UNIVERSIDADE D
COIMBRA

Sistemas Distribuídos 2024/2025
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Exercises 5 - RESTful services integration

1. Introduction

Representational State Transfer (REST) web services are used for making certain resources available to external systems. In this demo, you will understand the inner workings of REST and how to use REST APIs to incorporate external services in your application. The concept of REST services is based on the Hypertext Transfer Protocol (HTTP) protocol and its semantics as a means to share resources across the web. Each REST expression has a Verb and a Uniform Resource Identifier (URI). Although there are many verbs, such as HEAD or OPTIONS, the more interesting ones are GET, POST, PUT and DELETE. Your client interacts with other servers by using these verbs just as a browser would do.

2. Getting started

Due to the lack of a standard for strict output format (as Web Service Description Language (WSDL) works for WebServices) in REST services, the first step before using an API is to read its documentation and manually explore it. To help in the exploration phase there are many tools available, such as curl and rest- client, which can be installed and used from your PC. Below we will describe the usage of Postman which is the suggested tool to be used throughout this demo. Postman can be obtained from the following link https://www.postman.com/downloads/. The first step that should be performed after launching Postman is to open the Console, which allows you to see the raw HTTP requests and responses that are exchanged.

2.1. Use Postman to perform requests using the verbs and URIs defined below. Pay attention to the sent and received HTTP messages.

Verb	URI
POST	http://tinyurl.com/api-create.php?url=http://www.dei.uc.pt
GET	http://firebrowse.org/api/v1/Archives/StandardData?format=csv
GET	http://api.oceandrivers.com/v1.0/getEasyWind/EW013?period=latestdata
GET	https://blockchain.info/ticker

2.2. Which content types have you seen used in the responses?

3. Online Painting

Now that you have been introduced to the basics of REST APIs, it is time to solve a more advanced task. You are to use a simple REST API that allows the creation, modification and destruction of colored tiles in an online board. You can find a live representation of the board at http://ucx.dei.uc.pt/game and the web service is available at http://ucx.dei.uc.pt/game/board. Whenever you perform a change to the table, the result will be shown on the webpage.

3.1 Doing it by hand

The first step is to add a new tile in a color of your choice. For that purpose, we will use an endpoint available at the URI /board/{x}/{y} and use the POST verb. The values of {x} and {y} in the URI should be replaced with positive integer numbers that represent the location of the tile that you want to add. This endpoint receives also 3 parameters in its body, which are r, g and b as the components of the color.

```
POST /game/board/4/10/ HTTP/1.1
User-Agent: PostmanRuntime/11.39.2
Accept: */*
Cache-Control: no-cache
Host: localhost
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/w-www-form-urlencoded
Content-Length: 15
r=255&g=255&b=255
HTTP/1.1 201 Created
Date: Sun, 06 Apr 2025 06:49:32 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Keep-Alive: timeout=20
Connection: keep-alive
{"result":true}
```

Take a look at the status code in the response, 201, which means that the resource was created, and the content type of the response which was a JavaScript Object

Notation (JSON). The table below contains a general overview of the various status code groups.

Range	Meaning
2xx	Successful request
3xx	Redirect
4xx	Client-side error
5xx	Server-side error

After we have added our first tile to the board we can use the /board endpoint with the GET verb to obtain information about all the tiles in the board. You can use the /board/{x}/{y} endpoint if you only want to obtain information about a single tile.

```
GET/game/board HTTP/1.1
User-Agent: PostmanRuntime/11.39.2
Accept: */*
Cache-Control: no-cache
Host: localhost
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
HTTP/1.1 200 OK
Date: Sun, 06 Apr 2025 08:32:11 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Keep-Alive: timeout=20
Connection: keep-alive
{"board":[
    [
        {"r":255,"g":255,"b":255,"x":0,"y":0,"created": false},
        {"r":255,"g":255,"b":255,"x":1,"y":0,"created": false},
        {"r":255,"g":255,"b":255,"x":2,"y":0,"created": false},
        {"r":255,"g":255,"b":255,"x":3,"y":0,"created": false},
        {"r":255,"g":255,"b":255,"x":4,"y":0,"created": false},
        {"r":255,"g":255,"b":255,"x":5,"y":0,"created": false},
        (...)
    ]
]}
```

Since we already added a tile to a specific x and y position, if we try to add it again to the same place an error will be shown (go ahead, try it!). In this situation, we can either edit the color of that tile by calling /board/{x}/{y} using the PUT verb or delete the tile completely by using the DELETE verb on the same URI.

3.1.1 What is the purpose of using different verbs with the same endpoint? What are the most common verbs used with REST services and what do they mean?

3.2 Doing it programmatically

In the examples folder (python and java respectively), you will find the PicassoBot. It is a command-line interface to the online API that allows the user to draw basic shapes and which contains the following commands:

- **tiles**: Lists the positions and colors of all tiles that are not empty (have been created).
- **dot x y r g b**: Paints a dot in position x, y with the color r, g, b.
- **line x y dir len r g b**: Draws a line of length len starting in point (x, y), with direction (dir) that can be right, left, up, down and color in r, g, b.

Below you have 2 sections, one for Java and another for Python, you can follow one of the two at your choice.

3.2.1 Java

We start by defining the location of our server. In this case we have set it to a local instance of the game running.

```
// Defines the server address
static String serverAddress = "http://localhost:8080";
```

For the purpose of our class exercise you should use the following server: http://ucx.dei.uc.pt. Update the code accordingly.

The tiles command calls the print method that, as you might have guessed by now, does a GET HTTP request. These lines prepare the Connection that will be used to perform the communication between client and server.

```
URL url = URI.create(serverAddress + "/game/board").toURL();
HttpURLConnection connection =
    (HttpURLConnection) url.openConnection();
(...)
```

This line sets the Request Method to GET. Notice that the same URL may have different behaviors depending on the verb, so you should always be careful to set this accordingly.

```
connection.setRequestMethod("GET");
```

APIs reply errors in the 300-399 range if the page is no longer there, or if it was moved inside or between servers. You might want to know what kind of redirect was performed (permanent or temporary), and therefore the Client should not follow automatically the redirects.

```
connection.setInstanceFollowRedirects(false);
```

Resources might be available in different formats. Setting the accept header will ask the server for a particular type. In this case, we are using "application/xml", but if you wanted, you could ask for "application/json" and then use a JSON library to parse the response.

```
connection.setRequestProperty("Accept", "application/xml");
```

Some APIs use the User-agent to limit the content or provide slightly different features. It is also important to distinguish your REST client from browsers such as Firefox, Chrome and others that will have a different kind of output. Some APIs use the User-agent to know if you are using a mobile device to compress data even more (to reduce client expenses with a traffic plan).

```
connection.setRequestProperty("User-agent", "Pablo v1");
```

Since status codes in the 200-299 range are okay, but for anything else we want to debug the issue.

```
// If we get a Redirect or an Error (3xx, 4xx and 5xx)
if (connection.getResponseCode() >= 300) {
    // We want more information about what went wrong.
    debug(connection);
}
```

Note that if there is an error, the content will be available through the method getErrorStream() instead of the regular getInputStream(). This detail can be viewed in the debug function.

```
connection.getErrorStream();
```

Since we asked for XML, we need to parse that format and retrieve the contents we are looking for. The code below prepares the Document as an input source and creates a new XPath instance. XPath is a way of querying XML documents in a sense that asking for "/board/row/item" will return all elements inside each that is inside the . In practical terms, each corresponds to a cell in the board.

```
// Response body from InputStream.
InputSource inputSource =
    new InputSource(connection.getInputStream());

// XPath is a way of reading XML files.
XPathFactory factory = XPathFactory.newInstance();
XPath xPath = factory.newXPath();

// here we are querying the document (much like SQL)
// for all the item tags
// inside row elements.
NodeList nodes = (NodeList) xPath.evaluate(
        "/board/row/item",
        inputSource,
```

```
        XPathConstants.NODESET
    );
// The last argument defines the type of result we are looking for.
// Might be NODESET for a list of Nodes or NODE for a single node.
```

The evaluate method will return the result of the "/board/row/item" query on input- Source. The last parameter says that we are looking for more than one Node. If you knew beforehand that you only wanted one, you would use XPathConstants.NODE and cast it to Node.

As always, it is good to close system resources that are not used anymore. Notice that we have read when parsing the NodeList and not before that.

```
// We don't need the connection anymore once we get the nodes.
connection.disconnect();
```

Finally, we iterate through the nodes and retrieve the available attributes. We begin by checking the 'created' attribute to verify if that cell has a tile, and then we extract the color and position information. POST, PUT and DELETE are done in a pretty much similar way. It should be pointed out that POST and PUT may have an extra step since you can send data of things you want to create, or the new content of those resources.

```
for (int i = 0; i < nodes.getLength(); i++) {
    Node node = nodes.item(i);

    // Fetching the attributes of the item element
    String created =
        node.getAttributes().getNamedItem("created")
            .getTextContent();

    // We only want to print tiles that have been
    // created and painted
    if ("true".equals(created)) {
        String x = node.getAttributes().getNamedItem("x")
            .getTextContent();
        String y = node.getAttributes().getNamedItem("y")
            .getTextContent();
        String r = node.getAttributes().getNamedItem("r")
            .getTextContent();
        String g = node.getAttributes().getNamedItem("g")
            .getTextContent();
        String b = node.getAttributes().getNamedItem("b")
            .getTextContent();

        System.out.println(String.format(
            "Tile (%s,%s) with color (%s, %s, %s)", x, y, r, g, b)
        );
```

```
    }
}
```

Just like HttpConnection has an inputStream for getting the contents out, it also has an OutputStream to send contents to the server. It uses a special syntax. You might use "r=255" but since you must pass three values (r, g and b) you will use "r=255&g=10&b=100". Despite this little difference, the flow of requesting and processing a method is the same.

```
OutputStream os = connection.getOutputStream();
os.write(("r=" + r + "&g=" + g + "&b=" + b).getBytes());
os.flush();
```

3.2.1.1 Mistakes happen, so one should have a way to fix them. Add a new command undo to PicassoBot which will delete the last tile that it has painted. You should use the DELETE verb.

3.2.2 Python

We start by defining the location of our server. In this case we have set it to a local instance of the game running.

```
#  Defines the server address
server_address = "http://localhost:8080"
```

For the purpose of our class exercise you should use the following server: http://ucx.dei.uc.pt. Update the code accordingly.

The tiles command calls the print method that, as you might have guessed by now, does a GET HTTP request. These lines prepare the Connection that will be used to perform the communication between client and server. In python when we send a request the method is defined. In the second line we set the Request Method to GET. Notice that the same URL may have different behaviors depending on the verb, so you should always be careful to set this accordingly.

```
url = f"{self.server_address}/game/board"
response = requests.get(url, headers=headers, allow_redirects=False)
(...)
```

Also note in the second line that we are configuring the request in a way that no redirect requests will be followed. APIs reply errors in the 300-399 range if a page is no longer there, or if it was moved inside or between servers. You might want to know what kind of redirect was performed (permanent or temporary), and therefore the Client should not follow automatically the redirects.

The headers of a request allow us set multiple properties. In our case we are setting the following ones:

```
headers = {"Accept": "application/xml", "User-Agent": "Pablo v1"}
```

Resources might be available in different formats. Setting the Accept header will ask the server for a particular type. In this case, we are using “application/xml”, but if you wanted, you could ask for “application/json” and then use a JSON library to parse the response.

Some APIs use the User-agent to limit the content or provide slightly different features. It is also important to distinguish your REST client from browsers such as Firefox, Chrome and others that will have a different kind of output. Some APIs use the User-agent to know if you are using a mobile device to compress data even more (to reduce client expenses with a traffic plan).

Since status codes in the 200-299 range are okay, but for anything else we want to debug the issue.

```
# If we get a Redirect or an Error (3xx, 4xx and 5xx)
if response.status_code >= 300:
    print(f"Error: {response.status_code}")
    self.debug(response)
    return
```

Since we asked for XML, we need to parse that format and retrieve the contents we are looking for. The code below will parse our response body that we know that it is in XML format.

```
# Sends the REST request and stores the response
response = requests.get(url, headers=headers, allow_redirects=False)

# Gets the XML from the response text.
# ElementTree is a way of reading XML files.
root = ET.fromstring(response.text)
```

As always, it is good to close system resources that are not used anymore. Notice that we have read when parsing the NodeList and not before that.

```
# Always close the connection when it is no longer necessary
response.close()
```

Finally, we iterate through the nodes and retrieve the available attributes. We begin by checking the ‘created’ attribute to verify if that cell has a tile, and then we extract the color and position information. POST, PUT and DELETE are done in a pretty much similar way. It should be pointed out that you may have an extra step since you can send data of things you want to create, or the new content of those resources.

```
for item in root.findall(".//item"):
if item.get("created") == "true":
    x = item.get("x")
    y = item.get("y")
    r = item.get("r")
    g = item.get("g")
```

```
    b = item.get("b")
    print(f"Tile ({x},{y}) with color ({r}, {g}, {b})")
```

Just like issuing a GET request for getting the contents out, you can also send contents to the server. You need build the data structure (in our case a python dictionary) with the contents you want and add them to the request. Despite this little difference, the flow of requesting and processing a method is the same.

```
data = {"r": r, "g": g, "b": b}
# Send the request to the server
response = requests.post(url, headers=headers, data=data)
```

3.2.2.1 Mistakes happen, so one should have a way to fix them. Add a new command undo to PicassoBot which will delete the last tile that it has painted. You should use the DELETE verb.

4. Hacker news

HackerNews (https://news.ycombinator.com) is a website that allows users to post links to interesting articles, and other users to comment and vote on them. It is a very popular website used by many developers to keep up with the latest news in the industry. In this example, we will make a simple application that will allow us to read the latest news from HackerNews through its REST API. You can find the documentation in https://github.com/HackerNews/API.

4.1 Use Postman to perform requests to the HackerNews REST API using the verb GET and the URI https://hacker-news.firebaseio.com/v0/topstories.json. (Again) Pay attention to the sent and received HTTP messages.

Now that you have a better understanding of how the REST API works, you should implement a (Java or Python) program that will perform the some requests to Hackernews and expose that on its own API. A code snipped for Java (using spring) and Python (using fastapi) is also given.

4.2 The first URI from HackerNews API, "topstories", returns the IDs of the top stories in the JSON format. Use the IDs to retrieve the details of each item. The details of an item are available at the URI https://hacker-news.firebaseio.com/v0/item/id.json, where id is the ID of the item. The details of an item are also returned in JSON format. In the end you should implement an endpoint on your program "/hackernewstopstories" that returns the details of the 50 top stories.

4.3 The details of an item include the title, the author, the URL of the article, the number of comments, among others. Use the title to filter items using search terms passed through request parameters. For example, if the user passes the parameter "?$search = java$", the application should only return top items that contain the word "$java$" in the title.

Java

To run the springboot program you need to issue the following command “mvn spring-boot:run”

Python

To run the fastapi program you need to issue the following command “python src/main.py”