Project Documentation

RED: Student Enrolment System

ABSOLUTION DEVELOPMENT GROUP

Bahman Razmpa
Harsimran Singh Maan
Jaspreet Singh Thind
Jingbo Yu
Mengmeng Jiang
Wittawas Chotwanwirach

Date: 24 April 2013

Revision Summary

Date	Version	Description	Author
18/March/2013	1.0	Software Requirement Specifications	ADG Development
05/April/2013	2.0	Software Design Document	ADG Development
24/April/2013	3.0	Final Project Document	ADG Development

Table of Contents

REQUIREMENTS AND ANALYSIS DOCUMENT

1. Introduction

1.1. Purpose

This project is aimed to design a Student Enrolment Software System (codename: RED) for the engineering faculty at a university. RED replaces a manual data entry, storage and manipulation system and provides an end to end solution for the enrolment process. The system can be used by several types of user such as students, faculty members and administrators which in principle called the “User”. A user is privileged with ability to interact with specific set of features within RED based on the access level that is defined for that particular type of user. The system can be utilized for activities such as to enrol in a program or register for courses by a student, to manage the courses they teach by faculty members, and by administrators to manage the overall interaction of the rest of the users.

1.2. Document Conventions

The document contains standardized templates for use-cases and standard UML conventions for the diagrams. Priorities, if any, have been marked with the individual components. Some main components and diagrams may be followed by sub-components and sub diagrams which may elaborate the functionality of the system.

1.3. Intended Audience and Reading Suggestions

The document is intended for Developers, System Architects, System Engineers, Managers, End users, Testers, Documentation writers and the University Management. Reader may go through product functions and user characteristics to get an overview of the product features. The diagrams provide overview of the use cases and different modelling diagrams outline the high level procedures that the system would follow and the expected results. The end users may get an abstract view of the system after going through the document.

1.4. Product Scope

The overall product is scoped to help Students, Faculty and University administration to channelize the Enrolment process. RED provides functionality for each of its user but this is not limited to the list below:

<u>Student:</u>

- RED helps Students to register for courses for the upcoming semester.
- A student can find course information including: course department, pre-requisites, course description, deadlines and then register for course.
- Once the registration process is completed, a student is able to check the timetable, current list of registered courses, and can drop a course.
- A student is also able to check the grades of previous courses.

Faculty:

- Faculty is able to access the system to check the course information which they will be teaching, which includes the registration status of a course and the contact information of the registered students.
- At the end of each semester, faculty member can submit the grades of each course. Once the marks are submitted to the system, they are not allowed to modify unless the faculty send formal requests to the administrator.
- Faculty members can generate score reports (including historical data) of the courses they teach.
- Faculty may send requests to an admin to update the information provided for a specific course that they teach.

Administrator:

- An administrator can use RED in order to manage details about the courses, departments and programs.
- Admin can approve special request from the students or faculty.
- Admin can setup overall system policies and permissions.

Overall the system would be comprised of three major components:

Central Repository:

A database system capable of dynamically and efficiently storing all the information on the system such as user profiles, course information, registration history and etc.

Enrolment Service:

A robust API unit that connects the graphical user interface and the database systems and also enables other systems to interact with RED.

User Dashboard (GUI):

A user friendly graphical interface that allows the users to interact with the system and utilize its features to make intended use.

Each component inherently includes **Documentation**. This helps the end users to gain insights to the system functions.

1.5. References

The following books have been referred to for the conventions on Diagrams.

- Object Oriented Software Engineering Using UML Patterns and Java by Bernd Bruegge, Third Edition - 2012.
- Using UML, Software Engineering with Objects and Components, Perdita Stevens, Second Edition, Addison Wesley Publications.

The document template is a modified version of the IEEE template for Software Requirement Specifications document.

RED logo is designed by Jeremy Wallace.

2. Overall Description

2.1. Product Perspective

The system RED is a new enrolment software system. RED is an independent and self-contained system. RED has three main participating actors: Student, Faculty and Administrator. The actors interact with RED by their own tailored GUI once they have successfully logged in.

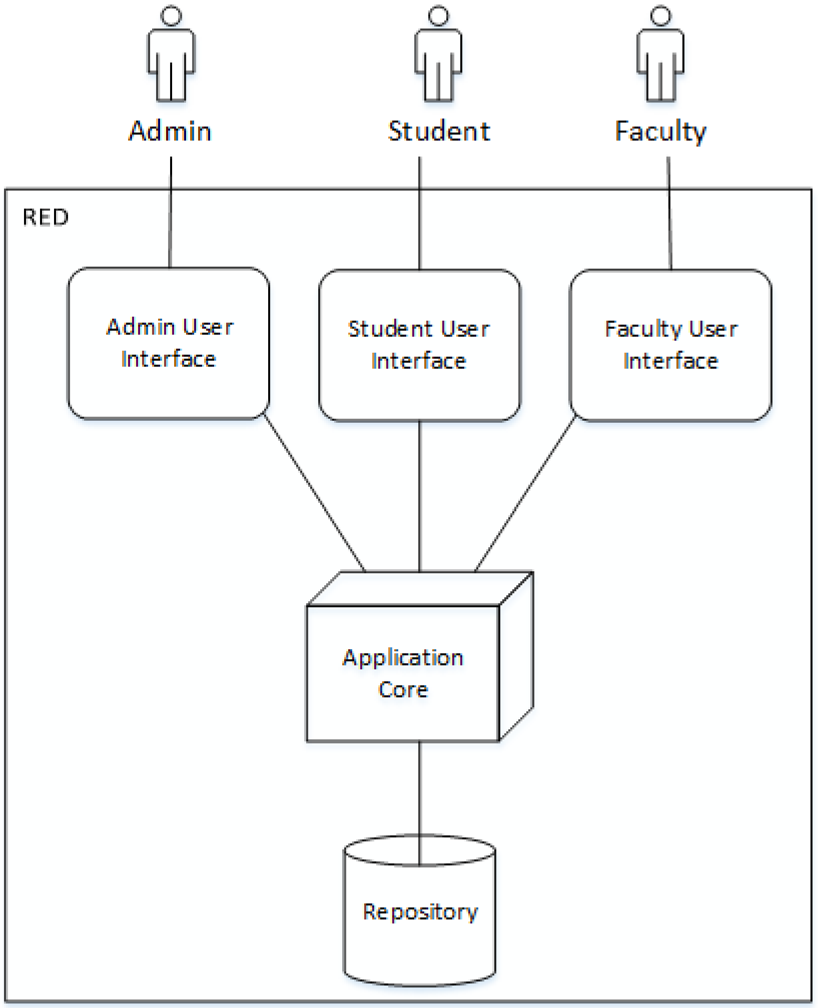


Fig 2.1.1 Product Overview

2.2. Product Functions

RED provides the following basic functions for the end users:

- Student
  - Register and drop courses
  - Display courses
  - Course search
  - View timetable
  - View degree info

- Faculty
  - View teaching timetable
  - Generate reports
  - Submit grades for the course
  - Submit special approval requests
- Administrator
  - Setup and manage users
  - Setup and manage departments, courses and programs

Every user is required to login to RED in order to use the system.

2.3. User Classes and Characteristics

RED is intended for uses by three different set of users: Students, Faculty and Administrators. The characteristics of each user are listed below:

Student:

- Student can be either undergraduate or graduate, full-time or part-time
- Some student might not have knowledge about Enrolment at all (freshmen)
- Student has basic computer knowledge

Faculty:

- Is a university professor or is an instructor
- Has administration and basic computer knowledge
- Has good knowledge about Enrolment process in general

Administrator:

- Has administration expertise and basic computer knowledge

2.4. Operating Environment

The system would be intended to run on windows (7 or higher) operating system. Any system that supports JRE (v1.7 or higher) should be able to run the software.

Minimum hardware requirements are as follows:

i. Disk Space: 100 MB
ii. RAM: 2 GB
iii. Processor: Intel Core i3 or AMD A4

2.5. Design and Implementation Constraints

The design of RED has the following constraints:

- Performance constraints:
  - Since it is impossible to design precise performance criteria, RED would be designed to be close to optimal.

    - Parallel operations may not be guaranteed if hardware is not scaled when the number of users increases.
    - Testing will be performed to ensure that RED meets the performance targets. If it does not, further optimizations will be made, or if it proves impossible, the targets may have to be renegotiated.

- Security constraints:
  Since RED would be designed to interact with a number of external systems, the security model would be flexible to include such interactions.

- Hardware and operating system constraints:
  The design of RED is created under the assumptions that the user has a processor capable of running Windows 7 or above and which also has access to a network.

- Implementation constraints:
  RED would be implemented using Java. Deviations from this may need to be discussed upfront and any licence fees involved with the use of any implementation technology are t9 be borne by the client.

2.6. User Documentation

Frequently Asked Questions (FAQ) and Glossary for students, professors and administrators should be accessible from the any window of the application.

2.7. Assumptions and Dependencies

The assumptions and dependencies relate to the capabilities of the RED are as follows:

- It is assumed that each student is already registered for the program and has an account with RED.
- RED's functionality is restricted to the Enrolment services only and may not extend to any other university operations.
- RED may depend on some open source or commercial modules and packages, the cost of which has to be borne by the client.

3. External Interface Requirements

3.1. User Interfaces

The product provides a Graphical user interface with command-line login functionality. The UI should have a menu, a breadcrumb navigator and would adhere to the flowing standards:

i. Help: A help icon must be available on every screen. The user should be able to invoke this at any time the user controls the system.
ii. Fields with fixed multiple choices should enable the user to select a choice rather than having to retype the options.
iii. Feedback: User actions should be acknowledged and every action should provide feedback to the user about the current state of the system.

3.2. Hardware Interfaces

RED should be implemented in a hardware agnostic manner and should not rely any specific hardware interfaces. It is however assumed that the system running the software is connected to a standard output display device and standard input keyboard a pointing device.

3.3. Software Interfaces

Only Java Runtime Engine and an R-DBMS repository are necessary to run the system. This however does not exclude any library packages and standard OS requirements.

3.4. Communication Interfaces

The system should provide an interface to interact with the university's central system. The system may also provide standard service interfaces to interact with other systems.

4. System Features

The following section describes the system features from the end-user's point of view.

4.1. General Features

These are a set of features that every user of the system can use:

- Log-in
- View/Edit Personal Info
- Personal Messages
- Glossary
- FAQ
- Log-out

4.1.1. Login

Feature name	***Login***
Participating actors	*User*
Flow of events	*1. User has opened RED.* *2. RED displays the login window asking for username and password.* *3. User enters the username and password and requests for login.* *User can also select to recover the password by initiating "**ForgotPassword**" use case.* *4. On a successful authentication RED presents the User with a Home screen with select features depending on the access level of User.*
Entry condition	*User has opened RED*
Exit condition	*User closes RED or is successfully logged in*
Exception	*If User enters incorrect username and/or password, the system displays an error and allows the user to re-enter their credentials.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand.*

4.1.2. Logout

Feature name	***Logout***
Participating actors	*User*
Flow of events	*1. User presses the Log Out button.* *2. RED presents the user with the Login screen.*
Entry condition	*User is logged into RED.*
Exit condition	*User successfully logged out of RED.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

4.1.3. Change Password

Feature name	***ChangePassword***
Participating actors	*User*
Flow of events	*1. User selects the "Change Password" option.* *2. RED provides a prompt to User asking to enter "Existing Password" or cancel request.* *3. User submits their correct "Existing Password".* *4. RED provides a new prompt for user to enter a "New Password" twice to confirm that the new password is entered correctly.* *5. User Enters the New Password and Selects SAVE.* *6. RED Saves the New Password and provides a message to the user "Password changed successfully"*
Entry condition	*If user enters an incorrect entry for Current Password, RED prompts user with an appropriate error and asks User to enter Existing Password again.*
Exit condition	*User is in "Personal Info" Tab and selects "Change Password" option.*
Exception	*User can select another tab from the main RED option row on the top.* *User can select "Back" option that takes them to the previous page.* *User can log out of the RED.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

4.1.4. Forgot Password

Feature name	***ForgotPassword***
Participating actors	*User*
Flow of events	*1. User selects Forgot password on the login screen* *2. RED prompts the user for security question and email address* *3. User answers the security question and enters the email address.* *4. RED sends and email to the user with a temporary password.*
Entry condition	*User is opens RED.*
Exit condition	*User receives a temporary password.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

4.1.5. Manage Personal Info

4.1.5.1. View Personal Info

Feature name	***ViewPersonalInfo includes <<ChangePassword>> & <<EditPersonalInfo>>***
Participating actors	*User*
Flow of events	*1. User selects the “Personal Info” tab.* *2. RED provides the User with their personal information recorded in the system. The page also includes features such as “Change Password” and “Edit Personal Info”.* *3. User can activate the **ChangePassword** and **EditPersonalInfo** use cases by selecting them.*
Entry condition	*Student is logged into RED and Selects “Personal Info” Tab.*
Exit condition	*Student can select another tab from the main RED option row on the top.* *Student can select “Back” option that takes them to the previous page.* *Student can log out of the RED.*
Exception	*At any point, if something is unclear to Student, RED should provide options to help him understand*
Quality requirements	*1. User selects the “Personal Info” tab.* *2. RED provides the User with their personal information recorded in the system. The page also includes features such as “Change Password” and “Edit Personal Info”.* *3. User can activate the **ChangePassword** and **EditPersonalInfo** use cases by selecting them.*

4.1.5.2.Edit Personal Info

Feature name	***EditPersonalInfo***
Participating actors	*User*
Flow of events	*1. User selects the "Edit Personal Info" option.* *2. RED provides a prompt to User asking for Existing Password or Cancel request.* *3. User enters the correct Existing Password.* *4. RED makes the fields in the Personal Info Page that are editable, dynamic for User to make appropriate changes, and provides a SAVE button at the bottom.* *5. User updates his personal information and selects SAVE.* *6. RED Saves the New Information and provides a suitable message to the user "Personal Information updated successfully" and activates "**ViewPersonalInfo**" use case.*
Entry condition	*If user enters an incorrect entry for Current Password, RED prompts user with an appropriate error and asks User to enter Existing Password again.*
Exit condition	*User is in "Personal Info" Tab and selects "Edit Personal Info" option.*
Exception	*User can select another tab from the main RED option row on the top.* *User can select "Back" option that takes them to the previous page.* *User can log out of the RED.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

4.1.6. Messages

4.1.6.1.Read Message

Feature name	***ReadMessages***
Participating actors	*User*
Flow of events	*1. User selects the "Message Center" tab.* *2. RED loads a page that shows subject list of all messages for the user in chronological date with most recent mail on top, and with the messages that are not read in BOLD font.* *3. User selects a particular message.* *4. RED opens that message showing the sender, date, and subject and message body.* *5. User can select to view next/previous message.* *6. RED loads the next or previous message.*
Entry condition	*User is logged into RED.*
Exit condition	*User can select another tab from the main RED option row on the top.* *User can log out of the RED.*
Exception	*If there are no new messages, RED writes on the screen that there are no new messages to display and shows only the old messages.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.1.1.1.Mark Message Read/Unread

Feature name	**MarkMessageReadOrUnread**
Participating actors	User
Flow of events	1. User selects the "Message Center" tab. 2. RED loads a page that shows subject list of all messages for the user in chronological date with most recent mail on top, and with the messages that are not read in BOLD font. 3. User selects a particular message. 4. RED opens that message showing the sender, date, and subject and message body. 5. User can select to view next/previous message. 6. RED loads the next or previous message.
Entry condition	User is logged into RED.
Exit condition	User can select another tab from the main RED option row on the top. User can log out of the RED.
Exception	If there are no new messages, RED writes on the screen that there are no new messages to display and shows only the old messages.
Quality requirements	At any point, if something is unclear to User, RED should provide options to help him understand

1.1.1.2.Delete Message

Feature name	**DeleteMessages**
Participating actors	User
Flow of events	1. User selects the "Message Center" tab. 2. RED loads a page with that shows subject list of all messages for the user in chronological date with most recent mail on top, and with the messages that are not read in BOLD font. 3. User selects a particular message any number of messages by tick marking the check box next to the name of that message and pressing the distinguished "Delete Mail" button. 4. RED deletes those mails selected and refreshes the mailing list.
Entry condition	User is logged into RED.
Exit condition	User can select another tab from the main RED option row on the top. User can log out of the RED.
Exception	If the mailing list is empty, the "Delete Mail" button is inactive.
Quality requirements	At any point, if something is unclear to User, RED should provide options to help him understand

1.1.2. Browse Home Screen

Feature name	***BrowseHomeScreen***
Participating actors	*User*
Flow of events	*RED displays home screen with all the options and TABS available to the User depending on their access level.*
Entry condition	*User logs into RED. Or,* *User selects "Home" tab from RED's static top selection menu.*
Exit condition	*User closes RED. Or* *Logs out of RED.*
Exception	*None*
Quality requirements	*The home page is well organized, easy to follow, inclusive of all the features and descriptive of key features the User can take advantage of.*

1.1.3. Browse Glossary

Feature name	***BrowseGlossary***
Participating actors	*User*
Flow of events	*1. User selects the "Glossary" tab.* *2. RED loads a page with the glossary of all the keywords used in the system on the left hand side.* *3. User can select the first letter of the keyword they are looking for, from the static row of alphabets on top of the glossary.* *4. RED shrinks down the glossary list to the keywords starting with that particular letter.* *5. User selects the keyword they are looking for.* *6. RED provides the description of that keyword inside a drop down menu below the keyword or by using a pop up window that contains the description.*
Entry condition	*User is logged into RED.*
Exit condition	*User can select another tab from the main RED option row on the top.* *User can log out of the RED.*
Exception	*If a particular letter does not have a keyword in the glossary that starts with that letter, the static top row will have that letter as disabled.*
Quality requirements	*None*

1.1.4. Browse FAQs

Feature name	***BrowseFAQ***
Participating actors	*User*
Flow of events	*1. User selects the "FAQ" tab.* *2. RED loads a page with the alphabetically ordered list of different topics that have FAQs related to them.* *3. User can select the appropriate topic that they need to find an FAQ for.* *4. RED drops down a menu with the alphabetically ordered list of all FAQs in that particular topic and their short answers immediately after them.*
Entry condition	*User is logged into RED.*
Exit condition	*User can select another tab from the main RED option row on the top.* *User can log out of the RED.*
Exception	*None*
Quality requirements	*None*

1.1.5. Search

Feature name	***Search***
Participating actors	*User*
Flow of events	*1. User select the search box and enters the search term.* *2. RED displays a list of all results related to the search term.*
Entry condition	*User is logged into RED.*
Exit condition	*User can select another tab from the main RED option row on the top.* *User can log out of the RED.*
Exception	*If not matching items are found, a message is displayed "No results found"*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.2. Features for Student

The following is the list of features that a Student user has access to:

- Browse Course List
- Add/Drop a Course
- View Time Table
- View Current Enrolment
- View Degree Information

1.2.1. Manage Courses

1.2.1.1.Browse Courses

Feature name	***BrowseCourse includes <<ViewCourseInfo>>***
Participating actors	*Student*
Flow of events	*1. Student selects the "Browse Course" tab.* *2. RED provides a page with the list of all the department abbreviations alphabetically, plus the full name of the department and the name of the Faculty they belong to.* *3. Student can browse through the list to select the name of the department that offers the course that they are looking for. User can also select the initial of the department's abbreviation from the top row list of alphabets.* *4. RED shrinks down the list to departments starting with that initial* *5. Student selects the department they are looking for.* *6. RED loads the list of all the courses offered by that department plus their course number and Title.* *7. Student selects the course they are looking for to activate "ViewCourseInfo" use case.*
Entry condition	*Student is logged into RED.*
Exit condition	*Student can select another tab from the main RED option row on the top. Or,* *Student can select "Back" option that takes them to the previous page. Or,* *Student can log out of the RED.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.2.1.2. View Course

Feature name	***ViewCourseInfo includes <<DropCourse>> & <<RegisterCourse>>***
Participating actors	*Student*
Flow of events	*1. Student selects a particular Course.* *2. RED loads a new page that includes: course description, course schedule, instructor name, list of pre-requisites, list of available tutorial or labs if any available, registration restrictions and start and end time(s) and REGISTER and DROP Buttons.* *3. Student can select "Register" or "Drop" button* *4. RED activates "RegisterCourse" or "DropCourse" use cases accordingly.*
Entry condition	*Student is logged into RED.* *Student selects a course at any time in any page or tab inside RED.*
Exit condition	*Student can select another tab from the main RED option row on the top.* *Student can select "Back" option that takes them to the previous page.* *Student can log out of the RED.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Student, RED should provide options to help him understand*

1.2.1.2.1. Register for course

Feature name	***RegisterCourse***
Participating actors	*Student*
Flow of events	*1. Student selects "Register" button.* *2. RED prompts a notification requesting the User to: "Confirm Registration?"* *3. Student confirms their registration request.* *4. RED registers student in the course, adds the section to his timetable and returns to the course information page with the "Register" button disabled. Drop Course button is now available.*
Entry condition	*Student has initiated "BrowseCourse" use-case.*
Exit condition	*Student cancels register confirmation request, RED returns to course info page.*
Exception	*If the student is NOT eligible to register for the course, the "Register Course" button is disabled.*
Quality requirements	*At any point, if something is unclear to Student, RED should provide options to help him understand*

1.2.1.2.2. Drop course

Feature name	***DropCourse***
Participating actors	*Student*
Flow of events	*1. Student selects "Drop course" button.* *2. RED displays a confirmation request to drop the course.* *3. Student confirms to drop the course.* *4. RED displays a suitable message to the Student indicating that "Course successfully dropped"*
Entry condition	*Student selects the "Drop course" option from the Course Info Page if already registered for the course.*
Exit condition	*The course is successfully dropped.* *The Student Logs Out of RED* *The Student navigates to some other Tab* *The Student presses CANCEL to go back to the landing Page*
Exception	*If Student cancels the confirmation request, RED reverts back to the most recent "Course Info Page".*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.2.2. View Timetable

Feature name	***ViewTimeTable includes <<ViewCourseInfo>>***
Participating actors	*Student*
Flow of events	*1. Student clicks on 'Time Table tab to activate the use case.* *2. RED opens the "Time Table" page which by default shows the timetable for the current term; highlighting blocks of time in a sample calendar for each course and indicating it with the course name and number. The Student may opt for a different term.* *3. Student selects a particular term that is NOT the current term and clicks SHOW button.* *4. RED queries the database and prints out the timetable for the particular term.* *5. Student can select a course on the list to activate "ViewCourseInfo" use case.*
Entry condition	*Student is logged into RED and clicks on 'Time Table' Tab from any Page*
Exit condition	*The student navigates to some other page.* *The student activates "LogOut".*
Exception	*If the Student is not enrolled in any courses for a particular chosen term, RED provides a suitable notification indicating: "TBC – To be Confirmed" and returns to "Time Table" page.*
Quality requirements	*Student should be able to invoke help option when necessary.* *Correct time table should be visible* *Time table should load within 1 second of invocation.*

1.2.3. View Enrolments

Feature name	***ViewEnrolments includes <<ViewCourseInfo>>***
Participating actors	*Student*
Flow of events	*1. Student selects the "MyCurrentEnrolments" Tab.* *2. RED displays a page with all course enrolments of the Student sorted by Term. Including courses from the first Term student has had a course registered in all the way up to the current school term and the immediate term after, only.* *3. Student can select a particular Term that they are interested in.* *4. RED trims the list down to the student enrollment history for that particular term.* *5. Student can select a course on the list to activate "ViewCourseInfo" use case.*
Entry condition	*Student is logged into the RED.* *Student selects the "My Current Enrolments" Tab.*
Exit condition	*The Student is presented with the list of enrolled courses or a suitable message respectively.*
Exception	*If the Student is NOT enrolled in any courses at all or for a particular term they are interested in, RED displays a notification indicating: "Student does not have an enrolment history for this selection."*
Quality requirements	*At any point, if something is unclear to Student, RED should provide options to help him understand*

1.2.4. View Degree Info

1.2.4.1. View Degree Requirements

Feature name	***ViewDegreeRequirements***
Participating actors	*Student*
Flow of events	*RED displays the list of courses required for student to complete the program*
Entry condition	*Student is logged into RED and selects the tab for "View degree requirements"*
Exit condition	*Student selects any other tab*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Student, RED should provide options to help him understand*

1.2.4.2.View Grade Summary

Feature name	***ViewGradeSummary***
Participating actors	*Student*
Flow of events	*1. Student selects the Grade Summary tab.* *2. RED provides the list of all courses student has taken plus their grade both in percentage format and in alphabetical format as well as the term and year they have taken them, since they were added to the system plus the courses they are registered in, which are either in progress or have not started.* *3. Student can select a particular year of enrolment from the top row.* *4. RED shrinks the grade summary list to the courses that are either registered to be taken or have been taken in that particular term.* *5. Student can select a course on the list to activate "**ViewCourseInfo**" use case.*
Entry condition	*Student is logged into RED.*
Exit condition	*Student selects any other tab*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Student, RED should provide options to help him understand*

1.3. Features for Faculty

The following is the list of features a Faculty Member has access to:

- Uploading Scores
- Making Special Approvals
- View Current Teaching Time Table
- Generate Course Report

1.3.1. View Timetable

Feature name	***ViewTimetable***
Participating actors	*Faculty*
Flow of events	*RED displays the schedule for all the courses for the faculty member*
Entry condition	*User is logged into RED and selects "View Timetable"*
Exit condition	*Faculty selects any other tab*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.3.2. Submit Special Approval Request

Feature name	***SubmitSpecialApprovalRequest***
Participating actors	*Faculty*
Flow of events	*1. Faculty selects the "Submit special Approval"* *2. RED prompts to enter the course and the student for whom the approval is to be granted* *3. Faculty enters the details and submits* *4. RED prompts the Faculty for confirmation* *5. Faculty confirms* *6. RED displays a success message*
Entry condition	*Faculty is logged into RED.*
Exit condition	*Faculty selects any other tab*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Faculty, RED should provide options to help him understand*

1.3.3. View Course

1.3.3.1.Generate Reports

Feature name	***GenerateReports***
Participating actors	*Faculty*
Flow of events	*1. Faculty selects the view reports button* *2. RED displays the average, highest grade, lowest grade, etc. for the course in the past years.*
Entry condition	*Faculty is logged into RED and has selected a course*
Exit condition	*Faculty selects any other tab*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Faculty, RED should provide options to help him understand*

1.3.3.2.View Registrants

Feature name	***ViewReigstrants***
Participating actors	*Faculty*
Flow of events	*1. Faculty selects "View registrants"* *2. RED displays a list of all students registered for the course*
Entry condition	*Faculty is logged into RED and has selected a course*
Exit condition	*Faculty selects any other tab*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Faculty, RED should provide options to help him understand*

1.3.4. Submit Grades

Feature name	***SubmitGrades***
Participating actors	*Faculty*
Flow of events	*1. Faculty selects the upload grades option* *2. RED prompts for the filename* *3. Faulty provides the filename* *4. RED reads the file and saves the grades*
Entry condition	*Faculty is logged into RED and has selected a course*
Exit condition	*Faculty selects any other tab*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Faculty, RED should provide options to help him understand*

1.4. Features for Admin

The following is the list of features that an Administrator account holder has access to. These features are considered lower in priority compared to feature sets for Student and Faculty Member users:

- Manage Courses
- Manage Users
- Manage Sections
- Manage Departments

1.4.1. Manage Courses

1.4.1.1.Create New Course

Feature name	***CreateCourse***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Create Course option* *2. RED displays a form asking for fields related to Course to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the new course. RED redirects the Administrator to the ViewCourseInfo page of the new course.*
Entry condition	*Administrator is logged into RED.*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.1.2.View Existing Course

Feature name	***ViewCourseInfo includes <<CreateCourse>> , <<UpdateCourse>> & <<DeleteCourse>>***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects a particular Course.* *2. RED loads a new page that includes: course description, course schedule, instructor name, list of pre-requisites, list of available tutorial or labs if any available, registration restrictions and start and end time(s) and Create, Update and Delete buttons.*
Entry condition	*Administrator is logged into RED and selects a particular course to view the details*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.1.3.Update Course

Feature name	***UpdateCourse***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Update Course option* *2. RED displays a form asking for fields related to Course to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the course. RED redirects the Administrator to the ViewCourseInfo page of the updated course.*
Entry condition	*Administrator is logged into RED and is viewing a course*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.1.4.Delete Course

Feature name	***DeleteCourse***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Delete Course option* *2. RED prompts for confirmation* *3. Administrator confirms to delete the course* *4. RED inactivates the course and displays a success message*
Entry condition	*Administrator is logged into RED and is viewing a course*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.2. Manage Users

1.4.2.1.Create New User

Feature name	***CreateUser***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Create User option* *2. RED displays a form asking for fields related to User to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the new User. RED redirects the Administrator to the ViewUserInfo page of the new User.*
Entry condition	*Administrator is logged into RED.*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.2.2. View Existing User

Feature name	***ViewUserInfo includes <<CreateUser>> , <<UpdateUser>> & <<DeleteUser>>***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects a particular User.* *2. RED loads a new page that includes complete details of the user and Create, Update and Delete buttons.*
Entry condition	*Administrator is logged into RED and selects a particular User to view the details*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.2.3. Update User

Feature name	***UpdateUser***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Update User option* *2. RED displays a form asking for fields related to User to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the User. RED redirects the Administrator to the ViewUserInfo page of the updated User.*
Entry condition	*Administrator is logged into RED and is viewing a User*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.2.4. Delete User

Feature name	***DeleteUser***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Delete User option* *2. RED prompts for confirmation* *3. Administrator confirms to delete the User* *4. RED inactivates the User and displays a success message*
Entry condition	*Administrator is logged into RED and is viewing a User*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.3. Manage Roles

1.4.3.1. Create New Role

Feature name	***CreateRole***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Create Role option* *2. RED displays a form asking for fields related to Role to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the new Role. RED redirects the Administrator to the ViewRoleInfo page of the new Role.*
Entry condition	*Administrator is logged into RED.*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.3.2. View Existing Role

Feature name	**ViewRoleInfo includes <<CreateRole>> , <<UpdateRole>> & <<DeleteRole>>**
Participating actors	*Administrator*
Flow of events	*1. Administrator selects a particular Role.* *2. RED loads a new page that includes complete details of the Role and Create, Update and Delete buttons.*
Entry condition	*Administrator is logged into RED and selects a particular Role to view the details*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.3.3. Update Role

Feature name	**UpdateRole**
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Update Role option* *2. RED displays a form asking for fields related to Role to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the Role. RED redirects the Administrator to the ViewRoleInfo page of the updated Role.*
Entry condition	*Administrator is logged into RED and is viewing a Role*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.3.4. Delete Role

Feature name	***DeleteRole***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Delete Role option* *2. RED prompts for confirmation* *3. Administrator confirms to delete the Role* *4. RED inactivates the Role and displays a success message*
Entry condition	*Administrator is logged into RED and is viewing a Role*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.4. Manage Departments

1.4.4.1. Create New Department

Feature name	***CreateDepartment***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Create Department option* *2. RED displays a form asking for fields related to Department to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the new Department. RED redirects the Administrator to the ViewDepartmentInfo page of the new Department.*
Entry condition	*Administrator is logged into RED.*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.4.2. View Existing Department

Feature name	***ViewDepartmentInfo includes <<CreateDepartment>> , <<UpdateDepartment>> & <<DeleteDepartment>>***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects a particular Department.* *2. RED loads a new page that includes complete details of the Department and Create, Update and Delete buttons.*
Entry condition	*Administrator is logged into RED and selects a particular Department to view the details*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.4.3. Update Department

Feature name	***UpdateDepartment***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Update Department option* *2. RED displays a form asking for fields related to Department to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the Department. RED redirects the Administrator to the ViewDepartmentInfo page of the updated Department.*
Entry condition	*Administrator is logged into RED and is viewing a Department*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.4.4.Delete Department

Feature name	***DeleteDepartment***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Delete Department option* *2. RED prompts for confirmation* *3. Administrator confirms to delete the Department* *4. RED inactivates the Department and displays a success message*
Entry condition	*Administrator is logged into RED and is viewing a Department*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.5. Manage Policies

1.4.5.1.Create New Policy

Feature name	***CreatePolicy***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Create Policy option* *2. RED displays a form asking for fields related to Policy to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the new Policy. RED redirects the Administrator to the ViewPolicyInfo page of the new Policy.*
Entry condition	*Administrator is logged into RED.*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.5.2.View Existing Policy

Feature name	***ViewPolicyInfo includes <<CreatePolicy>> , <<UpdatePolicy>> & <<DeletePolicy>>***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects a particular Policy.* *2. RED loads a new page that includes complete details of the Policy and Create, Update and Delete buttons.*
Entry condition	*Administrator is logged into RED and selects a particular Policy to view the details*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.5.3.Update Policy

Feature name	***UpdatePolicy***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Update Policy option* *2. RED displays a form asking for fields related to Policy to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the Policy. RED redirects the Administrator to the ViewPolicyInfo page of the updated Policy.*
Entry condition	*Administrator is logged into RED and is viewing a Policy*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.5.4. Delete Policy

Feature name	***DeletePolicy***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Delete Policy option* *2. RED prompts for confirmation* *3. Administrator confirms to delete the Policy* *4. RED inactivates the Policy and displays a success message*
Entry condition	*Administrator is logged into RED and is viewing a Policy*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

1.4.6. Manage Programs

1.4.6.1. Create New Program

Feature name	***CreateProgram***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Create Program option* *2. RED displays a form asking for fields related to Program to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the new Program. RED redirects the Administrator to the ViewProgramInfo page of the new Program.*
Entry condition	*Administrator is logged into RED.*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.6.2.View Existing Program

Feature name	***ViewProgramInfo includes <<CreateProgram>> , <<UpdateProgram>> & <<DeleteProgram>>***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects a particular Program.* *2. RED loads a new page that includes complete details of the Program and Create, Update and Delete buttons.*
Entry condition	*Administrator is logged into RED and selects a particular Program to view the details*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.6.3.Update Program

Feature name	***UpdateProgram***
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Update Program option* *2. RED displays a form asking for fields related to Program to be filled* *3. Administrator fills the form and submits* *4 RED validates the form and saves the Program. RED redirects the Administrator to the ViewProgramInfo page of the updated Program.*
Entry condition	*Administrator is logged into RED and is viewing a Program*
Exit condition	*Administrator selects any other tab.*
Exception	*Invalid entries should be re-entered.*
Quality requirements	*At any point, if something is unclear to User, RED should provide options to help him understand*

1.4.6.4. Delete Program

Feature name	**DeleteProgram**
Participating actors	*Administrator*
Flow of events	*1. Administrator selects Delete Program option* *2. RED prompts for confirmation* *3. Administrator confirms to delete the Program* *4. RED inactivates the Program and displays a success message*
Entry condition	*Administrator is logged into RED and is viewing a Program*
Exit condition	*Administrator selects any other tab.*
Exception	*None*
Quality requirements	*At any point, if something is unclear to Administrator, RED should provide options to help him understand*

2. Non-functional Requirements

2.1. Performance Requirements

The system should adhere to the following performance criteria:

i. User should receive feedback within 3 seconds of every action performed.
ii. Multiple users should be able to use the application at the same time.

2.2. Security and Safety Requirements

i. All data records that RED manipulates should be backed up daily.
ii. Access to all actions should be regulated by roles and permissions.
iii. Data privacy should be enforced.

2.3. Maintainability Requirements

The system should be well modularized with standard development models and the source code should be well documented.

2.4. Scalability Requirements

The number of users that can use the system at any given point should be scalable by adding in more system resources like faster or distributed hardware.

2.5. Software Quality Attributes

The software quality attributes are listed in preference order:

i. Usability
   Users should be able to use the system without any formal training and should conform to standard system usage practices.
ii. Portability
iii. Reliability
   All system operations should be unambiguous and provide user with prompt feedback.
iv. Maintainability
   All system modules should be loosely coupled.
v. Flexibility
vi. Security
vii. Scalability
viii. Testability

2.6. Business Rules

The following business rules may be enforced

- Restriction of user's right:

  The type of the user who could operate certain functions has been defined in the above features. Otherwise, the user is not allowed to perform the function.

Normally, students, faculty, or administrators could use the system except for the following cases:

- Legal procedures to update or to back up the system
- Support:

  The university could provide a help desk (the information for help desk is included in the user manual).

3. Other Requirements

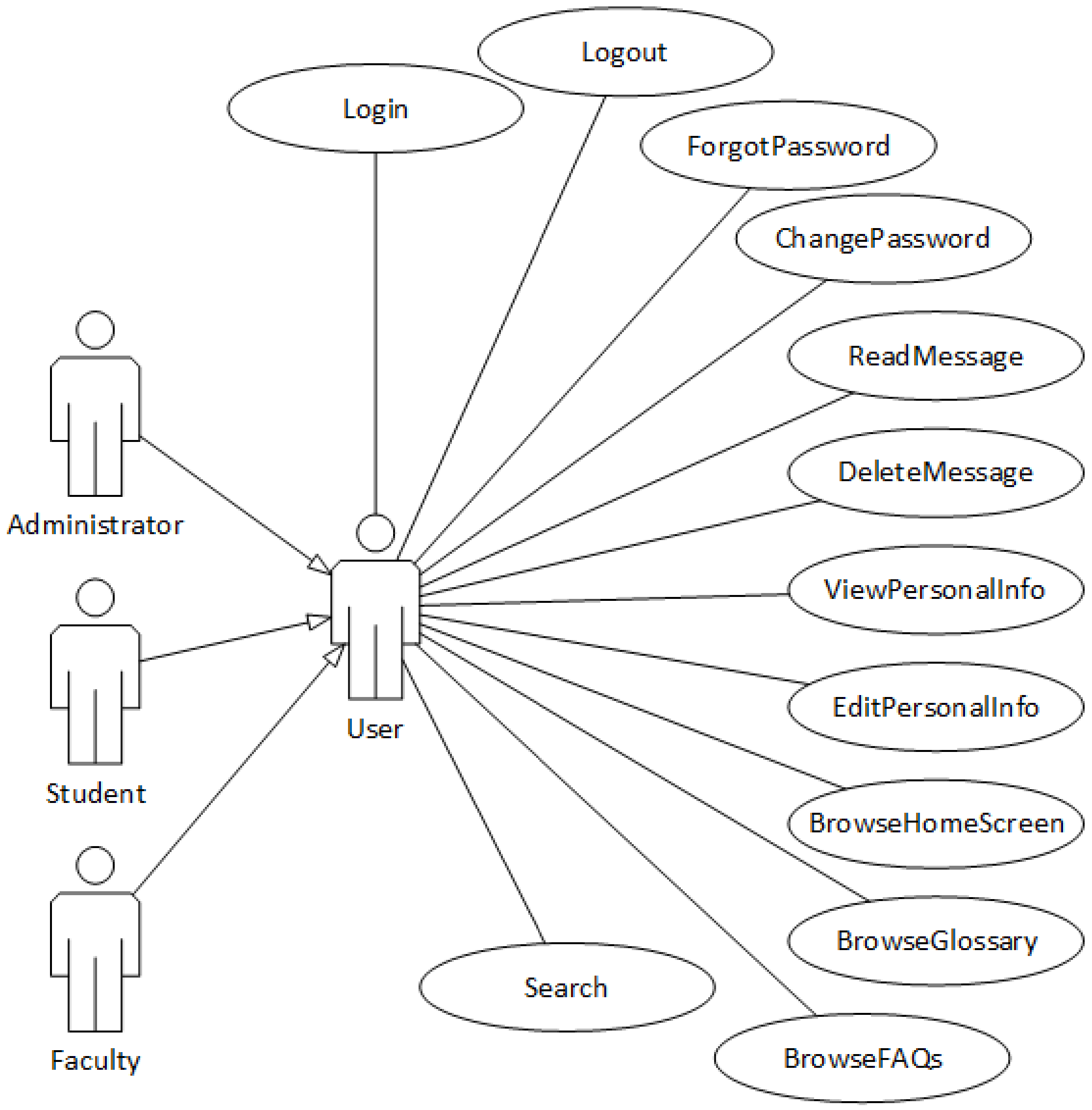
3.1. Internationalization

The application should be localizable and the user should have the option to change the locale from within the application. The application should remember the user's choice and should load with the last selected locale.

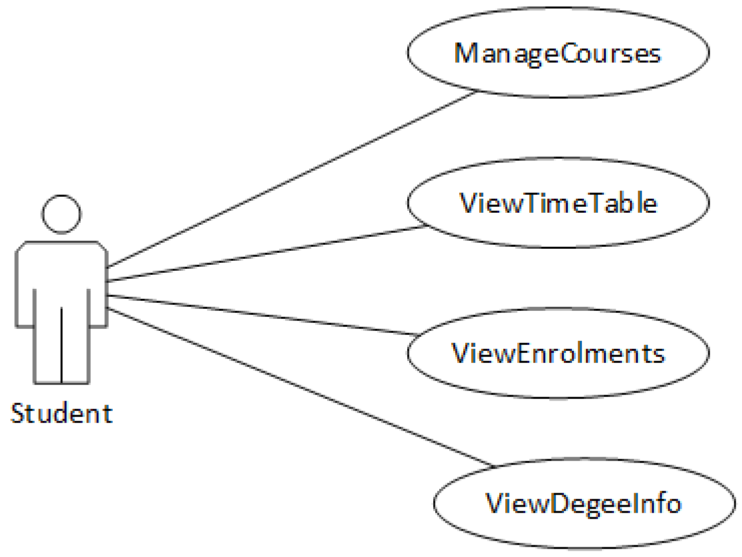
Appendix A – Models

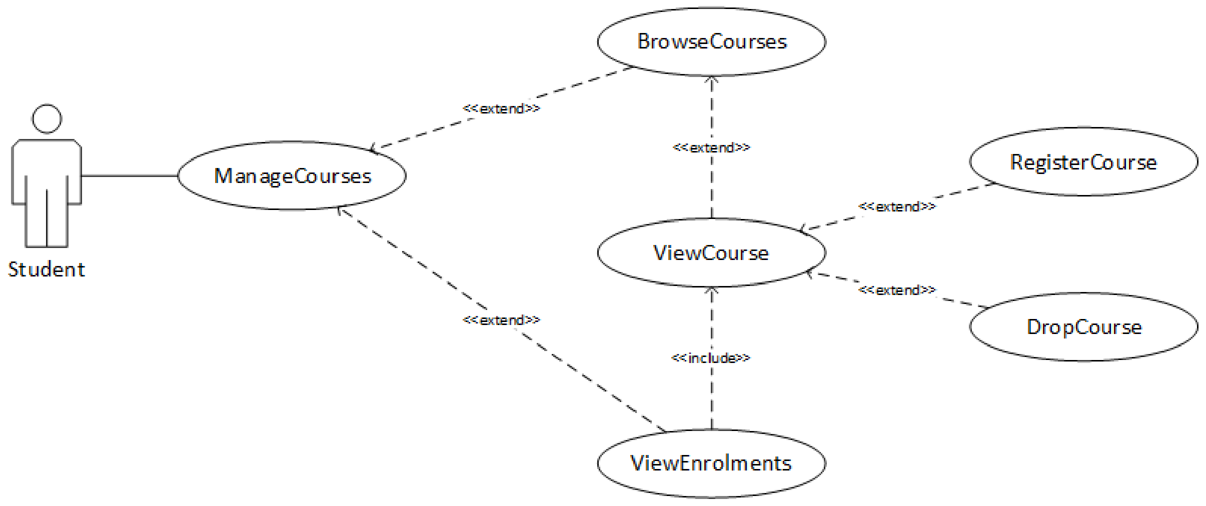
i. Use Case Diagrams

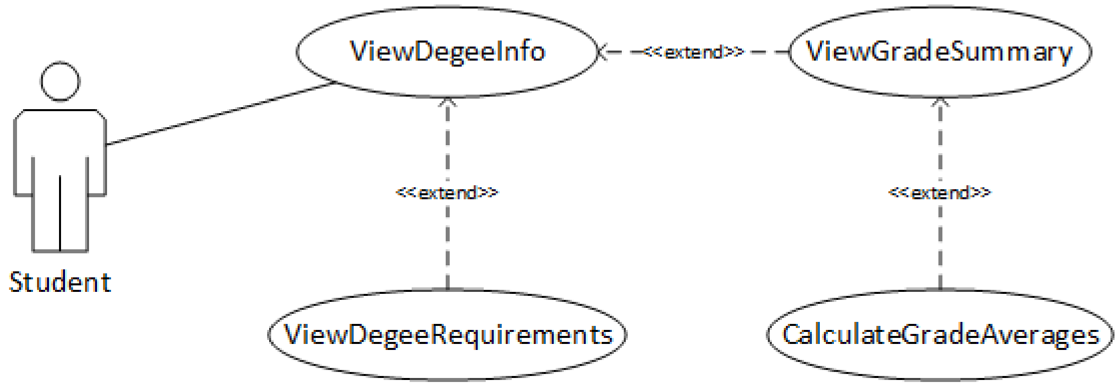
a. Every User



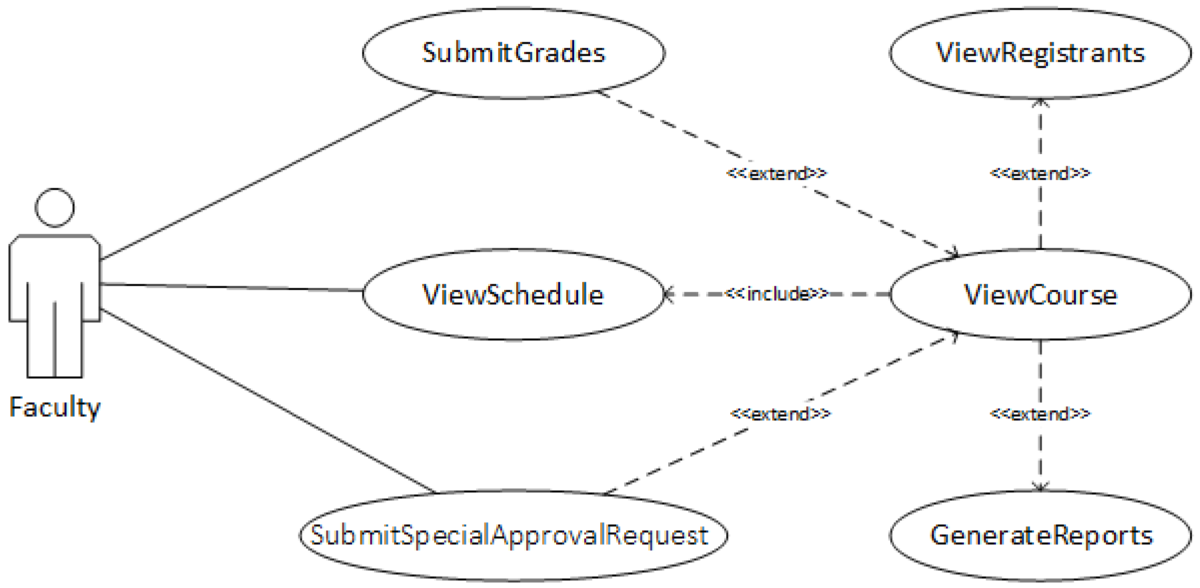
b. Student


ManageCourses
ViewTimeTable
ViewEnrolments
ViewDegeeInfo
Student

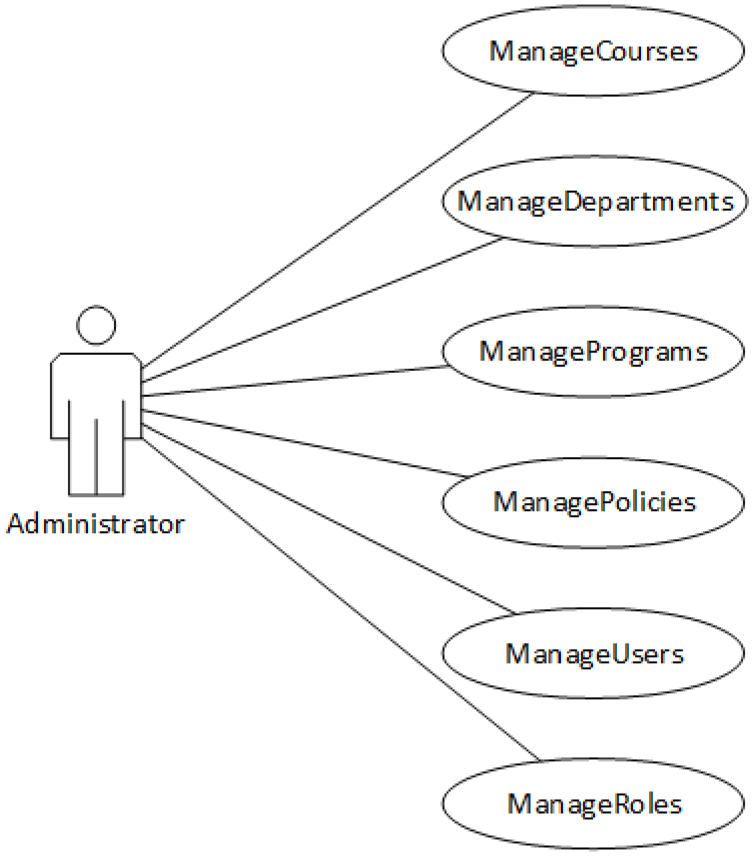

BrowseCourses
<<extend>>
<<extend>>
RegisterCourse
Student
ManageCourses
<<extend>>
ViewCourse
<<extend>>
DropCourse
<<extend>>
<<include>>
ViewEnrolments

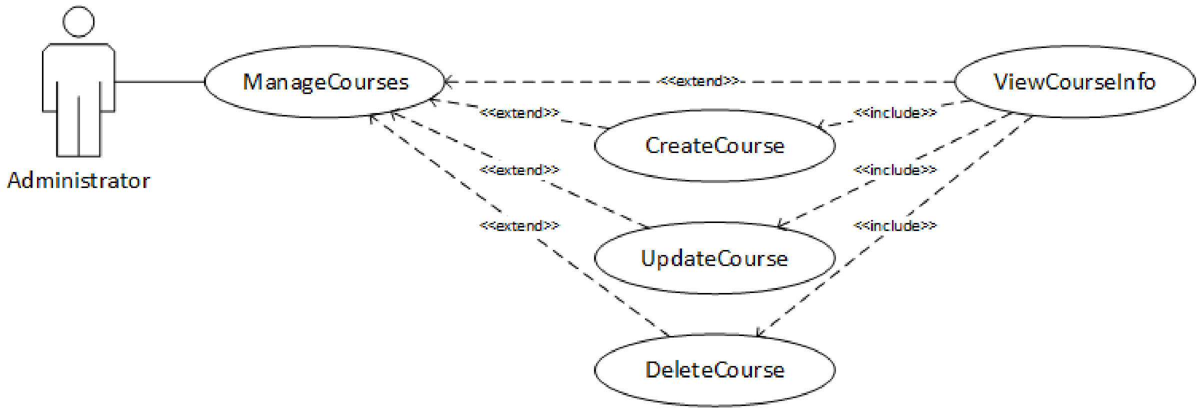

ViewDegeeInfo
<<extend>>
ViewGradeSummary
Student
<<extend>>
<<extend>>
ViewDegeeRequirements
CalculateGradeAverages

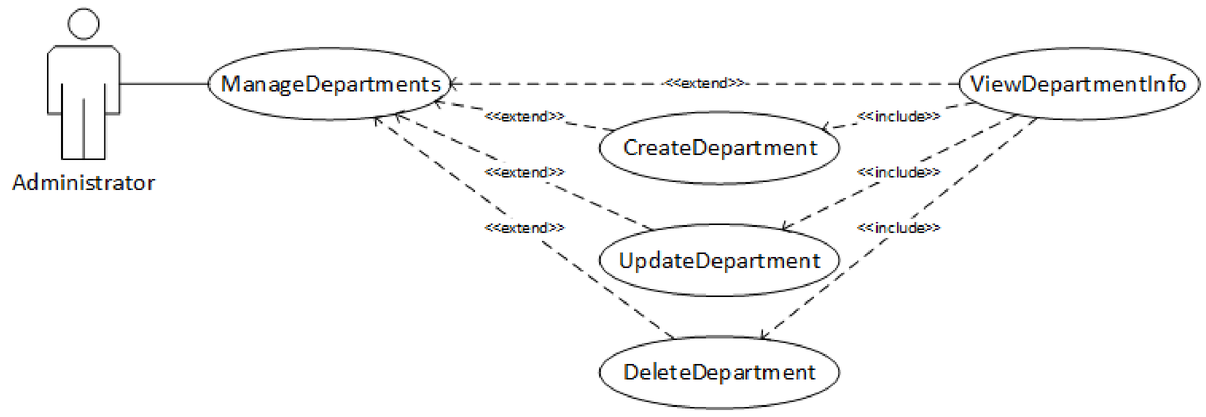
c. Faculty

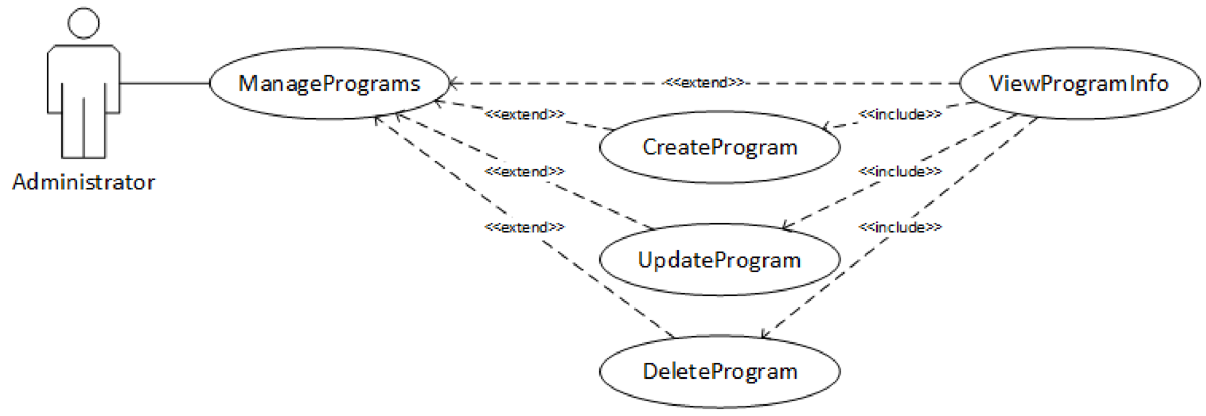

SubmitGrades
ViewRegistrants
Faculty
ViewSchedule
ViewCourse
SubmitSpecialApprovalRequest
GenerateReports
<<extend>>
<<extend>>
<<include>>
<<extend>>
<<extend>>

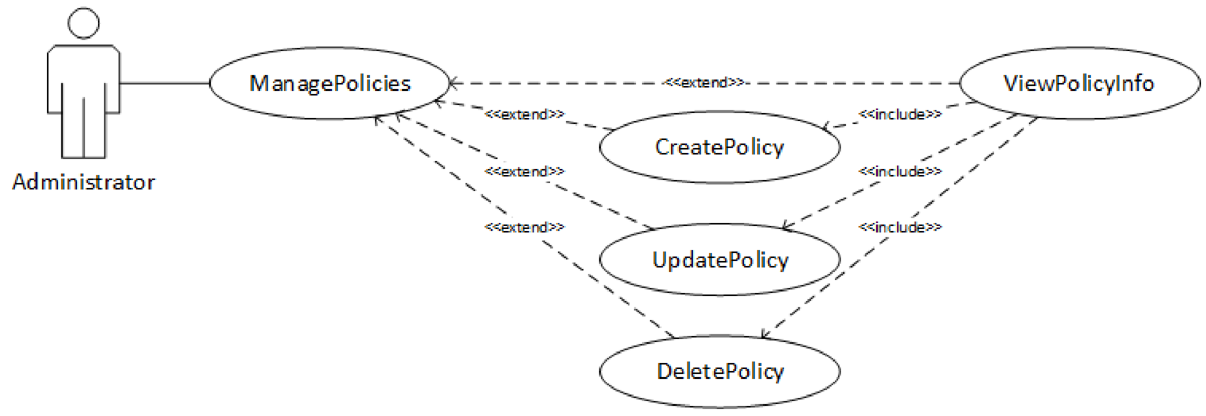
d. Administrator

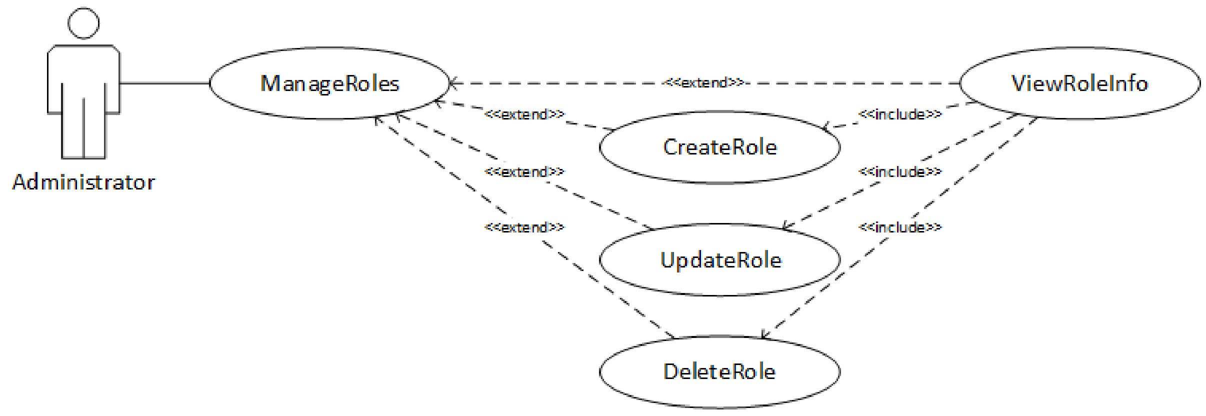




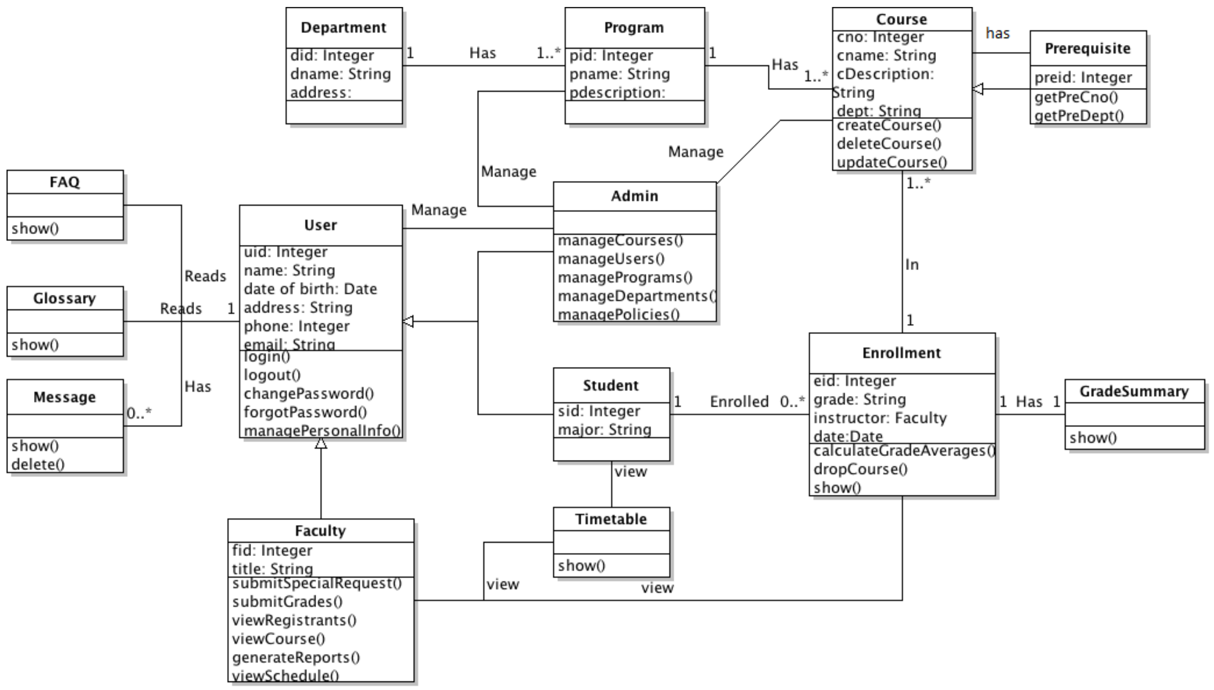

Administrator
ManageDepartments
ViewDepartmentInfo
CreateDepartment
UpdateDepartment
DeleteDepartment
<<extend>>
<<include>>


Administrator
ManagePrograms
ViewProgramInfo
CreateProgram
UpdateProgram
DeleteProgram
<<extend>>
<<include>>


Administrator
ManagePolicies
ViewPolicyInfo
CreatePolicy
UpdatePolicy
DeletePolicy
<<extend>>
<<include>>

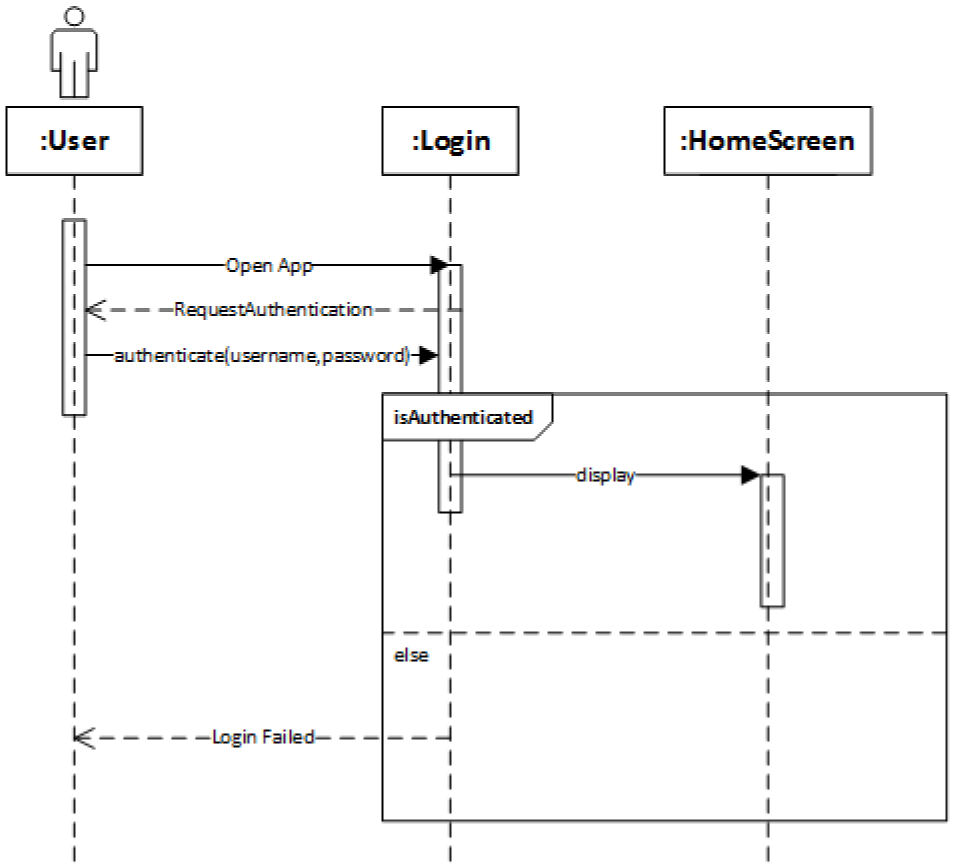

Administrator
ManageRoles
ViewRoleInfo
CreateRole
UpdateRole
DeleteRole
<<extend>>
<<include>>

ii. Class Diagram

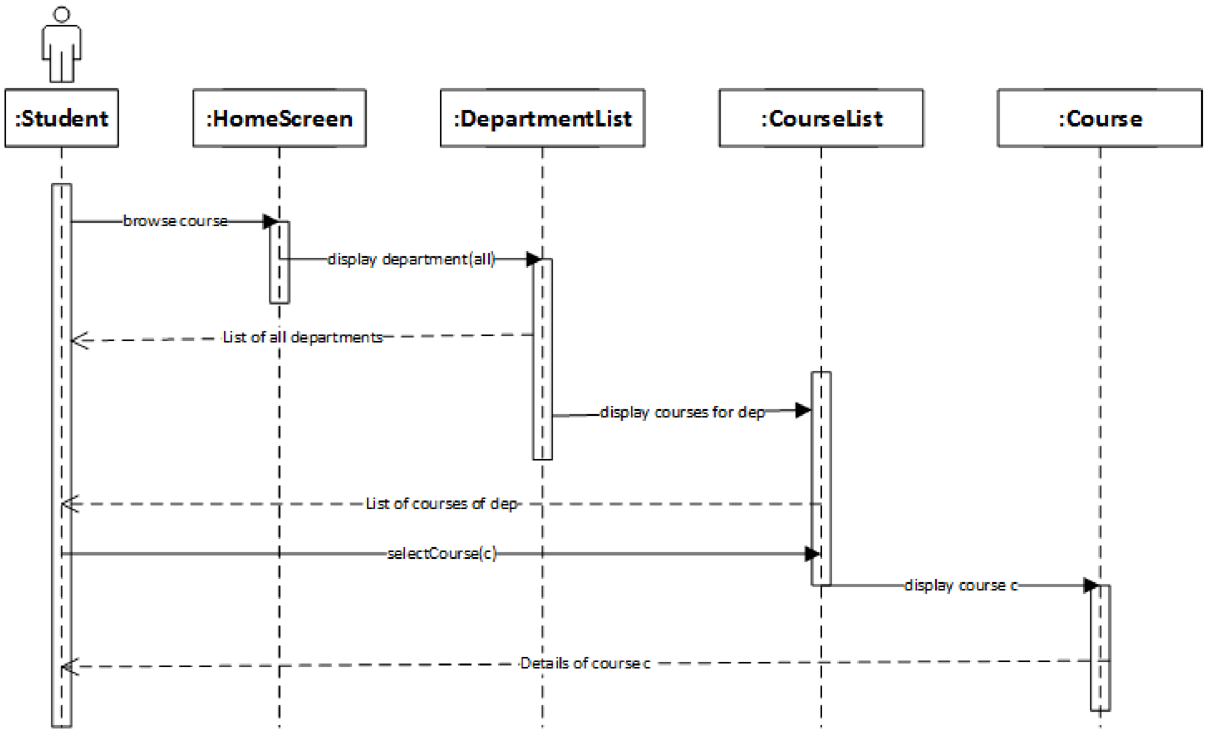


iii. Sequence Diagrams

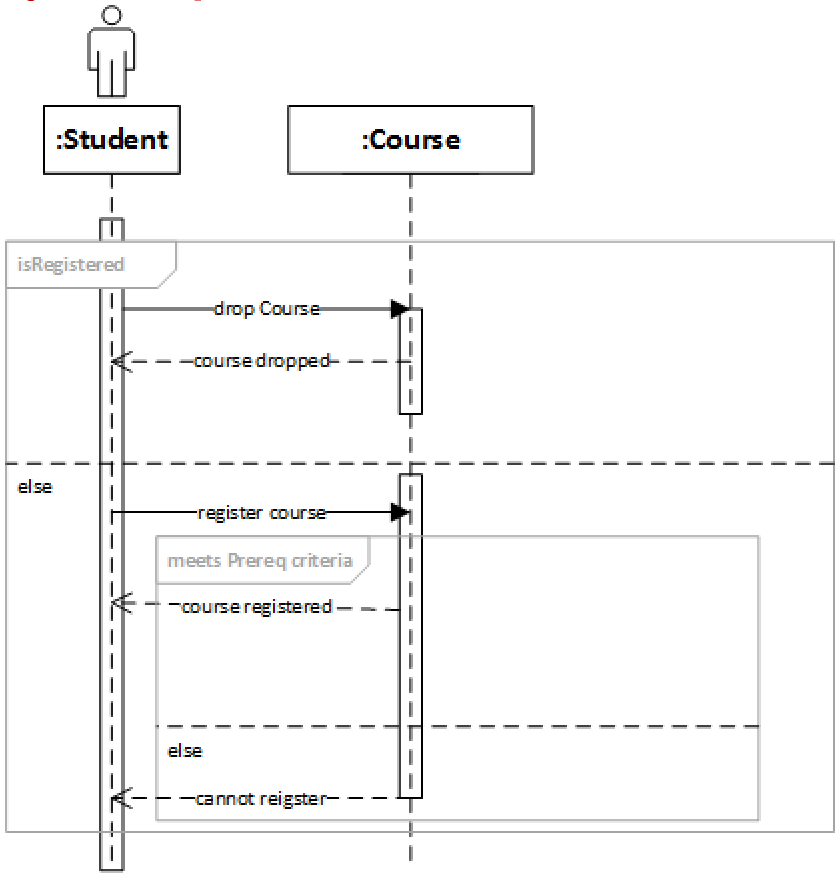
1. Login



2. Browse Courses



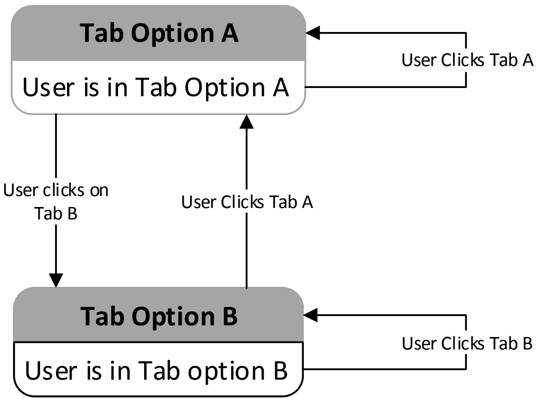
3. Register or drop a course



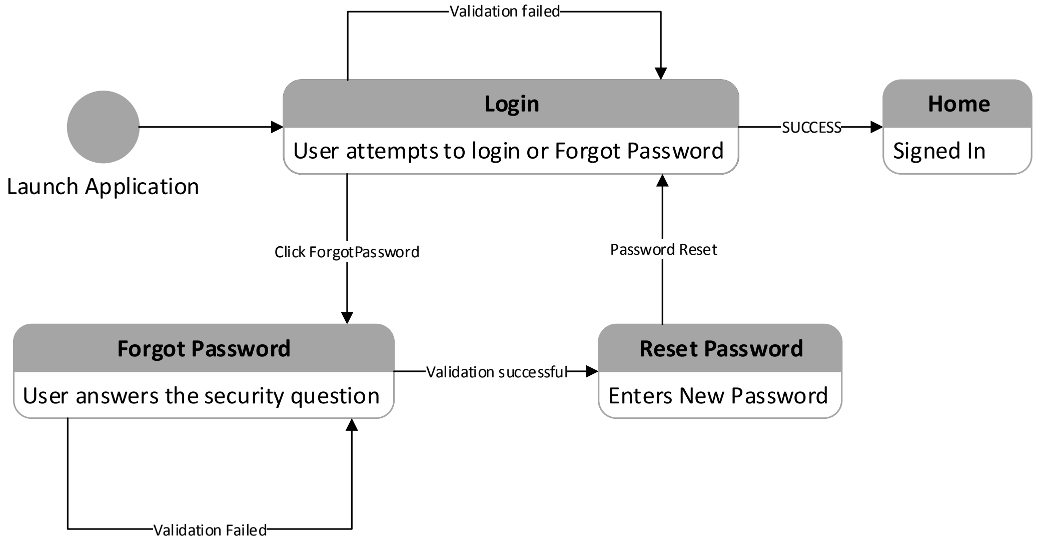
iv. State Diagrams

a. All Users

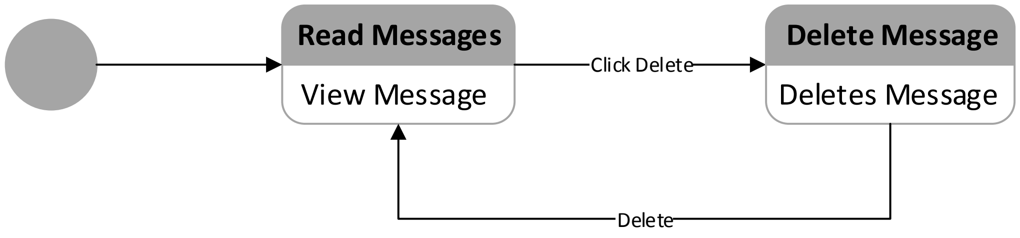
i. General Dashboard Operation



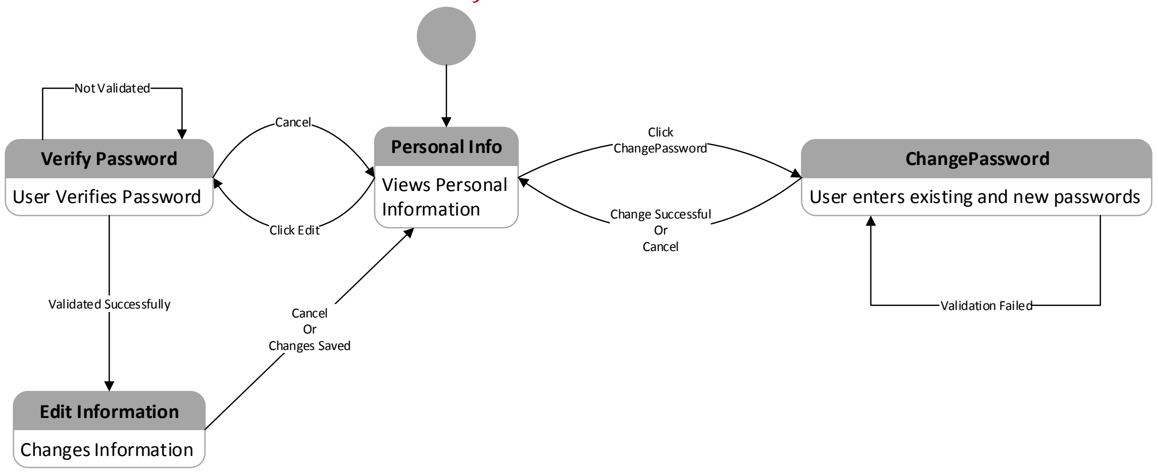
ii. Login/Forgot Password



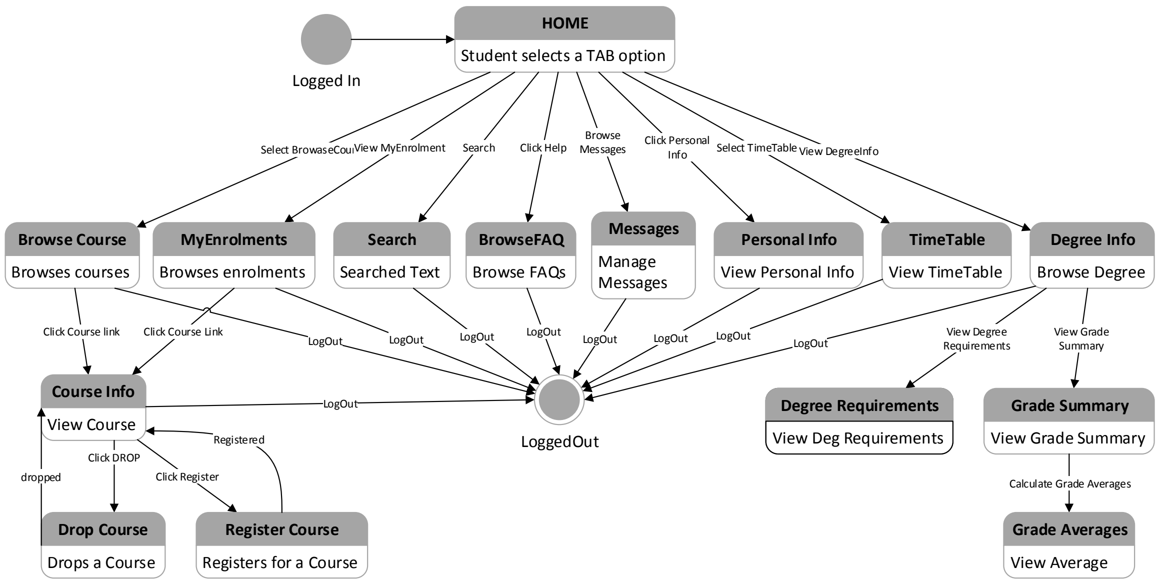
iii. Message Passing



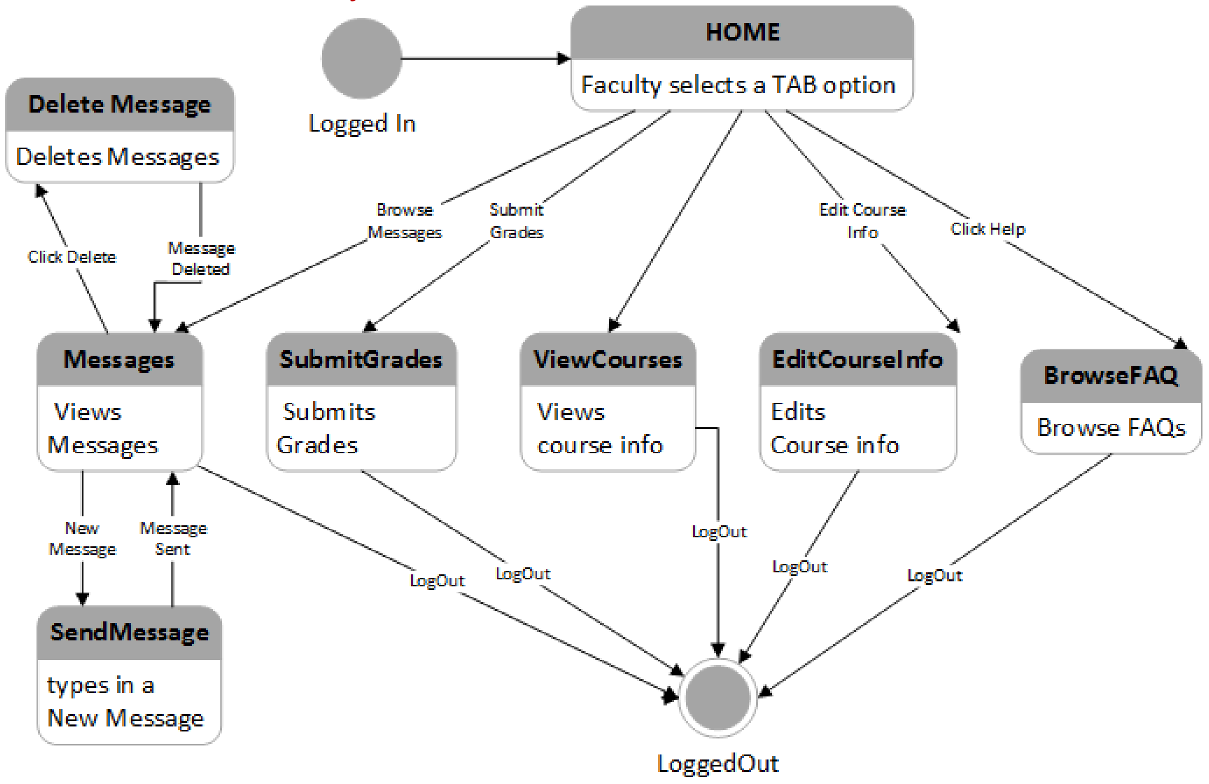
iv. Personal Information



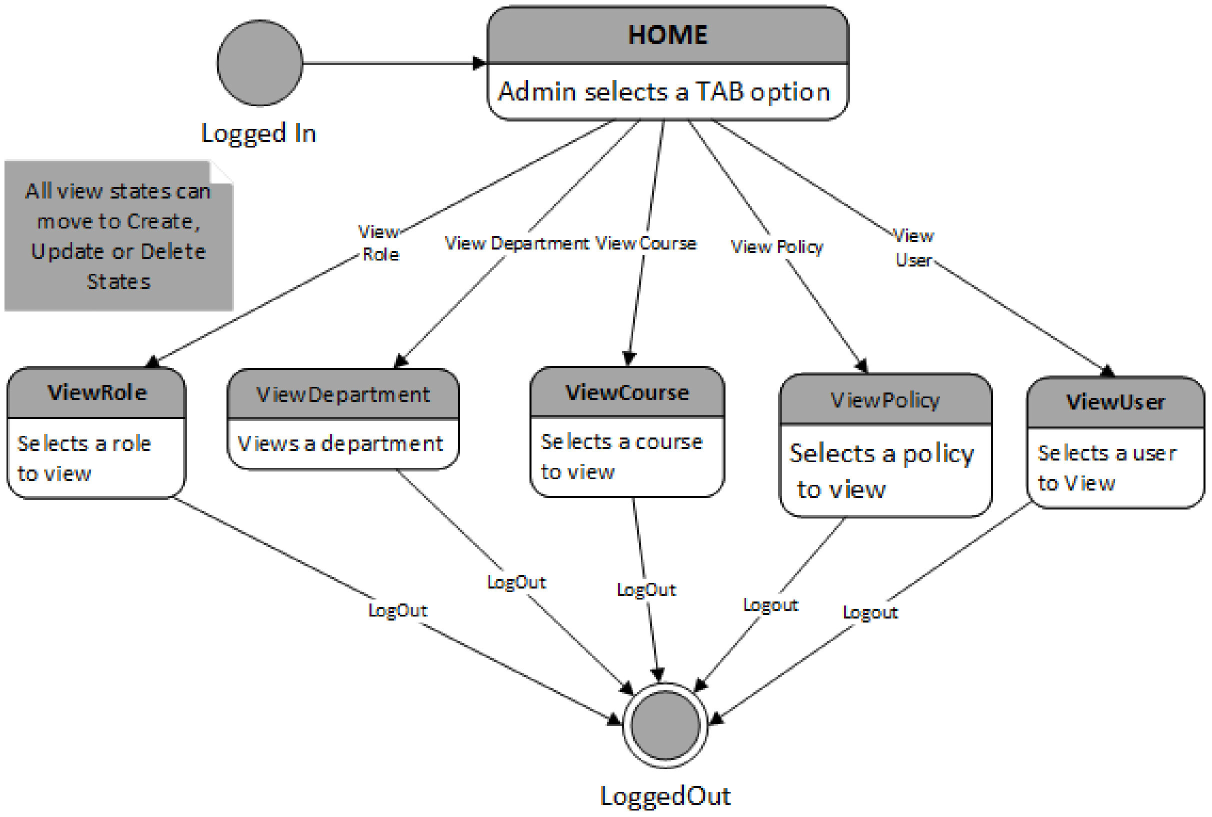
b. Student



c. Faculty



d. Administrator



SYSTEM DESIGN DOCUMENT

1. Introduction

The purpose of this document is to describe all the design goals in implementation of the student enrolment system (codename: RED) created for a small polytechnic-university. The purpose of this document is to provide a guideline to all the developers at Absolution Development Group (ADG) to streamline and standardize the design process.

1.1. Purpose of the System

RED should be able to carry on after the pre-enrolment activities including officially becoming a user member at the university which in turns allows the user to have access privilege to the system. RED may only be accessed by the authorized Users and each user-type is only allowed to participate in the related enrolment activities as defined later in this document. The design goals described here follow the detail included in the requirement elicitation and analysis document prepared originally for this application.

1.2. Design Goals

The following is the breakdown of the design goals based on requirements:

1.2.1Performance Requirements

- User should receive feedback within 3 seconds of every action performed.
- Multiple users should be able to use the application at the same time.
- The system should be able to store large amounts of data. The database shall offer optimal query speed with accurate results.

1.2.2 Security and Safety Requirements

- All data records that RED manipulates should be backed up daily.
- Access to all actions should be regulated by roles and permissions.
- Data privacy should be enforced.

1.2.3. Maintainability Requirements

- The system should be well modularized with standard development models.
- The source code should be well documented.
- The mapping from the code to specific requirements should be clear to other developers and follow standard conventions for each component in the system. This requirement should be followed throughout the whole design and implementation process.

1.2.4. Scalability Requirements

- The number of users that can use the system at any given point should be scalable by adding in more system resources like faster or more distributed hardware.
- The information stored in the database may increase overtime as new information is added such as new users and courses.
- The database scheme has to allow for addition of new features.

1.2.5 Software Quality Attributes

- **Usability:** Users should be able to use the system without any formal training and should conform to standard system usage practices (e.g. GUI is clear and unambiguous to all the users aware of basic computer operations).

- **Portability:** Users could install the system based on Windows or Linux Operating Systems.
- **Development cost:** To reduce the cost, all the tools and database servers used during the development are expected to be open source.

1.2.6 Internationalization

The application should be localizable and the user should have the option to change the locale from within the application. The application should remember the user's choice and should load with the last selected locale.

2. Market Research

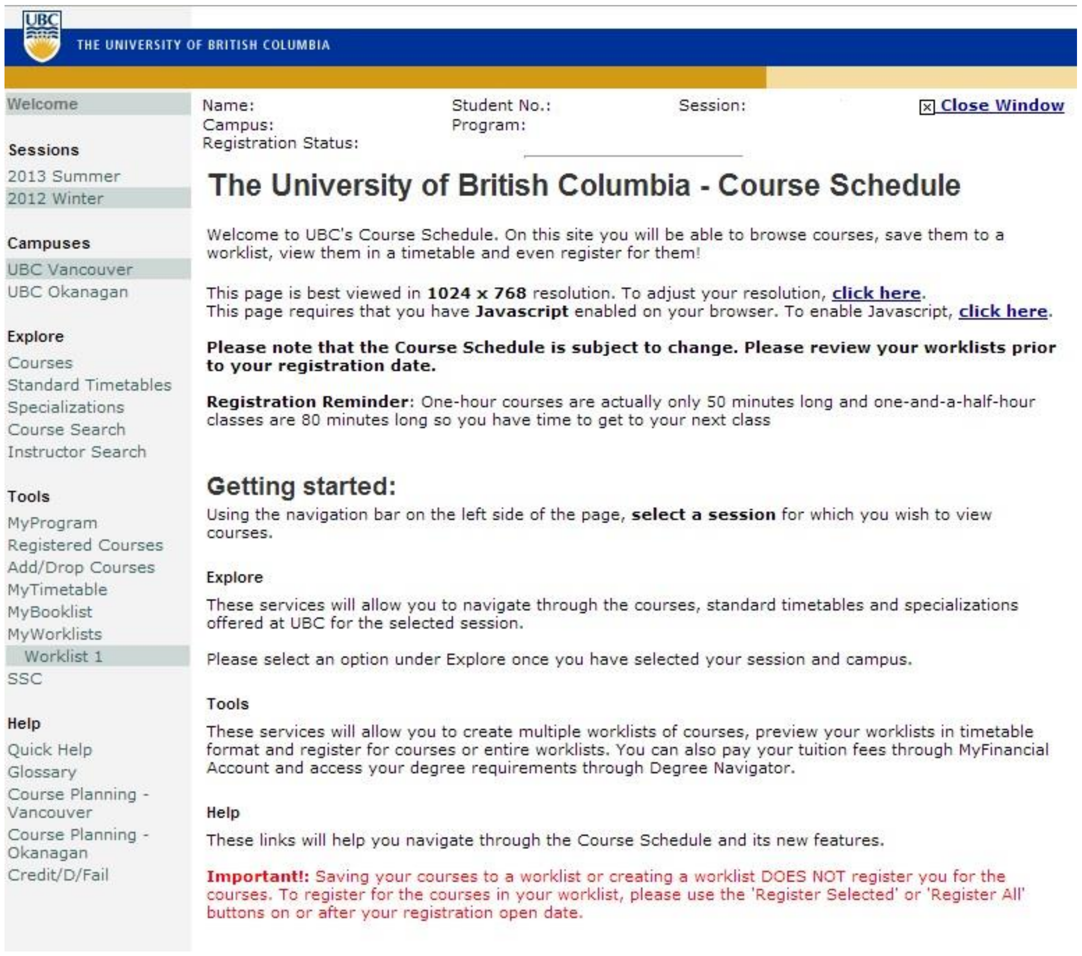
2.1. University of British Columbia Online Student Services System

The current enrolment system of University of British Columbia is a web-based application which is integrated as a part of Student Service Centre (SSC) website. The user has to have the Campus-Wide Login (CWL) account created. Once he clicks the login button, it will redirect to the CWL authentication website to authenticate the account. After a successful authentication, user will be redirect to the Student Service Centre home page. From there user can navigate to a variety of services according to his account type and privileges set up.

A student user can perform the following actions:

- View time table
- Browse course information
- Search for instructor
- Add/Drop a course
- Read through Glossary

The current software uses client/server with central database architectural style. User can use different kind of clients, in this case web browsers, to access to the Student Service Centre web servers through CWL authentication server. The SSC server will process the request and query the database if necessary. It will then return the HTML and JavaScript back to the user's web browser to display the content.


THE UNIVERSITY OF BRITISH COLUMBIA

Welcome

Sessions
2013 Summer
2012 Winter

Campuses
UBC Vancouver
UBC Okanagan

Explore
Courses
Standard Timetables
Specializations
Course Search
Instructor Search

Tools
MyProgram
Registered Courses
Add/Drop Courses
MyTimetable
MyBooklist
MyWorklists
Worklist 1
SSC

Help
Quick Help
Glossary
Course Planning - Vancouver
Course Planning - Okanagan
Credit/D/Fail

Name: Student No.: Session: Close Window
Campus: Program:
Registration Status:

The University of British Columbia - Course Schedule

Welcome to UBC's Course Schedule. On this site you will be able to browse courses, save them to a worklist, view them in a timetable and even register for them!

This page is best viewed in **1024 x 768** resolution. To adjust your resolution, click here.
This page requires that you have **Javascript** enabled on your browser. To enable Javascript, click here.

Please note that the Course Schedule is subject to change. Please review your worklists prior to your registration date.

Registration Reminder: One-hour courses are actually only 50 minutes long and one-and-a-half-hour classes are 80 minutes long so you have time to get to your next class

Getting started:

Using the navigation bar on the left side of the page, **select a session** for which you wish to view courses.

Explore

These services will allow you to navigate through the courses, standard timetables and specializations offered at UBC for the selected session.

Please select an option under Explore once you have selected your session and campus.

Tools

These services will allow you to create multiple worklists of courses, preview your worklists in timetable format and register for courses or entire worklists. You can also pay your tuition fees through MyFinancial Account and access your degree requirements through Degree Navigator.

Help

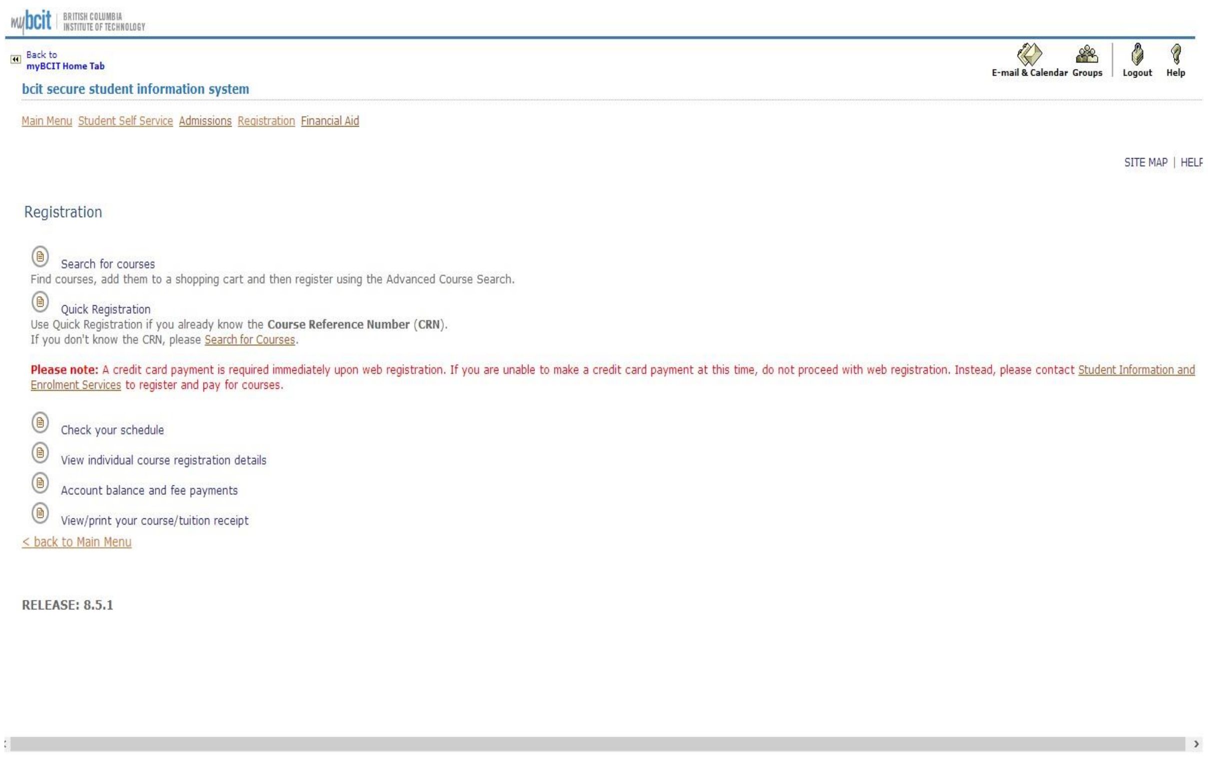
These links will help you navigate through the Course Schedule and its new features.

Important!: Saving your courses to a worklist or creating a worklist DOES NOT register you for the courses. To register for the courses in your worklist, please use the 'Register Selected' or 'Register All' buttons on or after your registration open date.

Figure 1 – UBC Student Registration Service

2.2. BC Institute of Technology (BCIT) Student Registration System

BCIT uses a similar web-based application to offer registration services. Students have to log in successfully using a valid account and there they can click on student self-registration feature among many other options. The BCIT system offers very similar features to that of the UBC in a less user-friendly layout. This system also takes advantage of client/server architecture with a central database.



Registration

Search for courses
Find courses, add them to a shopping cart and then register using the Advanced Course Search.

Quick Registration
Use Quick Registration if you already know the **Course Reference Number** (**CRN**).
If you don't know the CRN, please Search for Courses.

Please note: A credit card payment is required immediately upon web registration. If you are unable to make a credit card payment at this time, do not proceed with web registration. Instead, please contact Student Information and Enrolment Services to register and pay for courses.

Check your schedule

View individual course registration details

Account balance and fee payments

View/print your course/tuition receipt

< back to Main Menu

RELEASE: 8.5.1

Figure 2 – BCIT Student Enrolment Service

2.3. Interview with Bhavani Veylan – UBC Engineering Advisor

Ms. Bhavani Veylan at the Faculty of Applied Sciences at UBC guided our designers through a comprehensive tour of the administrator end of UBC's enrolment software. The first thing noticed here is that UBC offers a completely different user interface that is very rugged and less user-friendly than the one which students interact with. This application offers a variety of backend features that is not accessible to students such as access to information of all students in the engineering faculty including their records, personal information, history of communication, request log, status of their applications and more. The application also allows the Admin to generate many types of reports and lists such as: email lists, student performance reports.

3. Proposed Software Architecture

The following provides a description of the structure and implementation of the application that has been designed. The document continues with an overview of the system followed by more detailed information about the structure and the functionality of the application.

3.1. Overview

The RED Enrollment System consists of three major parts: Database, API and GUI. The persistent information is stored on a cloud server, the GUI handles user interactions and the API communicates between the two with appropriate functionalities. The application allows for three types of users while storing their access information in coupled tables in addition to storing all user passwords in an encrypted format.

User's interaction begins with running the application and authenticating themselves and ends with either closing the application or logging out. Every interaction between the user and the GUI is handled through message passing from the GUI thread to the API thread which in turn is dealt with by the controllers and models and communicated back and forth to and from the database.

3.2. Design Goals

- **Usability**:
  The goal here is for the interface to be user-friendly, intuitive and easy to use. The user does not need prior learning to work with the program therefore the application features need to be self-descriptive and issue informative error or warning messages that make it easy for the user to correct input problems. The buttons are displayed large with clear text and the main features are always displayed on the side for you user to be able to access them at any time.
- **Extensibility**:
  The system is designed to allow teams to create modules that can be added to the system in the future and deployed on the client environment. This is because the system is designed using separate object oriented modules and interfaces based on the Model-View-Controller (MVC) architecture.
- **Maintainability**:
  The system is implemented in such way to reduce training requirement for future developers as well as reduce preventive and corrective maintenance by issue and rationale tracking. Also the system does not require any particular calibration, alignment, or adjustment therefore it can be run in any of the specified environments (Windows 7 or higher here) with no significant setup over heard.
- **Performance**:
  The approach in designing this system is to identify constraints, determine features and identify the load (types of users and their usage requirements) and design and implement against these attributes. The major performance criterion here is the system's responsiveness; every feature has to be loaded in less than 3 seconds while displaying correct results.

3.3. Subsystem Decomposition

The application is divided into the following three major components:

(More detail in section 4.1.)

- **Database:** The database contains all tables based on the ER diagrams in Section 4.2.
- **Model-Controller:** The Controller is set of all classes written in Java that read user inputs and interaction and relay that information to the Models. The Models are Java classes that link the Controllers to the Database.
- **View/GUI:** The View which is also referred to as the Graphical User Interface (GUI) is part of the application that the user interacts with.

3.4. Persistent Data Management

The data stored in the database consists of two types: the Persistent Data and the Dynamic Data. The dynamic data refers to the information that is present in real-time during user's interaction with the system and is not stored anywhere in the database once the user is logged out or the application is closed. This type of data includes but is not limited to variables such as pages opened and buttons pressed.

The persistent data refers to the type of information that is stored in the database for future reference and is accessible even after the program is closed. Data such as course information, user information, passwords and enrollment history are examples of the persistent data.

The database is created in SQL using MySQL and in this application it is stored on an Amazon cloud server. The schemas of the database are modelled following the ER diagram presented in Section 4.2.

3.5. Access Control and Security

The enrollment system allows for access of three main types of user: Student, Faculty Member and Administrator. Each user has access to a specific set of features of the application based on their account type. For instance a student can register for classes and browse course information, the faculty can request to add a course or modify the description of a course they are teaching as well as view some information about the students that are enrolled in their class. An administrator has access to all of the above features as well as the ability to approve of requests for adding a course or special registration requests, adding/removing of a user as well as courses. This access information is stored in an access table in the database separate from the user table. Every user is assigned a typeID which in turn is stored with certain accessType in the Access table. In a separate table these accessTypes are linked with set of features that particular accessType is assigned to. This allows for modifiability in the future if for instance a new feature is added or a new accessType is defined.

Another important security feature is the authentication of users. This application is using the SHA-1 and DES encryption type to store the passwords of the users as part of the persistent data. All passwords are encrypted for storage.

3.6. Global Software Control

The application consists of two main threads that run concurrently: the GUI thread and the API thread which consists of the models and controllers. The GUI is always waiting for user input or displaying messages from the controllers while the API thread handles the communication between the GUI, controllers, model and the database. The two threads communicate between them however, since the GUI thread does not have access to the database and any stored information, it cannot alter the state of variables that are used by the API thread. The GUI thread only communicates with the API thread and vice versa, via message passing which allows for less complexity in terms of synchronization.

3.7. Boundary Conditions

The system starts-up once the application is run and the next step for the user to access the program is to login using valid authentication. The exceptions that can happen here are:

- The database server is not accessible: the program will not allow user to login and displays a message informing that the server is down.
- The user's account is suspended: the program will inform the user that their account is currently suspended.
- The user's login information is incorrect: the program will notify the user of authentication failure.

The system shuts-down once the application is closed however, the exit condition of the user here is where the user is logged out which happens in the following events:

- User selects to "log out": the program returns to the login page and notifies the user of a successful log out.
- User closes the application: the program changes the status of the user to logged out value automatically.

APPENDIX B – Class Diagrams

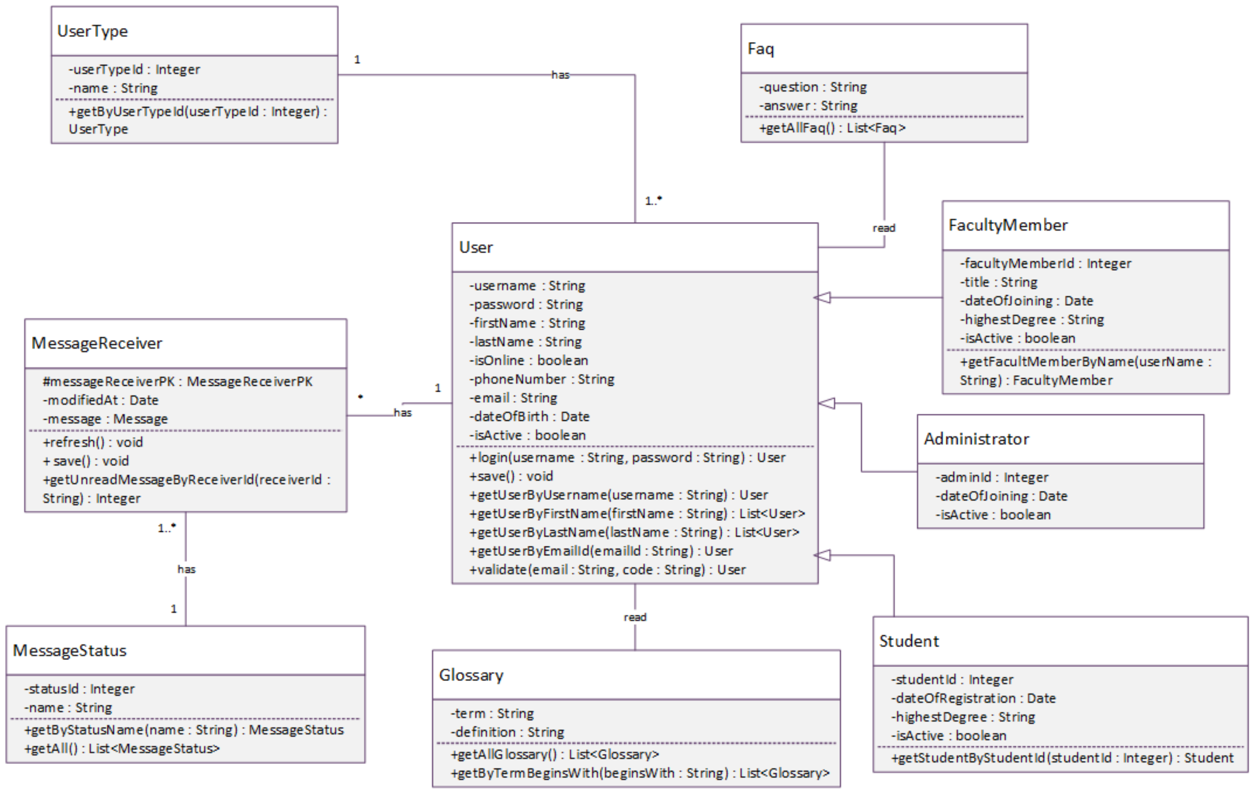

UserType
-userTypeId : Integer
-name : String
+getByUserTypeId(userTypeId : Integer) : UserType
1
has
1..*
Faq
-question : String
-answer : String
+getAllFaq() : List<Faq>
read
FacultyMember
-facultyMemberId : Integer
-title : String
-dateOfJoining : Date
-highestDegree : String
-isActive : boolean
+getFacultMemberByName(userName : String) : FacultyMember
User
-username : String
-password : String
-firstName : String
-lastName : String
-isOnline : boolean
-phoneNumber : String
-email : String
-dateOfBirth : Date
-isActive : boolean
+login(username : String, password : String) : User
+save() : void
+getUserByUsername(username : String) : User
+getUserByFirstName(firstName : String) : List<User>
+getUserByLastName(lastName : String) : List<User>
+getUserByEmailId(emailId : String) : User
+validate(email : String, code : String) : User
MessageReceiver
#messageReceiverPK : MessageReceiverPK
-modifiedAt : Date
-message : Message
+refresh() : void
+ save() : void
+getUnreadMessageByReceiverId(receiverId : String) : Integer
*
has
1
Administrator
-adminId : Integer
-dateOfJoining : Date
-isActive : boolean
1..*
has
1
MessageStatus
-statusId : Integer
-name : String
+getByStatusName(name : String) : MessageStatus
+getAll() : List<MessageStatus>
read
Glossary
-term : String
-definition : String
+getAllGlossary() : List<Glossary>
+getByTermBeginsWith(beginsWith : String) : List<Glossary>
Student
-studentId : Integer
-dateOfRegistration : Date
-highestDegree : String
-isActive : boolean
+getStudentByStudentId(studentId : Integer) : Student

Figure 1 – User Class Diagram

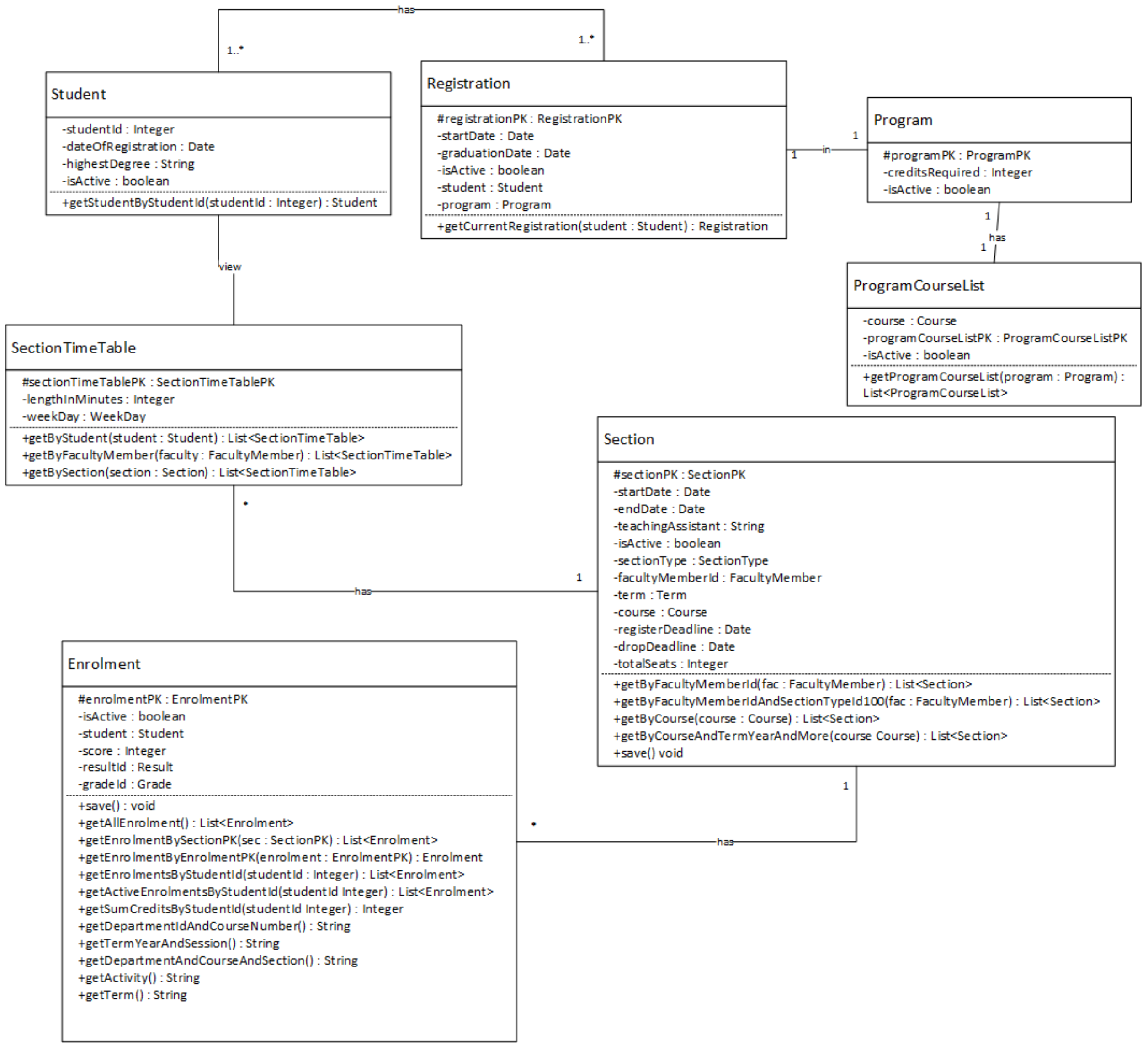

has
1..*
1..*
Student
-studentId : Integer
-dateOfRegistration : Date
-highestDegree : String
-isActive : boolean
+getStudentByStudentId(studentId : Integer) : Student
Registration
#registrationPK : RegistrationPK
-startDate : Date
-graduationDate : Date
-isActive : boolean
-student : Student
-program : Program
+getCurrentRegistration(student : Student) : Registration
1
in
1
Program
#programPK : ProgramPK
-creditsRequired : Integer
-isActive : boolean
1
has
1
ProgramCourseList
-course : Course
-programCourseListPK : ProgramCourseListPK
-isActive : boolean
+getProgramCourseList(program : Program) : List<ProgramCourseList>
view
SectionTimeTable
#sectionTimeTablePK : SectionTimeTablePK
-lengthInMinutes : Integer
-weekDay : WeekDay
+getByStudent(student : Student) : List<SectionTimeTable>
+getByFacultyMember(faculty : FacultyMember) : List<SectionTimeTable>
+getBySection(section : Section) : List<SectionTimeTable>
*
has
1
Section
#sectionPK : SectionPK
-startDate : Date
-endDate : Date
-teachingAssistant : String
-isActive : boolean
-sectionType : SectionType
-facultyMemberId : FacultyMember
-term : Term
-course : Course
-registerDeadline : Date
-dropDeadline : Date
-totalSeats : Integer
+getByFacultyMemberId(fac : FacultyMember) : List<Section>
+getByFacultyMemberIdAndSectionTypeId100(fac : FacultyMember) : List<Section>
+getByCourse(course : Course) : List<Section>
+getByCourseAndTermYearAndMore(course Course) : List<Section>
+save() void
Enrolment
#enrolmentPK : EnrolmentPK
-isActive : boolean
-student : Student
-score : Integer
-resultId : Result
-gradeId : Grade
+save() : void
+getAllEnrolment() : List<Enrolment>
+getEnrolmentBySectionPK(sec : SectionPK) : List<Enrolment>
+getEnrolmentByEnrolmentPK(enrolment : EnrolmentPK) : Enrolment
+getEnrolmentsByStudentId(studentId : Integer) : List<Enrolment>
+getActiveEnrolmentsByStudentId(studentId Integer) : List<Enrolment>
+getSumCreditsByStudentId(studentId Integer) : Integer
+getDepartmentIdAndCourseNumber() : String
+getTermYearAndSession() : String
+getDepartmentAndCourseAndSection() : String
+getActivity() : String
+getTerm() : String
*
has
1

Figure 2 – Student Class Diagram

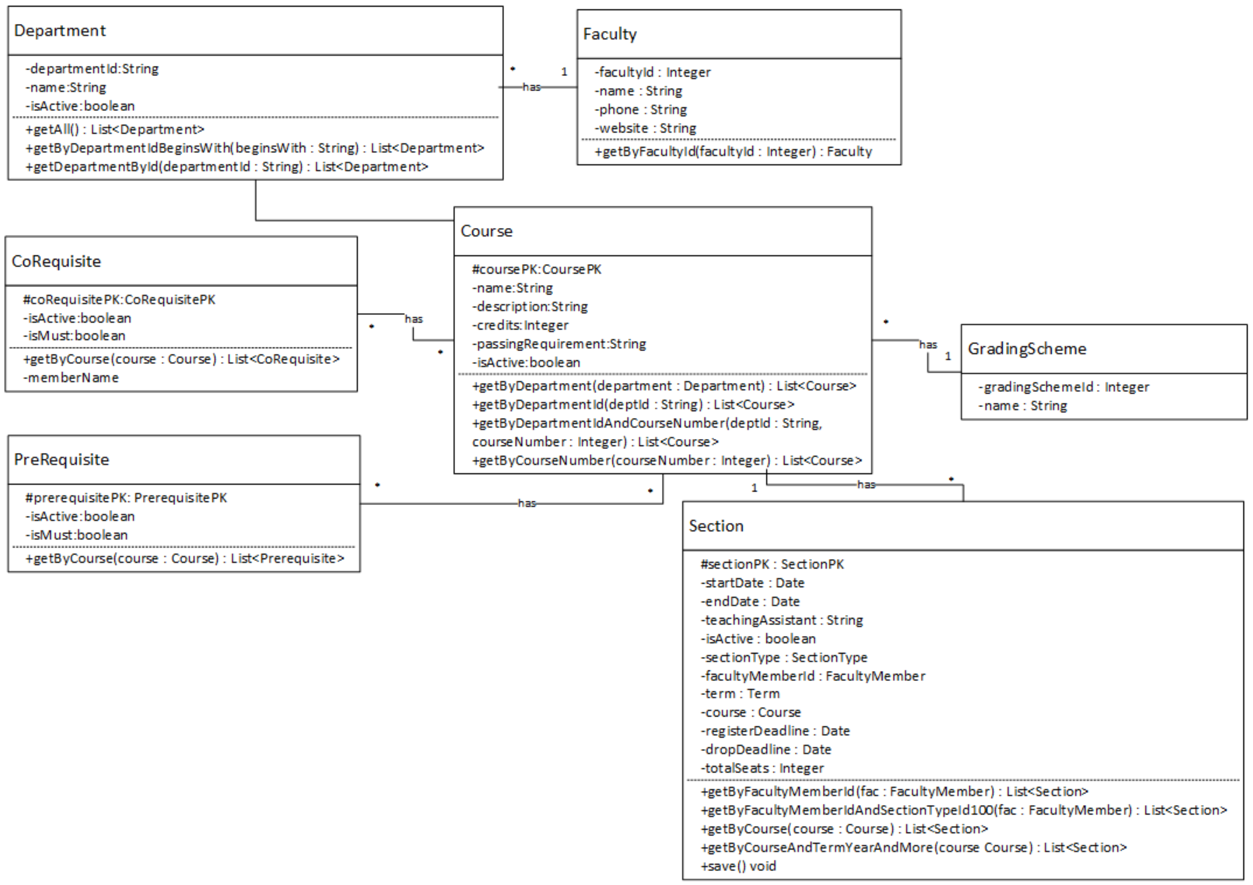

Department
-departmentId:String
-name:String
-isActive:boolean
+getAll() : List<Department>
+getByDepartmentIdBeginsWith(beginsWith : String) : List<Department>
+getDepartmentById(departmentId : String) : List<Department>
Faculty
-facultyId : Integer
-name : String
-phone : String
-website : String
+getByFacultyId(facultyId : Integer) : Faculty
has
*
1
Course
#coursePK:CoursePK
-name:String
-description:String
-credits:Integer
-passingRequirement:String
-isActive:boolean
+getByDepartment(department : Department) : List<Course>
+getByDepartmentId(deptId : String) : List<Course>
+getByDepartmentIdAndCourseNumber(deptId : String, courseNumber : Integer) : List<Course>
+getByCourseNumber(courseNumber : Integer) : List<Course>
CoRequisite
#coRequisitePK:CoRequisitePK
-isActive:boolean
-isMust:boolean
+getByCourse(course : Course) : List<CoRequisite>
-memberName
GradingScheme
-gradingSchemeId : Integer
-name : String
PreRequisite
#prerequisitePK: PrerequisitePK
-isActive:boolean
-isMust:boolean
+getByCourse(course : Course) : List<Prerequisite>
Section
#sectionPK : SectionPK
-startDate : Date
-endDate : Date
-teachingAssistant : String
-isActive : boolean
-sectionType : SectionType
-facultyMemberId : FacultyMember
-term : Term
-course : Course
-registerDeadline : Date
-dropDeadline : Date
-totalSeats : Integer
+getByFacultyMemberId(fac : FacultyMember) : List<Section>
+getByFacultyMemberIdAndSectionTypeId100(fac : FacultyMember) : List<Section>
+getByCourse(course : Course) : List<Section>
+getByCourseAndTermYearAndMore(course Course) : List<Section>
+save() void

Figure 3 – Course Class Diagram

APPENDIX C – Sequence Diagrams

Common Features:

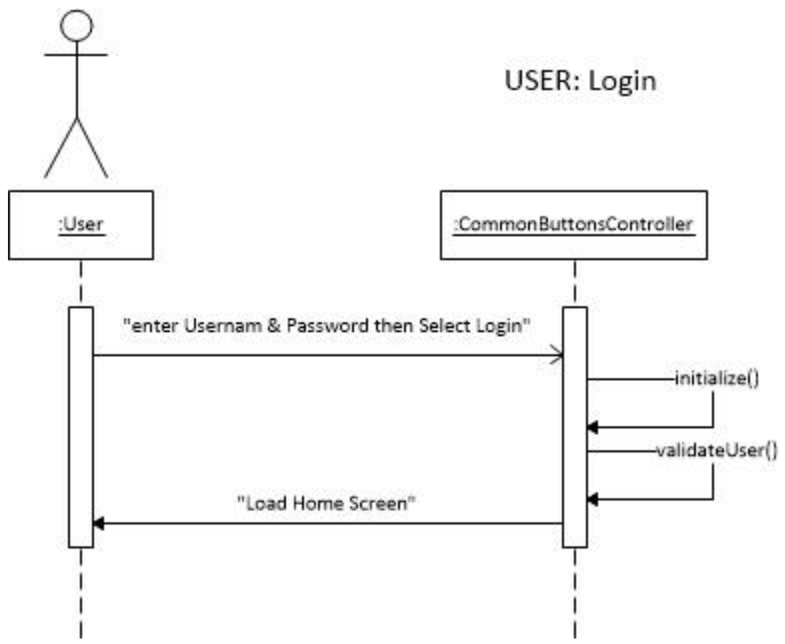


Figure 1 – User Login

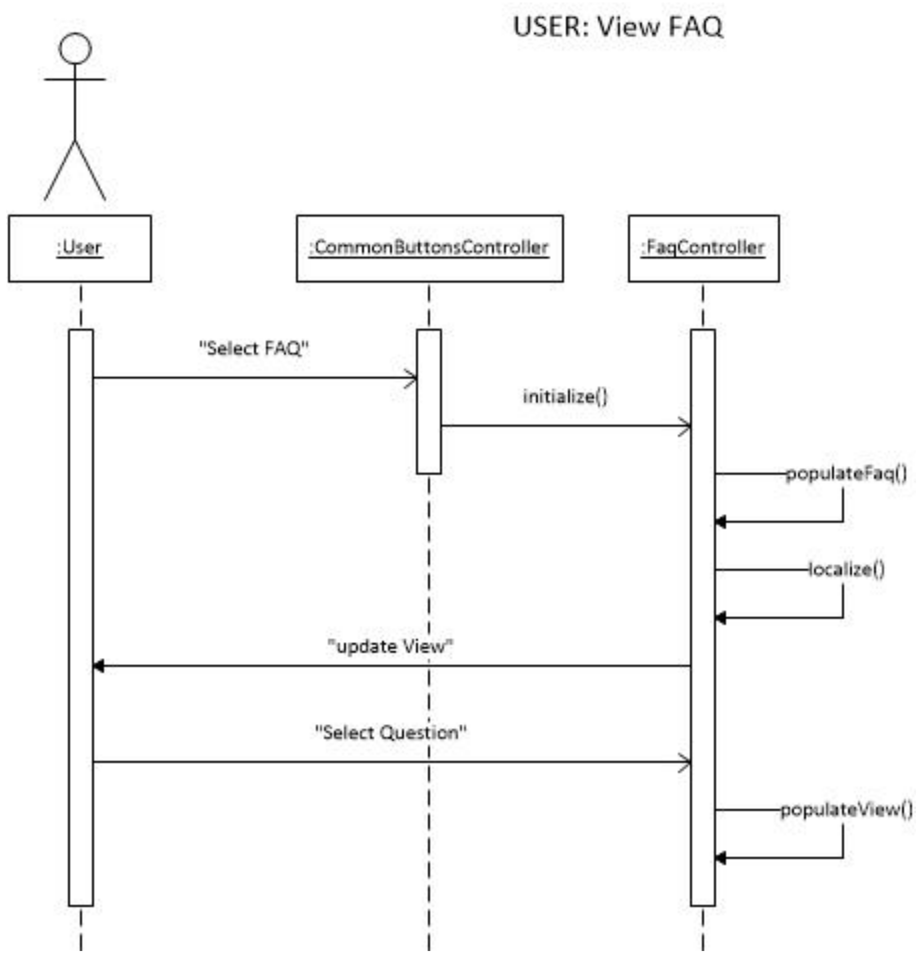


Figure 2 – View FAQ

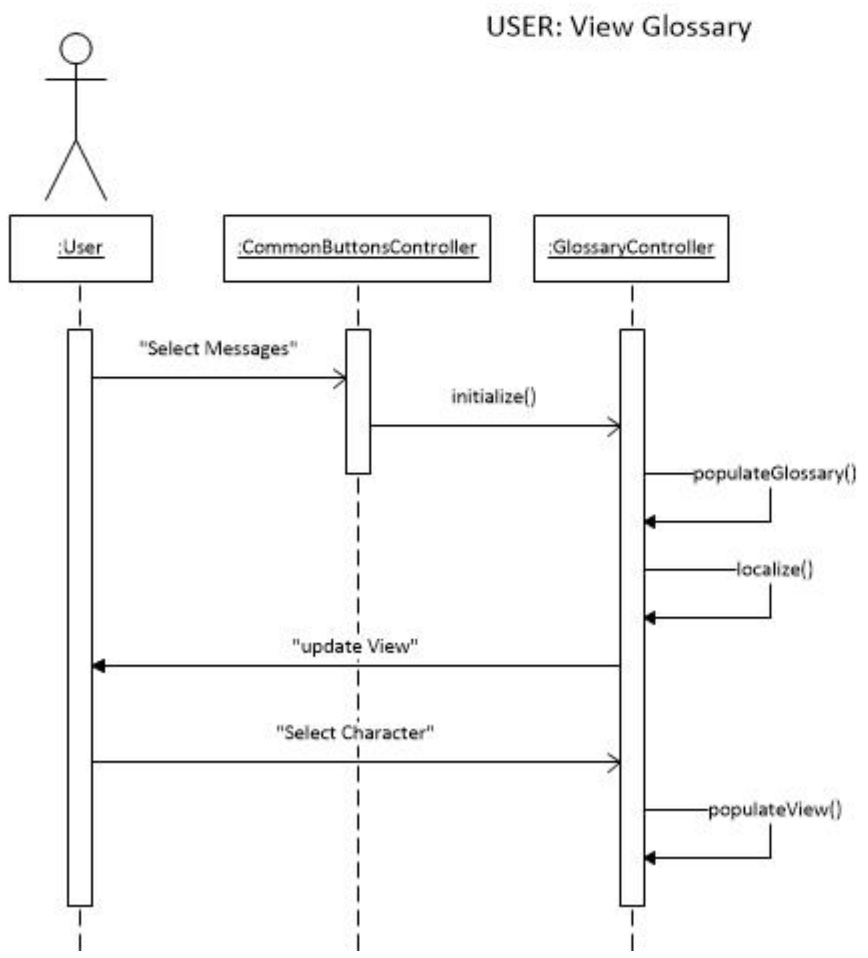


Figure 3 – View Glossary

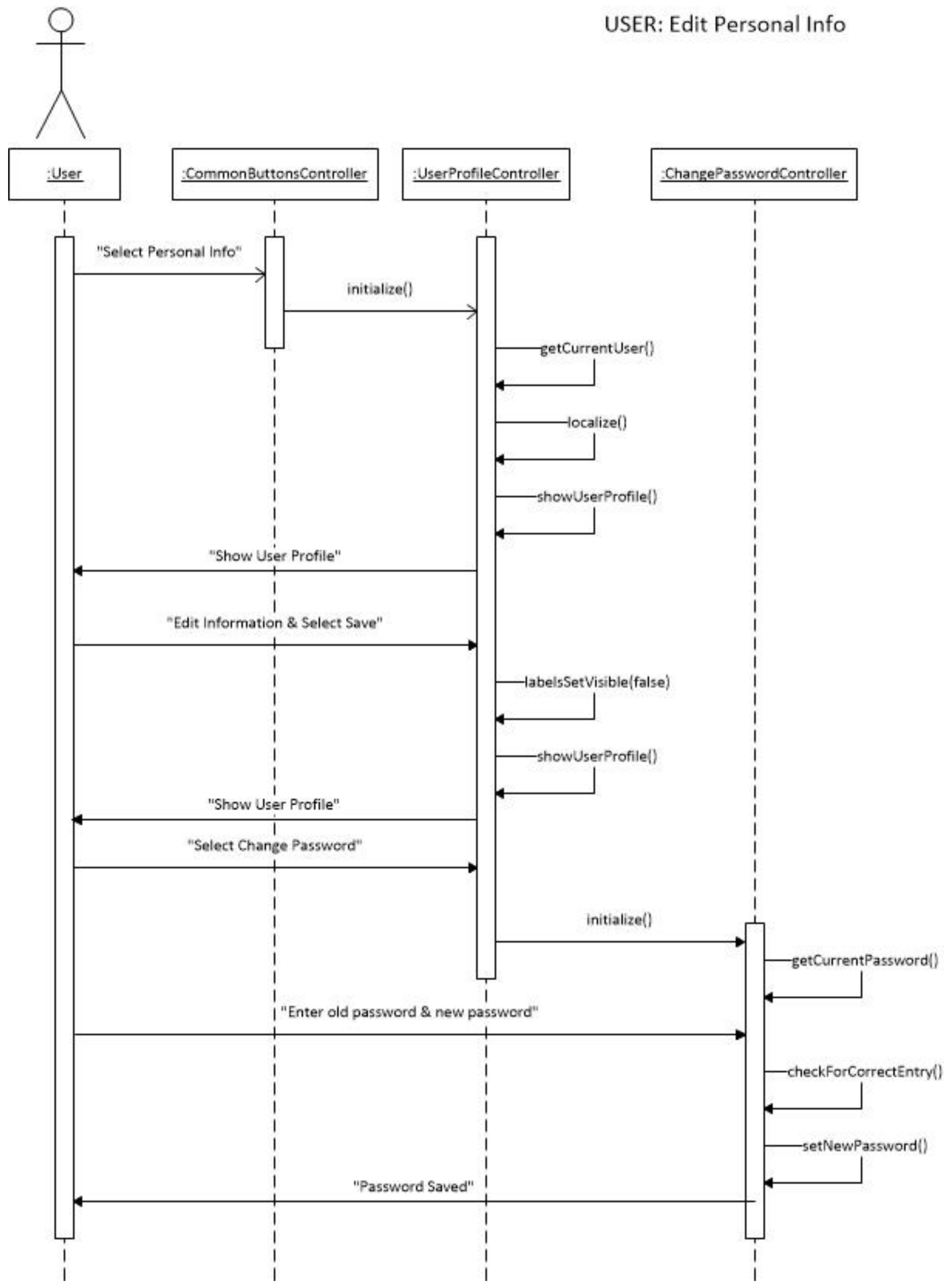


Figure 4 – View/Edit Personal Info

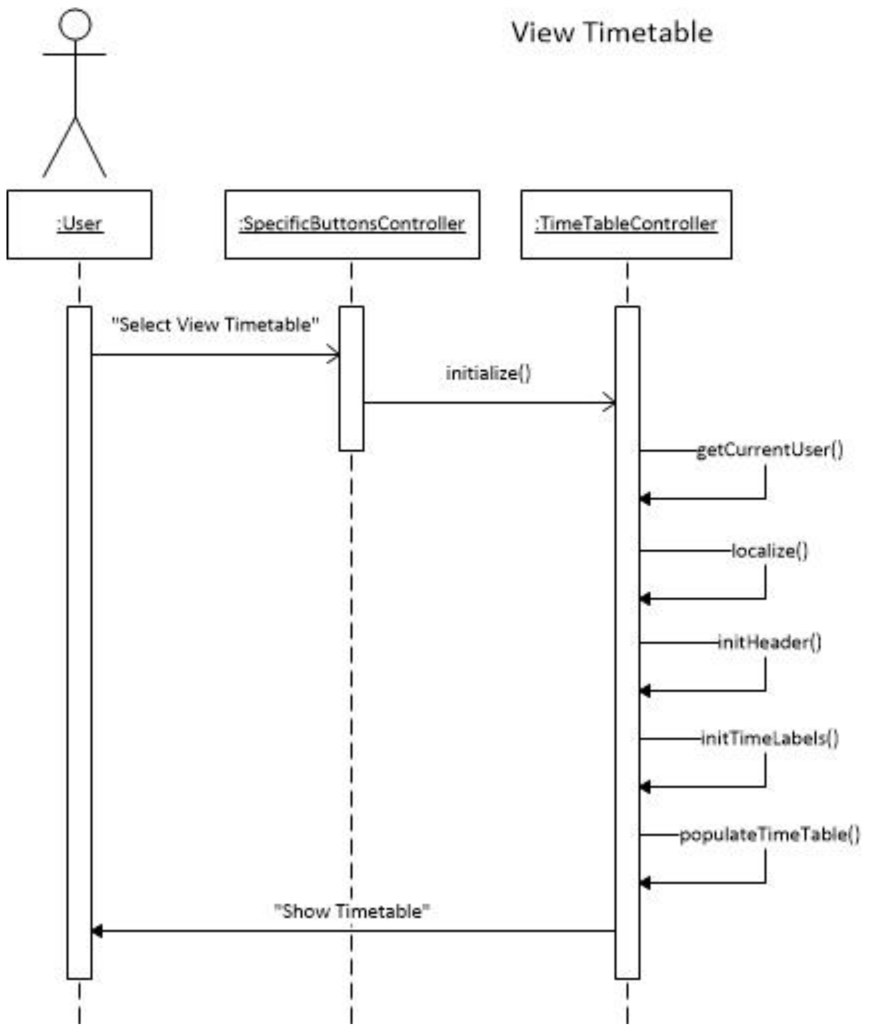


Figure 5 – View Timetable

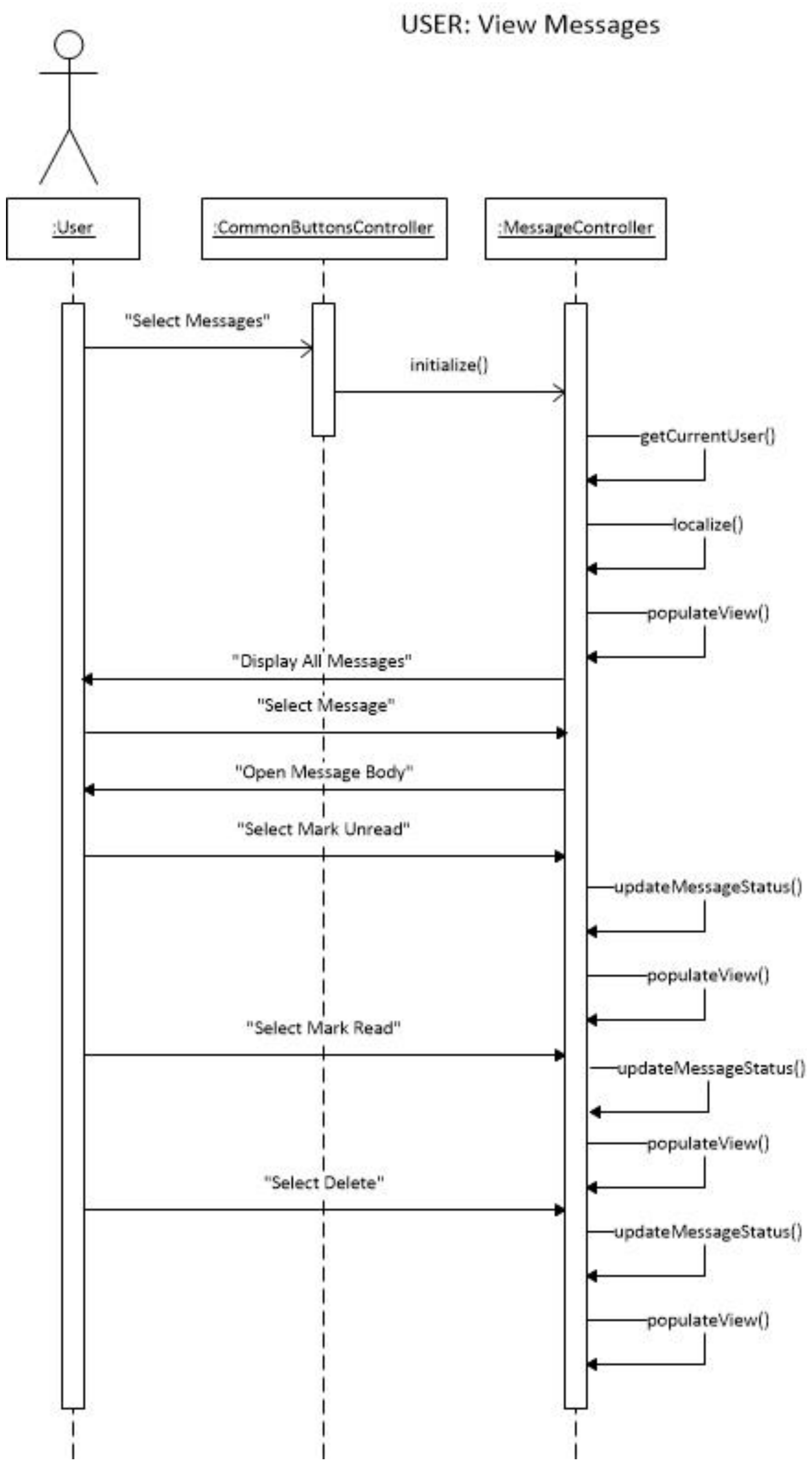


Figure 6 – Read Message

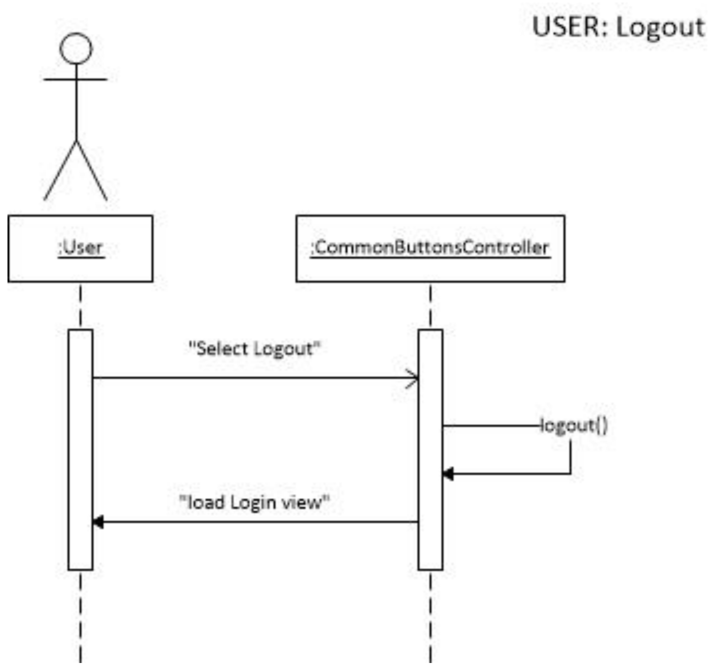


Figure 7 – Log Out

Student Features:

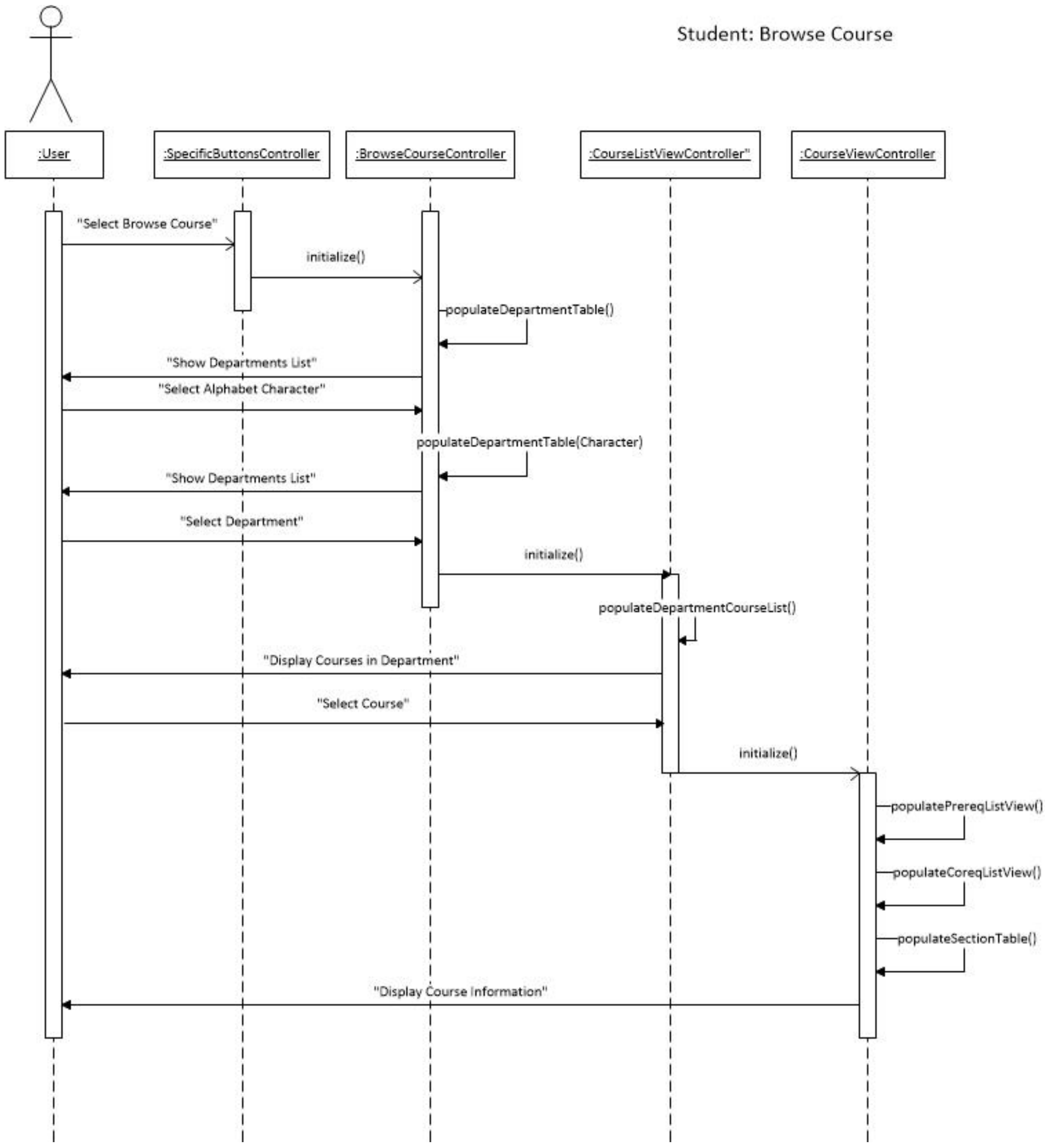


Figure 8 – Student Browse Course

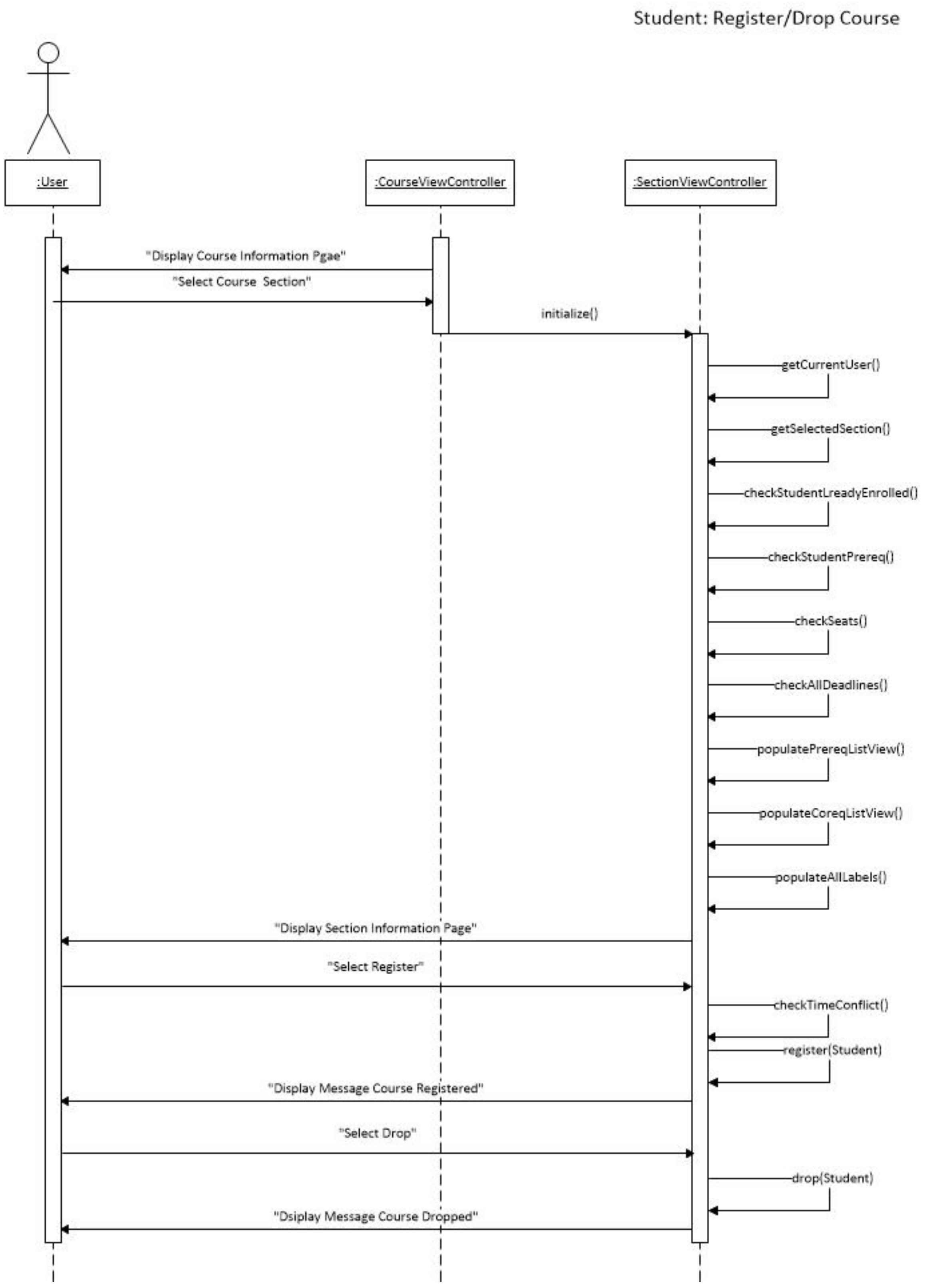


Figure 9 – Student Register/Drop Course

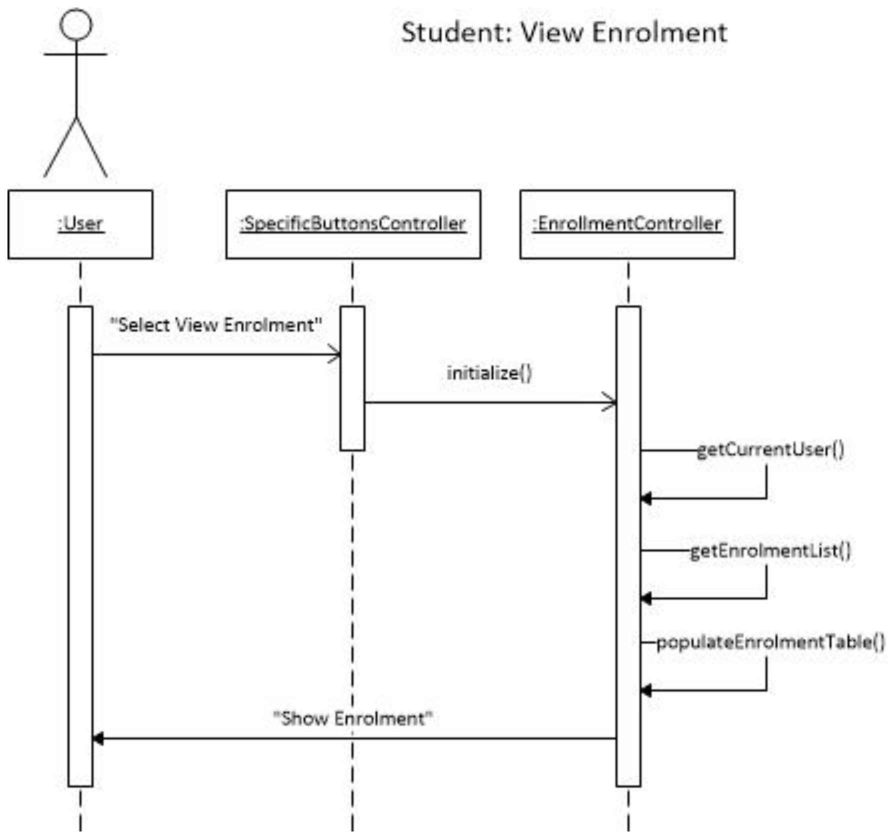


Figure 10 – Student View Enrolment

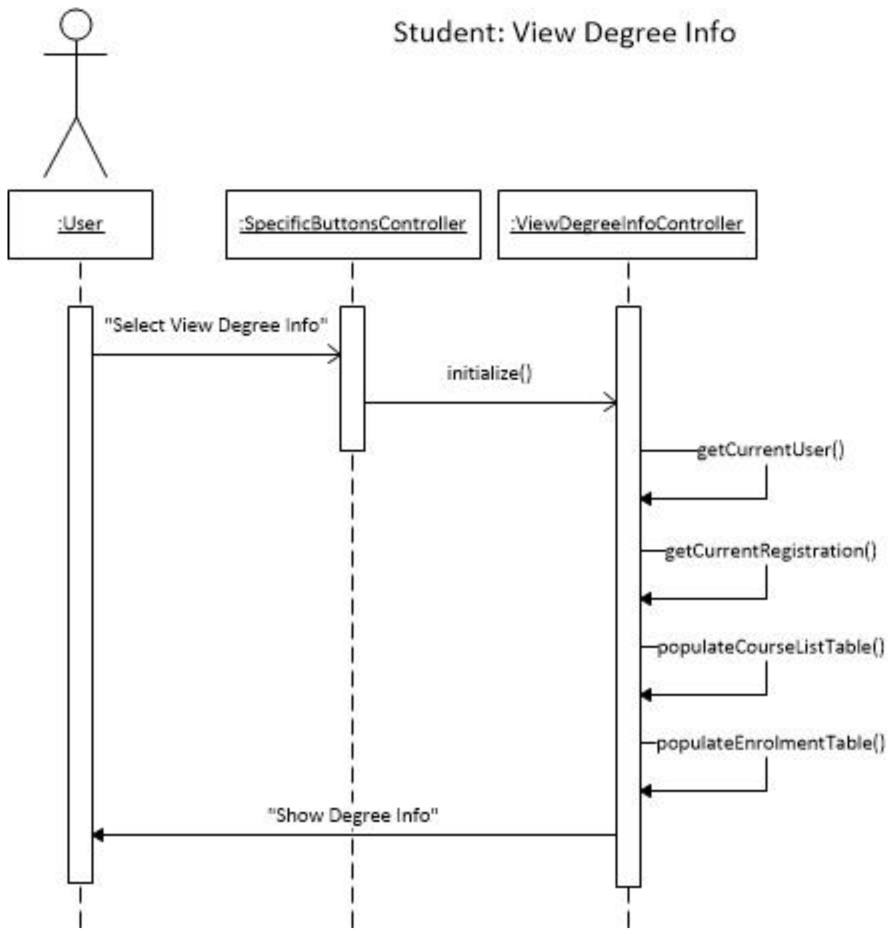

Student: View Degree Info
:User
:SpecificButtonsController
:ViewDegreeInfoController
"Select View Degree Info"
initialize()
getCurrentUser()
getCurrentRegistration()
populateCourseListTable()
populateEnrolmentTable()
"Show Degree Info"

Figure 11 – Student View Degree Info

Faculty Features:

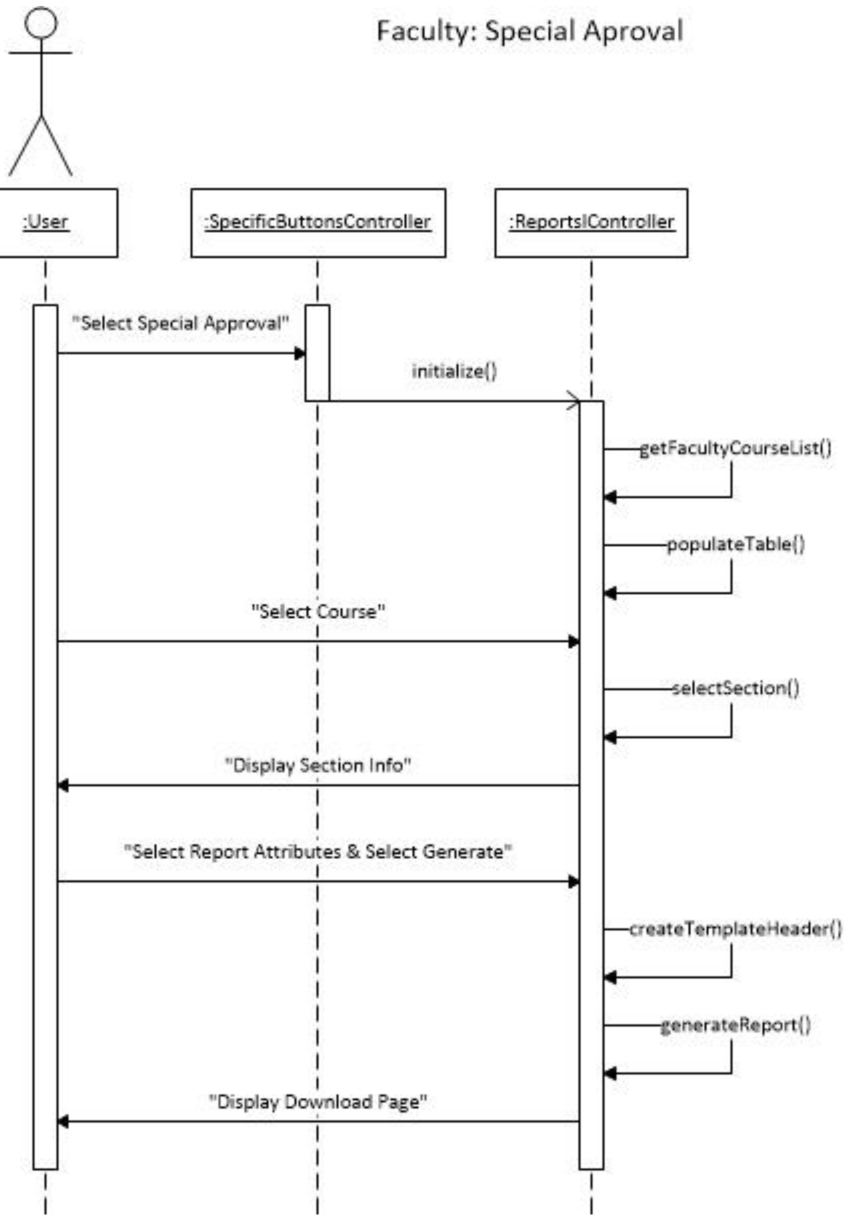


Figure 12 – Faculty Generate Reports

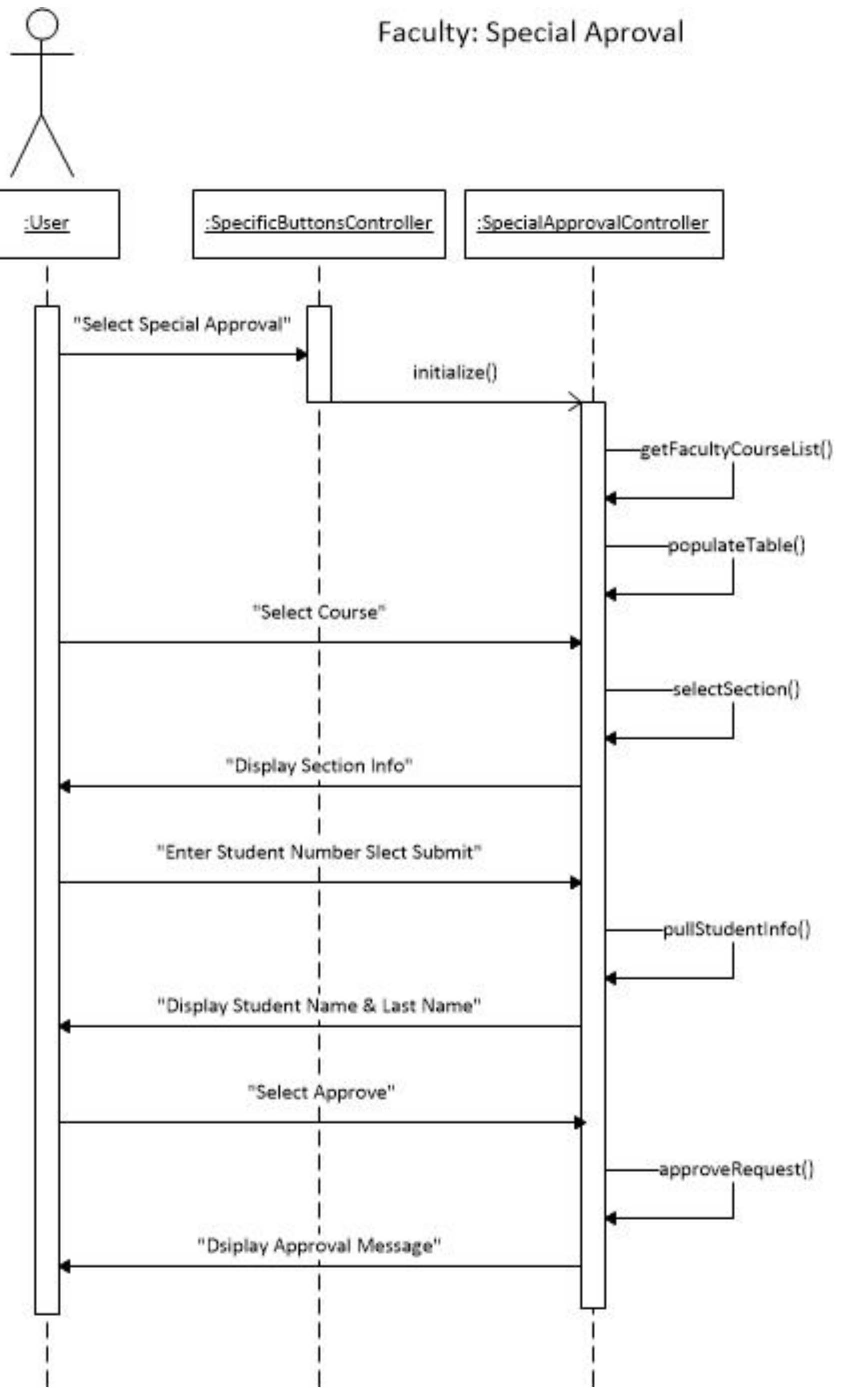
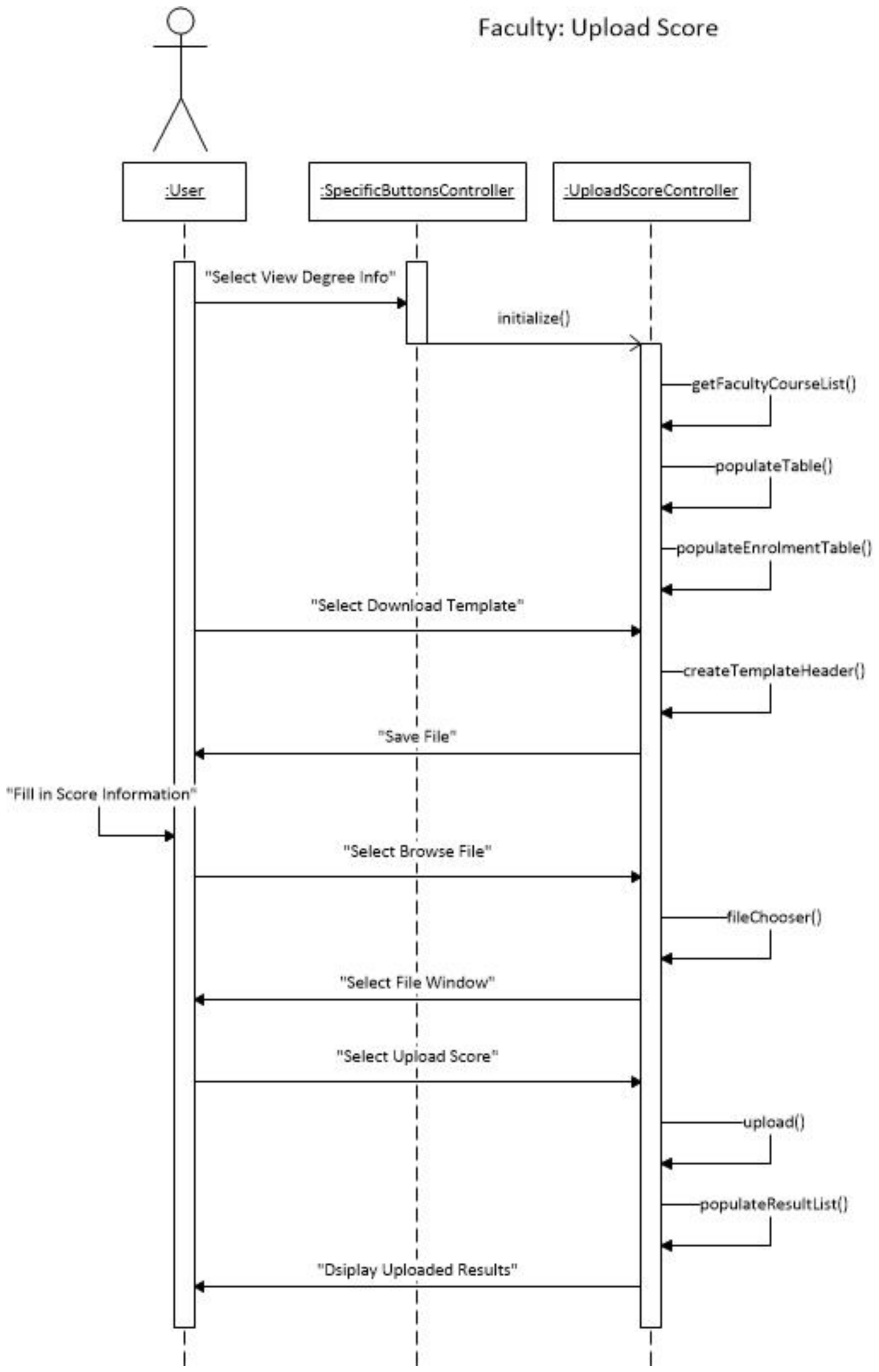
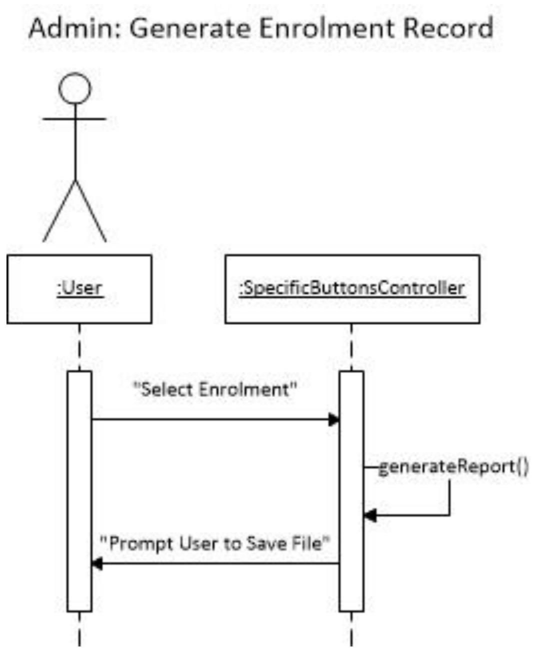


Figure 13 – Faculty Make Special Approval

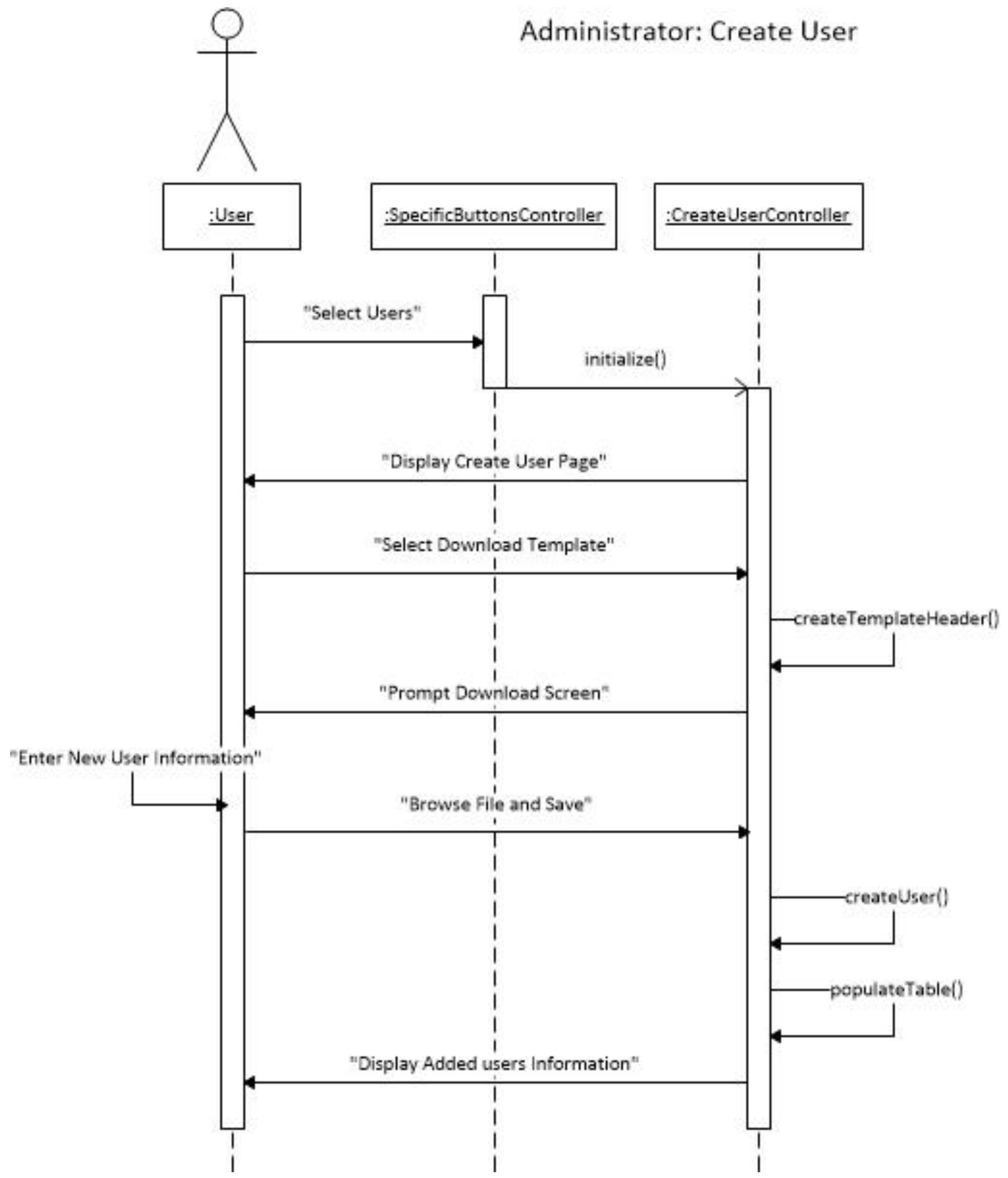


Faculty 14 – Upload Scores

Administrator Features:


Admin: Generate Enrolment Record
:User
:SpecificButtonsController
"Select Enrolment"
generateReport()
"Prompt User to Save File"

Faculty 15 – Administrator Enrolment


Administrator: Create User
:User
:SpecificButtonsController
:CreateUserController
"Select Users"
initialize()
"Display Create User Page"
"Select Download Template"
createTemplateHeader()
"Prompt Download Screen"
"Enter New User Information"
"Browse File and Save"
createUser()
populateTable()
"Display Added users Information"

Faculty 16 – Administrator Create Users

APPENDIX D - Deployment Diagram

The following two deployment diagrams display the overall components implemented into the system (Figure 9) and the application and server components implemented into a single system and database component implemented in Cloud (Figure 10):

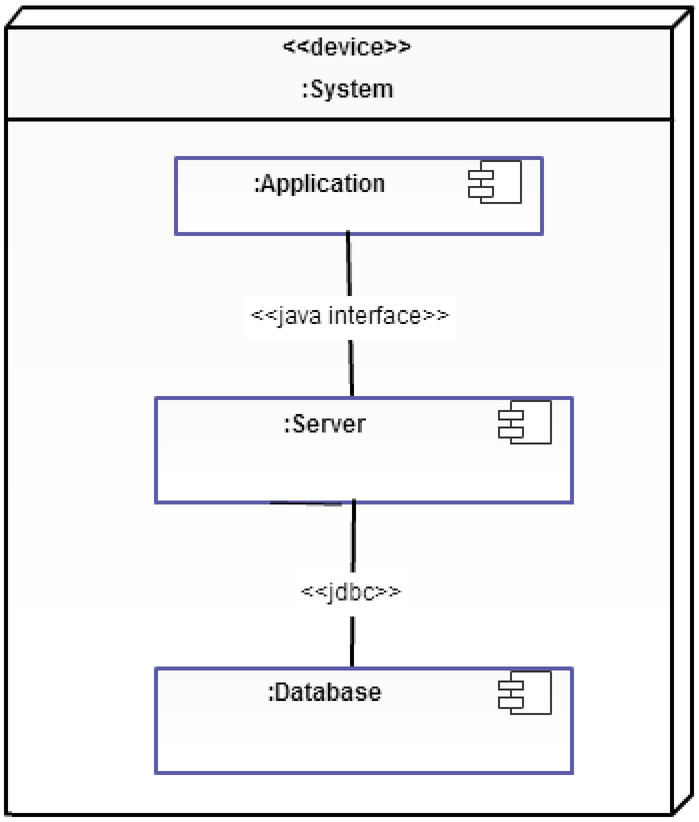


Figure 9 - All components implemented into a single system

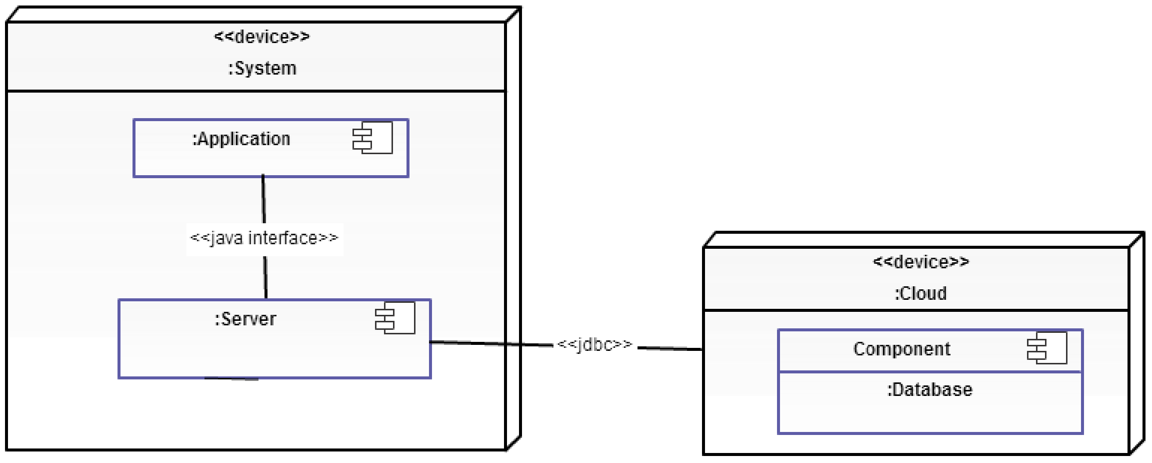


Figure 10 - Application and server components the database component in Cloud

APPENDIX E - Component Diagram

The following component diagrams display the overall system (Figure11) and the subsystem decomposition including the interface (Figure 12) and the database (Figures 13, 14 and 15):

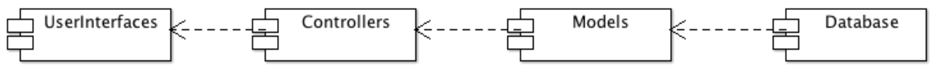


Figure 11 - System: RED

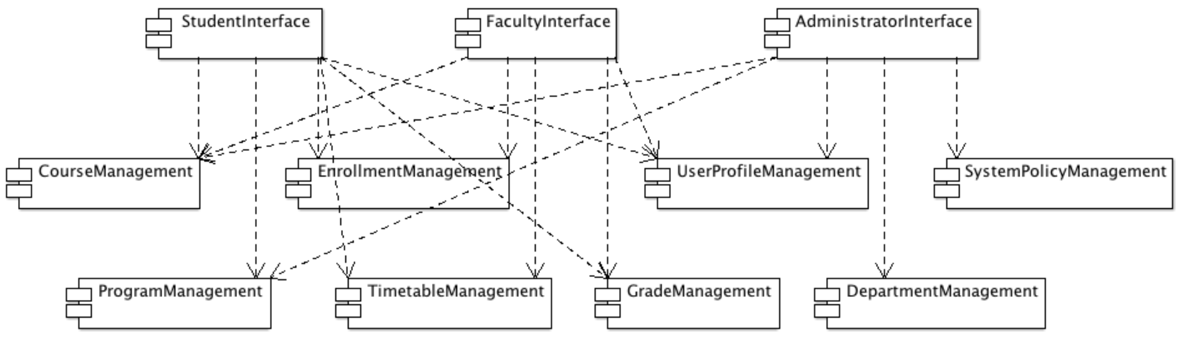


Figure 12: Subsystem Decomposition: User Interfaces

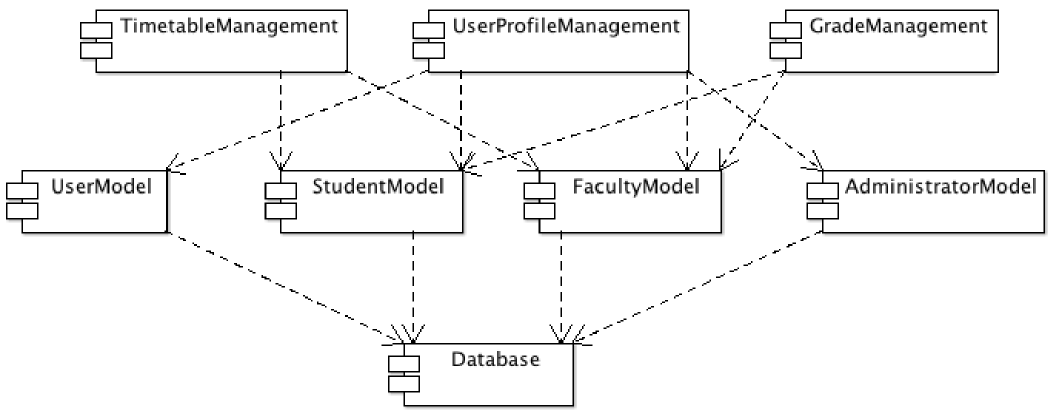


Figure 13: Subsystem Decomposition: Database

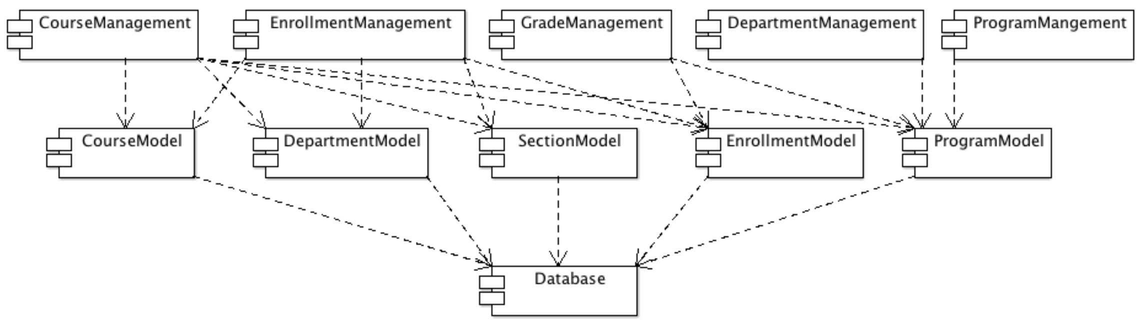


Figure 14: Subsystem Decomposition: Database

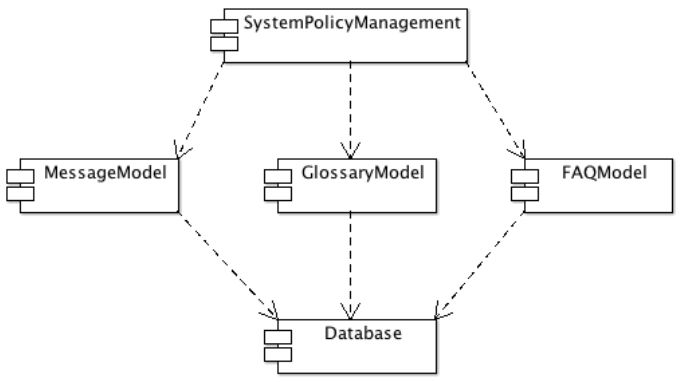


Figure 15: Subsystem Decomposition: Database

APPENDIX F – ER DIAGRAMS, DATABASE DESIGN & SCRIPTS

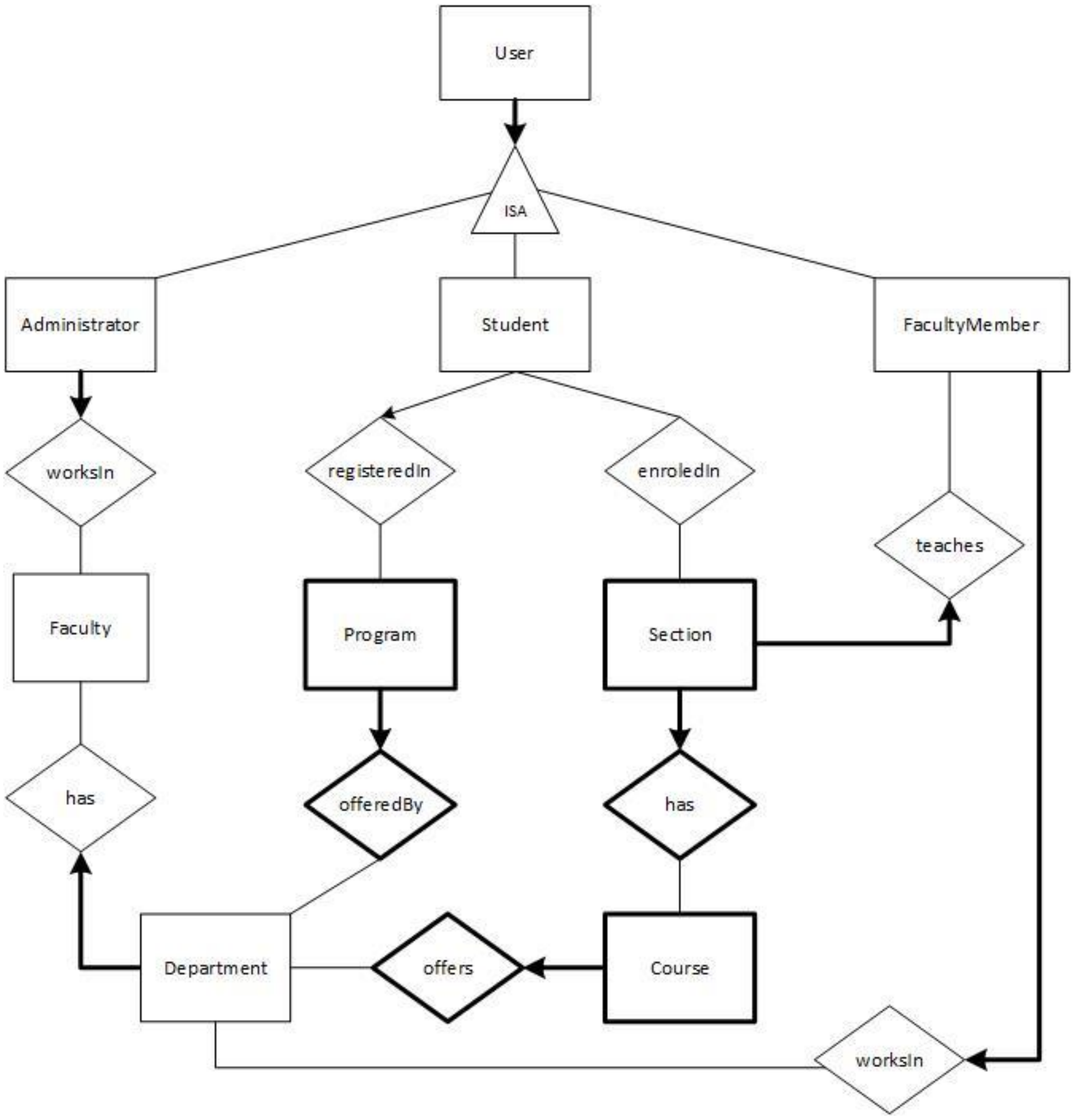


Figure 1: ER Diagram (MAIN)

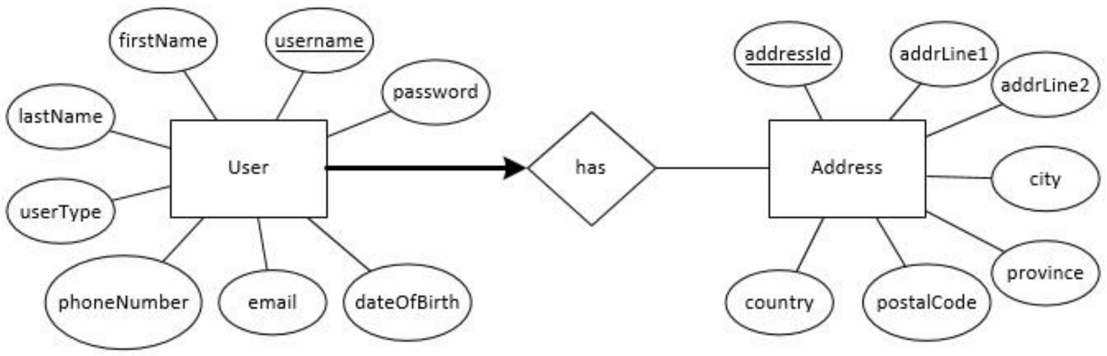


Figure 2: ER Diagram (User specific)

SQL Script

```
DROP TABLE IF EXISTS `UserType`;
CREATE TABLE `UserType` (
  `userTypeId` INT NOT NULL AUTO_INCREMENT,
  `name` varchar(25) NOT NULL,
  PRIMARY KEY (`userTypeId`),
  UNIQUE KEY `UserTypeUNIQname` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE UserType AUTO_INCREMENT = 100;

DROP TABLE IF EXISTS `Address`;
CREATE TABLE `Address` (
  `addressId` INT NOT NULL AUTO_INCREMENT,
  `addressLineFirst` varchar(45) NOT NULL,
  `addressLineSecond` varchar(45) ,
  city varchar(15)  NOT NULL,
  province varchar(16),
  `postalCode` varchar(6) NOT NULL,
  country varchar(25) NOT NULL,
  PRIMARY KEY (`addressId`)
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
 ALTER TABLE Address AUTO_INCREMENT = 100;
```

```sql
DROP TABLE IF EXISTS `User`;

CREATE TABLE `User` (

  `username` varchar(15) NOT NULL,

  `password` varchar(40) NOT NULL,

  `firstName` varchar(15) NOT NULL,

  `lastName` varchar(15) NOT NULL,

  `isOnline` bit(1) NOT NULL DEFAULT b'0',

  `userTypeId` INT NOT NULL,

   addressId INT NOT NULL,

  `phoneNumber` varchar(15) DEFAULT NULL,

  `email` varchar(45) NOT NULL,

  `dateOfBirth` date DEFAULT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

  PRIMARY KEY (`username`),

  CONSTRAINT `UserFKuserTypeId` FOREIGN KEY (`userTypeId`) REFERENCES `UserType`
(`userTypeId`) ON UPDATE CASCADE ON DELETE RESTRICT,

  CONSTRAINT `UserFKaddressId` FOREIGN KEY (`addressId`) REFERENCES `Address`
(`addressId`) ON UPDATE CASCADE ON DELETE RESTRICT,

  UNIQUE KEY `UserUNIQemail`  (`email`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;


ALTER TABLE User

ADD UNIQUE INDEX UserINDusernamePasswordIsActive (username,password,isActive)

USING BTREE;


ALTER TABLE User

ADD INDEX UserINfirstNamelastName (firstName, lastName)

USING BTREE;
```

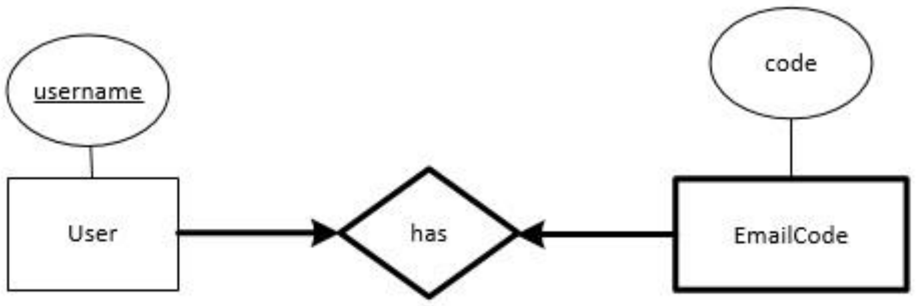


Figure 3: ER Diagram (Email Code)

SQL Script

```
DROP TABLE IF EXISTS `EmailCode`;

CREATE TABLE `EmailCode` (

  `username` varchar(15) NOT NULL,

  `code` VARCHAR(6) NOT NULL ,

   PRIMARY KEY (`username`),

   CONSTRAINT `EmailCodeFKusername` FOREIGN KEY (`username`) REFERENCES `User`
(`username`) ON UPDATE CASCADE ON DELETE RESTRICT

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

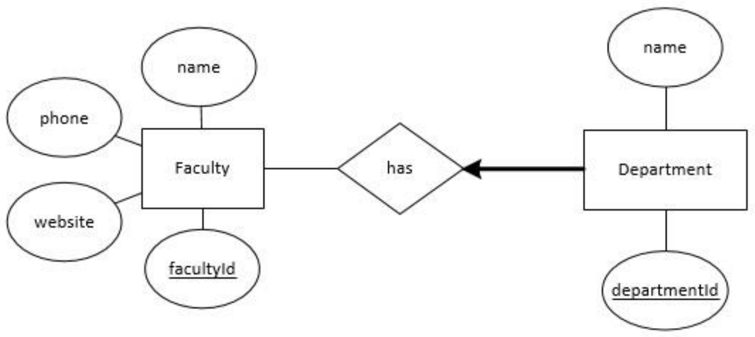


Figure 4: ER Diagram (Faculty & Department)

SQL Script

```
DROP TABLE IF EXISTS `Faculty`;

CREATE TABLE `Faculty` (

  `facultyId` INT NOT NULL AUTO_INCREMENT,

   name VARCHAR(50) NOT NULL,

  `phone` varchar(15) DEFAULT NULL,

  `website` varchar(50) DEFAULT NULL,

  PRIMARY KEY (`facultyId`),

  UNIQUE KEY `FacultyUNIQname` (name),

  CONSTRAINT FacultyCHKfacultyId CHECK (facultyId > 0)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE Faculty AUTO_INCREMENT = 100;


DROP TABLE IF EXISTS `Department`;

CREATE TABLE `Department` (

  `departmentId` varchar(4) NOT NULL,

  `name` varchar(100) NOT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

  facultyId INT NOT NULL,

  PRIMARY KEY (`departmentId`),

  CONSTRAINT `DepartmentFKfacultyId` FOREIGN KEY (`facultyId`) REFERENCES `Faculty`
(`facultyId`) ON UPDATE CASCADE ON DELETE RESTRICT

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

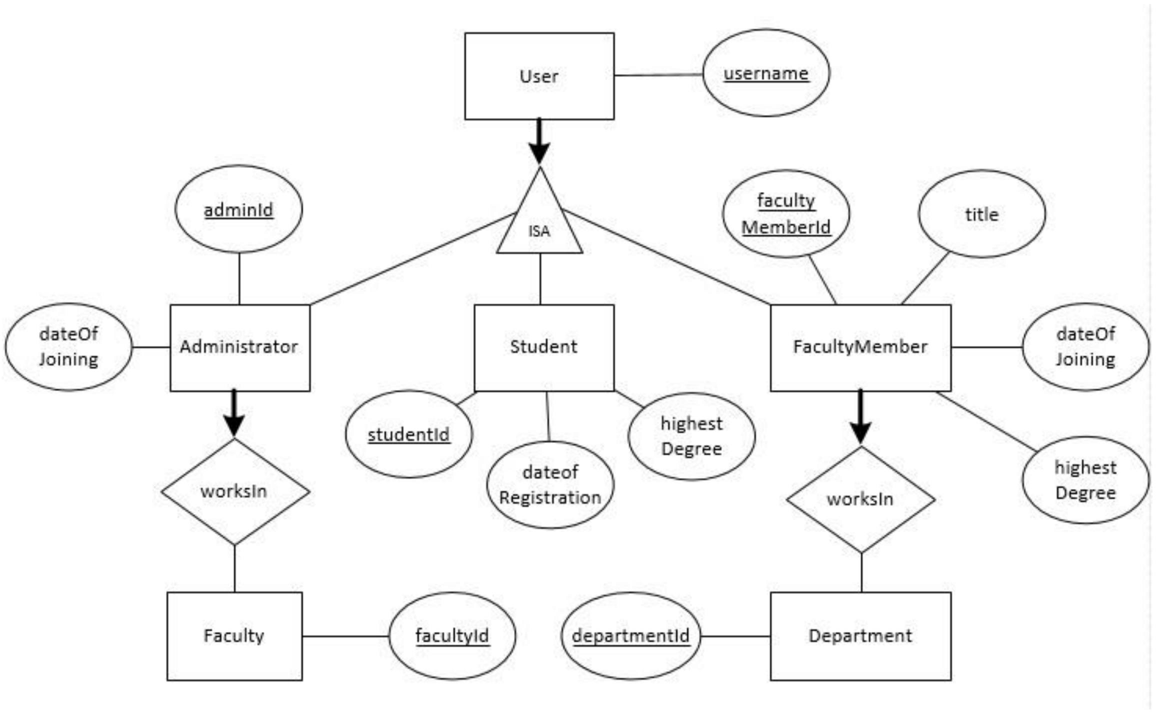


Figure 5: ER Diagram (User Types: Student, Administrator and FacultyMember)

SQL Script

```
DROP TABLE IF EXISTS `Student`;
CREATE TABLE `Student` (
  `studentId` INT NOT NULL AUTO_INCREMENT,
  `username` varchar(10) NOT NULL,
  `dateOfRegistration` date DEFAULT NULL,
  `highestDegree` varchar(20) DEFAULT NULL,
  `isActive` bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY (`studentId`),
  UNIQUE KEY `StudentUNIQusername` (`username`),
  CONSTRAINT `StudentFKusername` FOREIGN KEY (`username`) REFERENCES `User` (`username`)
ON DELETE CASCADE ON UPDATE CASCADE
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE Student AUTO_INCREMENT = 100;
```

```
DROP TABLE IF EXISTS `Administrator`;

CREATE TABLE `Administrator` (

  `adminId` INT NOT NULL AUTO_INCREMENT,

  `username` varchar(10) NOT NULL,

  `dateOfJoining` date NOT NULL,

  `hiringFacultyId` INT NOT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

  PRIMARY KEY (`adminId`),

  UNIQUE KEY `AdministratorUNIQusername` (`username`),

  CONSTRAINT `AdministratorFKusername` FOREIGN KEY (`username`) REFERENCES `User`
(`username`) ON DELETE CASCADE ON UPDATE CASCADE,

  CONSTRAINT `AdministratorFKhiringFacultyId` FOREIGN KEY (`hiringFacultyId`) REFERENCES
`Faculty` (`facultyId`) ON UPDATE CASCADE ON DELETE RESTRICT

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE Administrator AUTO_INCREMENT = 100;


DROP TABLE IF EXISTS `FacultyMember`;

CREATE TABLE `FacultyMember` (

  `facultyMemberId` INT NOT NULL AUTO_INCREMENT,

  `username` varchar(10) NOT NULL,

  `departmentId` varchar(4) NOT NULL,

  `title` varchar(20) NOT NULL,

  `dateOfJoining` date NOT NULL,

  `highestDegree` varchar(25) DEFAULT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

  PRIMARY KEY (`facultyMemberId`),

  UNIQUE KEY `FacultyMemberUNIQusername` (`username`),

  CONSTRAINT `FacultyMemberFKdepartmentId` FOREIGN KEY (`departmentId`) REFERENCES
`Department` (`departmentId`) ON UPDATE CASCADE ON DELETE RESTRICT,

  CONSTRAINT `FacultyMemberFKusername` FOREIGN KEY (`username`) REFERENCES `User`
(`username`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE FacultyMember AUTO_INCREMENT = 100;
```

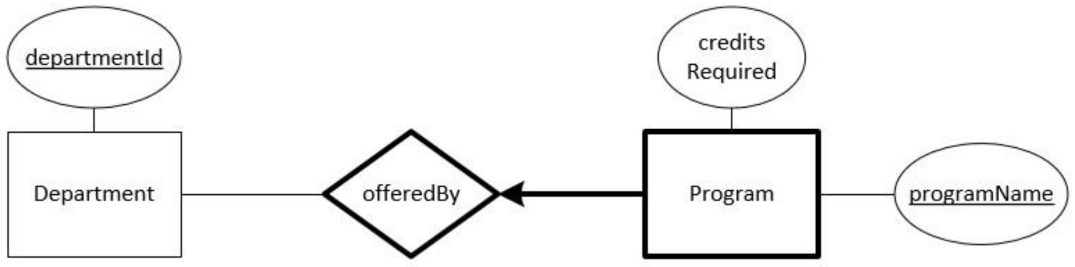


Figure 6 – ER Diagram(Program)

SQL Script

```
DROP TABLE IF EXISTS `Program`;
CREATE  TABLE `Program` (
  `programName` VARCHAR(30) NOT NULL ,
  `departmentId` VARCHAR(4) NOT NULL ,
  `isActive` BIT NOT NULL DEFAULT 1 ,
  `creditsRequired` INT NOT NULL ,
  PRIMARY KEY (`programName`, `departmentId`) ,
  CONSTRAINT `ProgramFKdepartmentId`
    FOREIGN KEY (`departmentId` )
    REFERENCES `Department` (`departmentId` )
    ON DELETE RESTRICT
    ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

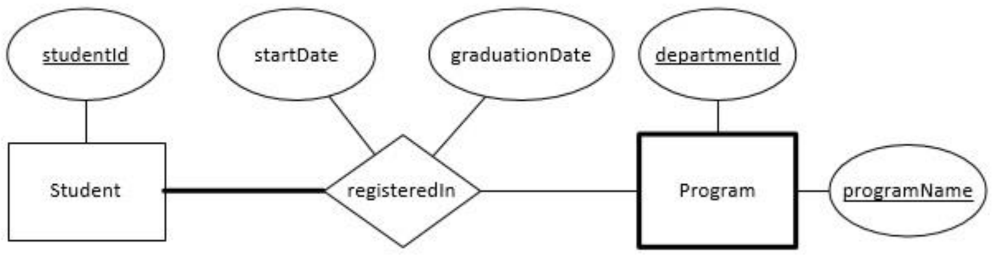


Figure 7 – ER Diagram (Registration)

SQL Script

```
DROP TABLE IF EXISTS `Registration`;
CREATE  TABLE `Registration` (
  `studentId` INT NOT NULL ,
  `programName` VARCHAR(30) NOT NULL ,
  `departmentId` VARCHAR(4) NOT NULL ,
  `startDate` DATE NOT NULL ,
  `graduationDate` DATE NOT NULL ,
  `isActive` BIT NOT NULL DEFAULT 1 ,
  PRIMARY KEY (`studentId`, `programName`, `departmentId`) ,
  CONSTRAINT `RegistrationFKstudentId`
    FOREIGN KEY (`studentId` )
    REFERENCES `Student` (`studentId` )
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `RegistrationFKprogramNamedepartmentId`
    FOREIGN KEY (`programName`,`departmentId` )
    REFERENCES `Program` (`programName`,`departmentId` )
    ON DELETE RESTRICT
    ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


ALTER TABLE Registration
ADD INDEX RegistrationINDstudentIdIsActive (studentId, isActive) USING BTREE;
```

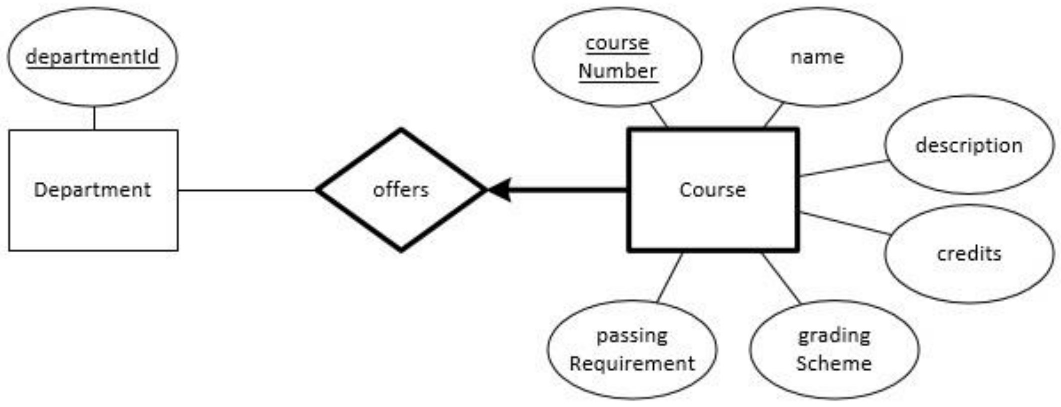


Figure 8 - ER Diagram (Course Specific)

SQL Script

```
DROP TABLE IF EXISTS `GradingScheme`;
CREATE TABLE `GradingScheme` (
 gradingSchemeId INT NOT NULL AUTO_INCREMENT,
 name  varchar(25) NOT NULL,
 PRIMARY KEY (`gradingSchemeId`),
 UNIQUE KEY `GradingSchemeUNIQname` (`name`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE GradingScheme AUTO_INCREMENT = 100;

DROP TABLE IF EXISTS `Course`;
CREATE TABLE `Course` (
  `courseNumber` INT NOT NULL,
  `departmentId` varchar(4) NOT NULL,
  `name` varchar(45) NOT NULL,
  `description` varchar(1000) NOT NULL,
  `credits` INT NOT NULL,
  `gradingSchemeId`INT NOT NULL,
  `passingRequirement` varchar(45) NOT NULL,
  `isActive` bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY (`courseNumber`,`departmentId`),

  CONSTRAINT `CourseFKdepartmentId` FOREIGN KEY (`departmentId`) REFERENCES `Department`
(`departmentId`) ON UPDATE CASCADE ON DELETE RESTRICT,

  CONSTRAINT `CourseFKgradingSchemeId` FOREIGN KEY (`gradingSchemeId`) REFERENCES
`GradingScheme` (`gradingSchemeId`) ON UPDATE CASCADE ON DELETE RESTRICT

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

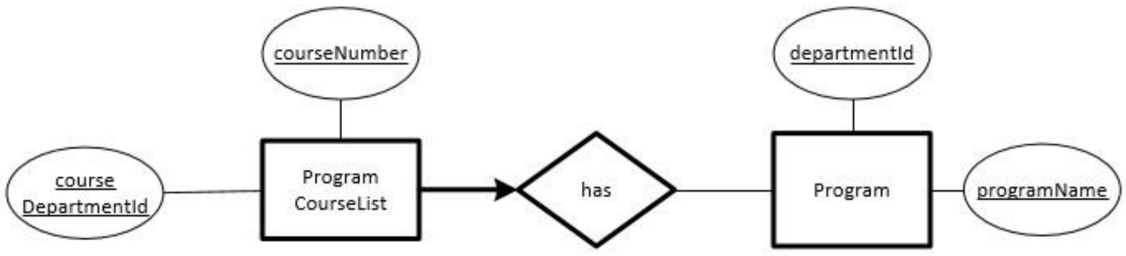


Figure 9 - ER Diagram (Program Course List)

SQL Script

```
DROP TABLE IF EXISTS `ProgramCourseList`;

CREATE TABLE `ProgramCourseList` (

  `programName` VARCHAR(30) NOT NULL ,

  `programDepartmentId` VARCHAR(4) NOT NULL ,

  `courseNumber` INT NOT NULL,

  `courseDepartmentId` varchar(4) NOT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

   PRIMARY KEY (`programName`,`programDepartmentId`,`courseNumber`,`courseDepartmentId`),

   CONSTRAINT `ProgramCourseListFKprogramTable` FOREIGN KEY
(`programName`,`programDepartmentId`) REFERENCES `Program` (`programName`,`departmentId`)
ON UPDATE CASCADE ON DELETE RESTRICT,

   CONSTRAINT `ProgramCourseListFKcourseTable` FOREIGN KEY
(`courseNumber`,`courseDepartmentId`) REFERENCES `Course` (`courseNumber`,`departmentId`)
ON UPDATE CASCADE ON DELETE RESTRICT

) ENGINE=InnoDB DEFAULT CHARSET=utf8;


ALTER TABLE ProgramCourseList

ADD INDEX ProgramCourseListINDprogramNameProgramDeptId (programName, programDepartmentId)
USING BTREE;
```

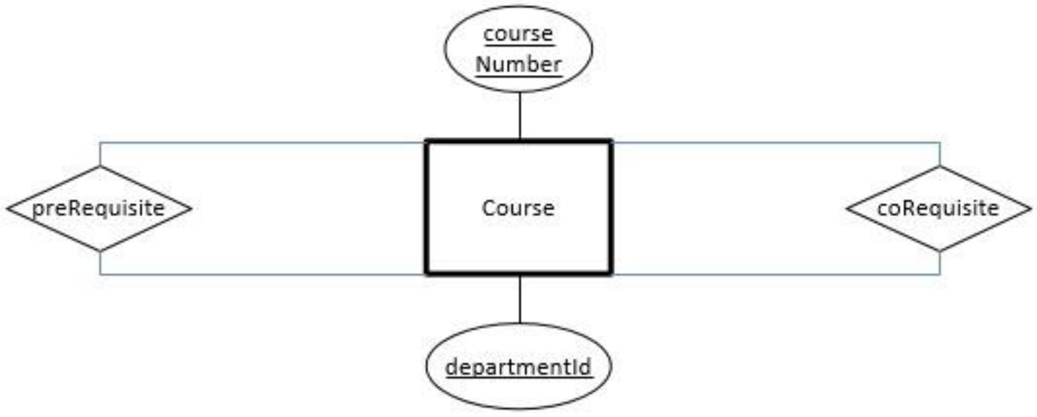


Figure 10: ER Diagram (Co-Requisites and Pre-Requisites)

SQL Script

```
DROP TABLE IF EXISTS `CoRequisite`;

CREATE TABLE `CoRequisite` (

  `courseNumber` int NOT NULL,

  `departmentId` varchar(4) NOT NULL,

  `coRequisiteNumber` int NOT NULL,

  `coRequisiteDeptId` varchar(4) NOT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

  isMust bit(1) NOT NULL DEFAULT b'1',

  PRIMARY KEY (`courseNumber`,`departmentId`,`coRequisiteNumber`,`coRequisiteDeptId`),

  CONSTRAINT `CoRequisiteFKcourseNumberdepartmentId` FOREIGN KEY
(`courseNumber`,`departmentId`) REFERENCES `Course` (`courseNumber`,`departmentId`) ON
DELETE RESTRICT ON UPDATE CASCADE,

  CONSTRAINT `CoRequisiteFKcoRequisiteDeptIdcoRequisiteDeptId` FOREIGN KEY
(`coRequisiteNumber`,`coRequisiteDeptId`) REFERENCES `Course`
(`courseNumber`,`departmentId`) ON DELETE RESTRICT ON UPDATE CASCADE

  ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```sql
DROP TABLE IF EXISTS `Prerequisite`;

CREATE TABLE `Prerequisite` (

  `courseNumber` int NOT NULL,

  `departmentId` varchar(4) NOT NULL,

  `preRequisiteNumber` int NOT NULL,

  `preRequisiteDeptId` varchar(4) NOT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

  isMust bit(1) NOT NULL DEFAULT b'1',

  PRIMARY KEY (`courseNumber`,`departmentId`,`preRequisiteNumber`,`preRequisiteDeptId`),

  CONSTRAINT `PrerequisiteFKcourseNumberdepartmentId` FOREIGN KEY
(`courseNumber`,`departmentId`) REFERENCES `Course` (`courseNumber`,`departmentId`) ON
DELETE RESTRICT ON UPDATE CASCADE,

  CONSTRAINT `PrerequisiteFKpreRequisiteNumberpreRequisiteDeptId` FOREIGN KEY
(`preRequisiteNumber`,`preRequisiteDeptId`) REFERENCES `Course`
(`courseNumber`,`departmentId`) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

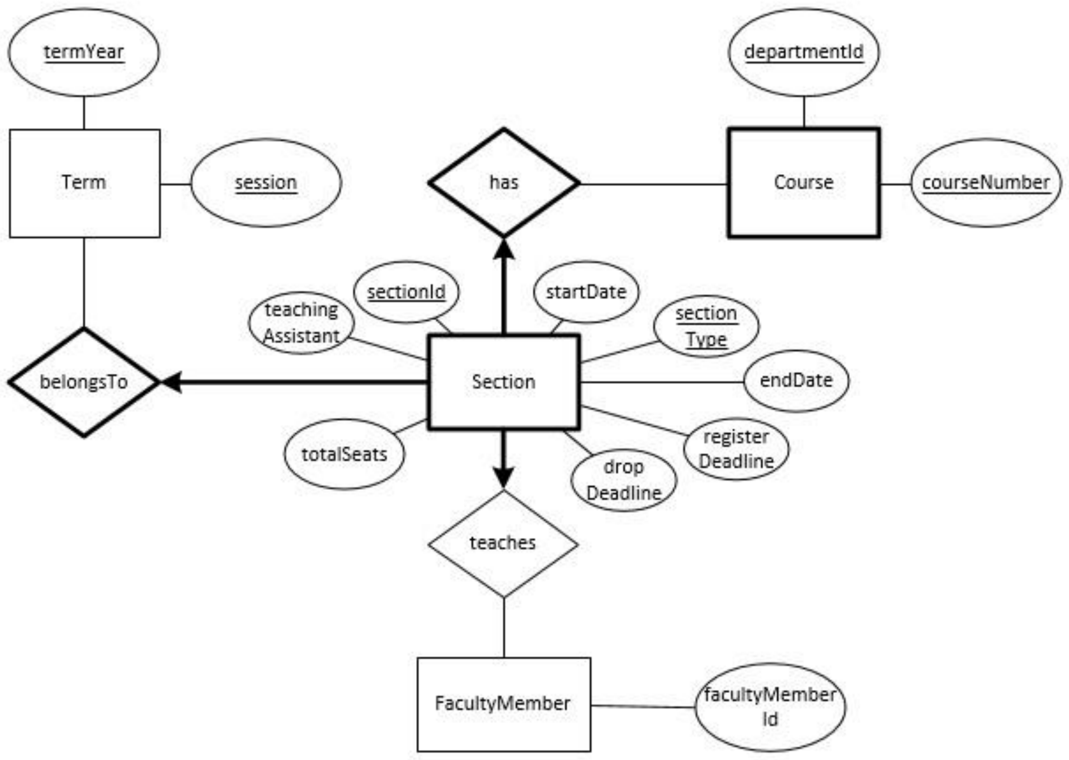


Figure 11: ER Diagram (Section Specific)

SQL Script

```
DROP TABLE IF EXISTS `Session`;
CREATE TABLE `Session` (
  `sessionId` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(10) NOT NULL,
  PRIMARY KEY (`sessionId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE `Session` AUTO_INCREMENT = 100;

DROP TABLE IF EXISTS `Term`;
CREATE TABLE `Term` (
  `termYear` SMALLINT NOT NULL,
  `sessionId` INT NOT NULL,
  `isActive` bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY (`termYear`,`sessionId`),
  CONSTRAINT TermCHKtermYear CHECK (termYear > 2000 AND termYear < 2999),
  CONSTRAINT `TermFKsessionId` FOREIGN KEY (`sessionId`) REFERENCES `Session`
(`sessionId`) ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `SectionType`;
CREATE TABLE `SectionType` (
  `sectionTypeId` INT NOT NULL AUTO_INCREMENT,
  `name` varchar(25) NOT NULL,
  PRIMARY KEY (`sectionTypeId`),
  UNIQUE KEY `SectionTypeUNIQname` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE SectionType AUTO_INCREMENT = 100;

DROP TABLE IF EXISTS `Section`;
CREATE TABLE `Section` (
  `sectionId` INT NOT NULL,
  `sectionTypeId` INT NOT NULL,
  `courseNumber` INT NOT NULL,
  `departmentId` varchar(4) NOT NULL,
  `termYear` SMALLINT NOT NULL,
  `sessionId` INT NOT NULL,
  `startDate` date NOT NULL,
  `endDate` date NOT NULL,
  `registerDeadline` date NOT NULL,
  `dropDeadline` date NOT NULL,
  `totalSeats` INT NOT NULL,
  `facultyMemberId` INT NOT NULL,
  `teachingAssistant` varchar(20) DEFAULT NULL,
  `isActive` bit(1) NOT NULL DEFAULT b'1',
  PRIMARY KEY
(`sectionId`,`courseNumber`,`departmentId`,`termYear`,`sessionId`,`sectionTypeId`),
  CONSTRAINT `SectionFKcourseNumberdepartmentId` FOREIGN KEY
(`courseNumber`,`departmentId`) REFERENCES `Course` (`courseNumber`,`departmentId`) ON
DELETE RESTRICT ON UPDATE CASCADE,

  CONSTRAINT `SectionFKtermYearsessionId` FOREIGN KEY (`termYear`,`sessionId`) REFERENCES
`Term` (`termYear`,`sessionId`) ON DELETE RESTRICT ON UPDATE CASCADE,

  CONSTRAINT `SectionFKfacultyMemberId` FOREIGN KEY (`facultyMemberId`) REFERENCES
`FacultyMember` (`facultyMemberId`) ON DELETE RESTRICT ON UPDATE CASCADE,

  CONSTRAINT `SectionFKsectionTypeId` FOREIGN KEY (`sectionTypeId`) REFERENCES
`SectionType` (`sectionTypeId`) ON DELETE RESTRICT ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

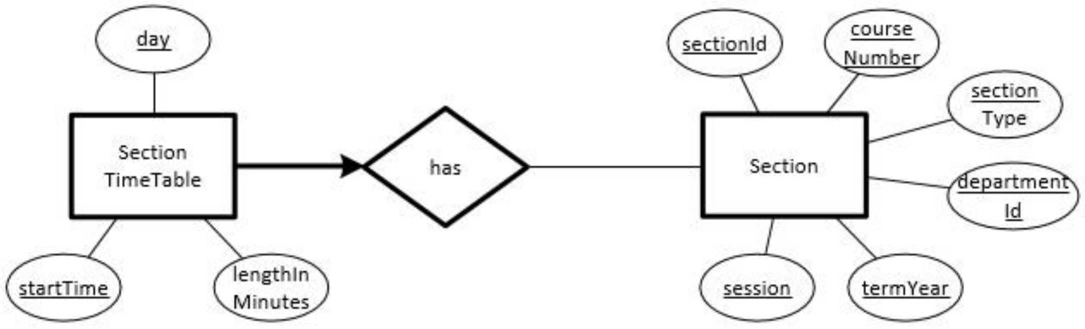


Figure 12 - ER Diagram (Section Time Table Specific)

SQL Script

```
DROP TABLE IF EXISTS `WeekDay`;
CREATE  TABLE `WeekDay` (
  `dayId` INT NOT NULL AUTO_INCREMENT,
  `weekDay` VARCHAR(10) NOT NULL ,
  PRIMARY KEY (`dayId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE WeekDay AUTO_INCREMENT = 100;

DROP TABLE IF EXISTS `SectionTimeTable`;
CREATE  TABLE `SectionTimeTable` (
  `sectionId` INT NOT NULL ,
  `courseNumber` INT NOT NULL ,
  `departmentId` VARCHAR(4) NOT NULL ,
  `termYear` SMALLINT NOT NULL,
  `sessionId` INT NOT NULL,
  `sectionTypeId` INT NOT NULL,
  `dayId` INT NOT NULL ,
  `startTime` TIME NOT NULL ,
  `lengthInMinutes` INT NOT NULL ,
  PRIMARY KEY (`sectionId`,`sectionTypeId`, `courseNumber`,
`departmentId`,`termYear`,`sessionId`, `dayId`, `startTime`),
  CONSTRAINT SectionTimeTableCHKlengthInMinutes CHECK (lengthInMinutes > 59),

  CONSTRAINT `SectionTimeTableFKSectionTable`

    FOREIGN KEY
(`sectionId`,`courseNumber`,`departmentId`,`termYear`,`sessionId`,`sectionTypeId`)

    REFERENCES `Section`
(`sectionId`,`courseNumber`,`departmentId`,`termYear`,`sessionId`,`sectionTypeId` )

    ON DELETE RESTRICT

    ON UPDATE CASCADE,

  CONSTRAINT `SectionTimeTableFKdayId`

    FOREIGN KEY (`dayId` )

    REFERENCES `WeekDay` (`dayId` )

    ON DELETE RESTRICT

    ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8;


ALTER TABLE SectionTimeTable

ADD INDEX SectionTimeTableINDdayIdStartTime (dayId, startTime) USING BTREE;
```

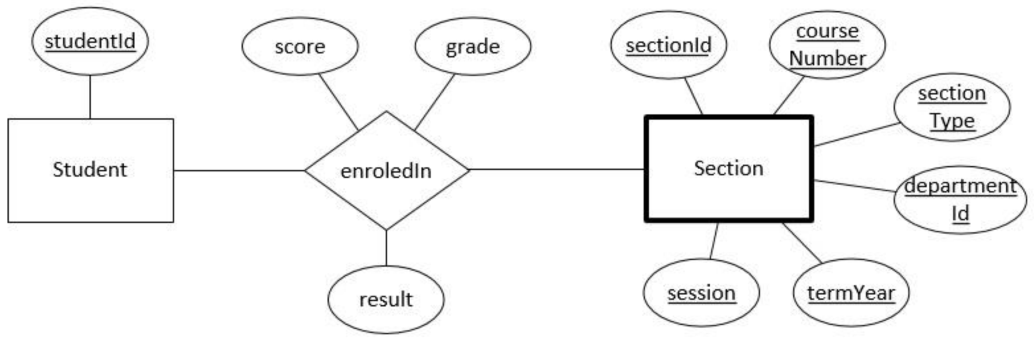


Figure 13: ER Diagram (Enrolment Specific)

SQL Script

```
DROP TABLE IF EXISTS `Grade`;
CREATE TABLE `Grade` (
  `gradeId` INT NOT NULL AUTO_INCREMENT,
  `name` varchar(1) NOT NULL,
   PRIMARY KEY (`gradeId`),
   UNIQUE KEY `GradeUNIQname` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE `Grade` AUTO_INCREMENT = 100;

DROP TABLE IF EXISTS `Result`;
CREATE TABLE `Result` (
  `resultId` INT NOT NULL AUTO_INCREMENT,
  `name` varchar(4) NOT NULL,
   PRIMARY KEY (`resultId`),
   UNIQUE KEY `ResultUNIQname` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE `Result` AUTO_INCREMENT = 100;
```

```
DROP TABLE IF EXISTS `Enrolment`;

CREATE  TABLE `Enrolment` (

  `studentId` INT NOT NULL ,

  `sectionId` INT NOT NULL ,

  `courseNumber` INT NOT NULL ,

  `departmentId` VARCHAR(4) NOT NULL ,

  `termYear` SMALLINT NOT NULL,

  `sessionId` INT NOT NULL,

  `sectionTypeId` INT NOT NULL,

  `score` INT DEFAULT NULL,

  `gradeId` INT DEFAULT NULL,

  `resultId` INT DEFAULT NULL,

  `isActive` bit(1) NOT NULL DEFAULT b'1',

  PRIMARY KEY (`studentId`, `sectionId`,`sectionTypeId`, `courseNumber`, `departmentId`,
`termYear`,`sessionId`) ,

  CONSTRAINT `EnrolmentFKstudentId`

    FOREIGN KEY (`studentId` )

    REFERENCES `Student` (`studentId` )

    ON DELETE RESTRICT

    ON UPDATE CASCADE,

  CONSTRAINT `EnrolmentFKsectionIdcourseNumberdepartmentId`

    FOREIGN KEY
(`sectionId`,`courseNumber`,`departmentId`,`termYear`,`sessionId`,`sectionTypeId` )

    REFERENCES `Section`
(`sectionId`,`courseNumber`,`departmentId`,`termYear`,`sessionId`,`sectionTypeId`)

    ON DELETE RESTRICT

    ON UPDATE CASCADE,

  CONSTRAINT `EnrolmentFKgradeId`

    FOREIGN KEY (`gradeId`) REFERENCES `Grade` (`gradeId`) ON DELETE RESTRICT ON UPDATE
CASCADE,

    CONSTRAINT `EnrolmentFKresultId`

    FOREIGN KEY (`resultId`) REFERENCES `Result` (`resultId`) ON DELETE RESTRICT ON
UPDATE CASCADE

       ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```sql
ALTER TABLE Enrolment

ADD INDEX EnrolmentINDstudentIdSectionTypeIdIsActive (studentId, isActive, sectionTypeId)
USING BTREE;

ALTER TABLE Enrolment

ADD INDEX EnrolmentINDstuidResidSectypidIsact (studentId, isActive, resultId,
sectionTypeId) USING BTREE;
```

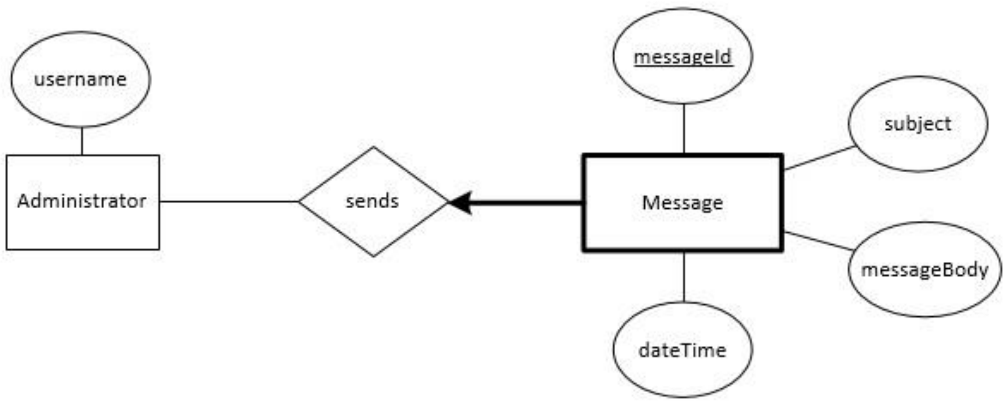


Figure 14 - ER Diagram (Message)

SQL Script

```
DROP TABLE IF EXISTS `Message`;
CREATE  TABLE `Message` (
  `messageId` INT NOT NULL AUTO_INCREMENT,
  `subject` VARCHAR(250) NULL ,
  `messageBody` VARCHAR(5000) NULL ,
  `senderId` VARCHAR(15) NOT NULL ,
  `dateTime` DATETIME NOT NULL ,
  PRIMARY KEY (`messageId`) ,
  CONSTRAINT `MessageFKsenderId`
    FOREIGN KEY (`senderId` )
    REFERENCES `User` (`username` )
    ON DELETE RESTRICT
    ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
ALTER TABLE Message AUTO_INCREMENT = 100;

ALTER TABLE Message
ADD INDEX MessageINDdateTime (dateTime) USING BTREE;
```

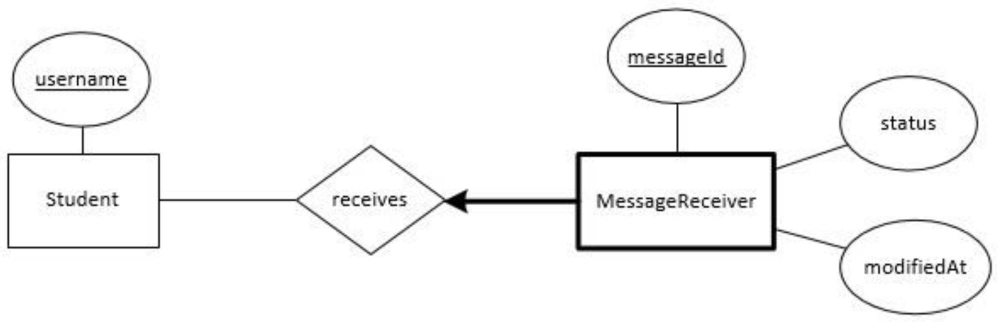


Figure 15 - ER Diagram (Message Receiver)

SQL Script

```
DROP TABLE IF EXISTS `MessageStatus`;
CREATE  TABLE `MessageStatus` (
  `statusId` INT NOT NULL ,
  `name` VARCHAR(15) NOT NULL ,
  PRIMARY KEY (`statusId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE MessageStatus
ADD UNIQUE INDEX MessageStatusINDname (name) USING BTREE;

DROP TABLE IF EXISTS `MessageReceiver`;
CREATE  TABLE `MessageReceiver` (
  `messageId` INT NOT NULL ,
  `receiverId` VARCHAR(15) NOT NULL ,
  `statusId` INT NOT NULL DEFAULT 0 ,
  `modifiedAt` DATETIME NOT NULL ,
  PRIMARY KEY (`messageId`, `receiverId`) ,
  CONSTRAINT `MessageReceiverFKmessageId`
    FOREIGN KEY (`messageId` )
    REFERENCES `Message` (`messageId` )
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `MessageReceiverFKreceiverId`
    FOREIGN KEY (`receiverId` )
    REFERENCES `User` (`username` )
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT `MessageReceiverFKstatusId`
    FOREIGN KEY (`statusId` )
    REFERENCES `MessageStatus` (`statusId` )
    ON DELETE RESTRICT
    ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

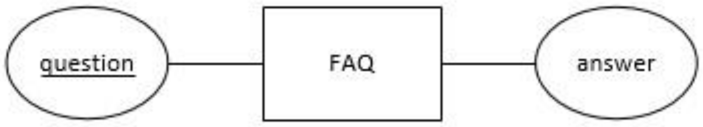


Figure 16: ER Diagram (FAQ)

SQL Script

```
DROP TABLE IF EXISTS `FAQ`;
CREATE  TABLE `FAQ` (
  `question` VARCHAR(250) NOT NULL ,
  `answer` VARCHAR(5000) NOT NULL ,
  PRIMARY KEY (`question`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

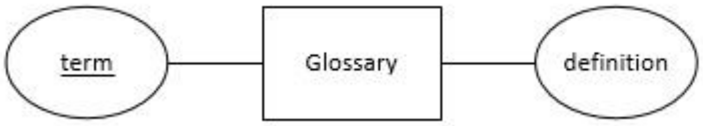


Figure 17: ER Diagram (Glossary)

SQL Script

```
DROP TABLE IF EXISTS `Glossary`;
CREATE  TABLE `Glossary` (
  `term` VARCHAR(100) NOT NULL ,
  `definition` VARCHAR(1000) NOT NULL ,
  PRIMARY KEY (`term`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

4. Glossary

A:

- Admin: is the secondary target user for the system who is an administrator at the university and/or is assigned an administrative access such as program advisors, directors and registration officers.
- API: Application Program Interface.

C:

- Course: the User generally views course as a literal meaning. A course in the RED system contains all the information about a particular classroom that the student can register is such as: department name, course number, instructor, hours, credits, prerequisites, etc. A course from the developer stand point is an entity (an object).
- Controller Objects: The Object of Controller subsystem in MVC architecture, which can send commands to its associated view objects to change the corresponding views objects of the model thus the change of GUI as well as send commands to the model objects to update the model's state.
- Corequisite: A co-requisite for a course states a condition or conditions which must be satisfied concurrently with registration in the course. This condition will normally be registration in a specified course unless credit is already held for the co-requisite course.
- Credit: Depending on the specific course, students obtain credits for each successfully completed course.

D:

- Degree: is the acknowledgement a student will receive after finishing each specific program.
- Department: is the name of the particular branch of the faculty that offers courses to students surrounding relevant topics to the specialization of that particular department.

E:

- Enrolment: Students may enrol in a degree, diploma, or university certificate program. The regulations in effect at the time of your initial enrolment are the regulations that govern your program.

F:

- Faculty: is the third target user for the system who is a faculty member or an instructor at the university and/or is assigned a faculty access.
- FAQ: Frequently Asked Questions are listed questions and answers, all supposed to be commonly asked in some context, and pertaining to a particular topic.

G:

- Grade: Is a particular level of rank. In RED system, the grade has 6 levels: A, B, C, D, E, F.
- GUI: Graphical user Interface.

J:

- JavaFX: A software platform for creating and delivering software system that can run across a lot of hardware devices.
- JDBC**:** This acronyms stand for Java Database Connectivity API which allows for standardized relational database access.

L:

- Laboratory: A laboratory is a facility that provides controlled conditions in which scientific research, experiments and measurement may be performed.
- Language: RED system provides users with three kinds of languages, which includes English, Chinese and Thai.
- Lecture: A lecture is an oral presentation intended to present information or teach student about a particular subject by a university professor or lecturer. Lectures are used to convey critical information, history, background, theories and equations.

M:

- Model Object: The Object of Model subsystem in MVC architecture, which include the persistent and long-lived information of the system.
- MVC Architectural Style: MVC stands for Model/View/Controller. In this three-tier architecture, the domain knowledge is maintained by model objects, displayed by view objects and implemented by controller objects.
- MySQL: Open source database management system from Oracle to maintain the database subsystem of the system.

N:

- Netbeans: An integrated development environment (IDE) from Oracle for developing primarily with Java in the system.

O:

- Outstanding Co-Requisite: The Co-Requisite courses that are not satisfied.
- Outstanding Prerequisite: The prerequisite courses that are not satisfied.

P:

- Personal Info: The account information of the user, including address, city, province, country, phone number, email, as well as the password information.
- Prerequisite: A prerequisite for a course states a condition which must be satisfied prior to registration in the course. This condition may consist of a) obtaining credit for another course or other courses, or b) having a particular registration status, such as registration in a program and/or in a specified year, or as a senior student (i.e., a student who has obtained 6.0 or more credits), or c) having a minimum GPA (e.g., thesis courses).
- Program: is a set of courses that a student has to take to specialize in a particular field.

R:

- RED: the codename for the student enrolment system.

S:

- Score: The percentage score each student obtained for the exam. This value is the score obtained out of a hundred. It is based on the scoring factors and is not the same as the % Correct.
- Search: Using the search function that RED system provided to find an item with specified properties (e.g. specified course) among a collection of items.
- Section: Each course will have one or more sections. A section is the particular class a student registers in. Characteristics of a section include the time, place and Term when it is taught and the professor that teaches it. Each section has enrolment limits, so some sections might be full, while others might be available.
- Session: There are three academic session in the calendar year: Fall: (12 weeks) September – December; Winter: (12 weeks) January – April; Summer: (12 weeks) May – August; The number of weeks listed above for each session are teaching weeks.
- Special Approval: Faculty is able to approve special request from a student to enrol the student in any section of the courses that he is teaching for the current session.
- Student: is the primary user for the system who is registered at the university and uses the RED system to register/drop courses, checks their enrolment and grade records, view course information and more.
- SQL: This acronyms stand for Structured Query Language as a special-purpose programming language to managing data held in a relational database management systems (RDBMS).

T:

- Term: Is composed of academic year sessions.
- Timetable: The timetable allows the user to view the sections that they are enrolled in in timetable format. Standard Timetable entries with display and course conflicts will show in red.
- Tutorial: A small class in which a teaching assistant or instructor gives intensive instruction or academic help in addition to a regular lecture.

U:

- User: is anybody who interacts with RED
- User Manual: A user manual is a technical communication document, a guidance that intends to give assistance to user using the RED system.

V:

- View Object: The Object of View subsystem in MVC architecture, which determines the GUI of the system and communicate with the user input directly.