11/24/2018

Verification and Code Review

Project Deliverable 5

Andrew Yue-Keung Leung, Anthony Leung, Rundong Liu, Sherry Hoi Yan Ma, Sean Hung Nguyen
MACROHARD RAINFOREST INC.

Table of Contents

Testing

Unit/Integration Testing

Our unit tests can be found in the **Team7** repository in the following directory:

Application/src/test/java/com/teqlip

In that directory, folders to the **database** unit test files and **gui** unit test files can be found.

To run all unit tests, use the **mvn test** command while in the **Application** directory.

Acceptance Testing

U1: Create account

1. Login as a TEQLIP user
2. Click Create/Manage Account
3. Enter a new username, role, email, etc. for the new account
4. Click submit
5. Check email for credentials. Password is automatically generated and sent there
6. Login with those credentials

U2: Upload template

1. Login as a TEQLIP user
2. Click Upload/Manage Files
3. Enter a name for the template, and either type in a path or browse for a path to the .xls file
4. Click upload
5. Login as an Organization user and click Download Files to see if the name of the added template is a selectable option in the dropdown bar

U3: Download template

1. Login as an Organization user
2. Click Download Files
3. Choose a template name from the scrollbar
4. Click Download
5. Check if the file is downloaded and is in the same directory as this application

U4: Upload filled out template data

1. Login as an Organization user
2. Click Upload Data
3. Select the path of the filled-out template data file
4. Click Upload
5. Follow **U5** to see if the upload of this data was successful

U5: View submitted anonymized template data

1. Login as a UTSC user
2. Choose a template name from the dropdown bar
3. Click Submit
4. A table with all data from that table should pop up

U6: Query

1. Login as a TEQLIP user
2. Click Query
3. Enter a query or choose a query from the saved queries dropdown bar
4. Click Execute Query
5. A table with the requested data should pop up

U7: Save query

1. Login as a TEQLIP user
2. Click Query
3. Enter a query
4. Click Save Query
5. Click Back then click Query again to check if the query is added to the saved queries dropdown bar

U8: Export query results as .csv file

1. Login as a TEQLIP user
2. Click Query
3. Enter a query or choose a query from the saved queries dropdown bar
4. Click Export CSV
5. Check if a new .csv file is created with the queried data and is added to the same directory as the application

U9: Delete account

1. Login as a TEQLIP user
2. Click Remove Accounts
3. Enter a username that corresponds to the account being deleted
4. Click Submit
5. Logout and try to login with the account being deleted (you should not be able to)

U10: Change password

1. Login as any user
2. Click Change Password
3. Enter the old password and the new password and the new password again for confirmation
4. Click Submit
5. Logout and try logging in with the old password (you should not be able to) and with the new password (you should be able to). Repeat for all three types of users

U11: Forgot password

1. On the login menu click “Forgot Password?”
2. Enter the email and username associated with the account
3. Click Submit
4. Check email for the new password. A new password is automatically generated and sent there
5. Login using the new password. The user may follow **U10** to change their password

Code Review

Strategy

While performing the code review, each team member should answer these main questions:

1. Can I understand what the code does just by reading it?
2. Are proper coding style and conventions being followed?
3. Is the code well-documented?
4. Are there multiple if/switch-case statements?
5. Is the code easily testable?

If a team member has done something particularly well that stands out, it should be commented upon, so all team members will know what direction to follow. If improvements can be made to a team member's code, constructive criticism should be provided so that all team members can benefit.

The following are some optional questions to consider during the code review:

1. Could we reuse code through generic functions and classes rather than copy and paste?
2. Are there places where we can use constants/enumerations instead of hard coding?
3. Does the code perform according to the design?
4. Does the code follow SOLID design principles?
5. Can we use design patterns to structure the code better?
6. Do I understand what the test cases are testing?
7. Do we have good test coverage?

Summary

Sean

I am reviewing the code that involves sending emails. The code includes multiple classes, including an interface and an abstract class. Different types of email such as NewAccountEmail, ChangePassEmail extends NoReplyEmail which is an example of code that follows the open/closed principle of SOLID design. There is also an email handler which makes the email function much more flexible in case of any change to the requirements. The functions within these classes are easy to understand, and only have a single responsibility. The design of the code makes it easy to test. There are no comments or docstrings to explains the details, however, it is still acceptable as the code is straightforward and the variables are named such

that there is little to no confusion when trying to understand it. Overall, I think Anthony did a great job designing and coding the email functionality. The code reflects his intention of ensuring good design principles.

Anthony

I will be doing my code review on the createOptionsPane() method in OrgMainMenu.java. The function doesn't have too many lines of code and it is very easy to determine what's being done. There seems to be good design in how the tasks are allocated to an external JGuiHelper. Only note is that the function returns a JComponent which seems to be a parent of the JPanel that is being created and currently returned. There is no additional function being done by this one class, so there is no need to return the parent type of the panel in question. There are String constants within the parameters of the helper function. This doesn't seem to be a problem for a small class and the names are unique. If these buttons were reused with the same name, it might be a good idea to create a constant for it.

Andrew

I will be reviewing TEQCreateAccountPanel.java. The code is easy to understand with fields being stored in one string array, and the code being separated into its own single-functionality methods with helpful names. The code does not have much documentation since it is only there where it is needed. The code makes good use of constants and enumerations but there are still some places that could use constants instead of hard-coded values. The code can easily be tested by using the GUI but would also require knowledge of what records exist or do not exist in the database. Overall, the code is good but there may be some issues in the future if we want to store more or less information on users. This would lead to having to change several parts of the code to accommodate for this change.

Sherry

I am reviewing the code which takes care of reading Excel files. The code is easy to understand since it is separated into four different classes, where classes are subclasses of another class (e.g. there are many rows in a sheet, and there are many sheets in an Excel book). This structure is very rigid and flexible since we can call different functions according to the type. The code adheres to proper coding style and conventions. There can be more comments that describe what the functions are used for, especially on the public functions since it would be useful when other team members need to use the functions in that class. There are some switch-case statements, but it is acceptable. The code is easy to test. The code follows SOLID design principles because each class represents a different part of element in the Excel file and

each function only handles one job. There can be more methods in ExcelRow and ExcelSheet which allow outside functions to access the inner data, for example, the size of an Excel row. In general, the code is organized and clear.

Rundong

I am reviewing the sending email functionality of the application. In the email package, there is an abstract class, an interface and a client which I believe to be very good coding structure for feature extension. In each class, responsibilities are distributed to as many separate functions as possible and each function is short enough to be clear and concise. Although these is no comment and Javadoc, the code is very easy to understand. The classes also have a very good inheritance style and they properly use the observer design pattern to connect with the email client. All the classes have a good format, including proper class name and proper package name. Briefly, this piece of code is well organized and written in a precise style. I learned a lot from this code design, but the only small inconveniences I had were that some of the final string variable names are not in upper-case and some lines are extremely long (more than 80 characters) which made the code difficult to view in an editor.