Internet Computing Open Assessment 1: REST Web Services

For this assessment, you are being asked to build a REST-based dictionary application. It is essentially the same application as the SOAP coursework but (a) there are some changes in the operations required; and (b) there are changes in the emphasis of the marking (for example, you can just use the SOAP coursework sample solution as a 'template' to get the synchronisation right, so there are fewer marks for that).

1.1. Task Specification

You need to write:

1. One Java Web Service REST application that provides the dictionary service functionality (see below).
2. Either: one Java application that acts as a client to all functions of the service; multiple Java applications each of which test one function of the service; or web form applications each of which test one function (or more than one if you're feeling adventurous) - or some combination of these. The aim is to ensure that you have a way of testing all functions of your service. I do not mind which combination of clients you choose - with *one* proviso: see below.

1. **Ping.** You should be able to call a simple no-argument service that just responds with "OK" - the point of this is to test the service is working.
2. **Return All Words.** Return all words and their current definitions - you should return data in either JSON (easiest) or XML (if feeling brave) format. If you write a Java client, you need to decide how to represent the words on the client (maybe an `ArrayList` of `Word` objects - where you define your own `Word` class?)
3. **Read a Word.** Return a single word and it's definition - again in JSON or XML. Once again, a Java client needs to decide how to represent a word (use the same `Word` class you wrote for handling all words and their definitions?)
4. **Delete a Word.** Delete a word, if present, from the dictionary.
5. **Add a Word.** Add a *new* word and it's definition to the dictionary.
6. **Update a Word.** Update an *existing* word's definition in the dictionary.

As with the SOAP coursework, you should synchronize operations appropriately - and I strongly suggest you follow the 'pattern' used in the sample solution for SOAP. HOWEVER, unlike the SOAP coursework, one of the read operations just reads a single word which is very quick. It's very unlikely to make sense cloning the dictionary for this, so I suggest in this case you synchronize the whole operation of (Hint!) as much of it as you need to. It DOES make sense to clone the dictionary for the operation that returns all words.

You are free to use whatever of the available REST parameter passing mechanisms you want, and whatever MIME/IMT types you want. But you should pay attention to the chapter that explained which parameter passing mechanisms and MIME/IMT types made sense in each case. You will need to use different MIME/IMT types and parameters if you write a web form application as a client than if you write a Java program (or it will be a lot easier if you do). You can also use any of the available HTTP commands GET, POST, DELETE and PUT, but there will be credit for using the most appropriate ones in each case.

1.2. Overview and Suggestions

This section points out some issues that you might have to consider, and which might be useful.

- **Patterns to Follows.** This coursework is basically a combination of the SOAP dictionary coursework, and the Book REST example. Read both carefully, and the notes that go with Book, and combine them to build a solution.
- **Singleton.** You will have to use the Spring framework and the Singleton pattern as described in the notes.
- **No complicated GUIs.** As before, there is no credit for GUIs.
- **Separate Projects.** A fairly common mistake last time was to put all the code into one project - the trouble with this is that it's very hard to work out if your code is working as a web service or as a 'normal' java application in more than one file. So use two projects for this.

- **Edit the generated client code.** Remember, if you write a Java client to get all words you need to *edit* the generated client code slightly. This is explained in the notes for the Book example, and you can see what needs to be done in the code (downloadable from Blackboard).
- **Client 'Reuse'.** Although it's normally frowned on, you can reuse the basic framework for your client applications from coursework 1 if you want - that is, any user-interface code; code to get/check input etc. etc. (obviously the actual service calls won't be identical, but they won't be much different). This would normally be 'self-plagiarism' and you wouldn't get credit (though you wouldn't get into trouble either). However, since modern software engineering encourages reuse and not reinventing/rewriting things for no reason, in this case it's fine.

1.3. Marking Scheme

As before the marking scheme will identify a number of categories, and define what it means to be in a particular band in each category. There are some changes from the last one.

- **Synchronization.** There are fewer marks for synchronization because I expect you to follow the template in the SOAP example - alternatively, you can do what you did for SOAP provided either (a) you got full marks for it; or (b) you know why you didn't.
- **Marks for responding to feedback.** As always for my 2nd coursework, there are some marks for explaining how you've responded to feedback from the first coursework. Alternatively, if you got 100%, there are marks for explaining what you think you've done better.
- **More marks for Java clients for get word and get all words.** Because a web 'client' for getting a single word and all words is ridiculously trivial (it's just a URI if you do it right - see the Book example), then there are a few more marks available for writing a Java client here.

1.4. Submission Guidelines

You will need to submit your coursework electronically via Blackboard.

Your solution should*either* be a `gzip'ed` and `tar'ed` file or a conventional ZIP file: please read and follow the instructions below (and don't use a RAR file; I still hate RAR files...).

- Include your complete Netbeans project folders - do whatever you did last time, for the SOAP coursework, because everyone's submission was fine: BUT make sure you send a zip file of files.
- Include a file containing simple instructions to run your program called readme.txt.
- **Do not submit your coursework by email unless I *explicitly* ask you to - I will delete it.** If Blackboard does not work for you for some reason, send me an email with `[submission]` (the word `submission' in square brackets) in the title and I will resolve the problem. But do not send your coursework with your email unless I explicitly ask you to do so.

I strongly recommend that after you create your submission file you copy it to another folder, and unpack it to make sure it actually contains what you think it contains. (This will also prevent you inadvertantly submitting a corrupt file or empty file - which a couple of people do every year.)

1.5. Assessment

Coursework 2 accounts for 20% of CS-253. Assessment and feedback will use the Blackboard rubric on the submission page.

1.6. Common Mistakes

The usual mistakes that are made are:

- Sending a corrupt or empty file - please make sure it unpacks and works properly.
- Ending up with files in a subdirectory (or series of subdirectories) after unpacking that is not related to the Java package names you used.
- Sending the class files instead of the source files - believe it or not it happens.

1.7. Dates

The deadline is 11.00 on *Tuesday 2nd May 2017*. To conform to University policy, late submissions will get a mark of zero. Under no circumstances will I grant extensions. However, if you believe you have good reasons for not being penalized, you can make a case for the penalty to be waived. I will attempt to ensure that grades and comments are available, and posted, by *Tuesday 9th May 2017*.