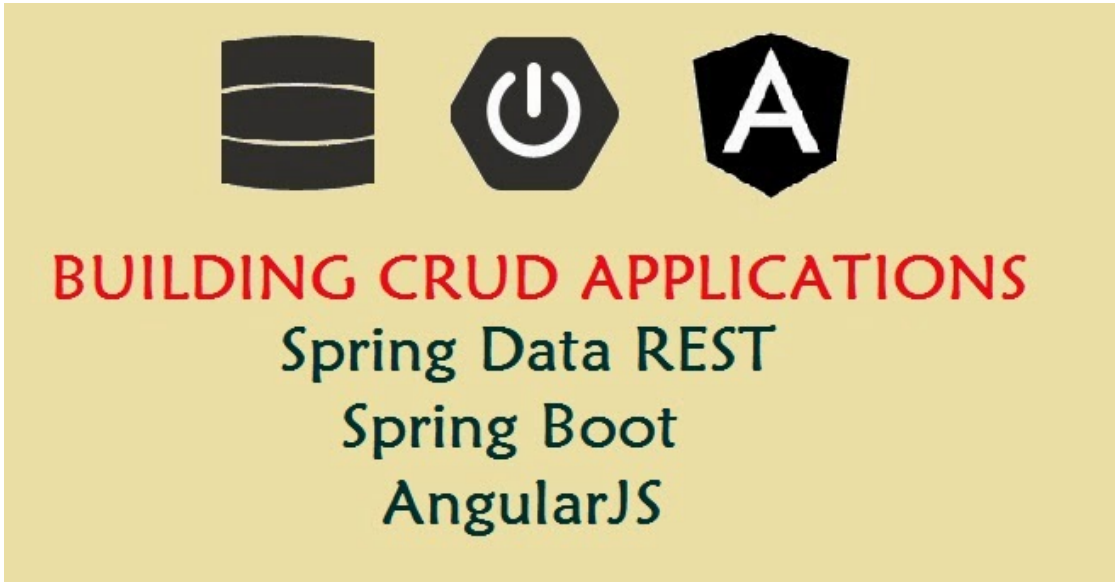
PROGRAMMINGFREE

CRUD using Spring Data Rest and AngularJS using Spring Boot

http://www.programming-free.com/2014/07/spring-data-rest-with-angularjs-crud.html



In my previous article, I explained how to make crud operations using plain spring restful web services and angularjs with a sample task manager application. That was very simple, yet it had several shortcomings that needed a fix in order to move towards reducing most of the boilerplate code, adhering to the latest methodologies suggested by Spring Framework and to follow better code practices. It is better late than never. Having mentioned that, the previous implementation will still work and to start with as a beginner, it is a good choice to have a read through it.

My sincere thanks to Greg L. Turnquist and Josh Long for helping me to understand the best approach and the way to go forward with Spring Framework. Special thanks to Greg, who also fine tuned task manager project to use Spring Data Rest and Spring Boot. In this article I am going to give you an overview of how to use Spring Boot to setup and build Spring project and how to use Spring Data REST to efficiently implement database operations in a RESTFul manner, with the same task manager application used in my previous article.

If you are not well versed in the latest releases of Spring and are used to the XML way of configuring things like me, it is time to learn "SPRING BOOT". Do not hesitate, as it is very simple and once you start using it you will never regret the decision of spending some time to learn it. Lets get started now.

DEMO DOWNLOAD

Article Recognitions*

- A mention about this article in official spring blog here.
- Big Link in Dzone

Spring Boot can be used with build tools such as Maven or Gradle. These build tools help you share jars between various applications, build your application and generate reports. There are many articles on how to install and use Maven and I am going to explain how to install Maven plugin to Eclipse for our use in this project.

Install Maven in Eclipse IDE

Open Eclipse IDE
Click Help -> Install New Software...
Click Add button at top right corner
At pop up: fill up Name as "M2Eclipse" and Location as "http://download.eclipse.org/technology/m2e/releases"
Now click OK
After that installation would be started.

Another way to install Maven plug-in for Eclipse:

Open Eclipse
Go to Help -> Eclipse Marketplace
Search by Maven
Click "Install" button at "Maven Integration for Eclipse" section
Follow the instruction step by step

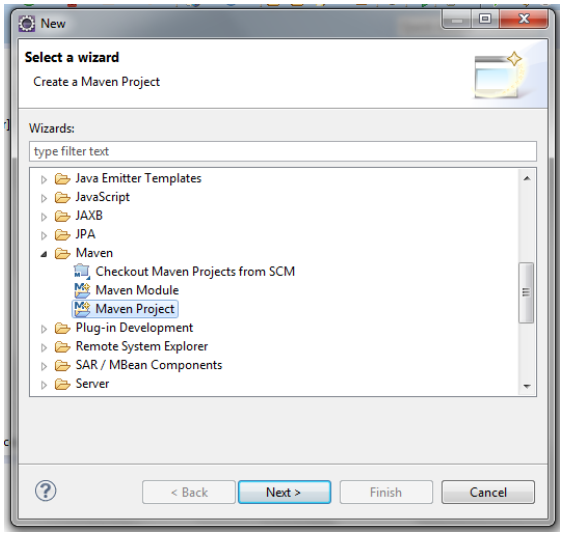
After successful installation do the followings in Eclipse:

1) Go to Window --> Preferences

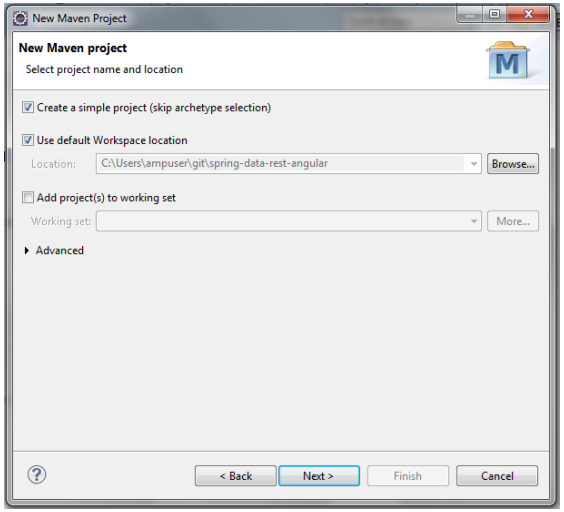
2) Observe, Maven is enlisted at left panel

Set up project with Spring Boot

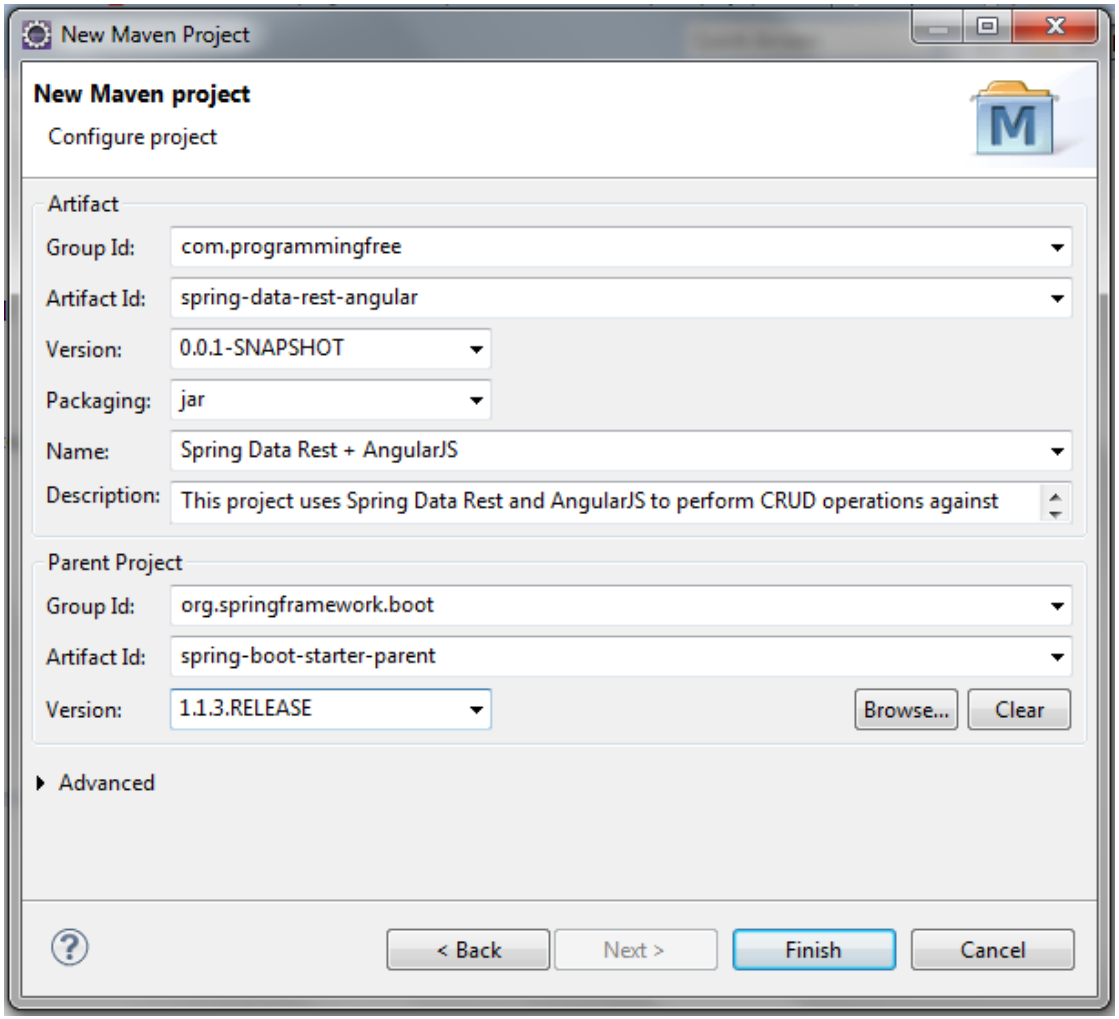
1. Go to New -> Maven Project in Eclipse,



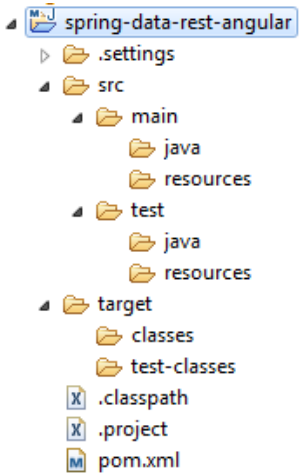
2. Click Next -> Check Create a simple project -> Give workspace location



3. Click Next -> Enter configurations as seen in the screenshot below



Now as the application configuration is done, on clicking "Finish", you can see a project with below standard directory structure created in the workspace,



Before proceeding with Spring Data Rest let me give you some pointers on what all Spring Boot provides,

- Production-ready services
- Basic health-check and monitoring functions (not used in this application)
- Pre-configured, embedded Tomcat server
- Executable JAR packaging with all dependencies included
- Very little overall configuration overhead (i.e. we just have to configure what we want to customize)

David Boross had written an excellent article out of his own experience with Spring Boot here. This contains information on what to expect from and how to use Spring Boot.

Usually, we use a servlet container such as Tomcat or Jetty to deploy and run our web application separately. While using Spring Boot it includes an embedded tomcat and all you have to do is, a Maven build that converts your whole application into an executable jar. Just run the jar file and access the application using default Tomcat's port (if you have not made an attempt to use a different port).

Let us go ahead and rewrite pom.xml file to include all dependencies,

```
<dependencies>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-rest</artifactId>
  </dependency>
```

```
 <dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
 </dependency>
</dependencies>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <start-class>com.programmingfree.springservice.Application</start-class>
 <java.version>1.7</java.version>
</properties>

<build>
 <plugins>
  <plugin>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
 </plugins>
</build>
```

There are four dependencies we have included,

- thymeleaf (html templating)
- spring data jpa
- spring data rest
- mysql connector for java

Apart from the dependencies, the start class is also mentioned for the maven to identify where to start from. Finally spring boot maven plugin ensures packaging of your application into an executable jar.

As we have the structure and setup ready, let us proceed with building the CRUD application using Spring Data Rest and AngularJS in the front end. Before that, let us complete the MySql table setup required to bring up the Task Manager Application.

MySql Table - Task Manager Application

Use the following sql script to create task_list table against which we are gonna perform CRUD operations.

```
create database taskmanager;
use taskmanager;
create table task_list(task_id int not null auto_increment, task_name varchar(100) not null, task_description text,task_priority varchar(20),task_status varchar(20),task_start_time datetime not null default CURRENT_TIMESTAMP,task_end_time datetime not null default CURRENT_TIMESTAMP,task_archived bool default false,primary key(task_id));
insert into task_list values(1,'Gathering Requirement','Requirement Gathering','MEDIUM','ACTIVE',curtime(),curtime() + INTERVAL 3 HOUR,0);
insert into task_list values(2,'Application Designing','Application Designing','MEDIUM','ACTIVE',curtime(),curtime() + INTERVAL 2
```

```
HOUR,0);
insert into task_list values(3,'Implementation','Implementation','MEDIUM','ACTIVE',curtime(),curtime() + INTERVAL 3 HOUR,0);
insert into task_list values(4,'Unit Testing','Unit Testing','LOW','ACTIVE',curtime(),curtime() + INTERVAL 4 HOUR,0);
insert into task_list values(5,'Maintanence','Maintanence','LOW','ACTIVE',curtime(),curtime() + INTERVAL 5 HOUR,0);
select * from task_list;
```

Spring Data REST

The difference between simple Spring MVC RESTFul Web Services and Spring Data REST is that it combines the RESTFul architecture with Spring Data JPA (Java Persistence)/Gemfire/MongoDB/Neo4j to provide an easy way to implement database operations. If you are not familiar with Spring Data JPA or any other persistence library such as Hibernate, do not worry. You just have to understand the basic concept of it to get started.

Java Persistence API (JPA) - Unlike writing a plain DAO that consists of plain JDBC code everywhere (my previous tutorial is an example of this scenario) full of PreparedStatements and SqlConnections etc, we just map the original fields in the database table to Java classes called Entities, provide SQL queries and let the persistence api handle the connections, query execution etc without writing much boilerplate code.

Spring Data JPA & Spring Data REST takes a step forward and handles the DAO layer around data repositories with out of the box implementation for most commonly used queries. Though you can use @Query Annotation to write your own queries, you would not require that in most of the cases.

Hope that is enough of theory now. Even if you do not understand a single line of it, I am sure you will make it out in the course of implementation. So let us first write our Entity class.

Task.java

```
01 package com.programmingfree.springservice;
02
03 import javax.persistence.Entity;
04 import javax.persistence.GeneratedValue;
05 import javax.persistence.GenerationType;
06 import javax.persistence.Table;
07 import javax.persistence.Column;
08 import javax.persistence.Id;
09
10 @Entity
11 @Table(name="task_list")
12 public class Task {
13
14  @Id
15  @GeneratedValue(strategy = GenerationType.AUTO)
16  @Column(name="task_id")
17  private int id;
18
19  @Column(name="task_name")
20  private String taskName;
21
```

```
22  @Column(name="task_description")
23  private String taskDescription;
24
25  @Column(name="task_priority")
26  private String taskPriority;
27
28  @Column(name="task_status")
29  private String taskStatus;
30
31  @Column(name="task_archived")
32  private int taskArchived = 0;
33
34  public int getTaskId() {
35   return id;
36  }
37
38  public void setTaskId(int taskId) {
39   this.id = taskId;
40  }
41
42  public String getTaskName() {
43   return taskName;
44  }
45
46  public void setTaskName(String taskName) {
47   this.taskName = taskName;
48  }
49
50  public String getTaskDescription() {
51   return taskDescription;
52  }
53
54  public void setTaskDescription(String taskDescription) {
55   this.taskDescription = taskDescription;
56  }
57
58  public String getTaskPriority() {
59   return taskPriority;
60  }
61
62  public void setTaskPriority(String taskPriority) {
63   this.taskPriority = taskPriority;
64  }
65
66  public String getTaskStatus() {
67   return taskStatus;
68  }
69
70  public void setTaskStatus(String taskStatus) {
71   this.taskStatus = taskStatus;
72  }
73
74  public int isTaskArchived() {
75   return taskArchived;
76  }
77
```

```
 public void setTaskArchived(int taskArchived) {
  this.taskArchived = taskArchived;
 }

 @Override
 public String toString() {
  return "Task [id=" + id + ", taskName=" + taskName
    + ", taskDescription=" + taskDescription + ", taskPriority="
    + taskPriority + ",taskStatus=" + taskStatus + "]";
 }

}
```

The above class has @Entity annotation and @Table annotation with table name as an argument to it. You can clearly see each and every field in the table is mapped to java variables with @Column providing the actual name in the table. Rest of the code are getters and setters which will be present in any domain class usually.

As the entity is ready, next step is to create a repository class that serves as the data access layer in Spring Data Rest. Usually in any DAO, you will have methods to create, read, update and delete from database. But Spring makes it easy for you, that all these operations are covered out of the box and you just need an interface that extends CrudRepository class.

TaskRepository.java

```
package com.programmingfree.springservice;

import java.util.List;

import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface TaskRepository extends CrudRepository<Task, Integer> {

 List<Task> findByTaskArchived(@Param("archivedfalse") int taskArchivedFalse);
 List<Task> findByTaskStatus(@Param("status") String taskStatus);

}
```

TaskRepository extends CrudRepository. The type of entity and ID it works with, Task and Long are specified as generic parameters to CrudRepository. By extending CrudRepository, TaskRepository inherits several methods for working with Task persistence, including methods for saving, deleting, and finding Task entities.

Spring Data JPA also allows you to define other query methods by simply declaring their method signature. In the case of TaskRepository, this is shown with a findByTaskArchived() method which takes archivedTask as parameter. In a typical Java application, you'd expect to write a class that implements TaskRepository. But that's what makes Spring Data JPA so powerful: You don't have to write an implementation of the repository interface. Spring Data JPA creates an implementation on

the fly when you run the application.

In the above code you can see @RepositoryRestResource annotation being used. This annotation is responsible for exposing this repository interface as a RESTFul resource. This is pretty much similar to @RestController which we used in plain Spring MVC REST to expose a controller as RESTFul resource.

Property File

By default Spring Boot will look for a property file in the package root directory called 'application.properties', this is a good place to customize your application. For example, as we have placed mysql connector jar in the pom.xml file, Spring Boot will look for mysql specific properties in this file. By Maven conventions, place this file into the src/main/resources directory so your build artefacts will be
generated correctly.

application.properties

```
# Replace with your connection string
spring.datasource.url=jdbc:mysql:

# Replace with your credentials
spring.datasource.username=root
spring.datasource.password=root

spring.datasource.driverClassName=com.mysql.jdbc.Driver
```

Make the application Executable

Although it is possible to package this service as a traditional WAR file for deployment to an external application server, the simpler approach demonstrated below creates a standalone application. You package everything in a single, executable JAR file, driven by a good old Java main() method. Along the way, you use Spring's support for embedding the Tomcat servlet container as the HTTP runtime, instead of deploying to an external instance. Remember we mentioned Application class as the start class in pom.xml file.

Application.java

```
package com.programmingfree.springservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
```

```
09 import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
10 import org.springframework.data.rest.webmvc.config.RepositoryRestMvcConfiguration;
11
12 @Configuration
13 @ComponentScan
14 @EnableJpaRepositories
15 @Import(RepositoryRestMvcConfiguration.class)
16 @EnableAutoConfiguration
17 public class Application {
18
19  public static void main(String[] args) {
20   SpringApplication.run(Application.class, args);
21
22  }
23
24 }
```

The @EnableJpaRepositories annotation activates Spring Data JPA. Spring Data JPA will create a concrete implementation of the TaskRepository and configure it to talk to a back end in-memory database using JPA.

Spring Data REST is a Spring MVC application. The @Import(RepositoryRestMvcConfiguration.class) annotation imports a collection of Spring MVC controllers, JSON converters, and other beans needed to provide a RESTful front end. These components link up to the Spring Data JPA backend.

The @EnableAutoConfiguration annotation switches on reasonable default behaviors based on the content of your classpath. For example, because the application depends on the embeddable version of Tomcat (tomcat-embed-core.jar), a Tomcat server is set up and configured with reasonable defaults on your behalf. And because the application also depends on Spring MVC (spring-webmvc.jar), a Spring MVC DispatcherServlet is configured and registered for you — no web.xml necessary.

Templating

We have loaded thymeleaf as a dependency which is nothing but an html templating engine. Initially when the application class is run, all classes in the package in which the application class resides will be scanned. Hence let us write a controller that redirects the control to front end template.

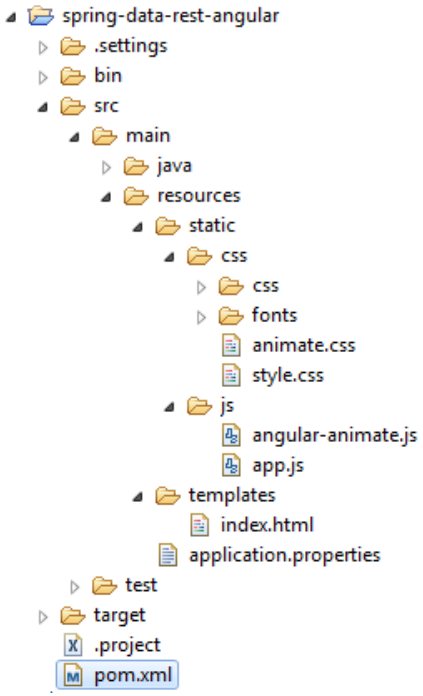
HomeController.java

```
01 package com.programmingfree.springservice;
02
03 import org.springframework.stereotype.Controller;
04 import org.springframework.web.bind.annotation.RequestMapping;
05
06 @Controller
07 public class HomeController {
08
09  @RequestMapping("/home")
```

```
 public String home() {
  return "index";
 }

}
```

Next let us place the static resources and template files according to the standard directory structure expected by Spring Boot to process them.



I had copied all css and javascript files from my previous project into static folder. I had used index.jsp in my previous tutorial and here I am using index.html template file.

index.html

```
<html ng-app="taskManagerApp">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>AngularJS Task Manager</title>
<link href='./css/style.css' rel="stylesheet" type="text/css" />
<link href='./css/css/font-awesome.css' rel="stylesheet" type="text/css" />
<link href='http://fonts.googleapis.com/css?family=Open+Sans' rel='stylesheet' type='text/css' />
<script data-require="angular.js@*" data-semver="1.3.0-beta.14" src="http://code.angularjs.org/1.3.0-beta.14/angular.js"></script>
  <script data-require="angular-animate@*" data-semver="1.3.0-beta.14" src="http://code.angularjs.org/1.3.0-beta.14/angular-
animate.js"></script>
```

```
<script type="text/javascript" src="./js/app.js"></script>
</head>
<body>

<div ng-controller="taskManagerController">
<h2 class="page-title">Task Manager using Spring Boot, Spring Data REST &amp;amp; AngularJS</h2>
<h4 class="page-title">Demo &amp;amp; Tutorial by <a href="">Priyadarshini</a></h4>
<a href="http://www.programming-free.com/2014/07/spring-data-rest-with-angularjs-crud.html" class="button-red" style="text-
align:center;width:70px;margin-left:45%;margin-right:40%">Tutorial</a>
 <div id="task-panel" class="fadein fadeout showpanel panel"  ng-show="toggle">
  <div class="panel-heading">
   <i class="panel-title-icon fa fa-tasks"></i>
   <span class="panel-title">Recent Tasks</span>
   <div class="panel-heading-controls">
    <button ng-click="toggle = !toggle" class="btn-panel">Add New Task</button>
    <button class="btn-panel" confirmed-click="archiveTasks()" ng-confirm-click="Would you like to archive completed tasks?">Clear
completed tasks</button>
   </div>
  </div>
  <div class="panel-body">
   <div class="task" ng-repeat="task in tasks">
    <span ng-if="task.taskPriority=='HIGH'" class="priority priority-red">
     {{task.taskPriority}}
    </span>
    <span ng-if="task.taskPriority=='MEDIUM'" class="priority priority-yellow">
     {{task.taskPriority}}
    </span>
    <span ng-if="task.taskPriority=='LOW'" class="priority priority-green">
     {{task.taskPriority}}
    </span>
    <div class="action-checkbox">
                    <input id="{{task._links.self.href}}" type="checkbox" value="{{task._links.self.href}}" ng-
checked="selection.indexOf(task._links.self.href) > -1" ng-click="toggleSelection(task._links.self.href)" />
                    <label for="{{task._links.self.href}}" ></label>
                </div>
    <div ng-if="task.taskStatus=='COMPLETED'">
     <a href="#" class="checkedClass">
      {{task.taskName}}
     <span class="action-status">{{task.taskStatus}}</span>
     </a>
    </div>
    <div ng-if="task.taskStatus=='ACTIVE'">
     <a href="#" class="uncheckedClass">
      {{task.taskName}}
      <span class="action-status">{{task.taskStatus}}</span>
     </a>
    </div>
   </div>
  </div>
 </div>
 <div id="add-task-panel" class="fadein fadeout addpanel panel" ng-hide="toggle">
  <div class="panel-heading">
   <i class="panel-title-icon fa fa-plus"></i>
   <span class="panel-title">Add Task</span>
   <div class="panel-heading-controls">
    <button ng-click="toggle = !toggle" class="btn-panel">Show All Tasks</button>
```

```
    </div>
   </div>
   <div class="panel-body">
    <div class="task" >
     <table class="add-task">
      <tr>
       <td>Task Name:</td>
       <td><input type="text" ng-model="taskName"/></td>
      </tr>
      <tr>
       <td>Task Description:</td>
       <td><input type="text" ng-model="taskDesc"/></td>
      </tr>
      <tr>
       <td>Task Status:</td>
       <td>
        <select ng-model="taskStatus" ng-options="status as status for status in statuses">
         <option value="">-- Select --</option>
             </select>
       </td>
      </tr>
      <tr>
       <td>Task Priority:</td>
       <td>
        <select ng-model="taskPriority" ng-options="priority as priority for priority in priorities">
         <option value="">-- Select --</option>
        </select>
       </td>
      </tr>
      <tr>
       <td>
<button ng-click="addTask()" class="btn-panel-big">Add New Task</button></td>
      </tr>
     </table>
    </div>
   </div>
  </div>
 </div>
</body>
</html>
```

app.js

```
var taskManagerModule = angular.module('taskManagerApp', ['ngAnimate']);

taskManagerModule.controller('taskManagerController', function ($scope,$http) {

 var urlBase="";
 $scope.toggle=true;
 $scope.selection = [];
 $scope.statuses=['ACTIVE','COMPLETED'];
 $scope.priorities=['HIGH','LOW','MEDIUM'];
```

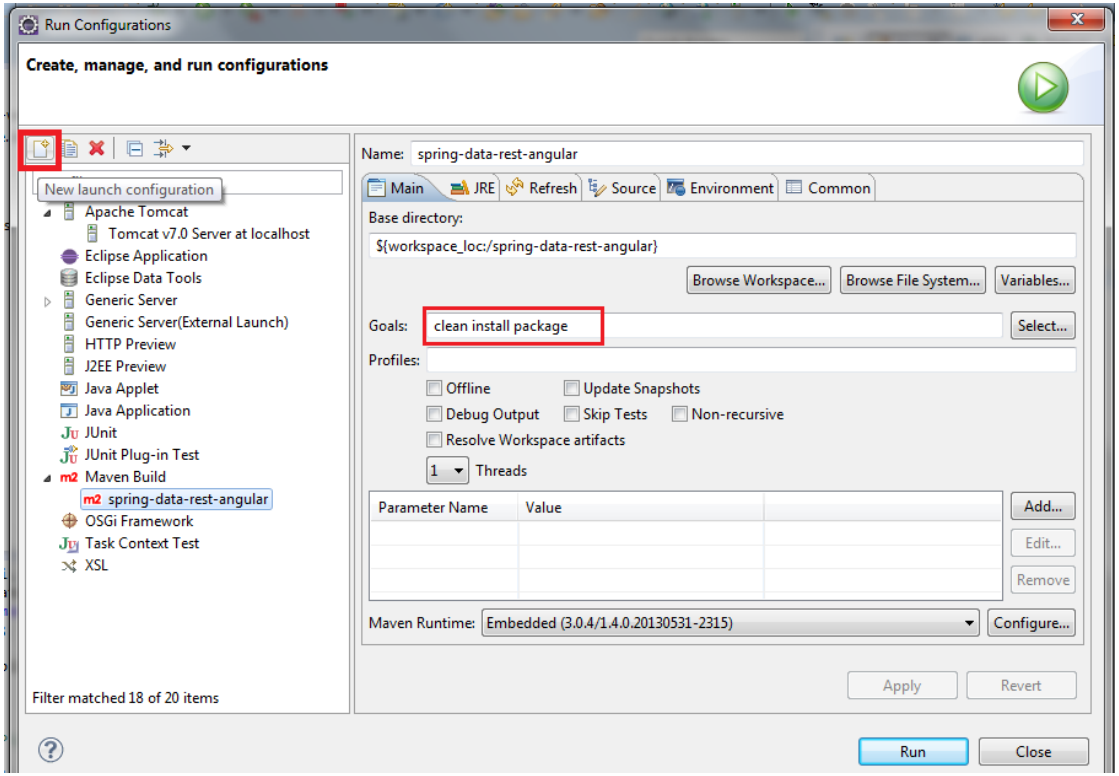
```
010 $http.defaults.headers.post["Content-Type"] = "application/json";
011
012    function findAllTasks() {
013        //get all tasks and display initially
014        $http.get(urlBase + '/tasks/search/findByTaskArchived?archivedfalse=0').
015            success(function (data) {
016                if (data._embedded != undefined) {
017                    $scope.tasks = data._embedded.tasks;
018                } else {
019                    $scope.tasks = [];
020                }
021                for (var i = 0; i < $scope.tasks.length; i++) {
022                    if ($scope.tasks[i].taskStatus == 'COMPLETED') {
023                        $scope.selection.push($scope.tasks[i].taskId);
024                    }
025                }
026                $scope.taskName="";
027                $scope.taskDesc="";
028                $scope.taskPriority="";
029                $scope.taskStatus="";
030                $scope.toggle='!toggle';
031            });
032    }
033
034    findAllTasks();
035
036 //add a new task
037 $scope.addTask = function addTask() {
038  if($scope.taskName=="" || $scope.taskDesc=="" || $scope.taskPriority == "" || $scope.taskStatus == ""){
039   alert("Insufficient Data! Please provide values for task name, description, priortiy and status");
040  }
041  else{
042   $http.post(urlBase + '/tasks', {
043             taskName: $scope.taskName,
044             taskDescription: $scope.taskDesc,
045             taskPriority: $scope.taskPriority,
046             taskStatus: $scope.taskStatus
047         }).
048    success(function(data, status, headers) {
049    alert("Task added");
050             var newTaskUri = headers()["location"];
051             console.log("Might be good to GET " + newTaskUri + " and append the task.");
052             // Refetching EVERYTHING every time can get expensive over time
053             // Better solution would be to $http.get(headers()["location"]) and add it to the list
054             findAllTasks();
055      });
056  }
057 };
058
059 // toggle selection for a given task by task id
060   $scope.toggleSelection = function toggleSelection(taskUri) {
061     var idx = $scope.selection.indexOf(taskUri);
062
063     // is currently selected
064        // HTTP PATCH to ACTIVE state
065     if (idx > -1) {
```

```
066         $http.patch(taskUri, { taskStatus: 'ACTIVE' }).
067       success(function(data) {
068           alert("Task unmarked");
069                 findAllTasks();
070         });
071          $scope.selection.splice(idx, 1);
072        }
073
074        // is newly selected
075           // HTTP PATCH to COMPLETED state
076        else {
077          $http.patch(taskUri, { taskStatus: 'COMPLETED' }).
078       success(function(data) {
079        alert("Task marked completed");
080                 findAllTasks();
081         });
082          $scope.selection.push(taskUri);
083        }
084      };
085
086
087    // Archive Completed Tasks
088      $scope.archiveTasks = function archiveTasks() {
089             $scope.selection.forEach(function(taskUri) {
090                 if (taskUri != undefined) {
091                     $http.patch(taskUri, { taskArchived: 1});
092                 }
093             });
094             alert("Successfully Archived");
095             console.log("It's risky to run this without confirming all the patches are done. when.js is great for that");
096             findAllTasks();
097      };
098
099   });
100
101   //Angularjs Directive for confirm dialog box
102   taskManagerModule.directive('ngConfirmClick', [
103    function(){
104            return {
105                link: function (scope, element, attr) {
106                    var msg = attr.ngConfirmClick || "Are you sure?";
107                    var clickAction = attr.confirmedClick;
108                    element.bind('click',function (event) {
109                        if ( window.confirm(msg) ) {
110                            scope.$eval(clickAction);
111                        }
112                    });
113                }
114            };
115    }]);
```

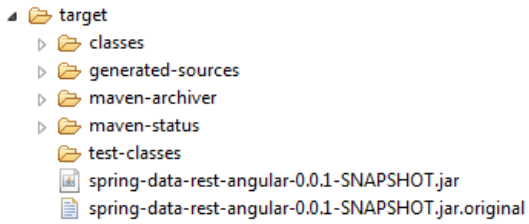
Note that the URI to query the exposed RESTFul repositories had changed in the app.js file. If you are doubtful about how angularjs works in index.html file and communicates with the backend, please refer to my previous article which will give you an idea of how AngularJS can be made to work with Spring MVC. If you are new to AngularJS, then you might want to have a look at the basics once to get started.

We are done with the coding. To build the project,

Right click on pom.xml file -> Run as -> Run Configurations -> Select Maven Build -> New Launch Configuration -> Select your project locations -> Mentions Goals as *clean install package* -> Run



After successfully running the project, refresh workspace once and you can see the executable jar file under target folder here,



Open command prompt to execute jar file and hit http://localhost:8080/home to see the task manager application working.

```
java - jar target/spring-data-rest-angular-0.0.1-SNAPSHOT.jar
```

DEMO DOWNLOAD

Keep yourself subscribed for getting programmingfree articles delivered directly to your inbox once in a month. Thanks for reading!

Posted by ***Priya Darshini***

POST A COMMENT

403 Forbidden

Request forbidden by administrative rules.

403 Forbidden

Request forbidden by administrative rules.

403 Forbidden

Request forbidden by administrative rules.

403 Forbidden

Request forbidden by administrative rules.

403 Forbidden

Request forbidden by administrative rules.