WebSystique
learn together

Spring Boot Rest API Example

Created on: December 27, 2016 | **Last updated on:** March 11, 2017 websystiqueadmin

ECOSPORT
Ford
PLATINUM EDITION
Touchscreen with Navigation so you never get lost.
Ford. Feels like family, for a chang
*T&C apply.
Go Further
KNOW MC

Spring Boot complements Spring REST support by providing default dependencies/converters out of the box. Writing RESTful service in Spring Boot is no-different than Spring MVC. If you are a REST Client [Rest Consumer], Spring Boot provides RestTemplateBuilder that can be used to customize the RestTemplate before calling the REST endpoints. To summarize, Spring Boot based REST service is exactly same as Spring based REST service, only differing in the way with we bootstrap the underlying application.

Post Spring CRUD REST Service contains a fairly general introduction to REST and shows a typical CRUD REST service using Spring @RestController and @RestTeamplate.

JSON REST service

Any Spring @RestController in a Spring Boot application will render JSON response by default as long as Jackson2 [jackson-databind] is on the classpath. In a web app [spring-boot-starter-web], it transitively gets included, no need to explicitly include it.

XML REST service

For enabling XML representations, Jackson XML extension (jackson-dataformat-xml) must be present on the classpath. Add the following dependency to your project:

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

Alternatively, if Jackson's XML extension is not available, one could also annotate the POJO with JAXB annotations.

Note : In order to get XML response instead of JSON, client is expected to send appropriate 'Accept' header with value 'text/xml' or 'application/xml'.

Short & Quick introduction to REST

REST stands for Representational State Transfer.It's an is an architectural style which can be used to design web services, that can be consumed from a variety of clients. The core idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls among them.

In Rest based design, resources are being manipulated using a common set of verbs.

Websystique
1,038 likes
WebSystique
Like Page
Share

WebSystique
Follow

- To Create a resource : HTTP POST should be used
- To Retrieve a resource : HTTP GET should be used
- To Update a resource : HTTP PUT should be used
- To Delete a resource : HTTP DELETE should be used

That means, you as a REST service developer or Client, should comply to above criteria, in order to be REST complained.

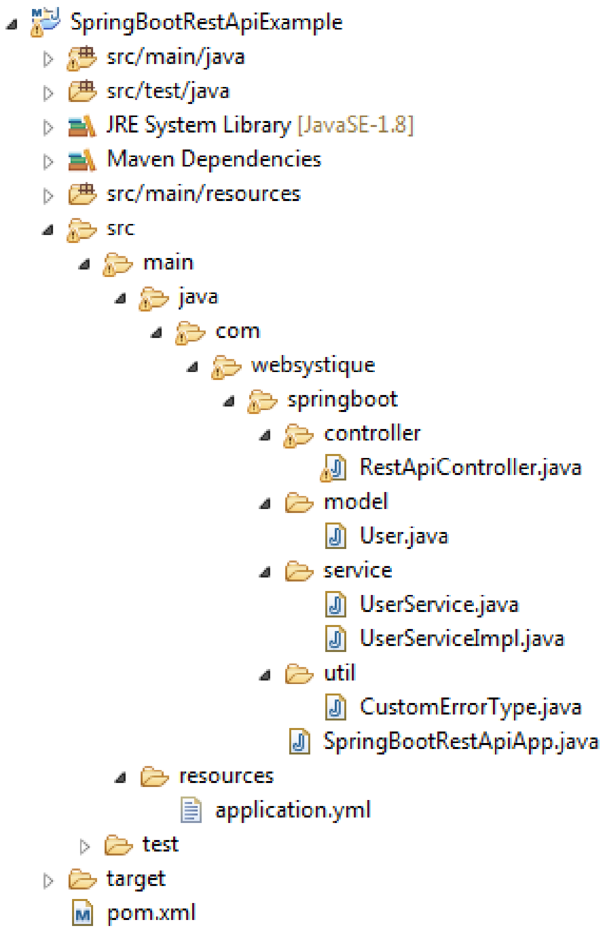
Often Rest based Web services return JSON or XML as response, although it is not limited to these types only. Clients can specify (using HTTP **Accept header**) the resource type they are interested in, and server may return the resource , specifying **Content-Type** of the resource it is serving. This StackOverflow link is a must read to understand REST in detail.

Following technologies stack being used:

- Spring Boot 1.4.3.RELEASE
- Spring 4.3.5.RELEASE [transitively]
- Maven 3.1
- JDK 1.8
- Eclipse MARS.1

Let's begin

Project Structure



```
SpringBootRestApiExample
    src/main/java
    src/test/java
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src/main/resources
    src
        main
            java
                com
                    websystique
                        springboot
                            controller
                                RestApiController.java
                            model
                                User.java
                            service
                                UserService.java
                                UserServiceImpl.java
                            util
                                CustomErrorType.java
                            SpringBootRestApiApp.java
            resources
                application.yml
        test
    target
    pom.xml
```

Recent Posts

- Spring Boot + AngularJS + Spring Data + JPA CRUD App Example
- Spring Boot Rest API Example
- Spring Boot WAR deployment example
- Spring Boot Introduction + hello world example
- Secure Spring REST API using OAuth2

1. Dependency Management [pom.xml]

```
<project xmlns="<a class="vglnk" href="http://maven.apache.org/POM/4.0.0" rel="nofollow"><span>h
    xsi:schemaLocation="<a class="vglnk" href="http://maven.apache.org/POM/4.0.0" rel="nofollow"
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.websystique.springboot</groupId>
    <artifactId>SpringBootRestApiExample</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <name>SpringBootRestApiExample</name>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.4.3.RELEASE</version>
    </parent>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <!-- Add typical dependencies for a web application -->
        <!-- Adds Tomcat and Spring MVC, along others, jackson-databind included transitively --
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin><!-- Include if you want to make an executable jar[FAT JAR which includes al
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2. Main class [Application]

Typical Spring Boot Application class, nothing special.

```
package com.websystique.springboot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication(scanBasePackages={"com.websystique.springboot"})// same as @Configuration
public class SpringBootRestApiApp {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootRestApiApp.class, args);
    }
}
```

3. REST Controller

Following is one possible Rest based controller, implementing REST API. I said possible, means Other's may implement it in another way, still (or even more pure way) conforming to REST style.

This is what our REST API does:

- **GET** request to /api/user/ returns a list of users
- **GET** request to /api/user/1 returns the user with ID 1
- **POST** request to /api/user/ with a user object as JSON creates a new user
- **PUT** request to /api/user/3 with a user object as JSON updates the user with ID 3
- **DELETE** request to /api/user/4 deletes the user with ID 4
- **DELETE** request to /api/user/ deletes all the users

```java
package com.websystique.springboot.controller;

import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
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 org.springframework.web.util.UriComponentsBuilder;

import com.websystique.springboot.model.User;
import com.websystique.springboot.service.UserService;
import com.websystique.springboot.util.CustomErrorType;

@RestController
@RequestMapping("/api")
public class RestApiController {

    public static final Logger logger = LoggerFactory.getLogger(RestApiController.class);
    @Autowired
    UserService userService; //Service which will do all data retrieval/manipulation work

    // -------------------Retrieve All Users---------------------------------------------

    @RequestMapping(value = "/user/", method = RequestMethod.GET)
    public ResponseEntity<List<User>> listAllUsers() {
        List<User> users = userService.findAllUsers();
        if (users.isEmpty()) {
            return new ResponseEntity(HttpStatus.NO_CONTENT);
            // You many decide to return HttpStatus.NOT_FOUND
        }
        return new ResponseEntity<List<User>>(users, HttpStatus.OK);
    }

    // -------------------Retrieve Single User------------------------------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    public ResponseEntity<?> getUser(@PathVariable("id") long id) {
        logger.info("Fetching User with id {}", id);
        User user = userService.findById(id);
        if (user == null) {
            logger.error("User with id {} not found.", id);
            return new ResponseEntity(new CustomErrorType("User with id " + id
                    + " not found"), HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<User>(user, HttpStatus.OK);
    }

    // -------------------Create a User-------------------------------------------

    @RequestMapping(value = "/user/", method = RequestMethod.POST)
    public ResponseEntity<?> createUser(@RequestBody User user, UriComponentsBuilder ucBuilder)
        logger.info("Creating User : {}", user);

        if (userService.isUserExist(user)) {
            logger.error("Unable to create. A User with name {} already exist", user.getName());
            return new ResponseEntity(new CustomErrorType("Unable to create. A User with name "
            user.getName() + " already exist."),HttpStatus.CONFLICT);
        }
        userService.saveUser(user);

        HttpHeaders headers = new HttpHeaders();
        headers.setLocation(ucBuilder.path("/api/user/{id}").buildAndExpand(user.getId()).toUri(
        return new ResponseEntity<String>(headers, HttpStatus.CREATED);
    }

    // ------------------- Update a User ------------------------------------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.PUT)
    public ResponseEntity<?> updateUser(@PathVariable("id") long id, @RequestBody User user) {
        logger.info("Updating User with id {}", id);

        User currentUser = userService.findById(id);
        if (currentUser == null) {
            logger.error("Unable to update. User with id {} not found.", id);
            return new ResponseEntity(new CustomErrorType("Unable to upate. User with id " + id
                    HttpStatus.NOT_FOUND);
        }

        currentUser.setName(user.getName());
        currentUser.setAge(user.getAge());
        currentUser.setSalary(user.getSalary());

        userService.updateUser(currentUser);
        return new ResponseEntity<User>(currentUser, HttpStatus.OK);
    }

    // ------------------- Delete a User-----------------------------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<?> deleteUser(@PathVariable("id") long id) {
        logger.info("Fetching & Deleting User with id {}", id);

        User user = userService.findById(id);
        if (user == null) {
            logger.error("Unable to delete. User with id {} not found.", id);
            return new ResponseEntity(new CustomErrorType("Unable to delete. User with id " + id
                    HttpStatus.NOT_FOUND);
        }
        userService.deleteUserById(id);
        return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
    }

    // ------------------- Delete All Users-----------------------------

    @RequestMapping(value = "/user/", method = RequestMethod.DELETE)
    public ResponseEntity<User> deleteAllUsers() {
        logger.info("Deleting All Users");

        userService.deleteAllUsers();
        return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
    }

}
```

Detailed Explanation :

@RestController : First of all, we are using Spring 4′s new @RestController annotation. This annotation eliminates the need of annotating each method with @ResponseBody. Under the hood, @RestController is itself annotated with @ResponseBody, and can be considered as combination of @Controller and @ResponseBody.

@RequestBody : If a method parameter is annotated with @RequestBody, Spring will bind the incoming HTTP request body(for the URL mentioned in @RequestMapping for that method) to that parameter. While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the HTTP request body into domain object [deserialize request body to domain object], based on ACCEPT or Content-Type header present in request.

@ResponseBody : If a method is annotated with @ResponseBody, Spring will bind the return value to outgoing HTTP response body. While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the return value to HTTP response body [serialize the object to response body], based on Content-Type present in request HTTP header. As already mentioned, in Spring 4, you may stop using this annotation.

ResponseEntity is a real deal. It represents the entire HTTP response. Good thing about it is that you can control anything that goes into it. You can specify status code, headers, and body. It comes with several constructors to carry the information you want to sent in HTTP Response.

@PathVariable This annotation indicates that a method parameter should be bound to a URI template variable [the one in '{}'].

Basically, @RestController , @RequestBody, ResponseEntity & @PathVariable are all you need to know to implement a REST API in Spring. Additionally, spring provides several support classes to help you implement something customized.

MediaType : Although we didn't, with @RequestMapping annotation, you can additionally, specify the MediaType to be produced or consumed (using **produces** or **consumes** attributes) by that particular controller method, to further narrow down the mapping.

```
package com.websystique.springboot.util;


public class CustomErrorType {

    private String errorMessage;

    public CustomErrorType(String errorMessage){
        this.errorMessage = errorMessage;
    }
    public String getErrorMessage() {
        return errorMessage;
    }

}
```

Testing the API

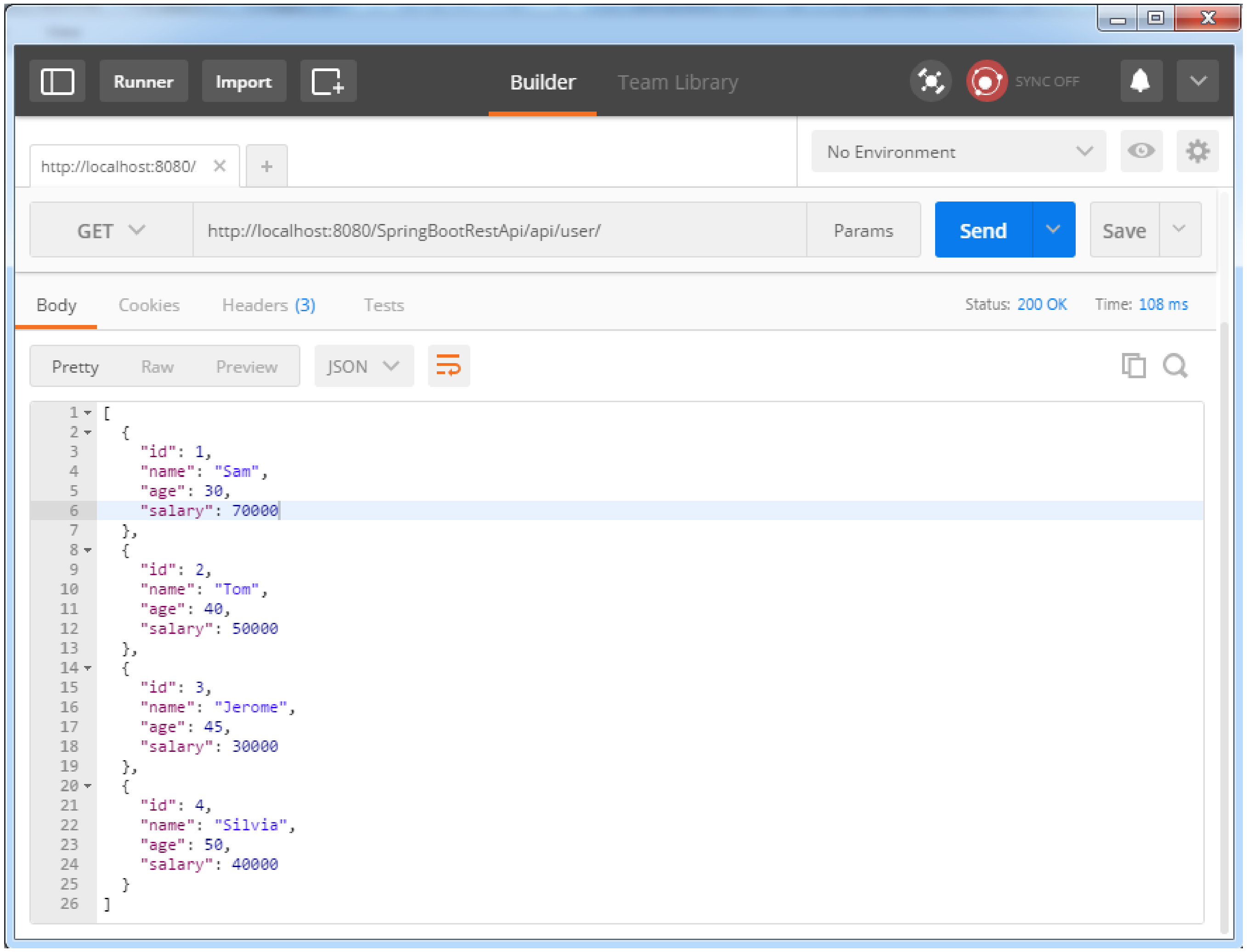
Let’s run the application[from IDE or on command line]. To test this API, i will use an external client POSTMAN (An extension from CHROME). We will write our own client in just few minutes.

Apart from IDE, you can also run this app using following approaches.

- **java -jar path-to-jar**
- on Project root , **mvn spring-boot:run**

1. Retrieve all users

Open POSTMAN tool, select request type [GET for this usecase], specify the uri **http://localhost:8080/SpringBootRestApi/api/user/** and Send., should retrieve all users.

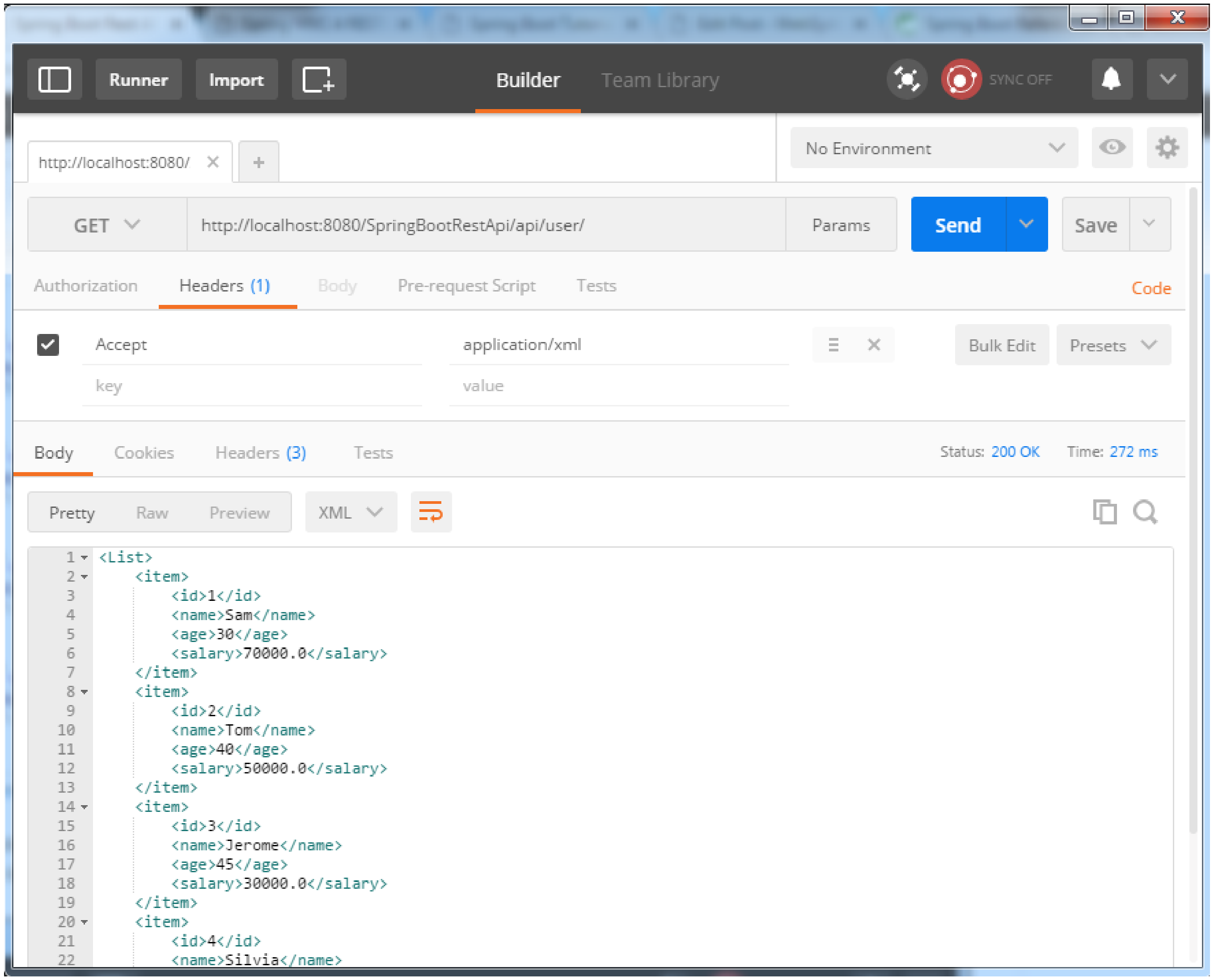


GET http://localhost:8080/SpringBootRestApi/api/user/

Status: 200 OK Time: 108 ms

```
[
  {
    "id": 1,
    "name": "Sam",
    "age": 30,
    "salary": 70000
  },
  {
    "id": 2,
    "name": "Tom",
    "age": 40,
    "salary": 50000
  },
  {
    "id": 3,
    "name": "Jerome",
    "age": 45,
    "salary": 30000
  },
  {
    "id": 4,
    "name": "Silvia",
    "age": 50,
    "salary": 40000
  }
]
```

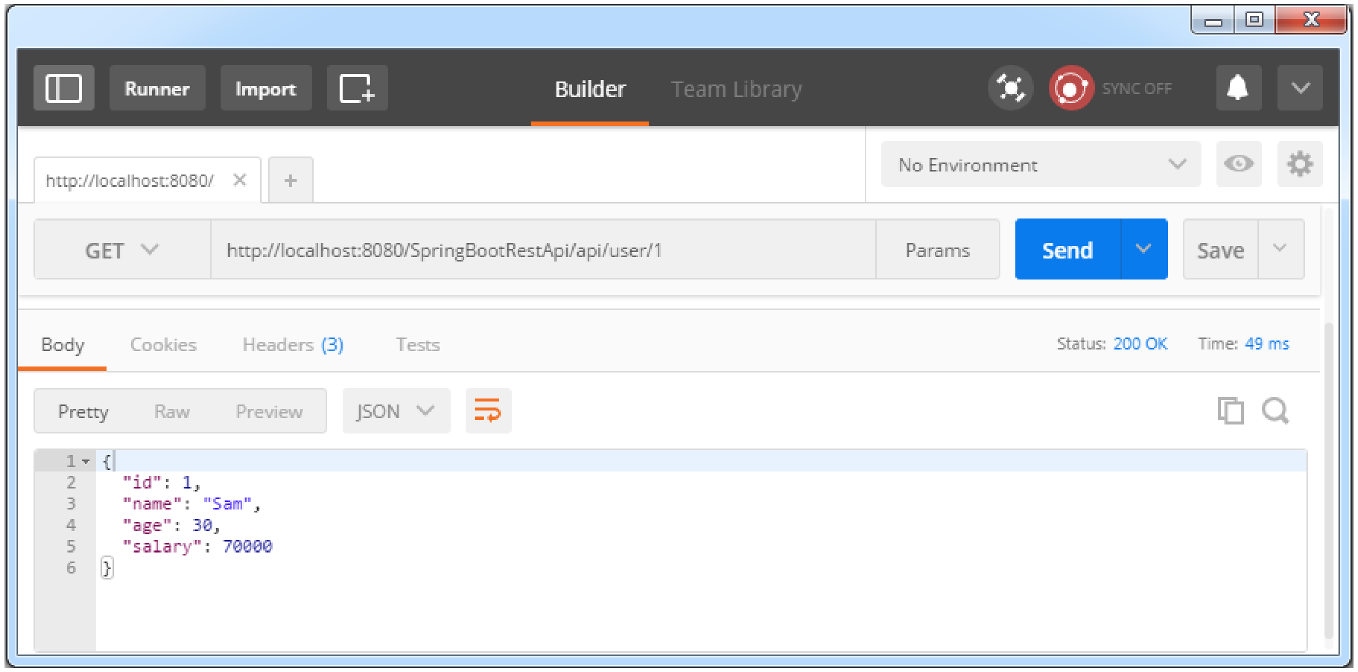
Now, let's retry the GET, with an Accept header this time with value 'application/xml'. You should get XML response, thanks to `jackson-dataformat-xml` available in classpath, server was able to return an XML response based on 'Accept header'.



Accept	application/xml
key	value

```
<List>
    <item>
        <id>1</id>
        <name>Sam</name>
        <age>30</age>
        <salary>70000.0</salary>
    </item>
    <item>
        <id>2</id>
        <name>Tom</name>
        <age>40</age>
        <salary>50000.0</salary>
    </item>
    <item>
        <id>3</id>
        <name>Jerome</name>
        <age>45</age>
        <salary>30000.0</salary>
    </item>
    <item>
        <id>4</id>
        <name>Silvia</name>
```

2. Retrieve a single user
Use GET, specify the id of the user you are looking for and send.

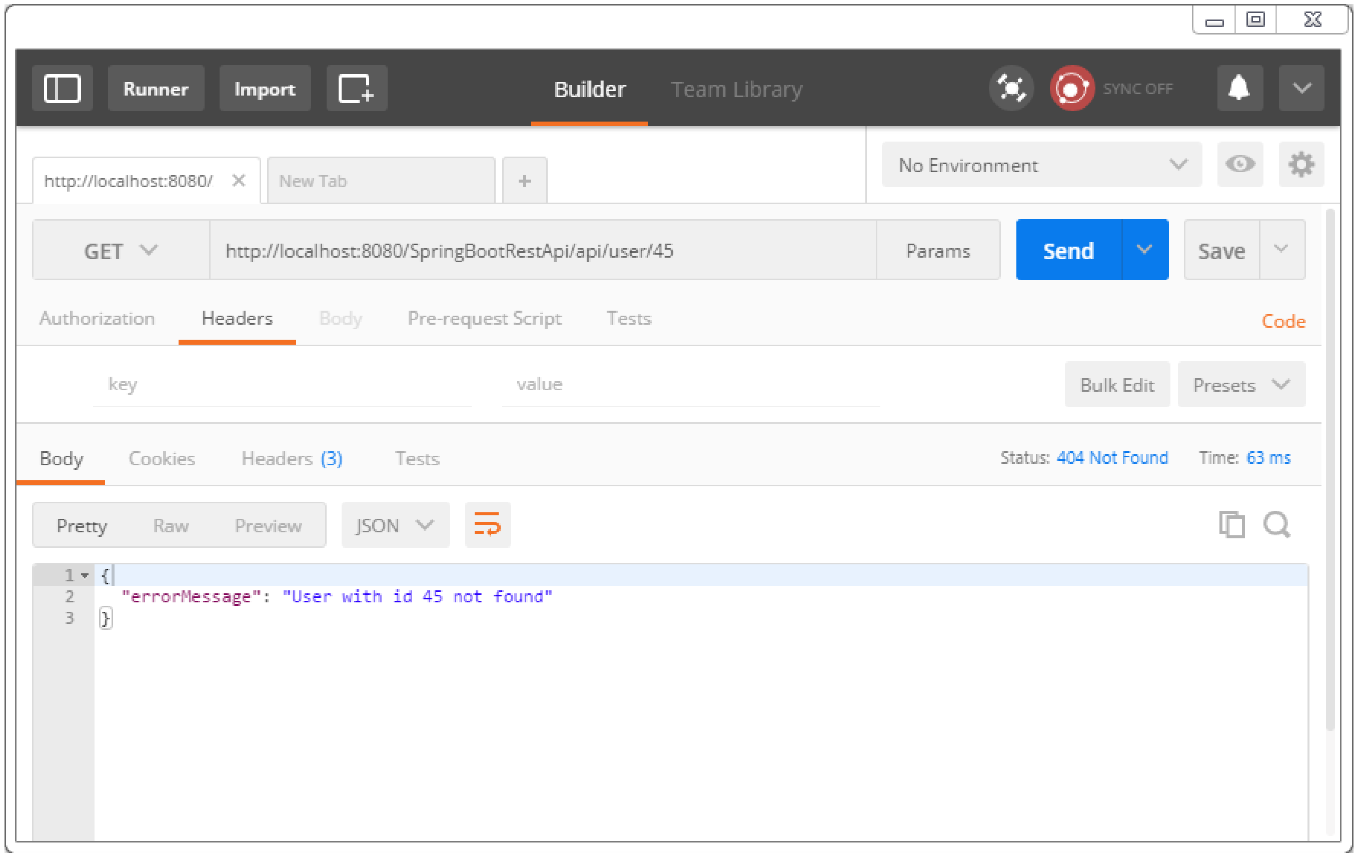


GET http://localhost:8080/SpringBootRestApi/api/user/1

Status: 200 OK Time: 49 ms

```
{
  "id": 1,
  "name": "Sam",
  "age": 30,
  "salary": 70000
}
```

3. Retrieve an unknown user
Use GET, specify the wrong id of the user, and send.

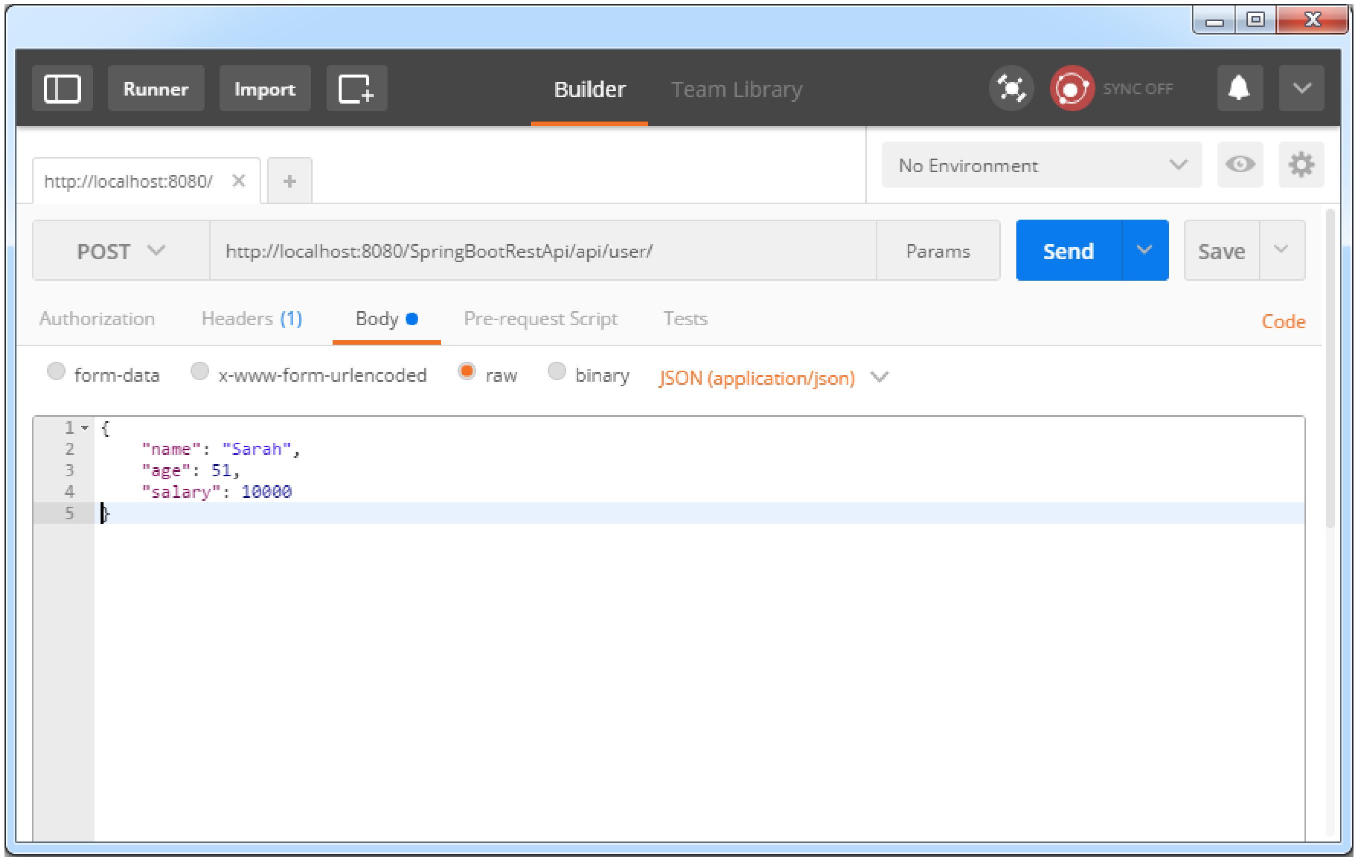


GET http://localhost:8080/SpringBootRestApi/api/user/45

Status: 404 Not Found Time: 63 ms

```
{
  "errorMessage": "User with id 45 not found"
}
```

4. Create a User
Use POST, specify the content in body, select content-type as ‘application/json’

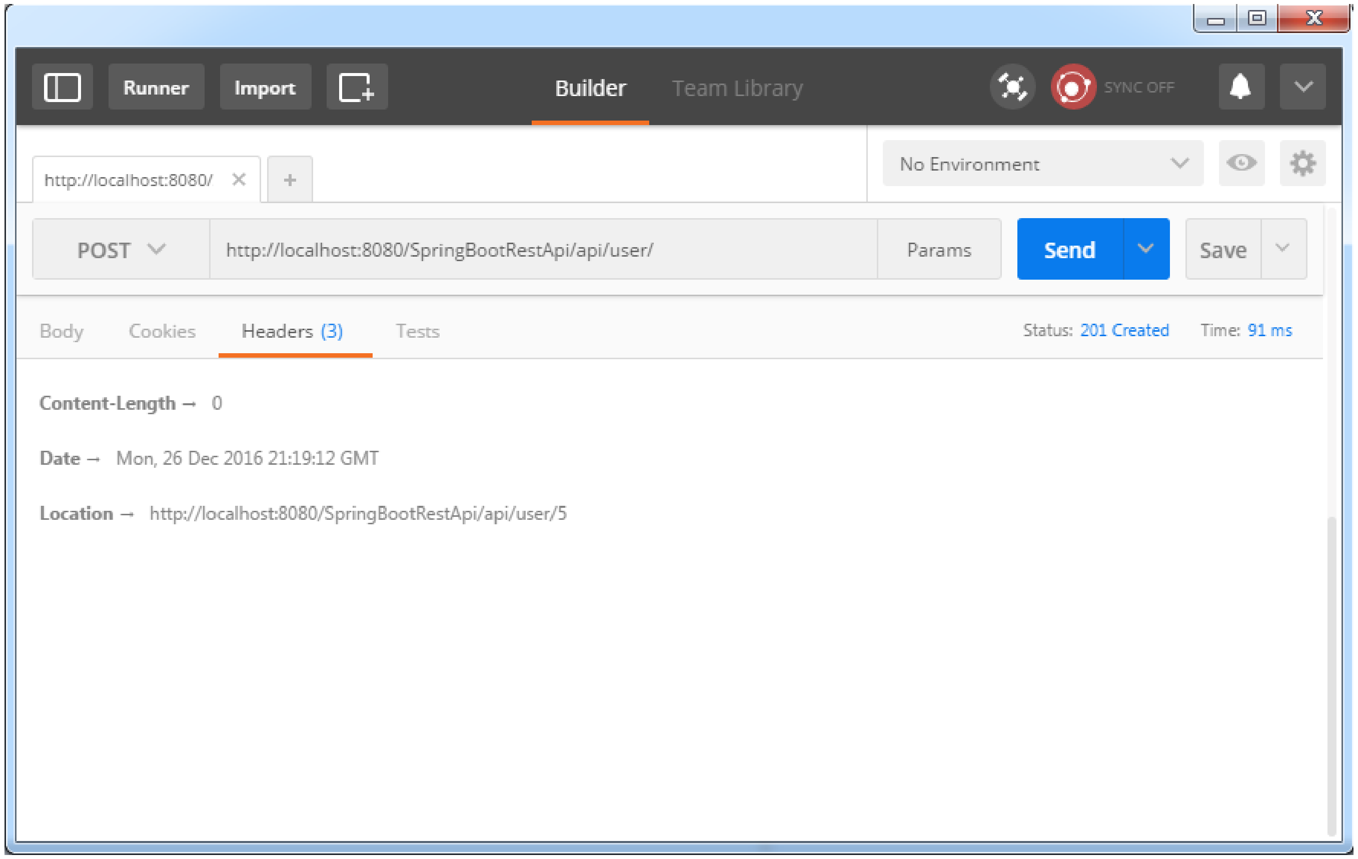


POST http://localhost:8080/SpringBootRestApi/api/user/

raw: JSON (application/json)

```
{
    "name": "Sarah",
    "age": 51,
    "salary": 10000
}
```

Send. New user would be created and will be accessible at the location mentioned in **Location** header.

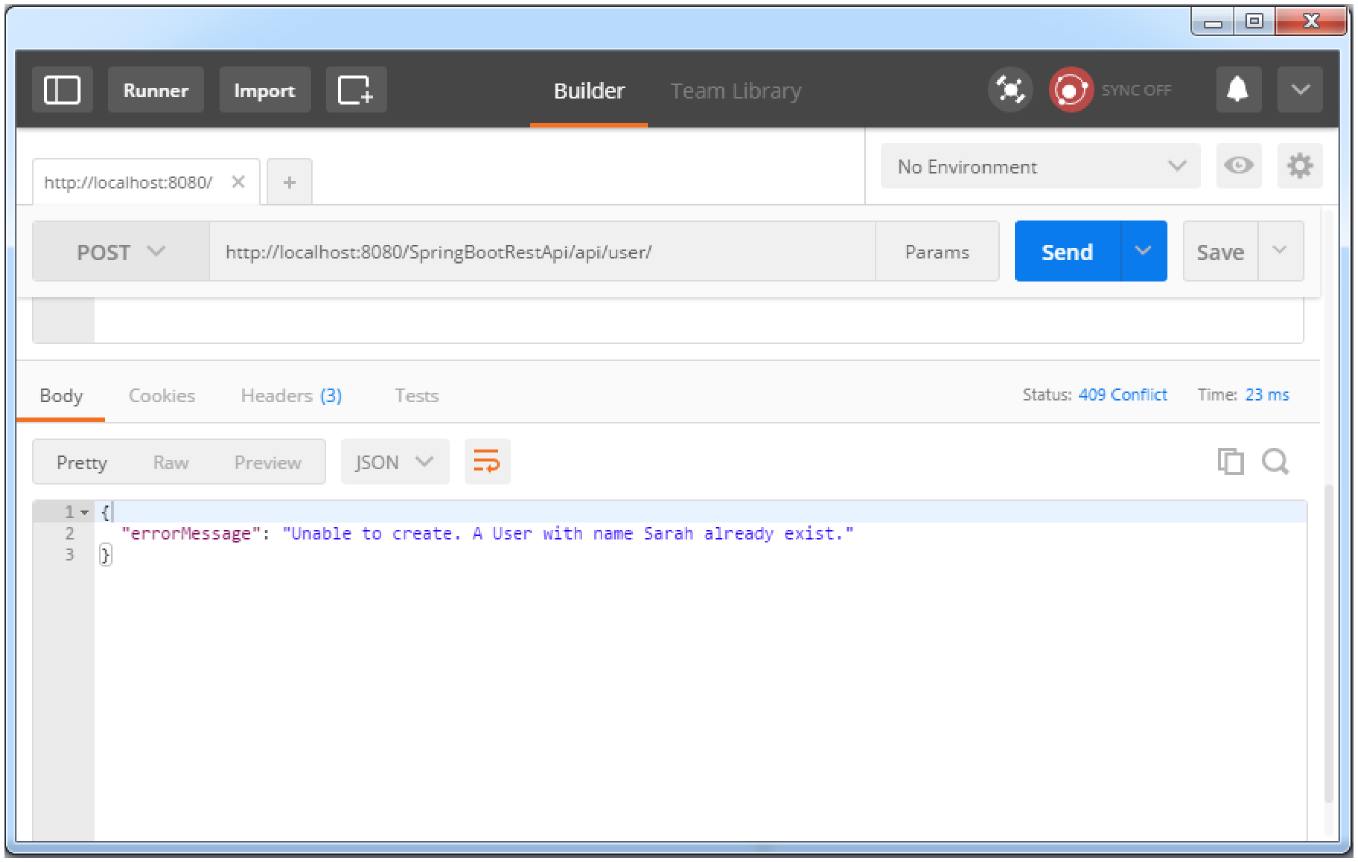


POST http://localhost:8080/SpringBootRestApi/api/user/

Status: 201 Created Time: 91 ms

```
Content-Length → 0
Date → Mon, 26 Dec 2016 21:19:12 GMT
Location → http://localhost:8080/SpringBootRestApi/api/user/5
```

5. Create a User with an existing user-name
Use POST, specify the content in body with name of an existing user,Send, should get a 409/conflict.

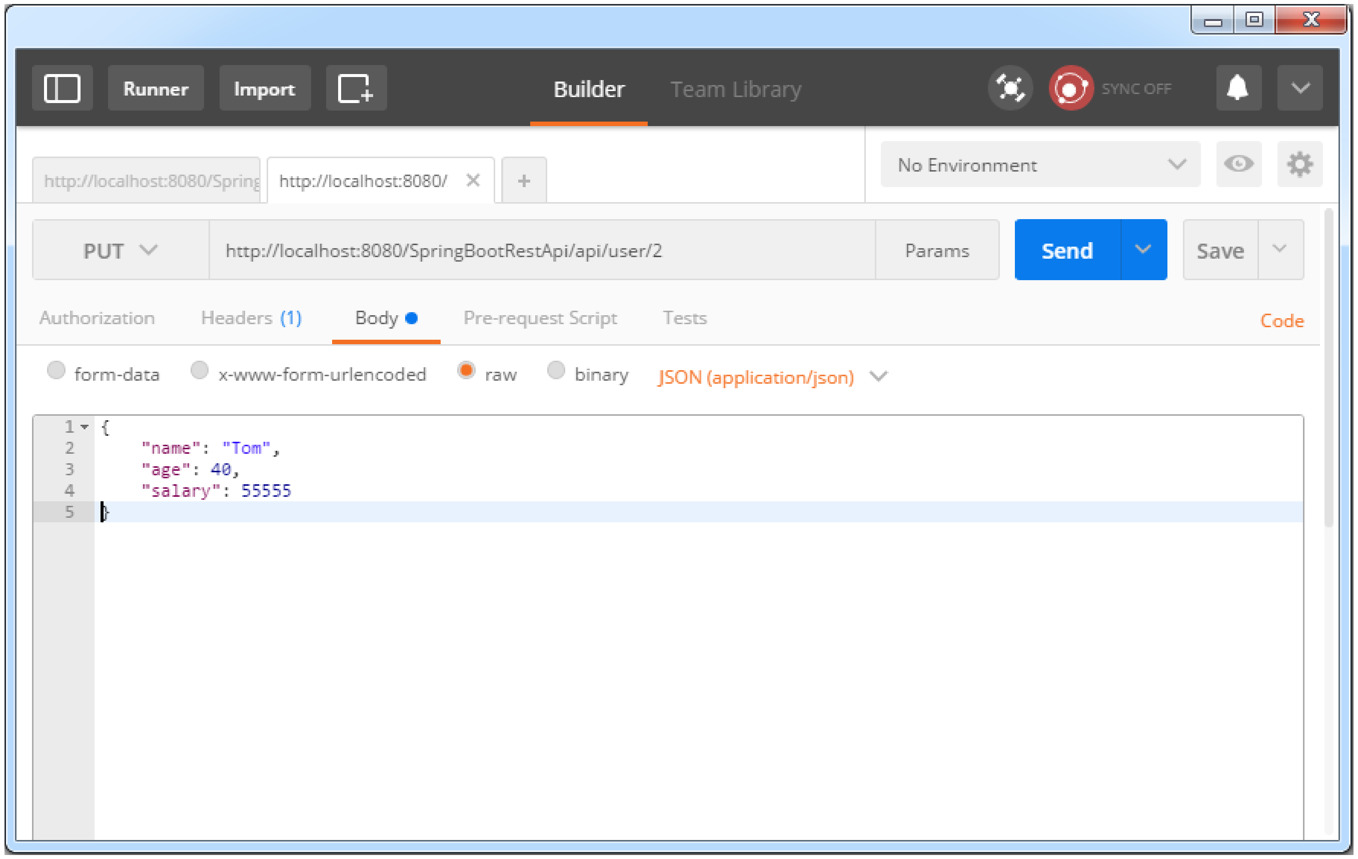


POST http://localhost:8080/SpringBootRestApi/api/user/

Status: 409 Conflict Time: 23 ms

```
{
  "errorMessage": "Unable to create. A User with name Sarah already exist."
}
```

7. Update an existing user
Use PUT, specify the content in body and type as ‘application/json’.

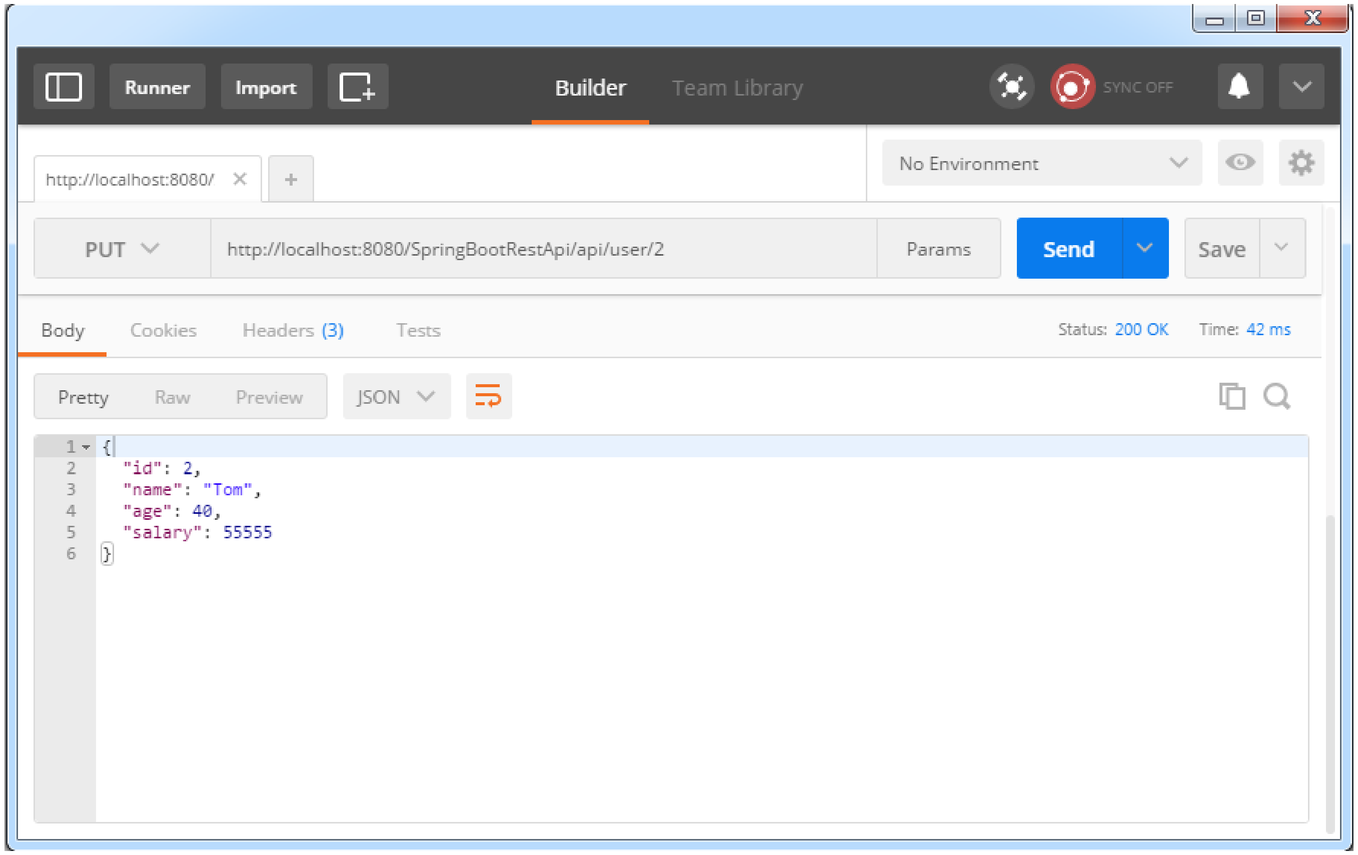


PUT http://localhost:8080/SpringBootRestApi/api/user/2

raw — JSON (application/json)

```
{
    "name": "Tom",
    "age": 40,
    "salary": 55555
}
```

Send, the user should be updated at server.



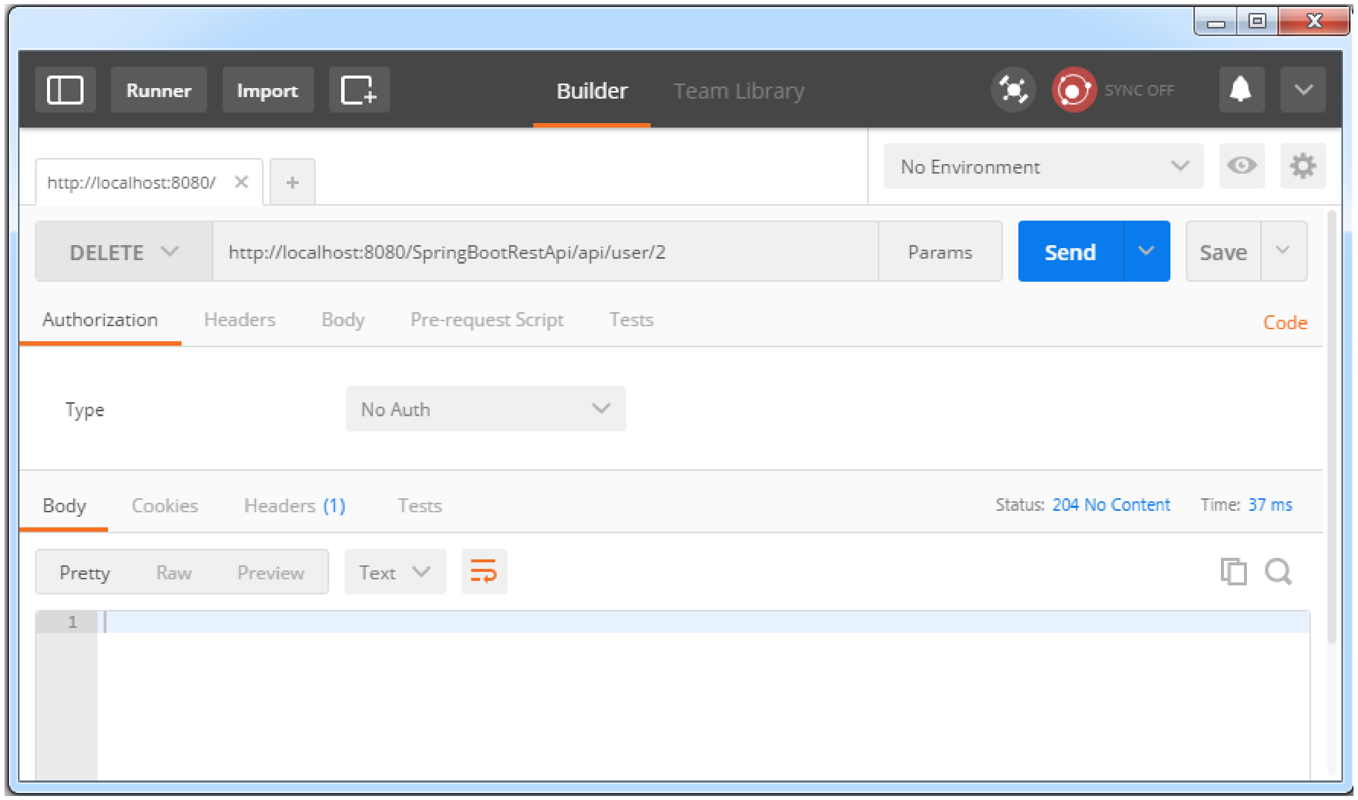
PUT http://localhost:8080/SpringBootRestApi/api/user/2

Status: 200 OK Time: 42 ms

```
{
  "id": 2,
  "name": "Tom",
  "age": 40,
  "salary": 55555
}
```

8. Delete an existing user

Use DELETE, specify the id in url, send. User should be deleted from server.

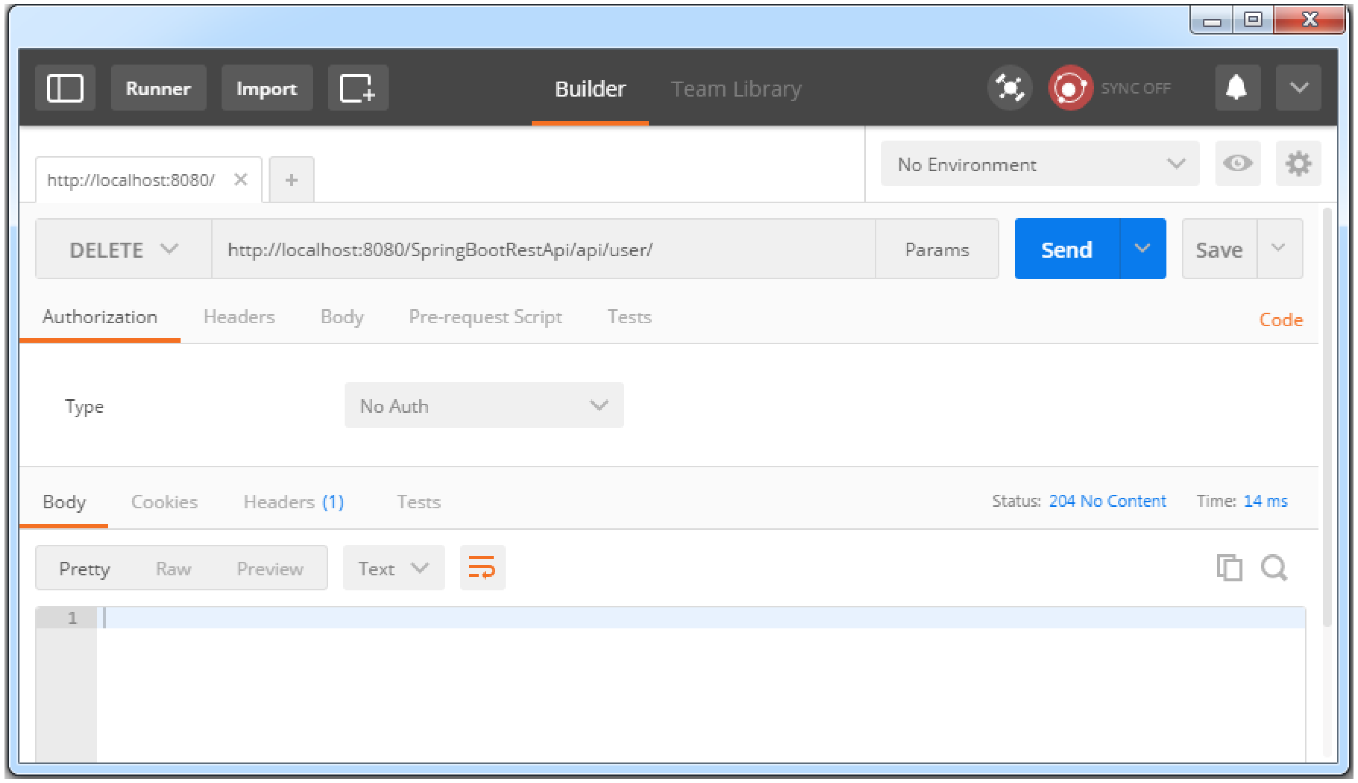


DELETE http://localhost:8080/SpringBootRestApi/api/user/2

Type: No Auth

Status: 204 No Content Time: 37 ms

8. Delete all users
Use DELETE, do not specify any id, send. All users should be deleted from server.

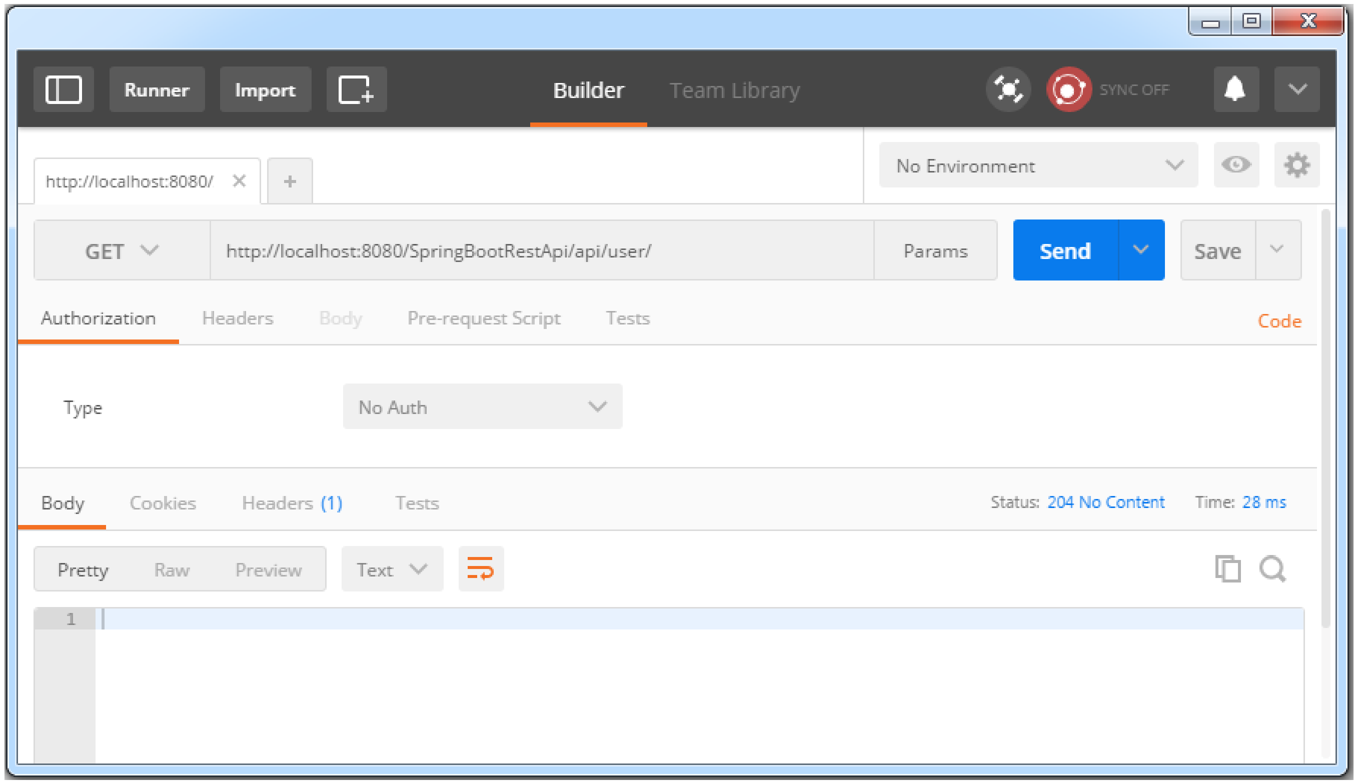


DELETE http://localhost:8080/SpringBootRestApi/api/user/

Type No Auth

Status: 204 No Content Time: 14 ms

8. Verify the results
Use GET to retrieve all users, send, should not get any user in response.



4. Writing REST Client using RestTemplate

Postman tool we used above is a wonderful Client to test Rest API. But if you want to consume REST based web services from your application, you would need a REST client for your application. One of the most popular HTTP client is Apache HttpComponents **HttpClient**. But the details to access REST services using this are too low level.

Spring's `RestTemplate` comes to Rescue. RestTemplate provides higher level methods that correspond to each of the six main HTTP methods that make invoking many RESTful services a one-liner and enforce REST best practices.

Spring Boot provides RestTemplateBuilder that can be used to customize the RestTemplate before calling the REST endpoints. Since in this post we are not customizing the REST template [No additional header e.g.], we may prefer to directly use RestTemplate.

Below shown are HTTP methods and corresponding RestTemplate methods to handle that type of HTTP request.

HTTP Methods and corresponding RestTemplate methods:

- HTTP GET : getForObject, getForEntity
- HTTP PUT : put(String url, Object request, String…urlVariables)
- HTTP DELETE : delete
- HTTP POST : postForLocation(String url, Object request, String… urlVariables), postForObject(String url, Object request, Class responseType, String… uriVariables)
- HTTP HEAD : headForHeaders(String url, String… urlVariables)
- HTTP OPTIONS : optionsForAllow(String url, String… urlVariables)
- HTTP PATCH and others : exchange execute

Custom Rest client , consuming the REST services created earlier.

```
package com.websystique.springboot;

import java.net.URI;
import java.util.LinkedHashMap;
import java.util.List;

import com.websystique.springboot.model.User;
import org.springframework.web.client.RestTemplate;


public class SpringBootRestTestClient {

    public static final String REST_SERVICE_URI = "<a class="vglnk" href="http://localhost:8080/

    /* GET */
    @SuppressWarnings("unchecked")
    private static void listAllUsers(){
        System.out.println("Testing listAllUsers API-----------");

        RestTemplate restTemplate = new RestTemplate();
        List<LinkedHashMap<String, Object>> usersMap = restTemplate.getForObject(REST_SERVICE_UR

        if(usersMap!=null){
            for(LinkedHashMap<String, Object> map : usersMap){
                System.out.println("User : id="+map.get("id")+", Name="+map.get("name")+", Age="
            }
        }else{
            System.out.println("No user exist----------");
        }
    }

    /* GET */
    private static void getUser(){
        System.out.println("Testing getUser API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user = restTemplate.getForObject(REST_SERVICE_URI+"/user/1", User.class);
        System.out.println(user);
    }

    /* POST */
    private static void createUser() {
        System.out.println("Testing create User API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user = new User(0,"Sarah",51,134);
        URI uri = restTemplate.postForLocation(REST_SERVICE_URI+"/user/", user, User.class);
        System.out.println("Location : "+uri.toASCIIString());
    }

    /* PUT */
    private static void updateUser() {
        System.out.println("Testing update User API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user  = new User(1,"Tomy",33, 70000);
        restTemplate.put(REST_SERVICE_URI+"/user/1", user);
        System.out.println(user);
    }

    /* DELETE */
    private static void deleteUser() {
        System.out.println("Testing delete User API----------");
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.delete(REST_SERVICE_URI+"/user/3");
    }


    /* DELETE */
    private static void deleteAllUsers() {
        System.out.println("Testing all delete Users API----------");
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.delete(REST_SERVICE_URI+"/user/");
    }
    public static void main(String args[]){
        listAllUsers();
        getUser();
        createUser();
        listAllUsers();
        updateUser();
        listAllUsers();
        deleteUser();
        listAllUsers();
        deleteAllUsers();
        listAllUsers();
    }
}
```

Now restart the app in order to repopulate the data, and then run this client.

output:

```
Testing listAllUsers API-----------
21:59:39.198 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for
21:59:39.258 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept h
21:59:39.613 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "<a clas
21:59:39.615 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [interface java.
User : id=1, Name=Sam, Age=30, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
Testing getUser API----------
21:59:39.730 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for
21:59:39.806 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept h
21:59:39.839 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "<a clas
21:59:39.840 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [class com.websy
User [id=1, name=Sam, age=30, salary=70000.0]
Testing create User API----------
21:59:39.860 [main] DEBUG org.springframework.web.client.RestTemplate - Created POST request for
21:59:39.879 [main] DEBUG org.springframework.web.client.RestTemplate - Writing [User [id=0, nam
21:59:40.043 [main] DEBUG org.springframework.web.client.RestTemplate - POST request for "<a cla
Location : <a class="vglnk" href="http://localhost:8080/SpringBootRestApi/api/user/5" rel="nofol
Testing listAllUsers API-----------
21:59:40.057 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for
21:59:40.059 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept h
21:59:40.066 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "<a clas
21:59:40.066 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [interface java.
User : id=1, Name=Sam, Age=30, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0
Testing update User API----------
21:59:40.083 [main] DEBUG org.springframework.web.client.RestTemplate - Created PUT request for
```

```
21:59:40.086 [main] DEBUG org.springframework.web.client.RestTemplate - Writing [User [id=1, nam
21:59:40.105 [main] DEBUG org.springframework.web.client.RestTemplate - PUT request for "<a clas
User [id=1, name=Tomy, age=33, salary=70000.0]
Testing listAllUsers API-----------
21:59:40.116 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for
21:59:40.118 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept h
21:59:40.129 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "<a clas
21:59:40.129 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [interface java.
User : id=1, Name=Tomy, Age=33, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0
Testing delete User API----------
21:59:40.148 [main] DEBUG org.springframework.web.client.RestTemplate - Created DELETE request f
21:59:40.161 [main] DEBUG org.springframework.web.client.RestTemplate - DELETE request for "<a c
Testing listAllUsers API-----------
21:59:40.183 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for
21:59:40.185 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept h
21:59:40.191 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "<a clas
21:59:40.191 [main] DEBUG org.springframework.web.client.RestTemplate - Reading [interface java.
User : id=1, Name=Tomy, Age=33, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0
Testing all delete Users API----------
21:59:40.210 [main] DEBUG org.springframework.web.client.RestTemplate - Created DELETE request f
21:59:40.217 [main] DEBUG org.springframework.web.client.RestTemplate - DELETE request for "<a c
Testing listAllUsers API-----------
21:59:40.233 [main] DEBUG org.springframework.web.client.RestTemplate - Created GET request for
21:59:40.235 [main] DEBUG org.springframework.web.client.RestTemplate - Setting request Accept h
21:59:40.245 [main] DEBUG org.springframework.web.client.RestTemplate - GET request for "<a clas
No user exist----------
```

5. Service & model

```
package com.websystique.springboot.service;


import java.util.List;

import com.websystique.springboot.model.User;

public interface UserService {

    User findById(long id);

    User findByName(String name);
    void saveUser(User user);

    void updateUser(User user);

    void deleteUserById(long id);

    List<User> findAllUsers();

    void deleteAllUsers();

    boolean isUserExist(User user);

}
```

```
package com.websystique.springboot.service;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.atomic.AtomicLong;

import org.springframework.stereotype.Service;

import com.websystique.springboot.model.User;



@Service("userService")
public class UserServiceImpl implements UserService{

    private static final AtomicLong counter = new AtomicLong();

    private static List<User> users;

    static{
        users= populateDummyUsers();
    }

    public List<User> findAllUsers() {
        return users;
    }

    public User findById(long id) {
        for(User user : users){
            if(user.getId() == id){
                return user;
            }
        }
        return null;
    }
    public User findByName(String name) {
        for(User user : users){
            if(user.getName().equalsIgnoreCase(name)){
                return user;
            }
        }
        return null;
    }

    public void saveUser(User user) {
        user.setId(counter.incrementAndGet());
        users.add(user);
    }

    public void updateUser(User user) {
        int index = users.indexOf(user);
        users.set(index, user);
    }

    public void deleteUserById(long id) {

        for (Iterator<User> iterator = users.iterator(); iterator.hasNext(); ) {
            User user = iterator.next();
            if (user.getId() == id) {
                iterator.remove();
            }
        }
    }

    public boolean isUserExist(User user) {
        return findByName(user.getName())!=null;
    }

    public void deleteAllUsers(){
        users.clear();
    }

    private static List<User> populateDummyUsers(){
        List<User> users = new ArrayList<User>();
        users.add(new User(counter.incrementAndGet(),"Sam",30, 70000));
        users.add(new User(counter.incrementAndGet(),"Tom",40, 50000));
        users.add(new User(counter.incrementAndGet(),"Jerome",45, 30000));
        users.add(new User(counter.incrementAndGet(),"Silvia",50, 40000));
        return users;
    }

}
```

```
package com.websystique.springboot.model;

public class User {
    private long id;

    private String name;

    private int age;

    private double salary;

    public User(){
        id=0;
    }

    public User(long id, String name, int age, double salary){
        this.id = id;
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + (int) (id ^ (id >>> 32));
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        User other = (User) obj;
        if (id != other.id)
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + name + ", age=" + age
                + ", salary=" + salary + "]";
    }


}
```

Conclusion

Rest API with Spring Boot is no-different than with Spring MVC, only the underlying application differs. Spring boot quietly simplifies it, providing all the sugar required, while still not getting in your way, reducing the development time by many-fold, certainly worth giving a try. Make sure to check our other posts on Spring Boot, we will be covering lots of concepts here. Feel free to write your thoughts in comment section.

Download Source Code

Download Now!

References

- Spring Boot
- Spring framework
- YAML Documentation

websystiqueadmin

If you like tutorials on this site, why not take a step further and connect me on Facebook , Google Plus & Twitter as well? I would love to hear your thoughts on these articles, it will help me improve further our learning process.

If you appreciate the effort I have put in this learning site, help me improve the visibility of this site towards global audience by sharing and linking this site from within and beyond your network. You & your friends can always link my site from your site on www.websystique.com, and share the learning.

After all, we are here to learn together, aren't we?

Related Posts:

1. **Secure Spring REST API using Basic Authentication**
2. **Secure Spring REST API using OAuth2**
3. **Spring MVC 4 RESTFul Web Services CRUD Example+RestTemplate**
4. **Spring Boot + AngularJS + Spring Data + JPA CRUD App Example**

spring-boot. permalink.

← Spring Boot WAR deployment example

Spring Boot + AngularJS + Spring Data + JPA CRUD App Example →