



Read DZone's 2019 Machine Learning Trend Report to see the future impact machine learning will have.

[Read Now](#) ▶

# Step-By-Step Spring Boot RESTful Web Service Complete Example

by Abhijit Pritam Dutta · Sep. 05, 18 · Integration Zone · Tutorial

After the huge response and viewership for my earlier article, I have decided to write a new article with all the REST calls example respectively **GET, POST, PUT, and DELETE**.

Let's start this project step by step.

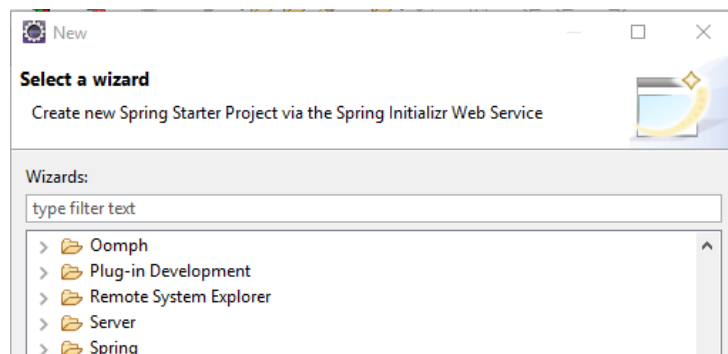
Prerequisites for this project:

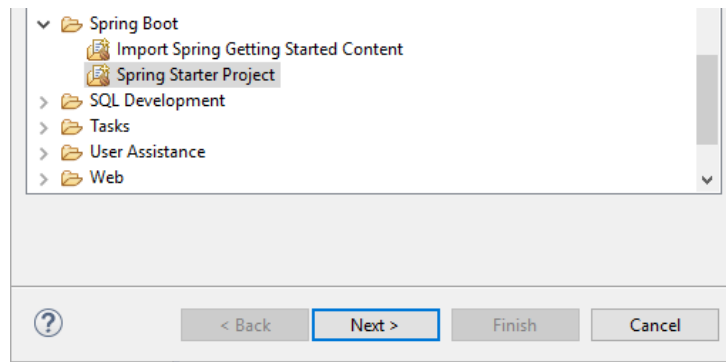
- If you have Eclipse, download the STS plug-in from here.
- If you don't have Eclipse, download STS from here.
- Download the latest JDK from here.
- Also for testing please download and install SOAPUI tool from here.

The first example I am going to explain is about **HTTP GET** request, the second example will be about **HTTP POST** request, the third example about **HTTP PUT** request, and the fourth example is for **HTTP DELETE** request. In all of these examples, I am going to use JSON Representation.

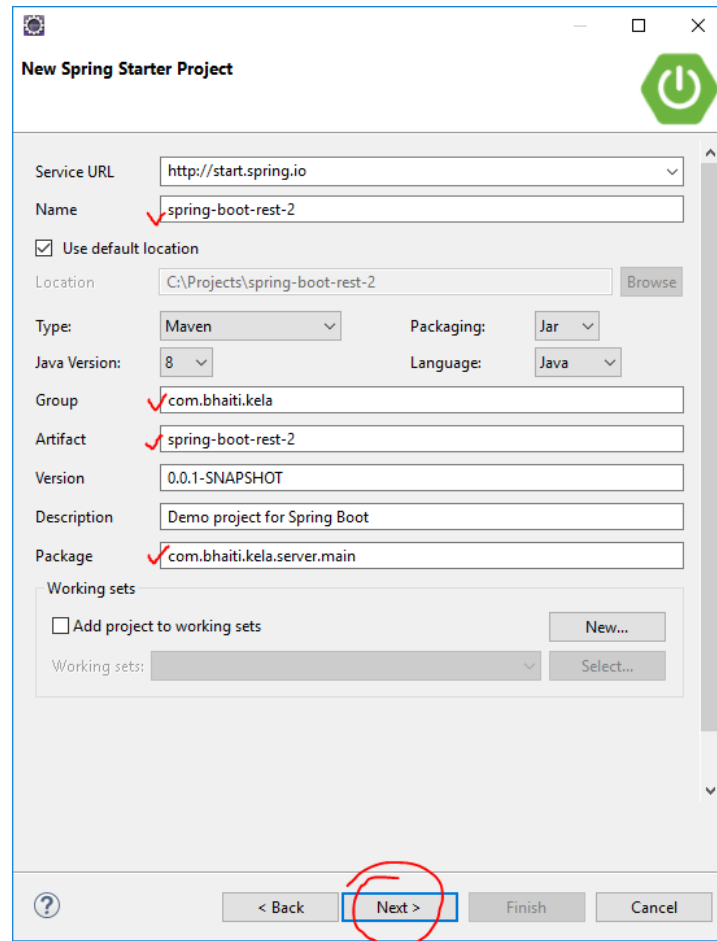
You can download this project from here.

1. First, create a folder in your C drive: **C:\Projects**
2. Open eclipse and select work space as: **C:\Projects**
3. From the File menu, select "New" and then "other;" and from wizard, expand "Spring Boot" and select "Spring Starter Project" (**File->New->other->wizard->Spring Starter Project**).

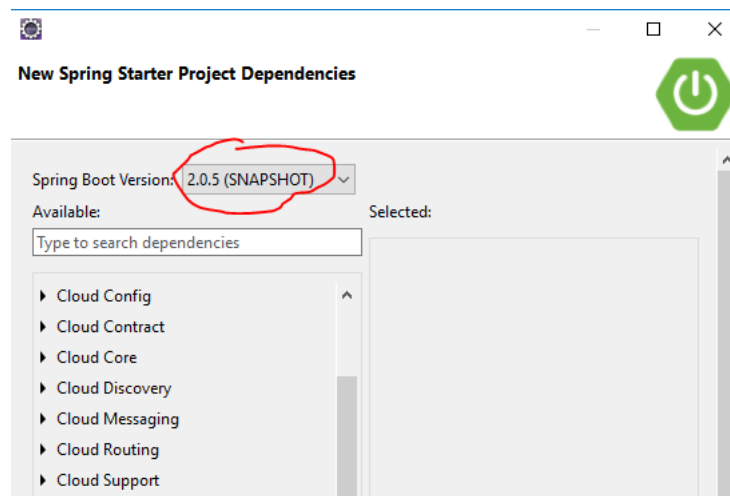


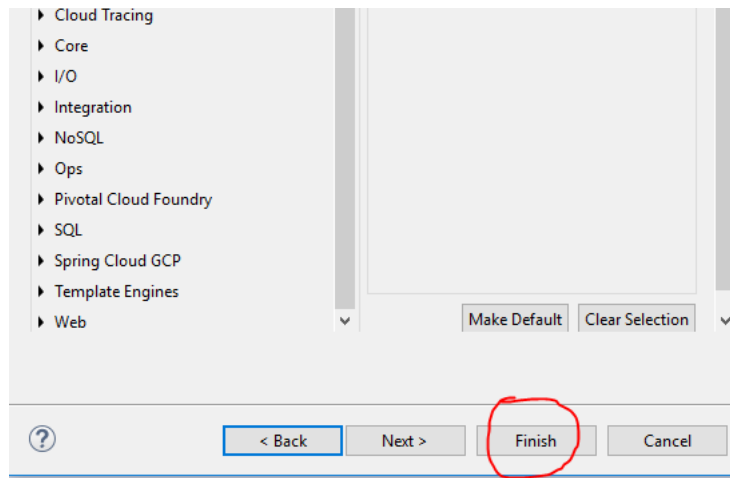


Now click the **Next** button and provide the below information and click the **Next** button again.

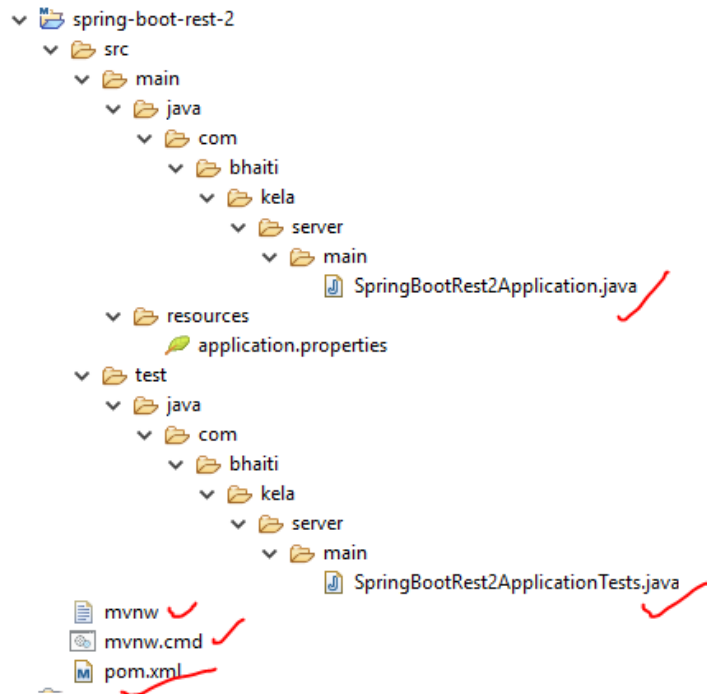


Now, provide/select the below information and click the **Finish** button.





Now you can see the below project structure in your project's explorer window.



Now, look that the **SpringBootRest2Application.java** file, which is created by the **STS plug-ins**.

```

1  package com.bhaiti.kela.server.main;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class SpringBootRest2Application {
8
9      public static void main(String[] args) {
10         SpringApplication.run(SpringBootRest2Application.class, args);
11     }
12 }

```

This is a Spring Boot main class. A Spring Boot REST application loads through this class. We can also see that this class is created with the annotation **@SpringBootApplication**. As per the Spring documentation, the annotation **@SpringBootApplication** is equivalent to using **@Configuration**, **@EnableAutoConfiguration**, and **@ComponentScan**, and these annotations are frequently used together. Most of the time, in Spring Boot development, the main class is always annotated with all three of these important annotations.

So we are going to modify the **@SpringBootApplication** (given below in the Java class) with a component path. Without that, the application cannot find out the controller classes. We will learn more about controller classes in a few minutes.

### @SpringBootApplication(scanBasePackages = {"com.bhaiti"})

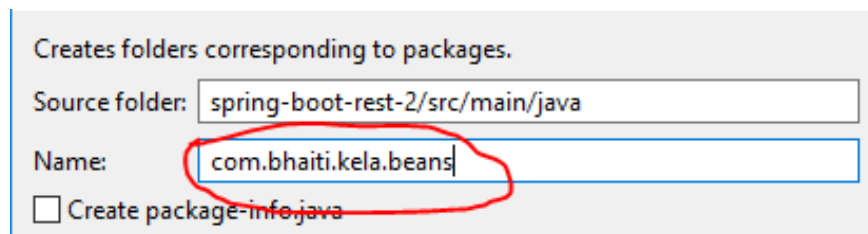
```

1  @SpringBootApplication(scanBasePackages = {"com.bhaiti"})
2  public class SpringBootRest2Application {
3
4      public static void main(String[] args) {
5          SpringApplication.run(SpringBootRest2Application.class, args);
6      }
7  }

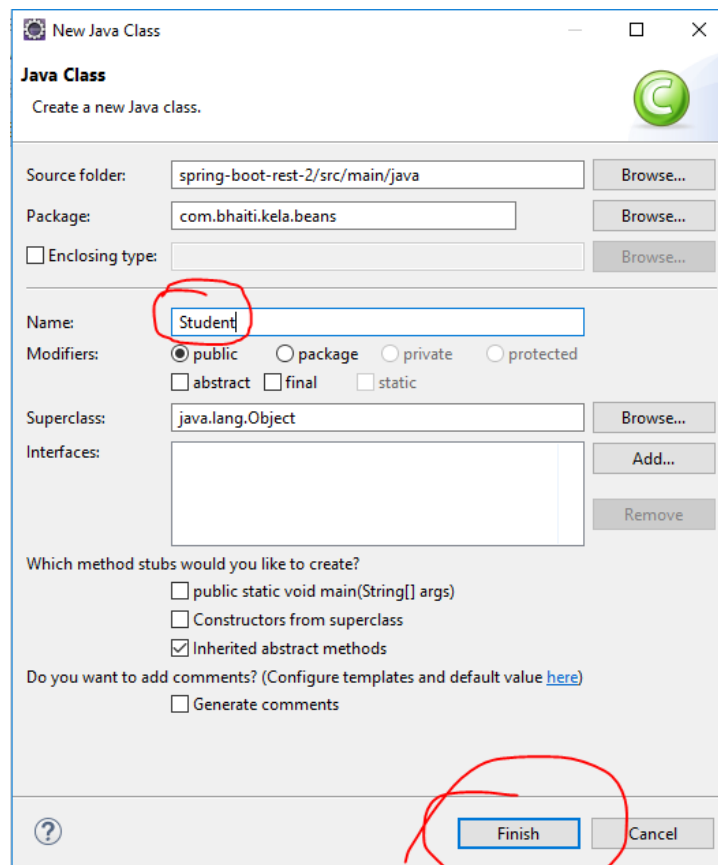
```

Now we are going to create our beans classes, which we will use for our **GET, POST, PUT, and DELETE** REST calls. Create package, **com.bhaiti.beans**, and add classes into that package like below:

4. Right-click on the project and select New and then package (**New=>Package**). Enter the above package name and click Finish.



5. Now, right-click on the package **com.bhaiti.beans** and select **New->class** and provide the class name, **student**, like below:



- In the empty class, just add below members:

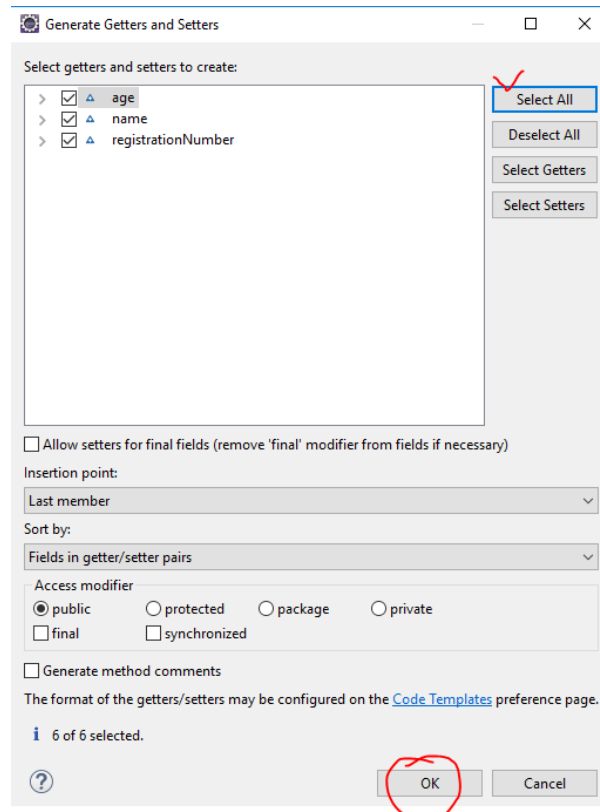
```

1  package com.bhaiti.kela.beans;

```

```
2
3 public class Student {
4
5     String name;
6     int age;
7     String registrationNumber;
8 }
```

- Now create **getter** and **setter** methods for these members. For that, from the Project Explorer window, right-click on the class "Student" and select "Source" and select "Generate Getter and Setter."



```
1 package com.bhaiti.kela.beans;
2
3 public class Student {
4
5     String name;
6     int age;
7     String registrationNumber;
8
9     public String getName() {
10    return name;
11    }
12    public void setName(String name) {
13    this.name = name;
14    }
15    public int getAge() {
16    return age;
17    }
18    public void setAge(int age) {
19    this.age = age;
20    }
```

```

--
21     public String getRegistrationNumber() {
22         return registrationNumber;
23     }
24     public void setRegistrationNumber(String registrationNumber) {
25         this.registrationNumber = registrationNumber;
26     }
27 }

```

6. Now follow steps 5 and create class call **StudentRegistration** and modify it like below:

```

1  package com.bhaiti.kela.beans;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class StudentRegistration {
7
8      private List<Student> studentRecords;
9
10     private static StudentRegistration stdregd = null;
11
12     private StudentRegistration(){
13         studentRecords = new ArrayList<Student>();
14     }
15
16     public static StudentRegistration getInstance() {
17
18         if(stdregd == null) {
19             stdregd = new StudentRegistration();
20             return stdregd;
21         }
22         else {
23             return stdregd;
24         }
25     }
26
27     public void add(Student std) {
28         studentRecords.add(std);
29     }
30
31     public String upDateStudent(Student std) {
32
33     for(int i=0; i<studentRecords.size(); i++)
34         {
35             Student stdn = studentRecords.get(i);
36             if(stdn.getRegistrationNumber().equals(std.getRegistrationNumber())) {
37                 studentRecords.set(i, std);//update the new record
38                 return "Update successful";
39             }
40         }
41
42     return "Update un-successful";
43 }
44
45     public String upDateStudent(Student registrationNumber) {

```

```
45 public String deleteStudent(String registrationNumber) {
46
47     for(int i=0; i<studentRecords.size(); i++)
48         {
49             Student stdn = studentRecords.get(i);
50             if(stdn.getRegistrationNumber().equals(registrationNumber)){
51                 studentRecords.remove(i);//update the new record
52                 return "Delete successful";
53             }
54         }
55
56     return "Delete un-successful";
57 }
58
59 public List<Student> getStudentRecords() {
60     return studentRecords;
61 }
62
63 }
```

7. Now add a class calls **StudentRegistrationReply** and modify like below. This class will be used to reply a response back to the client application

```
1 package com.bhaiti.kela.beans;
2
3 public class StudentRegistrationReply {
4
5     String name;
6     int age;
7     String registrationNumber;
8     String registrationStatus;
9
10    public String getName() {
11        return name;
12    }
13    public void setName(String name) {
14        this.name = name;
15    }
16    public int getAge() {
17        return age;
18    }
19    public void setAge(int age) {
20        this.age = age;
21    }
22    public String getRegistrationNumber() {
23        return registrationNumber;
24    }
25    public void setRegistrationNumber(String registrationNumber) {
26        this.registrationNumber = registrationNumber;
27    }
28    public String getRegistrationStatus() {
29        return registrationStatus;
30    }
31    public void setRegistrationStatus(String registrationStatus) {
32        this.registrationStatus = registrationStatus;
33    }
34 }
```

```
33     }
34
35 }
```

8. Now we will introduce two controllers, one to serve the **GET** request and the second one to serve the **POST** request. With the **GET** request, we will retrieve all Student Registration information, and with the **POST** request, we will add student information into our application. In spring's approach to build a **RESTful** web services, **HTTP** requests are handled by a controller. Controller classes/components are easily identified by the **@RestController** annotation, and the below **StudentRetrieveController** will handle **GET** requests for **/student/allstudent** by returning a list of **Student** class objects in **JSON** format.

- Now just follow **step 4** and **step 5** and create the package **com.bhaiti.kela.controllers** and add the class **StudentRetrieveController** to it and import the class **Student** and modify the class like below:-

```
1  package com.bhaiti.kela.controllers;
2
3  import java.util.List;
4
5  import org.springframework.stereotype.Controller;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8  import org.springframework.web.bind.annotation.ResponseBody;
9
10 import com.bhaiti.kela.beans.Student;
11 import com.bhaiti.kela.beans.StudentRegistration;
12
13 @RestController
14 public class StudentRetrieveController {
15
16     @RequestMapping(method = RequestMethod.GET, value="/student/allstudent")
17
18     @ResponseBody
19     public List<Student> getAllStudents() {
20         return StudentRegistration.getInstance().getStudentRecords();
21     }
22
23 }
```

The **@RequestMapping** annotation maps all HTTP operations by default and, in this example, it ensures that HTTP requests to **/student/allstudent** are mapped to the **getAllStudents()** method.

Now we are done with everything for a **GET** RESTful call. Let's implement a RESTful **POST** call.

- Now it's time to introduce the controller class to handle the **POST** request. Follow **step 5** and add below controller class **StudentRegistrationController** in package **com.bhaiti.kela.controllers**

```
1  package com.bhaiti.kela.controllers;
2
3  import java.util.List;
4
5  import org.springframework.stereotype.Controller;
6  import org.springframework.web.bind.annotation.RequestBody;
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import org.springframework.web.bind.annotation.RequestMethod;
9  import org.springframework.web.bind.annotation.ResponseBody;
10
11 import com.bhaiti.kela.beans.*;
```



```
12
13 @Controller
14 public class StudentRegistrationController {
15
16     @RequestMapping(method = RequestMethod.POST, value="/register/student")
17
18     @ResponseBody
19     public StudentRegistrationReply registerStudent(@RequestBody Student student) {
20         System.out.println("In registerStudent");
21         StudentRegistrationReply stdregreply = new StudentRegistrationReply();
22         StudentRegistration.getInstance().add(student);
23         //We are setting the below value just to reply a message back to the caller
24         stdregreply.setName(student.getName());
25         stdregreply.setAge(student.getAge());
26         stdregreply.setRegistrationNumber(student.getRegistrationNumber());
27         stdregreply.setRegistrationStatus("Successful");
28
29         return stdregreply;
30     }
31
32 }
```

Till now, we are done with everything for a **GET and POST** RESTful call. Let's test this application first. After the test, we will learn about **PUT and DELETE** calls as well. First, we need to compile the application.

- **Compilation:** To compile this project with Maven, we will add the below information into the POM file: Double click and open the POM file from project explorer window and add below information under “<dependencies>” section.

```
1     <dependency>
2         <groupId>org.springframework.boot</groupId>
3         <artifactId>spring-boot-starter</artifactId>
4     </dependency>
5
6     <dependency>
7         <groupId>org.springframework.boot</groupId>
8         <artifactId>spring-boot-starter-test</artifactId>
9         <scope>test</scope>
10    </dependency>
11
12    <dependency>
13        <groupId>org.springframework.boot</groupId>
14        <artifactId>spring-boot-starter</artifactId>
15    </dependency>
16
17    <dependency>
18        <groupId>org.springframework</groupId>
19        <artifactId>spring-web</artifactId>
20    </dependency>
21
22    <dependency>
23        <groupId>org.springframework.boot</groupId>
24        <artifactId>spring-boot-starter-web</artifactId>
25    </dependency>
26
27    <dependency>
```

```

28     <groupId>org.springframework.boot</groupId>
29     <artifactId>spring-boot-starter-test</artifactId>
30     <scope>test</scope>
31 </dependency>

```

And we'll also add the below information for the property file for this project under the build section of our **POM.xml** file (to change the port number at the moment):

```

1     <resource>
2         <directory>src/main/resources</directory>
3         <filtering>true</filtering>
4     </resource>

```

your POM.xml file finally looks like below:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" >
4     <modelVersion>4.0.0</modelVersion>
5
6     <groupId>com.bhaiti.kela</groupId>
7     <artifactId>spring-boot-rest-2</artifactId>
8     <version>0.0.1-SNAPSHOT</version>
9     <packaging>jar</packaging>
10
11     <name>spring-boot-rest-2</name>
12     <description>Demo project for Spring Boot</description>
13
14     <parent>
15         <groupId>org.springframework.boot</groupId>
16         <artifactId>spring-boot-starter-parent</artifactId>
17         <version>2.0.5.BUILD-SNAPSHOT</version>
18         <relativePath/> <!-- lookup parent from repository -->
19     </parent>
20
21     <properties>
22         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23         <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
24         <java.version>1.8</java.version>
25     </properties>
26
27     <dependencies>
28     <dependency>
29         <groupId>org.springframework.boot</groupId>
30         <artifactId>spring-boot-starter</artifactId>
31     </dependency>
32
33     <dependency>
34         <groupId>org.springframework.boot</groupId>
35         <artifactId>spring-boot-starter-test</artifactId>
36         <scope>test</scope>
37     </dependency>
38
39     <dependency>
40         <groupId>org.springframework.boot</groupId>
41         <artifactId>spring-boot-starter</artifactId>

```

```
42 </dependency>
43
44 <dependency>
45     <groupId>org.springframework</groupId>
46     <artifactId>spring-web</artifactId>
47 </dependency>
48
49 <dependency>
50     <groupId>org.springframework.boot</groupId>
51     <artifactId>spring-boot-starter-web</artifactId>
52 </dependency>
53
54 <dependency>
55     <groupId>org.springframework.boot</groupId>
56     <artifactId>spring-boot-starter-test</artifactId>
57     <scope>test</scope>
58 </dependency>
59
60 </dependencies>
61
62 <build>
63     <plugins>
64     <plugin>
65         <groupId>org.springframework.boot</groupId>
66         <artifactId>spring-boot-maven-plugin</artifactId>
67     </plugin>
68     </plugins>
69     <resources>
70         <resource>
71             <directory>src/main/resources</directory>
72             <filtering>true</filtering>
73         </resource>
74     </resources>
75 </build>
76
77 <repositories>
78 <repository>
79     <id>spring-snapshots</id>
80     <name>Spring Snapshots</name>
81     <url>https://repo.spring.io/snapshot</url>
82     <snapshots>
83     <enabled>true</enabled>
84     </snapshots>
85 </repository>
86 <repository>
87     <id>spring-milestones</id>
88     <name>Spring Milestones</name>
89     <url>https://repo.spring.io/milestone</url>
90     <snapshots>
91     <enabled>>false</enabled>
92     </snapshots>
93 </repository>
94 </repositories>
```

```
95
96 <pluginRepositories>
97 <pluginRepository>
98 <id>spring-snapshots</id>
99 <name>Spring Snapshots</name>
10
10 <url>https://repo.spring.io/snapshot</url>
10
10 <snapshots>
1 <enabled>true</enabled>
2
10 </snapshots>
3
10 </pluginRepository>
4
10 <pluginRepository>
5
10 <id>spring-milestones</id>
10
10 <name>Spring Milestones</name>
7
10 <url>https://repo.spring.io/milestone</url>
8
10 <snapshots>
9
11 <enabled>false</enabled>
0
11 </snapshots>
1
11 </pluginRepository>
2
11 </pluginRepositories>
3
11
4
11
5
11
6 </project>
```

9. Now open file **application.properties** under **C:\Projects\spring-boot-rest-2\src\main\resources** and add the below lines in it:

**server.port=8083**

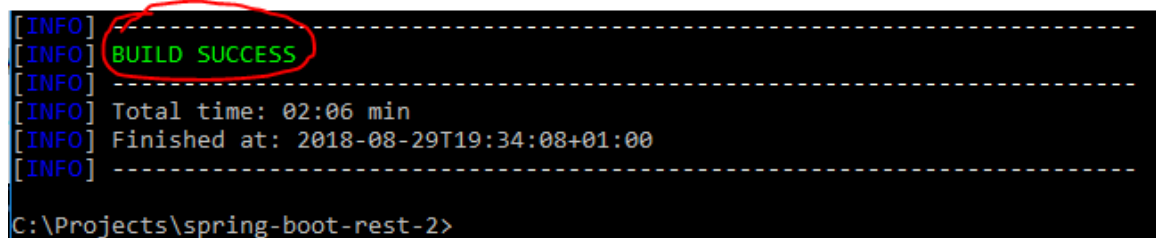
**spring.profiles.active=@spring.profiles.active@**

10. Now open the command prompt window and go to the project home directory in the command prompt. In my case, it looks like this:

**cd C:\Projects\spring-boot-rest-2**

**mvnw clean package**

If everything goes fine, you can see the below result:

A terminal window with a black background and white text. The text shows the output of a Maven build. The first line is "[INFO] -----". The second line is "[INFO] BUILD SUCCESS", where "BUILD SUCCESS" is highlighted in green and circled in red. The third line is "[INFO] -----". The fourth line is "[INFO] Total time: 02:06 min". The fifth line is "[INFO] Finished at: 2018-08-29T19:34:08+01:00". The sixth line is "[INFO] -----". The prompt "C:\Projects\spring-boot-rest-2>" is visible at the bottom.

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:06 min
[INFO] Finished at: 2018-08-29T19:34:08+01:00
[INFO] -----
C:\Projects\spring-boot-rest-2>
```

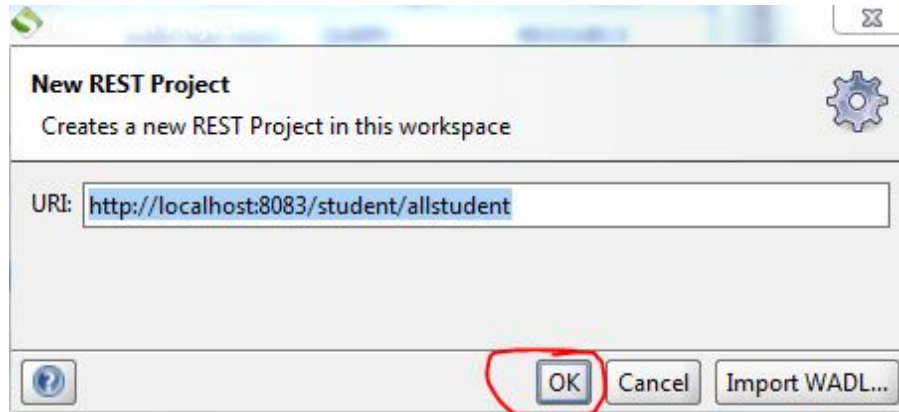
Now run the server:

```
java -jar target\spring-boot-rest-2-0.0.1-SNAPSHOT.jar
```

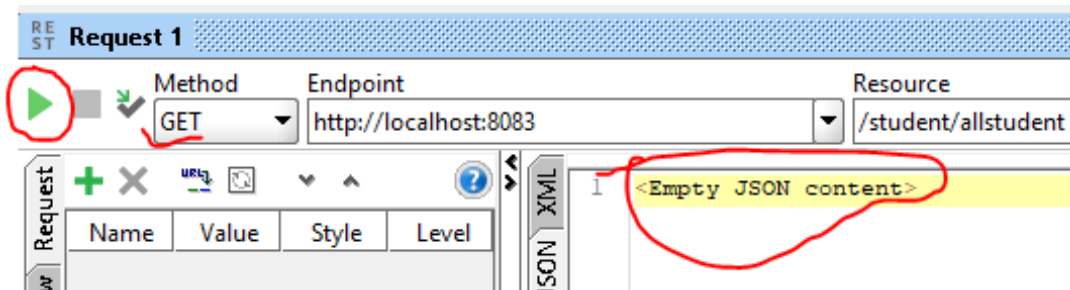
```
ort(s): 8083 (http) with context path ""
2018-08-29 19:36:58.351 INFO 22828 --- [main] c.b.k.s.main.SpringBootRest2Application : Started SpringBootR
est2Application in 34.087 seconds (JVM running for 38.504)
```

11. Once the server starts successfully, we will test get **GET** request first. Open your **SOAPUI** tool. In **SOAPUI** tool from file menu select **New REST Project** and put the below URL in the address bar and press OK. (**File=>New REST Project**)

**http://localhost:8083/student/allstudent**



12. Now, from the SOAPUI project explorer menu, expand your project and double click on Request1 and click the green arrow button:

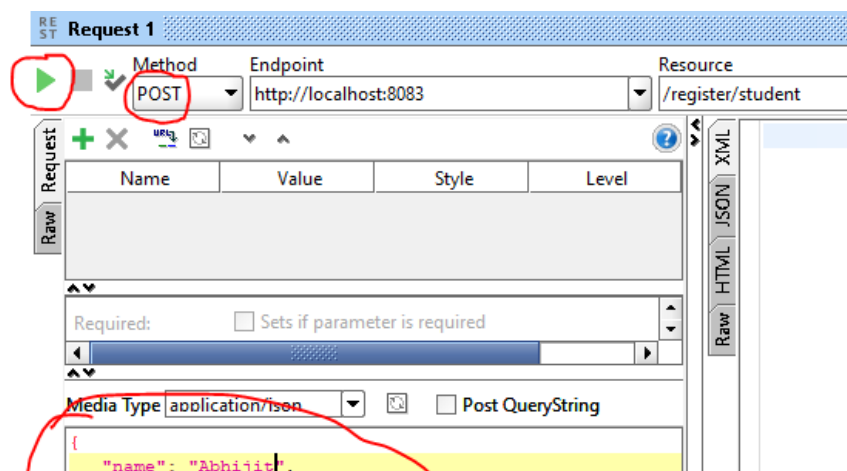


Now you can see the above information. The reason is our student list is empty at the moment, and to store student information, we need to insert some values in it. For that, we will use the POST service that we have already implemented.

13. Follow step 10 and insert the below URL in the address box and click OK.

**http://localhost:8083/register/student**

14. Now you have to select **POST** this time from **Method combo box** for a post request. Once you select **POST Media Type**, sub pane will emerge from where you have to select media type to **application/json** like below and put the below **json body** in it and click the **green arrow** button, which will add the information into our application.



```

    "age": 25,
    "registrationNumber": "12345"
  }
}

```

```

{
  "name": "Abhijit",
  "age": 25,
  "registrationNumber": "12345",
  "registrationStatus": "Successful"
}

```

15. Now go to the GET request project (step 12) and click the green arrow button. You can see one record like below:

The screenshot shows the REST Client interface. The Method is GET and the Endpoint is http://localhost:8083. The Resource is /student/allstudent. The response is a JSON array with one object:

```

[[{"name": "Abhijit", "age": 25, "registrationNumber": "12345"}]]

```

16. Now go back to **POST** request test (step 14) and add at least three records and call the **GET** request (step 11) and see:

```

Resource
  /student/allstudent
1 [
2   {
3     "name": "Abhijit",
4     "age": 25,
5     "registrationNumber": "12345"
6   },
7   {
8     "name": "Pritam",
9     "age": 26,
10    "registrationNumber": "12346"
11  },
12  {
13    "name": "Dutta",
14    "age": 27,
15    "registrationNumber": "12347"
16  }
17 ]

```

Until now, we have learned how to implement GET and POST services. Now we will learn PUT and DELETE requests.

17. Now introduce the controller classes to handle PUT and DELETE requests. Follow Step 6 above and add the below two controller classes in it.

Create **StudentUpdateController** class and modify it like below:

```
1 package com.bhaiti.kela.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestBody;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestMethod;
7 import org.springframework.web.bind.annotation.ResponseBody;
8
9 import com.bhaiti.kela.beans.Student;
10 import com.bhaiti.kela.beans.StudentRegistration;
11
12 @Controller
13 public class StudentUpdateController {
14
15     @RequestMapping(method = RequestMethod.PUT, value="/update/student")
16
17
18     @ResponseBody
19     public String updateStudentRecord(@RequestBody Student stdn) {
20         System.out.println("In updateStudentRecord");
21         return StudentRegistration.getInstance().updateStudent(stdn);
22     }
23
24 }
```

Create **StudentDeleteController** and modify it like below:

```
1 package com.bhaiti.kela.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RequestMethod;
6 import org.springframework.web.bind.annotation.ResponseBody;
7 import org.springframework.web.bind.annotation.PathVariable;
8
9 import com.bhaiti.kela.beans.StudentRegistration;
10
11 @Controller
12 public class StudentDeleteController {
13
14     @RequestMapping(method = RequestMethod.DELETE, value="/delete/student/{regdNum}")
15
16
17     @ResponseBody
18     public String deleteStudentRecord(@PathVariable("regdNum") String regdNum) {
19         System.out.println("In deleteStudentRecord");
20         return StudentRegistration.getInstance().deleteStudent(regdNum);
21     }
22 }
```

NB: In reality, you don't need four controller classes to handle four different REST service calls. One controller class will suffice, but for clarity, I have introduced four different controller classes in this article.

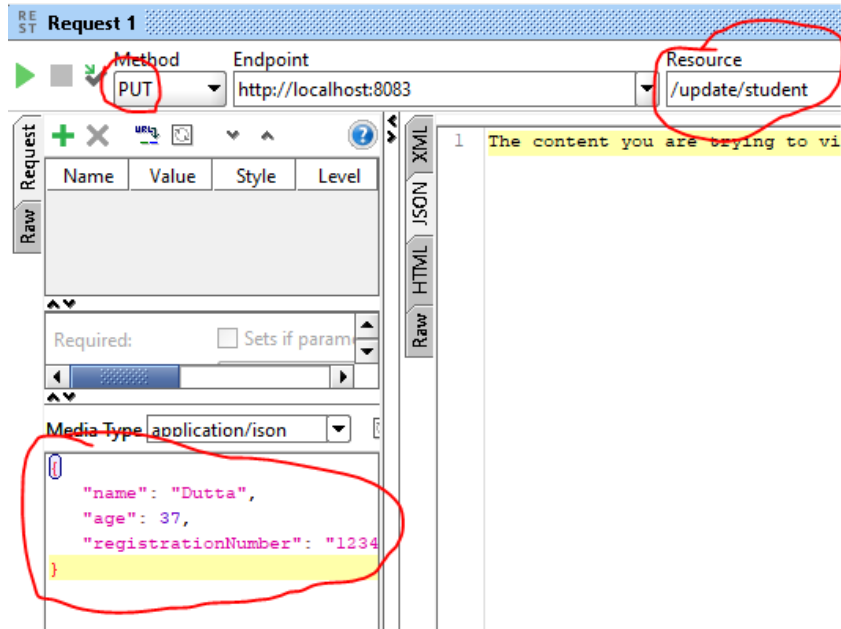
18. Now stop the server (by using Control-C), and compile the code, and run the server again.

19. First, insert three to four records into the system by using POST call and retrieve the records by using GET test mentioned in step

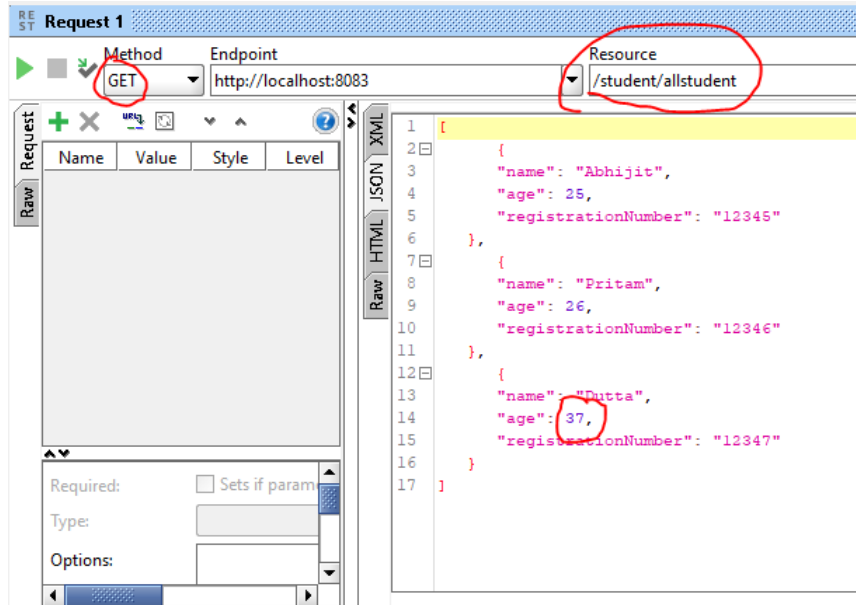
12.

20. Now we will modify a record here, for that create a new REST project in SOAPUI tool and use the below URL and this time select PUT from method combo box and modify the records as shown below:

http://localhost:8083/update/student

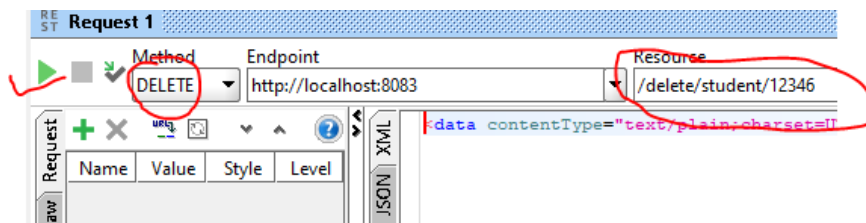


Once you click **green arrow** button this information will be updated in the system. Now see the result, just do the GET call and check.



21. Now finally we will do the **DELETE** service call test. Just create a REST project in SOAPUI and use below URL and select DELETE from method combo box and enter the record like below and click the green arrow button and to check the final outcome just call the GET service call.

http://localhost:8083/delete/student/12346 (the last numeric value is registrationNumber)





Request 1

Method: GET | Endpoint: http://localhost:8083 | Resource: /student/allstudent

| Name | Value | Style | Level |
|------|-------|-------|-------|
|------|-------|-------|-------|

```
1 [
2   {
3     "name": "Abhijit",
4     "age": 25,
5     "registrationNumber": "12345"
6   },
7   {
8     "name": "Dutta",
9     "age": 37,
10    "registrationNumber": "12347"
11  }
12 ]
```

If you liked this article, please do not forget to click the like button and let me know your thoughts in the comments section. Thanks!

## Like This Article? Read More From DZone

[Spring RESTful Web Services Validation: A Complete Blueprint](#)

[Securing REST Services With OAuth2 in Spring Boot](#)

[How to Write Your Own Spring Boot REST Service](#)

[Free DZone Refcard Open Source API Management](#)

Topics: SPRING BOOT , REST API , SPRING 4.1 , JAVA , RESTFUL API , RESTFUL WEB SERVICES , JSON , INTEGERATION , TUTORIAL

Opinions expressed by DZone contributors are their own.