**Project Report**

**DAL DISCUSSION FORUM**

For the course

**CSCI 5708: Quality Assurance**

Course Instructor

**Faisal Abbas and Robert Hawkey**

Submitted by

|  |  |  |
| --- | --- | --- |
| **V**ivek Shah | **B00799155** | **vv453822@dal.ca** |
| **S**mit Saraiya | **B00811636** | **Sm252977@dal.ca** |
| **K**ush Rao | **B00801194** | **ks927356@dal.ca** |
| **Sharon Alva** | **B00813350** | [**sh535850@dal.ca**](mailto:sh535850@dal.ca) |

**Master of Applied Computer Science**

**Faculty of Computer Science**

**Dalhousie University**

**Halifax, Nova Scotia, Canada B3H 4R2**

## How you implemented continuous integration (tools used; settings; is anything hardcoded? Why?)

Jenkins has been used to implement continuous integration in the project. A build has been configured in Jenkins that connects to the github repository as seen in Figure 1. We have maintained two different jobs one which points to git develop branch and one points to git master branch. The build that points to master has been set up such that once the build is successful it is deployed on Heroku.

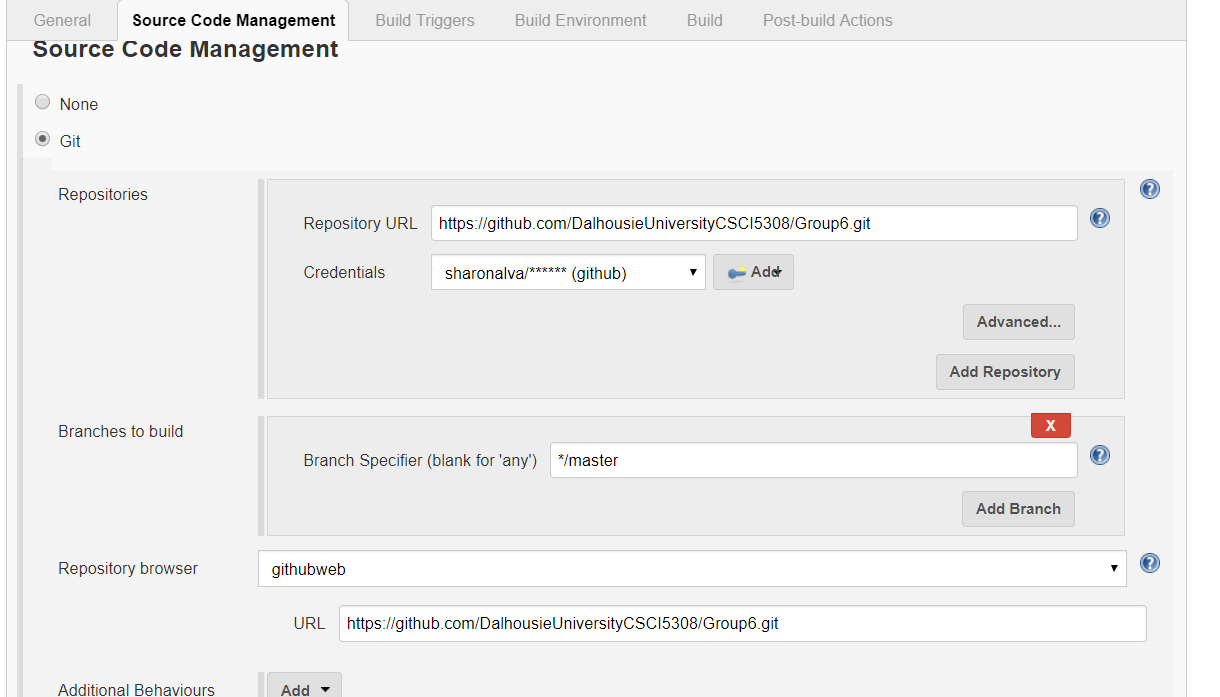


Figure 1: Link jenkins to git repo

The build is triggered every two minutes and email-ext plugin has been configured to send email to all team members for unsuccessful builds

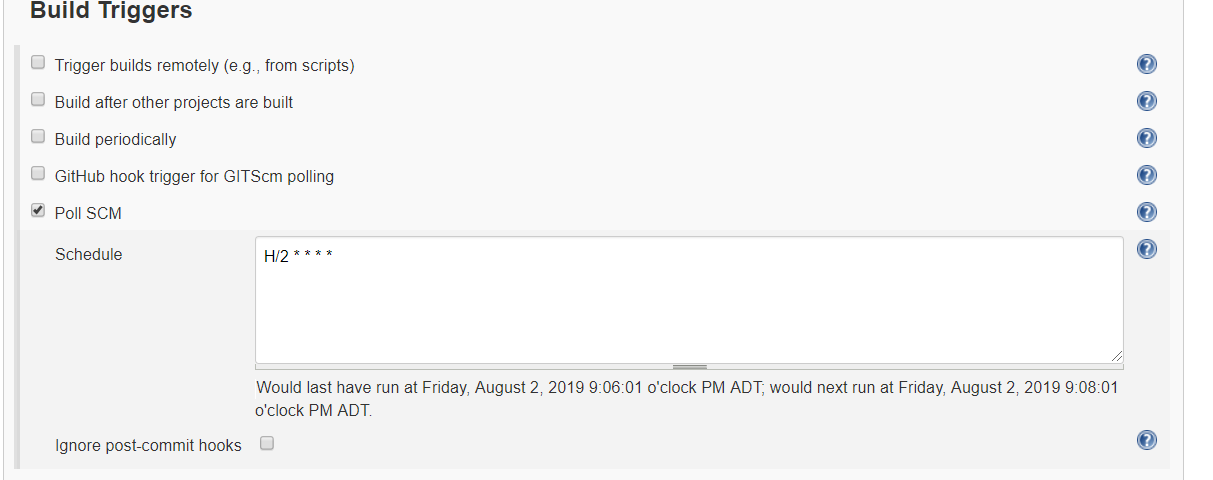


Figure 2: Build scheduled to run every two minutes

We have hardcoded the different database connection parameters in a single file. Which are commented and uncommented based on the environment in which the project is deployed in.

To resolve this problem, we tried setting up different application files for each environment such as application-dev.properties, application-pro.properties and used the spring active profile to read the correct properties file at runtime. We were not able to implement it successfully due to time constraints.

## What design patterns were used and why

* Singleton Pattern

The Singleton Design Pattern was used in the Config classes like DatabaseConfig and AppConfig. As we only needed a single instance to be created such that it is easier to manage the connections. Also, the application configuration only needs to be loaded in the start and they should be accessible throughout the application. Hence, we used Singleton Pattern

* Factory Pattern

The factory design pattern is used for instantiating object of DAO and Service Class. Using the pattern make the code more flexible. The DAO implementation from database can be changed to file by changing a single file. This makes it more flexible

* There was a possibility to use template pattern for registration module and login module both. Writing a single method for validation for both login and registration would help in code redundancy. But as login module validations were already in place, so different validations were used. This can be clubbed into one in future.

## How separation of the presentation layer, business layer and data layers were achieved

We have used Model-View- Controller architecture for separation of presentation, data and business layer.

**Model** - Model represents state of the business logic of the application

**View** - View represent the way in which we display data. In our application, we have made use of Java server pages and Thymleaf

**Controller** – The controller acts as an interface between view and model

## What naming strategy and coding conventions were agreed upon and why

For naming convention, we have used “camelCase” for maintenance and readability of the code.

## Examples of refactoring’s performed

### Extract Class

The Class PostService.java had too many methods that could be split into individual class based on their responsibility

The PostService.java class was divided into three class based on their functionality. They are CommentService, ReplyService and ImageService. This helped make the code clean and easier to understand as well as separated the concerns.

### Introduce Null Object

The PostController.java and RegisterController.java was refactored to make use of the @ModelAttribute which created an Object of the class. This helped eliminate the null checks for each individual attribute. As seen in Figure 3 the code in red show the code before refactoring and the code in green is the new clean code.

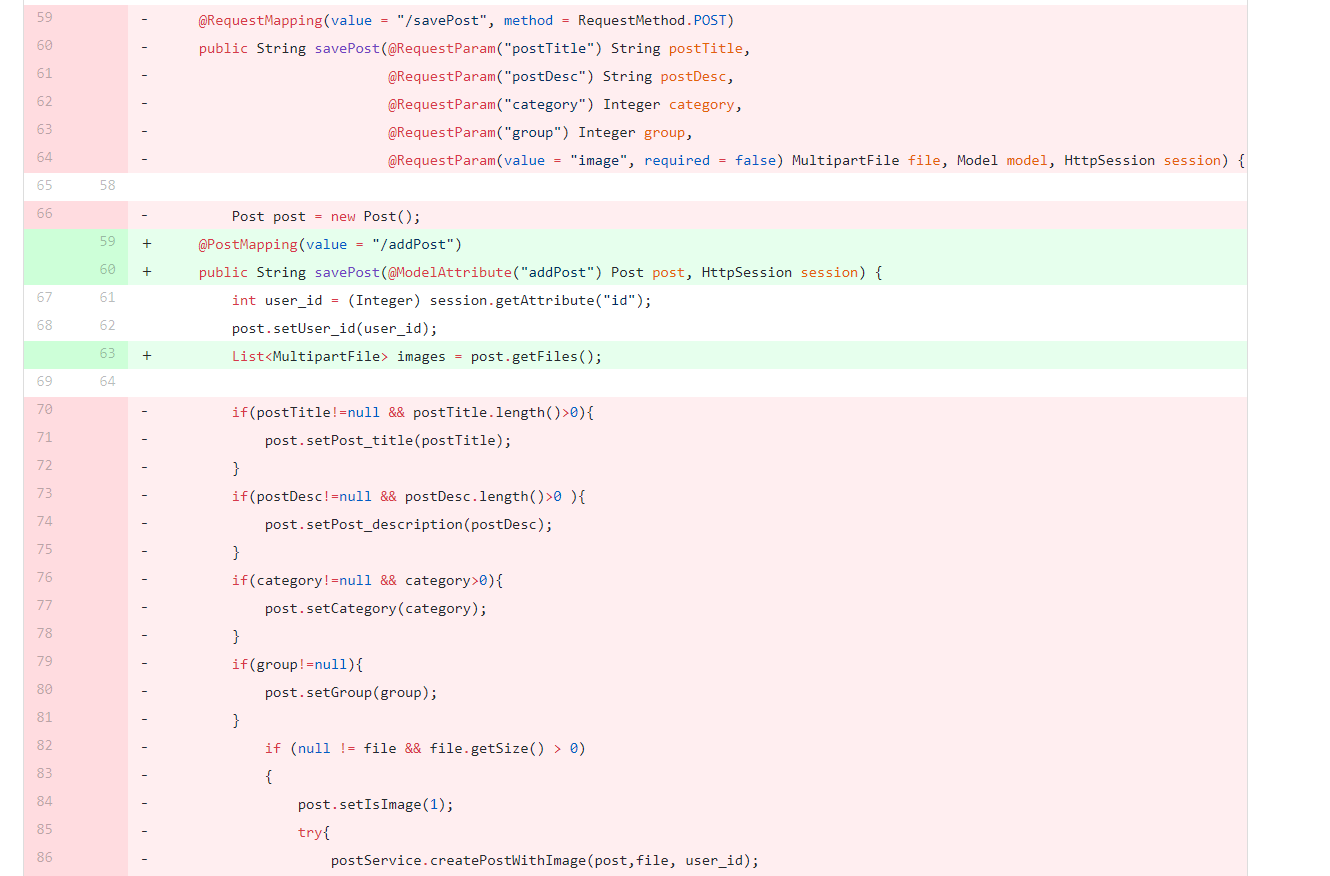


Figure 3: Insert Null Object Refactoring

### Remove Middle Man

To update user karma points originally Observer Design Pattern was used. The notify method was called to perform updateKarmaPoints in the Observer. The Observer here only acted as a Middle Man delegating the task and also added additional overhead to bind to the user for each session. Hence the Observer pattern was scrapped. It was replaced by a simple method that called the UserService to update user karma points

### Add Parameter

As seen in Figure 4, the createPostWithImage method had multiple parameters that has been replaced with a single Object



Figure 4:Refactoring Add Parameter

### Replace conditional with Polymorphism

Instead of a switch statement for password validation rules, we have created an interface and implement it to the individual password validation rule classes.

## A list of what is considered technical debt in the current state of the project and how these could be resolved

|  |  |  |
| --- | --- | --- |
| File Name | Technical Debt | Solution |
| AmazonClient.java | * The class has all the configuration as well as implementation of methods. The class violates Separation of Concern. * The folder based on environment is hardcoded into the code. | * The Class can be split into two classes one for Configuration and one for implementing the method to upload images on cloud * The environment value can be set in the application configuration file and be accessed in the code, making the code clean and reducing the chances of error |
| UserService.java  Method: updateUserKarmaPoints | The method takes input parameter as karmaPoints and post id both do not uniquely identify the user. The method can be moved to another class | The method can be moved to another class that only updates the karma points of the user based on the post id. A stored procedure that retrives the user details and performs the update of karma points based on post id can be written. By this the method performs that task with the values provided as input making it simpler to understand |
| UserDAO.java | The sql insert statement could be replaced with procedure call | The sql statemets could be replaced with procedure call making the code less prone to SQL injection |
| All DAO Classes | There is a common pattern of setting parameters in the SQL statement and reading values from the resultset. | This can be solved by using the template method design pattern |
| PersonalGroupController.java | If else condition to check subscription could be replaced by polymorphism | The conditional smell cold be removed by state design pattern |

* In Database config class the loggers could have been more specific just as the DAO classes. Custom log messages for the exceptions could have been done.
* Refactoring could be done in more detail manner for the DAO classes. Could have used Row mapper for storing the result set and then could minimized the code duplication.
* Composite Design pattern could also have been implemented for Posts – Comments – Replies. As they form a hierarchy, Post would be the root node and reply would be the leaf node. It would make the code more flexible and manageable
* The state Design Pattern could have been implemented for the approval of the pending request to join a personal group. Initially the user’s request will be in pending state, and then after the approval by admin it will be in APPROVED state.

## A clear list of each member’s contribution to the project. Each student must clearly list all the classes, presentation layer files and stored procedures they wrote for the project. If there is doubt on the authorship of code, git history will be considered the final source of truth. Because of this, it is essential that you understand how to use git and do all your own git work. Do not give code to other group members to commit for you.

## The list must clearly label each .java file and who wrote it

## If multiple people worked on a given .java file, list which methods in the class were written by which group member

|  |  |  |  |
| --- | --- | --- | --- |
| **Author Name** | **File Name** | **Methods** | **Stored Procedure** |
| Sharon Alva | PostController.java |  |  |
|  | PostDAO.java | createPost  getAllActive  updatePostStatus | addPost  getAllActivePosts  updatePostStatus |
|  | IPostDAO.java |  |  |
|  | PostImageDAO.java |  | addImage  getImagesByPostId |
|  | IPostImageDAO.java |  |  |
|  | UserDAO.java | getUserIdByPostID  getOriginalKarmaPoints  updateUserKarmaPoints |  |
|  | PostService.java | createPost  createPostWithImage  updatePostStatus  getAllActivePosts  getInactivePosts |  |
|  | IPostService |  |  |
|  | AppConfig.java | getIntValue  getValue |  |
|  | AmazonClient.java |  |  |
|  | IImageService.java |  |  |
|  | ImageService.java |  |  |
|  | IUserService.java | updateUserKarmaPoints |  |
|  | UserService.java | updateUserKarmaPoints |  |
|  | PostImage.java |  |  |
|  | PostServiceTest.java |  |  |
|  | ImageServiceTest.java |  |  |
|  | PostDAOMock.java |  |  |
|  | PostImageDAOMock.java |  |  |
|  | ControllerException.java |  |  |
|  | DAOException  .java |  |  |
|  | ServiceException.java |  |  |
|  | Post.jsp |  |  |
| Smit Ashish Saraiya | DAOFactory.java |  |  |
|  | IDAOFactory.java |  |  |
|  | IServiceFactory.java |  |  |
|  | ServiceFactory.java |  |  |
|  | AdminDAO.java |  | getAdmin  fetchAllSubscriptionRequests  approveSubscriptionRequest  getPostsByMaxReports |
|  | IAdminDAO.java |  |  |
|  | CommentDAO.java |  | getCommentsByPostId  addComment  getCommentByName |
|  | ICommentDAO.java |  |  |
|  | HomeDAO.java |  | getAllPosts  addReportingPost  fetchreportedPostsByUserID  getSearchPost  getPostsByGroupId |
|  | IHomeDAO.java |  |  |
|  | PersonalGroup.java |  |  |
|  | IPersonalGroup.java |  |  |
|  | ReplyDAO.java |  | addReply |
|  | IReplyDAO.java |  |  |
|  | SubscriptionDAO.java |  | getSubscriptionGroupList  addSubscriptionRequest  fetchSubscriptionByUserId  fetchAllApprovedRequests  fetchSubscriptionByID |
|  | ISubscription.java |  |  |
|  | DatabaseConfig.java |  |  |
|  | Admin Controller.java |  |  |
|  | Customerror Controller.java |  |  |
|  | Homepage Controller.java |  |  |
|  | PersonalGroup Controller.java |  |  |
|  | Post Detail Controller.java |  |  |
|  | Subscription Controller.java |  |  |
|  | Subscription Detail Controlller.java |  |  |
|  | Views.java |  |  |
|  | Admin Service.java |  |  |
|  | IAdminService.java |  |  |
|  | HomeService.java |  |  |
|  | IHomeService.java |  |  |
|  | PersonalGroupService.java |  |  |
|  | IPersonalGroupService.java |  |  |
|  | SubscriptionService.java |  |  |
|  | ISubscriptionService.java |  |  |
|  | CommentDAOMock.java |  |  |
|  | CommentServiceTest.java |  |  |
|  | SubscriptionDAOMock.java |  |  |
|  | SubscriptionServiceTest.java |  |  |
|  | HomeDAOMock.java |  |  |
|  | PostServiceTest.java |  |  |
|  | PostDAOMock.java |  |  |
|  | CommentTest.java |  |  |
|  | ReplyTest.java |  |  |
|  | PostTest.java |  |  |
|  | SubscriptionTest.java |  |  |
|  | CommentServie.java |  |  |
|  | ICommentService.java |  |  |
|  | ReplyService.java |  |  |
|  | IReplyService.java |  |  |
|  |  |  |  |
| Vivek Shah | DashboardDAO.java |  | getPostsByUserID  deletePostById  updatePostById  getPostByUserID  getPostByGroupId |
|  | IDashboardDAO.java |  |  |
|  | RegisterDAO.java |  |  |
|  | IRegisterDAO.java |  |  |
|  | DashboardController.java |  |  |
|  | RegisterController.java |  |  |
|  | User.java |  |  |
|  | DashboardService.java |  |  |
|  | IDashboardService.java |  |  |
|  | RegisterService.java |  |  |
|  | IRegisterService.java |  |  |
|  | RegisterValidator.java |  |  |
|  | ValidationCode.java |  |  |
|  | DigitRule.java |  |  |
|  | EmailValidator.java |  |  |
|  | Length.java |  |  |
|  | LowerCaseRule.java |  |  |
|  | NonAlphanumeric.java |  |  |
|  | IPasswordRule.java |  |  |
|  | PasswordValidator.java |  |  |
|  | StringRules.java |  |  |
|  | UpperCaseRule.java |  |  |
|  | validation.properties |  |  |
|  | UserTest.java |  |  |
|  | DashboardDAOMock.java |  |  |
|  | DashboardServiceTest.java |  |  |
|  | RegisterDAOMock.java |  |  |
|  | RegisterServiceTest.java |  |  |
| Kush Rao | UserDAO.java |  | updateUser |
|  | VerificationDAO.java |  |  |
|  | ProfileController.java |  |  |
|  | PublicController.java |  |  |
|  | User.java [ entities] |  |  |
|  | Verification.java [entities] |  |  |
|  | User.java [model] |  |  |
|  | SendForgotPasswordEmailResponse.java |  |  |
|  | SendForgotPasswordEmailRequest.java |  |  |
|  | SendEmailResponse.java |  |  |
|  | SendEmailRequest.java |  |  |
|  | ResetPasswordResponse.java |  |  |
|  | ResetPasswordRequest.java |  |  |
|  | LoginRequest.java |  |  |
|  | LoginResponse.java |  |  |
|  | LoginService.java |  |  |
|  | ResetPasswordService.java |  |  |
|  | SendEmailService.java |  |  |
|  | SendForgotPasswordEmailService.java |  |  |
|  | AppConfig.java |  |  |
|  | AuthenticationProvider.java |  |  |
|  | WebSecurityConfigurer.java |  |  |
|  | ServletInitializer.java |  |  |
|  | DaldiscussionApplication.java |  |  |
|  | UserService.java |  |  |
|  |  |  |  |