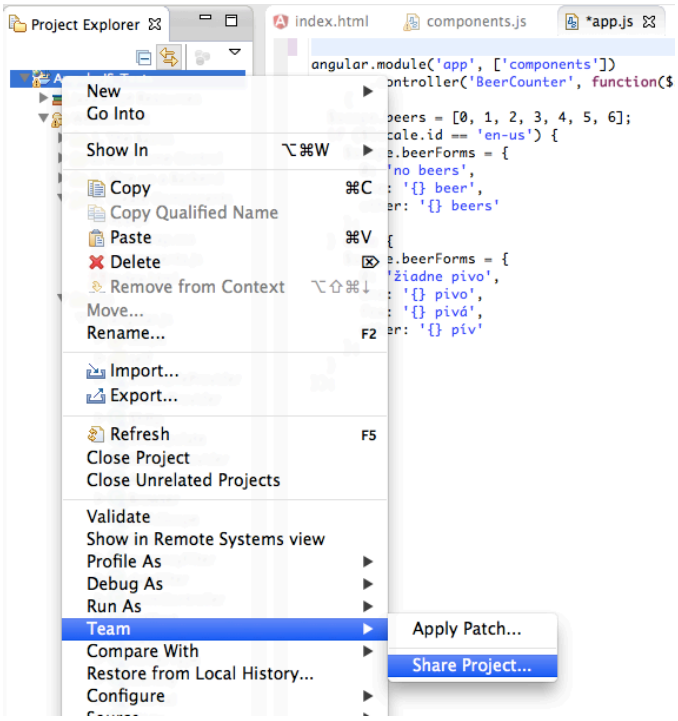
… and we're back in USA currency and English language:



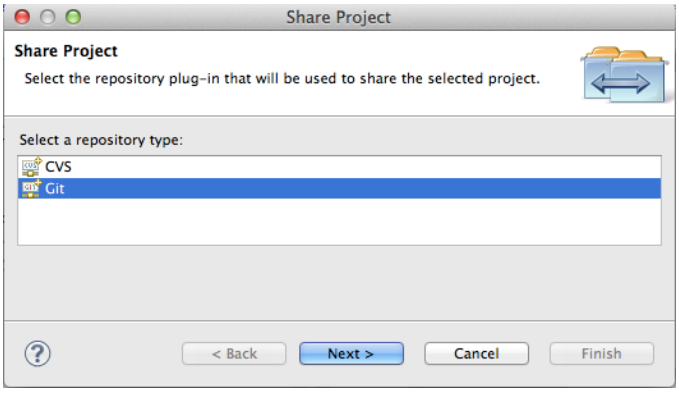
4) viewing Git Diffs in Eclipse

At this stage I wanted to see **git diffs **inside eclipse (for example '*current changes vs last committed code*'), but the option I was looking for ('*Compare With -> HEAD Revision*'**) was not there.

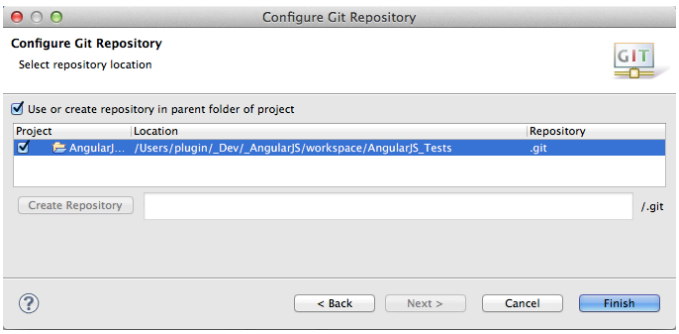
The problem was that I needed to _'Share current project', _which can be done via the ***Team*** menu:



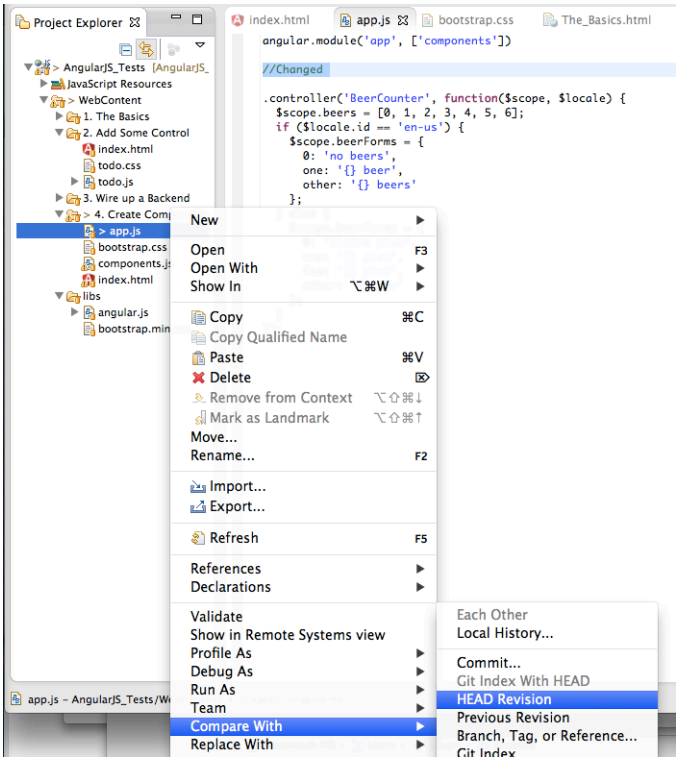
After choosing the **Share Project** ... ****menu option**, **I was asked to chose the Git repository type (which is a bit weird since this was already a Git repository)



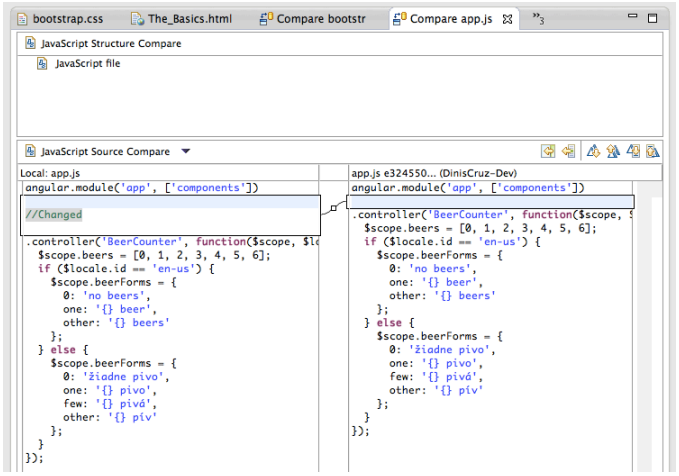
... and chose to use the repository in the parent folder of the selected project:



Once that completed, I was able to see the ***Compare With -> HEAD Revision*** option:



... which provides this really nice ***Git diff*** UI:



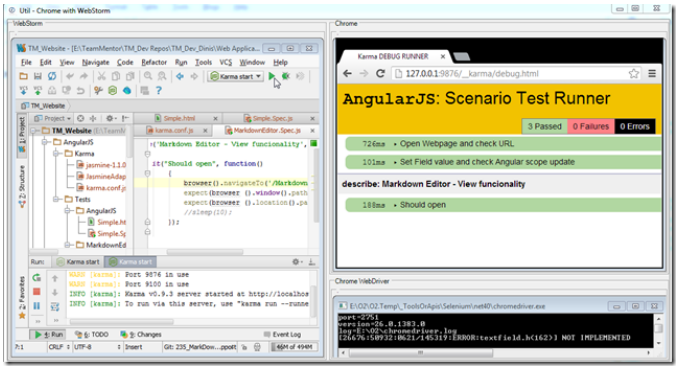
1.6 AngularJS code editor using UI-Bootstrap and CodeMirror (done without using jQuery)

I'm adding a number of AngularJS views to TeamMentor, and here is a simple HTML based source code editor inspired on the How to Integrate Codemirror With Angular UI post and ui-codemirror repository.

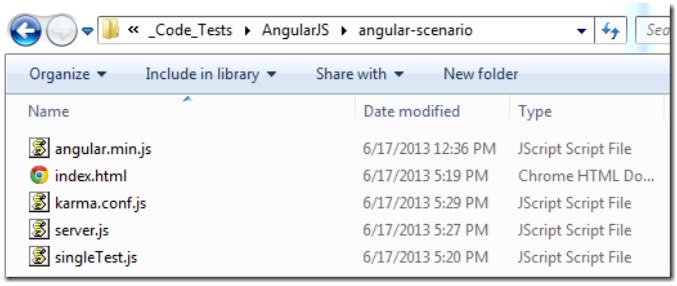
In the end, these are the APIs I used:

- http://angularjs.org/
- http://angular-ui.github.io/bootstrap/
- http://twitter.github.io/bootstrap
- http://codemirror.net/ (just the core bit)

And this is what it looks like:



The source code editor is showing the contents of the current page (dynamically fetched using Angular ***$http.get***) and the bottom yellow div is showing (in real-time) the contents of the source code editor:



What is nice about this example is that I didn't use jQuery at all!

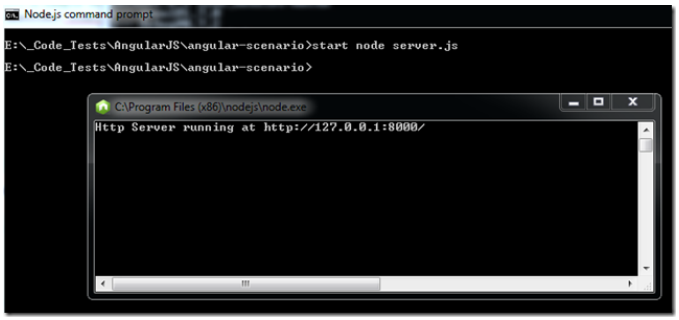
The great posts http://stackoverflow.com/a/15012542 and AngularJS for jQuery Developers explain why learning to code in Angular without JQuery is so important.

Basically, it's better not have jQuery available, since them, the only way to solve a problem, is the 'AngularJS way' :)

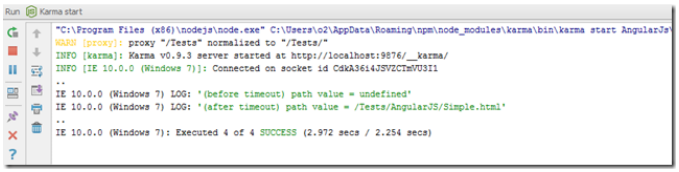
How it works:

Here is a brief explanation of the code behind this PoC (included in full at the end of this page):

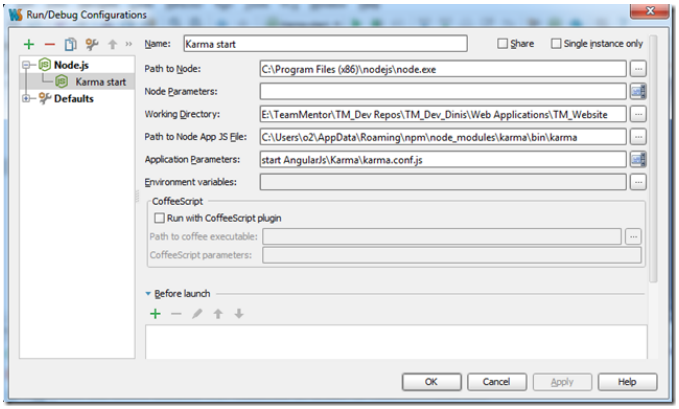
First we load the **Javascript** and CSS:



... them set up ***AngularJS*** module and ***codeMirror*** value

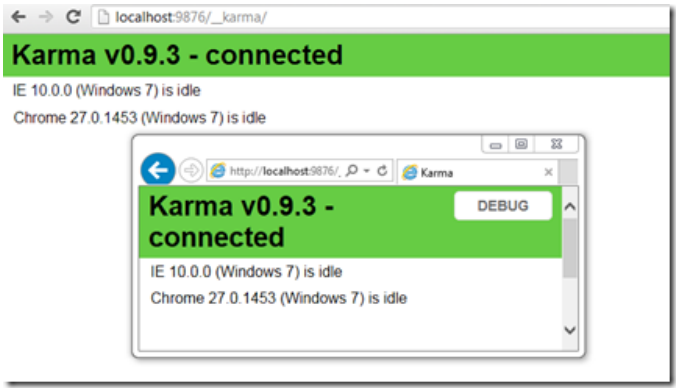


... use a ***controller*** to get the code to show (using ***$http.get***) and assign it to the the_ **$scope.code**_ variable



... configure **angularJS** in the HTML by setting the ***textarea*** element to be a ***codemirror*** (linked to the **$scope.code **model)

... finally show the current value of **$scope.code** in side an bootstrap ***alert*** element



Complete Source code:

```
<!DOCTYPE html>

<html>
    <head>
        <title>CodeMirror with AngularJS</title>
        <link href="/Content/bootstrap.min.css" rel="stylesheet">
        <script src="/Scripts/angular.min.js" type="text/javascript"></script>
        <script src="/Scripts/angular-ui.js" type="text/javascript"></script>
        <script src="/Scripts/ui-bootstrap-tpls-0.3.0.min.js" type="text/javascript"></scr\
ipt>

        <link href="/Content/codemirror-3.01/codemirror.css" rel="stylesheet"/>
        <link href="/Content/codemirror-3.01/theme/rubyblue.css" rel="stylesheet"/>
        <script src="/Scripts/codemirror-3.01/codemirror.js" type="text/javascript"></scri\
pt>
        <script src="/Scripts/codemirror-3.01/mode/javascript.js" type="text/javascript"><\
/script>


        <script type="text/javascript">
            var myApp = angular.module('myApp', ['ui', 'ui.bootstrap']);

            myApp.value('ui.config',
                {
                    codemirror:
                        {
                            mode: 'javascript',
                            lineNumbers: true,
                            matchBrackets: true,
                            theme: 'rubyblue'
                        }
                });
```

```
33
34              function codeCtrl($scope,$http)
35              {
36                  $scope.docLocation = document.location.href;
37                  $http.get($scope.docLocation)
38                       .success(function (data)
39                          {
40                              $scope.code = data;
41                          });
42                  //$scope.code = "var a = 'somecode'; \n//which also shows above</h1>";
43              }
44          </script>
45      </head>
46      <body ng-app="myApp">
47          <div class="well well-large">
48              <div class="container">
49                  <h2>CodeMirror working with AngularJS and Bootstrap</h2></div>
50              </div>
51              <div ng-controller="codeCtrl">
52                  <div class="container">
53
54                  <h4>Code Editor:</h4>
55                  <p>With the the contents of this page (i.e.: {{docLocation}} )</p>
56
57                  <textarea ui-codemirror ng-model="code"></textarea>
58
59                  <br/><hr/><br/>
60
61                  <h4>Bootstrap alert style</h4>
62                  <p>
63                      Showing in real time the contents of the code shown above (make a chan\
64  ge to try it)
65                  </p>
66                  <alert type="success">{{code}}</alert>
67              </div>
68          </div>
69      </body>
70  </html>
```

Table of Contents | Code

2 KarmaJS

This section has the following chapters:

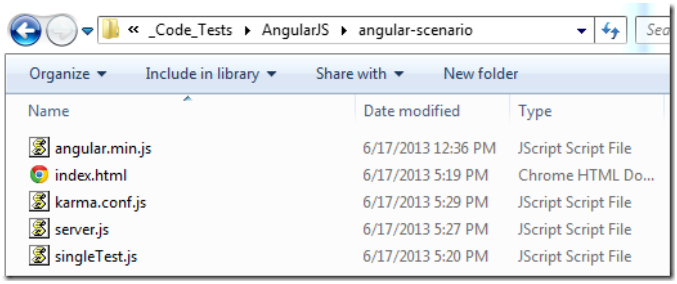
Table of Contents | Code

2.1 A small AngularJS Jasmine test executed by KarmaJS

When I try to understand how a particular technology works I always like to create a simple test case with a small number of moving parts.

This post shows such example for an AngularJS page, a Jasmine test, a NodeJS web server and a KarmaJS execution cycle.

The files used/customised were based on the KarmaJS test/e2e/angular-scenario example:



Let's look at each file's contents and what they do

1) **angular.min.js** is the latest version of AngularJS

2) **index.html** is a simple AngularJS example:

```
<!DOCTYPE html>
<html xmlns:ng="http://angularjs.org" id="ng-app" ng-app>
<head>
  <meta charset="utf-8">
  <title>Sample Angular App</title>
  <script src="angular.min.js"></script>
</head>
<body>
  <div>
    <label>Name:</label>
    <input type="text" ng-model="yourName" placeholder="Enter a name here">
    <hr>
    <h1>Hello {{yourName}}!</h1>
  </div>
</body>
</html>
```

3) **karma.conf.js** is a reduced to the normal KarmaJS config file

```
module.exports = function(karma)
    {
        karma.configure(
              {
                // generic
                frameworks    : ['ng-scenario'],
                urlRoot        : '/__karma/',
                autoWatch    : true,
                plugins     : ['karma-ng-scenario'],

                //project specific
                proxies : { '/': 'http://localhost:8000/'},
                files : ['singleTest.js'],

                //run specific
                singleRun : true,
              });
    };
```

4) **singleTest.js** checks if AngularJS is working as expected

```
/** A Sample Angular E2E test */

describe('SimpleTest', function()
{

    it('A small Angular Test', function()
        {
            browser().navigateTo('/index.html');
            input('yourName').enter('A Pirate!');
            expect(element('.ng-binding').text()).toEqual('Hello A Pirate!!');
        });
});
```

5) **server.js** is a working Web NodeJS server (also reduced for easier reading):

```
var sys = require('sys'),
    http = require('http'),
    fs = require('fs'),
    url = require('url'),
    events = require('events');

var DEFAULT_PORT = 8000;

function main(argv)
{
    new HttpServer({
                        'GET': createServlet(StaticServlet),
                        'HEAD': createServlet(StaticServlet)
                    }).start(Number(argv[2]) || DEFAULT_PORT);
}

function escapeHtml(value)
{
    return value.toString().
    replace('<', '&lt;').
    replace('>', '&gt').
    replace('"', '&quot;');
}

function createServlet(Class)
{
    var servlet = new Class();
    return servlet.handleRequest.bind(servlet);
}

/**
* An Http server implementation that uses a map of methods to decide
* action routing.
*
* @param {Object} Map of method => Handler function
*/
function HttpServer(handlers)
{
    this.handlers = handlers;
    this.server = http.createServer(this.handleRequest_.bind(this));
}

HttpServer.prototype.start = function(port) {
    this.port = port;
    this.server.listen(port);
```

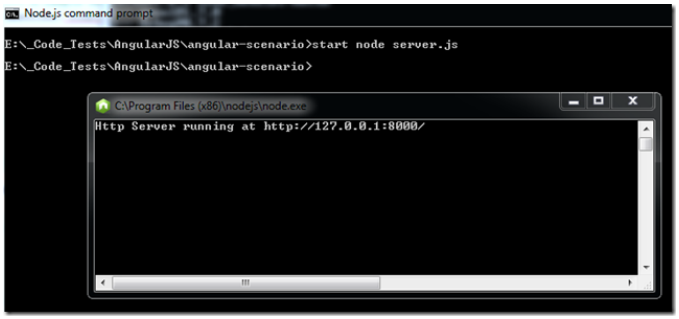
```
46      sys.puts('Http Server running at http://127.0.0.1:' + port + '/');
47  };
48
49  HttpServer.prototype.parseUrl_ = function(urlString) {
50      var parsed = url.parse(urlString);
51      parsed.pathname = url.resolve('/', parsed.pathname);
52      return url.parse(url.format(parsed), true);
53  };
54
55  HttpServer.prototype.handleRequest_ = function(req, res)
56  {
57      var logEntry = req.method + ' ' + req.url;
58      if (req.headers['user-agent'])
59      {
60          logEntry += ' ' + req.headers['user-agent'];
61      }
62      sys.puts(logEntry);
63      req.url = this.parseUrl_(req.url);
64      var handler = this.handlers[req.method];
65      if (!handler)
66      {
67          res.writeHead(501);
68          res.end();
69      } else
70      {
71      handler.call(this, req, res);
72      }
73  };
74
75  /**
76  * Handles static content.
77  */
78  function StaticServlet() {}
79
80  StaticServlet.MimeMap =
81      {
82          'txt': 'text/plain',
83          'html': 'text/html',
84          'css': 'text/css',
85          'xml': 'application/xml',
86          'json': 'application/json',
87          'js': 'application/javascript',
88          'jpg': 'image/jpeg',
89          'jpeg': 'image/jpeg',
90          'gif': 'image/gif',
```

```
91          'png': 'image/png',
92          'manifest': 'text/cache-manifest'
93      };
94
95  StaticServlet.prototype.handleRequest = function(req, res)
96      {
97          var self = this;
98          var path = ('./' + req.url.pathname).replace('//','/').replace(/%(..)/g, function(\
99  match, hex)
100             {
101                 return String.fromCharCode(parseInt(hex, 16));
102             });
103         var parts = path.split('/');
104
105         fs.stat(path, function(err, stat)
106             {
107                 if (err)
108                     return self.sendMissing_(req, res, path);
109                 return self.sendFile_(req, res, path);
110             });
111     };
112
113 StaticServlet.prototype.sendFile_ = function(req, res, path)
114     {
115         var self = this;
116         var file = fs.createReadStream(path);
117         res.writeHead(200,
118         {
119             // CSP headers, uncomment to enable CSP
120             //"X-WebKit-CSP": "default-src 'self';",
121             //"X-Content-Security-Policy": "default-src 'self'",
122             'Content-Type': StaticServlet.
123             MimeMap[path.split('.').pop()] || 'text/plain'
124         });
125         if (req.method === 'HEAD')
126         {
127             res.end();
128         } else
129         {
130             file.on('data', res.write.bind(res));
131             file.on('close', function()
132             {
133                 res.end();
134             });
135             file.on('error', function(error)
```

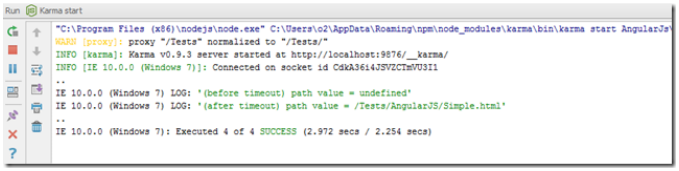
```
136             {
137                 self.sendError_(req, res, error);
138             });
139         }
140     };
141 
142 StaticServlet.prototype.sendError_ = function(req, res, error)
143     {
144         res.writeHead(500, {
145                               'Content-Type': 'text/html'
146                           });
147         res.write('<!doctype html>\n');
148         res.write('<title>Internal Server Error</title>\n');
149         res.write('<h1>Internal Server Error</h1>');
150         res.write('<pre>' + escapeHtml(sys.inspect(error)) + '</pre>');
151         sys.puts('500 Internal Server Error');
152         sys.puts(sys.inspect(error));
153     };
154 
155 StaticServlet.prototype.sendMissing_ = function(req, res, path)
156     {
157         path = path.substring(1);
158         res.writeHead(404, {
159                               'Content-Type': 'text/html'
160                            });
161         res.write('<!doctype html>\n');
162         res.write('<title>404 Not Found</title>\n');
163         res.write('<h1>Not Found</h1>');
164         res.write('<p>The requested URL ' + escapeHtml(path) +
165                   ' was not found on this server.</p>');
166         res.end();
167         sys.puts('404 Not Found: ' + path);
168     };
169 
170 // Must be last,
171 main(process.argv);
```

To see this in action

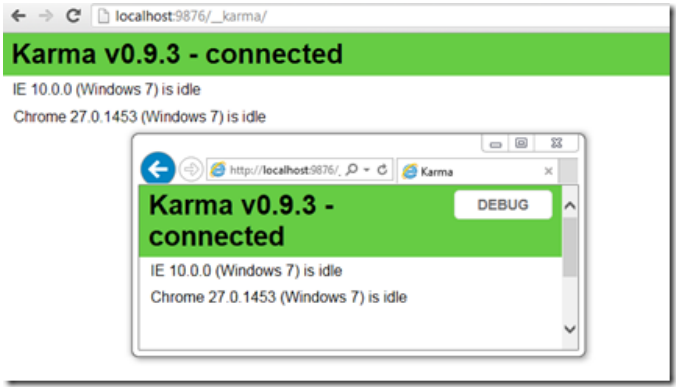
1) start the web server using ***start node server.js***



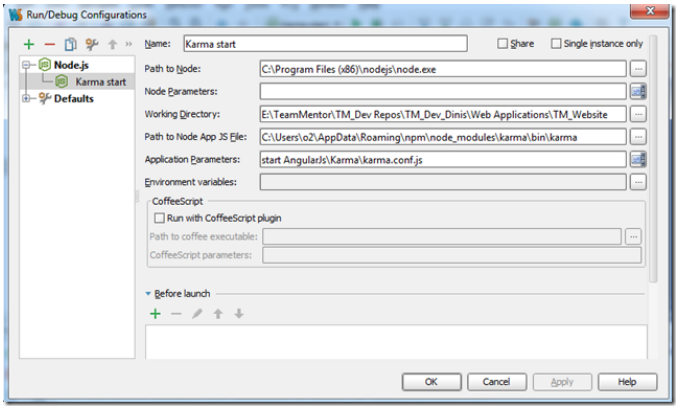
2) start karma with ***karma start karma.confi.js***



3) on a local browser open *http://localhost:9876/__karma*_ _which will execute the test



Since we have **singleRun **set to** true **(in ***karma.conf.js**), the karma process ends after each execution, which means that the captured browsers will go into a 'wait state' (i.e. waiting for another karma server to come back to life)



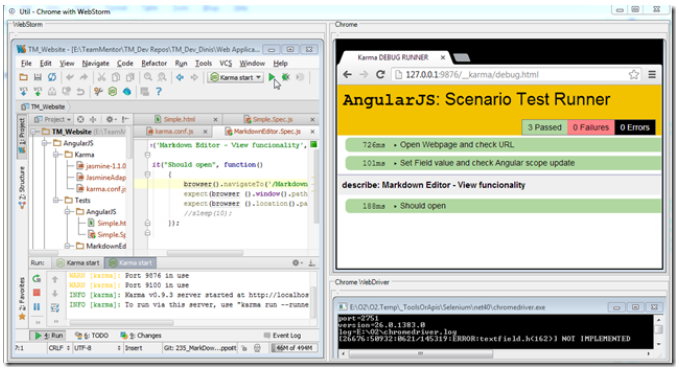
4) notice that back in karma, the tests executed ok:

```
E:\_Code_Tests\AngularJS\angular-scenario>karma start karma.conf.js
INFO [karma]: Karma v0.9.3 server started at http://localhost:9876/__karma/
INFO [Chrome 27.0.1453 (Windows 7)]: Connected on socket id 5WJJYzMys60uMjrZ_-2v
Chrome 27.0.1453 (Windows 7): Executed 1 of 1 SUCCESS (0.368 secs / 0.184 secs)
```

So here it is, a petty small example of a really powerful combination of technologies (and UnitTests workflows)

**NOTE: while creating this post, I wrote an O2 Platform C# script to help **

this script created a nice integrated test UI



Which allowed me to (in the same window) make changes and see its impact

C# script of UI shown above

```
var topPanel = "PoC - Karma and Angular run".popupWindow(1200,700);
//var topPanel = panel.clear().add_Panel();
//var textBox = topPanel.clear().add_RichTextBox();
var codeDir  = @"E:\_Code_Tests\AngularJS\angular-scenario";
var karma      = @"C:\Users\o2\AppData\Roaming\npm\node_modules\karma\bin\karma";

var testPage = "http://localhost:8000/index.html";
var testRunner = "http://localhost:9876/__karma/";

var karmaConfig = codeDir.pathCombine("karma.conf.js");
var serverConfig = codeDir.pathCombine("server.js");
var unitTestFile = codeDir.pathCombine("e2eSpec.js");

Process karmaProcess = null;

Action startWebServer =
    ()=>{
            Processes.startProcessAndRedirectIO("node", serverConfig,codeDir,(line)=>line.\
info());
        };

Action runKarma =
    ()=>{
            Action<string> consoleOut =
```

```
25                  (consoleLine)=> consoleLine.info(); //textBox.append_Line(consoleLine);
26
27              var command = "start \"{0}\" ".format(karmaConfig);
28              karmaProcess = "node".startProcess("\"" + karma + "\" " + command, consoleOut)\
29 ;
30          };
31
32 if (testPage.GET().notValid())
33 {
34     "Staring WebServer".info();
35     startWebServer();
36     1000.wait();
37 }
38
39 var toolStrip = topPanel.insert_Above_ToolStrip();
40 var codeEditor_Test = topPanel.add_SourceCodeEditor();
41 var ie_UnitTestRunner = codeEditor_Test.insert_Right().add_WebBrowser_with_NavigationBar()\
42 ;
43 var ie_Site = ie_UnitTestRunner.insert_Below().add_WebBrowser_with_NavigationBar();
44 var codeEditor_Config = codeEditor_Test.insert_Below().add_SourceCodeEditor();
45
46 codeEditor_Test.open(unitTestFile);
47 codeEditor_Config.open(karmaConfig);
48 ie_Site.open(testPage);
49 ie_UnitTestRunner.open(testRunner);
50 toolStrip.add_Button("Run","btExecuteSelectedMethod_Image".formImage(),()=>runKarma());
51
52
53 runKarma();
54
55 //using System.Diagnostics
```

2.2 Creating an Eclipse UI to run AngularJS e2e tests using Karma

This post shows how I created a nice set of views in Eclipse to quickly see the execution result of AngularJS e2e (end-to-end) tests, without leaving the Eclipse UI.

The image below shows this UI in action, where:

- The source code of the test is shown in the Eclipse Java editor
- Just below is the ***console out*** of the Karma runner (which is detecting files changes)
- On the top-right is the hooked browser (i.e. the one that will run the tests)
- On the middle-right is the simple AngularJS ***Hello World*** page
- On the bottom-right is the debug view of the hooked browser (which is what you get if you click on the ***Debug*** Button included on the top-right page)

Here are the tools and Eclipse Plugins used to create this UI:

- Eclipse Kepler for Java EE Developers with:
  - Eclipse Groovy REPL Scripting Environment 1.6.0 (update site)
  - AngularJS Eclipse Plugin (update site)
  - Nodeclipse (update site)
  - Ansi Console (update site)
- NodeJs with
  - Karma

The code used on this sample is based on my past attempts of using Karma (see A small AngularJS Jasmine test executed by KarmaJS , Running KarmaJS's AngularJS example test/e2e/angular-scenario (on Chrome) and KarmaJS related posts)

Although the AngularJS and NodeJs Eclipse plugins provide nice helpers and views, they didn't provide (yet) the kind of UI that I was looking for. Namely they didn't support the use of KarmaJS to run AngularJS tests.

But since I now have the ability to quickly manipulate and create new Eclipse views without restarting Eclipse (using the Groovy REPL script environment) it was not very hard to create the UI that I wanted (see Eclipse Plugin that allows the execution of REPL Groovy Scripts (in the current Eclipse Instance) and Fluent API for Eclipse+SWT).

Basically the brief was to:

- Create new (Eclipse) view with a browser showing _http://localhost:9879/__karma/ _(the KarmaJs browser runner/hook)
- Create new view with ***a browser showing http://localhost:9879/index.html*** (the AngularJS page)
- Create new view with a browser showing _http://localhost:9879/__karma/debug.html _(debug view of Karma runner), with an toolbar button to refresh it.

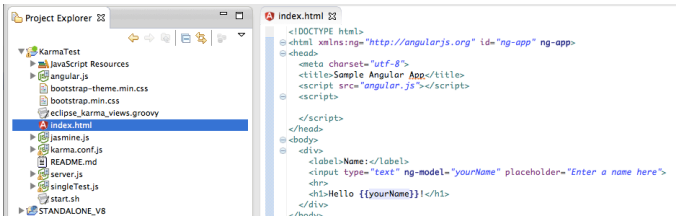
Here is the gist with the code that creates this 3 Eclipse views:

Hopefully this Groovy script is easier to read (the idea of the Fluent API that I added to the Groovy REPL was allow the easy reading and understanding of scripts like this).

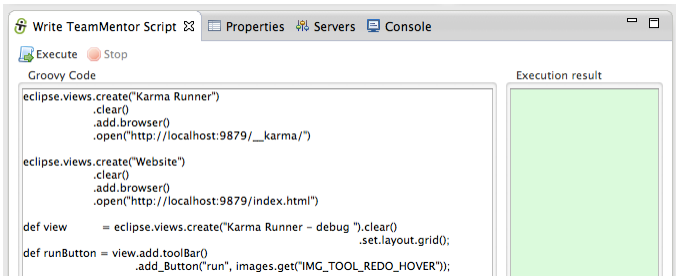
1) Setting up the environment

Now lets look at how this environment was setup:

We start with a simple AngularJS project test page that will just show the model-binding capabilities of AngularJS (yes I know that all those JS files should be in separate folders :))



Then I opened up Groovy REPL script environment_ _and wrote the script (shown above) that creates the multiple views:

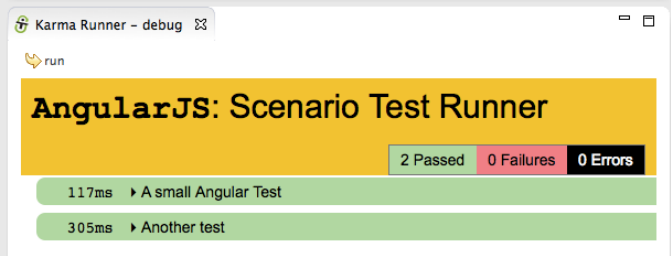


Clicking on _Execute _will create 3 views:

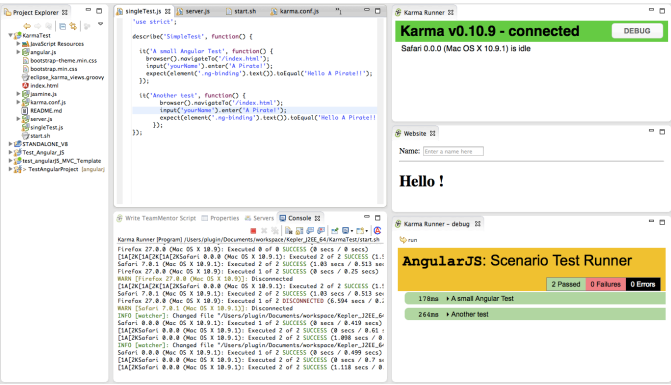
a) the ***Karma Runner*** view:

b) the **_Website** _view:

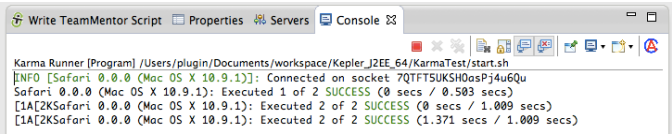
c) the ***Karma Runner - debug*** view:



... which (as seen in the first screenshot of this post) looks like this:



Part of this UI (but not created using the Groovy script) is an Eclipse *Console _with the _'Karma Runner process'* ***console*** **out **:



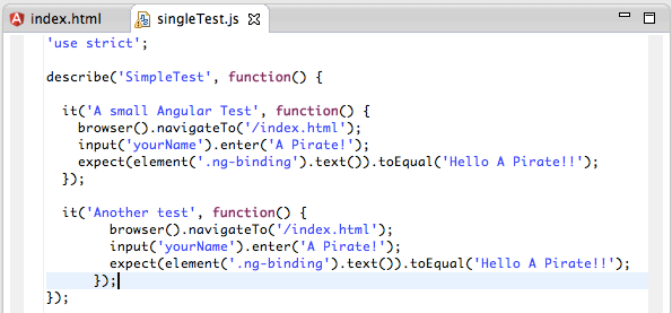
The AngularJS tests were executed when the ***Karma Runner*** view was opened, because the Karma process (configured via Karma.config.js) is set to wait for new browser's hooks/runners and (on connection or code changes) execute the configured tests.

2) Making changes to existing tests

To see Karma in action, let's make a test fail :)

The current AngularJS page is basically just echoing/binding the contents of the ***Name*** TextBox into the ***H1*** tag:

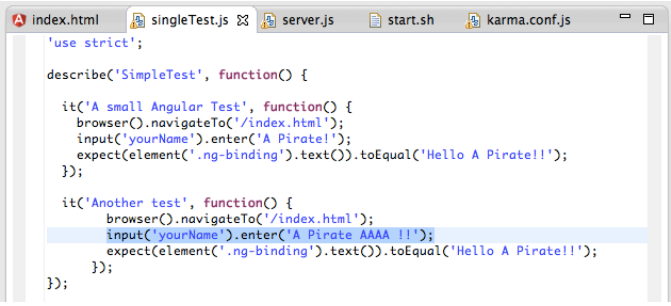
Here is the test that is currently being executed, which is an e2e test that runs on the browser (just like Selenium or Watin).



Hopefully this type of Fluent API is easy to read:

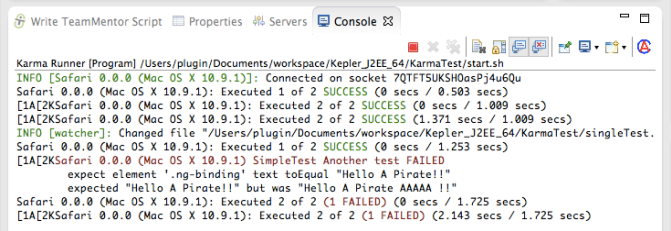
- browser navigates to ***/index.html***
- value is inserted into the ***yourName*** input field (the TextBox)
- the value of the binded element (***.ng-binding***) is checked to see if it matches

To make this test fail, let's change just the **yourName **value:

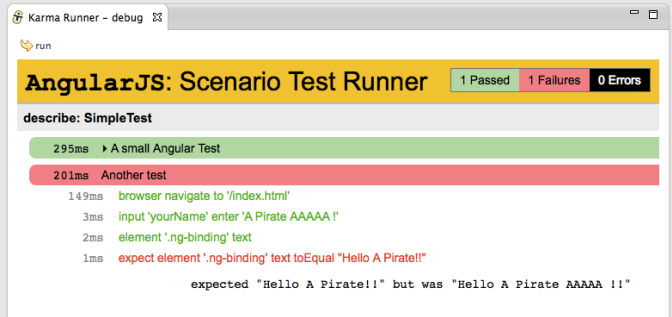


Immediately after saving the test with the changes shown above, Karma refreshes the hooked browsers in order to trigger the execution of the tests.

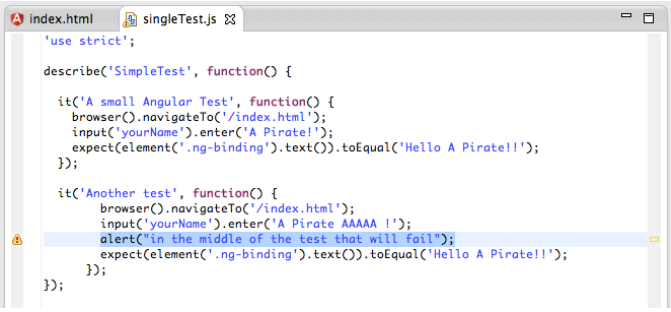
And as the image below will show, there is now one failed test:



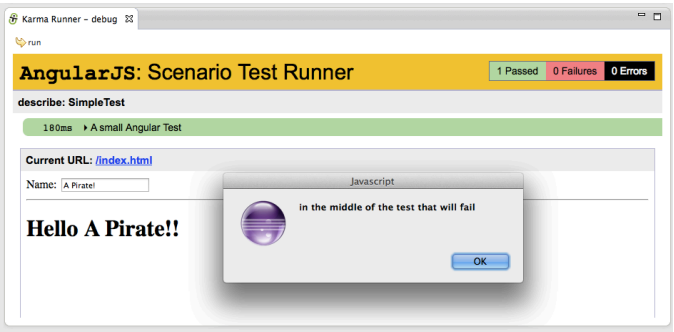
In cases like this, the _**Karma Runner - debug** _is very useful since it will show more details on what happened, and where the test failed:



Just to confirm that the page is actually being loaded and the tests are happening in a real browser instance, if we add an Javascript alert to the current test:



... we will get an alert box in the **Karma Runner - debug** (note that on the screenshot below we' are seeing the image of the final state of execution of the previous test (in this case the *'A small Angular Test'*)

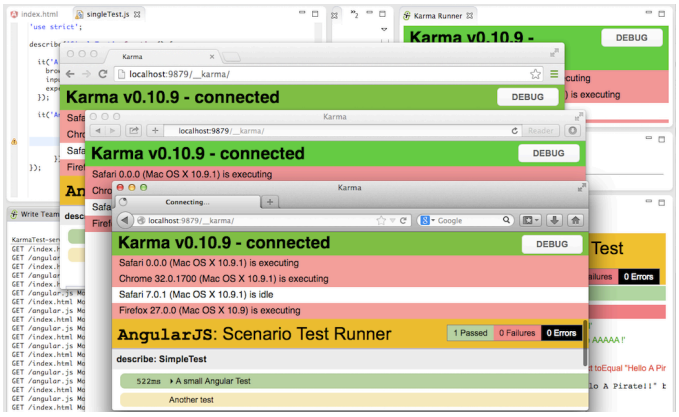


3) Hooking multiple browsers and only failing on Firefox

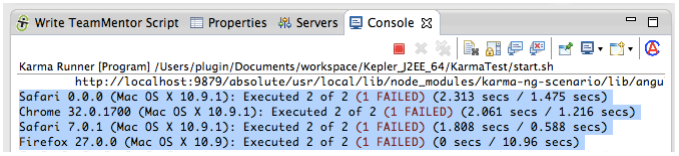
In order to run the tests in multiple browsers all we need to do is to open the _http://localhost:9879/__karma/_page (in the target browser), and a test execution will be triggered.

Note how on the image below:

- Eclipse is in the background (containing the views previously created and showing the ***console out*** of the Karma test runner process)
- There are 3 stand lone browser windows (Chrome, Safari and Firefox)
- Firefox (top most window) shows the tests being executed (in green the test executed, in yellow the test being executed)

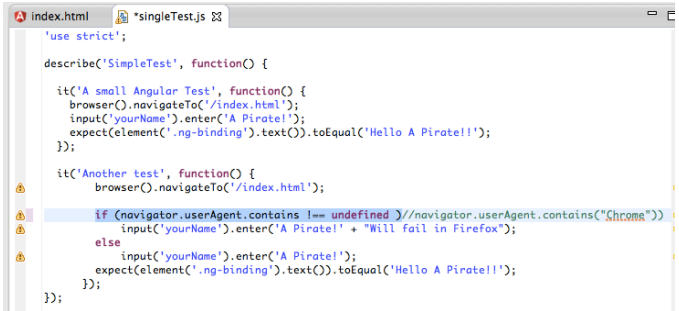


After execution, a quick look at the Karma runner shows that the modified test failed (as expected) on all browsers:

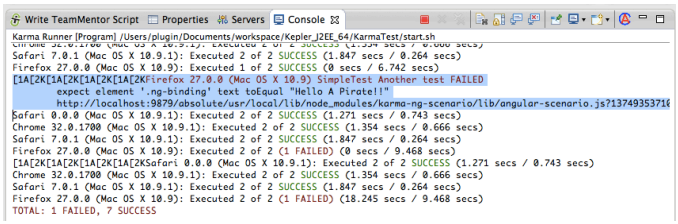


Just to make sure all is working as expected, let's create a test that will fail only in one browser.

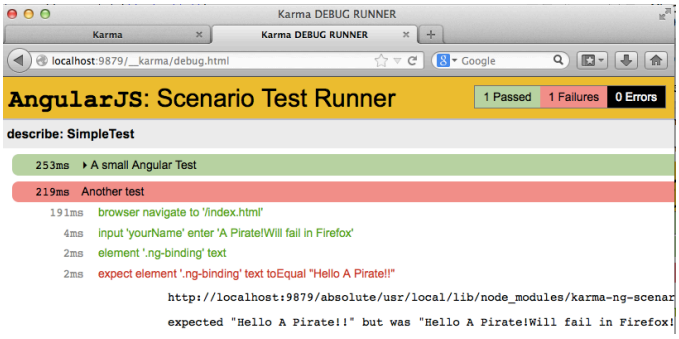
For example Firefox is the only one that has defined the ***navigator.userAgent.contains*** Javascript function (so we can use this to detect Firefox, and create a test that will only fail if executed inside it):



After saving the changes, the tests will be executed on all 4 browsers, with the one test that failed being the one executed in Firefox:



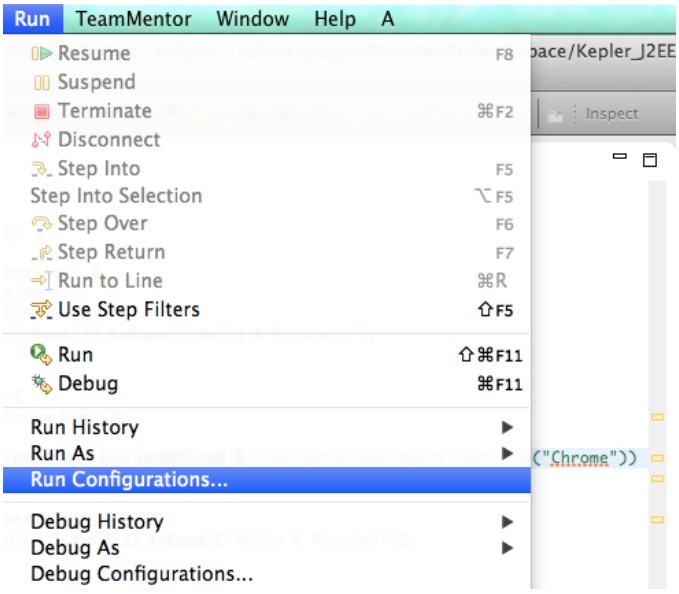
Refreshing the Firefox debug runner, shows the failed test and assert:



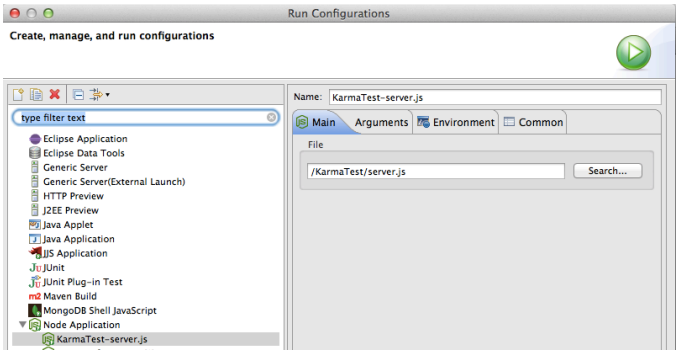
4) Setup

In addition to creating the views programatically, I also setup an Eclipse_ **Run configurations** ** ***for*** ** _-***NodeJS*** and an _External Tools Configuration _for ***KarmaJS***.

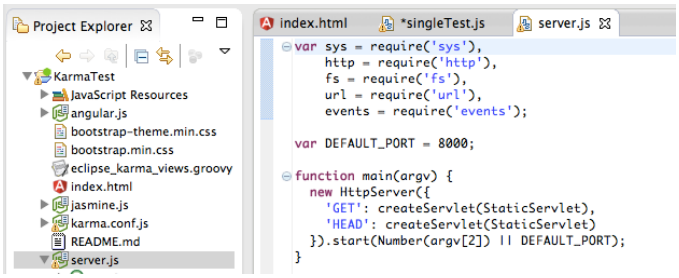
The NodeJS configuration was done here:



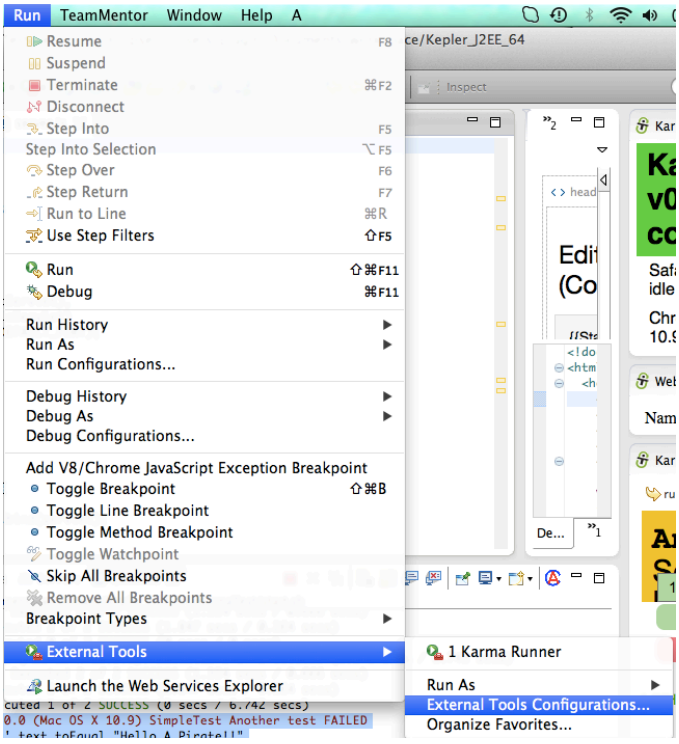
... where I created a Node Application runner that was pointed to ***server.js***



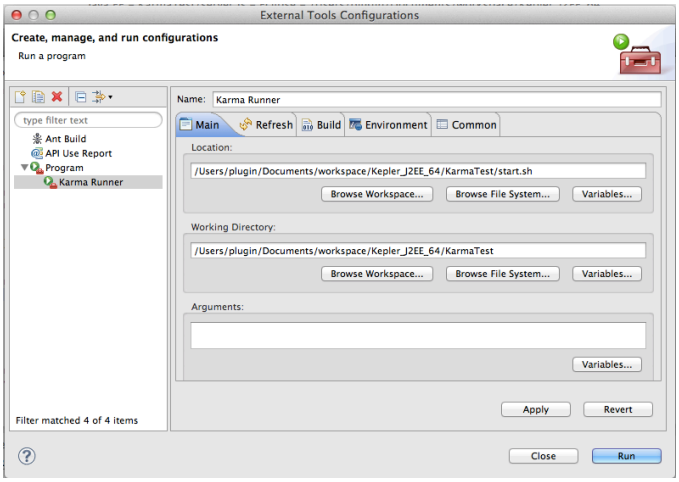
... which contains a simple NodeJS web server (we could also had used NodeJS Express, but that would had added a lot of other 'stuff' to this project)



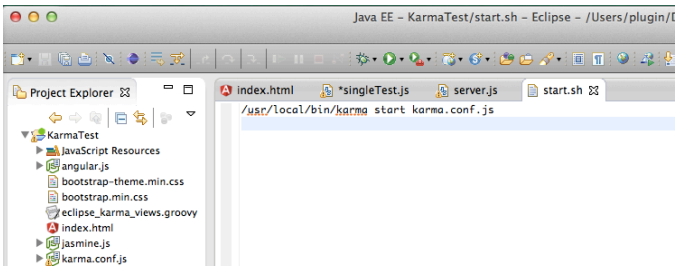
The KarmaJS config was done as an ***External Tools Configuration***



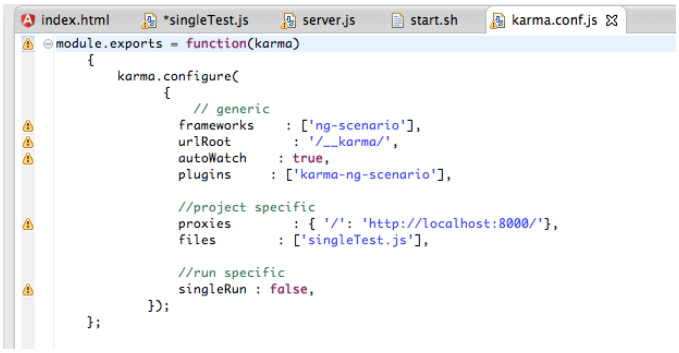
... where I simply run the ***start.sh*** bash file



... which starts ***KarmaJS***:



... using this simple ***karma.conf.js*** configuration



Note: I started the Karma process like this because there was an issue with ***KarmaJS*** finding the **NodeJS **executable from Eclipse (and at the time I didn't had time to debug why that was happening)

External References:

- AngularJS docs on E2E Testing
- AngularJS Testing with Karma and Jasmine
- Test runners for AngularJS - how to run the tests from eclipse IDE and CI server without too much complication?
- How to run KarmaJS from Eclipse
- Setting-up AngularJS, Angular Seed, Node.js and Karma

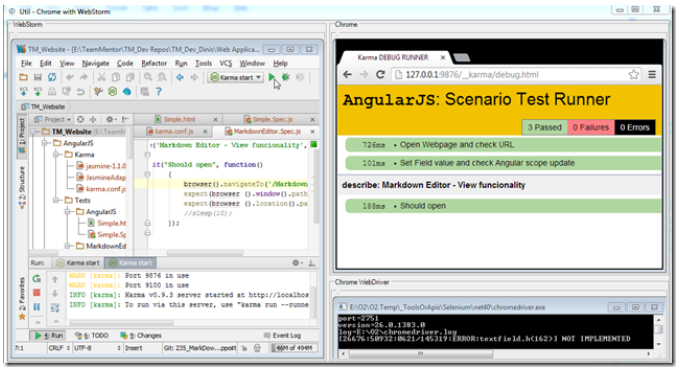
- Can’t run e2e tests with karma (for angular)

Table of Contents | Code

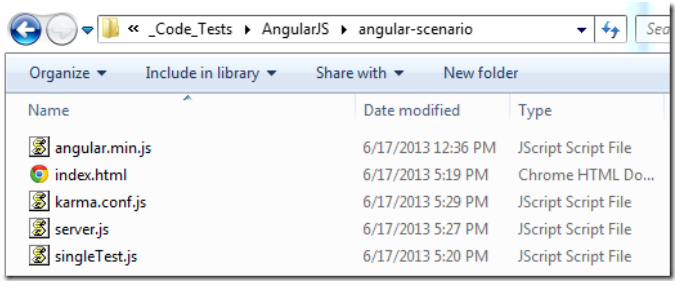
2.3 Running KarmaJS's AngularJS example test/e2e/angular-scenario (on Chrome)

To learn and get an idea of how Karma (the *'Spectacular Test Runner for JavaScript'*) works, and how it can be used to create browser automations tests, here are the steps I took to get the test/e2e/angular-scenario example to work.

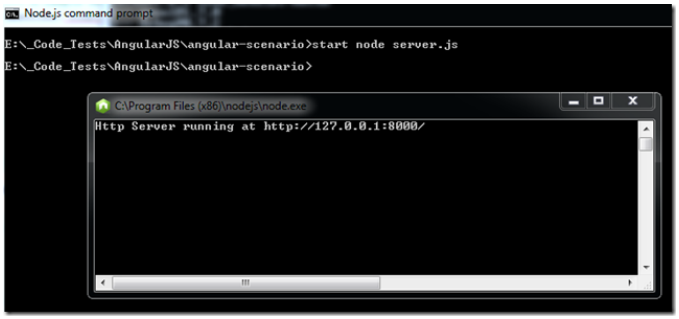
It all started with a clone of: git@github.com:karma-runner/karma.git



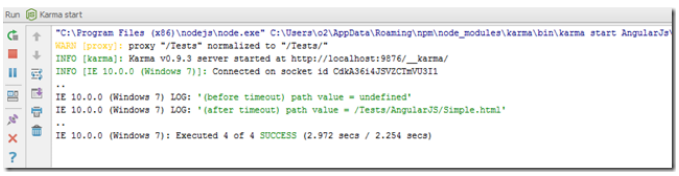
With this KarmaJS test example (see below), being the one that we are going to use:



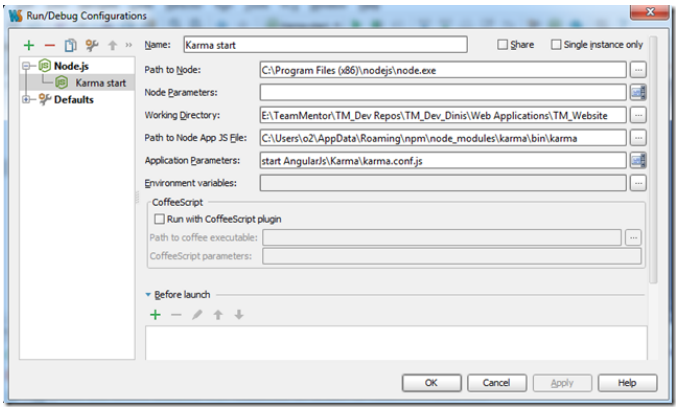
I then opened an nodejs command prompt and navigated to the folder shown above:



... used **_node server.js** _to start a local webserver

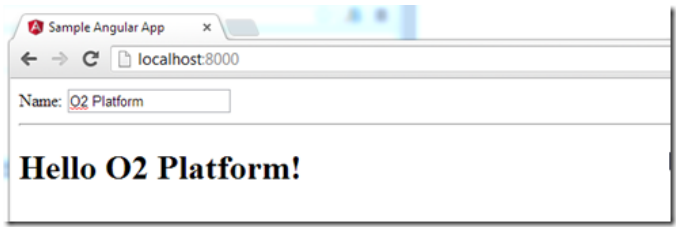


... on port 8000:

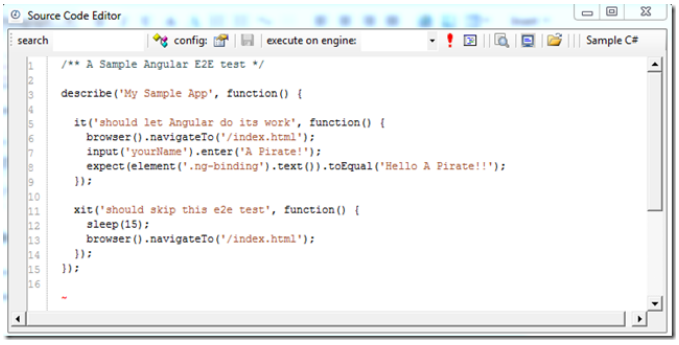


The test case we are using (on KarmaJS's ***test/e2e/angular-scenario***) is a simple AngularJS example, which just consumes the angular-min.js file model attribute:

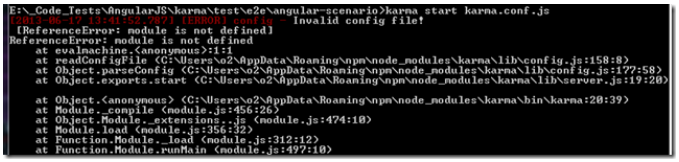
... and uses angular to populate the ***{{yourName}}*** value dynamically (wired to the input field via the ***ng-model="yourName"***)



Next we are going to run this Jasmine ('*Behavior-**D**riven **D**evelopment framework for testing JavaScript code*') test using KarmaJS

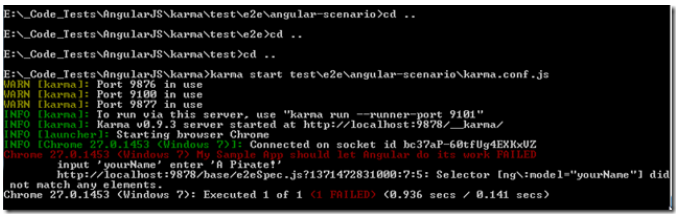


... which *should* work with just: **karma start karma.conf.js**



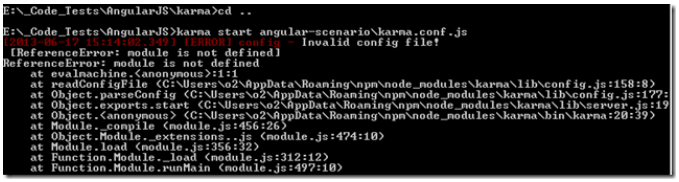
... but it didn't

There is a module dependency missing, which in this case can be resolved by running this command from the root of the karma repository:

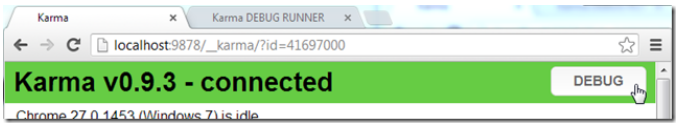


UPDATE: the issue above was caused by the fact that I had an the official released version of karma installed globally which is the one that was running when I tried it on the 'test/e2e/angular-scenario' folder'

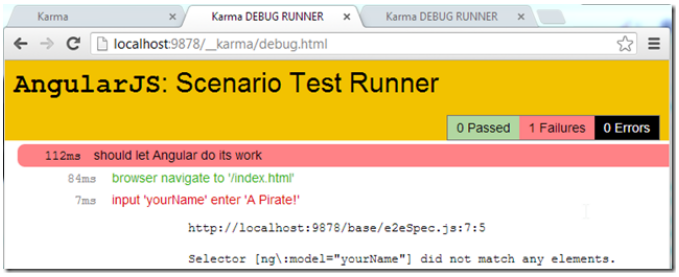
And now (based on an option from the karma.conf.js) a Chrome window opened up:



Clicking on the ***Debug*** button:

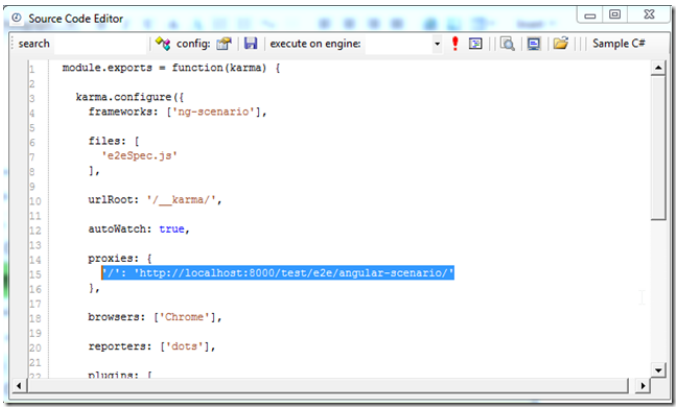


... we can more details on the test that failed:

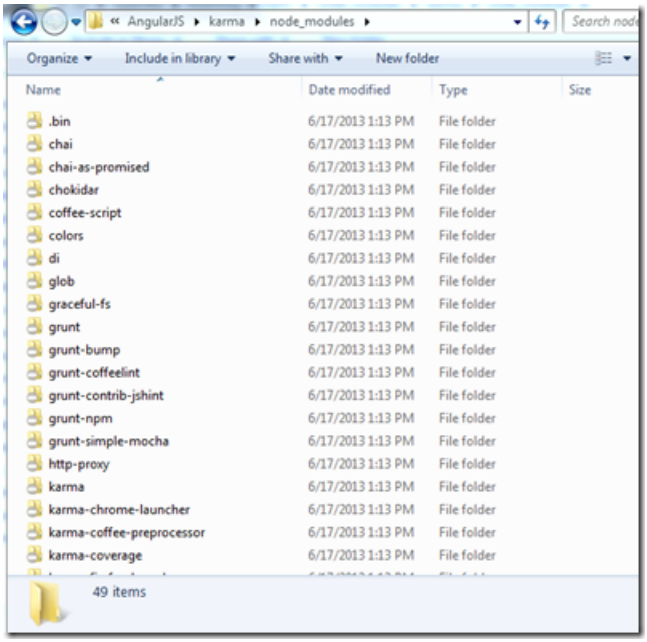


In this case the problem is that the 'proxy mapping' that karma does is not correct

If we look at the karma.config.js file

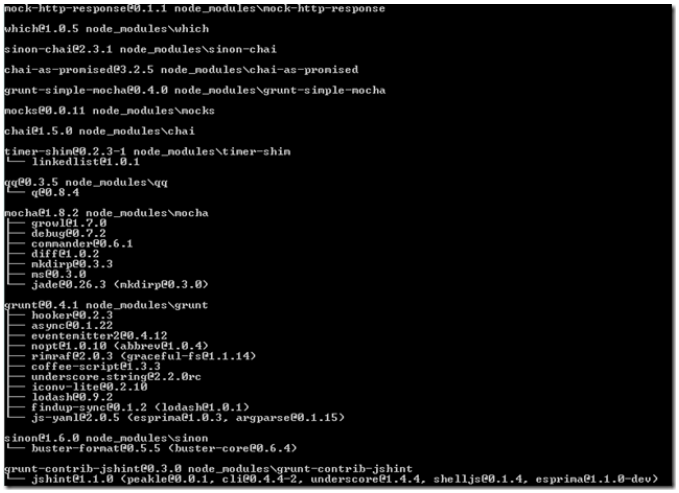


…. and the unit test ***brower().navigateTo*** command



… we can see that karma will try to open ***/index.html*** from ***http://localhost:8000/test/e2e/angular-scenario/index.html***

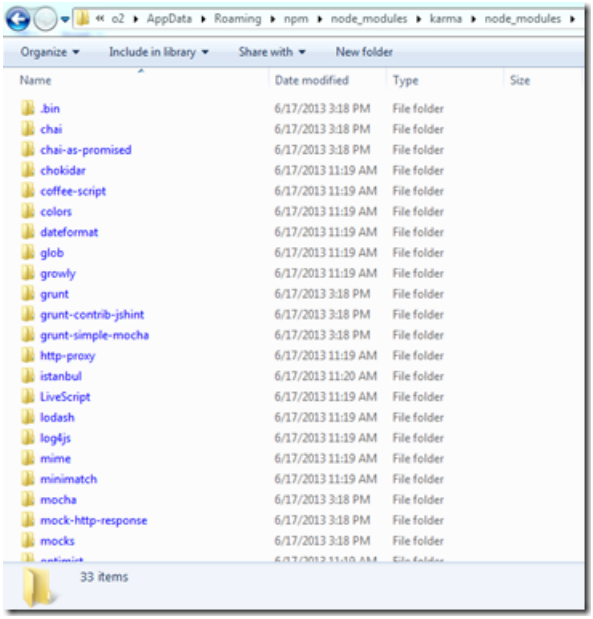
That is not going to work since the page we want it is on http://localhost:8000/index.html (which happened because we started **_node server.js** _on the** .../test/e2e/angular-scenario **folder)



One way to solved this, is to change the 'proxies' mapping to *'/' : 'http://localhost:8000/'*

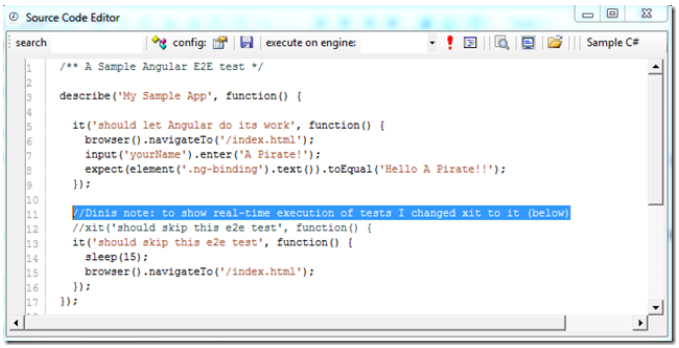
... and after stopping and starting the karma server:

all tests pass :)

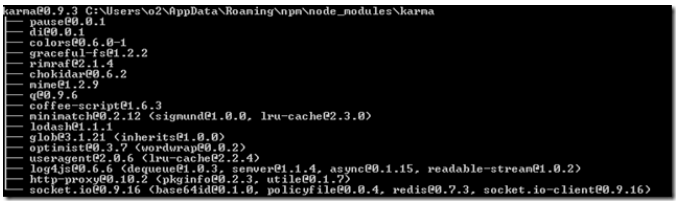


Changing and executing tests in real time

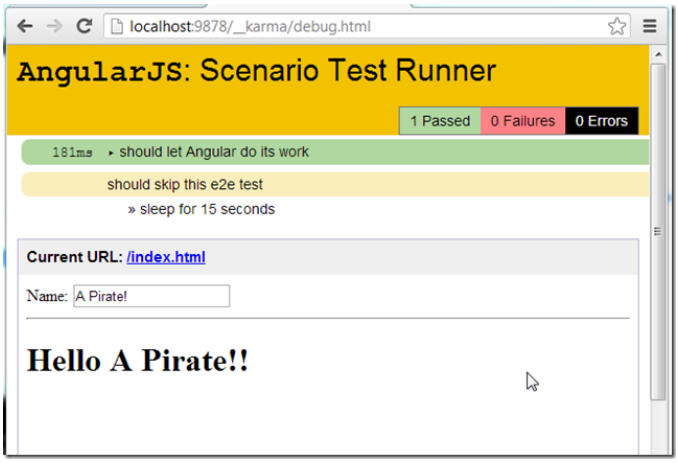
What is really cool with this set-up is that (as long as the karma process is running), because the **autoWatch** value (in k_**arma.config.j**_s) was set to true, if I make a change to the test file:



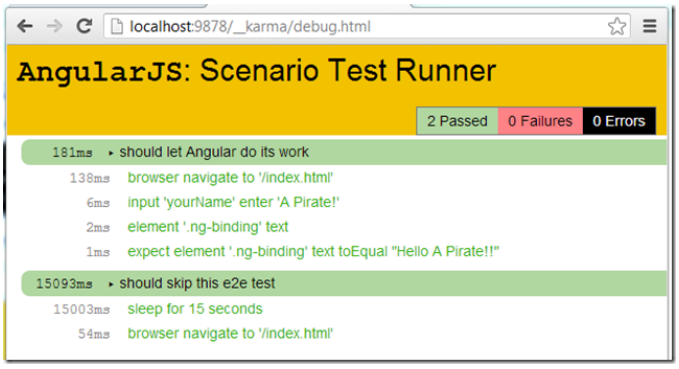
... the karma runner will detect the changes and rerun the tests (note that there are 2 tests executed now)



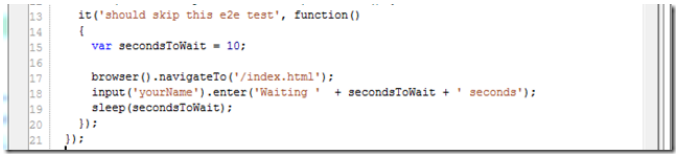
This 2nd test shows an interesting behaviors since it will make the test wait for 15 seconds (with the browser window opened):



Note how the time execution time for the 2nd test was ~15 secs

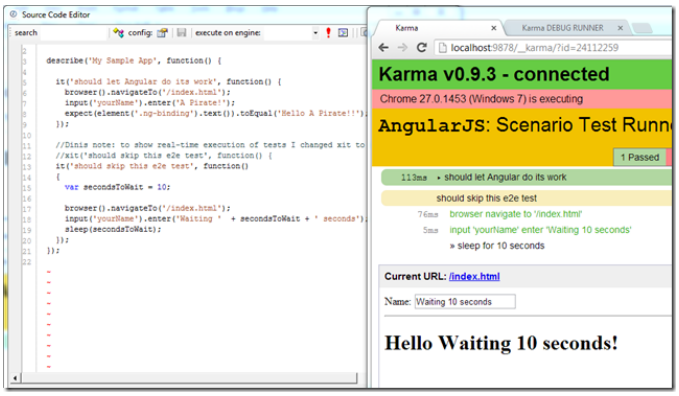


And if we modify the 2nd test to:



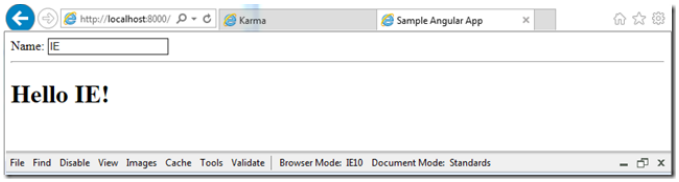
```
13    it('should skip this e2e test', function()
14    {
15      var secondsToWait = 10;
16
17      browser().navigateTo('/index.html');
18      input('yourName').enter('Waiting '  + secondsToWait + ' seconds');
19      sleep(secondsToWait);
20    });
21  });
```

... the execution test UI will look like this (note that the execution was triggered when I saved the test file :)



**NOTE 1: **to solve the ENOENT error shown the first screenshot of localhost:8000, we just needed to a

favicon.ico file to the **lib/nodeserver **folder



... and now the error doesn't happen anymore

Note 2: **when trying to run Karma for the first time, I had a prob with grunt where it was failing with an ***Error: spawn ENOENT*:

... this was resolved by installed the 32bit version of nodeJS and running ***npm install*** on the karma folder (after cloning it)

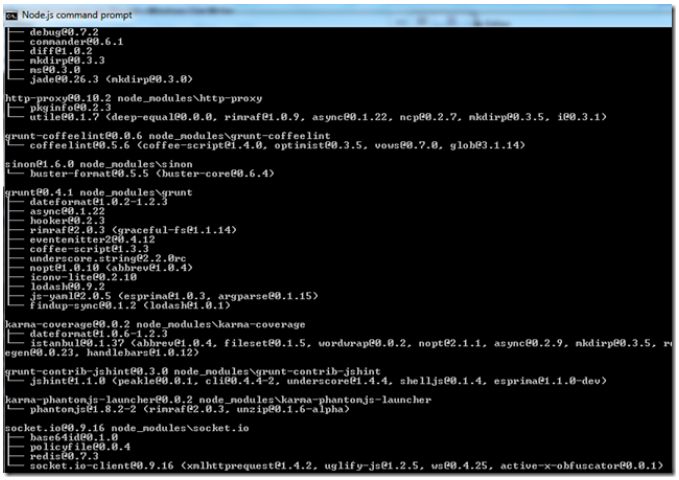


Table of Contents | Code

3 Firebase

This section has the following chapters:

- First PoC of sending TeamMentor's server-side request URLS to Firebase
- Trying out Firebase (Beta) hosting solution and good example of Firebase Security rules

---

Table of Contents | Code

3.1 First PoC of sending TeamMentor's server-side request URLS to Firebase (and seeing it in realtime in an AngularJS page)

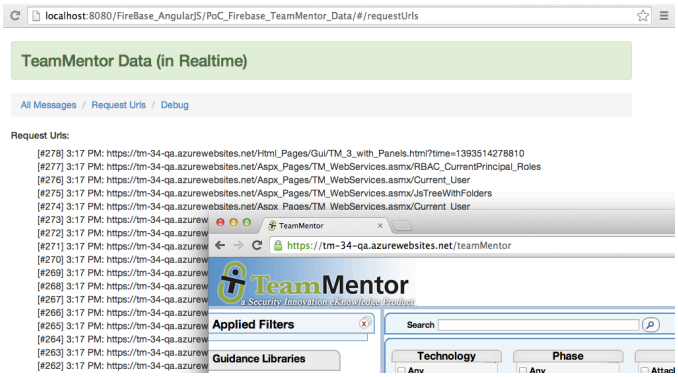
After getting my head around how Firebase works (see Using Firebase to sync data with a webpage (via Javascript, REST and Firebase Admin panel) and Trying our Firebase (Beta) hosting solution and good example of Firebase Security rules), I really wanted to see how it could work on a key feature that I've been wanting to add to TeamMentor for ages: **Realtime ****viewing of ****traffic and logs**

And it worked :)

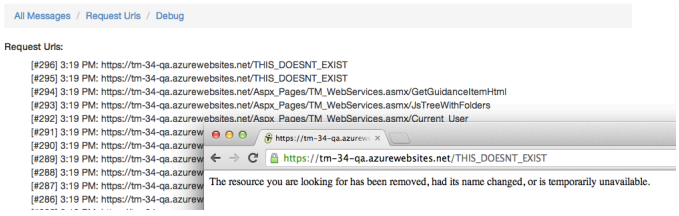
This is really exciting!!! (can you tell :)), specially since I can see so many great uses of this type of technique and technology in TeamMentor (for example it will allow for much better understanding on how the content is used, and for better collaboration between its readers (and authors))

I'm going to write a number of blog posts that explain how this works in detail, but the core of it is the C# code shown below (gist here) which is running on a test TeamMentor instance, and basically hooks into the ASP.net HTTP pipeline in order to send the current URL to Firebase (using Firebase's REST API):

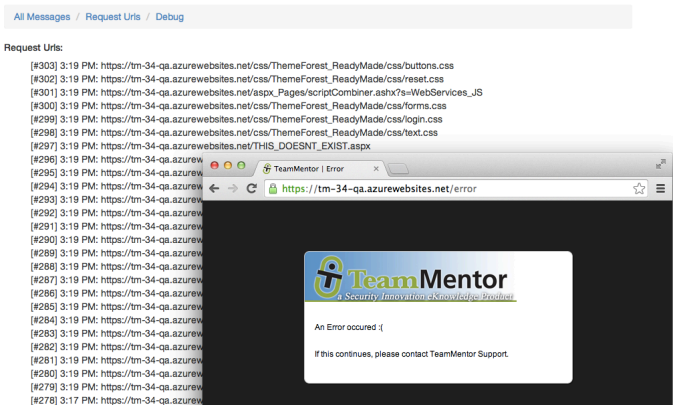
Here is the AngularJS+Firebase Html app that I created in Eclipse which shows the data received from the TeamMentor website (i.e. all requests made by a client visiting its home page):



... here are the logs for a page that doesn't exist:



... here are the requests for the TeamMentor error page:



Quick look at the C# code executed on the TeamMentor server **(gist here):**

Here is how the HTTP pipeline is hooked (using the TMEvents helper object from TM)

```
if (TMEvents.OnApplication_BeginRequest.size() > 1)
    TMEvents.OnApplication_BeginRequest.remove(1);

logDebugMsg("mapping TMEvents.OnApplication_BeginRequest to send url to Firebase");

TMEvents.OnApplication_BeginRequest.Add(
        ()=>{
                logRequestURL(HttpContextFactory.Request.Url.str());
            });
return TMEvents.OnApplication_BeginRequest.size();

//using System.Threading;
//using TeamMentor.CoreLib;
//O2Ref:TeamMentor.CoreLib.dll
```

... the ***logDebugMsg*** and ***logRequestURL*** lambda functions are used to set the Firebase object to store the received messages:

```
Action<string> logDebugMsg =
        (message)=>{
                        sendData("debugMsg", message);
                    };

Action<string> logRequestURL =
        (url)=>{
                        sendData("requestUrl", url);
                };
```

... the *sendData* lambda function is used to configure the Firebase target app and authorisation token:

```
Action<string,string> sendData =
        (area, data) => {
                            var app       = "tm-admin-test";
                            var authToken = "1luXXuQHpzhrG2LzVlDNu17tOBu0paTqR6bNhFZm";
                            sendData_Raw(app, authToken, area, data);
                        };
```

... the **sendData_Raw **lambda function is the one that sends the REST/POST request to the mapped Firebase app:

```
Action<string,string,string,string> sendData_Raw =
        (app,token, area, data)=>
                {
                    ThreadPool.QueueUserWorkItem((o)=>
                       {
                            var url      = "https://{0}.firebaseio.com/{1}.json?auth={2}".format(app, area, token);
                            var now      = DateTime.Now.ToShortTimeString(); // DateTime.Now. TimeOfDay;
                            var postData = "\"{0}: {1}\"".format(now, data.replace("\"", "'"));
                            url.POST(postData);
                        });
                };
```

Let me know if you have any questions regarding this example.

Firebase recovers connection

In terms of being able to recover from going offline, Firebase seems to do a good job at reconnecting back to the server once the host box is back online (see image below which happened between me leaving my house and connecting into my current WIFI location)

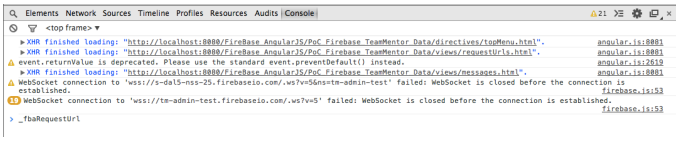


Table of Contents | Code

3.2 Trying out Firebase (Beta) hosting solution and good example of Firebase Security rules

Since Firebase now offers a Beta hosting service (and I was looking for a quick way to publish one of the firebase PoCs I'm working at the moment), I decided to take it for a spin.

I have to say that I'm really impressed with the end result, and as you will see below, there entire process (from install to published website) was really smooth.

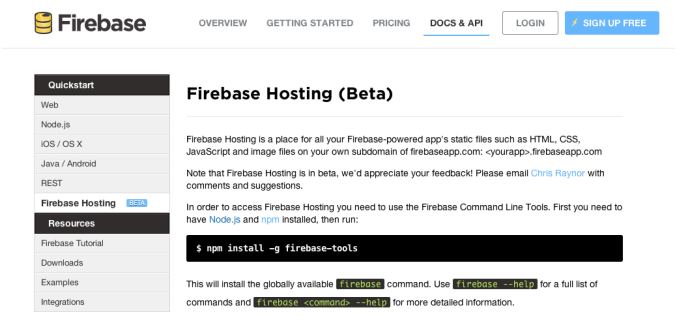
Note 1: in the example below I already had created an Firebase app to hold the data (see Using Firebase to sync data with a webpage (via Javascript, REST and Firebase Admin panel) for details on how to create one)

Note 2: at the time I wrote this post, the website created is/was (depending on when you are reading this) hosted at https://tm-admin-test.firebaseapp.com/

Starting with the instructions from Firebase hosting page:

**

**



1) Install the firebase tools

```
~Czens-MacBook-Air:FireBase_AngularJS plugin$ sudo npm install -g firebase-tools
npm http GET https://registry.npmjs.org/firebase-tools
npm http 304 https://registry.npmjs.org/firebase-tools
npm http GET https://registry.npmjs.org/bitballoon
npm http GET https://registry.npmjs.org/prompt
npm http GET https://registry.npmjs.org/tar
npm http GET https://registry.npmjs.org/open
npm http GET https://registry.npmjs.org/optimist
npm http 200 https://registry.npmjs.org/tar
npm http GET https://registry.npmjs.org/tar/-/tar-0.1.19.tgz
npm http 200 https://registry.npmjs.org/tar/-/tar-0.1.19.tgz
```

....

```
npm http GET https://registry.npmjs.org/request/-/request-2.9.203.tgz
npm http 200 https://registry.npmjs.org/eyes/-/eyes-0.1.8.tgz
npm http 200 https://registry.npmjs.org/stack-trace/-/stack-trace-0.0.9.tgz
npm http 200 https://registry.npmjs.org/async
npm http 200 https://registry.npmjs.org/request/-/request-2.9.203.tgz
/usr/bin/firebase -> /usr/lib/node_modules/firebase-tools/bin/firebase
firebase-tools@0.1.6 /usr/lib/node_modules/firebase-tools
├── open@0.0.4
├── optimist@0.6.1 (wordwrap@0.0.2, minimist@0.0.8)
├── tar@0.1.19 (inherits@2.0.1, block-stream@0.0.7, fstream@0.1.25)
├── bitballoon@0.1.2 (base64-js@0.0.6, glob@3.2.9)
└── prompt@0.2.12 (revalidator@0.1.6, pkginfo@0.3.0, utile@0.2.1, read@1.0.5, winston@0.6.2)
```

**

**

2) Run the firebase BootStrap

```
zens-MacBook-Air:FireBase_AngularJS plugin$ firebase bootstrap
Login required
Email: dinis.cruz@owasp.org
Password:
```

... after logging in (above), we are asked to chose the 'firebase' app to use (below)

```
---------- YOUR FIREBASES ----------
incandescent-fire-1320
teammentor
tm-admin-test
xss
------------------------------------
Firebase: tm-admin-test
```

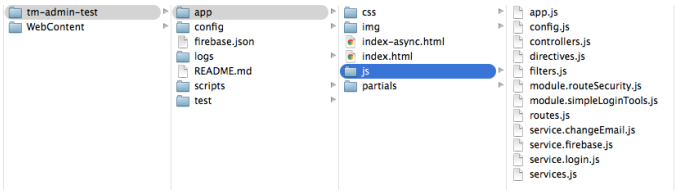
... then pick the template to use (I chose the ***angular*** one)

```
------- AVAILABLE TEMPLATES --------
angular
chat
------------------------------------
Template: angular
```

... and that's it:

```
Bootstrapping into directory 'tm-admin-test'
Downloading and unpacking template
Writing firebase.json settings file
BOOTSTRAPPING SUCCESSFUL - Instructions and setup guide here: https://github.com/firebase/angularFire-seed
zens-MacBook-Air:FireBase_AngularJS plugin$
```

Here are the files created locally (all under the ***tm-admin-test*** folder)

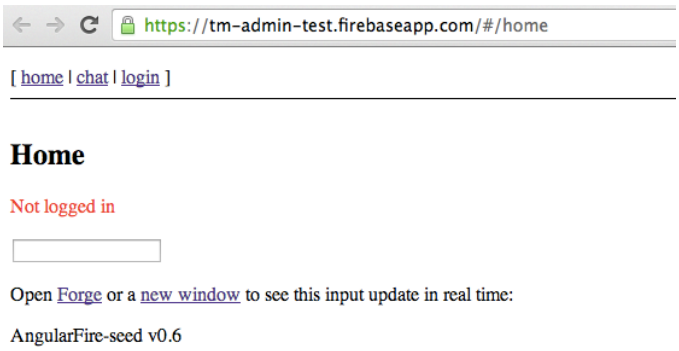


3) Deploy the application

```
zens-MacBook-Air:tm-admin-test plugin$ firebase deploy
Updating security rules...
Deploying...
DEPLOYED SUCCESSFULLY - View your app at: https://tm-admin-test.firebaseapp.com
zens-MacBook-Air:tm-admin-test plugin$
```

... which is also super quick.

After completion the default browser is opened with the created website, which looked like this:

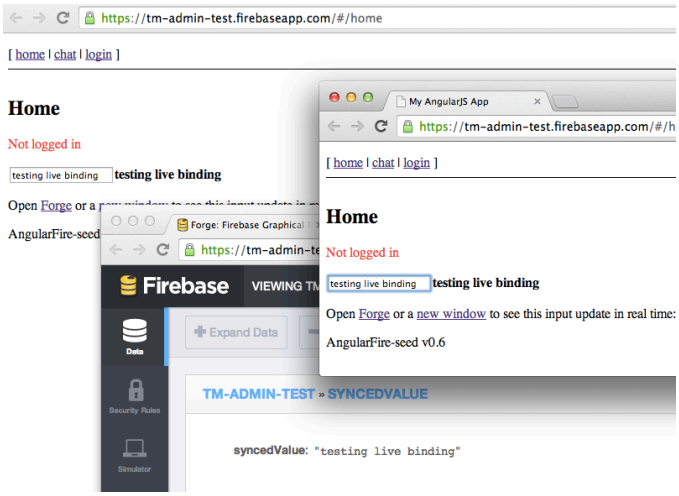


4) Testing the test app

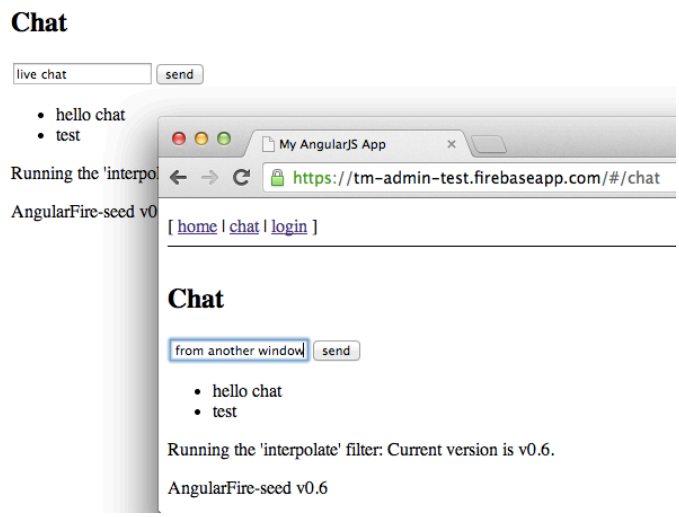
**

The created/published test app has a simple Firebase 3-way data binding example, where changes in an **_angular-binded _model, are propagated to the server, and distributed to any connected clients (the image below shows two browsers and the firebase admin panel, all synced in real time (regardless where the data is

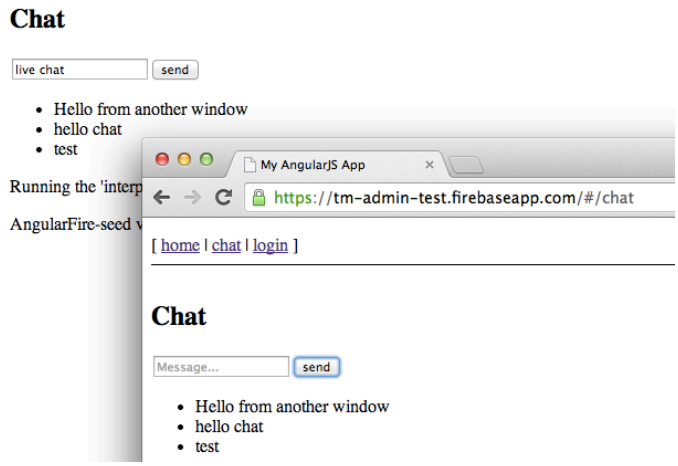
changed))
**
**



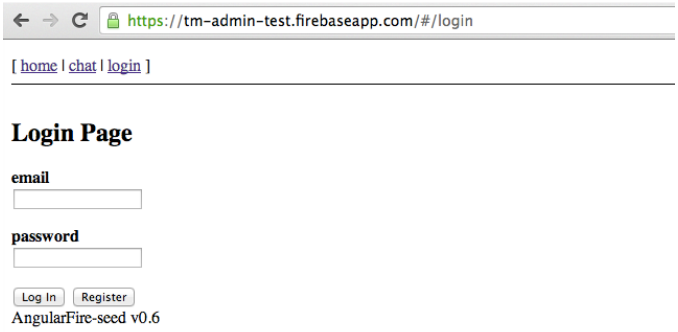
There is a basic **Chat **page:



... which allows new messages to be added and viewed in real time:



Also included is a Login page (with account creation capabilities):

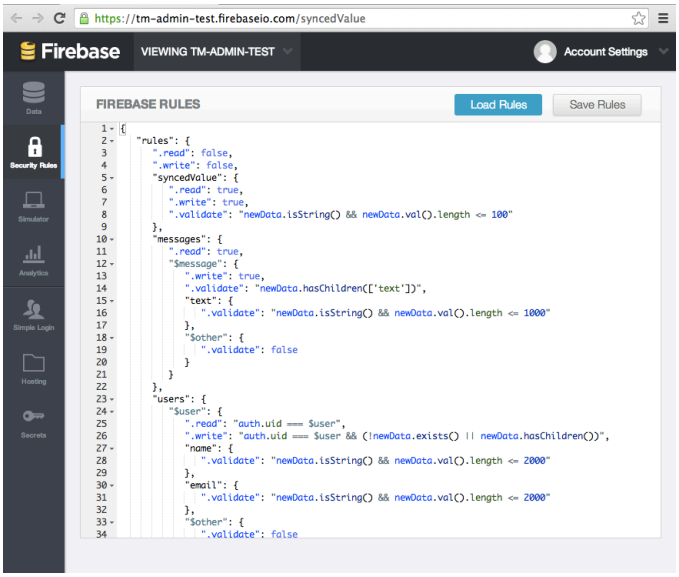


5) Review the security rules

At this stage it's good to take a look at the security rules added to this Firebase app.

In the image below we can see that:

- Anonymous users ***can read*** all data, but ***not write*** by default
- The ***syncedValue*** object ***can be read and written*** by all
  - there is a validation on this field that ensures that it is not empty and not bigger that 100 chars
- The ***messages*** object (i.e. 'firebase section/subdomain'):
  - can be read by all
  - each message must include a property called_ text _with a max length of 1000 chars
  - (I think) the ***.validate = false*** means that no other properties can be added
- the ***users*** object:
  - each ***$user*** child object:
    * can be read if the current user matches its name (i.e. it is logged in as that user)
    * each user can write into two fields: ***name*** and ***email*** (both with a max length of 2000)



6) Create an account

To test the provided authorisation solution, let's start by trying to login with an account that doesn't exist:

email
a@a.com

password
•••

Log In Register

Error: The specified user does not exist.

AngularFire-seed v0.6

... then click on the ***Register*** button (which just adds a ***confirm pass*** field)

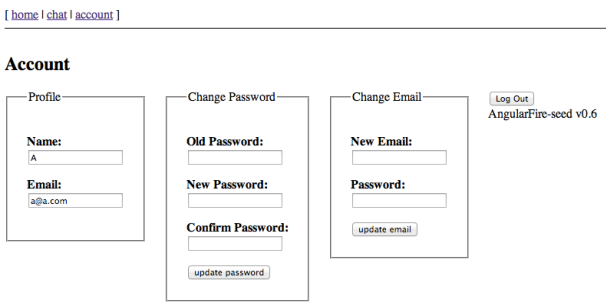
Login Page

email
a@a.com

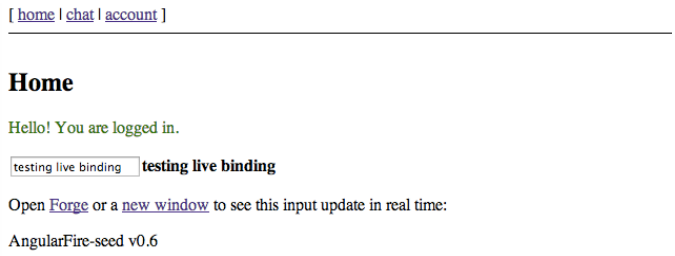
password
•••

confirm pass
•••

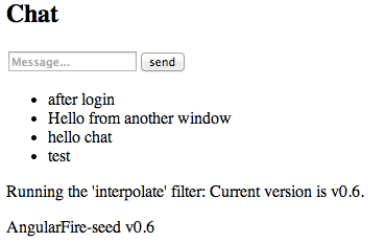
Here is what the 'post register/login' page looks like (note that the top-menu-bar ***login*** link was also changed to ***account***)



Clicking on *home* takes us to the first page, which also shows a different message (now that we are logged in)



Interestingly the ***Chat*** page doesn't seem to take into account that we are logged in now (would be nice to show the current user in the message posted)

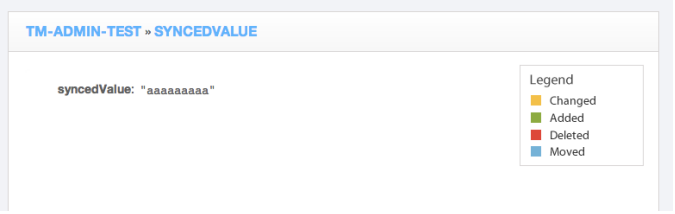


6) Take a look at data (as stored in the assigned Firebase app)

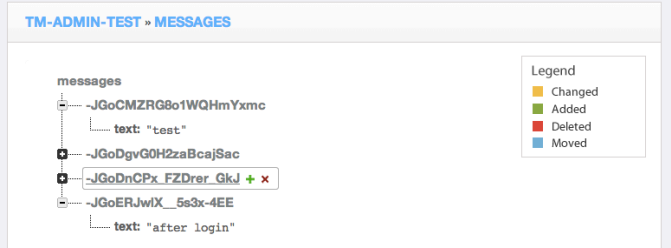
**

**Using the control panel for the current app:

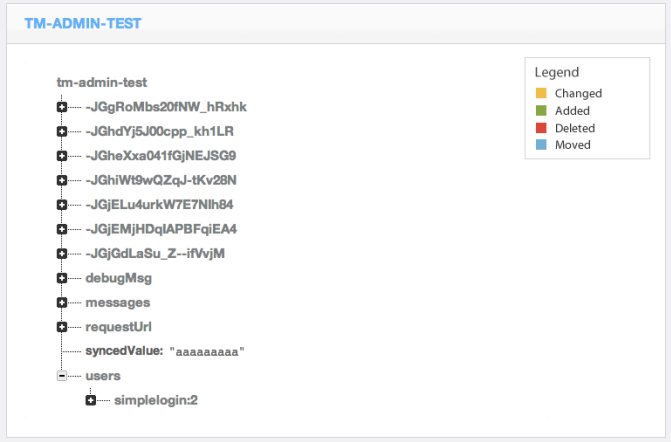
a) here is the ***syncedValue*** object



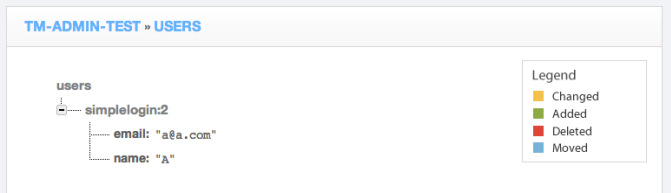
b) here is the ***messages*** object (which is a firebase kind-of-array, based on a name/value pair. The name is the ***node*** text (for example_ -JGoCMZRRGo1WQHmYxmc*) and the value is the child _text* node value (for example _test _)):



c) There was a new ***users*** node/object in the root of the app's data:

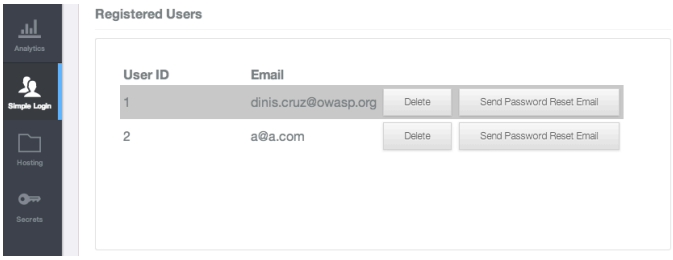


... which contains the user provided (and 'editable by that user') data (*email* and *name*)



And where is the user's credentials?

The user created was added to the current list of 'registered users' , and it must be stored somewhere inside firebase servers since we don't have access to it (hopefully it is stored using a nice strong hashing algorithm)



Wrapping up

**

**Kudos to the Firebase team for providing such easy-to-use hosting solution (next step for me is to try to use it in the TeamMentor PoC I'm currently working on)

**

**For reference the AngularJS+Firebase website that was created by the Firebase cli tool, is the one available at https://github.com/firebase/angularFire-seed (which contains a nice **README.md **file with more details)

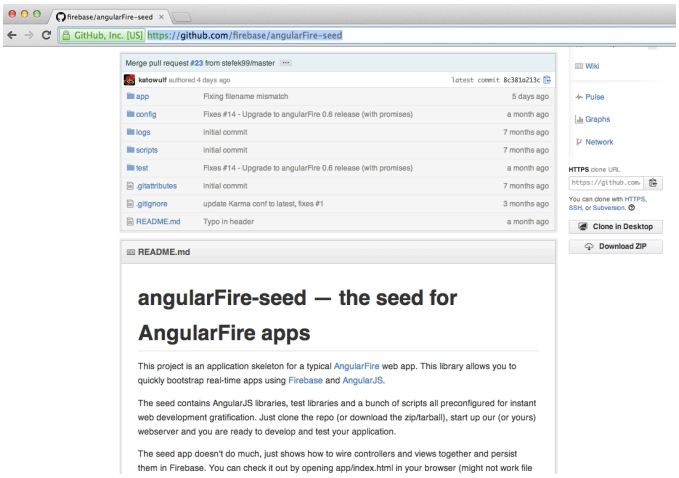


Table of Contents | Code

4 Misc Tricks

This section has the following chapters:

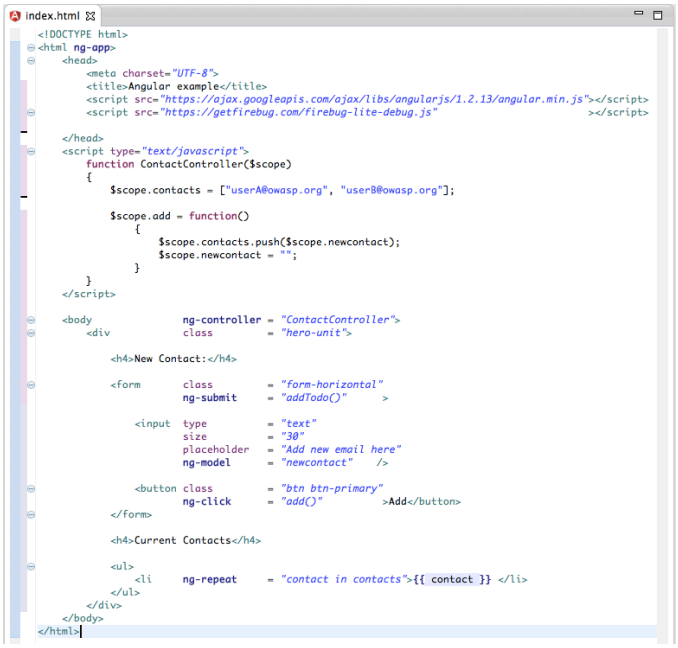
Table of Contents | Code

4.1 Programatically changing an AngularJS scope variable and adding Firebug Lite to an AngularJs app

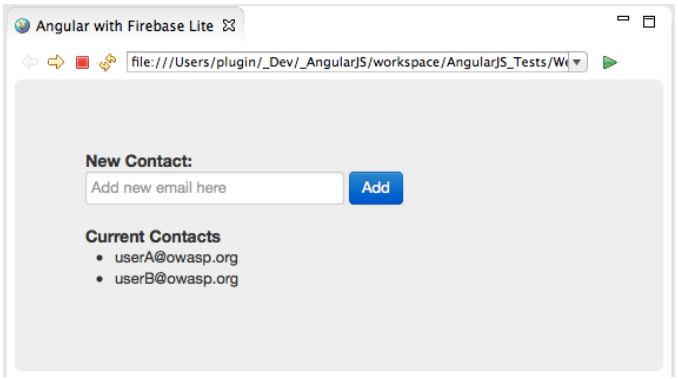
In this post I'm going to show two really nice tricks that help when developing AngularJS applications:

- adding Firebug Lite to the current browser
- changing the scope value outside a normal AngularJS controller, service or module

Let's say that we are inside Eclipse and have this simple AngularJS app (gist here)

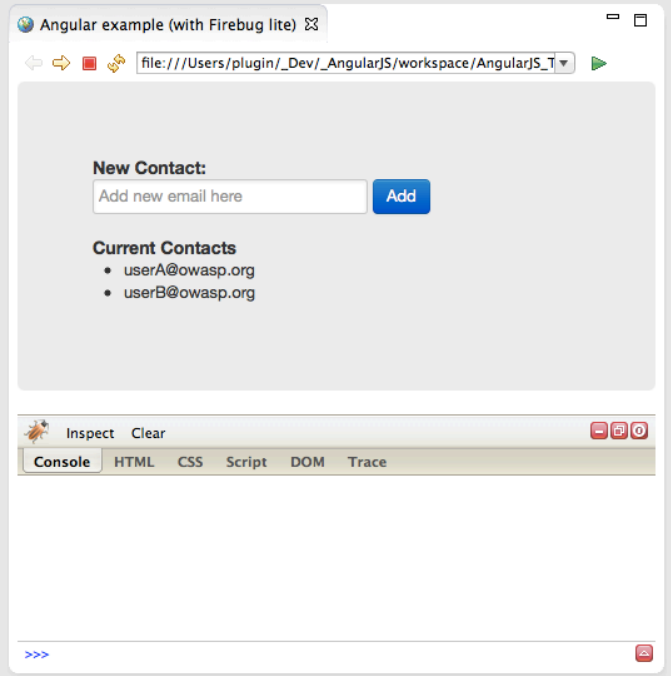


... which looks like this when executed:



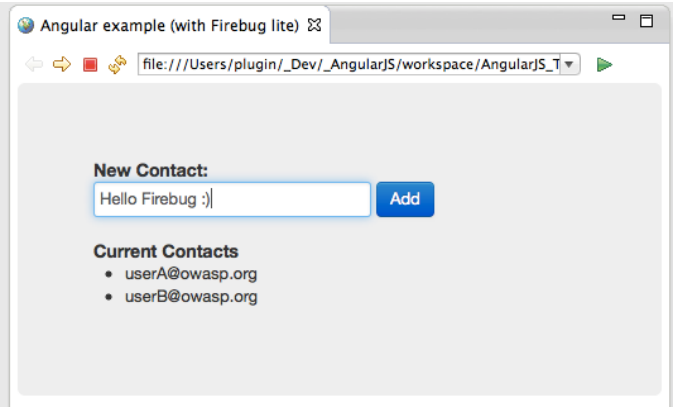
To add Firebug Lite to this page, all we need to do is to add a script reference to ***https://getfirebug.com/firebug-lite-debug.js***

... and after refresh we will have a nice Firebug Lite console (and other nice goodies) at the bottom of our page :)

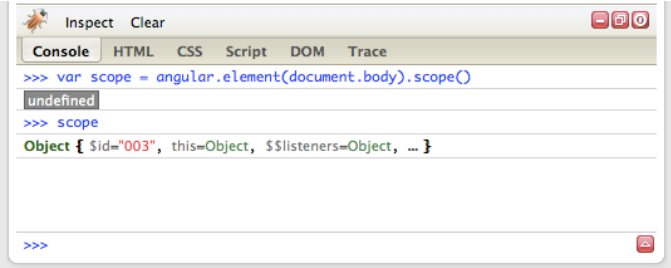


Next lets see how to access and change the AngularJS ***$scope*** of the ***ContactController***.

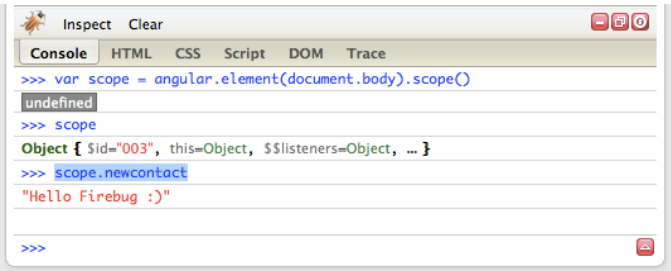
The objective is to access programatically the ***New Contact*** value (set below to ***Hello Firebug :)***)



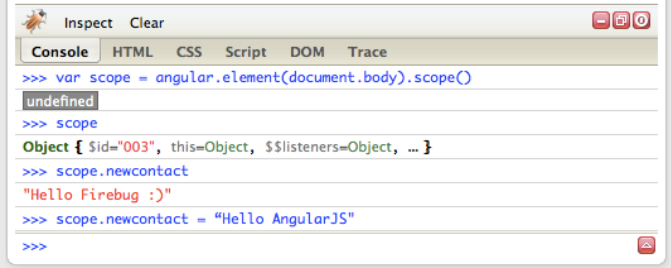
In Firebug Lite, we can access the scope object by executing: ***var scope = angular.element(document.body).scope()***



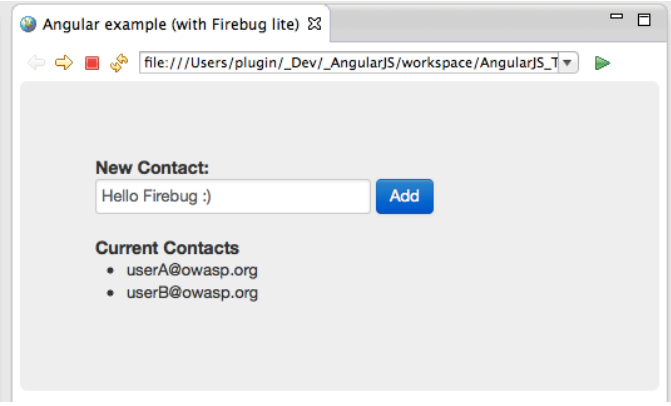
… and the **New Contact **value value using: ****scope.newcontact***



If we change the ***New Contact*** value here (in Firebug Lite) using ***scope.newcontact = "Hello AngularJS"***



… we will notice that the value will not automagically (ala AngularJS way) change in the binded (via ***ng-model***) input field



The reason that didn't happen is because the change was done outside the AngularJS $digest cycle.

The solution is to call the ***scope.$apply()*** function (and the input field will be updated):

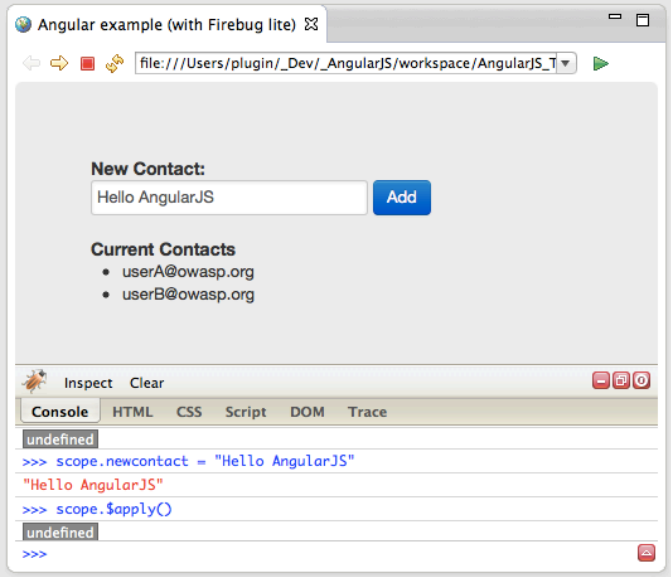
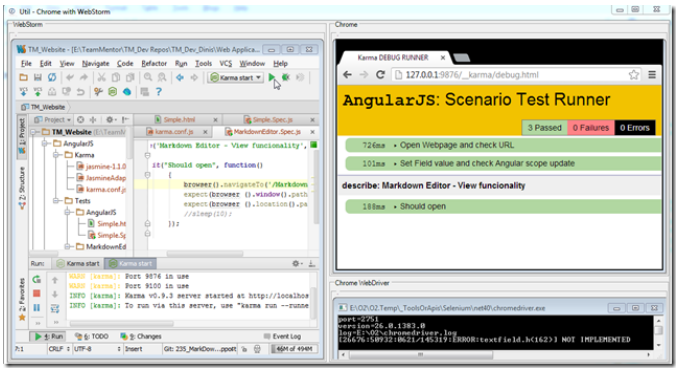


Table of Contents | Code

4.2 Hubspot current.js code includes JQuery on it

Although I'm using Angular.js on the HubSpot TBot page (see Submitting TM users to HubSpot via TBOT interface (using Angular JS)) I'm still more comfortable/knowledgeable in jQuery, so I decided to use it to populate the HubSpot fields.

So my first action was to load jQuery at the top of the TBot page:

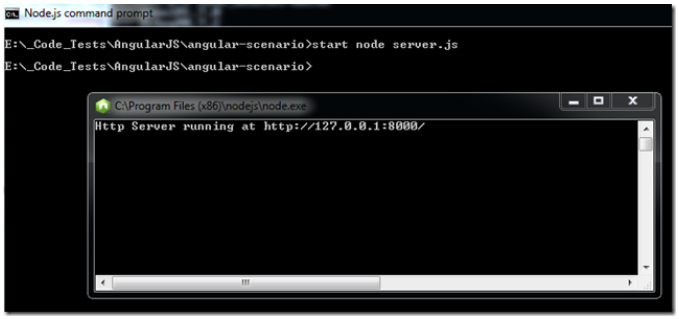


which allowed me to do this:

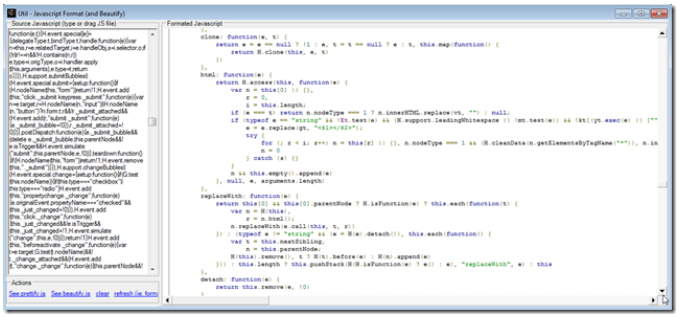
```
function TM_HubSpot($scope,$http)
    {

        var loadData = function()
            {
                userName = window.location.search.substring(1);
                if (userName !== "")
                {
                        $scope.userName = userName;             ;
                        $http.get('/rest/user/' + userName).success(function(data)
                            {
                                _data = data;
                                $scope.userData = data
                                $(".hs-input[name='firstname']").val(data.FirstName);
                                $(".hs-input[name='lastname']").val(data.LastName);
                                $(".hs-input[name='email']").val(data.Email);
                                $(".hs-input[name='company']").val(data.Company);
                                $(".hs-input[name='jobtitle']").val(data.Title);
                                $(".hs-input[name='country']").val(data.Country);
                                $(".hs-input[name='state']").val(data.State);
                            });
                }
                else
                {
                    $scope.userName =  "No username provided";
                }
```

(note the mix of JQuery and AngularJS code)

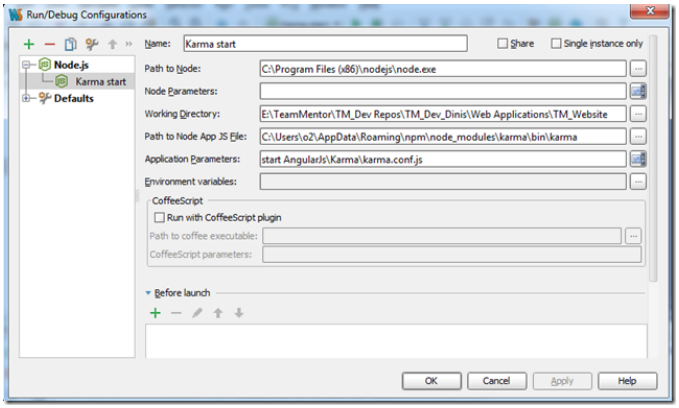
But then as I looked through the http://js.hubspot.com/forms/current.js file (which you will also notice that for TM, I saved it on the */_Customizations* folder)



In the ***Util - Javascript Format (and Beautify).h2*** O2 Platform tool:

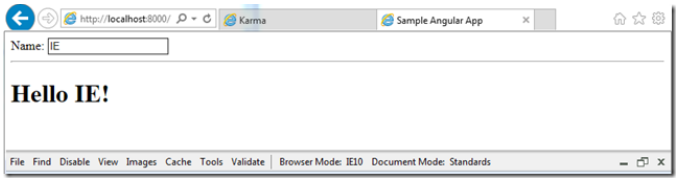


I noticed that they had embedded ***jQuery _**in the **_current.js*** code, namely at the ***window.hsJQuery*** variable

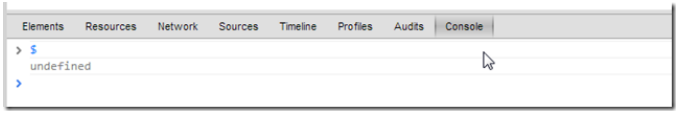


this means that we can use jQuery to access the page's DOM

Now, if we remove the jQuery script import:

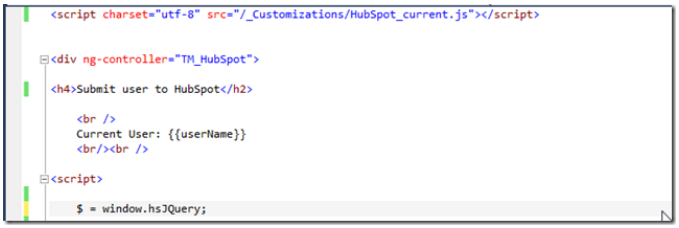


the ***$*** variable will not be available:



But if we assign ***$*** to ***window.hsJQuery***, we are back in action:

Finally, we add that assignment to the TBot page



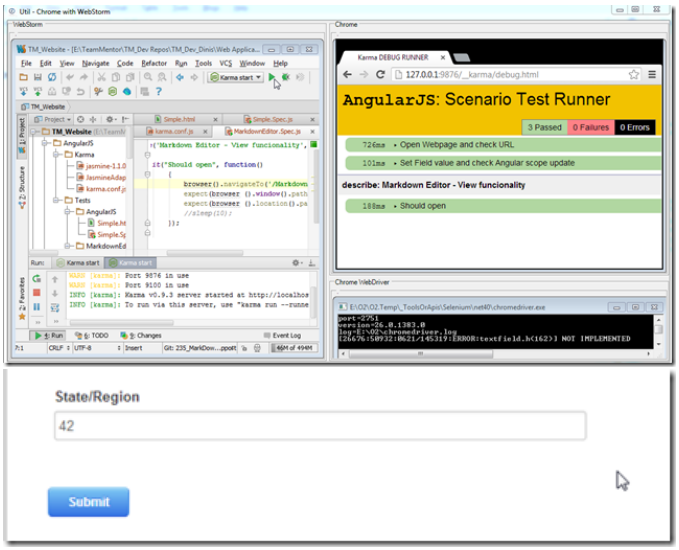
And the previous code (that used the ***$***) is working again (now using the JQuery version from ***current.js***)

Table of Contents | Code

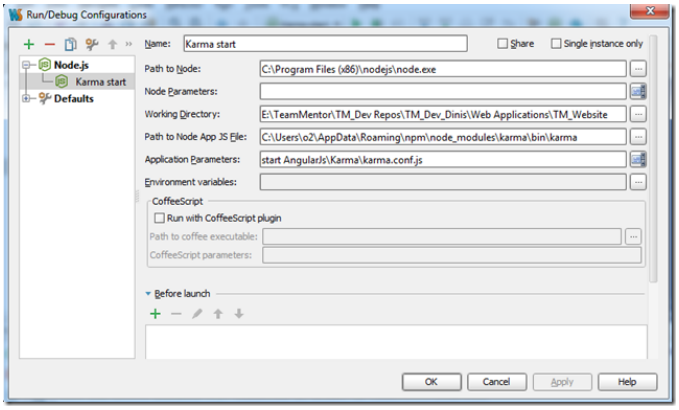
4.3 Submitting TM users to HubSpot via TBOT interface (using Angular JS)

Following the need to submit TM new users to HubSpot, I just created an TBot razor page (TM admin script) that uses Angular JS to get data about a particular user and populates a form that can then be submitted to HubSpot.

Here is what the Form looks like for the admin user (value provided on the url):

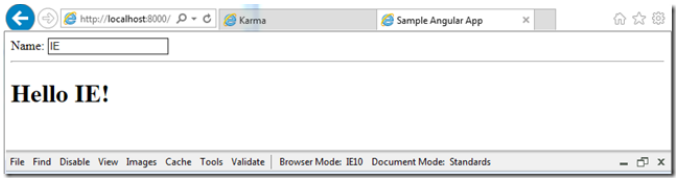


Pressing the Submit button will send the data to HubSpot which is captured like this:



This works using the new HubSpot form API and Structure (which is quite nice).

The form is created using this HubSpot provided script:



```
E:\_Code_Tests\AngularJS\angular-scenario>karma start karma.conf.js
INFO [karma]: Karma v0.9.3 server started at http://localhost:9876/__karma/
INFO [Chrome 27.0.1453 (Windows 7)]: Connected on socket id 5WJJYzMys60uMjrZ_-2v
Chrome 27.0.1453 (Windows 7): Executed 1 of 1 SUCCESS (0.368 secs / 0.184 secs)
```

Which is then auto-populated using Angular.js

```
var loadData = function()
    {
        var userName = window.location.search.substring(1);
        if (userName != "")
        {
                $scope.userName = userName;                ;
                $http.get('/rest/user/' + userName).success(function(data)
                    {
                        _data = data;
                        $scope.userData = data
                        $(".hs-input[name='firstname']").val(data.FirstName);
                        $(".hs-input[name='lastname']").val(data.LastName);
                        $(".hs-input[name='email']").val(data.Email);
                        $(".hs-input[name='company']").val(data.Company);
                        $(".hs-input[name='jobtitle']").val(data.Title);
                        $(".hs-input[name='country']").val(data.Country);
                        $(".hs-input[name='state']").val(data.State);
                    });
        }
        else
            $scope.userName =  "No username provided";
    }
```

Table of Contents | Code

5 IDEs

This section has the following chapters:

- Eclipse Groovy REPL script to sync a Browser with file changes
- Eclipse Groovy script to remove the 'busy' image from the WebBrowser Editor
- Using Chrome inside a native VisualStudio pane (using Window Handle Hijacking)
- Using WebStorm with Chrome and ChromeDriver (to view KarmaJS execution results)
- When the best way to automate Chrome is to use ... Chrome
- Adding KarmaJS support to WebStorm to automagically run tests on file changes

---

Table of Contents | Code

5.1 Eclipse Groovy REPL script to sync a Browser with file changes (with recursive folder search via Java's WatchService)

Since I am using Eclipse to develop using AngularJS (see Creating an Eclipse UI to run AngularJS e2e tests using Karma), I needed a way to refresh the browser window every-time I made changes to any AngularJS related file (note that due to the nature of the AngularJS projects, I need the change to trigger on any change made inside the root folder and all its subfolders).

Since there didn't seem to be an easy way to do this (*'auto browser refresh on file changes'*) in Eclipse, I used the Eclipse Grovy REPL Scripting Environment to develop a script/macro that:

- Based on a title of an opened eclipse editor file:
- ... find the full path of that file, and:
- ... create aJava WatchService that monitors the file's folder and subfolders, and:
- ... when a StandardWatchEventKinds.ENTRY_MODIFY is received :
  - Create/Open a new Eclipse view with a browser (called ***Synced Browser***), and:
  - ...refresh the index page

For reference here is the groovy code for this script (gist here):

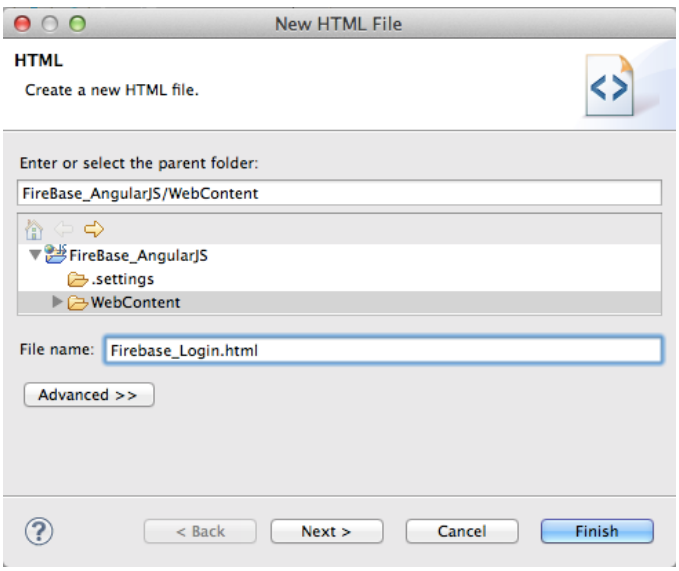
Originally I had tried to use Eclipse file change events (like on this SO thread), but that didn't work as well as the WatchService.

A next step is to create a mini UI to allow the configuration of the target files (maybe as a view added to the next version of the Groovy REPL Eclipse plugin)

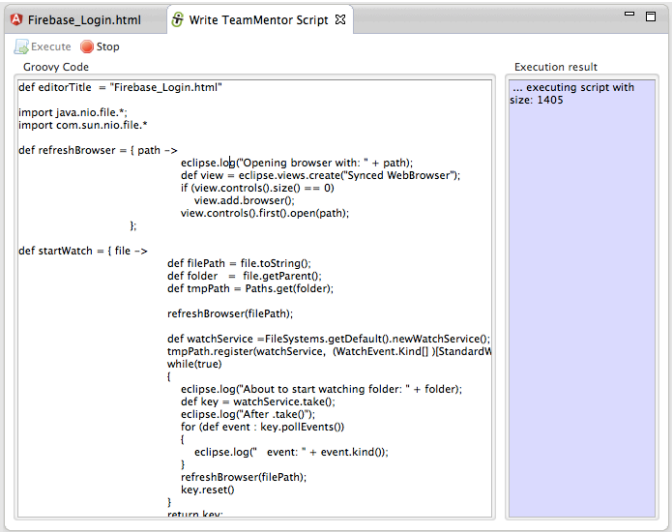
Seeing it in action
**

**Here is how to test this script:

1) create a Web project with an Html file on it:



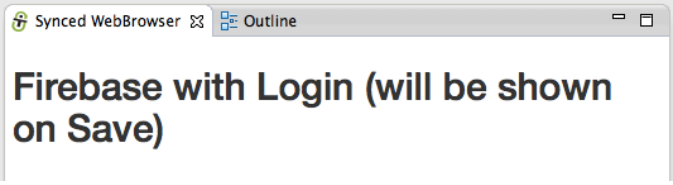
2) run the Groovy code in the REPL window (note that the image below is using a different root file and the version of script is an older one (which didn't contain the recursive folder monitoring)):



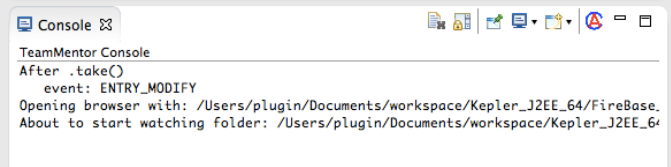
... on execution you will see a new view (called *Synced WebBrowser*) show up in your current Eclipse instance.

3) make some changes on the Html file

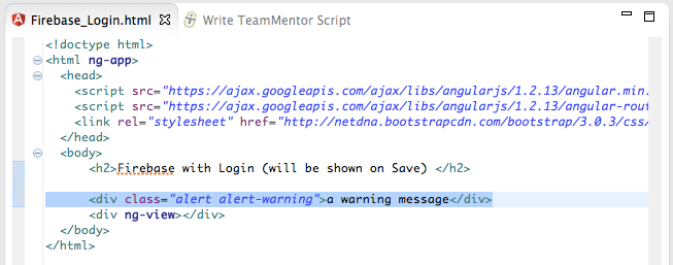
4) and note that the ***Synced WebBrowser*** view will be refreshed automatically (it takes about 500ms to 1s for the change to be picked up (see this SO answer for why I had to use the ***SensitivityWatchEventModifier.HIGH*** setting on the WatchService))



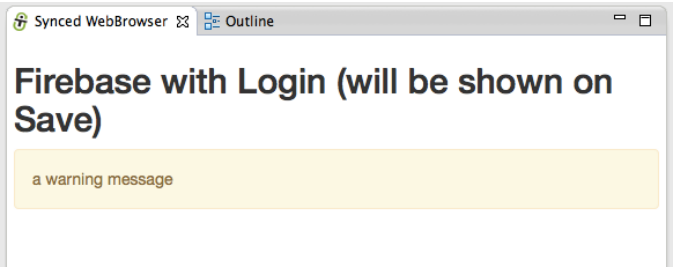
5) if you open the ***TeamMentor Console***, you will also see a number of log messages that help to see what is going on:



6) here is another example where I added a new Bootstrap css div:



7) which was immediately (~500ms) shown on save



8) note that the the log message shows the events being triggered and the resetting of the ***WatcherService***:

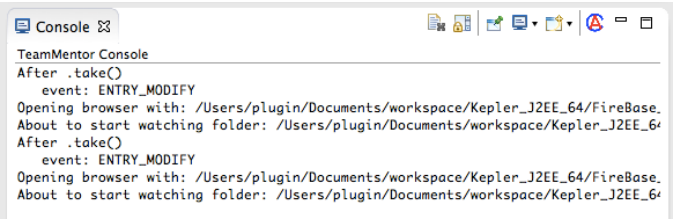


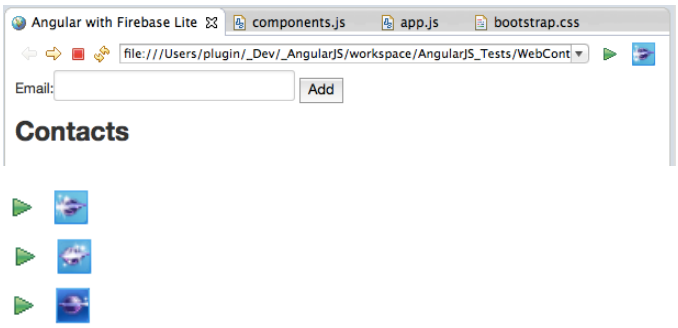
Table of Contents | Code

5.2 Eclipse Groovy script to remove the 'busy' image from the WebBrowser Editor

Now that I'm doing AngularJS and Firebase development inside Eclipse, there was a little 'thing' that was starting to drive me crazy: ***The animation icon on the top right of the Eclipse WebBrowser!***
**

Apart from the mosaic 2000s look (which is kinda ok), there is a big problem with pages that keep running for a while: ****the animation doesn't stop!*

_
_This means that if you are developing inside Eclipse, there is this 'thing' (i.e. the top right icon) that keeps moving and demand my brain's attention:

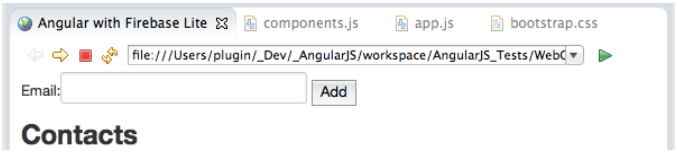


Since there didn't seem to be an preference or option to disable this behaviour, it was time to open up the Eclipse Groovy REPL Scripting Environment and fix it with a script :)

The solution

After a quick prototyping, I come up with this script to remove all icons from all opened WebBrowser editors (gist here):

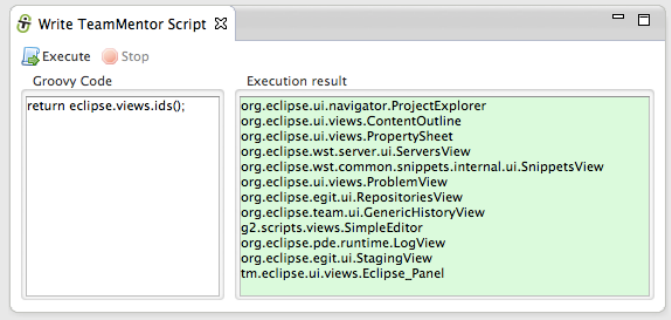
After execution the icon is not there anymore :)



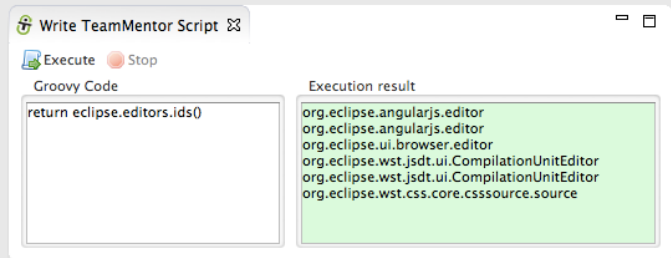
How I found the solution

Here are the steps I took to find the _busy _object to set the ***visible*** value to false

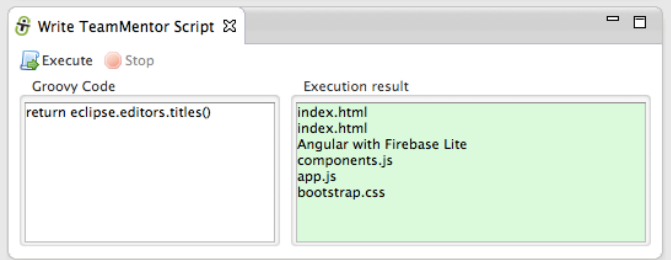
1) in the Groovy REPL I started by getting a list the ***ids*** of all current opened Eclipse views, but there didn't seem to be any web browser in there:



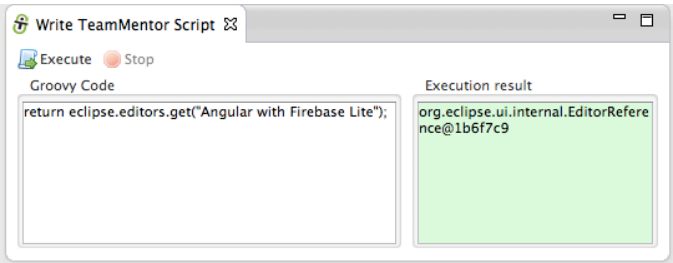
2) then I looked at the list the ***ids*** of all current opened editors, and found one that sounded promising: **org.eclipse.ui.browser.editor**



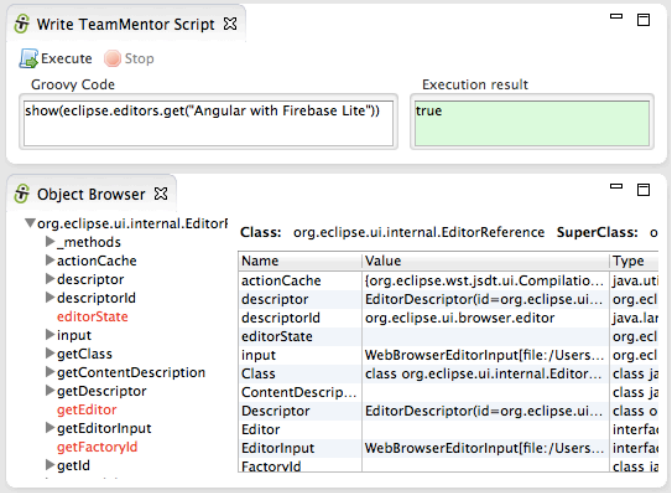
3) to confirm that that was the one I wanted, I looked at the ***titles* **of the current editors, and confirmed that the browser window I want to capture was there (the title was "*Angular with Firebase Lite*");



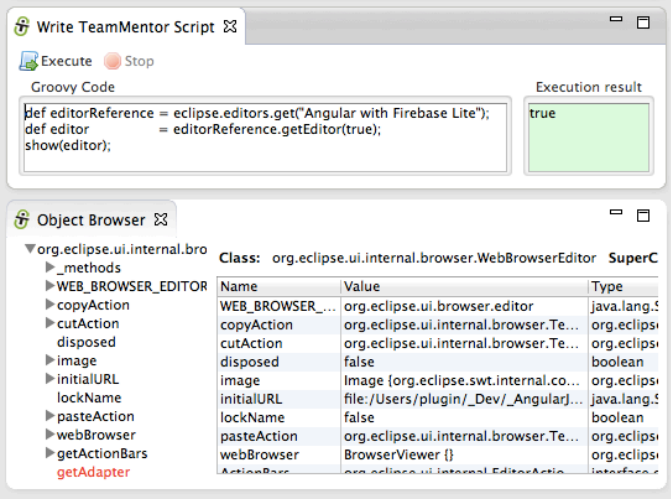
4) now that I knew the title, it was easy to get an ***EditorReference*** to it:



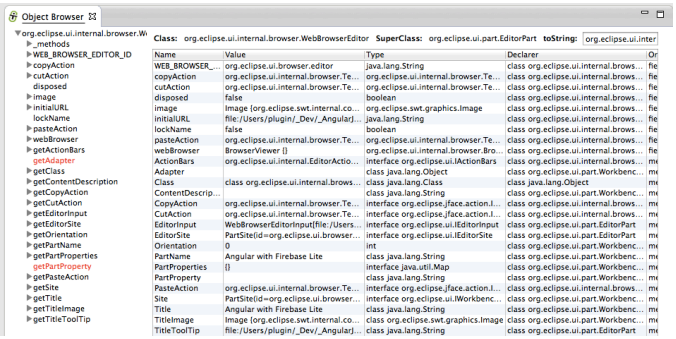
5) everytime I have an object reference that I want to take a look, I call the _show({object}) _viewer, since that will give me a nice ***TreeViewer*** of all methods, fields and properties (see Viewing Eclipse and SWT objects (Workbench, Display and Shell) using Groovy's ObjectBrowser and using TeamMentor's Plugin ObjectBrowser for more details how this works)



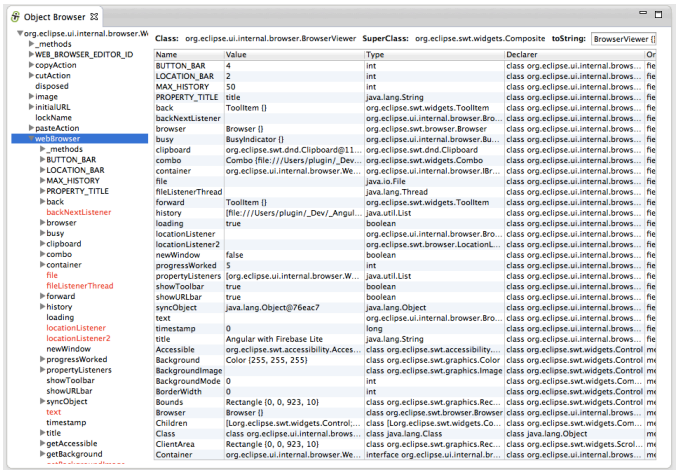
6) in this case we don't want the _EditorReference _object. What we really want is the actually editor, which can be retrieved by invoking the ***getEditor(true) _**method (note how the object we are currently seeing in the **_Object Browser*** is the ***org.eclipse.ui.internal.WebBrowserEditor***)



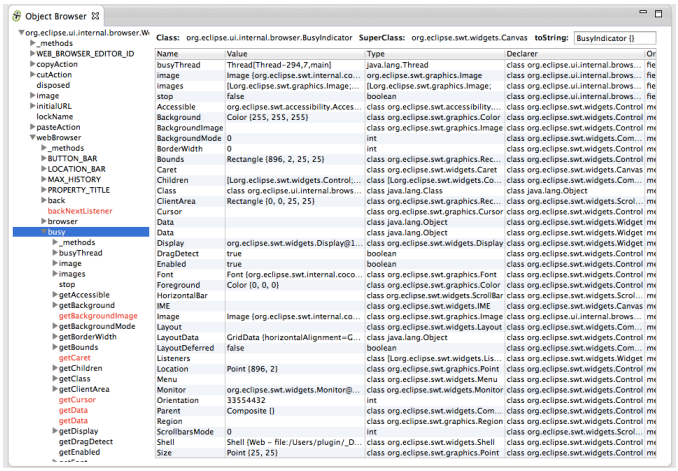
Here is a better view of this object:



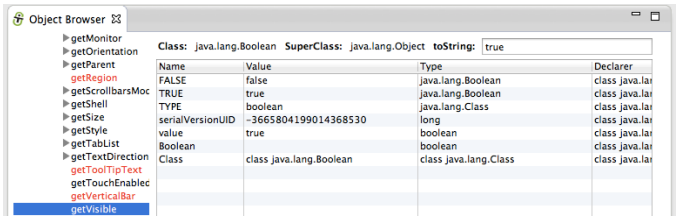
7) looking at the **org.eclipse.ui.internal.WebBrowserEditor **object, the one that called my attention was the ****webBrowser*** field (which is an ***org.eclipse.ui.internal.browser.BrowserViewer*** object)



8) Inside the ***webBrowser*** object I found the ***busy*** field (which is an ***org.eclipse.ui.internal.browser.BusyIndicator*** object)

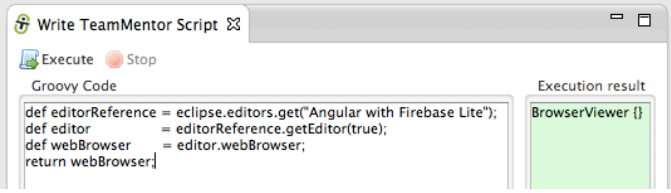


9) and finally inside the** busy **object I found the ****visible*** property (ie the ***getVisible*** and **setVisible **methods)

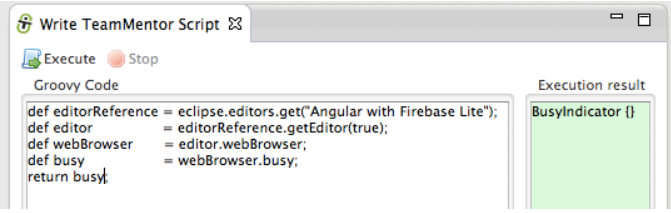


10) back in the REPL, it was time to do the same thing programatically.

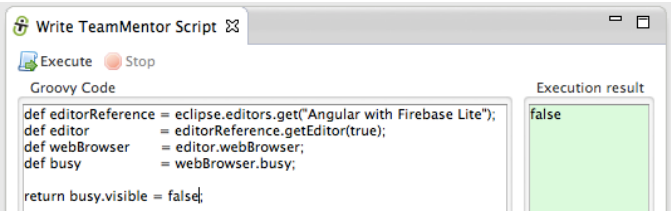
First I got a reference to the ***webBrowser*** field:



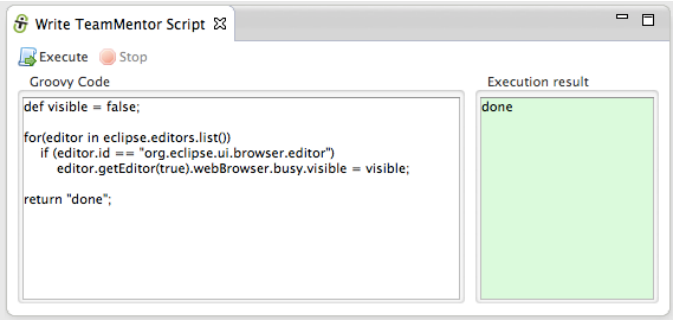
... and then the ***busy*** field:



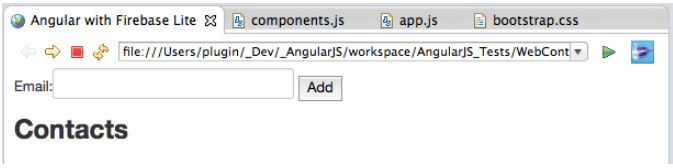
11) Once I had the reference to the ***busy*** field, it was easy to make it invisible (by just setting the ***visible*** property to ***false***)



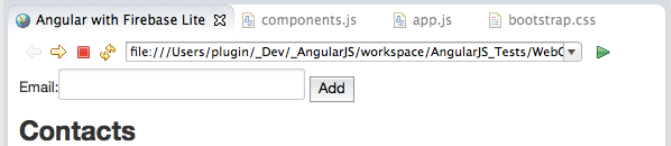
12) final step was to write a generic script that would work on all opened browser editor windows (a robust script would use the same trick I used in the integration with the Fortify plugin, where a callback is received on every new view opened (i.e. we could wait for new instances of the **org.eclipse.ui.browser.editor **to be opened, and apply the fix then))



To demonstrate the script in action, here it is the browser before:



... and here it is after:



Note: another way to to stop the constant animation was to just set the ***stop*** value to *false*, but this would only work for the current opened page (i.e. the animation would be back on page reload)

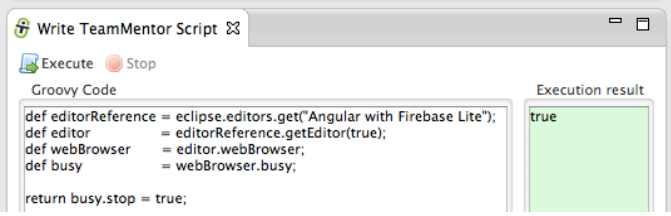
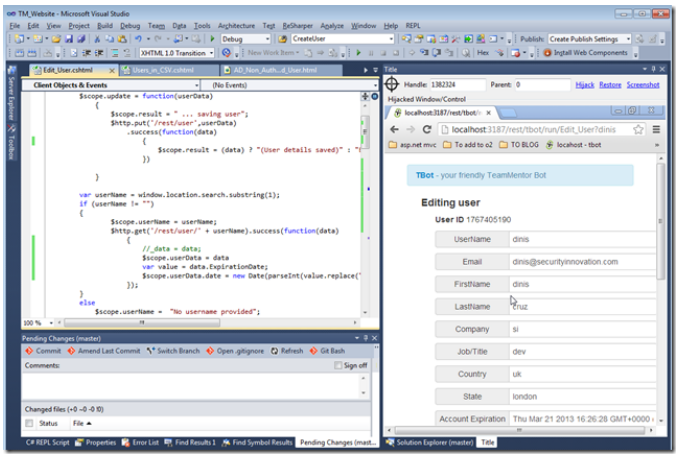


Table of Contents | Code

5.3 Using Chrome inside a native VisualStudio pane (using Window Handle Hijacking)

To help me debug and visualize an AngularJS page I was developing, I used the O2's Window Handle Hijack technique to insert an Chrome window inside VisualStudio 2010.

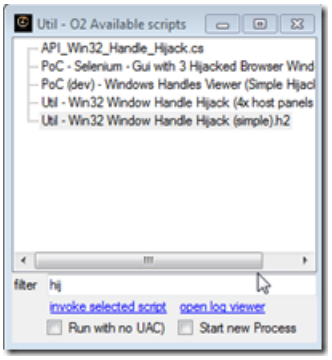
Here it is in action:



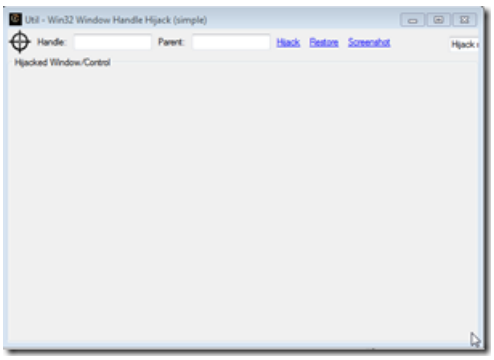
On the right you can see a full chrome window, inserted inside a VisualStudio dockable pane.

On the left you can see the AngularJs file (rendered from a RazorSharp template) that I can edit and quickly view its output on the right-hand-side Chrome window (with no web recompilation needed)

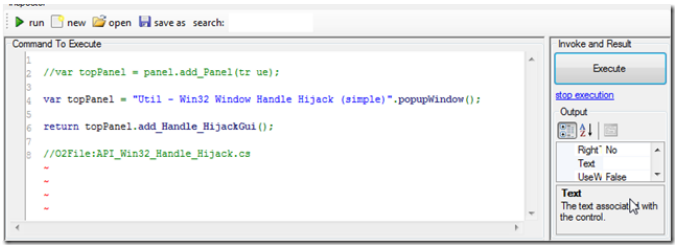
To create this, I searched in O2 Platform for the ***Util - Win32 Window Handle Hijack (simple).h2*** script



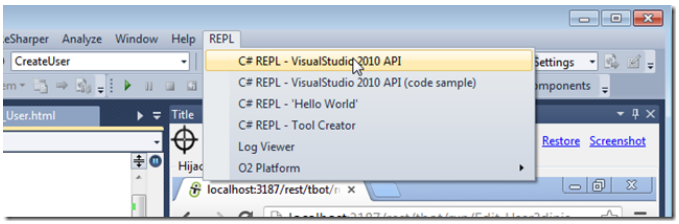
which looks like this:



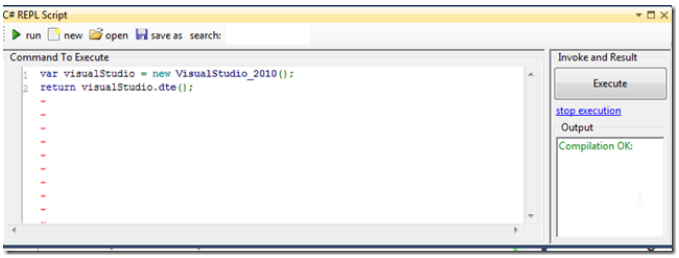
Had a look at its source code to see how it was created (note the Extension Method ***add_Handle_HijackGui***):



Then inside a VisualStudio, I opened a C# REPL script:

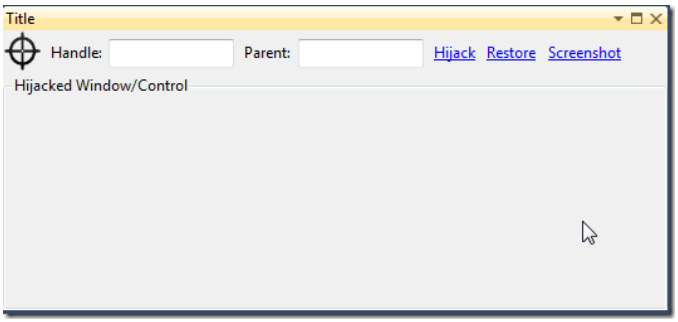


This gave me access to the ***VisualStudio*2010_** API

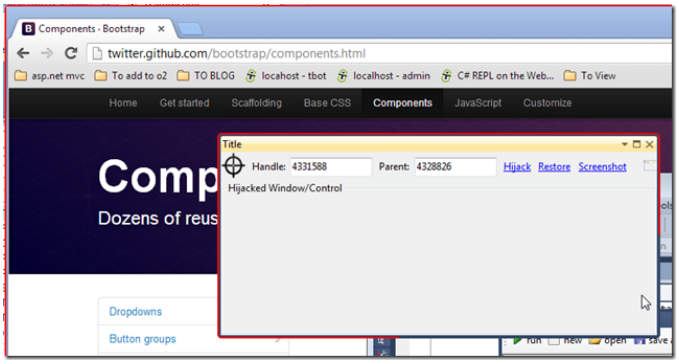


where I can use the Extension Method ***add_Handle_HijackGui***

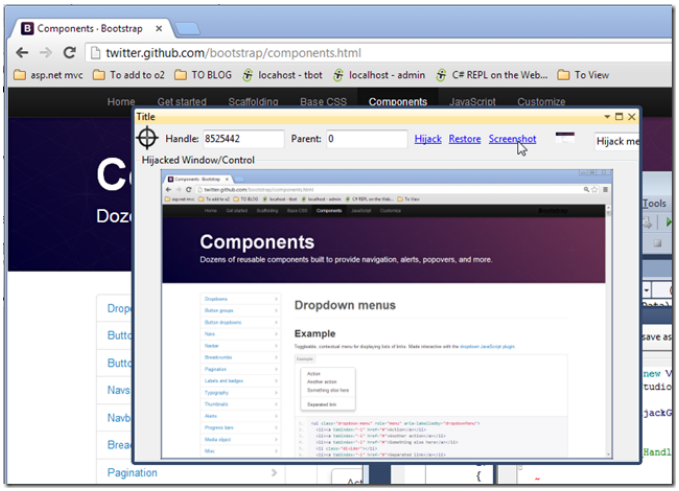
to create a native VisualStudio pane with the ***Windows Handle Hijack Gui***



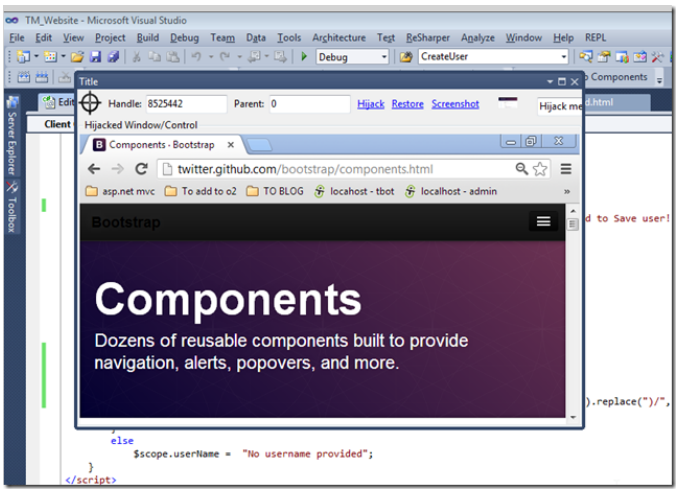
With this GUI, we can grab any Window's Window, by dragging the target icon (top left) into the window we want to use/hijack:



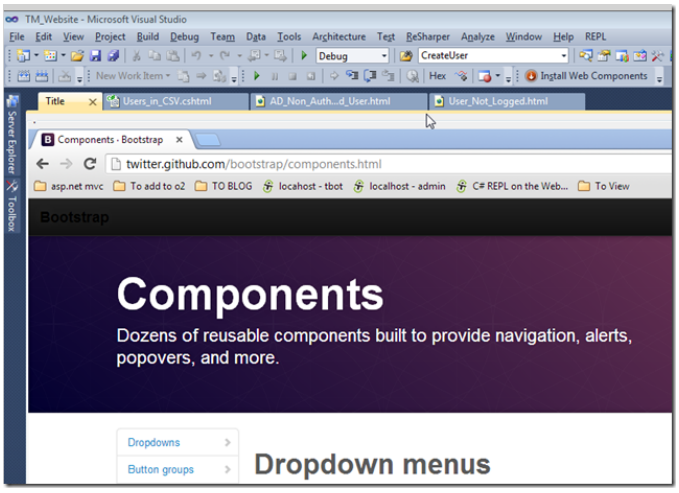
Tip: before Hikacking a window, it is a good idea to take a screenshot and see if we have the right one:



Once we're sure, just hit the **_Hijack** _link, and we will have have a fully functional Chrome window that we can place anywhere inside VisualStudio's GUI.



For example, we can place it in the documents area as one of the source code files
(tip: double click on the 'Hijacked Window/Control' text to hide the hijack controls)



As a final example, here is what it looks like if we just Hijack the browser's website window (without the navigation and top bars)

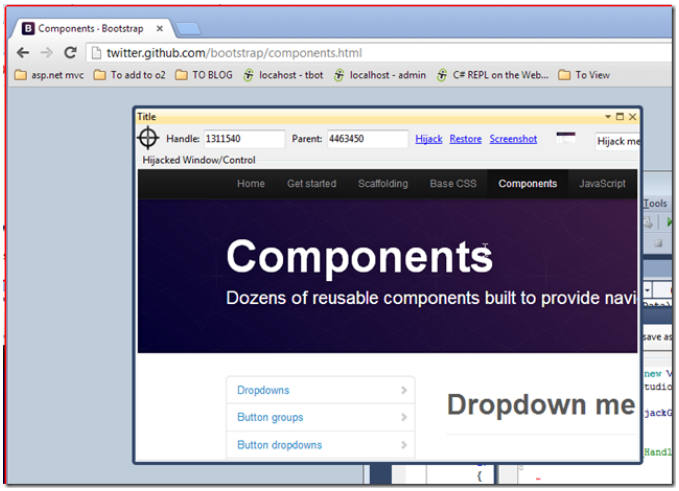
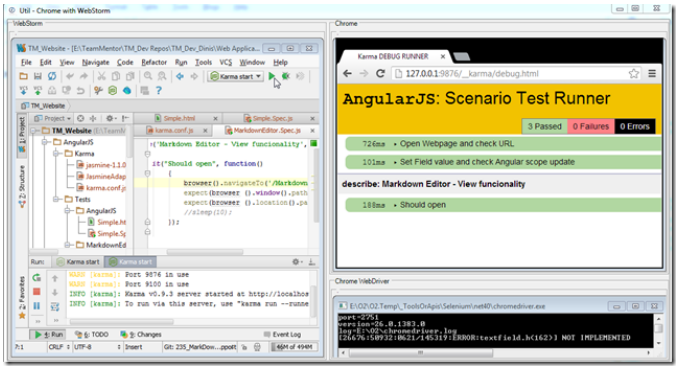


Table of Contents | Code

5.4 Using WebStorm with Chrome and ChromeDriver (to view KarmaJS execution results)

Following from the example described inWhen the best way to automate Chrome is to use ... Chrome , here is a more practical scenario where I'm creating a GUI that has both WebStorm and Chrome running side-by-side

Here is what it looks like:



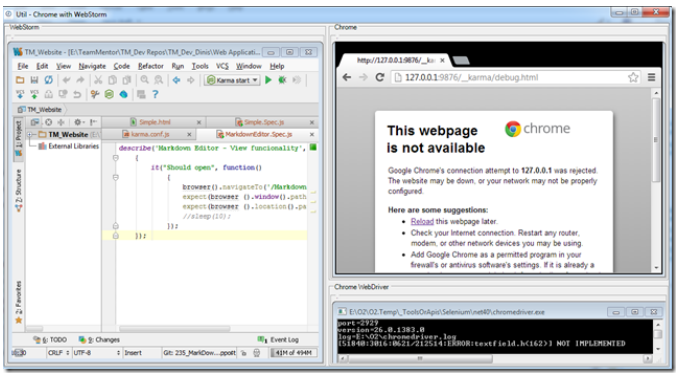
What makes this example practical is the KarmaJS auto execution on file change, just like it was described in Adding KarmaJS support to WebStorm to automagically run tests on file changes

Here is the code that creates this GUI (with some functionality since the last example, but it still needs to a bit of work to make it more robust):

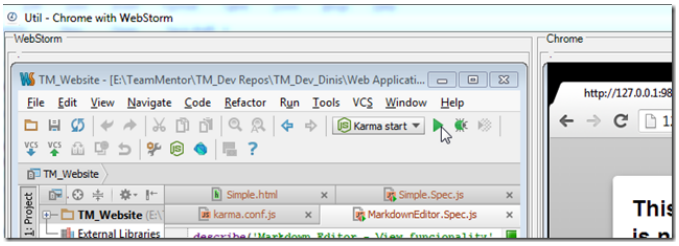
```
1  //var topPanel      = panel.clear().add_Panel();
2  var topPanel  = "Util - Chrome with WebStorm".popupWindow(1200,600);
3  var webStormPanel = topPanel.add_GroupBox("WebStorm").add_Panel();
4
5  //we need to stop any existing WebStorm processes
6  foreach(var process in Processes.getProcessesCalled("WebStorm"))
7  process.stop().WaitForExit();
8
9  // start webstorm
10 var webStorm = @"C:\Program Files (x86)\JetBrains\WebStorm 6.0.2\bin\WebStorm.exe".startPr\
11 ocess();
12
13
14 var chromeHijack = new API_Chrome_Hijack().open_ChromeDriver()
15 .add_Chrome_To_Panel(topPanel.insert_Right());
16 //.add_WebDriver_ScriptMe_To(replPanel);
17
18 //wait for main window handle
19 webStorm.waitFor_MainWindowHandle();
20 //hack to deal with the splash screen
21 3000.sleep();
22
```

```
23 //hijack WebStorm window
24 chromeHijack.hijack_Process_Into_Panel(webStormPanel,webStorm);
25
26 chromeHijack.ChromeDriver.Navigate().GoToUrl("http://127.0.0.1:9876/__karma/debug.html");
27
28
29 //O2File:API_Chrome_Hijack.cs
30 //O2Ref:WebDriver.dll
```

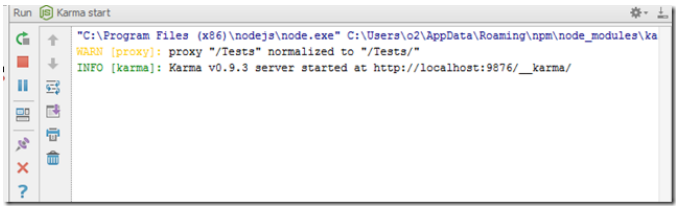
When executed for the first time, this GUI looks like this:



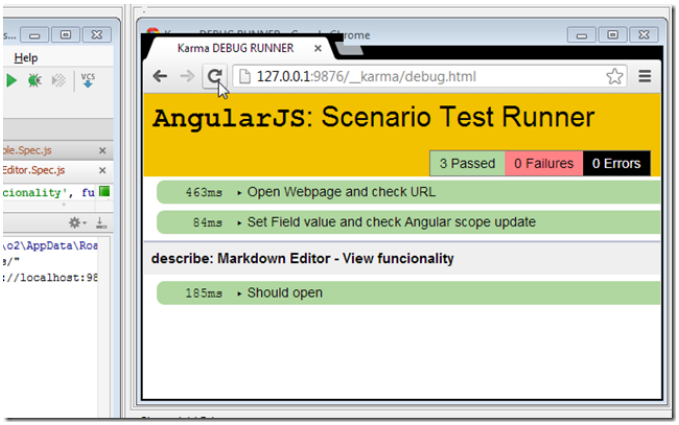
Them if we start KarmaJS:



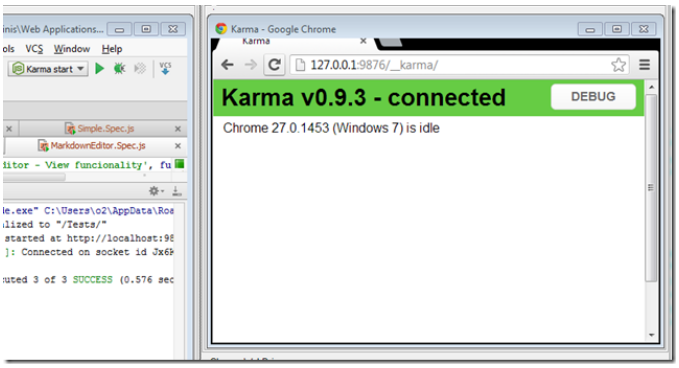
... a listener will start (waiting for browsers to connect):



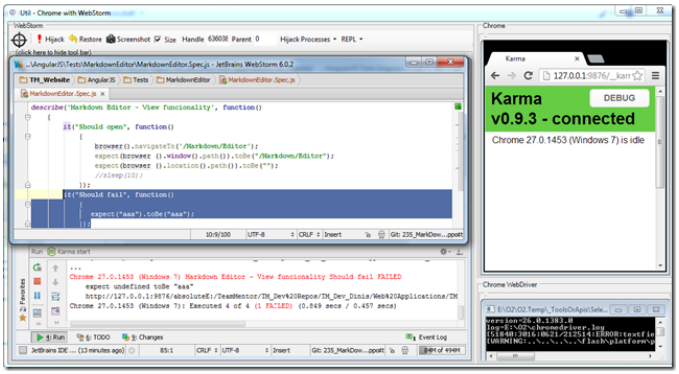
Refreshing the browser will 'capture it' and manually execute the tests:



Opening up the normal page will start the 'auto test execution loop':



... and if we add a new test and save the file, the unit test execution will occour



Note: there is some kind of weird event race-condition happening and the keyboard key's events are not being fired inside the captured WebStorm. So in the case shown above, I open the file to edit in a WebStorm popup window, where the keyboard events fire-up ok (and I was able to make the changes to the unit test file)

5.5 When the best way to automate Chrome is to use ... Chrome (with examples on Google search, direct AngularJS scope manipulation and ChromeDriver javascript access)

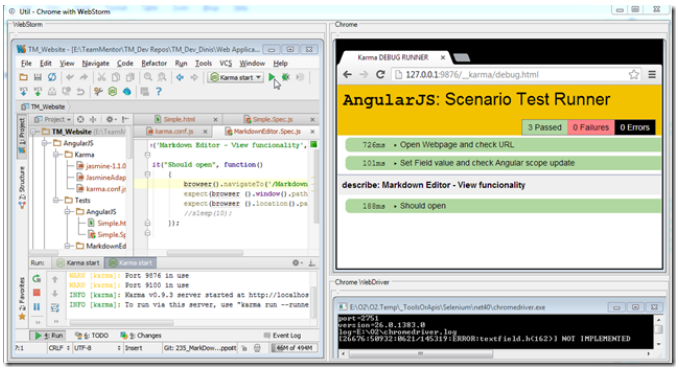
On the topic of Web Automation, I always wanted to have a REPL environment for Chrome like I have for IE (using Watin).

In the past I have explored multiple solutions, for example the use of CefSharp (see here and here). But that was never the real thing, and there was always a couple issues (caused by the fact that the 'real' chrome wasn't being used).

For a while, in the back on my mind the solution was simple and obvious: ***Use the real Chrome process in a way that it can be programmatically accessed from an O2's C# repl environment!***

Well, the good news is that is exactly what I have done :)

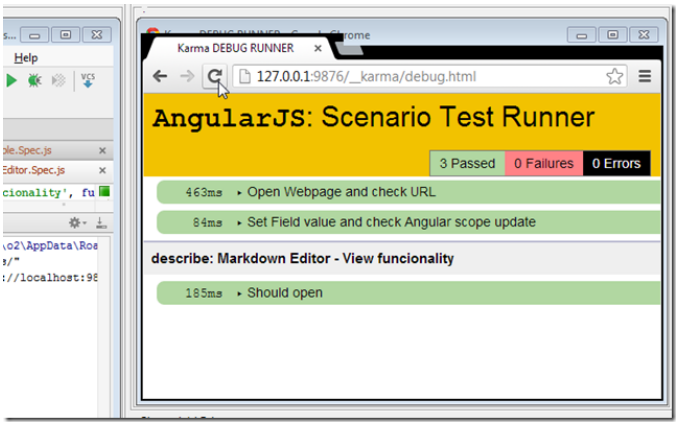
I just created the Gui you can see below, which uses the Window-Hikacking technique to inject an (Selenium's ChromeDriver started) Chrome process' window in a Panel, and pass its reference (as a variable) to an O2 REPL environment.



The script is called_ **Util - Chrome Browser with REPL.h2**_ and it has:

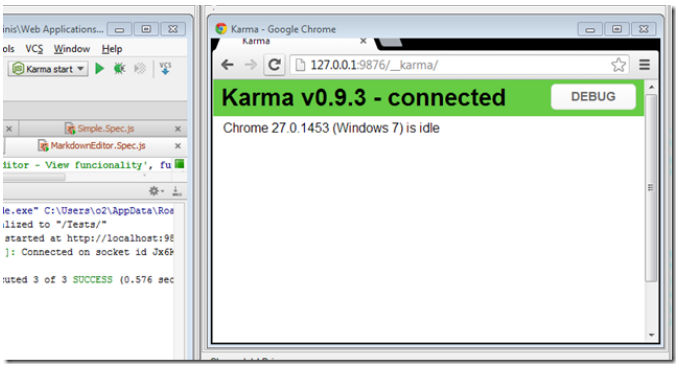
- The **cmd.exe** window from the **ChromeDriver.exe** process in the bottom right
- The **Chrome** window (started by the ChromeDriver) in the top right
- A **C# REPL** (with the CromeDriver object passed in as a parameter) in the left

Now, on the left-hand-side C# REPL, I can write a script like this:



Which will:

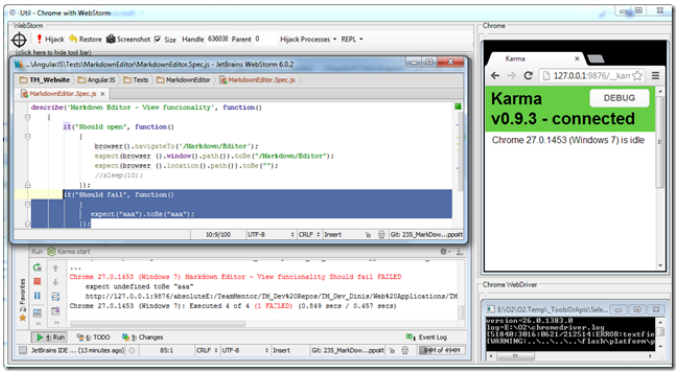
- Open (in Chrome) the http://www.google.com web page
- Find the 'q' element (the search text input)
- Send the Keys 'O2 Platform' to that element
- Find the element 'btnG' (the search button)
- Click on that element



Automating an AngularJS page

**

**The next example is more interesting where we are going to open AngularJS page and programmatically change a $scope variable

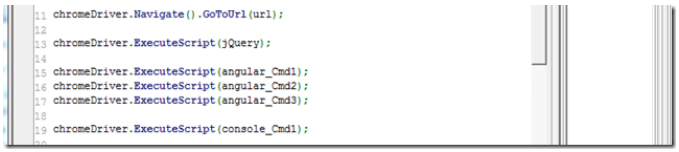


Lets look at the code to see what is going on.

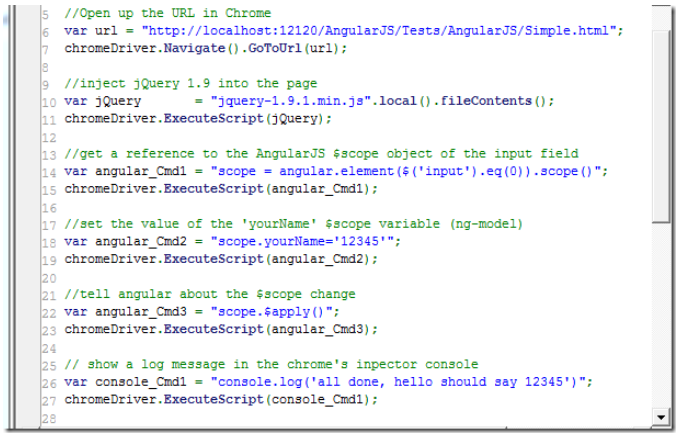
The first line opens up an the Url with the sample AngularJS example described here and here

... next I create a number of variables with the url and commands to execute:

... here:



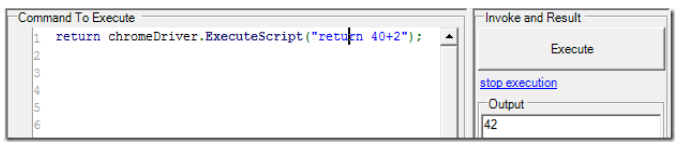
It will probably be easier if I break these commands individually:



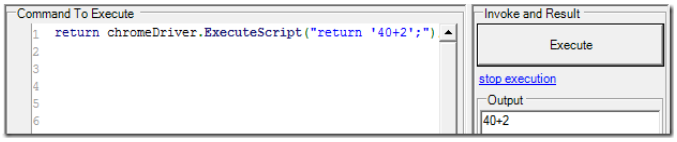
And this is all programmed in a nice REPL environment, which makes it really easy to try thing out.

For example, here are a couple interesting two way data exchange between the C# script and the Chrome javascript:

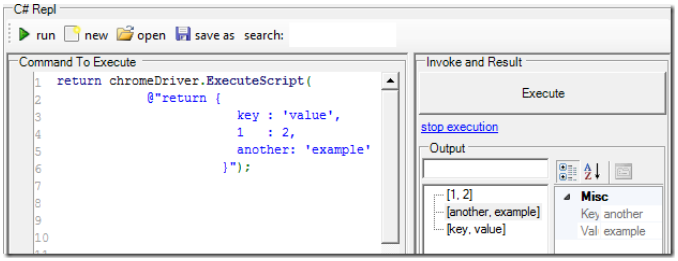
1) get an int value



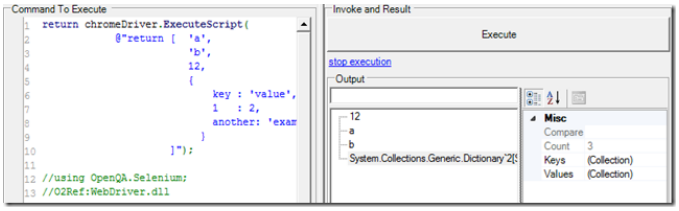
2) get an string value:



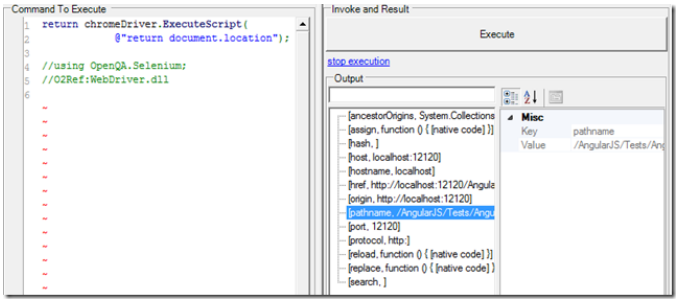
3) get an simple object (converted into dictionary)



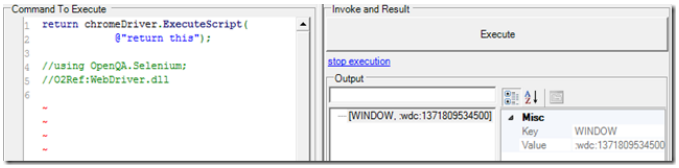
4) get an array



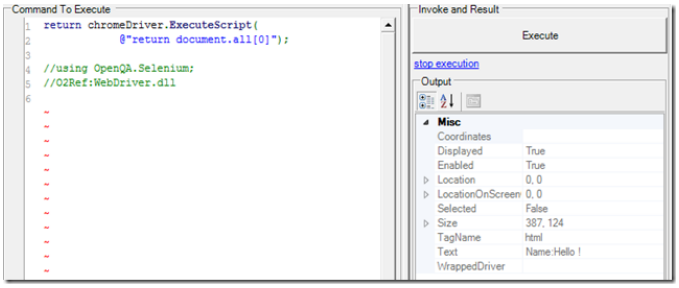
5) get the document.location object



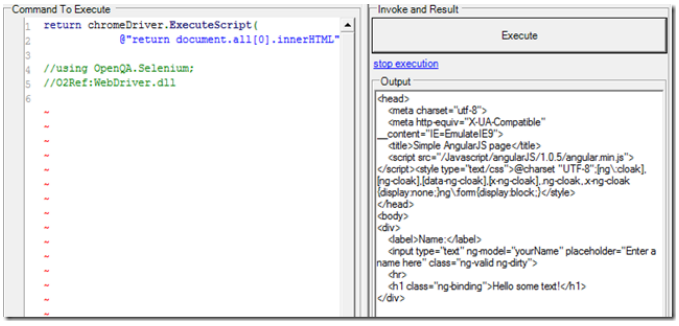
6) get the 'this' object



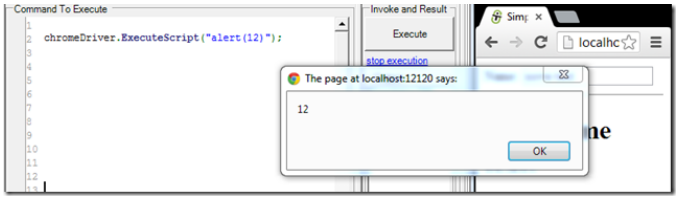
**7) get the first document.element **



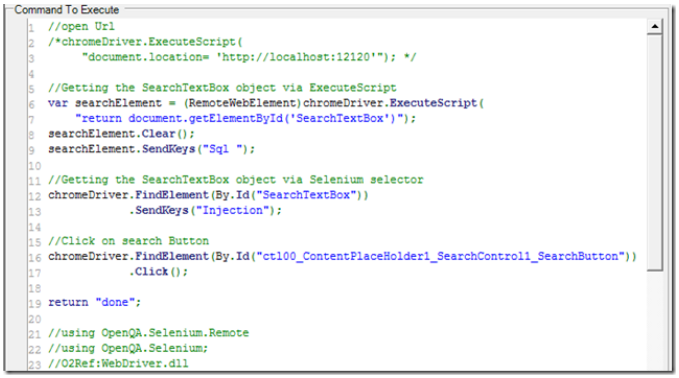
8) get the html of the first document.element



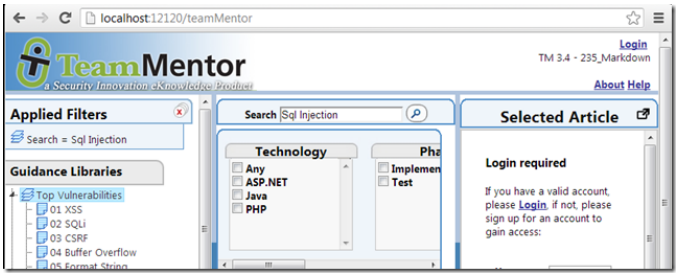
9) show an alert



10) getting and using an Html element via ExecuteScript



which will populate the search text value and click on the search icon



What is cool about all these examples is that we are running the code on the local installation of Chrome, ie it is the actually chrome process that we are using, which is great for testing, debugging and developing :)

Scripts used in this post

A) C# code that created the GUI (start chromeDriver and chrome processes and hijack they main window)

```
//var topPanel      = panel.clear().add_Panel();
var topPanel = "Util - Chrome with REPL".popupWindow(1200,600);

var replPanel = topPanel.add_GroupBox("C# Repl").add_Panel();
var chromePanel = topPanel.insert_Right(replPanel.width() / 2 , "Chrome");
var chromeDriver = chromePanel.parent().insert_Below(150,"Chrome WebDriver");

var firstScript =
@"chromeDriver.open(""http://www.google.com"");
chromeDriver.FindElement(By.Name(""q""))
.SendKeys(""O2 Platform"");
chromeDriver.FindElement(By.Name(""btnG""))
.Click();
return ""done"";

//using OpenQA.Selenium;
//O2Ref:WebDriver.dll
//O2File:API_ChromeDriver.cs";

var chromeHijack = new API_Chrome_Hijack()
.open_ChromeDriver();

chromeHijack.ChromeDriver.script_Me(replPanel).set_Code(firstScript);
var hijacked_Chrome = chromePanel.add_Handle_HijackGui(false)
.hijackProcessMainWindow(chromeHijack.ChromeProcess);
var hijacked_ChromeDriver = chromeDriver.add_Handle_HijackGui(false)
.hijackProcessMainWindow(chromeHijack.ChromeDriverProcess);

//O2File:API_Chrome_Hijack.cs
//O2File:API_Win32_Handle_Hijack.cs
```

```
31
32  //O2Ref:WebDriver.dll
33
34  **B) first code example (open Google and do a search)**
35
36
37      chromeDriver.open("http://www.google.com");
38      chromeDriver.FindElement(By.Name("q"))
39                  .SendKeys("O2 Platform");
40      chromeDriver.FindElement(By.Name("btnG"))
41                  .Click();
42      return "done";
43
44  //using OpenQA.Selenium;
45  //O2Ref:WebDriver.dll
46  //O2File:API_ChromeDriver.cs
47
48  **C) 2nd code example, open AngularJS page and programmatically change a $scope variable**\
49
50
51
52      var url          = "http://localhost:12120/AngularJS/Tests/AngularJS/Simple.html";
53
54  //var jQuery = "http://code.jquery.com/jquery-1.10.1.min.js".GET();
55  var jQuery = "jquery-1.9.1.min.js".local().fileContents();
56  var angular_Cmd1 = "scope = angular.element($('input').eq(0)).scope()";
57  var angular_Cmd2 = "scope.yourName='12345'";
58  var angular_Cmd3 = "scope.$apply()";
59
60  var console_Cmd1 = "console.log('all done, hello should say 12345')";
61
62  chromeDriver.Navigate().GoToUrl(url);
63
64  chromeDriver.ExecuteScript(jQuery);
65
66  chromeDriver.ExecuteScript(angular_Cmd1);
67  chromeDriver.ExecuteScript(angular_Cmd2);
68  chromeDriver.ExecuteScript(angular_Cmd3);
69
70  chromeDriver.ExecuteScript(console_Cmd1);
71
72  return "done";
73
74  // doesn't work to open chrome's inspector
75  // chromeDriver.Keyboard.SendKeys(OpenQA.Selenium.Keys.F12);
```

```
76
77  //using OpenQA.Selenium;
78  //O2Ref:WebDriver.dll
79  **D) last example where search filed was retrieved using two different techniques:**
80
81
82      //open Url
83      /*chromeDriver.ExecuteScript(
84           "document.location= 'http://localhost:12120'"); */
85
86  //Getting the SearchTextBox object via ExecuteScript
87  var searchElement = (RemoteWebElement)chromeDriver.ExecuteScript(
88  "return document.getElementById('SearchTextBox')");
89  searchElement.Clear();
90  searchElement.SendKeys("Sql ");
91
92  //Getting the SearchTextBox object via Selenium selector
93  chromeDriver.FindElement(By.Id("SearchTextBox"))
94  .SendKeys("Injection");
95
96  //Click on search Button
97  chromeDriver.FindElement(By.Id("ctl00_ContentPlaceHolder1_SearchControl1_SearchButton"))
98  .Click();
99
100 return "done";
101
102 //using OpenQA.Selenium.Remote
103 //using OpenQA.Selenium;
104 //O2Ref:WebDriver.dll
```

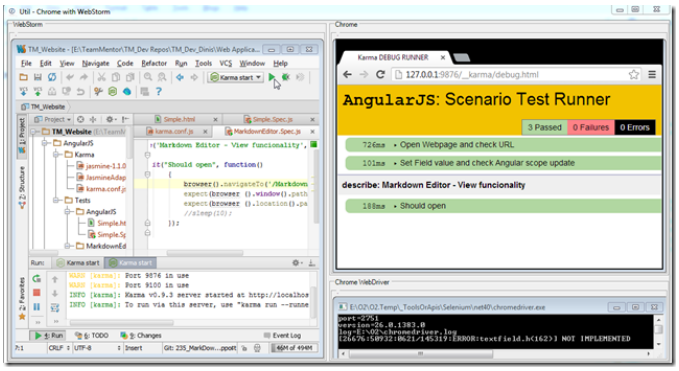
5.6 Adding KarmaJS support to WebStorm to automagically run tests on file changes (and test UI with SublimeText, Chrome and Cmd.exe)

On the ***AngularJs* **and** *KarmaJS*** theme (see A small AngularJS Jasmine test executed by KarmaJS and the related posts linked at the bottom), here is my first attempt at using Karma to test AngularJS code inside TeamMentor.

I'm using WebStorm instead of VisualStudio, since for Javascript coding WebStorm is MUCH better/faster/-cleverer, specially since it has good support for AngularJs and Jasmine (with KarmaJS support easily added, as we are about to see).

Also shown below is a cool tool I created that hijacks windows from SublimeText, Chrome and Cmd.exe windows into the same UI (an O2 Platform .NET Script)

Here is the directory structure:



... with **Karma.config.js **looking like this:

```
module.exports = function(karma)
{
    karma.configure(
        {
            frameworks: ['ng-scenario'],

            files:
                        [
                            '../Tests/**/*.Spec.js'
                        ],

            urlRoot: '/__karma/',

            autoWatch: true,

            proxies: {
                        '/' : 'http://localhost:12120/',
```

```
                        '/Tests': 'http://localhost:12120/AngularJs/Tests/'

                },

            //browsers: ['Chrome'],

             reporters: ['dots'],    //reporters: ['progress'],
             plugins: [
                        'karma-ng-scenario',
                        'karma-chrome-launcher'
                    //,'karma-firefox-launcher'
                    ] ,
           //logLevel: karma.LOG_DEBUG
             logLevel: karma.LOG_INFO
        });
};
```

... **Simple.html **like this:

```
<!DOCTYPE html>
<html xmlns:ng="http://angularjs.org" id="ng-app" ng-app>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" __content="IE=EmulateIE9" />
    <title>Simple AngularJS page</title>
    <script src="/Javascript/angularJS/1.0.5/angular.min.js"></script>
</head>
<body>
<div>
    <label>Name:</label>
    <input type="text" ng-model="yourName" placeholder="Enter a name here">
    <hr>
    <h1>Hello {{yourName}}!</h1>
</div>
</body>
</html>
```

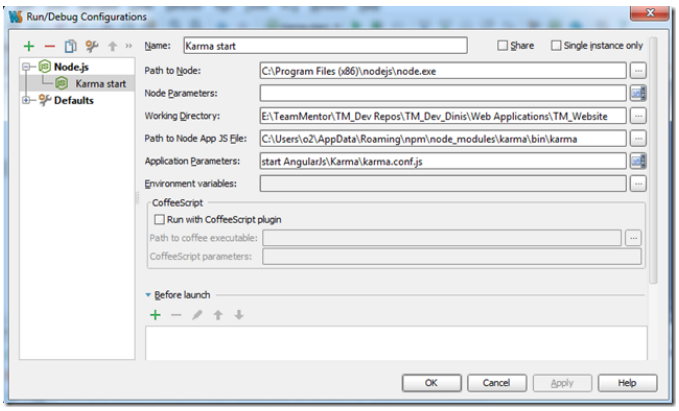
... **Simple.Spec.js** like this:

```
describe('Test Angular Simple page', function()
    {
        beforeEach(function()
            {
                browser().navigateTo('/Tests/AngularJS/Simple.html');
            });
        it('Open Webpage and check URL', function()
            {
                expect(browser ().window().path()).toBe("/Tests/AngularJS/Simple.html");
                expect(browser ().location().path()).toBe("");
                browser().navigateTo('/Tests/AngularJS/aaa.html');
                expect(browser ().window().path()).not().toBe("/Tests/AngularJS/Simple.htm\
l");
                expect(browser ().window().path()).toBe("/Tests/AngularJS/aaa.html");
            });

        it('Set Field value and check Angular scope update', function()
            {
                var testValue = "This is a value";
                var expectedValue = "Hello " + testValue + "!";
                input('yourName').enter(testValue);
                expect(element('.ng-binding').text()).toEqual(expectedValue);
            });

        it('get path value manually (using setTimeout)',function()
            {
                var path = browser().window().path();
                console.log("(before timeout) path value = " + path.value );
                setTimeout(function()
                    {
                        console.log("(after timeout) path value = " + path.value );
                        //expect("value").toBe("/123");

                    },200);
                sleep(1);
            });
    });
```

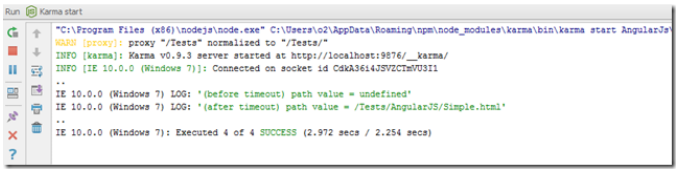
... and **MarkDown.Editor.Spec.js** currently with just

```
describe('Markdown Editor - View funcionality', function()
    {
        it("Should open", function()
            {
                browser().navigateTo('/Markdown/Editor');
                expect(browser ().window().path()).toBe("/Markdown/Editor");
                expect(browser ().location().path()).toBe("");
                //sleep(10);
            });
    });
```

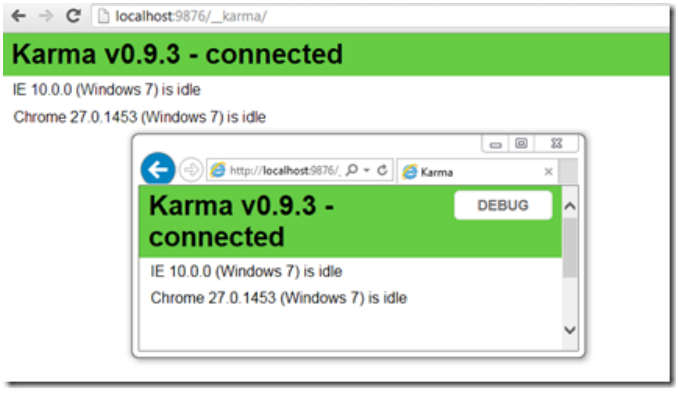
This is how I configured Karma to run on WebStorm:



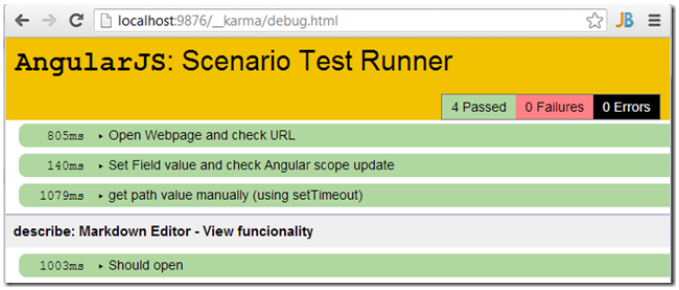
Here is KarmaJS execution log:



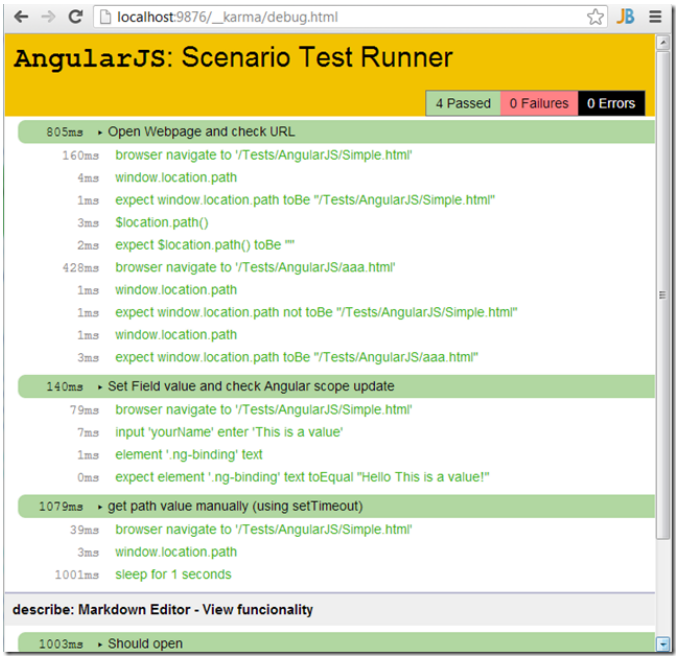
Here are two connected KarmaJS Runners (one in IE and one in Chrome)



And finally, here is the (super useful) ***AngularJS: Scenario Test Runner*** view, in a collapsed state:



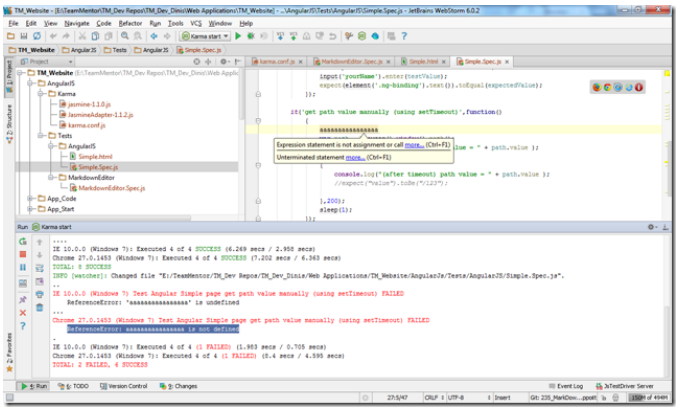
... and in an expanded state:



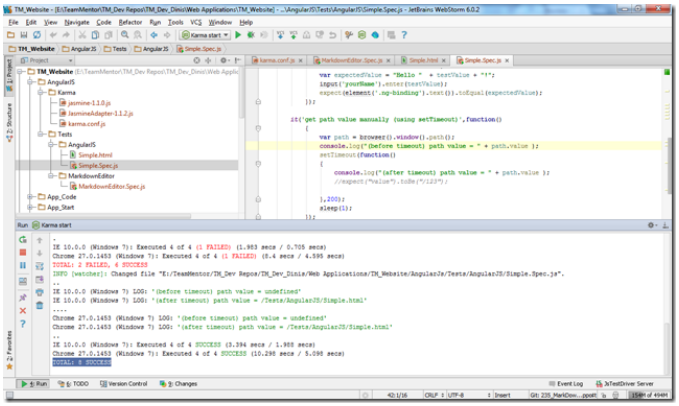
With this set-up KarmaJS is configured to run continuously and to monitor any file changes, which is really powerful, since it allows for continuous development and testing.

For example (to see the automagically execution in action), here is what the WebStorm UI looks like:

** ... with an Javascript error:** (KarmaJS execution triggered on Save) :



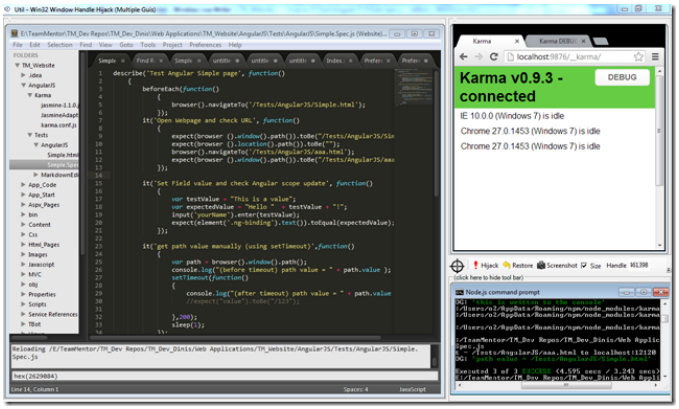
**... without the error: ** (KarmaJS execution triggered on Save)



This is a really nice environment to quickly develop AngularJS apps, specially since the tests are super quick to execute and we can control what tests are executed (for example by creating multiple karma.conf.js files)

O2 Platform Test GUI (created by hijacking 3 processe' windows)

I also created a test GUI using the O2 Platform's window-handle Hijack capabilities, which looked like this:



In the image above:

- The host process is the O2 Platform Util - Win32 Window Handle Hijack (4x host panels).h2 script/tool (which is a .Net app)
  - The left-hand side the **Sublime Text** editor (which is a C++, Pyhton app)
  - The top-right is ***Chrome*** (which is C++ app)
  - The bottom-right is **cmd.exe** (C++ app) running the command ***karma start karma.conf.js **in the **E:TeamMentorTM_Dev ReposTM_Dev_DinisWeb ApplicationsTM_WebsiteAngularJSKarma*** *folder*

All in a nice, self-contained process/environment

Note that there are 4 separate process at play here (O2 Platform, Sublime, Chrome and Cmd.Exe w/ Karma)

Related AngularJS and KarmaJS posts:

- A small AngularJS Jasmine test executed by KarmaJS
- KarmaJS AngularJS Scenario Test Runner execution variations in IE 7,8,9 and 10 when using AngularJS
- Debugging a weird case of missing module in AngularJS and KarmaJS
- Running KarmaJS's AngularJS example test/e2e/angular-scenario (on Chrome)
- AngularJS code editor using UI-Bootstrap and CodeMirror (done without using jQuery)
- Hubspot current.js code includes JQuery on it
- Submitting TM users to HubSpot via TBOT interface (using Angular JS)
- Using Chrome inside a native VisualStudio pane (using Window Handle Hijacking)
- If AngularJS doesn't work on your O2 Platform IE scripts (the fix is to change browser compatibility mode)

Table of Contents | Code

6 Troubleshooting

This section has the following chapters:

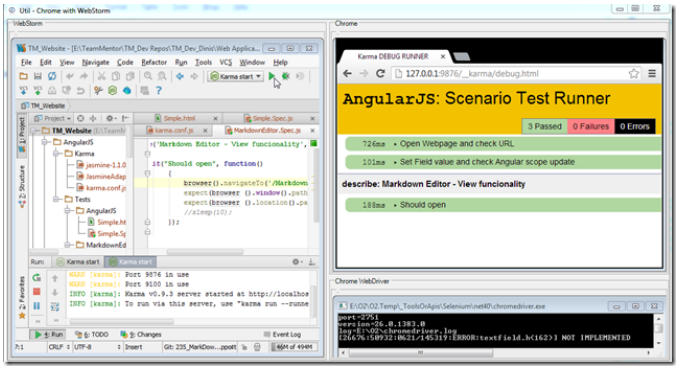
- KarmaJS AngularJS Scenario Test Runner execution variations in IE 7,8,9 and 10 when using AngularJS
- If AngularJS doesn't work on your O2 Platform IE scripts (the fix is to change browser compatibility mode)
- Debugging a weird case of missing module in AngularJS and KarmaJS

6.1 KarmaJS AngularJS Scenario Test Runner execution variations in IE 7,8,9 and 10 when using AngularJS

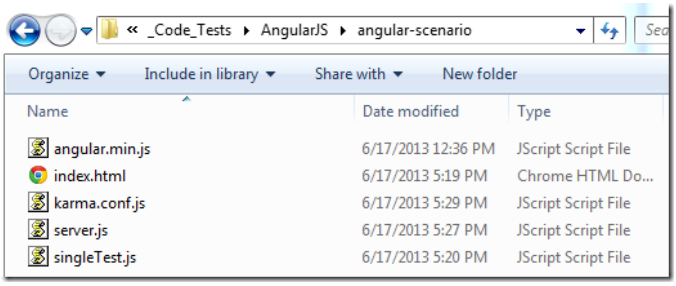
While trying to get Karma JS to work, I found a number of different behaviours for its AngularJS Scenario Test Runner in IE's multiple *'compatibility modes'*.

TLDR: some of the Jasmine and AngularJS test apis don't work (although Angular does seem to work ok)

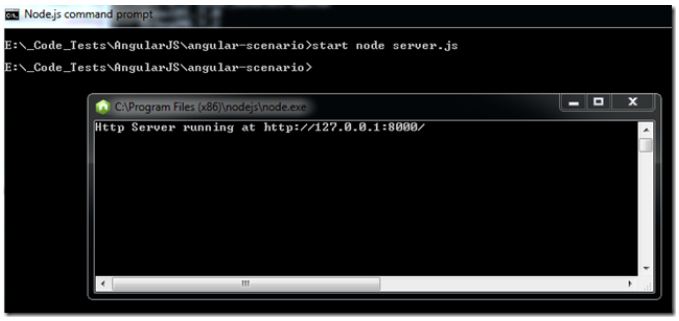
Here is the default web page I was using:



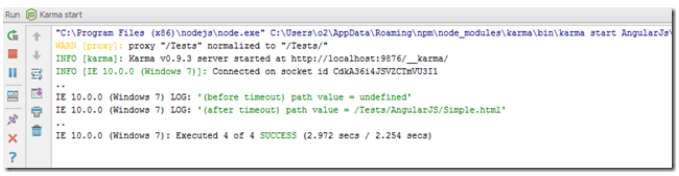
Here is the test executed



Here is KarmaJS starting and successfully executing the tests

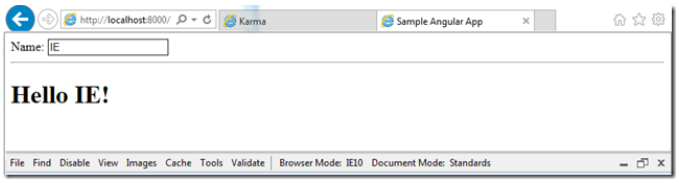


... in this captured IE browser session:

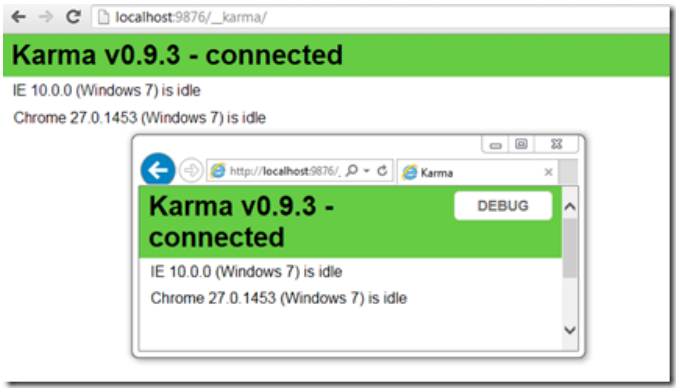


Just to confirm that the target page works in the multiple IE configurations, here it is running in:

IE 10 , IE9, IE 8:



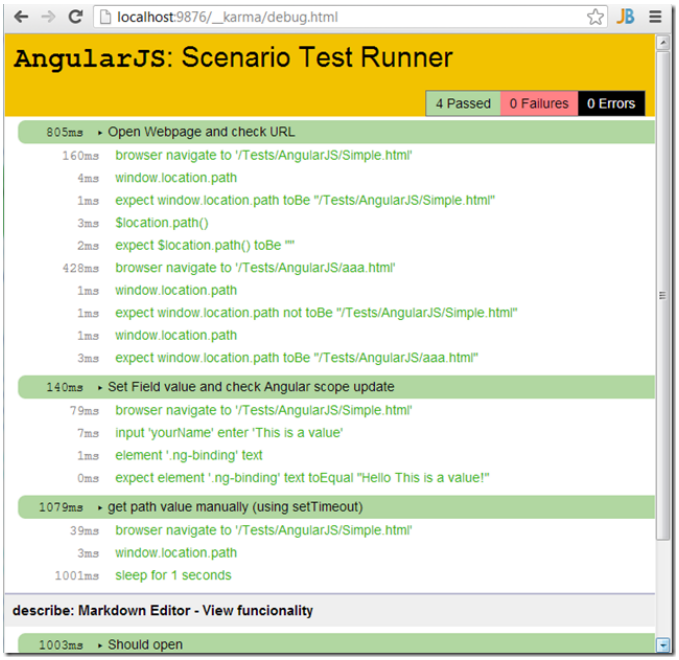
... and even in IE 7:



Now lets click on the _DEBUG _button to open the KarmaJS's_ **AngularJS Scenario Test Runner**_ view and see what happens in multiple IE compatibility modes.

IE 10 Works:

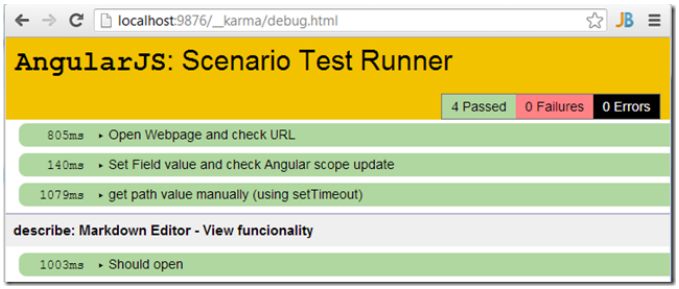
**



**

****IE 10 Compatibility View**

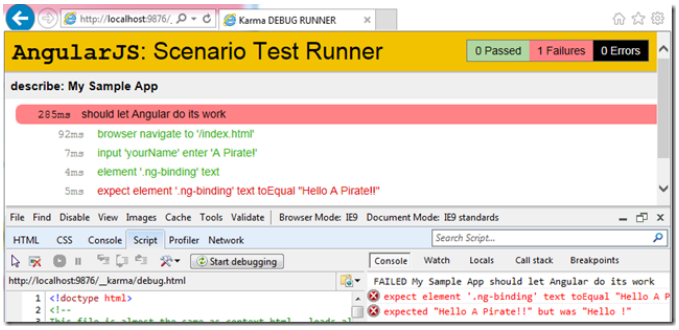
**



**

****IE 9 Fails:**

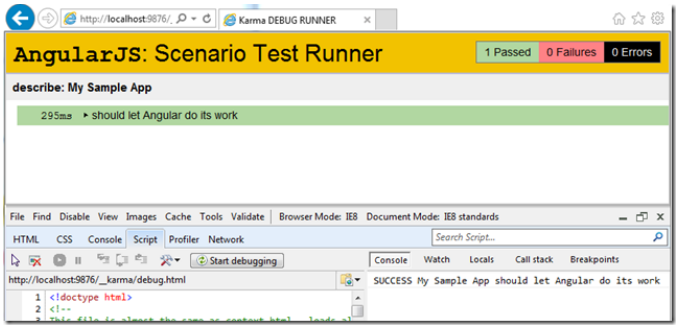
**



**

****IE 8 Works (WTF!!)**

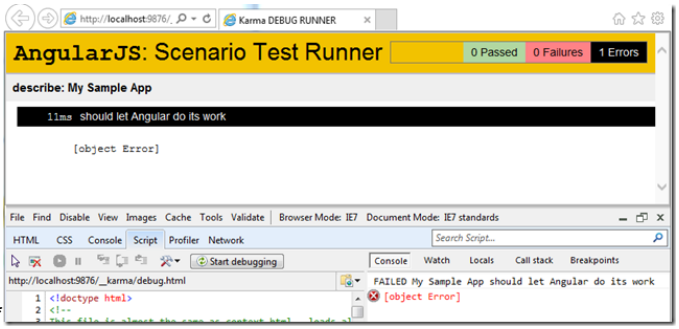
**



**

****IE 7 Fails**

**



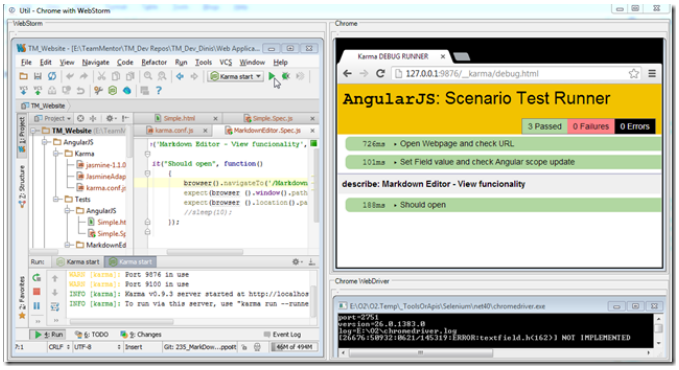
**

So unfortunately it looks like this technique can't be used to run e2e (end-to-end) tests on AngularJS apps using KarmaJS

Table of Contents | Code

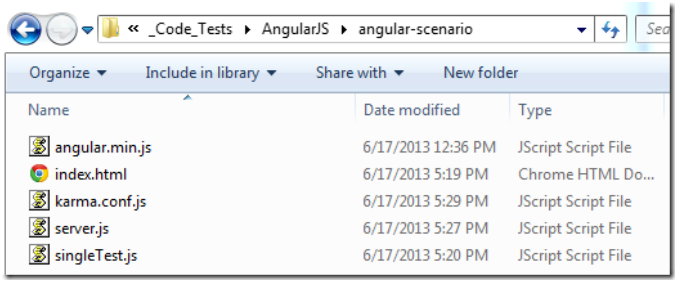
6.2 If AngularJS doesn't work on your O2 Platform IE scripts (the fix is to change browser compatibility mode)

If when trying to open an AngularJS page inside an O2 Platform script, you see:

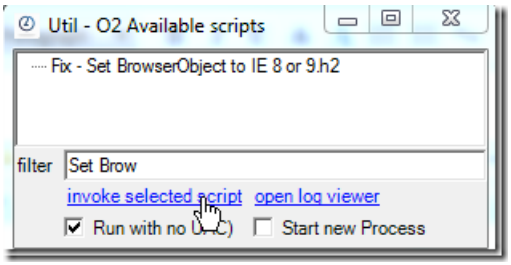


... this means that the IE browser embedded in that .NET process is set to run under IE 7

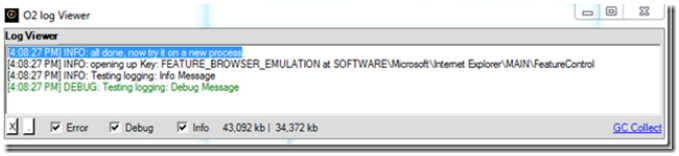
To confirm it, try opening the http://www.whatismybrowser.com and you should see something like:



As mentioned in the set .NET WebBrowser Control to use latest version of IE post to. change it on your system, run this script



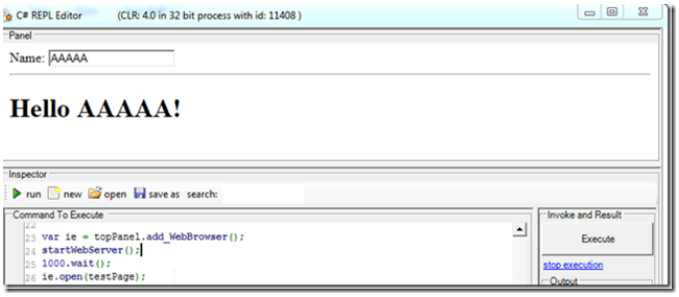
... as admin with no UAC



...and now after restarting the O2 Platform process, IE should be on Internet **Explorer 9 compatibility mode** **

** 

... and AngularJS should work:



**Note 1: **if you control the site you are testing, you can also add this also works to make it work (with the advantage that it is not exe specific)

```
_<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE9" />_
```

Table of Contents | Code

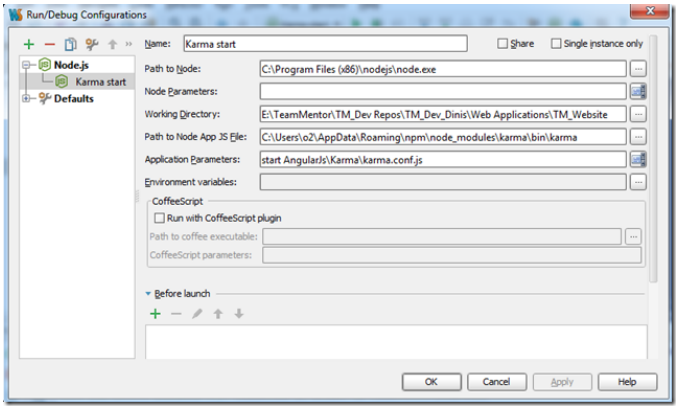
6.3 Debugging a weird case of missing module in AngularJS and KarmaJS

When I was trying the Running KarmaJS's AngularJS example test/e2e/angular-scenario (on Chrome) I hit on the the following weird behaviour.

TLDR; the solution was to run ***npm install –g _****_karma@canary***

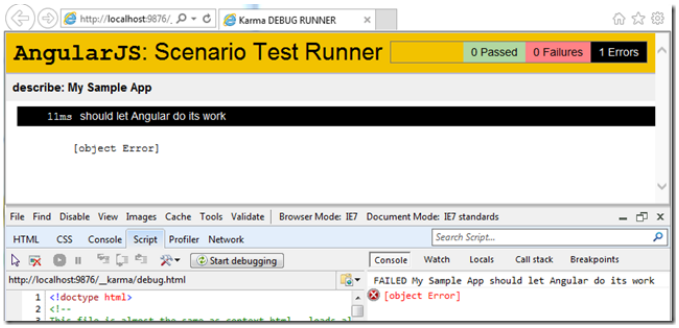
Setup
**

**Chrome window opened in:

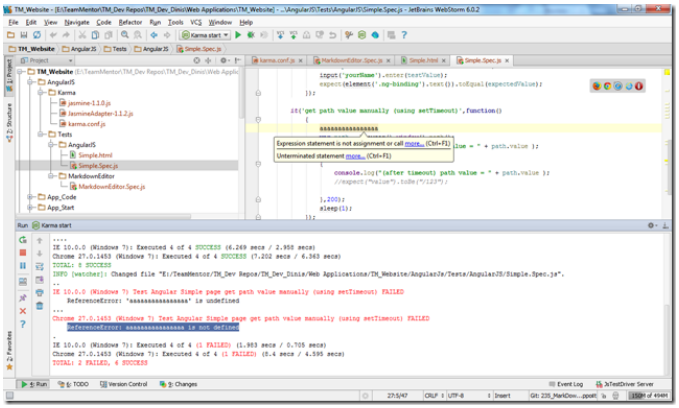


… a local website at port 8000:

… which is a nodeJS powered website, started using ***node server.js*** (below)



A simple (as I could make it)_ **Karma.config.js** _file



…. and AngularJS test

```
/** A Sample Angular E2E test */

describe('My Sample App', function() {

  it('should let Angular do its work', function() {
    browser().navigateTo('/index.html');
    input('yourName').enter('A Pirate!');
    expect(element('.ng-binding').text()).toEqual('Hello A Pirate!!');
  });
```

**

****Scenario A) Running from folder with karma clone (and npm install)**

**

****_karma start ..\angular-scenario\karma.conf.js _**works OK

```
E:\_Code_Tests\AngularJS\karma>karma start ..\angular-scenario\karma.conf.js
DEBUG [config]: autoWatch set to false, because of singleRun
DEBUG [plugin]: Loading plugin karma-ng-scenario.
INFO [karma]: Karma v0.9.3 server started at http://localhost:9876/__karma/
DEBUG [watcher]: Resolved files:
        E:/_Code_Tests/AngularJS/karma/node_modules/karma-ng-scenario/lib/angular-sc
        E:/_Code_Tests/AngularJS/karma/node_modules/karma-ng-scenario/lib/adapter.js
        E:/_Code_Tests/AngularJS/angular-scenario/e2eSpec.js
DEBUG [karma]: New browser has connected on socket 9aNFLIxUCPEn_NPleui-
INFO [Chrome 27.0.1453 (Windows 7)]: Connected on socket id 9aNFLIxUCPEn_NPleui-
DEBUG [karma]: All browsers are ready, executing
DEBUG [web server]: serving: E:/_Code_Tests/AngularJS/karma/node_modules/karma/stat
DEBUG [proxy]: proxying request - /index.html to localhost:8000
DEBUG [proxy]: proxying request - /angular.min.js to localhost:8000
Chrome 27.0.1453 (Windows 7): Executed 1 of 1 SUCCESS (5.388 secs / 4.173 secs)
DEBUG [launcher]: Disconnecting all browsers
```

Scenario B) running from parent folder

**

****_karma start angular-scenario\karma.conf.js _**fails with a ****module is not defined error***

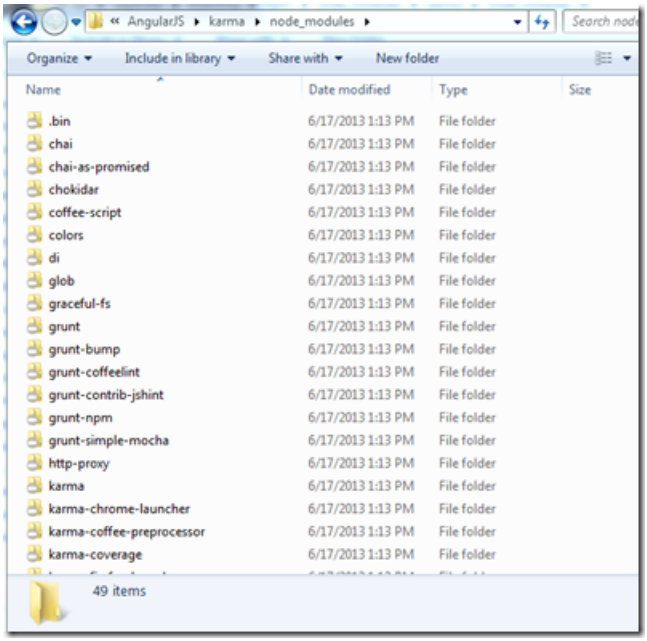
_

```
E:\_Code_Tests\AngularJS\karma>cd ..

E:\_Code_Tests\AngularJS>karma start angular-scenario\karma.conf.js
[2013-05-12 15:14:02.349] [ERROR] config   Invalid config file!
[ReferenceError: module is not defined]
ReferenceError: module is not defined
    at evalmachine.<anonymous>:1:1
    at readConfigFile (C:\Users\o2\AppData\Roaming\npm\node_modules\karma\lib\config.js:158:8)
    at Object.parseConfig (C:\Users\o2\AppData\Roaming\npm\node_modules\karma\lib\config.js:177:
    at Object.exports.start (C:\Users\o2\AppData\Roaming\npm\node_modules\karma\lib\server.js:19
    at Object.<anonymous> (C:\Users\o2\AppData\Roaming\npm\node_modules\karma\bin\karma:20:39)
    at Module._compile (module.js:456:26)
    at Object.Module._extensions..js (module.js:474:10)
    at Module.load (module.js:356:32)
    at Function.Module._load (module.js:312:12)
    at Function.Module.runMain (module.js:497:10)
```

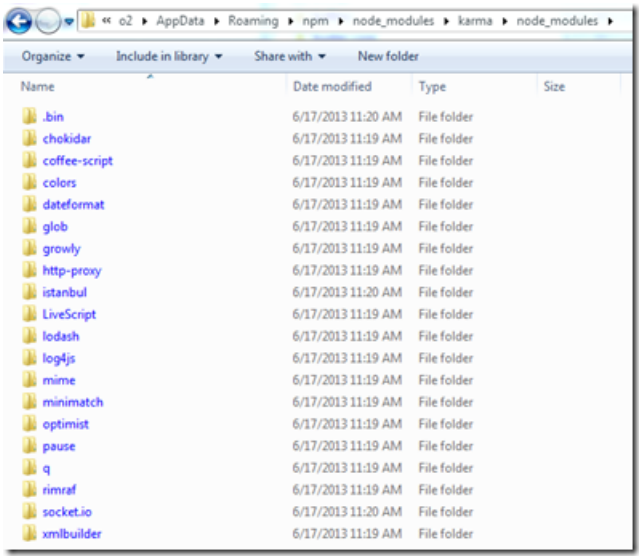
_

So what I think is happening is that because I run_ **npm install*** *on the karma folder (the one I got from a GitHub clone), there are more modules in there than in the global karma (which I got when I installed karma using **_npm install –g karma*)

At the moment there are 49 modules in the GitHub karma:



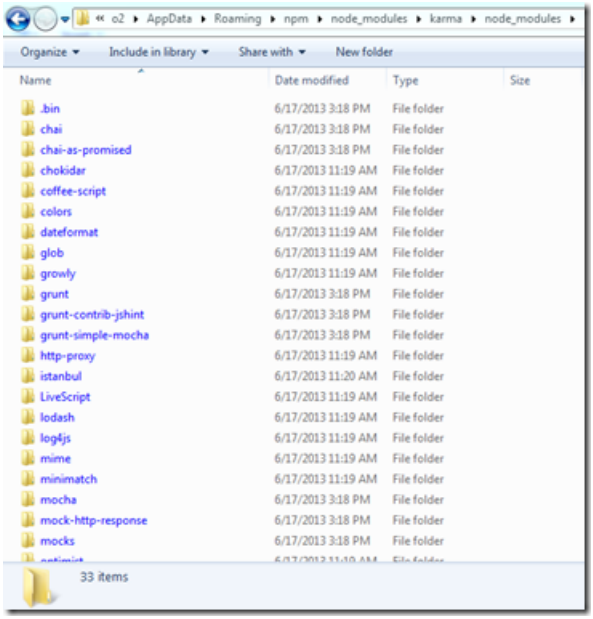
And 20 modules in the global karma install



So let's try running npm install on this folder

```
mock-http-response@0.1.1 node_modules\mock-http-response

which@1.0.5 node_modules\which

sinon-chai@2.3.1 node_modules\sinon-chai

chai-as-promised@3.2.5 node_modules\chai-as-promised

grunt-simple-mocha@0.4.0 node_modules\grunt-simple-mocha

mocks@0.0.11 node_modules\mocks

chai@1.5.0 node_modules\chai

timer-shim@0.2.3-1 node_modules\timer-shim
└── linkedlist@1.0.1

qq@0.3.5 node_modules\qq
└── q@0.8.4

mocha@1.8.2 node_modules\mocha
├── growl@1.7.0
├── debug@0.7.2
├── commander@0.6.1
├── diff@1.0.2
├── mkdirp@0.3.3
├── ms@0.3.0
└── jade@0.26.3 (mkdirp@0.3.0)

grunt@0.4.1 node_modules\grunt
├── hooker@0.2.3
├── async@0.1.22
├── eventemitter2@0.4.12
├── nopt@1.0.10 (abbrev@1.0.4)
├── rimraf@2.0.3 (graceful-fs@1.1.14)
├── coffee-script@1.3.3
├── underscore.string@2.2.0rc
├── iconv-lite@0.2.10
├── lodash@0.9.2
├── findup-sync@0.1.2 (lodash@1.0.1)
└── js-yaml@2.0.5 (esprima@1.0.3, argparse@0.1.15)

sinon@1.6.0 node_modules\sinon
└── buster-format@0.5.5 (buster-core@0.6.4)

grunt-contrib-jshint@0.3.0 node_modules\grunt-contrib-jshint
└── jshint@1.1.0 (peakle@0.0.1, cli@0.4.4-2, underscore@1.4.4, shelljs@0.1.4, esprima@1.1.0-dev)
```

And now there are 33 modules installed:



but that still didn't work :(

At this stage I remembered reading something about installing the latest version of karma (globally), which is probably what I'm getting from the github clone, and that could explain the different number of modules.

So I executed ***npm install –g _****_karma@canary***

```
E:\_Code_Tests\AngularJS\angular-scenario>npm install -g karma@canary
npm http GET https://registry.npmjs.org/karma
npm http 304 https://registry.npmjs.org/karma
npm http GET https://registry.npmjs.org/di
npm http GET https://registry.npmjs.org/chokidar
npm http GET https://registry.npmjs.org/coffee-script
npm http GET https://registry.npmjs.org/rimraf
npm http GET https://registry.npmjs.org/glob
npm http GET https://registry.npmjs.org/minimatch
npm http GET https://registry.npmjs.org/socket.io
npm http GET https://registry.npmjs.org/optimist
npm http GET https://registry.npmjs.org/http-proxy
npm http GET https://registry.npmjs.org/q
npm http GET https://registry.npmjs.org/pause/0.0.1
npm http GET https://registry.npmjs.org/mime
```

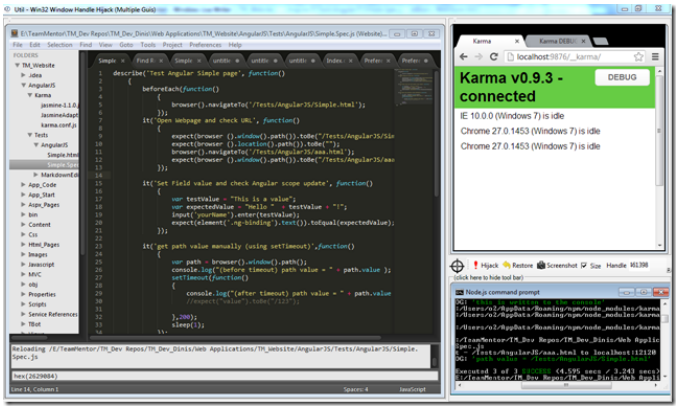
which installed ok:

```
karma@0.9.3 C:\Users\o2\AppData\Roaming\npm\node_modules\karma
├── pause@0.0.1
├── di@0.0.1
├── colors@0.6.0-1
├── graceful-fs@1.2.2
├── rimraf@2.1.4
├── chokidar@0.6.2
├── mime@1.2.9
├── q@0.9.6
├── coffee-script@1.6.3
├── minimatch@0.2.12 (sigmund@1.0.0, lru-cache@2.3.0)
├── lodash@1.1.1
├── glob@3.1.21 (inherits@1.0.0)
├── optimist@0.3.7 (wordwrap@0.0.2)
├── useragent@2.0.6 (lru-cache@2.2.4)
├── log4js@0.6.6 (dequeue@1.0.3, semver@1.1.4, async@0.1.15, readable-stream@1.0.2)
├── http-proxy@0.10.2 (pkginfo@0.2.3, utile@0.1.7)
└── socket.io@0.9.16 (base64id@0.1.0, policyfile@0.0.4, redis@0.7.3, socket.io-client@0.9.16)
```

Now runnig the **scenario B) **case throws a different error:

```
E:\_Code_Tests\AngularJS>karma start angular-scenario\karma.conf.js
DEBUG [config]: autoWatch set to false, because of singleRun
DEBUG [plugin]: Loading plugin karma-ng-scenario.
WARN [plugin]: Cannot find plugin "karma-ng-scenario".
  Did you forget to install it ?
  npm install karma-ng-scenario --save-dev

C:\Users\o2\AppData\Roaming\npm\node_modules\karma\node_modules\di\lib\injector.js:9
      throw error('No provider for "' + name + '"! (Resolving: ' +

Error: No provider for "framework:ng-scenario"! (Resolving: framework:ng-scenario)
    at error (C:\Users\o2\AppData\Roaming\npm\node_modules\karma\node_modules\di\lib\in
```

It looks that we also need install the ***karma-ng-scenario*** module



And now it works :)

```
E:\_Code_Tests\AngularJS>karma start angular-scenario\karma.conf.js
DEBUG [config]: autoWatch set to false, because of singleRun
DEBUG [plugin]: Loading plugin karma-ng-scenario.
INFO [karma]: Karma v0.9.3 server started at http://localhost:9876/__karma/
DEBUG [watcher]: Resolved files:
        C:/Users/o2/AppData/Roaming/npm/node_modules/karma-ng-scenario/lib/angular-scenario.js
        C:/Users/o2/AppData/Roaming/npm/node_modules/karma-ng-scenario/lib/adapter.js
        E:/_Code_Tests/AngularJS/angular-scenario/e2eSpec.js
DEBUG [karma]: New browser has connected on socket 4-p7Xg16B9uxe6y_kA2s
INFO [Chrome 27.0.1453 (Windows 7)]: Connected on socket id 4-p7Xg16B9uxe6y_kA2s
DEBUG [karma]: All browsers are ready, executing
DEBUG [web server]: serving: C:/Users/o2/AppData/Roaming/npm/node_modules/karma/static/context.html
DEBUG [web server]: serving: C:/Users/o2/AppData/Roaming/npm/node_modules/karma-ng-scenario/lib/adap
ter.js
DEBUG [web server]: serving: C:/Users/o2/AppData/Roaming/npm/node_modules/karma-ng-scenario/lib/angu
lar-scenario.js
DEBUG [proxy]: proxying request - /index.html to localhost:8000
DEBUG [proxy]: proxying request - /angular.min.js to localhost:8000
Chrome 27.0.1453 (Windows 7): Executed 1 of 1 SUCCESS (0.416 secs / 0.235 secs)
DEBUG [launcher]: Disconnecting all browsers
```

Table of Contents | Code

Appendix Post's Details

This is the chronological order of the posts:

March 2013

- Using Chrome inside a native VisualStudio pane (using Window Handle Hijacking)

April 2013

- Submitting TM users to HubSpot via TBOT interface (using Angular JS)
- Hubspot current.js code includes JQuery on it

June 2013

- AngularJS code editor using UI-Bootstrap and CodeMirror (done without using jQuery)
- Running KarmaJS's AngularJS example test/e2e/angular-scenario (on Chrome)
- Debugging a weird case of missing module in AngularJS and KarmaJS
- If AngularJS doesn't work on your O2 Platform IE scripts (the fix is to change browser compatibility mode)
- KarmaJS AngularJS Scenario Test Runner execution variations in IE 7,8,9 and 10 when using AngularJS
- A small AngularJS Jasmine test executed by KarmaJS
- Adding KarmaJS support to WebStorm to automagically run tests on file changes (and test UI with SublimeText, Chrome and Cmd.exe)
- When the best way to automate Chrome is to use … Chrome (with examples on Google search, direct AngularJS scope manipulation and ChromeDriver javascript access)
- Using WebStorm with Chrome and ChromeDriver (to view KarmaJS execution results)

February 2014

- Creating an Eclipse UI to run AngularJS e2e tests using Karma
- Trying out Firebase (Beta) hosting solution and good example of Firebase Security rules
- First PoC of sending TeamMentor's server-side request URLS to Firebase (and seeing it in realtime in an AngularJS page)
- Eclipse Groovy REPL script to sync a Browser with file changes (with recursive folder search via Java's WatchService)
- A really SIMPLE and clean AngularJS+Firebase example
- Using AngularJS in Eclipse, Part 1) The Basics
- Using AngularJS in Eclipse, Part 2) Add Some Control
- Using AngularJS in Eclipse, Part 3) Wire up a Backend
- Using AngularJS in Eclipse, Part 4) Create Components

March 2014

- Eclipse Groovy script to remove the 'busy' image from the WebBrowser Editor
- Programatically changing an AngularJS scope variable and adding Firebug Lite to an AngularJs app

---

Table of Contents | Code