REST API Security
Learning More about REST tags

Guadalupe Ortiz Bellot

Department of Computer Science and Engineering

UCA
Universidad de Cádiz

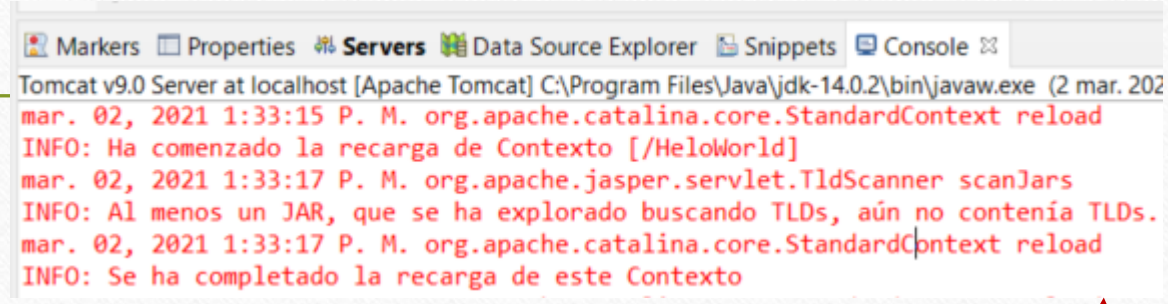
Content

1. @Path
2. @PathParam
3. @QueryParam
4. @XmlRootElement
5. @Post
6. @FormParam
7. @PUT
8. @DELETE

Content

1. **@Path**
2. @PathParam
3. @QueryParam
4. @XmlRootElement
5. @Post
6. @FormParam
7. @PUT
8. @DELETE

1. @Path (i)



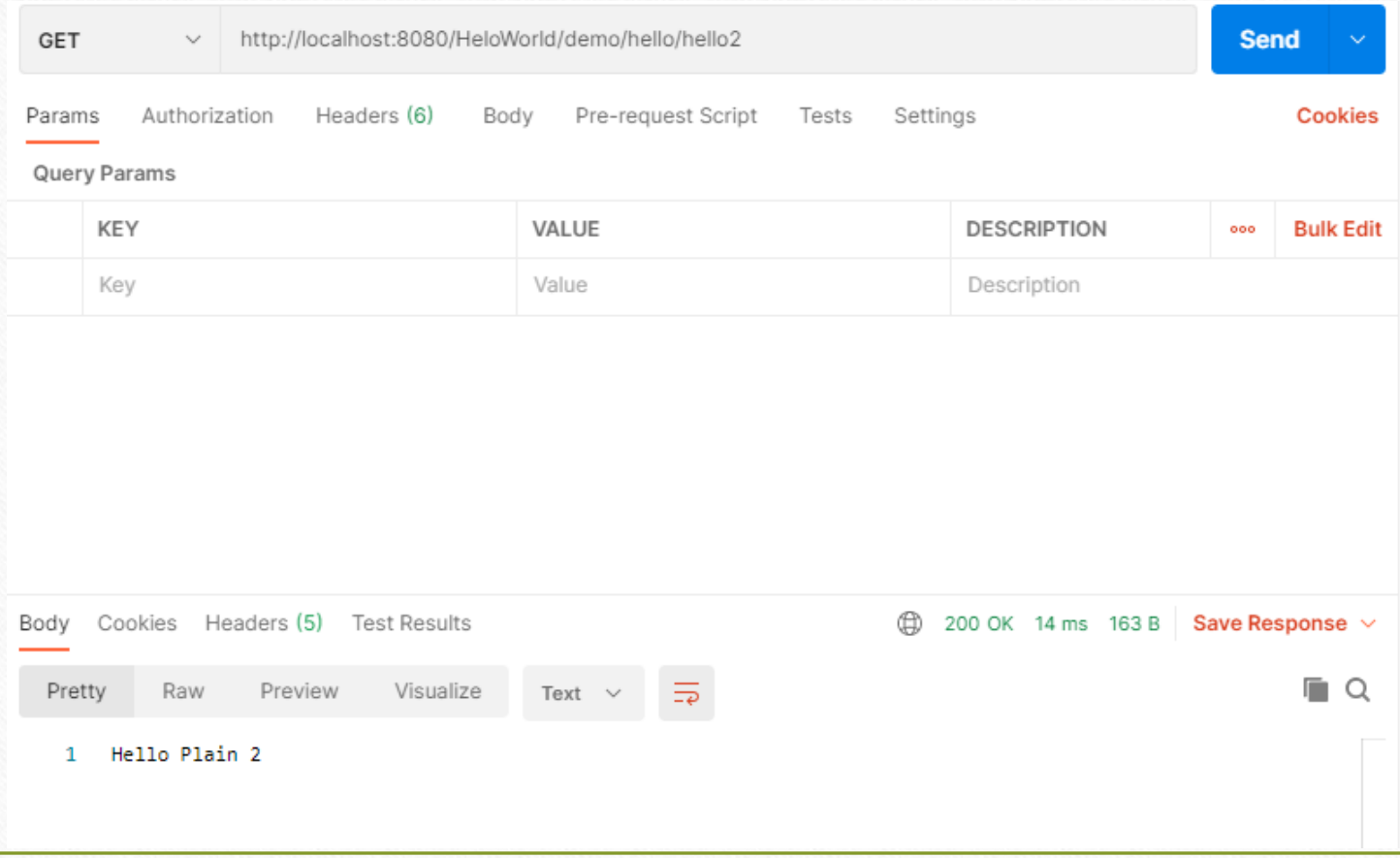
- Add the following function inside the class Hello.java:

```
@GET
@Path("/hello2")
@Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello2(){return "Hello Plain 2";}
```

- Pay attention: we have added the path to the function
- Pay attention: **do not forget to save the file before testing.**
- **Pay attention: After saving the changes, Tomcat server will reload the context to add the new function (wait until it finishes restarting). There is no need to restart, but you have to wait until the context is reloaded (see Tomcat console)**
- Now, in order to test it, we have to add the path to the root path: http://localhost:8080/HelloWorld/demo/hello/hello2
- Test it
- NOTE: the URI contains the class path followed by the method path:

1. @Path (ii)

- URI: http://localhost:8080/HelloWorld/demo/hello/hello2



Content

1. @Path
2. **@PathParam**
3. @QueryParam
4. @XmlRootElement
5. @Post
6. @FormParam
7. @PUT
8. @DELETE

2. @PathParam (i)

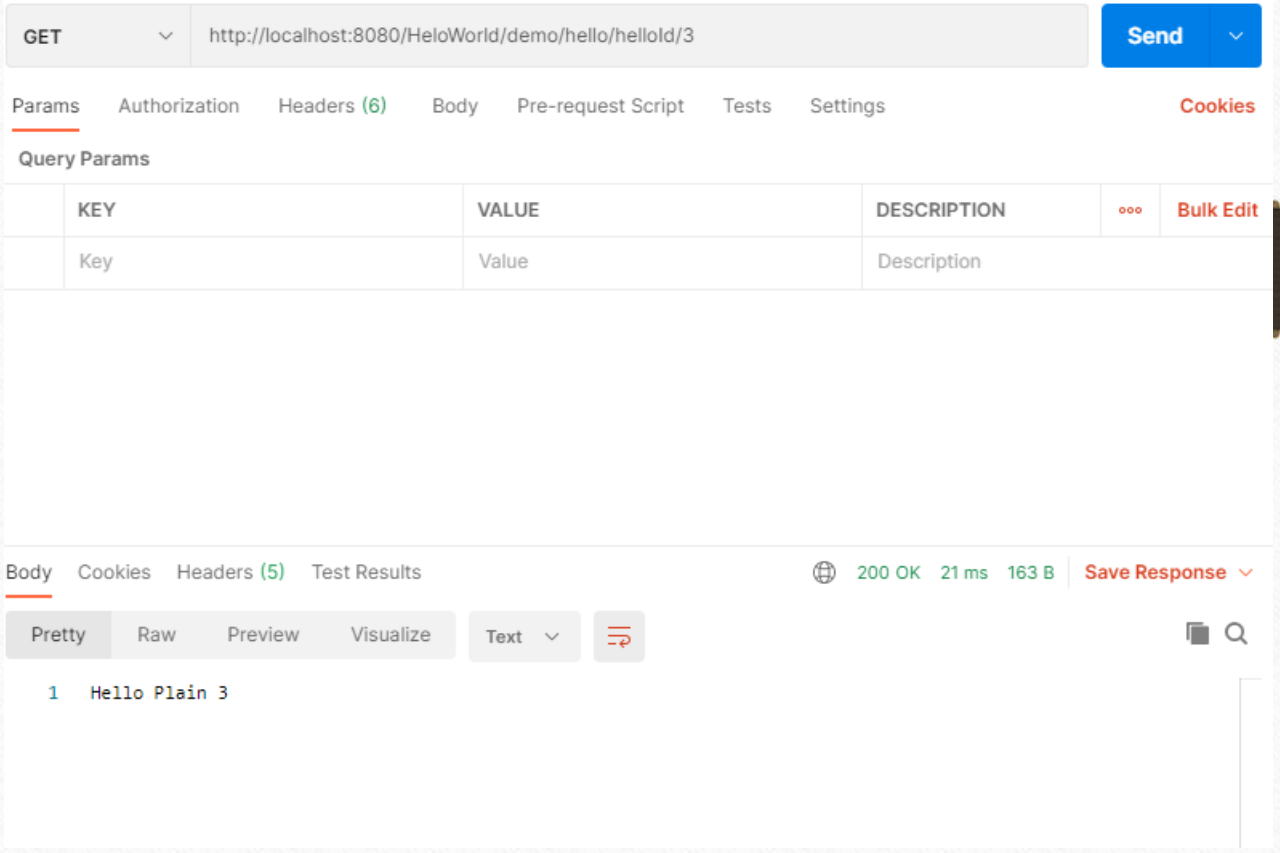
- Add the following function in the Hello.java class:

```
@GET
@Path("/helloId/{oid}")
@Produces(MediaType.TEXT_PLAIN)
  public String sayHelloWithId(@PathParam("oid")int id)
{return "Hello Plain " + id;}
```

- We have added a parameters identifier between keys to the path
- In the function parameters we have identified such parameter introducing `@PathParam("oid")` before the corresponding parameter type. *oid* is the identifier used in the path
- In order to test it we have to replace {oid} by a value, i.e: http://localhost:8080/HelloWorld/demo/hello/helloId/3
- Test it. Note: new labels require new Import (import javax.ws.rs.PathParam;). **Pay attention, always with the root javax.ws**
- If you want to add multiple parameters separate them by / : `@Path("/helloDate/{year}/{month}/{day}")`

2. @PathParam (ii)

- URI: http://localhost:8080/HelloWorld/demo/hello/helloId/3



Content

1. @Path
2. @PathParam
3. **@QueryParam**
4. @XmlRootElement
5. @Post
6. @FormParam
7. @PUT
8. @DELETE

3. @QueryParam (i)

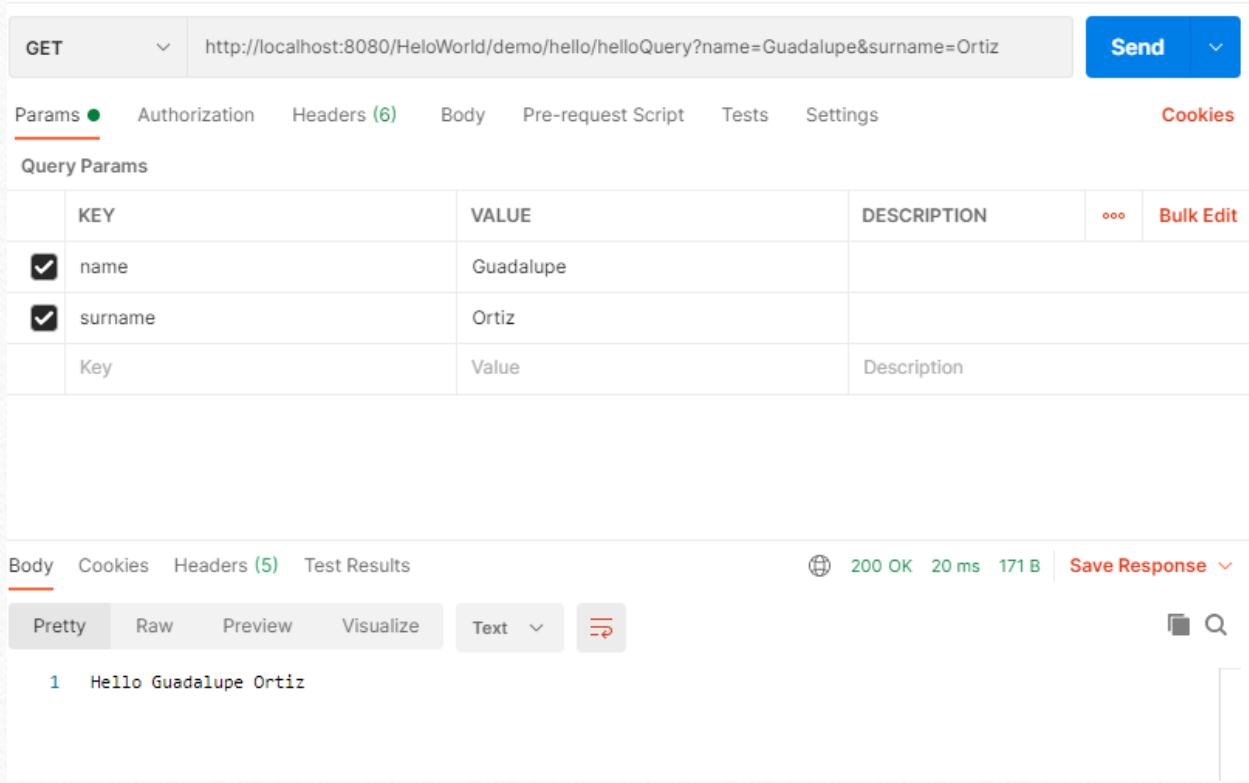
- We can use it to send directly the parameters from an HTML form by GET.
- Add the following function in the Hello.java class:

```
@GET
@Path("/helloQuery")
@Produces(MediaType.TEXT_PLAIN)
  public String sayHelloWithQuery(@QueryParam("name")String
name,@QueryParam("surname")String surname )
{return "Hello " + name + " "+ surname;}
```

- In order to test it you have to add the parameters in the path as follows :
- http://localhost:8080/HelloWorld/demo/hello/helloQuery**?name=Guadalupe&surname=Ortiz**
- Test it

3. @QueryParam (ii)

- URI: http://localhost:8080/HelloWorld/demo/hello/helloQuery?name=Guadalupe&surname=Ortiz



Content

1. @Path
2. @PathParam
3. @QueryParam
4. **@XmlRootElement**
5. @Post
6. @FormParam
7. @PUT
8. @DELETE

4. @XmlRootElement (i)

- @XmlRootElement is added to a class which will be used as return parameter. It will allow us to benefit from automatic transformations into XML or JSON type.
- **Create a new class MyDate:**

```
package packageName;

import
javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class MyDate {
  private int day;
  private int month;
  private int year;
```

Use the package name you chose

```
public int getDay() {return day;}
public void setDay(int day)
  {this.day = day;}
public int getMonth() {return month;}
public void setMonth(int month)
  {this.month = month;}
public int getYear() {return year;}
public void setYear(int year)
  {this.year = year;} }
```

4. @XmlRootElement (ii)

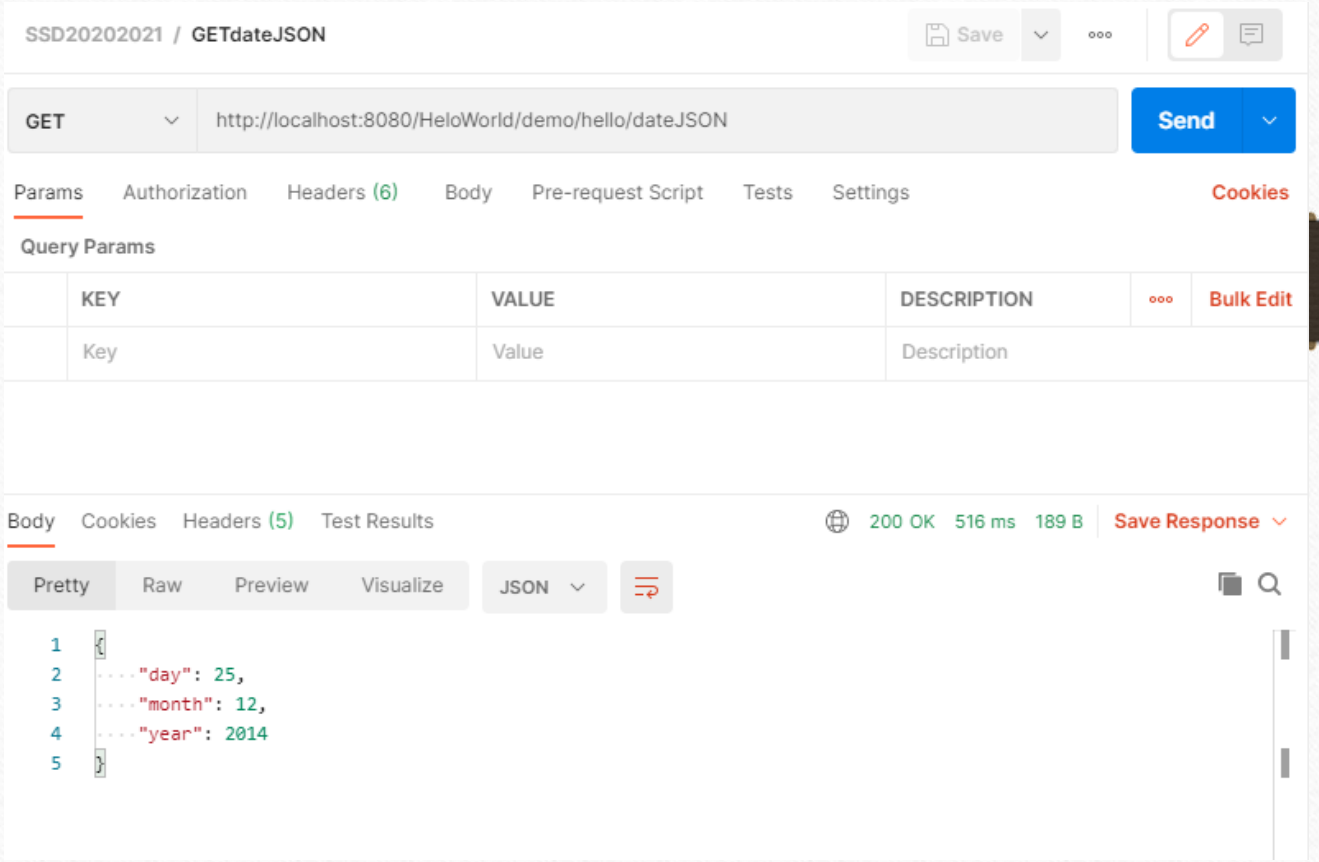
- Add a new method **to the Hello class**:

```
@GET
@Path("/dateJSON")
@Produces({"application/json"})
public MyDate getDate_JSON() {
    MyDate oneDate = new MyDate();
    oneDate.setDay(25);
    oneDate.setMonth(12);
    oneDate.setYear (2014);
    return oneDate;}
```

- Test it
- NOTE. You can use both `@Produces({"application/json"})` and `@Produces({MediaType.APPLICATION_JSON})`
- You can also do it for XML : `@Produces({"application/xml"})` or `@Produces({MediaType.APPLICATION_XML})`

4. @XmlRootElement (iii)

- URI: http://localhost:8080/HelloWorld/demo/hello/dateJSON



Content

1. @Path
2. @PathParam
3. @QueryParam
4. @XmlRootElement
5. **@Post**
6. @FormParam
7. @PUT
8. @DELETE

5. @POST (i)

- Add a new method:

```
@POST
@Path("/name")
@Consumes(MediaType.TEXT_PLAIN)
@Produces(MediaType.TEXT_PLAIN)
public String HelloName(String myName){
return "Hello "+myName;}
```

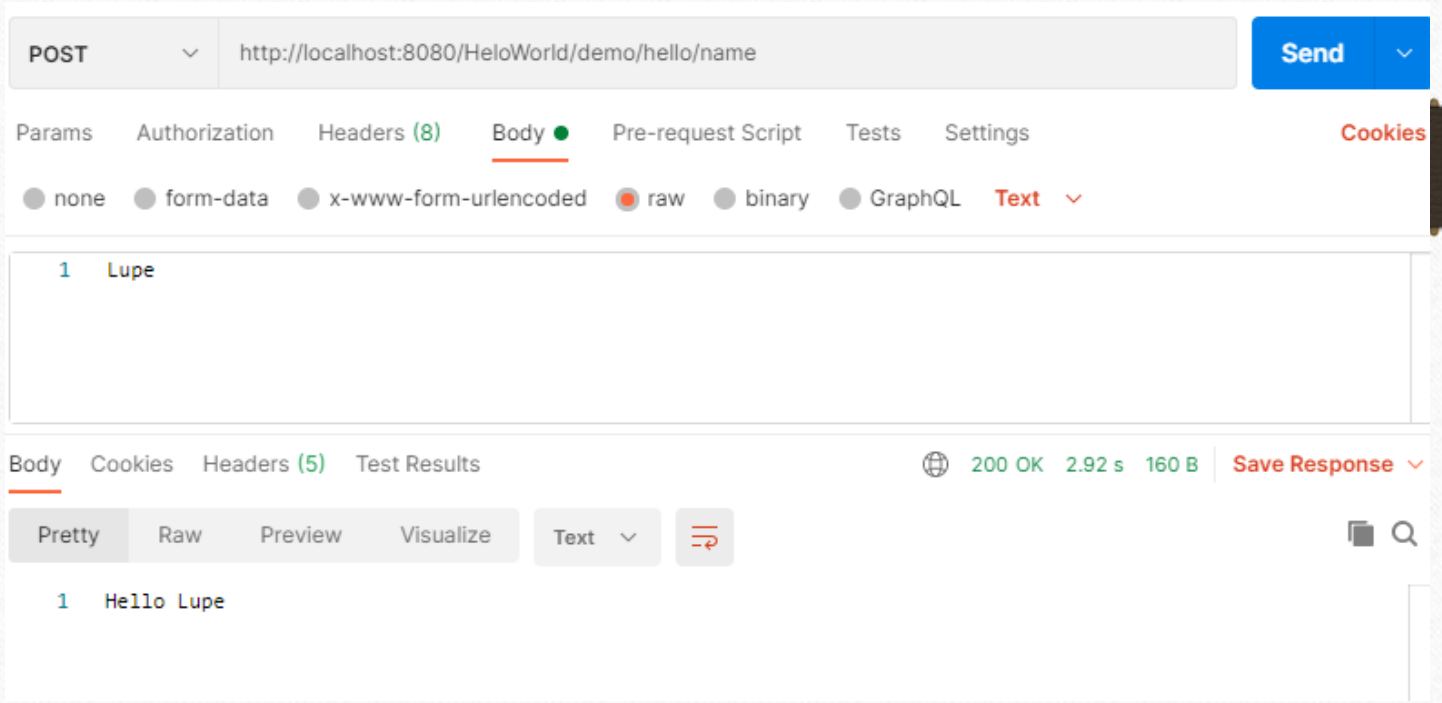
- We have added the tag that identifies the data type that we are going to send in the invocation (@Consumes)
- Test it:
  - Select POST
  - In the drop-down

  select text/plain
  - In the Payload field

  Write the text to be sent

5. @POST (ii)

- URI: http://localhost:8080/HelloWorld/demo/hello



5. @POST (iii)

- We are going to see now an example with JSON format
  1. Please remember that the JSON must be between keys and is composed of a set of keys separated by commas, between quotation marks and followed by colon and the corresponding values.

     For example:

     {

     “Name”:”Peter”,

     “Age”:33

     }
  1. Besides, in order to facilitate the automatic conversion to/from the class, the keys must be the same as the corresponding class private attribute names.

5. @POST (iv)

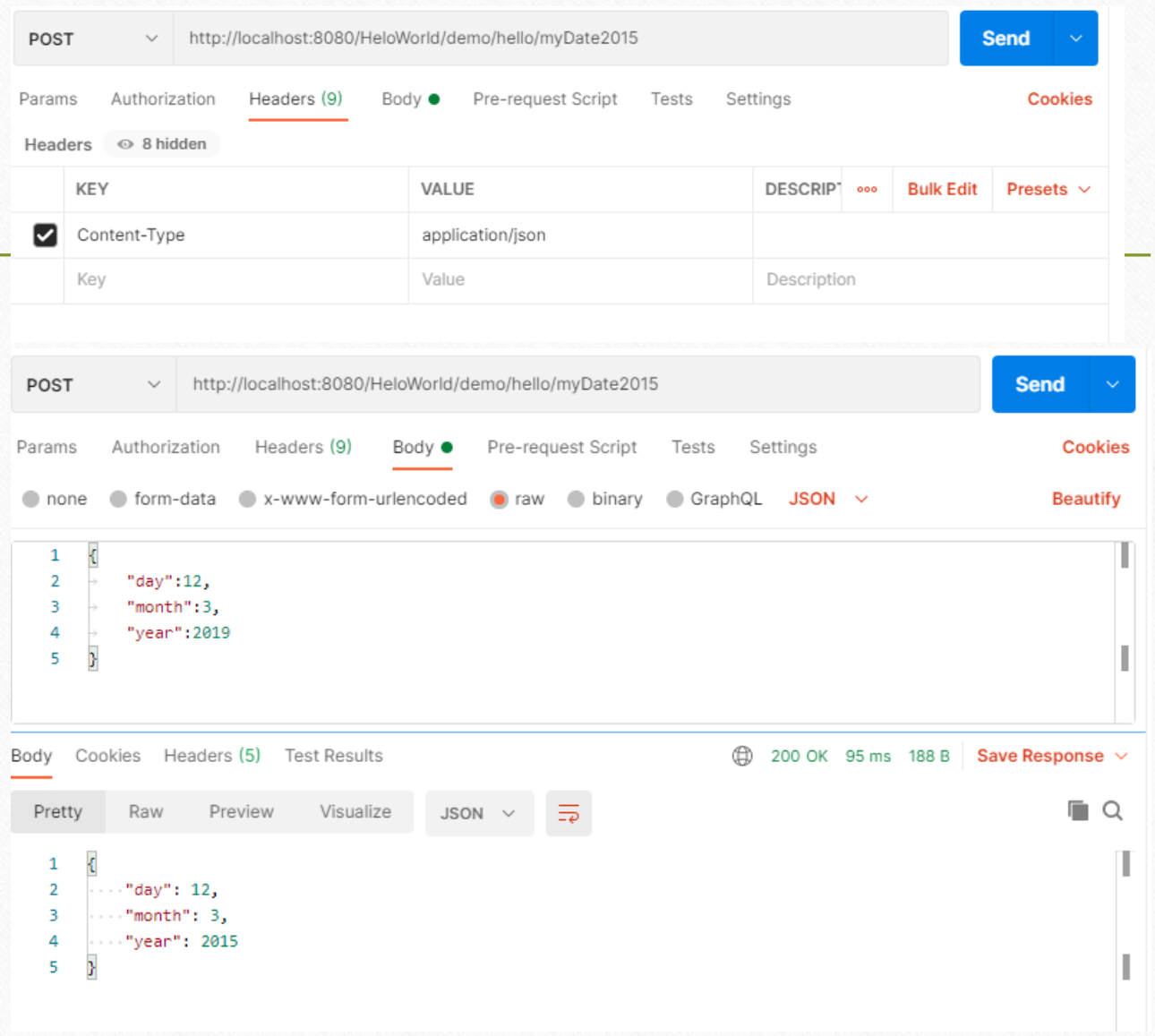
- Add a new method:

```
@POST
@Path("/myDate2015")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public MyDate dateToString(MyDate
myDate){
myDate.setYear(2015);
return myDate;}
```

- Test it.
- Note: Select **application/json** as **content-type**.
- Ensure that data are submitted in JSON format in the **raw payload**. Remember that the keys must be the same as the corresponding class private attribute names (in the example day, month, year). See previous slide to understand JSON format.
- Test the same function submitting a JSON and receiving an XML document.

5. @POST (v)

- URI: http://localhost:8080/HelloWorld/demo/hello/myDate2015



Content

1. @Path
2. @PathParam
3. @QueryParam
4. @XmlRootElement
5. @Post
6. **@FormParam**
7. @PUT
8. @DELETE

6. @FormParam (i)

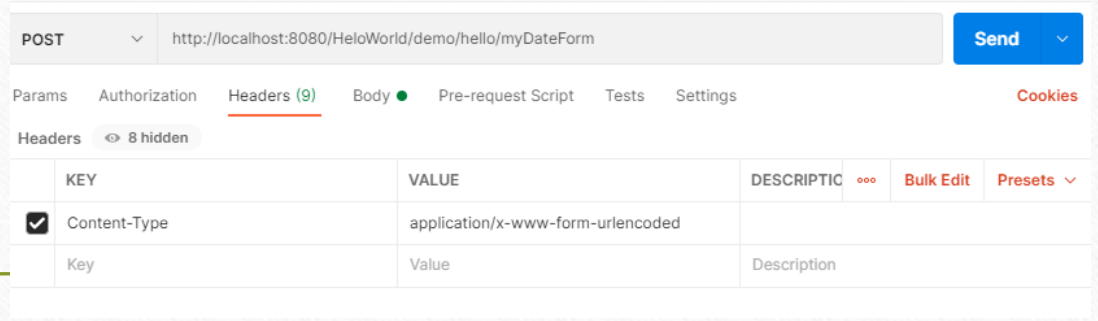
- It is used to send parameters (through POST) directly from an HTML form
- Add a new method:

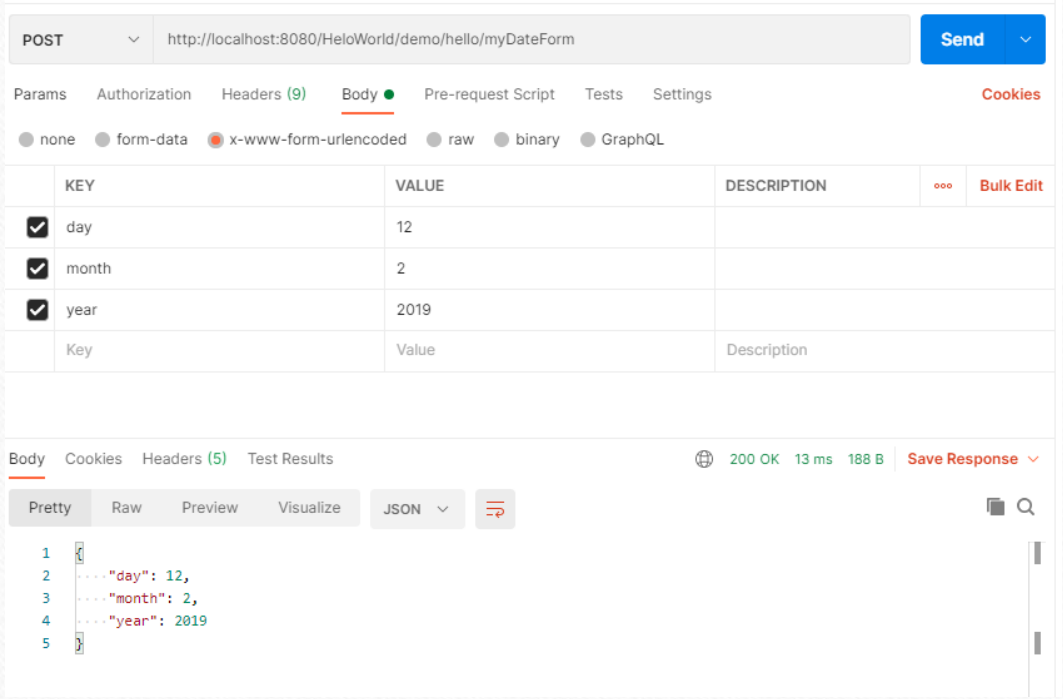
```
@POST
@Path("/myDateForm")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
```

```
public MyDate dateToText(@FormParam("day") int myDay, @FormParam("month") int myMonth, @FormParam("year") int myYear){
MyDate myDate = new MyDate();
myDate.setDay(myDay);
myDate.setMonth(myMonth);
myDate.setYear(myYear);
return myDate;
}
```

6. @FormParam (ii)

- URI: http://localhost:8080/HelloWorld/demo/hello/myDateForm





Content

1. @Path
2. @PathParam
3. @QueryParam
4. @XmlRootElement
5. @Post
6. @FormParam
7. **@PUT**
8. @DELETE

7. @PUT (i)

- In order to be able to modify, delete and test that data is correctly updated we are going to create a static structure in the Hello class:

```
private static Map<String,
MyDate> myMap = new HashMap<>();
```

-
```
static{
        MyDate myDate = new MyDate();
        myDate.setDay(25);
        myDate.setMonth(12);
        myDate.setYear(2014);
        myMap.put("ChristmasDay", myDate);
        myMap.put(" ChristmasEve", myDate);
        myMap.put("NewYearsEve", myDate);
        myMap.put("SantasDays", myDate);
        }
```

7. @PUT (ii)

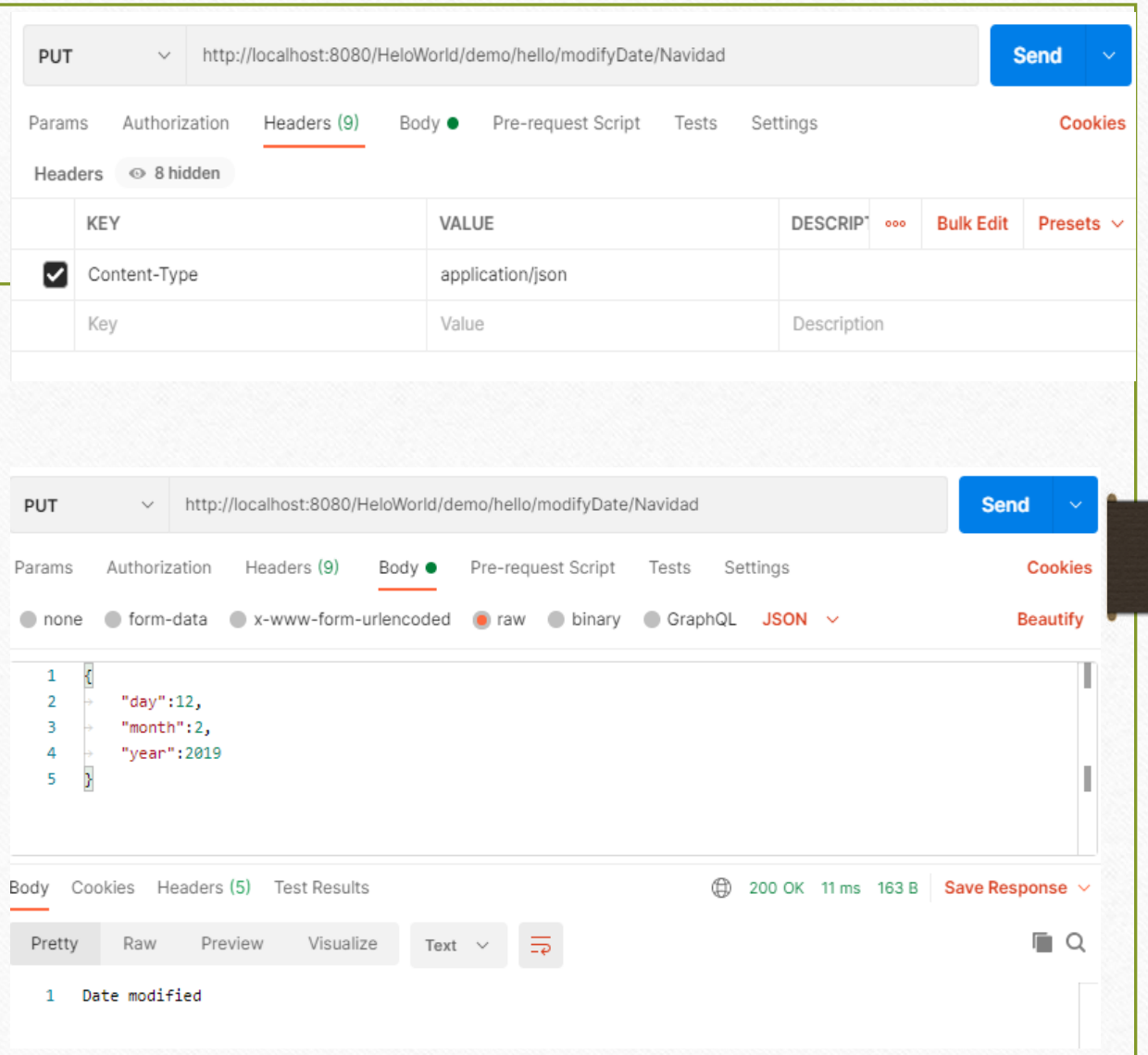
- Add a new method:

```
@PUT
@Path("/modifyDate/{date}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.TEXT_PLAIN)
  public String
modifyImportantDate(@PathParam("date")
String key, MyDate myDate) {
myMap.put(key, myDate);
return "Date modified"; }
```

- Pay attention: In order to test it you have to include the name of the date to be updated in the path (for instance, Christmas day) and as raw payload the json with the new date.
- After that you can test that everything worked by doing a GET request of the date in question.
- How would the delete be…?

7. @PUT (iii)

- URI: http://localhost:8080/HelloWorld/demo/hello/modifyDate/SantasDays



Content

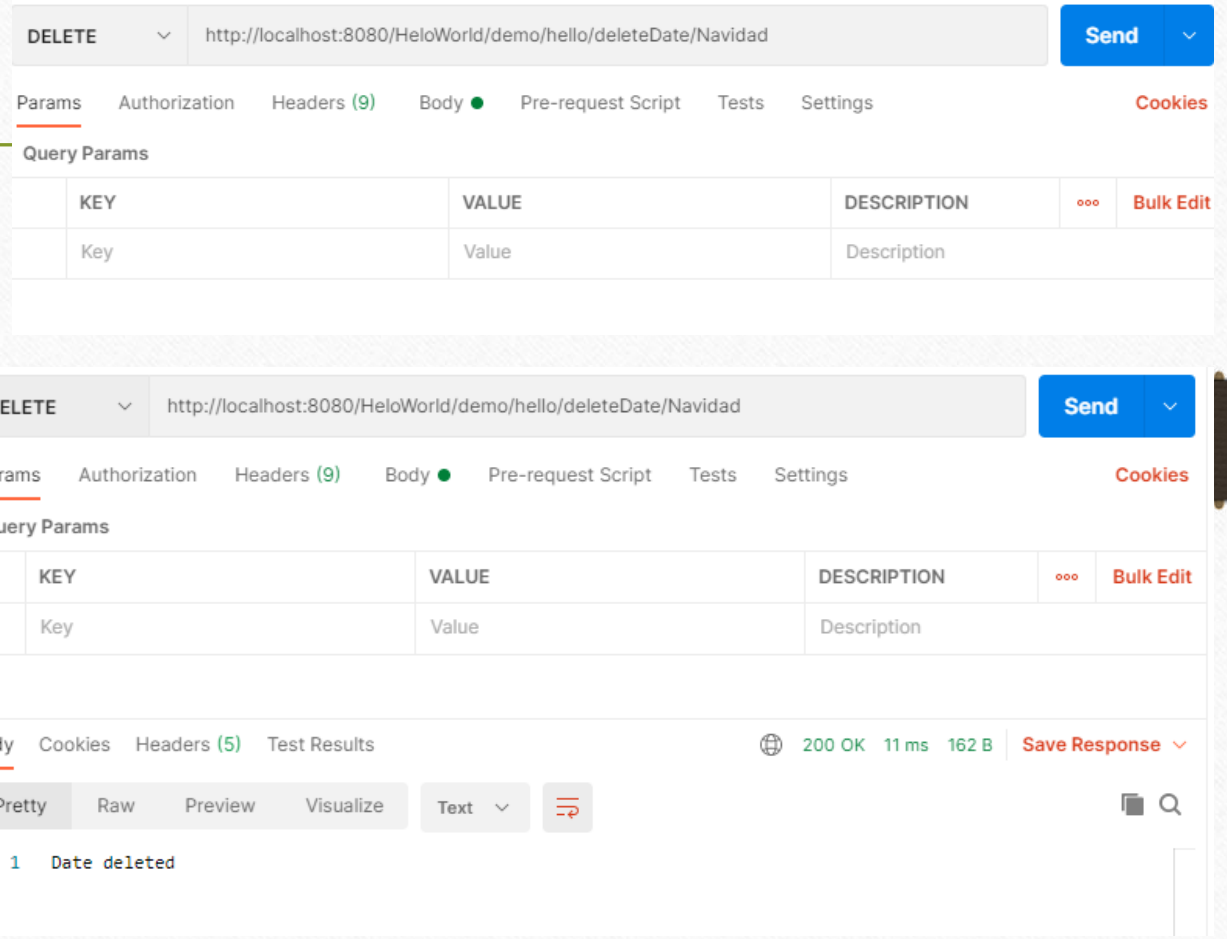
1. @Path
2. @PathParam
3. @QueryParam
4. @XmlRootElement
5. @Post
6. @FormParam
7. **@PUT**
8. @DELETE

8. @DELETE (i)

- Add a new method:

```
@DELETE
@Path("/deleteDate/{date}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.TEXT_PLAIN)
  public String
deleteDate(@PathParam("date") String
key) {
  myMap.remove(key);
return "Date deleted"; }
```

- URI: http://localhost:8080/HelloWorld/demo/hello/deleteDate/SantasDays

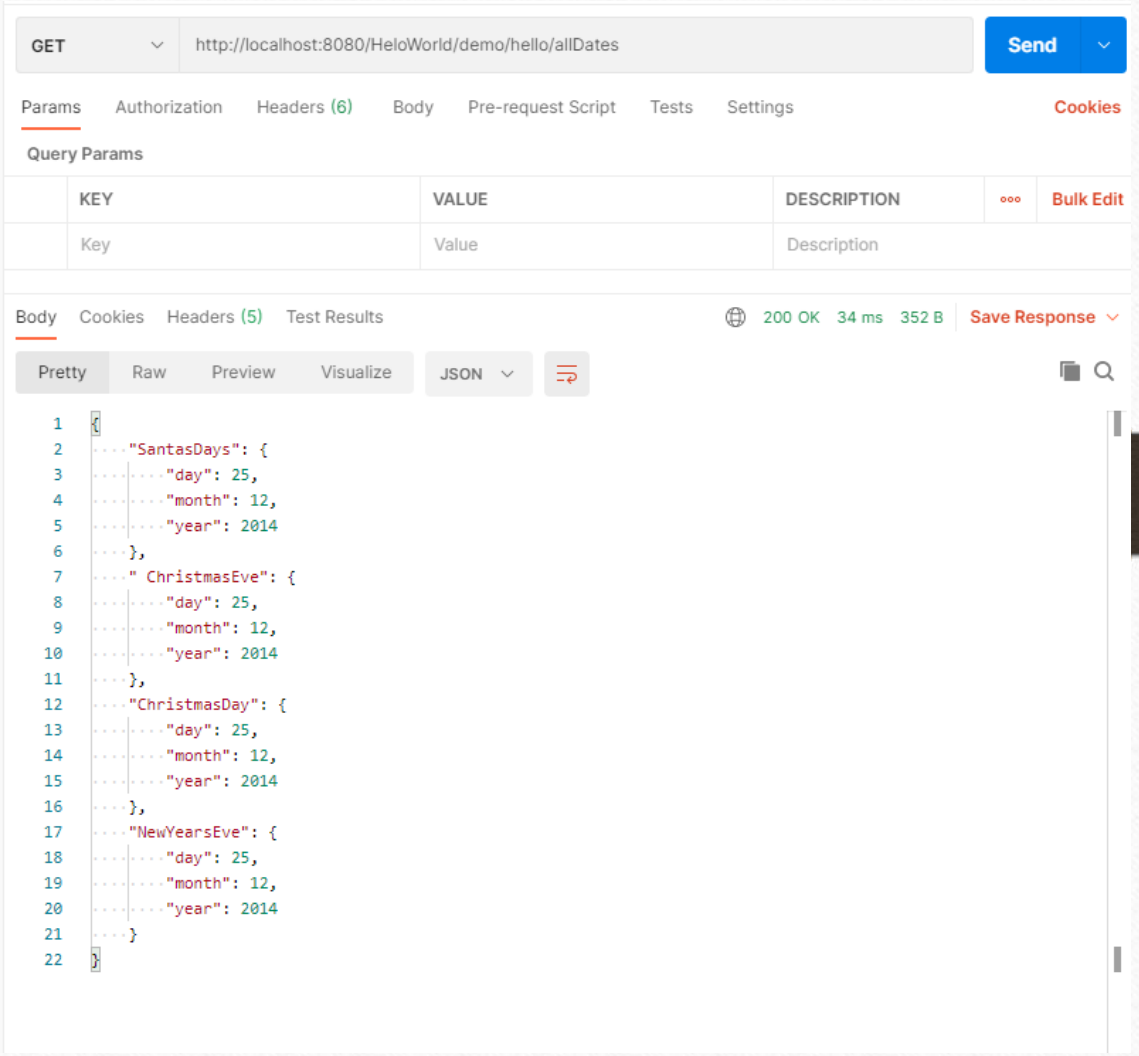


8. @DELETE (ii)

You can implement a method to get ***all*** the information stored to verify how *put* and *delete* are working properly.

```
@GET
@Path("/allDates")
@Produces(MediaType.APPLICATION_JSON)
public Map<String, MyDate> allDates() {
return myMap;
}
```

- URI: http://localhost:8080/HelloWorld/demo/hello/allDates



Support Bibliography and References

- Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON. By: Masoud Kalali; Bhakti Mehta. Publisher: Packt Publishing Pub. Date: October 15, 2013. Print ISBN-13: 978-1-78217-812-5
- REST with Java (JAX-RS) using Jersey - Tutorial Lars Voguel https://www.vogella.com/tutorials/REST/article.html
- Postman API Client - Postman Inc. https://www.postman.com/product/api-client/