Software Requirements Specification for

Robust Systems Student Employment Agency

<Version 2.1>

Prepared by Andrew Er

RMIT University
April 2012

Revision History

Date	Version	Description	Author
20 March 2012	<1.0>	First Draft	Andrew ER
27 March 2012	<1.1>	Added Major Part Second Draft	Andrew ER
22 April 2012	<2.0>	Added Major Parts based on the requirements	Andrew ER
25 April 2012	<2.1>	Fix classes and added description	Andrew ER

Table of Contents

Software Requirements Specification

1. Introduction

1.1 Purpose

The purpose of this document is to describe the software requirement specifications for Robust System Student Employment Agency. The intended audience of this document includes the administrator, employers and the students using this system.

1.2 Scope

The software system to be produced is a Robust System Student Employment Agency, which will be referred as “RSSEA” thoughout this document. This program offers a complete and easy way for employers to give an opportunity for students to get hired. The goal of this is that the system will manage all the job lists, which are requested by the employers, and it is available for display by students who are looking for a job.

1.3 Definition, Acronyms and Abbreviations

“RSSEA” - Robust System Student Employment Agency

1.4 Overview

1.4.1 Introduction:

Provide an overview of the application, describe the document structure and point the individual objectives.

1.4.2 Overall Description:

Provide the specification of the system model, the classes’ model, and the list any assumed factors that used within this document.

1.4.3 Other Nonfunctional Requirements

Provide some other constraints that apply to factors such as performance.

1.5 Intended Audience

This document is intended for all users including students, employers, and administrators. The user needs to understand the basic system architecture and its specifications. Below are the probable uses for each type of users.

Students : The students who are seeking for a job can review a list of jobs available for the students. Students may also upload their resume into the system for reference.

Employer : The employer may create jobs into the job list for student to search

Administrator: Administrator manages the system by maintaining records such as the employer, student, job and resume.

2. Overall Description

The Robust Student Employment Agency is a new system that replaces the current manual telephone and mail service for students who wish to look for a job. The context diagram in Figure 1 illustrates the external entities. The system interface for the first release <v1.0> will be shown in this documentation. The system is expected to evolve over several releases with more user-friendlier interface as well as a tighter security giving user easy access to the system.

2.1 Use Case Model Survey

2.1.1 Use-case diagram

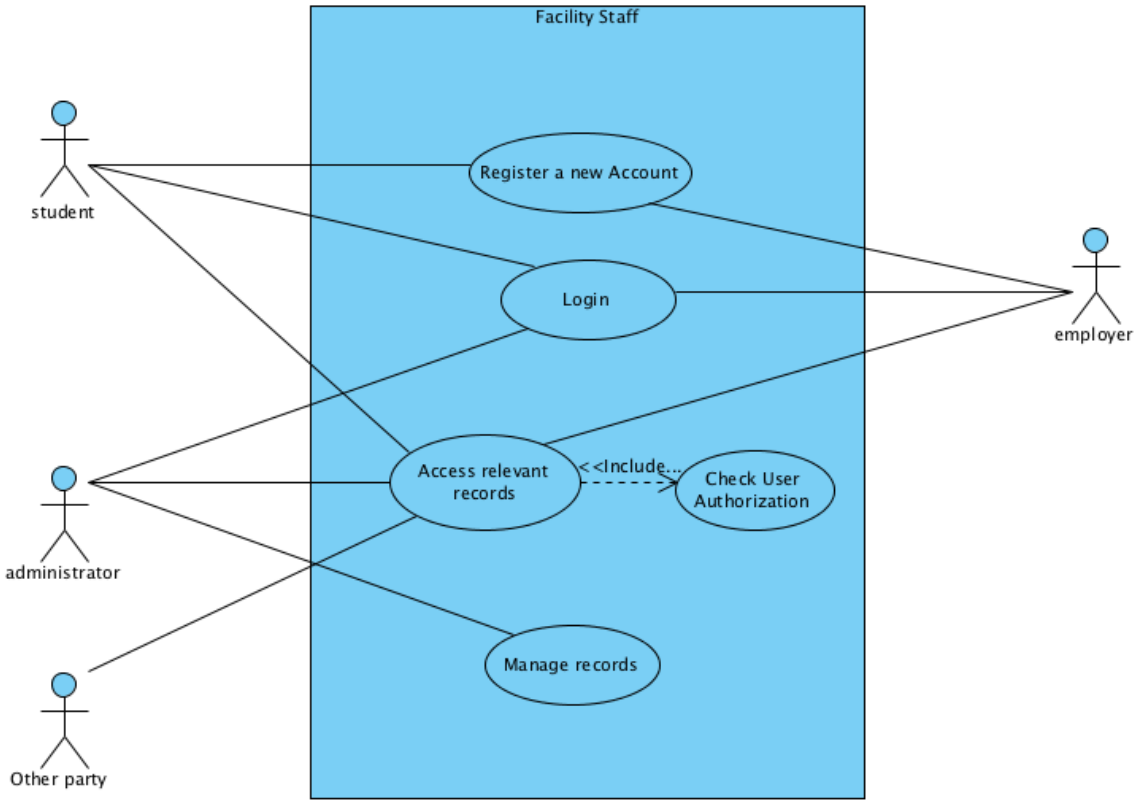


Figure 1.1 Use case diagram of the Facility Staff with the System

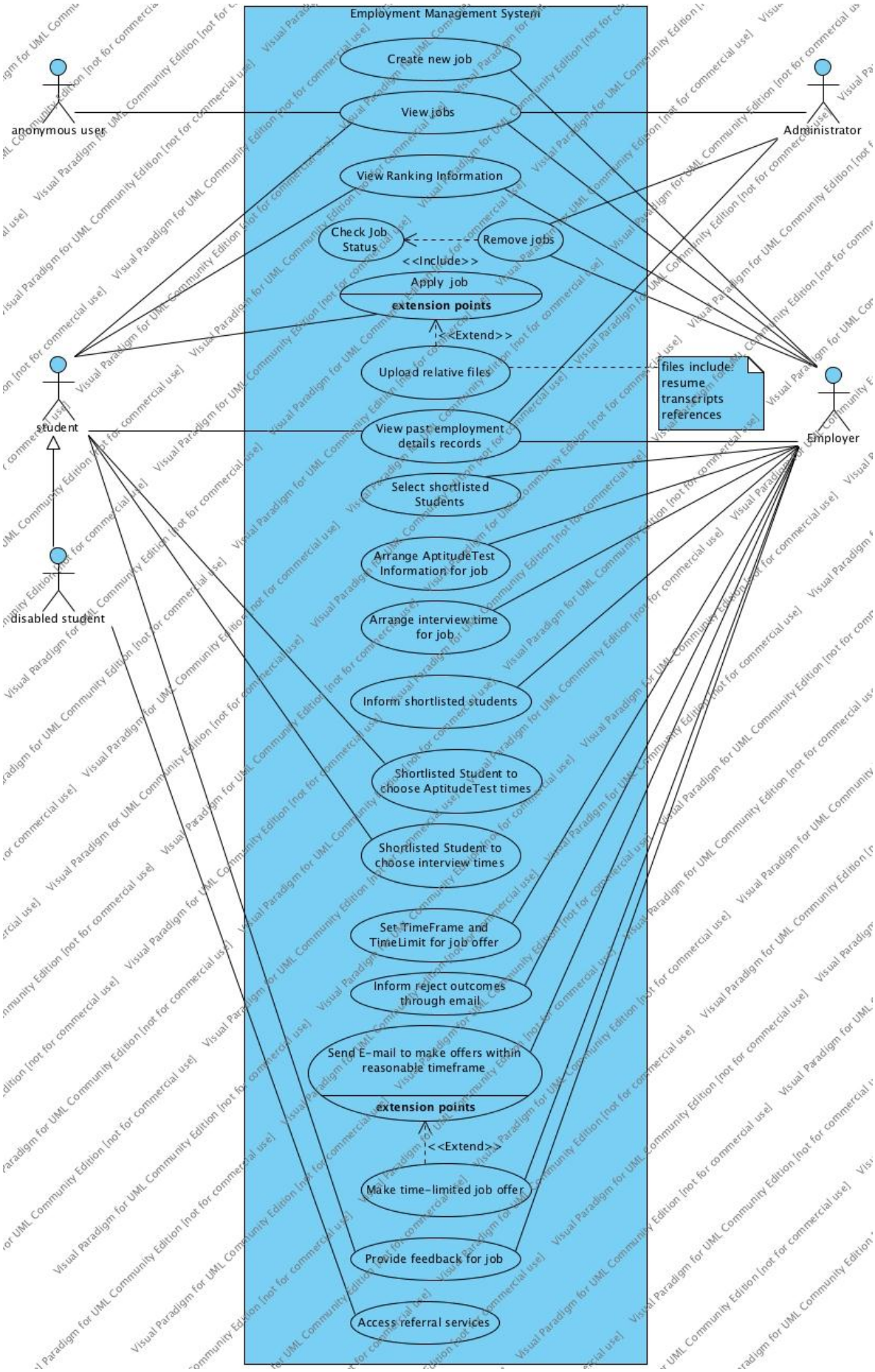


Figure 1.2 Use case diagram of the Job Management and the system

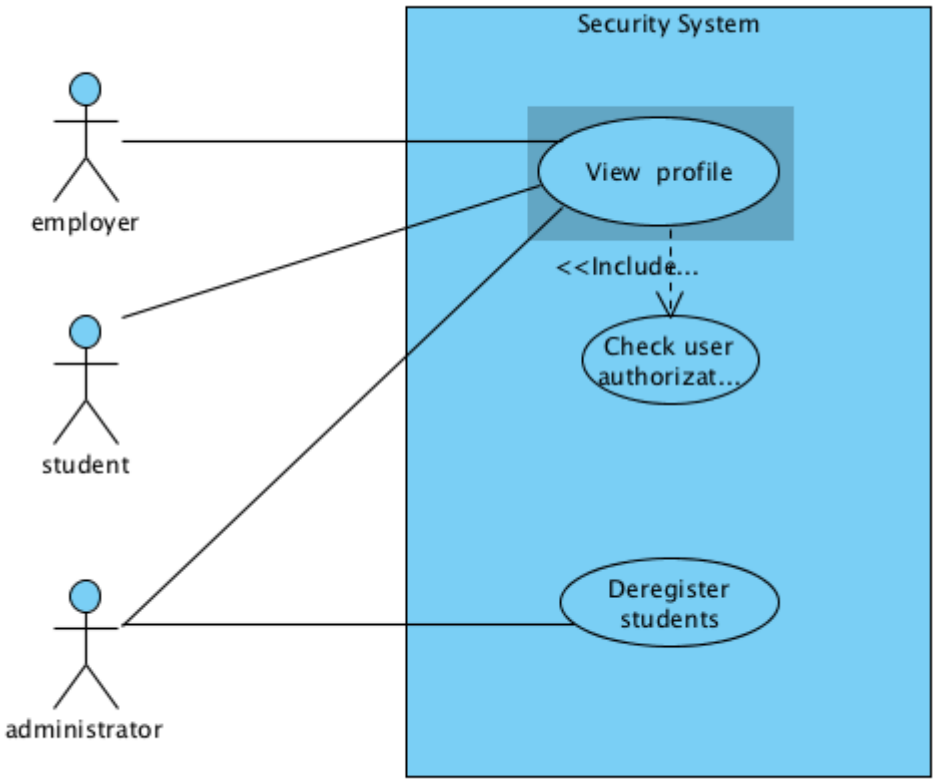


Figure 1.3 Use case diagram of the security system and the system

2.1.2 Description of Use-case diagram

Facility Staff Section

Use Case Name	Register account
Iteration:	Filled
Summary:	The user can register their personal information with the system
Basic Course of Events:	1. The system will first display the registration page. 2. The user, which in this case can be the student or the employer, will have to select the account type that the user will be registering. 3. If the user is a a. Student; The system will provide a registration page where the student needs to provide personal information such as: i. Full Name ii. Date of Birth iii. Mobile Number iv. Email Address v. Gender vi. Upload Resume or Transcript (Optional) vii. Personalized User ID viii. Personalized User Password b. Employer; The system will provide a registration page where the employer needs to provide personal Information such as: i. Full Name ii. Date of Birth iii. Mobile Number iv. Email Address v. Gender vi. Company Name vii. Company Description viii. ABN x. Personalized User ID xi. Personalized User Password 4. After submitting the completed registration form, the system will check the all the input for validity. 5. The system submits all the information to the database storage and inform the user that the registration was a success and the system will generate a unique ID for the different user : a. Student will start with the letter “S” followed by 7-digit unique number. b. Employer will start with the letter “E” followed by 7-digit unique number. and displays them onto the screen 6. Exit the use case

Alternative Paths:	1. In step 4, if the user enters invalid information during the registration, the system will repeat step 3 again. 2. Such Information during registration is considered invalid if a : a. Student ‘s i. Full Name is: a. Empty. b. More than 50 characters. c. Have already existed in the system ii Date of Birth is empty iii. Mobile Number is: a. Empty. b. Number is outside Australia. c. Number is more than 10 characters. iv. Email Address: a. Invalid. b. Have already existed in the system v. Gender is empty vi. Personalized User/Password is empty. b. Employer ‘s i. Full Name is: a. Empty. b. More than 50 characters. ii Date of Birth is empty iii. Mobile Number is: a. Empty. b. Number is outside Australia. c. Number is more than 10 characters. iv. Email Address is invalid. v. Gender is empty vi. Personalized User/Password is empty. vii. Company Name is empty viii. Company Description is empty x. ABN is empty
Exception Paths:	-
Trigger:	A new user wants to register into the system
Preconditions	The user who is trying to register must be new to the system
Post-conditions:	Information entered by the user is appropriately stored in the database.

Table 1.1 Use case description for Register an account

Use Case Name	Login
Summary:	The user (in this case the student, employers and the administrator) can log in into the system and the system decides which menu to display depending on the user type.
Basic Course of Events:	1. The user will need to go into the login page. 2. The log in page will require the user to enter the personalized user ID and user Password into the field box provided. 3. After submitting the user ID and password, the system will check the all the input information for validity 4. If all the input information is valid, the system will generate different menu (access control) according to different user type: a. For Student whose user ID start with S will generate student privilege functions b. For Employer whose user ID start with E will generate employer privilege functions c. For Administrator whose user ID start with A will generate Administrator privilege functions 5. When login successfully it will redirect to Home page for user.
Alternative Paths:	In step 3, if the user enters the invalid information during the login, the system will allow the same user to retry at most three times. After three times, the system will lock the current user account and the user has to contact with Administrator for further information.
Exception Paths:	
Trigger:	User who want to login into system
Assumptions:	-
Preconditions	
Post-conditions:	User can see Result page (Welcome Page or Error Page)

Table 1.2 Use case description for Login

Use Case Name	Manage records
Summary:	Administrator wants to modify, delete or create student record and employer record.
Basic Course of Events:	1. Admin requests to create student or employer records. 2. The system return student or employer creation form. 3. Admin fill in all the fields required respectively and submit. 4.The system validate all the input data and update the database; 5. The system returns creation success information to the admin. 6. Admin requests to delete specified student or employer records. 7. The system returns deleting confirmation message to the admin. 8. Admin confirms to delete records. 9. The system deletes records in database and return deleting success message to the admin. 10. Admin requests to modify a specific student or employer record. 11. The system return corresponding information of the record to the admin. 12. Admin modifies fields and requests to update record. 13. The system validates all fields of the record and updates the database; 14. The system return updating success message to the admin.
Alternative Paths:	at step 4 and step 13, if there is any invalid fields in the input, the system would return error message to the admin and ask for correct not-allowed input data.
Exception Paths:	-
Trigger:	The admin requests to update create or delete student or employer records.
Preconditions	-
Post-conditions:	The system successfully updates the database.

Table 1.3 Use case description for managing record

Security Section:

Use Case Name	View profiles
Iteration:	Filled
Summary:	View profile information depending on different role (including check user authorization use case)
Basic Course of Events:	1. A visitor can only be one of 4 roles, viewer, student, employer and administrator, click to view other account profile. 2. System retrieves role information from the visitor. 3. System retrieves authority information based on the role of the visitor; 4. System shows corresponding profile information to the visitor.
Alternative Paths:	-
Exception Paths:	-
Trigger:	A visitor chooses to view other account information.
Assumptions:	-
Preconditions	-
Post-conditions:	-

Table 1.4 Use case description for view profile

Use Case Name	Deregister Student
Iteration:	Filled
Summary:	Allow administrator deletes a student account
Basic Course of Events:	1. A user login as administrator. 2. Go into Deregister Page; 3. Administrator provides deregister student ID. 4. The system performs delete operation for specific student.
Alternative Paths:	During step 4, if the provided student information is not validated, the system does not perform delete operation and return error message to user.
Exception Paths:	-
Trigger:	Administrator who want to deregister a student
Assumptions:	-
Preconditions	Only administrator can deregister a student.
Post-conditions:	Student record still in database but in lock status.

Table 1.5 Use case description for deregister student

Employment Management Section:

Use Case Name	Create a new job
Iteration:	Filled
Summary:	Allow an employer to create a new job.
Basic Course of Events:	1. An employer requests to create a new job. 2. The system return job creation form; 3. The employer fills in the form including fields such as: a. Job description; b. Employee requirement; c. Job expiry date. 4. The system validates required fields and updates the database. 5. The system will display successful job creation.
Alternative Paths:	At step 3, if the employer gives any invalid information format, the system would show warning message to specific fields.
Exception Paths:	
Trigger:	A registered employer wanting to create a job.
Assumptions:	Only employer can create a new job
Preconditions	A registered employer has login.
Post-conditions:	A new job record is properly stored in database.

Table 1.6 Use case description for creating a new job

Use Case Name	View Jobs
Iteration:	Filled
Summary:	Allows everyone including administrator, students, employers and anonymous user to view all the job listing
Basic Course of Events:	1. A user request to view the entire job listing that is available. 2. The system returns a list of jobs available.
Alternative Paths:	-
Exception Paths:	-
Trigger:	Any user wants to see a job detail
Assumptions:	-
Preconditions	There are job for the system to display them
Post-conditions:	-

Table 1.7 Use case description for view job

Use Case Name	View ranking
Summary:	A visitor wants to view either an employer's or a student's rating;
Basic Course of Events:	1. A visitor requests to view a employer's or a student's rating. 2. The system returns an employer's or a student's rating.
Alternative Paths:	-
Exception Paths:	During step 2, if an error occurs in searching an employer's or a student's rating records, the system would return database operation error page.
Trigger:	A visitor requests to view either an employer's or a student's rating;
Preconditions	-
Post-conditions:	Ranking information is successfully to the visitor.

Table 1.8 Use case description for view ranking

Use Case Name	Remove jobs
Iteration:	Filled
Summary:	Allow an employer (the job owner) or administrator (the job on expiry) to remove existing jobs. This Use Case includes Check job status use case.
Basic Course of Events:	1. An employer or administrator logs in into the system. 2. Go to the remove job page; 3. The user inputs job ID that he/she want to remove. 4. According to the user type and job status, the system perform following process: a. If user is an employer, the system will check whether the current employer created this job. If yes, he/she can perform the deletion operation. b. If user is an administrator, the system will check whether this job has expired. If yes, administrator and perform delete operation. 5. After performing the delete operation, the system will record all relative deleted records into backup file. 6. The system shows successful job deletion.
Alternative Paths:	At step 4, if any of the condition is not satisfied, the system will display error message, which includes (user type privilege access error) the reason to user.
Exception Paths:	-
Trigger:	A registered employer or administrator wants to delete a job.
Assumptions:	-
Preconditions	A registered employer or administrator must first log in.
Post-conditions:	Specific job record is properly deleted from database table.

Table 1.9 Use case description for remove job.

Use Case Name	Apply job
Iteration:	Filled
Summary:	Registered students and disabled students to apply for jobs
Basic Course of Events:	1. Registered student or a disabled student views a vacant job description 2. Registered student or a disabled student choose to apply a job; 3. The system will check if the student has uploaded either a resume or a transcript or a reference. 4. The system will return a job application number and let the student know that the job is successfully applied.
Alternative Paths:	At step 3, if the registered student or a disabled student has not uploaded resume and transcript, the system would require the student to upload either one before applying.
Exception Paths:	At step 2 and step 3, if error occurs in updating database, the system returns error information about server problem to the student.
Trigger:	a registered student or a disabled student to apply a job
Assumptions:	A registered student or a disabled student can apply a job that he/she hasn't applied it before.
Preconditions	A student must be registered and login.
Post-conditions:	Job application requested by the student properly recorded in database.

Table 1.10 Use case description for applying job

Use Case Name	Upload the resume, transcripts and reference
Iteration:	Filled
Summary:	Registered students who has not uploaded or want to update their resume, transcripts or reference can upload them.
Basic Course of Events:	1. The user must first login as a student and must direct to the resume, transcripts and reference management page. 2. The user can upload his/her resume and transcripts. a. If he/she never uploads them before, the system will create new records in database. b. If resume and transcript have been uploaded before, the new files will overwrite the old files. 3. The system returns upload success message to user.
Alternative Paths:	During stage 2, if there are any errors (such as wrong required file format or network problem), the system will display the an error message to user.
Exception Paths:	-
Trigger:	A registered student who want to upload or update resume and transcripts.
Assumptions:	All registered student can upload resume and transcripts.
Preconditions	Student must be registered and login.
Post-conditions:	-

Table 1.11 Use case description for upload the resume, transcript and reference

Use Case Name	View past employment details records
Iteration:	Filled
Summary:	Registered student, employer or administrator want to view past employment records for specific student
Basic Course of Events:	1. The user must first login as different role and go into employment detail management page. 2. The user can view past employment detail according to different privileges. a. The student role can only view their own past employment detail. b. The employer role can view employment detail for all the students who apply his/her jobs. c. The administrator can view employment detail for any registered student. 3. The system returns employment detail to user.
Alternative Paths:	-
Exception Paths:	-
Trigger:	Registered user who want to view student's employment detail according to his/her privilege.
Assumptions:	-
Preconditions	The user must be registered and login.
Post-conditions:	-

Table 1.12 Use case description for view past employment details records

Use Case Name	Select shortlisted student
Iteration:	Filled
Summary:	Employer shortlists potential students.
Basic Course of Events:	1. The employer request to show all applied students for a specific job. 2. The system returns all applied students. 3. The employer selects student IDs. 4. The system record shortlisted students.
Alternative Paths:	At step 3, the system will return an error if no student ID is selected as employer must have at least chose one student.
Exception Paths:	At step 4, if error occurs in updating database, the system returns error information about server problem to the student.
Trigger:	A registered employer wanting to create a job.
Assumptions:	-
Post-conditions:	Shortlisted students are properly stored in database

Table 1.13 Use case description for select shortlisted student

Use Case Name	Arrange Aptitude Test Information for job
Iteration:	Filled
Summary:	Shortlisted full time students are given an aptitude test
Basic Course of Events:	1. The shortlisted full time students are given a notification that there would be an aptitude test 2. The student will sit for the test under staff supervision.
Alternative Paths:	If the student it unable to attend the aptitude test, the student must arrange another time for the test to be taken 48 hours before the assigned time for the aptitude test.
Exception Paths:	-
Trigger:	Selected students that applied wants a full time job
Assumptions:	The student must be shortlisted and selected by the employer
Post-conditions:	-

Table 1.14 Use case description for arrange aptitude test information for job

Use Case Name	Arrange interview time
Iteration:	Filled
Summary:	An employer arranges available interview time option for a specific job.
Basic Course of Events:	1. An employer selects specific job to arrange interview time. 2. The employer provides interview time slots for specific job. 3. The system validates time format and semantics. 4. The system properly records interview time in database.
Alternative Paths:	At step 3, if the employer doesn't provide valid time information, the system returns error message and ask to return to step 2.
Exception Paths:	At step 4, if error occurs in updating database, the system returns error information about server problem to the employer.
Trigger:	A registered employer wants to arrange time options.
Assumptions:	-
Post-conditions:	Available interview time slots are properly stored in database

Table 1.15 Use case description for arrange interview time

Use Case Name	Inform shortlisted students through email
Iteration:	Filled
Summary:	To inform selected students to make interview appointment by email.
Basic Course of Events:	1. An employer request to view all shortlisted students with specific conditions. 2. An employer specifies a subset of shortlisted students. 3. An employer provides an email template. 4. The system sends emails to all specified shortlisted students. 5. The system update status of students who are successfully received emails. 5. The system return email outcome.
Alternative Paths:	At step 4, if some of the students email address cannot be reached, the system would show a list of students who fail to receive email notification.
Exception Paths:	-
Trigger:	An employer wants to notify shortlisted students to choose interview time.
Assumptions:	-
Preconditions	Registered employer is login beforehand.
Post-conditions:	The system properly updates shortlisted students status.

Table 1.16 Use case description for inform shortlisted students though email

Use Case Name	Shortlisted Student to choose Aptitude Test times
Iteration:	Filled
Summary:	Students who are selected for the aptitude test are required to choose a test time.
Basic Course of Events:	1. The student must first login into the system and the system verifies that the student is shortlisted and available to seat for the test. 2. The student will be directed to the Timetabling page where the student can select the time that they would want to take the test 3. The student confirms the time selected 4. The system records it and saves into the database.
Alternative Paths:	-
Exception Paths:	If there is something wrong in retrieving the times or saving the selected timetables, the system will return an error
Trigger:	Shortlisted students that are looking to work at a full time job that requires them to seat for an aptitude test
Assumptions:	The student is already shortlisted and looking for a full time job
Preconditions	The student must first login into the system
Post-conditions:	The system updates the selected timetable by the students to the database.

Table 1.17 Use case description for shortlisted student to choose aptitude test time

Use Case Name	Choose interview time
Iteration:	Filled
Summary:	Shortlisted students choose interview time.
Basic Course of Events:	1. Students request interview time setting page. 2. the system display interview time-setting page with available time options for a specific job 3. Students choose interview time slot. 4. The system validates time and check vacant position. 5. The system returns success information to students.
Alternative Paths:	at step 4, a. If the student gives invalid time format, the system return error message; b. if the vacant position has been taken before the student request, the system return position-taken error message to the student. c. If the student is not a shortlisted student for a specific job, the system would return authority-related error message.
Exception Paths:	At step 4, if any error occurs in database operation, the system would show server-related error message to the student.
Trigger:	A shortlisted student wants to select an interview time.
Assumptions:	-
Preconditions	1. The student has been login. 2. The student is shortlisted.
Post-conditions:	The system properly records the interview time for shortlisted students.

Table 1.18 Use case description for choose interview time

Use Case Name	Make TimeFrame and TimeLimit for casual job offer
Iteration:	Filled
Summary:	To set TimeFrame and TimeLimit for specific casual job
Basic Course of Events:	1. an employer views all the available jobs which are created by himself. 2. The employer selects only one job which he want to set parameters for them. 3. The employer sets the specific TimeFrame for this job. 4. The employer also can set TimeLimit for this job(Optional Parameter). 5. the system store these information into database.
Alternative Paths:	-
Exception Paths:	-
Trigger:	a employer who want to set the time information for a specific job offer
Assumptions:	-
Preconditions	Registered employer logins into system.
Post-conditions:	the system properly store time-related information for a specific job.

Table 1.19 Use case description for make timeframe and timelimit for casual job offer

Use Case Name	Provide feedback for a job
Iteration:	Filled
Summary:	Provide a feedback from the employer to the student
Basic Course of Events:	1. The employer must first log in into the system as in employer 2. The employer is directed to the feedback page 3. The employer type a feedback 4. The employer send the feedback
Alternative Paths:	-
Exception Paths:	-
Trigger:	The employer wants to give a feedback to the student
Assumptions:	-
Preconditions	The user must be logged in as an Employer
Post-conditions:	The feedback is sent to the student

Table 1.20 Use case description for provide feedback for a job

Use Case Name	Access referral services
Iteration:	Filled
Summary:	Referral services that assist disabled applicants
Basic Course of Events:	1. Disabled applicants can use the website to access the page 2. The system can determine the disability of the user and suggest jobs that are available 3. The user selects the job
Alternative Paths:	-
Exception Paths:	-
Trigger:	A disabled applicant wants to find a job
Assumptions:	Disabled applicants are already registered in the system
Preconditions	User type must be disabled.
Post-conditions:	-

Table 1.21 Use case description for access referral services

Job Seeker Management

Use Case Name	Seek for employment
Iteration:	Filled
Summary:	Registered students search jobs
Basic Course of Events:	1. A registered student input search conditions, such as a. job type b. job location 2. The user click search button 3. The system returns search results
Alternative Paths:	-
Exception Paths:	-
Trigger:	A registered student search for jobs
Assumptions:	All registered student can search job.
Preconditions	A student must be registered and login.
Post-conditions:	-

Table 1.22 Use case description for seek for employment

Use Case Name	View past employment details records
Iteration:	Filled
Summary:	Registered student, employer or administrator want to view past employment records for specific student
Basic Course of Events:	1. user login as different role and go into employment detail management page. 2. the user can view past employment detail according to privilege. a. student role can only view their own past employment detail. b. employer role can view employment detail for all the students who apply his/her jobs. c. administrator can view employment detail for any registered student. 3. the system returns employment detail to user.
Alternative Paths:	-
Exception Paths:	-
Trigger:	a registered user who want to view student's employment detail according to his/her privilege.
Assumptions:	-
Preconditions	a user must be registered and login.
Post-conditions:	-

Table 1.23 Use case description for view past employment details records

Use Case Name	Rank
Summary:	A student wants to rate the employer after the completion of the job. A employer will to rate the student after the job completion.
Basic Course of Events:	1. A student requests to rate his employer; 2. The system returns the rating page; 3. The student fills in ranking fields required. 2. The system checks the student profile whether he/she has completed the employment. 3. The system validates required fields. 4. The system updates the database. 5. An employment requests to rate a student and give feedback. 6. The system returns the rating page including feedback. 7. The employer fills in all fields required including marks and feedback text; 8. The system validates required fields. 9. The system updates the database.
Alternative Paths:	- at step 3 and 4, if the field data entered is invalid, the system would return the ranking page with error messages.
Exception Paths:	-
Trigger:	A student or an employer wants to rank each other.
Preconditions	The student work for the employer and has finished the job.
Post-conditions:	Ranking information is successfully stored in database.

Table 1.24 Use case description for Rank

Use Case Name	View ranking
Summary:	A visitor wants to view either an employer's or a student's rating;
Basic Course of Events:	1. A visitor requests to view a employer's or a student's rating. 2. The system returns an employer's or a student's rating.
Alternative Paths:	-
Exception Paths:	at step 2, if error occurs in searching an employer's or a student's rating records, the system would return database operation error page.
Trigger:	A visitor requests to view either an employer's or a student's rating;
Preconditions	-
Post-conditions:	Ranking information is successfully to the visitor.

Table 1.25 Use case description for view ranking

Use Case Name	Select job seekers
Iteration:	Filled
Summary:	An employer selects potential students.
Basic Course of Events:	1. Employer request to show all applied students for a specific casual job. 2. System displays all applied students. 3. Employer provide selected student IDs. 4. System record selected students.
Alternative Paths:	During step 1, if the employer doesn't provide any student ID, the system returns error message implying the employer should give at least 1 student ID.
Exception Paths:	During step 4, if error occurs in updating database, the system returns error information about server problem to the student.
Trigger:	Registered employer who want to selected job seekers
Assumptions:	-
Post-conditions:	Selected students are properly stored in database

Table 1.26 Use case description for select job seekers

Use Case Name	Make offers for casual employment within reasonable timeframe
Iteration:	Filled
Summary:	Inform selected casual job students outcome. (Includes: Send email to job seeker with useful information use case)
Basic Course of Events:	1. Employer queries all selected students. 2. Employer provide an email template to (In this process, employer may provide two different templates: the one is make offer template, the other is reject template). a. make an offer template. The system will fill the timeframe field and also include employment details and the average ranking for that employer so that an informed decision can be made. b. make a reject template. The system generates some reject content for job seekers. 4. System send emails to all casual job seekers in specific timeframe. 5. System update status of all job seekers.
Alternative Paths:	During step 4, if some of the students email address cannot be reached, the system would show a list of students who fail to receive email notification.
Exception Paths:	-
Trigger:	Employer who wants to inform outcomes to all job seekers.
Assumptions:	-
Preconditions	Registered employers login.
Post-conditions:	System properly updates the status of all job seekers.

Table 1.27 Use case description for Make offers for casual employment within reasonable timeframe

Use Case Name	Make a time-limited casual job offer (inherit from Make offers for casual employment within reasonable timeframe use case)
Iteration:	Filled
Summary:	To inform selected casual job students outcome with time limit factor.
Basic Course of Events:	1. To complete all the from previous use case (make offers for casual employment within reasonable timeframe) 2. Also add the time limited element into the make offer email.
Alternative Paths:	-
Exception Paths:	-
Trigger:	Employer who want to set the time information for a specific job offer
Assumptions:	-
Preconditions	Registered employer logins into system.
Post-conditions:	The system properly stores time-related information for a specific job.

Table 1.28 Use case description for Make a time-limited casual job offer (inherit from Make offers for casual employment within reasonable timeframe use case)

2.4 User Interface

The context diagram in Figure 3 illustrates the system interface for the release <v1.0>

Figure 3.1 Login

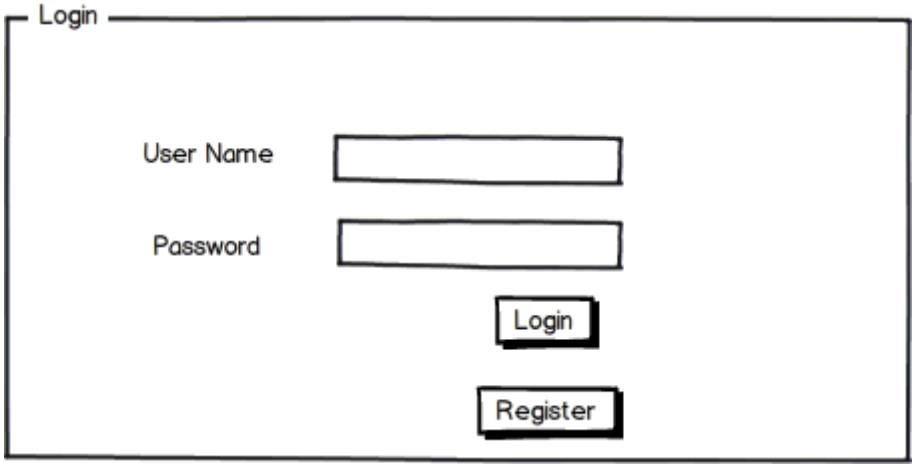


Figure 3.2

Register User Interface (for Student or Employer role)

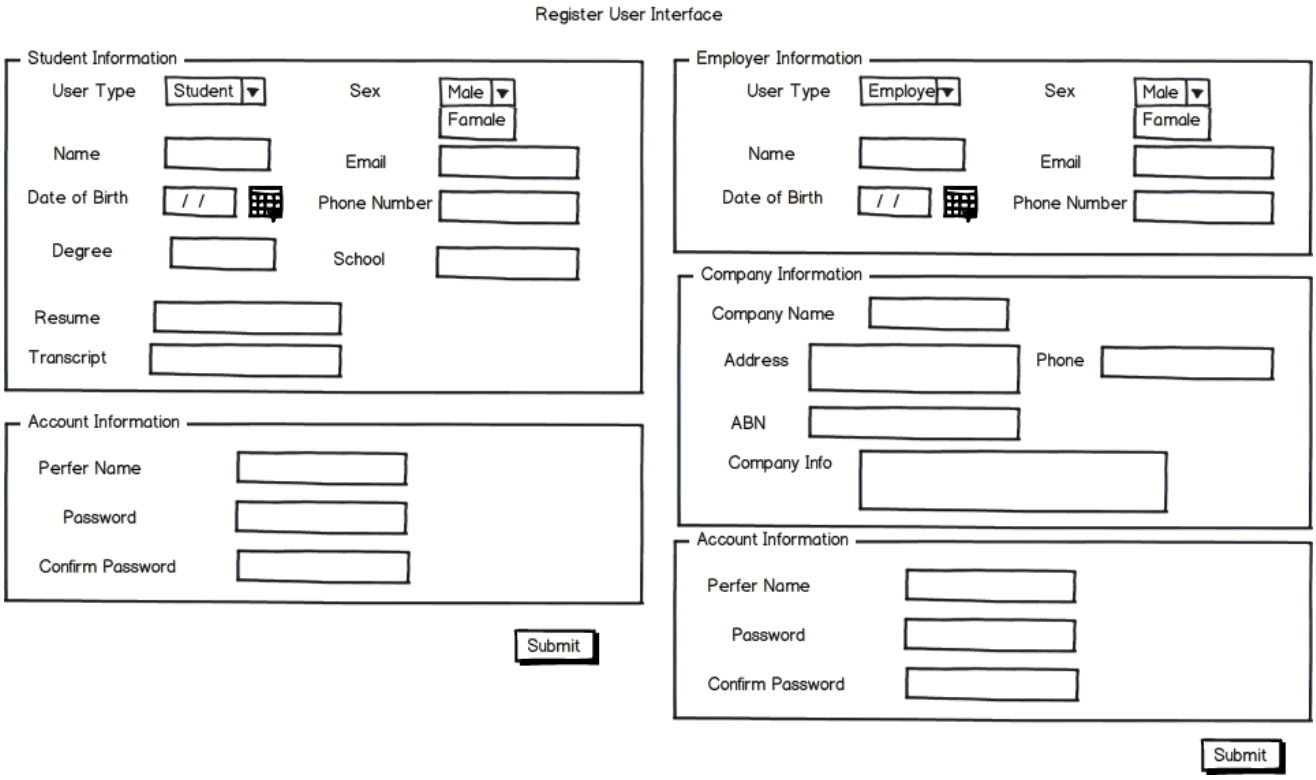


Figure 3.3

Create New Job

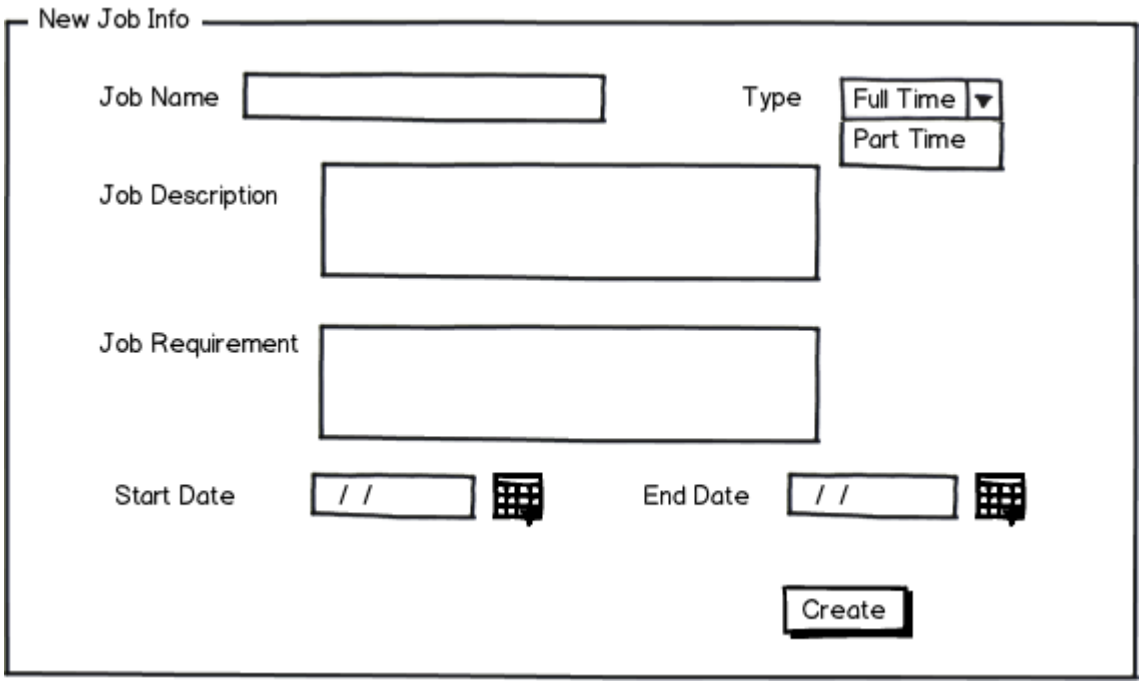


Figure 3.4

Select Shortlisted Students

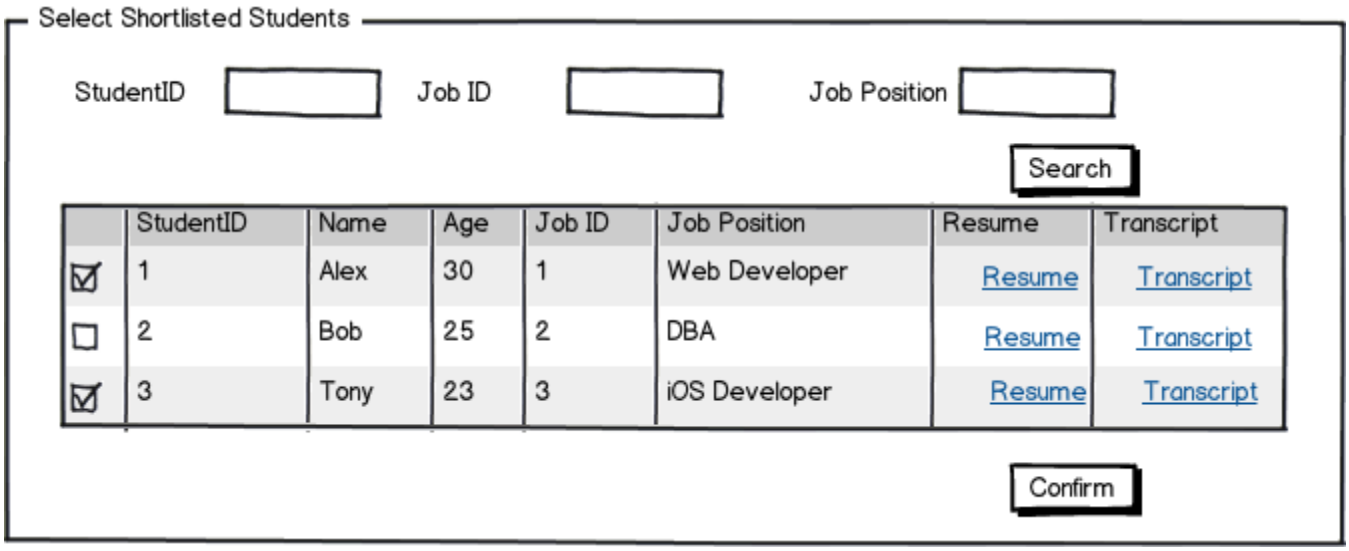


Figure 3.5

Arrange Interview Time

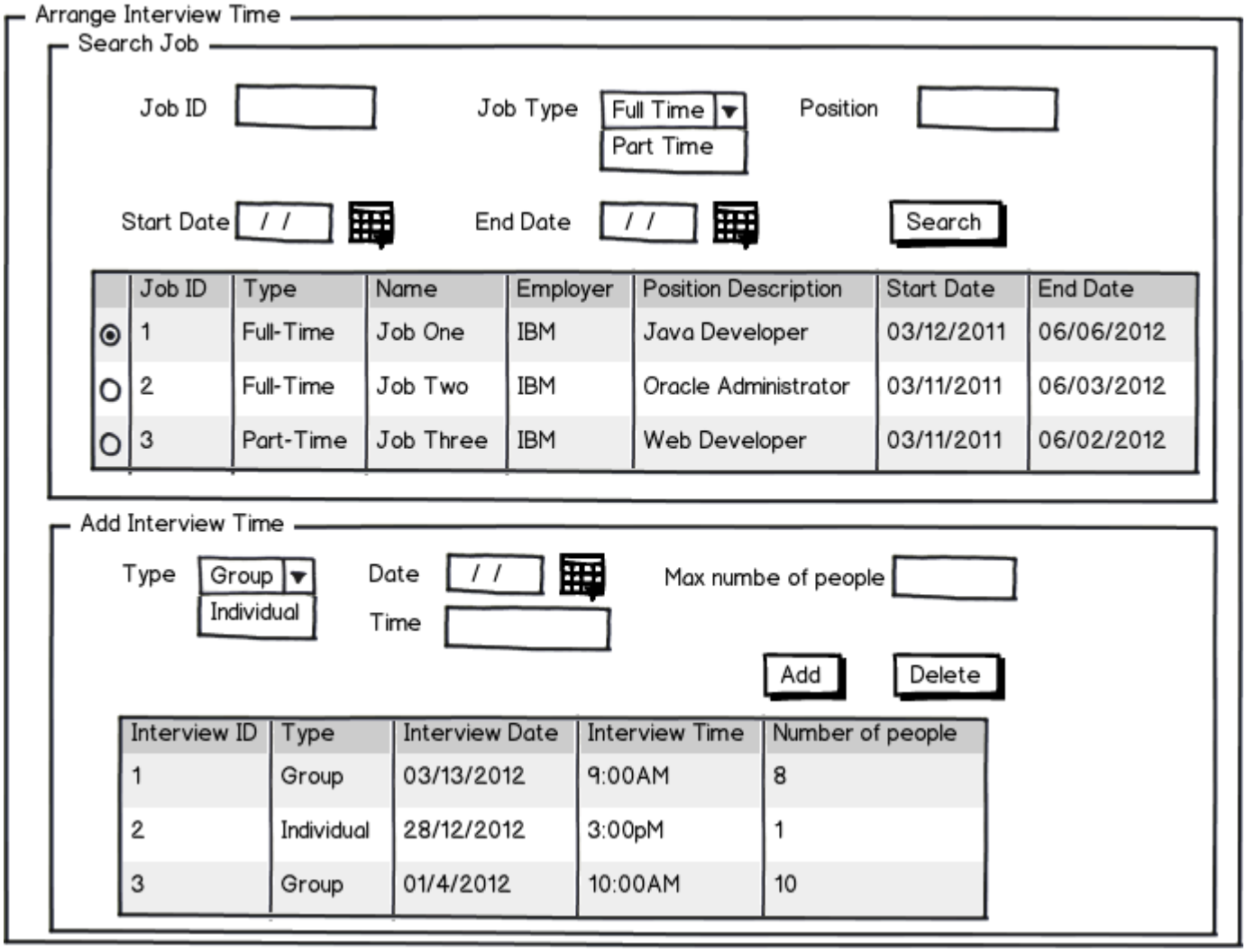


Figure 3.6

Inform Shortlisted Students

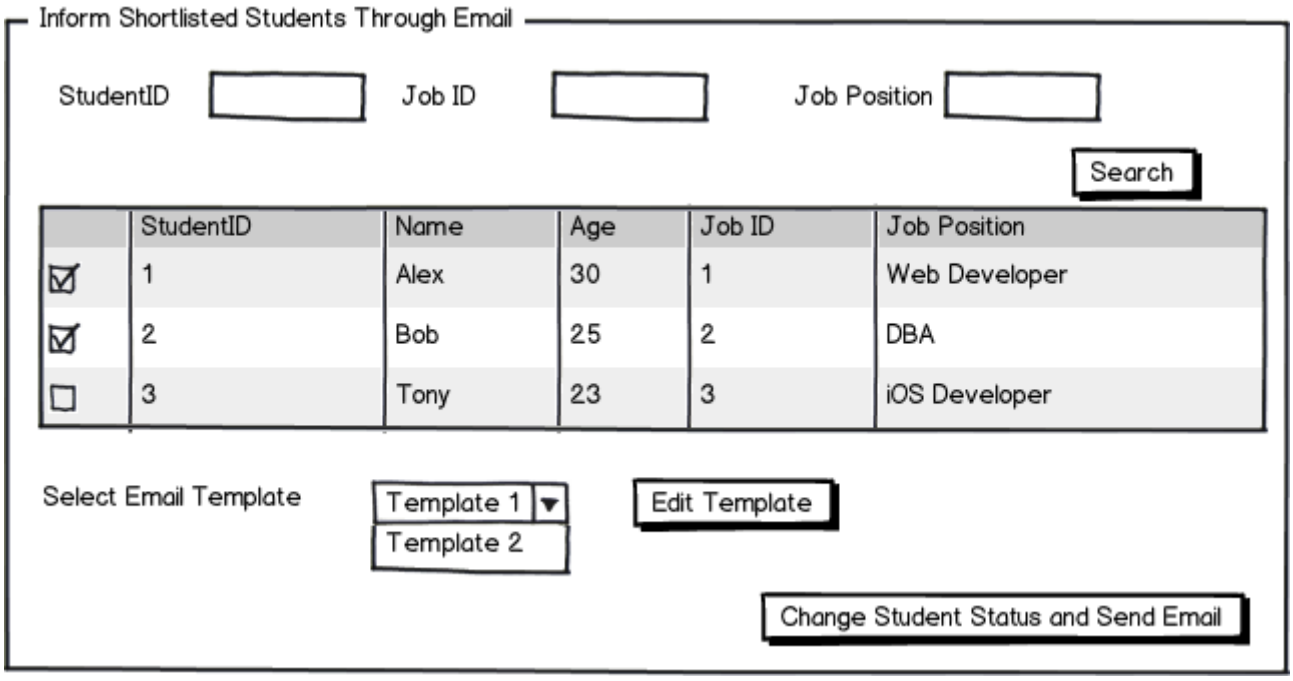


Figure 3.7

Choose Interview Time

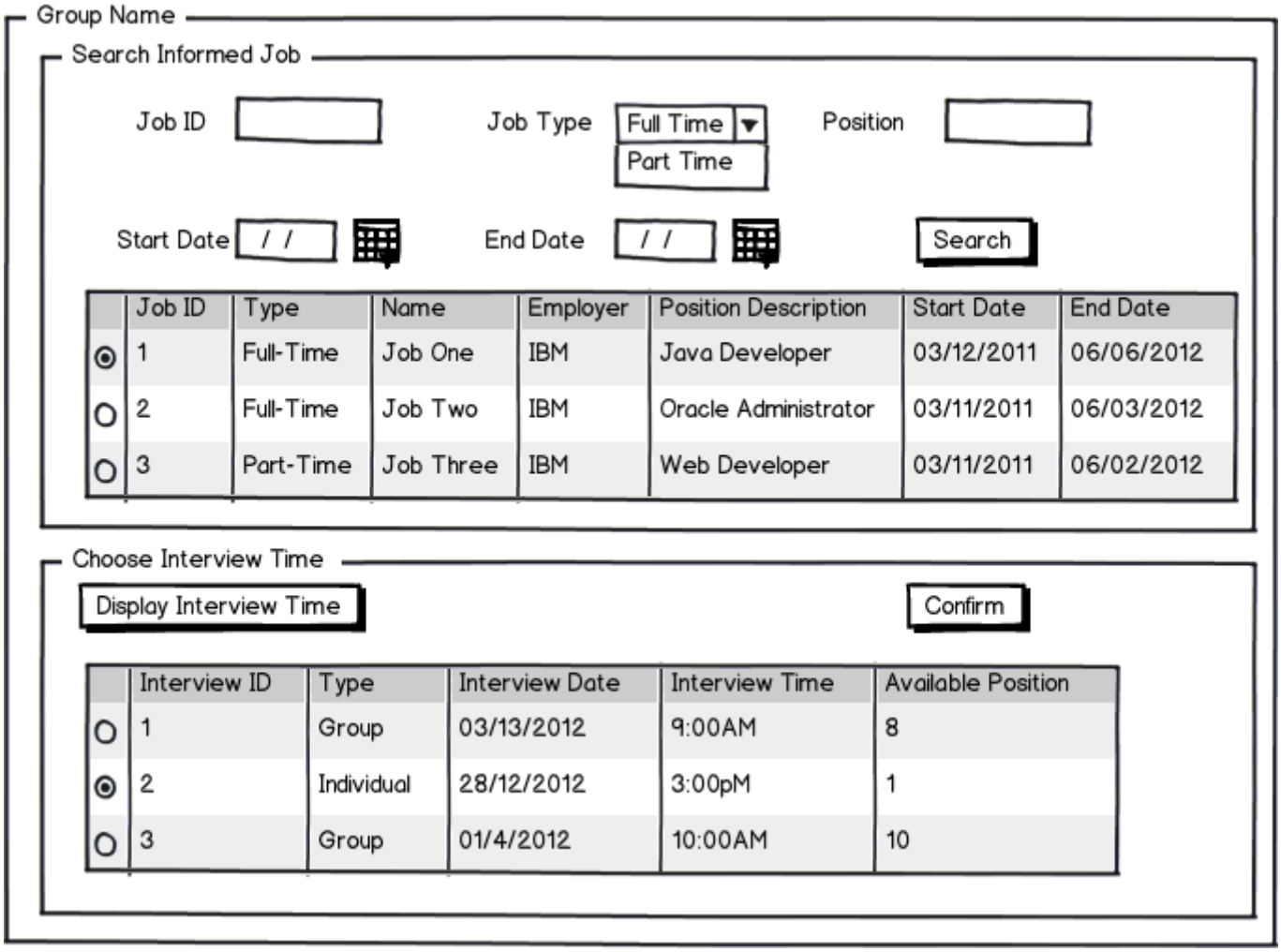


Figure 3.8

Inform Outcome to Student

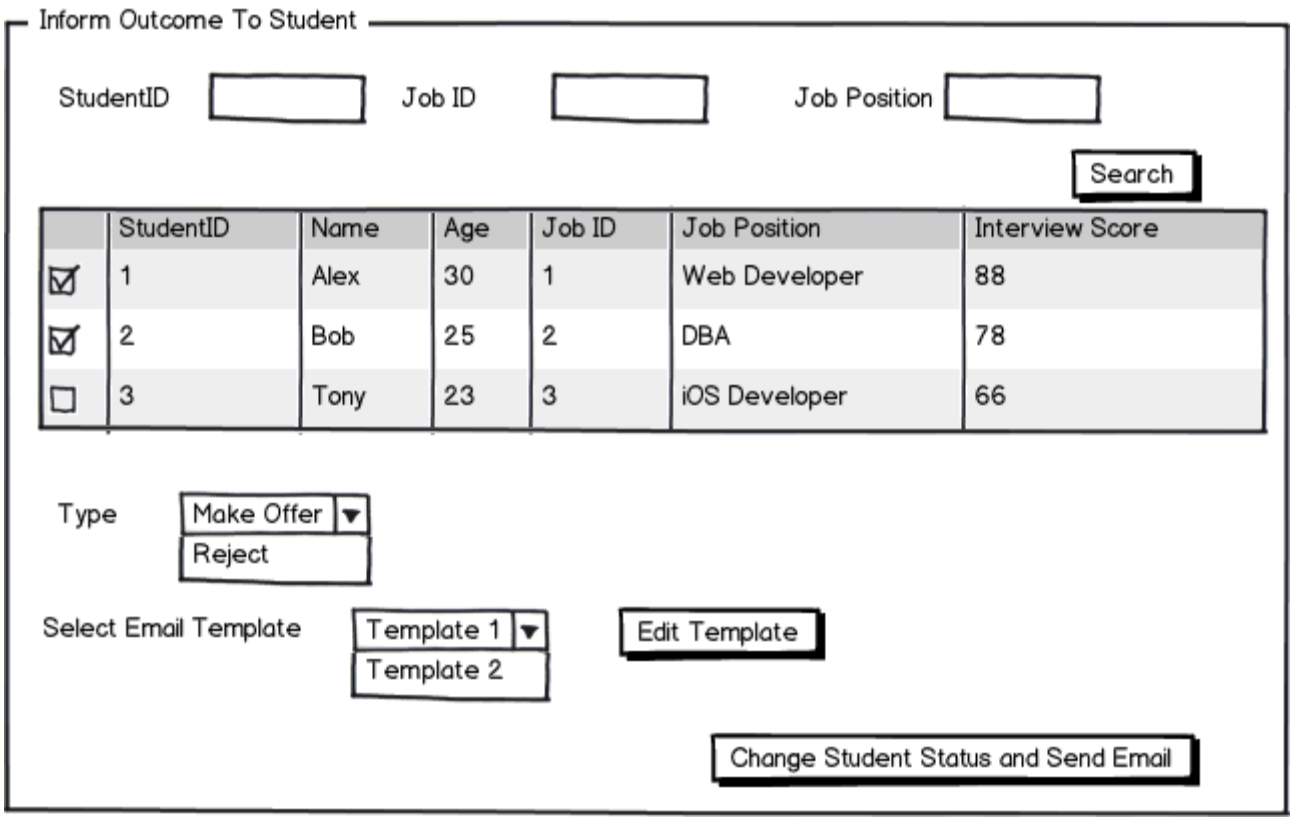


Figure 3.9

Set Job Timeframe and Time Limit

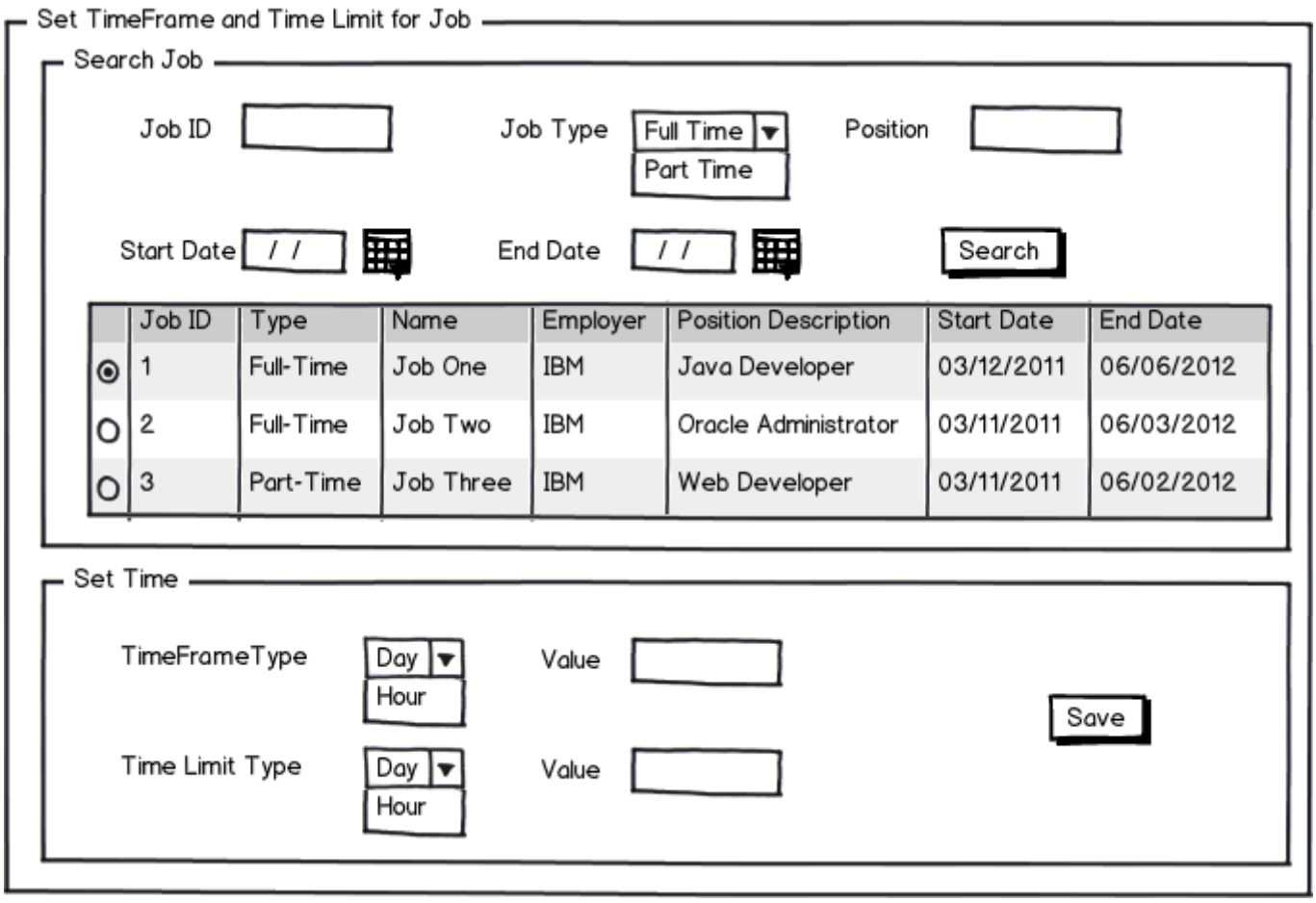
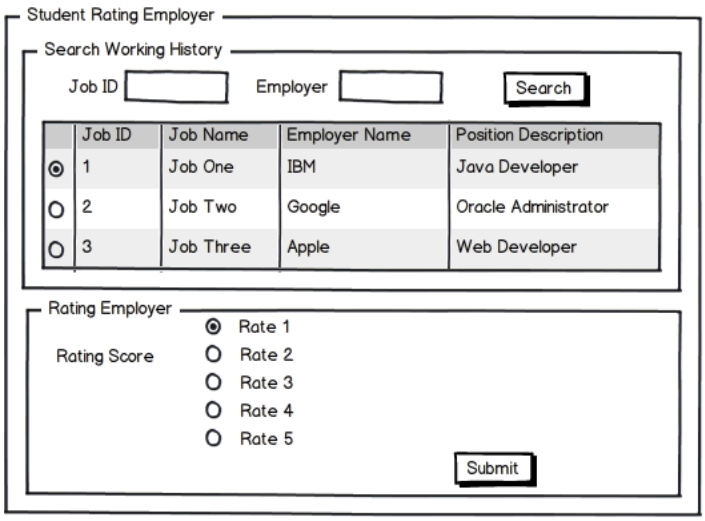


Figure 3.10
Ranking



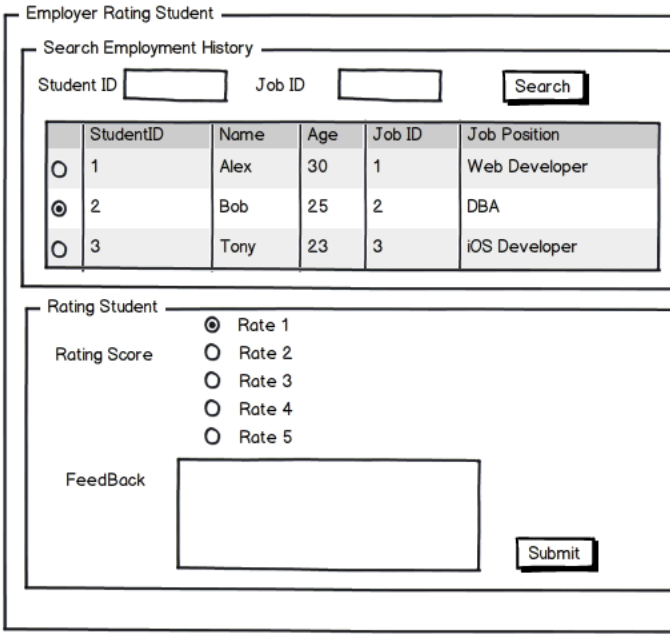
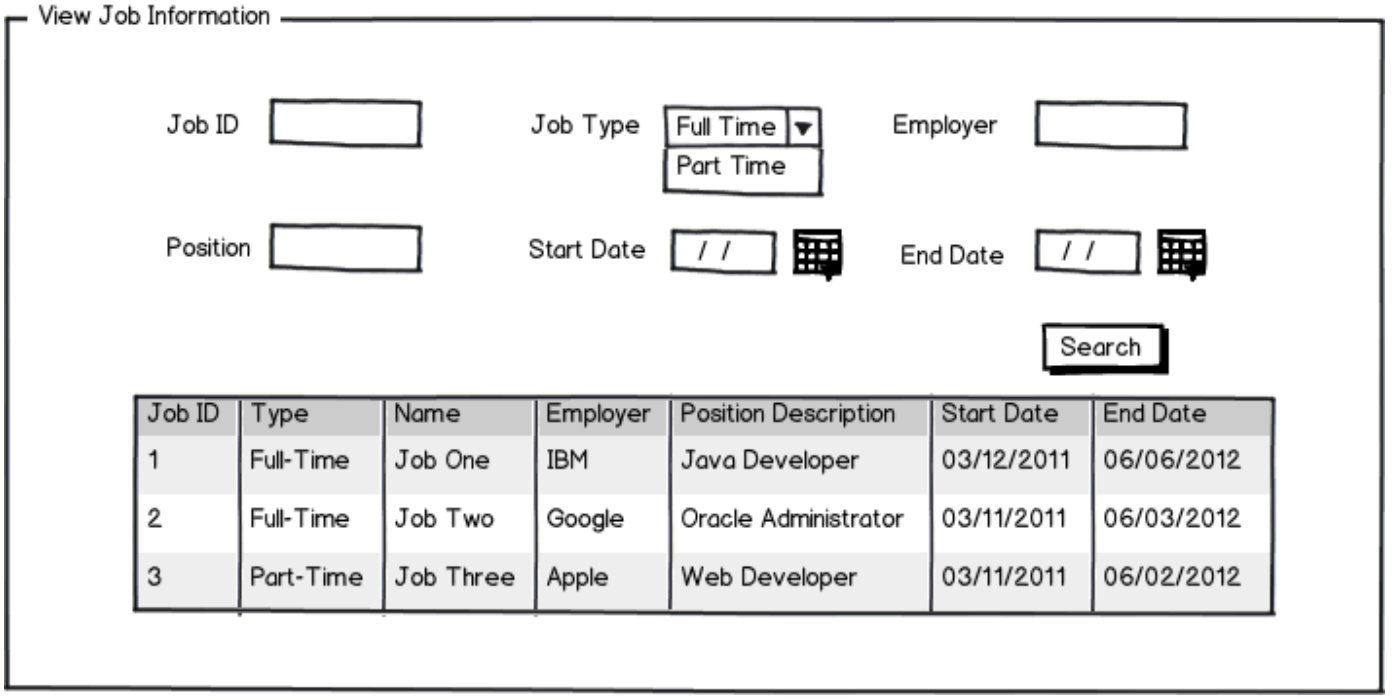
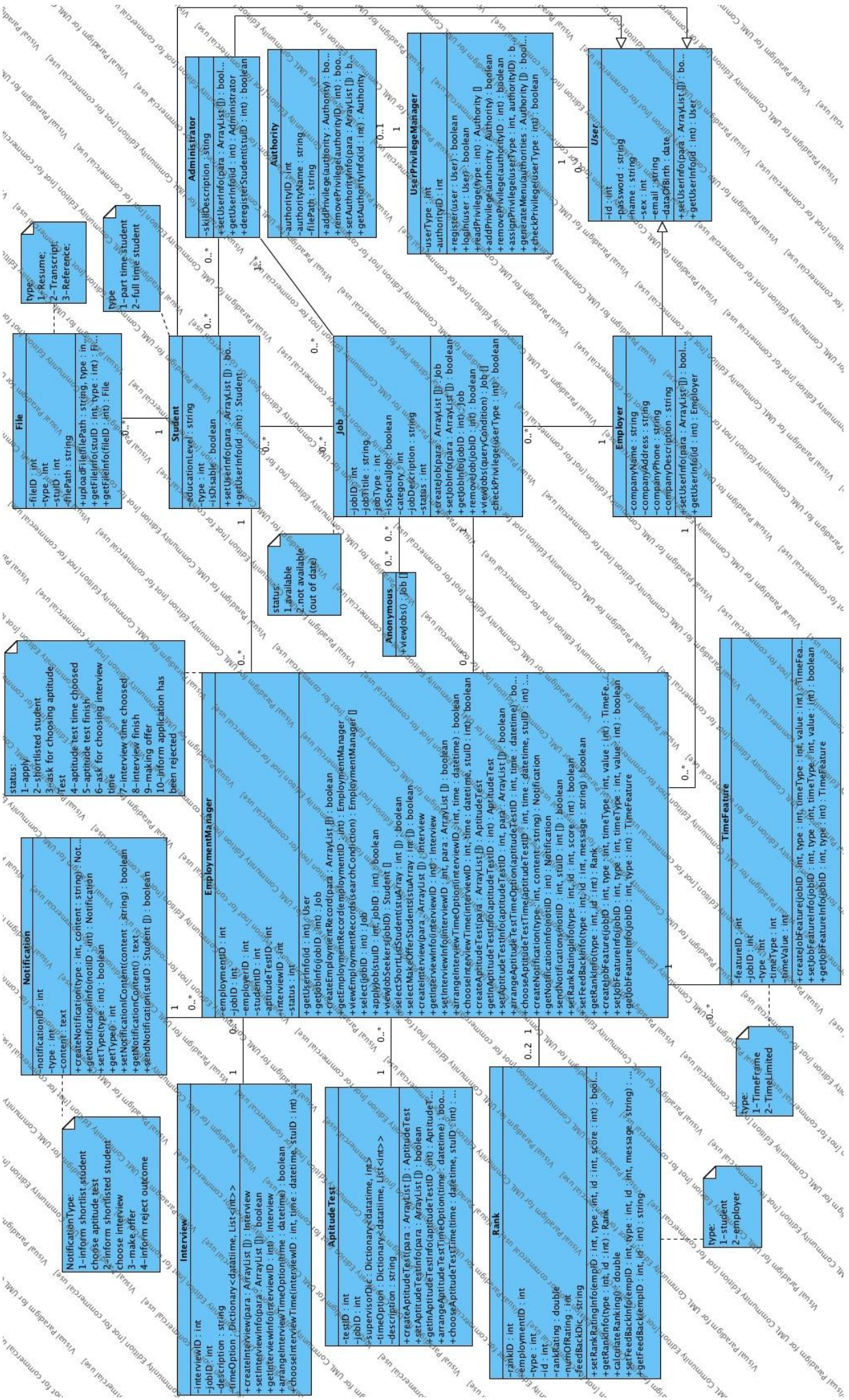


Figure 3.11

View Job Information



2.3 Class diagram


Administrator
-skillDescription : string
+setUserInfo(para : ArrayList []) : bool...
+getUserInfo(id : int) : Administrator
+deregisterStudent(stuID : int) : boolean
Authority
-authorityID :int
-authorityName : string
-filePath : string
-fileInfo : string
+addPrivilege(authority : Authority) : bo...
+removePrivilege(authority:D : int) : boo...
+setAuthorityInfo(para : ArrayList []) : b...
+getAuthorityInfo(id : int) : Authority
UserPrivilegeManager
-userType : int
-authorityID : int
+register(user : User) : boolean
+logIn(user : User) : boolean
+readPrivilege(type : int) : Authority []
+addPrivilege(authority : Authority) : boolean
+removePrivilege(authorityID : int) : boolean
+assignPrivilege(userType : int, authorityID) : b...
+generateMenu(authorities : Authority []) : bool...
+checkPrivilege(userType : int) : boolean
User
-id : int
-password : string
-name : string
-sex : int
-email : string
-dateOfBirth : date
+setUserInfo(para : ArrayList []) : bo...
+getUserInfo(id : int) : User
type:
1-Resume;
2-Transcript;
3-Reference;
type
1-part time student
2-full time student
File
-fileID : int
-type : int
-stuID : int
-filePath : string
+uploadFile(filePath : string, type : in...
+getFileInfo(stuID : int, type : int) : fi...
+getFileInfo(fileID : int) : File
Student
-educationLevel : string
-type : int
-isDisable : boolean
+setUserInfo(para : ArrayList []) : bo...
+getUserInfo(id : int) : Student
Job
-jobID: int
-jobTitle : string
-jobType : int
-isSpecialJob: boolean
-category : int
-jobDescription : string
-status : int
+createJob(para : ArrayList []) : Job
+setJobInfo(para : ArrayList []) : boolean
+getJobInfo(jobID : int) : Job
+removeJob(jobID : int) : boolean
+viewJobs(queryCondition) : Job []
-checkPrivilege(userType : int) : boolean
Employer
-companyName : string
-companyAddress : string
-companyPhone : string
-companyDescription : string
+setUserInfo(para : ArrayList []) : bool...
+getUserInfo(id : int) : Employer
status:
1-available
2-not available
(out of date)
Anonymous
+viewJobs() : Job []
status:
1-apply
2-shortlisted student
3-ask for choosing aptitude test
4-aptitude test time choosed
5-aptitude test finish
6-ask for choosing interview time
7-interview time choosed
8-interview finish
9-making offer
10-inform application has been rejected
EmploymentManager
-employmentID : int
-jobID : int
-employerID : int
-studentID : int
-aptitudeTestID: int
-interviewID :int
-status : int
+getUserInfo(id : int) : User
+getJobInfo(jobID : int) : Job
+createEmploymentRecord(para : ArrayList []) : boolean
+getEmploymentRecord(employmentID :int) : EmploymentManager
+viewEmploymentRecords(searchCondition) : EmploymentManager []
+selectJob(jobID : int) : Job
+applyJob(stuID : int, jobID : int) : boolean
+viewJobSeekers(jobID) : Student []
+selectShortListStudents(stuArray : int []) : boolean
+selectMakeOfferStudents(stuArray : int []) : boolean
+createInterview(para : ArrayList []) : Interview
+getInterviewInfo(interviewID : int) : Interview
+setInterviewInfo(interviewID : int, para : ArrayList []) : boolean
+arrangeInterviewTimeOption(interviewID :int, time : datetime) : boolean
+chooseInterviewTime(interviewID : int, time : datetime, stuID : int) : boolean
+createAptitudeTest(para : ArrayList []) : AptitudeTest
+getAptitudeTestInfo(aptitudeTestID : int) : AptitudeTest
+setAptitudeTestInfo(aptitudeTestID : int, para : ArrayList []) : boolean
+arrangeAptitudeTestTimeOption(aptitudeTestID : int, time : datetime) : bo...
+chooseAptitudeTestTime(aptitudeTestID : int, time : datetime, stuID : int) :...
+createNotification(type : int, content : string) : Notification
+getNotificationInfo(notID : int) : Notification
+sendNotification(notID : int) : boolean
+setFeedBackInfo(type : int, id : int, message : string) :boolean
+getRankInfo(type : int, id : int) : Rank
+createJobFeature(jobID : int, type : int, timeType : int, value : int) : TimeFe...
+setJobFeatureInfo(jobID : int, type : int, timeType : int, value: int) : boolean
+getJobFeatureInfo(jobID : int, type : int) : TimeFeature
TimeFeature
-featureID : int
-jobID : int
-type : int
-timeType : int
-timeValue : int
+createJobFeature(jobID : int, type : int, timeType : int, value : int) : TimeFea...
+setJobFeatureInfo(jobID : int, type : int, timeType : int, value : int) : boolean
+getJobFeatureInfo(jobID : int, type : int) : TimeFeature
Notification
-notificationID : int
-type : int
-content : text
+createNotification(type : int, content : string) : Not...
+getNotificationInfo(notID : int) : Notification
+setType(type : int) : boolean
+getType() : int
+setNotificationContent(content : string) : boolean
+getNotificationContent() : text
+sendNotification(stuID : Student []) : boolean
NotificationType:
1-inform shortlist student
choose aptitude test
2-inform shortlisted student
choose interview
3-make offer
4-inform reject outcome
Interview
-interviewID : int
-jobID : int
-description : string
-timeOption : Dictionary<datatime, List<int>>
+createInterview(para : ArrayList []) : Interview
+setInterviewInfo(para : ArrayList []) : boolean
+getInterviewInfo(interviewID : int) : Interview
+arrangeInterviewTimeOption(time : datetime) : boolean
+chooseInterviewTime(interviewID : int, time : datetime, stuID : int) : ...
AptitudeTest
-testID : int
-jobID : int
-supervisorDic : Dictionary<datatime, int>
-timeOption : Dictionary<datatime, List<int>>
-description : string
+createAptitudeTest(para : ArrayList []) : AptitudeTest
+setAptitudeTestInfo(para : ArrayList []) : boolean
+getAptitudeTestInfo(aptitudeTestID : int) : AptitudeT...
+arrangeAptitudeTestTimeOption(time : datetime) : boo...
+chooseAptitudeTestTime(time : datetime, stuID : int) : ...
Rank
-rankID : int
-employmentID : int
-type : int
-id : int
-rankRating : double
-numOfRating : int
-feedBackDic : string
+setRankRatingInfo(empID : int, type : int, id : int, score : int) : bool...
+getRankInfo(type : int, id : int) : Rank
+calculateRanking() : double
+setFeedBackInfo(empID : int, type : int, id : int, message : string) : ...
+getFeedBackInfo(empID : int, id : int) : string
type:
1-TimeFrame
2-TimeLimited
type:
1-student
2-employer

2.3.1 Class diagram description

EmploymentManager Class

+getUserInfo(id : int) : User
Administrator can get information from a user

+getJobInfo(jobID : int) : Job
User can get the job information.

+createEmploymentRecord(para : ArrayList []) : Boolean
The system will maintain a record of all past employment details

+getEmploymentRecord(employmentID : int) : EmploymentManager
Gets the records for all past employment recorded by the system.

+viewEmploymentRecords(searchCondiction) : EmploymentManager []
Each students can only view their own past employment. Employers who employs the students can view their past employments. Administrator can view the student's past employements.

+applyJob(stuID : int, jobID : int) : boolean
Registered students who have already logged in as a student can use this function to apply for a job

+viewJobSeekers(jobID) : Student []
Employers can view students who are still seeking for a job.

+selectShortListStudent(stuArray : int []) : Boolean
The employer gets to select students and shortlist them.

+selectMakeOfferStudents(stuArray : int []) : Boolean
The employer gets to choose whether to make an offer to the students.

+createInterview(para : ArrayList []) : Interview
The employer can create an interview session with the shortlisted student.

+getInterviewInfo(interviewID : int) : Interview
Shortlisted students can retrieve the interview information such as the Interview ID

+setInterviewInfo(interviewID : int, para : ArrayList []) : boolean
The employer can set the interview information (eg. The Date and Time)

+arrangeInterviewTimeOption(interviewID : int, time : datetime) : boolean
The employer can arrange a few interview time for the shortlisted student to select.

+chooseInterviewTime(interviewID : int, time : datetime, stuID : int) : boolean
Shortlisted students can use this function to choose the interview time.

+createAptitudeTest(para : ArrayList []) : AptitudeTest

The employer can create Aptitude test for shortlisted full time job seekers.

+getInAptitudeTestInfo(aptitudeTestID : int) : AptitudeTest
Gets the Aptitude test information.

+setAptitudeTestInfo(aptitudeTestID : int, para : ArrayList []) : Boolean
The employer can set the information on about the aptitude test.

+arrangeAptitudeTestTimeOption(aptitudeTestID : int, time : datetime) :Boolean
The employer can arrange the times where the students can select when to take the aptitude test.

+chooseAptitudeTestTime(aptitudeTestID : int, time : datetime, stuID : int) : Boolean
The shortlisted students can choose when to take the aptitude test time from the given time options by the employer

+createNotification(type : int, content : string) : Notification
The user can create a notification to another user by using the method.

+getNotificationInfo(notiID : int) : Notification
This function is use the get the notification information.

+sendNotifications(notiID : int, stuID : int []) : Boolean
The system sends any notification to the sender using this function.

+setRankRatingInfo(type : int, id : int, score : int) : boolean
Sets the rank rating information for both the employer and the student.

+setFeedBackInfo(type : int, id : int, message : string) : Boolean
Employer can send a feedback to the student by calling this function.

+getRankInfo(type : int, id : int) : Rank
Get the ranking information on the students or the employer.

+createJobFeature(jobID : int, type : int, timeType : int, value : int) : TimeFeature
The employer can create a job feature upon create a job into the job listing.

+setJobFeatureInfo(jobID : int, type : int, timeType : int, value : int) : boolean
The employer sets the information about this job that the employer listed.

+getJobFeatureInfo(jobID : int, type : int) : TimeFeature
Students can get the job's information by calling this method.

User Class

+setUserInfo(para : ArrayList []) : Boolean
Administrator can set the user information so that he/she can manage.

+getUserInfo(id : int) : User
Administrator can get information from all of the users.

Student Class

+setUserInfo(para : ArrayList []) : Boolean
Administrator and the student can set the student information.

+getUserInfo(id : int) : Student
Administrator and Employer can use this to get information about the student.

Employer Class

+setUserInfo(para : ArrayList []) : Boolean
Administrator and Employer can set the employer information

+getUserInfo(id : int) : Employer
Administrator can use this to get information about the employer. Student can use this to get limited information about the employer.

Administrator Class

+setUserInfo(para : ArrayList []) : Boolean
Only administrator can use this to set the administrator information.

+getUserInfo(id : int) : Administrator
Only administrator can use this to get information about an administrator

+deregisterStudent(stuID : int) : Boolean
Administrator can deregister student with a series of bad reports and will be blacklisted.

Anonymous Class

+viewJobs() : Job []
Anonymous user can view all the job listings.

UserPrivilegeManager Class

+register(user : User) : Boolean
This function is used to register the user and store the information into the database. The user type is already been pre-defined by the administrator.

+logIn(user : User) : Boolean
This function is used to log in all different type of users into the system

+readPrivilege(type : int) : Authority []
When user is trying to login into the system, this function is used to read what type of privileges to identity the user type.

+addPrivilege(authority : Authority) : Boolean
This function is used to give privilege to different user to perform different action

+removePrivilege(authorityID : int) : Boolean

This function is used to revoke privilege to different user to perform different action

+assignPrivilege(userType : int, authorityID) : Boolean
Once the user is determined, the system will use this function to assign all the default privileges depending on the user type.

+generateMenu(authorities : Authority []) : boolean
Different menu will be generated depending on the role of the user that logged in.

+checkPrivilege(userType : int) : Boolean
This function will check on what privileges that each user have before determining what the user can do.

Authority Class

+addPrivilege(authority : Authority) : Boolean
This function is used to give privilege to different user to perform different authority actions.

+removePrivilege(authorityID : int) : Boolean
This function is used to revoke privilege to different user to perform different authority actions.

+setAuthorityInfo(para : ArrayList []) : Boolean
The authority function is pre defined in the database to store information on what action that can be performed.

+getAuthorityInfo(id : int) : Authority
Gets the authority information on about the action performed.

Interview Class
+createInterview(para : ArrayList []) : Interview
The employer can create an interview session with the shortlisted student.

+setInterviewInfo(para : ArrayList []) : boolean
The employer can set the interview information (eg. The Date and Time)

+getInterviewInfo(interviewID : int) : Interview
Shortlisted students can retrieve the interview information such as the Interview ID

+arrangeInterviewTimeOption(time : datetime) : Boolean
The employer can arrange a few interview time for the shortlisted student to select.

+chooseInterviewTime(interviewID : int, time : datetime, stuID : int) : boolean
Shortlisted students can use this function to choose the interview time.

Aptitude Test Class

+createAptitudeTest(para : ArrayList []) : AptitudeTest
The employer can create Aptitude test for shortlisted full time job seekers

+setAptitudeTestInfo(para : ArrayList []) : Boolean
The employer can set the information on about the aptitude test.

+getInAptitudeTestInfo(aptitudeTestID : int) : AptitudeTest
Gets the Aptitude test information

+arrangeAptitudeTestTimeOption(time : datetime) : Boolean
The employer can arrange the times where the students can select when to take the aptitude test.

+chooseAptitudeTestTime(time : datetime, stuID : int) : Boolean
The shortlisted students can choose when to take the aptitude test time from the given time options by the employer

Rank Class

+setRankRatingInfo(empID : int, type : int, id : int, score : int) : Boolean
Sets the information of the rank rating.

+getRankInfo(type : int, id : int) : Rank
Get the ranking information on the students or the employer.

+calculateRanking() : double
Calculates the ranking before and after a student or employer ranks.

+setFeedBackInfo(empID : int, type : int, id : int, message : string) : boolean
Employer can send a feedback to the student by calling this function.

+getFeedBack(empID : int, id : int) : string
Gets the feed back if this function is called.

TimeFeature Class

+createJobFeature(jobID : int, type : int, timeType : int, value : int) : TimeFeature
The employer can create a job feature upon create a job into the job listing.

+setJobFeatureInfo(jobID : int, type : int, timeType : int, value : int) : Boolean
The employer sets the information about this job that the employer listed.

+getJobFeatureInfo(jobID : int, type : int) : TimeFeature
Students can get the job's information by calling this method.

Notification Class

+createNotification(type : int, content : string) : Notification
The employer can create a notification to inform a student. (Eg. To inform the student whether he/she was accepted or rejected)

+getNotificationInfo(notiID : int) : Notification
Gets the information for the notification.

+setNotificationlContent(content : string) : Boolean
The system have already predefined the content for a notification

+getNotificationContent() : text
The system can get the notification content by using this function.

+sendNotification(stuID : Student []) : Boolean
The system can send the notification to the intended sender.

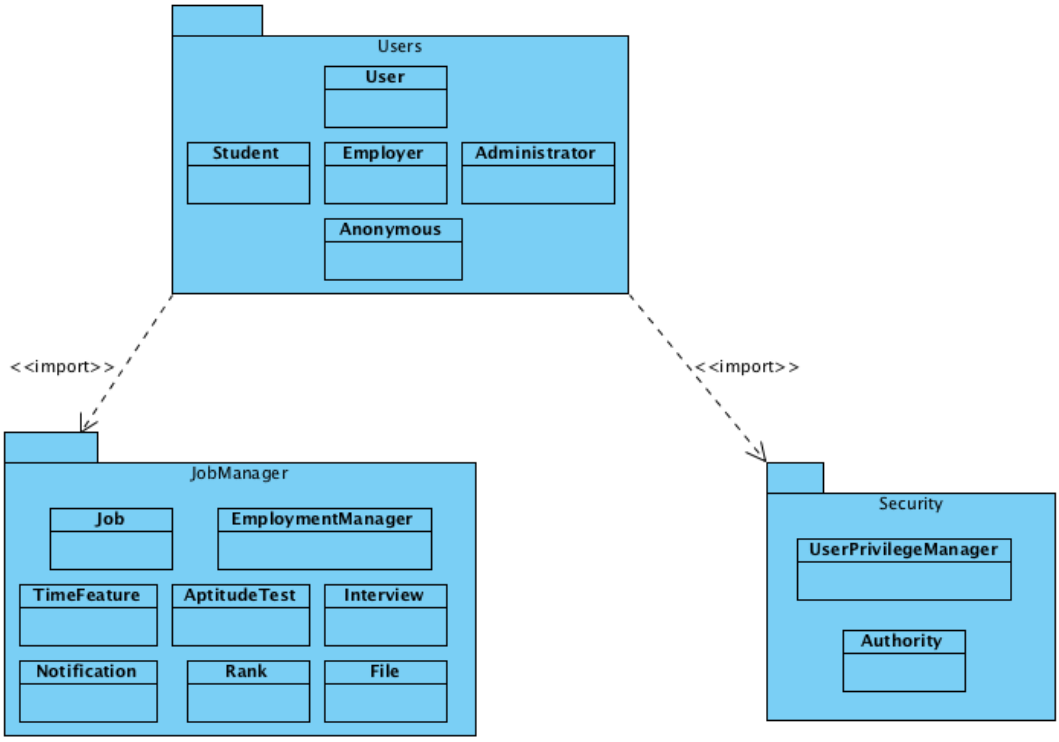
File Class

+uploadFile(filePath : string, type : int) : int
This function is used to upload either the student's resume, transcript or reference.

+getFileInfo(stuID : int, type : int) : File []
This function is used to get the information of a file. An employer can use this function to read about a student.

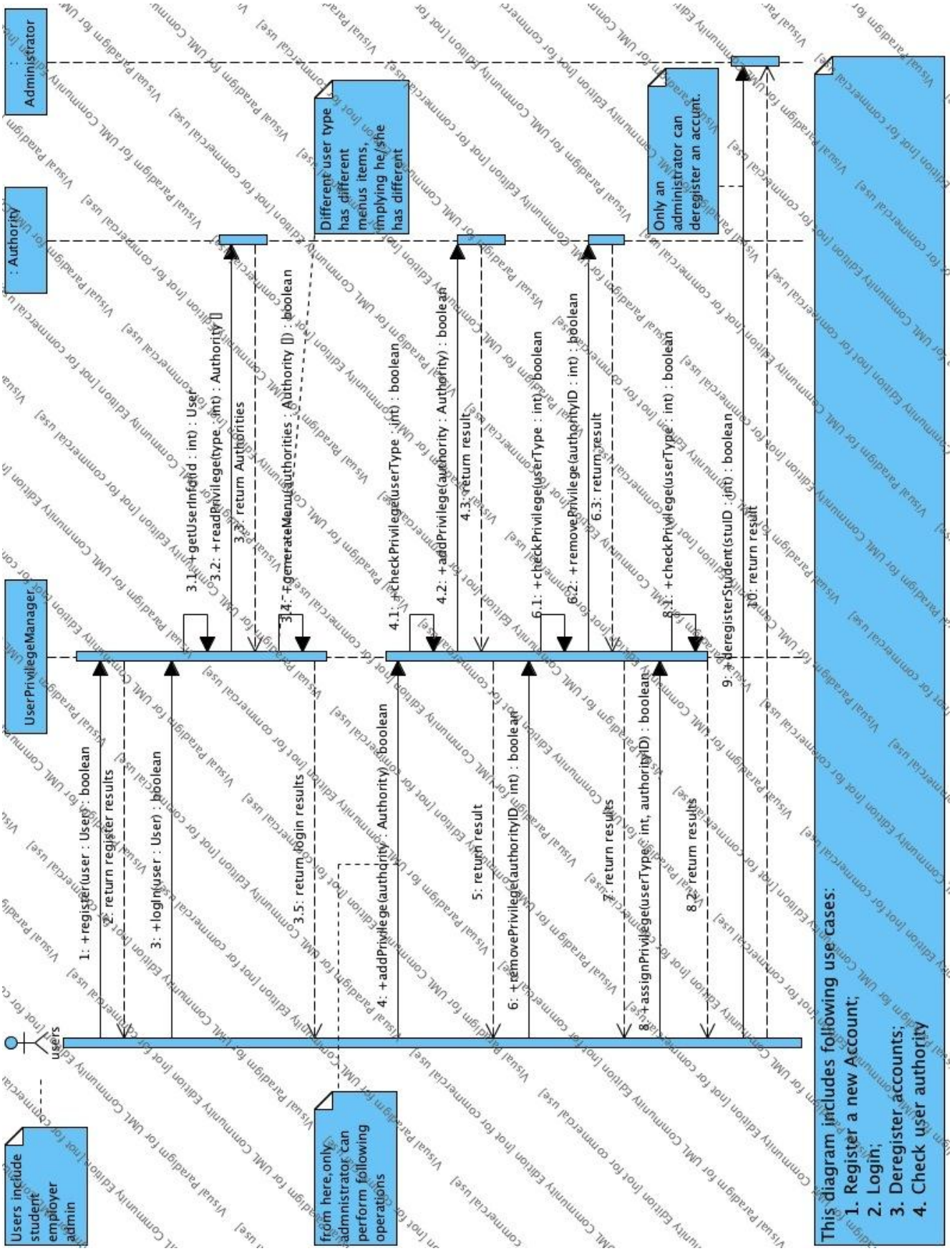
+getFileInfo(fileID : int) : File
This function is used to get the information of a file. An employer can use this function to read about a student.

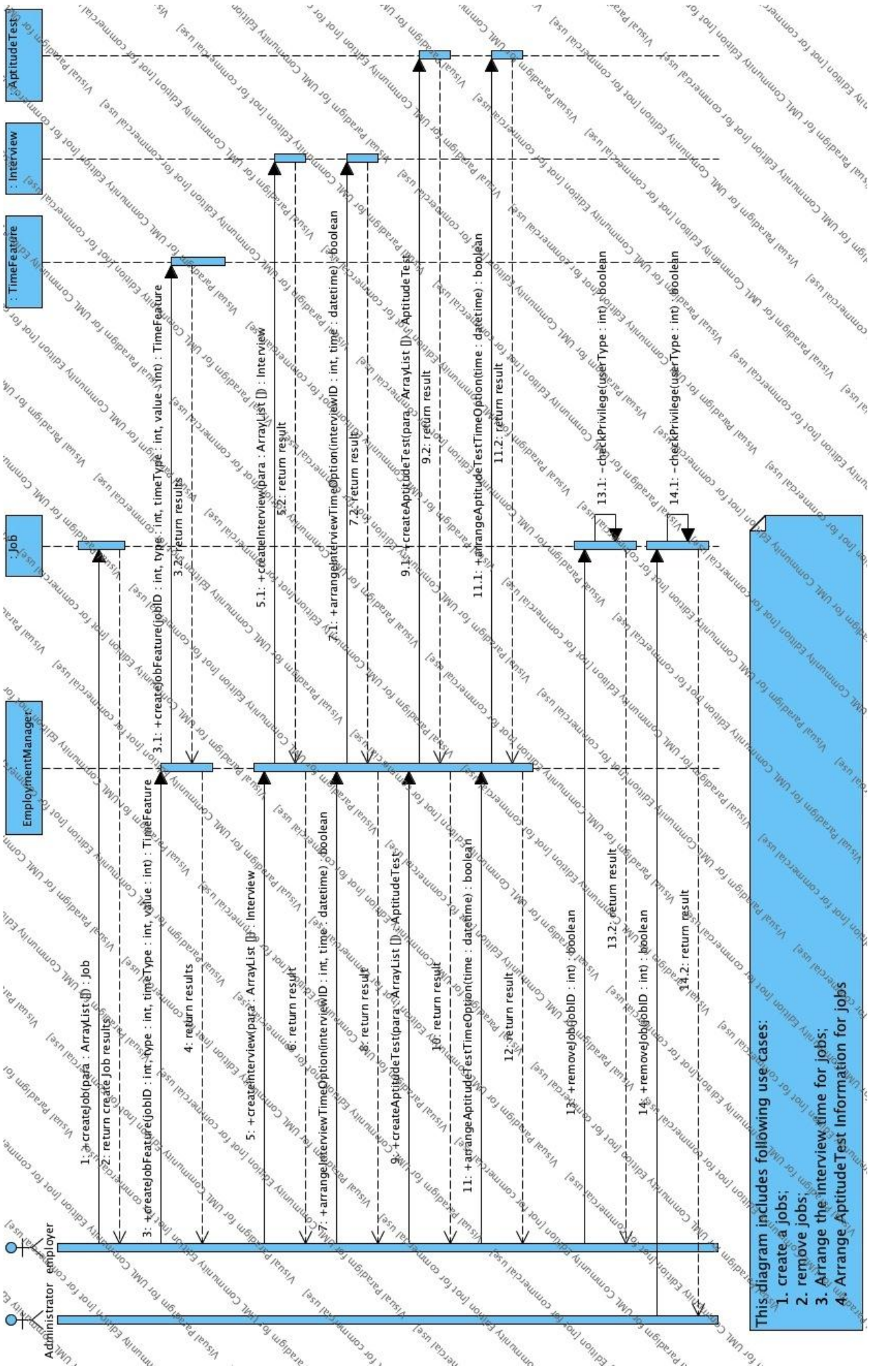
2.4 Package diagram

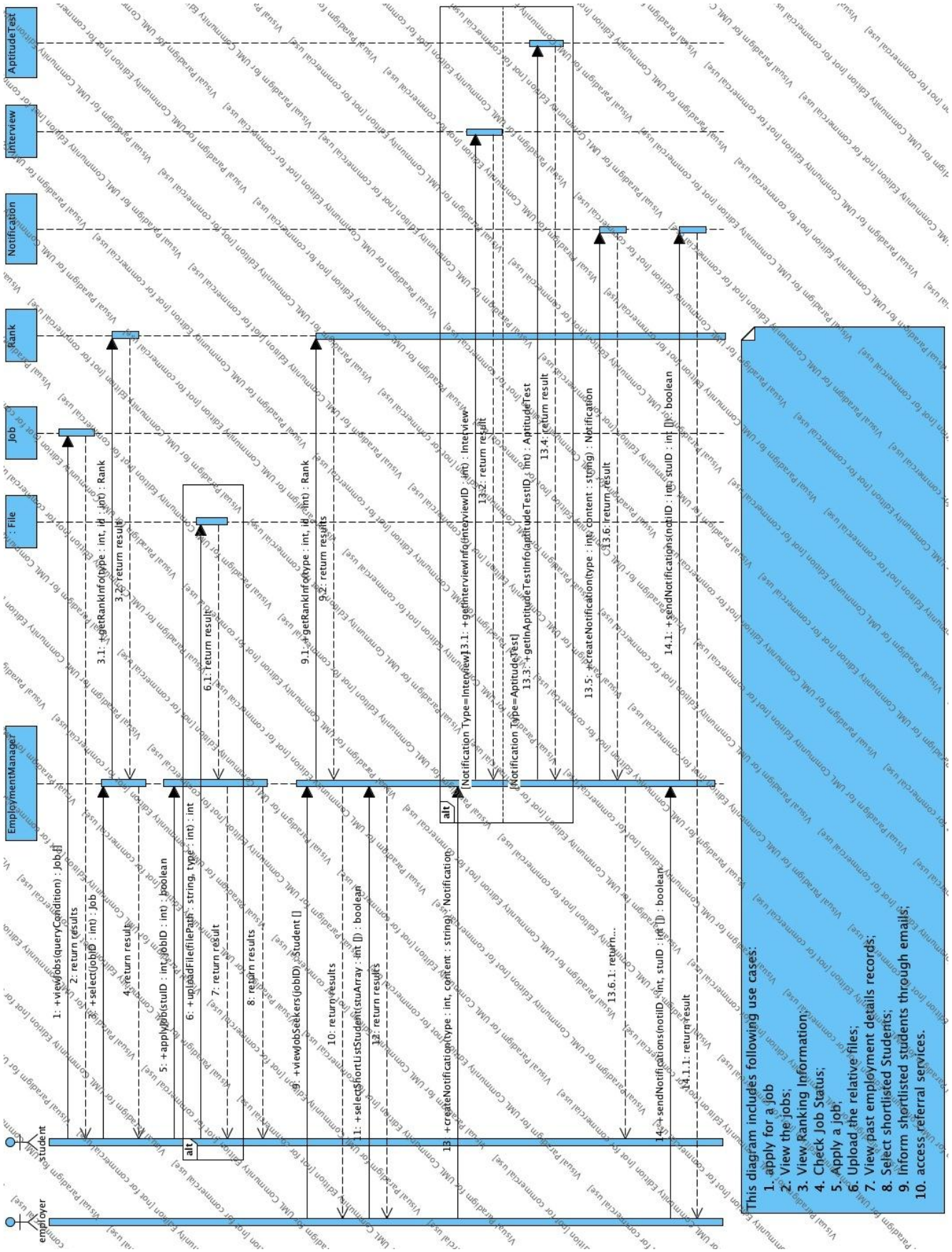


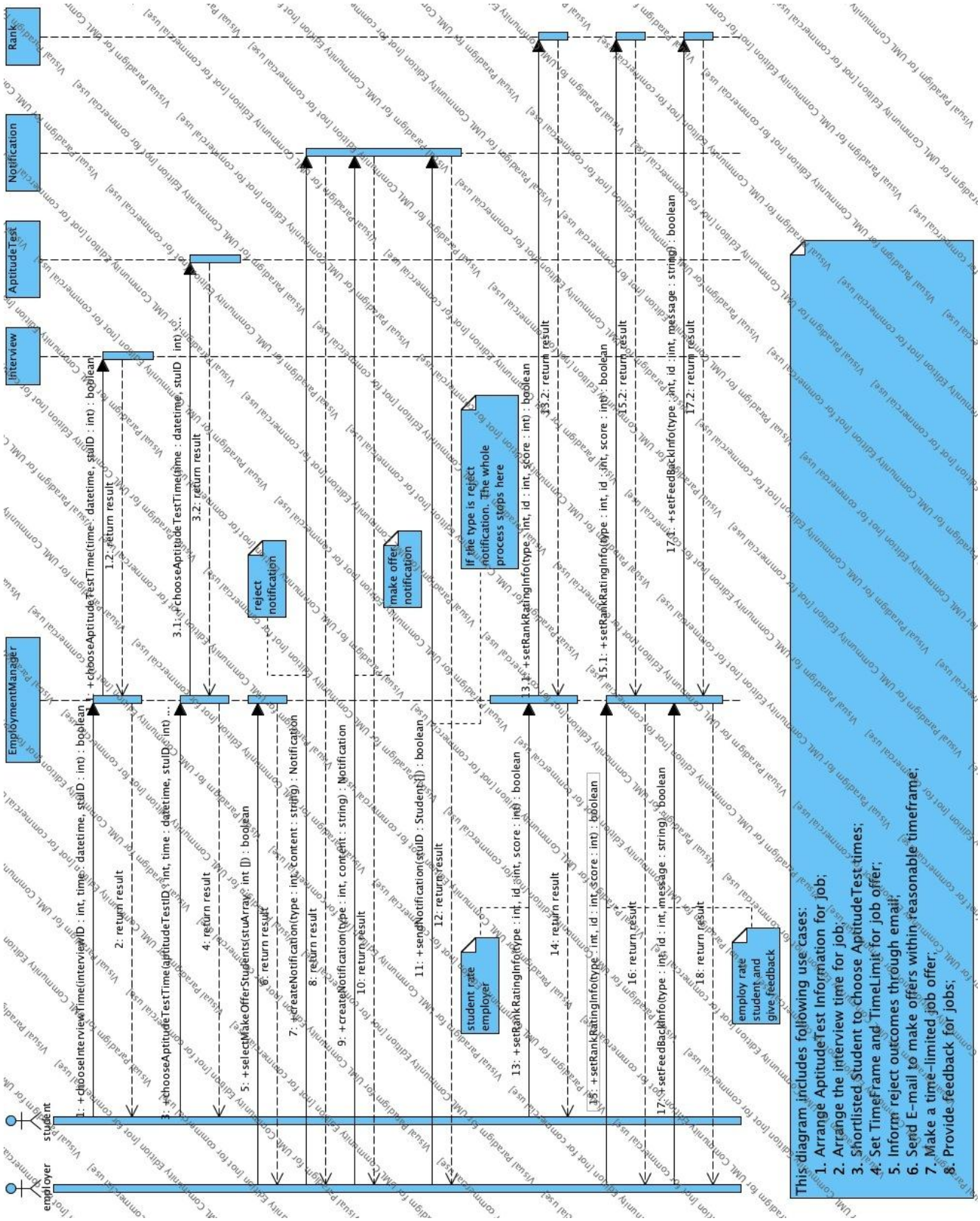
In this package diagram, there are only mainly three different package in this system; the User, JobManager and Security packages. The Security package handles the classes necessary to control the privileges of different type of users that logs in to the system. The Users package consists of the type of user that will be using the system such as the student, employer, administrator and the anonymous users. The JobManager package in the other hand is use to manage the all jobs for the employers to the students; (e.g. whether the student is finding for a job or the employer is looking for employee) will all be managed in this package.

2.5 Sequence diagrams

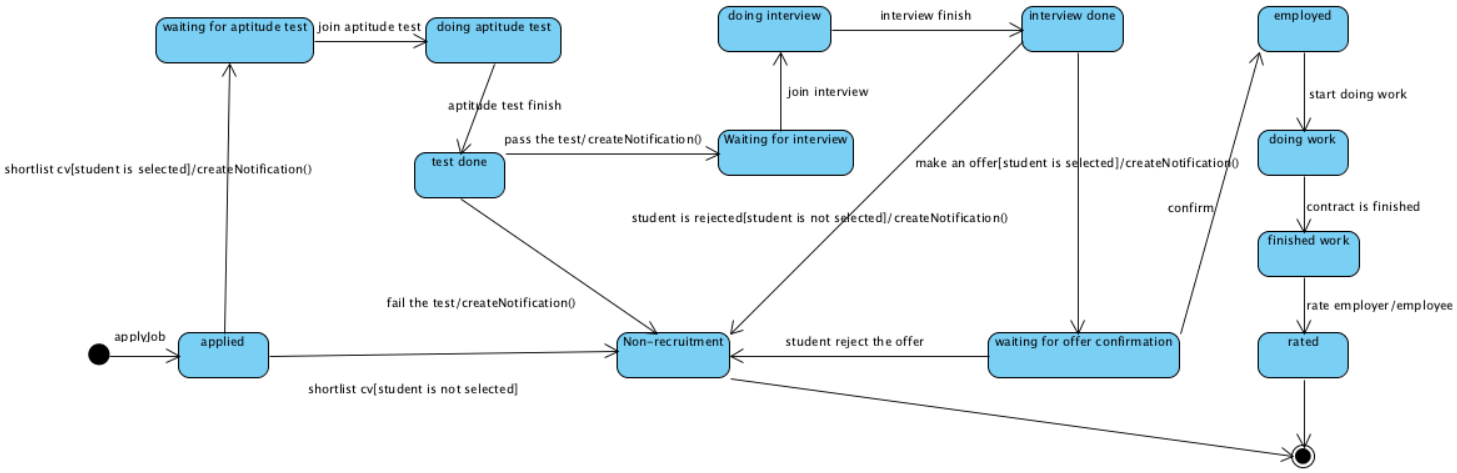

: Administrator
: Authority
UserPrivilegeManager
users
Different user type has different menus items, implying he/she has different
Only an administrator can deregister an accunt.
1: +register(user : User) : boolean
2: return register results
3: +logIn(user : User) : boolean
3.1: +getUserInfo(id : int) : User
3.2: +readPrivilege(type : int) : Authority []
3.3: return Authorities
3.4: +generateMenu(authorities : Authority []) : boolean
3.5: return login results
4: +addPrivilege(authority : Authority) boolean
4.1: +checkPrivilege(userType : int) : boolean
4.2: +addPrivilege(authority : Authority) : boolean
4.3: return result
5: return result
6: +removePrivilege(authorityID : int) : boolean
6.1: +checkPrivilege(userType : int) : boolean
6.2: +removePrivilege(authorityID : int) : boolean
6.3: return result
7: return results
8: +assignPrivilege(userType : int, authorityID) : boolean
8.1: +checkPrivilege(userType : int) : boolean
8.2: return results
9: +deregisterStudent(stuID : int) : boolean
10: return result
Users include student employer admin
from here,only administrator can perform following operations
This diagram includes following use cases:
1. Register a new Account;
2. Login;
3. Deregister accounts;
4. Check user authority


: AptitudeTest
: Interview
: TimeFeature
: Job
EmploymentManager
Administrator
employer
1: +createJob(para : ArrayList []) : Job
2: return create Job results
3: +createJobFeature(jobID : int, type : int, timeType : int, value : int) : TimeFeature
4: return results
5: +createInterview(para : ArrayList []) : Interview
6: return result
7: +arrangeInterviewTimeOption(interviewID : int, time : datetime) : boolean
8: return result
9: +createAptitudeTest(para : ArrayList []) : AptitudeTest
10: return result
11: +arrangeAptitudeTestTimeOption(time : datetime) : boolean
12: return result
13: +removeJob(jobID : int) : boolean
13.2: return result
14: +removeJob(jobID : int) : boolean
14.2: return result
3.1: +createJobFeature(jobID : int, type : int, timeType : int, value : int) : TimeFeature
3.2: return results
5.1: +createInterview(para : ArrayList []) : Interview
5.2: return result
7.1: +arrangeInterviewTimeOption(interviewID : int, time : datetime) : boolean
7.2: return result
9.1: +createAptitudeTest(para : ArrayList []) : AptitudeTest
9.2: return result
11.1: +arrangeAptitudeTestTimeOption(time : datetime) : boolean
11.2: return result
13.1: -checkPrivilege(userType : int) : boolean
14.1: -checkPrivilege(userType : int) : boolean
This diagram includes following use cases:
1. create a jobs;
2. remove jobs;
3. Arrange the interview time for jobs;
4. Arrange AptitudeTest Information for jobs


AptitudeTest
Interview
Notification
Rank
Job
: File
EmploymentManager
student
employer
1: +viewJobs(queryCondition) : Job []
2: return results
2: +select(jobID : int) : Job
3: return result
3.1: +getRankInfo(type : int, id : int) : Rank
3.2: return result
4: return result
5: +applyJob(stuID : int, jobID : int) : boolean
6: +uploadFile(filePath : string, type : int) : int
6.1: return result
7: return result
8: return results
9: +viewJobSeekers(jobID) : Student []
9.1: +getRankInfo(type : int, id : int) : Rank
9.2: return results
10: return results
11: +selectShortListStudent(stuArray : int []) : boolean
12: return results
alt
[Notification Type=Interview]
13 : +createNotification(type : int, content : string) : Notification
13.1: +getInterviewInfo(interviewID : int) : Interview
13.2: return result
13.3: +getInAptitudeTestInfo(aptitudeTestID : int) : AptitudeTest
13.4: return result
[Notification Type=AptitudeTest]
13.5: +createNotification(type : int, content : string) : Notification
13.6: return result
13.6.1: return...
14: +sendNotifications(notiID : int, stuID : int []) : boolean
14.1: +sendNotifications(notiID : int, stuID : int []) : boolean
14.1.1: return result
This diagram includes following use cases:
1. apply for a job
2. View the jobs;
3. View Ranking Information;
4. Check Job Status;
5. Apply a job;
6. Upload the relative files;
7. View past employment details records;
8. Select shortlisted Students;
9. Inform shortlisted students through emails;
10. access referral services.

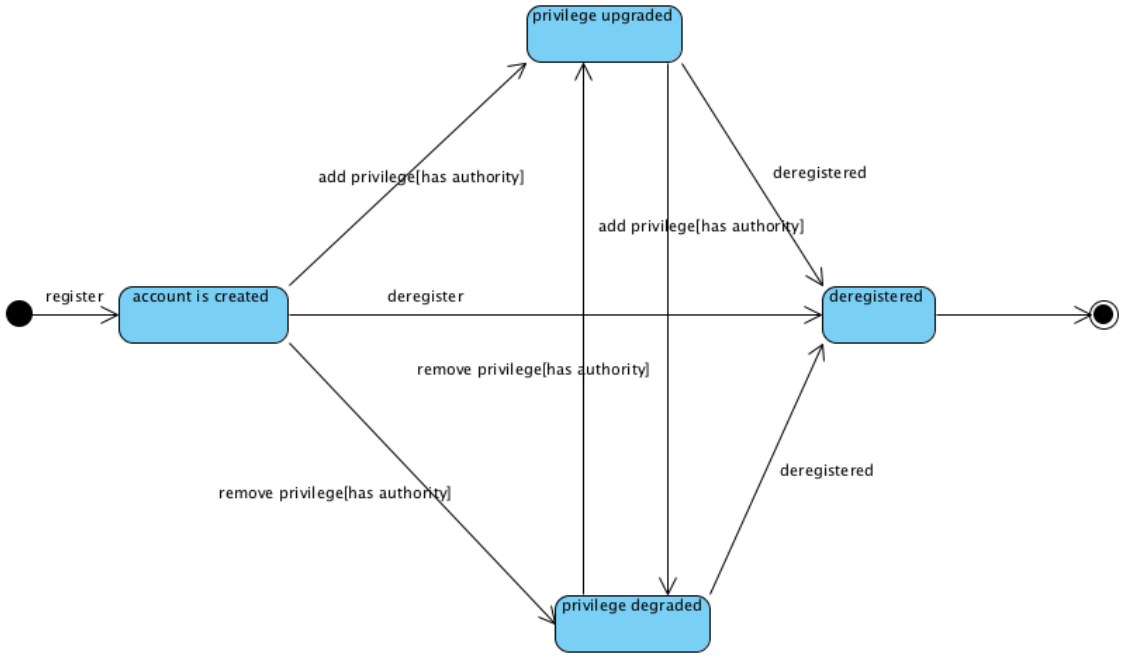

Rank
Notification
AptitudeTest
Interview
EmploymentManager
student
employer
1: +chooseInterviewTime(interviewID : int, time : datetime, stuID : int) : boolean
2: return result
3: +chooseAptitudeTestTime(aptitudeTestID : int, time : datetime, stuID : int) : ...
4: return result
5: +selectMakeOfferStudents(stuArray : int []) : boolean
6: return result
7: +createNotification(type : int, content : string) : Notification
8: return result
9: +createNotification(type : int, content : string) : Notification
10: return result
11: +sendNotification(stuID : Student[]) : boolean
12: return result
13: +setRankRatingInfo(type : int, id : int, score : int) : boolean
14: return result
15: +setRankRatingInfo(type : int, id : int, score : int) : boolean
16: return result
17: +setFeedBackInfo(type : int, id : int, message : string) : boolean
18: return result
1.1: +chooseInterviewTime(time : datetime, stuID : int) : boolean
1.2: return result
3.1: +chooseAptitudeTestTime(time : datetime, stuID : int) : ...
3.2: return result
reject notification
make offer notification
If the type is reject notification. The whole process stops here
13.1: +setRankRatingInfo(type : int, id : int, score : int) : boolean
13.2: return result
15.1: +setRankRatingInfo(type : int, id : int, score : int) : boolean
15.2: return result
17.1: +setFeedBackInfo(type : int, id : int, message : string) : boolean
17.2: return result
student rate employer
employ rate student and give feedback
This diagram includes following use cases:
1. Arrange Aptitude Test Information for job;
2. Arrange the interview time for job;
3. Shortlisted Student to choose AptitudeTest times;
4. Set TimeFrame and TimeLimit for job offer;
5. Inform reject outcomes through email;
6. Send E-mail to make offers within reasonable timeframe;
7. Make a time-limited job offer;
8. Provide feedback for jobs;

2.6 State diagram



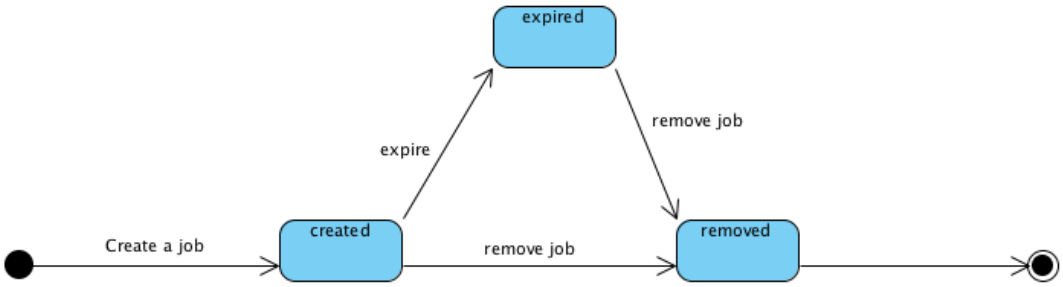
Employment State Diagram

1.When student applies a job, the state of job becomes applied.
2. When employer finishes shortlist, state may become ‘waiting for aptitude test’ or ‘non-recruitment’ which depends on if the student is selected or not.
3. When student starts aptitude test, the state becomes ‘doing aptitude test’.
4. When aptitude test finishes, the state becomes ‘done’.
5. If student passes the test , the state becomes ‘waiting for interview’ or state becomes ‘non recruitment’ if he/she fails.
6. When student starts interview, the state becomes ‘doing interview’. When finishes, state becomes ‘interview done’.
7. If student is selected, state becomes ‘waiting for offer confirmation’ or state becomes ‘non recruitment’ if he/she is not selected.
8. If student confirms the offer, state becomes ‘employed’ or state becomes ‘non-recruitment’.
9.When student starts work, state becomes ‘doing work’.
10. When contract is finished, the state becomes ‘finished work’.
11. When student and employer both do the rating, state becomes ‘rated’.



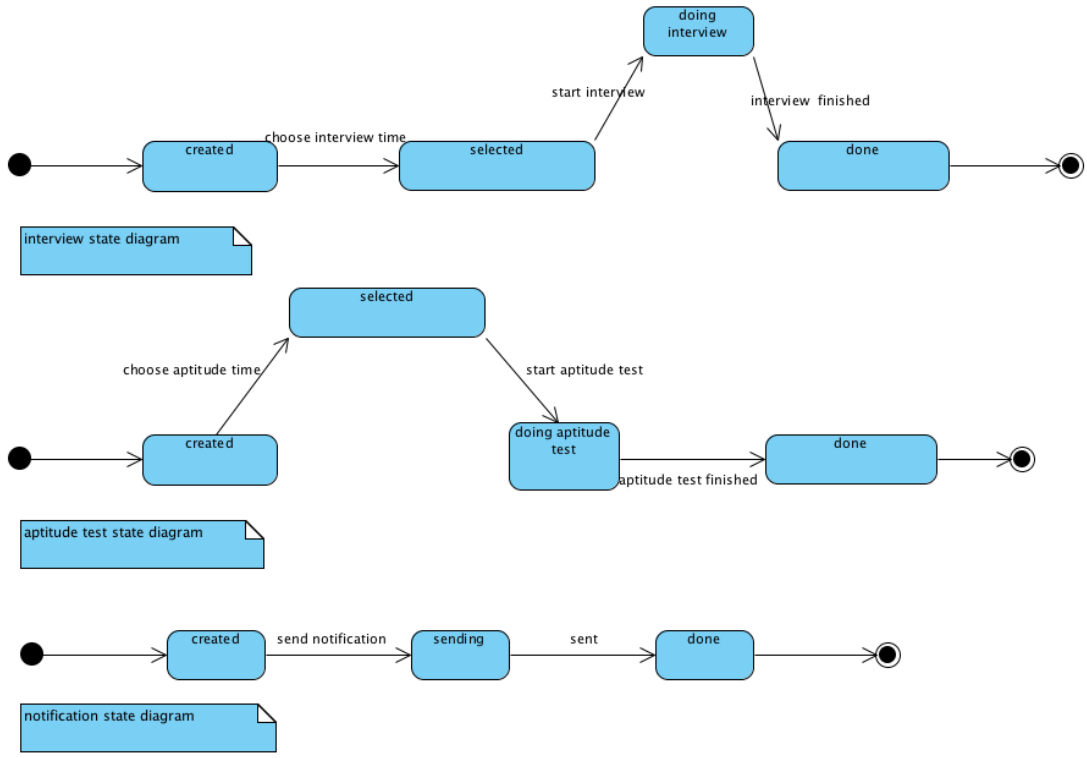
Account State Diagram

1. When student do the registration, the account is created. Administrator may add privilege or remove privilege of a account.
2. Account may be deregistered by administrator.



Job State Diagram

1. When a job is created, state of job becomes ‘created’.
2. When a job is expired, job state becomes ‘expired’.
3. When employer or system removes the job, job state becomes ‘removed’.



Interview State Diagram

1. When an interview is created, the state of interview becomes ‘created’.
2. When an interview time is selected, interview state becomes ‘selected’.
3. When the student starts interview then the interview state becomes ‘doing interview’ and when he/she finishes the interview then the state becomes ‘done’.

Aptitude Test Sequence Diagram

1. When an aptitude test is created, the state of aptitude test becomes ‘created’.
2. When an aptitude test time is selected, aptitude test state becomes ‘selected’.
3. When the student starts aptitude test then the aptitude test state becomes ‘doing aptitude test ’ and when he/she finishes the aptitude test then the state becomes ‘done’.

2.7 Database Table

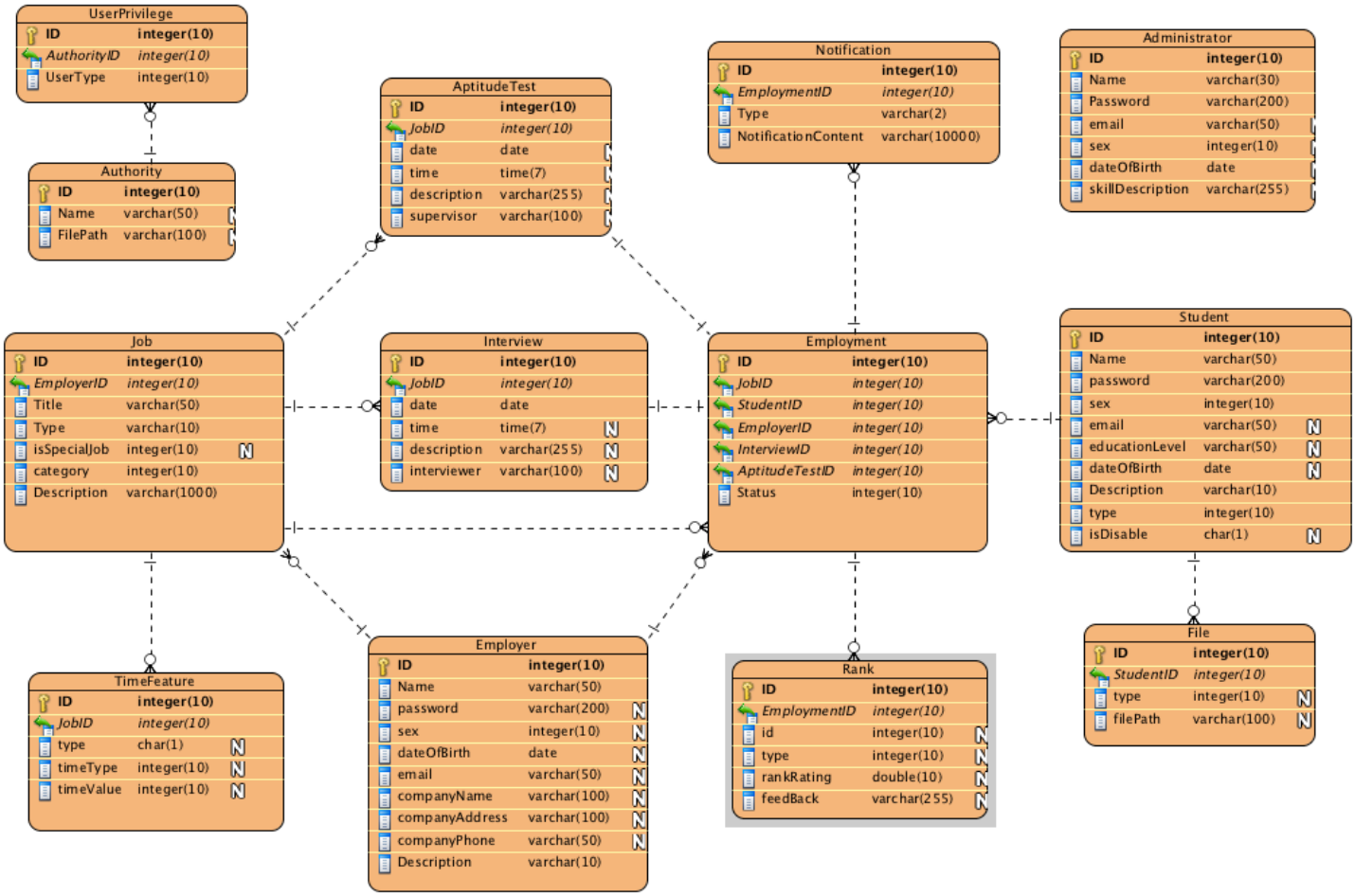
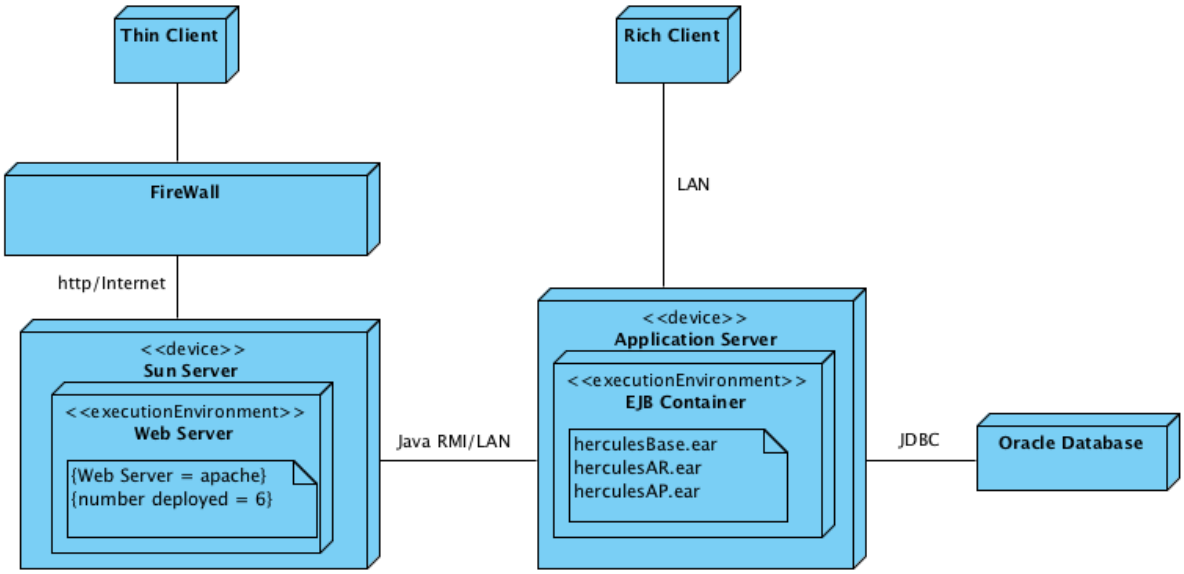


Figure 1.3 shows the database table of the entire system.

The Student, Administrator and Employer tables are use to store all the information about the user.
The UserPrivilege and Authority table is use to store secure information that implements different type of users on which type of privileges given to use the system.
The File table is use to store the uploaded files (e.g. resume, transcript and reference) by the students.
The Job table is to store the job listing information. One of the important information that it can store is the type of job (whether it is a full time job or part time job). Another information that it can store is whether the job is for disabled students or for normal students.
The Interview table is to store all interview information. The Aptitude table is to store all aptitude information.
The Notification table is to store the notification history for specific student. One of the important information stored in this table is the type of notification (e.g. whether the notification is about acceptance or the rejection of an application job).
The Rank table is to store the student's and the employer's ranking information. One of the important information stored is the rank rating where either the employer will rate the student or via versa.
The TimeFeature table is used to store the time type (either a time frame or time limited) for a specific job.

2.8 Deployment diagram



The thin client in this deployment diagram can be represented like a typical browser where as the rich client can be represented like a desktop application server. The web server used in this system is apache and we have included six nodes. The web server is also protected by a firewall as well. The thin client can use the http protocol to connect to the web server. We use the EJB Container as the framework for our application server. The web server can use Java RMI to talk directly to the application server. Rich Clients are able to connect to the application server. The application server is also connected directly to the Oracle database by the JDBC protocol.

2.9 Assumptions And Dependencies

2.3.1 Assume that a full time employment and casual time employment will go though a same process when the employers are recruiting students.

2.3.2 The students who are applying for a casual time job do no need to go though an interview process unlike students who are applying for a full time job employment.

2.3.3 Assume that only students who are applying for a casual time employment are given a certain time frame to accept an offer given by an employer.

2.3.4 Assume that only employer who are recruiting students for a casual time job employment will be given a time-limited offer for these students to accept.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 Facility Staff

3.1.1.1 The system must allow employers and existing students to register.

3.1.1.2 The system must provide access to relevant records (such as records for employers, students, jobs and resumes) for students, employers, administrators and other interested parties

3.1.1.3 The system must allow administrative staff to manage the records

3.1.2 Job Management

3.1.2.1 The system will allow new jobs to be created

3.1.2.2 The system will allow jobs to be removed by employers (only their own) or administrators.(on expiry)

3.1.2.3 The system will allow everyone to view the jobs but only registered students to apply.

3.1.2.4 The system should allow shortlisted students to be informed through emails.

3.1.2.5 The system will allow selected applicants to choose interview times.

3.1.2.6 The system should allow students to be informed of the outcomes through emails.

3.1.3 Security

3.1.3.1 User must log in for the system to identify what type of user before deciding on which privileges that the user has that can be given.

3.1.3.2 The system must limit access to information depending on the role of the person. Only management staff must be granted access to confidential information and given the privileges to deregister students.

3.1.4 Job Seekers Management

3.1.4.1 The system will allow registered students to seek for casual employment (hourly, short-term or internships). All job seekers should be allowed to upload the resume and transcripts.

3.1.4.2 The system will maintain a record of all past employment details for each student

3.1.4.3 The system will provide average ranking of students by past employers to prospective (casual) employers.

3.1.4.4 The employer should be presented to select, offer jobs and provide feedback. The user interface is designed for employers to select job seekers and make offers for casual employment. Different user profile (busy employers) and the type of employment offered is also considered.

3.1.4.5 Those completing casual jobs should be allowed to rank the employer.

3.1.4.6 The system should allow employers to make a time-limited casual job offer (for example, job offer must be accepted within next xx hours). These offers must be sent as emails to job seekers. These emails should include employment details and the average ranking of for that employer so that an informed decision can be made.

3.2 Non-Functional Requirements

3.2.1 System Management and Performance

3.2.1.1 Performance requirements

3.2.1.1.1 Response time should not exceed 10ms in 95% of the cases.

3.2.1.2 Safety requirements

3.2.1.2.1 No safety requirements have been identified.

3.2.1.3 Security Requirements

3.2.1.3.1 The system will provide audit trails for all administrative users of the system

3.2.1.3.2 The system will limit access to employer details to prevent students directly contacting them.

3.2.1.3.3 The system must limit access to information depending on the role of the person.

3.2.1.4 Availability Requirements

3.2.1.4.1 The system will be available on a 24/7 basis, with allowable maintenance windows of 1 hour per fortnight.

3.2.1.5 Scalability Requirements

3.2.1.5.1 It is expected in the initial stage there will be 1000 registered students and 100 potential employers. These numbers are expected to double each year for the next four years.

3.3 Classification of functional requirements

Functionality	Type
3.1.1.1 The system must allow employers and existing students to register.	Essential
3.1.1.2 The system must provide access to relevant records for students, employers, administrators and other interested parties	Essential
3.1.1.3 The system must allow administrative staff to manage the records	Essential
3.1.2.1 The system will allow new jobs to be created	Essential
3.1.2.2 The system will allow jobs to be removed by employers (only their own) or administrators.(on expiry)	Essential
3.1.2.3 The system will allow everyone to view the jobs but only registered students to apply.	Essential
3.1.3.1 The system will limit access to employer details to prevent students directly contacting them.	Desirable
3.1.3.2 The system must limit access to information depending on the role of the person. Only management staff must be granted access to confidential information and given the privileges to deregister students.	Essential
3.1.4.1 The system will allow registered students to seek for casual employment (hourly, short-term or internships). All job seekers should be allowed to upload the resume and transcripts.	Essential
3.1.4.2 The system will maintain a record of all past employment details for each student	Essential
3.1.4.3 The system will provide average ranking of students by past employers to prospective (casual) employers.	Essential

3.1.4.4 The employer should be presented with uncluttered screens and easy way to select, offer jobs and provide feedback. The user interface designed should allow an average employer to select job seekers and make offers for casual employment within reasonable timeframe. You should consider the user profile (busy employers) and the type of employment offered when deciding on a reasonable timeframe.	Optional
3.1.4.5 Those completing casual jobs should be allowed to rank the employer.	Essential
3.1.4.6 The system should allow employers to make a time-limited casual job offer (for example, job offer must be accepted within next xx hours). These offers must be sent as emails to job seekers. These emails should include employment details and the average ranking of for that employer so that an informed decision can be made.	Essential
3.2.1.1 The system will be available on a 24/7 basis, with allowable maintenance windows of 1 hour per fortnight.	Desirable
3.2.1.2 The system will provide audit trails for all administrative users of the system.	Essential
3.2.1.3 Response time should not exceed 10ms in 95% of the cases. It is expected in the initial stage there will be 1000 registered students and 100 potential employers. These numbers are expected to double each year for the next four years.	Desirable

4. Supporting Information

4.1 Appendix

Student ID = Student ID provided by the system uniquely. Consist of the letter "S" followed by seven digits.

Employer ID = Employer ID provided by the system uniquely. Consist of the letter "E" followed by seven digits.

Administrator ID = Administrator ID provided beforehand. Consist of the letter "A" followed by seven digits.

Register = Includes personal details to be entered into the field box

Time-limited = Extremely short period of time

Audit Trails = System logging all activities

Manage = Modifying an object / Delete an object

Rank = Rating