Installation Web Developer   
Desktop

Guide

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Name** | **Date** |
| Author |  | Duc Nguyen | 13-May-2014 |
| Review |  |  | 03-Mar-2014 |
| Approval |  |  |  |

**Content**

[1. Introduction 4](#_Toc387770147)

[1.1 Target audience 4](#_Toc387770148)

[1.2 Glossary 4](#_Toc387770149)

[1.3 References 4](#_Toc387770150)

[2. Install Git-SCM: 5](#_Toc387770151)

[3. Install Ruby and Compass 8](#_Toc387770152)

[4. Install NodeJS 10](#_Toc387770153)

[5. Creating boilder-plate AngularJS project with WebStorm 12](#_Toc387770154)

[6. Install xComponent 16](#_Toc387770155)

[7. I18n filter in component 18](#_Toc387770156)

[8. Template cache 19](#_Toc387770157)

[9. Documentation 21](#_Toc387770158)

[10. Minification Convention 26](#_Toc387770159)

[11. Build problemes 28](#_Toc387770160)

[11.1 Yeoman boilerplate project 28](#_Toc387770161)

[11.2 Build problems with xedu-webapp 31](#_Toc387770162)

[12. Junit reporter 33](#_Toc387770163)

[13. Naming convention 34](#_Toc387770164)

**Changes**

|  |  |  |
| --- | --- | --- |
| 1.0 | 21-Feb-2014 | Initial version |
|  |  |  |
|  |  |  |
|  |  |  |

# Introduction

This document is tutorial for developers to develop web application.

## Target audience

This document is intended primarily for integrators and application developers.

It does not address non-technical people and customers.

Its distribution must remain internal and confidential.

## Glossary

|  |  |
| --- | --- |
| User credentials | Establish the identity of a user, for example in the form of username and password. |
| npm | Node package management |

## References

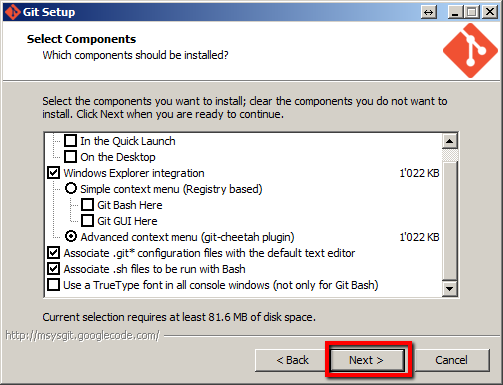
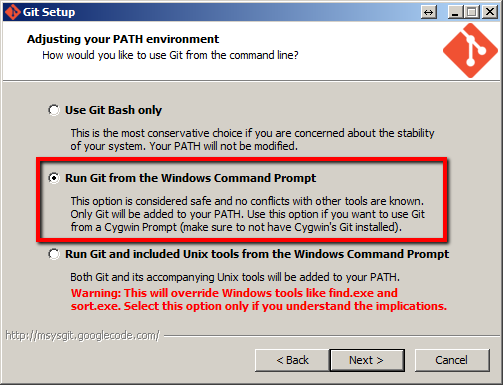
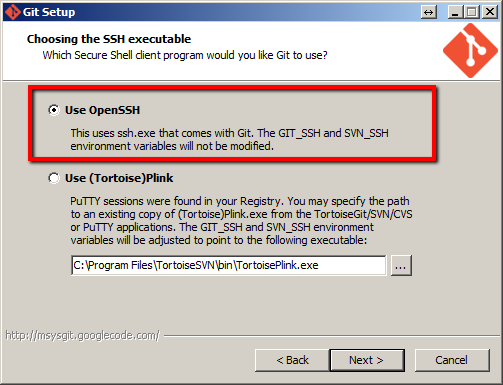
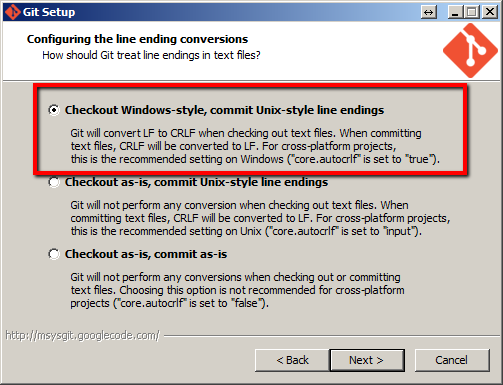
**InstallationWebDeveloperDesktop-***<version>***.docx**

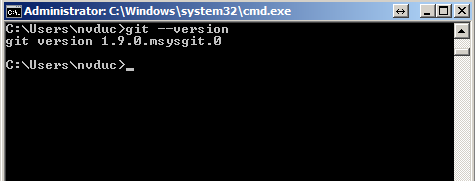
# Install Git-SCM:

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:



# Install Ruby and Compass

We using compass for compile SCSS files to CSS files.

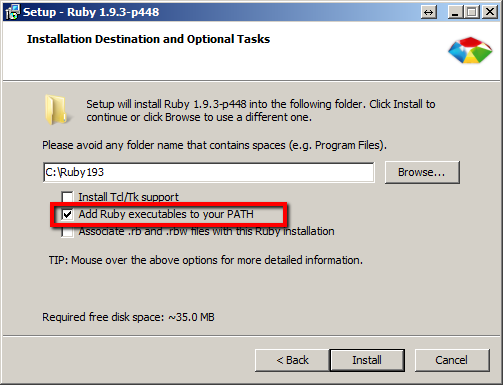
Compass runs on any computer that has Ruby installed.

We have to install Ruby first.

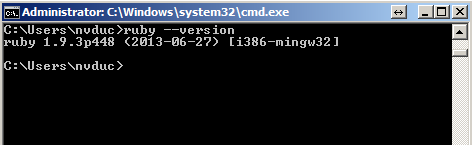
Link: <https://www.ruby-lang.org/en/installation/>

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**

1. Check installed successfully:  
   

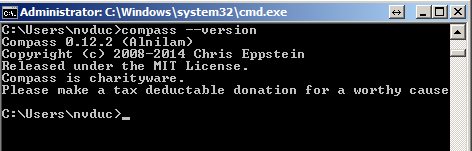
Installation steps for Compass:

1. Run these commands to install Compass:

$gem update --system

$gem install compass

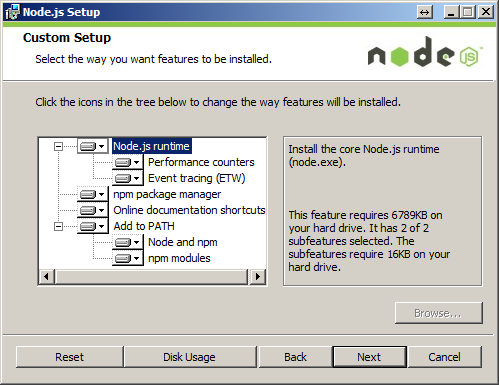
1. Check installed successfully:



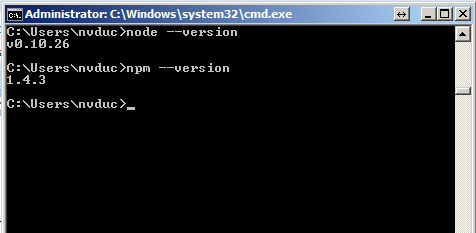
# Install NodeJS

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**.

1. Check installation successfully:  
   

Install some modules:

1. Grunt:  
   $npm install -g grunt-cli
2. Bower:

$npm install -g bower

1. Yeoman:  
   $npm install -g yo

$npm install -g generator-angular

**package.json**

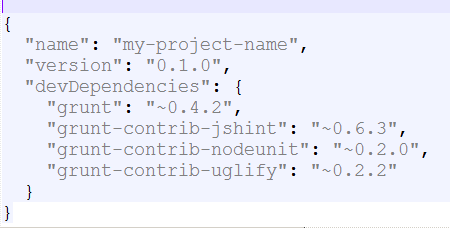
The *package.json* file belongs in the root directory of your project, next to the Gruntfile, and should be committed with your project source. Running **$npm install** in the same folder as a *package.json* file will install the correct version of each dependency listed therein.

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.

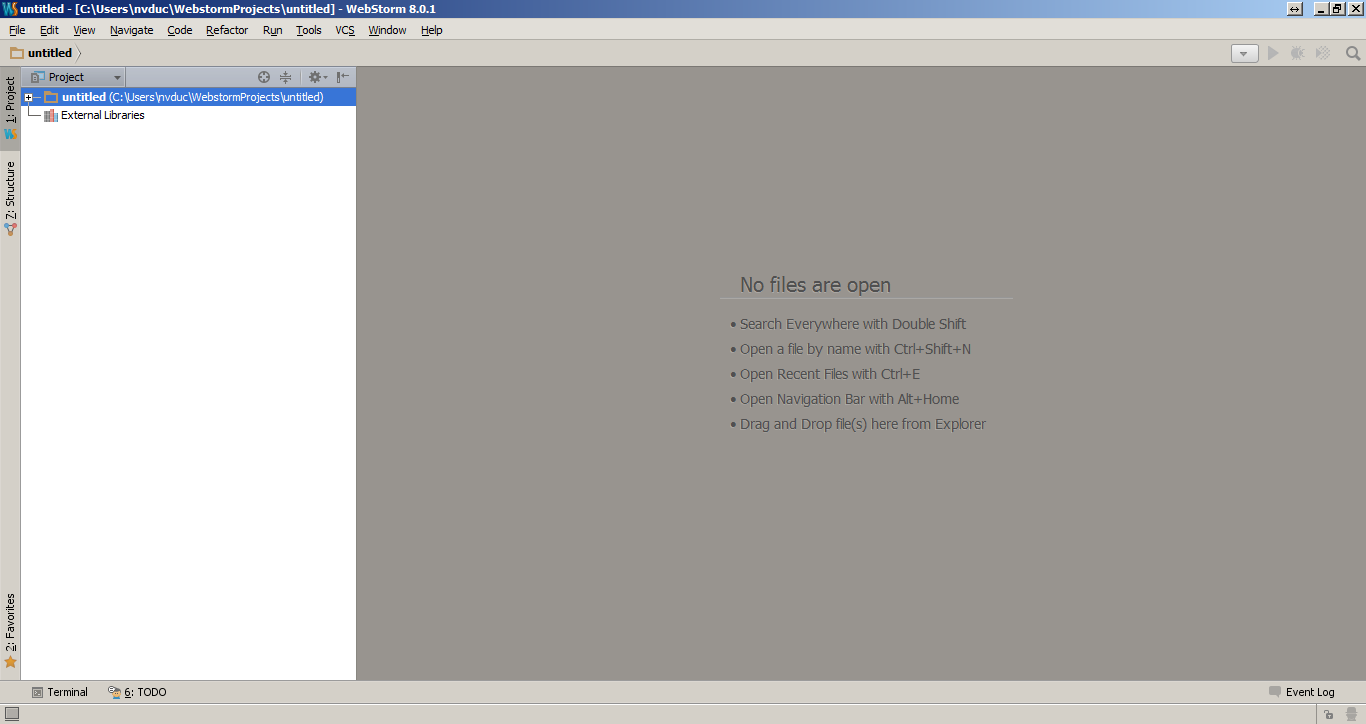
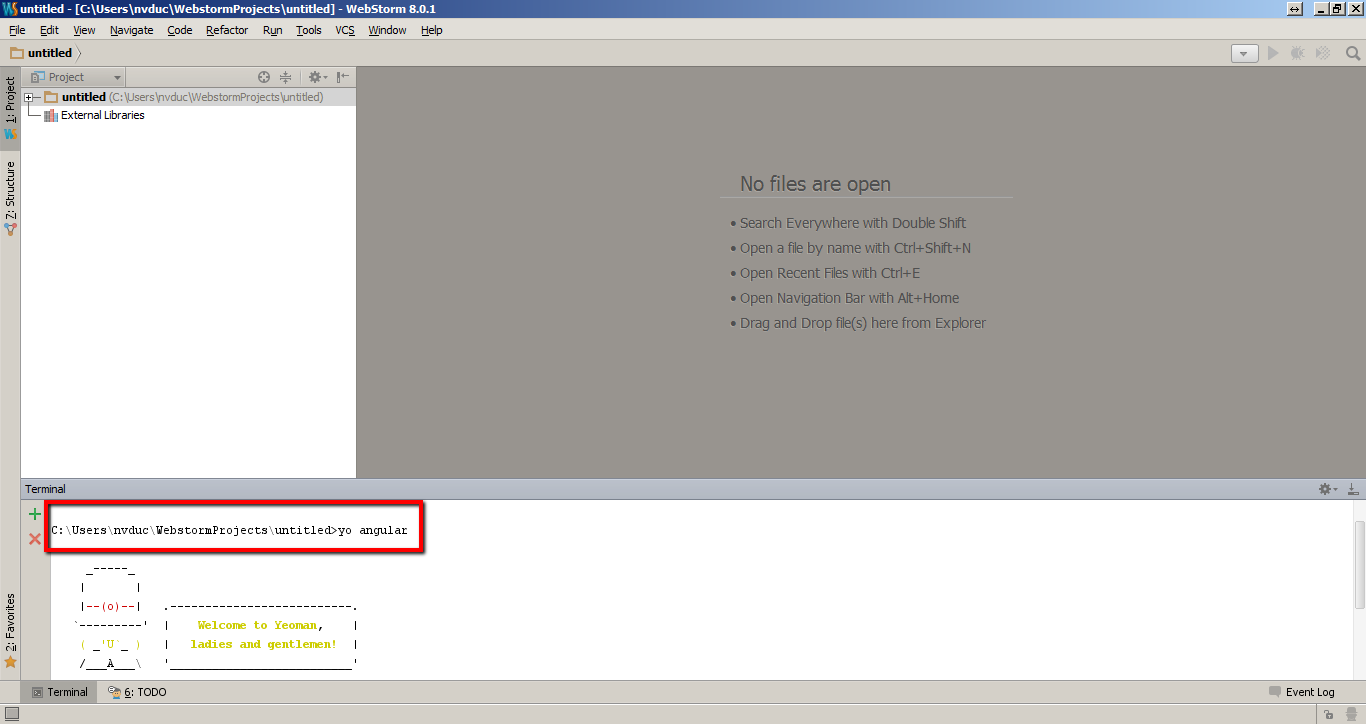
The **$npm init** command will create a basic *package.json* file.

Example:

****

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

# Creating boilder-plate AngularJS project with WebStorm

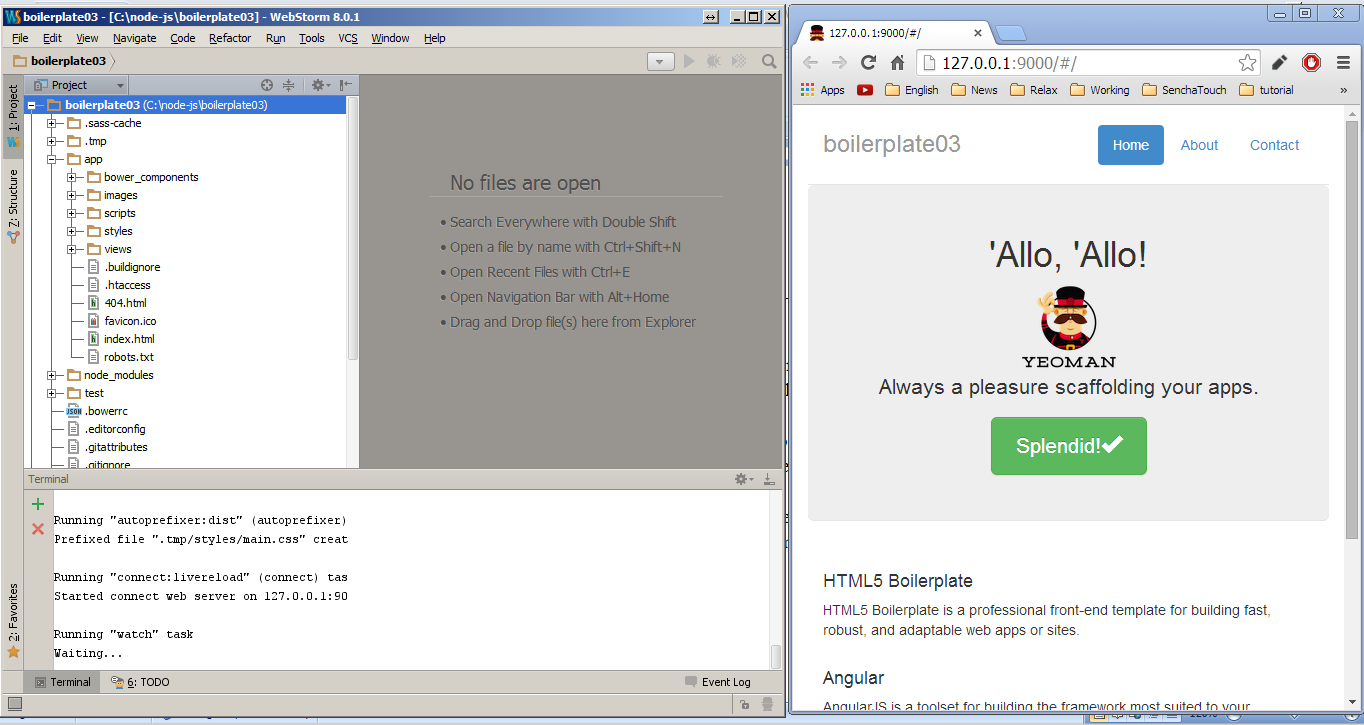
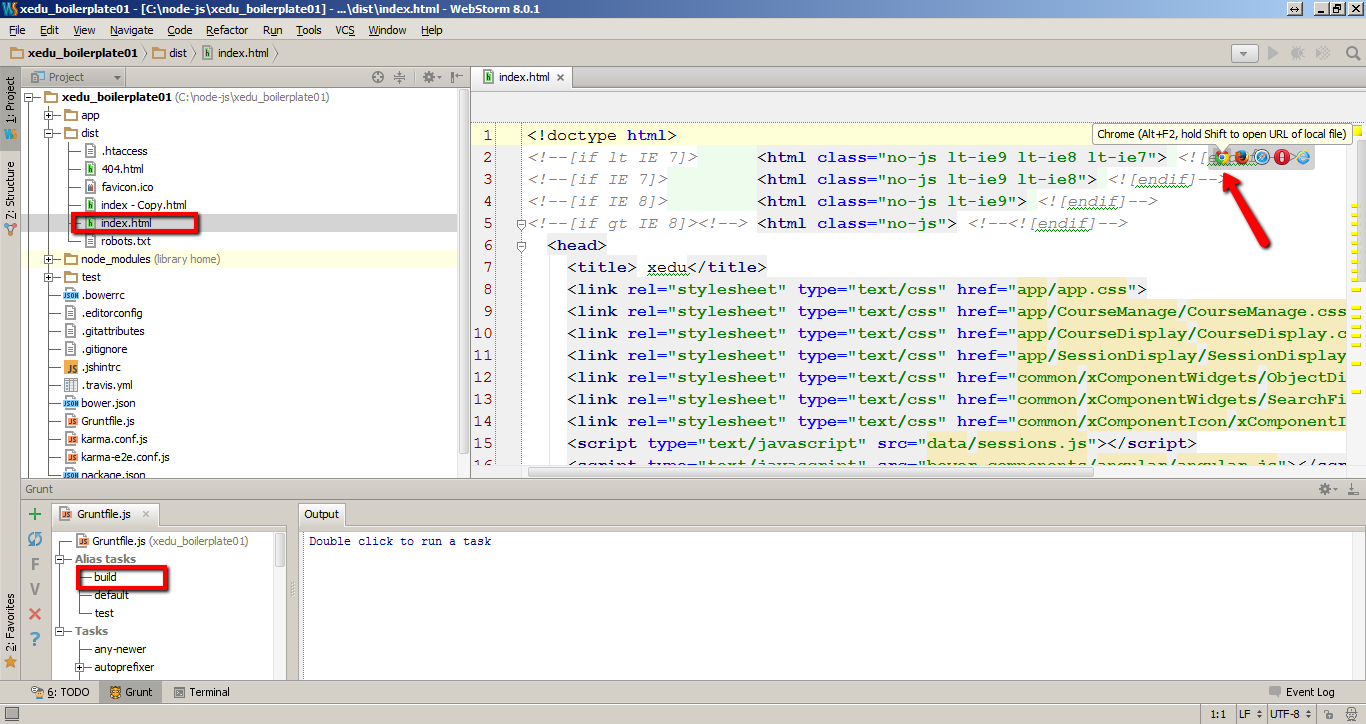
1. Creating new project:  
   Important! Close your WebStorm if it was opening. Reopen it and create new project.  
     
   
2. Tool 🡪 Open Terminal. Run command **$yo angular**  
   

Select some dependencies if you need.

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

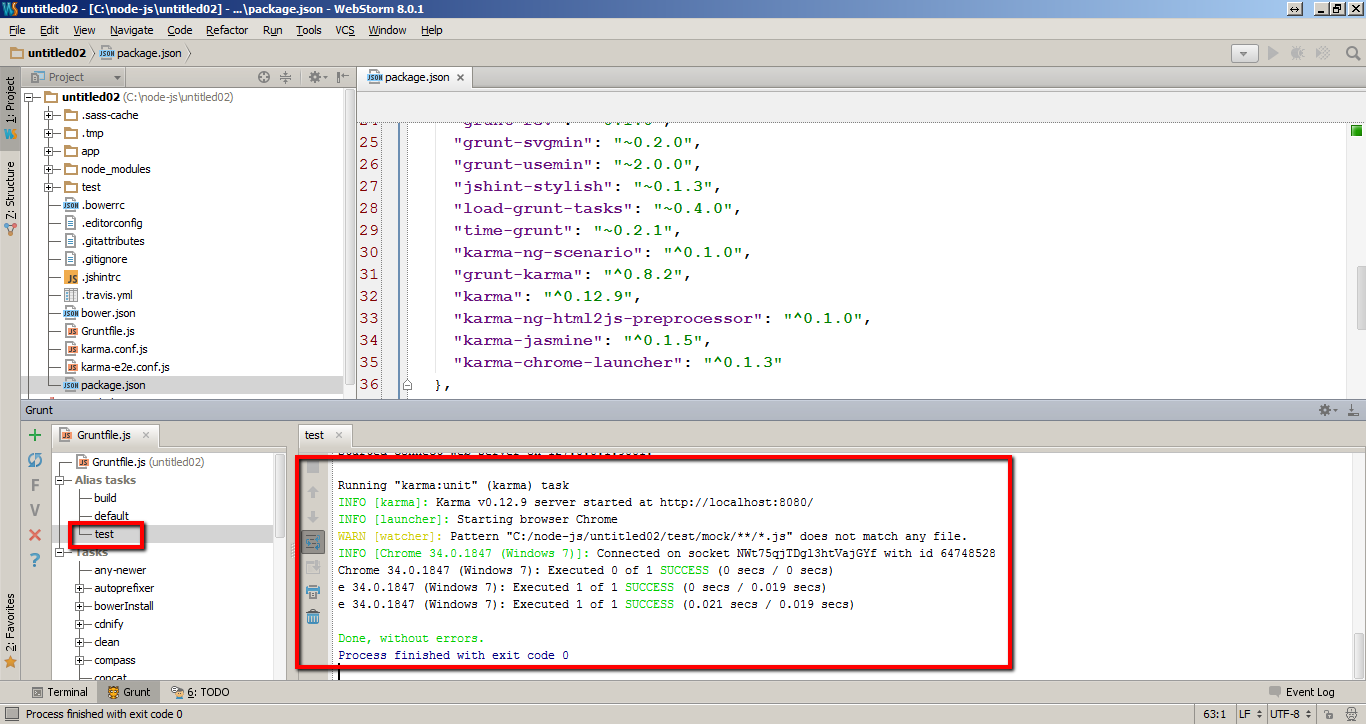
If you have some problems with **grunt** and **bower**. Run these command

**$npm install** and **$bower install** after that run **$grunt serve** again.

1. Now run **$grunt serve** command for check new Boilerplate created.  
   
2. 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.
3. 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.

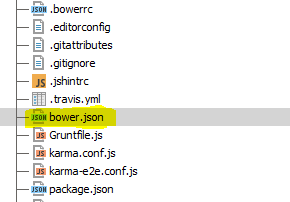
Open Grunt Console 🡪 Click test.

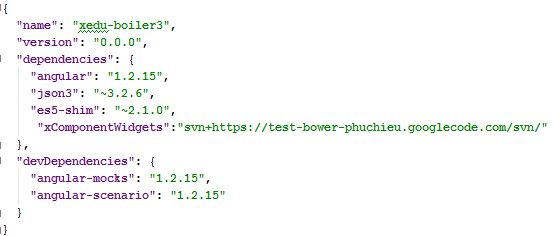


# Install xComponent

Add this line into dependencies property in bower.json

"xComponentWidgets":"svn+https://[svn\_link\_of\_xComponentWidgets]"

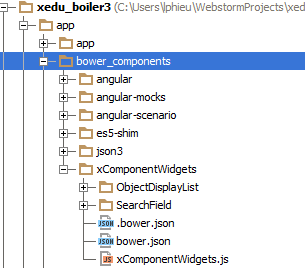




Run in terminal

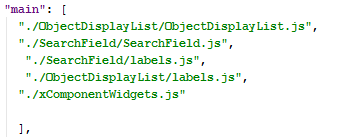
$ bower install

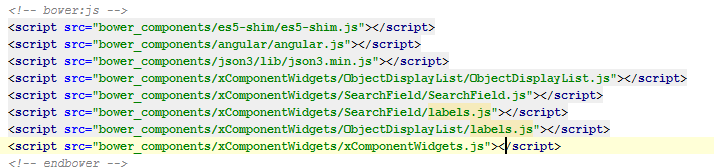
It will download component from svn and put it in bower\_component folder



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

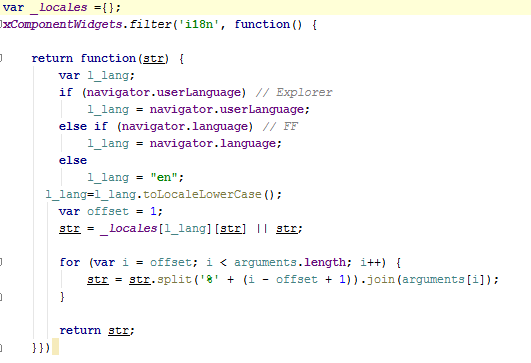
$ grunt bowerInstall



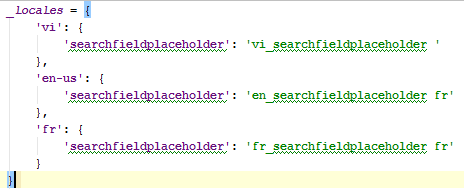


# I18n filter in component

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



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



Using:

In html file:

{{'Hello, world'|i18n}}

In js file:

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

# Template cache

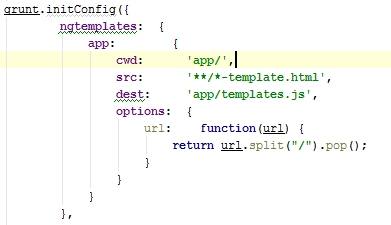
Install the plugin:

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

Enable the plugin within your Gruntfile:

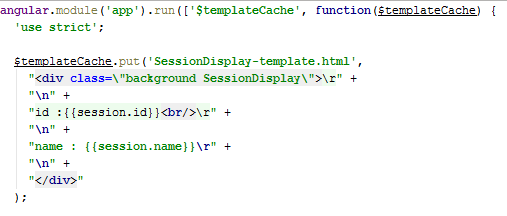
grunt.loadTasks('grunt-angular-templates');

Register HTML Templates by create a task in Gruntfile.js

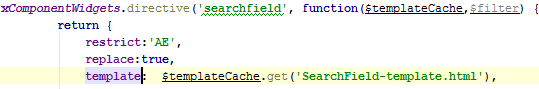


Create template.js

$ grunt ngtemplates



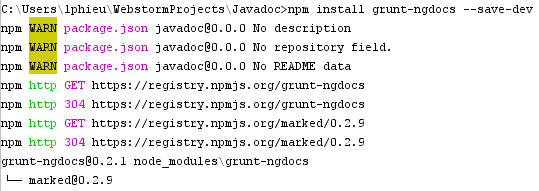
Use template in component like this



# Documentation

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



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**:** {

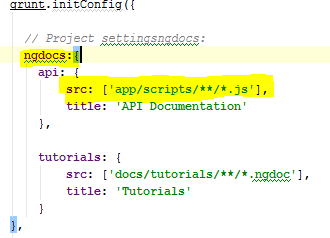
src**:** ['src/\*\*/\*.js', '!src/\*\*/\*.spec.js'], //important

title**:** 'API Documentation'

}

}

EX



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*

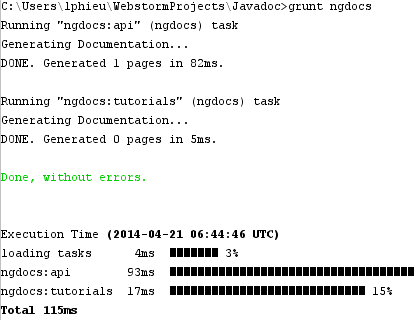
});

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

Build documents folder

Run “grunt ngdocs” interminal

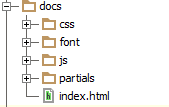
grunt ngdocs



Or add ngdocs task into another task (suchs as build, test task…) by add “ngdocs” in registerTask



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

# Minification Convention

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 ourPhoneListCtrl:

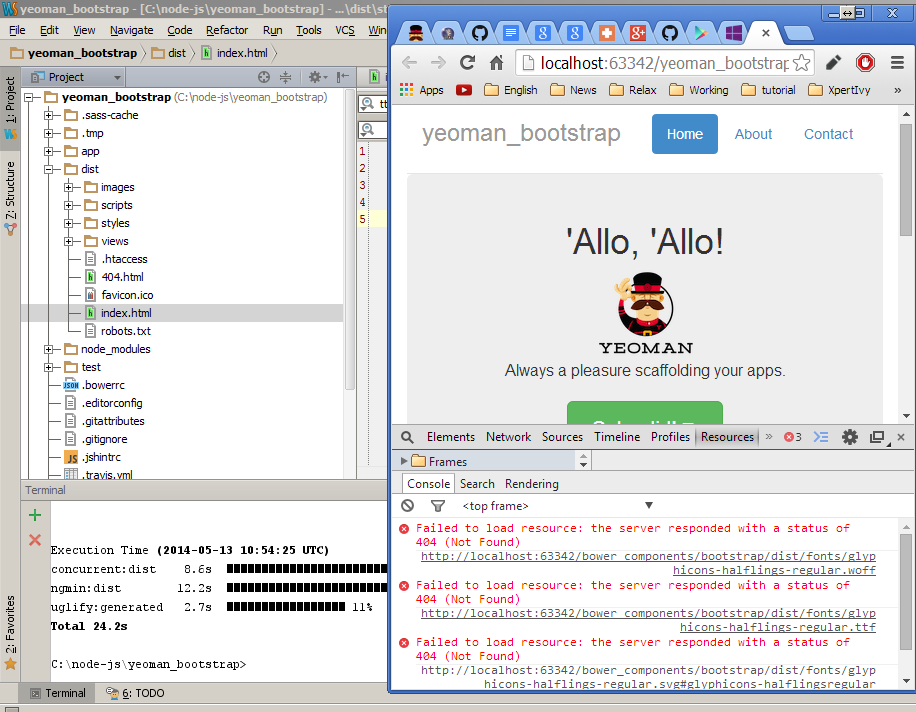
**app/js/controllers.js:**

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. }]);

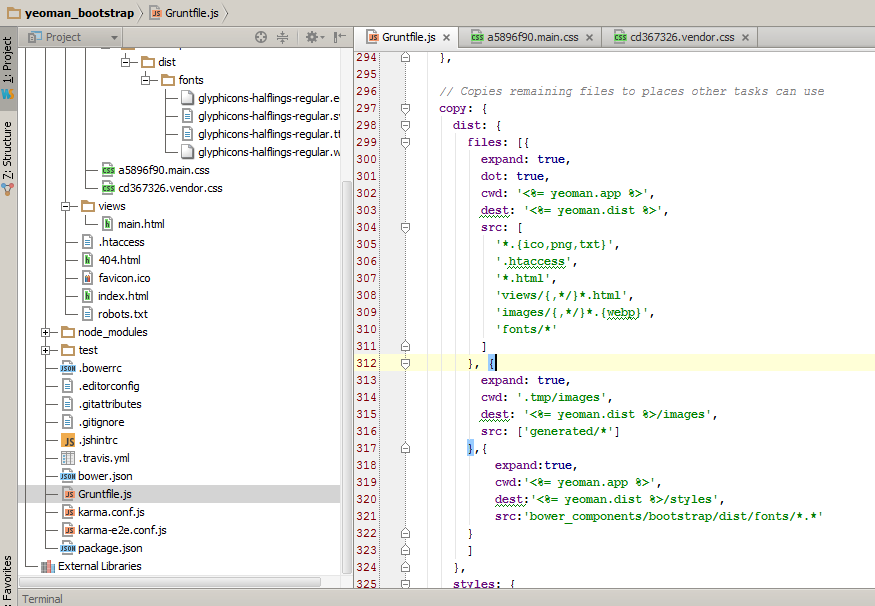
# Build problems

## Yeoman boilerplate project

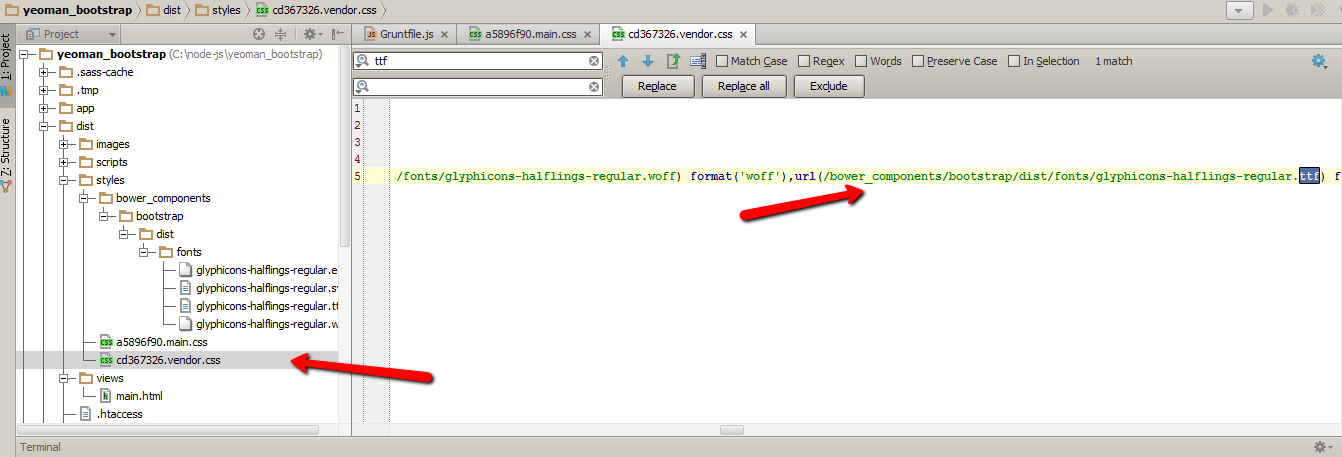
There are problems because grunt build didn’t copy font folder of bootstrap components.



We will copy it by add some like in copy task of grunt file:



We still have problem because **usemine** change URL wrong local:



Add task to change URL correct:

**$npm install grunt-text-replace --save-dev**

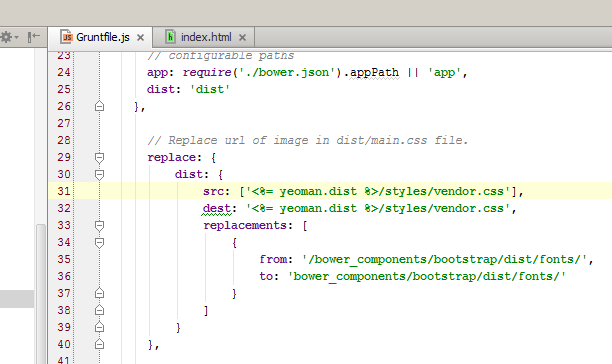


Figure 1. Add replace task to Gruntfile.js

Add new line to task build:



Figure 2. Add replace task before replace:dist

## Build problems with xedu-webapp

In xedu-webapp project add task

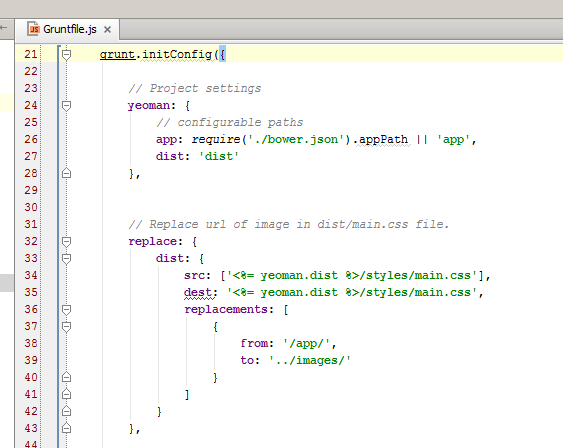


Figure 3. Add task to Gruntfile.js of xedu-webapp

Add new sub task to build task:



Figure 4. Add replace task to build task

# Junit reporter

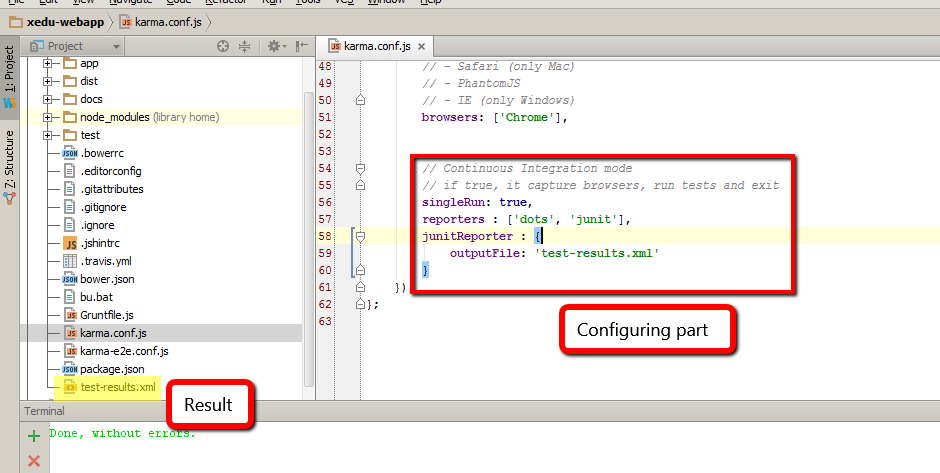
When we using Hudson for continuous integration our project, we need to report result of all test-case for monitoring developer process.

We will use A Karma plugin. The plugin help Karma report results in junit xml format.

Link: https://www.npmjs.org/package/karma-junit-reporter

Install command:

**$npm install karma-junit-reporter**



# 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.).