WebSystique
learn together

Spring MVC 4 RESTFul Web Services CRUD Example+RestTemplate

August 11, 2015 websystiqueadmin

In this post we will write a CRUD Restful WebService using Spring MVC 4, and write a REST client with RestTemplate to consume those services. We will also test those services using external clients.

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.

- 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

Recent Posts

Spring 4 Caching Annotations Tutorial
Spring 4 Cache Tutorial with EhCache
Spring 4 MVC+JPA2+Hibernate Many-to-many-Example
Spring 4 Email using Velocity,Freemarker Template library
Spring 4 Email With Attachment Tutorial

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.

Other interesting posts you may like

- Spring 4 MVC+JPA2+Hibernate Many-to-many Example
- Spring 4 Caching Annotations Tutorial
- Spring 4 Cache Tutorial with EhCache
- Spring 4 Email Template Library Example
- Spring 4 Email With Attachment Tutorial
- Spring 4 Email Integration Tutorial
- Spring MVC 4+JMS+ActiveMQ Integration Example
- Spring 4+JMS+ActiveMQ @JmsLister @EnableJms Example
- Spring 4+JMS+ActiveMQ Integration Example
- Spring MVC 4+Apache Tiles 3 Integration Example
- Spring MVC 4+Spring Security 4 + Hibernate Integration Example
- Spring MVC 4+AngularJS Example
- Spring MVC 4+AngularJS Routing with ngRoute Example
- Spring MVC 4+AngularJS Routing with UI-Router Example
- Spring MVC 4+Hibernate 4 Many-to-many JSP Example
- Spring MVC 4+Hibernate 4+MySQL+Maven integration example using annotations
- Spring MVC4 FileUpload-Download Hibernate+MySQL Example
- TestNG Mockito Integration Example Stubbing Void Methods
- Maven surefire plugin and TestNG Example
- Spring MVC 4 Form Validation and Resource Handling

Rest Based 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

```
package com.websystique.springmvc.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
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.springmvc.model.User;
import com.websystique.springmvc.service.UserService;

@RestController
public class HelloWorldRestController {

    @Autowired
    UserService userService;  //Service which will do all data retrieval/


    //-------------------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<List<User>>(HttpStatus.NO_CONTENT);
        }
        return new ResponseEntity<List<User>>(users, HttpStatus.OK);
    }


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

    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET, pro
    public ResponseEntity<User> getUser(@PathVariable("id") long id) {
        System.out.println("Fetching User with id " + id);
        User user = userService.findById(id);
        if (user == null) {
            System.out.println("User with id " + id + " not found");
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
```

```
        }
        return new ResponseEntity<User>(user, HttpStatus.OK);
    }



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

    @RequestMapping(value = "/user/", method = RequestMethod.POST)
    public ResponseEntity<Void> createUser(@RequestBody User user,    Uri
        System.out.println("Creating User " + user.getName());

        if (userService.isUserExist(user)) {
            System.out.println("A User with name " + user.getName() + " a
            return new ResponseEntity<Void>(HttpStatus.CONFLICT);
        }

        userService.saveUser(user);

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


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

    @RequestMapping(value = "/user/{id}", method = RequestMethod.PUT)
    public ResponseEntity<User> updateUser(@PathVariable("id") long id, @
        System.out.println("Updating User " + id);

        User currentUser = userService.findById(id);

        if (currentUser==null) {
            System.out.println("User with id " + id + " not found");
            return new ResponseEntity<User>(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<User> deleteUser(@PathVariable("id") long id) {
        System.out.println("Fetching & Deleting User with id " + id);

        User user = userService.findById(id);
        if (user == null) {
            System.out.println("Unable to delete. User with id " + id + "
            return new ResponseEntity<User>(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() {
        System.out.println("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 4. Additionally, spring provides several support classes to help you implement something customized.

MediaType : 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.

Deploy and Test this API, let’s dig deeper into how this thing works

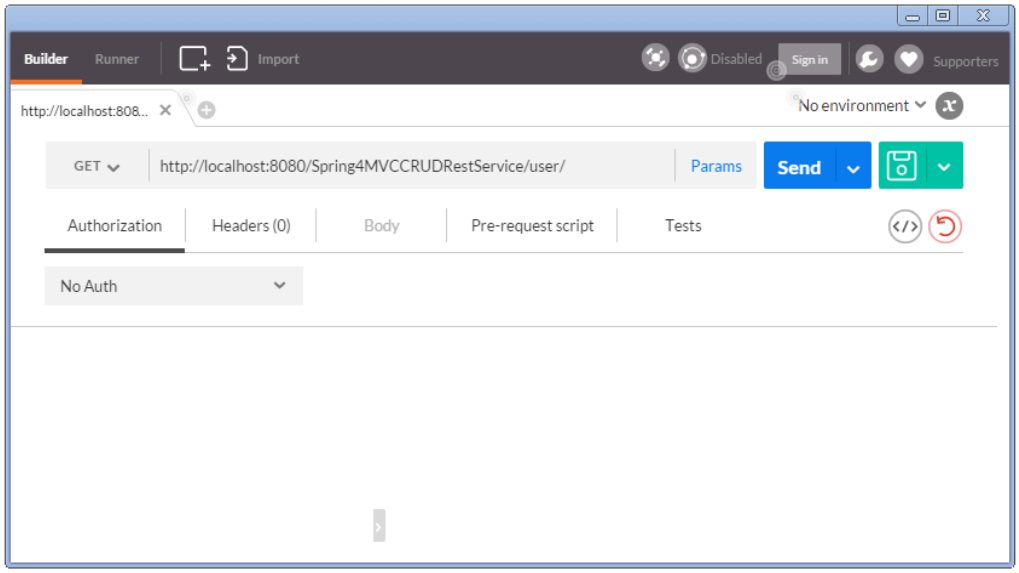
At the at end of day, it’s just a plain controller class, part of a deploy-able

application.[Complete downloadable application code is shown further down in post which you can deploy straight-away in your container]. I am going to deploy it, in order to see things live and discuss each operation in detail. Deployed Application is available at http://localhost:8080/Spring4MVCCRUDRestService.

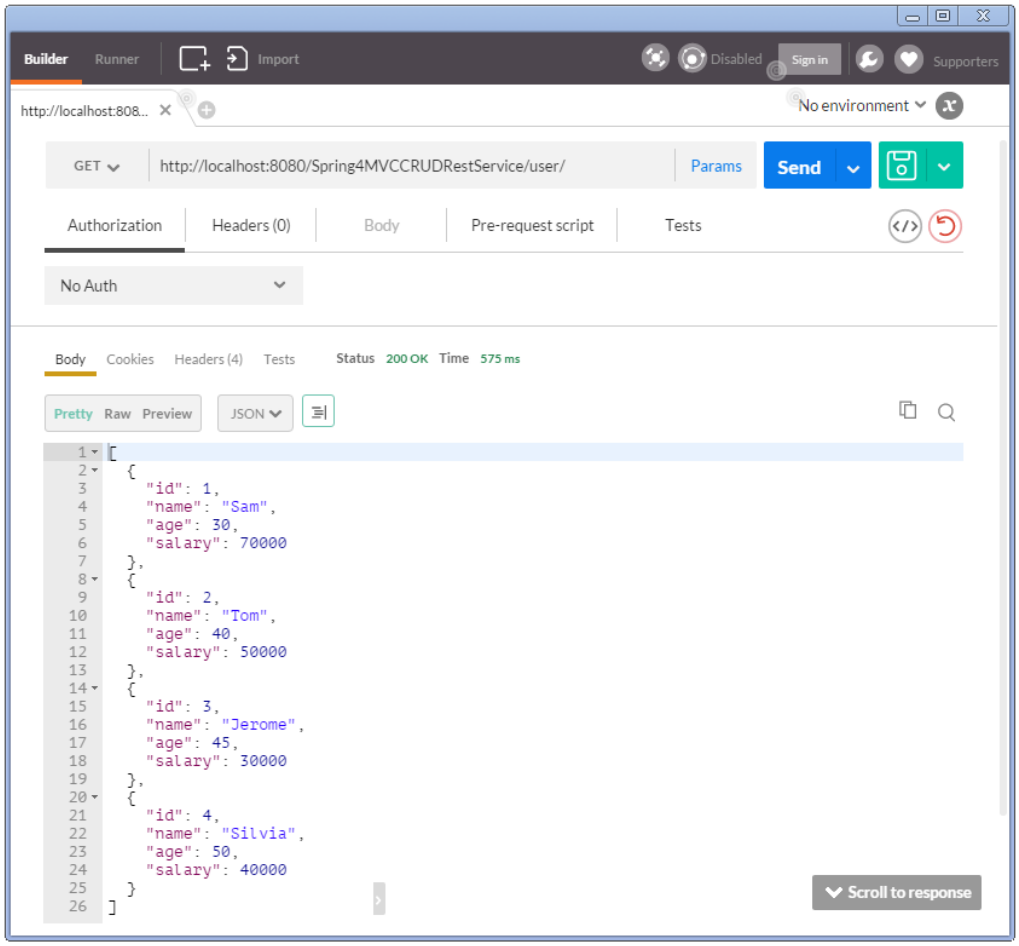
To test this API, i will use an external client POSTMAN (An extension from CHROME). We will write our own client in just few minutes.

1. Retrieve all users

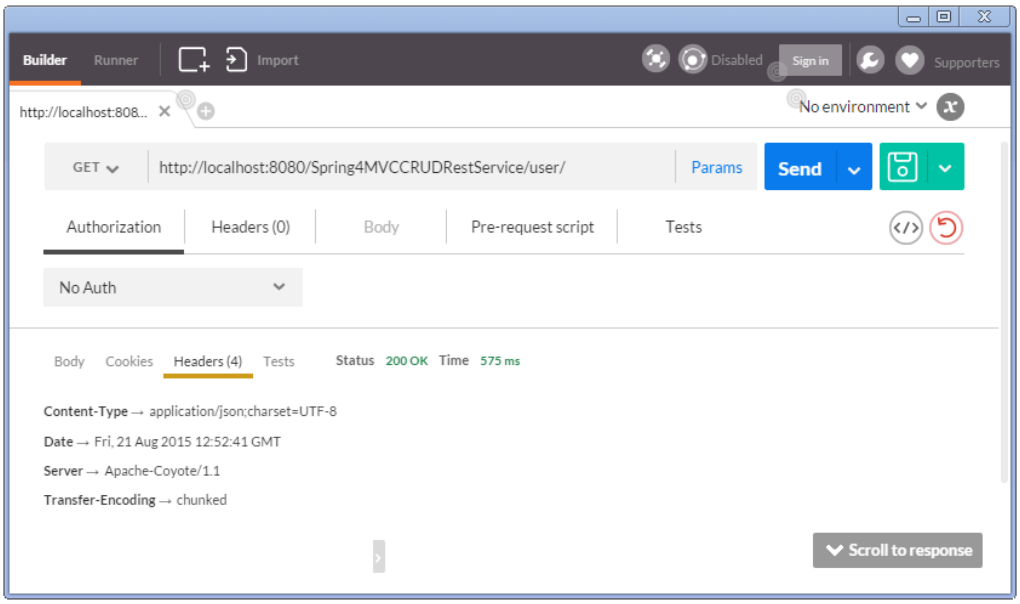
Open POSTMAN tool, select request type [GET for this usecase], specify the operation uri.



Notice that we did not specify any HTTP header here. Click on Send, you will receive list of all users.



Also notice the HTTP 200 response. Additionally check headers.



You might be wondering how the response is sent as JSON string, and the Content-Type header in response confirms that. Glad you asked. This is due to the fact that we have included Jackson library in our project.

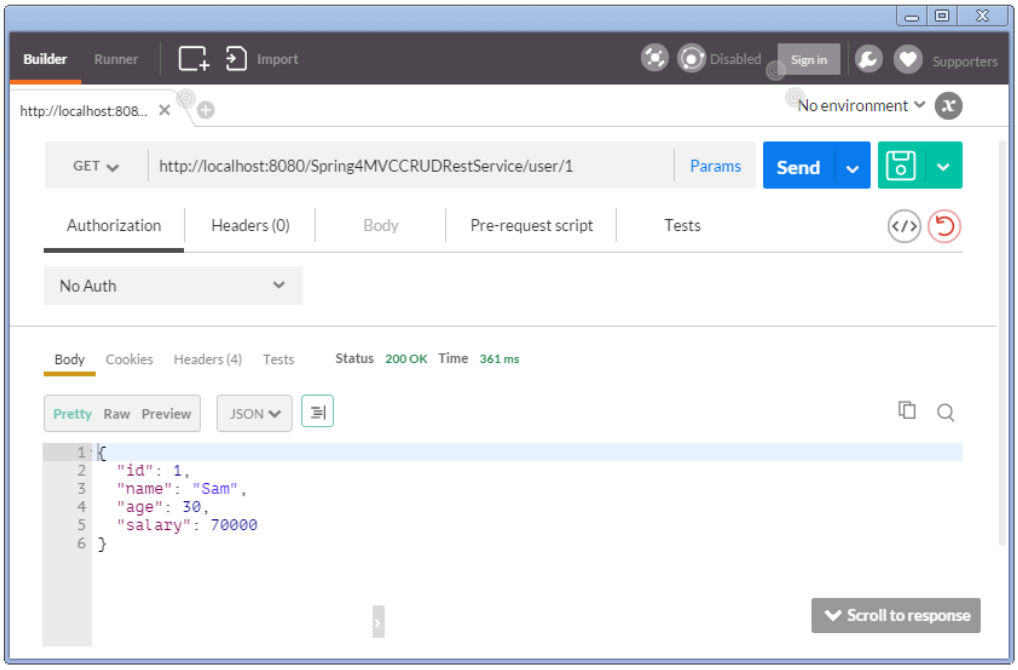
```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.5.3</version>
</dependency>
```

Since spring finds this library in class path, it invokes inbuilt MappingJackson2HttpMessageConverter converter to convert the response (List of objects) into JSON.

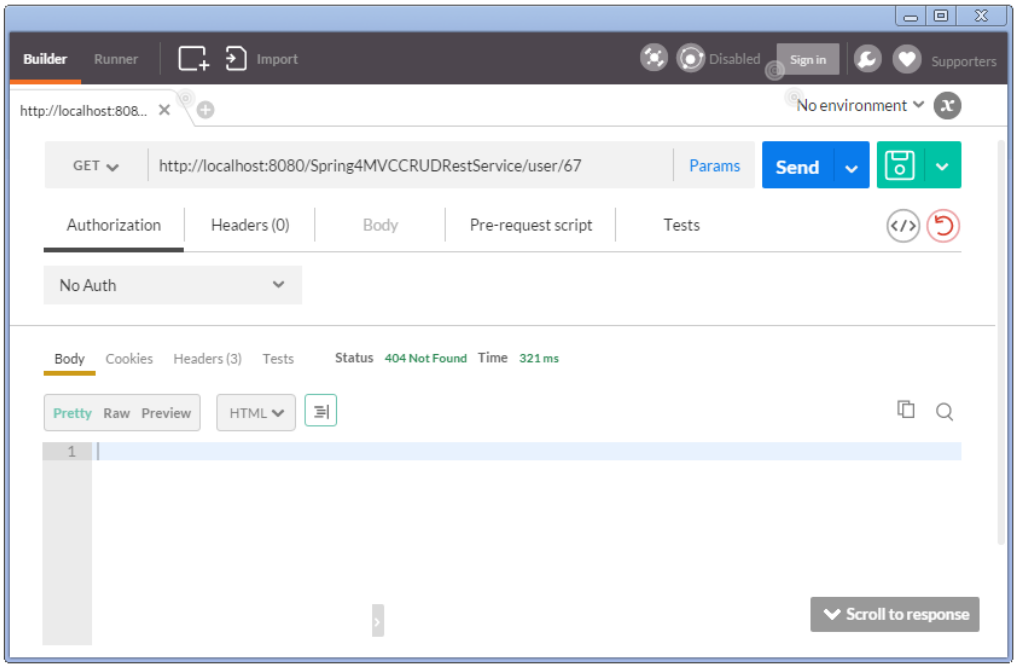
Good thing about Spring inbuilt converters are that most of the time they just need certain library in classpath in order to perform conversion. Of course sometime we do need to adapt our API/application as well. For instance, if we want to serve XML as well, we should annotate User class with proper JAXB annoations.

2. Retrieve Single User

Specify a GET with /user/1 , click on send.

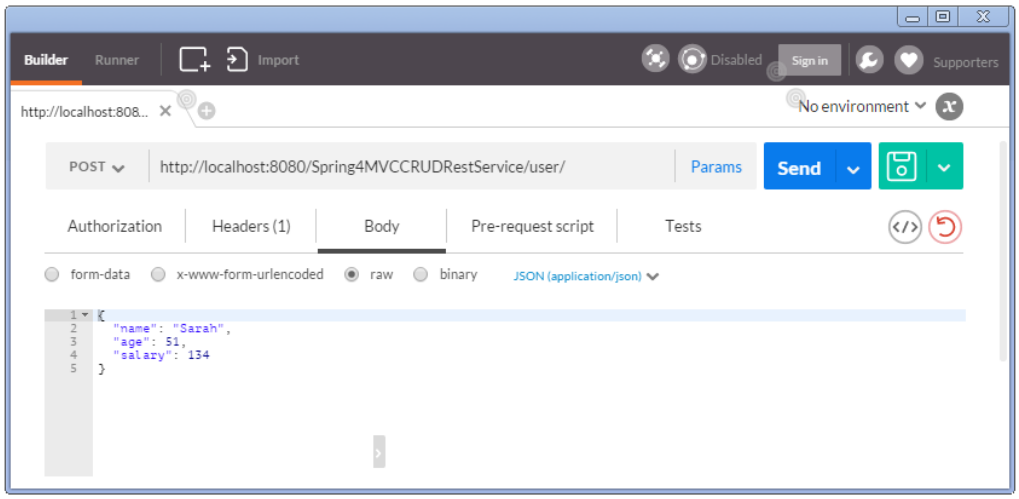


Now try to send a GET with invalid identifier, you should receive a HTTP 404.

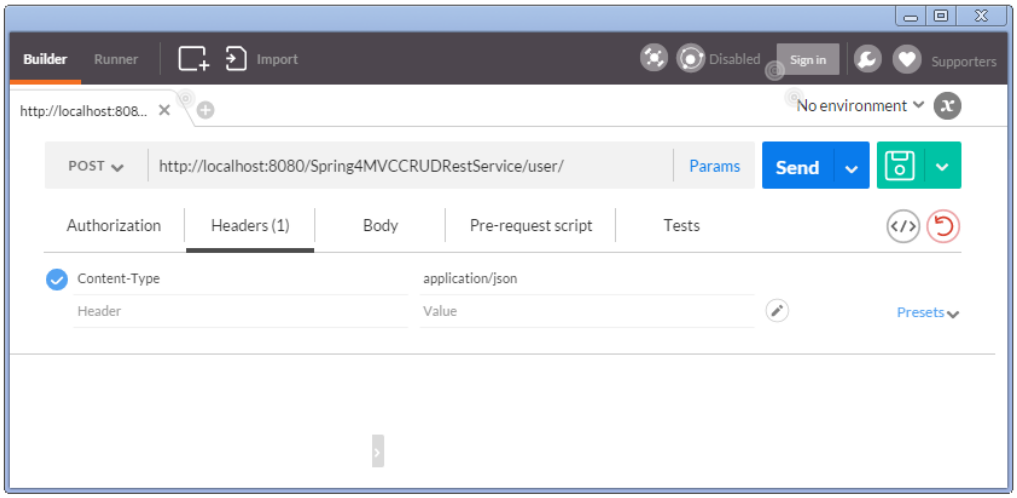


3. Create a User

Select the method as POST, specify uri as /user/, specify body in POSTMAN body tab, select the type [application/json].



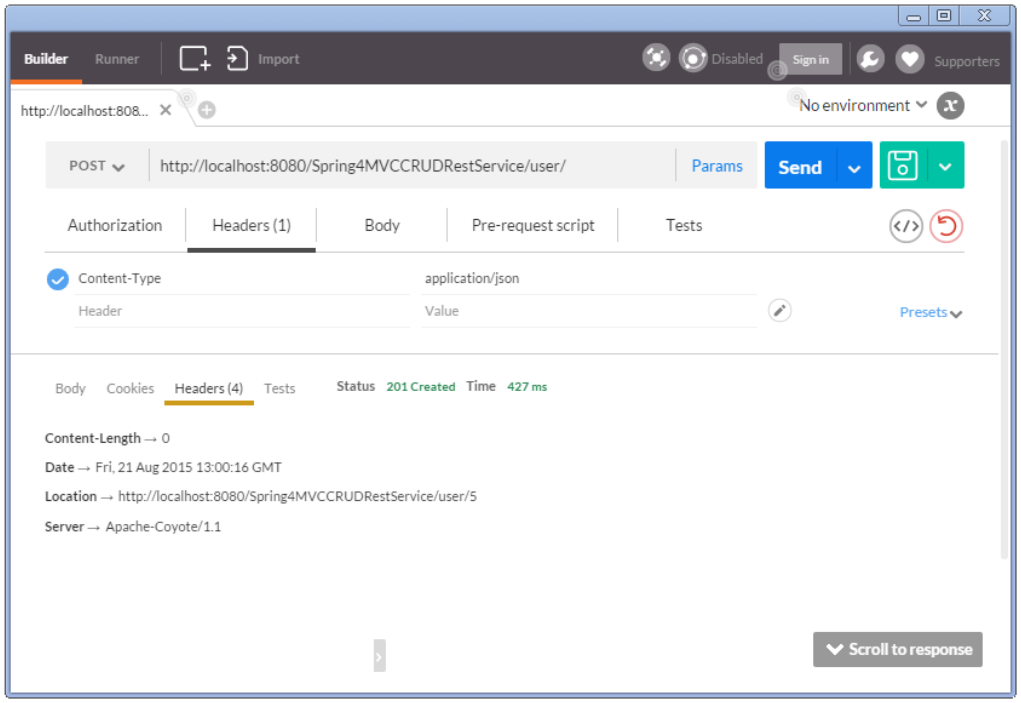
You might have noticed that POSTMAN automatically adds a header **Content-Type**.



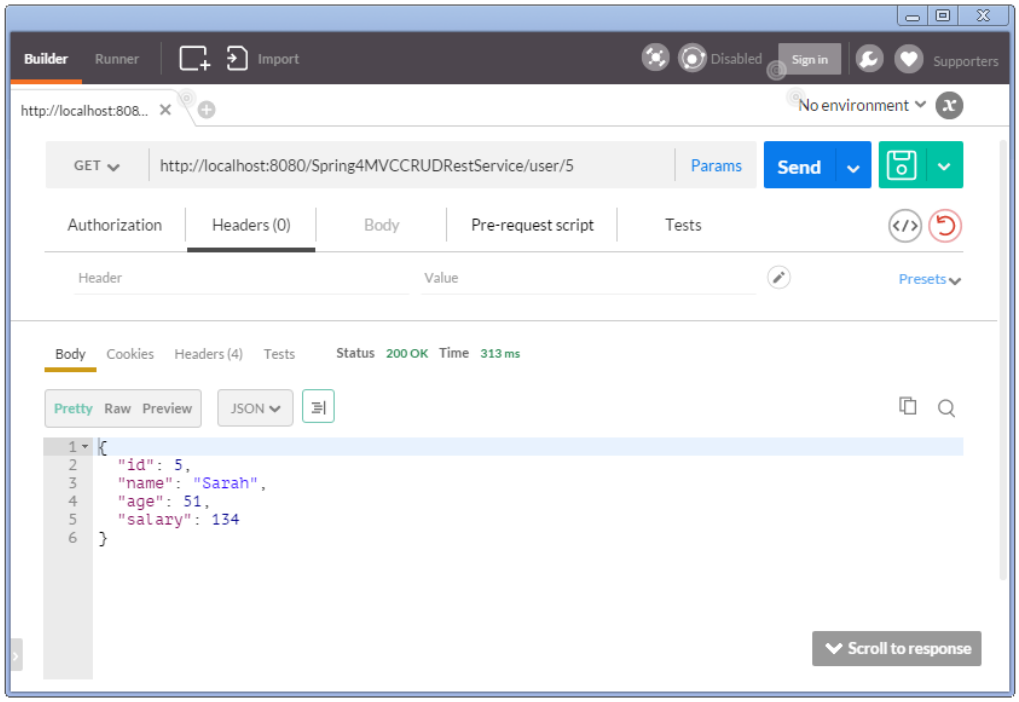
Along with POST and PUT request, clients send the data to the server and they should specify the actual content type of the data being sent.

Remember : Accept header says about what type client can understand. Content-Type header says what type of data actually is of.

Send. You should see HTTP 200 response with no body (as API don't send anything in body). But you should find a **Location header** specifying the location the newly created user can be found at.

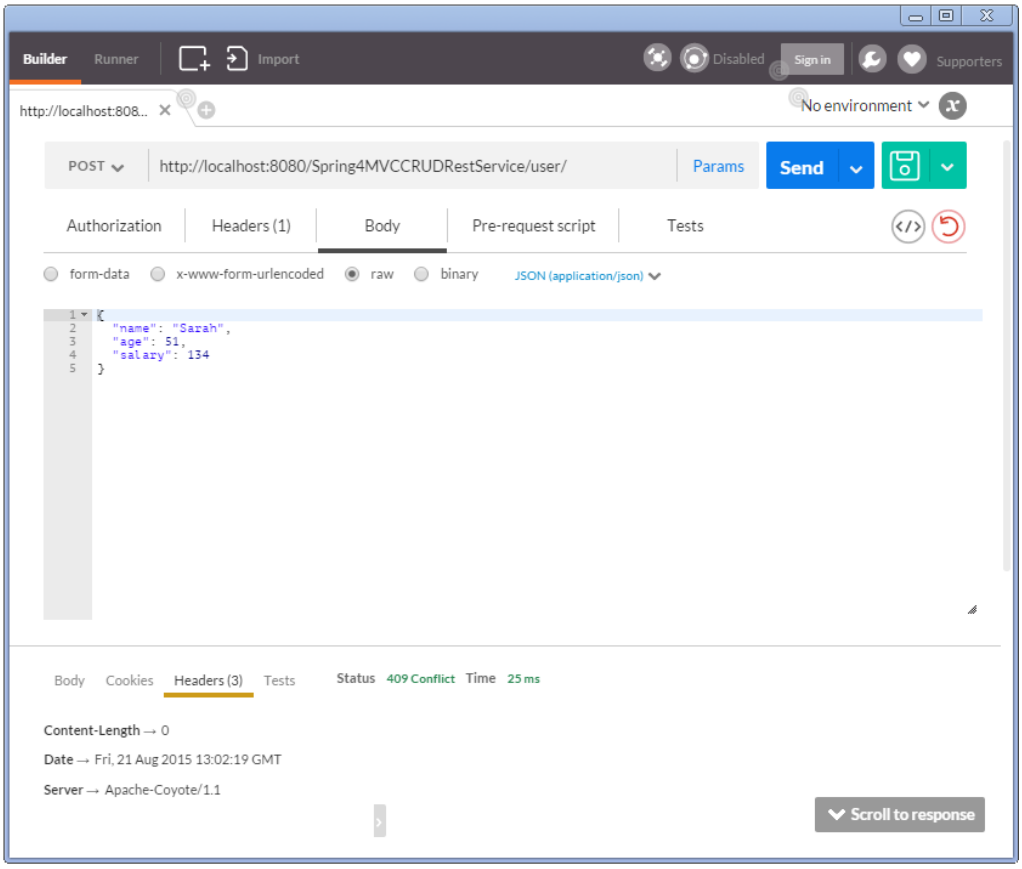


You can now fetch the newly created user.



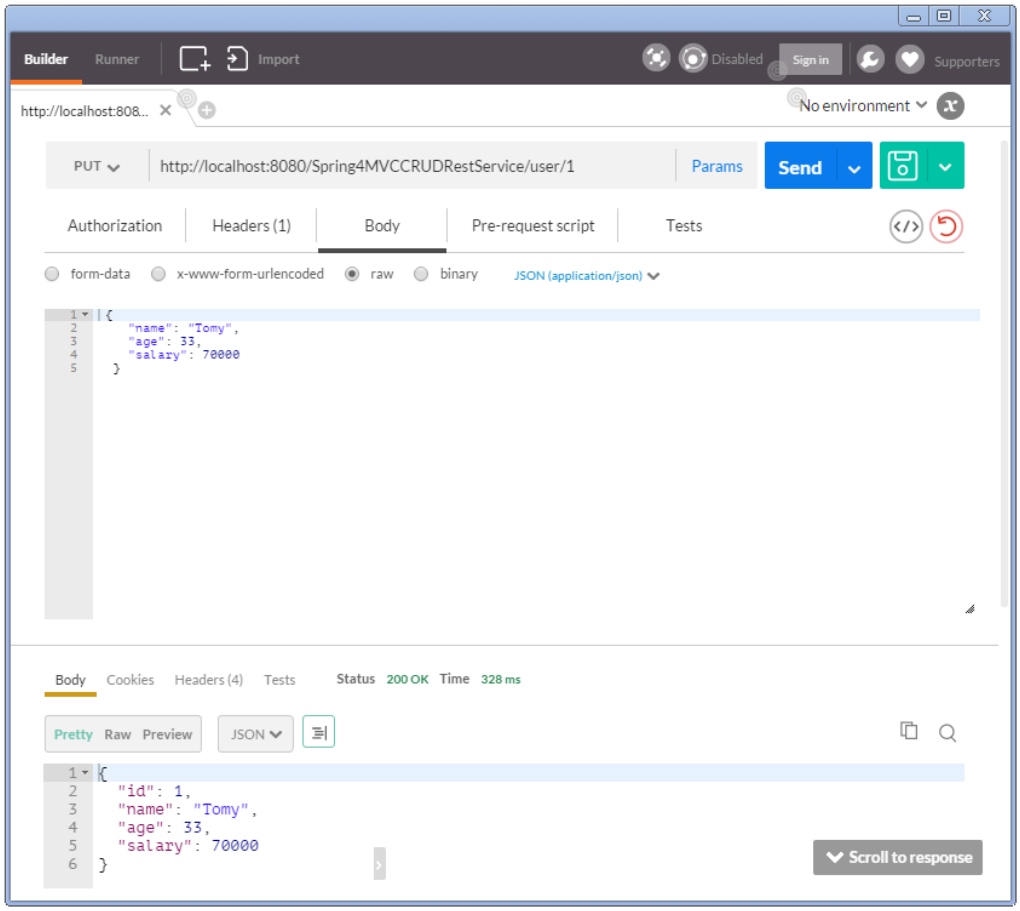
This way of implementation is common in REST. But no one stops you if you do want to send the content in Response body of a POST/PUT request. Will that still be REST complaint API? It's a debatable point.

Anyway, Lets try to create the same user again.You should get HTTP Conflict response.



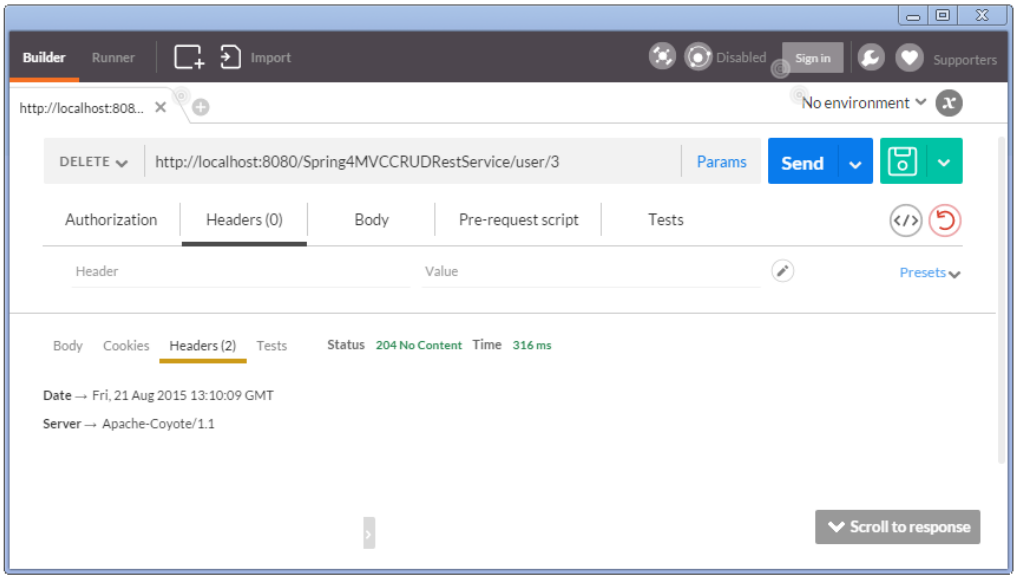
4. Update a User

Send a HTTP PUT request to update a user. Send along the new user details to be put in.

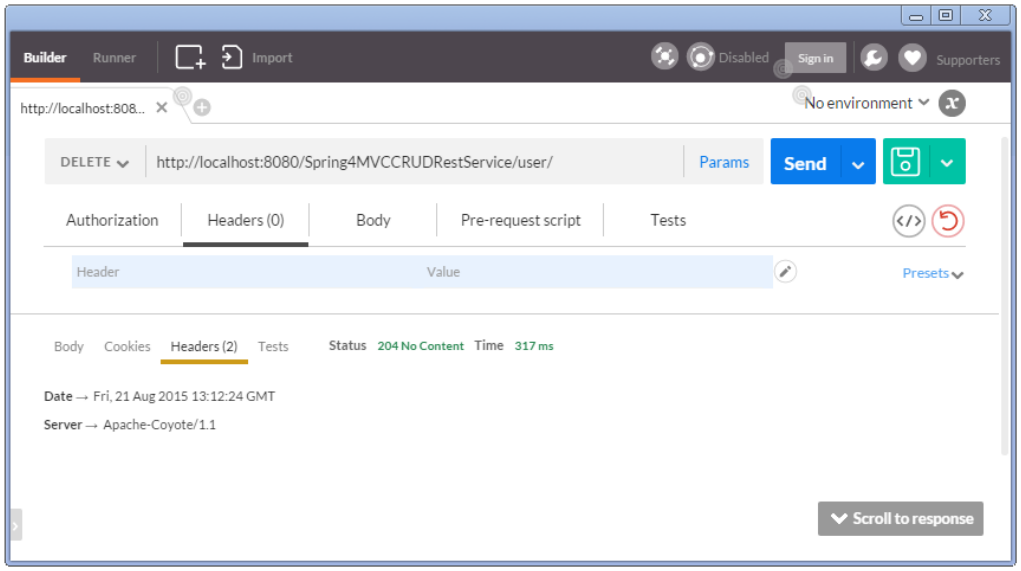


Notice that we have received response body this time. This is because the method implementation in controller is sending it. Again, one may decide not to send the updated details in response body, and just send the location header(as in create).

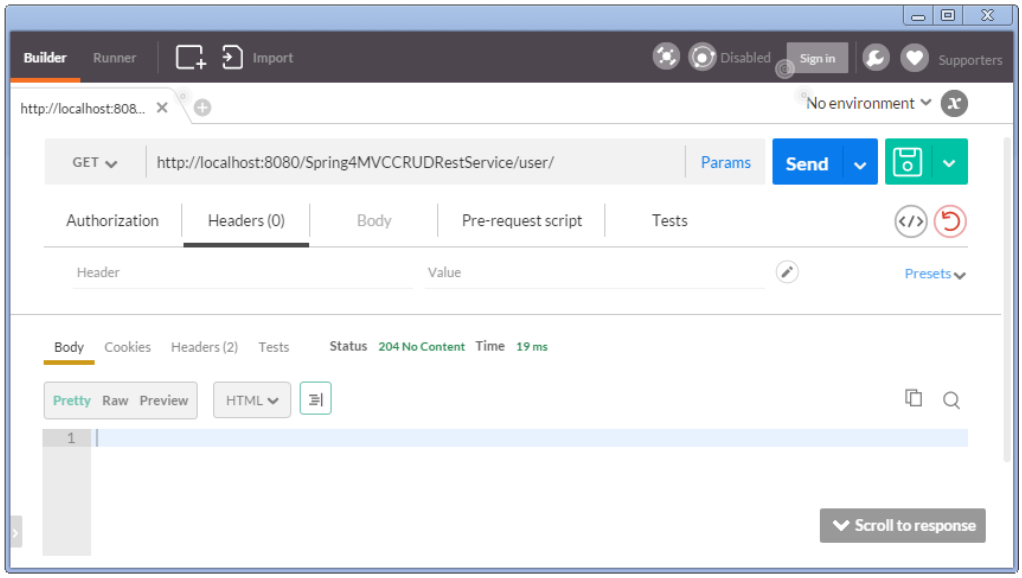
5. Delete a User



6. Delete All Users



7. Verify users after delete-all



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.

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.springmvc;

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

import org.springframework.web.client.RestTemplate;

import com.websystique.springmvc.model.User;

public class SpringRestTestClient {

    public static final String REST_SERVICE_URI = "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.getFo

        if(usersMap!=null){
            for(LinkedHashMap<String, Object> map : usersMap){
                System.out.println("User : id="+map.get("id")+", Name="+m
            }
        }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",
        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/",
        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();
    }
}
```

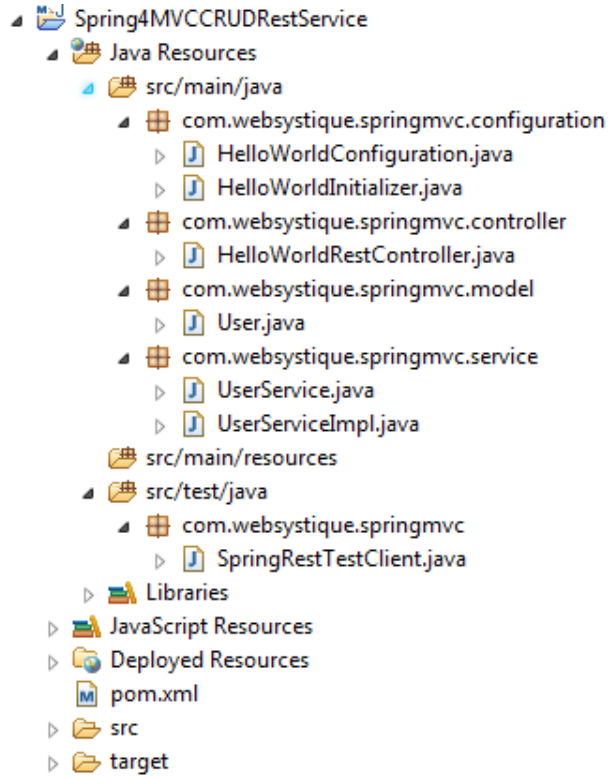
Restart server(In our example, data on server side is fixed.). Run above program.

Output from above Client program

```
Testing listAllUsers API-----------
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----------
User [id=1, name=Sam, age=30, salary=70000.0]
Testing create User API----------
Location : http://localhost:8080/Spring4MVCCRUDRestService/user/5
Testing listAllUsers API-----------
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----------
User [id=1, name=Tomy, age=33, salary=70000.0]
Testing listAllUsers API-----------
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----------
Testing listAllUsers API-----------
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----------
Testing listAllUsers API-----------
No user exist----------
```

Complete Example

Project Structure



Declare project dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apac
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.websystique.springmvc</groupId>
  <artifactId>Spring4MVCCRUDRestService</artifactId>
  <packaging>war</packaging>
  <version>1.0.0</version>
  <name>Spring4MVCCRUDRestService Maven Webapp</name>

    <properties>
        <springframework.version>4.2.0.RELEASE</springframework.version>
        <jackson.version>2.5.3</jackson.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${springframework.version}</version>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.version}</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
```

```
            <version>3.1.0</version>
        </dependency>

    </dependencies>


    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.2</version>
                    <configuration>
                        <source>1.7</source>
                        <target>1.7</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>2.4</version>
                    <configuration>
                        <warSourceDirectory>src/main/webapp</warSourceDir
                        <warName>Spring4MVCCRUDRestService</warName>
                        <failOnMissingWebXml>false</failOnMissingWebXml>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>

        <finalName>Spring4MVCCRUDRestService</finalName>
    </build>
</project>
```

User Service

```
package com.websystique.springmvc.service;

import java.util.List;

import com.websystique.springmvc.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();

    public boolean isUserExist(User user);

}
```

```
package com.websystique.springmvc.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 org.springframework.transaction.annotation.Transactional;

import com.websystique.springmvc.model.User;

@Service("userService")
@Transactional
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;
    }

    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;
    }

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

}
```

Model class

```
package com.websystique.springmvc.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 + "]";
    }


}
```

Configuration class

```
package com.websystique.springmvc.configuration;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.websystique.springmvc")
public class HelloWorldConfiguration {


}
```

Initialization Class

```
package com.websystique.springmvc.configuration;
```

```
import org.springframework.web.servlet.support.AbstractAnnotationConfigDi

public class HelloWorldInitializer extends AbstractAnnotationConfigDispat

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { HelloWorldConfiguration.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

Adding CORS support to your REST API

While accessing the REST API, you might face issues concerning Same Origin Policy.

Errors like :
" No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://127.0.0.1:8080' is therefore not allowed access." OR
" XMLHttpRequest cannot load http://abc.com/bla. Origin http://localhost:12345 is not allowed by Access-Control-Allow-Origin." are common in such case.

Solution is Cross-Origin Resource Sharing. Basically, on server side, we can return additional CORS access control headers with response, which will eventually allow further inter-domain communication.

With Spring, we can write a simple filter which adds those CORS specific headers in each response.

```
package com.websystique.springmvc.configuration;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;


public class CORSFilter implements Filter {

    public void doFilter(ServletRequest req, ServletResponse res, FilterC
        System.out.println("Filtering on.....................................
        HttpServletResponse response = (HttpServletResponse) res;
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET, PU
        response.setHeader("Access-Control-Max-Age", "3600");
```

```
            response.setHeader("Access-Control-Allow-Headers", "x-requested-w
            chain.doFilter(req, res);
        }

        public void init(FilterConfig filterConfig) {}

        public void destroy() {}

    }
```

Then we can simply configure it in our Spring configuration like shown below:

```
    package com.websystique.springmvc.configuration;

    import javax.servlet.Filter;

    import org.springframework.web.servlet.support.AbstractAnnotationConfigDi

    public class HelloWorldInitializer extends AbstractAnnotationConfigDispat

        @Override
        protected Class<?>[] getRootConfigClasses() {
            return new Class[] { HelloWorldConfiguration.class };
        }

        @Override
        protected Class<?>[] getServletConfigClasses() {
            return null;
        }

        @Override
        protected String[] getServletMappings() {
            return new String[] { "/" };
        }

        @Override
        protected Filter[] getServletFilters() {
            Filter [] singleton = { new CORSFilter()};
            return singleton;
        }

    }
```

That's it. With these two additional steps, clients will be able to communicate with your REST API without worrying about Cross domain issues.

Download Source Code

Download Now!

With CORS support:

Download Now!

References

- Spring framework

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. **Spring 4 MVC+AngularJS CRUD Example using $http service**
2. **Spring 4 MVC+AngularJS CRUD Application using ngResource**
3. **Spring MVC @RequestBody @ResponseBody Example**
4. **Angularjs Server Communication using ngResource-CRUD Application**

springmvc. permalink.

← Spring MVC @RequestBody @ResponseBody Example

Spring MVC 4 File Upload Example using Commons fileupload →

Found when accessing

http://localhost:7001/Spring4MVCCRUDRestService/user/

I using your code as-is. I'm using Weblogic if that matters btw..

Any idea what I may doing wrong? Or how I can debug this?

1 • Reply • Share ›

websystique Mod ➚ nmpg • 9 months ago

Hi,

Are you deploying on Weblogic externally or using it from within Eclipse? You might want to go through Setting Tomcat with Eclipse. Although the post is for setting up Tomcat, similar setup should be applicable for other servers/containers. Let me know if the outcome.

• Reply • Share ›

nmpg ➚ websystique • 9 months ago

I'm using Netbeans, and deploying to external Weblogic server (but through Netbeans IDE..). It works just fine with others projects (some from here actually), but for some reason does not work with this one.

By the way, I just tried http://websystique.com/springm... and whenever accessing http://localhost:7001/Spring4MVCAngularJSExample/ I get Error 403 --Forbidden..

I'm not even sure how I can debug this.. Any thoughts on what I may doing wrong?

• Reply • Share ›

Gil Shapir • 4 days ago

GREAT ARTICLE - learned a lot from the trial & error - before it worked... Some useful tips, in a case someone is desperate as I was:

1) for the 404 really follow EVERY single step for Tomcat. My issue was that I neglected the Deployment Assembly Maven Dependencies thing (as mentioned here:
http://websystique.com/misc/ho...

2) In a case pom.xml reports error die lack of web.xml, using this link solved the issue:
http://stackoverflow.com/quest...,
basically modify the pom with:
<plugin>
<groupid>org.apache.maven.plugins</groupid>
<artifactid>maven-war-plugin</artifactid>
<version>2.6</version>
<configuration>

<failonmissingwebxml>false</failonmissingwebxml>
</configuration>
</plugin>

Reply • Share ›

Srinivasa Praneeth • 21 days ago

Hi,

Thank you very much... it is an Awesome tutorial it helped me a lot.

Reply • Share ›

Chriso Lopez C • 24 days ago

Hi, I downloaded your project and trying to run it from Eclipse on Glassfish, I keep getting a 404 Not found.

Reply • Share ›

websystique Mod ➚ Chriso Lopez C • 22 days ago

Hi Chriso, It could be related to your environment setup. Have a look at Setup tomcat+Eclipse. Although that post is for tomcat, you can follow the similar procedure to setup your server with Eclipse.

Reply • Share ›

SG • a month ago

Hi, thanks for this tutorial. This is really wonderful. I have a question more to do with the design part of the components. For the registration page, I mean the actual html/jsp client we need to have a non-rest controller ? I don't want to call the JSP/HTML and would like to go through Springs MVC? what will be the best way to handle this.

Reply • Share ›

websystique Mod ➚ SG • a month ago

Hi SG, you can implement everything in REST way. Even in this example post, the createUser method is written in a REST way, returning ResponseEntity with actual HTTP status code. We should implement our logic in a consistent way. Let me know if i misunderstood your question.

Reply • Share ›

SG ➚ websystique • 25 days ago

Sorry If my question was not clear. I am talking about the initial view page to input data for the CreateUser method. Say for example user.jsp is the initial page, which is holding form input element to enter user information which in turns call the CreateUser REST api. In below code I am using a normal Spring Controller instead of REST. I believe below approach is a correct.

```
@Controller // Using a non REST Controller.
@RequestMapping("/registerUser")
public class HomeController {
```

```
@RequestMapping(method=RequestMethod.GET)
public String HomePage(Model model){
return "user";
}
}
```

Reply • Share ›

Hzmr • a month ago

hi, may i know what is the difference between using spring with restful web services and using spring without it?

Reply • Share ›

websystique Mod ➚ Hzmr • a month ago

Hi hzms,
Spring provides different ways to implement applications. In case your application is build with REST architecture in mind, you take advantage of Spring REST support to implement your application. If not, you go for traditional Spring MVC controllers, returning views [instead of data & HTTP status codes].

Reply • Share ›

ashish • a month ago

hi very nice tutorial.. but so many new things... How can I integrate this module with mybatis..?

Reply • Share ›

websystique Mod ➚ ashish • a month ago

Hi Ashish, i don't have knowledge of mybatis yet, sorry, probably something for future.

Reply • Share ›

Maun Bisma • a month ago

Hi, nice tutorial,,,
I already try your tutorial http://websystique.com/springm... when I add the restfull controller like this tutorial, i get error 405 about POST request not support,,,, can you give me advice what wrong?

Thank you,,,

Reply • Share ›

websystique Mod ➚ Maun Bisma • a month ago

Hi Maun, in that post, we are using Spring Security with CSRF enabled. Make sure that in your page, the expressions are evaluated [set <%@ page isELIgnored="false" %> in JSP].

Reply • Share ›

sonia rao • 2 months ago

Hi,I have tried this example ...its asking for Username And password in postman...can anyone help with this??

Reply • Share ›

websystique Mod → sonia rao • a month ago

Hi Sonia, this example does not required any authentication anywhere. Could it be that you are mixing it with requests for other posts [Spring Security]?

Reply • Share ›

Daniel • 2 months ago

Hi, how does it come that I did not declare this jackson dependency and it still returns a JSON with the header Content-Type →application/json;charset=UTF-8 .... I checked all the jars in my project and it does NOT seem like some other dependency transitively brought it.. any idea? Great post by the way, thanks!

ANSWER: My bad, actually spring-webmvc does bring that dependency

Reply • Share ›

Girish Balodi • 2 months ago

Ultimate guid step by step... :)

Reply • Share ›

Hendi Santika • 2 months ago

Hi. I've been enjoying your tutorials, they're great :)

How ever I want to know the way you handle those api in form / web page. So that I can continue learning your nice tutorial more understand implementations.

Reply • Share ›

websystique Mod → Hendi Santika • a month ago

Hi Hendi, Sorry i missed it. For web pages in a Spring MVC based application, normally if we get the response from server as plain JSON, our preference [in 2016] should be using AngularJS [within jsp as shown SpringMVC+AngularJS Example or without jsp but with a template[Velocity/Freemarker] as shown Spring 4 MVC+AngularJS Routing Example using UI-Router]. But if for any reason you are stuck with plain JSP[no angularJS], you should consider using JQuery with JSP to get the response from server [using $.ajax()] and display in appropriate element [node] within your page.

Reply • Share ›

Kali Prasad Padhy • 2 months ago

Hi,
Can you tell me why we use AbstractAnnotationConfigDispatcherServletInitializer to intalize HelloWorldInitializer class.

Thanks

Reply • Share ›

websystique Mod ➚ Kali Prasad Padhy • 2 months ago

Hi,
Functionally, AbstractAnnotationConfigDispatcherServletInitializer provides a way to handle the configuration you would normally specify in traditional web.xml: register a DispatcherServlet. It provides hooks to register classes & filters with global ContextLoaderListener as well as with DispatcherServlet. More detailed description can be found at Spring Docs.

▲ | ▼ • Reply • Share ›

greo • 3 months ago

i get error 404 in netbeans and eclipse, i try server tomcat in eclipse and server tomcat and glassfish in netbeans, how to fix that :(?

▲ | ▼ • Reply • Share ›

websystique Mod ➚ greo • 3 months ago

As i am using Eclipse, i would suspect the tomcat configuration. Did you already go through each step from post Setup Tomcat with Eclipse?

▲ | ▼ • Reply • Share ›

Sof Ham • 3 months ago

Hi I'm getting "Etat HTTP 404 - /Spring4MVCCRUDRestService/user/" when calling http://localhost:8282/Spring4MVCCRUDRestService/user/

please note that tomcat is correctly installed.

▲ | ▼ • Reply • Share ›

websystique Mod ➚ Sof Ham • 3 months ago

Hi Sof, Is your tomcat deployment listening on 8282 or 8080?

▲ | ▼ • Reply • Share ›

Abhijit Marne Deshmukh • 3 months ago

thank you so much for providing this great example. it is really helpful.

▲ | ▼ • Reply • Share ›

websystique Mod ➚ Abhijit Marne Deshmukh • 3 months ago

Thanks for the comments Abhijit.

▲ | ▼ • Reply • Share ›

Binh Thanh Nguyen • 3 months ago

Thanks, nice tips.

▲ | ▼ • Reply • Share ›

sanji • 3 months ago

Hi, very nice tutorial I learned so much from it :)

When creating a user via Postman I get an HTTP 201 Created (which is obvious) instead of HTTP 200 OK like in your examples (less obvious...)

is there any explanations about that ?

Second, I'm using Spring Tool Suite (latest version) I wanted to deploy the app with spring-boot (have all the necessary dependencies in maven) so when I type the command "mvn spring-boot:run" I get this output http://pastebin.com/AAYnCCnN server returns 404 when I type http://localhost:8080/stbwebservice/user/ in the browser on the other hand when I deploy it as a war and run it on a standalone tomcat8 it works... How can I solve that ?

Thanks again.

▲ | ▼ • Reply • Share ›

Rupali Solaskar • 4 months ago

Hi. I am running same code on my laptop and also following every step of setting tomcat with eclipse but i got an error -"Server Tomcat v8.0 Server at localhost failed to start." i am not getting that because of what error comes. after every step performing when i tried to start tomcat at that time this error occurs.

please help me.

Thanks

▲ | ▼ • Reply • Share ›

Gísli Leifsson • 4 months ago

Hi there and thanks for a great series of tutorials! Been following quite a few.

For those of you who are dealing with a 404 error, the answer might be extremely simple.
In the controller class, the request mapping for listAllUsers looks like this:
@RequestMapping(value = "/user/", method = RequestMethod.GET)
Notice that it's "/user/" and not "/user". This means that if you end the URL with "/user", it gives you a 404 error. If you end it with "/user/" however, it will work. At least it behaves like this in Tomcat 8.

Simply change the request mapping to "/user" and it will work as expected.

▲ | ▼ • Reply • Share ›

Sekhar ➚ Gísli Leifsson • 2 months ago

http://localhost:8077/Spring4MVCCRUDRestService/user/ it is working fine. we need to use user/
i downloaded tomcat 8.0 and builded it using maven 3.0 and pasted the war file in the webapps folder and clicked on start.bat. And as given in the postmaster i given extra / it worked fine for me

▲ | ▼ • Reply • Share ›

Ashmin Pathak ➚ Gísli Leifsson • 3 months ago

Thanks..

▲ | ▼ • Reply • Share ›

Rupali Solaskar • 4 months ago

Hi I am using your above code but it is not executing. I am getting 404 error, I am running it in Eclipse -Luna can any one help me how to remove error.

• Reply • Share ›

websystique Mod ➚ Rupali Solaskar • 4 months ago

Hi Rupali,

Probably your local tomcat+Eclipse setup is not correct. Please have a look at Setting Tomcat with Eclipse, should be helpful.

• Reply • Share ›

Rupali Solaskar ➚ websystique • 4 months ago

its done!!!!!!
Thank you.!!!!!!!!!!!!

Yes you were correct there was problem in my setting of tomcat with Eclipse.

Great Work. its very helpful.

• Reply • Share ›

Muhammed Abdul • 4 months ago

Hello, I have also added Spring Security basic authentication as mentioned in your previous tutorials,

But i am not able to access it if i provide a header with authorization and Basic <base64encoded credentials="">

Can you help? Do i need to change the security class (i am using the same security class as in ur previous tutorials without any changes)

• Reply • Share ›

wang • 5 months ago

A good example for study spring rest

Thanks a lot!

• Reply • Share ›

lab • 5 months ago

I am using netbean, and deploying in tomcat 8. However i get error 404 - not found accessing resources. How I can solve it?

• Reply • Share ›

websystique Mod ➚ lab • 5 months ago

Hi, I don't have a netbeans setup, but the problem itself seems similar to eclipse+tomcat when the tomcat is not configured properly with your IDE. To get an idea, you may want to refer to Setting Tomcat with Eclipse, and check if your IDE missing some

specific configuration.

Reply • Share ›

Mamuka Arabuli • 5 months ago

I was unable to produce xml . simple call always returns json , I added @XmlRootElement but no result :

406 Not Acceptable

Reply • Share ›

Antonio D.A. (adoalonso) ➚ Mamuka Arabuli • 2 days ago

You should add the dependency to com.fasterxml.jackson.dataformat::jackson-dataformat-xml

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</gro
  <artifactId>jackson-dataformat-xml</artifactId
  <version>2.7.5</version>
</dependency>
```