ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στα Δίκτυα Υπηρεσιών

Assisting Lecture 6 - Java Restful Web Services Examples (JAX-RS)

Μύρων Παπαδάκης
Τμήμα Επιστήμης Υπολογιστών

CS-592: Introduction Service Networks Spring 2015

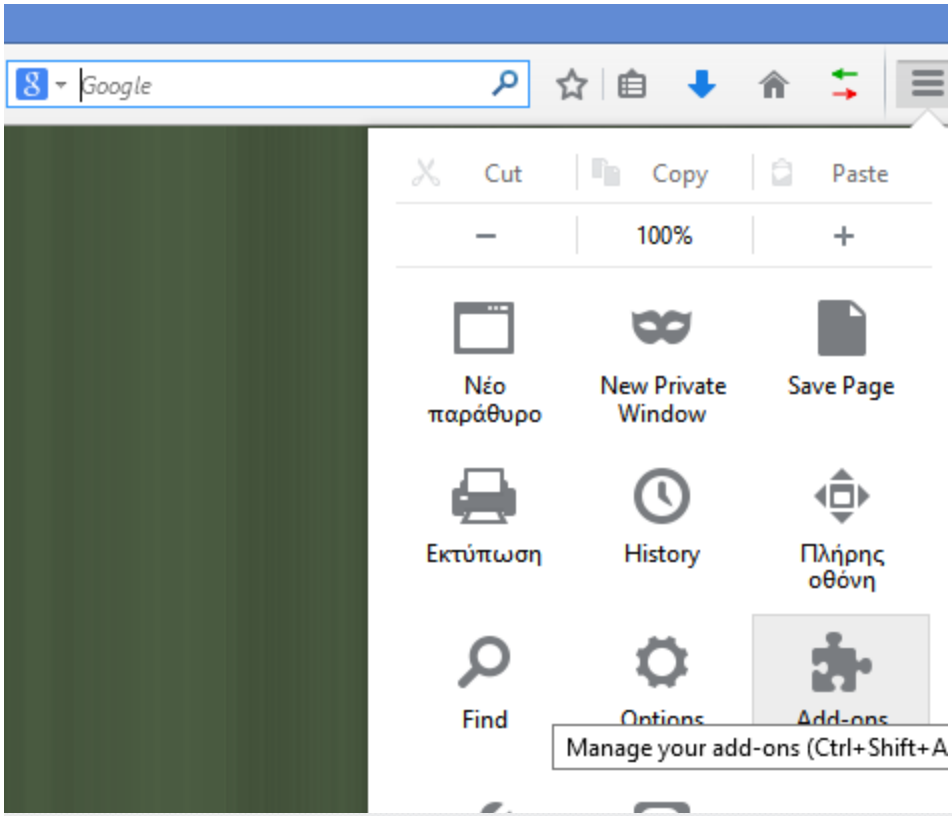
Rest Web Services Examples and JAX-RS

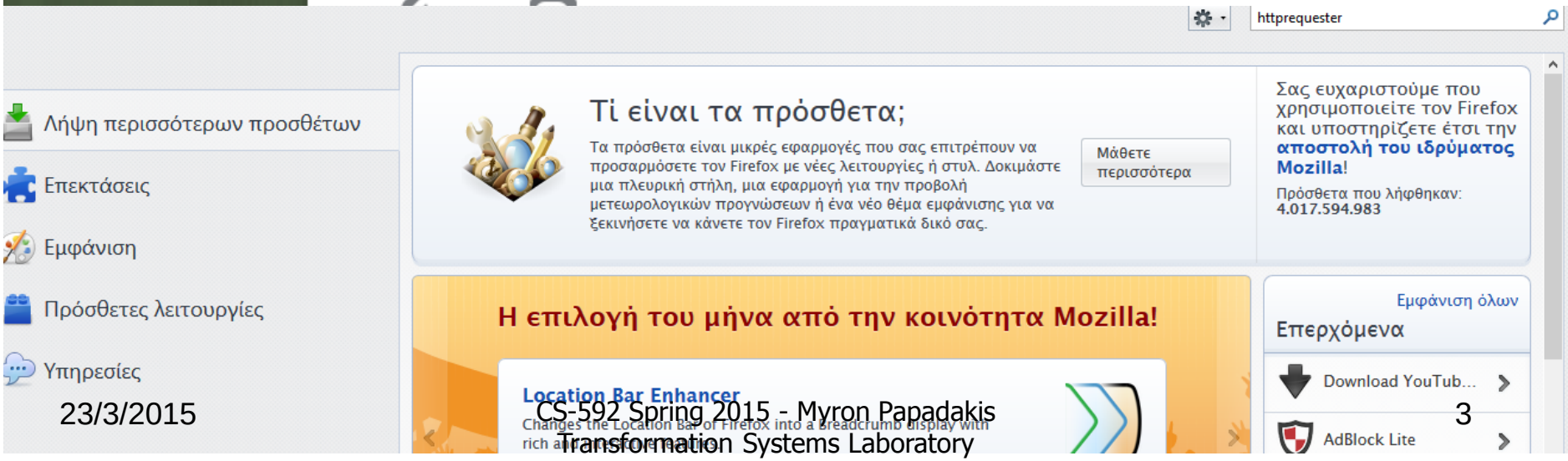
Myron Papadakis (myrpap@gmail.com)

Outline

- Jax Rs Tutorial
- Netbeans Example (Hello Rest)
- Netbeans Imported Example
- Netbeans Example – More complex (Book)

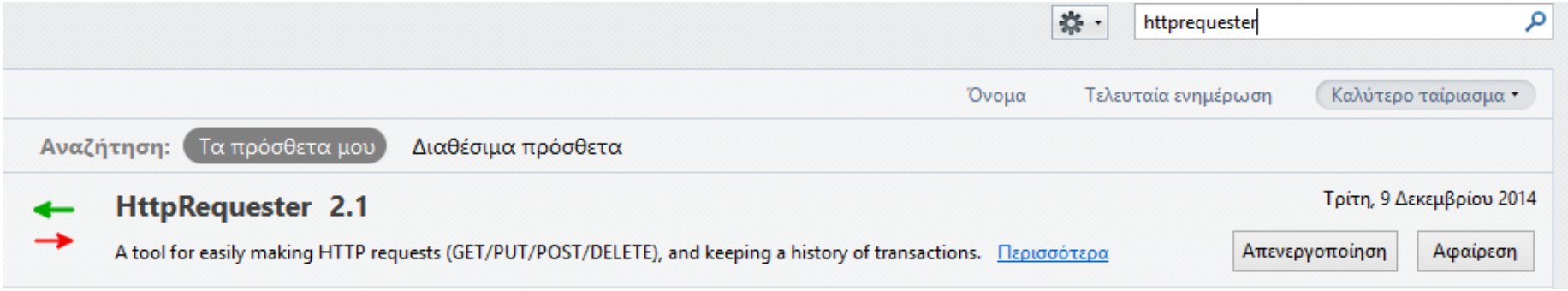
Useful Firefox Addons





Useful Firefox Addons

- In my pc says already installed, in your case it will prompt you to install it
  - Restart the firefox too



Java API for RESTful Web Services

JAX-RS

Recall: what makes up a restful web service

The definition of RESTful web service consists of

1. The base URI for the web service
   - e.g. `http://example.com/resources/`
2. The MIME type of the data supported by the web service
   - e.g. JSON, XML, YAML
3. The set of operations supported by the web service using HTTP methods
   - e.g. POST, GET, PUT, DELETE

JAX-WS for RESTful Web Services

- The Java API for XML Web Services (JAX-WS) provides full support for building and deploying RESTful Web services

- Sun article about programming RESTful Web Services with JAX-WS: http://www.oracle.com/technetwork/articles/javase/index-137171.html

- **But there is another specification JAX-RS**
  - **Java API for RESTful Web Services**

JAX-RS

- JSR 311: http://jcp.org/en/jsr/detail?id=311
- Part of the Java EE 6 platform, JAX-RS fully supports REST principles
- Uses annotations to simplify the development of RESTful web services
- Allow you to expose simple POJOs as web resources

Jersey

- Sun offers the open source, production quality Reference Implementation for JAX-RS code-named **Jersey**

- Jersey also provides an API so that developers may extend Jersey to suite their needs

The Resource Class (@Path)

- JAX-RS resource is any POJO that is annotated with **`@Path`** with relative URI path as value
  - The base URI is the application context

```
import javax.ws.rs.Path;

@Path("/stockquote")
public class StockResource {
    ........
    .......
     public String getStockInfo() {
         return "This is Stock Information";
     }
}
```

Resource Methods

- Resource methods are public methods of a resource class that you identify with a request method designator
  - **`@GET, @PUT, @POST, @DELETE`**
  - **`@HEAD, @OPTIONS`**
- The return values of methods with request designator annotations are generally
  - void
  - a Java language type
  - **`javax.ws.rs.core.Response`**

Example (http://www.javapassion.com/webservices/jaxrs.pdf)

```
// Assume the application context is
// http://example.com/catalogue, then
//
// GET http://example.com/catalogue/widgets
//      - handled by the getList method
// GET http://example.com/catalogue/widgets/nnn
//      - handled by the getWidget method.

@Path("widgets")
public class WidgetsResource {

@GET
String getList() {...}

@GET @Path("{id}")
String getWidget(@PathParam("id") String id)
{...}
```

URI Path Template

- URI path templates are URIs with variables embedded within the URI syntax

```
// Will respond to http://example.com/bookmarks/1234
@Path("/bookmarks/{bookmark}")
public class BookmarkResource {

@GET
public String getBookmark(
        @PathParam("bookmark") String bookmarkId) {
...
}
```

- The value of the bookmark variable may be obtained by adding the **`@PathParam`** on method parameter

Resource method parameters

- Resource methods can be annotated with one of the following annotations:

Annotation	Description
@MatrixParam	Extracts the value of a URI matrix parameter
@QueryParam	Extracts the value of a URI query parameter
@PathParam	Extracts the value of a URI template parameter
@CookieParam	Extracts the value of a cookie
@HeaderParam	Extracts the value of a header
@FormParam	Extract the value of a form
@Context	Injects an instance of a supported resource

http://www.devx.com/Java/Article/42873/0/page/2

Sub Resources

- Apart from the resource class, it is possible also to annotate **methods** of a resource class with the `@Path` annotation (sub resource methods)

```
@Path("/sayHello")
public class SayHello {

    public SayHello() {
    }

    @GET
    @Path("lastname")
    public String hello() {
          ..............
    }
}
```

GET request from the URI **`/sayHello/lastname`** will be handled by the **`hello()`** **sub-resource method** in the **`SayHello`** resource class

MIME Types Specification

- A resource class can produce or consume any type of MIME
- Use `@Produces` to specify the MIME type for the response
  - a representation that can be produced by a resource and sent back to the client
- Use `@Consumes` to specify the MIME type for the request
  - a representation of the specific content types that a resource can accept from an HTTP request entity

Example: MIME specification

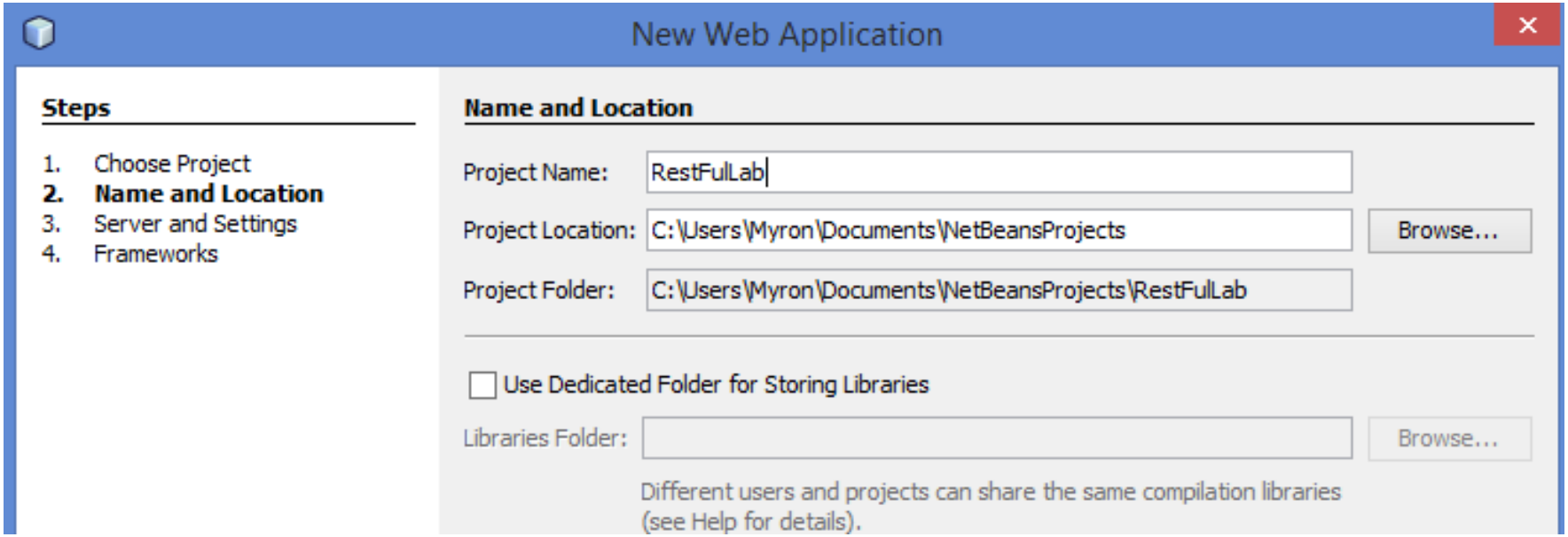
```
@Path("/sayHello")
@Produces("application/xml")
public class SayHelloResource {

    @GET
    public String getXml() {...}

    @GET
    @Produces("text/html")
    public String getHtml() {...}

    @PUT
    @Consumes("application/xml")
    public void putXml(String content) {...}
}
```

Netbeans RestFul Example

Similar to the Hello Project Example (Netbeans import)

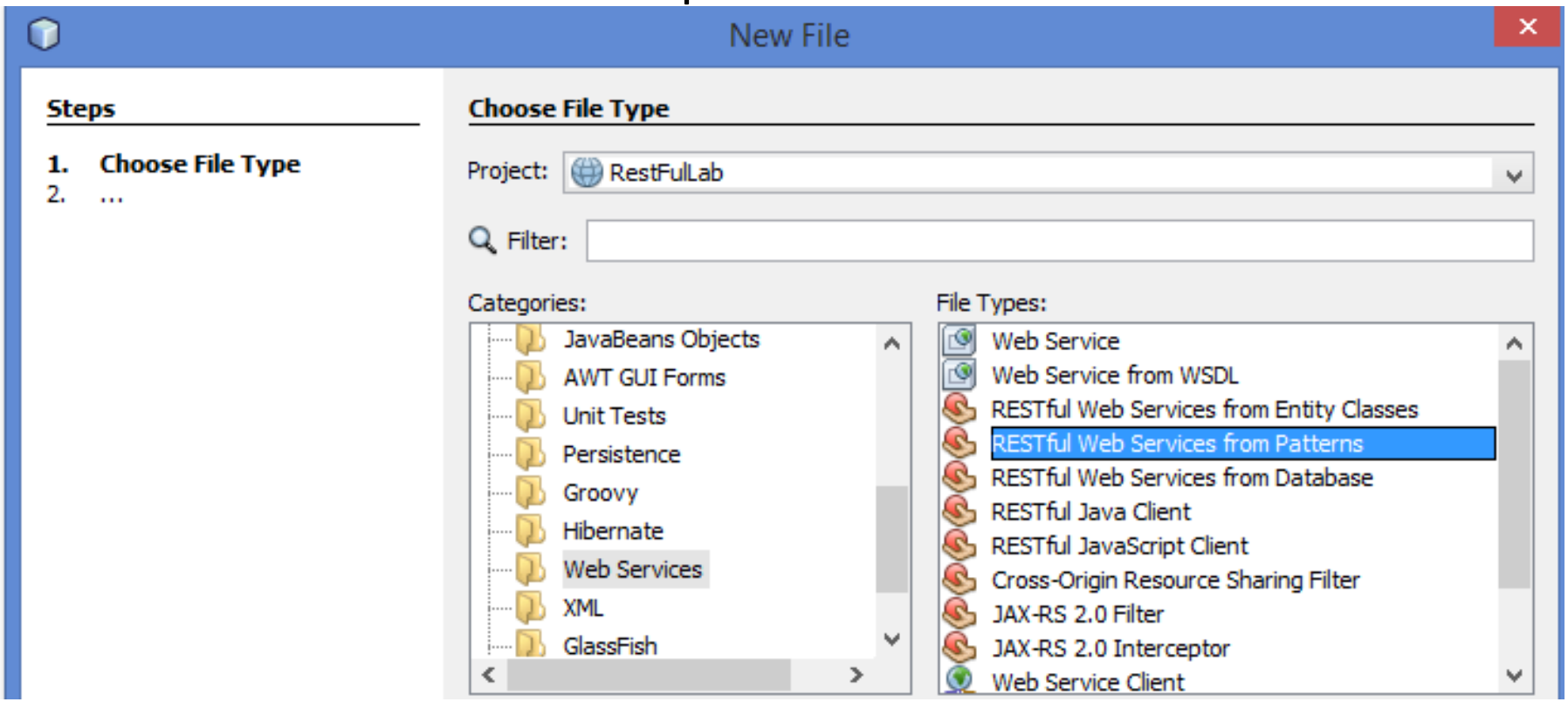
Netbeans Example > Create a Web Application (1)

- The first step is to create a web application. Name the web application RESTfulLab.

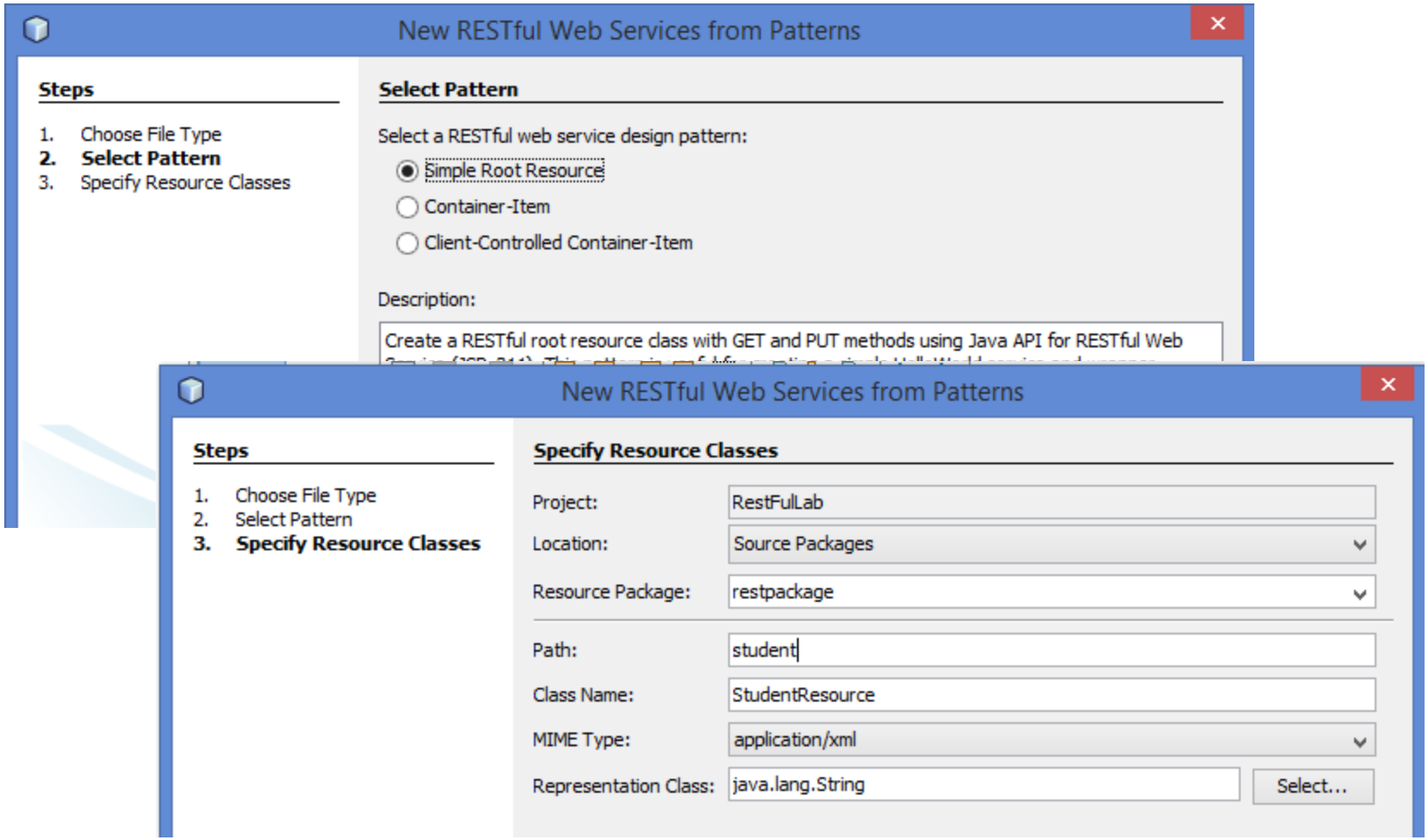


Netbeans Example > (2) Add a RESTful service.

- You should do this by selecting the option to create the RESTful web service from pattern.

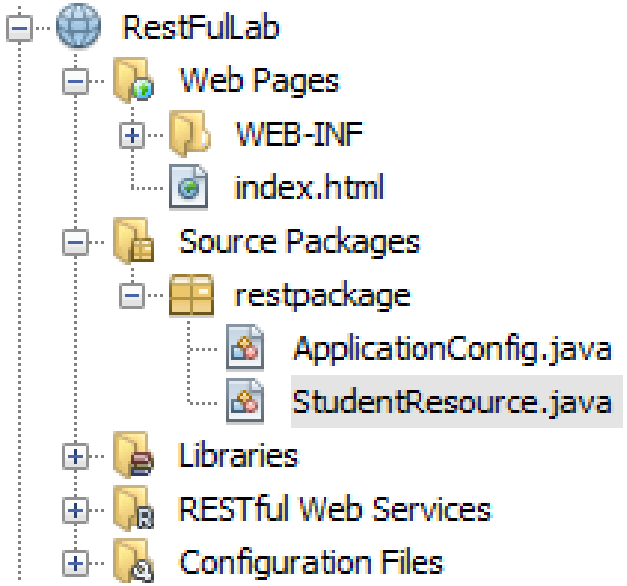


Netbeans Example > (3) Specify the Resource Classes



Netbeans Example > (3) Specify the Resource Classes

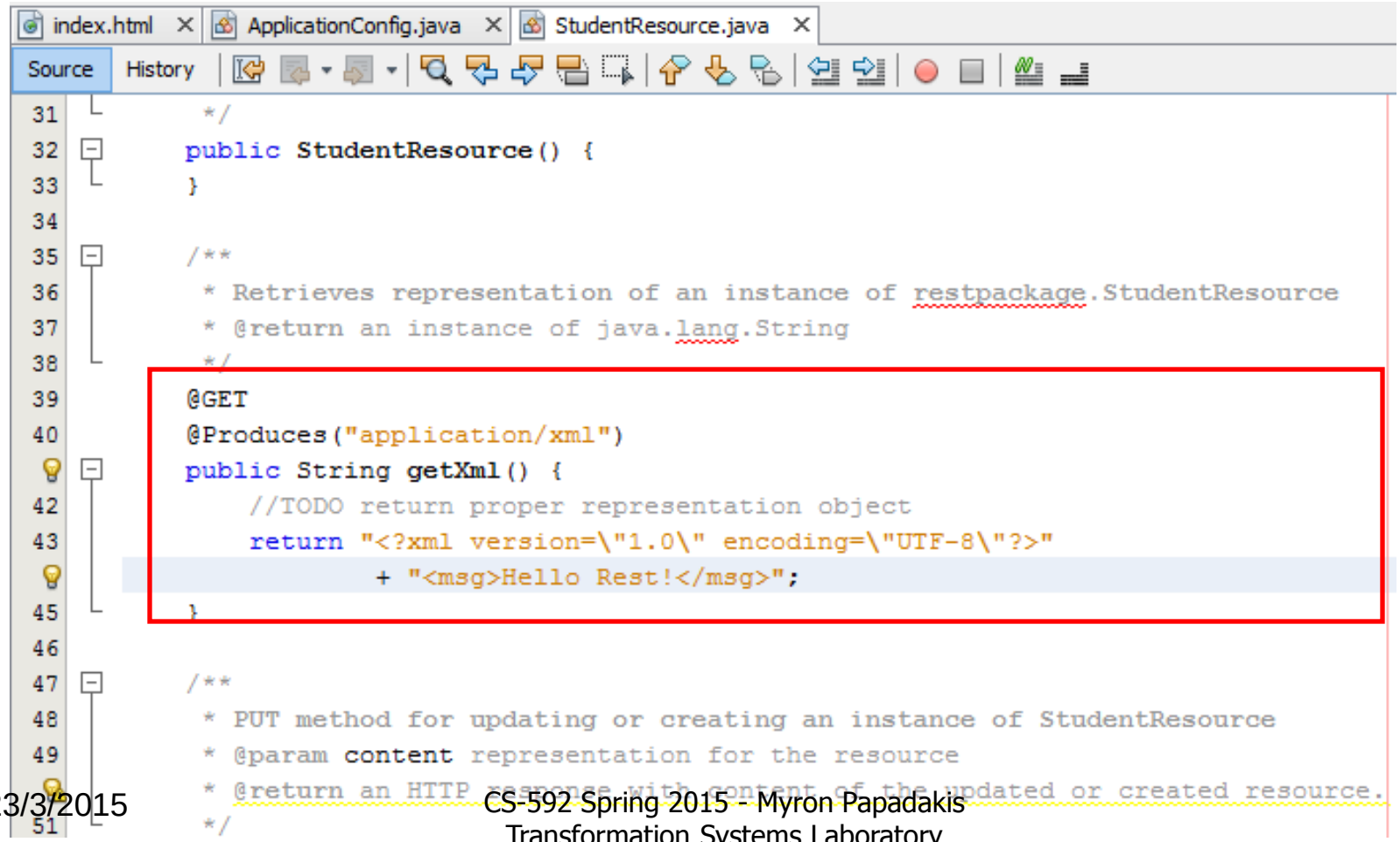
- Once this is done, keep clicking through the pages of the wizard, accepting the defaults on each page. You should wind up with a skeleton of a RESTful web service that looks like what you see on the next slide



```java
@Path("student")
public class StudentResource {

    @Context
    private UriInfo context;

    /**
     * Creates a new instance of StudentResource
     */
    public StudentResource() {
    }

    /**
     * Retrieves representation of an instance of restpackage.StudentResource
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("application/xml")
    public String getXml() {
        //TODO return proper representation object
        throw new UnsupportedOperationException();
    }

    /**
     * PUT method for updating or creating an instance of StudentResource
     * @param content representation for the resource
     * @return an HTTP response with content of the updated or created resource
```

Starting to modify the code

- Change the body of the getXml method to return a “Hello String”



Deploying and Testing a Restful Web Service

- Next, do a **Clean and Build**, and then right-click on the project node and select **Deploy**.

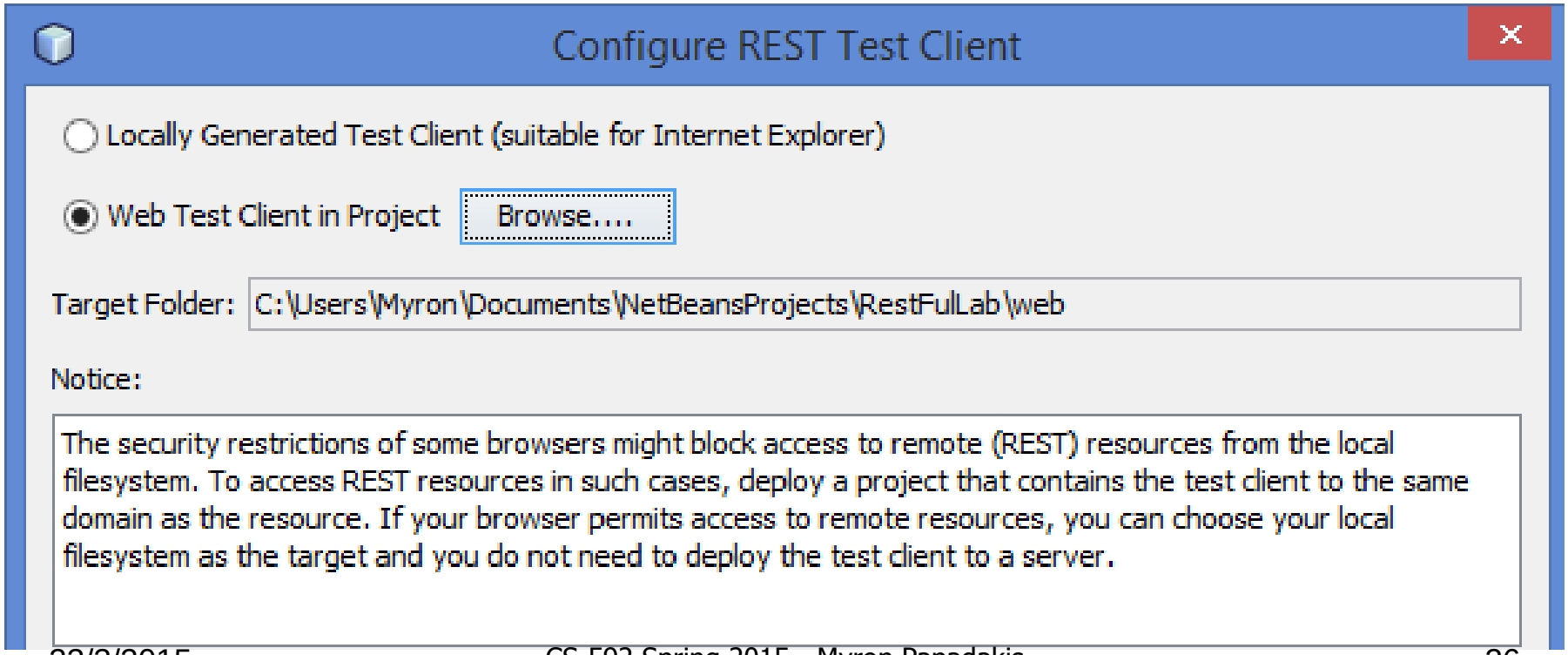
- We are now ready to test the web service.
  - Netbeans can use a browser as a test client for a RESTful service. There is a caveat though, only Internet Explorer works correctly to test RESTful services.

- You are now read to test the web service. Right-click on the project node, and select

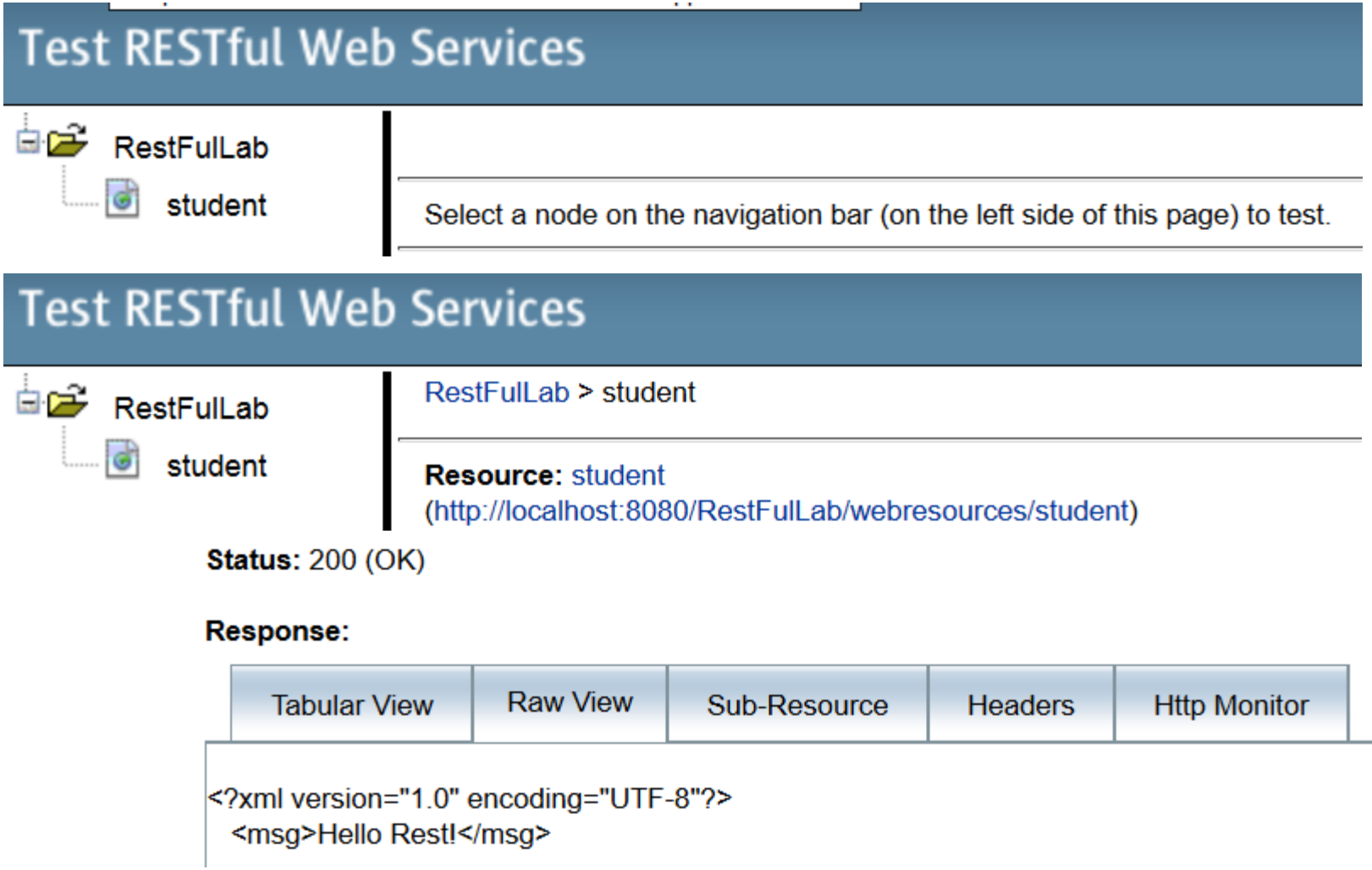
- **Test RESTful web services**

Configure REST Test Client

- When the first Option (Locally Generated Test Client) does not work select the second and select the project that you just deployed

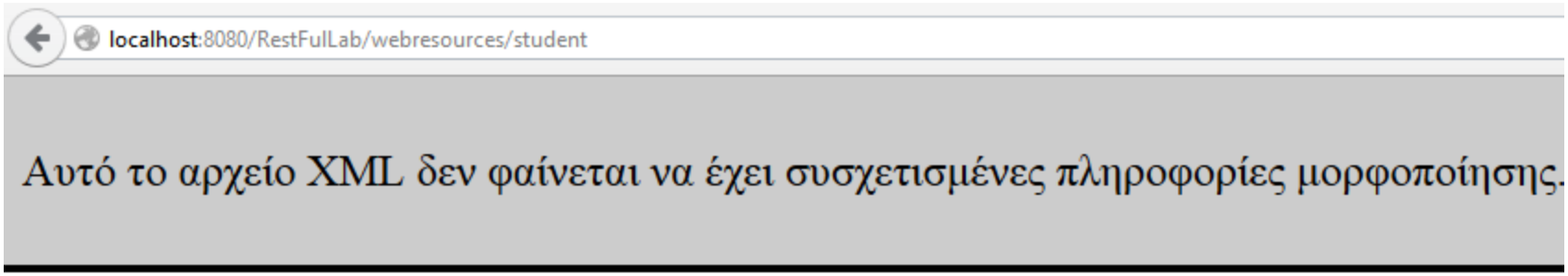


Test Restful Web Services



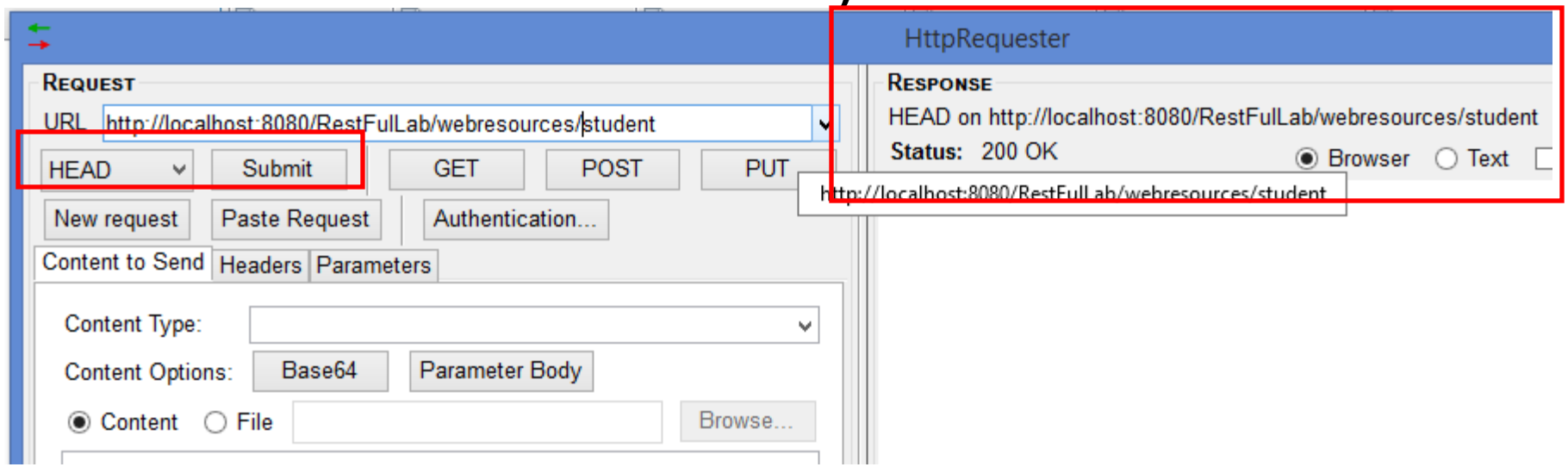
Testing the WS (Alternate)

- http://localhost:8080/RestFulLab/webresources/student

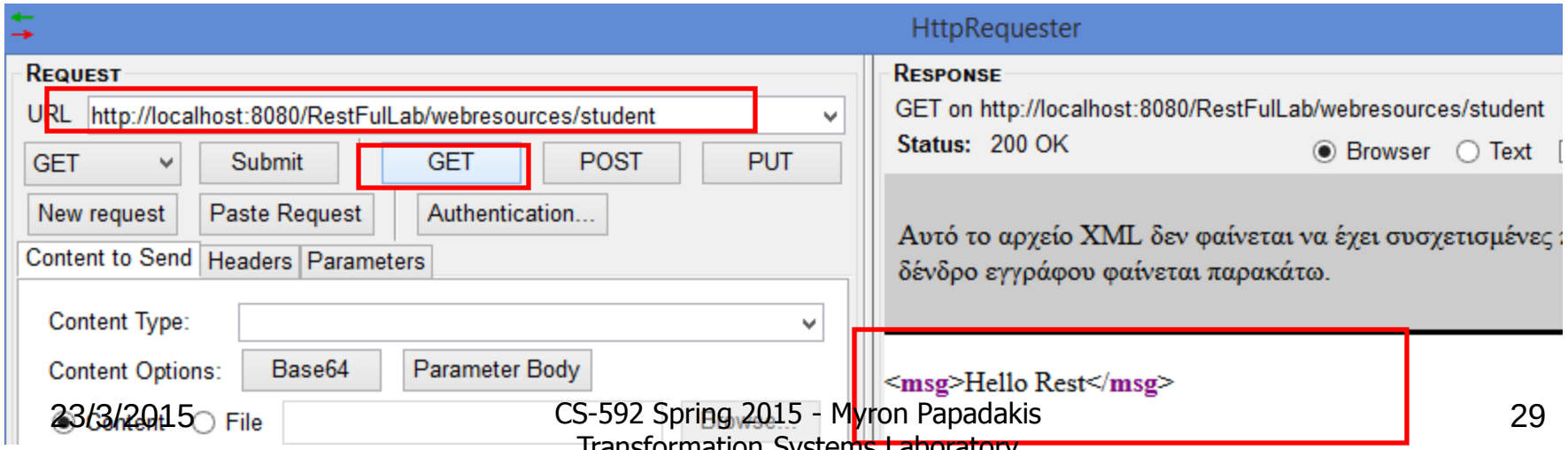


<msg>Hello Rest!</msg>

Alternate Testing (Using the Firefox Add-on)



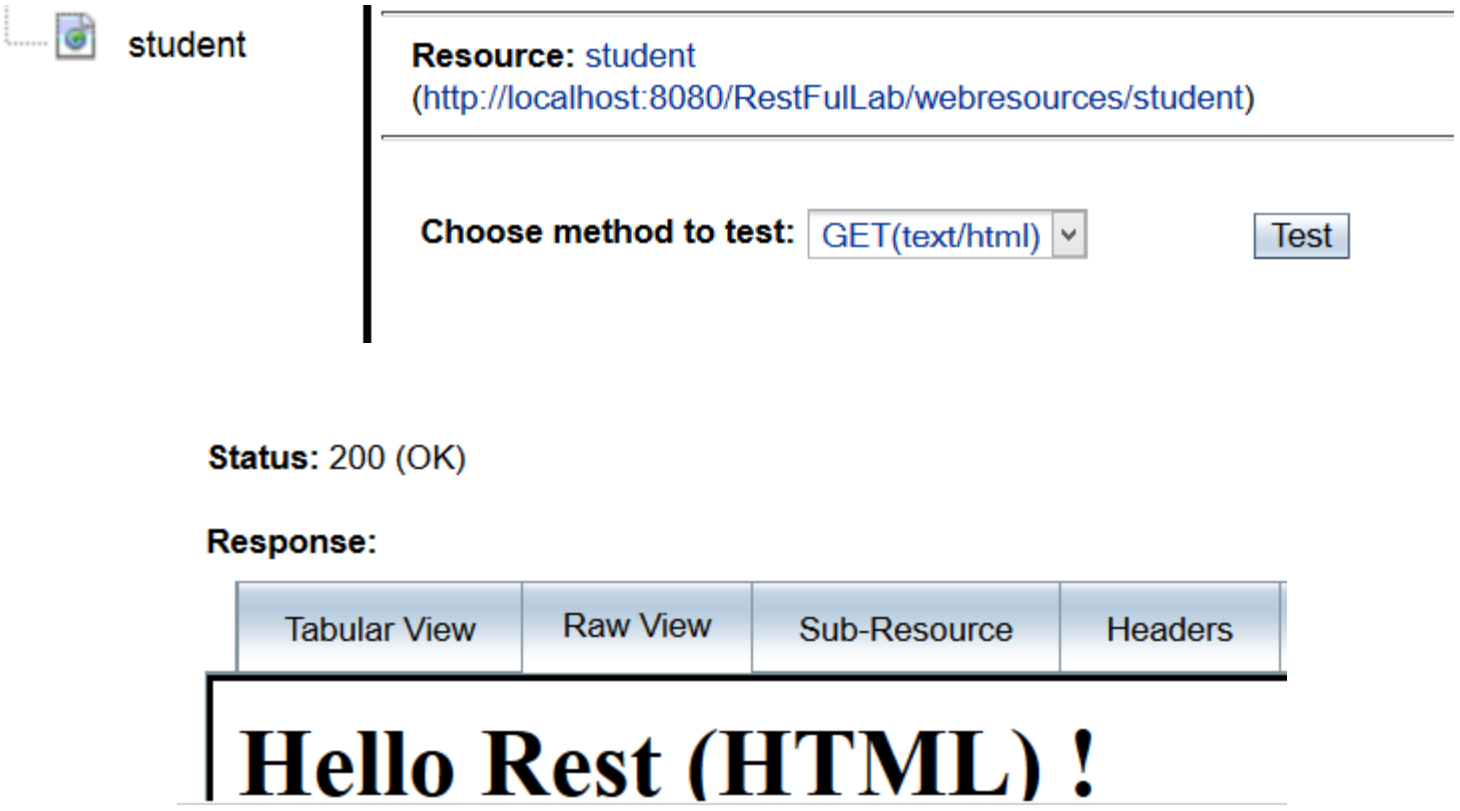
http://localhost:8080/RestFulLab/webresources/student



Change the Example (XML to HTML)

```
 */
@GET
@Produces("text/html")
public String getHtml() {
    //TODO return proper representation object
    return "<html><body><h1>Hello Rest (HTML) !</h1></body></html>";
}

/**
 * PUT method for updating or creating an instance of StudentResource
 * @param content representation for the resource
 * @return an HTTP response with content of the updated or created resource.
 */
@PUT
@Consumes("text/html")
public void putHtml(String content) {
```

Test the HTML Example



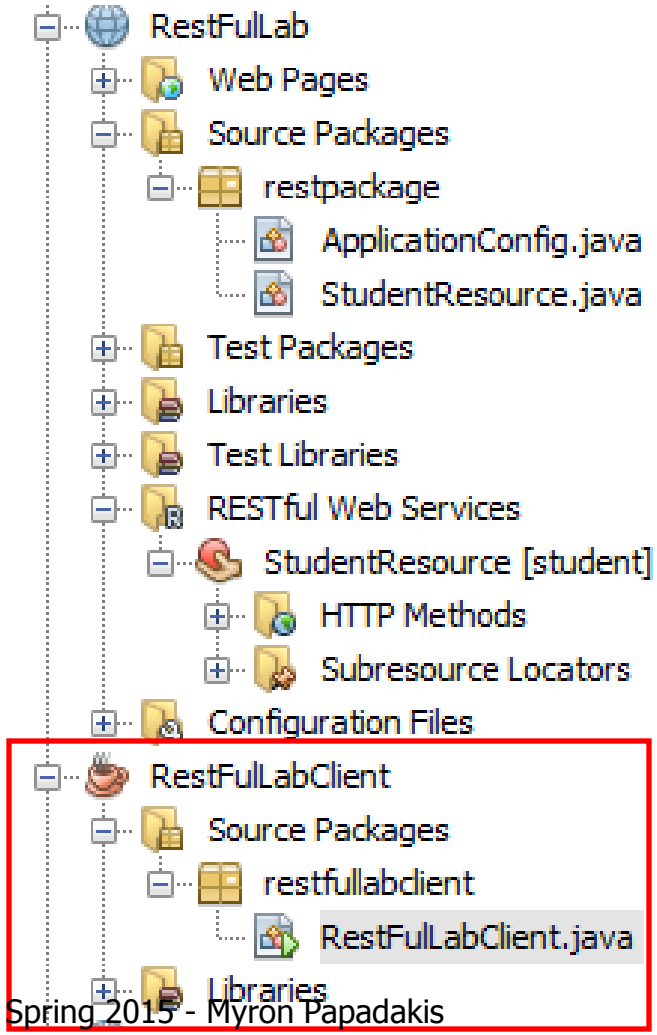
Test the HTML Example

- Immediately by providing the URL of the resource

Restful Java Client in Netbeans

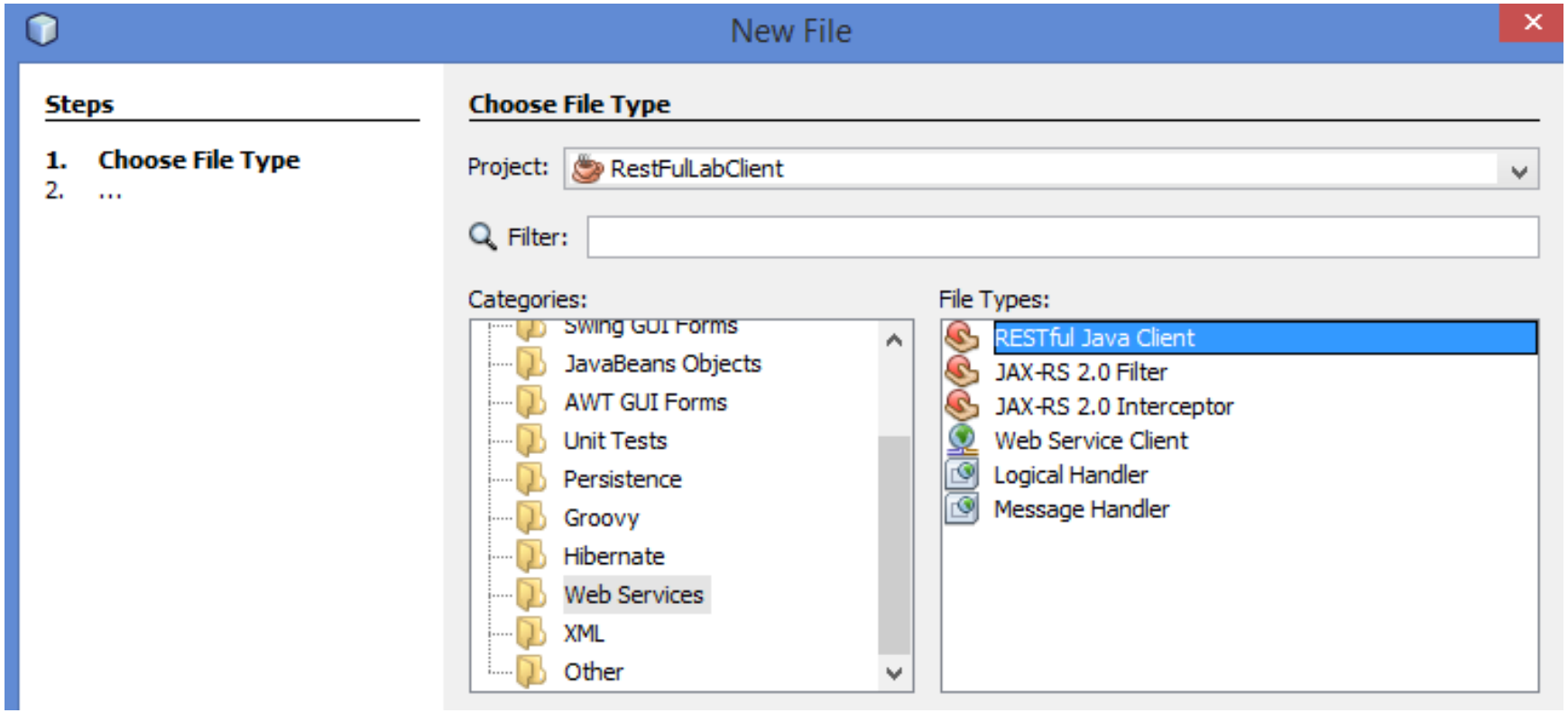
Creating the Client Project

- Create a new Java Application Project "RestFullLabClient"



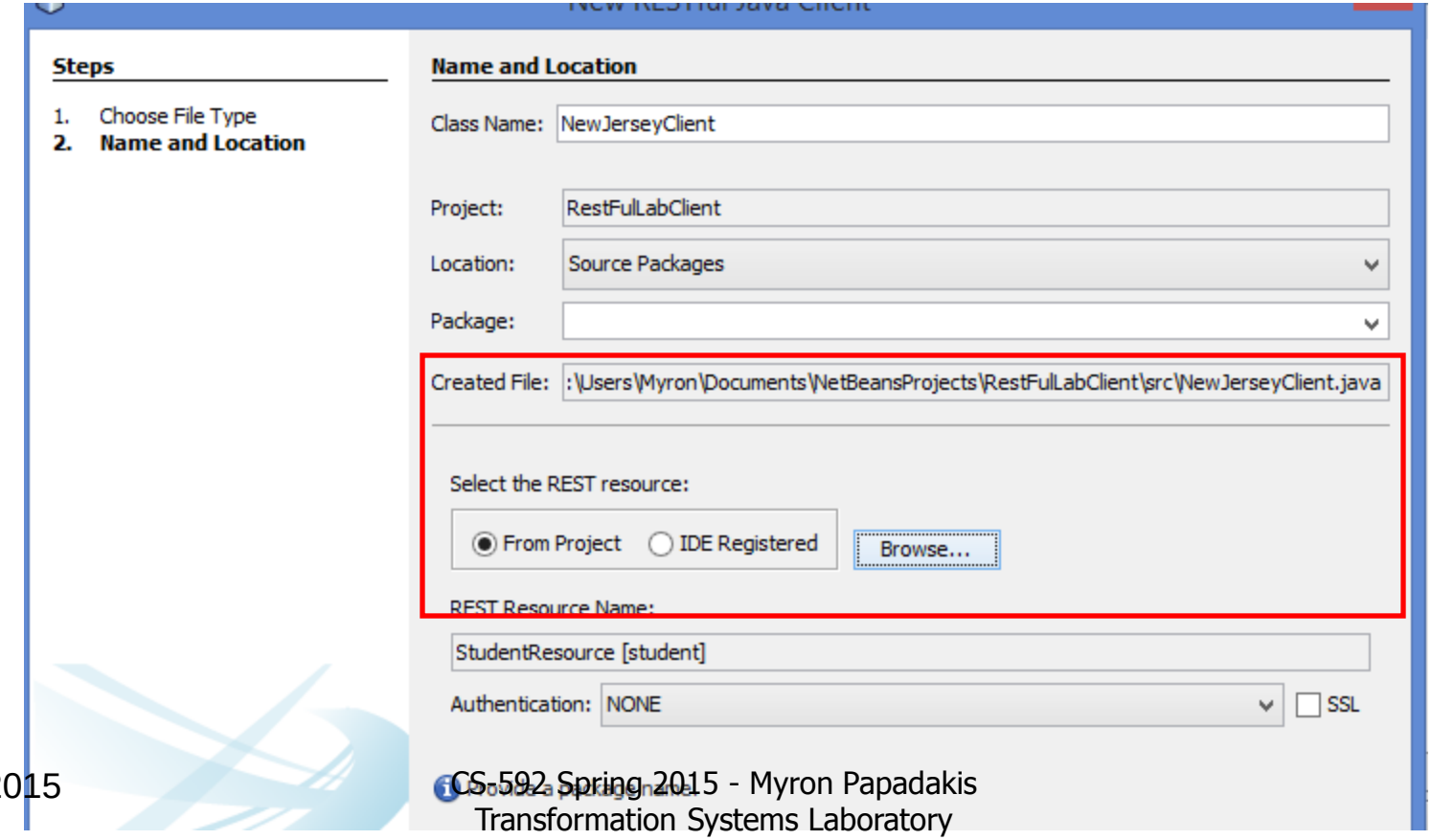
Creating the Client Project

- File > New > Other > RestFul Client



Creating the Client Project

- Select the Rest Resource > From Project > ResourceLab Project > StudentResource and press Finish



Client

- Something like the following stub will be created

```
public class NewJerseyClient {
    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/RestFulLab/webresources";

    public NewJerseyClient() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget = client.target(BASE_URI).path("student");
    }

    public String getXml() throws ClientErrorException {
        WebTarget resource = webTarget;
        return resource.request(javax.ws.rs.core.MediaType.TEXT_XML).get(String.class);
    }

    public void putXml(Object requestEntity) throws ClientErrorException {
        webTarget.request(javax.ws.rs.core.MediaType.TEXT_XML).put(javax.ws.rs.client.Entity.entity(
    }

    public void close() {
        client.close();
    }
```

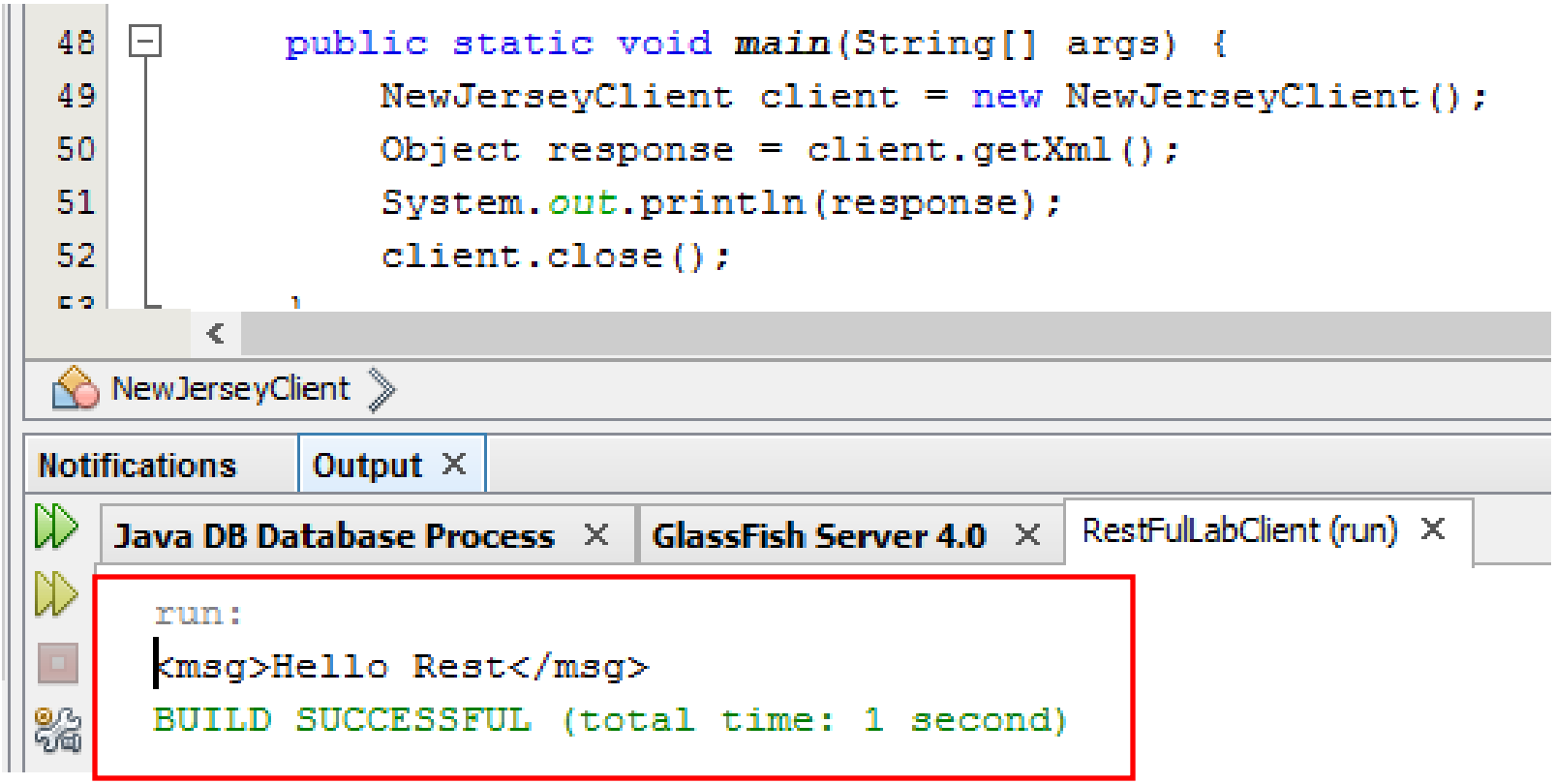
Client

- Take a look at the top of the file

```
package clientpackage;

import javax.ws.rs.ClientErrorException;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.WebTarget;

/**
 * Jersey REST client generated for REST resource:StudentResource [student]<br>
 * USAGE:
 * <pre>
 *        NewJerseyClient client = new NewJerseyClient();
 *        Object response = client.XXX(...);
 *        // do whatever with response
 *        client.close();
 * </pre>
 *
 * @author Myron
 */
```

Running the client

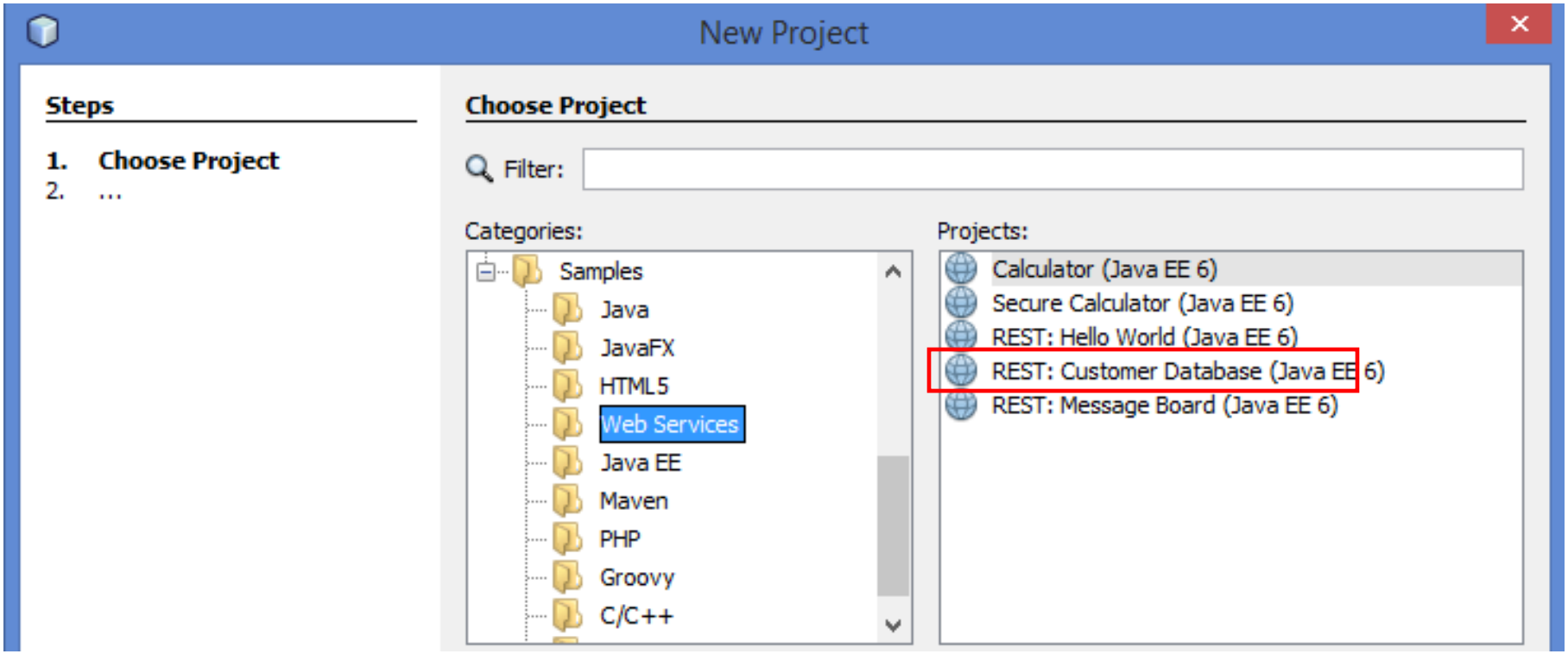
- In order to run the client the server must be running…



Netbeans Restful Example 2

Import a Sample Project

Example 2: Import Sample Project



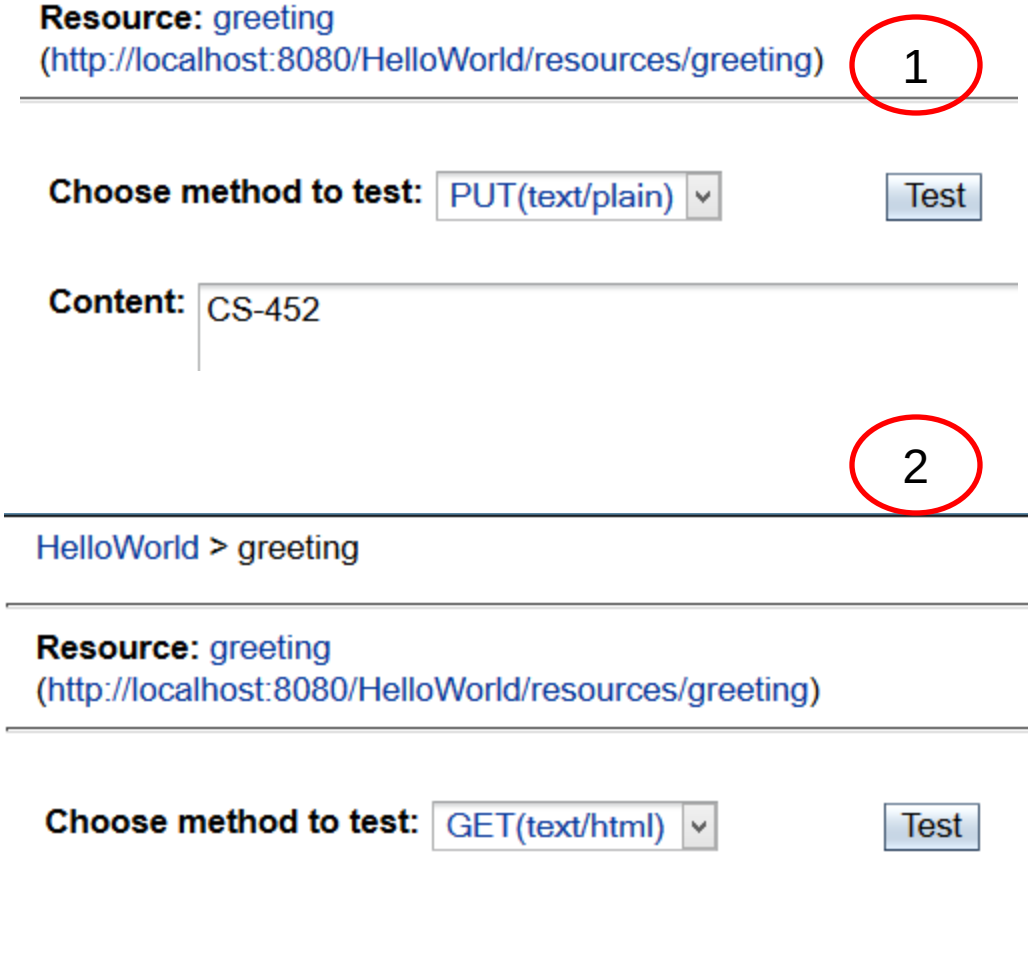
```java
@Stateless
@Path("/greeting")
public class HelloWorldResource {

    @EJB
    private NameStorageBean nameStorage;
    /**
     * Retrieves representation of an instance of helloworld.HelloWorldResource
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("text/html")
    public String getGreeting() {
        return "<html><body><h1>Hello "+nameStorage.getName()+"!</h1></body></html>";
    }

    /**
     * PUT method for updating an instance of Hello
     * @param content representation for the resour
     * @return an HTTP response with content of the
     */
    @PUT
    @Consumes("text/plain")
    public void setName(String content) {
        nameStorage.setName(content);
    }
```

```java
@Singleton
public class NameStorageBean {

    // name field
    private String name = "World";

    public String getName() {
        return name;
    }

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

Test the Sample Project

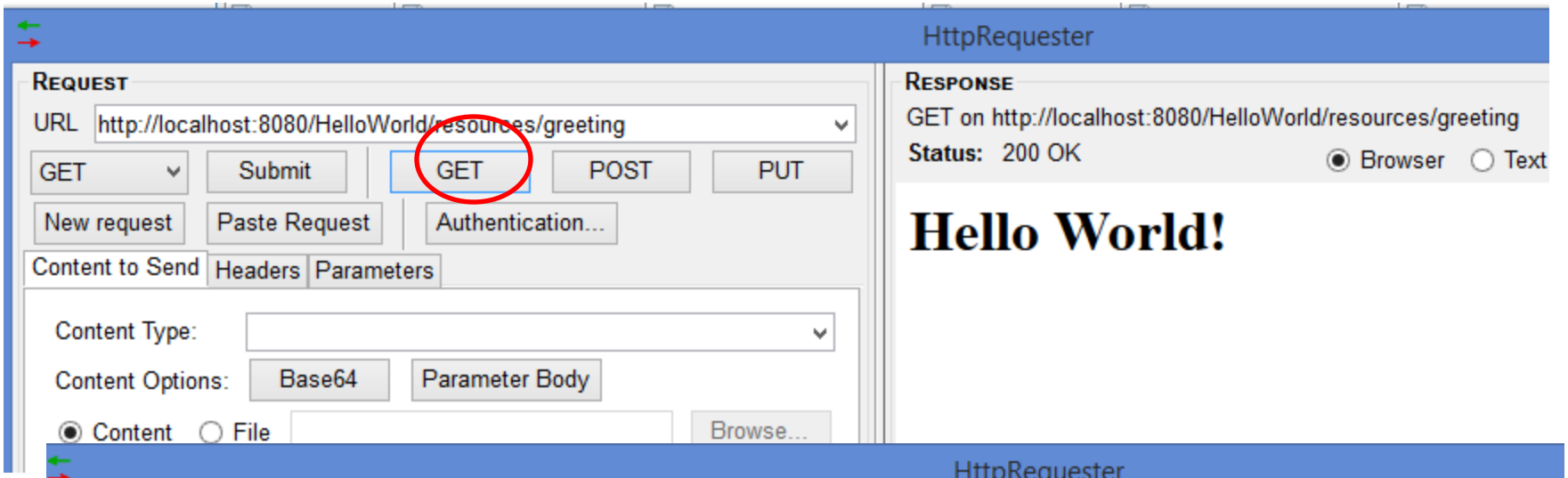

```
@Singleton
public class NameStorageBean {

    // name field
    private String name = "World";

    public String getName() {
        return name;
    }

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

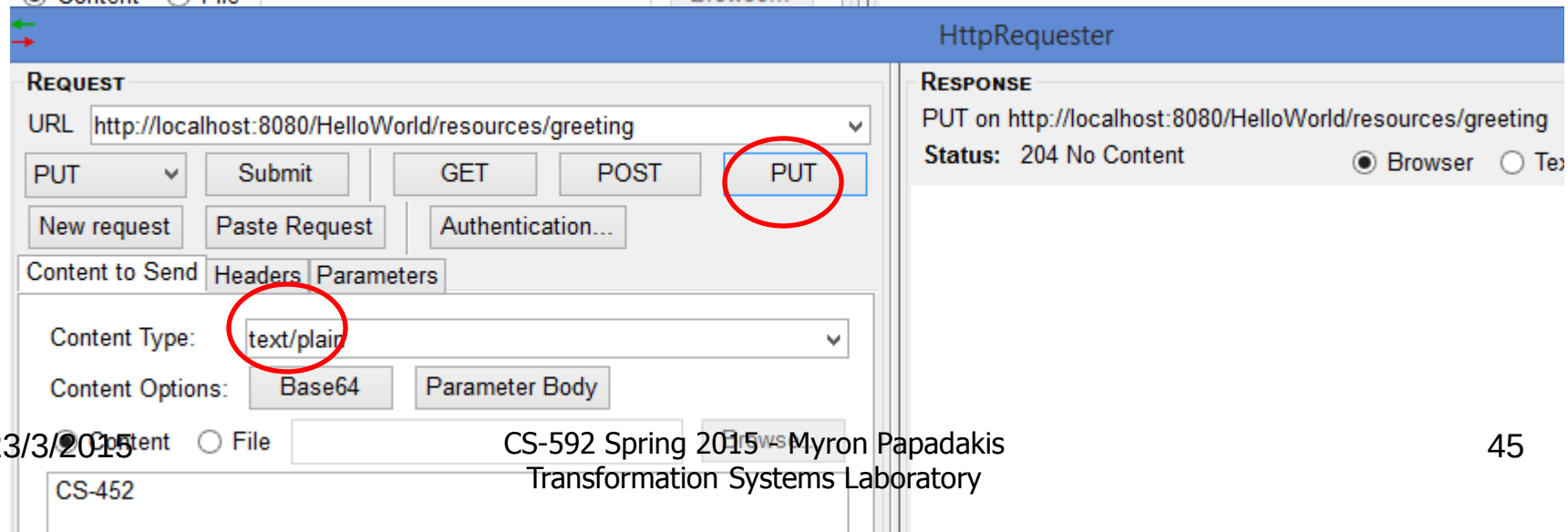
Test the Sample Project > Put a value and get back the response



Alternate Testing

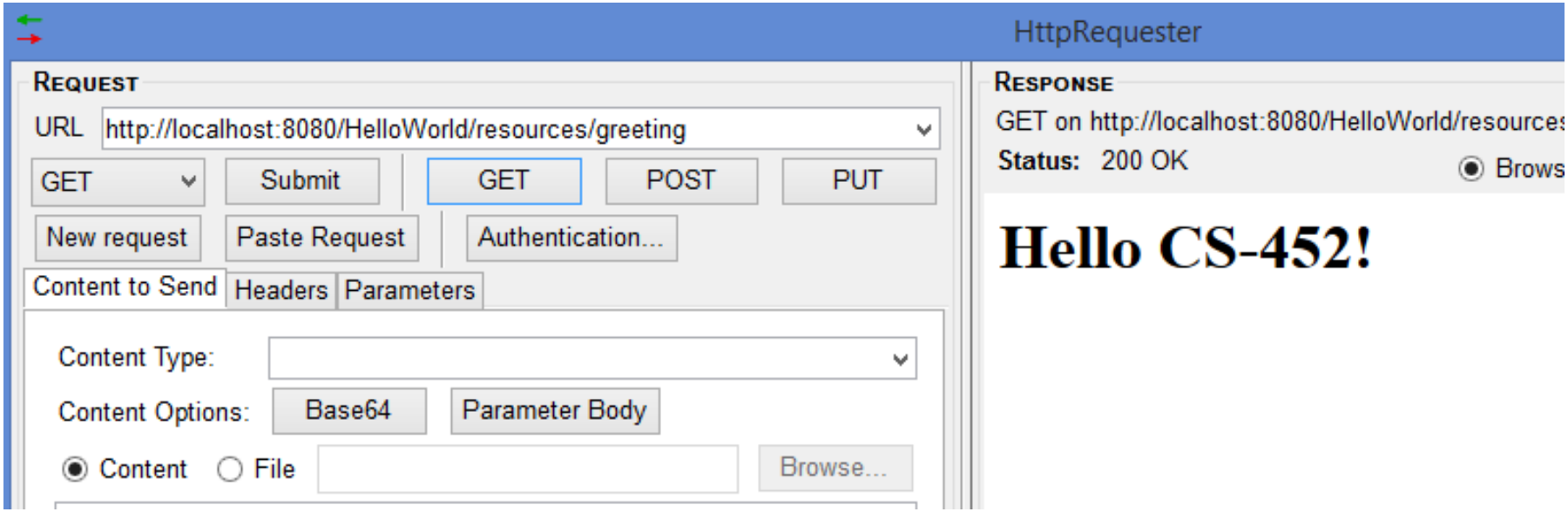
- http://localhost:8080/HelloWorld/resources/greeting





Alternate Testing

- After putting the content we check it by selecting GET again



More Complex Example

Book Example

Book Restful Web Service

- For starters develop a Restful Service that
  - Returns all books
  - Returns a book with a given id
  - Adds a book
- Obviously we will need a structure for the books (list, map or whatever)
- Firstly, the Restful Web Service must create a dummy book
- Each book has
  - Id
  - Name
  - Author
  - Isbn
  - Price

Book Restful Web Service

- Files that we will need to write
  - BookResource.java
  - Book.java

- In this example, we will be using **XML as the serialization format**, i.e. we will send and receive Book entities from the web service using XML.

- The **@XMLRootElement** annotation on the Book class is a JAXB annotation which allows **JAXB to convert this entity from Java to XML and back.** It is possible to annotate the fields and methods within the class to customize the serialization, but for our tutorial the JAXB defaults are fine.

Book.java

Book.java | BookResource.java

Source | History

```
1   package bookpackage;
2
3   import javax.xml.bind.annotation.XmlRootElement;
4
5   @XmlRootElement(name = "book")
6   public class Book {
7
8           private int id;
9           private String bookName;
10          private String bookAuthor;
11          private String bookISBN;
12
13          public int getId() {return id;}
14          public void setId(int id) {this.id = id;}
15          public String getBookName() {return bookName;}
16          public void setBookName(String bookName) {this.bookName = bookName;}
17          public String getBookAuthor() { return bookAuthor;}
18          public void setBookAuthor(String bookAuthor) {this.bookAuthor = bookAuthor;}
19          public String getBookISBN() {return bookISBN;}
20          public void setBookISBN(String bookISBN) {this.bookISBN = bookISBN;}
```

The resource will thus be hosted at "/bookresource".

- **Singleton lifecycle ensures that only one instance of this class will created by Jersey per web-application.**

```
@Path("bookresource")
@Singleton
public class BookResource {

    @Context
    private UriInfo context;

    private HashMap<Integer, Book> bookMap = new HashMap<Integer, Book>(

    public BookResource() {
        Book book = new Book();
        book.setBookAuthor("Bhaveh Thaker");
        book.setBookName("Introduction to RESTful Web Services");
        book.setBookISBN("ISBN 10: 0-596-52926-0");
        addBook(book);
    }

    @GET
    @Path("books")
    public List<Book> getBooks() {
        List<Book> books = new ArrayList<Book>();
        books.addAll(bookMap.values());
        return books;
    }

    @GET
    @Path("{id}")
    public Book getBook(@PathParam("id") int bookId) {
        return bookMap.get(bookId);
    }
```

Notes

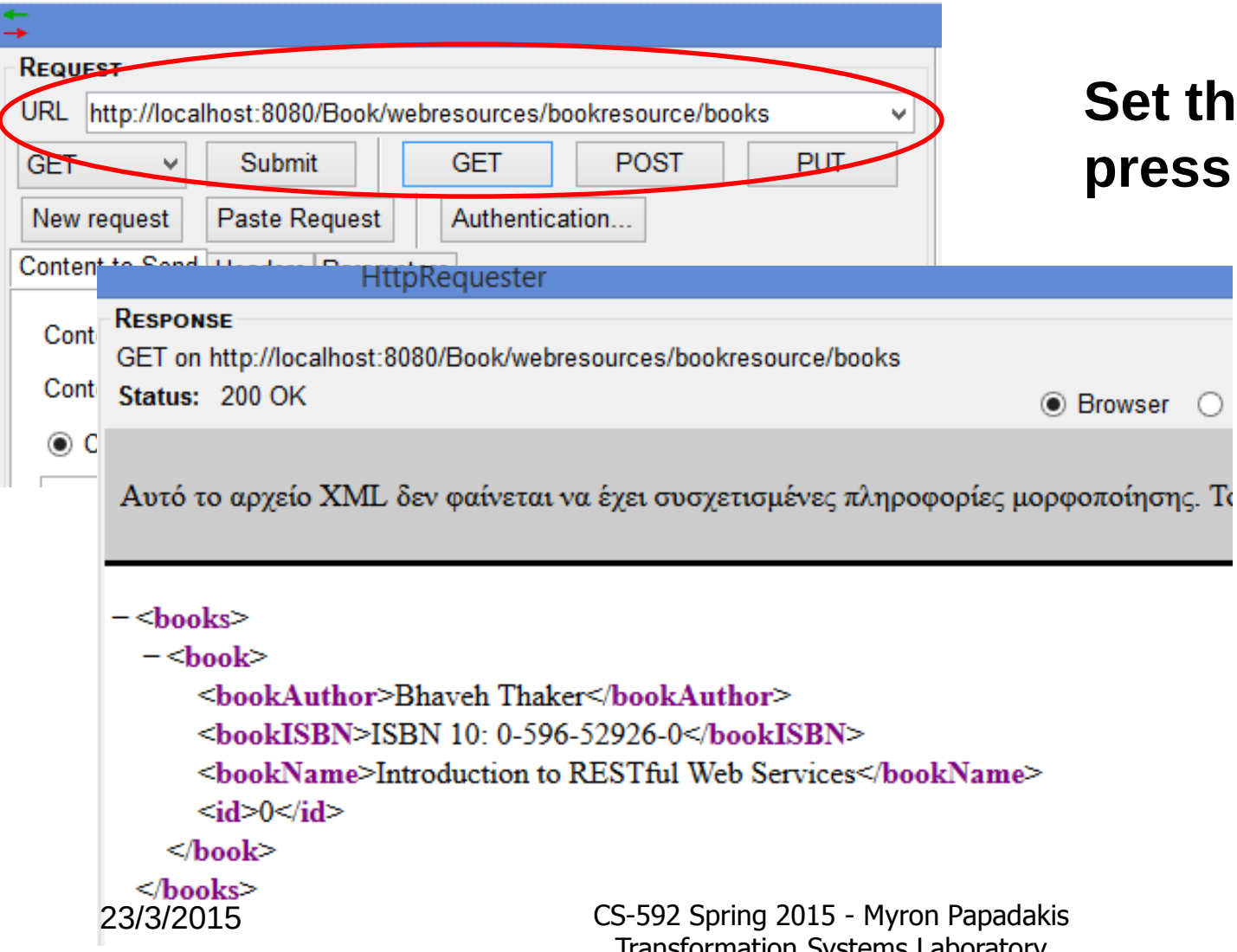
- The Consumes and Produces combos can be used to specify the default mime type(s) of data which this resource can accept and generate. These values can be overridden by individual methods in the class.
  - We will be serializing to XML, so we use the application/xml mime type.

- The URL path field specifies the path at which this method can be reached, relative to the containing resource.
  In this case we specify ***{id}***, which means this resource method can be reached at **/bookresource/*{id}***.
  - The curly braces denote a URI variable.
  - These variables are substituted at runtime in order for a resource to respond to a request based on the substituted URI.

Notes

- Since we need the value of the id variable, we use the PathParam annotation to map it to the cId parameter.
- In **addBook()** case, we're responding to a POST request and expect **application/xml** input which would be deserialized into the *book* parameter.
- The ***book*** parameter is an **Entity** parameter (unannotated) and is mapped directly from the message body of the incoming request.

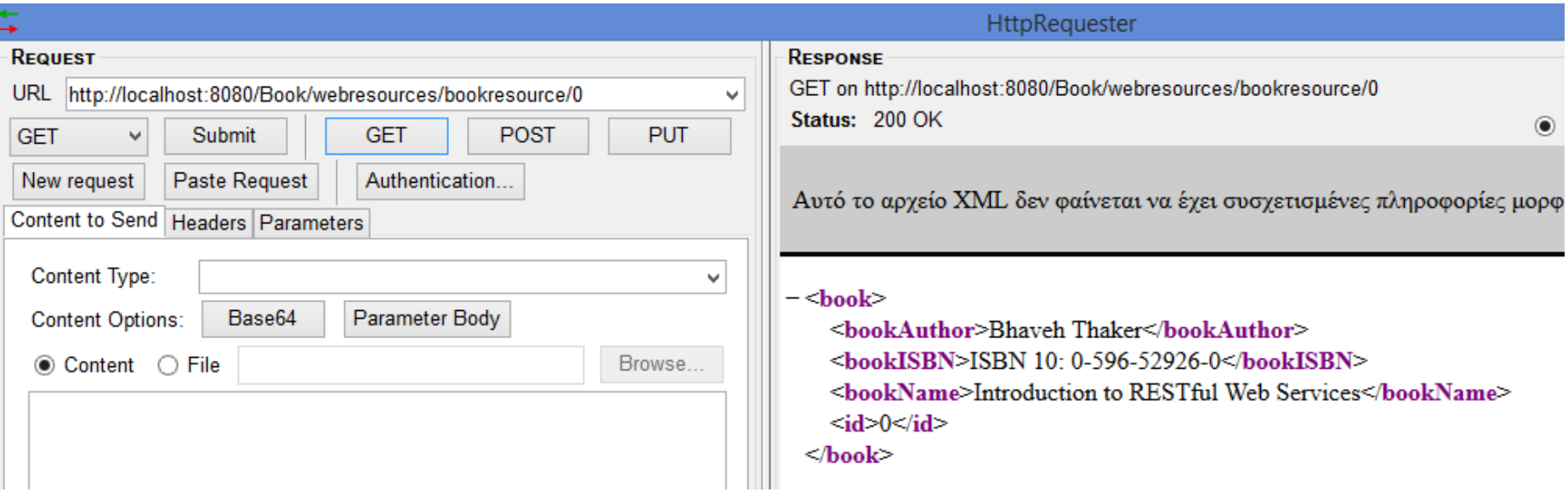
BookResource > Show all Books



Set the url and press get

BookResource > Show Book by id

- http://localhost:8080/Book/webresources/bookresource/0

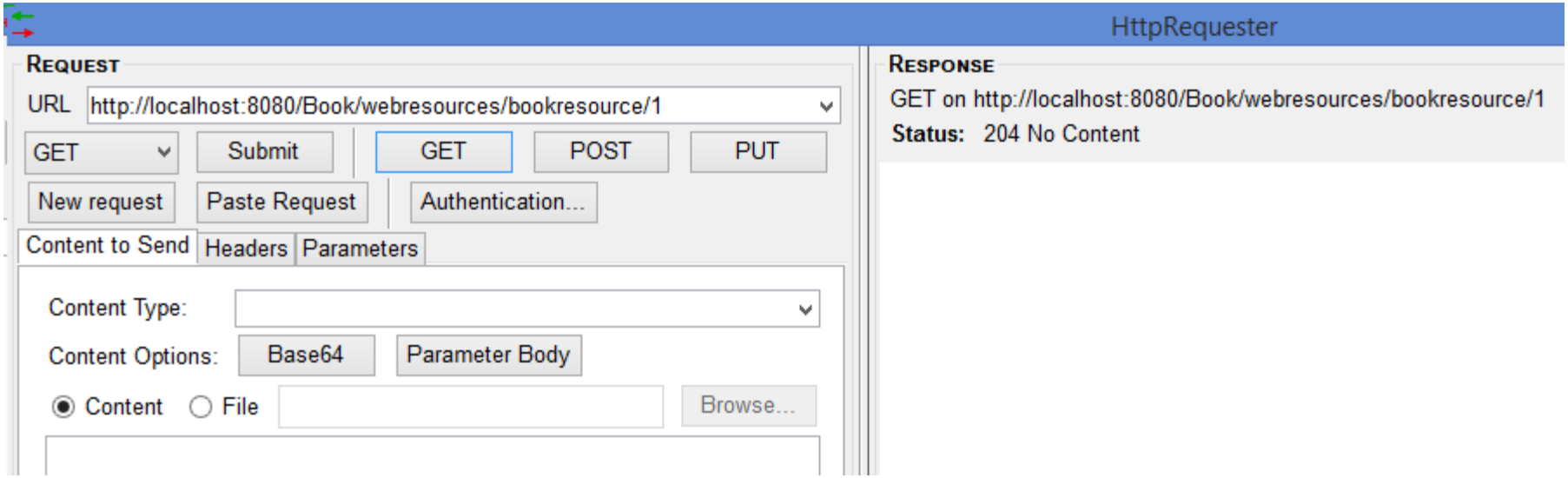


The book is returned ☺

```
@GET
@Path("{id}")
public Book getBook(@PathParam("id") int bookId) {
    return bookMap.get(bookId);
}
```

BookResource > Show Book by id

- http://localhost:8080/Book/webresources/bookresource/1



There is no book with id 1..

BookResource > Add a book

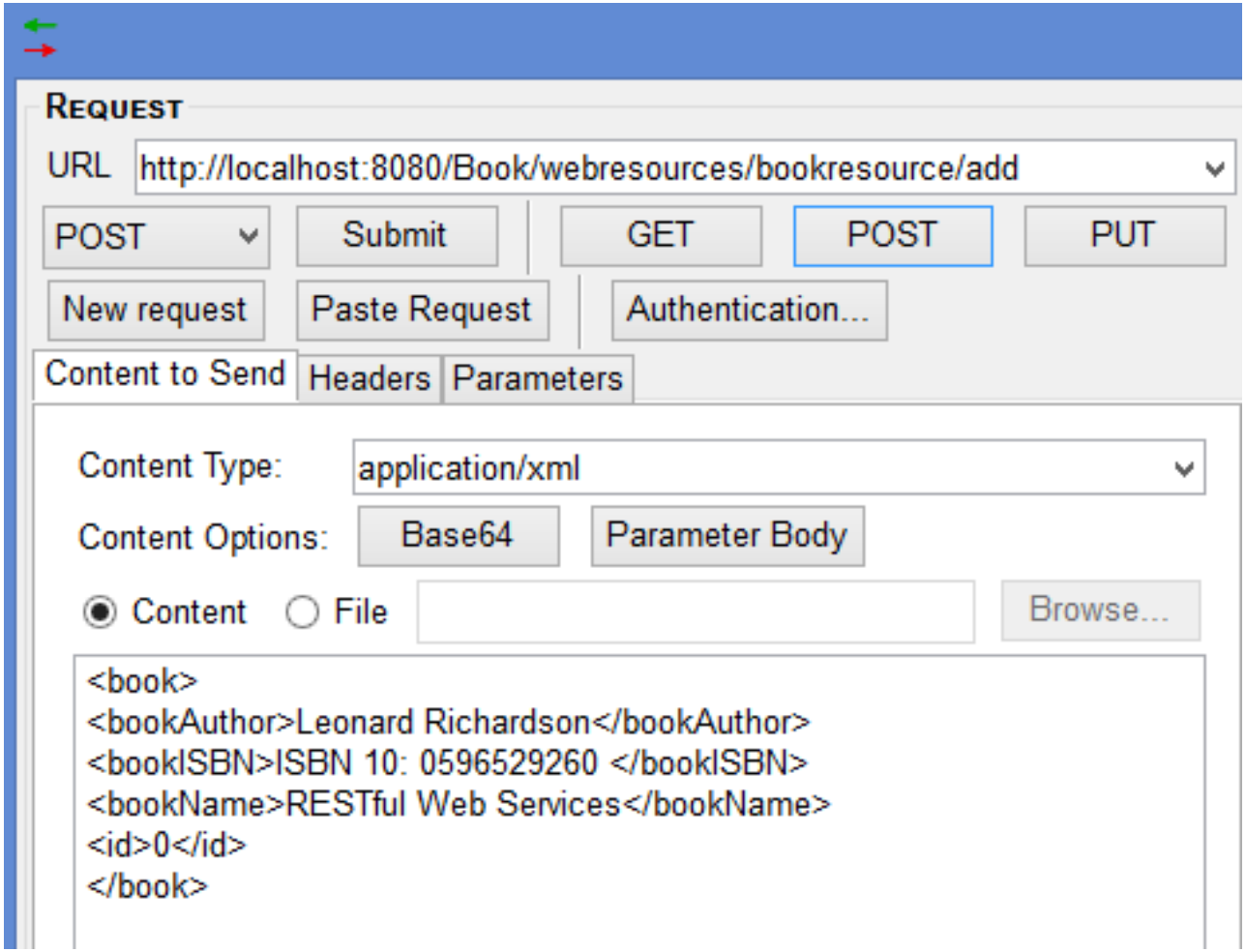
- Suppose we want to add a (2nd) book
- The id is increased according to the size of the structure (map, list or whatever we use…)
- We must make a post request
- Consider we want to send the following book as a request

```
<book>
<bookAuthor>Leonard Richardson</bookAuthor>
<bookISBN>ISBN 10: 0596529260</bookISBN>
<bookName>RESTful Web Services</bookName>
</book>
```

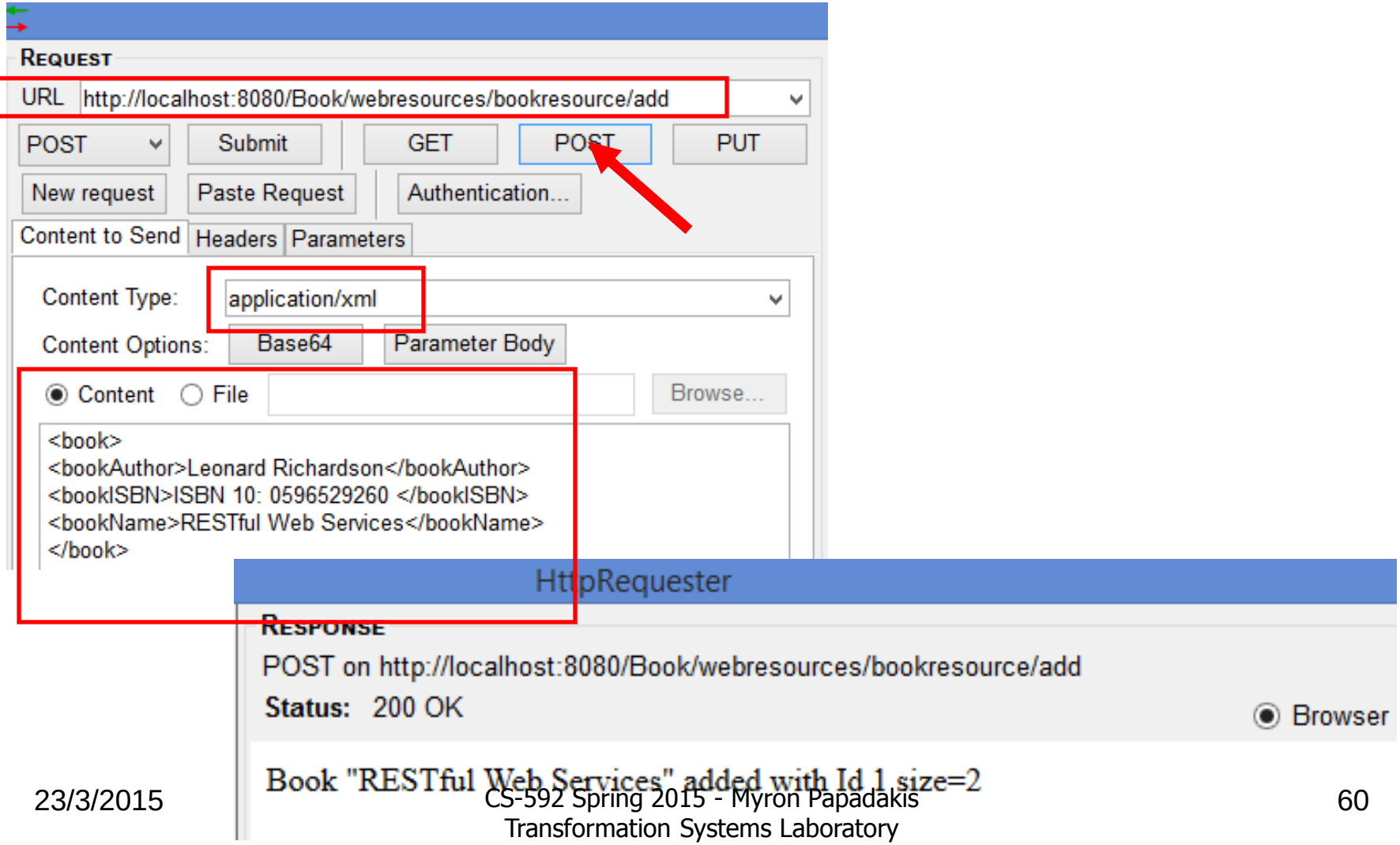
BookResource > Add a book > Code

```
@POST
@Path("add")
public String addBook(Book book) {
    int id = bookMap.size();
    if (book == null) {
        return "Book is null";
    } else {
        book.setId(id);
        bookMap.put(id, book);
        return "Book \"" + book.getBookName() + "\" added with Id "
                + id + " size=" + bookMap.size();
    }

}
```

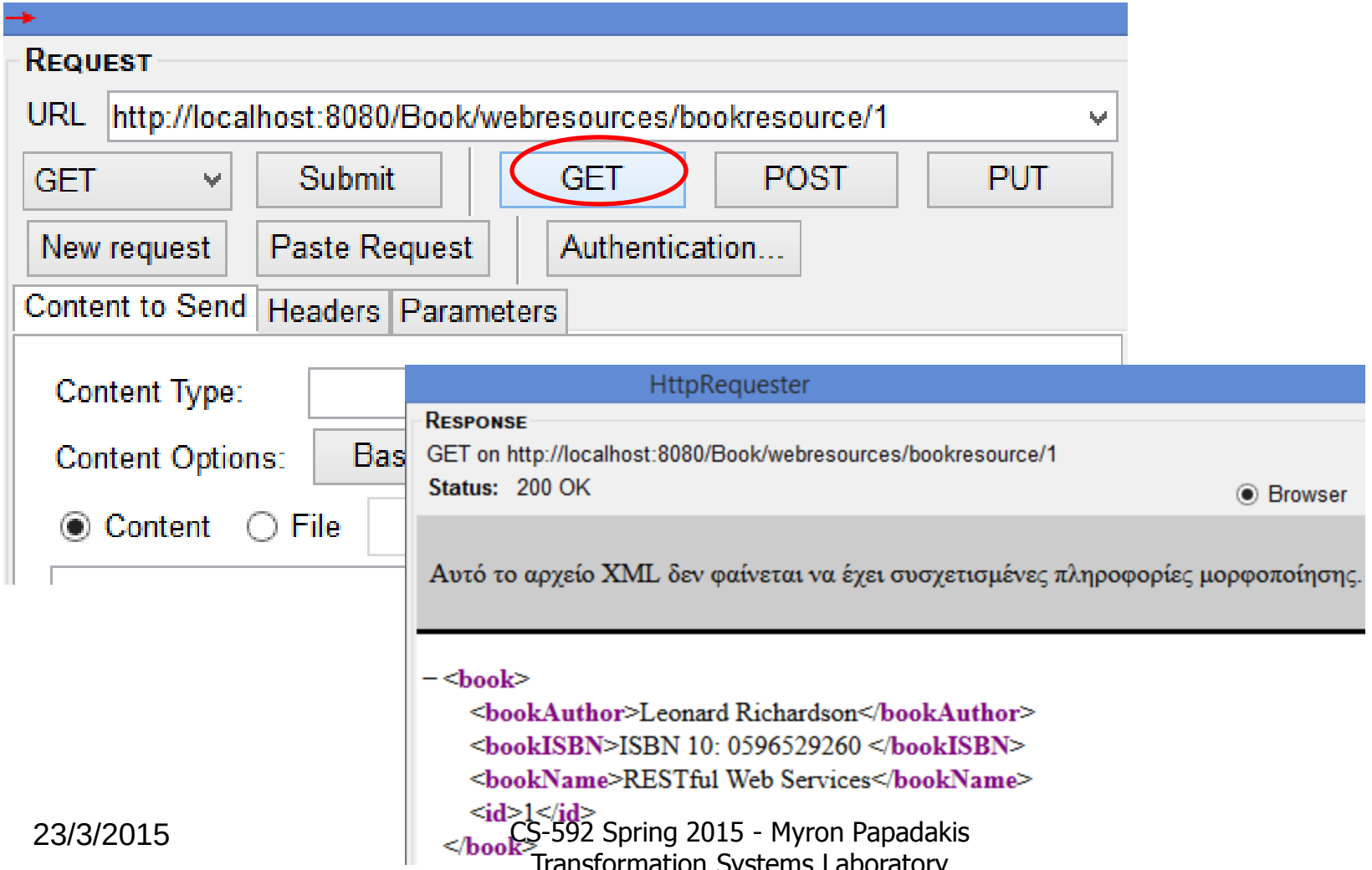
XML Post


Request
URL
http://localhost:8080/Book/webresources/bookresource/add
POST
Submit
GET
POST
PUT
New request
Paste Request
Authentication...
Content to Send
Headers
Parameters
Content Type:
application/xml
Content Options:
Base64
Parameter Body
Content
File
Browse...
<book>
<bookAuthor>Leonard Richardson</bookAuthor>
<bookISBN>ISBN 10: 0596529260 </bookISBN>
<bookName>RESTful Web Services</bookName>
<id>0</id>
</book>

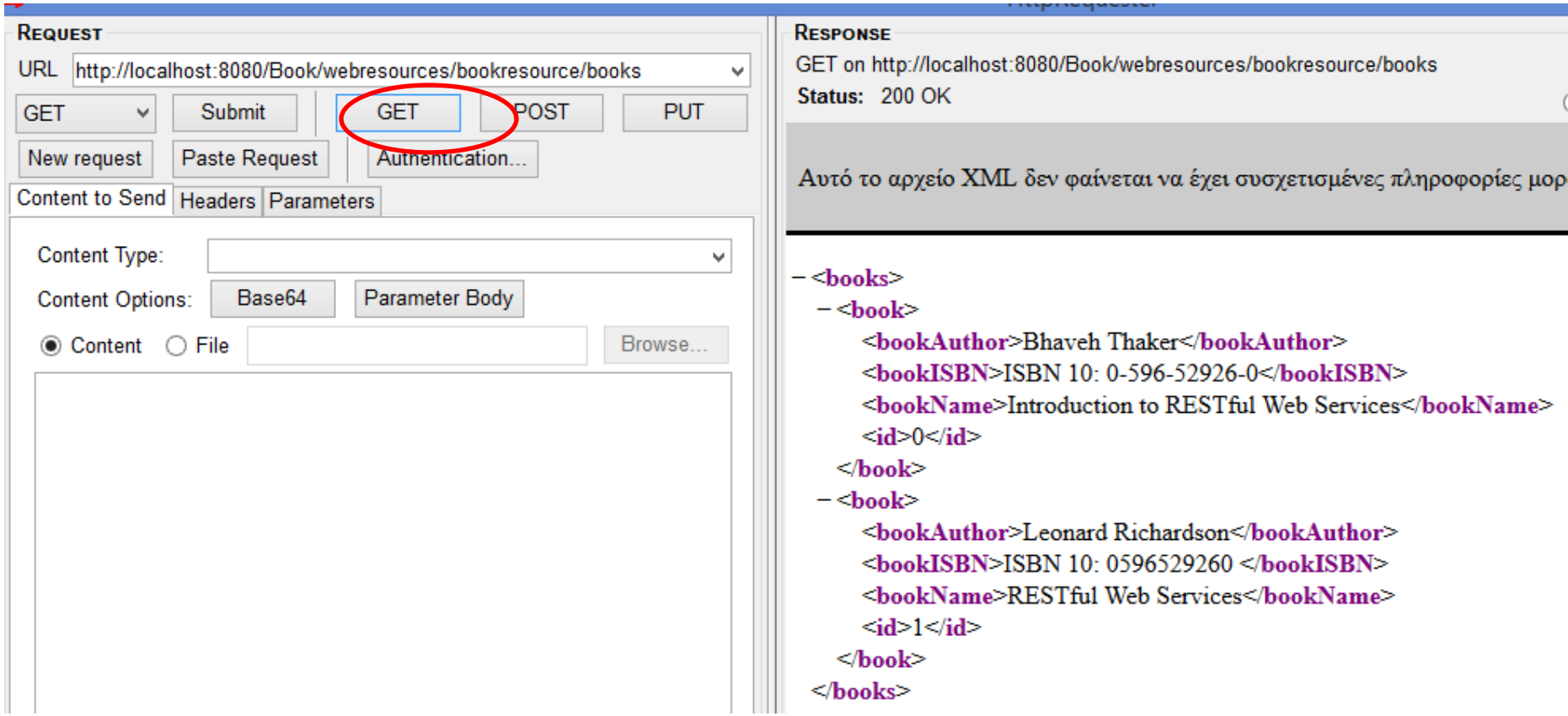
HttpRequester > Post (a book as XML)



HttpRequester > Post (a book as XML) > Verify it is added



HttpRequester > Get all books



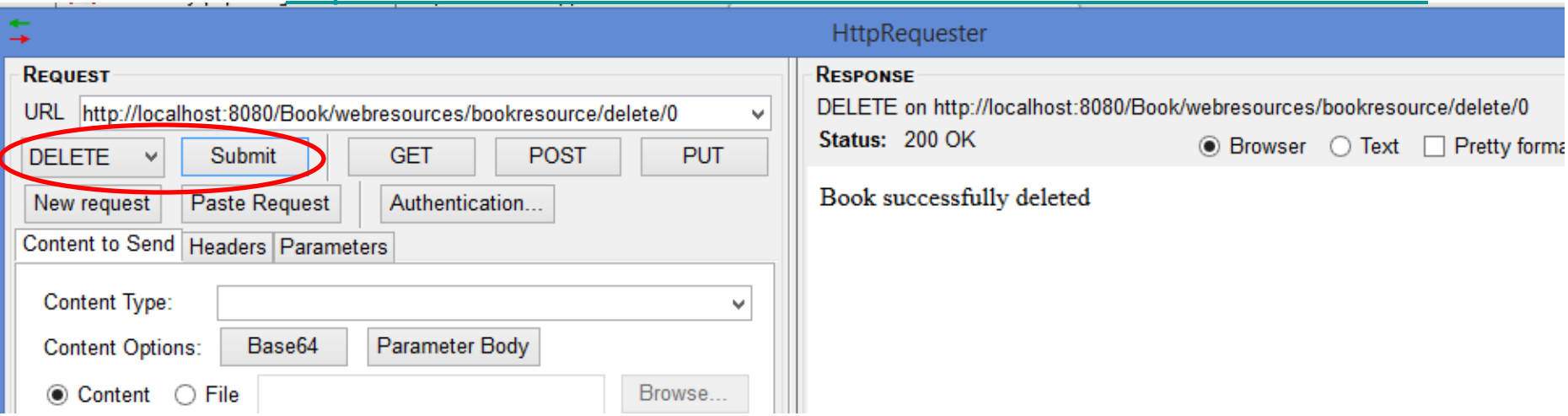
HttpRequester > Delete a book

- If we want to delete a book what should we do?

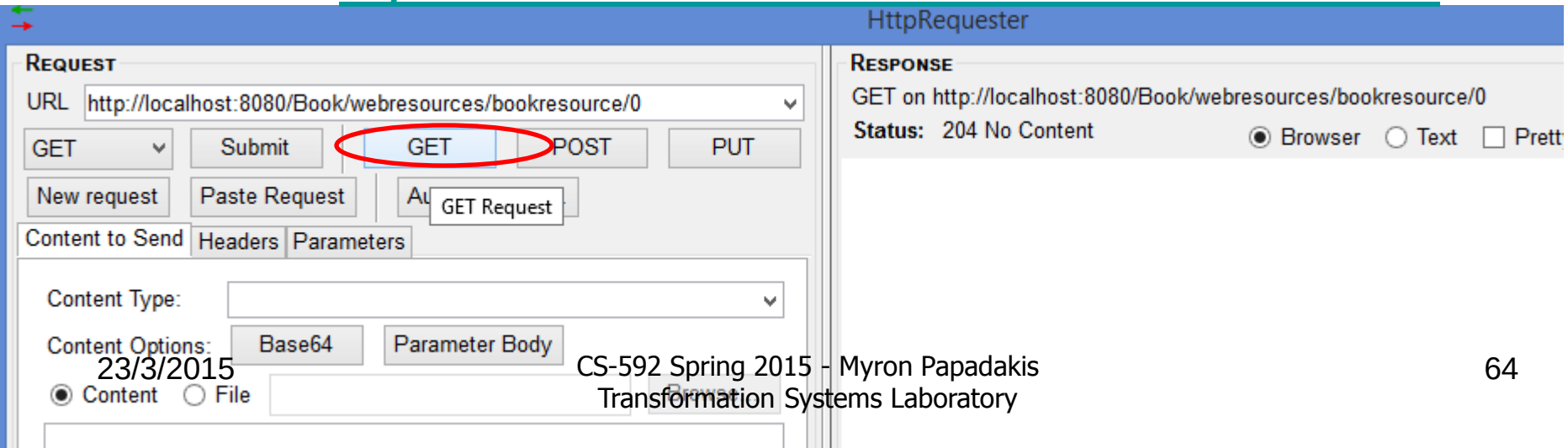
- Add a method for deleting a book.
- Need a parameter for the id of the book
  - @DELETE…
  - Go to the map and locate the book id
  - Delete it

HttpRequester > Delete a book

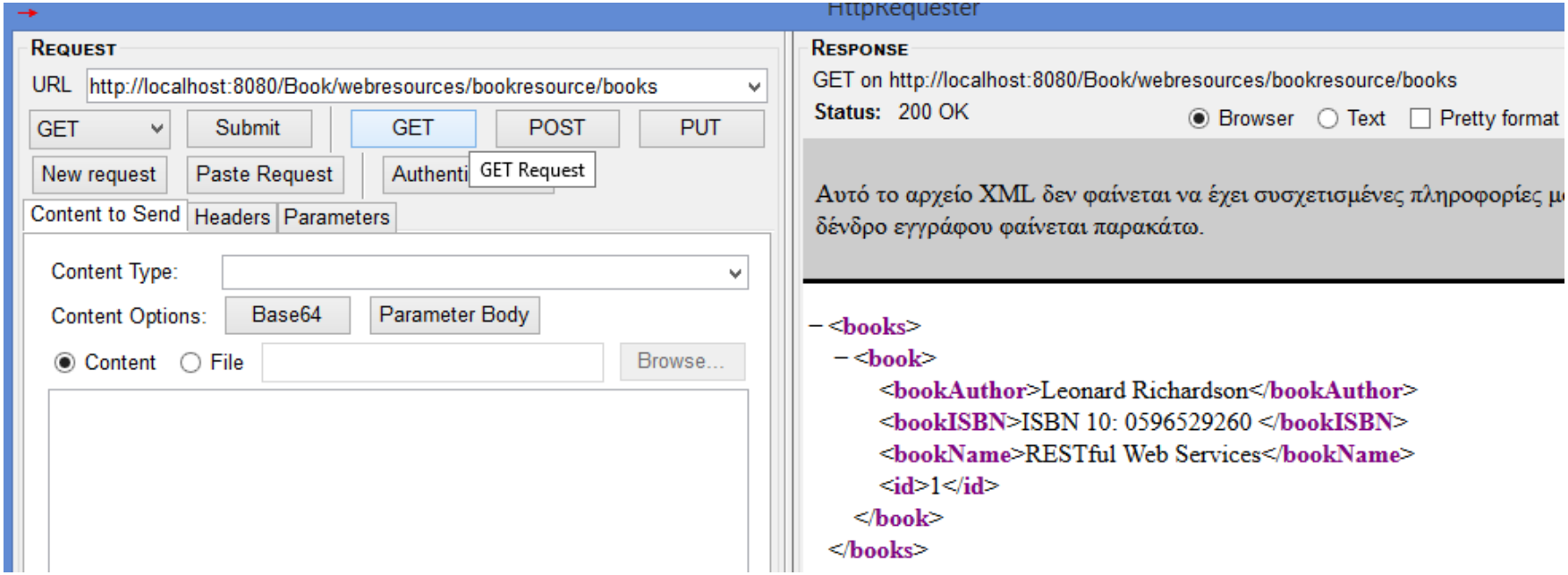
http://localhost:8080/Book/webresources/bookresource/delete/0



http://localhost:8080/Book/webresources/bookresource/0

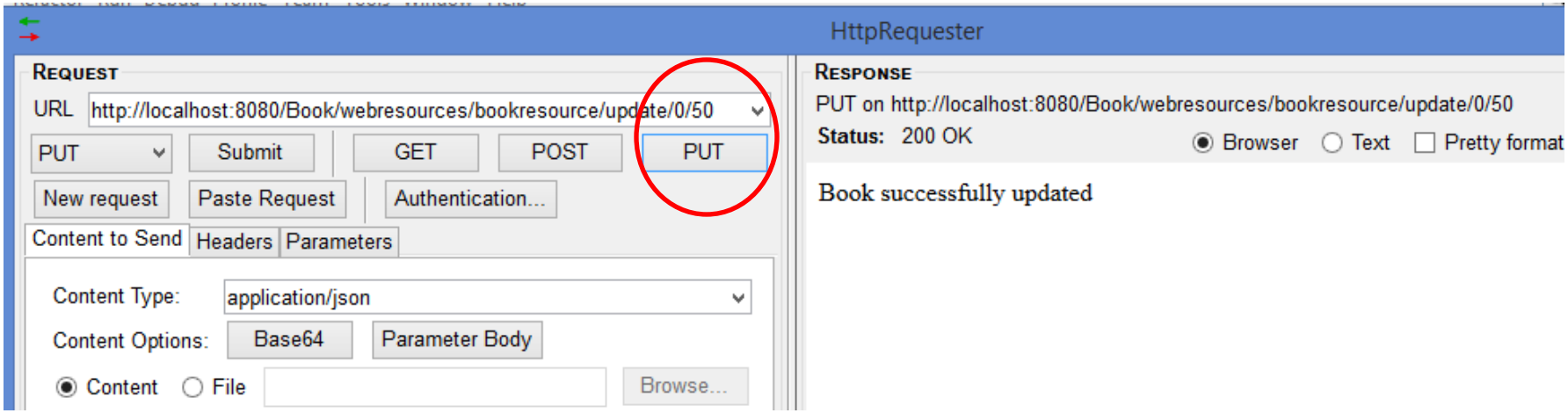


HttpRequester > Delete a book

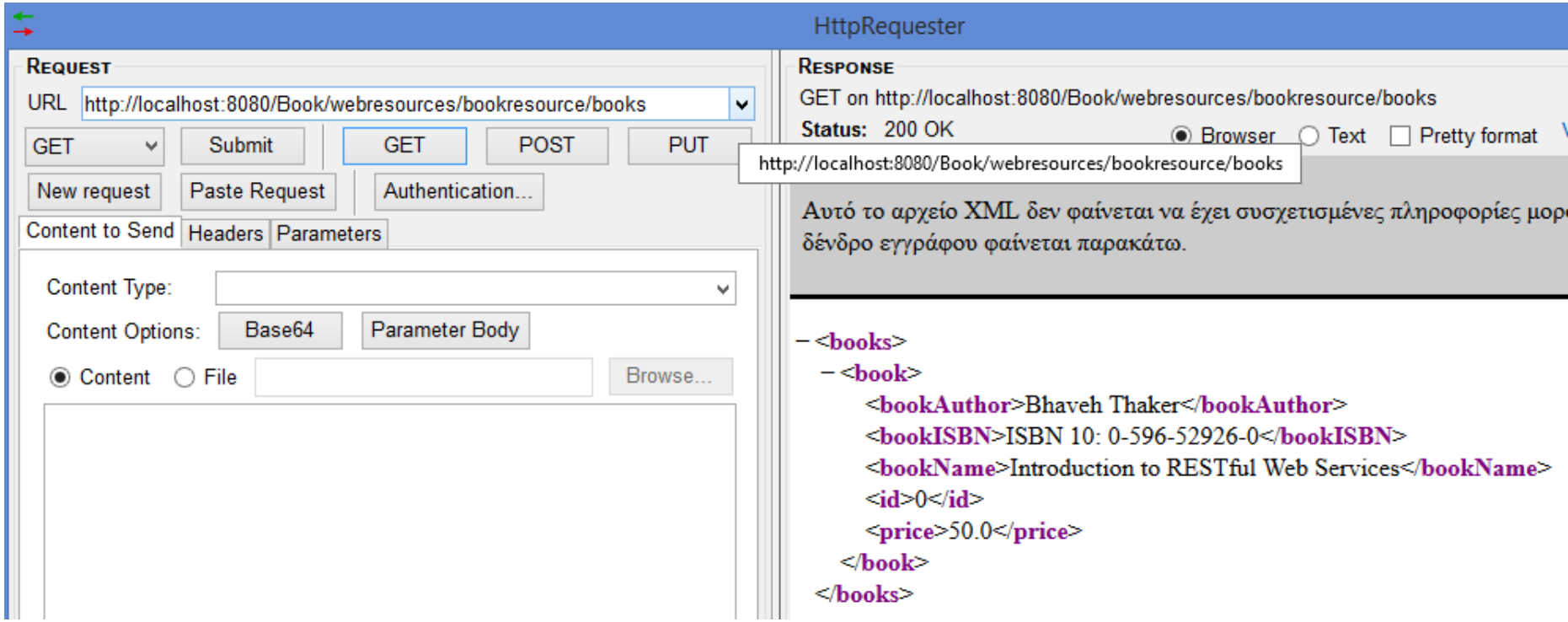


HttpRequester > Update a book

- Need to some code to the resource (@PUT) for updating the book
  - Two parameters (bookId,price)
- http://localhost:8080/Book/webresources/bookresource/update/0/50



Update a book (its price)



Rest Example with JAXB + JSON +JQuery

Similar to Book Example

JAXB *(Just some notes….)*

JAXB Introduction

- NetBeans 6.5 provides support for web services using JAX-WS and JAXB.
- JAX-WS delegates the mapping of the Java language data types to JAXB API.
  - JAXB stands for Java Architecture for XML Binding.
  - JAXB converts XML schemas to Java Content trees and vice versa.
  - Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of Web service

JAXB Introduction

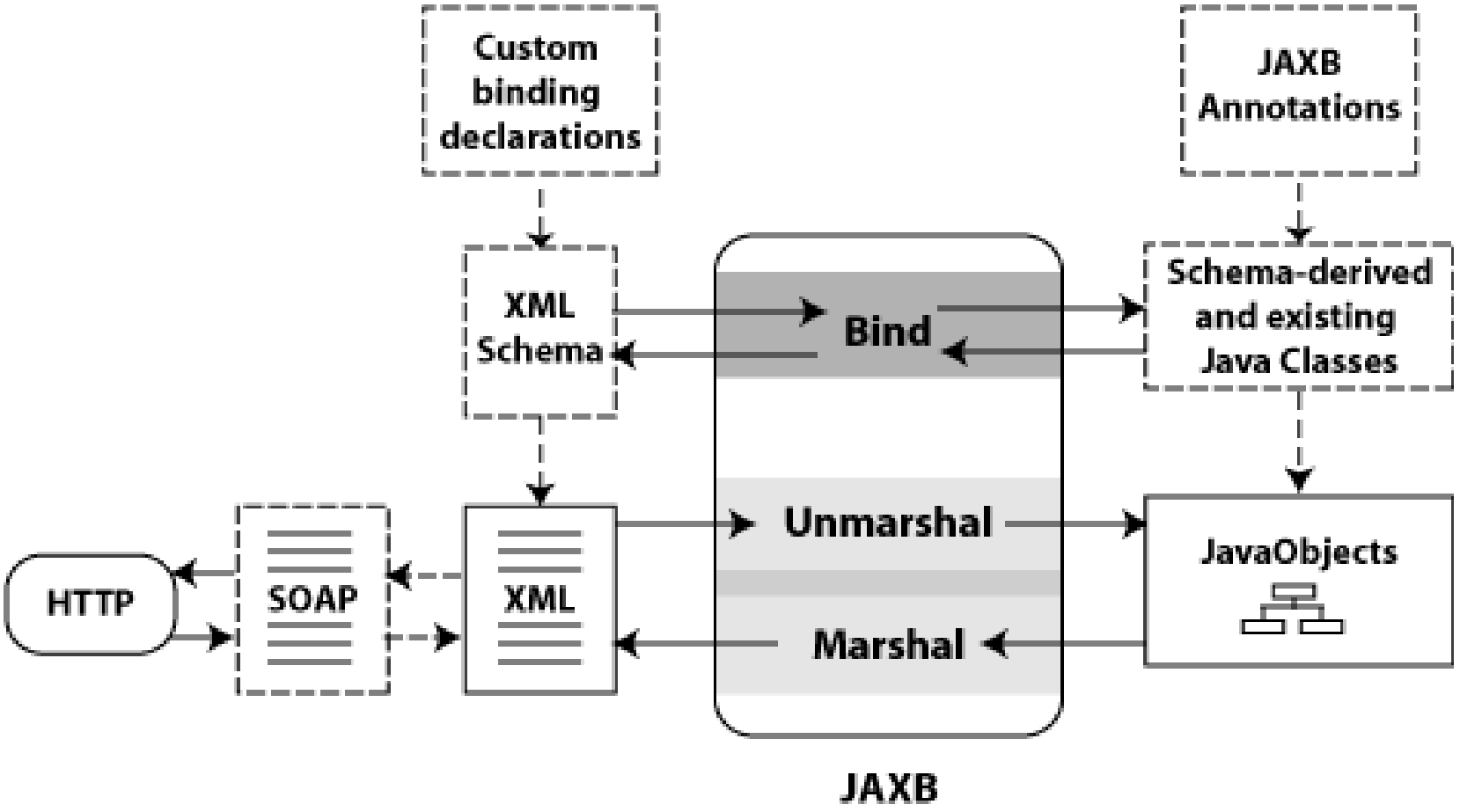
- Basically JAXB is the translator of XML schemas (and data types) to Java and vice versa.
  - Since the WSDL describes the XML schema there is a need to translate that schema to Java.
  - It is the job of JAXB to do just that

- JAXB uses annotations to indicate the central elements.

Overview of Data Binding Using JAXB

- Web Service applications need a way to access data that are in XML format directly from the Java application.
- Specifically, the XML content needs to be converted to a format that is readable by the Java application.
- **Data binding describes the conversion of data between its XML and Java representations.**
- **JAX-WS uses Java Architecture for XML Binding (JAXB) to manage all of the data binding tasks**.
  - Specifically, JAXB binds Java method signatures and WSDL messages and operations and allows you to customize the mapping while automatically handling the runtime conversion.

Data Binding with JAXB



JAXB in Bottom-up (and Top-down)

- **Start from Java**: Using this programming model, you create the Java classes.
  - At run-time, JAXB ***marshals*** the Java objects to generate the XML content which is then packaged in a SOAP message and sent as a Web Service request or response.

- **Start from WSDL**: Using this programming model, the XML Schemas exist and JAXB *unmarshals* the XML document to generate the Java objects.

JAXB Architecture

XML Schema Type	Java Data Type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd.long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float

JAXB vs. DOM and SAX

- JAXB is not part of the standard Java distribution, but is available from Sun as part of the Java Web Services Developer Pack (Java WSDP)

- JAXB is a higher level construct than DOM or SAX
  - DOM represents XML documents as generic trees
  - SAX represents XML documents as generic event streams
  - JAXB represents XML documents as Java classes with properties that are specific to the particular XML document
    - E.g. book.xml becomes Book.java with getTitle, setTitle, etc.

- JAXB thus requires almost no knowledge of XML to be able to programmatically process XML documents!

Is JAXB enough?

- Marshall and unmarshall the XML file

TodoMap class

```
package todopackage;


import java.util.TreeMap;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement (name="todos")
@XmlAccessorType(XmlAccessType.FIELD)
public class TodoMap {

    private TreeMap<Integer, Todo> todoMap = new TreeMap<Integer, Todo>();

    public TreeMap<Integer, Todo> getTodoMap() {
        return todoMap;
    }

    public void setTodoMap(TreeMap<Integer, Todo> todoMap) {
        this.todoMap = todoMap;
    }
}
```

JAXB marshall and unmarshall

```
public class Util {
    static String filename ="C:/todoDB/todo.xml";

    public static void marshall(TodoMap todoMap, String filename) {
        try {
            JAXBContext jaxbContext = JAXBContext.newInstance(TodoMap.class);
            Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
            jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            jaxbMarshaller.marshal(todoMap, new File(filename));
        } catch (JAXBException ex) {
            Logger.getLogger(Util.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    public static TodoMap unMarshal(String filename) {
        try {
            JAXBContext jaxbContext = JAXBContext.newInstance(TodoMap.class);
            Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
            TodoMap tMap = (TodoMap) jaxbUnmarshaller.unmarshal( new File(filename) );
            return tMap;
        } catch (JAXBException ex) {
            Logger.getLogger(Util.class.getName()).log(Level.SEVERE, null, ex);
        }
        return null;
    }
}
```

TodoResource Class > Marshall or Unmarshall?

```
@GET
@Path("todos")
@Produces({MediaType.APPLICATION_JSON})
public List<Todo> getTodos() {...5 lines }

@GET
@Produces({MediaType.APPLICATION_JSON,MediaType.APPLICATION_XML})
@Path("{todoId}")
public Todo getTodo(@PathParam("todoId") int todoId) {...3 lines }

//Here new todos are replaced because of id
@POST
@Consumes({MediaType.APPLICATION_JSON})
@Path("add")
public Response addTodo(Todo todoObj) {...18 lines }

@DELETE
@Path("delete/{delId}")
public String deleteTodo(@PathParam("delId") int todoId) {...8 lines }

@PUT
@Path("update/{todoId}/{date}")
public String updateTodo(@PathParam("todoId") int todoId,
        @PathParam("date") String date) {...9 lines }
```

TODOResource

```
@Path("todo")
@Singleton
public class TODOResource {

    private TodoMap todoMap = null;

    public TODOResource() {
        File f = new File(Util.filename);
        TreeMap<Integer, Todo> map = null;//moved here saturday
        if (!f.exists()) {
            todoMap = new TodoMap();
            map = new TreeMap<Integer, Todo>();
            todoMap.setTodoMap(map);
            createDummyTODO();
        } else {
            //Otherwise the map can be loaded in the memory
            todoMap = Util.unMarshalingExample(Util.filename);
        }

    }
```

TODOResource

```
private int getFreeSlot() {
    if (todoMap.getTodoMap().isEmpty()) {
        return 1;
    } else {
        return todoMap.getTodoMap().lastKey() + 1;
    }
}

private void createDummyTODO() {
    Todo t = new Todo();
    t.setDescription("I have to make this assignment");
    t.setDate("Today at 11:00");
    t.setId(getFreeSlot());
    t.setTitle("CS-592 Assignment1");
    addTodo(t);
}

@GET
@Path("todos")
@Produces({MediaType.APPLICATION_JSON})
public List<Todo> getTodos() {
    List<Todo> todosList = new ArrayList<Todo>();
    todosList.addAll(todoMap.getTodoMap().values());
    return todosList;
}
```

TODOResource

```
@GET
@Produces({MediaType.APPLICATION_JSON,MediaType.APPLICATION_XML})
@Path("{todoId}")
public Todo getTodo(@PathParam("todoId") int todoId) {
    return todoMap.getTodoMap().get(todoId);
}

private boolean containsTodo(Todo todoObj) {
    String title = todoObj.getTitle();
    Collection<Todo> todos = todoMap.getTodoMap().values();
    for (Todo value : todos) {
        if (title.equals(value.getTitle())) {
            return true;
        }
    }
    return false;
}
```

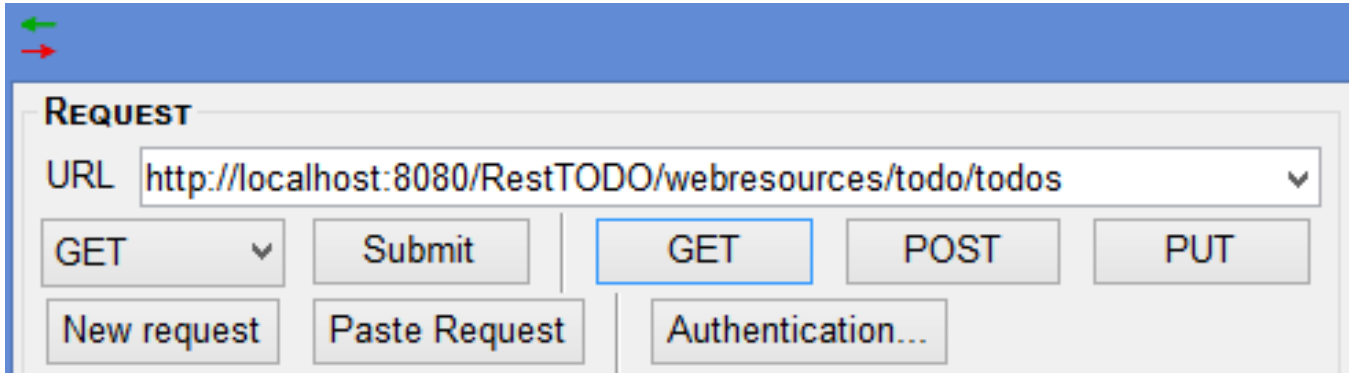
TODOResource > addTodo

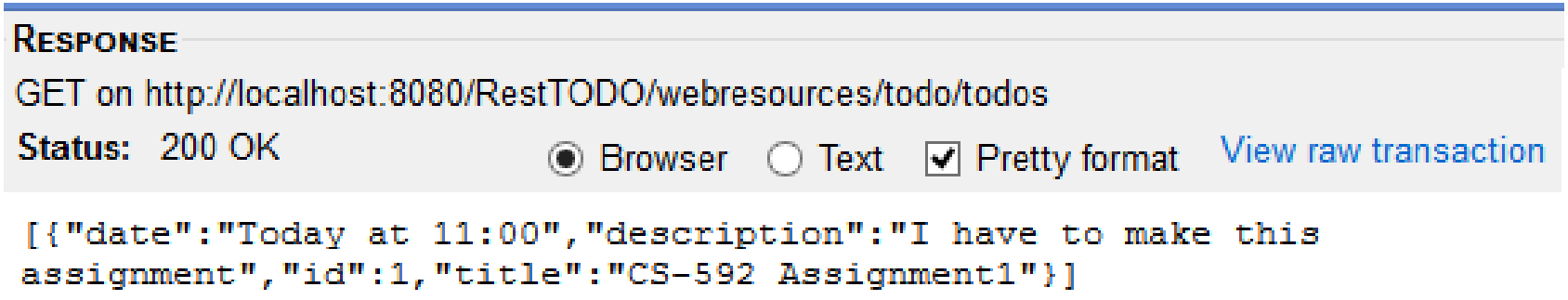
```
@POST
@Consumes({MediaType.APPLICATION_JSON})
@Path("add")
public Response addTodo(Todo todoObj) {
    int id = getFreeSlot();
    String result = null;
    if (todoObj == null) {
        result = "You sent no content. Please post a correct TODO (in json format).";
        return Response.status(Response.Status.NO_CONTENT).entity(result).build();
    } else if (todoMap.getTodoMap().containsKey(id)) {//containsTodo(todoObj)){//
        result = "This todo already exists";
        return Response.status(400).entity(result).build();
    } else {
        todoObj.setId(id);
        todoMap.getTodoMap().put(id, todoObj);
        result = "todoObj \"" + todoObj.getDescription() + "\" was created with Id "
                + todoObj.getId();
        Util.marshall(todoMap, Util.filename);

        return Response.status(Response.Status.CREATED).entity(result).build();
    }
}
```

TODOResource > addTodo

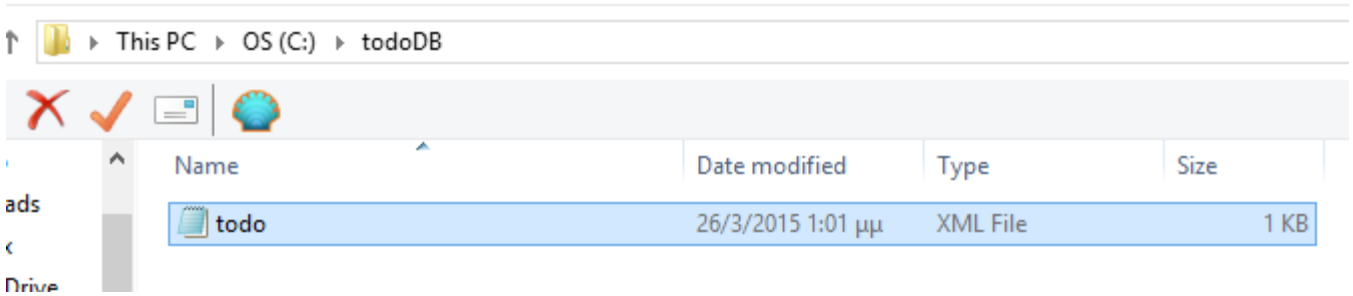
```
@POST
@Path("add/{title}/{date}/{description}")
public String addTodo(
        @PathParam("title") String title, @PathParam("date") String date1,
        @PathParam("description") String description1) {
    int id = getFreeSlot();
    if (todoMap.getTodoMap().containsKey(id)) {
        return "This todo already exists";
    } else {
        Todo todoObj = new Todo();
        todoObj.setId(id);
        todoObj.setTitle(title);
        todoObj.setDescription(description1);
        todoObj.setDate(date1);
        todoMap.getTodoMap().put(id, todoObj);
        Util.marshall(todoMap, Util.filename);

        return "todoObj \"" + todoObj.getDescription() + "\" added with Id "
                + id;
    }
}
```

First run


Request
URL
http://localhost:8080/RestTODO/webresources/todo/todos
GET
Submit
GET
POST
PUT
New request
Paste Request
Authentication...

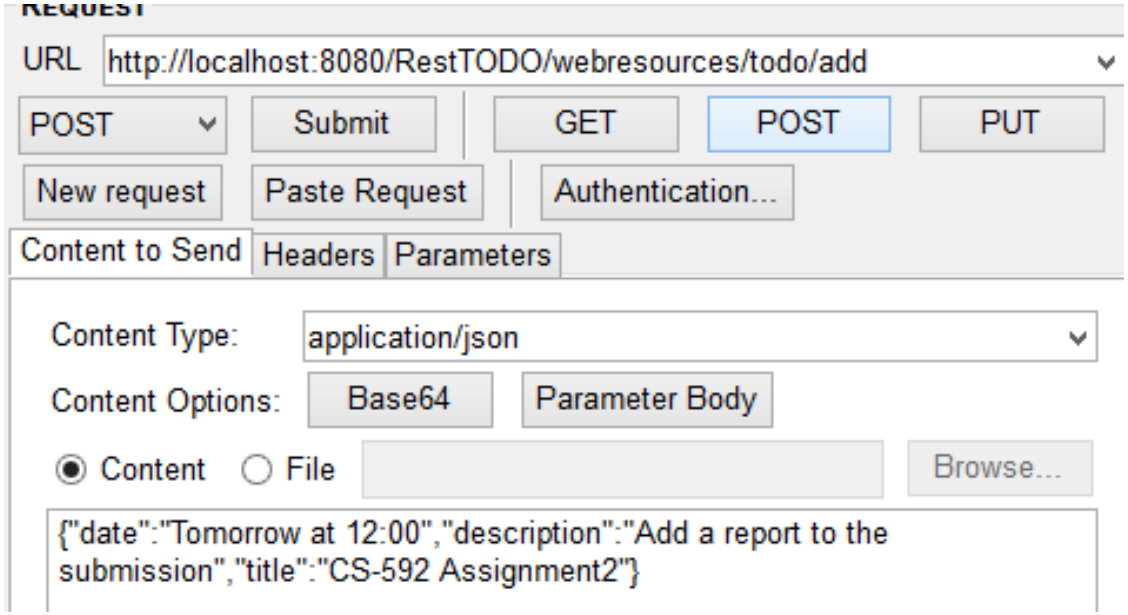

Response
GET on http://localhost:8080/RestTODO/webresources/todo/todos
Status: 200 OK
Browser
Text
Pretty format
View raw transaction
[{"date":"Today at 11:00","description":"I have to make this assignment","id":1,"title":"CS-592 Assignment1"}]

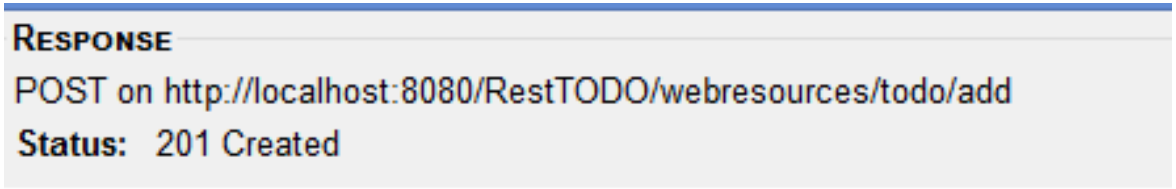
Directory & File Creation




```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<todos>
    <todoMap>
        <entry>
            <key>1</key>
            <value>
                <date>Today at 11:00</date>
                <description>I have to make this assignment</description>
                <id>1</id>
                <title>CS-592 Assignment1</title>
            </value>
        </entry>
    </todoMap>
</todos>
```

Add a Todo

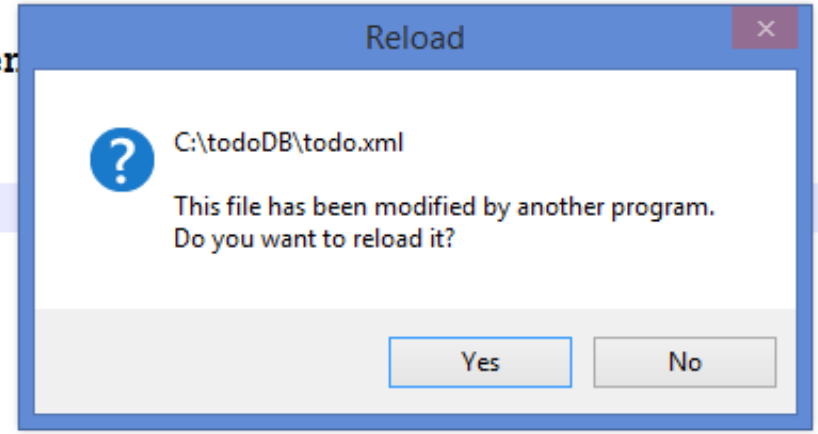




todoObj "Add a report to the submission" was created with Id 2

Add a Todo > XML is updated

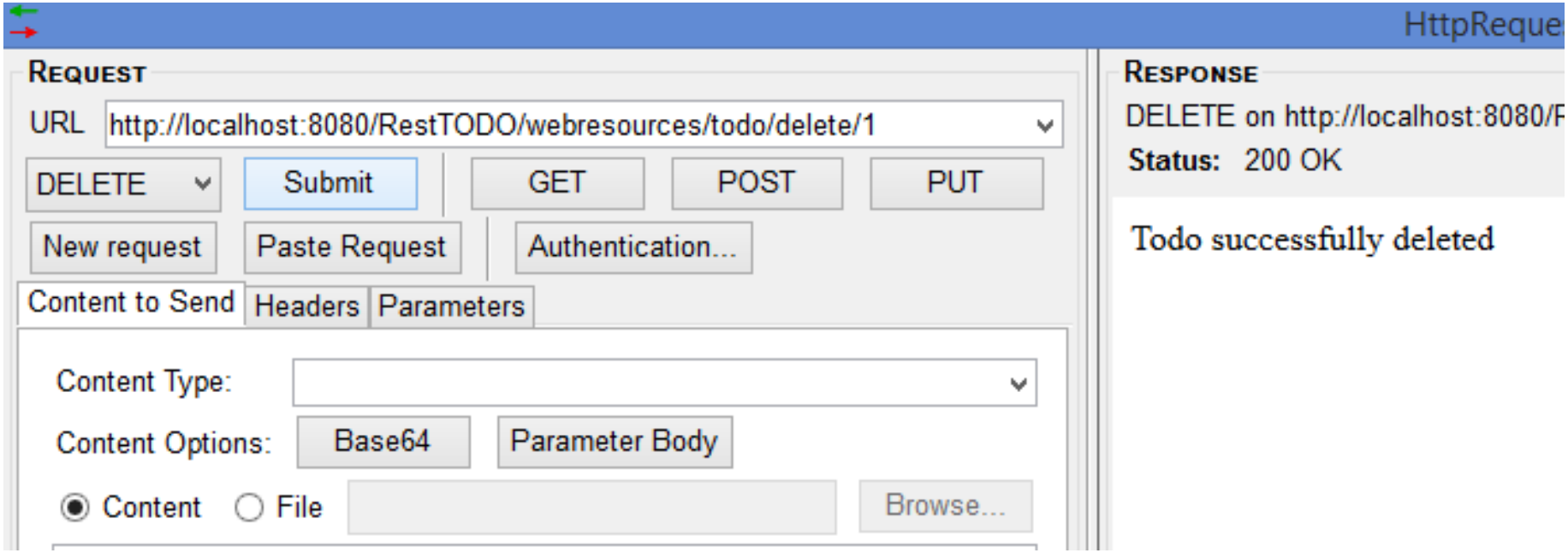
```
?xml version="1.0" encoding="UTF-8" standalone="yes"?>
todos>
    <todoMap>
        <entry>
            <key>1</key>
            <value>
                <date>Today at 11:00</date>
                <description>I have to make this assignment</description>
                <id>1</id>
                <title>CS-592 Assignmer
            </value>
        </entry>
    </todoMap>
/todos>
```



Add a Todo > XML is updated

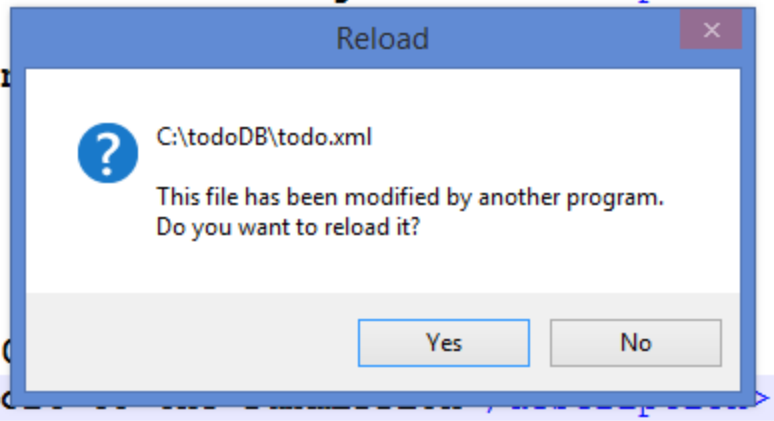
```
<todos>
    <todoMap>
        <entry>
            <key>1</key>
            <value>
                <date>Today at 11:00</date>
                <description>I have to make this assignment</description>
                <id>1</id>
                <title>CS-592 Assignment1</title>
            </value>
        </entry>
        <entry>
            <key>2</key>
            <value>
                <date>Tomorrow at 12:00</date>
                <description>Add a report to the submission</description>
                <id>2</id>
                <title>CS-592 Assignment2</title>
            </value>
        </entry>
    </todoMap>
</todos>
```

Delete a Todo



Delete a Todo

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<todos>
    <todoMap>
        <entry>
            <key>1</key>
            <value>
                <date>Today at 11:00</date>
                <description>I have to make this assignment</description>
                <id>1</id>
                <title>CS-592 Assignmen
            </value>
        </entry>
        <entry>
            <key>2</key>
            <value>
                <date>Tomorrow at 12:0
                <description>Add a repo
                <id>2</id>
                <title>CS-592 Assignment2</title>
            </value>
        </entry>
    </todoMap>
</todos>
```



Delete a Todo > Updated XML file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<todos>
    <todoMap>
        <entry>
            <key>2</key>
            <value>
                <date>Tomorrow at 12:00</date>
                <description>Add a report to the submission</description>
                <id>2</id>
                <title>CS-592 Assignment2</title>
            </value>
        </entry>
    </todoMap>
</todos>
```

Τέλος Ενότητας

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[1] http://creativecommons.org/licenses/by-nc-nd/4.0/

•Ως **Μη Εμπορική** ορίζεται η χρήση:

–που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο

–που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο

–που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

•Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

.

Σημείωμα Αναφοράς