Distributed Programming II

A.Y. 2019/20

Lab6

All the material needed for this assignment is included in the *.zip* archive where you have found this file. Please extract the archive to the same working directory where you have already extracted the material for part a) and where you will work. In this way, you should have the xsd and design documentation developed in part a) in the `[root]/WebContent/xsd` and `[root]/doc` folders respectively.

Purpose

The aim of this Lab is to experiment with the implementation of RESTful web services in Java, using the JAX-RS framework. The exercise gives you the opportunity to apply the knowledge acquired about REST web services implementation in Java.

Exercise 1

1. Write a **simplified** Java implementation (using the JAX-RS framework) of the web service designed in Lab5, by completing the implementation sketch discussed in the classroom and provided in the package, with the addition of the management of bookshelves (the management of journal information is not required, and is omitted). The implementation should include all the main functions defined in your design, but, in order to keep it simple, if there are some features that are functionally not essential (e.g. paging, or some query parameters that have been introduced merely for performance or scalability reasons), they should be omitted in the implementation. If needed, you can make changes to the implementation sketch, but you should not modify the interface already defined for creating and deleting items, because it is used by the final acceptance tests. In the implementation, the maximum number of items that can be put in a bookshelf must be set to 20.

The web service has to be packaged into a war archive that can be deployed to Tomcat. This is done automatically by the ant script `[root]/build.xml`, which includes the target named `package-service` that builds and packages the service. The name of the service packaged in this way will be "BiblioSystem", and its base URL will depend on the configuration of Tomcat (it will be http://localhost:8080/BiblioSystem/rest on the Labinf machines and on the VM). A WADL description will be available at the standard Jersey-provided WADL location (on the Labinf machines, this is http://localhost:8080/BiblioSystem/rest/application.wadl), and a link to the swagger documentation will be available from the index web page. The service must have no dependency on the actual location of Tomcat, nor on the Tomcat port.

In order to populate the service with data from the data generator (the same used in Assignment 1), a client has been provided. It can be started by running the command

```
ant populate-service
```

The service does not have to provide data persistency for bookshelf-related information (data will be stored in main memory) but it has to manage concurrency, i.e. several clients can operate concurrently on the same service without restrictions.

The service must be developed entirely in package `it.polito.dp2.BIB.sol3.service` (and sub-packages), and the corresponding source code must be stored under

[root]/src/it/polito/dp2/BIB/sol3/service. Under this location, you will initially find the implementation sketch of the service discussed in the classroom, which you should use as a starting point for your solution. Note: the implementation of the DB module based on Neo4j is only provided in binary form, as a jar file, along with its documented interface ([root]/src/it/polito/dp2/BIB/sol3/service/db/DB.java).

The ant script [root]/sol_build.xml contains the build-service target for building the service. You can modify it as you need, while you must not modify the main script [root]/sol_build.xml which contains the other targets for packaging and deploying the service and for managing all the other operations. Customization files, if necessary, can be stored under [root]/custom, while XML schema files can be stored under [root]/xsd. Of course, the schema file already developed in the classroom can also be used and modified as necessary.

Important: the ant script defines basedir="." and all paths used by the script must be under ${basedir} and must be referenced in a relative way starting from ${basedir}.

The package-service target available in build.xml calls the build-service target of sol_build.xml and then packages the service into the archive [root]/war/BiblioSystem.war. The descriptor of the service used for the package is under [root]/WebContent. The contents of this folder can be customized by you. The deployment of the package to the Tomcat server can be done by calling the deployWS target available in the build.xml script. Note that this target re-builds the war. Before running this target, you need to start Tomcat. This has to be done by calling the start-tomcat ant target on build.xml (which also sets the expected system properties; **do not run Tomcat in another way otherwise the system properties may not be set correctly**).
Complete the documentation you already developed for part a), by adding a text file that illustrates the most relevant implementation choices you made (including any assumptions you made, and which parts of the design, if any, were omitted in your implementation). Limit the size of this document to 1 page. Store this document in the file [root]/doc/doc.txt.

2. Test your service by means of the postman client (or any other HTTP client) and debug it (use wireshark in order to monitor the execution during debugging).

Exercise 2

1. Implement a client for your web service, which can provide read access to information about items and bookshelves, and also the possibility to add or remove items in bookshelves, and to add or remove bookshelves themselves. The client must be developed as an extension of the client provided in the package, and it must implement the interface it.polito.dp2.BIB.ass3.Client (available in source form in the zip archive of this assignment, with documentation included in the source files).

The classes of the library must be entirely in package it.polito.dp2.BIB.sol3.client and their sources stored in [root]/src/it/polito/dp2/BIB/sol3/client. The library must include the factory class named it.polito.dp2.BIB.sol3.client.ClientFactory, which extends the abstract factory it.polito.dp2.BIB.ass3.ClientFactory and, through the method newClient(), creates an instance of your concrete class implementing the it.polito.dp2.BIB.ass3.Client interface. As in the sample client, the actual base URL and port of the web service to be used by the client must be obtained by reading the it.polito.dp2.BIB.ass3.URL and it.polito.dp2.BIB.ass3.PORT system properties. If these properties are not set, the defaults http://localhost:8080/BiblioSystem/rest and 8080 must be used.

In order to compile your client, use the target named `build-client`, defined in `sol_build.xml`, and modify it if necessary. All the class files must be saved under `[root]/build`. You can assume that Tomcat is running with your BiblioSystem already deployed when the `build-client` target is called (of course, the target may fail if this is not the case). In your script, you can assume that Tomcat is available on the localhost, with the port number specified by the ant property PORT (inherited automatically by the script). Do not use hard-coded values for the port. Customization files, if necessary, can be stored under `[root]/custom`.

2. Test your client by writing a main program that uses it.
3. Run the final acceptance tests by running the ant command

   ```
   ant run-tests
   ```

   Note that this command also redeploys the service, but you have to start Tomcat and Neo4j before running it.

Submission instructions

A single *.zip* file must be submitted, including all the files that have been produced in Labs 5 and 6. The *.zip* file to be submitted must be produced by issuing the following command (from the `[root]` directory):

```
ant make-zip
```

Do not create the *.zip* file in other ways, in order to avoid the contents of the zip file are not conformant to what is expected by the automatic submission system. Note that the *.zip* file **will not** include the files generated automatically.

Deadline for submission: January 26, 2020 23:59 CET