

1. Git-SCM: .....	2
2. Install Ruby and Compass: .....	5
3. NodeJS:.....	7
4. Creating Boilerplate AngularJS project with WebStorm IDE: .....	9
5. Install xComponentWidgets.....	13
6. l18n filter in component .....	15
7. Template cache.....	16
8. Install grunt-ngdocs.....	18
9. Naming convention:.....	24

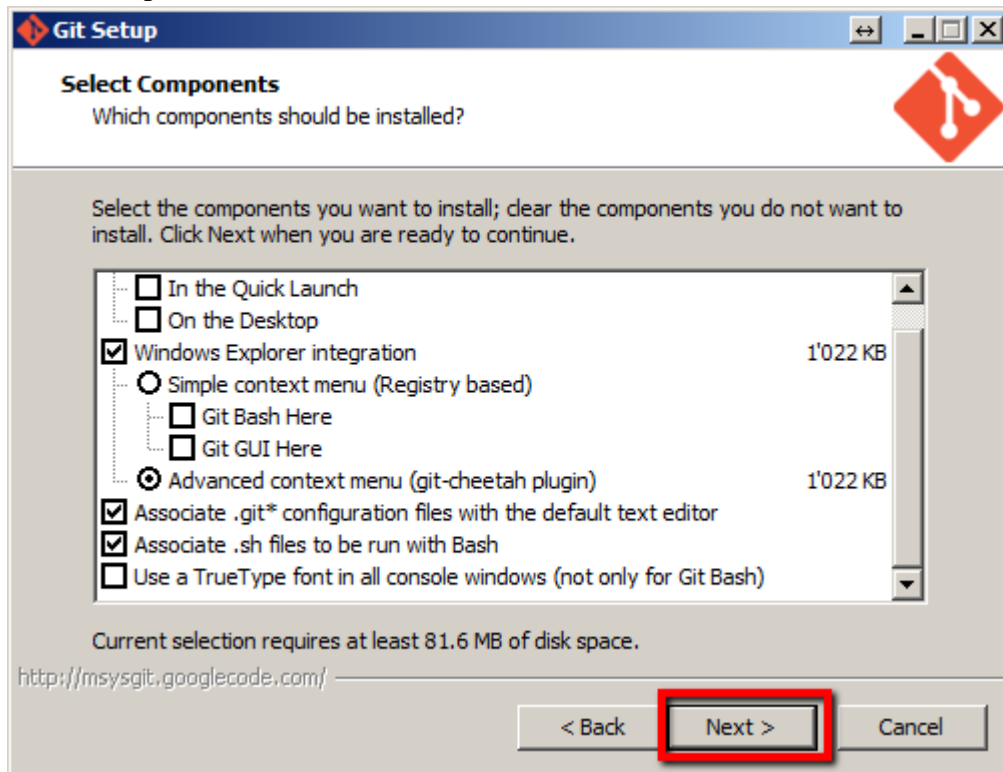
## 1. Git-SCM:

Link: <http://git-scm.com/>

Why do we need **Git-SCM**: Because **bower** will get projects on git-hub.

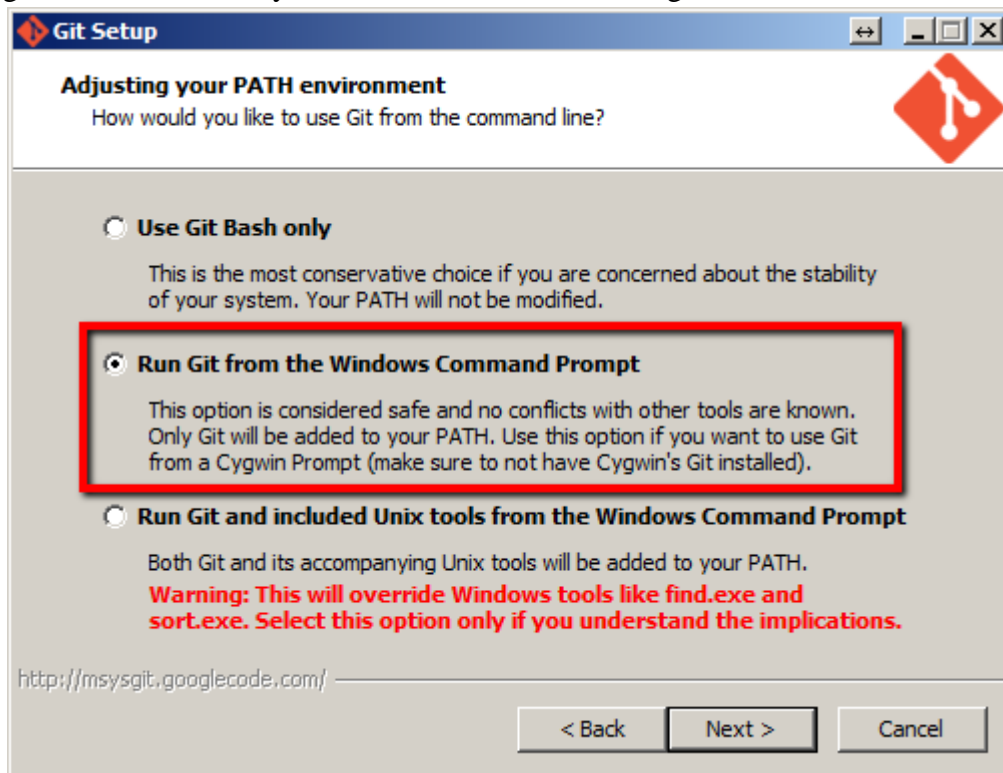
Installation steps for Git-SCM version 1.9.0:

1. Select component:

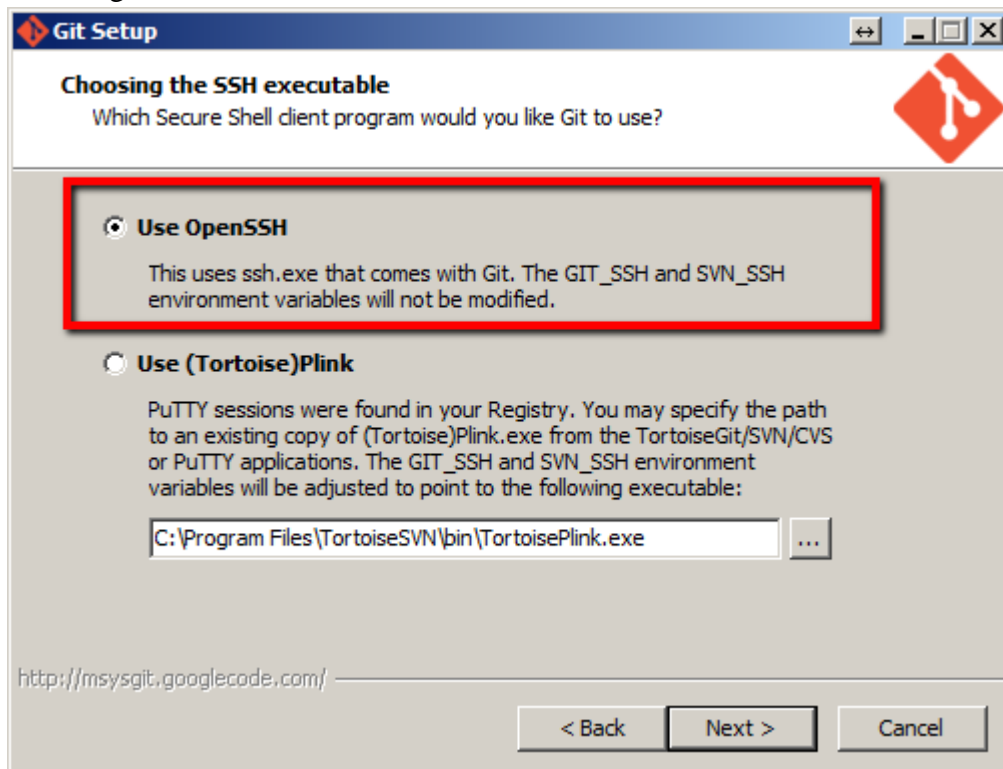


2. Adjusting your PATH environment:

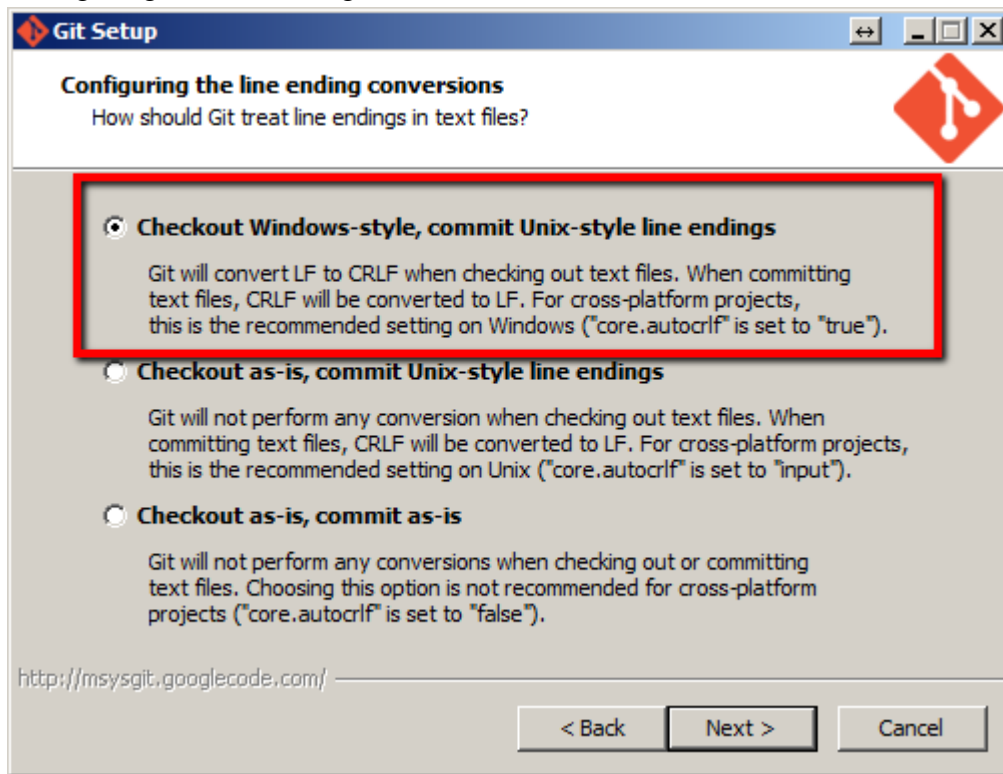
Please make use you read it carefully. If you choose **Use Git Bash Only**. You must add git to PATH manually. Because **bower** need to call git command.



3. Choosing the SSH executable:



4. Configuring the line ending conversions:



5. Test installed successful with GIT-SCM in PATH:

```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\nvduc>git --version
git version 1.9.0.msysgit.0
C:\Users\nvduc>_
```

## 2. Install Ruby and Compass:

We use compass for compile SCSS files to CSS files.

Compass runs on any computer that has Ruby installed.

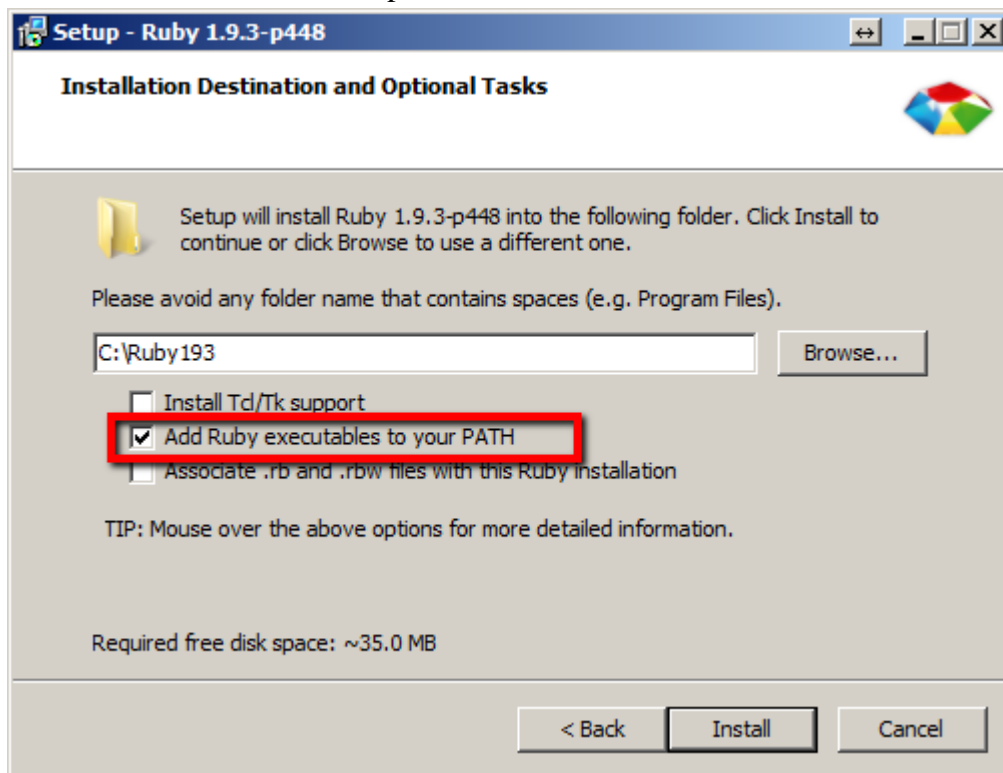
We have to install Ruby first.

Link: <https://www.ruby-lang.org/en/installation/>

Download RubyInstaller package.

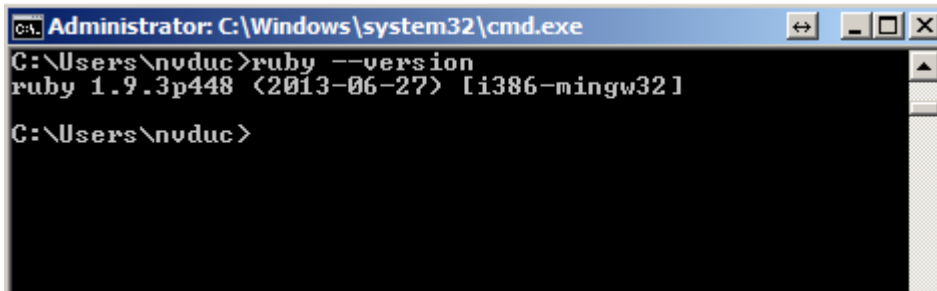
Installation steps for Ruby version 1.9.3

### 1. Installation Destination and Optional Tasks:



We need to check add to **PATH**. We will use it for install **Compass**

2. Check installed successfully:



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\nvduc>ruby --version
ruby 1.9.3p448 (2013-06-27) [i386-mingw32]
C:\Users\nvduc>
```

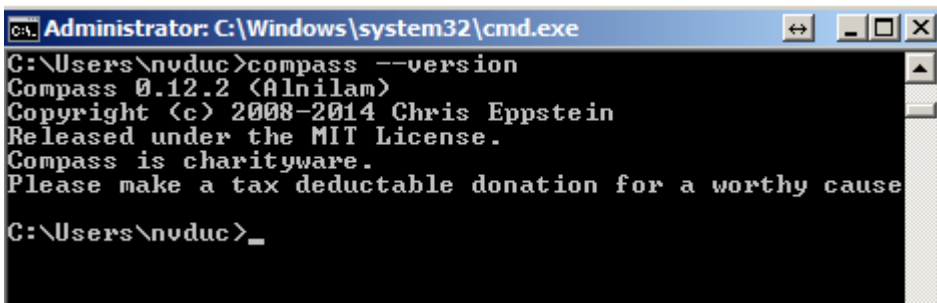
Installation steps for Compass:

1. Run these commands to install Compass:

```
$gem update --system
```

```
$gem install compass
```

2. Check installed successfully:



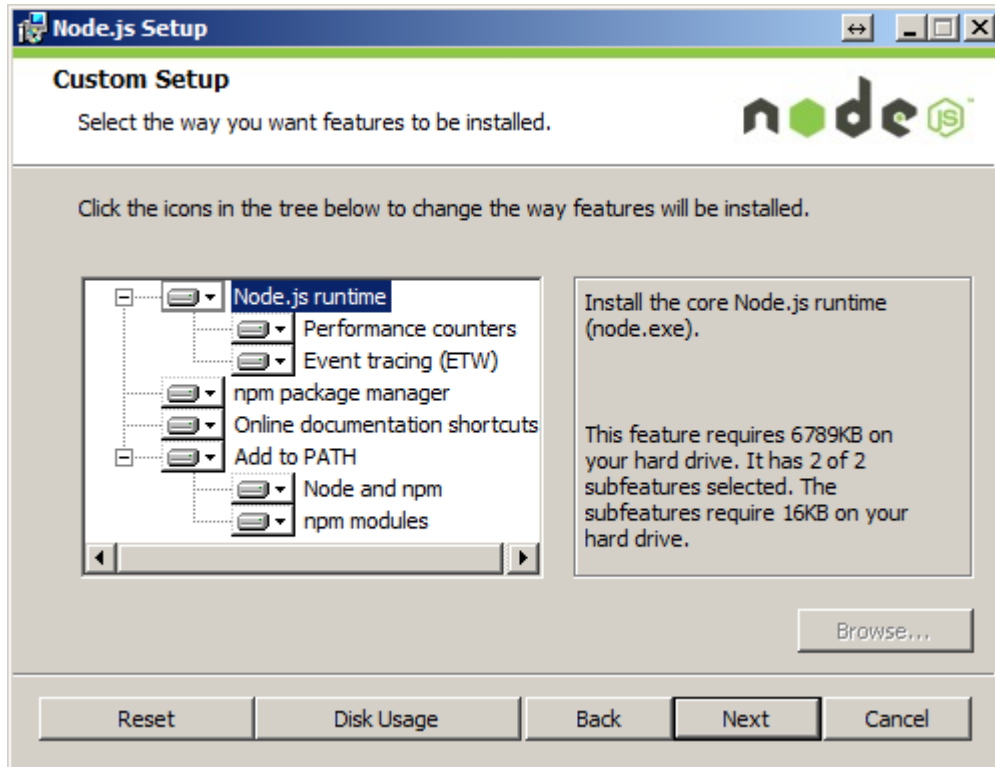
```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\nvduc>compass --version
Compass 0.12.2 (Alnilam)
Copyright (c) 2008-2014 Chris Eppstein
Released under the MIT License.
Compass is charityware.
Please make a tax deductible donation for a worthy cause
C:\Users\nvduc>_
```

### 3. NodeJS:

Many components like Grunt, Bower, Yeoman, Karman... build on top of NodeJS .That's why we need to installed NodeJS.

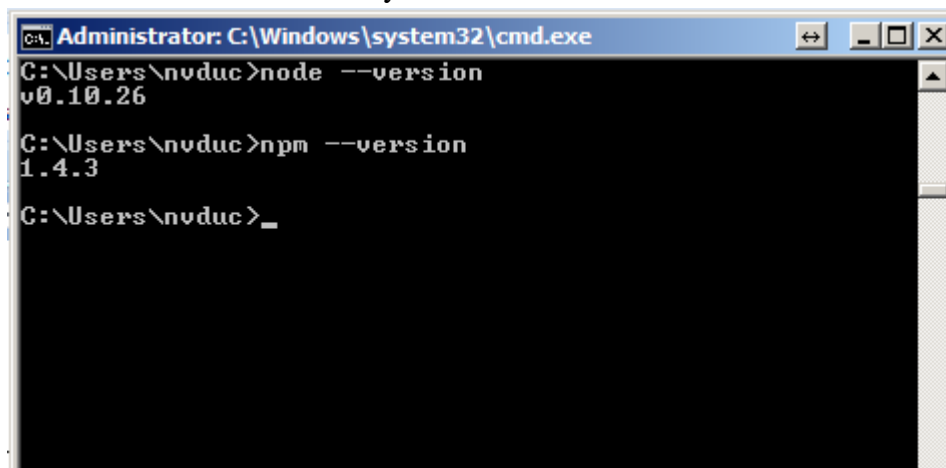
Installation steps with NodeJS version v0.10:

#### 1. Custom Setup:



Important select is **Add to PATH**.

#### 2. Check installation successfully:



Install some modules:

1. Grunt:  
\$npm install -g grunt-cli
2. Bower:  
\$npm install -g bower
3. Yeoman:  
\$npm install -g yo  
\$npm install -g generator-angular

## package.json

The *package.json* file belongs in the root directory of your project, next to the Gruntfile, and should be committed with your project source. Running **\$npm install** in the same folder as a *package.json* file will install the correct version of each dependency listed therein.

There are a few ways to create a *package.json* file for your project:

Most grunt-init templates will automatically create a project-specific *package.json* file.

The **\$npm init** command will create a basic *package.json* file.

Example:

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.2",
    "grunt-contrib-jshint": "~0.6.3",
    "grunt-contrib-nodeunit": "~0.2.0",
    "grunt-contrib-uglify": "~0.2.2"
  }
}
```

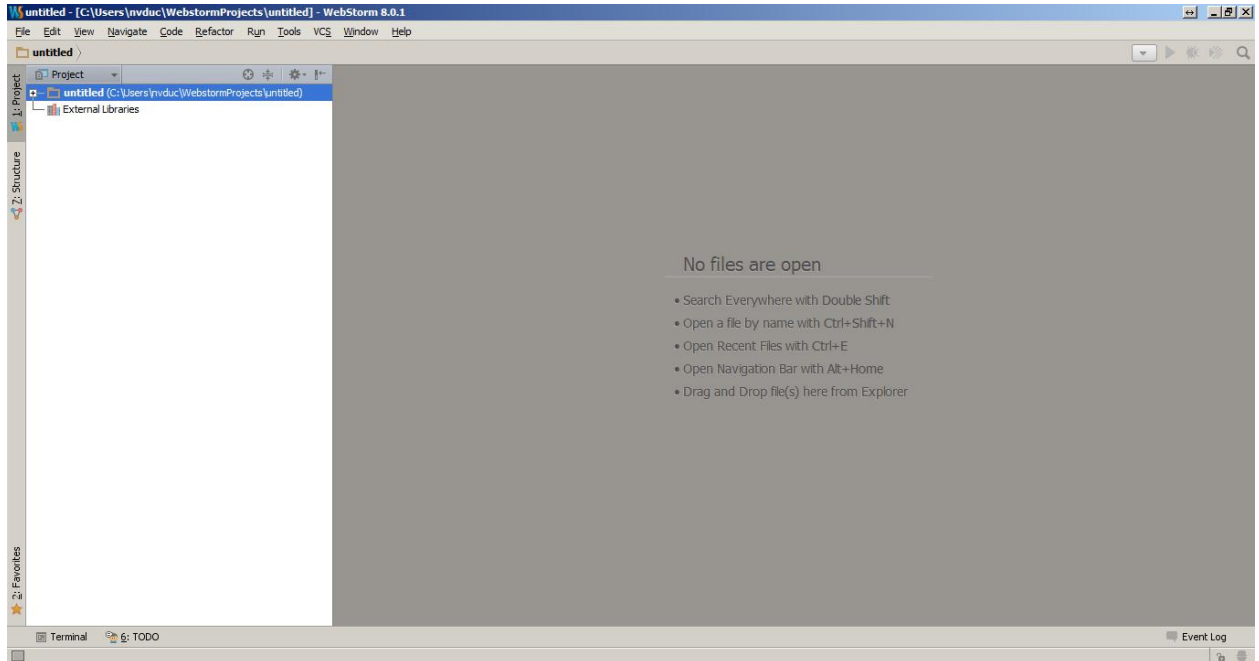
Run **\$npm install** if you added new packages manually.



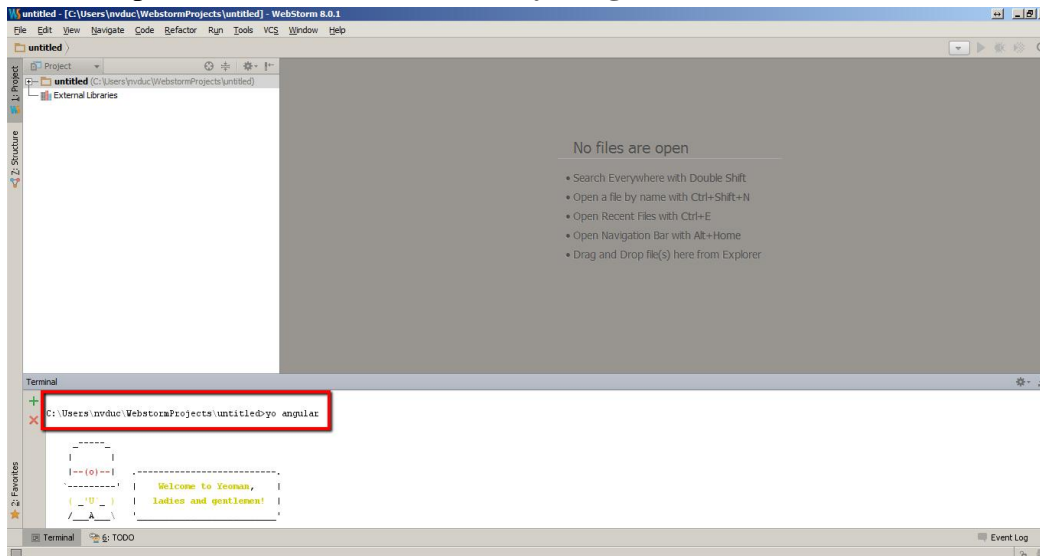
## 4. Creating Boilerplate AngularJS project with WebStorm IDE:

### 1. Creating new project:

Important! Close your WebStorm if it was opening. Reopen it and create new project.



### 2. Tool → Open Terminal. Run command **\$yo angular**

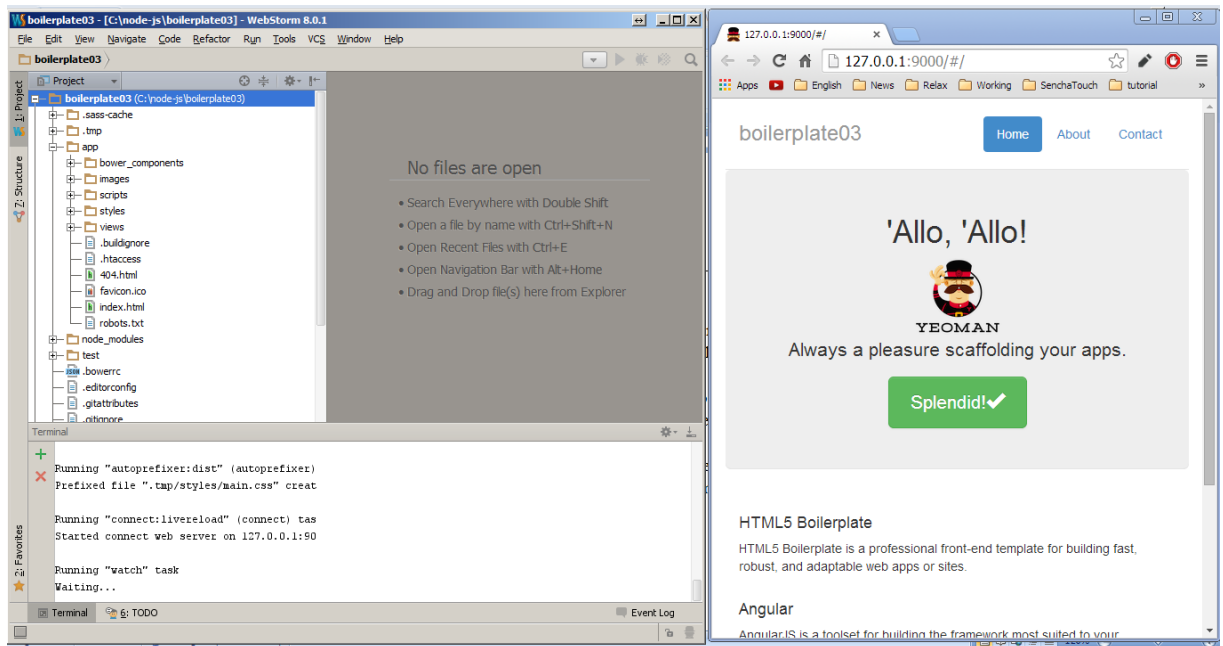


Select some dependencies if you need.

If you got errors with command: **\$yo angular**. Back to part 3 and recheck installation components part again.

If you have some problems with **grunt** and **bower**. Run these command **\$npm install** and **\$bower install** after that run **\$grunt serve** again.

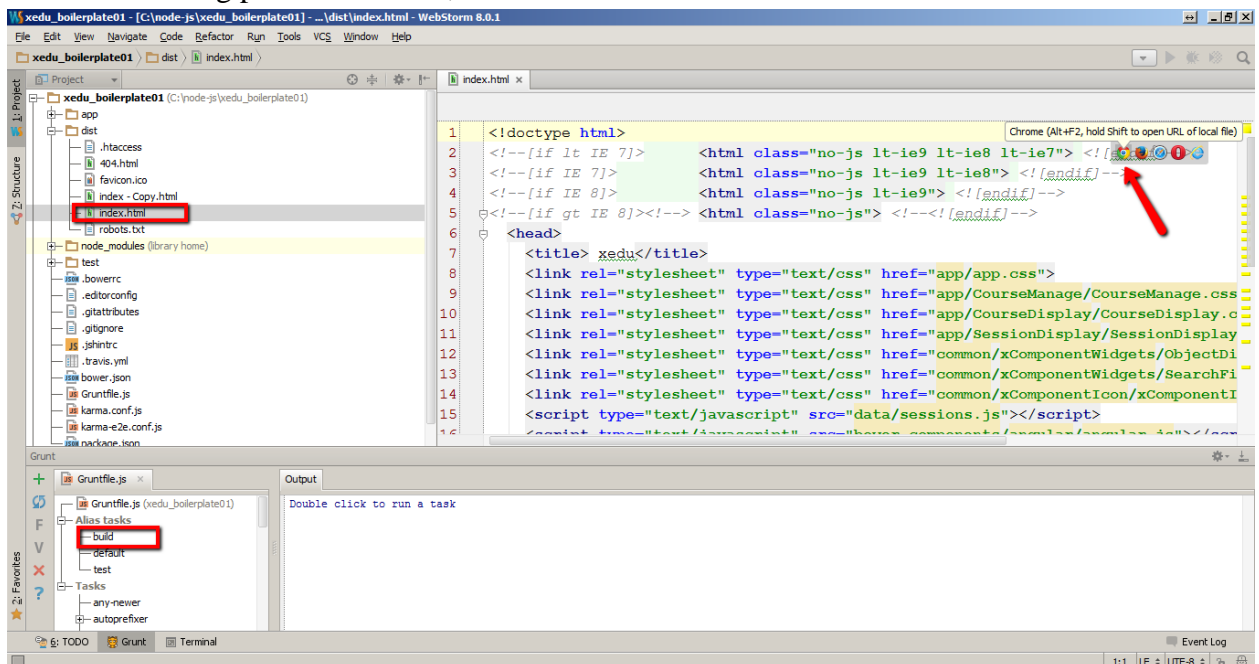
3. Now run **\$grunt serve** command for check new Boilerplate created.



4. Build project with Grunt Console.

Tool → Open Grunt Console → Click **Build** command in grunt console.

After the building process finished, we can have this screen.



You can test the distribution project by click on Chrome icon.

5. Testing Project:

In WebStorm command line run commands for install some dependencies:

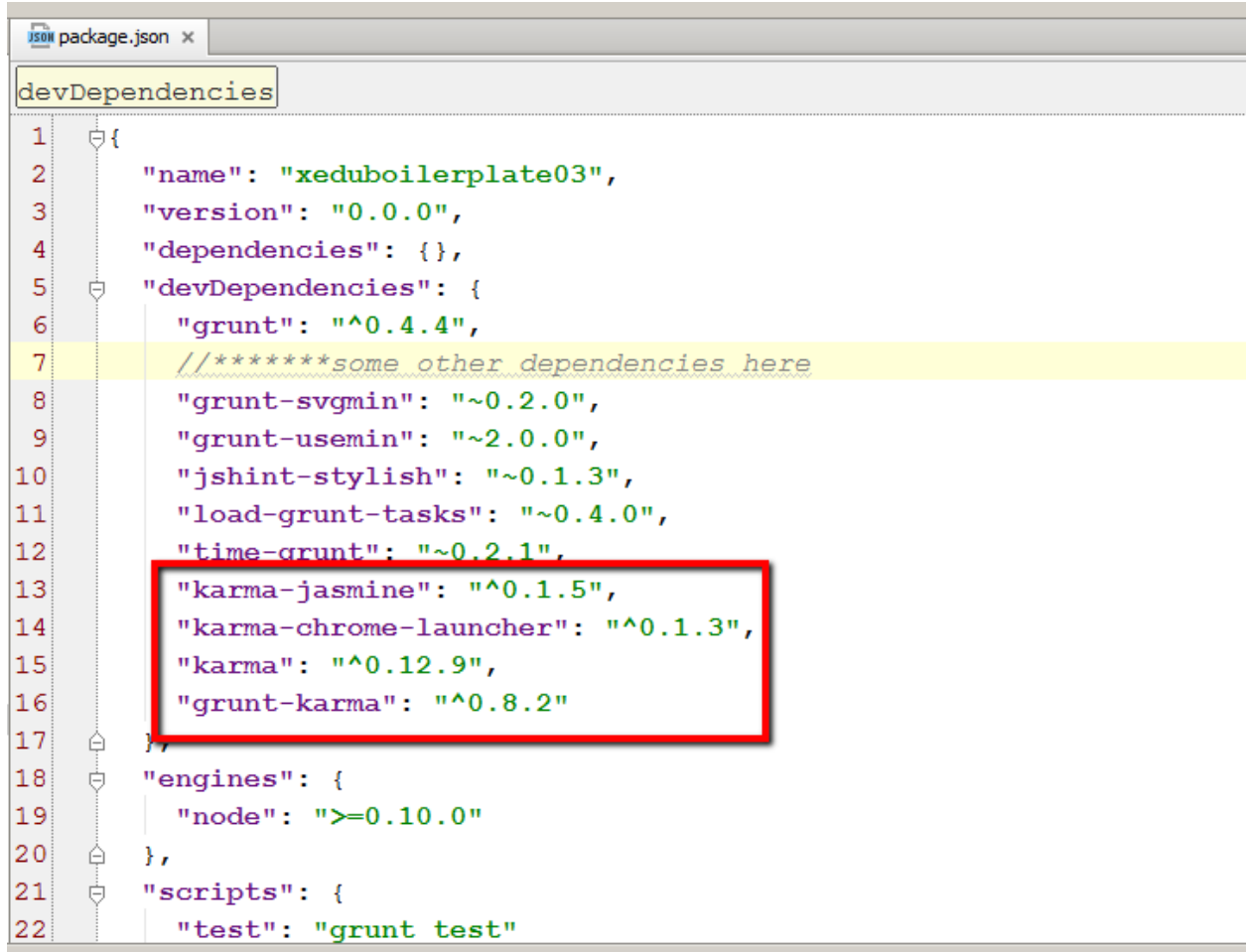
**\$npm install karma --save-dev**

**\$npm install grunt-karma --save-dev**

**\$npm install karma-jasmine --save-dev**

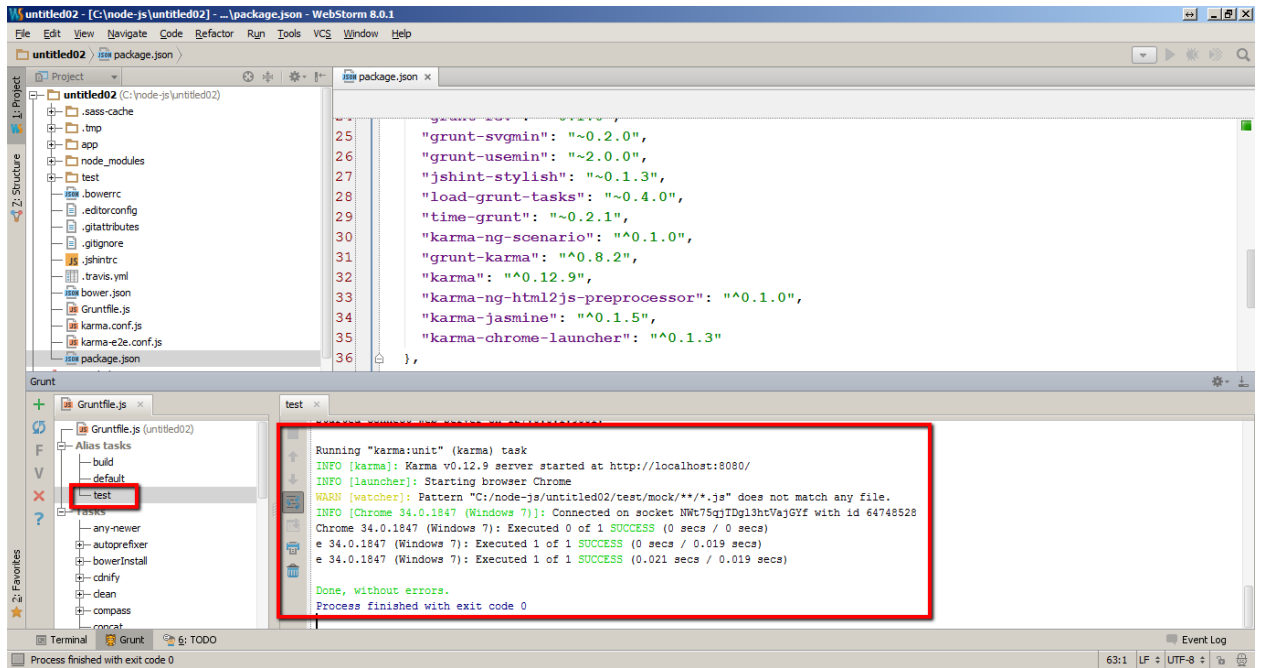
**\$npm install karma-chrome-launcher --save-dev**

After install we have dependencies in package.json.



```
package.json x
devDependencies
1 {
2   "name": "xeduboilerplate03",
3   "version": "0.0.0",
4   "dependencies": {},
5   "devDependencies": {
6     "grunt": "^0.4.4",
7     *****some other dependencies here
8     "grunt-svgmin": "~0.2.0",
9     "grunt-usemin": "~2.0.0",
10    "jshint-stylish": "~0.1.3",
11    "load-grunt-tasks": "~0.4.0",
12    "time-grunt": "~0.2.1",
13    "karma-jasmine": "^0.1.5",
14    "karma-chrome-launcher": "^0.1.3",
15    "karma": "^0.12.9",
16    "grunt-karma": "^0.8.2"
17  },
18  "engines": {
19    "node": ">=0.10.0"
20  },
21  "scripts": {
22    "test": "grunt test"
```

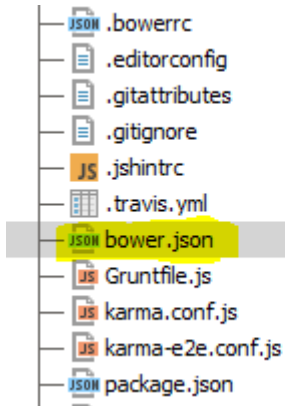
Open Grunt Console → Click test.



## 5. Install xComponentWidgets

Add this line into dependencies property in bower.json

```
"xComponentWidgets": "svn+https://[svn_link_of_xComponentWidgets]"
```

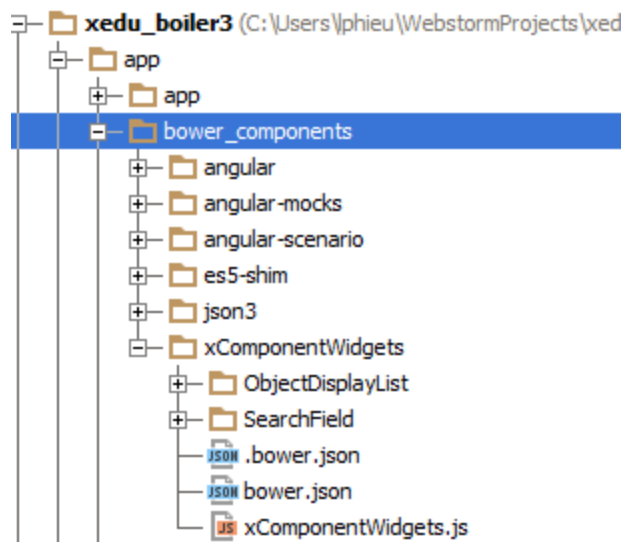


```
{
  "name": "xedu-boiler3",
  "version": "0.0.0",
  "dependencies": {
    "angular": "1.2.15",
    "json3": "~3.2.6",
    "es5-shim": "~2.1.0",
    "xComponentWidgets": "svn+https://test-bower-phuchieu.googlecode.com/svn/"
  },
  "devDependencies": {
    "angular-mocks": "1.2.15",
    "angular-scenario": "1.2.15"
  }
}
```

Run in terminal

```
$ bower install
```

It will download component from svn and put it in bower\_component folder



In bower.json file of xComponentWidgets, there are a list of files in main property, these files will be included automatically into index.html by “grunt bowerInstall” command

```
$ grunt bowerInstall
```

```
"main": [
  "./ObjectDisplayList/ObjectDisplayList.js",
  "./SearchField/SearchField.js",
  "./SearchField/labels.js",
  "./ObjectDisplayList/labels.js",
  "./xComponentWidgets.js"
],

<!-- bower:js -->
<script src="bower_components/es5-shim/es5-shim.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/json3/lib/json3.min.js"></script>
<script src="bower_components/xComponentWidgets/ObjectDisplayList/ObjectDisplayList.js"></script>
<script src="bower_components/xComponentWidgets/SearchField/SearchField.js"></script>
<script src="bower_components/xComponentWidgets/SearchField/labels.js"></script>
<script src="bower_components/xComponentWidgets/ObjectDisplayList/labels.js"></script>
<script src="bower_components/xComponentWidgets/xComponentWidgets.js"></script>
<!-- endbower -->
```

## 6. I18n filter in component

In xComponentWidgets.js, we define i18n filter and global variable locales:

```
var _locales ={};
xComponentWidgets.filter('i18n', function() {

    return function(str) {
        var l_lang;
        if (navigator.userAgent) // Explorer
            l_lang = navigator.userAgent;
        else if (navigator.language) // FF
            l_lang = navigator.language;
        else
            l_lang = "en";
        l_lang=l_lang.toLocaleLowerCase();
        var offset = 1;
        str = _locales[l_lang][str] || str;

        for (var i = offset; i < arguments.length; i++) {
            str = str.split('%' + (i - offset + 1)).join(arguments[i]);
        }

        return str;
    }
})
```

In each component, we create a js file for label, the content is look like this

```
_locales = {
    'vi': {
        'searchfieldplaceholder': 'vi_searchfieldplaceholder '
    },
    'en-us': {
        'searchfieldplaceholder': 'en_searchfieldplaceholder fr'
    },
    'fr': {
        'searchfieldplaceholder': 'fr_searchfieldplaceholder fr'
    }
}
```

Using:

In html file:

```
{{'Hello, world'|i18n}}
```

In js file:

```
$filter('i18n')('String in js');
```

## 7. Template cache

Install the plugin:

```
$ npm install grunt-angular-templates --save-dev
```

Enable the plugin within your Gruntfile:

```
grunt.loadTasks('grunt-angular-templates');
```

Register HTML Templates by create a task in Gruntfile.js

```
grunt.initConfig({  
  ngtemplates: {  
    app: {  
      cwd: 'app/',  
      src: '**/*-template.html',  
      dest: 'app/templates.js',  
      options: {  
        url: function(url) {  
          return url.split("/").pop();  
        }  
      }  
    }  
  },  
},
```

Create template.js

```
$ grunt ngtemplates
```

```
angular.module('app').run(['$templateCache', function($templateCache) {  
  'use strict';  
  
  $templateCache.put('SessionDisplay-template.html',  
    "<div class=\"background SessionDisplay\">\r" +  
    "\n" +  
    "id :{{session.id}}<br/>\r" +  
    "\n" +  
    "name : {{session.name}}\r" +  
    "\n" +  
    "</div>"  
  );  
});
```



Use template in component like this

```
xComponentWidgets.directive('searchfield', function($templateCache,$filter) {  
    return {  
        restrict:'AE',  
        replace:true,  
        template: $templateCache.get('SearchField-template.html'),  
    }  
})
```

## 8. Install grunt-ngdocs

From the same directory as your project's Gruntfile and package.json, install this plugin with the following command:

```
npm install grunt-ngdocs --save-dev
```

```
C:\Users\lphieu\WebstormProjects\Javadoc>npm install grunt-ngdocs --save-dev
npm WARN package.json javadoc@0.0.0 No description
npm WARN package.json javadoc@0.0.0 No repository field.
npm WARN package.json javadoc@0.0.0 No README data
npm http GET https://registry.npmjs.org/grunt-ngdocs
npm http 304 https://registry.npmjs.org/grunt-ngdocs
npm http GET https://registry.npmjs.org/marked/0.2.9
npm http 304 https://registry.npmjs.org/marked/0.2.9
grunt-ngdocs@0.2.1 node_modules\grunt-ngdocs
└─ marked@0.2.9
```

### Configure Gruntfile.js

Once that's done, add this line to your project's Gruntfile:

```
grunt.loadNpmTasks('grunt-ngdocs');
```

Inside your `Gruntfile.js` file, add a section named *ngdocs*. Here's a simple example:

```
ngdocs: {
  api: {
    src: ['app/**/*.js'],
    title: 'API Documentation'
  },
  tutorials: {
    src: ['docs/tutorials/**/*.ngdoc'],
    title: 'Tutorials'
  }
}
```

And with all options: (you can remove options that you don't know, it will be used with default value)

```
ngdocs: {
  options: {
    dest: 'docs',
    scripts: ['./app.min.js'],
    html5Mode: true,
    startPage: '/api',
    title: "My Awesome Docs",
    image: "path/to/my/image.png",
    imageLink: "http://my-domain.com",
    titleLink: "/api",
    bestMatch: true,
    analytics: {
      account: 'UA-08150815-0',
      domainName: 'my-domain.com'
    },
    discussions: {
      shortName: 'my',
      url: 'http://my-domain.com',
      dev: false
    }
  },
  tutorial: {
    src: ['content/tutorial/*.ngdoc'],
    title: 'Tutorial'
  },
  api: {
    src: ['src/**/*.js', '!src/**/*.spec.js'], //important
    title: 'API Documentation'
  }
}
```

EX

```
grunt.initConfig({  
  // Project settings  
  ngdocs: {  
    api: {  
      src: ['app/scripts/**/*.js'],  
      title: 'API Documentation'  
    },  
    tutorials: {  
      src: ['docs/tutorials/**/*.ngdoc'],  
      title: 'Tutorials'  
    }  
  },  
},
```

Option meaning: <https://github.com/m7r/grunt-ngdocs>

A doc comment looks like this:

```
/**
 * @ngdoc directive
 * @name rfx.directive:rAutogrow
 * @element textarea
 * @function constructor
 *
 * @description
 * Resize textarea automatically to the size of its text content.
 *
 * **Note:** ie<9 needs pollyfill for window.getComputedStyle
 *
 * @example
 * <example module="rfx">
 *   <file name="index.html">
 *     <textarea ng-model="text" r-autogrow class="input-block-level"></textarea>
 *     <pre>{{text}}</pre>
 *   </file>
 * </example>
 */
angular.module('rfx', []).directive('rAutogrow', function() {
  //some nice code
});
```

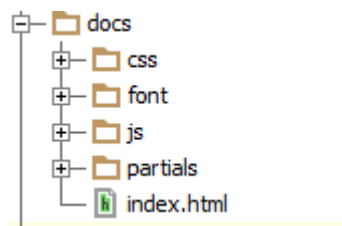
Documentation how-to <https://github.com/m7r/grunt-ngdocs/wiki>

Run “grunt ngdocs” interterminal

[illegible]

```
grunt.registerTask('build', [
  'ngdocs',
  'clean:dist',
  'bowerInstall',
  'useminPrepare',
  'concurrent:dist',
  'autoprefixer',
  'concat',
  'ngmin',
  'copy:dist',
  'cdnify',
  'cssmin',
  'uglify',
  'rev',
  'usemin',
  'htmlmin'
]);
```

After create docs successfully, “docs” folder will be created in root folder (sam level with app folder)



Run index.html in browser to see result

## 9. Naming convention:

- Each filename should describe the file's purpose by including the component or view sub-section that it's in, and the type of object that it is as part of the name. For example, a date-picker directive would be in components/date-picker/date-picker-directive.js.
- Controllers, Directives, Services, and Filters, should include controller, directive, service, and filter in their name.
- File names should be lowercase, following the existing JS Style recommendations. HTML and CSS files should also be lowercase.
- Unit tests should be named ending in \_test.js, hence "foo-controller\_test.js" and "bar-directive\_test.js"
- We prefer, but don't require, that partial templates be named *something-tpl.html* rather than *something.ng*. (This is because, in practice, most of the templates in an Angular app tend to be partials.).

## 10. A Note on Minification:

Since Angular infers the controller's dependencies from the names of arguments to the controller's constructor function, if you were to minify the JavaScript code, all of its function arguments would be minified as well, and the dependency injector would not be able to identify services correctly.

We can overcome this problem by annotating the function with the names of the dependencies, provided as strings, which will not get minified. There are two ways to provide these injection annotations:

- Create a `$inject` property on the controller function which holds an array of strings. Each string in the array is the name of the service to inject for the corresponding parameter. In our example we would write:

```
1. function PhoneListCtrl($scope, $http) {...}
2. PhoneListCtrl.$inject = ['$scope', '$http'];
3. phonecatApp.controller('PhoneListCtrl', PhoneListCtrl);
```

- Use an inline annotation where, instead of just providing the function, you provide an array. This array contains a list of the service names, followed by the function itself.

```
1. function PhoneListCtrl($scope, $http) {...}
2. phonecatApp.controller('PhoneListCtrl', ['$scope', '$http', PhoneListCtrl]);
```



Both of these methods work with any function that can be injected by Angular, so it's up to your project's style guide to decide which one you use.

When using the second method, it is common to provide the constructor function inline as an anonymous function when registering the controller:

```
1. phonecatApp.controller('PhoneListCtrl', ['$scope', '$http', function($scope, $http) {...}]);
```

From this point onward, we're going to use the inline method in the tutorial. With that in mind, let's add the annotations to our [PhoneListCtrl](#):

**app/js/controllers.js:**

```
1. var phonecatApp = angular.module('phonecatApp', []);
2. phonecatApp.controller('PhoneListCtrl', ['$scope', '$http',
3.   function ($scope, $http) {
4.     $http.get('phones/phones.json').success(function(data) {
5.       $scope.phones = data;
6.     });
7.     $scope.orderProp = 'age';
8.   }]);
```

Have fun!