CSCC01 Team Code Review

Team Name: Ctrl-Alt-Elite
Team Number: 11
Date: November 12th, 2018

Table of Contents

Code Review Videos

Code Review - Debriefing Meeting
https://www.youtube.com/watch?v=k5RTcFJyUJk

Code Review - Group Review Session
https://www.youtube.com/watch?v=_ej1VAiWz4A

Jun

For this code review, I reviewed the Command classes under model.database.api.

I didn’t have good design in the beginning, just put everything that communicates with MySQL database server in one single class QueryOnDatabase. However, it got messy very soon. I decide to using the abstract class Command that contains a abstract function handle() and let other specific Command classes (CreateCommand, SelectCommand etc.) to extend the Command class.
In this way, not only the code’s readability is improved, but also it follows the single responsibility principle.

Leo

For this code review, I review the Uploading CSV Files feature on the U3-T2-Update-CreateTemplateModelImpl.createUsingFile branch, which I implemented.

At first, I assumed that this would be a simple task to just update the existing function calls, which created a ExcelFile object to parse the template information from the Excel files, to use the CSVFile object and functions. However, I then realized, with input from Jun and Vishwa, that I would instead create an interface where both ExcelFIle and CSVFIle implement. Therefore, I created an interface called TemplateFileInterface with all same functions with respect to both file formats, where I refactored the implementations to TemplateFileExcelImpl and TemplateFileCsvImpl.

This improved the design of the code because it removed the need for large blocks of code after checking the type of file the user has uploaded and call their specific functions. Instead, it just checks initially the file type to instantiate either a TemplateFileExcelImpl or TemplateFileCsvImpl and stores it in a TemplateFileInterface. This allowed for the same functions to be called, which makes it easier to extend if needed in the future.

Vishwa

For this code review, I review the code for CreateTemplate user story which was initially thought to have a model interface like the presenter that helps communicate between presenter and the backend. After implementing the interface, I learned it was not going to be very good from a design point of view because having an interface for the model for each user story means duplicated code where the same code will go in different model interfaces if something is needed from the database like fetching template names. Therefore, after discussing it with my team members, I decided to avoid the interfaces for the model layer and instead have Usecases that can solve the purpose - getting data to and from the database using the entities. An advantage of usecases was that they did not know anything about the external agencies, view or framework. All they want is data into so they can start working. Moreover, we were able to reuse the usecases where needed without much duplicated code. Therefore, I believe thinking about a particular implementation from a design perspective right after developing really helps reveal some of the issues that can come up later in the project due to changes which are inevitable.

Tayyab

For this code review, I chose to look at the code for the User Interface classes that I implemented. After reviewing the code, I believe that it is best if I break down the giant constructors for the UI classes because as of right now the constructors for them are way too long. Breaking down the code helps improve design because it increases modularity. I noticed that many calls in the constructors are repeated in most, if not all the other UI classes so they would be better suited to a class such as UIHelpers. In addition, I noticed that there are many branches for features that have been already been developed and shipped a long time ago, so they should be deleted.

Angela

For this specific code review, I have chosen the code implemented in my branch U7-ReportsRunningSQL.
It was decided for the beginning that we were going to follow the MVP pattern. Every time a new functionality needs to be implemented, we try to keep them consistent with already existent code.

During the first read of my code, I made some style improvements and spotted a couple of duplication, for which I implemented new methods and integrate them accordingly. I reviewed the unit tests, to make sure all the main capabilities were tested and I was not missing anything important.

Made sure my code was well commented, specially each method and class to understand the behaviour of the code. Reviews and spotted any redundancies in my code and thought about any other way of coding the same functionality.
Scanned the code one more time to reaffirm that none of the already implemented methods and classes could be reused as part of my code.