SOEN 487 – Winter 2020
Assignment-II

10% of the Final Grade

Due Date: March 24, 2020 – 11:59 PM
Group Assignment: A maximum of three per team

Combining Web Services

Objectives

The purpose of this assignment is to create a small backend application that provides SOAP as well RESTful web service that support complex data types.

Overview

In this assignment you are extending the basic service you created for a library system in assignment 1, by adding a persistency layer as well expanding the domain of the service. This assignment consists of two domains: Book Repository, Loans.
This implementations in this assignment are not yet secure. Therefore, no authentication is required (however, see also tasks 7).

Book Repository

The Book Repository is a RESTful service that maintains information of books, as in the assignment 1, with the support of four data formats: Text, Html, Json, and XML. Details are provided in the next sections.
The information about the books are:

- title, description, isbn, author, and publisher

Loans

The Loans Service is SOAP RPC style service that maintains the loans.
The information about the loans are:

- book title, person who borrowed the book (member), date of borrowing, and return date.

Project Outline

This assignment consists of seven tasks as listed in the following sections to be implemented in seven separate projects:

- The Library Core
  A java class library containing the common classes / data structures that are shared between the client and the service

- The Loan Core
  A java class library containing the common classes / data structures that are shared between the client and the service

- The Library System
  A java class library that that implements the library system (The Business)

- The Loan System
  A java class library that that implements the loan system (The Business)

- The Library Service
  A java application implementing a **self-contained** RESTful web service that uses the library system.

- The Loan Service
  A java application implementing a **self-contained** SOAP RPC-style web service that uses the loan system.

- The Client
  A simple java web application that enables the library clerk to view books and record borrowings.

Task 1. The Library Core

Create the library core project and include common data structures (i.e. Entity Class(es)) that are shared between client and the service in here.
No business classes to be implemented here.
No references to Loan Core.

Task 2. The Loan Core

Create the library core project and include common data structures (i.e. Entity Class(es)) that are shared between client and the service in here.
No business classes to be implemented here.
No references to Library Core.

Task 3. The Library System

Design and implement a class called Library that takes care of storing and modifying the library data. Data is stored in a **database**.
Use a database of your choice.
Make sure the connection string and all required parameters are stored in a **config file**.
Implement standard CRUD operations. Reference Library Core in task 1.
From the business layer's perspective, books are identified by a unique identifier (i.e. a string such as call number). Visit https://library.concordia.ca/ for some sample data.
Use Exceptions for error handling. In case of errors, an instance of LibraryException is thrown.

Task 4. The Loan System

Design and implement two class called memberManager and LoanManager that takes care of storing and modifying the member and loan data.
Data is stored in a **database**. Use a database of your choice.
You may share the same scheme as in Task 3, however, no references to the Library System must be used.
The Loan System and The Library system are completely isolated modules.
The link to the library data is via the book unique identifier (i.e. "call no." string).
Make sure the connection string and all required parameters are stored in a **config file**.
Reference Library Core in task 2.
Implement the following functions:

- Standard CRUD operations on Member data.
- Borrow a Book (given the book unique identifier, and Member id)
- Edit Book Loan
- Return Book
- Delete a Book Loan

Use Exceptions for error handling. In case of errors, an instance of LoanException is thrown.

Task 5. The Library Service
Using the class library created in task 3, implement a self-contained RESTful service to provide the following functionalities:

- list (shows the list of current books, id and title)
- get book (returns the book info by the id)
- add book (adds a new book to the system)
- update book (updates specific book, id and the data to be given as arguments)
- delete book (deletes a book by id)

Create and implement the Library RESTful Web Service that implements all four verbs:

GET, **POST**, **PUT**, and **DELETE**.

Use appropriate REST method for CRUD operations.
Use proper annotation.
For non JSON/XML data types, you may use String data type for all attributes.
The following data types are supported:

- JSON: may be used for both consuming and producing.
- XML: may be used for both consuming and producing
- TEXT: may be used for producing only. You may implement toString() method for the entity class(es) in task 1.
- HTML: may be used for producing only.
  Use XSLT to transform entity data into HTML.

For all functions, support all possible data formats for returning the result: JSON, XML, TEXT, and HTML.
For consuming data, show at least two methods of implementation to support complex data type (JSON and XML), as well as parameterized argument (i.e. query param, form param, etc.)
For producing data, enable all four, as listed above.
Handle exceptions properly. The web service must not crash. You may use web exceptions to report the error or generate the error in a readable format by the client (i.e. json, xml, text, etc., depending on the accept header)

Task 6. The Loan Service
Using the class library created in task 4, implement a self-contained RPC-Style SOAP service to provide the following functionalities:

- list members
- get member info (id, name, contact)
- add member
- update member
- delete member

- list loans (given a specific book)
- list loans (given a member id)
- get loan (returns load details by id)
- borrow (create a loan) -- book must be available
- update loan
- return item (book is returned)
- delete loan

Use your own design. Make sure the business logic is properly implemented.
Handle exceptions properly. The web service must not crash. Use soap faults to report errors.

Task 7. The Web Client

This task implements a thin client that uses the two services in tasks 6 and 7 that is used by a library clerk.
In this tasks you need to provide the functionalities that are listed in the services.
Use your own design for UI.
You may use any client technology, as you wish; even a non-java language.
Keep it user friendly.

Make sure the connection strings to the web services are configurable (use config file).
Test availabilities of the services. For instance, in case either services are not, warn the user.
Use a predefined super-user (stored in the config file) to authenticate the library user in the web client.

Task 8. The Class / Package Diagram
Draw UML diagrams of your system, by highlighting modules, layers (projects), packages, and classes. Show all associations and dependencies.
You may use separate diagrams, as not all may be fit in one UML diagram.

Deliverables

IMPORTANT: You are allowed to work on a team of 3 students at most (including yourself). You and your teammate must be in the same section. Any teams of 4 or more students will result in 0 marks for all team members. If your work on a team, ONLY one copy of the assignment is to be submitted for both members. You must make sure that you upload the assignment to the correct directory of **Assignment 2** on Moodle. Assignments uploaded to the wrong directory will be discarded and no resubmission will be allowed.

Naming convention for uploaded file: Create one zip file, containing all needed files for your assignment using the following naming convention:
The zip file should be called *a#_studentID*, where # is the number of the assignment *studentID* is your student ID(s) number. For example, for the first assignment, student 12345678 would submit a zip file named *a1_12345678.zip*. If you work on a team and your IDs are 12345678 and 34567890, you would submit a zip file named *a1_12345678_34567890.zip.*

Submit your assignment electronically via Moodle based on the instruction given by your instructor as indicated above. **Please see course outline for submission rules and format, as well as for the required demo of the assignment.** A working copy of the code and a sample output should be submitted for the tasks that require them. A text file with answers to the different tasks should be provided. Put it all in a file layout as explained below, archive it with any archiving and compressing utility, such as WinZip, WinRAR, tar, gzip, bzip2, or others. **You must keep a record of your submission confirmation.** This is your proof of submission, which you may need should a submission problem arises.

Grading Scheme

T#	MX	MK	
1&2	/6		*common structure (lightweight, no business)*
3	/10		*persistency (2), configuration (1), functionality (5), exceptions (2)*
4	/15		*member system (5), loan system (5), persistency (2), configuration (1), exceptions (2)*
5	/29		*REST (5), HTML and XSLT (6), JSON (5), XML (5), TEXT (3), Parameter arguments (2), error handling (3)*
6	/17		*SOAP (5), faults (3), members (4), loans (5)*
7	/16		*correct UX/UI (7), error handling (3), service availability (3), configuration (2), super-user (1)*
8	/7		*UML diagram, dependencies, low coupling,*
Total:	/100		

(T# - task number, MX - max (out of), MK - your mark)

References

1. https://www.vogella.com/tutorials/REST/article.html
2. https://library.concordia.ca/
3. https://www.w3schools.com/xml/xsl_intro.asp