College of the North Atlantic
CP3566 – Applied Java Programming
Assignment 3

Due Date: Friday, February 11th

JAX-RS Web Services

Part 1: Database Setup

We will use the same database that was used for assignment 1 and 2. If you want to create a fresh instance of it, or need the original SQL file for any reason, see the assignment 1 folder. If you have additional data in your database from assignment 1 and 2 testing and development, that's totally fine.

Part 2: Restful API and Supporting Classes

NOTE: Ensure that you use a Java Enterprise project, choose the REST Service template, make sure you also choose version "Jakarta EE 9".

In this assignment, you will create a restful API that enables a user to get information from your books database, as well as enable a user to perform a variety of the standard CRUD functionality. Note that several aspects of implementation will be left up to you. You must meet the following objectives:

1. The API(s) should be accessible through an application path called "**app**" (i.e. the URL mapping for the entire application).
2. Your main Restful class should be called **LibraryController**. It should be accessible through a path called "**library**".
3. It is recommended to have a class to handle your database connections, such as the **DBConnection** class from assignment2. Such a class will handle the details of setting up a database connection.

NOTE: For the following objectives, any supporting classes/methods, and specific implementation are up to you.

4. The default URI/URL (i.e. relative path **app/library** - no specific path parameters) should display a simple message with a little information about the intentions of the API.
5. If the URI is set to "books" (i.e. relative path **app/library/books)**, a method will return a listing of all books from the database (along with relevant author data). Data should be returned in JSON format.

6. If the path parameter is set to "authors" (i.e. relative path **app/library/authors)**, a method will return a listing of all authors from the database (along with relevant book data). Data should be returned in JSON format.
7. Related to the above, you should also be able to specify the id of either a book or an author, and return just that book/author data.
    1. In this instance, the relative URI for getting one book would be **library/book/{id}** where "id" can be dynamically assigned.
    2. The relative URI for getting one author would be **library/author/{id}** where "id" can be dynamically assigned.
8. Three methods will be related to **POST** requests to the API. They will be differentiated by the path specified, and each will add content to the database:
    1. If the path is set to "**addbook**" (i.e. the relative URI would be **library/addbook**), then an **addBook** method will accept JSON data related to the elements of a book. It should return a JSON representation of the data that was added.
    2. If the path is set to "**addauthor**" (i.e. the relative URI would be **library/addauthor**), then an **addAuthor** method will accept JSON data related to the elements of an author. It should return a JSON representation of the data that was added.
    3. If the path is set to "**associateAuthor**" (i.e. the relative URI would be **library/associateauthor**), then an **associateAuthor** method will accept JSON data with containing the id's of an author and a book, and from a database perspective, will create the linkage between them. It should return a JSON representation of the data that was added.
9. Two methods will be related to **PUT** requests to the API. They will be differentiated by the path specified, and each will modify content in the database:
    1. If the path is set to "**modbook**" (i.e. the relative URI would be **library/modbook**), then a **modBook** method will accept JSON data related to the elements of a book. Based on the id of the book specified, the associated book record in the database will be updated with any new data. It should return a JSON representation of the data that was modified.
    2. If the path is set to "**modauthor**" (i.e. the relative URI would be **library/modauthor**), then a **modAuthor** method will accept JSON data related to the elements of an author. Based on the id of the author specified, the associated author record in the database will be updated with any new data. It should return a JSON representation of the data that was modified.
10. Two methods will be related to **DELETE** requests to the API. They will be differentiated by the path specified, and each will delete content in the database:
    1. If the path is set to "**delbook/{id}**" (i.e. the relative URI would be **library/delbook/{id}**, where "id" can be dynamically assigned), then a **delBook** method will remove the book based on the id of the book specified. This method should simply return a message stating that the book was removed.
    2. If the path is set to "**delauthor/{id}**" (i.e. the relative URI would be **library/delauthor/{id}**, where "id" can be dynamically assigned), then a **delAuthor** method will remove the author based on the id of the author specified. This method should simply return a message stating that the author was removed.

Remember, you can test all of your API calls using Postman, as demoed in class.
When complete and tested, export your project as a zip file and submit to the assignment 3 dropbox.