**[BytesTree](http://www.bytestree.com/" \o "BytesTree)**

Technology Blog

* [Home](http://bytestree.com/)
* [Spring](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/)
* [Hibernate](http://www.bytestree.com/tag/hibernate/)
* [Web Services](http://www.bytestree.com/tag/web-services/)



Top of Form



Bottom of Form

**RESTFUL WEB SERVICE CRUD OPERATIONS WITH SPRING BOOT**

 Article Views: 4,377

**Share this:**

* 2
* ***2***

*Shares*

This article shows an example to create RESTful Web Service to perform CRUD operations using Spring Boot framework.

**Contents**

* [Overview](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Overview)
  + [HTTP Methods](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#HTTP_Methods)
* [Video Tutorial](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Video_Tutorial)
* [Technology Stack](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Technology_Stack)
* [Project Setup](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Project_Setup)
* [Restful Web Service](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Restful_Web_Service)
* [Service Layer](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Service_Layer)
* [DAO Layer](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#DAO_Layer)
* [Running the Application](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Running_the_Application)
* [Testing with Postman](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Testing_with_Postman)
* [Source Code](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#Source_Code)

Overview

RESTful web services are based on REST architecture which uses HTTP as communication protocol. A RESTful Web Service is nothing but set of exposed web resources identified by URIs.  Standard HTTP methods like GET, PUT, DELETE and POST are used to access and manipulate these web resources. If you are familiar with SOAP web services then a thing to note that RESTful web service do not have any contract of operations, hence no WSDL.

**You may also interested in:**  
[RESTful Web Services Unit Testing with Spring Boot](http://www.bytestree.com/spring/restful-web-services-unit-testing-spring-boot/)  
[RESTful Web Services Integration Testing with Spring Boot](http://www.bytestree.com/spring/restful-web-services-integration-testing-spring-boot/)  
[RESTful Web Services Authentication and Authorization](http://www.bytestree.com/spring/restful-web-services-authentication-authorization/)

HTTP Methods

The below table shows mapping of CRUD operation and HTTP method:

|  |  |
| --- | --- |
| **CRUD Operation** | **HTTP Method** |
| Create | POST |
| Read | GET |
| Update | PUT |
| Delete | DELETE |

Video Tutorial

If you want see a demo for quick start to create RESTful Web Services in Spring Boot, below video tutorial is suitable for you. Continue reading further if you want more details.

 Technology Stack

Technology stack used in this example is:

* Spring Boot 1.4.1.RELEASE
* Spring Data JPA
* Database – PostgreSQL
* JDK 8

Project Setup

As we are using Spring Boot, project configuration is very strait forward. We don’t even need a web server to host our RESTful Web Services as Spring Boot has embedded Tomcat Server. Let’s start with the pom.xml first.

**pom.xml**

Mains things to configure here are:

* spring-boot-starter-parent
* Dependencies
  + Spring Data JPA
  + Spring Boot Web Starter
  + Database libraries for PostgreSQL

**application.properties**

In this mention following things:

* ContextPath of Web Service – /rest
* database connection details
* Spring JPA configuration
* Logging level of application

We want hibernate to create database tables on startup so we configure spring.jpa.hibernate.ddl-auto=update in application.properties.

**Application.java**

This class loads all configuration, libraries and starts the application. It has @SpringBootApplicationannotation which is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes.

|  |  |
| --- | --- |
|  | package com.bytestree.restful; |
|  |  |
|  | import org.springframework.boot.SpringApplication; |
|  | import org.springframework.boot.autoconfigure.SpringBootApplication; |
|  |  |
|  | @SpringBootApplication |
|  | public class Application { |
|  |  |
|  | public static void main(String[] args) { |
|  | SpringApplication.run(Application.class, args); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/bytestree/798e3758c1144e5d700f5527b5670e8b/raw/ab598dc9f1c54f91a9ae381d9e1ca8822cdc5a9f/Application.java)[**Application.java**](https://gist.github.com/bytestree/798e3758c1144e5d700f5527b5670e8b#file-application-java) hosted with  by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | # Server |
|  | server.contextPath = /rest |
|  |  |
|  | # Database |
|  | spring.datasource.driverClassName=org.postgresql.Driver |
|  | spring.datasource.url=jdbc:postgresql://localhost:5432/postgres |
|  | spring.datasource.username=postgres |
|  | spring.datasource.password=postgres |
|  |  |
|  | # JPA |
|  | spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect |
|  | spring.jpa.hibernate.ddl-auto=update |
|  | spring.jpa.show-sql=true |
|  |  |
|  | # Logging |
|  | logging.level.com.bytestree.restful=DEBUG |

[**view raw**](https://gist.github.com/bytestree/798e3758c1144e5d700f5527b5670e8b/raw/ab598dc9f1c54f91a9ae381d9e1ca8822cdc5a9f/application.properties)[**application.properties**](https://gist.github.com/bytestree/798e3758c1144e5d700f5527b5670e8b#file-application-properties) hosted with  by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" |
|  | xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"> |
|  | <modelVersion>4.0.0</modelVersion> |
|  | <groupId>com.bytestree.restful</groupId> |
|  | <artifactId>spring-restful-service</artifactId> |
|  | <version>1.0.0-SNAPSHOT</version> |
|  | <packaging>jar</packaging> |
|  |  |
|  | <name></name> |
|  | <description></description> |
|  | <parent> |
|  | <groupId>org.springframework.boot</groupId> |
|  | <artifactId>spring-boot-starter-parent</artifactId> |
|  | <version>1.4.1.RELEASE</version> |
|  | <relativePath/> |
|  | </parent> |
|  |  |
|  | <properties> |
|  | <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding> |
|  | <maven.compiler.source>1.8</maven.compiler.source> |
|  | <maven.compiler.target>1.8</maven.compiler.target> |
|  | </properties> |
|  |  |
|  | <dependencies> |
|  | <dependency> |
|  | <groupId>org.springframework.boot</groupId> |
|  | <artifactId>spring-boot-starter-data-jpa</artifactId> |
|  | </dependency> |
|  | <dependency> |
|  | <groupId>org.springframework.boot</groupId> |
|  | <artifactId>spring-boot-starter-web</artifactId> |
|  | </dependency> |
|  |  |
|  | <!-- database --> |
|  | <dependency> |
|  | <groupId>org.postgresql</groupId> |
|  | <artifactId>postgresql</artifactId> |
|  | </dependency> |
|  | </dependencies> |
|  |  |
|  | <build> |
|  | <plugins> |
|  | <plugin> |
|  | <groupId>org.springframework.boot</groupId> |
|  | <artifactId>spring-boot-maven-plugin</artifactId> |
|  | </plugin> |
|  | </plugins> |
|  | </build> |
|  | </project> |

[**view raw**](https://gist.github.com/bytestree/798e3758c1144e5d700f5527b5670e8b/raw/ab598dc9f1c54f91a9ae381d9e1ca8822cdc5a9f/pom.xml)[**pom.xml**](https://gist.github.com/bytestree/798e3758c1144e5d700f5527b5670e8b#file-pom-xml) hosted with  by **[GitHub](https://github.com/)**

Restful Web Service

Now its time to write restful web service class. It’s nothing but simple Spring Controller class having @RestController annotation at class level. First let’s take a look at the code. The explanation on annotation and methods is after that.

|  |  |
| --- | --- |
|  | package com.bytestree.restful.controller; |
|  |  |
|  | import java.util.Arrays; |
|  | import java.util.List; |
|  |  |
|  | import org.apache.log4j.Logger; |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.http.HttpStatus; |
|  | import org.springframework.http.ResponseEntity; |
|  | import org.springframework.web.bind.annotation.PathVariable; |
|  | import org.springframework.web.bind.annotation.RequestBody; |
|  | import org.springframework.web.bind.annotation.RequestMapping; |
|  | import org.springframework.web.bind.annotation.RequestMethod; |
|  | import org.springframework.web.bind.annotation.RestController; |
|  |  |
|  | import com.bytestree.restful.model.Employee; |
|  | import com.bytestree.restful.service.EmployeeService; |
|  |  |
|  | @RestController |
|  | @RequestMapping("/employee") |
|  | public class EmployeeController { |
|  |  |
|  | final static Logger logger = Logger.getLogger(EmployeeController.class); |
|  |  |
|  | @Autowired |
|  | EmployeeService empService; |
|  |  |
|  | @RequestMapping(method = RequestMethod.POST) |
|  | public ResponseEntity<Employee> addEmployee(@RequestBody Employee employee) { |
|  | empService.save(employee); |
|  | logger.debug("Added:: " + employee); |
|  | return new ResponseEntity<Employee>(employee, HttpStatus.CREATED); |
|  | } |
|  |  |
|  |  |
|  | @RequestMapping(method = RequestMethod.PUT) |
|  | public ResponseEntity<Void> updateEmployee(@RequestBody Employee employee) { |
|  | Employee existingEmp = empService.getById(employee.getId()); |
|  | if (existingEmp == null) { |
|  | logger.debug("Employee with id " + employee.getId() + " does not exists"); |
|  | return new ResponseEntity<Void>(HttpStatus.NOT\_FOUND); |
|  | } else { |
|  | empService.save(employee); |
|  | return new ResponseEntity<Void>(HttpStatus.OK); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | @RequestMapping(value = "/{id}", method = RequestMethod.GET) |
|  | public ResponseEntity<Employee> getEmployee(@PathVariable("id") Long id) { |
|  | Employee employee = empService.getById(id); |
|  | if (employee == null) { |
|  | logger.debug("Employee with id " + id + " does not exists"); |
|  | return new ResponseEntity<Employee>(HttpStatus.NOT\_FOUND); |
|  | } |
|  | logger.debug("Found Employee:: " + employee); |
|  | return new ResponseEntity<Employee>(employee, HttpStatus.OK); |
|  | } |
|  |  |
|  |  |
|  | @RequestMapping(method = RequestMethod.GET) |
|  | public ResponseEntity<List<Employee>> getAllEmployees() { |
|  | List<Employee> employees = empService.getAll(); |
|  | if (employees.isEmpty()) { |
|  | logger.debug("Employees does not exists"); |
|  | return new ResponseEntity<List<Employee>>(HttpStatus.NO\_CONTENT); |
|  | } |
|  | logger.debug("Found " + employees.size() + " Employees"); |
|  | logger.debug(employees); |
|  | logger.debug(Arrays.toString(employees.toArray())); |
|  | return new ResponseEntity<List<Employee>>(employees, HttpStatus.OK); |
|  | } |
|  |  |
|  |  |
|  | @RequestMapping(value = "/{id}", method = RequestMethod.DELETE) |
|  | public ResponseEntity<Void> deleteEmployee(@PathVariable("id") Long id) { |
|  | Employee employee = empService.getById(id); |
|  | if (employee == null) { |
|  | logger.debug("Employee with id " + id + " does not exists"); |
|  | return new ResponseEntity<Void>(HttpStatus.NOT\_FOUND); |
|  | } else { |
|  | empService.delete(id); |
|  | logger.debug("Employee with id " + id + " deleted"); |
|  | return new ResponseEntity<Void>(HttpStatus.GONE); |
|  | } |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/bytestree/2da8891c01aaa8cf7f2bb409cccdea54/raw/cd1f523ecc60cbb2cdc8f7cf2899f7a70548a5f6/EmployeeController.java)[**EmployeeController.java**](https://gist.github.com/bytestree/2da8891c01aaa8cf7f2bb409cccdea54#file-employeecontroller-java) hosted with  by **[GitHub](https://github.com/)**

**Request Mapping**

The @RequestMapping annotation is used at class level and method level. The class level annotation defines the path(“/employee“) for which this controller is used. Since we have “/rest” as application context path, any request with “/rest/employee” will handle by this controller.

Each method in this class performs one CRUD operation. In method level @RequestMapping annotation RequestMethod(or HTTP Method) parameter is set. It defines a CRUD operation performed by this method. It is set according to mapping table in Overview > HTTP Methods section of this post.

**Response**

You may have observed that every method is returning [ResponseEntity](http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html). ResponseEntity represents HTTP response returned from these methods. It contains status code, headers, and body of response. In our example we are just setting the Status code and body. In some method we don’t have to send any object as a response. For those cases we just set the body as [Void](https://docs.oracle.com/javase/7/docs/api/java/lang/Void.html) within ResponseEntity. The Status code is set as per the response received from Service layer (EmployeeService). Which status code to set is pretty self explanatory.

Service Layer

The service layer in this example is performing CRUD operations using DAO layer. The CRUDServiceinterface has all methods for CRUD operations.  Entity class should be defined to use this interface. In our case EmployeeService extends this interface with Employee as entity class. DefaultEmployeeService is a concrete class implementing EmployeeService interface. In all cases service method simply calls required method in DAO layer.

|  |  |
| --- | --- |
|  | package com.bytestree.restful.service; |
|  |  |
|  | import java.io.Serializable; |
|  | import java.util.List; |
|  |  |
|  | public interface CRUDService<E> { |
|  |  |
|  | E save(E entity); |
|  |  |
|  | E getById(Serializable id); |
|  |  |
|  | List<E> getAll(); |
|  |  |
|  | void delete(Serializable id); |
|  | } |

[**view raw**](https://gist.github.com/bytestree/ef68b8578696dae5c5b4afde7aed4a8b/raw/f0420a7bc078118c17bf5912c11a94431b35eb71/CRUDService.java)[**CRUDService.java**](https://gist.github.com/bytestree/ef68b8578696dae5c5b4afde7aed4a8b#file-crudservice-java) hosted with  by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | package com.bytestree.restful.service; |
|  |  |
|  | import java.io.Serializable; |
|  | import java.util.List; |
|  |  |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.stereotype.Service; |
|  |  |
|  | import com.bytestree.restful.model.Employee; |
|  | import com.bytestree.restful.repository.EmployeeRepository; |
|  |  |
|  | @Service |
|  | public class DefaultEmployeeService implements EmployeeService { |
|  |  |
|  | @Autowired |
|  | private EmployeeRepository employeeRepository; |
|  |  |
|  | @Override |
|  | public Employee save(Employee entity) { |
|  | return employeeRepository.save(entity); |
|  | } |
|  |  |
|  | @Override |
|  | public Employee getById(Serializable id) { |
|  | return employeeRepository.findOne((Long) id); |
|  | } |
|  |  |
|  | @Override |
|  | public List<Employee> getAll() { |
|  | return employeeRepository.findAll(); |
|  | } |
|  |  |
|  | @Override |
|  | public void delete(Serializable id) { |
|  | employeeRepository.delete((Long) id); |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/bytestree/ef68b8578696dae5c5b4afde7aed4a8b/raw/f0420a7bc078118c17bf5912c11a94431b35eb71/DefaultEmployeeService.java)[**DefaultEmployeeService.java**](https://gist.github.com/bytestree/ef68b8578696dae5c5b4afde7aed4a8b#file-defaultemployeeservice-java) hosted with  by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | package com.bytestree.restful.service; |
|  |  |
|  | import com.bytestree.restful.model.Employee; |
|  |  |
|  | public interface EmployeeService extends CRUDService<Employee> { |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/bytestree/ef68b8578696dae5c5b4afde7aed4a8b/raw/f0420a7bc078118c17bf5912c11a94431b35eb71/EmployeeService.java)[**EmployeeService.java**](https://gist.github.com/bytestree/ef68b8578696dae5c5b4afde7aed4a8b#file-employeeservice-java) hosted with  by **[GitHub](https://github.com/)**

DAO Layer

We are using Spring Data JPA for dao operations which makes all DAO operations very easy. We just have to create a repository interface for each entity and extend it to JpaRepository. No need to write or implement any method. The database connection and other configuration is already present in application.properties file. Spring Data JPA will take of it. Isn’t is so easy? You can explore Spring Data JPA more to know all advantages it provide over normal hibernate DAO implementation. And here is the code for it:

|  |  |
| --- | --- |
|  | package com.bytestree.restful.model; |
|  |  |
|  | import javax.persistence.Column; |
|  | import javax.persistence.Entity; |
|  | import javax.persistence.GeneratedValue; |
|  | import javax.persistence.GenerationType; |
|  | import javax.persistence.Id; |
|  | import javax.persistence.Table; |
|  |  |
|  | @Entity |
|  | @Table(name = "employee") |
|  | public class Employee implements java.io.Serializable { |
|  |  |
|  | private static final long serialVersionUID = 4910225916550731446L; |
|  |  |
|  | @Id |
|  | @GeneratedValue(strategy = GenerationType.IDENTITY) |
|  | @Column(name = "id", unique = true, nullable = false) |
|  | private Long id; |
|  |  |
|  | @Column(name = "firstname", length = 50) |
|  | private String firstname; |
|  |  |
|  | @Column(name = "lastname", length = 50) |
|  | private String lastname; |
|  |  |
|  | @Column(name = "designation", length = 20) |
|  | private String designation; |
|  |  |
|  | @Column(name = "salary") |
|  | private Integer salary; |
|  |  |
|  | public Employee() { |
|  | } |
|  |  |
|  | public Employee(Long id) { |
|  | this.id = id; |
|  | } |
|  |  |
|  | public Employee(Long id, String firstname, String lastname, String designation, Integer salary) { |
|  | this.id = id; |
|  | this.firstname = firstname; |
|  | this.lastname = lastname; |
|  | this.designation = designation; |
|  | this.salary = salary; |
|  | } |
|  |  |
|  | public Employee(String firstname, String lastname, String designation, Integer salary) { |
|  | this.firstname = firstname; |
|  | this.lastname = lastname; |
|  | this.designation = designation; |
|  | this.salary = salary; |
|  | } |
|  |  |
|  | public Long getId() { |
|  | return this.id; |
|  | } |
|  |  |
|  | public void setId(Long id) { |
|  | this.id = id; |
|  | } |
|  |  |
|  | public String getFirstname() { |
|  | return this.firstname; |
|  | } |
|  |  |
|  | public void setFirstname(String firstname) { |
|  | this.firstname = firstname; |
|  | } |
|  |  |
|  | public String getLastname() { |
|  | return this.lastname; |
|  | } |
|  |  |
|  | public void setLastname(String lastname) { |
|  | this.lastname = lastname; |
|  | } |
|  |  |
|  | public String getDesignation() { |
|  | return this.designation; |
|  | } |
|  |  |
|  | public void setDesignation(String designation) { |
|  | this.designation = designation; |
|  | } |
|  |  |
|  | public Integer getSalary() { |
|  | return this.salary; |
|  | } |
|  |  |
|  | public void setSalary(Integer salary) { |
|  | this.salary = salary; |
|  | } |
|  |  |
|  | @Override |
|  | public String toString() { |
|  | StringBuffer sb = new StringBuffer(); |
|  | sb.append("Id: ").append(this.id).append(", firstName: ").append(this.firstname).append(", lastName: ") |
|  | .append(this.lastname).append(", Designation: ").append(this.designation).append(", Salary: ") |
|  | .append(this.salary); |
|  | return sb.toString(); |
|  | } |
|  |  |
|  | @Override |
|  | public boolean equals(Object obj) { |
|  | if (this == obj) |
|  | return true; |
|  | if (id == null || obj == null || getClass() != obj.getClass()) |
|  | return false; |
|  | Employee toCompare = (Employee) obj; |
|  | return id.equals(toCompare.id); |
|  | } |
|  |  |
|  | @Override |
|  | public int hashCode() { |
|  | return id == null ? 0 : id.hashCode(); |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/bytestree/9911a80694256ded7e91f73e199f30f0/raw/c7eb7c254a3224740440c4ece967054e176eb6d5/Employee.java)[**Employee.java**](https://gist.github.com/bytestree/9911a80694256ded7e91f73e199f30f0#file-employee-java) hosted with  by **[GitHub](https://github.com/)**

|  |  |
| --- | --- |
|  | package com.bytestree.restful.repository; |
|  |  |
|  | import org.springframework.data.jpa.repository.JpaRepository; |
|  | import org.springframework.stereotype.Repository; |
|  |  |
|  | import com.bytestree.restful.model.Employee; |
|  |  |
|  | @Repository |
|  | public interface EmployeeRepository extends JpaRepository<Employee, Long> { |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/bytestree/9911a80694256ded7e91f73e199f30f0/raw/c7eb7c254a3224740440c4ece967054e176eb6d5/EmployeeRepository.java)[**EmployeeRepository.java**](https://gist.github.com/bytestree/9911a80694256ded7e91f73e199f30f0#file-employeerepository-java) hosted with  by **[GitHub](https://github.com/)**

Running the Application

Before running the application make sure database connection details are correctly mentioned in application.properties file. You don’t have to create any table. The configuration we are using will  create it if not exists.

To run the application you just have to run the Application.java file. If you are using Eclipse IDE, you can run the Application.java by right click > Run As > Java Application. You don’t have to deploy it on tomcat and start the server. Sprint Boot has embedded tomcat. As you run the application it will take care of everything.

Testing with Postman

We are now ready to test our Restful Web Service. We will use Postman chrome app for testing. If you don’t have it already installed in your Chrome browser then please install it first.

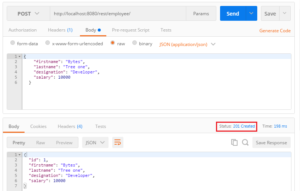
**Create/Add Data**

To add data you need to send **POST** request to <http://localhost:8080/rest/employee/>. The body of this request is as follows:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {  "firstname": "Bytes",  "lastname": "Tree one",  "designation": "Developer",  "salary": 10000  } |

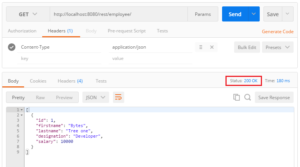
For all operations we need to set the Header “ Content-Type ” to “ application/json “.

[](http://www.bytestree.com/wp-content/uploads/2016/07/RestfulService_Create.png)

Restful Web Service – Create

**Read Data**

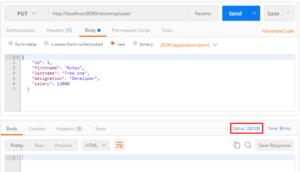
For this you need to send **GET** request on <http://localhost:8080/rest/employee/> .  No Body need to set here (Header still need to set).

[](http://www.bytestree.com/wp-content/uploads/2016/07/RestfulService_Read.png)

Restful Web Service – Read

**Update Data**

For this you need to send **PUT** request on <http://localhost:8080/rest/employee/>. The Body should be the updated request with “id” set to ID of employee we want to update.

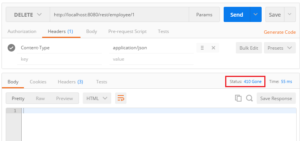
[](http://www.bytestree.com/wp-content/uploads/2016/07/RestfulService_Update.png)

Restful Web Service – Update

To see the updated data, run the Read Data operation again.

**Delete Data**

For this you need to send **DELETE** request to <http://localhost:8080/rest/employee/1> . Here the “1” at the end indicates the ID of employee you want to delete.

[](http://www.bytestree.com/wp-content/uploads/2016/07/RestfulService_Delete.png)

Restful Web Service – Delete

Source Code

The complete source code (maven project) :

Spring Boot 1.4.1.RELEASE + JDK 8: [spring-restful-service-core.zip (493 downloads)](http://www.bytestree.com/download/243/)

**Share this:**

* 2
* ***2***

*Shares*

 September 17, 2016  [BytesTree](http://www.bytestree.com/author/bytestree/)  [Spring](http://www.bytestree.com/category/spring/)  [3 Comments](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/#disqus_thread)

[Spring](http://www.bytestree.com/tag/spring/) [Web Services](http://www.bytestree.com/tag/web-services/)

[← Generic DAO in Hibernate](http://www.bytestree.com/hibernate/generic-dao-hibernate-example/)

[RESTful Web Services Unit Testing with Spring Boot →](http://www.bytestree.com/spring/restful-web-services-unit-testing-spring-boot/)

**RECENT POSTS**

* [Spring 4 + Hibernate 5 Example](http://www.bytestree.com/hibernate/spring-4-hibernate-5-example/)
* [RESTful Web Services Authentication and Authorization](http://www.bytestree.com/spring/restful-web-services-authentication-authorization/)
* [RESTful Web Services Integration Testing with Spring Boot](http://www.bytestree.com/spring/restful-web-services-integration-testing-spring-boot/)
* [RESTful Web Services Unit Testing with Spring Boot](http://www.bytestree.com/spring/restful-web-services-unit-testing-spring-boot/)
* [RESTful Web Service CRUD Operations with Spring Boot](http://www.bytestree.com/spring/restful-web-service-crud-operation-spring-boot-example/)

**TAGS**

[DAO](http://www.bytestree.com/tag/dao/) [Hibernate](http://www.bytestree.com/tag/hibernate/) [Spring](http://www.bytestree.com/tag/spring/) [Spring Security](http://www.bytestree.com/tag/spring-security/) [Testing](http://www.bytestree.com/tag/testing/) [Web Services](http://www.bytestree.com/tag/web-services/)

**CATEGORIES**

* [Hibernate](http://www.bytestree.com/category/hibernate/) (2)
* [Spring](http://www.bytestree.com/category/spring/) (5)

**CATEGORIES**

* [Hibernate](http://www.bytestree.com/category/hibernate/)
* [Spring](http://www.bytestree.com/category/spring/)

All rights reserved.

* [TWITTER](https://twitter.com/bytestree)
* [FACEBOOK](https://www.facebook.com/bytestree)
* [GOOGLE+](https://plus.google.com/u/0/107289920102419681264/)
* [RSS](http://www.bytestree.com/feed/)