RESTful webservice setup with Jersey framework

Spring MVC, AngularJS and HTML5 setup in the same project

Prepared by:

Ashish Kumar Mondal

Contents

[Introduction 4](#_Toc415843736)

[Steps to create RESTful project 4](#_Toc415843737)

[Source Code 5](#_Toc415843738)

[Troubleshooting for RESTful webservice setup 5](#_Toc415843739)

[Add spring framework also with this RESTful webservice 6](#_Toc415843740)

[Source code for Spring MVC + RESTful web service 7](#_Toc415843741)

[Add HTML5 in the same project 7](#_Toc415843742)

[Add angular JS in the same project 7](#_Toc415843743)

[Project Structure 8](#_Toc415843744)

[Include AngularJS within the project 8](#_Toc415843745)

[Step1: Include the angular js with the project 8](#_Toc415843746)

[Step2: Mark the jsp file with angularjs 9](#_Toc415843747)

[Step3: Create another .js file which will call GET/POST web service 9](#_Toc415843748)

[Call GET and POST RESTful web service 9](#_Toc415843749)

[GET RESTful webservice call 9](#_Toc415843750)

[POST RESTful webservice call 10](#_Toc415843751)

[Sample Screen 11](#_Toc415843752)

[Source Code 11](#_Toc415843753)

[Troubleshooting for angularjs setup 11](#_Toc415843754)

[Important concepts in AngularJS 12](#_Toc415843755)

[ng-directives 12](#_Toc415843756)

[AngularJS application 14](#_Toc415843757)

[AngularJS filter 15](#_Toc415843758)

[Angular $http (call GET/POST RESTful webservice) 16](#_Toc415843759)

[Form Validation using AngularJS 16](#_Toc415843760)

[Integrate log4J with this Web application 17](#_Toc415843761)

[Spring Security Implementation in html5 login page 18](#_Toc415843762)

[Steps 18](#_Toc415843763)

[Source code 19](#_Toc415843764)

[Screenshots 20](#_Toc415843765)

[Spring Dependency Injection 21](#_Toc415843766)

[Annotation based DI 21](#_Toc415843767)

[Source code 21](#_Toc415843768)

# Introduction

This document will help user to setup a RESTful webservice with Jersey framework.

**Assumption**: J2EE eclipse (e.g. Eclipse Kepler), JDK1.7 and maven is already available in the system

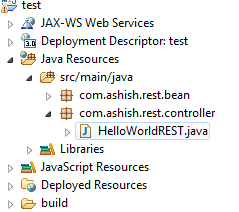
(If you need maven setup, follow my another document in Github "Maven build setup for your project.docx"

# Steps to create RESTful project

Best way to learn the RESTful web service is download a small helloworld project and check the pom.xml, web.xml, java file details as mentioned below

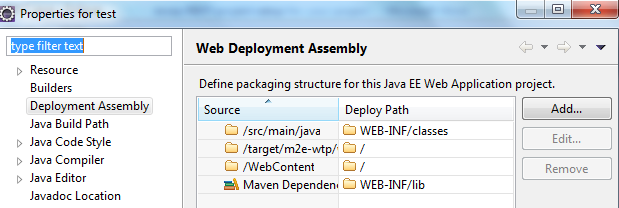
However, you can create a new project from scratch as mentioned below.

1. Create a Dynamic web project with module version 3.0 and java source directory must be src/main/java



1. Convert the project into maven project (right click on project-> Configure -> Convert to Maven project)
2. Create a package under java source (src/main/java): com.ashish.rest.controller
   1. Right click on project->properties->Deployment Assembly->Add->Java Build Path Entries->Maven Dependencies (Note: Deploy path should be WEB-INF/lib by default)

The meaning of the below entry is the dependent jars will go to WEB-INF/lib folder in the deployable



1. pom.xml should have below contains



1. Write the below java file



1. Modify web.xml with the following content



1. You may need to update maven project by the following option (right click on the project->Maven->update maven project)

## Source Code



## Troubleshooting for RESTful webservice setup

|  |  |  |
| --- | --- | --- |
| **SL No** | **Issue** | **Solution** |
| 1 | SEVERE: Servlet /JAXRS-HelloWorld threw load() exception  java.lang.ClassNotFoundException: com.sun.jersey.spi.container.servlet.ServletContainer | Right click on project->properties->Deployment Assembly->Add->Java Build Path Entries->Maven Dependencies (Note: Deploy path should be WEB-INF/lib by default) |
| 2 | com.sun.jersey.api.container.ContainerException: The ResourceConfig instance does not contain any root resource classes. | a) com.sun.jersey.config.property.packages doesn’t exist in your web.xml  <servlet>  <servlet-name>jersey-helloWorld-serlvet</servlet-name>  <servlet-class>  com.sun.jersey.spi.container.servlet.ServletContainer </servlet-class>  <init-param>  <param-name>  com.sun.jersey.config.property.packages  </param-name>  <param-value>com.ashish.rest.controller</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  b) com.sun.jersey.config.property.packages included a resource that doesn’t include any jersey services. In above case, "com.ashish.rest.controller” doesn’t contains any jersey services.  c) The project's java source directory must be under src/main/java folder as the project is of type Maven |
| 3 | Caused by: com.sun.jersey.api.MessageException: A message body writer for Java class com.ashish.rest.bean.Employee, and Java type class com.ashish.rest.bean.Employee, and MIME media type application/json was not found | Add below dependency in pom.xml  <dependency>  <groupId>com.sun.jersey</groupId>  <artifactId>jersey-json</artifactId>  <version>1.9.1</version>  </dependency>  <dependency>  <groupId>com.sun.jersey</groupId>  <artifactId>jersey-bundle</artifactId>  <version>1.18.1</version>  </dependency>  Add below entry in web.xml  <init-param>  <param-name>  com.sun.jersey.api.json.POJOMappingFeature  </param-name>  <param-value>true</param-value>  </init-param> |
|  |  |  |

# Add spring framework also with this RESTful webservice

1. Add spring mvc dependencies in pom.xml
2. Add the following xsd at the header of the pom.xml (value of xsi:schemaLocation=)

<http://maven.apache.org/maven-v4_0_0.xsd>



1. Edit default web.xml with the spring mvc related entries as attached



1. Add a file called dispatcher-servlet.xml under WEB-INF folder



1. Add index.jsp inside WebContent folder



1. Add helloworld.jsp under WEB-INF/views folder



## Source code for Spring MVC + RESTful web service



# Add HTML5 in the same project

Add html5 tags in a JSP called html5AngularJsView.jsp. Below is the way to achieve this

1. Add the following lines at the beginning of the jsp file

<%@page contentType=*"text/html"* pageEncoding=*"UTF-8"*%>

<!DOCTYPE html> <!-- DOCTYPE for html5 -->

1. Then add charset="UTF-8" in the head section of the html5

<head>

<meta charset=*"UTF-8"*>

</head>



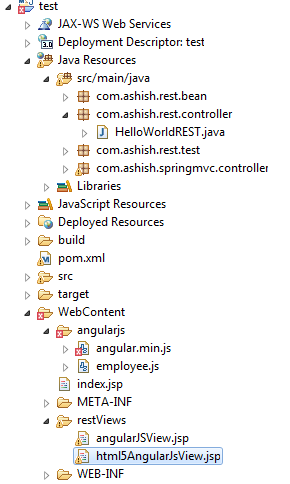
# Add angular JS in the same project

Angularjs is very popular for CRUD operation. It is one of the best framework for data binding with browser.

In this example I have shown

* Project Structure
* How to include angular js with the project.
* How to call GET and POST method of RESTful web service

## Project Structure



## Include AngularJS within the project

The purpose of the project is to call GET and POST RESTful web service which is returning JSON data from server. If you are very new to the angular JS then go through [this section](#_Important_concepts_in) first

### Step1: Include the angular js with the project

Include *angular.min.js* inside the <head> tag . Add the file as local copy or the copy from google CDN server.

<head>

<meta charset=*"UTF-8"*>

<title>Integrate HTML5 and Angular JS</title>

<!-- <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script> -->

<script src=*"*<%=request.getContextPath()%>*/angularjs/angular.min.js"*></script>

<script src="<%=request.getContextPath()%>/angularjs/employee.js"></script>

</head>

### Step2: Mark the jsp file with angularjs

Mark the JSP file with the angular js using ng-app attribute in <html> tag

<html ng-app="empRecordApp">

### Step3: Create another .js file which will call GET/POST web service

Created a file called employee.js which is placed inside ***WebContent/angularjs/*** folder



## Call GET and POST RESTful web service

### GET RESTful webservice call

In the jsp file have the below code

<div ng-controller=*"getEmployee"*>

<div>

<p>The Employee ID is {{employeeData.empId}}</p>

<p>The Employee Name is {{employeeData.name}}</p>

</div>

<div>

EmpId: <input type=*"text"* placeholder=*"Emp Id"* ng-model=*"employeeData.empId"*>

Name: <input type=*"text"* placeholder=*"Name"* ng-model=*"employeeData.name"*>

</div>

</div>

In the employee.js file write the below code

/\*

\* In this below example, the GET webservice got called inside the controller

\*/

**var** empRecordAppCtrl = angular.module('empRecordApp', []);

empRecordAppCtrl.controller('getEmployee', **function**($scope, $http) {

$http.get("http://localhost:8080/test/rest/hello/getEmployee/123")

.success(**function**(response) {

$scope.employeeData = response;

});

});

Explanation: ng-controller directive is mentioned in the <div> tag which calls the *getEmployee* controller mentioned in the *employee.js* file. The controller calls the GET RESTful webservice

### POST RESTful webservice call

In the jsp file have the below code

<div ng-controller=*"getSalary"*>

<div>

EmpId: <input type=*"text"* placeholder=*"Name"* ng-model=*"employeeData.name"*>

Salary: <input type=*"text"* placeholder=*"Salary"* ng-model=*"employeeData.salary"*>

</div>

<div>

<p>The Employee Name is {{employeeData.name}}</p>

<p>The Employee Salary is {{employeeData.salary}}</p>

</div>

In the employee.js file write the below code

/\*

\* In this below example, the POST webservice got called inside the controller

\*/

**var** empRecordAppCtrl = angular.module('empRecordApp', []);

empRecordAppCtrl.controller('getSalary', **function**($scope, $http) {

$http.post("http://localhost:8080/test/rest/hello/getSalary", {empId:'123'})

.success(**function**(response) {

$scope.employeeData = response;

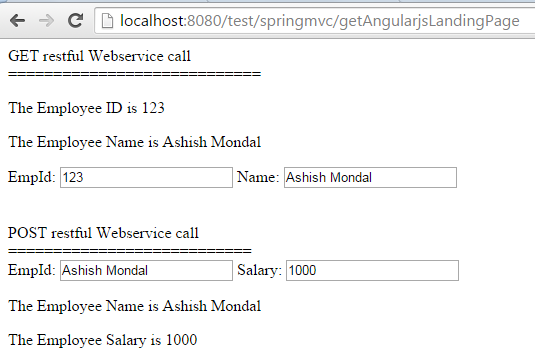
});

});

Explanation: ng-controller directive is mentioned in the <div> tag which calls the *getSalary* controller mentioned in the *employee.js* file. The controller calls the POST RESTful webservice

### Sample Screen



## Source Code



## Troubleshooting for angularjs setup

|  |  |  |
| --- | --- | --- |
| **SL No** | **Issue** | **Solution** |
| 1 | Not able to access javascript file placed under WebContent folder in Tomact7+springmvc | Servlet mapping changed from  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  TO  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/springmvc/\*</url-pattern>  </servlet-mapping>  Otherwise it was looking for URL mapping so <http://localhost:8080/test/angularjs/employee.js> was not accessible |
| 2 |  |  |
| 3 |  |  |

## Important concepts in AngularJS

* AngularJS extends HTML with ng-directives

### ng-directives

|  |  |  |
| --- | --- | --- |
| **ng-directives** | **Description** | **Example** |
| **ng-app** | Defines an AngularJS application   * This is root element of angularJS application * This will **auto-bootstrap** (automatically initialize) the application when a web page is loaded * can have a value (like ng-app="*directiveExample* "), to connect code modules | <!DOCTYPE html>  <html>  <head>  <script src= *"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"*></script>  </head>  <body>  <div ng-app=*"directiveExample"*>  <p>Input something in the input box:</p>  <p>Name: <input type=*"text"* ng-model=*"name"*></p>  ng-bind example: <p ng-bind=*"name"*></p>  An alternative to ng-bind is: {{name}}  </div>  </body>  </html>  **Screenshot** |
| **ng-model** | Binds the value of HTML controls (input, select, textarea) to application data.   * Provide type validation for application data (number, email, required). * Provide status for application data (invalid, dirty, touched, error). * Provide CSS classes for HTML elements * Bind HTML elements to HTML forms |
| **ng-bind** | Binds application data to the HTML view |
| **ng-init** | defines **initial values** for an AngularJS application | <div ng-app=*""* ng-init=*"names=['Ashish','Dona','Ujan']"*>  <p>Looping with ng-repeat:</p>  <ul>  <li ng-repeat=*"x in names"*>  Name: {{ x }} <br/>  Alternative way: <span ng-bind=*"x"*/>  </li>  </ul>  </div>  **Screenshot** |
| **ng-repeat** | repeats an HTML element |
| **ng-disabled** | binds AngularJS application data to the disabled attribute of HTML elements | <!DOCTYPE html>  <html>  <head>  <script src= *"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"*></script>  </head>  <body>  <div ng-app=*""* ng-init=*"mySwitch=true"*>  <p>  <button ng-disabled=*"mySwitch"*>Click Me!</button>  </p>  <p>  <input type=*"checkbox"* ng-model=*"mySwitch"*/>Button  </p>  <p>  Button disabled: {{ mySwitch }}  </p>  </div>  </body>  </html>  **Screenshots** |
| **ng-show/ng-hide** | shows or hides an HTML element | <body>  <div ng-app=*"myUserCtrlApp"* ng-controller=*"personCtrl"*>  <div ng-if=*"displayUser"*>  <button ng-click=*"toggleUserDtls()"*>Display user</button>  </div>  <div ng-if=*"!displayUser"*>  <button ng-click=*"toggleUserDtls()"*>Hide user</button>  </div>  <p ng-hide=*"displayUser"*>  First Name: <input type=*text* ng-model=*"fName"*><br>  Last Name: <input type=*text* ng-model=*"lName"*><br><br>  Full Name: {{fName + " " + lName}}  </p>  </div>  <script>  **var** app = angular.module('myUserCtrlApp', []);  app.controller('personCtrl', **function**($scope) {  $scope.fName = "Ashish",  $scope.lName = "Mondal"  $scope.displayUser = **false**;  $scope.toggleUserDtls = **function**() {  $scope.displayUser = !$scope.displayUser;  }  });  </script>  </body>  **Screenshots** |
| **ng-click** | defines an AngularJS click event |
| **ng-if** | Defines condition |

### AngularJS application

AngularJS application has two major parts called **modules** and **controllers**

**Module(ng-app)** defines the Angular js application

**Controller(ng-controller)** controls AngularJS application

**Example**

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <script src= *"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"*></script>  </head>  <body>  <p>Try to change the name</p>  <div ng-app=*"angularJSApp"* ng-controller=*"agularJSController"*>  First Name: <input type=*"text"* ng-model=*"firstName"*><br/>  Last Name: <input type=*"text"* ng-model=*"lastName"*><br/>  <br/>  Full Name: {{firstName + " " + lastName}}  </div>  <script>  **var** app = angular.module('angularJSApp', []); // This is angular js module  app.controller('agularJSController', **function**($scope) { // This is angular js controller  $scope.firstName= 'Ashish';  $scope.lastName= 'Mondal';  });  </script>  </body>  </html>  **Screenshot** |

### AngularJS filter

Filter is used to transform data. Following filters are available

|  |  |  |  |
| --- | --- | --- | --- |
| **Filter** | **Description** | | **Example** |
| currency | Format a number to a currency format | **HTML File Contains**  <!DOCTYPE html>  <html ng-app=*"myFilterApp"*>  <head>  <title></title>  <script type=*"text/javascript"* src=*"lib/angular.js"*></script>  <script type=*"text/javascript"* src=*"js/controller.js"*></script>  </head>  <body>  <form>  <div ng-controller=*"myFilterController"*>    Enter principal amount : <input type=*"text"* ng-model=*"principal"* placeholder=*"Principle Amount"* maxlength=*"10"* autocomplete=*"off"*/> <br/>  Enter rate of interest : <input type=*"text"* ng-model=*"rate"* placeholder=*"Rate of Interest"* maxlength=*"2"* autocomplete=*"off"*/><br/>  Enter Tenure of the loan (Years) : <input type=*"text"* ng-model=*"duration"* placeholder=*"Tenure of the Loan"* maxlength=*"2"* autocomplete=*"off"*/><br/>  <br/><br/><span>Interest amount using ng-bind</span>  <h3 ng-bind=*"((principal \* rate \* duration) /100) | currency"*></h3>  <h2>Interest amount is {{((principal \* rate \* duration) /100) | currency}}</h2>    </div>  </form>  </body>  </html>  **js/controller.js Contains**  **var** myFilterController = angular.module('myFilterApp', []);  myFilterController.controller('myFilterController', **function**($scope) {  $scope.principal = '';  $scope.rate='';  $scope.duration='';  $scope.interest = $scope.principal \* ($scope.rate/100) \* $scope.duration;    });  **Screenshot** | |
| filter | Select a subset of items from an array |
| lowercase | Format a string to lower case |
| orderBy | Orders an array by an expression |
| uppercase | Format a string to upper case |

### Angular $http (call GET/POST RESTful webservice)

The [above code setup](#_Include_AngularJS_within) will help you to understand the code

### Form Validation using AngularJS

Below example validates username and email id.

In the below example *$dirty,$invalid* properties are used. **$dirty** means user has interacted with the field, and **$invalid** means the field contains invalid

<form ng-app=*""* ng-init=*"data={'user':'ashish', 'email':'amo@gmail.com'}"* name=*"myForm"* novalidate>

<p>Username:<br>

<input type=*"text"* name=*"user"* ng-model=*"data.user"* required>

<span style="color:*red*" ng-show=*"myForm.user.$dirty && myForm.user.$invalid"*>

<span ng-show=*"myForm.user.$error.required"*>Username is required.</span>

</span>

</p>

<p>Email:<br>

<input type=*"email"* name=*"email"* ng-model=*"data.email"* required>

<span style="color:*red*" ng-show=*"myForm.email.$dirty && myForm.email.$invalid"*>

<span ng-show=*"myForm.email.$error.required"*>Email is required.</span>

<span ng-show=*"myForm.email.$error.email"*>Invalid email address.</span>

</span>

</p>

<p>

<input type=*"submit"*

ng-disabled=*"myForm.user.$dirty && myForm.user.$invalid ||*

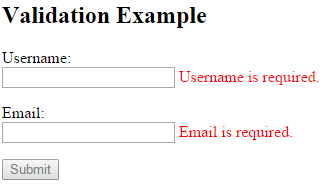
*myForm.email.$dirty && myForm.email.$invalid"*>

</p>

</form>

**Screenshots**





# Integrate log4J with this Web application

1. Add maven dependency in the pom.xml file

<dependency>

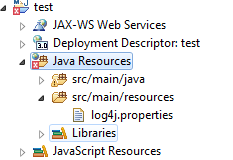
<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

1. Create a resource folder under Java Resources (src/main/resources) as shown below. So maven build will automatically pickup properties files from this folder



1. Add log4j.properties file as attached below. Define the log level as shown below and output file path. Open the attachment for more details

*log4j.rootLogger=DEBUG, stdout, file*

*log4j.appender.file.File=D:\\Ashish\\log4j-application.log*



# Spring Security Implementation in html5 login page

Spring can be configured to provide login and logout capabilities to an application. Spring provides a default login page that can be made available by simply turning on a variable in the spring configuration file. However, in most cases we would like to use our own login page and then delegate the request to spring login URL

The following topics will be covered

* Configure a custom Login page (Spring MVC)
* Configure a custom 403 page (Spring MVC)
* Configure two-tier security based on regular users and administrators
* Use annotation based authorization directly on the Java classes (or methods)
* Use HTTP basic as your authentication mechanism

## Steps

1. **Add maven dependencies for spring security as mentioned in the pom.xml**



1. **Below configuration added in web.xml to integrate spring security**

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/dispatcher-servlet.xml

/WEB-INF/security-applicationContext.xml

</param-value>

</context-param>

………..

<listener>

<listener-class>

org.springframework.web.context.request.RequestContextListener

</listener-class>

</listener>

…………

<!-- Security Filter -->

<!-- Any request for RESTful service or SpringMVC service has to pass through spring security filter -->

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

<init-param>

<param-name>contextAttribute</param-name>

<param-value>org.springframework.web.context.WebApplicationContext.ROOT</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>



1. Add security-applicationContext.xml under WEB-INF folder. Each and every line is explained inside the attached file.



1. Add the following jsp files inside WEB-INF/views folder



1. Add LoginController.java controller

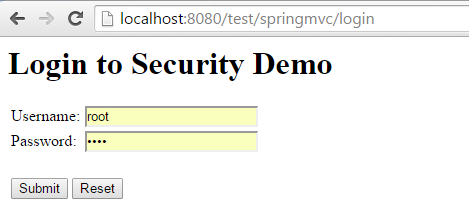
## Source code

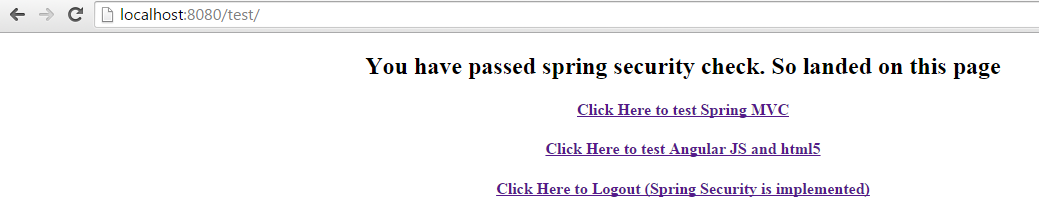
Attached source code has followings technologies integrated

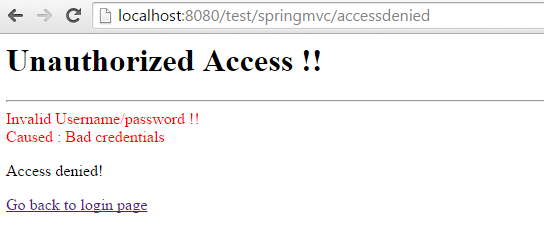
1. Jersey RESTful webservice
2. Spring mvc
3. Spring security
4. Log4j
5. angularJs
6. html5



## Screenshots







# Spring Dependency Injection

## Annotation based DI

Important annotated classes are @Autowire and @Service

In this example:

Class LoginService implements LoginInterface and injected in the LoginController

Below are the configurations

1. dispatcher-servlet.xml file should have below entry

<!-- With this below configuration, annotation based dependency injection can be achieved -->

<context:component-scan base-package=*"com.ashish.springmvc"* />

<mvc:annotation-driven />

1. LoginService.java should have below code

/\*\*

\* This is service class. **@Service** informs IOC container about the dependency injection

\* **@author** Ashish Mondal

\*

\*/

@Service

**public** **class** LoginService **implements** LoginInterface {

**public** **boolean** isLoginSuccess() {

**return** **true**;

}

}

1. LoginController should have below code to complete dependency injection

// Below annotation is responsible for dependency injection

@Autowired

**public** **void** setLoginService(LoginInterface loginService) {

**this**.loginService = loginService;

}

### Source code

This source code has the following

1. Jersey RESTful webservice
2. Spring mvc
3. Spring security
4. Log4j
5. angularJs
6. html5
7. Annotation based Dependency Injection

