SOEN 387 – Fall 2019
Assignment-II

9% of the Final Grade

Due Date: Nov 1, 2019 – 11:59 PM
Group Assignment: A maximum of three per team

A Book Repository Implementation

Objectives

Understanding layered architecture, repository implementation, using databases, applying MVC approach, binary data and formatted text.

Project Outline

In this assignment, you are asked to create the following projects:

- The Repository Core (implementing the business logic tier with no assumptions on the front-end platform.

- The front-end Web Project (implementing the front-end UI using servlets and jsp)

Use separate packages (or sub packages) for the two projects.
i.e.

com.your-name.soen387.repository.core,
com.your-name.soen387.repository.webui, … // these are only suggestions

Part A - The Repository Core

The repository core implements a basic repository to store information for books. The information contains:

- id (integer, automatically generated)
- title (string)
- description (string)
- isbn (string) -- unique
- author (firstname and lastname both strings)
- publisher (publisher company and address, both strings)
- cover (image using a format of your choice)

The repository implements the following functions:

- list all books
- get book info (by id)
- get book info (by isbn)
- add new book (id will be returned)
- update book info (id and book details received as argument)
- set/reset book cover image
- delete book (by id)

- delete all books

Entity Classes
The following entity classes are implemented and used by the repository core:
- **Book** (id, isbn, title, description, author, cover)
- **Author** (firstname, lastname)
- **CoverImage** (mimetype, imagedata)

The Session Class
This class, implemented in the business logic, is a place holder representing a business session between the client (front-end) and the server (business core). The session class has the following functionalities:
- getCurrentUser()
- isUserLoggedIn()
- login(userid, password)
- logout()

The session class uses a pre-defined user/password readonly list stored in users.json file located somewhere in the project.
The path to the file is stored in a config file.
Password are stored in MD5 hashes.

Note that this session class has nothing to do with the http session.

The Repository Class

The interface definition of the Repository Class is defined in an interface called **IBookRepository**.
The Repository is then implemented in a singleton object called **BookRepository**.
The static method returning the repository returns a reference to IBookRepository.
Note that none of the methods in the Repository class must be exposed to the outside objects. Functions are only accessed via IBookRepository interface.
Also note that the repository class must be thread safe. Implement synchronization guards, if necessary.
The static method that returns the repository reference receives the "context" as its argument so that in case the context is null, it returns null reference (meaning the repository is not available).

Note: the first argument to every method in the repository is a reference to the "**Session**" class, as specified above). Every function checks if the user has logged in to the session or not; and if not, it will throw a RepositoryException with proper message (see below).

The RepositoryException Class

All repository functions must handle errors through an exception class called RepositoryException, which is implemented in the business core.

Part B - The Front-end UI

The front-end UI consists of the following pages (views):

- Login screen (only displayed when user is not logged in)
- Home screen (which lists the current books in the system and links to the add/update pages, and an option to delete a book or all books, as well as logout option)
  Note: for deleting, a confirmation must be given by the user; Use client-side script
- View book details (showing the details + the book cover)
- Add/Update book into (including the cover image)
  Note: updating book info may not necessarily changes the book cover image
- Logout page: a black page that displays the logout message and provides options to close the window or go back to the login page
- An image display **servlet** (or jsp) to dynamically show the image cover or specific book id (this servlet is used in view / update screen)

Note: showing image thumbnails on the home screen (books grid) is not required. In case it is implemented, it is considered as **BONUS**.

Tasks

The assignment consists of the following five tasks:

1. Creating the Database Layer
2. Creating the Business Core
3. Creating Junit Test
4. Creating the Front-End UI
5. The System Model

Task 1. Creating the Database Layer
Target topics: database scripts (table creation)

Create a file called database-init.sql and stored all DDL commands that sets up your database. This is normally a file with one CREATE TABLE statement. In case you implement stored procedures, include them in this file as well.

Use a database of your choice that contains only one table called BOOK. The BOOK table includes all the necessary columns to store the book information including author, publisher, and image cover (and details).

Use BLOB data type for image data.
Do not use AUTO INCREMENT for id. Implement your own auto increment.
Note: when executing multiple statements on the database, make sure you respect atomic transactions (in case of implementing the auto increment by running a SELECT statement followed by an INSERT statement).
You may alternatively handle this at the database level by using stored proc, for instance.

Task 2 Creating the Business Core

Create a java class library project. This project must not reference any data structures defined in the web UI. Any external libraries / jars that are used in this layer must explicitly added to the project (i.e. json library for handling user information, jdbc, …) Implement the entity classes, the session class, the exception class, as well as the repository interface and implementation.

Task 3 Creating JUnit Test

Use Test-Driven Development (TDD) approach for the implementation of the Business Code. For every repository function, create necessary JUnit test cases with proper before and after configuration.

Implement at least 9 key test functions. Below are some examples:

- Create a new book (check if id > 0)
- Create a duplicate isbn (expect error)
- set book cover image, get book cover image, verify equal
- delete non existing book (expect error)
- …

Submit your test report.

Task 4 Creating the Front-End UI

In this task, you create a java web project. Reference the repository core in here.
In this project no additional business functionality is implemented. All business functionalities must be implemented in the core project.
This project hosts the jsps as well as the servlet(s).
Use MVC approach to implement the front-end. Refer to the section "**The Front-end UI**" for the details.

- In additional to the authentication check that is done in the business layer, in the UI layer, every page must also check if the user is logged in or not.
- In case the user is not logged in, the app must take the user to the login page. This must be done in every page.
- Use http session to host the core "Session" object.
- To access the repository, you may either host it in an application variable or use the singleton pattern.
- Implement proper 404 page, so that in case the user types an invalid address it guides the user to the home page.
- Home page may be used as the default page. In case the user is not logged in, it automatically goes to the login page.

Task 5 The System Model

We normally do this prior to coding. However, this task is intentionally done at the end of the project, in order to give you a better understanding of the system components.

In this task you are creating the following diagrams:

- The Use Case Diagram
- The Add Book Scenario
- The Set Book Cover Scenario
- The System Sequence Diagram (SSD) "Set Book Cover Scenario"
- The Sequence Diagram (SD) for "Set Book Cover Scenario"
- The System Class Diagram (Business Core only)
- The Package Diagram

You may use reverse engineering to create many of the above diagrams.

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** using EAS. 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 EAS 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	/10		*database script (ddl, tables, and other objects)*
2	/35		*business core, including: entity classes (4), session (2), user-mngt (3), configuration (2), repository (6), exception (2), concurrency (1), sql-injection (2), atomicity (2), singleton (2), error handling (2), cohesion & coupling (2), design (5)*
3	/10		*junit test (9), test report (2)*
3	/35		*front-end: functionality (20), image servlet (5), encoding and security (5), client scripting (5),*
4	/10		*use-case and scenarios (1+2×1.5), sequence diagrams (1+2), class diagram (2), package diagram (1)*
Total:	/100		

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

References

1. https://draw.io/
2. https://www.visual-paradigm.com/VPGallery/import/Rose.html