**Group 2**

Course Management & Study Planner

**Ikjun Jang(ixj120230)**

**Mohit Mittal(mxm138630)**

**Swati Sharma(sxs136131)**

**Jinisha Shah(jas130630)**

**Rachita Agasthy(rxa124130)**

Table of Contents

1. [Purpose of Document](#_Toc340506951) 2
2. [System Requirements](#_Toc340506956) 2
3. [Domain Model](#_Toc340506973) 4
4. [Use Case Model](#_Toc340506951) 7

a. Use Cases

b. Use Case Diagrams

c. System Sequence Diagrams

d. Operation Contracts

1. [Design Model](#_Toc340506956) 24

a. Interaction Diagrams - Sequence Diagrams

b. Design Class diagram

c. Patterns Used and explanation

1. [Software Snapshots](#_Toc340506968) 36
2. [Lessons Learnt](#_Toc340506968) 42

# Purpose of Document

This document describes the design of Course management and Study planner system. First we list the functional and supplementary requirements of the system followed by the domain model. We describe the components of the domain model and the class diagram we have. Then we have the use case model where we identify the different users and use cases with respect to them. Use case model has the use case texts, use case diagrams. We have also come up with system sequence diagrams for the use cases and operation contracts for the use cases that are a little complicated.

The next part of the document has the sequence diagram which are interactions diagrams for our system describing how the classes and objects would interact. Then we also have the class design diagram representing the actual classes that we have used in the code. At the end of the document we have included the snap shots of our working system.

# System Requirements

## Vision

As we all know there is already an existing course management system that is being used by our university. One of the major problems faced by students while registering for courses online is that, if the courses have pre requisites the students are not allowed to register online. The manual process of registration is tiring for both students and advisors who help in registration. In our system we aim to provide an interface for both students and advisors to help in the course registration process.

The other thing that we have focused on is based on the activities part of the courses like assignments, home works, projects and their deadlines we will be creating a study plan for students which would create a time table for them helping them complete the activities in time.

## Functional Requirements

The tool will be mainly focused on providing an online portal to Students, Professors, Student Advisors and Administrators helping them in the process of course management, course activity management, handling student requests and offered course management respectively.

### Student Requirements

1. Student should be able to register/drop/swap for courses online.
2. In case of course registration, system should be capable of checking if the student is registering for a course outside his/her program and also check if he has satisfied the prerequisites for the course being registered. If not a request for registering should be generated and sent to the advisor.
3. In case of swapping a course, the system should be able to register for the new course first and then the student must be dropped from the already registered course.
4. Student should be able to view the activities related to the course that he/she has registered for and then based on the selected activities the student should have an option to create a study plan to complete the activities in time.
5. Once the study plan is generated the student should be able to save or delete the study plan.

### Professor Requirements

1. Professor should be able view the courses that he/she is offering and course activities related to them.
2. Professor should also be able to add/edit/delete a course activity related to a course that he/she is offering.

### Advisor Requirements

1. Advisor should be able to view the requests generated by the system on behalf of the students while registering for a course.
2. Advisor should also be provided with an interface to accept or reject the requests based on the priority.

### Administrator Requirements

1. Administrator should be able to add/delete/edit details regarding courses.
2. Administrators will also be able add/delete/edit the offered courses details.

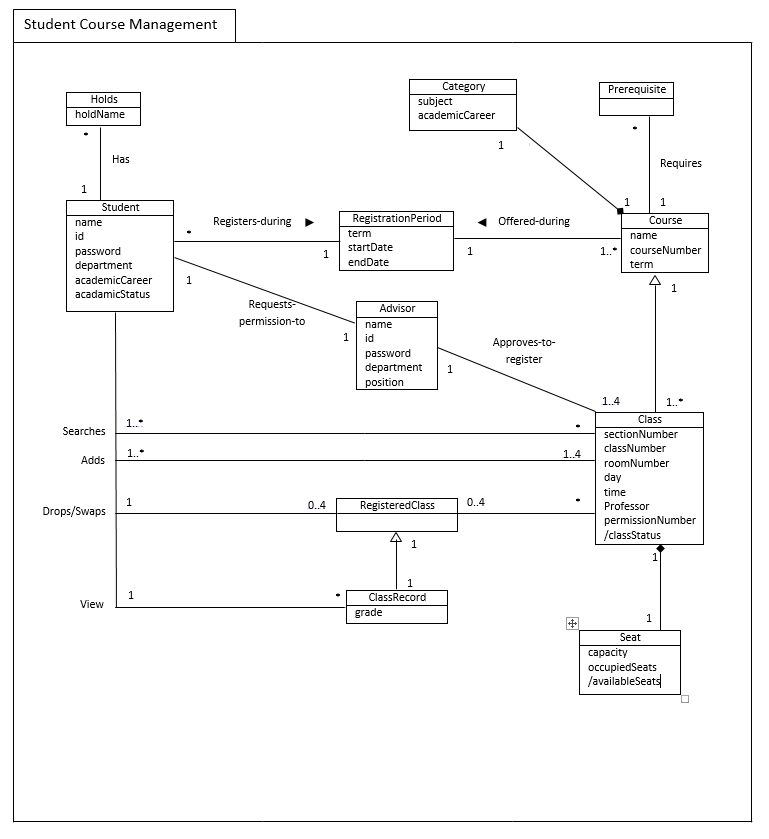
## Supplementary Specifications

1. System should be available as a web application.
2. Logging and error handling – All the users must be able to login to the system and have their respective views displayed to them.
3. Usability – Users must have a clear view of the web page. The components should be sized appropriately so that the text are readable and understandable.
4. Performance – The response time of the system should be within one to two seconds, system should not keep the user waiting for a long time.
5. Technologies – Plan to use most of the open source software – apache tools, Java, Maven and MySql so on.
6. Reliability – If the system says a student has been registered for a course then the registration should be successful. In case of swap of course, if the student is not able to register for a new course he/she should not be dropped from the already registered course.

# Domain Model

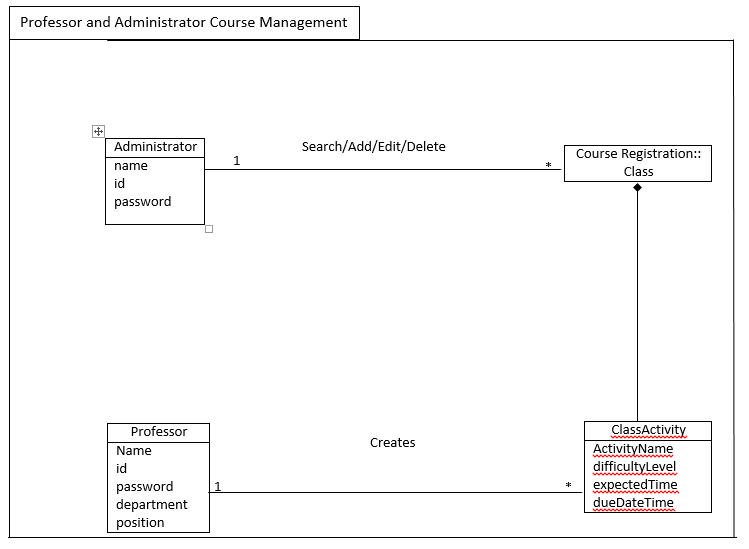
Domain model is a very classic method of modelling in object oriented analysis. Apart from capturing the details of a domain it inspires design of other software objects. Here is the domain model for our system.

Course Management System and Study Planner



* What has been revised from previous student course management package

-We have excluded the cart and waitlist mainly because we wanted to focuse more on core functionalities such as study planner and student request handling.



* What has been included from previous domain model

-We have included domains and associations for administrator and professor course management.

* What has been omttied from previous domain model

-We have omitted study planner package as we have not needed to revise any contents in the package.

Glossary

* academicCareer: Graduate/Undergraduate
* academicStatus: Good/Probation/Drop
* Subject: Field of study (e.g., computer science)
* classStatus: Open/Closed

# Use Case Model

## High Level Use Cases

1. **Use Case Name**: Register Courses

**Scope**: Online Course Management System

**Level:** User-goal

**Actors**: Student

**Main Flow**:

* Student can login into the system and check the course list.
* The student can request the advisor to get enrolled in particular class.

1. **Use Case Name**: Swap/Drop Courses

**Scope**: Online Course Management System

**Level**: User-goal

**Actors**: Student

**Main Flow**:

* User can login into the system and check the course list.
* The user can swap a course with any course in the course book list.
* He can request the system to drop course which is there on his registered course list.

1. **Use Case Name**: Add/Edit/Delete Courses

**Scope**: Online Course Management System

**Level**: User-goal

**Actors**: Admin (Primary)

**Main Flow**: The admin can add/delete/edit course

1. **Use Case Name**: HandleRequest

**Scope**: Online Course Management System

**Level**: User-goal

**Actors**: Advisor (Primary)

**Main Flow**:

* The advisor receives student request for adding course.
* Based on certain criteria the advisor decides to accept/reject student request

1. **Use Case Name**: Create study planner

**Scope**: Online Course Management System

**Level**: User-goal

**Actors**: Student (Primary), System (Secondary)

**Main Flow**:

* The student logins into the system and requests the study planner.
* Based on the course activities the system generates the study planner for student.

1. **Use Case Name:** Add/Edit/Delete Course Acitivities.

**Scope**: Online Course Management System

**Level:** User-goal

**Actors**: Professor (Primary)

**Main Flow**:

* The Professor can add details to the course he is taking at any time. This can include updating the deadline, adding assignment

## Low level Use Cases

1.

**Use Case Name**: Register Course

**Scope**: Online Course Management System

**Level**: User-goal

**Actors**: Student

**Stakeholders**:

* Student: Login into the system, Check the courses, Add Courses, Delete Courses

Preconditions: The user should have a valid username and password to login to the system.

**Main Success Flow**:

1. The User can login into the system.
2. View all the details such as section number, the class number, professor teaching the course.
3. He can also view the class capacity and wait list.
4. Advisor approves the student request to add a particular class to his course list.

**Alternate flows (details left out)**:

* Invalid User
* Wrong course request
* Insufficient details
* Quit

**Post-Conditions (Success Guarantee):**

* Successful Completion
* The user has added course and database is updated.
* Failure Condition
* The database has been updated accordingly.

**Special Requirements:**

The system update database every time when there is an acceptance or rejection.

2.

**Use Case Name**: Swap/Drop Courses

**Scope:** Online Course Management System

**Level:** User-goal

**Actors:** User (Primary), System(Secondary)

**Stakeholders:**

* User: Login into the system, Check the courses list, Swap course or drop course
* System: Allows the student to swap or drop particular course.

**Preconditions:**

* The user should have a valid username and password to login into the system.
* To swap or drop course, the user should have some course registered on his system

**Main Success Flow**:

1. The User can login into the system.
2. For Swapping, he can check the course list.
3. After checking the course list, he can swap with any course registered on his system.
4. The system approves the swap done by the student.
5. For dropping any course, user can request the system for a drop.
6. The system approves the student request
7. That particular course is dropped from user’s registered course list

**Alternate flows (details left out):**

* Invalid User
* Performing wrong swap
* Insufficient details
* Quit

**Post-Conditions (Success Guarantee):**

* Successful Completion
* The user has swapped course and database is updated.
* Failure Condition
* User has successfully dropped a course
* The database has been updated accordingly.

3.

**Use Case Name:** Add/ Delete / Edit Courses

**Scope:** Online Course Management System

**Level**: User-goal

**Actors:** Admin (Primary)

**Stakeholders:**

* Admin: Login into the system and perform some actions

Preconditions: The admin should have a valid username and password to login into the system.

**Main Success Flow:**

1. The admin logs into the system.
2. Admin can add a particular course.
3. Admin can delete a particular course if there are not many students enrolled in the course.
4. Admin can edit the course such as the course timings, class in which the course is taking place, etc.

**Alternate flows (details left out):**

* Invalid admin
* Quit the system

**Post-Conditions (Success Guarantee):**

* Successful Completion
* The admin added a course.
* The admin has deleted a course.
* The admin adds invalid course details.
* The database has been updated accordingly.

**Special Requirements**:

The system updates database every time if any add, delete or edit activity is performed.

4.

**Use Case Name**: Handle Requests

**Scope:** Online Course Management System

**Level:** User-goal

**Actors**: Advisor (Primary)

**Stakeholders**:

* Advisor: Login into the system, receives the student request and decides to accept/reject.

Preconditions: The admin should have a valid username and password to login into the system.

**Main Success Flow**:

1. The advisor logs into the system.
2. Advisor receives student request to get added to a particular course.
3. Advisor checks if the student has any pre-requisites.
4. If the student has a pre-requisite for the course requested, the advisor rejects the student’s request.
5. Else he approves the student request.

**Alternate flows (details left out)**:

* Invalid advisor.
* Failure in checking the pre-requisite

**Post-Conditions (Success Guarantee)**:

* Successful Completion
* The advisor approves the student request.
* The database has been updated accordingly.

**Special Requirements**:

The system must have a proper interface to existing database.

5.

**Use Case Name**: Create Study Plan

**Scope**: Online Course Management System

**Level:** User-goal

**Actors**: Student (Primary)

**Stakeholders**:

* Student: Logs in and request study planner
* System: Generates the study plan

**Preconditions:**

* The student should have a valid username and password to log in.
* The student should have certain courses registered on his system.

**Main Success Flow**:

1. The student logins into the system.
2. After logging in, the student requests for the study planner.
3. The system checks all the course activities of the student like when is the next assignment due, which is the next upcoming exam.
4. Based on all this, the system will generate a study planner for the student.

**Alternate flows (details left out):**

* Invalid username
* No course on the system and student requests the study plan

**Post-Conditions (Success Guarantee):**

* Successful Completion
* Generation of study plan

**Special Requirements**:

The course activities of student should be accessible.

6.

**Use Case Name**: Add/ Edit/ Delete Course Activities

**Scope:** Online Course Management System

**Level:** User-goal

**Actors**: Professor (Primary)

**Stakeholders:**

* Professor: Adds course details including homework’s, project details, marks, etc.

Preconditions: The professor should have valid username and password to login into the system and edit the course details.

**Main Success Flow**:

1. The Professor logs in into the system
2. If there is any change in the submission deadline he updates in the course activity.
3. At the end of the term he assigns grade to each student.

**Alternate flows (details left out)**:

* Invalid username
* Wrong date of submission
* Delay in giving grades

**Post-Conditions (Success Guarantee)**:

* Successful Completion

**Special Requirements**:

The student should get an email every time there is some change in the course activity.

## Use Case Diagram

Student

Course Management and Study Planner System

Advisor

Administrator

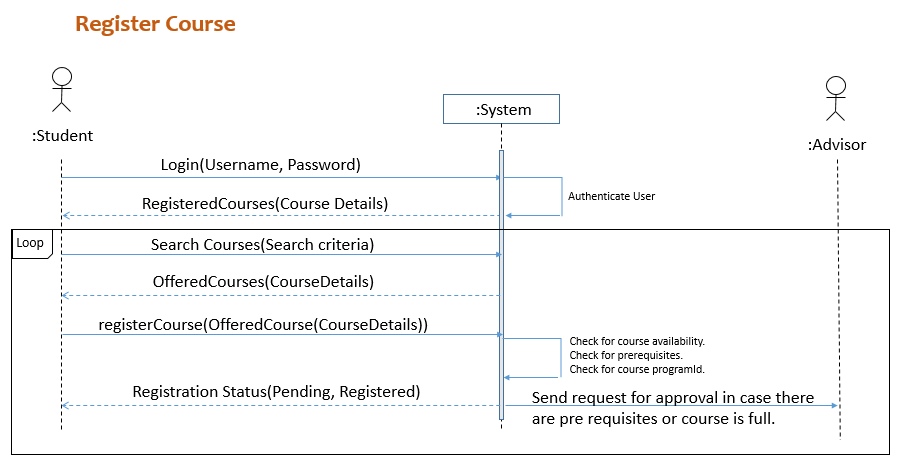
Course Management and Study Planner System

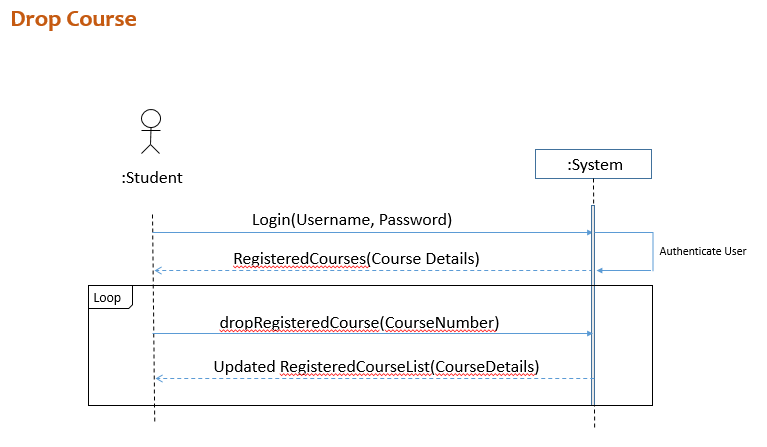
Professor

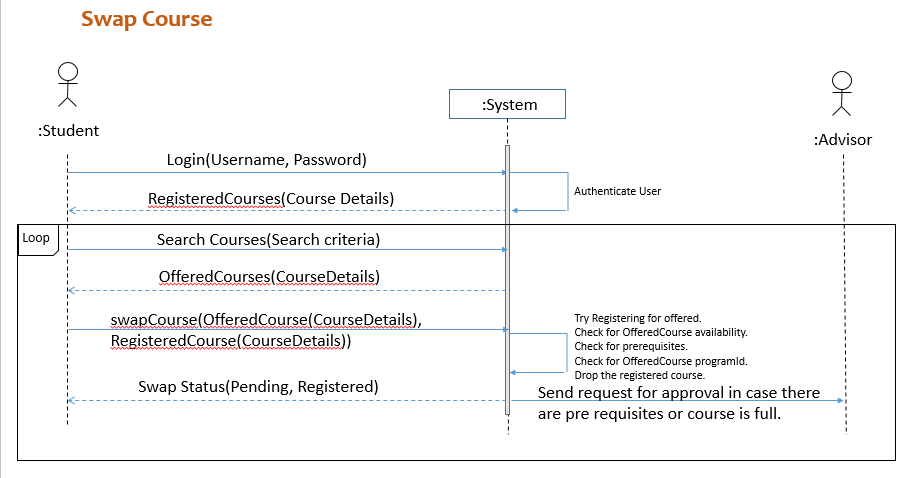
# System Sequence Diagram

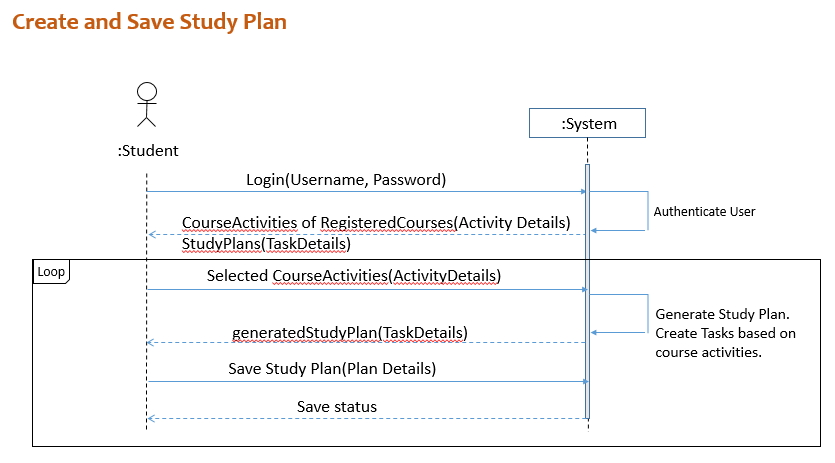
These are diagrams which depict input and output to the system in question based on particular use case scenarios. This aides in creation of operation contracts and object design.

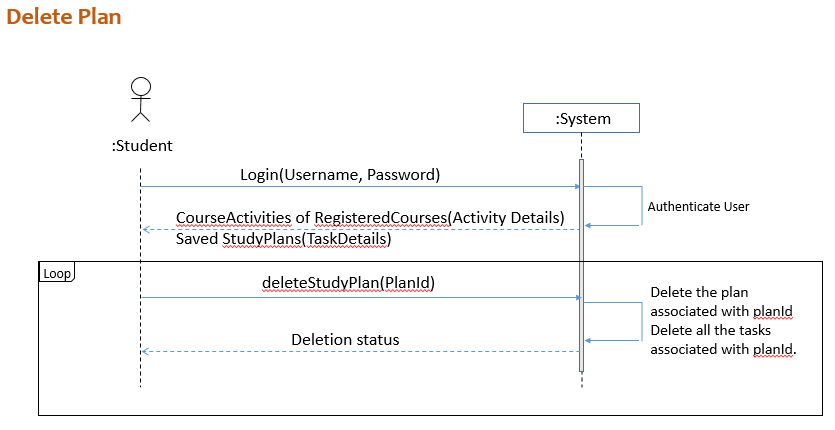
## Student Related



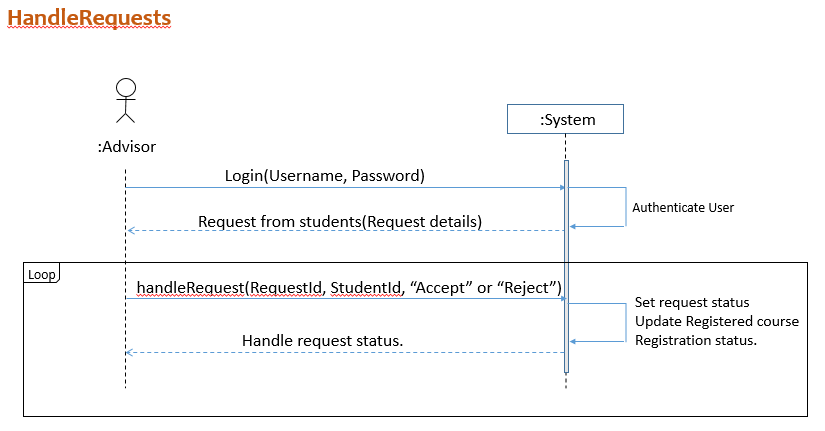




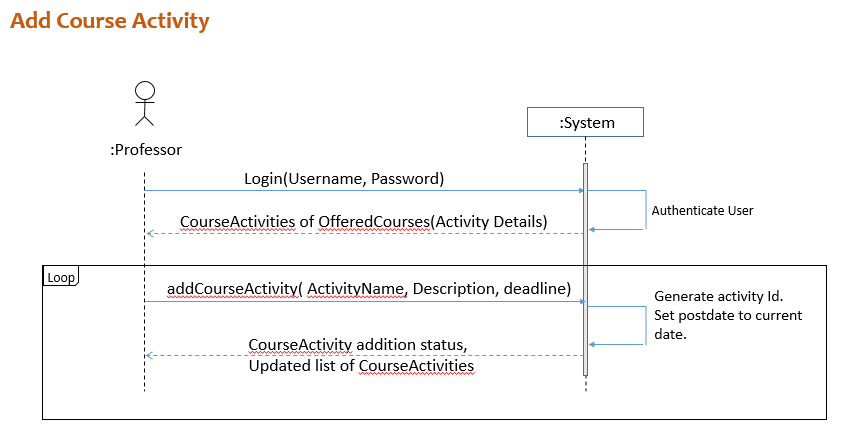


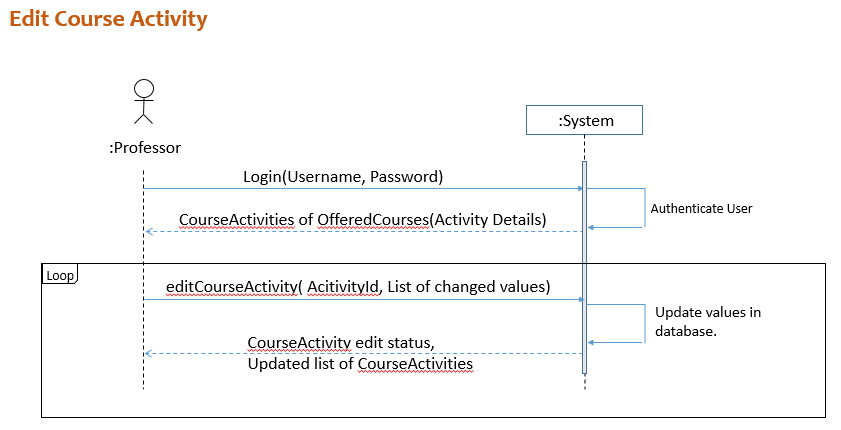


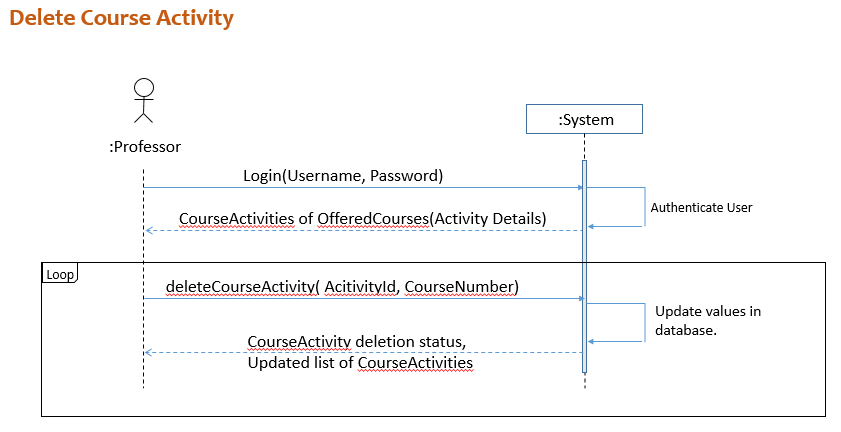
## Advisor related



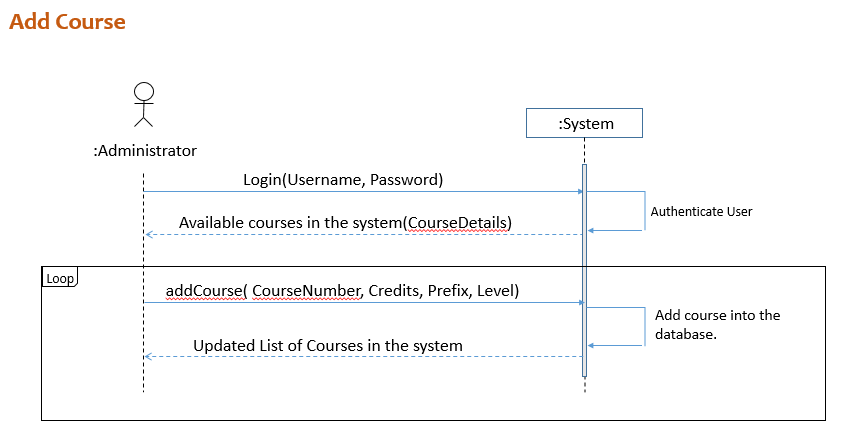
## Professor Related

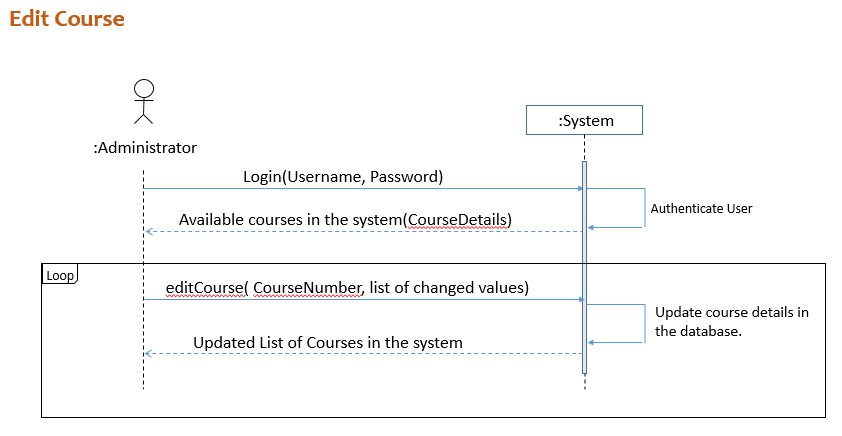


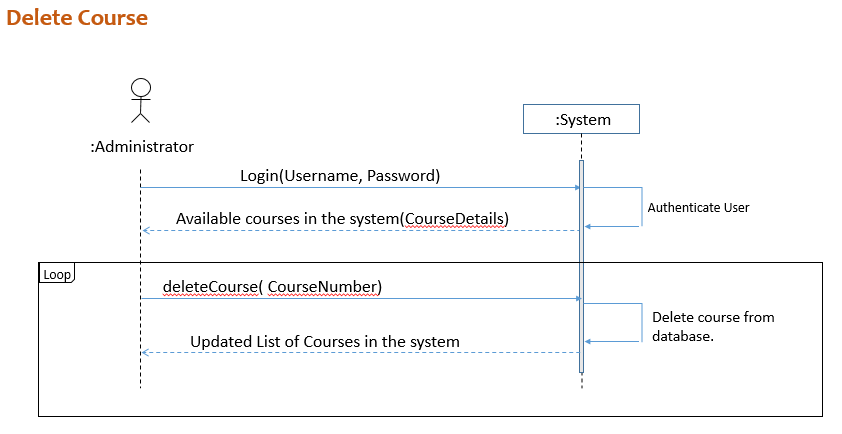


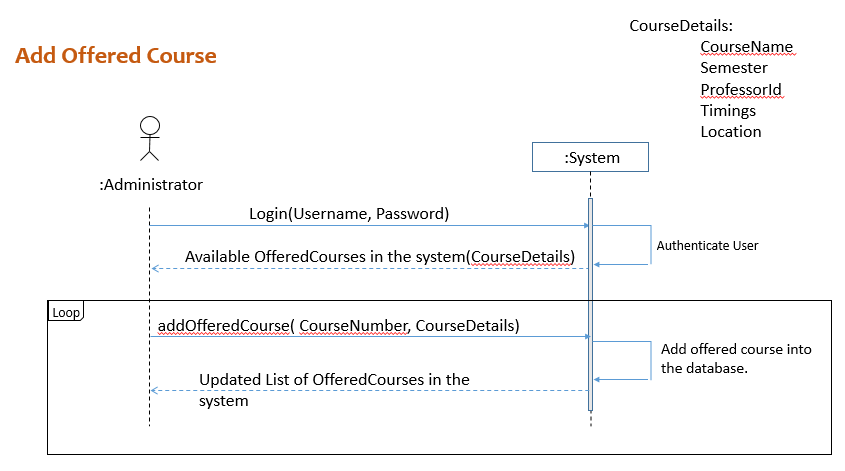


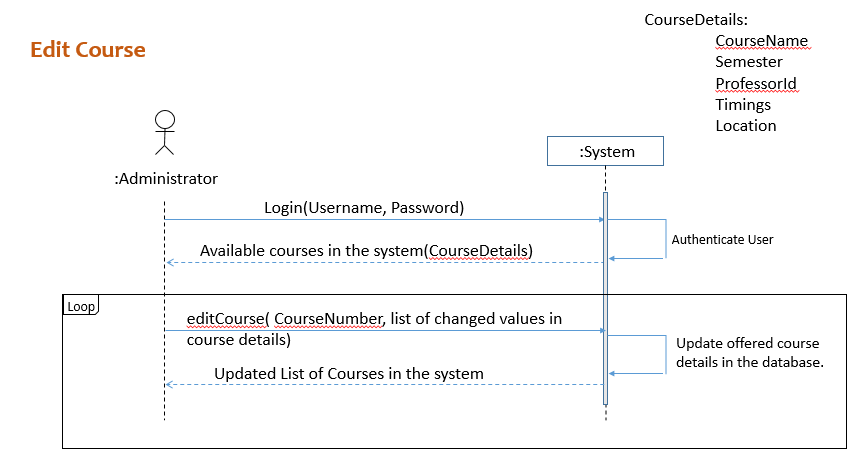
## Administrator related

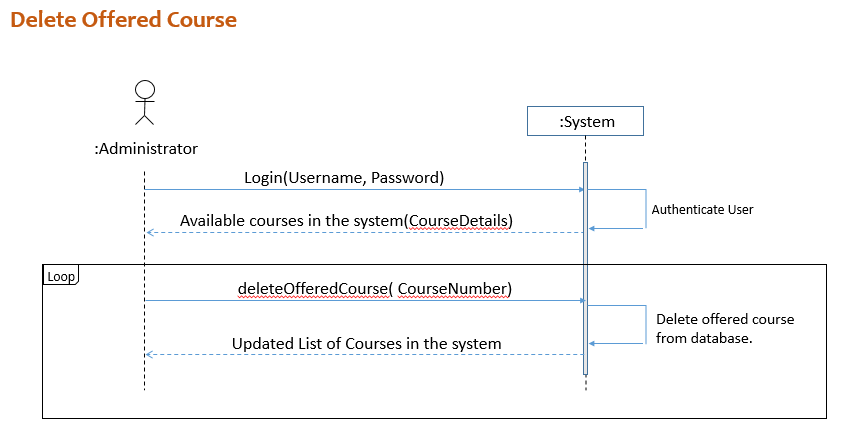












# Operation Contracts

Operation contracts are usually used to describe processes that are hard to understand. We have used operation contracts to describe the register course, swap course and create study planner use cases which are a bit complicated to understand just using the use case diagrams and system sequence diagrams.

**OC1 : registerCourses**

Operation: registerCourses

Cross references: Use case - Student

Pre-condition:

* The course should be offered in the current semester.
* Course should be open for registration and there must be seats available for the course offered.

Post-condition:

* RegisteredCourse instance was created.
* The course instance was associated with the Student.
* If the student has not satisfied the prereqs or registering for a course outside his/her program a Request object for the advisor is created.
* The seats available attribute of the offered course is reduced by 1.
* If the seats reduced attribute is 1 the course status attribute is set to closed.

**OC2 : swapCourses**

Operation: swapCourses

Cross references: Use case - Student

Pre-condition: The Student must have courses registered for that particular term.

Post-condition:

* Post conditions of the registerCourse contract holds here also for the registeredCourse instance created.
* The seats\_available is increased by 1 for the already registered course instance.
* If the course\_status was closed then it is changed to available.

**OC3: createStudyPlan**

Operation: createStudyPlanner

Cross references: Use case - Student

Pre-condition:

* The student must be registered with some courses for the semester.
* The registered course should have some course activities associated with them.

Post-condition:

* an instance of StudyPlan is created
* For every task in the study plan, an instance of task is created.
* The created instance of study plan in associated with the user.
* The created instances of task are also associated with the user.

**OC4: handleRequests**

Operation: handleRequests

Cross references: Use case - Advisor

Pre-condition:

* The advisor should have requests from students whose status are pending.
* The requests should be ordered based on the request generation timings.

Post-condition:

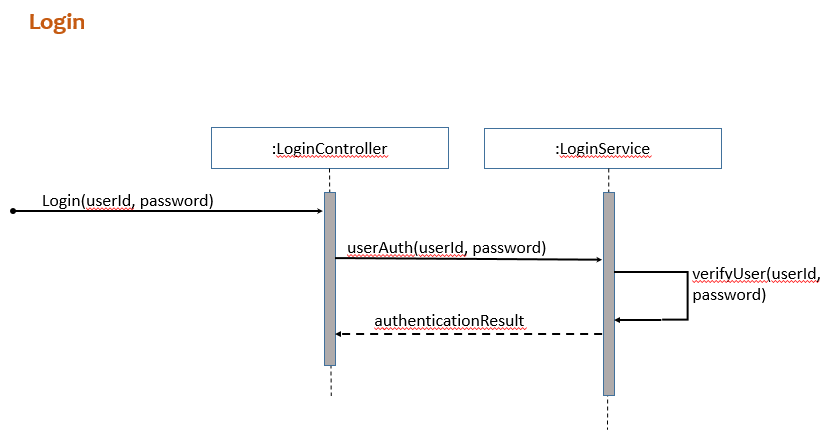
* The studentRequest to advisor association is removed/added based on accept/reject decision made by the advisor.
* The registration status of the registered course instance associated with the student is updated to registered or dropped based on accepted or rejected decision made by the advisor respectively.

# Design Model

## Sequence diagrams and Use case realization with respect to them.

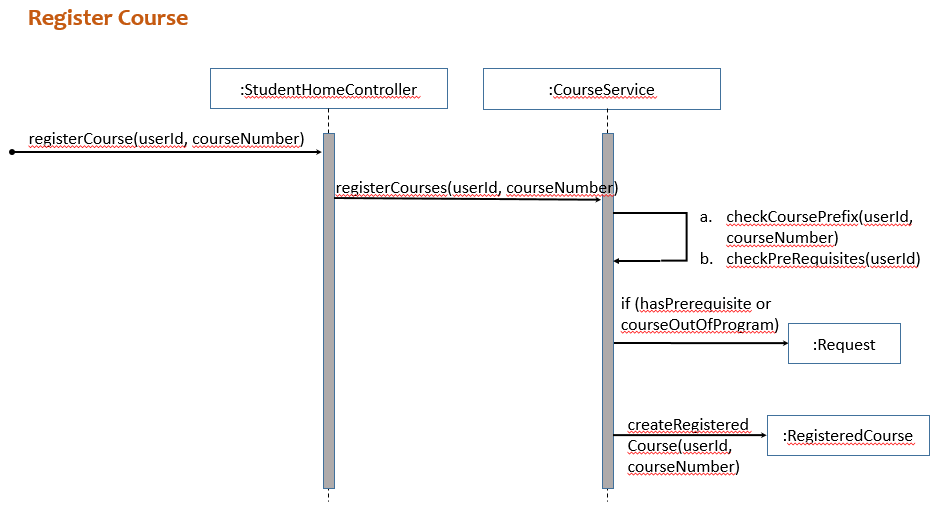
#### Login use case

All the users login to the system and a view based on their role is displayed.

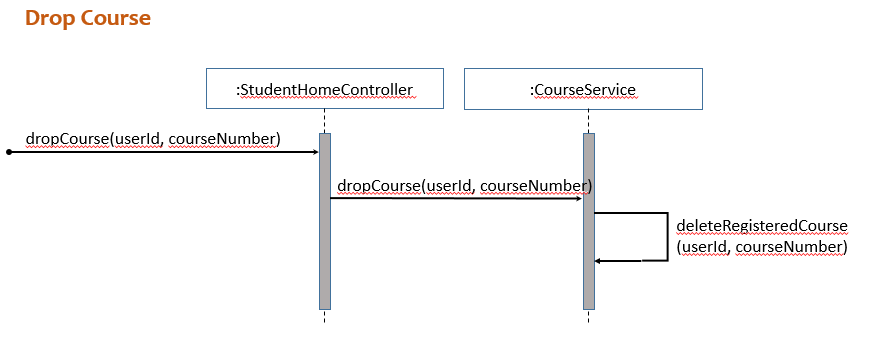


#### Student Use cases

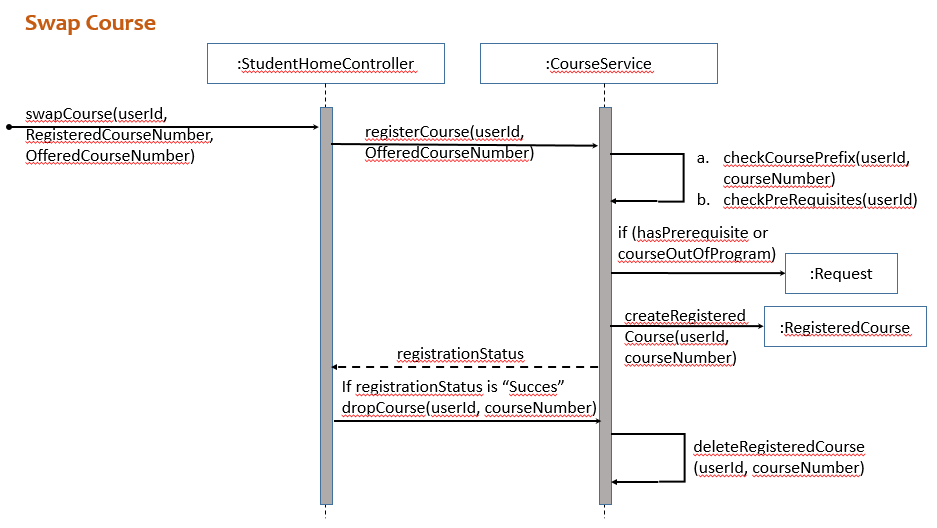
1. The student can register for offered courses and below are flow that takes place in the software which realizes this process.



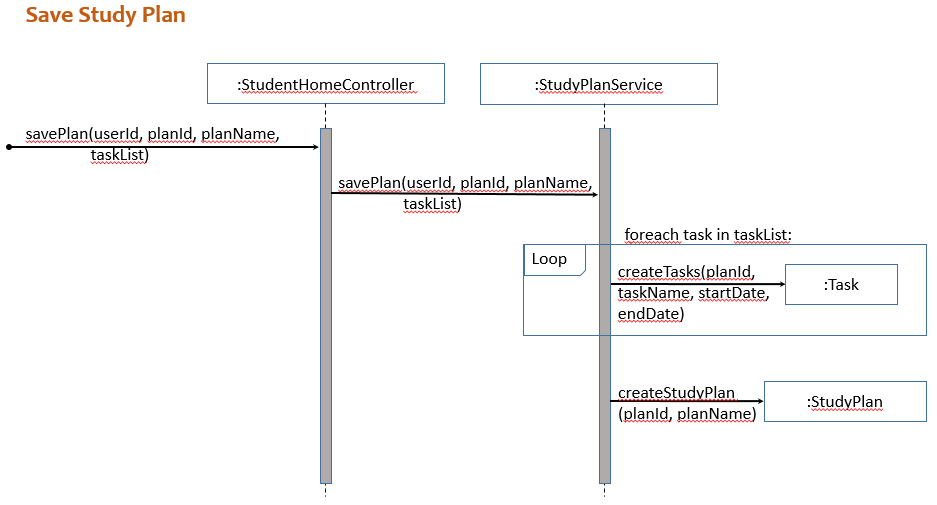
1. Student can drop the course and here are the sequence of events.



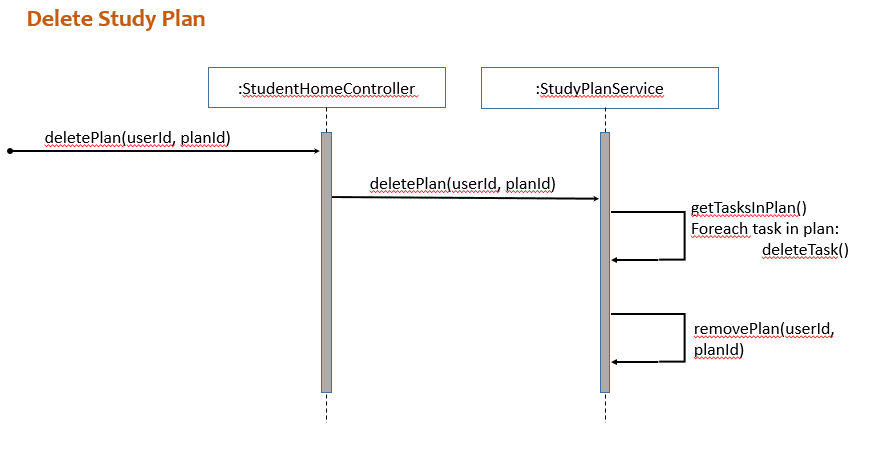
1. Student can swap a RegisteredCourse with an OfferedCourse and the following are the steps involved.



1. Student can create a study plan and save the same.

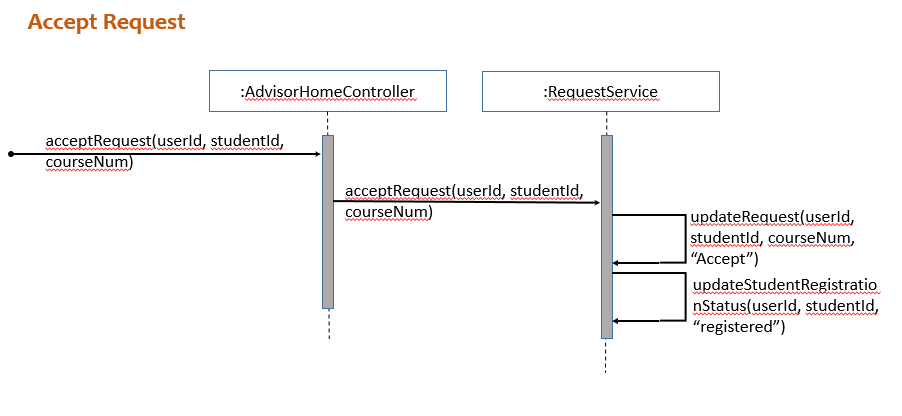


1. Student can delete a study plan.

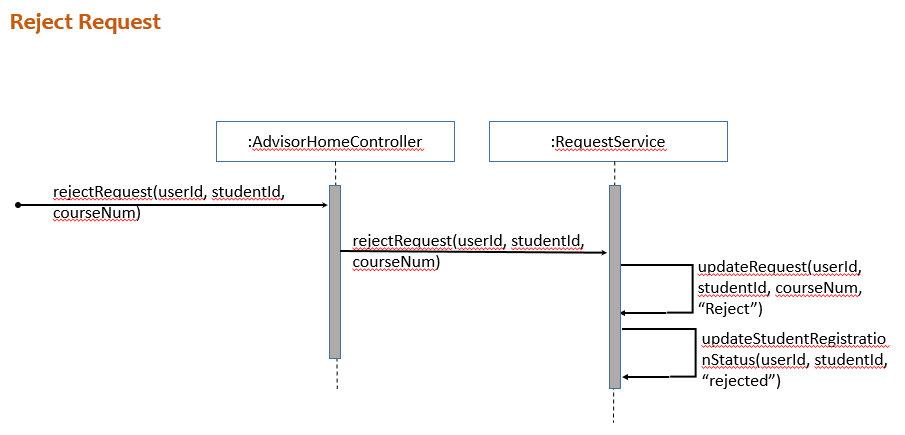


#### Advisor Related

1. Advisor can accept a request that is presented to him.

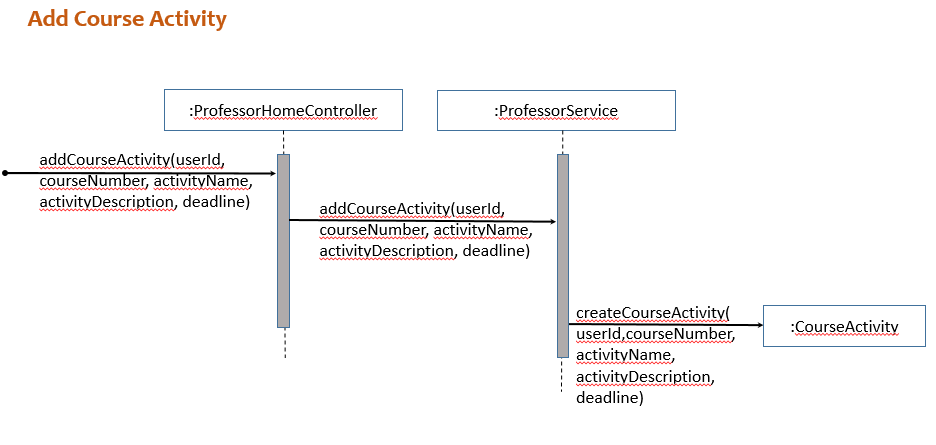


1. Advisor can reject a request that is posed to him.

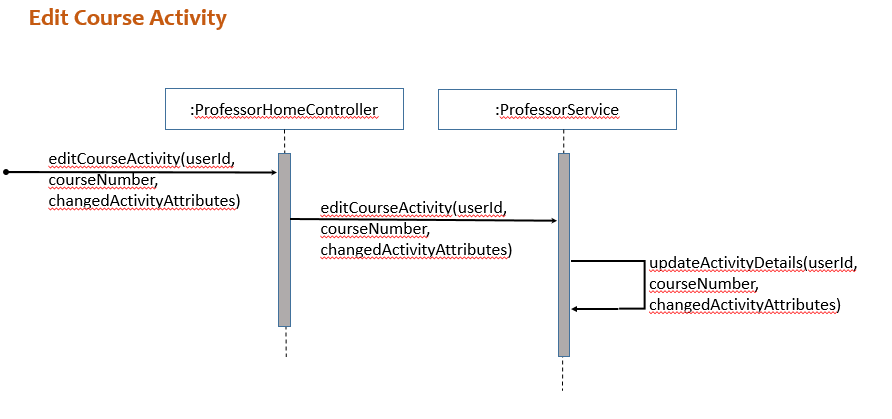


#### Professor Related

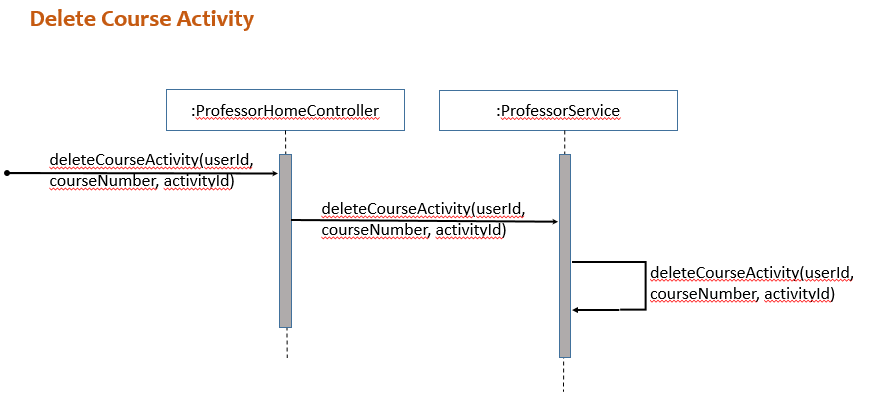
1. Professor can add a course activity related to the course that he is offering.



1. Professor can edit a course activity.

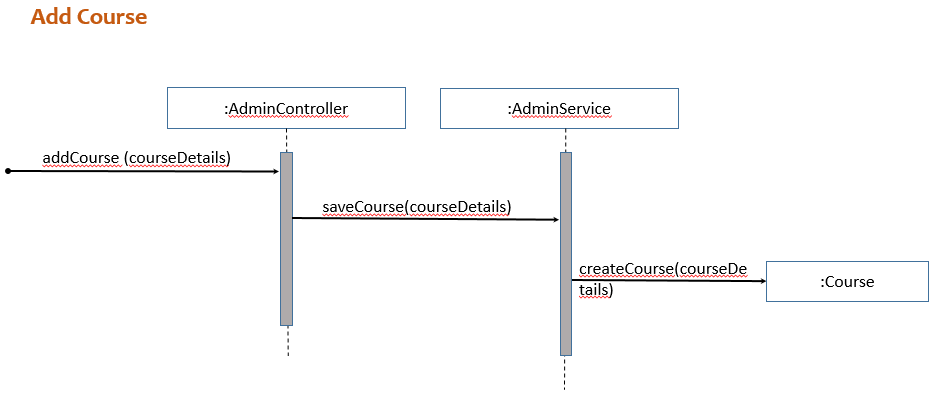


1. Professor can edit a course activity.

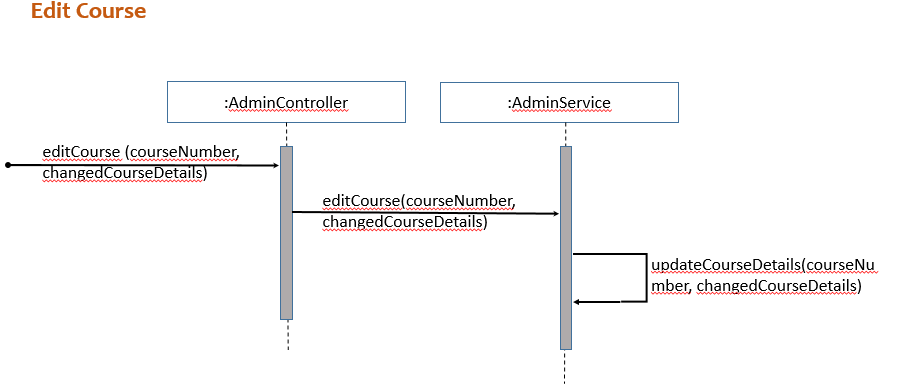


#### AdministratorRelated

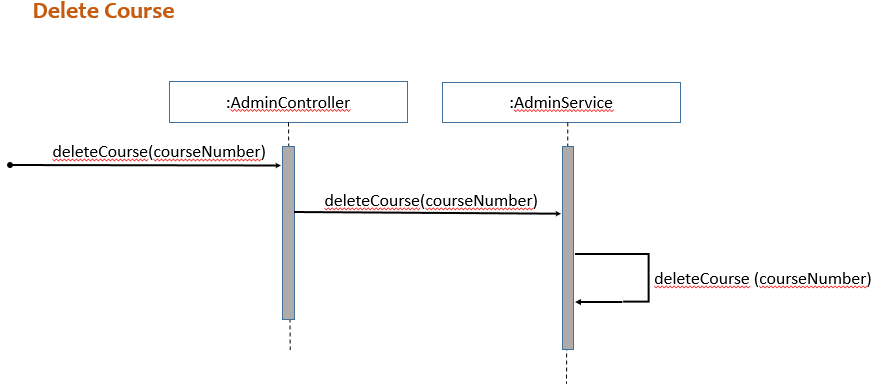
1. Add Course



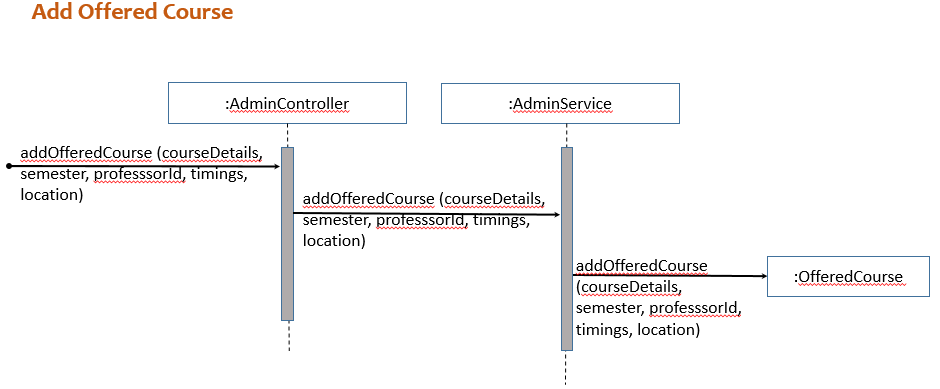
1. Edit Course



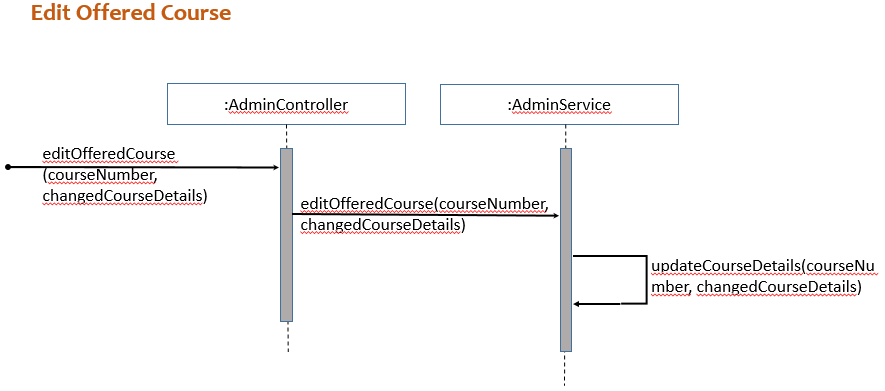
1. Delete Course



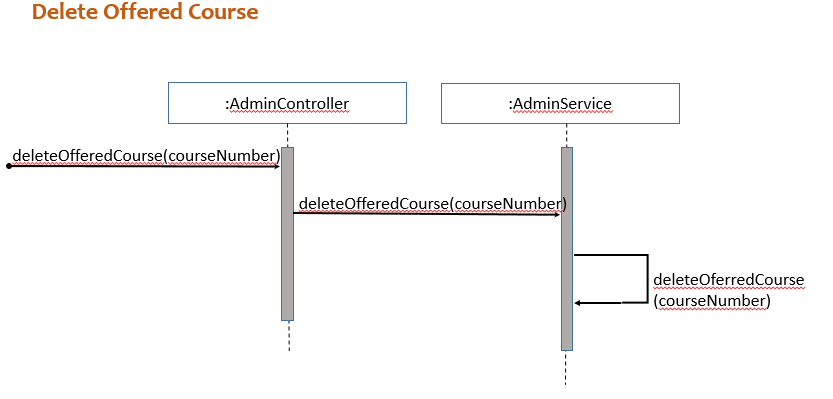
1. Add OfferedCourse



1. Edit OfferedCourse

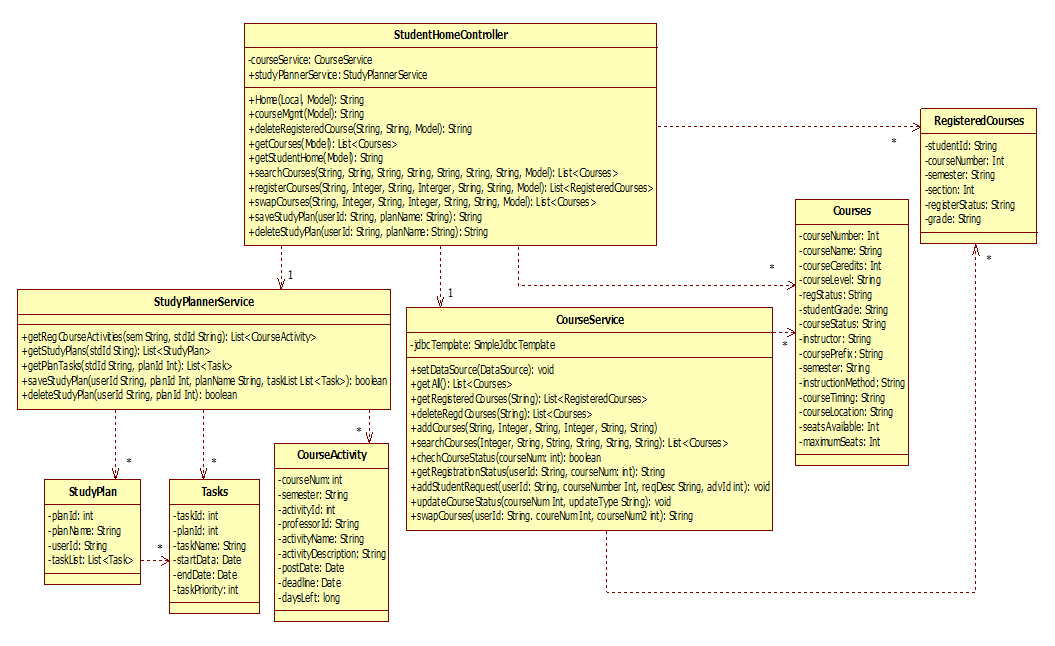


1. Delete OfferedCourse



# Design Class Diagram

The diagram below shows all the classes that were used in implementation of course management for Student and study planner creation for student.



Below are the classes that help in course activity management for the professors.



Classes that help in implementation of advisor accepting or rejecting requests from students.



The classes used for the implementation of Admin related use cases where he can add/delete or edit courses and offered courses are as shown below.



* We have a controller class for each user which receives the request for a particular process from the UI.
* The Controller then makes a call to the method which is part of service class that helps in completing a particular process.
* The methods in the service class in turn contacts the database for modifying the data and helps in persistence of the changes made to the system.

# Patterns Used

* Model-View-Controller (MVC) Model Layers

User Interface

StudentHome.jsp

ProfessorHome.jsp

AdvisorHome.jsp

AdminHome.jsp

Home.jsp

CourseService

LoginController

Domain

Domain Service

AdminService

RequestService

ProfessorService

AdvisorController

ProfessorController

StudentController

AdminController

Controller

UserService

StudyPlannerService

CourseActivityService

CourseActivity

StudyPlan

Task

Course

OfferedCourse

Request

User

As you see in the above figure, our design is based on MVC model.

* **Viewer Layer**: This layer displays UI and input/output user actions.
* **Controller Layer**: Every input from user is first dealt with by controllers.
* **Model Layer**: Does actual application logics such as database access.

### Controller GRASP pattern

All those controller classes in the figure play the role to separate UI layer from program logic. This is the key in our design because, whenever UI or program logic is changed separately, the one does not affect the other one, which means low coupling and high cohesion.

### Pure Fabrication & Indirection

The domain service layer does everything related to database service and does nothing else (pure fabrication). Hence, every other class other than domain service cannot access database directly and should access database only via class in domain service layer (indirection).

### Low Coupling

As you can see in the above diagram the design layers are shown. The way we have mapped is each class in the Layer is dependent on only one or two class in the layer below it. The mappings are as shown below

1. Advisor Controller depends on Request Service depends on Request
2. Admin Controller depends on Admin Service depends on Course and OfferedCourse
3. Student Controller depends on CourseService and StudyPlannerService.
4. CourseService depends on Course.
5. StudyPlannerService depends on StudyPlan depends on Task.

### High Cohesion

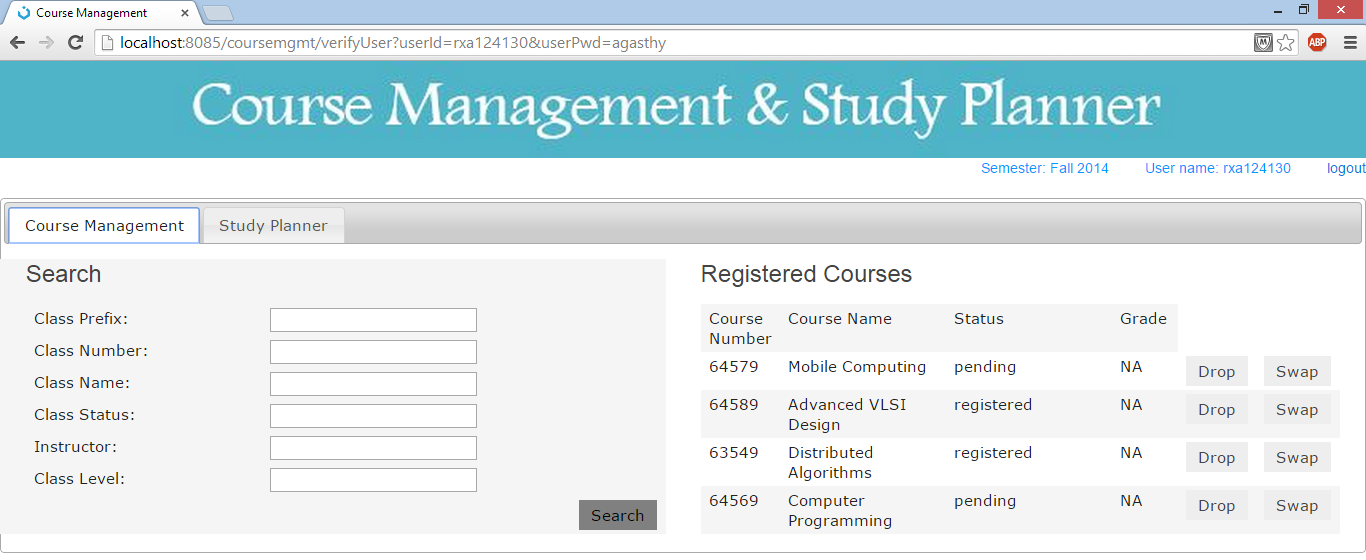
Based on our design we can easily say that each class is highly focused on doing the responsibilities assigned to it. We have a controller and a service for each type of user we have identified and they house methods which realize particular usecases associated with the respective user.

Apart from that the domain classes we have are highly reusable. For example we use Course class objects in both CourseService and AdminService. Similarly CourseActivity class objects in ProfessorService and StudyPlannerService.

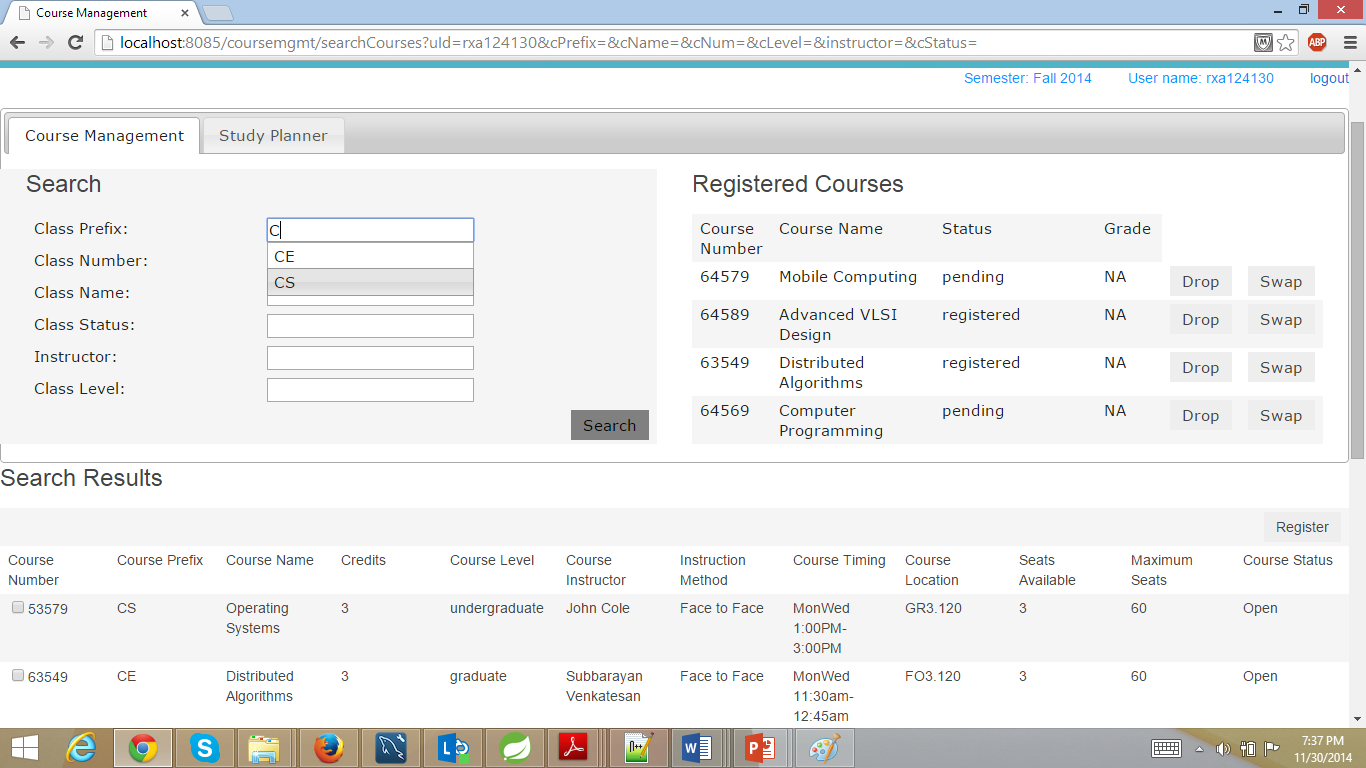
# Software Snapshots

## Student Home Page

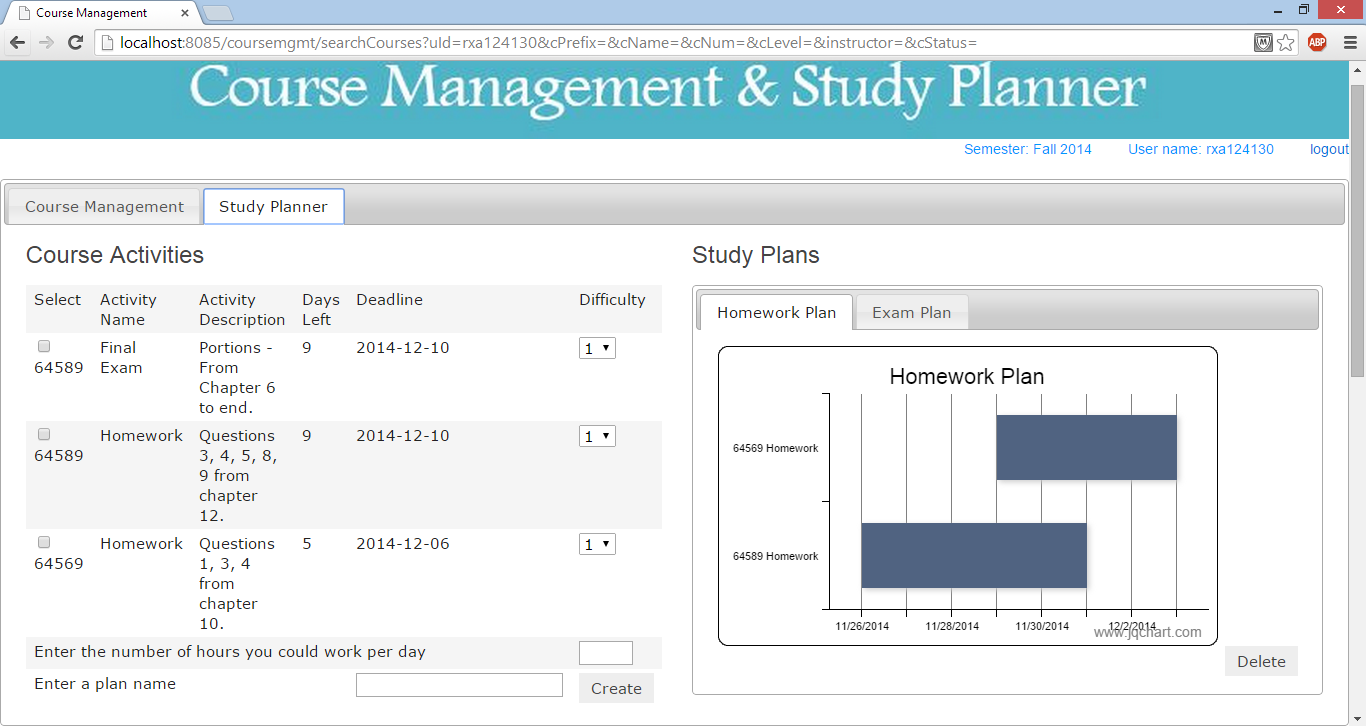
1. Displays registered courses and interface for searching courses.



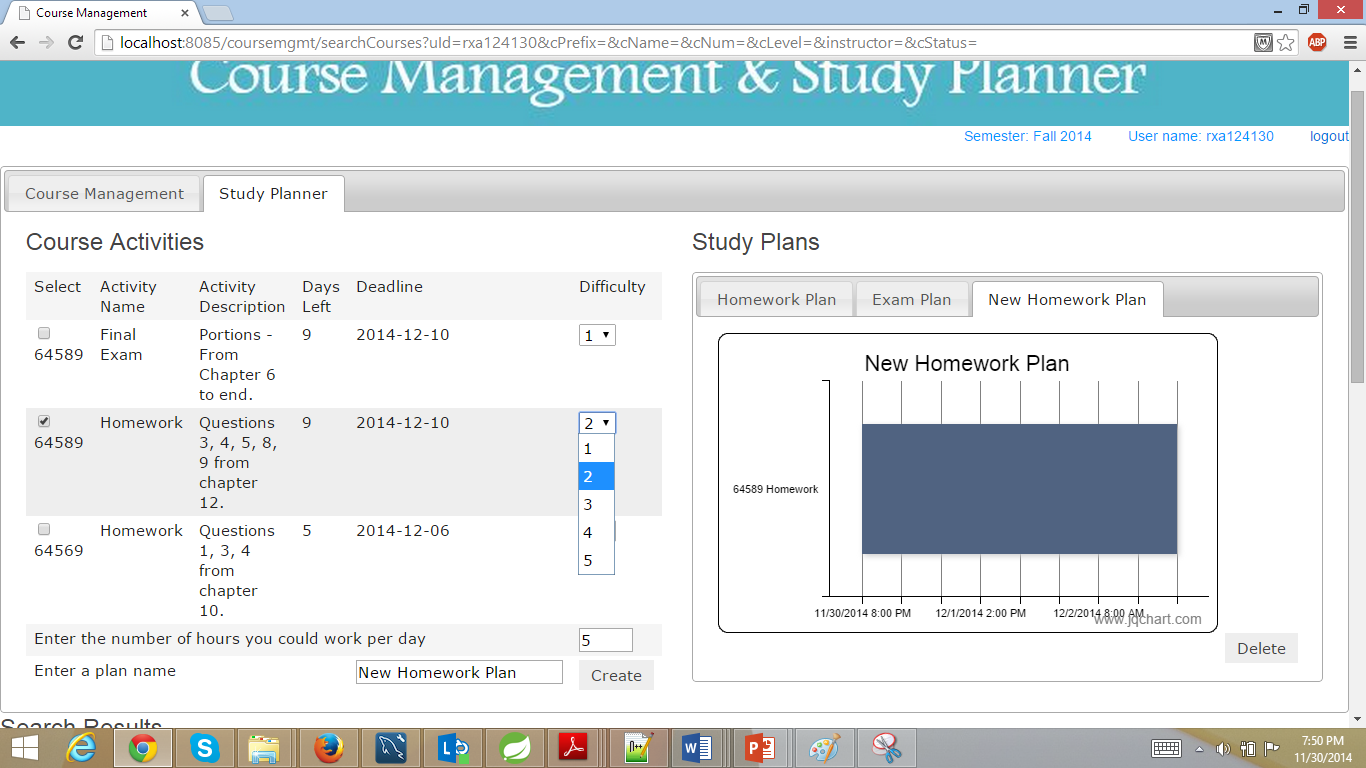
1. You can provide search filters or you can just click on search to get all the offered courses.



1. You can click on drop/swap button next to the registered courses. Then you can select the searched course and then click on register to register for courses. As shown above.
2. You can view course activities and then already created study plans.



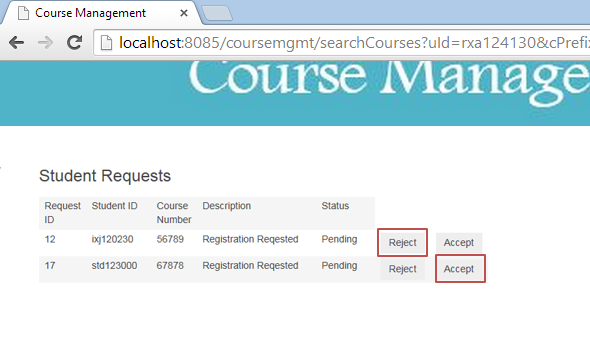
1. You can create a new plan by entering all the details above.



Can also delete the plan by clicking on it.

## Advisor Home Page

Advisor can view the student requests and handles them as below.



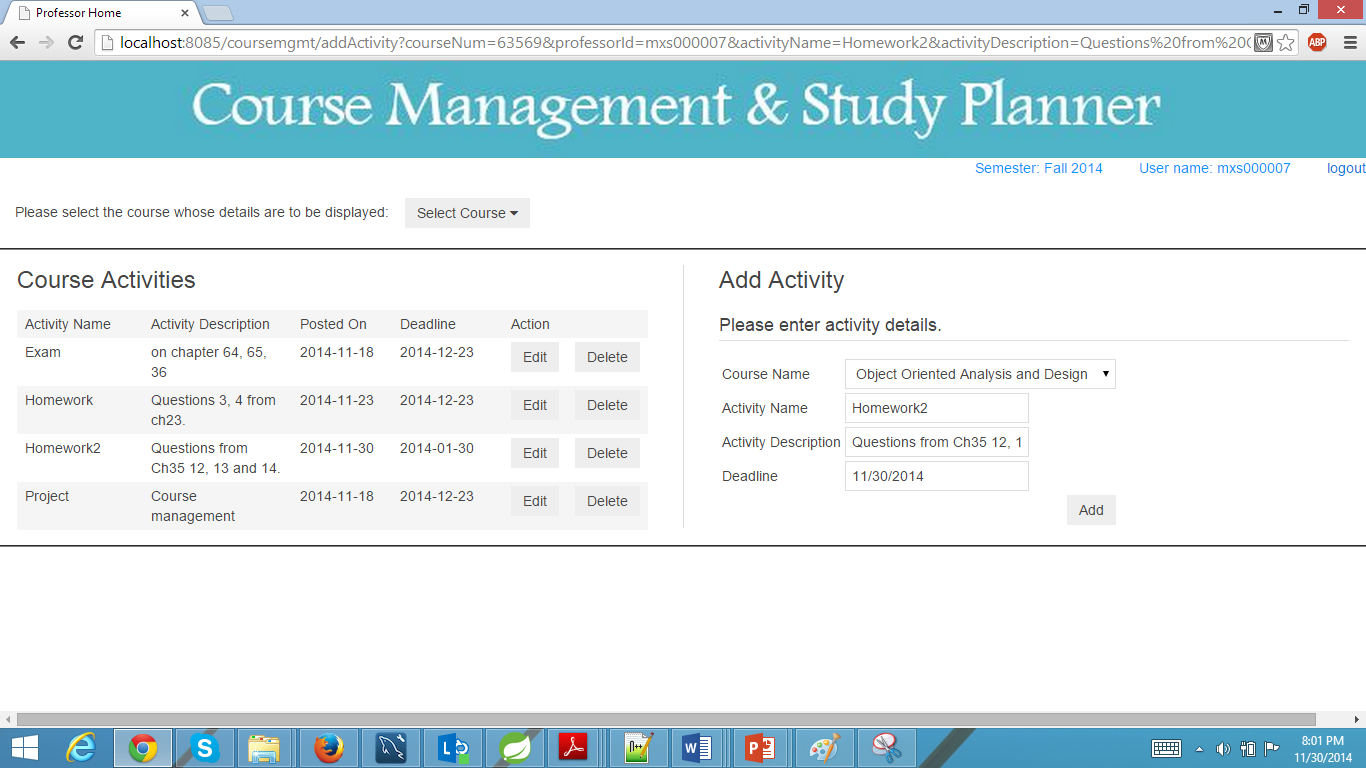


## Professor Home Page

Professor can view the course activities as soon as he logs in.

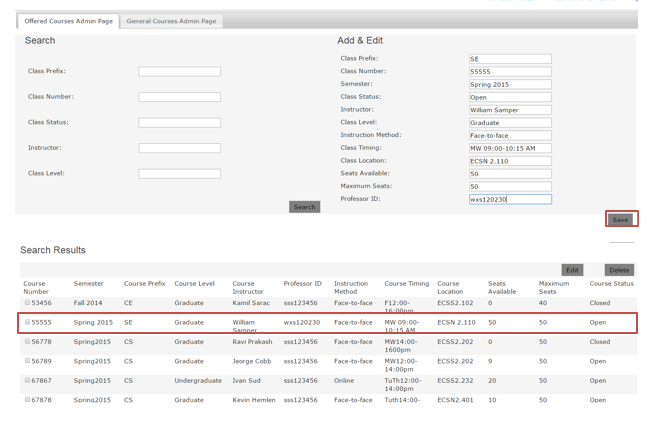


Professor can add or edit data as below.

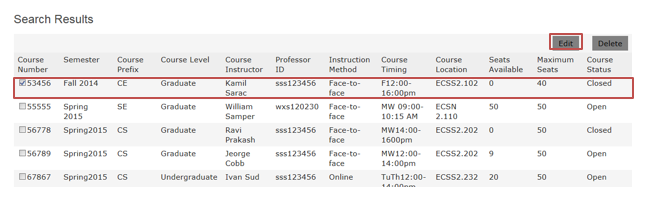


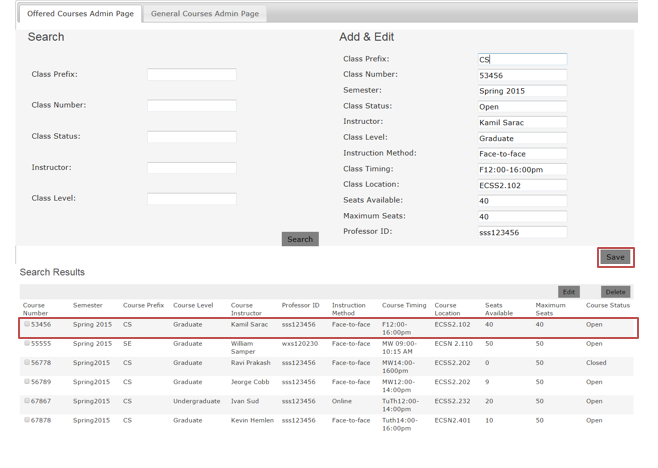
## Administrators Home Page

Admin can manage course related information as shown below.



Can edit offered courses as below.





# Lessons Learnt

**Domain Knowledge**

We have learnt a lot from this course. Object-oriented approach has been long around software developers, but still it is not easy to fully benefit from the OOAD concept and methodology. From our experience, software development has been always a sequence of modifications from analysis to implementation.

In this course we have learned what Agile Method mean and how it works. Especially, as we were working on our projects in iterations, we have realized that this approach is much more efficient way to develop a program. Specifically, we benefit much from learning how we apply design patterns.

Every time we got an opportunity to modify the system and make it better and also add new things to the system so that it fulfills more requirements.

We got the opportunity of playing multiple roles of customer, software architect and a software developer.

## Pitfalls

In fact, agile and evolutionary methodologies were not familiar with us and we have been confused sometimes on what we should exactly do for the artifacts for each iteration. That is, since the course management system is already well-known for us, we usually did not have many things to change from inception.

UML Notations and Getting the diagrams right - Also, we struggled to draw diagrams correctly. (E.g., confusion between system sequence diagram and sequence diagram).

Need for change - we have implemented most of the use cases we have identified. In the process we had to change our analysis and design for every iteration.

Identifying the problem pattern and then applying the same for designing the solution was a challenge for us as we were new to the concept and learnt about it much later in the course when 50% of the development was done.