GLHF - Refactoring Report

ENSE 470

April 2018

Maksym Zabutnyy

Earl Macalino

Jennifer Herasymuik

1.0 GLHF's MVP

Team GLHF have managed to complete their MVP and based on their legend there were things added to make it more complete product. Beyond the functionality of submitting and approving the form the team added notifications, sorting, tracking, feedback, and searching. Going to the homepage prompts a login, which for now is generic and signup will not be implemented until the second release. The request page is multi-functional, the user is able to submit a request using the form and also check the history of submitted tickets. The style is very crisp, simple, and neat. The form component is missing a border and does looked trumped next to the table. The roles that deal with approval also have an easy way to interface with the forms to progress the task. Overall, the release has great functionality and features, you are able to view request information that is nicely broken down.

2.0 Github Repository

Team GLHF's GitHub repository is in very good condition. The files are organized by milestones or relevant information, great organizational skills. There are a few 'README' files where necessary which usually help someone who is navigating the GitHub. One critical point is that there are two repositories for the project, one dealing with the web-app and the other with milestones and class assignments. The web-app repository contains the project and is also nicely organized but it definitely is a hindrance to have two repositories. Even though creates better order when developing, having a second repository really spreads the resources that are should be together. Being in a repository with all the information about a project but not seeing the project and having to have a different repository open is a little detrimental. Regardless, the web-app repository is also very organized and the commit messages were short and to the point.

3.0 Code and Refactoring

Overall the code is very clean. Looking at the code it is very legible and easy to understand. The front end uses xhtml, css, and javascript like any normal web app. The backend is programmed using Java, which is an easy to understand programming language that we are familiar with. All of the files were split up appropriately so no page contains a lot of code, which can be overwhelming but was not an issue here. The app uses Model View Controller, which isn't obvious at the start since the folder names don't tell you immediately whether it contains models, view, or controllers. However after some reading one can tell which folder is which. Another hige positive is how they named their functions and variables. This is very helpful, as there are no code comments in the backend to help another programmer understand the code.

Since the naming was done so well, code comments are almost not required for a lot of it. An example of this can be seen in the figure below:

```
public class AnalystDashboardBean implements Serializable{

	private Request selectedRequest;
	private List<Request> requestList;

	private User loggedUser;

	private int pendingCount;

	private Boolean finished;
	private List<String> statusList;

	public List<Request> getRequestList() {return requestList;}
	public void setRequestList(List<Request> requestList) {this.requestList = requestList;}

	public Request getSelectedRequest() {return selectedRequest;}
	public void setSelectedRequest(Request selectedRequest) {this.selectedRequest = selectedRequest;}
```

Overall the code is very easy to understand and a new programmer looking at it would be able to generally understand what is going on. The only possible code smell is the use of separate approver and analyst pages and functions. This could be considered an Object-Orientation Abuser. There are two authentication bean functions, one for analysts as shown in figure above (AnalystDashboardBean.java) and one for the approvers (ApproverDashboardBean.java). The approver dashboard bean looks very similar and has some of the exact functions as the analyst one. The code is very similar and may be able to be combined into one class. If these functions are combined, that means the two dashboard pages (one for analyst, one for approver) could also be combined since again they are very similar as well. It may be possible to create a super class and have analyst and approver be subclasses of the superclass. This could reduce the code duplication between the two dashboard beans. This could also make the Factory pattern they are using more obvious. However, this is the only code smell that could be found in the code since it is well written. More code examples can be viewed on the project repo: https://github.com/chan200v/ense470_webapp.

4.0 Design Patterns

The observer pattern is one of the patterns GLHF said they used. GLHF does in fact properly use the observer pattern, which can be viewed in the figure below:

```
public void doNotifyApprovers() {
	String text;
	for (User approver : approverList) {
		text = "Hello, "+approver.getName()+"\n\nYou have a new pending request.";
		EmailSender.SendTo(approver.getEmail(), text);
	}
}
```

We agree that they do use it since they do utilise email notifications to notify the user. The Factory Pattern is another design pattern they mentioned. They used factory pattern to instantiate different sessions with each type of users e.g(approver dashboard for approver, and requestbean for user). Not really a factory pattern because they did not utilize the interface. The last design pattern they used was the Chain of Responsibility design pattern. An example of this in their code can be seen below:

```
public void submitRequest() {
    request.setRequester(loggedUser);
    request.setStatus('P');
    request.setRequestDate(LocalDateTime.now());

    if (request.getEmail().isEmpty())
            request.setEmail(loggedUser.getEmail());

            try {
                    RequestDAO requestDAO = new RequestDAO();
            requestDAO.merge(request);
            Messages.addGlobalInfo("Successfully submitted the request.");
            }catch (RuntimeException error){
                    Messages.addGlobalError("An error has occurred submitting the request.");
                    error.printStackTrace();
            }
    request = new Request();
    getUserRequests();
    doNotifyApprovers();
}
```

They used this pattern for the request which we agree cause for example on a part of the chain an analyst could pass it to approver or cancel the request and break the chain.