****

**Faculty of Computer Science**

**Dalhousie University**

**CSCI 5308**

**Quality Assurance**

**Project Report**

**Job Portal - MyCareer**

**Submitted By: (Group 8)**

**Abinaya Raja - B00799562 (abinaya.raja@dal.ca)**

**Aniruddha Chitley - B00808320 (aniruddha.chitley@dal.ca)**

**Jessica Castelino- B00804805 (jessica.castelino**[**@dal.ca**](mailto:ys273139@dal.ca)**)**

**Sarmad Noor-B00799557 (**[**sarmad.noor@dal.ca**](mailto:sarmad.noor@dal.ca)**)**

# Introduction

We have developed a job portal known as MyCareer for the university students who are looking for full time as well as co-op job opportunities. The target user base for our application are the graduate Computer Science students studying at Dalhousie University. There are three user personas involved in this project – Co-op coordinators, employers and students. The functionalities of each of these user personas is mentioned below.

**Co-op administrators:**

1. Co-op administrators can login to the website.
2. Authenticated co-op administrators can perform the following tasks:
   * 1. Approve or decline registration requests from employers after background check
     2. Add new students to give them access the portal
     3. Revoke the access of students to the portal
     4. Delete the employer to revoke their access from the portal

**Employers:**

1. The employers can register themselves on the website
2. The employers who have received an approval from co-op administrators can login to the portal and perform the below tasks:
   * 1. Post new jobs
     2. View jobs posted in the past
     3. Close a job posting that is already created, temporarily close a posted job, and permanently delete the job posting
     4. Edit an existing job posting
     5. View the details of the students who have applied for the job and access the application package (resume, cover letter, etc)
     6. Change the status of the applicant (under review, accepted, rejected, etc)
     7. Filter applicants based on the application status
     8. Manage the company profile information and personal information

**Students:**

1. Students can login to the website
2. Authenticated students can perform the following tasks:
   * 1. View the job postings posted by employers
     2. View the jobs they have applied for in the past along with the status
     3. Apply for jobs and upload relevant documents
     4. Filter jobs based on location and co-op work term

# Tools used

* Eclipse, VSCode – IDE for development
* MySQL workbench – connecting to MSQL remote database
* PUTTY – to connect to Azure remote server.
* GitBash

## Client-Side Technology:

* HTML
* CSS
* JQuery
* JSP

## Server-Side Technology:

* Java

## Framework:

* Spring Boot

## Database:

* MySQL

## Source Code Control:

* Github

## Continuous Integration Tool:

* Jenkins

## Ticket Controlling Tool:

* Trello

## Cloud Server Used:

* Heroku
* Azure (for demonstration of server logs)

# Continuous Integration Implementation

Jenkins is used for continuous integration and deployment on Heroku cloud.

The build and deployment steps are not hardcoded, but they are configured in Jenkins configuration console.

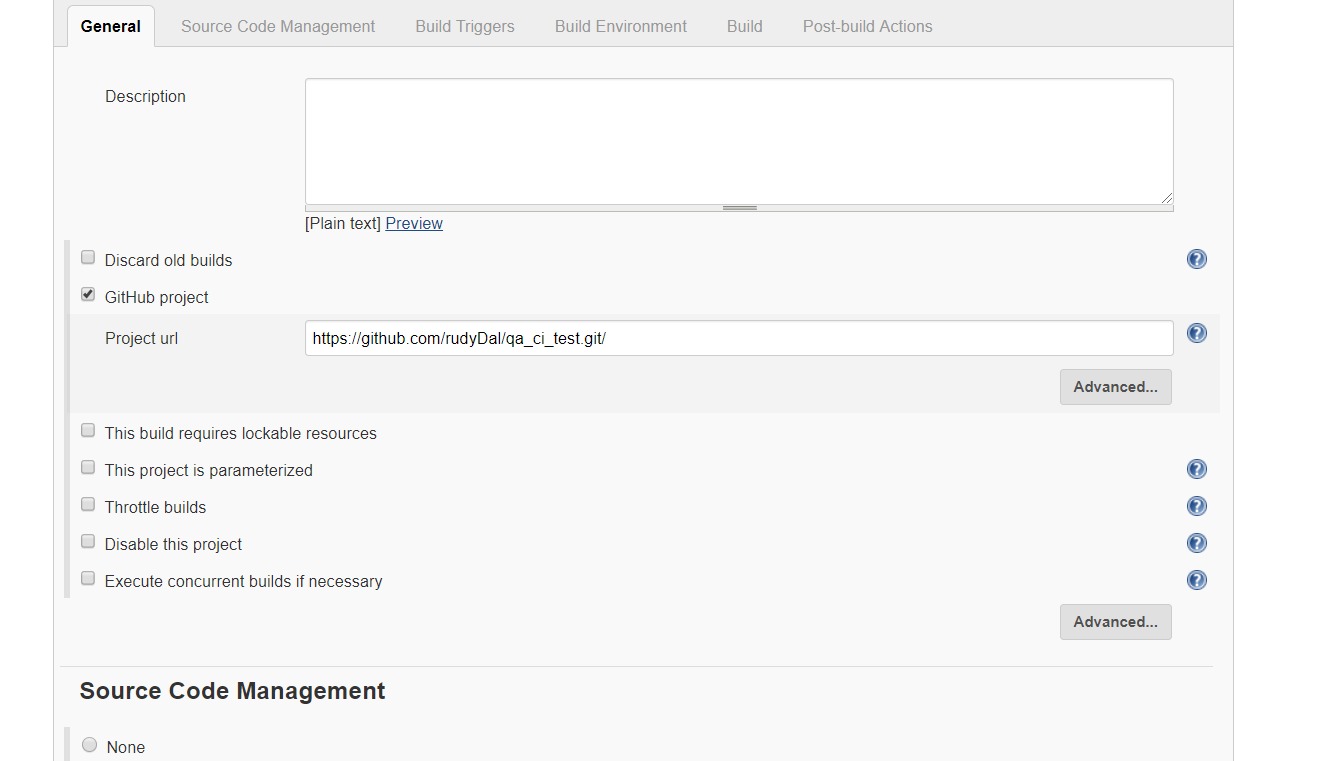


Figure : Git repo URL to fetch the code

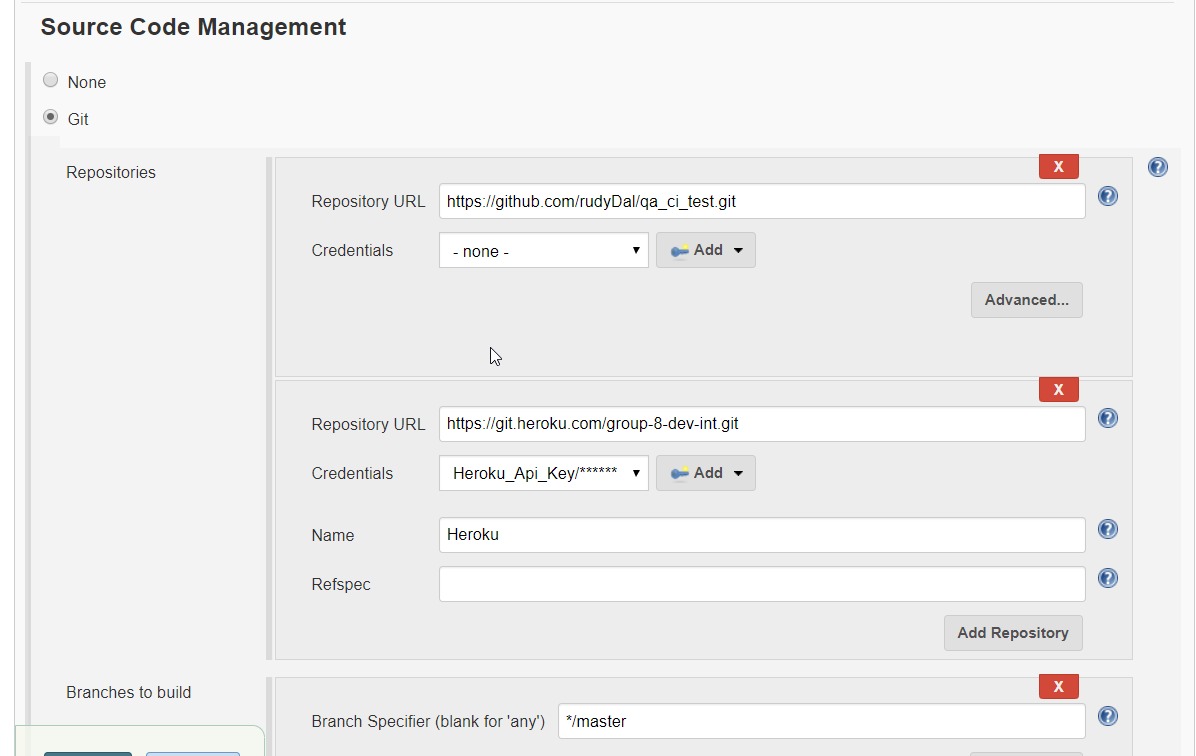


Figure : Code repository and cloud URL

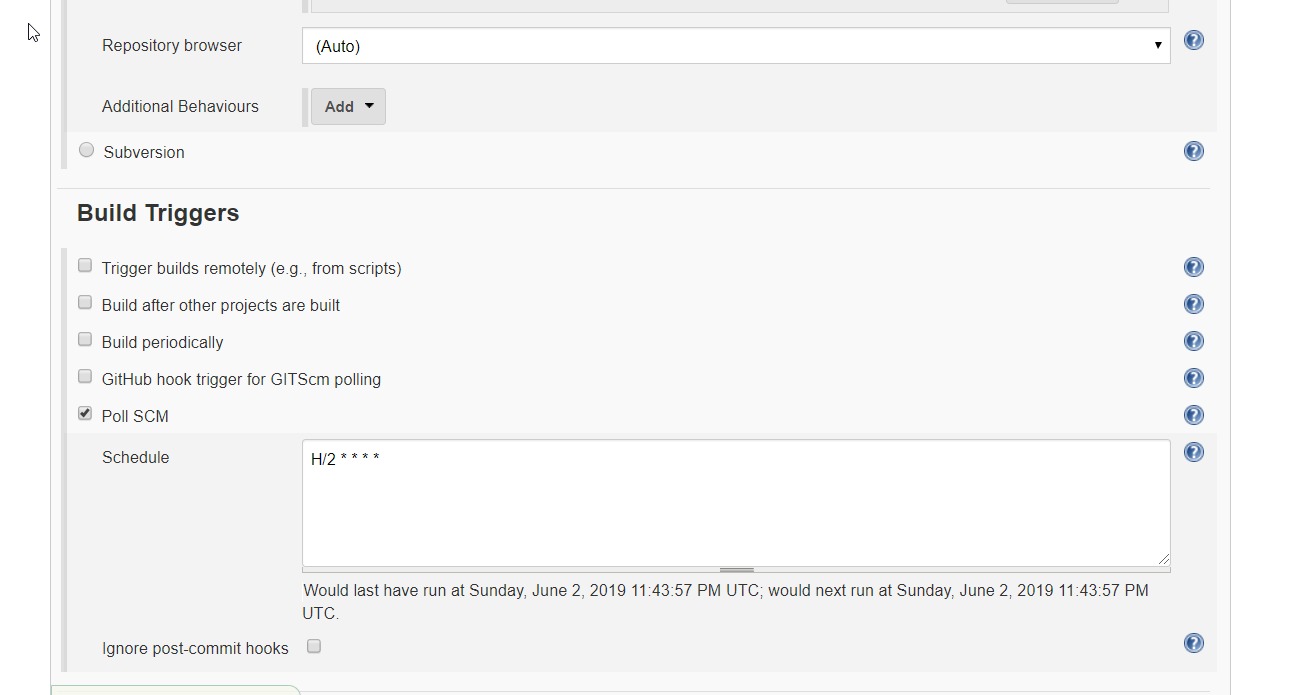


Figure : Build Triggers

# Configuration based Logic

**Use case 1:** Showing jobs to students based on pre-requisite courses for the job matching the student’s completed courses.

**Use case 2:** Showing all the courses to the student posted by employers on My Career portal.

**myCareer.properties**

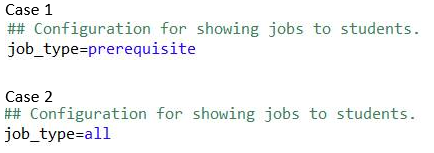


Figure : Configuration values



Figure : Folder structure for configurable business logic

**FetchAllJobs.java** is the class corresponding to the configuration “all” and **FetchPrequisiteCourseJobs.java** is the class corresponding to the configuration “prerequisite”. In the **ConfigLogicClassLoader.java,** a Hash map is defined which consists of configuration and corresponding class mapping. Now based on the value of the “job\_type” property in **mycareer.properties** file, the object of relevant class is fetched from the HashMap and correct procedure is returned.

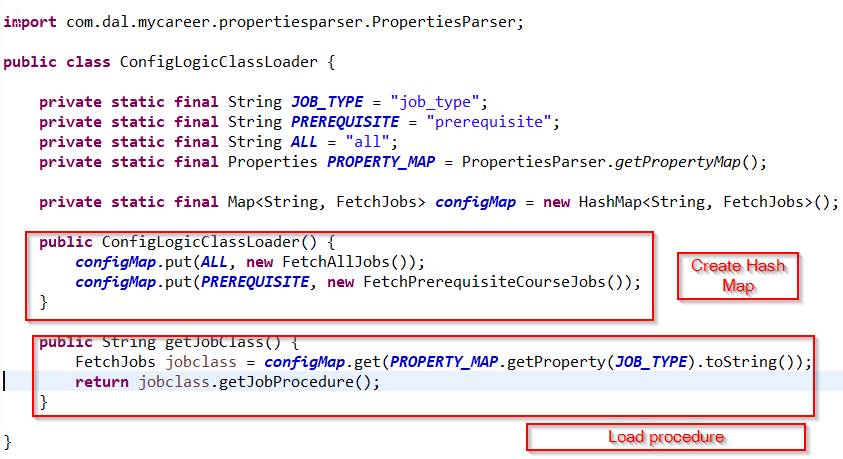


Figure : Configuration Loader

# Design Patterns

* **Template method pattern**

Template method pattern is used to reuse the code for creating connection, preparing call statement, and execution of stored procedure; meanwhile facilitating the type of operations to be performed and ways to handle the parameters and results to be decided at runtime. We choose to use template method in this case because it avoids repeating of code wherever data is to be stored to or retrieved from the database. It facilitates a centralized class where the common activities can be performed, thereby reducing the number of files/classes/methods to be changed for any reason.

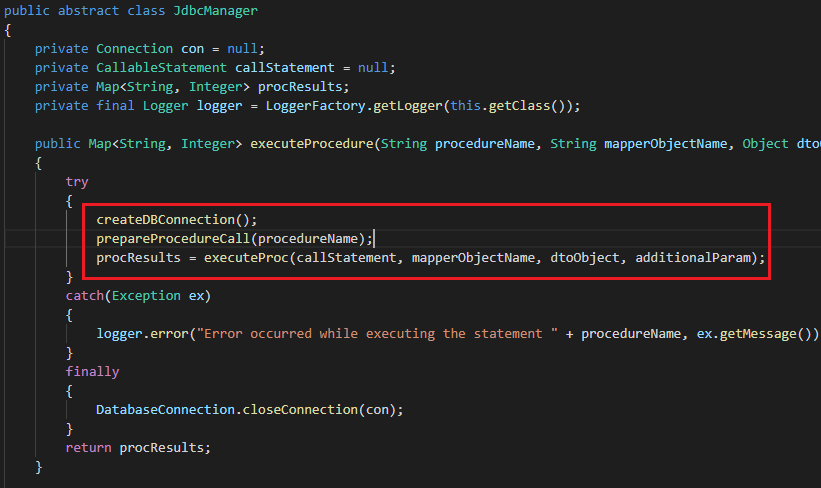


Figure : Template method design pattern

JdbcManager abstract class contains the “executeProcedure” that contains common tasks to be performed while executing the store procedure.

We created 4 classes InsertHandler, SelectHandler, UpdateHandler, and DeleteHandler to perform the operations insert, select, update, and delete respectively. These classes have their own way of performing the operations it is supposed to do.

* **Singleton pattern**

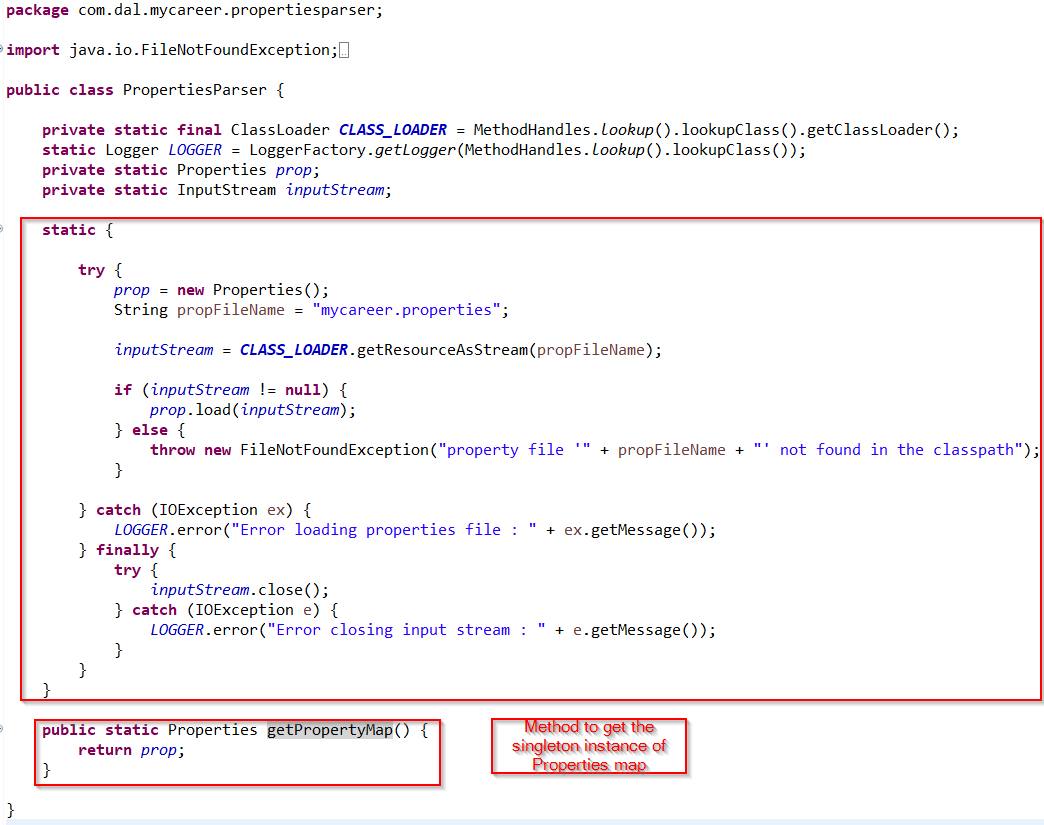


Figure : Singleton design pattern in PropertiesParser.java class

In the above case the when the classes loader loads the **PropertiesParser.class**, a Properties map of **mycareer.properties** file is created and this singleton instance is used by other classes reading from the properties map.

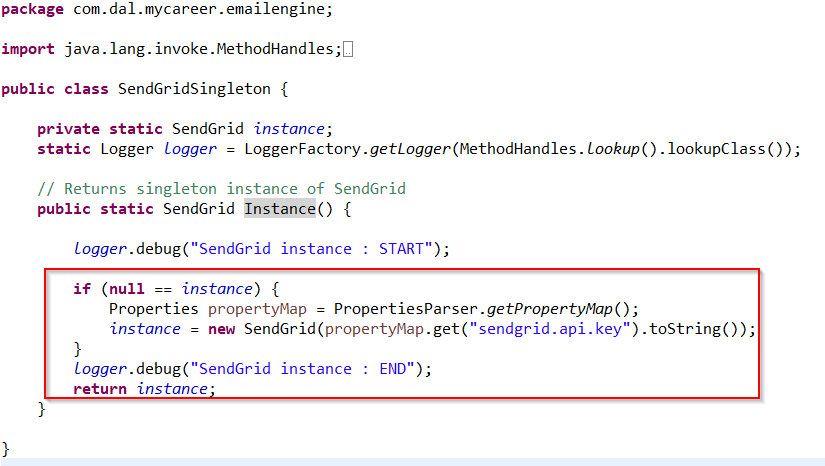


Figure : Singleton design pattern in SendGrid class

The Instance() method returns a singleton instance of third-party email service (SendGrid) which is used by other methods to send email to Students and Employers after successful onboarding.

# Separation of Layers

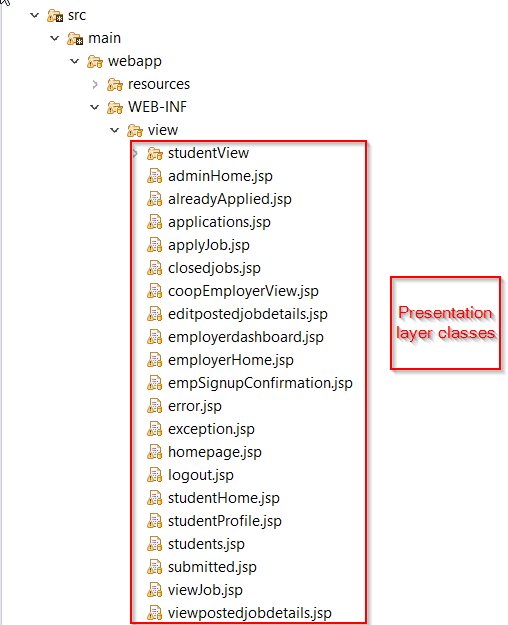


Figure : Presentation Layer

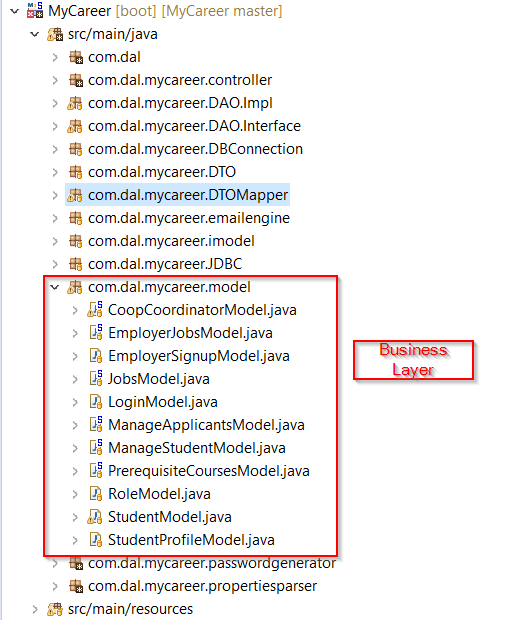


Figure : Business Layer

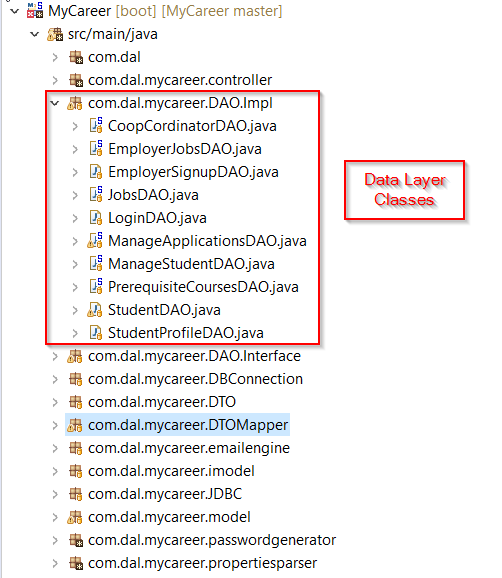


Figure : Data Layer classes

A screenshot of a cell phone

Description automatically generated

Figure : Database UML diagram

Unit test case overview

We have 36 test cases inside the hierarchy **src/test/java**. The package for each test class is same as that of the Java class for that test.

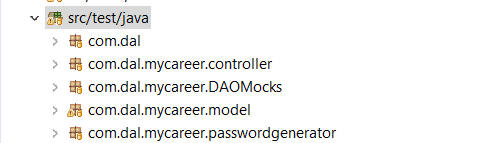


Figure : JUnit test classes hierarchy

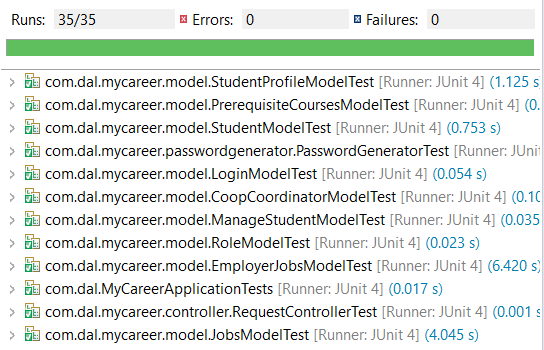


Figure : Test case execution report

# Naming/Spacing Convention

We have agreed to use camel case for method names and variable names and Pascal case for class names. We agreed to have a single line of space between each method.

# Refactoring performed

* Reduced the duplicate code for executing stored procedures by implementing template method pattern.
* Created class level constants for frequently used string in the class rather than hardcoding the string everywhere in the class.
* Created separate classes for various models and moved the methods to these classes. In order to avoid Single responsibility principle violation by any model classes.
* Moved the common fragment of code outside the conditional fragments.
* Extracted variables from complicated expressions in a temporary variable that explains its purpose.
* Grouped parameters and introduced a parameter object wherever possible.
* Replaced error code with exceptions that are handled by a Global Exception Handler.
* Created getting and setting methods to access fields of an object so that coupling can be removed.
* Split temporary variables to ensure that every assignment a separate variable exists, unless it is not a loop variable.

# Technical debt

* We almost completed the entire project when we learnt about the design patterns. The T**emplate method design** pattern we implemented in the project requires complete rewrite of data layer code. Since we did not get enough time to consume the classes that we introduced to perform database operations throughout the project, we have completed the refactoring of the data layer code in 50% of the data layer classes and demonstrated the implementation of Template method design pattern in **Employer JobDAO.java.** Due to time crunch we could not implement this Design pattern in other DAO classes.

We have generalized the classes (**InsertHandler.java, SelectHandler.java, UpdateHandler.java, and DeleteHandler.java**) and methods that we introduced to handle the database operations to support the execution of stored procedures that perform any DML operations. Hence, the refactoring of the rest of the data layer classes can be completed easily by consuming these classes.

# Member’s Contribution

|  |  |  |
| --- | --- | --- |
| **Member Name** | **List of classes** | **Methods** |
| Abinaya Raja | EmployerJobsController.java | activeJobs |
|  | getActiveJobs |
|  | getClosedJobs |
| JobsController.java | closeJob |
|  | openJob |
| ManageApplicationsController.java | viewApplicants |
|  | updateApplicationStatus |
| ManageStudentController.java | RegisterStudent |
|  | DeleteStudent |
| EmployerJobsDAO.java | getActiveJobs |
|  | getClosedJobs |
|  | fetchJobByStatus |
| JobsDAO.java | updateJobStatus |
| ManageApplicationsDAO.java | getApplications |
|  | updateApplicationStatus |
| ManageStudentDAO.java | RegisterStudent |
| PrerequisiteCoursesDAO.java | addStudentCompletedPrereq |
| JobsMapper.java |  |
| StudentDetailsMapper.java |  |
| StudentsMapper.java |  |
| DeleteHandler.java |  |
| ProcedureParamLoader.java |  |
| SelectHandler.java |  |
| UpdateHandler.java |  |
| EmployerJobsModel.java | getActiveJobs |
|  | getClosedJobs |
| JobsModel.java | updateJobStatus |
| ManageApplicationsModel.java | getApplications |
|  | updateApplicationStatus |
| ManageStudentModel.java | RegisterStudent |
| EmployerJobsDAOMock | getActiveJobs |
|  | getClosedJobs |
| JobsDAOMock.java |  |
| ManageStudentDAOMock.java | setLstRegisteredStudent |
|  | getLstRegisteredStudent |
|  | RegisterStudent |
|  | DeleteStudent |
| PrerequisiteCoursesDAOMock.java |  |
| EmployerJobsModelTest.java | getActiveJobsTest |
|  | getClosedJobsTest |
| ManageStudentModelTest.java | RegisterStudentSuccessfulRegistrationTest |
|  | DeleteStudentTest |
|  | DeleteStudentCountTest |
|  | DeleteStudentAndFindTest |
| JobsModelTest.java | updateJobStatusToOpenTest |
|  | updateJobStatusToClosedTest |
| Aniruddha Chitley | PropertiesParser.java |  |
| ConfigLogicClassLoader.java |  |
| FetchAllJobs.java |  |
| FetchJobs.java |  |
| FetchPrerequisiteCourseJobs.java |  |
| StudentProfileModel.java |  |
| RoleModel.java |  |
| LoginModel.java |  |
| PasswordGenerator.java |  |
| EmployerApprovalEmail.java |  |
| EmployerRejectionEmailImpl.java |  |
| SendGridSingleton.java |  |
| SendGridEmailService.java | SendHTML() |
| SendEmail() |
| StudentProfileController.java |  |
| LoginController.java | Login() |
| RequestController.java | LoadHome() |
| EmployerHome() |
| Logout() |
| EmployerSignUpConteroller.java |  |
| LoginDAO.java |  |
| StudentProfileDAO.java |  |
| DatabaseConnection.java | GetConnection() |
| CloseConnection() |
| CloseDatabaseComponents() - overloaded method |
| Jessica Castelino | RecruiterRequestsController |  |
| GlobalExceptionController |  |
| StudentJobsController |  |
| StudentJobApplicationController |  |
| ManageApplicationsController | downloadResume() |
| JobsController | viewStudentSelectedJob |
| RecruiterRegistrationRequestDAO |  |
| ManageApplicationsDAO | fetchDocument |
| StudentApplicationDAO |  |
| StudentDetailsDAO |  |
| StudentJobsDAO |  |
| JobsDAO | fetchJob |
| ManageApplicationsModel | downloadFile |
| RecruiterRegistrationRequestModel |  |
| JobsModel | fetchJob |
| StudentApplicationModel |  |
| StudentJobsModel |  |
| Sarmad | ManageActiveRecruitersController.java |  |
| EmployerJobsController.java | saveJob |
|  | viewPostedJob |
|  | editPostedJob |
|  | updateJobDetails |
| ManageStudentController.java | getRegisteredStudents |
| PrerequisiteCoursesController.java |  |
| ManageActiveRecruitersDAO.java |  |
| EmployerJobsDAO.java | InsertJobDetails |
|  | viewPostedJobDetails |
|  | updatejobDetails |
| ManageStudentDAO.java | getRegisteredStudents |
|  | isNewStudent |
| PrerequisiteCoursesDAO.java | insertJobPrerequisiteCourses |
|  | getPrerequisiteCourses |
| DatabaseConnection.java | closeDatabaseComponents |
| DTOMapper.java |  |
| JobDetailsMapper.java |  |
| JdbcManager.java |  |
| InsertHandler.java |  |
| ManageActiveRecruitersModel.java |  |
| EmployerJobsModel.java | InsertJobDetails |
|  | viewPostedJobDetails |
|  | updateJobDetails |
| ManageStudentModel.java | getRegisteredStudents |
| PrerequisiteCoursesModel.java |  |
| ManageActiveRecruitersDAOMock.java |  |
| EmployerJobsDAOMock.java | InsertJobDetails |
|  | viewPostedJobDetails |
|  | updatejobDetails |
| ManageStudentDAOMock.java | getRegisteredStudents |
| ManageActiveRecruitersModelTest.java |  |
| EmployerJobsModelTest.java | viewPostedJobDetailsTest |
|  | updatejobDetailsTest |
|  | InsertJobDetailsTest |
| ManageStudentModelTest.java | getRegisteredStudentsCountTest |
|  | getRegisteredStudentsAfterRegistrationTest |
| PrerequisiteCoursesModelTest.java | getPrerequisiteCourses |

|  |  |
| --- | --- |
| **Member Name** | **List of Presentation layer** |
| Abinaya Raja | employerHome.jsp (job list base page) |
| closedjobs.jsp |
| applications.jsp |
| students.jsp (Register student popup) |
| Aniruddha Chitley | homepage.jsp |
| logout.jsp |
| error.jsp |
| StudentProfile.jsp |
| Jessica | adminHome.jsp |
| studentHome.jsp |
| applyJob.jsp |
| alreadyApplied.jsp |
| Exception.jsp |
| submitted.jsp |
| viewJob.jsp |
|  |
| Sarmad | employerHome.jsp (create job pop up) |
| editpostedjobdetails.jsp |
| viewpostedjobdetails.jsp |
| coopEmployerView (active employer) |
| students.jsp (Registered students list base page) |

|  |  |
| --- | --- |
| **Member Name** | **List of Stored procedures and SQL Functions** |
| Abinaya Raja | insertStudentCompletedCourses |
| sp\_insertStudent |
| sp\_updateApplicationStatus |
| getActiveJobsForEmployer |
| getClosedJobsForEmployer |
| sp\_updateJobStatus |
| sp\_deleteStudent |
| getRequiredCoursesForJob (function) |
| getCompletedCoursesForStudent (function) |
| Aniruddha Chitley | IsValidLogin |
| studProfile |
| Jessica | fetchJob |
| fetchDocument |
| alreadyApplied |
| withdrawApplication |
| applyForJob |
| fetchStudentDetails |
| getAppliedJobList |
| getAllJobList |
| Sarmad | checkDupicateStudent |
| fetchActiveRecruiters |
| fetchRegisteredStudents |
| getPostedJobDetails |
| updatejobdetails |
| sp\_insertjobdetails |
| sp\_deleteActiveEmployer |
| sp\_getPrerequisiteCourses |
| insertjobRequirementRecord |
| getCompletedCoursesIds (function) |

Other Project related Tasks:

* Aniruddha
  + CI/CD setup with Jenkins and Heroku.
  + Logging Implementation.
* Jessica
  + CI/CD setup with Jenkins and Heroku.
  + Deployment on Azure as logs were not visible on Heroku
* Sarmad
  + Database setup
* Abinaya
  + Database setup