Linnæus University
School of Computer Science, Physics and Mathematics

Degree Project

Thesis Blog on Rails

Khan, Junaid Ud Din

Date: 2010-03-03

Level: Bachelor Thesis

Course Code: DA3003

Abstract

Communication handicap has been observed between student and supervisor during research work owing to various reasons; unavailability of any party on desired time, distance etc. Web base platform could be a solution to these communication problems in that, students can get guidance and feedback from their supervisors in easy, time-effective, cost-effective way. Students could upload his/ her problems, questions on the forum and supervisors would read from that platform and give suggestions, advices and recommendations to students. In this way, supervisors can check students' progress at any given time slice and students would also get timely feedback. My project is based on Ruby on Rails (RoR), it is a web-base platform, students and supervisor can communicate by this platform without face to face meeting.

Key words: online thesis blog, RoR, Rails, Ruby on rails

Table of contents

1. Introduction

This section will describe the purpose of the thesis followed by the problem which will be solved by giving a prototype for a simple web portal. Later we discuss what approach we will adopt to achieve desired task.

1.1 Purpose

Purpose of this thesis is to provide a platform to students and supervisors to overcome the handicap between them in form of a web base portal, with which the obstacles on the communication will reduce. By using this platform student can upload his (or her) problems, questions and so on to the supervisor. The supervisor will read that from the platform, encounters the problems, replies and give suggestions, advices and recommendations back to the students by this platform. In a convenient way student can get most suitable and appropriate response from supervisor.

Focus while developing this blog was to make it is easy and user friendly with all essential attributes.

To accomplish the task rails application is used as web application.

1.2 Problem

When students work with a research project (Thesis), they need guidelines from the supervisor and they have to interact with supervisor couple of times during thesis project. In such scenarios, student will contact the supervisor more frequently than before; the problem is that supervisor has many other mandatory works to do. He (or she) might not be there round the clock for supervision.

Such situation causes a communication handicap between students and supervisor by which will result in obstacles on student’s task achieving.

1.3 Approach Problem fixing

When we talk about any problem, first we believe that some solution exist for the problem we just have to find out. To come up with solution we pass through some steps.

1.3.1 Problem perspective

We just focus the problem, define the specific problem area. Communication handicap between supervisor and student is problem area.

1.3.2 Possible solutions

After defining problem look at the possible solutions for the problem. Online blog seem most appropriate solution to eliminate the handicap.

We may have some other tools to deal with over stated problem like Messengers, MSN space etc. but all readymade software has its own function and features. We need a tailor made environment which address only features relevant to desire task. Which will help users to be more relevant and to the point while using this blog.

CVS, SVN. Wiki are for developing software and specially that kind of software which are under development by different developers at different locations. In our scenario thesis might not be developing software. Mail, MSN space or blogs all have their own add on features, not customized for specific task (our scenario). There is no facility to check student gradual progress in calendar in blogs or in mail.

1.3.3 Selection of technology

After selecting problem’s solution, selection of technology or technique is next milestone. Ruby on Rails (RoR) has been chose as tool to design solution blog of defined problem. The strengths of RoR will elaborate in chapter 4.

1.4 Structure

This report is mainly divided into two sections one is theoretical portion in which we discuss about the problem addressed in report, purpose of the report and approach which will be used to fix the problem. As the solution is a web base application next will describe about the history of web applications and architecture of web base application to make better understanding for user with web base application and architecture. It continued with introduction of Ruby (Language in which development will be made) along with its architecture and simple syntax. Later discussion about Ruby on Rails (RoR) application and brief discussion about its structure and architecture, database it support, security concerns. Last part of theoretical section talk about why blog is better than other option of communication.

Second section will explain what real layout this blog will have. It will explain the user interface and features of blog. All interfaces will be explained with help of use case.

Report will end with discussion on report and some future work in it for its better performance.

2. Web Application
In this chapter we will discuss about web application history, web application definition and web application architecture for better understanding with web applications.

2.1 History
Previous method of client-server computing was such that every application worked as a user interface having its own client program and need to be installed on individual computer. HTML is mostly supported by all browsers and web pages are displayed as static documents. Even at static pages user can navigate through the contents. Technological advancement has changed the definition of web applications which are not static anymore.

Now a chain of interactive documents is generated dynamically, supported by common browsers. Scripting languages (JavaScript/VBScript) make user interface dynamic at client side. Web form elements embedded in the page markup return user input. Now web browser acts as universal client; interprets and display pages during the session.

For understanding with rails we should have some introduction with web application as rails is a web application

2.2 Definition
We can generally define web base application as following

I) Web application or webapp is an application that is accessed via Web browser over a network such as the Internet or an intranet.

II) Web applications are applications accessed by users through various means, but the primary method is the web browser.

2.3 Web Application Architecture
Web application architecture has three tiers as shown in fig 2.1

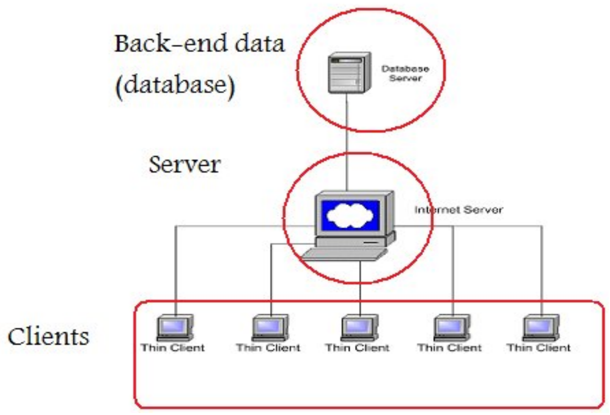


Fig 2.1: web application architecture

2.3.1 Client
Web application's interface is called client and mostly developed in HTML. Static pages of web application are developed in HTML. Many basic tasks are performed at client end such as text and output formatting, tables and forms building, setting links, image insertion.

CSS (Cascading Style Sheets) are embedded in HTML and used for formatting interface. In CSS pre defined formatting schemes are called for applying certain format.

At client side validation, navigation and calculation within form are achieved by using JavaScript/VBScript.

2.3.2 Server

Server performs essential role to integrate database, application and web services. Active Server Pages/Java Server Pages (ASP/JSP) are tools to develop server side scripting.

Web base client server environment is different from conventional client server. Establishing connection and retrieving data is easier and simple in web base client server. At time of establishing connection server assigns a session ID to client in hidden file and some information is saved as cookies at client. Client requests for data to server and disconnect the connection after receiving and storing the requested data. Client requests the server if it needs further data or information.

2.3.3 Back-end

Back end is actually a database which is collection of data/information physically stored in computer. Storing data in computer is based on a model. Relational Database Management System (RDBMS) is most commonly used model for storing data in database.

3. Ruby

Ruby is the interpreted scripting language for quick and easy object-oriented programming. It is free, not only free of charge, but also free to use, copy, modify, and distribute it.

It has many features

- Ruby has simple syntax
  Creating database is as simplest in rails never before. Rails works with MySQL.
  > ruby script/generate model student (Database Name)
  running this command will create database student with table students.
- It has exception handling features to make it easy to handle errors.
  Rescue clause intercepts the exception
  ```
  begin
          student=Students.find(params[:id])
  rescue ActiveRecord::RecordNotFound
          logger.error("Attempt to login invalid user #{params[:id]"})
          flash[:notice] = "Invalid User"
          redirect_to=>:login
  end
  ```
  If Student table does not contain given id, it will generate exception, rescue clause will intercept it and write in log the message Attempt to login invalid user and ***#{params[:id]"}*** will write user id written by user.
  flash[:notice] = "Invalid User"
  it will display message "Invalid User" and ***redirect_to=>:login*** will redirect page back to login page.
- Operators are syntax sugar for the methods. You can redefine them easily.
- It is a complete, full, pure object oriented language.
  Ruby follows strict concepts of Object Oriented programming languages (OOPS) and thus has objects, methods and classes available for programming. It has all the ways by which we can handle these objects and classes in a smooth manner.
- Its features blocks in its syntax (code surrounded by '{' ... '}' or 'do' ... 'end'). These blocks can be passed to methods, or converted into closures.
  3.times {puts "simplest form of closures"}
  Output will be
  ```
  simplest form of closures
  simplest form of closures
  simplest form of closures
  ```
  passing arguments to a closure
  students = ['john', 'peter', 'khan'].collect {|item| item.upcase}
  puts students.join(" or ") + "one of all"
  output
  ```
  JOHN or PETER or KHAN one of all
  ```
- A true mark-and-sweep garbage collector. It works with all Ruby objects. You do not have to care about maintaining reference counts in extension libraries.
- Needs no variable declarations. It uses simple naming conventions to denote the scope of variables.
  @time simply declare a variable time
  In controller just assign value to variable ***@time=Time.now,*** in .rthml file displays the variable value ***<%= @time %>***
- Can load extension libraries dynamically if an OS allows.

- OS independent threading. Thus, for all platforms on which Ruby runs, you also have multithreading, regardless of if the OS supports it or not, even on MS-DOS.
- It is highly portable: it is developed mostly on Linux, but works on many types of UNIX, DOS, Windows 95/98/Me/NT/2000/XP/Vista, MacOS, BeOS, OS/2, etc.

4. Ruby on Rails (Rails)

Ruby on Rails is an open source web application framework which is written in Ruby and also called as RoR or Rails. Ruby architecture follows Model-View-Controller (MVC) architecture.

It is heading over its simplicity, providing less code and minimum configuration platform for developing real world applications.

Initially Rails was distributed through Ruby Gems. Ruby Gems are distribution channel for Ruby libraries and application.

Framework

Web applications are easier to create with help of reusable structured program called web framework. These reusable programs deal with tasks required in writing a web application. For example

- Validating form data
- Displaying error messages
- Saving data to the database
- Handling page flow and navigation etc.

5. Ruby on Rails MVC Architecture

Architecture of Ruby on Rails as follows in fig 5.1

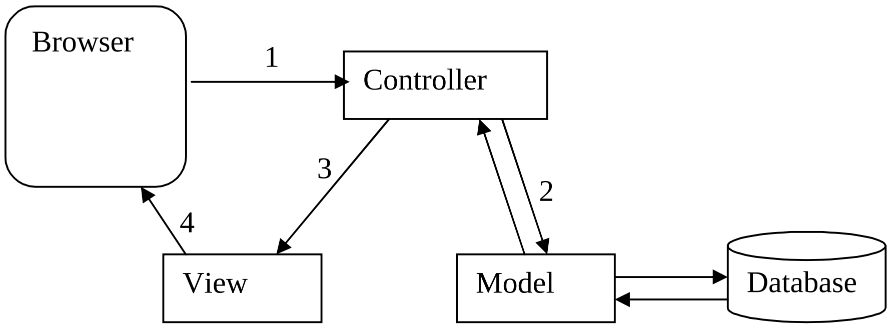


Fig 5.1 Architecture of Ruby on Rails

The architecture of Ruby on Rails (RoR) as shown in fig 5.1 is explained bellow.

5.1 Controller

Controller classes replies user interaction in MVC architect. Application is also called by controller classes which later control the data and data is forwarded to model. View displays the data. Controller methods are mostly initiated by web browser.

5.2 Model

Classes representing RDBMS tables make data bases that drive MVC web application. Model classes in RoR are handled by Active records. Thus it can be said that there must be a related class in the application for each table existing in database. Functions required to find, delete, create and update rows in the table are present in the relevant class.

Usually programmer should have the Active Records::Base class as subclass, then selection of particular RDBMS table and columns existing in the table are handled by the program automatically.

Class definitions define the Object-relational mapping commands relating the classes. For instance, there is a class named Image with the definition “has-many:comments and we an object of class image called ‘a’, application of a.commands will return an array with all Comments objects with the image_id equal to a.id.”

Model class also specify and implement data validation handlers (e.g. validates_uniqueness_of :checksum) and any update-related handlers (e.g. after_destroy :remove_file, before_update :update_related_details).

5.3 View

Display logic or data from the controller classes are displayed in View. It is embedded in HTML and consists of minimal amount of code.

View can be handled in many ways. The underlying view code is part of the Action Pack. Similar to JSP, Rails use embedded Ruby (.rhtml) files, which are combination of HTML and Ruby code in parallel.HTML and XML can also be constructed programmatically with Builder, Liquid template system, Markaby, or Haml.

HTML code should be written to display user output of each method in the controller. Controller action which displays layouts and fragments is described separately from page layout.

I) Web Server Support
The lightweight WEBrick web server included with Ruby is often used as the application server, for development. Apache, Cherokee, Nginx or Lighttpd with FastCGI is recommended for commercial use, but any web server with CGI or FastCGI support will work.

II) Database Support
For data storage an RDBMS system is recommended as the Rails architecture strongly favor database use. If RDBMS is not possible Rails also support SQLite library. Programmers define the way to access the database as rails handles access to all database automatically. Direct SQL queries are also possible, if necessary. Over different database system Rails is maintaining database-neutrality, application portability and pre-existing databases usability. Framework alone could not guarantee due to different feature sets of the RDBMSes. Several RDBMS systems are supported by RoR including IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, and SQLite.

6. Security in Blog

Security has been applied on different level to make it possible that unauthorized usage of data can be avoided. Security structure is designed in focusing all layers of application.

6.1 Encryption of Password

Passwords are encrypted from simple text so that it can be saved from hacking.

```
def password
        @password
end


def password=(pwd)
        @password=pwd
        return if pwd.blank?
        create_new_salt
        self.hashed_password = User.encrypted_password(self.password, self.salt)
end
```

Fig 6.1 Code for checking password presence

In the fig 6.1 code system will check the presence the passwords. If a user not enter any password it return back and if password is given it will create new salt and pass to function for encryption.

```
def self.authenticate(name, password)
        user=self.find_by_name(name)
        if user
        expected_password = encrypted_password(password, user.salt)
                if user.hashed_password != expected_password
                        user=nil
                end
        end
end


private
def self.encrypted_password(password, salt)
        string_to_hash= password + "wibble" + salt
        Digest::SHA1.hexdigest(string_to_hash)
```

```
end


def create_new_salt

                self.salt=self.object_id.to_s + rand.to_s

end
```

Fig 6.2 Encrypting password

In fig 6.2 code encryption has been made to make password secure so that it can not be visible in database tables.

6.2 Declaring Private Methods

Non-action methods are declared as private to secure form outside triggers to execution it.

```
private

def self.encrypted_password(password, salt)

                string_to_hash= password + "wibble" + salt

                Digest::SHA1.hexdigest(string_to_hash)

end


def create_new_salt

                self.salt=self.object_id.to_s + rand.to_s

end
```

Fig 6.3 Data safety

In fig 6.3 code declaring keyword “private” make upcoming methods private and cannot be triggered outside the class. It makes data manipulation safe.

6.3 Session variables

Session variables being empty instead of overloading, so that after logoff copy, paste URL will not works to manipulate data by any other users.

```
def logout
                 session[:user_id]=nil
                 flash[:notice]="Logged Out"
                 redirect_to(:action => "index")
  end
```

Fig 6.4 Securing URL execution from copy, paste

In fig 6.4 code has made possible for each user logoff, session variable being empty as most of data manipulations based on session variable which is always userid.

6.4 Quarries are bind with variables

All quarries are bind with session or local variables, values are assigned to variables after user and password authentication.

```
def submit_new_problem
                @UserID=session[:user_id]
                @StudentID=Studentdata.find(params[:id])def delete_user
                 if request.post?
                                user=User.find(params[:userid])
```

Fig 6.5 Query execution

In fig 6.5 shows submitting new problem require user_id which is session variable and session variable could only be assigned a value if user and password was authenticated. It makes data manipulation secure.

Same strategy is applied in all quarries to make data more secure.

7. Execution

This section describes the implementation of the Thesis Blog application with a short description.

A blog is an access-control system and we have three types of users to access blog namely;

1) **Admin**
2) **Supervisor**
3) **Student**

Below the different functions offered by the system to the above mentioned user groups are explained.

We shall take one example whereby, we shall start the blog using from the scratch. Start from the login screen, then, log in with first user. The first user will be an administrator type of user. The login screen's screenshot with use case will be explained-how it works. Next screen with use case will show the functions an administrator can perform. Later, it will be explained, how the administrator could add and delete supervisor and students. How thesis is to be assigned, how student would post on blog, how supervisor would check and reply, how student would update calendar, and how supervisor would monitor student's day to day activity.

Every step is explained with the help of screenshot and use cases to let reader visualize how the blog would work and to develop a better understanding with architect of the blog on each step.

7.1 Login

Enter user id and password. It will redirect automatically to first page according to user type (Admin, supervisor, student), if user name or password error occurs it displays message and redirect to login page again. Only successive login will be redirected.

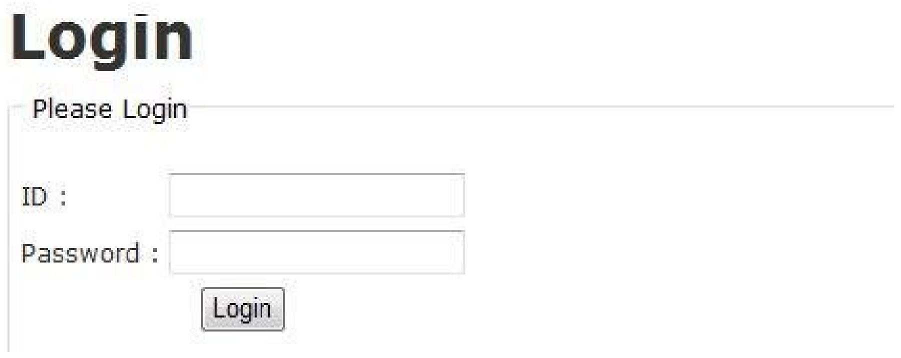


Fig 7.1 Login Screen

Login use case

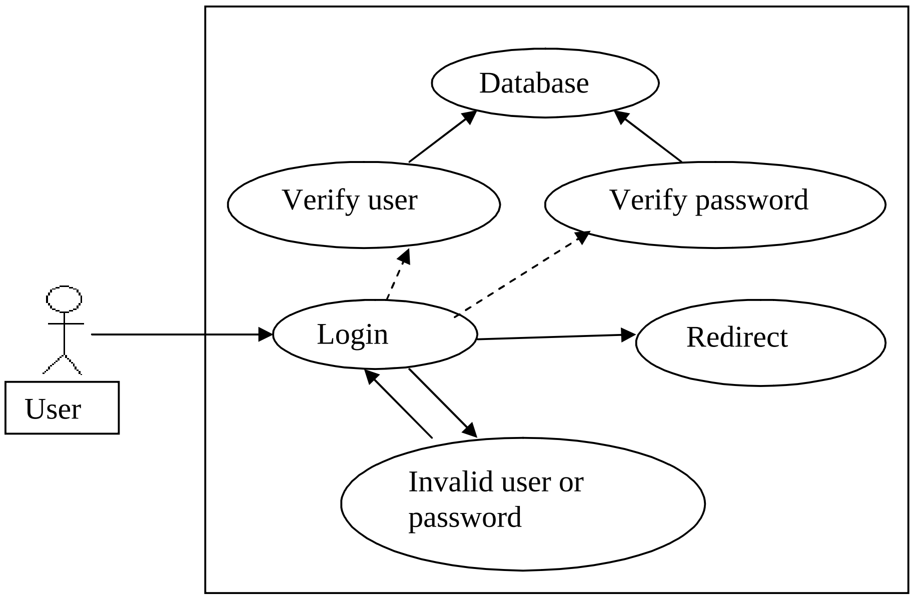
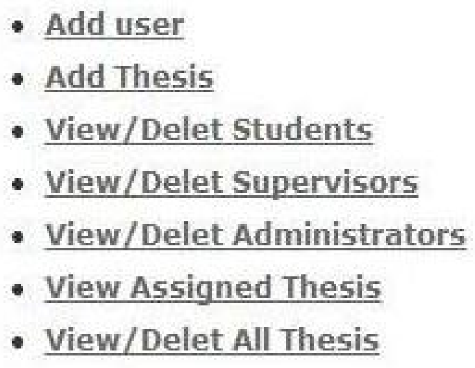


Fig 7.2 Login Use case

7.2 Admin

When user login in as a admin user group it has following rights to perform task

7.2.1 Add user
7.2.2 view/delete Student
7.2.3 Add Thesis
7.2.4 view/delete supervisor
7.2.5 view/delete admin
7.2.6 View assigned thesis
7.2.7 view/delete all theses



- Add user
- Add Thesis
- View/Delet Students
- View/Delet Supervisors
- View/Delet Administrators
- View Assigned Thesis
- View/Delet All Thesis

Fig 7.3 First Admin user screenshot

Redirect to admin Use case

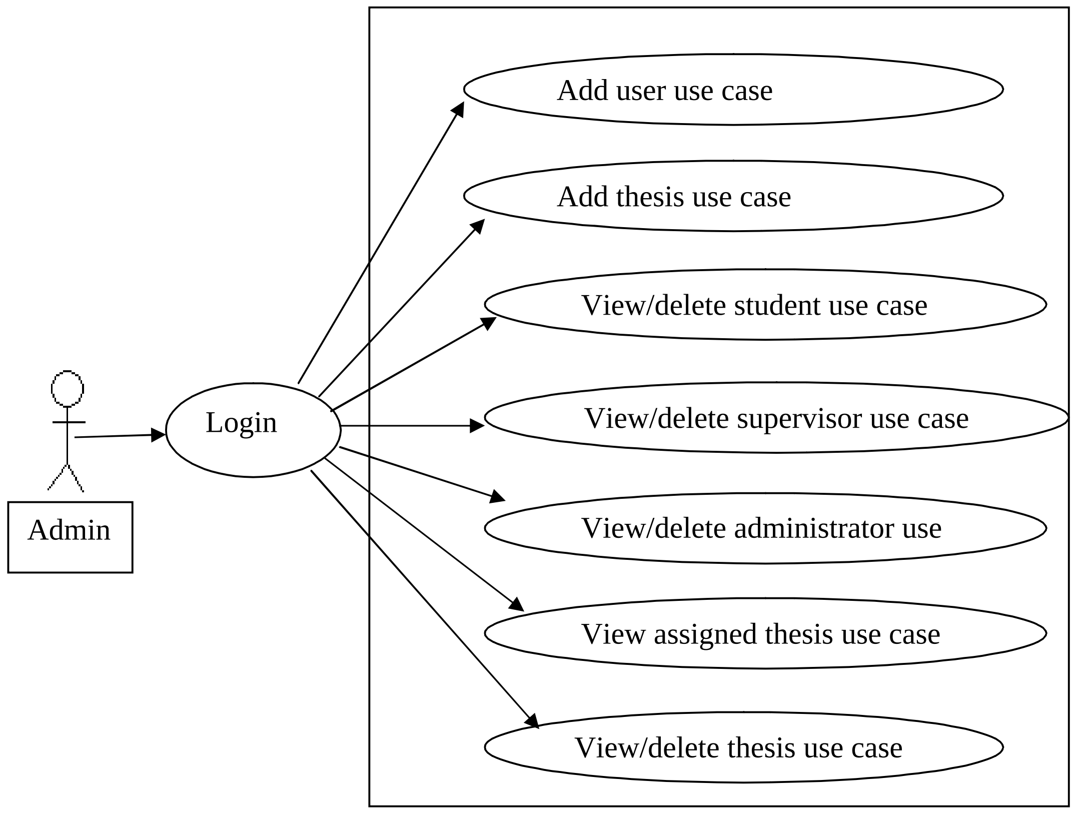
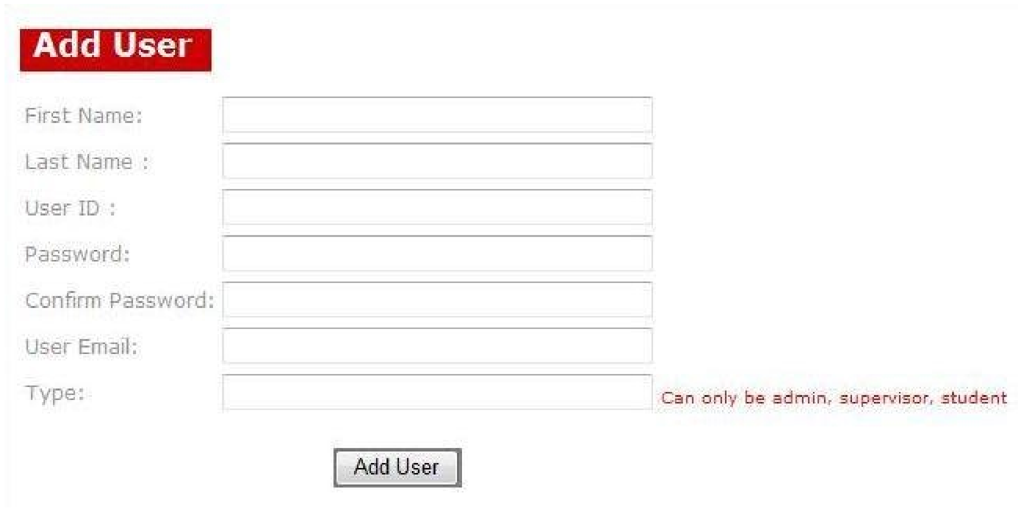

Admin
Login
Add user use case
Add thesis use case
View/delete student use case
View/delete supervisor use case
View/delete administrator use
View assigned thesis use case
View/delete thesis use case

Fig 7.4 Admin user case

7.2.1 Add user
Add basic and common information of each user. At User type there ate three options Admin, supervisor and student.

After adding common information page redirects to new page for add data of each type of user.



Add User

First Name:	
Last Name :	
User ID :	
Password:	
Confirm Password:	
User Email:	
Type:	Can only be admin, supervisor, student

Fig 7.5 Add user screenshot

Add user use case

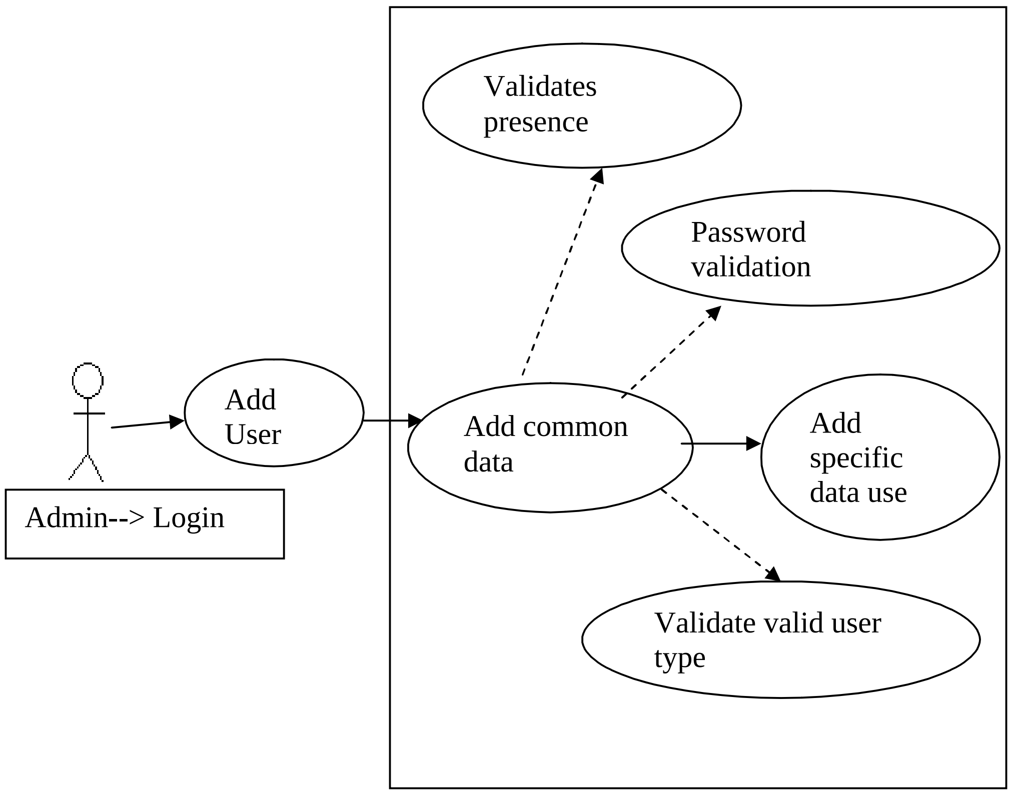
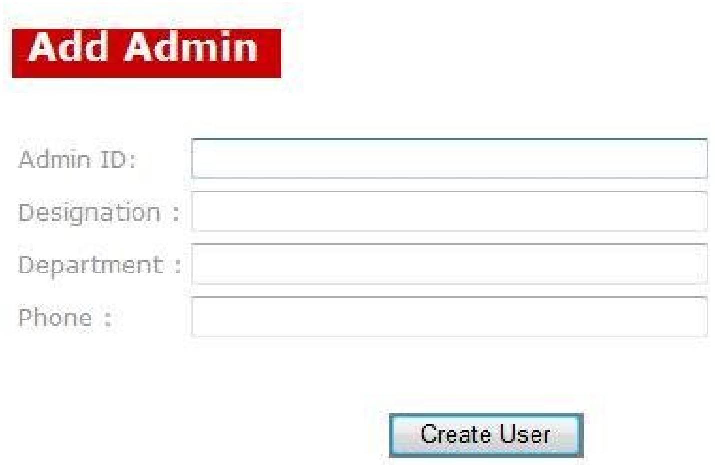


Fig 7.6 Add user use case

7.2.1.1 Add Admin

If user type was admin at time of user creation it will redirect to new page for entering specific information about administrator. Admin ID will displayed automatically and have to add designation, department and phone number.



Add Admin

Admin ID:	
Designation :	
Department :	
Phone :	

Fig 7.7 Add Admin screenshot

Add Specific data use case for admin user

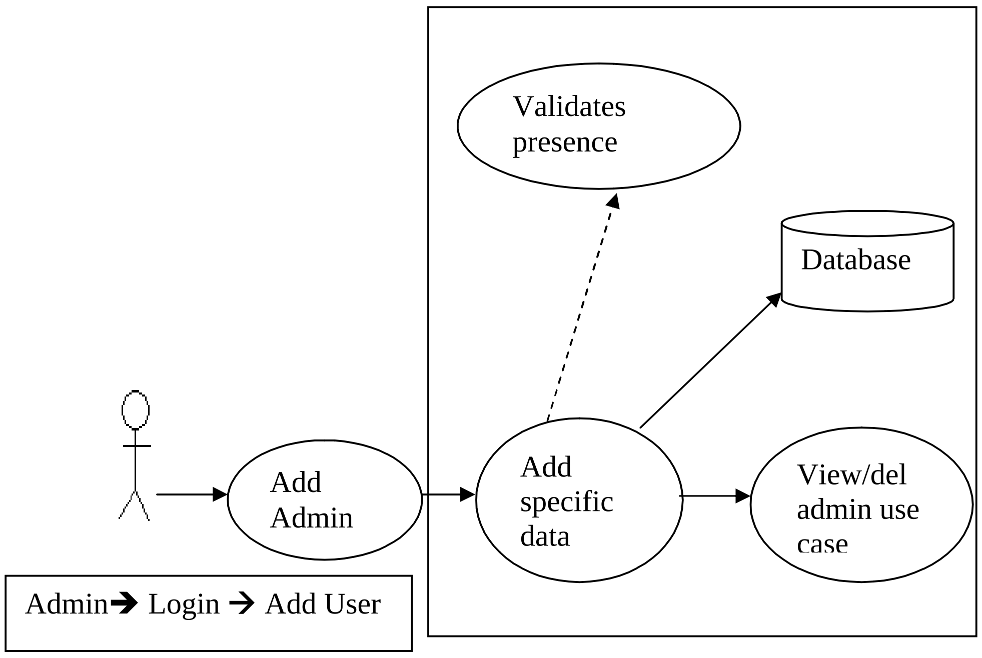


Fig 7.8 Add Admin use case

7.2.1.2 Add supervisor

If user type was supervisor at time of user creation it will redirect to new page for entering specific information about supervisor. Supervisor ID will displayed automatically and have to add designation, department and phone number.

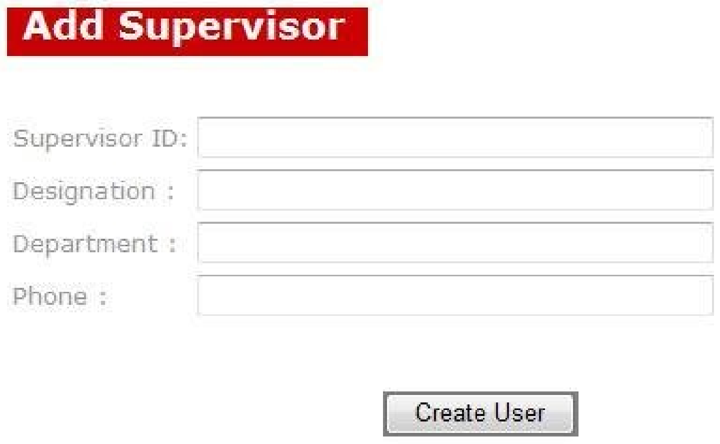


Fig 7.9 Add Supervisor screenshot

Add Specific data use case for supervisor user

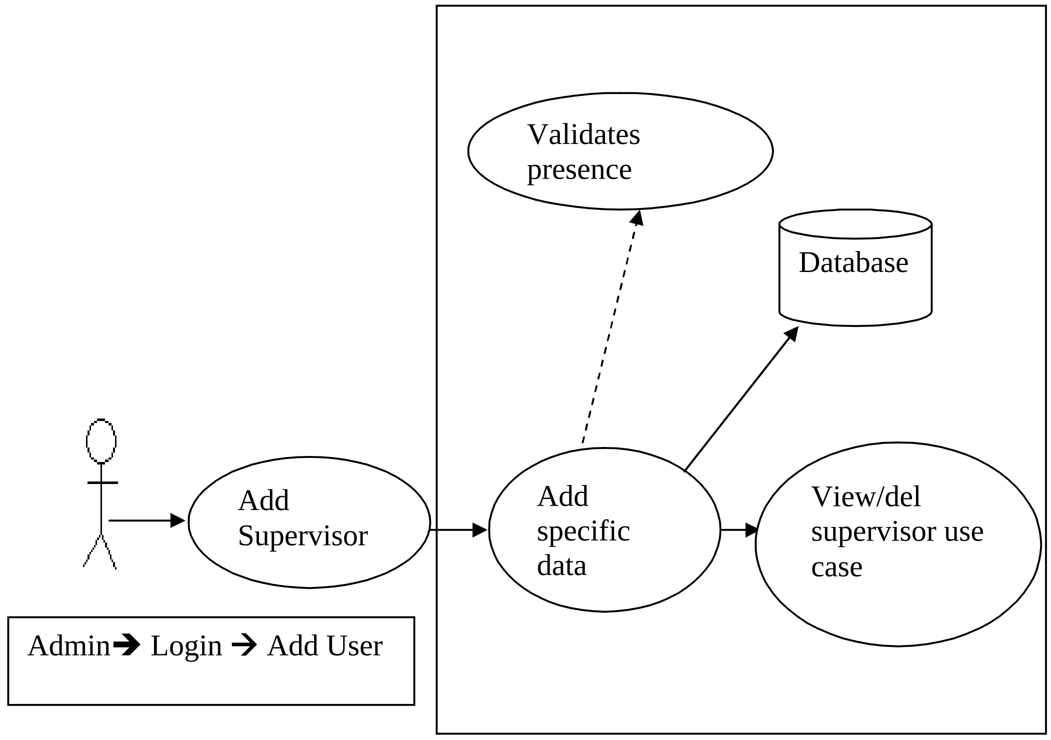


Fig 7.10 Add Supervisor use case

7.2.1.3 Add Student

If user type was student at time of user creation it will redirect to new page for entering specific information about student. Student Id will be displayed automatically and have to add supervisor ID, thesis ID, department and school in which student is studying.

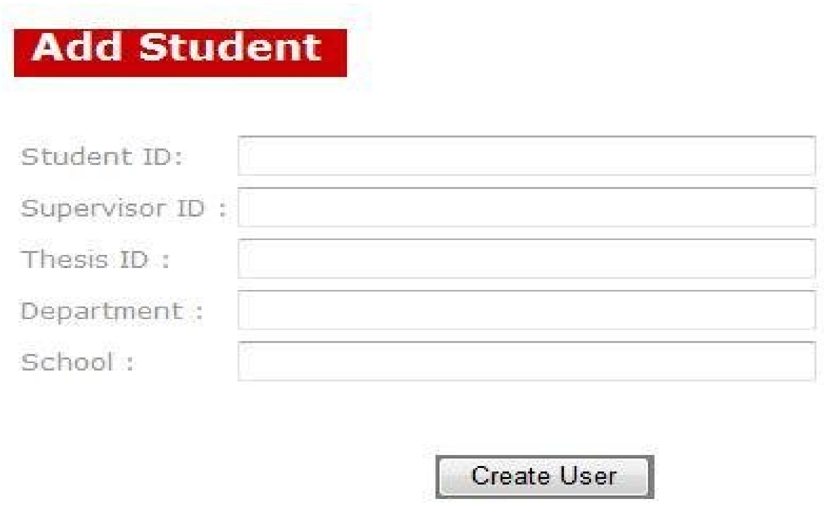


Fig 7.11 Add Student screenshot

Add Specific data use case for student user

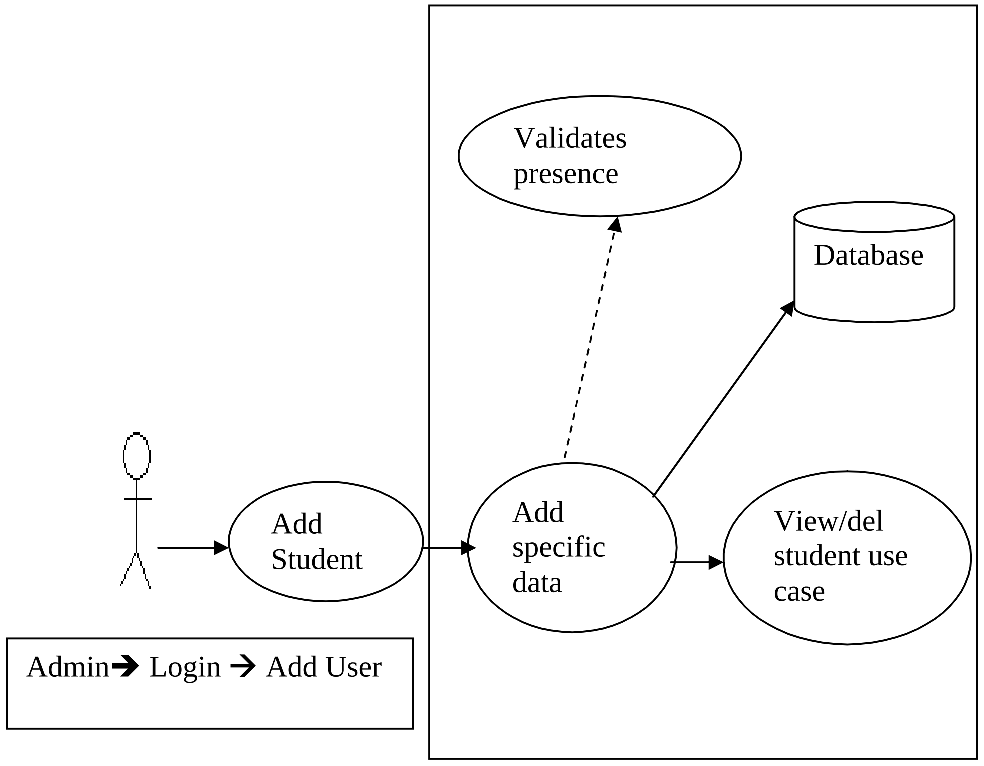
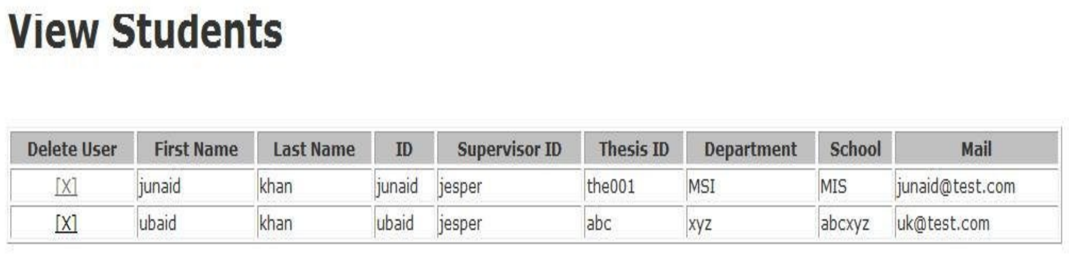


Fig 7.12 Add student use case

7.2.2 View/delete Student

If the added user belongs to student group it will show all students added previously just after adding specific data of any user. Users can be deleted by clicking [X], it will reconfirm do you really want to delete. After deleting user same page will be displayed with list of remaining users.

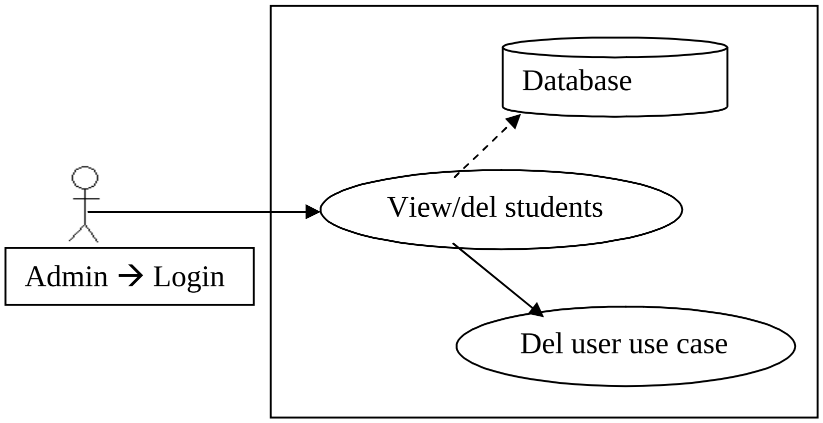


View Students

Delete User	First Name	Last Name	ID	Supervisor ID	Thesis ID	Department	School	Mail
[X]	junaid	khan	junaid	jesper	the001	MSI	MIS	junaid@test.com
[X]	ubaid	khan	ubaid	jesper	abc	xyz	abcxyz	uk@test.com

Fig 7.13 view/delete student screenshot

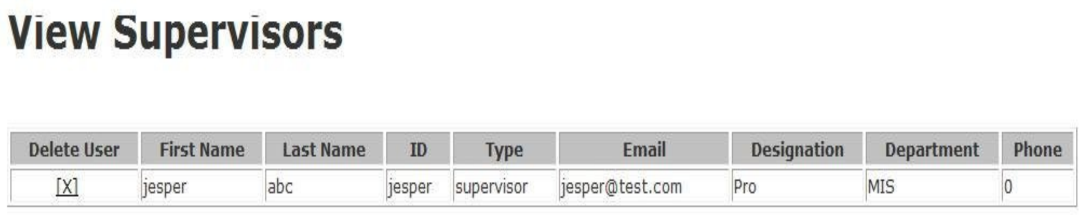
View/del student user case



7.14 view/delete student use case

7.2.3 View/delete Supervisor

If the added user belongs to supervisor group it will show all supervisor added previously. Supervisor can be deleted by clicking [X], it will reconfirm do you really want to delete. After deleting supervisor same page will be displayed with list of remaining supervisor.



View Supervisors

Delete User	First Name	Last Name	ID	Type	Email	Designation	Department	Phone
[X]	jesper	abc	jesper	supervisor	jesper@test.com	Pro	MIS	0

Fig 7.15 view/delete supervisor screenshot

View/del supervisor use case

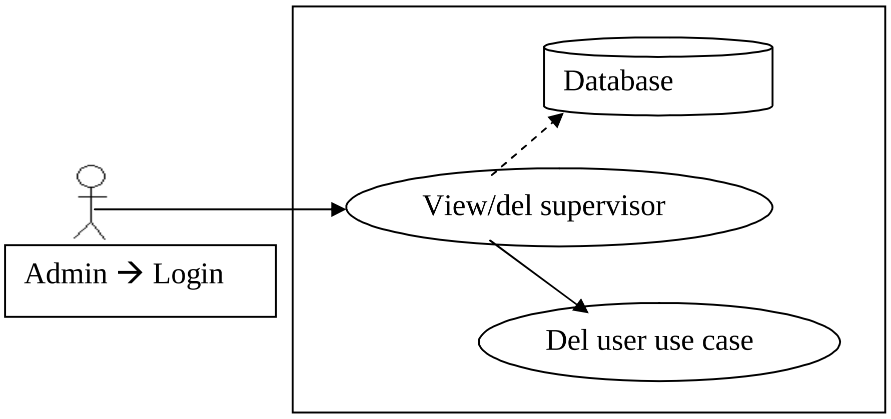
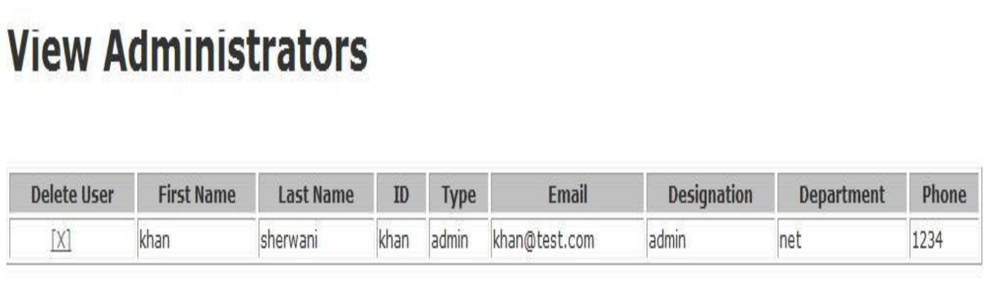


Fig 7.16 view/delete supervisor use case

7.2.4 View/delete Admin

If the added user belongs to admin group it will show all administrators added previously. Administrator can be deleted by clicking [X], it will reconfirm do you really want to delete. After deleting supervisor same page will be displayed with list of remaining supervisor.

Last admin can not be deleted. If you tried to do so it will generate error that last admin can't be deleted.



View Administrators

Delete User	First Name	Last Name	ID	Type	Email	Designation	Department	Phone
[X]	khan	sherwani	khan	admin	khan@test.com	admin	net	1234

Fig 7.17 view/delete admin screenshot

View/del admin use case

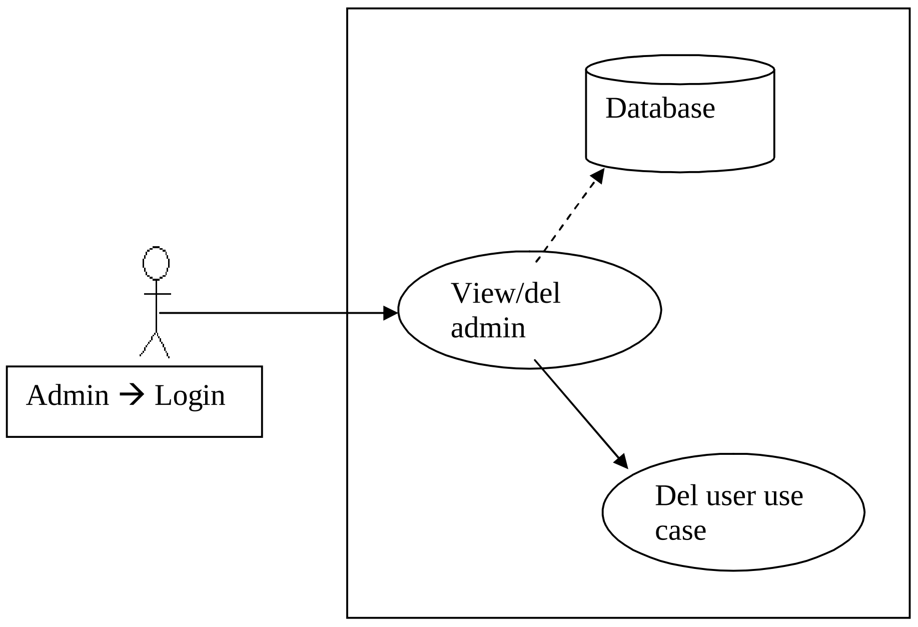


Fig 7.18 Add/delete Admin use case

7.2.5 Del user case

In all delete user case *del user use case* executes, which first check user in database than delete user from database and redirect page to view/del user page with remaining user.

Delete user use case

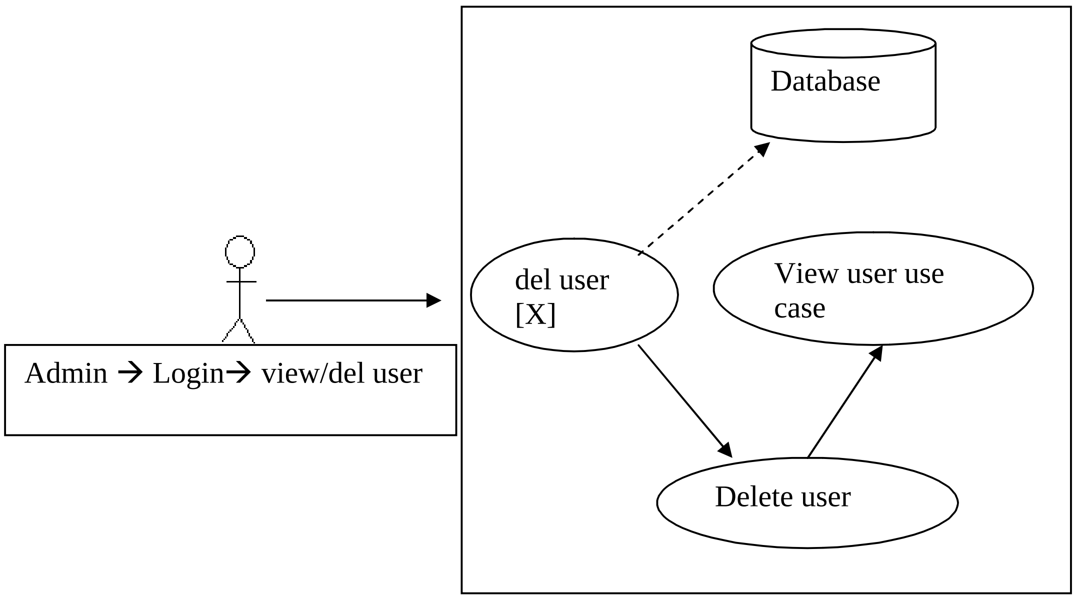
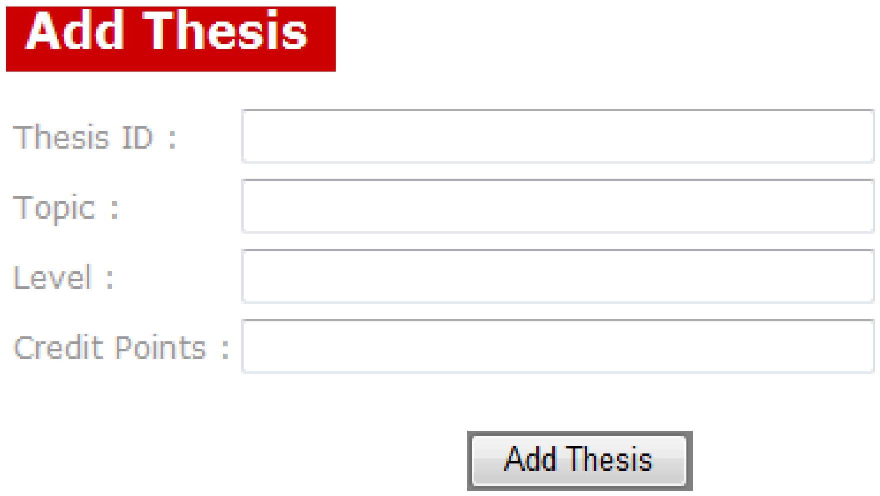


Fig 7.19 Delete user use case

In case of deleting admin, it will check if it is last user of admin than it will popup a message that it is last admin user and cannot be deleted.

7.2.6 Add thesis

Clicking on add thesis will open a page where all information about thesis is saved. Adding thesis will redirect to view thesis page after adding new thesis.



Add Thesis

Thesis ID :	
Topic :	
Level :	
Credit Points :	

Fig 7.20 Add thesis screenshot

Add thesis use case

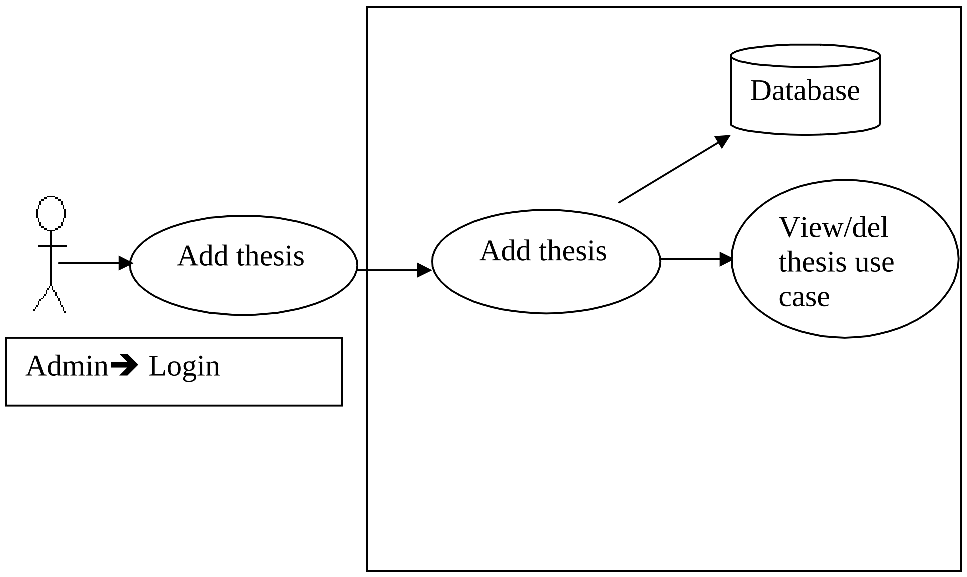
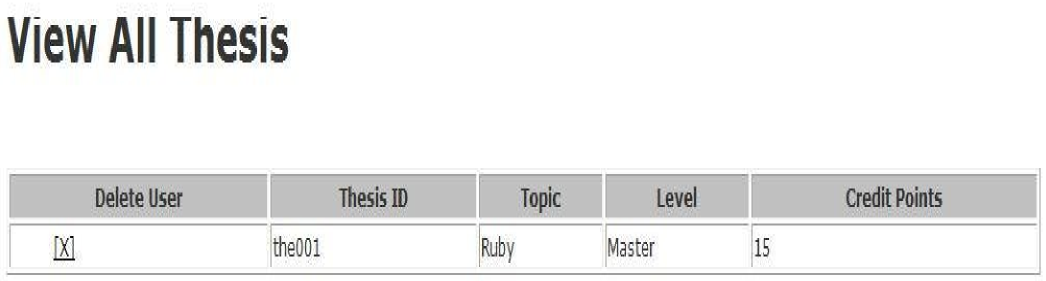


Fig 7.21 Add thesis use case

7.2.7 View/del Thesis
It will show all theses in all schools. Thesis can be deleted by clicking [X], it will reconfirm do you really want to delete. After deleting thesis same page will be displayed with list of remaining thesis.



View All Thesis

Delete User	Thesis ID	Topic	Level	Credit Points
[X]	the001	Ruby	Master	15

Fig 7.22 view/delete thesis screenshot

View/del these use case

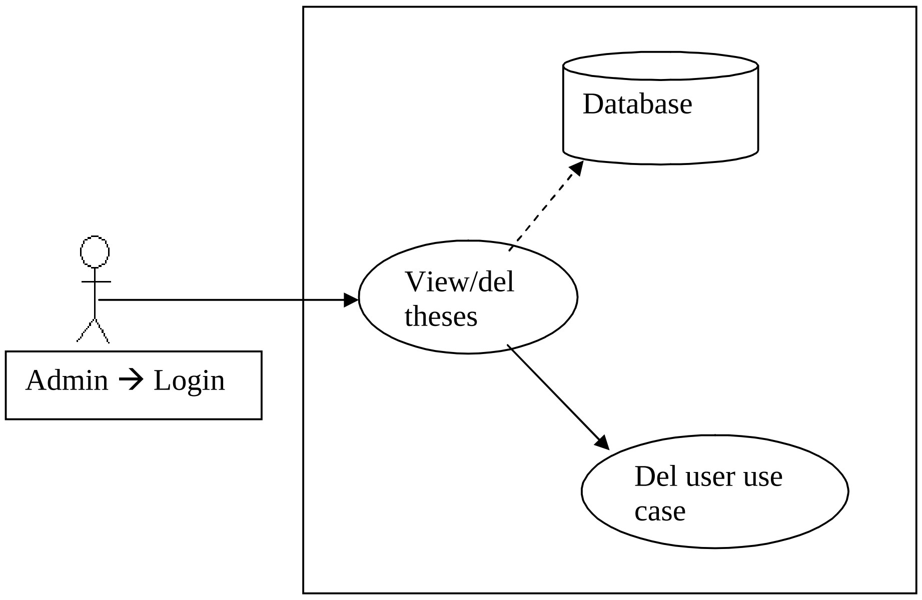
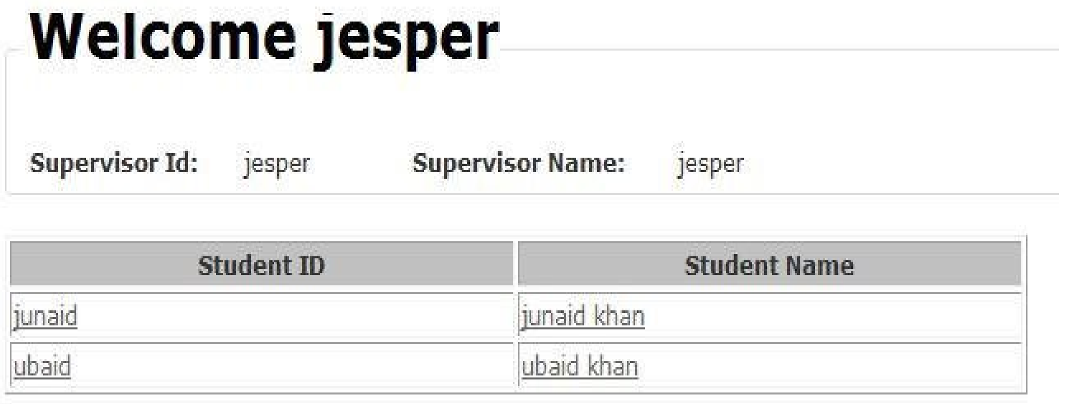


Fig 7.23 View/delete thesis use case

7.3 Supervisor

Supervisor user has to supervisor one or more students at a time. He can view all students under his supervision. He also can view all problems and submitted documents. Supervisor has right to delete any of post by students. He can view student calendar of incremental progress. Supervisor can reply any of student post and post new suggestions, guideline or any news or upcoming schedule for student.

After login in a user belongs to supervisor group will redirect to a page having list of all users enrolled in his supervision.



Welcome jesper

Supervisor Id: jesper   Supervisor Name: jesper

Student ID	Student Name
junaid	junaid khan
ubaid	ubaid khan

Fig 7.24 Supervisor user screenshot

Supervisor login use case

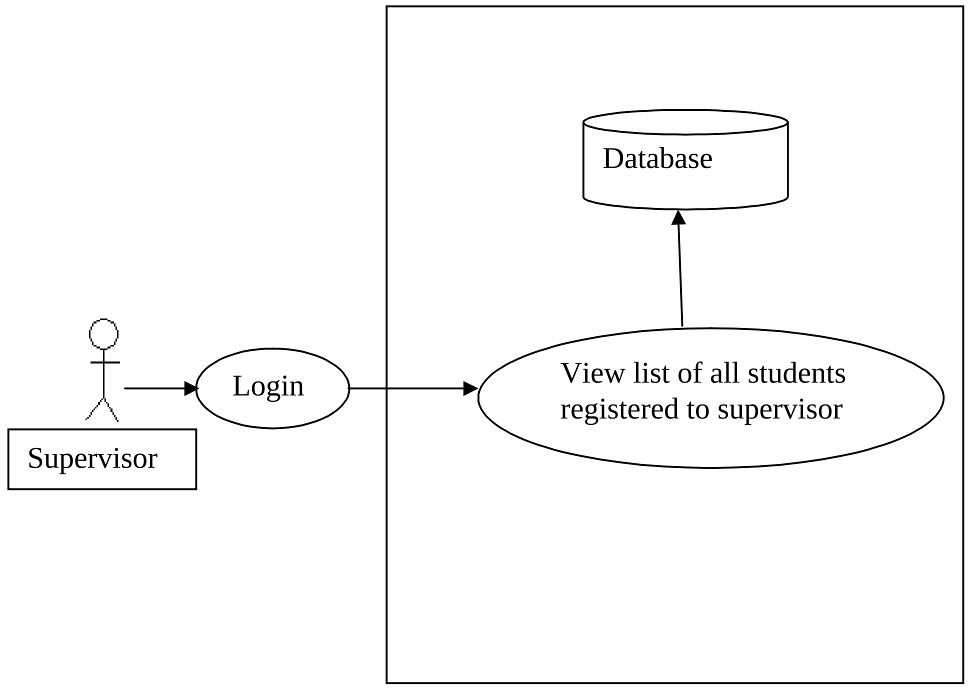
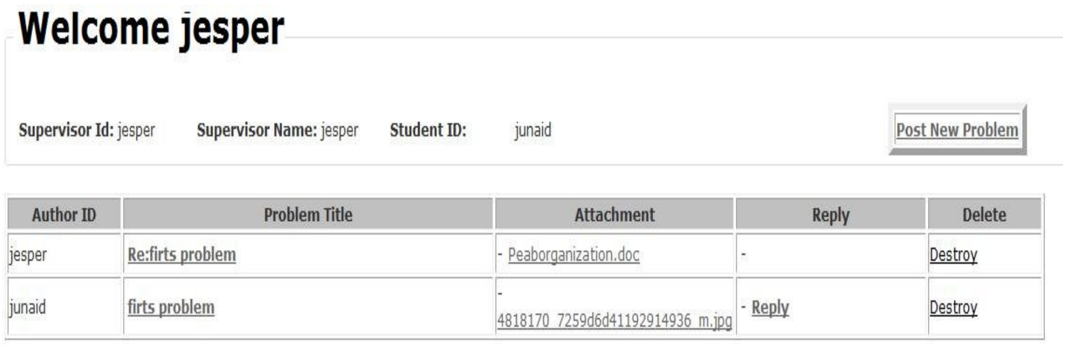


Fig 7.25 Supervisor login use case

7.3.1 Student blog

Clicking on student id or name all post from student or supervisor will be displayed in list. Supervisor can open any post by student or download attachment sent by student. Supervisor can reply post without opening the post detail or downloading attachment just by clicking on reply. Reply option is enabled on only post sent by student in supervisor login. Supervisor can delete any post either by his own or by student. New post or unread post's problem title is written in black font color and read post problem title is in gray text.



Welcome jesper

Supervisor Id: jesper **Supervisor Name:** jesper **Student ID:** junaid

Post New Problem

Author ID	Problem Title	Attachment	Reply	Delete
jesper	Re:firts problem	- Peaborganization.doc	-	Destroy
junaid	firts problem	- 4818170_7259d6d41192914936_m.jpg	- Reply	Destroy

Fig 7.26 student blog screenshot

Student blog use case

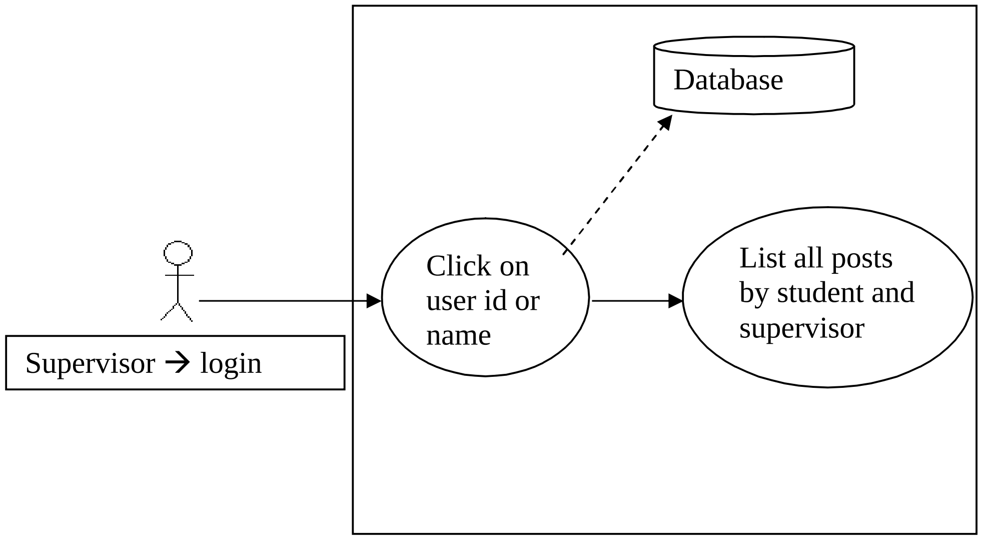
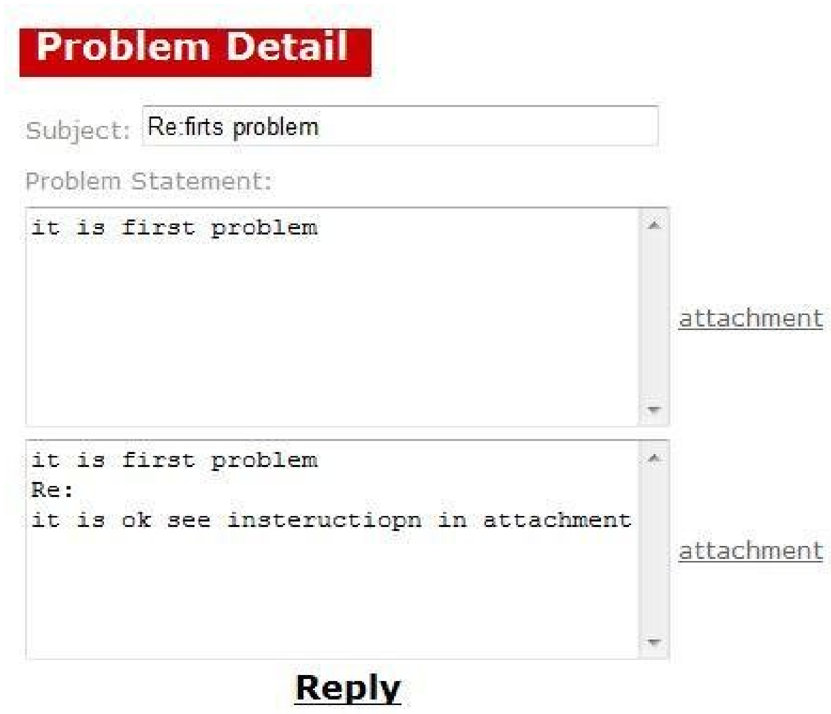


Fig 7.27 Student blog use case

7.3.2 Problem Detail

Clicking on problem title a new window will open with all history of that post, if exists, beside with attachments of each message in history. Reply option is available there in bottom if wants to reply.



Problem Detail

Subject: Re:firts problem

Problem Statement:

```
it is first problem
```

attachment

```
it is first problem
Re:
it is ok see insteructiopn in attachment
```

attachment

Reply

Fig 7.28 Problem detail screenshot

Problem detail use case

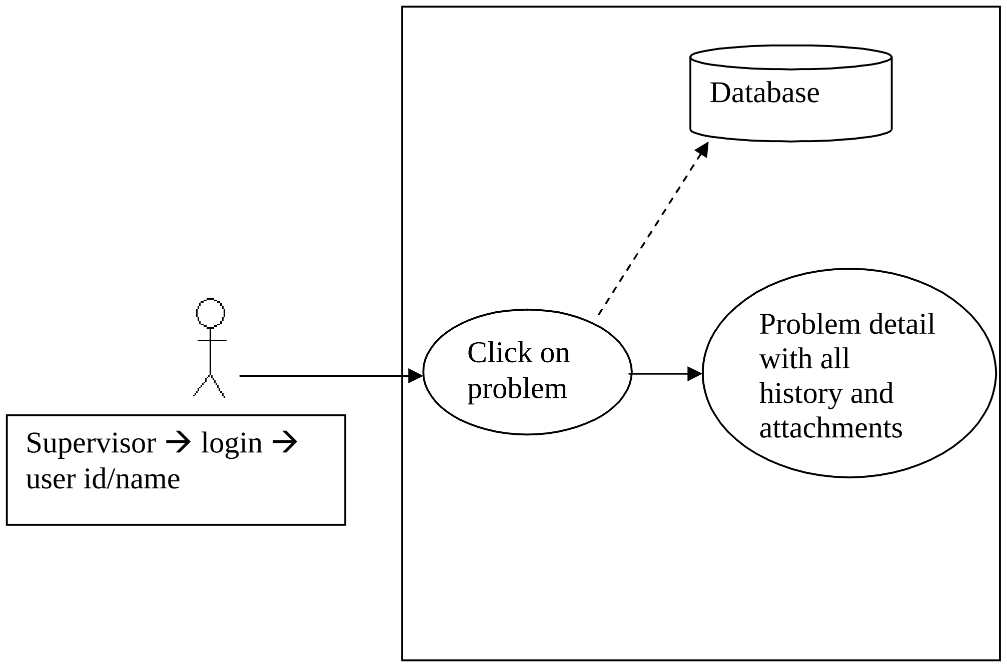
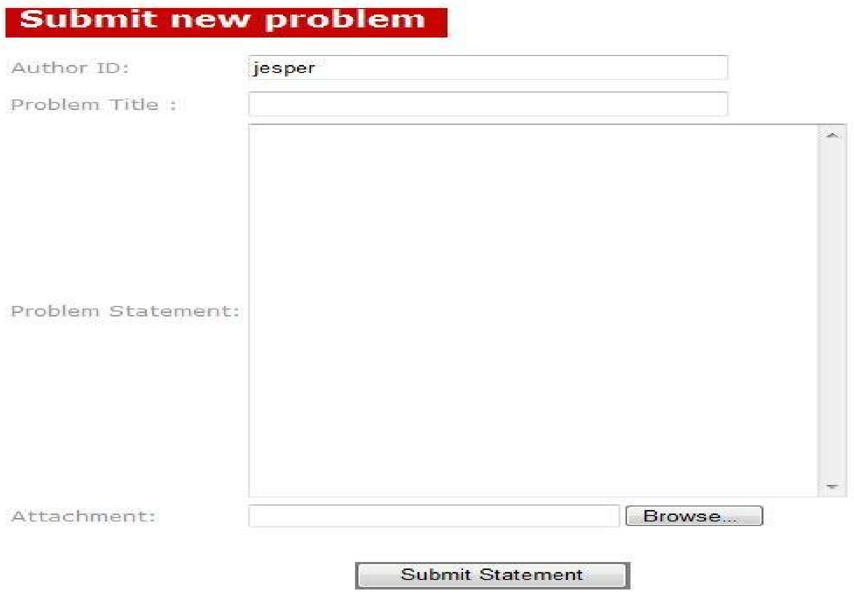

Database
Click on problem
Problem detail with all history and attachments
Supervisor → login → user id/name

Fig 7.29 Problem detail use case

7.3.3 Submit new problem

On right top of student posts detail, Post new problem is available, if supervisor wants to post any new comment, suggestion or attachment to student it can be. Author id will be written in textbox automatically.



Submit new problem

Author ID:	jesper
Problem Title :	
Problem Statement:	
Attachment:	

Fig 7.30 Submit new problem screenshot

post new problem use case

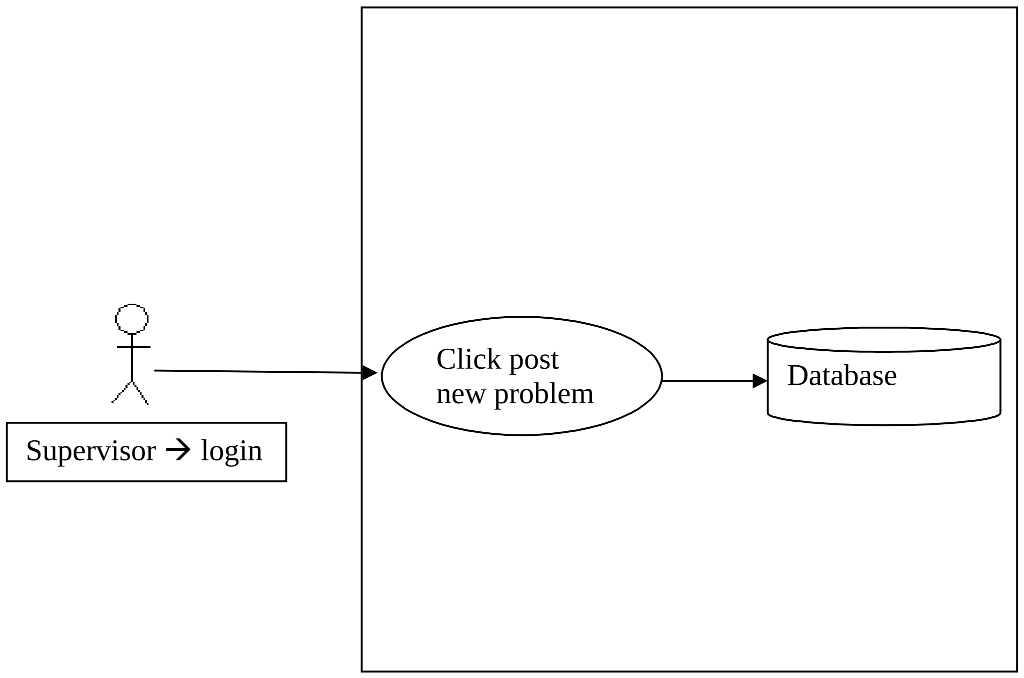
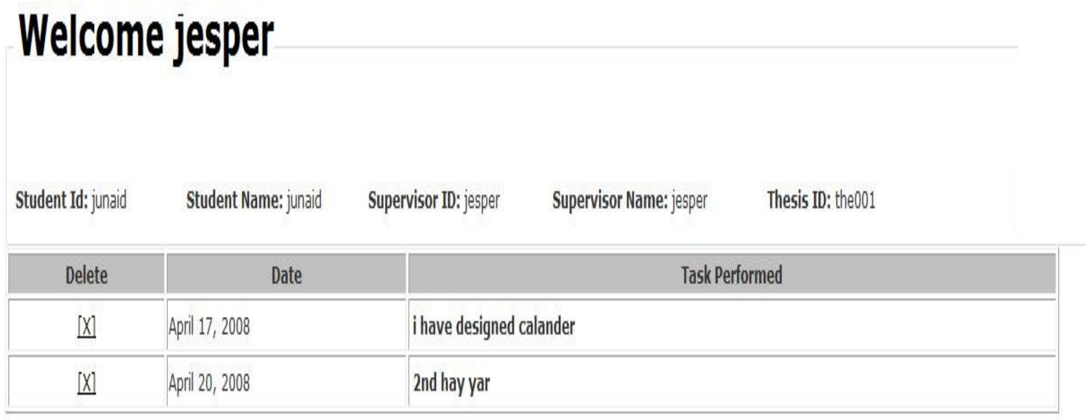


Fig 7.31 Post new problem use case

7.3.4 Calendar

Beside post new problem, calendar is available. On clicking calendar contents of page will change and calendar will displayed with detail of task performed date vise.



Student Id: junaid **Student Name:** junaid **Supervisor ID:** jesper **Supervisor Name:** jesper **Thesis ID:** the001

Delete	Date	Task Performed
[X]	April 17, 2008	i have designed calander
[X]	April 20, 2008	2nd hay yar

Fig 7.32 Calendar screenshot

Calendar view use case

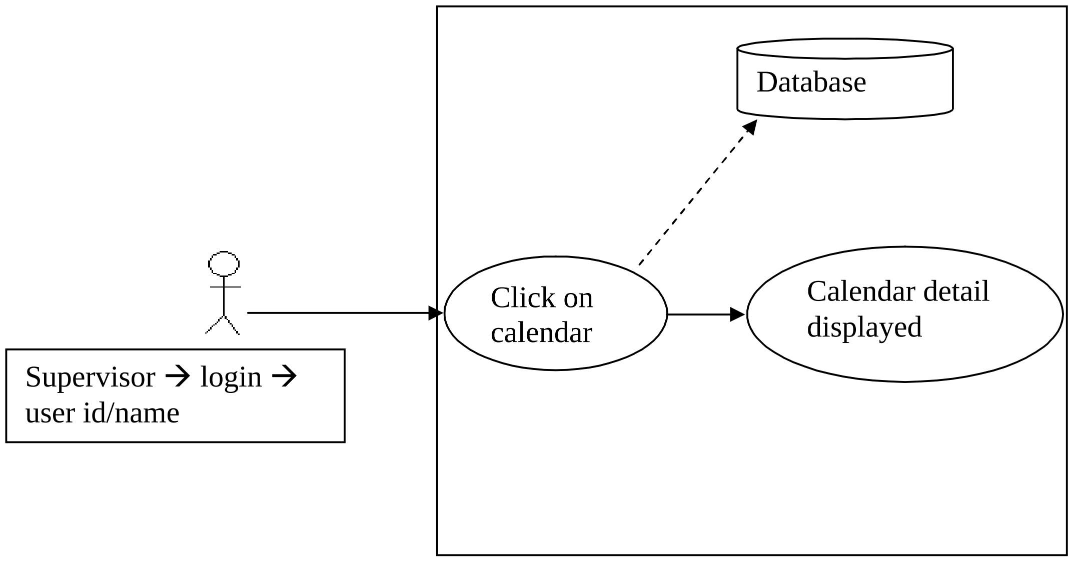
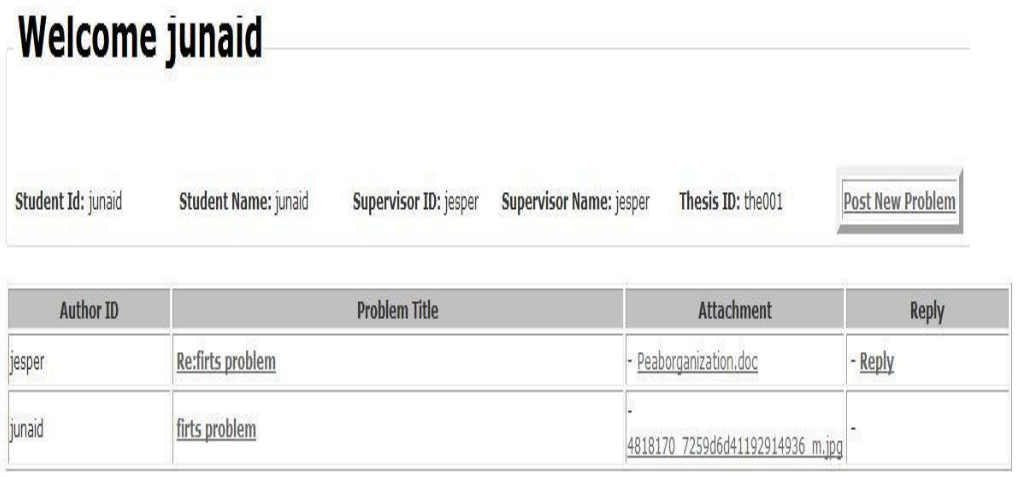


Fig 7.33 Calendar use case

7.4 Student

Student user has least right on blog. If login user is student all posts by supervisor or student himself are there in list. Student can open any post by supervisor or download attachment sent by supervisor. Student can reply post without opening the post detail or downloading attachment just by clicking on reply. Reply option is enabled on only post sent by supervisor in student login. Student can't delete any post either by his own or by student. New post or unread post's problem title is written in black font color and read post problem title is in gray text.



Welcome junaid

Student Id: junaid **Student Name:** junaid **Supervisor ID:** jesper **Supervisor Name:** jesper **Thesis ID:** the001

Post New Problem

Author ID	Problem Title	Attachment	Reply
jesper	Re:firts problem	- Peaborganization.doc	- Reply
junaid	firts problem	- 4818170_7259d6d41192914936_m.jpg	-

Fig 7.34 Student page screenshot

Student login use case

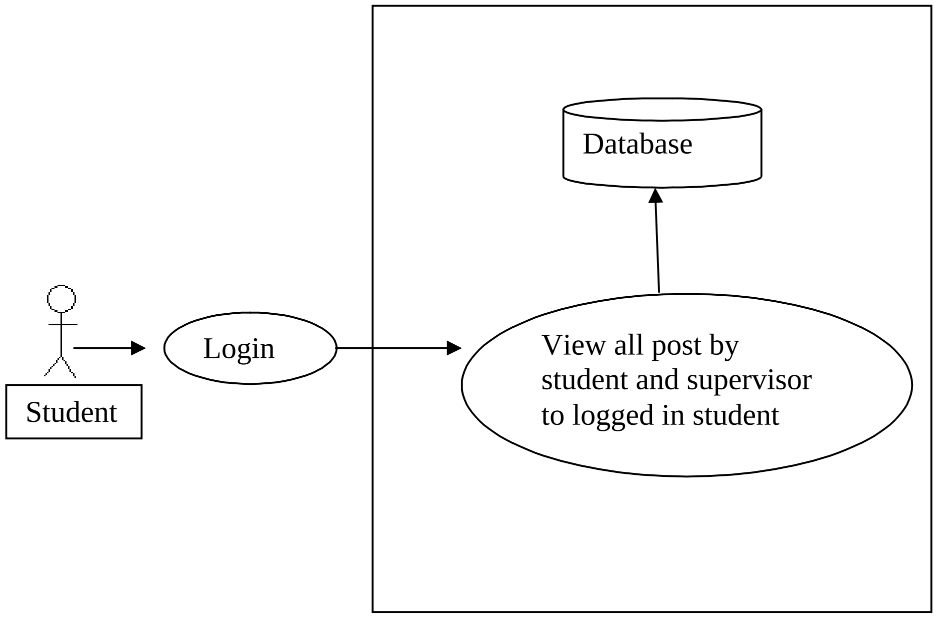
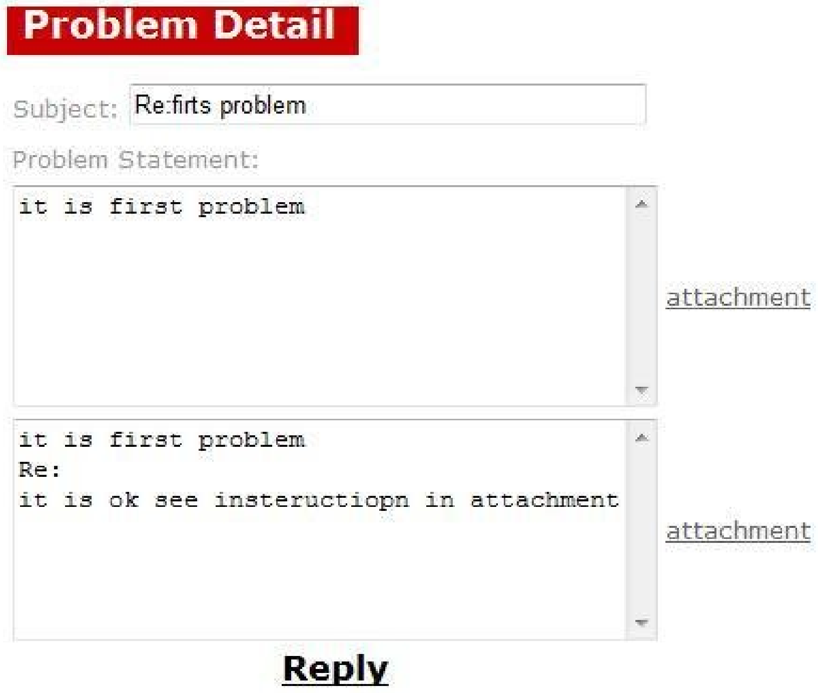

Database
Login
View all post by student and supervisor to logged in student
Student

Fig 7.35 Student login use case

7.4.1 Problem Detail

Clicking on problem title a new window will open with all history of that post, if exists, beside with attachments of each message in history. Reply option is available there in bottom if wants to reply.



Subject: Re:firts problem

Problem Statement:

```
it is first problem
```

```
it is first problem
Re:
it is ok see insteructiopn in attachment
```

Fig 7.36 problem detail screenshot

Problem detail use case

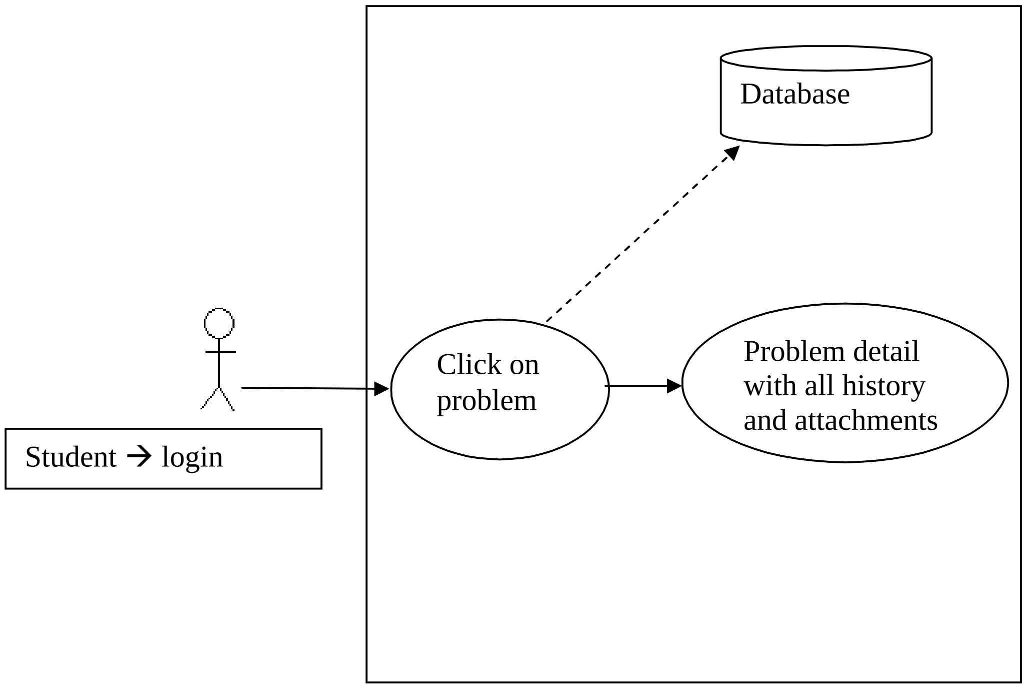
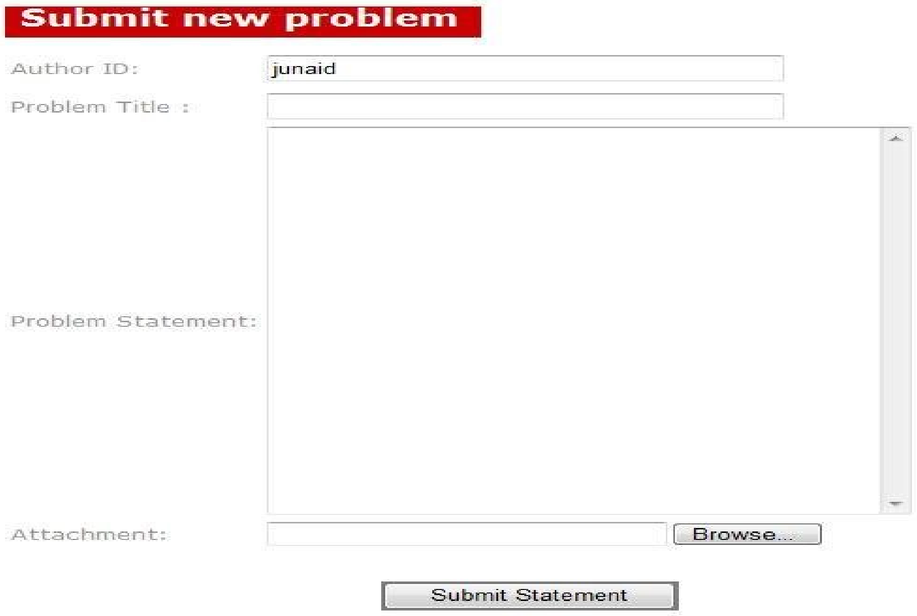


Fig 7.37 problem detail use case

7.4.2 Submit new Problem

On right top of student posts detail, Post new problem is available, if student wants to post any new problem or attachment to supervisor it can be.
Author id will be written in textbox automatically.



Submit new problem

Author ID:	junaid
Problem Title :	
Problem Statement:	
Attachment:	

Fig 7.38 Submit new problem screenshot

Post new problem use case

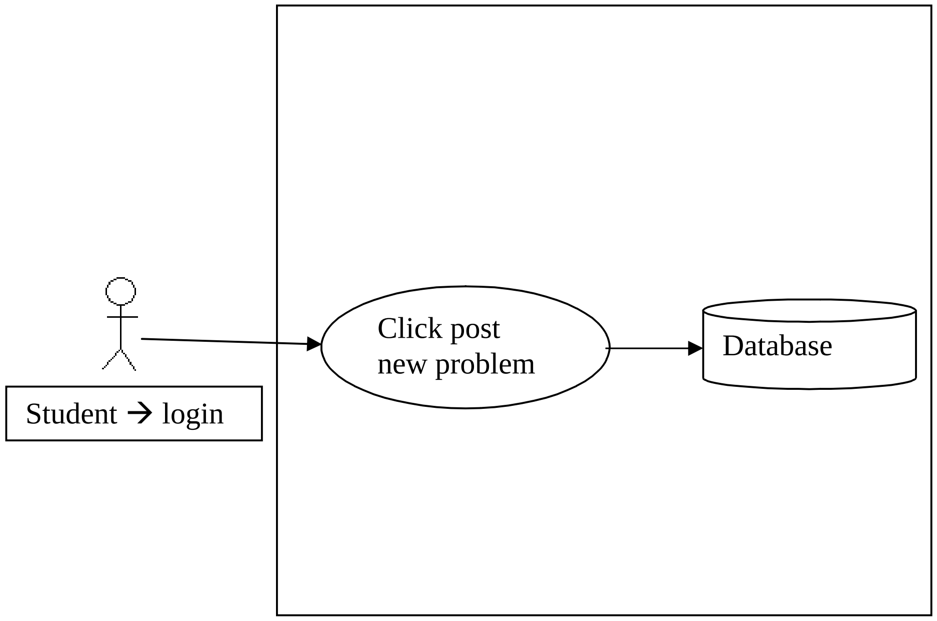
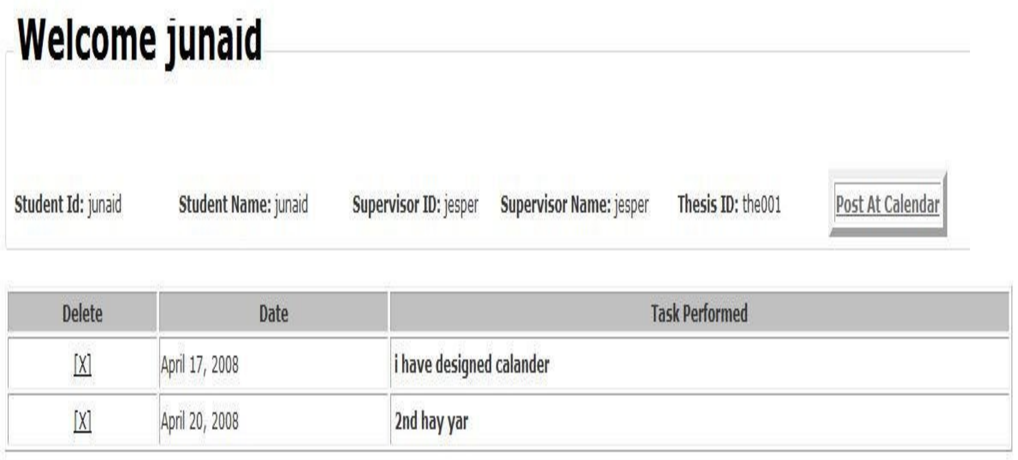


Fig 7.39 Post new problem use case

7.4.3 Calendar

Beside post new problem, calendar is available. On clicking calendar contents of page will change and calendar will displayed with detail of task performed date vise. All tasks complete by student can be see here.



Welcome junaid

Student Id: junaid **Student Name:** junaid **Supervisor ID:** jesper **Supervisor Name:** jesper **Thesis ID:** the001

Delete	Date	Task Performed
[X]	April 17, 2008	i have designed calander
[X]	April 20, 2008	2nd hay yar

Fig 7.40 Calendar screenshot

Calendar detail use case

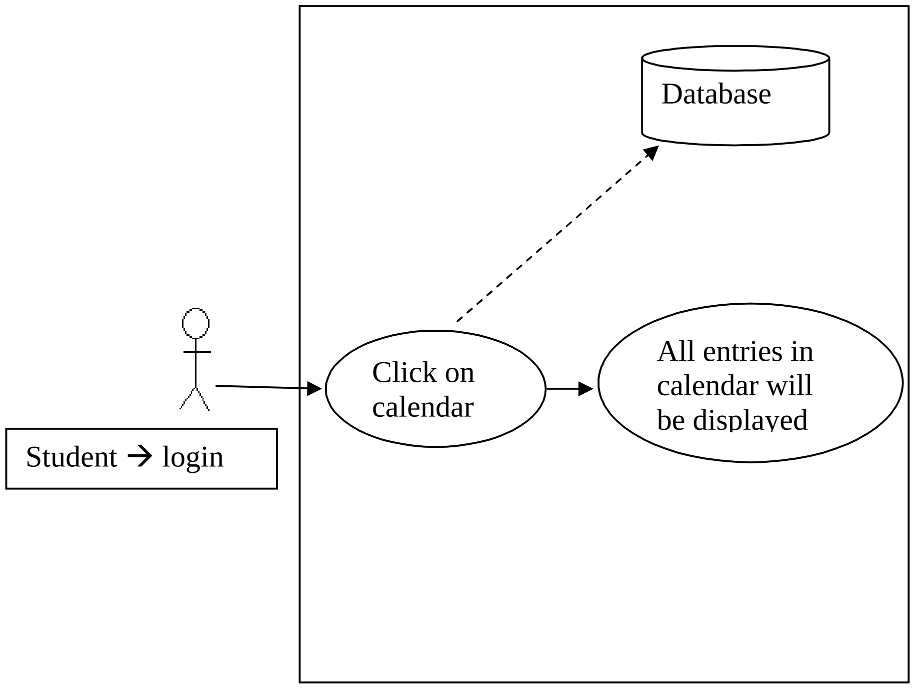
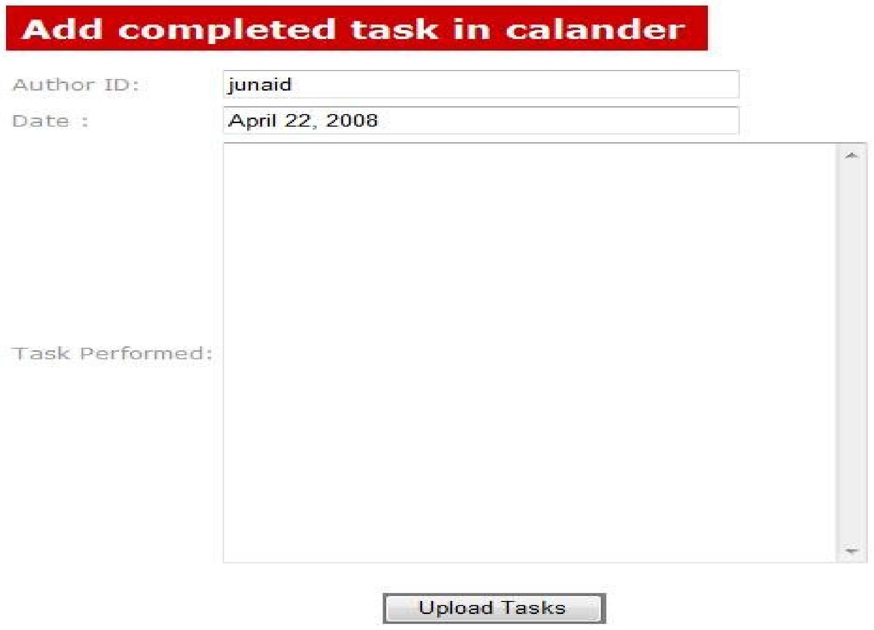


Fig 7.41 Calendar detail use case

7.4.4 Post at Calendar

At right top of page Post At calendar is available to add new post at calendar, only student can add at calendar because it contains information about assigned task at given date.



Add completed task in calander

Author ID:	junaid
Date :	April 22, 2008
Task Performed:	

Fig 7.42 Post at calendar screenshot

Post new at calendar use case

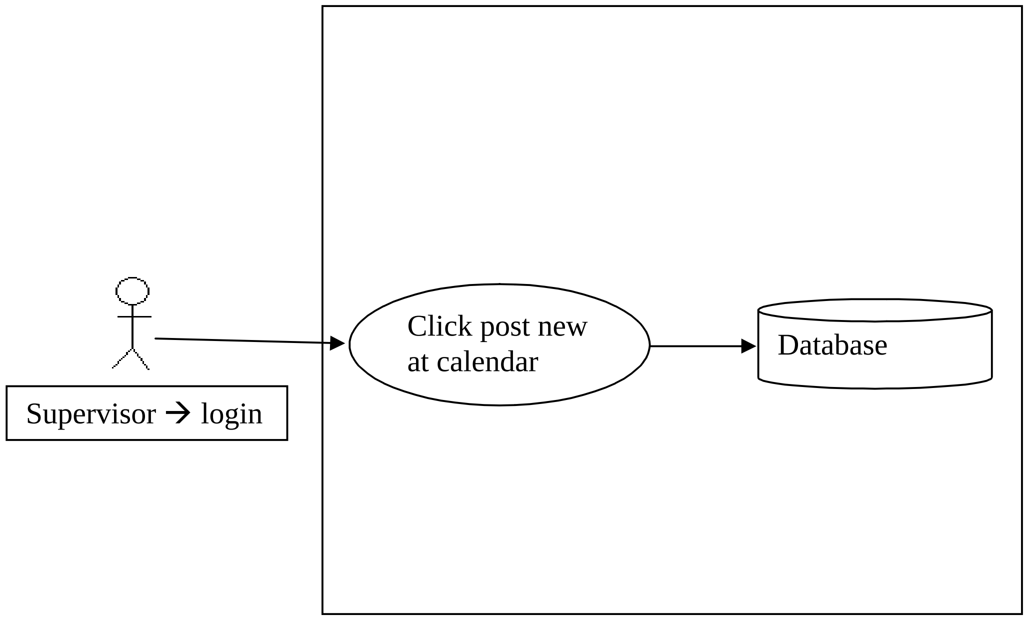


Fig 7.43 Post new at calendar use case

7.5. Logout
On every page's right top log off option is available. Clicking on this user will be logged off from current session and first login screen will be refreshed where users has to enter user id and password.

Fig 7.44 Logout screenshot

Logout Use case

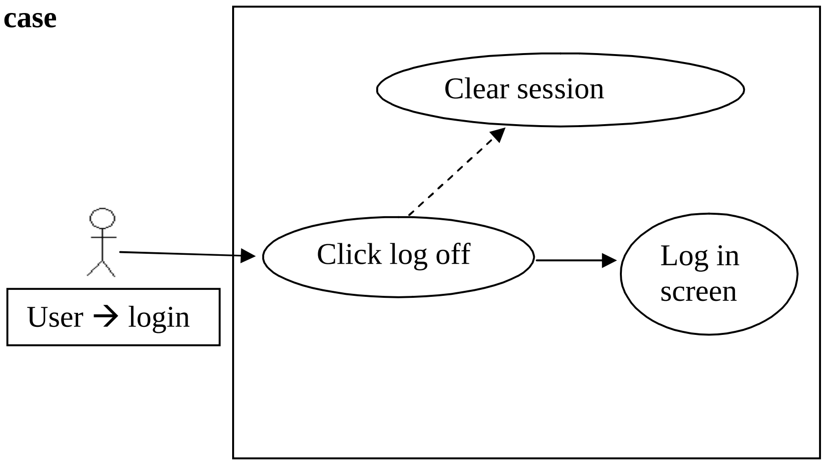


Fig 7.44 Logout use case

8. Discussion

The blog we made helps different user types in using data. The application helps a lay man or a new user to understand and apply ROR.

8.1 Conclusion

Generally administrator, supervisor and student can be possible users of this webblog. The authority depends on the type of user he or she is. The architecture of the application has been made after conducting tests. These tests consist on applying the blog on different types of users and data sitting on different platforms. The results obtained out of these tests exactly correspond to the theoretical framework of this paper.

The solution is appropriate between supervisor and students to buffer communication disorder/spikes as discussed in chapter 1. The blog helps supervisor to communicate and monitor the periodic task completion of students. On the other hand students can post problems and complete tasks to obtain feedback of supervisor.

8.2 Learning

In the beginning of this project I had little experience of web development in HTML, JavaScript, ASP and SQL. Not only this but also a thin knowledge and understanding with Ruby on Rails (RoR) as an open source web application. I was expecting it will work at server endpoint as ASP and should have to link manually with backend database. I also learned that it is extremely useful to handle all aspects of client, server and database automatically without any manual work.

Since it is a new and developing application, there is not enough material available on internet and there are very few books in library too. It was not easy to find most suitable solution for any specific problem.

At the beginning, it was tough time for me to install and configure. With due efforts, I found some key issues to install and run as follows:

- IIS and rails use same ports, in which first stop is IIS and then rails.
- It can be start only from Ruby's console, simple command prompt execute the command but not start Ruby properly.
- Best practice is to make new folder at root to save rails project.
- Never give space in the name of the folder where rails project are saved.
- By using internet explorer, open ruby page with web address http://localhost:3000/, if it opens then the installation and configuration is correct.
- Saving image in database through rails was the toughest part of this development. It saves image in three columns in database table with name, content type and data.
- Submitting form which contains attachment, which is also different, depends on the data type of attachment. Multipart property must be enabled when dealing with submitting form.

I am still exploring rails and try to find out solution for saving data from combo box.

I learned a lot from working with RoR.

9. Future work

This blog is designed and developed with basic level of functionalities. Some future work can be made to make it more effective, productive and to navigate it better.

9.1 Management Tools

The blog can be user friendly and easy to manage, when multiple hundred posts are in the blog, by adding following features.

- **List only newly added post**
  At present all posts are displayed in queue, descending order by date. It can be improved if there is an option of showing only newly uploaded posts. We can divide page in two portion one for unread posts and other for newly uploaded posts.

- **Delete more than one post at a time**
  At present supervisor has option to delete only one post at a time, which is time consuming. Check box option can be added along with each post for multiple selections to delete posts.

- **Archive by months and years**
  All posts are in the same queue irrespective of old and newly posted. It can be convenient if there is an option of archive mails, by months and years. Small calendar can be added to archive posts.

9.2 Convenience Tool

For better data representation of posts and easy to view important mails in the blog, following features can be added.

- **Rich-text editor**
  Users have only simple textbox to write in the blog. Adding text editor will make things easy to highlight important areas of written text.

- **Mark important posts**
  At present all posts are displayed in a static way in queue, there is no option to mark any important post. Flag can be added to mark important mails.

9.3 Communication tool

To make it possible for students to communicate in real time with their supervisor following feature could be added.

- **Online chat**
  Though this blog eliminate handicap between student and supervisor, still its efficiency can be improved by adding real time communication feature. Online chat could be added to provide real time communication.

9.4 Notification tool

For prompt update to supervisor or student to see what new post has been arrived in the blog following feature could be added.

- **Email on new post**
  At present supervisor and student has to often visit the blog to see new posts. There could be notification facility which can be added. Email can reduce these frequent visits to the blog. RSS (Really Simple Syndication) can be 2nd appropriate solution of notification of newly uploaded post.

10. References

Thomas, David, 2006, Agile web development with Rails, Published by Raleigh, N.C.: Pragmatic Bookshelf