Addis Ababa Institute of Technology

Center of Information Technology and Scientific Computing

Department of Software Engineering

SiraFelagi.com

Software Design Specification

Team Members

- Abebe Kayimo………………………………………….ATR/9006/08
- Milkias Tonji…………………………………………….ATR/8137/08
- Etsegenet Melese…………………………………….ATR/8845/08
- Eyosias Desta…………………………………………..ATR/3173/08
- Kena Wakwoya………………………………………..ATR/4296/08
- Kalkidan Getnet………………………………………ATR/6219/08
- Melkam Beyene………………………………………ATR/3392/08

Advisors: Natnael A

Table of Contents

List of Tables

List of figures

Definitions, Acronyms, Abbreviations

- UML: Unified Modeling Language..
- PHP: a server side programing language
- Cont. : controller class

1. Introduction

1.1 Purpose

The purpose of System Design document is to show a slightly detailed technical implementation of system requirements.

1.2 General Overview

The software we are designing is a web app called SiraFelagi. It will be used to connect job seekers to vacant jobs offered by companies. It also allows companies to give online examinations to filter most of the applicants before interviewing them.

The implementation will be done with a three tier architecture. Details are further provided in this document.

1.3 Development Methods & Contingencies

An **MVC** architecture is adopted for the implementation of this project. This will help us to clearly address all necessary features with a separation of responsibilities among the developers of this system.

We will use an Object-Oriented programing scheme for designing with the following technologies:

- Back end development with java
- Front end development with XML markup language
- A database system running on MySQL server.
- Apache Web server to run the PHP scripts.

The development might run into some shortcoming as a result of incompatible technologies. Some features in the system might require a latest versions of technologies that might not work on the above stated ones. So, a work around for this hindrance will be to update our frameworks to a newer versions.

2. System Architecture

2.1 Subsystem decomposition

The upper level component in this project is the **User Interface** of the system. All other subcomponent are derived and will be handled by the user's interaction to our system through the provided User Interfaces.

2.1.1 User Validation

This component ensures that the services of the system are only provided for the authenticated users. Its sub-component include:

- **Password Management**: Access to the main features of the system is only for authenticated users. The log in scheme is based on the type of the user that wishes an access to the system. I.e. Applicant, Company or Admin.
  This sub-component also allows users to change their password, and also reset it to a new one if they forget it.

2.1.2 Resource Management

In our system, as described in earlier documentations, users are categorized as Companies, Applicants (jobs seekers) and admins. This sub-component manages resources that can be provided to a user based on his/her respective user type.
Thus, attempts of getting to a service by writing valid URLs will be denied if the user is not the type to whom the requested service is provided. Inside this, sub-component we have: ➢ **Documents Management**: This component is for the job seekers, i.e. applicants where they can upload, change and delete their CVs and other related documents.

2.13. Notification

This is one of the direct interaction that companies have with their job candidates. It helps to inform the applicant that he has been called for further interview for the job he/she has applied to.

Applications submitted to the companies will also notify the employer through this subcomponent.

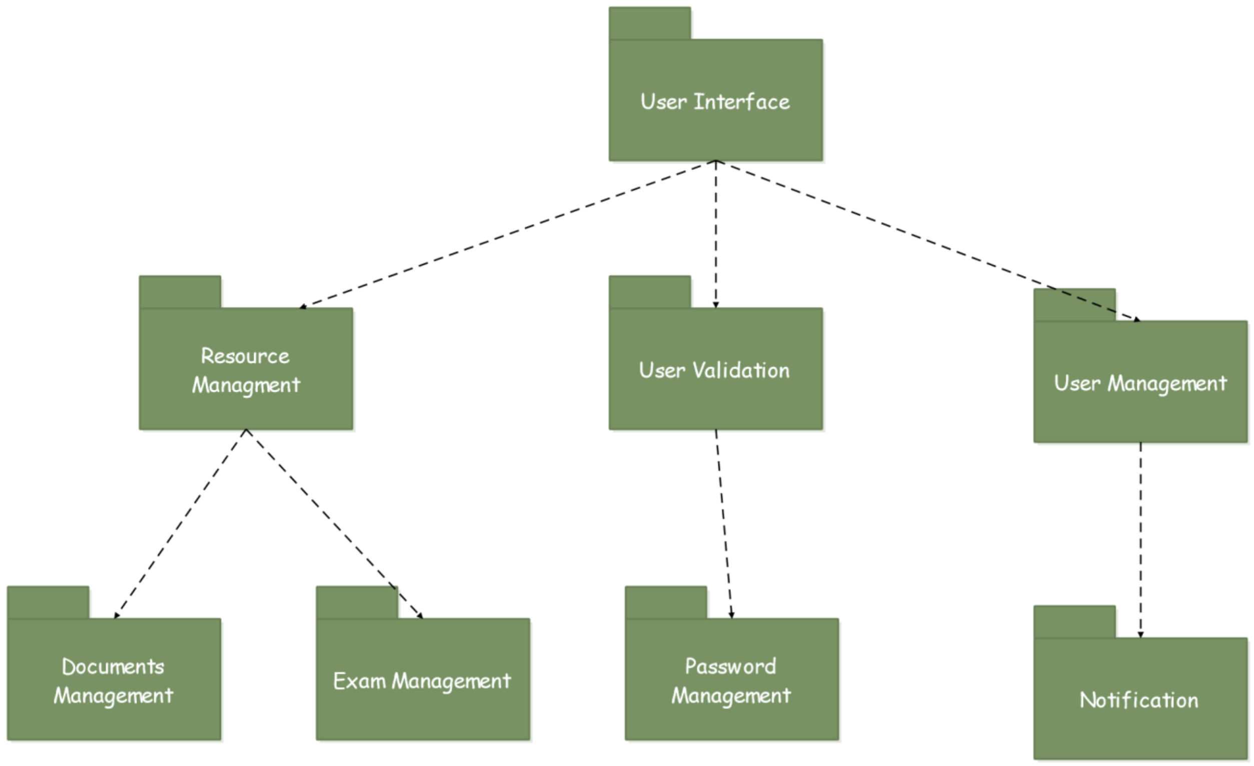


Fig. 2.1 UML Component Diagram

2.2 Hardware/software mapping

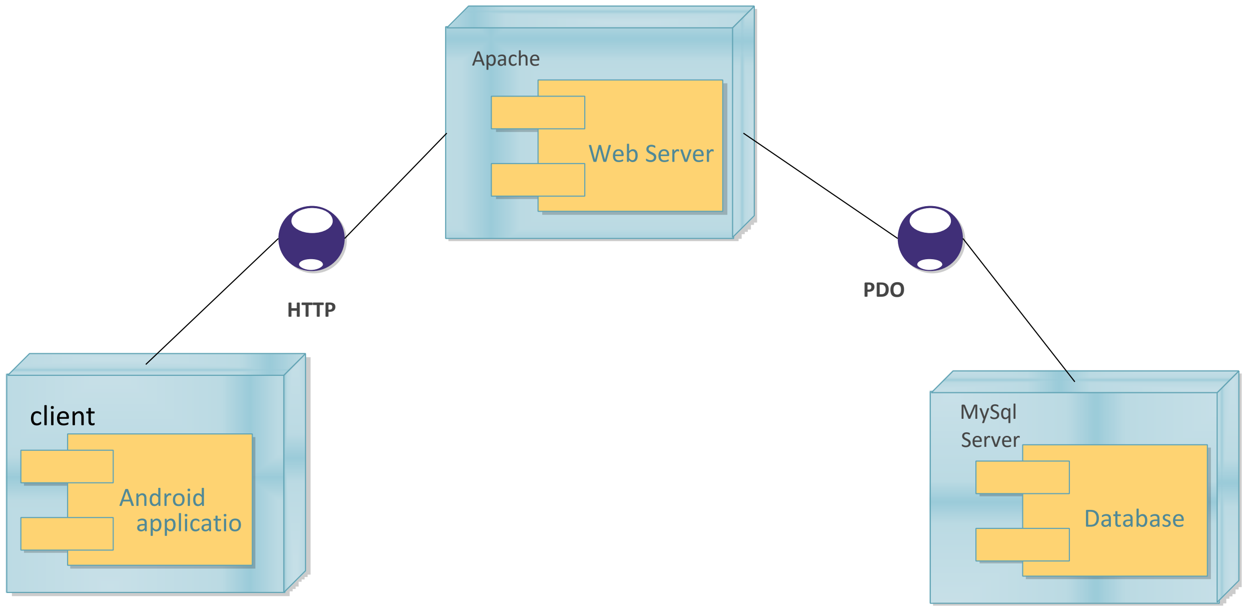
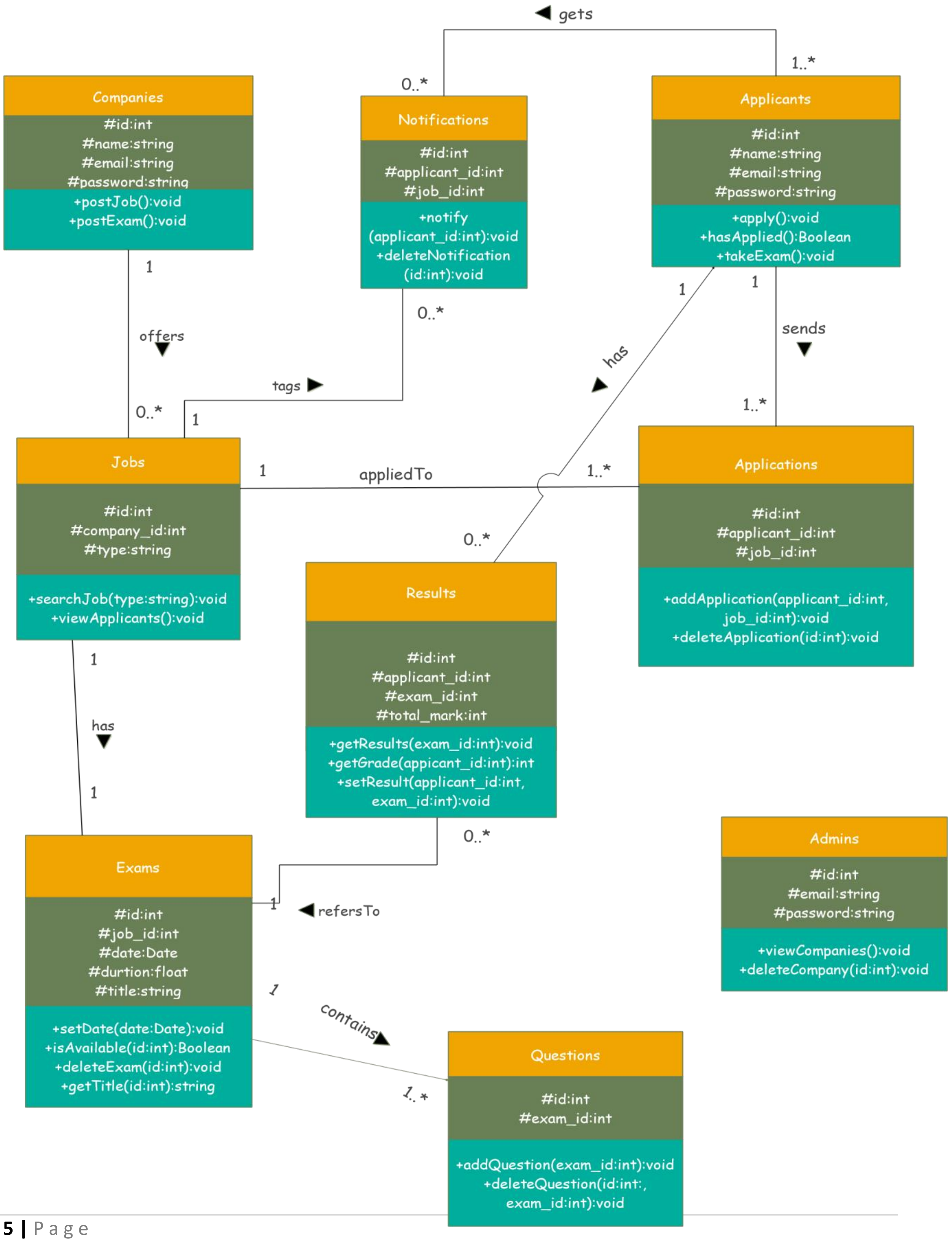


Fig. 2.2 UML Deployment Diagram

3. Object Model

3.1 Class Diagram

Provided on the next page ……


gets
0..*
1..*
Companies
#id:int
#name:string
#email:string
#password:string
+postJob():void
+postExam():void
Notifications
#id:int
#applicant_id:int
#job_id:int
+notify
(applicant_id:int):void
+deleteNotification
(id:int):void
Applicants
#id:int
#name:string
#email:string
#password:string
+apply():void
+hasApplied():Boolean
+takeExam():void
1
offers
0..*
tags
1
0..*
1
has
1
sends
1..*
Jobs
#id:int
#company_id:int
#type:string
+searchJob(type:string):void
+viewApplicants():void
1
appliedTo
1..*
Applications
#id:int
#applicant_id:int
#job_id:int
+addApplication(applicant_id:int,
job_id:int):void
+deleteApplication(id:int):void
0..*
Results
#id:int
#applicant_id:int
#exam_id:int
#total_mark:int
+getResults(exam_id:int):void
+getGrade(appicant_id:int):int
+setResult(applicant_id:int,
exam_id:int):void
1
has
1
0..*
Exams
#id:int
#job_id:int
#date:Date
#durtion:float
#title:string
+setDate(date:Date):void
+isAvailable(id:int):Boolean
+deleteExam(id:int):void
+getTitle(id:int):string
1
refersTo
Admins
#id:int
#email:string
#password:string
+viewCompanies():void
+deleteCompany(id:int):void
1
contains
1..*
Questions
#id:int
#exam_id:int
+addQuestion(exam_id:int):void
+deleteQuestion(id:int:,
exam_id:int):void

3.2 Sequence Diagram

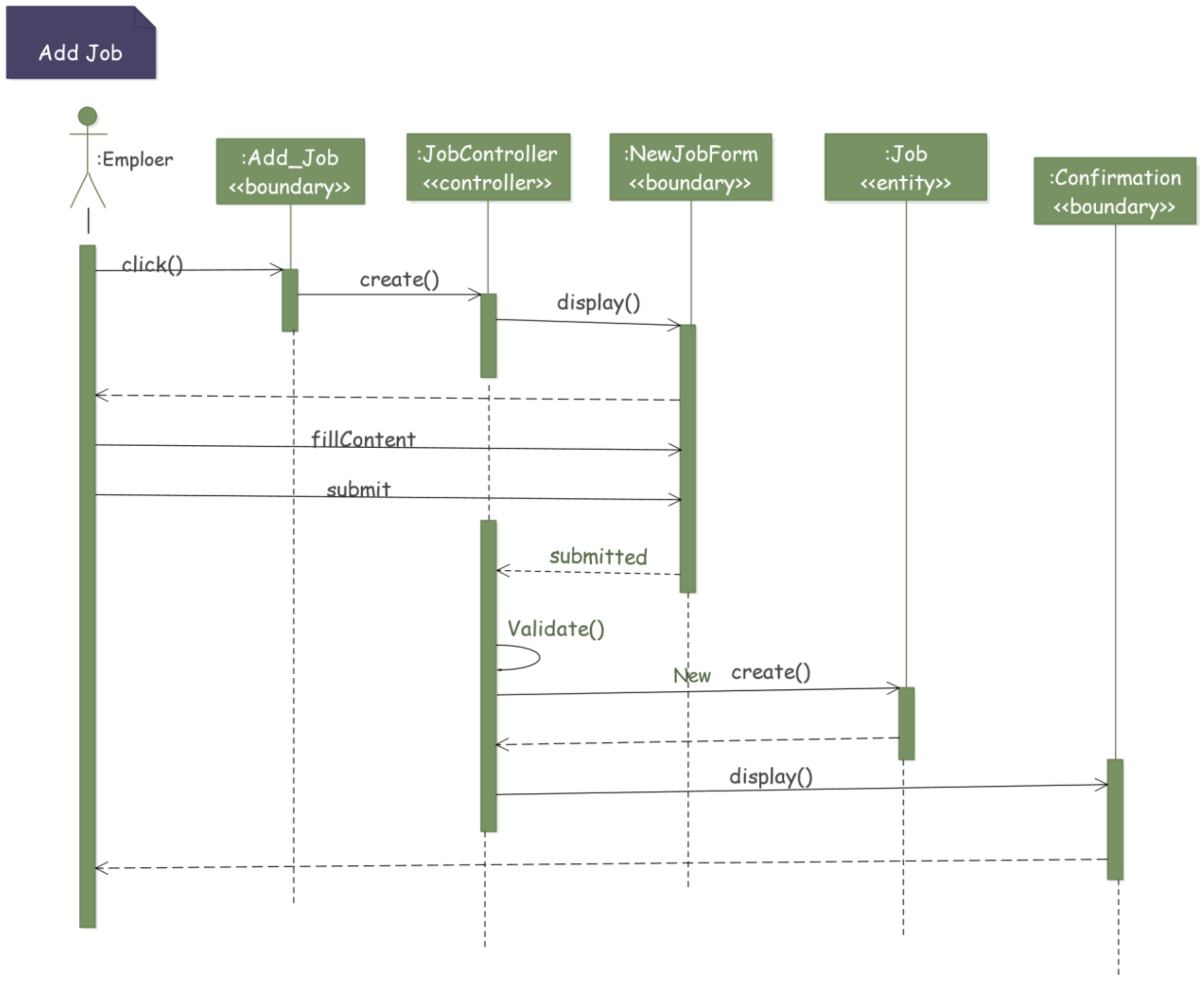


Fig 3.2. UML Sequence Diagram for Posting New Job

- This Sequence diagram is an implementation for the use case 04 named “Send job Advertisement” in our SRS Document.

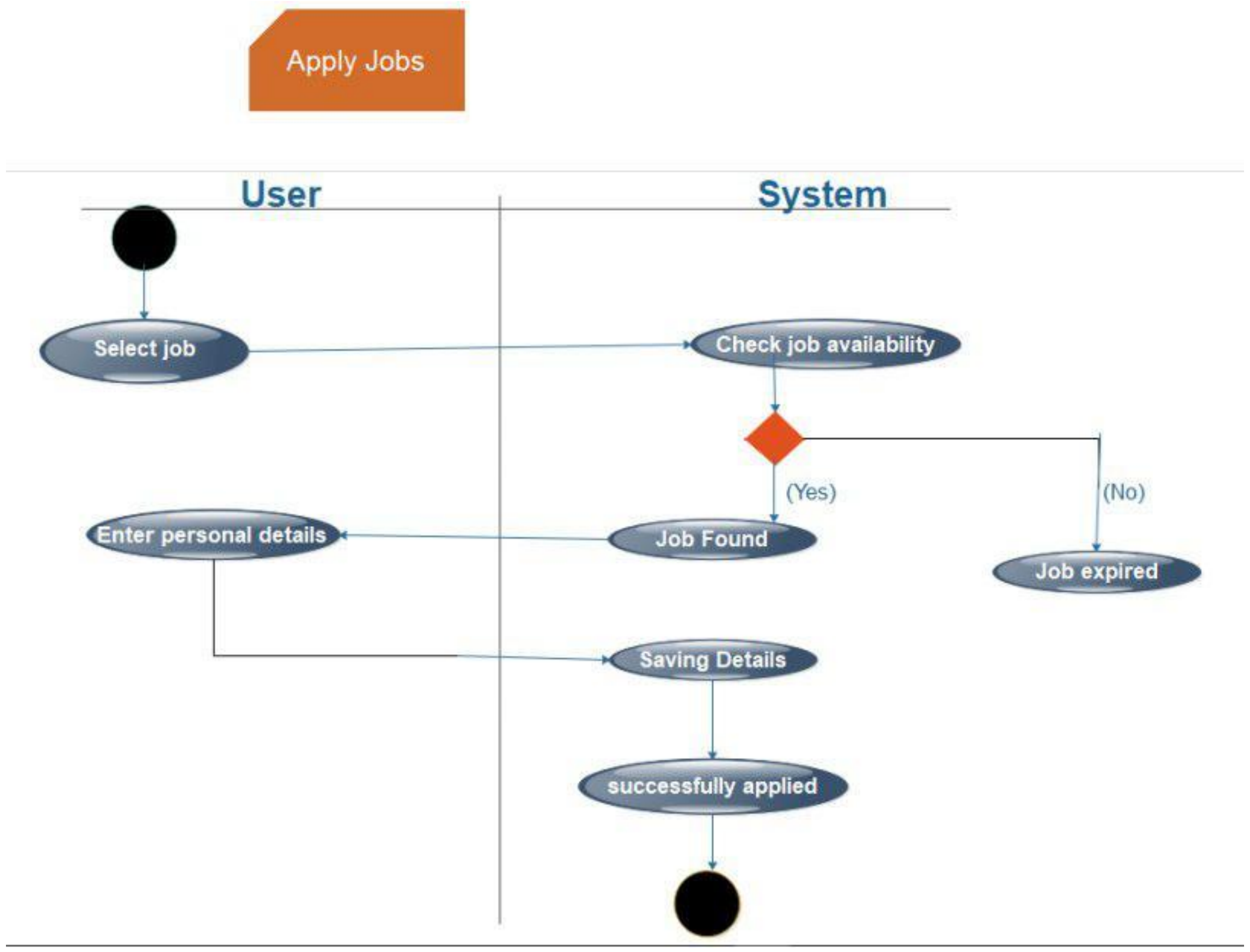


Fig 3.3 UML Sequence Diagram for apply jobs

3. Detailed Design

Table 1: Applicants class

Applicants
#id : int

#name: String #email: email #password: password

+getName();
+setPassword();
+getEmailAddress();
+getId();
+has Applied();
+confirmApply();

Table 2: Applicants class attributes

Attribute	Type	Visibility	Invariants
Id	Integer	Protected	ID<>NULL and must contain only integers
Name	String	Protected	Name<>NULL and must contain first and last name and shouldn't contain special characters and integers.
Email	String	Protected	Email<>NULL ✓ Must contain exactly one @ symbol. ✓ Must contain dot(.) ✓ The length of the string before @ must be at least 1 character long. And alphanumeric characters along with dot(.) and underscore(_) are allowed. ✓ The length of the string

			between @ and dot (.) must be at least 3 characters long. Only alphanumeric.
Password	String	protected	Password <>NULL, it must be greater than 4 digits and it can contain special characters, integers and characters.

Table 3: Operation description for Applicants class attributes

Operation	Visibility	Return type	Argument	Pre-condition	Post-condition
getName()	Public	String	No	The applicant object should be in database.	The student name should be retrieved.
setPassword()	Public	Void	Password	The applicant should be registered	Password is changed
getEmailAddress()	Public	Void	No	The applicant email should be in database	The applicant's email is retrieved
getId()	Public	Integer	No	The applicant should be registered	
hasApplied()	Public	Boolean	ID	.	Check if the applicant has applied

confirmApply()	Public	Void	No	NO	Applicants application is saved in the database

Table 4: Company class

Company
#id : int #name: String #email: email #password: password

+getName();
+setPassword();
+getId();
+postJob();

Table 5: company class attributes

Attribute	Type	Visibility	Invariants
Id	Integer	Protected	ID<>NULL and must contain only integers
Name	String	Protected	Name<>NULL and must contain first and last name and shouldn't contain special characters and integers.
Email	String	Protected	Email<>NULL ✓ Must contain exactly one @ symbol. ✓ Must contain dot(.) ✓ The length of the string before @ must be at least 1 character long. And alphanumeric characters along with dot(.) and underscore(_) are allowed. ✓ The length of the string between @ and dot (.) must be at least 3 characters long. Only alphanumeric.
Password	String	Protected	Password <>NULL, it must be greater than 4 digits and it can contain special characters, integers and characters.

Table 6: Operation description for company class attributes

Operation	Visibility	Return type	Argument	Pre-condition	Post-condition
getName()	Public	String	No	The company object should be in database.	The student name should be retrieved
postJob()	Public	Void	No	.The company must be logged in	Job is posted on the website
isRegistered()	Public	Boolean	No	No	Check if the company is registered
confirmRegistered()	Public	Void	No	No	Company information is saved in the database

Table 7: Job class

JOB
#id : int #company_id:int #type: String

+getType();
+deleteJob();
+addJob();

Table 8: Job class Attributes

Attribute	Type	Visibility	Invariants
Id	Integer	Protected	ID<>NULL and must contain only integers
Type	String	Protected	**Type**<>NULL and must contain name and description and shouldn't contain special characters.
Company_id	Integer	Protected	ID<>NULL and must contain only integers

Table 9: Operation Description for Job class attributes

Operation	Visibility	Return type	Argument	Pre-condition	Post-condition
getType()	Public	String	No	The type should be available in the database	Job is posted on the website
setType ()	public	Void	String	No	The type of the Job is changed

Table 10: Notification class

Notification
#id : int
#applicant_id:int #job_id: int #title:String #description:String

+getType();
+getDescription();
+deleteNotification();
+addNotification();

Table 11: Notification class Attributes

Attribute	Type	Visibility	Invariants
Id	Integer	Protected	ID<>NULL and must contain only integers
Title	String	Protected	**TITLE**<>NULL and must contain characters.
Description	Integer	Protected	Description<>NULL and must contain

Table 12: Operation Description for Notification class

Operation	Visibility	Return type	Argument	Pre-condition	Post-condition
getTitle()	Public	String	Id	No	The notification
					title should be retrieved
getDescription()	Public	String	No	No	Notification description is retrieved
Notify()	public	Void	No	No	The users are notified
deleteNotification()	Public	Void	ID	Id of the notification must exist in the database	The notification is deleted

References

Bibliography

Ion Somerville Software Engineering edition 9 **Web resource**

http://www.creately.com/blog/diagrams/sequence-diagram-tutorial/

http://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm