SOEN 387: WebBased Enterprised Applications Design

Assignment 3 on Architectural Patterns

Fall 2020, sections F

November 17, 2020

Contents

1 General Information

Date posted: Tuesday November 17th, 2020.

Date due: Tuesday December 8th, 2020, by 23:59[1].

Weight: 8% of the overall grade.

2 Introduction

This assignment targets implementing web applications with an emphasis on 1) layered architecture, 2) design, and 3) architectural patterns.

2.1 Overview

This assignment is the second iteration on the "Message Board System", implemented in assignment 2. You are free to use the code base you developed in assignment 2, refactor it, or completely rewrite it as a new project.

Note: The detailed description of the functionality of the system is specified in assignment 2. Section 4 lists the changes as well as the additional features.

3 Ground rules

You are allowed to work on a team of 4 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

[1] see submission notes

result in penalties or no credit.

4 Your Assignment

This assignment is the second iteration on the "Message Board System" that was implemented in the previous assignment. It consists of the following sections:

1) Handling Users and Groups, 2) Test Driven Development, 3) Using Patterns, and 4) Using Web Presentation Patterns.

4.1 Handling Users and Groups

In the previous assignment, we did not address the data privacy. In this assignment, access to the posts are restricted to the users who have permission to see them. In order to do so, the following are to be implemented.

4.1.1 The Users and Groups Info

In addition to the users info in assignment 2, the following additional information needs to be handled. Similar to the previous assignment, the information is readonly and the system does not let the user modify them.
The requirements to implement security groups are as follows.

1. The system uses a predefined list of groups that is stored in a file (very similar to the list of users in the previous assignment). You may use any file format i.e. json, xml, etc.

2. Each group may have an optional parent group which indicates inclusion relationship from the parent to the child. If a group does not have a parent, it is considered as a top level group.

3. The user-membership identifies which the groups each user blongs to (i.e. membership records).

Examples:

```
Users:
    john    (data fields as in the previous assignment)
    jane    ( " )
    jack123 ( " )

Groups:
    admins
    concordia
    encs (parent: concordia)
    comp (parent: encs)
    soen (parent: encs)

Membership:
    john: admins, concordia
    jane: encs
    jack123: soen
```

You may combine all about information in a single file or implement each in separate files.

4.1.2 System Administrators

A predefine group (stored in the application configuration file) specifies the group name of the system administrators. A system administrator is a user with full permissions that can delete / update any information in the system.
Example: `admins`; as a result, `john` is the system administrator and has full control in the system.

4.1.3 The Login Process

During the login process, the user is authenticated and its group membership is loaded and stored in the user session. Administrators are implicitly member of all groups in the system.
Examples:

- `john` is a system administrator and therefore is a member of `admins`, `concordia`, `encs`, `comp`, and `soen`.

- `jane` implicitly belongs to `encs`, `comp`, and `soen`.

4.1.4 Creating and Viewing Posts

When users create posts, they may specify a group for their post, by which only users under that group may see the post. Specifying the group is optional. If a group is not specified, the post is considered as "public". Users are allowed to view public posts as well as the posts under the groups they are member of.

4.1.5 Updating Posts

Users are allowed to update or delete their own posts. Admins are allowed to update or delete any posts.

4.2 Test Driven Development

In this assignment you are following a TDD approach for implementing the above features. During the demo you need to show the results of the test cases.
Important: Although the specs in 4.1.1 indicates the data is readonly, your program must be able to detect erroneous data. Examples of such cases are given below:

1. a group membership that has undefined group or user
2. a group definition with a non-existing parent

Question: Does your program prevent circular parent-child definition? How?

4.3 Using Patterns

4.3.1 Using Separated Interface

The user Management is implemented using *Separated Interface (476)*. To do so, you need to create an interface (namely UserManager) that authenticates the user, and provides the user information including the group membership. The interface is to be implemented in a separate project.

Note that the UserManager interface is defined in the business layer, while the implementation is done in a separate layer. See 2.

4.3.2 Using a Singleton Factory Pattern

In order to instantiate the UserManager in the business layer, use a *Factory* patterns, also implemented as a *Singleton*. The factory pattern simply looks for the class name of the implementation in a config file and uses `class.forName()` to dynamically load the java class. You need to use `class.newInstance()` (or a similar) method to instantiate the class dynamically (see 5).

4.3.3 Implementing the UserManager

The implementation of the UserManager is done in a separate class library project. While the busness layer is not aware of any references to this project, this class library requires a reference to the business layer, as it contains the definition of the UserManager interface.
Note: Your implementation may require some parameter (i.e. the path to the file(s)) that is/are to be received via the business layer. To do so, the factory class in the business layer passes the path to the file(s) as argument(s) to the constructor of this implementation class.

4.4 Using Web Presentation Patterns

In addition to view the post, users have option to download a post as xml. To do so:

1. Use *Template View (350)* to implement the view post page.
2. The view post page has a link that enables the user to download / view the post as xml.
3. Use *Transform View (361)* to transform the post data into XML.

5 What to Submit

The whole assignment is submitted by the due date under the corresponding assignment box. It has to be completed by ALL members of the team in one submission file.

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 4 students at most (including yourself). Any teams of 5 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.

6 Grading Scheme

Users and Groups Functionality	20 marks
Processing Users and Groups Data	15 marks
TDD	15 marks
Separated Interface	10 marks
UserManager Factory	10 marks
Web Presentation Patterns	20 marks
Error Handling	10 marks

Total: 100 marks.

References

1. Martin Fowler, "Patterns of Enterprise Application Architecture (EAA)", 2002, ISBN-10: 0321127420, ISBN-13: 978-0321127426
2. Separated Interface (476) :
   `https://martinfowler.com/eaaCatalog/separatedInterface.html`
3. Template View (350):
   `https://martinfowler.com/eaaCatalog/templateView.html`
4. Transform View (361):
   `https://martinfowler.com/eaaCatalog/transformView.html`
5. Invoking Constructors using Reflection:
   `https://java2blog.com/invoke-constructor-using-reflection-java/`