SER422 Spring 2018 ONLINE Lab 3 – REST **Due: Tuesday, April 3 2018 at 11:59pm**

You will implement a REST API based on the PhoneServlet example given in the class GitHub repo in folder Servlets. Specifically, in the edu.asupoly.ser422.phone package there are 2 classes, "PhoneBook" and "PhoneEntry". Your assignment is to design and implement a REST API for these 2 resources that satisfies the following:

1. Provide an ability to retrieve a specific PhoneEntry based on a phone number
2. Provide an ability to create a new PhoneEntry
3. Provide an ability to add a PhoneEntry to a specific PhoneBook. This should error if the PhoneEntry is already contained in a different PhoneBook.
4. Provide an ability to update a PhoneEntry firstname and/or lastname.
5. Provide an ability to remove a PhoneEntry (and remove it from a PhoneBook if it is listed in one) from the system.
6. Provide the ability to retrieve a specific PhoneBook and all entries within the PhoneBook.
7. Provide an ability to retrieve a subset of the PhoneEntry resources in a PhoneBook based on a substring of the firstname, the lastname, or both.
8. Provide an ability to retrieve all unlisted (not in a PhoneBook) PhoneEntry resources.

You need to implement 2 resources, "PhoneBookResource" and "PhoneEntryResource" using Jersey and Jax-RS. The design of your 9 endpoints, including the URI, HTTP Verbs and Response Codes, and request/response payload formats are up to you under the design constraints given below, and in accordance with the grading criteria also described below.

You may certainly start by copying and pasting over the PhoneBook and PhoneEntry source code files referenced above, but know that this code will need to be modified to use as services supporting your resource endpoint implementations. For example:

- The API specification above indicates there are multiple PhoneBooks, and that a specific PhoneEntry is only listed in at most one PhoneBook at a time.
- It is possible to have unlisted PhoneEntrys, meaning a PhoneEntry that is not contained in a PhoneBook.
- The PhoneBook presumes a specific backing file, "phonebook.txt". As you will have to manage multiple PhoneBooks, you will need to decide how to refactor the persistent storage to handle this.

Additional Functional Requirements:

A. Add API documentation using the apidocjs tool available at apidocjs.com. The API documentation should be comprehensive, meaning include all methods, all verbs you process, all errors, and example response formats. The Booktown REST example has some apidocjs documentation in it, as well as an ant target to run it. This documentation should serve as the means by which we understand how to exercise your REST API.

Design Constraints:

1. You must have classes edu.asupoly.ser422.lab3.resources.PhoneBookResource and PhoneEntryResource.
2. The resource classes should not directly implement the functionality to manipulate phone books and phone entries, but rather this should happen in separate classes. For example I've already suggested you start with PhooneBook.java and PhoneEntry.java from a servlet example. You might also choose to adapt code into the internal service class model used in the Booktown REST example (e.g. BooktownService). The actual internal structures are up to you, and you should exercise good design principles (loose coupling, configurable, etc.), but the firm requirement is to delegate from your resource classes to somewhere else in your code to avoid direct business logic in your resource classes.
3. Your endpoint design, including what URIs, what types of parameters, what verbs, what response code, handling errors, and payload formats should use API Design Best Practices. You may use XML or JSON (or both) for payload formats. Your grading criteria indicate how design is mapped to points awarded.

Grading Criteria:

- The apidocjs documentation is worth 12 points, and is expected to specify verbs, response codes, errors, and example payloads. There will be a 1-point deduction for each omission on any endpoint until this bucket is down to zero.
- Each of the 8 endpoints listed above is worth 8 points each (64 points total points). For each endpoint:
    - (1 pts) Use the proper verb
    - (2 pts) Return the proper response code(s) and proper response headers
    - (2 pts) Properly handle errors specific to that endpoint (detect the error and return the proper response code)
    - (1 pts) Construct a proper response payload (Jersey has defaults for you)
    - (2 pts) Construct a proper endpoint URL, including appropriate use of different parameter types. Specifically, whether a parameter should be a path parameter, a query string parameter, or a payload parameter
- The remaining 24 points is based on the correct functional implementation of the success case of each endpoint. To state another way, 3 points per endpoint to ensure you implement the proper internal business logic to do what the endpoint actually asks! This is different than the previous, which simply awards points based on proper RESTful characteristics, not on the functional correctness of the underlying implementation!

Jersey defaults will not necessarily yield full credit for these criteria. Specifically, Jersey does not always return the proper response code for some types of errors, and you will need to know if you should implement a custom error handler. You should also never return a stacktrace. You also need to decide what to do in case of an internal server error (like internally you throw an Exception, or catch one from the runtime, like a java.io.IOException when working with files).

Some very important things to note:

1. Your lab submission must DEPLOY AND RUN WITHOUT ERROR. I expect you will provide a build.xml and build.properties for Ant, or a full build.gradle. Each should have targets for with targets init, compile, clean, dist, and deploy. You should submit the source code, property files, web.xml, build scripts, API documentation and README.txt. You should not submit IDE project files, compiled source code (.class files), or the Jersey libraries – we have those and will put them in a lib folder under your build to keep submission sizes smaller. Not provided a proper submission format and buildfile is an automatic 10 point deduction. This includes having Operating System specific paths or instructions.
2. Be sure to use your README.txt to tell us anything need to know.

Extra Credit (up to 25 points):

1. 5 points for implementing content negotiation – that is, looking at the request header to determine accepted content-types, and returning that type (XML or JSON) based on the prioritized list of preferences expressed in that request header. Note that Jersey can help with this very easily.
2. 15 points for implementing JSON+HAL as the payload format. With JSON+HAL, you have to use a custom serializer/deserializer process to produce JSON that include proper navigable links to other resources.

SUBMISSION:

- Please put a README.txt in the root of your project directory.
- Your submission should be named lab3.asurite.zip for upload to Blackboard.
- If you do the EC, your README must indicate which part(s) you did. You may choose to submit a second file for the EC if you do not want to risk rolling those functionalities into the main submission. If so, submit a separate zipfile named lab3_ec.asurite.zip.
- You may submit as many times as you like, there is NO reason to ask for a late submission!
- If you use jar to archive your files, please be sure to change the extension from .jar to .zip before submitting.
- Your API warfile should be named lab3_ASURITE.war, where ASURITE is your asurite id.
- The html files generated by apidocjs should be submitted in a folder named "apidocjs" under the root of your source directory. Your build script deploy target should copy those files to the root of your web application context.