SOEN 487: Web Services and Applications

Assignment 2 Combining Web-Services

Winter 2021, sections NN

February 9, 2021

Contents

1 General Information

Date posted: Wednesday, February 10th, 2021.

Date due: Tuesday, march 16th, 2021, by 23:59.[1].

Weight: 12% of the overall grade.

2 Introduction

This assignment targets implementing integrated web service. In this assignment you create a small back-end application that provides SOAP-based as well RESTful web services that support complex data types.

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 application consists of two domains: 1) Albums and 2) Change Logs.

See section 4 for the details.

3 Ground rules

You are allowed to work on a team of 3 students at most (including yourself). Each team should designate a leader who will submit the assignment electronically. See Submission Notes for the details.

ONLY one copy of the assignment is to be submitted by the team leader. Upon submission, you must book an appointment with the marker team and demo the assignment. All members of the team must be present during the demo to receive the credit. Failure to do so may result in zero credit.

This is an assessment exercise. You may not seek any assistance from others while expecting to receive credit. **You must work strictly within your team).** Failure to do so will result in penalties or no credit.

[1] see submission notes

4 Overview

This assignment consists of a back-end application whose API is exposed through web-services, as well as a thin client that is connected to the back-end system.
This back-end application consist of two domains: 1) Albums and 2) Change Logs.

The back-end system in this assignment essentially delivers a repository system for storing albums data. The data is persistent and changes are tracked. Any changes in the data is logged and the log is also available for preview.

The front-end maybe implemented in any platform using any framework or programming language, with the following condition: The thin client provide accesses and manipulates the system data strictly through the web service API. No direct call to the business, or data is allowed.

This implementations in this assignment are not yet secure. Therefore, no authentication is to be implemented.

5 Your Assignment

The assignment consists of the following deliverables:

1. **The Albums Core:** providing core interfaces and structures.
2. **The Albums Persistence Layer:** to handle data persistency in a database (or file system).
3. **The Albums Repository:** the business [domain] layer.
4. **The Albums Service:** consists of a JSON RESRful service for albums and SOAP service for change logs.
5. **The Albums Client App:** a thin clients
6. **The UML Diagrams**

5.1 The Albums Core

To be implemented as a java "thin" class library, this project delivers common interfaces and POJO objects. It may be regarded as a "client" library[2]. As such, service interfaces may also be defined in here.
Below are some key objects that are to be defined here. You may add more.

- Album
- Artist (to be aggregated in Album)
- RepException
- LogEntry

See the sections 5.2 and 5.3 for details.

5.2 The Albums Persistence Layer

The data in this assignment is persistent. To implement the data persistency, you may use any of the Fowler's Data Source Architectural Patterns (see 6):

> Table Data Gateway (144), Row Data Gateway (152), Active Record (160),
> Data Mapper (165).

Using a relational database is recommended, however not mandatory. You may use any database system of your choice (relational, object oriented, or even a document oriented database).
The albums consist of the following data items:

> a unique International Standard Recording Code (ISRC), a title, an optional content description, released year, artist's first and last names[3], and a cover image[4].

Any changes to the Album data must be logged. A log entry consists of the following info:

[2]This class library may optionally be referenced on the client side for data serialization and/or accessing the web service interface i.e. (using SOAP API).
[3]The artist information is stored in the same table as in the albums; no normalization is required.
[4]Note that you need to store image data as well as its MIME type.

time-stamp, type of change (CREATE, UPDATE, or DELETE), and the record key (ISRC).

5.3 The Albums Repository

This java class library is the business layer. It essentially provides a repository system for albums data, similar to assignment 1, with the following changes:

i) One repository class handles all data (albums and artists together)
ii) All changes in the system are logged
iii) Albums may optionally have a cover image

The following business functionality is provided by this layer.

- Create a new Album
- Update Album
- Delete Album
- Get Album Info
- Get Albums List

- Update Album Cover Image
- Delete Album Cover Image
- Get Album Cover Image

- Get Change Logs
- Clear Logs (currently not available)

Any changes to the data automatically generates a log entry, which will contain the time-stamp (date and time), type of change, as well as the record key. In case of errors, a *RepException* is to be raised by the business layer.

The Cover Image may be returned as `byte[]`, or the business layer may support writing into an output stream. Note that in either case, the MIME type must also be returned. The client uses this info to render the binary data as a proper image MIME type.

The Get Change Logs method may receive three optional parameters: from date-time, to date-time, and change type. If none is given, all log entries are returned.

Albums are sorted alphabetically by title; log entries are sorted chronologically by the time-stamp.

The Clear Logs method is not supported at the moment; as a result, the method simply raises a *RepException*, indicating "the method is not yet supported".

5.4 The Albums Service

The Albums Services is implemented in two parts: 1) a RESTful service providing the Album data, and 2) a SOAP service providing the log entries, both of which use the very same class library implemented in section 5.3.

The RESTful service handles the album data, including its aggregate artist info. It essentially exposed the API (implemented section 5.3) as a JSON RESTful API. The SOAP-based service provides the log entries.

Note: You are required to catch ALL runtime exceptions that may be thrown by the underlying layers and handle them properly. The RESTful service may return the error details as an HTTP error or as a JOSN object. The errors in SOAP service must be generated as *Faults*. In both cases, you may reuse the "RepException" class you designed in section 5.1.

5.5 The Albums Client App

This thin client, to be implemented using any platform or programming language, uses the Albums service and lets the user view or change albums data. The Album client App strictly uses the albums service, implemented in section 5.4. No direct link to the business, database, or any files within the server file system is allowed.

In case of implementing in Java, yet optional, the core library (implemented in section 5.1) may be referenced.

The Client App demonstrates the following functionalities:

1. Search for an Album:

   receives a search pattern, calls the 'list' method, filters the data, and displays the result[5].
2. View Album:

   to display the album info in a separate page, including the album cover image, if available.
3. Create a new or modify an existing Album
4. Delete an Album
5. Update or Delete an Album Cover Image
6. View Change Logs:

   displaying the log entriess within a given range and/or filtered by the change type

5.6 The UML Diagrams

Upon completion of your project, generate and include the "UML class / package diagram" of the implemented classes in your project. Include all classes that are used in the Core, The Business, the Persistency, and the Service layers.
Using a sequence diagram, show how a cover image is updated by the client.

6 What to Submit

The whole assignment is submitted by the due date under the corresponding assignment box. One submission per team.

[5]No search method is to be implemented in the service layer

Submission Notes

Clearly include the names and student IDs of all members of the team in the submission. Indicate the team leader.

IMPORTANT: You are allowed to work on a team of 3 students at most (including yourself). 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. You must make sure that you upload the assignment to the correct assignment box on Moodle. No email submissions are accepted. Assignments uploaded to the wrong system, wrong folder, or submitted via email will be discarded and no resubmission will be allowed. Make sure you can access Moodle prior to the submission deadline. The deadline will not be extended.

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#_studids, where # is the number of the assignment, and studids is the list of student ids of all team members, separated by (_). For example, for the first assignment, student `12345678` would submit a zip file named `a1_12345678.zip`. If you work on a team of two and your IDs are `12345678` and `34567890`, you would submit a zip file named `a1_12345678_34567890.zip`.
Submit your assignment electronically on Moodle based on the instruction given by your instructor as indicated above: `https://moodle.concordia.ca`
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.

7 Grading Scheme

The Albums Core	5 marks
The Albums Persistence Layer	10 marks
The Albums Repository	15 marks
The Albums RESTful Service	15 marks
The Albums SOAP Service	15 marks
The Albums Client App	20 marks
Error Handling	10 marks
UML	10 marks

Total: 100 marks.

References

1. REST with Java (JAX-RS) using Jersey:

   `https://www.vogella.com/tutorials/REST/article.html`
2. SOAP with Java (JAX-WS):

   `https://www.baeldung.com/jax-ws`
3. SOAP Faults:

   `https://www.tutorialspoint.com/soap/soap_fault.htm`
4. REST Content Handlers (ch. 6):

   `https://dennis-xlc.gitbooks.io/restful-java-with-jax-rs-2-0-2rd-edition/`
5. REST client tool:

   `https://www.jetbrains.com/help/go/rest-client-tool-window.html`
6. Martin Fowler's EAA Catalog:

   `https://martinfowler.com/eaaCatalog/`