Software Requirements Specification

Project Title:

Learning Management System (LMS) for Aitchison College

Course name: Object-Oriented Analysis & Design

Course Code: CSCS 352

Section: B

Development Team:

1) Mohammad Usman

2) Mohammad Sharif Hayat

Date of Submission: 31st May, 2019

Submitted To: Dr. Saba Khalil Toor

Table of Contents

1. Introduction:

1.1 Introduction

Before we can move further, we must first define what is meant by a "Learning Management System". In essence, an LMS is a software application used for the administration and tracking of an educational course. An LMS integrates all the necessary aspects of an educational system onto a single platform, so that the progress of the educational course can be monitored and controlled easily. They help to create, adopt, administer, distribute and manage all of the activities related to e-learning training. Some prime examples of LMS are: Moodle, Edmodo, Turnitin and Google Classroom.

1.2 About

The project handles all the information of the student's education and learning. Also it manages resources which were managed and handled by manual manpower previously. The main purpose of the project is to integrate distinct sections of the organization in a consistent manner so that complex functions can be handled smoothly by any technical or non-technical persons.

In this case, the organization for which the LMS is being developed is the Aitchison College. The LMS is supposed to handle the educational information of a high school. That means it will cater to all the subjects being offered and all the various sections within the school.

1.3 Requirements

The Administrator role is as follows:

- Manage Teacher
- Manage Student
- Search Student
- Search Teacher
- Manage Section

The Teacher role is as follows:

- Manage Task (Quiz, Assignment)
- Search Student
- Mark Attendance
- Upload Content
- Post/Comment
- Check Progress

The Student role is as follows:

- Assessment (Quiz, Assignment)
- Download Content
- Post/Comment
- Check Progress

2. Use-Case Model:

2.1 Actors

Primary Actors

- Administrator
- Teacher
- Student

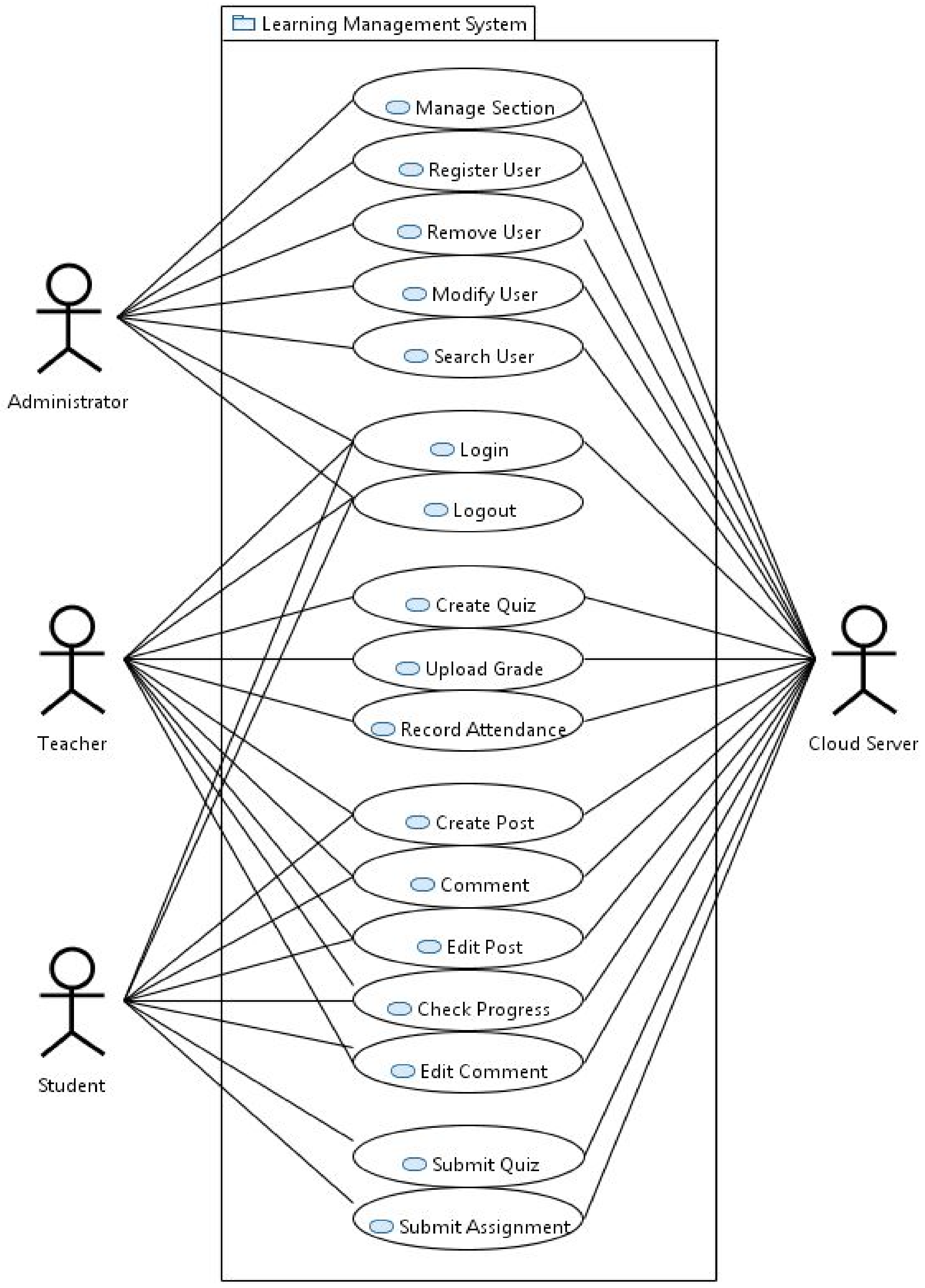
Secondary Actors

- Cloud Server

2.2 Use Cases

1. Login (All)
2. Logout (All)
3. Register User (Administrator)
4. Create Post (Teacher & Student)
5. Comment (Teacher & Student)
6. Create Quiz (Teacher)
7. Submit Quiz (Student)
8. Submit Assignment (Student)
9. Upload Grade (Teacher)
10. Remove User (Administrator)
11. Modify User (Administrator)
12. Edit Post (Teacher & Student)
13. Edit Comment (Teacher & Student)
14. Search User (Administrator)
15. Record Attendance (Teacher)
16. Check Progress (Student & Teacher)
17. Manage Section (Administrator)

2.3 Use Case Diagram



Note: As our project has 17 use cases, patterns are only applied to the first 10 use cases and the class diagrams are shown after applying the patterns. Moreover, the domain models contain some software elements as Aitchison College is already using a Web Application. Also note that as we ourselves are the developers, so no operation contracts were needed to clear the understanding of any method.

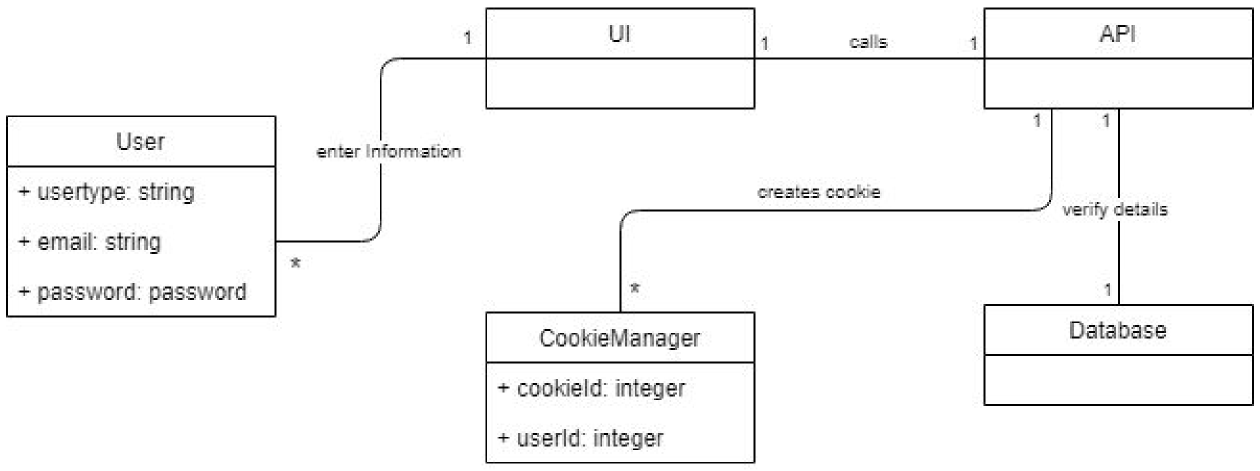
2.2.1 Login

This function allows the user to enter into the application. Registered User can login to the system by entering valid user id and password. It is the login session for the Administrator, teacher and student. Only the authorized person can login. Priority is high.

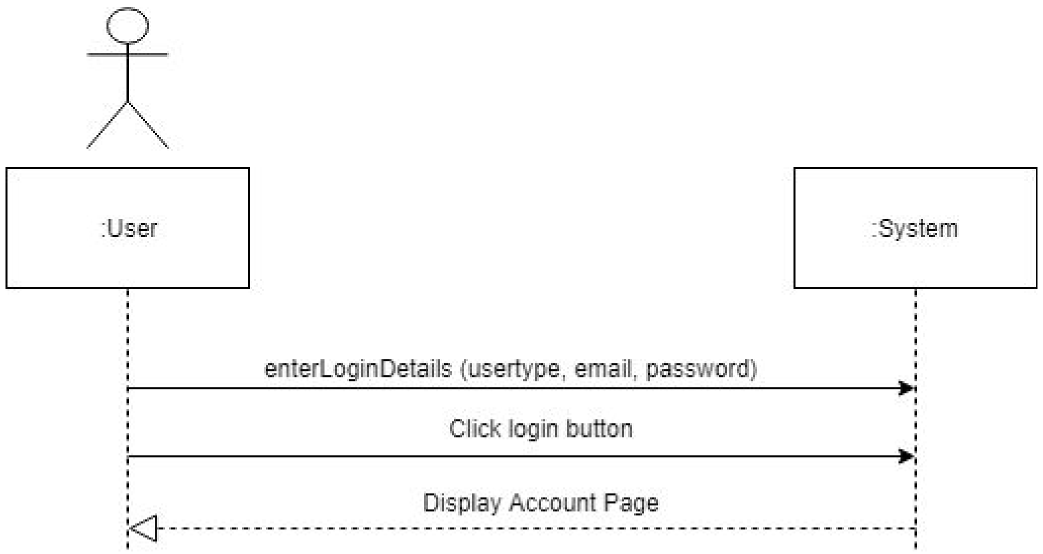
Use Case ID:	UC-1.1		
Use Case Name:	Login		
Created By:		**Last Updated By:**	
Date Created:		**Last Revision Date:**	
Actors:	Administrator/Teacher/Student		
Description:	Login is required by the administrator, teacher as well as the student in order to use the particular features of the system. The end-user is directed to his/her account and is ready to use the system.		
Trigger:	When the end-user wishes to log into the Learning Management System.		
Preconditions:	Nil		
Postconditions:	1. Instance of Session class is instantiated. 2. The end-user is directed to his/her account and is ready to use the system.		
Normal Flow:	1. End-user selects user type. 2. End-user enters Email address. 3. End-user enters Password. 4. End-user clicks on Login button. 5. System validates if the end-user is registered. 6. System creates a session for the end-user. 7. End-user is directed to his/her account.		
Alternative Flows:	Nil		
Exceptions:	2a. In step 2 of the normal flow, if typed email address is not registered 1. Message to end-user that typed email address is not registered and to re-enter email address. 2. End-User re-enters email address. 3. Use case resumes on step 4 of normal flow.		

	2b. In step 2 of the normal flow, if the end-user enters an invalid Email address 1. Message to the end-user that typed email address is invalid and to re-enter email address. 2. End-user enters correct email address. 3. Use case resumes on step 4 of normal flow. 3a. In step 3 of the normal flow, if the end-user enters wrong password 1. Message to the end-user that typed password is incorrect and to re-enter password. 2. End-user enters correct password. 3. Use case resumes on step 4 of normal flow.
Includes:	Nil
Frequency of Use:	150 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall not be able to enter invalid input 4. The end-user shall be able to login within 1 seconds after filling out all the fields 5. The system shall accept 50 requests per second 6. The system shall authenticate the end-user according to the information stored in the cloud server. 7. The system shall not leave any cookies on the end-user's computer containing any of the end-user's confidential information.
Assumptions:	Nil
Notes and Issues:	Nil

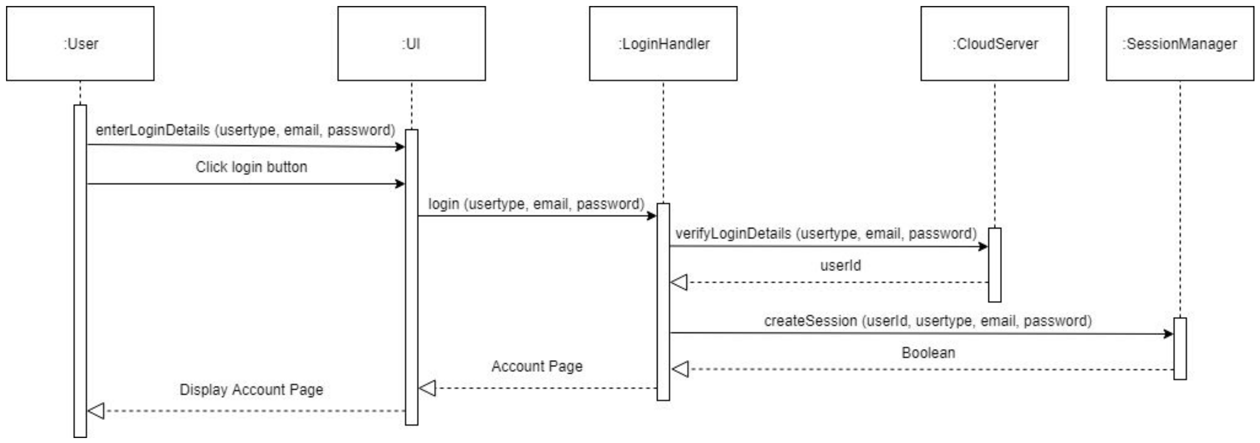
Domain Model:



System Sequence Diagram:

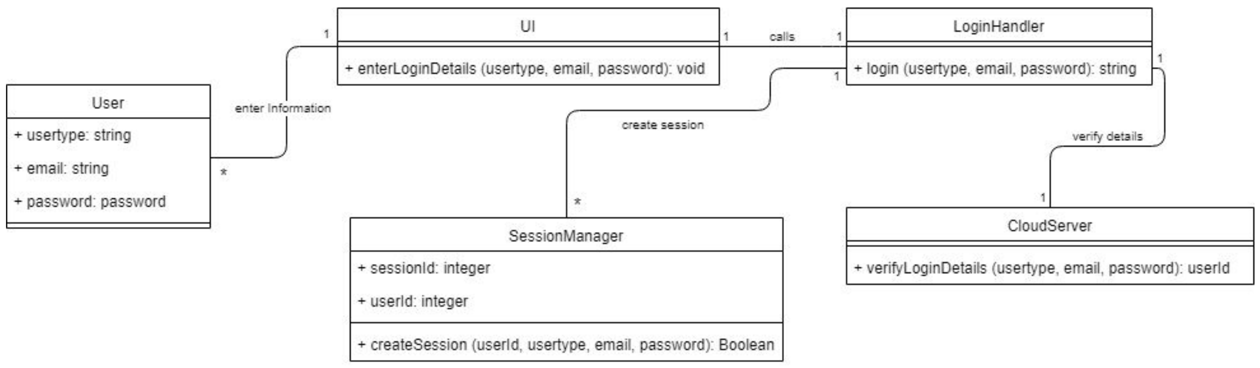


Sequence Diagram:



:User
:UI
:LoginHandler
:CloudServer
:SessionManager
enterLoginDetails (usertype, email, password)
Click login button
login (usertype, email, password)
verifyLoginDetails (usertype, email, password)
userId
createSession (userId, usertype, email, password)
Boolean
Account Page
Display Account Page

Class Diagram:



UI
+ enterLoginDetails (usertype, email, password): void
LoginHandler
+ login (usertype, email, password): string
User
+ usertype: string
+ email: string
+ password: password
SessionManager
+ sessionId: integer
+ userId: integer
+ createSession (userId, usertype, email, password): Boolean
CloudServer
+ verifyLoginDetails (usertype, email, password): userId
enter Information
calls
create session
verify details
1
*
1
1
1
*
1
1

Patterns:

1. **Controller:**
    a. **Problem:**
    What first object beyond the UI layer receives and coordinates a system operation?

    b. **Solution:**
    Assign the responsibility to the class LoginHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

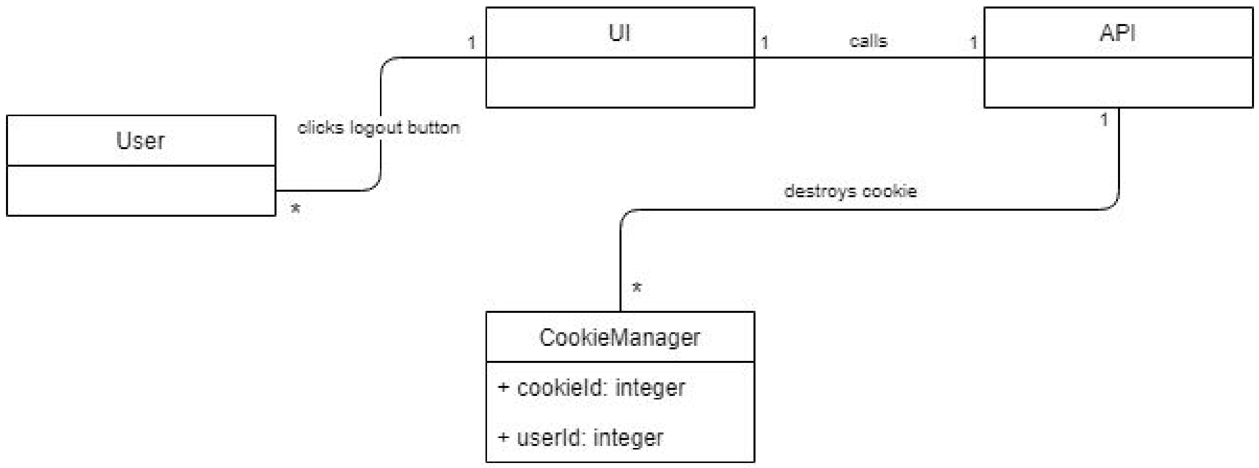
2.2.2 Logout

It terminates the user session. This is by the Administrator, teachers and students who can logout after they have finished using the system. Priority is high.

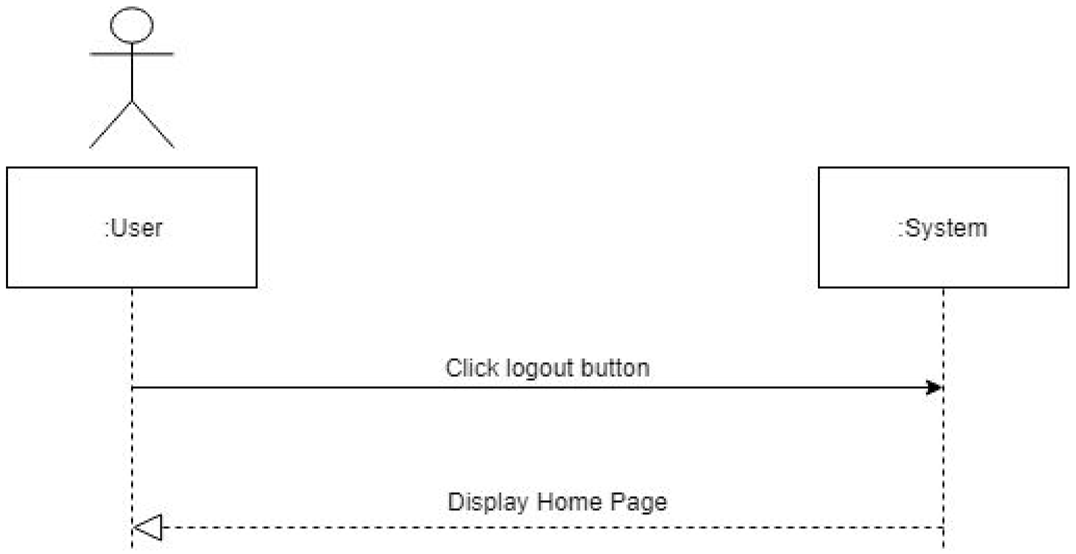
Use Case ID:	UC-1.2		
Use Case Name:	Logout		
Created By:		**Last Updated By:**	
Date Created:		**Last Revision Date:**	
Actors:	Administrator/Teacher/Student		
Description:	This terminates the end-user's session and is logged out of the system. The end-user will be directed to their account.		
Trigger:	When the end-user clicks on the Logout button located at the top right on the Header besides the Check Progress Button.		
Preconditions:	1. The End-user must be logged in.		
Postconditions:	1. Instance of session is destroyed. 2. The end-user is directed to the Home page.		
Normal Flow:	1. The end-user clicks on the Logout button. 2. System destroys the session of the end-user. 3. The end-user is directed to the Home page.		
Alternative Flows:	Nil		
Exceptions:	Nil		
Includes:	Nil		
Frequency of Use:	150 per day		
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to logout within 1 second 4. The system shall accept 50 requests per second 5. The end-user will be directed to the Home page.		

	6. The system shall terminate the end-user's session. 7. The system shall direct the end-user to the Home page after logging out. 8. The system shall automatically log out all users after a period of inactivity. 9. The system shall not leave any cookies on the end-user's computer containing any of the end-user's confidential information.
Assumptions:	Nil
Notes and Issues:	Nil

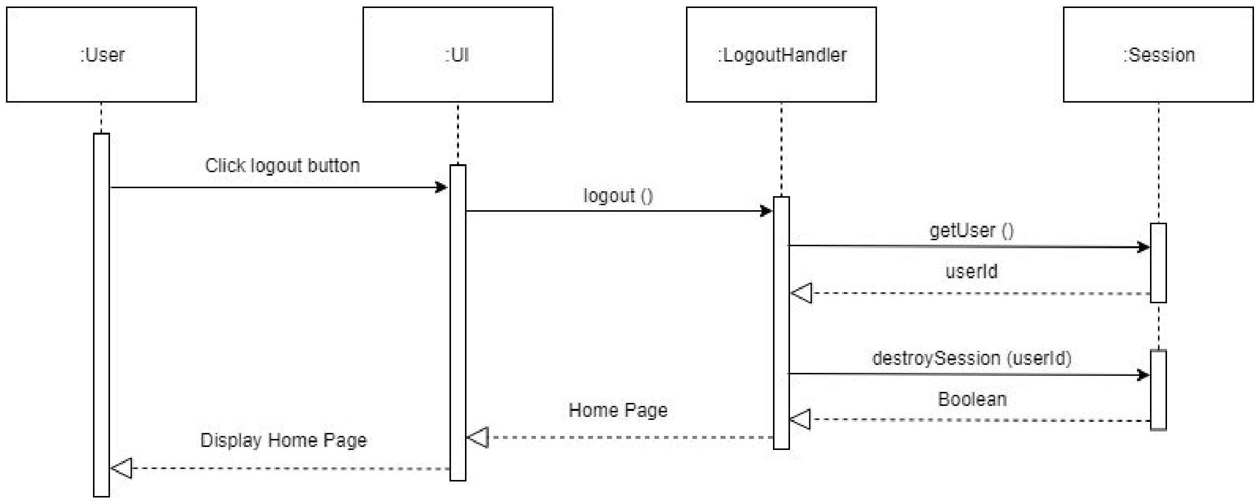
Domain Model:



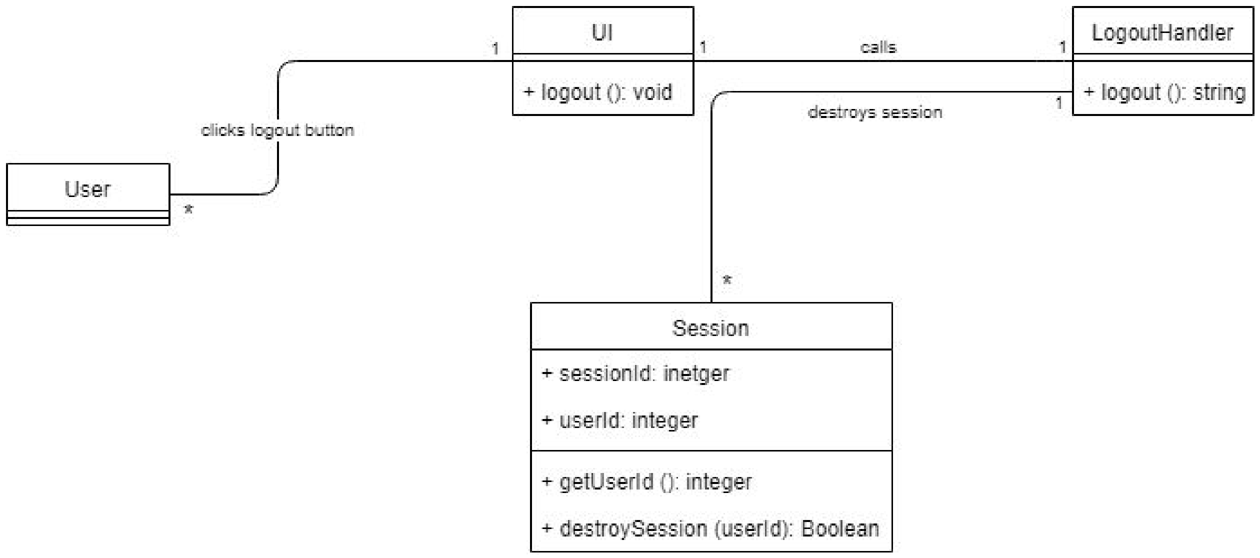
System Sequence Diagram:



Sequence Diagram:



Class Diagram:



Patterns:

1. **Controller:**
   a. **Problem:**
      What first object beyond the UI layer receives and coordinates a system operation?

   b. **Solution:**
      Assign the responsibility to the class LogoutHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2.2.3 Register User

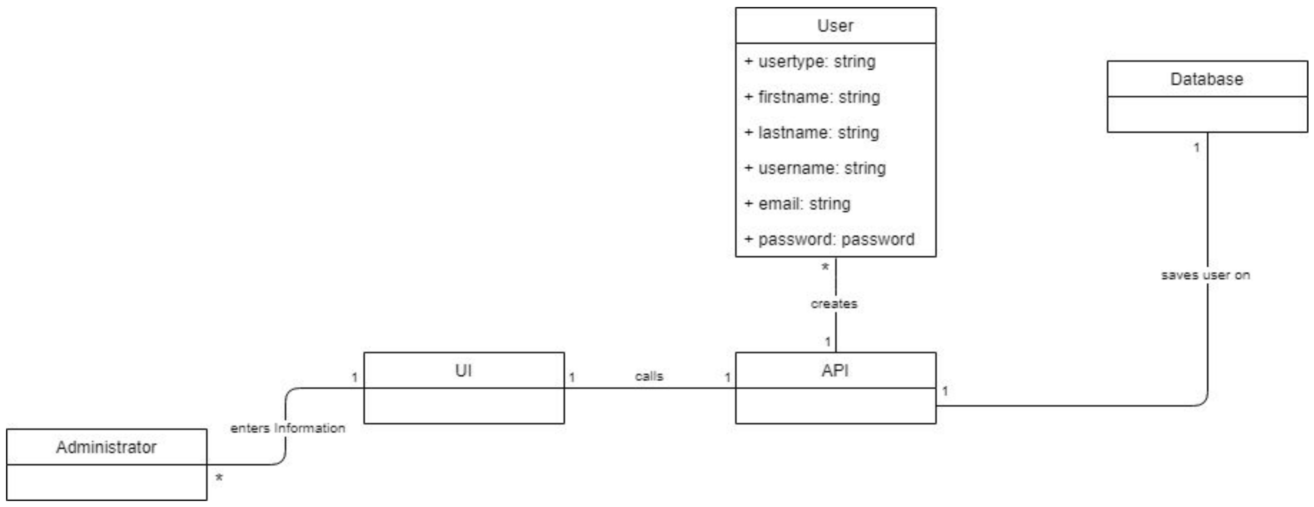
This is used by the administrator who can register any type of user (Administrator, Teacher, Student). If an end-user wants to use the various features provided by the system then (s)he must be registered. Priority is high.

Use Case ID:	UC-1.3		
Use Case Name:	Register User		
Created By:		**Last Updated By:**	
Date Created:		**Last Revision Date:**	

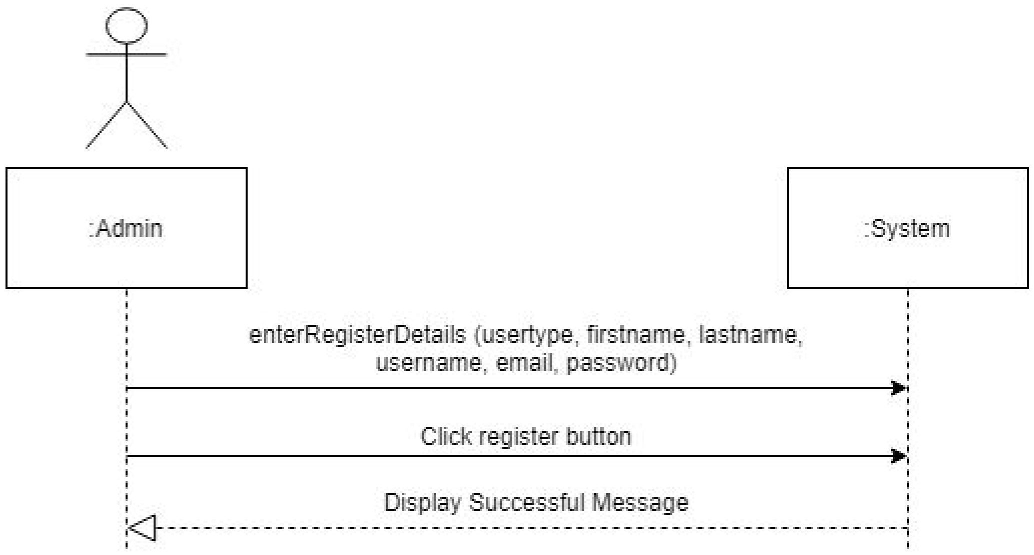
Actors:	Administrator
Description:	Registration is required for the administrator/student/teacher in order to use the particular features of the system. All of the information of the new user will be stored in the cloud server.
Trigger:	When the Administrator opens the "Registration" panel.
Preconditions:	1. The Administrator must be logged in.
Postconditions:	1. Instance of User is instantiated. 2. All of the information of the new user is stored in the cloud server. 3. All of the fields are emptied out.
Normal Flow:	1. End-user selects user type. 2. End-user enters First Name. 3. End-user enters Last Name. 4. End-user enters Username. 5. End-user enters Email address. 6. End-user enters Password. 7. End-user clicks on Register button. 8. System validates if a user with the same username is not already registered. 9. System validates if a user with the same email address is not already registered. 10. End-user is shown a registration successful message.
Alternative Flows:	Nil
Exceptions:	2a. In step 2 of the normal flow, if the Administrator enters an invalid First Name 1. Message to administrator that typed first name is invalid and to re-enter first name. 2. Administrator enters correct first name. 3. Use case resumes on step 7 of normal flow. 3a. In step 3 of the normal flow, if the Administrator enters an invalid Last Name

	1. Message to administrator that typed last name is invalid and to re-enter last name. 2. Administrator enters correct last name. 3. Use case resumes on step 7 of normal flow. 4a. In step 4 of the normal flow, if the Administrator enters a username which is already registered 1. Message to administrator that the typed username is already in use and to re-enter username. 2. Administrator enters another email address. 3. Use case resumes on step 7 of normal flow. 5a. In step 5 of the normal flow, if the Administrator enters an email which is already registered 1. Message to administrator that the typed email is already in use and to re-enter email address. 2. Administrator enters another email address. 3. Use case resumes on step 7 of normal flow. 5a. In step 5 of the normal flow, if the Administrator enters an invalid Email address 1. Message to administrator that typed email address is invalid and to re-enter email address. 2. Administrator enters correct email address. 3. Use case resumes on step 7 of normal flow. 6a. In step 6 of the normal flow, if the Administrator enters an invalid password 1. Message to administrator that typed password is invalid and to re-enter password. 2. Administrator enters correct password. 3. Use case resumes on step 7 of normal flow.
Includes:	Nil
Frequency of Use:	100 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall not be able to enter invalid input 4. The end-user shall be able to register within 5 seconds after filling out all the fields 5. The password should be at least 6 characters long containing characters (0-9, a-z, A-Z, _, #) 6. The system shall accept 5 requests per second 7. The system should be able to contain 200,000 users.
Assumptions:	Nil
Notes and Issues:	Nil

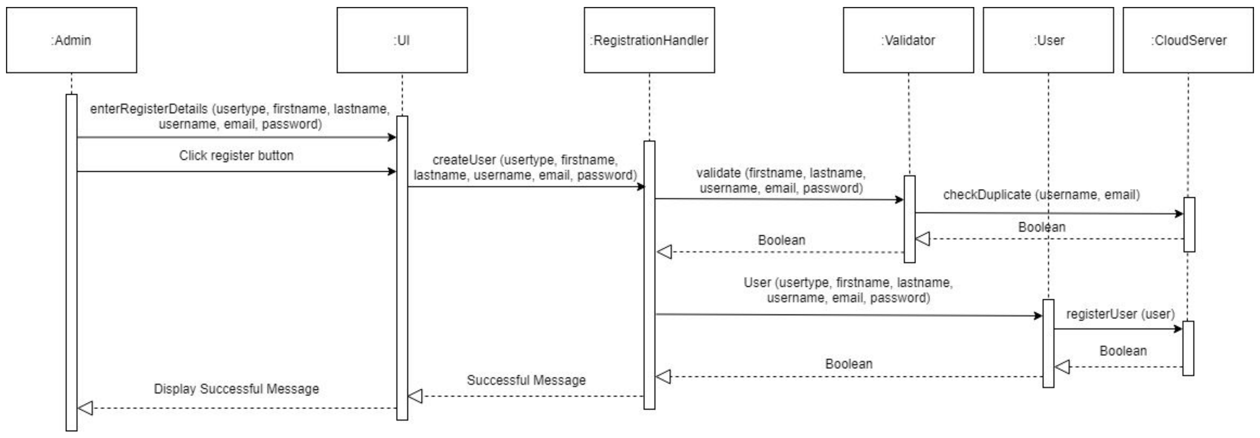
Domain Model:



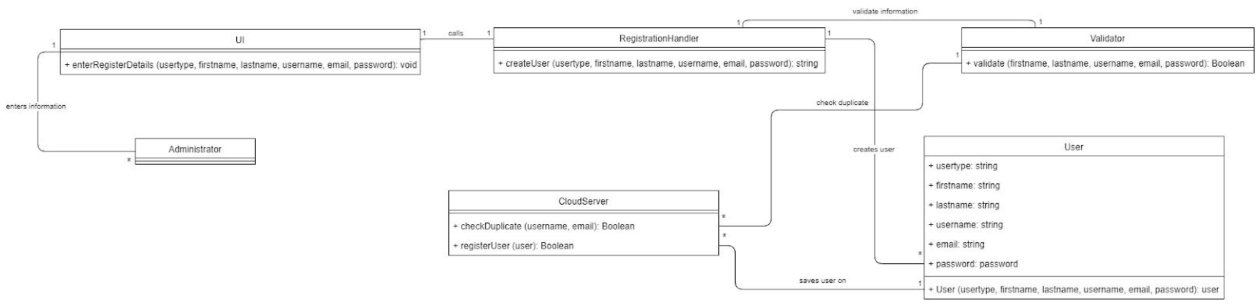
System Sequence Diagram:


:Admin
:System
enterRegisterDetails (usertype, firstname, lastname, username, email, password)
Click register button
Display Successful Message

Sequence Diagram:


:Admin
:UI
:RegistrationHandler
:Validator
:User
:CloudServer
enterRegisterDetails (usertype, firstname, lastname, username, email, password)
Click register button
createUser (usertype, firstname, lastname, username, email, password)
validate (firstname, lastname, username, email, password)
checkDuplicate (username, email)
Boolean
Boolean
User (usertype, firstname, lastname, username, email, password)
registerUser (user)
Boolean
Boolean
Successful Message
Display Successful Message

Class Diagram:


validate information
1
1
UI
+ enterRegisterDetails (usertype, firstname, lastname, username, email, password): void
1
calls
1
RegistrationHandler
+ createUser (usertype, firstname, lastname, username, email, password): string
1
Validator
+ validate (firstname, lastname, username, email, password): Boolean
1
enters information
1
Administrator
*
check duplicate
creates user
User
+ usertype: string
+ firstname: string
+ lastname: string
+ username: string
+ email: string
+ password: password
+ User (usertype, firstname, lastname, username, email, password): user
CloudServer
+ checkDuplicate (username, email): Boolean
+ registerUser (user): Boolean
*
*
*
saves user on
1

Patterns:

1. **Controller**
    a. **Problem:**
    What first object beyond the UI layer receives and coordinates a system operation?
    b. **Solution:**
    Assign the responsibility to the class RegistrationHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Low Coupling**
    a. **Problem:**
    How to support low dependency, low change impact and increased reuse in saving the User on Cloud Server after creation of User?
    b. **Solution:**
    Assign the responsibility to the class CloudServer to save information of an instance of User to the Cloud Server instead of User which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

2.2.4 Create Post

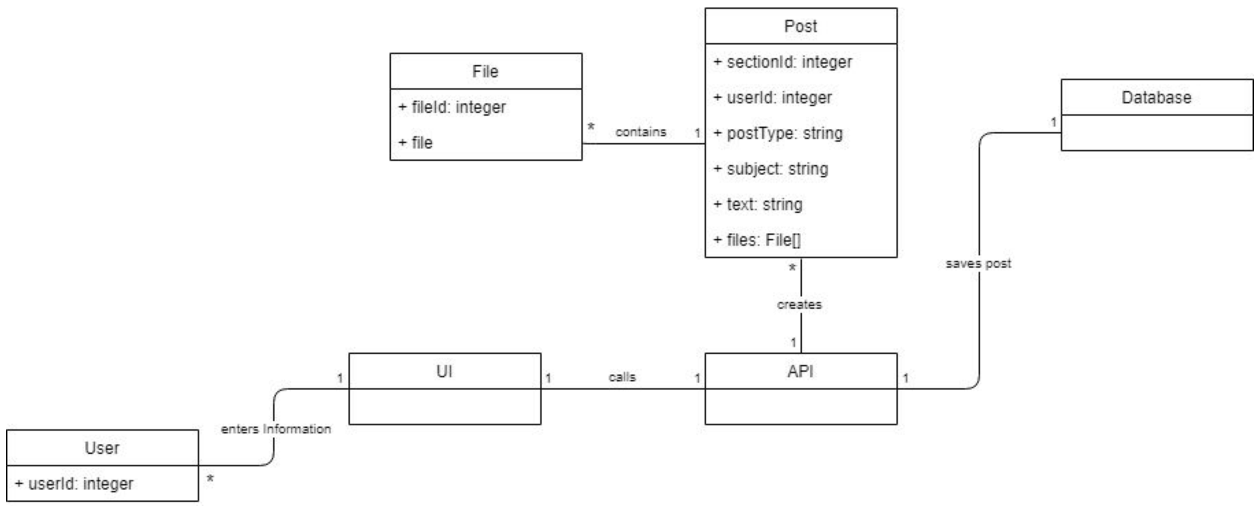
This is used by teacher/student. Teacher can use it to post an announcement, lecture slides, content or assignment. Student can use it to post a query or to make a new thread. Priority is medium.

Use Case ID:	UC-1.4		
Use Case Name:	Create Post		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	

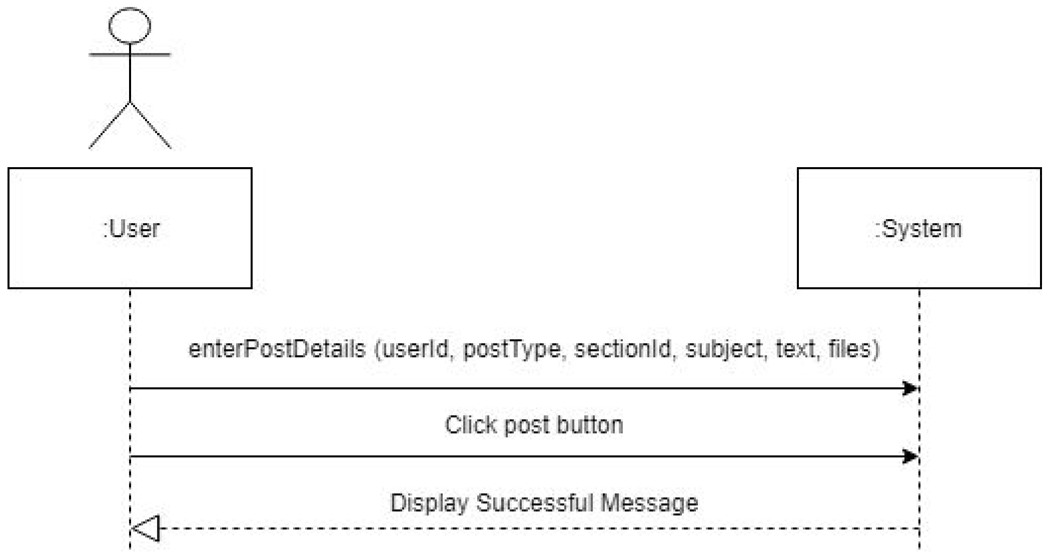
Actors:	Teacher/Student
Description:	Teacher and student uses this feature of the system to communicate on various important subjects.
Trigger:	When the end-user clicks the “Create Post” button on a particular section page
Preconditions:	1. The end-user must be logged in.
Postconditions:	1. All of the information regarding the post is stored in the cloud server.
Normal Flow:	1. End-user selects Post type. 2. End-user selects Section. 3. End-user enters Post subject. 4. End-user enters Post text. 5. End-user clicks on Attach Files button to upload file(s) (optional). 6. End-user clicks on Post button. 7. System validates if individual file size is less than 5MB. 8. End-user is shown a successful message.
Alternative Flows:	Nil
Exceptions:	2a. In step 2 of the normal flow, if the end-user enters an invalid post subject including empty 1. Message to end-user that typed post subject is invalid and to re-enter post subject. 2. End-user enters correct post subject. 3. Use case resumes on step 6 of normal flow. 3a. In step 3 of the normal flow, if the end-user enters an invalid post text including empty 1. Message to end-user that typed post text is invalid and to re-enter post text. 2. End-user enters correct post text. 3. Use case resumes on step 6 of normal flow. 4a. In step 4 of the normal flow, if the file size is larger than 5MB

	1. Message to end-user that the file size is larger than 5MB and to attach file(s) again within the size limit. 2. End-User clicks on Attach Files button to upload file(s) within the size limit. 3. Use case resumes on step 6 of normal flow.
Includes:	Nil
Frequency of Use:	200 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall not be able to upload malicious files 4. The end-user shall be able to post within 2 seconds after filling out all the fields 5. The system shall accept 50 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

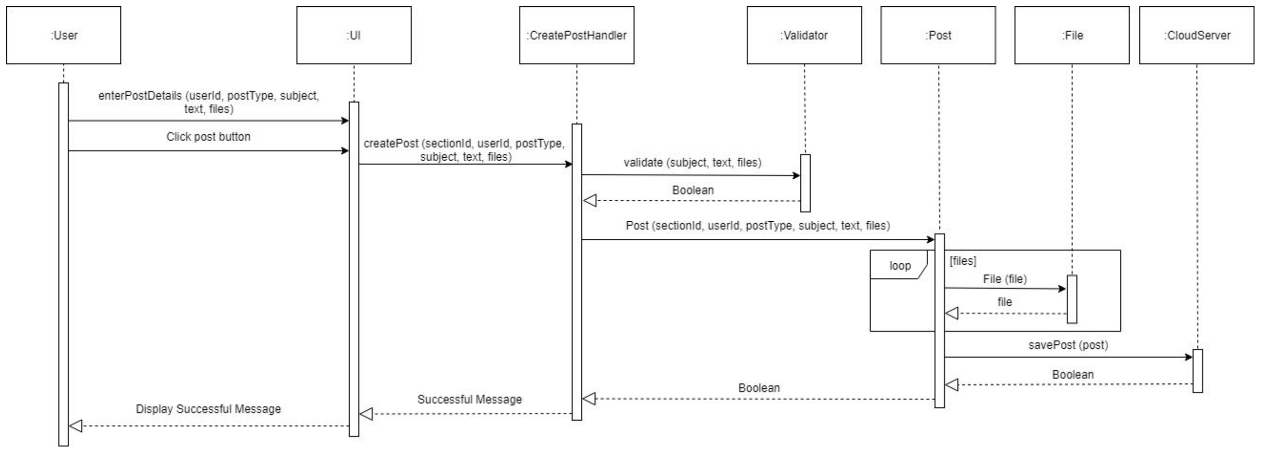
Domain Model:



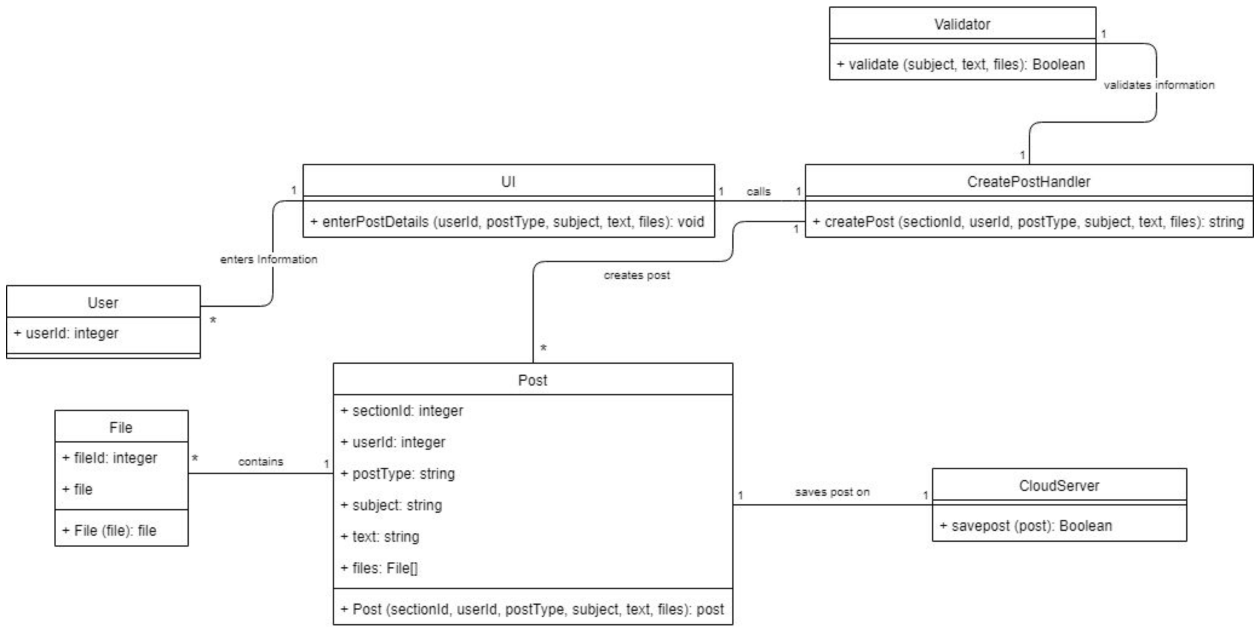
System Sequence Diagram:


:User
:System
enterPostDetails (userId, postType, sectionId, subject, text, files)
Click post button
Display Successful Message

Sequence Diagram:


:User
:UI
:CreatePostHandler
:Validator
:Post
:File
:CloudServer
enterPostDetails (userId, postType, subject, text, files)
Click post button
createPost (sectionId, userId, postType, subject, text, files)
validate (subject, text, files)
Boolean
Post (sectionId, userId, postType, subject, text, files)
loop
[files]
File (file)
file
savePost (post)
Boolean
Boolean
Successful Message
Display Successful Message

Class Diagram:



Patterns:

1. **Controller**
   a. **Problem:**
   What first object beyond the UI layer receives and coordinates a system operation?
   b. **Solution:**
   Assign the responsibility to the class CreatePostHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Low Coupling**
   a. **Problem:**
   How to support low dependency, low change impact and increased reuse in saving the Post on Cloud Server after creation of Post?
   b. **Solution:**
   Assign the responsibility to the class CloudServer to save information of an instance of Post to the Cloud Server instead of Post which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

2.2.5 Comment

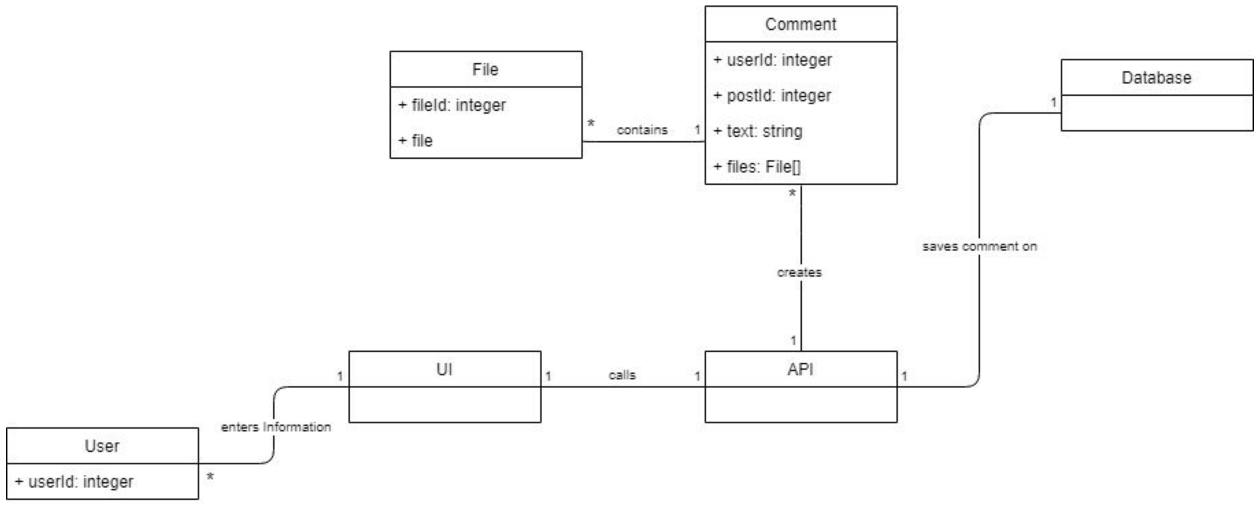
This is used by teacher/student. End-user can use it to comment on a post. Priority is low.

Use Case ID:	UC-1.5		
Use Case Name:	Comment		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	

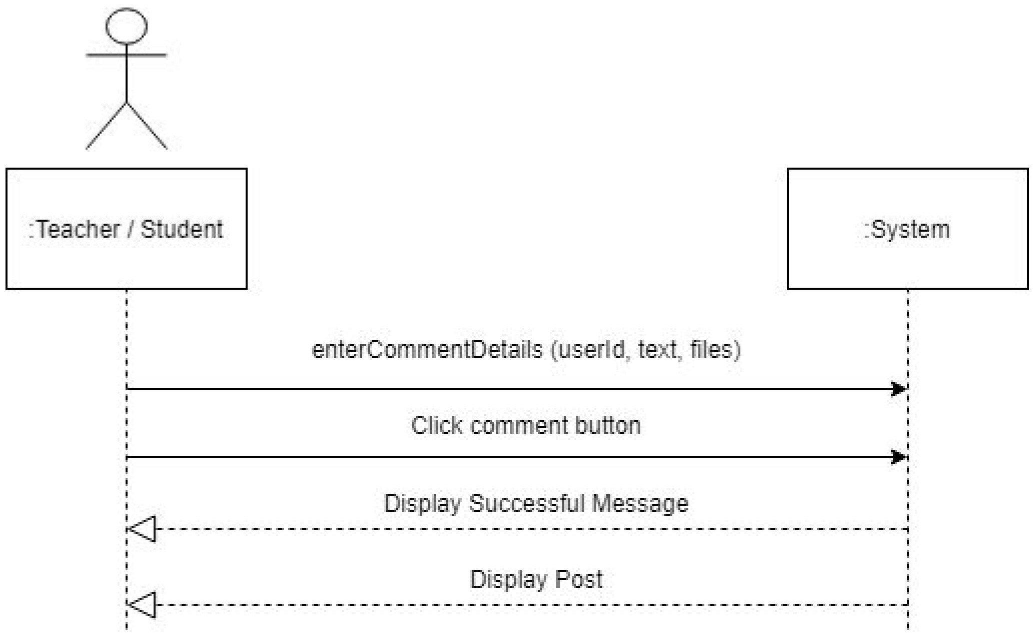
Actors:	Teacher/Student
Description:	Teacher and student use this feature to follow up on an already created post to resolve queries and ambiguities.
Trigger:	When the end-user clicks on comment button below a post.
Preconditions:	1. The end-user must be logged in.
Postconditions:	1. Instance of Comment is instantiated. 2. All of the information regarding the comment is stored in the cloud server. 3. End-user is directed to the post.
Normal Flow:	1. End-user enters Comment text. 2. End-user clicks on Attach Files button to upload file(s) (optional). 3. End-user clicks on Comment button. 4. System validates if individual file size is less than 5MB. 5. End-user is shown a successful message.
Alternative Flows:	Nil
Exceptions:	1a. In step 1 of the normal flow, if the end-user enters an invalid comment including empty 1. Message to end-user that typed comment is invalid and to re-enter comment. 2. End-user enters correct comment. 3. Use case resumes on step 3 of normal flow. 2a. In step 2 of the normal flow, if the file size is larger than 5MB 1. Message to end-user that the file size is larger than 5MB and to attach file(s) again within the size limit. 2. End-User clicks on Attach Files button to upload file(s) within the size limit. 3. Use case resumes on step 3 of normal flow.
Includes:	Nil
Frequency of Use:	200 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall not be able to upload malicious files

	4. The end-user shall be able to comment within 2 seconds after filling out all the fields 5. The system shall accept 50 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

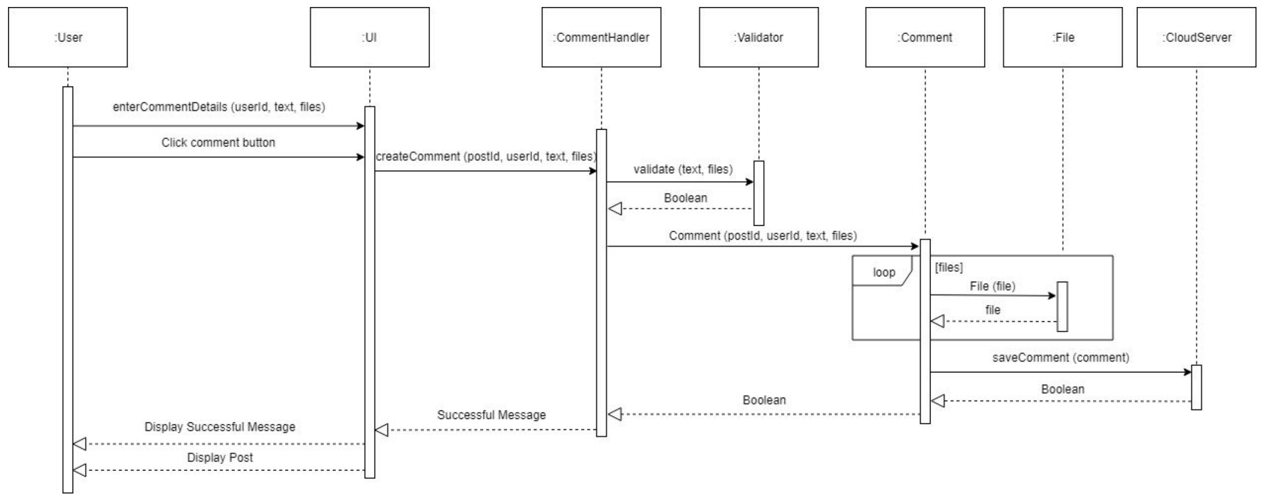
Domain Model:



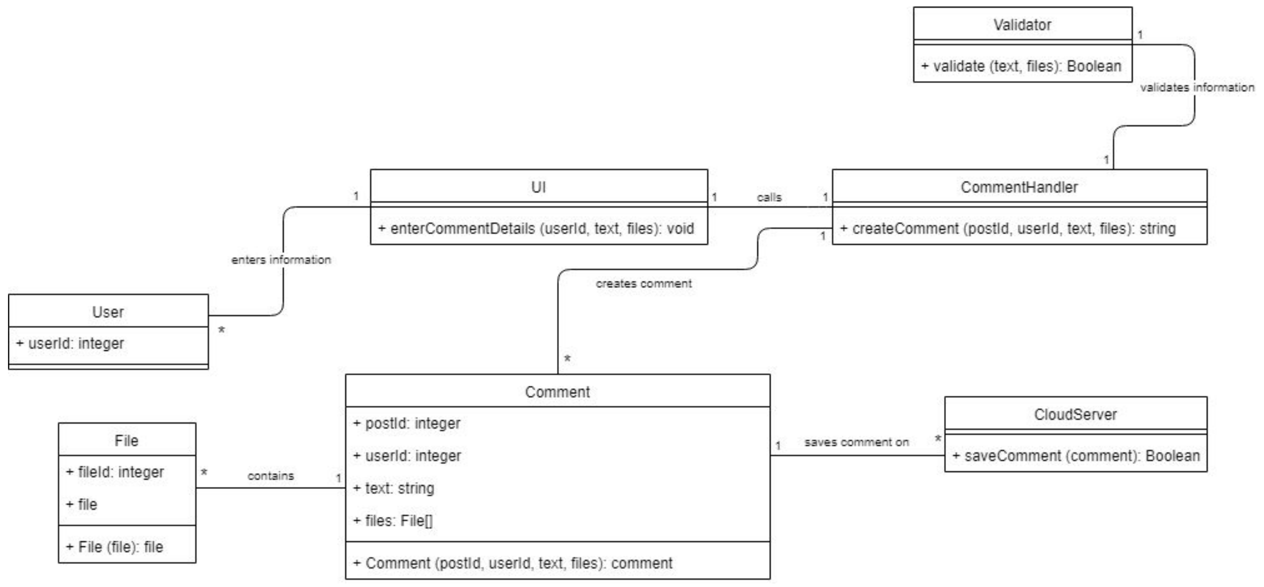
System Sequence Diagram:


:Teacher / Student
:System
enterCommentDetails (userId, text, files)
Click comment button
Display Successful Message
Display Post

Sequence Diagram:



Class Diagram:



Patterns:

1. **Controller**
   a. **Problem:**
      What first object beyond the UI layer receives and coordinates a system operation?
   b. **Solution:**
      Assign the responsibility to the class CommentHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Low Coupling**
   a. **Problem:**
   How to support low dependency, low change impact and increased reuse in saving the Comment on Cloud Server after creation of Comment?
   b. **Solution:**
   Assign the responsibility to the class CloudServer to save information of an instance of Comment to the Cloud Server instead of Comment which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

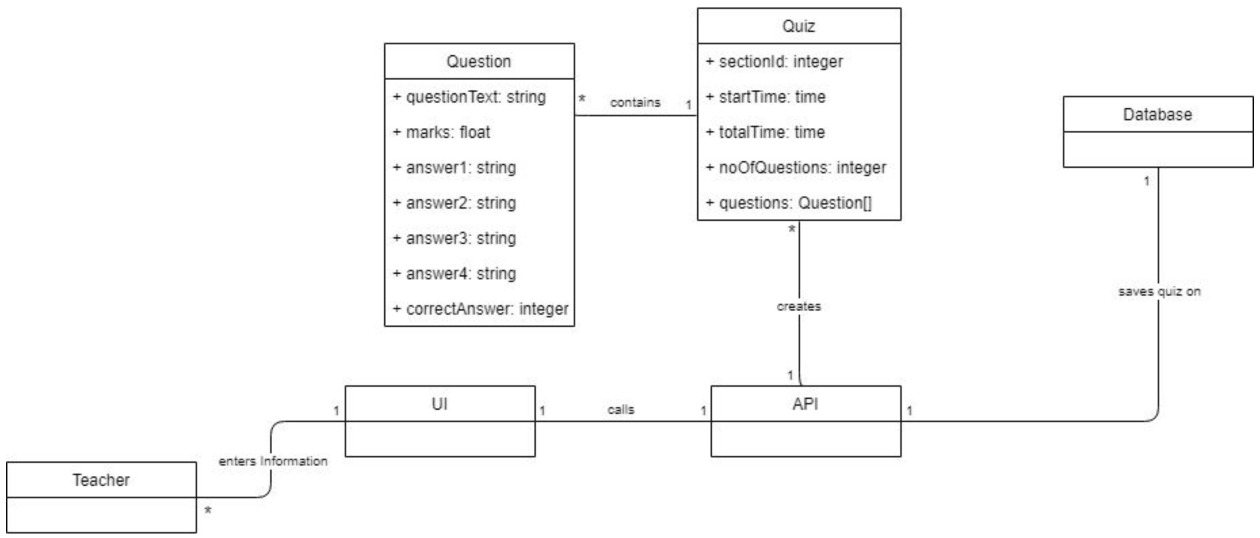
2.2.6 Create Quiz

This is used by teacher. End-user can use it to create a quiz for a specific section. Priority is medium.

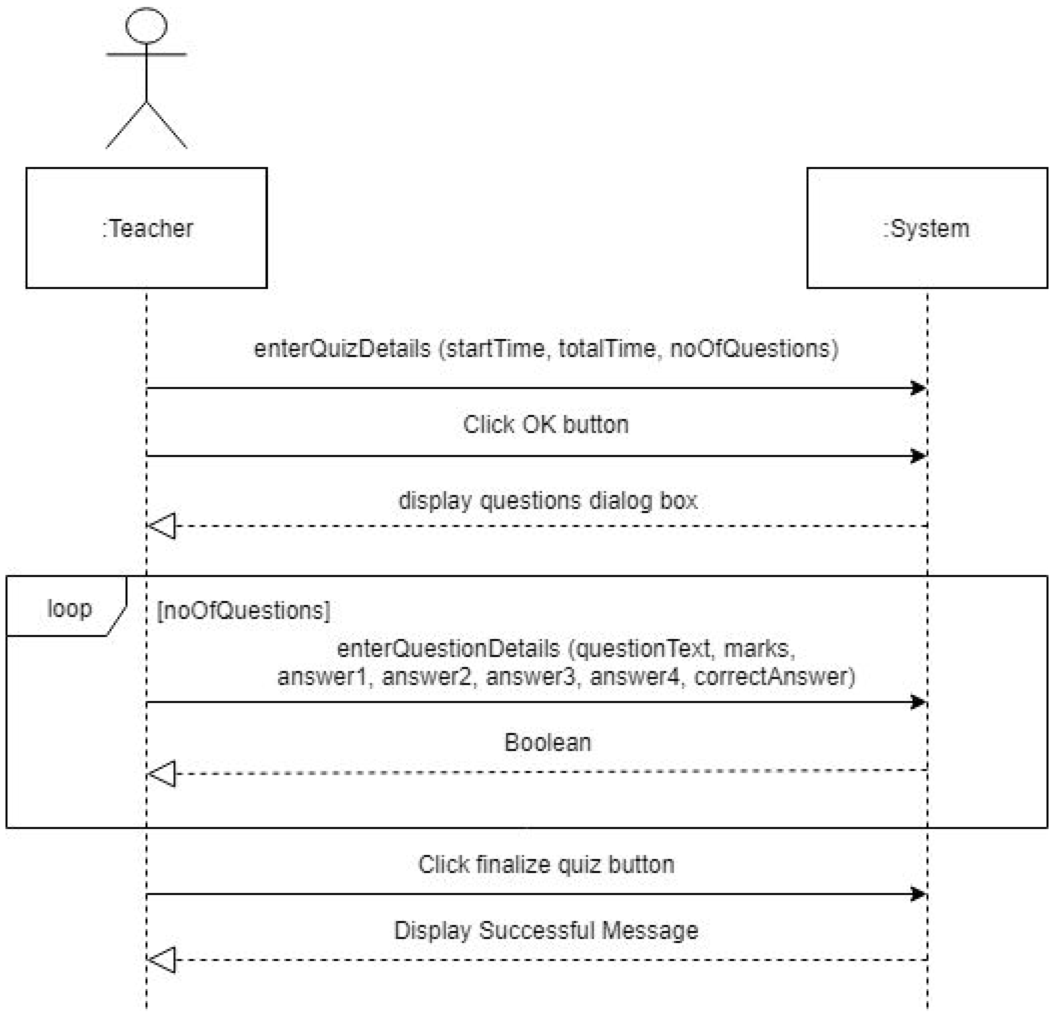
Use Case ID:	UC-1.6		
Use Case Name:	Create Quiz		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Teacher		
Description:	Teacher uses this feature to create quiz for a specific section with various questions and also providing their answers for an immediate calculation of grade upon submission by student.		
Trigger:	When the end-user clicks on create quiz button located on the header.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. Instance of Quiz is instantiated. 2. All of the information regarding the quiz is stored in the cloud server. 3. End-user is directed to the section page.		
Normal Flow:	1. End-user selects section. 2. End-user enters start time. 3. End-user enters time limit. 4. End-user selects number of questions. 5. End-user clicks OK button. 6. End-user enters question requested by the system. 7. End-user enters four answers. 8. End-user selects the correct answer. 9. End-user repeat steps -7 based on number of questions. 10. End-user clicks on Finalize Quiz button. 11. End-user is shown a successful message.		
Alternative Flows:	Nil		

Exceptions:	5a. In step 5 of the normal flow, if the end-user enters an invalid question including empty 1. Message to end-user that typed question is invalid and to re-enter question. 2. End-user enters correct question. 3. Use case resumes on step 9 of normal flow. 6a. In step 6 of the normal flow, if the end-user enters an invalid answer including empty 1. Message to end-user that typed answer is invalid and to re-enter answer. 2. End-user enters correct answer. 3. Use case resumes on step 9 of normal flow. 7a. In step 7 of the normal flow, if the end-user does not select one of the answers 1. Message to end-user to select the correct answer. 2. End-user selects the correct answer. 3. Use case resumes on step 9 of normal flow.
Includes:	Nil
Frequency of Use:	200 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to create quiz within 2 seconds after filling out all the fields 4. The system shall accept 5 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

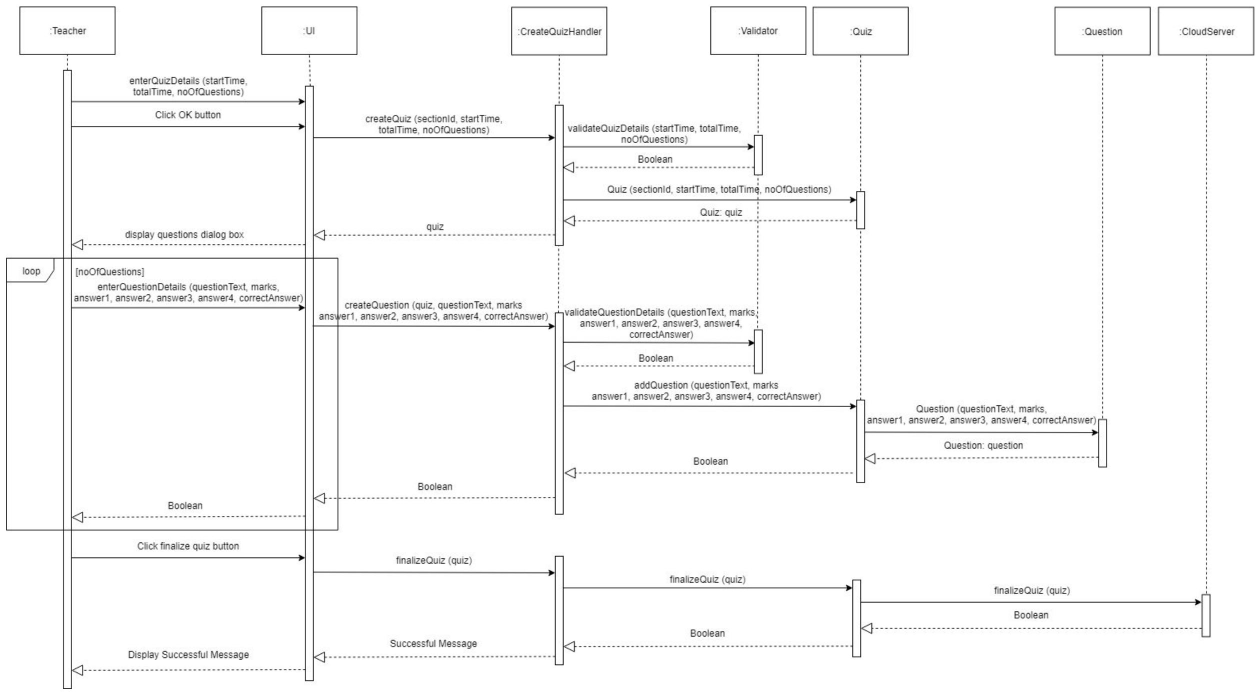
Domain Model:



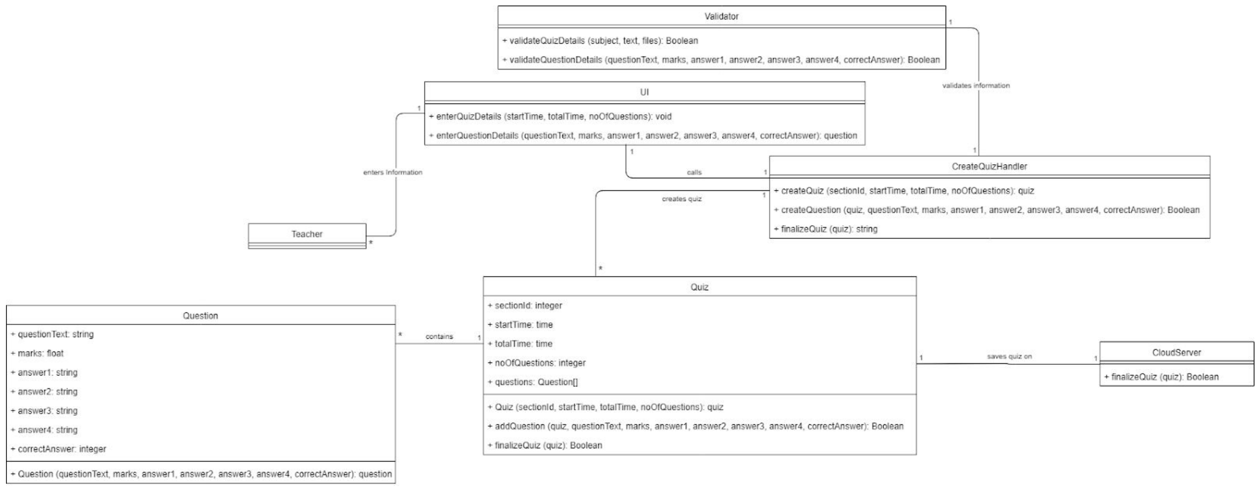
System Sequence Diagram:


:Teacher
:System
enterQuizDetails (startTime, totalTime, noOfQuestions)
Click OK button
display questions dialog box
loop
[noOfQuestions]
enterQuestionDetails (questionText, marks, answer1, answer2, answer3, answer4, correctAnswer)
Boolean
Click finalize quiz button
Display Successful Message

Sequence Diagram:


:Teacher
:UI
:CreateQuizHandler
:Validator
:Quiz
:Question
:CloudServer
enterQuizDetails (startTime, totalTime, noOfQuestions)
Click OK button
createQuiz (sectionId, startTime, totalTime, noOfQuestions)
validateQuizDetails (startTime, totalTime, noOfQuestions)
Boolean
Quiz (sectionId, startTime, totalTime, noOfQuestions)
Quiz: quiz
quiz
display questions dialog box
loop
[noOfQuestions]
enterQuestionDetails (questionText, marks, answer1, answer2, answer3, answer4, correctAnswer)
createQuestion (quiz, questionText, marks answer1, answer2, answer3, answer4, correctAnswer)
validateQuestionDetails (questionText, marks, answer1, answer2, answer3, answer4, correctAnswer)
Boolean
addQuestion (questionText, marks answer1, answer2, answer3, answer4, correctAnswer)
Question (questionText, marks, answer1, answer2, answer3, answer4, correctAnswer)
Question: question
Boolean
Boolean
Boolean
Click finalize quiz button
finalizeQuiz (quiz)
finalizeQuiz (quiz)
finalizeQuiz (quiz)
Boolean
Boolean
Successful Message
Display Successful Message

Class Diagram:


Validator
+ validateQuizDetails (subject, text, files): Boolean
+ validateQuestionDetails (questionText, marks, answer1, answer2, answer3, answer4, correctAnswer): Boolean
1
validates information
1
UI
+ enterQuizDetails (startTime, totalTime, noOfQuestions): void
+ enterQuestionDetails (questionText, marks, answer1, answer2, answer3, answer4, correctAnswer): question
1
enters Information
*
Teacher
1
calls
1
CreateQuizHandler
+ createQuiz (sectionId, startTime, totalTime, noOfQuestions): quiz
+ createQuestion (quiz, questionText, marks, answer1, answer2, answer3, answer4, correctAnswer): Boolean
+ finalizeQuiz (quiz): string
1
creates quiz
*
Quiz
+ sectionId: integer
+ startTime: time
+ totalTime: time
+ noOfQuestions: integer
+ questions: Question[]
+ Quiz (sectionId, startTime, totalTime, noOfQuestions): quiz
+ addQuestion (quiz, questionText, marks, answer1, answer2, answer3, answer4, correctAnswer): Boolean
+ finalizeQuiz (quiz): Boolean
Question
+ questionText: string
+ marks: float
+ answer1: string
+ answer2: string
+ answer3: string
+ answer4: string
+ correctAnswer: integer
+ Question (questionText, marks, answer1, answer2, answer3, answer4, correctAnswer): question
*
contains
1
1
saves quiz on
1
CloudServer
+ finalizeQuiz (quiz): Boolean

Patterns:

1. **Controller**
   a. **Problem:**
      What first object beyond the UI layer receives and coordinates a system operation?

   b. **Solution:**
      Assign the responsibility to the class CreateQuizHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Creator**
   a. **Problem:**
      Who should be responsible for creating a new instance of Question class?
   b. **Solution:**
      Assign the responsibility to the class Question because Quiz contains Questions which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

3. **Low Coupling**
   a. **Problem:**
      How to support low dependency, low change impact and increased reuse in saving the Quiz on Cloud Server after creation of Quiz?
   b. **Solution:**
      Assign the responsibility to the class CloudServer to save information of an instance of Quiz to the Cloud Server instead of Quiz which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

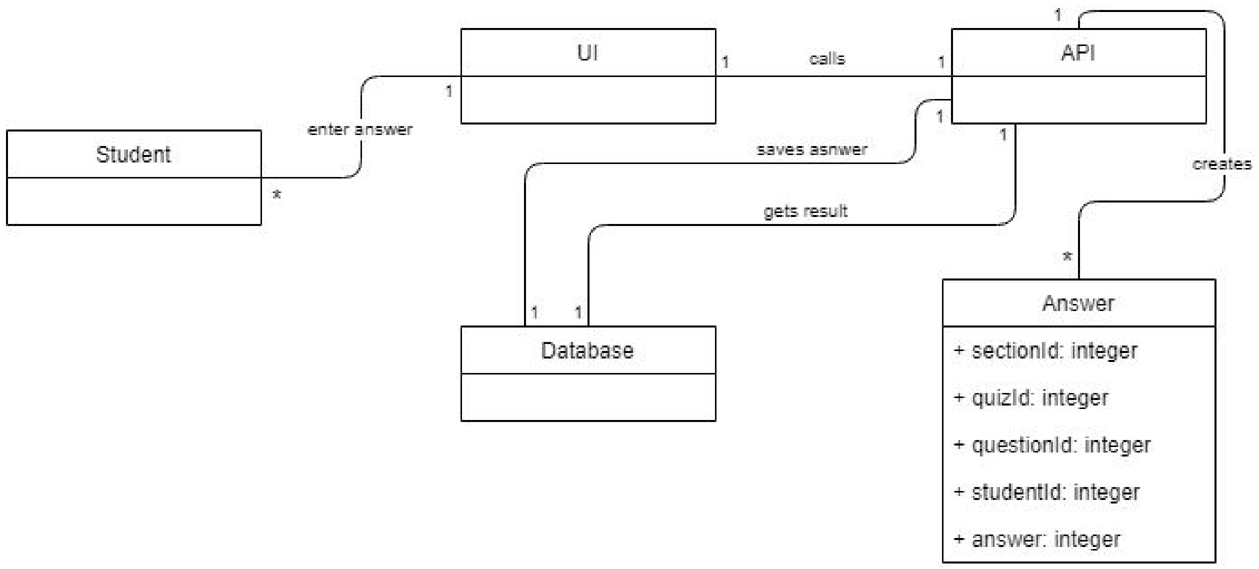
2.2.7 Submit Quiz

This is used by student. End-user can use it to submit a quiz for a specific section. Priority is medium.

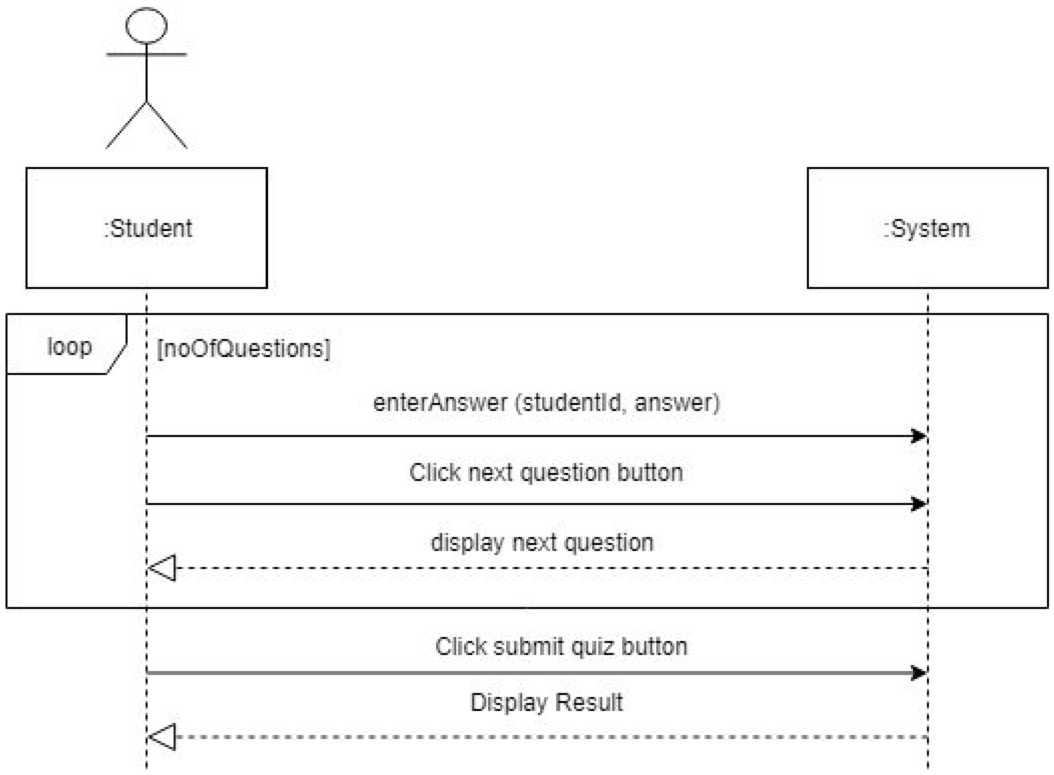
Use Case ID:	UC-1.7		
Use Case Name:	Submit Quiz		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	

Actors:	Student
Description:	Student uses this feature to submit quiz for a specific section answering all the questions and receiving result of the quiz on the spot.
Trigger:	When the end-user clicks on attempt quiz button located on a specific section page.
Preconditions:	1. The end-user must be logged in.
Postconditions:	1. Instances of Answer are instantiated based on the number of questions. 2. All of the information regarding the answers and quiz marks are stored in the cloud server. 3. End-user is shown the result of the quiz.
Normal Flow:	1. End-user selects an answer for a question (radio button). 2. End-user clicks on next question button. 3. End-user repeat steps 1-2 based on number of questions. 4. End-user clicks on Submit Quiz button. 5. End-user is shown the result of the quiz.
Alternative Flows:	Nil
Exceptions:	1a. In step 1 of the normal flow, if the end-user does not select one of the answers before pressing the next question button 1. Message to end-user to select an answer. 2. End-user selects an answer. 3. Use case resumes on step 3 of normal flow.
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available during a quiz 3. The end-user shall be able to submit quiz within 1 second after filling out all the fields 4. The system shall accept 200 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

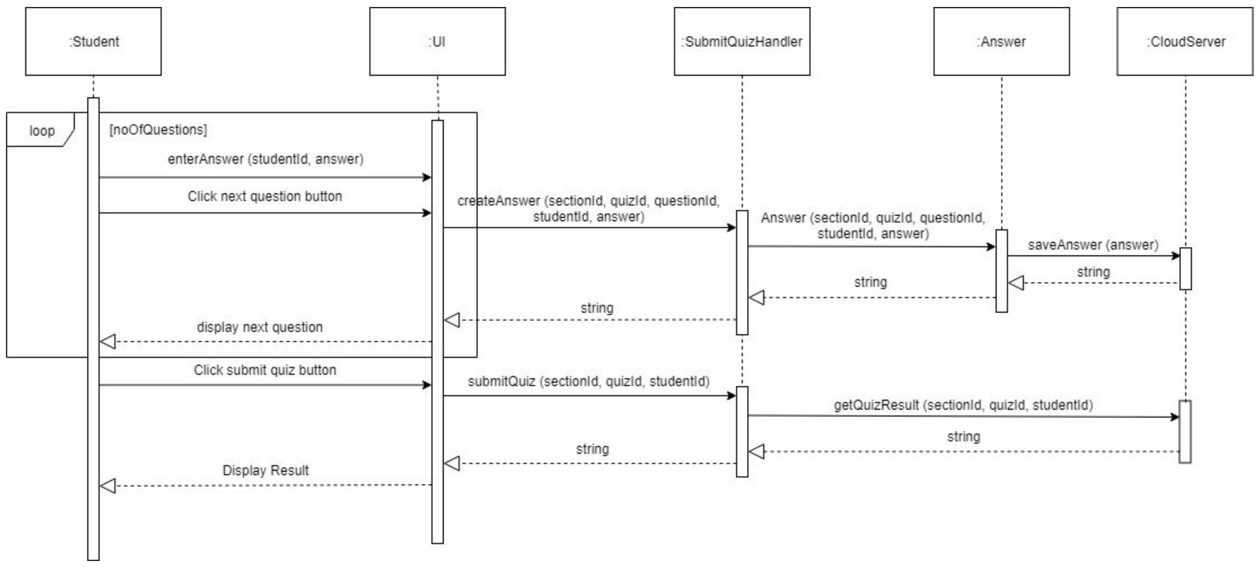
Domain Model:



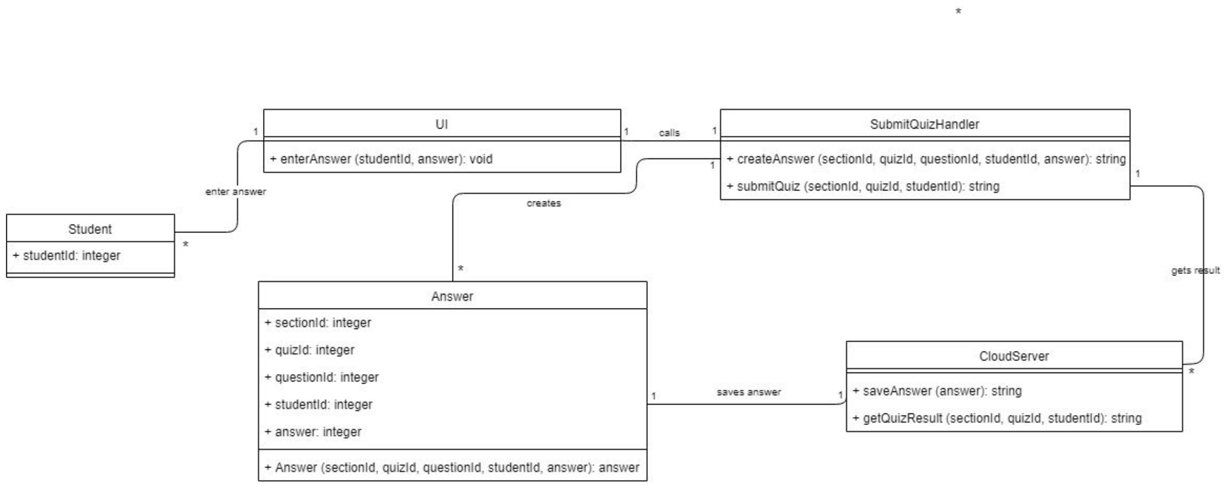
System Sequence Diagram:



Sequence Diagram:


:Student
:UI
:SubmitQuizHandler
:Answer
:CloudServer
loop
[noOfQuestions]
enterAnswer (studentId, answer)
Click next question button
createAnswer (sectionId, quizId, questionId, studentId, answer)
Answer (sectionId, quizId, questionId, studentId, answer)
saveAnswer (answer)
string
string
string
display next question
Click submit quiz button
submitQuiz (sectionId, quizId, studentId)
getQuizResult (sectionId, quizId, studentId)
string
string
Display Result

Class Diagram:


UI
+ enterAnswer (studentId, answer): void
SubmitQuizHandler
+ createAnswer (sectionId, quizId, questionId, studentId, answer): string
+ submitQuiz (sectionId, quizId, studentId): string
Student
+ studentId: integer
Answer
+ sectionId: integer
+ quizId: integer
+ questionId: integer
+ studentId: integer
+ answer: integer
+ Answer (sectionId, quizId, questionId, studentId, answer): answer
CloudServer
+ saveAnswer (answer): string
+ getQuizResult (sectionId, quizId, studentId): string
calls
enter answer
creates
gets result
saves answer
1
1
1
1
1
1
1
1
*
*
*
*

Patterns:

1. **Controller**
   a. **Problem:**
      What first object beyond the UI layer receives and coordinates a system operation?
   b. **Solution:**
      Assign the responsibility to the class SubmitQuizHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Low Coupling**
    a. **Problem:**
    How to support low dependency, low change impact and increased reuse in saving the Answer on Cloud Server after creation of Answer?
    b. **Solution:**
    Assign the responsibility to the class CloudServer to save information of an instance of Answer to the Cloud Server instead of Answer which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

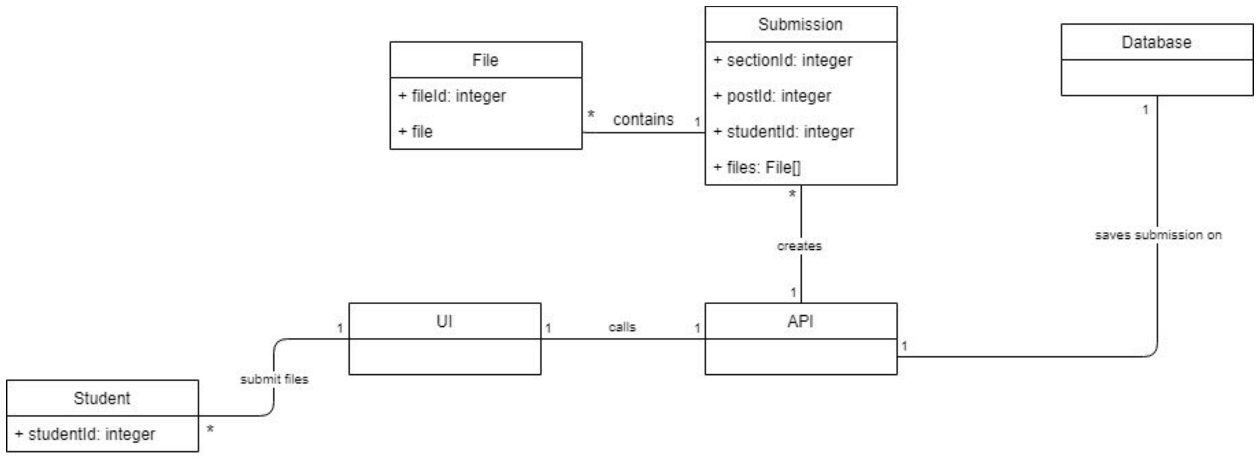
2.2.8 Submit Assignment

This is used by student. End-user can use it to submit an assignment for a specific section. Priority is medium.

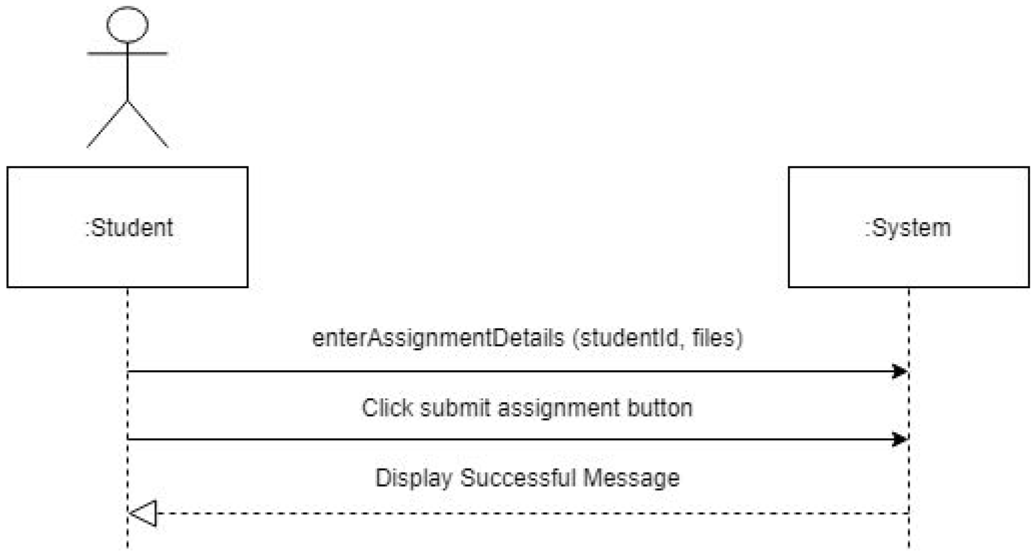
Use Case ID:	UC-1.8		
Use Case Name:	Submit Assignment		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Student		
Description:	Student uses this feature to submit assignment for a specific section by uploading the required files.		
Trigger:	When the end-user clicks on submit assignment button besides an assignment located on a specific section page.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. Instance of Submission is instantiated. 2. All of the assignment files are stored in the cloud server. 3. End-user is directed to the section page.		
Normal Flow:	1. End-user clicks on Attach Files button to upload file(s). 2. End-user clicks on Submit Assignment button. 3. End-user is shown a successful message.		
Alternative Flows:	Nil		
Exceptions:	1a. In step 1 of the normal flow, if the file size is larger than 10MB 1. Message to end-user that the file size is larger than 10MB and to attach file(s) again within the size limit. 2. End-User clicks on Attach Files button to upload file(s) within the size limit. 3. Use case resumes on step 2 of normal flow.		
Includes:	Nil		
Frequency of Use:	500 per day		

Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available during assignment time limit 3. The end-user shall not be able to upload malicious files 4. The end-user shall be able to submit assignment within 1 second after clicking the submit button 5. The system shall accept 200 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

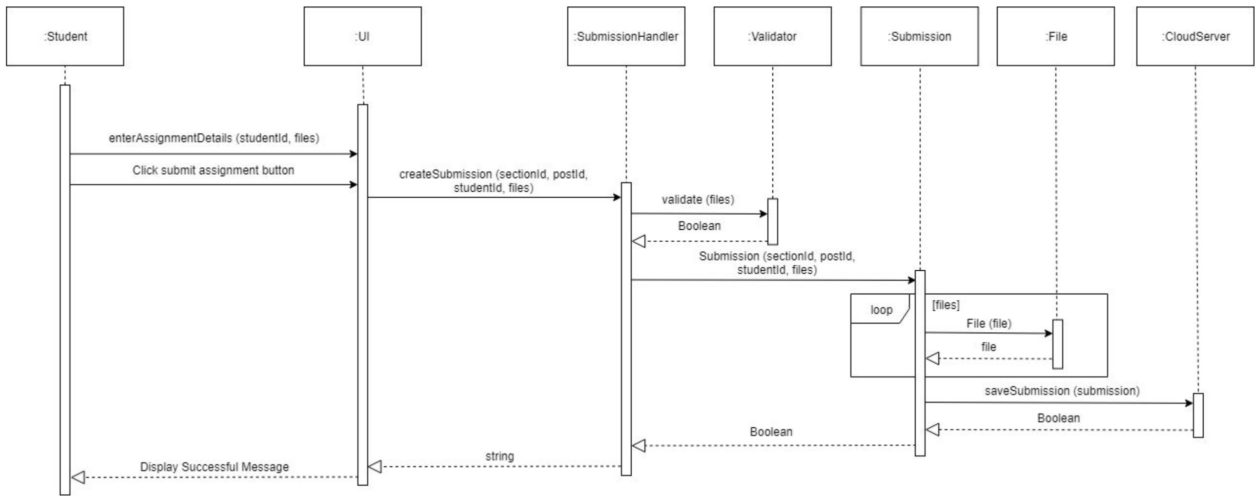
Domain Model:



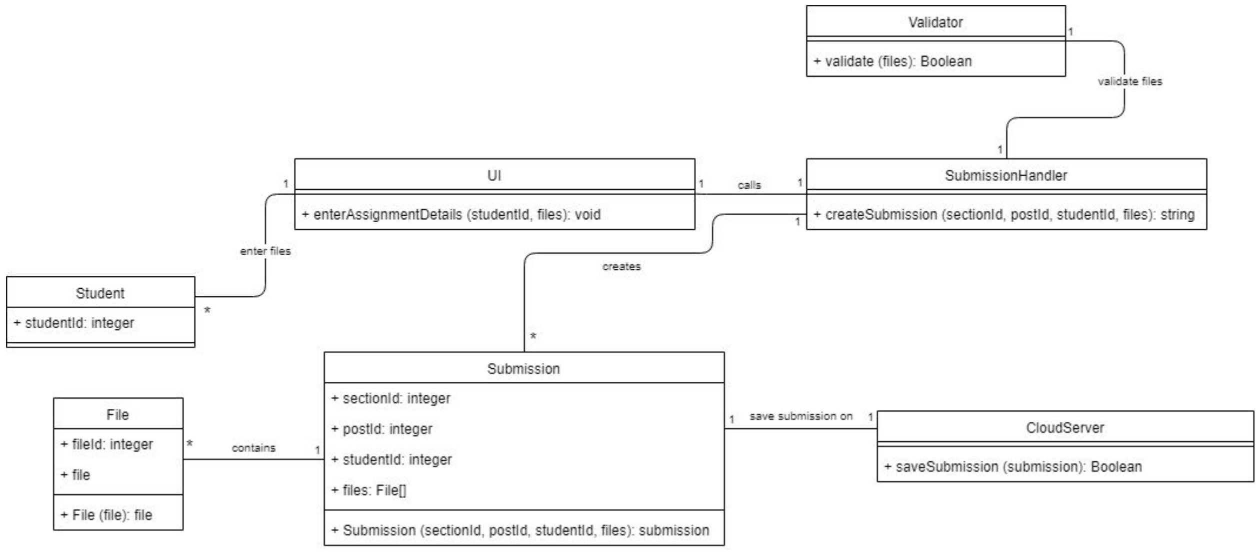
System Sequence Diagram:


:Student
:System
enterAssignmentDetails (studentId, files)
Click submit assignment button
Display Successful Message

Sequence Diagram:


:Student
:UI
:SubmissionHandler
:Validator
:Submission
:File
:CloudServer
enterAssignmentDetails (studentId, files)
Click submit assignment button
createSubmission (sectionId, postId, studentId, files)
validate (files)
Boolean
Submission (sectionId, postId, studentId, files)
loop
[files]
File (file)
file
saveSubmission (submission)
Boolean
Boolean
string
Display Successful Message

Class Diagram:



Patterns:

1. **Controller**
   a. **Problem:**
      What first object beyond the UI layer receives and coordinates a system operation?
   b. **Solution:**
      Assign the responsibility to the class SubmissionHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Low Coupling**
    a. **Problem:**
    How to support low dependency, low change impact and increased reuse in saving the Submission on Cloud Server after creation of Submission?
    b. **Solution:**
    Assign the responsibility to the class CloudServer to save information of an instance of Submission to the Cloud Server instead of Submission which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

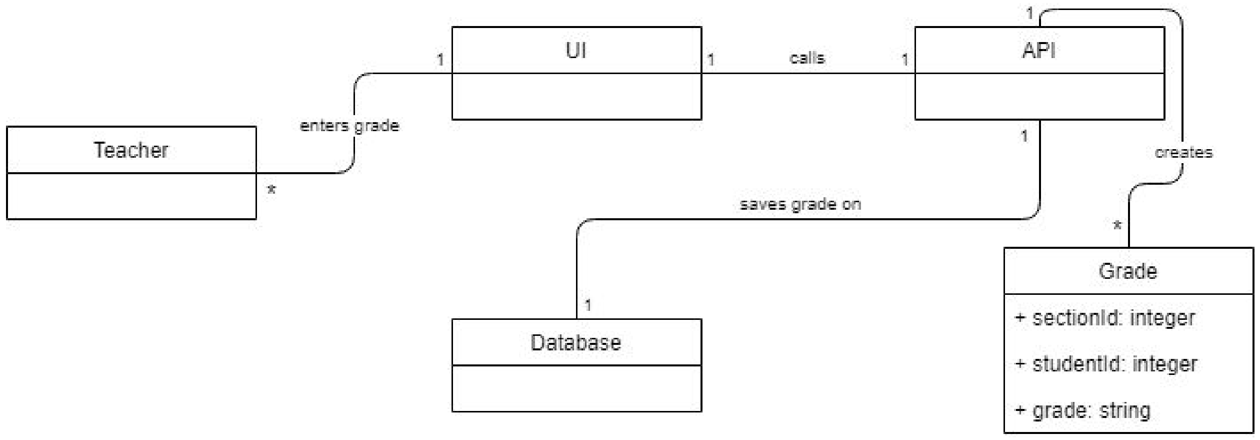
2.2.9 Upload Grade

This use case is used by a Teacher in order to upload a letter Grade for a specific assignment, for all students.

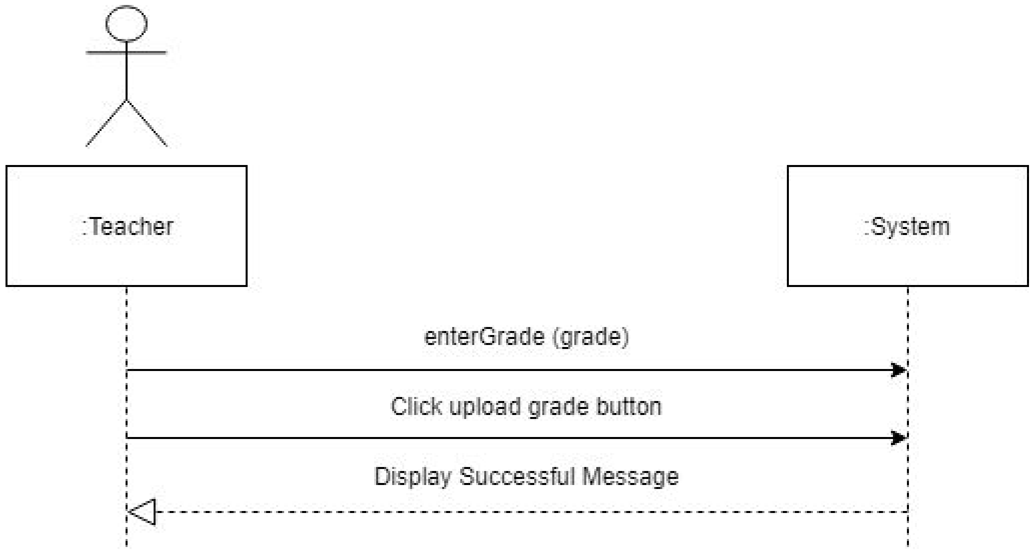
Use Case ID:	UC-1.9		
Use Case Name:	Upload Grade		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	

Actors:	Teacher
Description:	A Teacher uses this feature to upload a letter Grade for a particular student and an assignment.
Trigger:	When the end-user clicks on upload grade button besides the student name in a particular section.
Preconditions:	1. The end-user must be logged in.
Postconditions:	1. Instance of Grade is instantiated. 2. The Grade is stored in the cloud server.
Normal Flow:	1. End-user selects grade. 2. End-user clicks on Upload Grade button. 3. End-user is shown a successful message.
Alternative Flows:	Nil
Exceptions:	Nil
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to upload grade within 0.5 seconds after clicking the upload grade button 4. The system shall accept 200 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

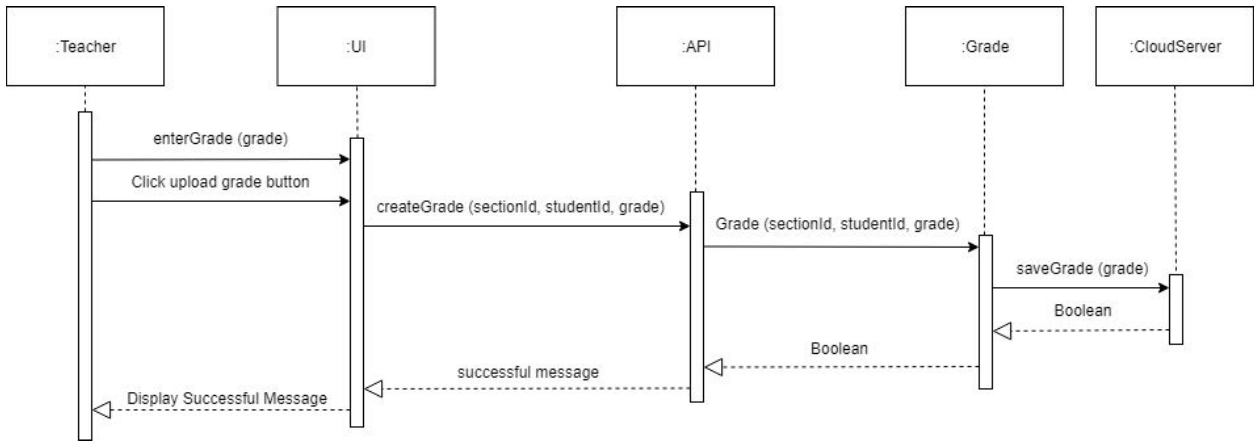
Domain Model:



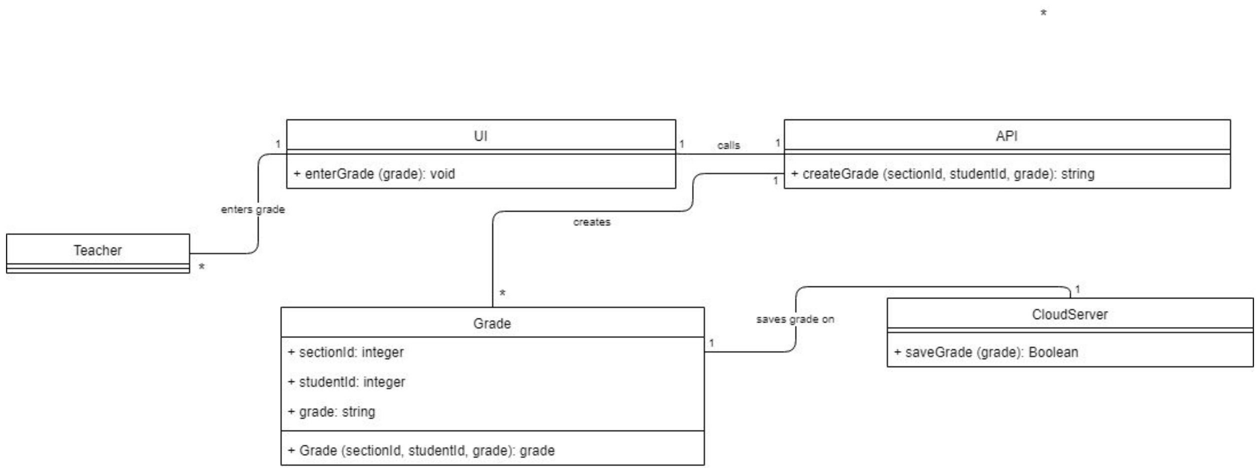
System Sequence Diagram:



Sequence Diagram:


:Teacher
:UI
:API
:Grade
:CloudServer
enterGrade (grade)
Click upload grade button
createGrade (sectionId, studentId, grade)
Grade (sectionId, studentId, grade)
saveGrade (grade)
Boolean
Boolean
successful message
Display Successful Message

Class Diagram:


UI
+ enterGrade (grade): void
API
+ createGrade (sectionId, studentId, grade): string
calls
enters grade
creates
Teacher
Grade
+ sectionId: integer
+ studentId: integer
+ grade: string
+ Grade (sectionId, studentId, grade): grade
saves grade on
CloudServer
+ saveGrade (grade): Boolean
1
*

Patterns:

1. **Controller**
    a. **Problem:**
    What first object beyond the UI layer receives and coordinates a system operation?
    b. **Solution:**
    Assign the responsibility to the class UploadGradeHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Low Coupling**
    a. **Problem:**
    How to support low dependency, low change impact and increased reuse in saving the Grade on Cloud Server after creation of Submission?

b. **Solution:**

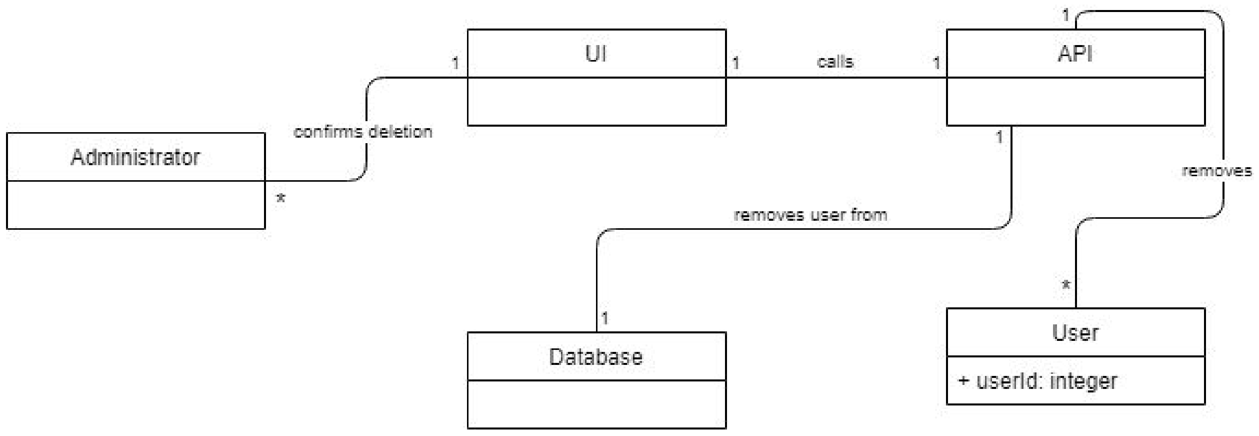
Assign the responsibility to the class CloudServer to save information of an instance of Grade to the Cloud Server instead of Grade which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

2.2.10 Remove User

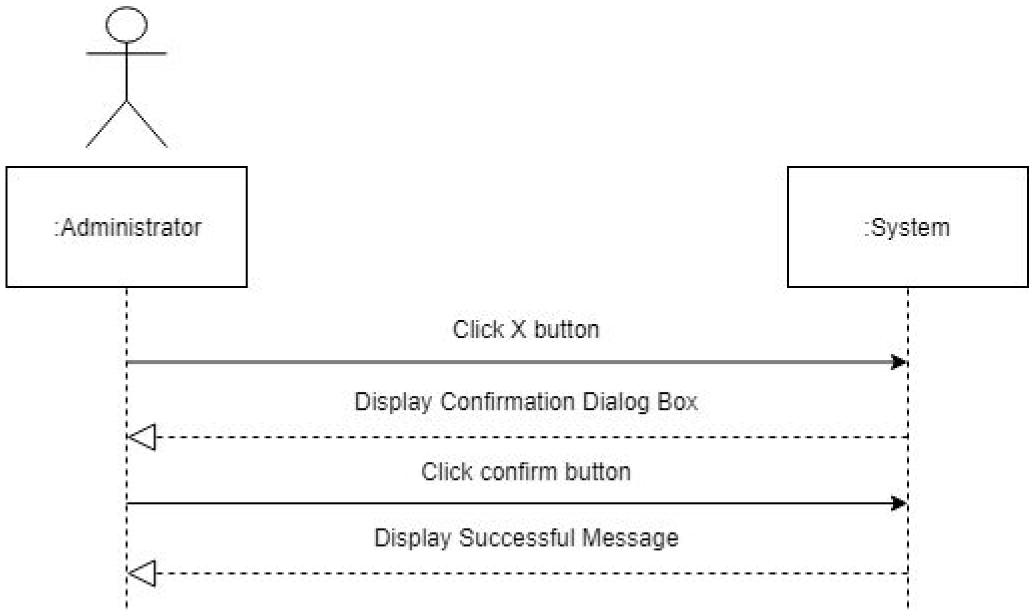
This use case is used by an Administrator in order to delete a user from the LMS.

Use Case ID:	UC-1.10		
Use Case Name:	Remove User		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Administrator		
Description:	An Administrator uses this feature to remove a particular user of the LMS.		
Trigger:	When the end-user clicks on ‘X’ button besides the teacher/student name in a column containing all users.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. The end-user’s information is removed from the cloud server.		
Normal Flow:	1. End-user clicks the ‘X’ button besides a particular user. 2. System shows a confirmation dialog box. 3. End-user clicks on Confirm button. 4. End-user is shown a successful message.		
Alternative Flows:	Nil		
Exceptions:	Nil		
Includes:	Nil		
Frequency of Use:	500 per day		
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to remove end-user within 0.5 seconds after clicking the confirm button 4. The system shall accept 200 requests per second		
Assumptions:	Nil		
Notes and Issues:	Nil		

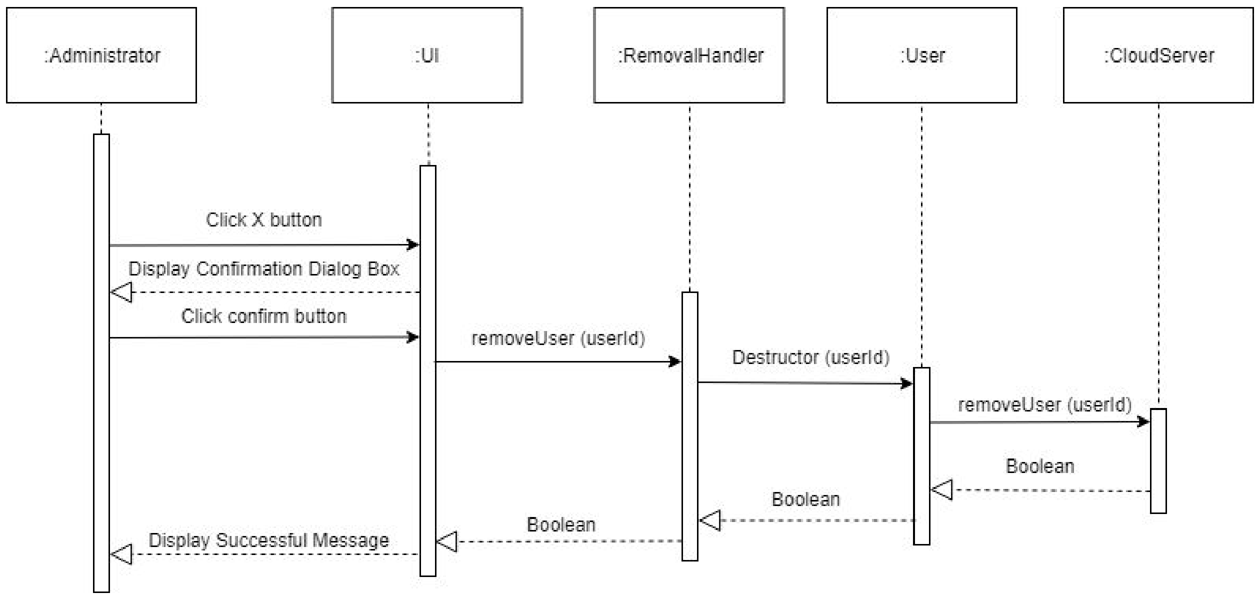
Domain Model:



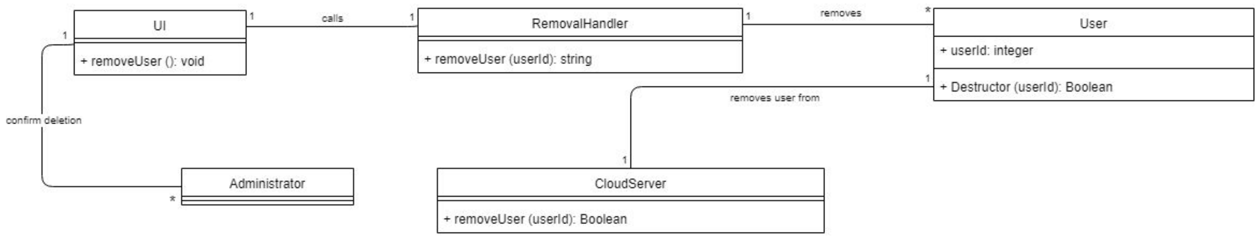
System Sequence Diagram:



Sequence Diagram:



Class Diagram:



Patterns:

1. **Controller**
    a. **Problem:**
    What first object beyond the UI layer receives and coordinates a system operation?
    b. **Solution:**
    Assign the responsibility to the class RemovalHandler which will result in increased potential for reuse and pluggable interfaces. It will ensure that system operations occur in a legal sequence.

2. **Low Coupling**
    a. **Problem:**
    How to support low dependency, low change impact and increased reuse in removing the User from the Cloud Server after destroying the instance of User?

b. **Solution:**

Assign the responsibility to the class CloudServer to remove information of an instance of User from the Cloud Server instead of User which will result in convenience in reuse, improved understandability in isolation and not affected by changes in other components.

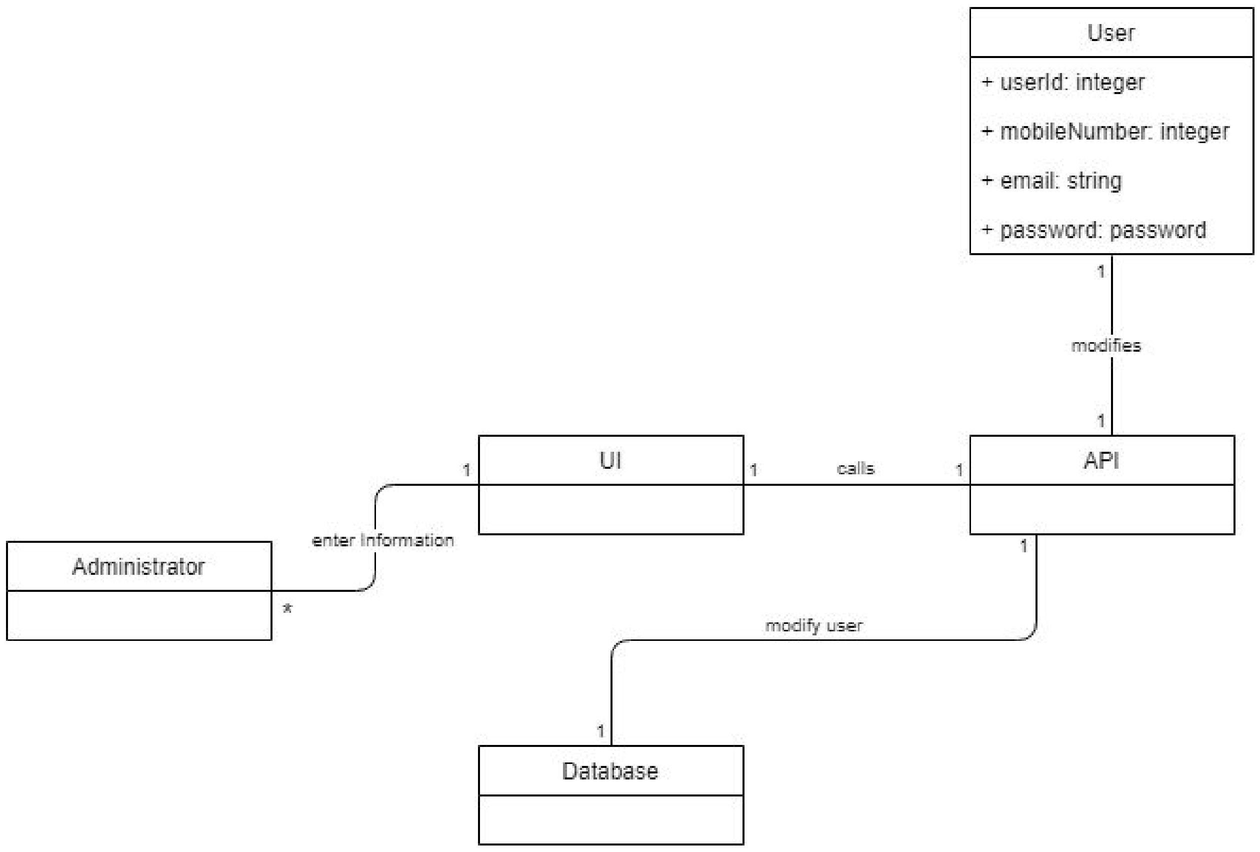
2.2.11 Modify User

This use case is used by an Administrator to modify the account information of a particular end-user.

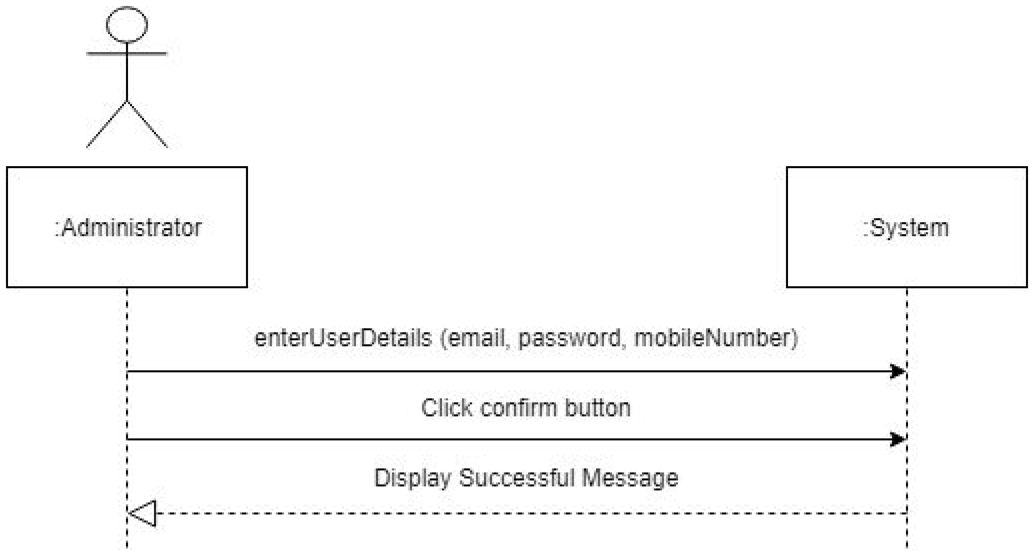
Use Case ID:	UC-1.11		
Use Case Name:	Modify User		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Administrator		
Description:	An Administrator uses this feature to update or modify the general information of a particular user of the LMS.		
Trigger:	When the end-user clicks the 'Modify' button for a specific user.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. The end-user's information is updated on the cloud server.		
Normal Flow:	1. End-user clicks the 'Modify' button in front of a particular user. 2. End-user enters new email (optional). 3. End-user enters new password (optional). 4. End-user enters mobile number (optional). 5. End-user clicks on Confirm button. 6. End-user is shown a successful message.		
Alternative Flows:	Nil		
Exceptions:	2a. In step 2 of the normal flow, if the Administrator enters an email which is already registered 1. Message to administrator that the typed email is already in use and to re-enter email address. 2. Administrator enters another email address. 3. Use case resumes on step 5 of normal flow. 2a. In step 2 of the normal flow, if the Administrator enters an invalid Email address 1. Message to administrator that typed email address is invalid and to re-enter email address. 2. Administrator enters correct email address. 3. Use case resumes on step 5 of normal flow.		

	3a. In step 3 of the normal flow, if the Administrator enters an invalid password 1. Message to administrator that typed password is invalid and to re-enter password. 2. Administrator enters correct password. 3. Use case resumes on step 5 of normal flow. 4a. In step 4 of the normal flow, if the Administrator enters a mobile number which is already registered 1. Message to administrator that typed mobile number is invalid and to re-enter mobile number. 2. Administrator enters another mobile number. 3. Use case resumes on step 5 of normal flow. 4a. In step 4 of the normal flow, if the Administrator enters an invalid mobile number 1. Message to administrator that typed mobile number is invalid and to re-enter mobile number. 2. Administrator enters correct mobile number. 3. Use case resumes on step 5 of normal flow.
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to update end-user within 0.5 seconds after clicking the confirm button 4. The password should be at least 6 characters long containing characters (0-9, a-z, A-Z, _, #) 5. The system shall accept 5 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

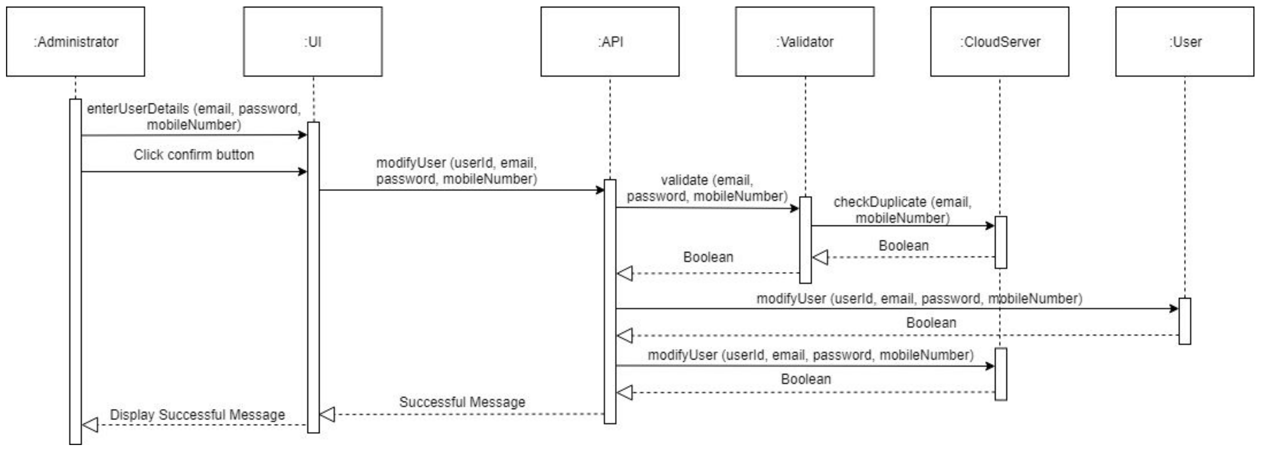
Domain Model:



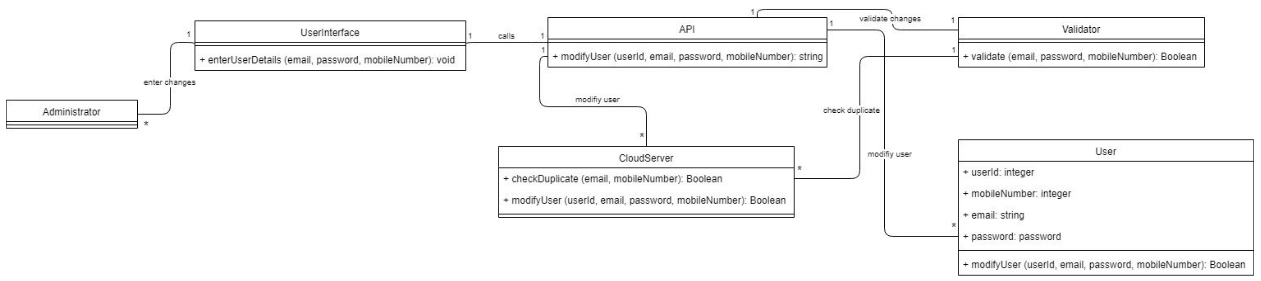
System Sequence Diagram:


:Administrator
:System
enterUserDetails (email, password, mobileNumber)
Click confirm button
Display Successful Message

Sequence Diagram:


:Administrator
:UI
:API
:Validator
:CloudServer
:User
enterUserDetails (email, password, mobileNumber)
Click confirm button
modifyUser (userId, email, password, mobileNumber)
validate (email, password, mobileNumber)
checkDuplicate (email, mobileNumber)
Boolean
Boolean
modifyUser (userId, email, password, mobileNumber)
Boolean
modifyUser (userId, email, password, mobileNumber)
Boolean
Successful Message
Display Successful Message

Class Diagram:


UserInterface
+ enterUserDetails (email, password, mobileNumber): void
Administrator
enter changes
calls
API
+ modifyUser (userId, email, password, mobileNumber): string
validate changes
Validator
+ validate (email, password, mobileNumber): Boolean
modify user
check duplicate
modify user
CloudServer
+ checkDuplicate (email, mobileNumber): Boolean
+ modifyUser (userId, email, password, mobileNumber): Boolean
User
+ userId: integer
+ mobileNumber: integer
+ email: string
+ password: password
+ modifyUser (userId, email, password, mobileNumber): Boolean

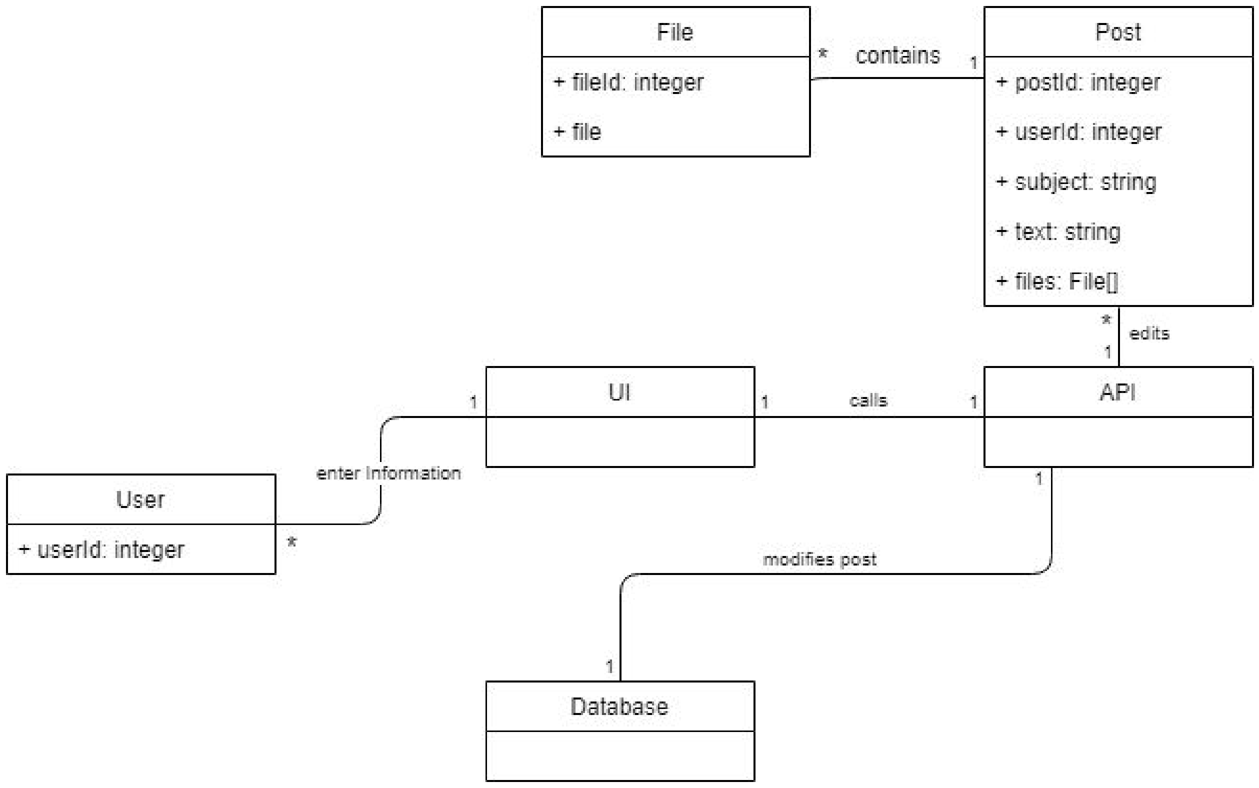
2.2.12 Edit Post

Teacher/Student uses this feature to modify the contents of a particular Post.

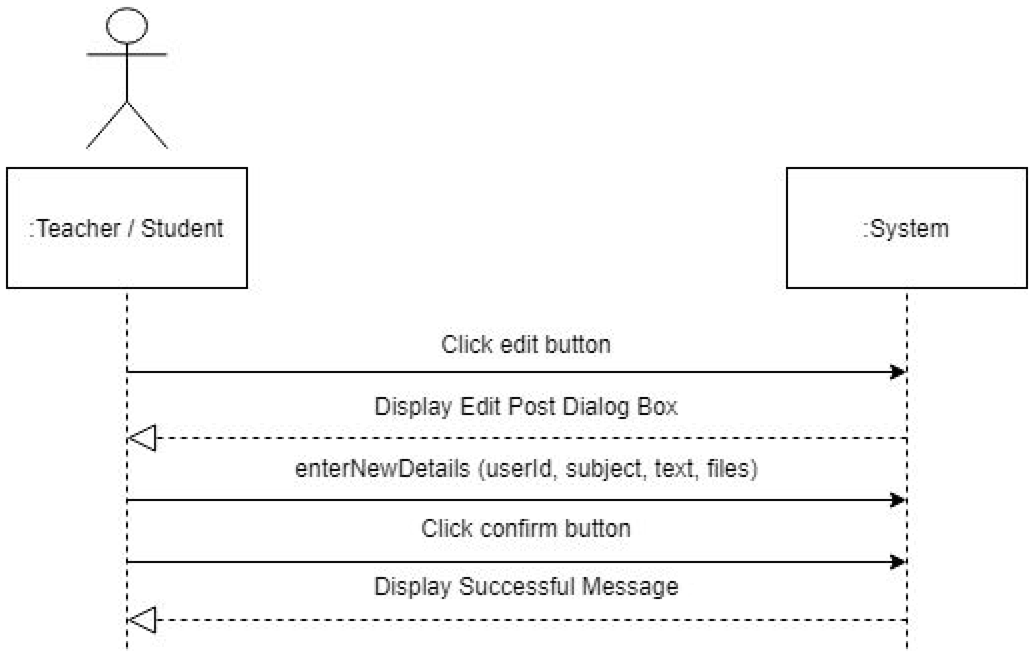
Use Case ID:	UC-1.12		
Use Case Name:	Edit Post		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Teacher/Student		
Description:	End-user uses this feature to edit a Post.		
Trigger:	When the end-user clicks the ‘Edit’ button besides his/her post.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. The post’s information is updated on the cloud server.		
Normal Flow:	1. End-user clicks the ‘Edit’ button in front of a post. 2. System displays an edit post subject, post text and files dialog box. 3. End-user enters the changes in the provided field. 4. End-user clicks on Confirm button. 5. End-user is shown a successful message.		

Alternative Flows:	Nil
Exceptions:	2a. In step 2 of the normal flow, if the end-user enters an invalid subject, text (including empty) or if file size is greater than 5MB 1. Message to end-user about the error. 2. End-user enters correct subject, text or file(s). 3. Use case resumes on step 4 of normal flow.
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to update post within 0.5 seconds after clicking the confirm button 4. The system shall accept 25 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

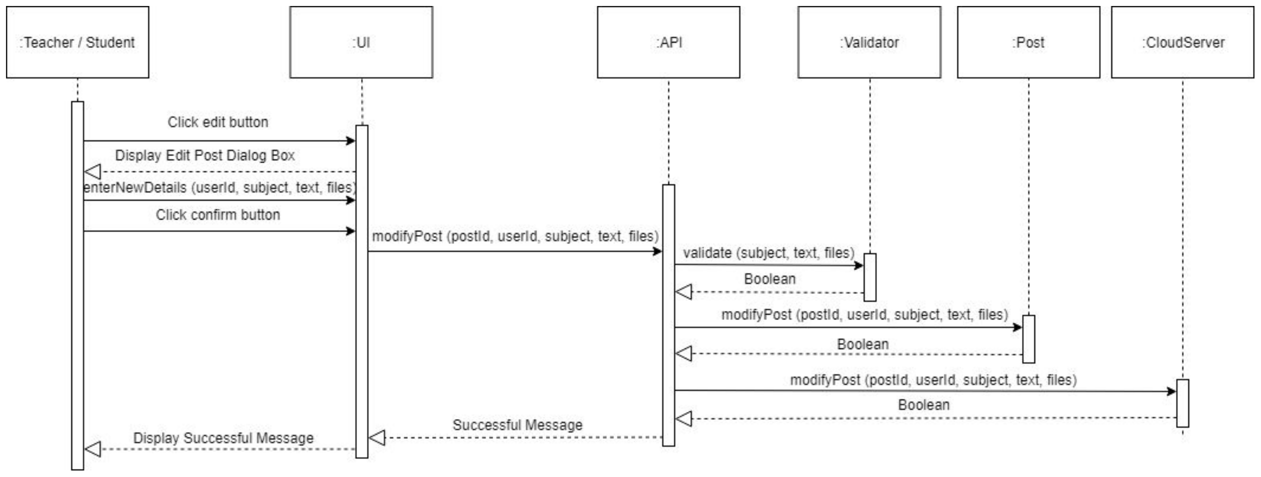
Domain Model:



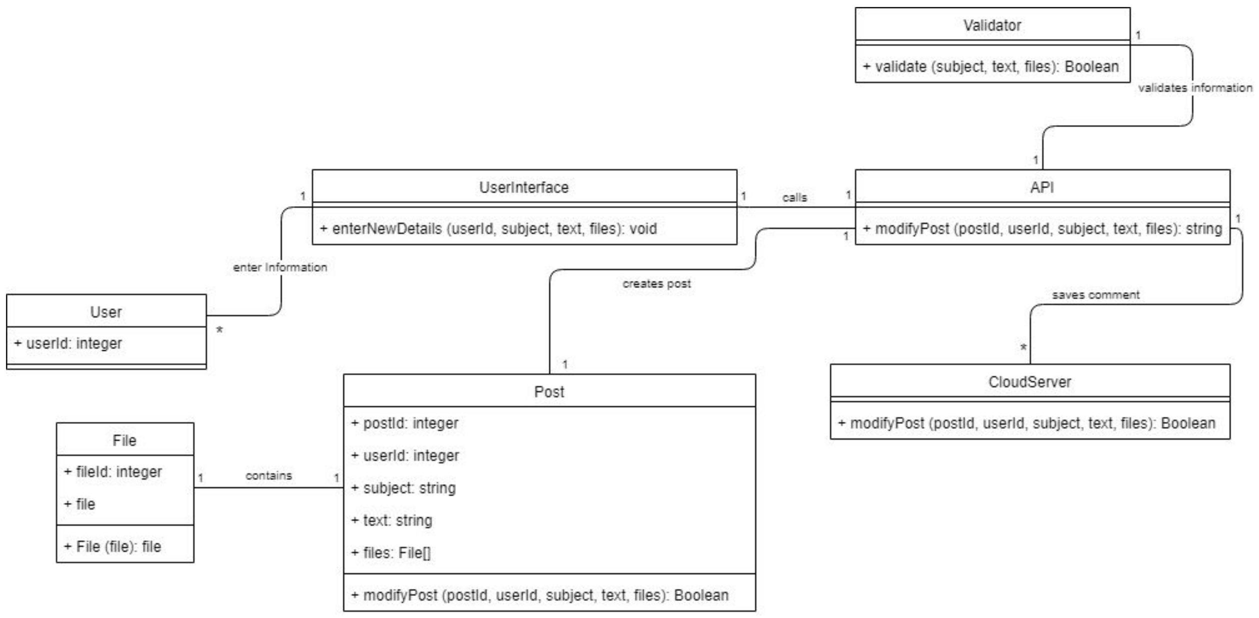
System Sequence Diagram:


:Teacher / Student
:System
Click edit button
Display Edit Post Dialog Box
enterNewDetails (userId, subject, text, files)
Click confirm button
Display Successful Message

Sequence Diagram:


:Teacher / Student
:UI
:API
:Validator
:Post
:CloudServer
Click edit button
Display Edit Post Dialog Box
enterNewDetails (userId, subject, text, files)
Click confirm button
modifyPost (postId, userId, subject, text, files)
validate (subject, text, files)
Boolean
modifyPost (postId, userId, subject, text, files)
Boolean
modifyPost (postId, userId, subject, text, files)
Boolean
Successful Message
Display Successful Message

Class Diagram:


Validator
+ validate (subject, text, files): Boolean
validates information
UserInterface
+ enterNewDetails (userId, subject, text, files): void
calls
API
+ modifyPost (postId, userId, subject, text, files): string
enter Information
User
+ userId: integer
creates post
saves comment
CloudServer
+ modifyPost (postId, userId, subject, text, files): Boolean
Post
+ postId: integer
+ userId: integer
+ subject: string
+ text: string
+ files: File[]
+ modifyPost (postId, userId, subject, text, files): Boolean
File
+ fileId: integer
+ file
+ File (file): file
contains
1
*

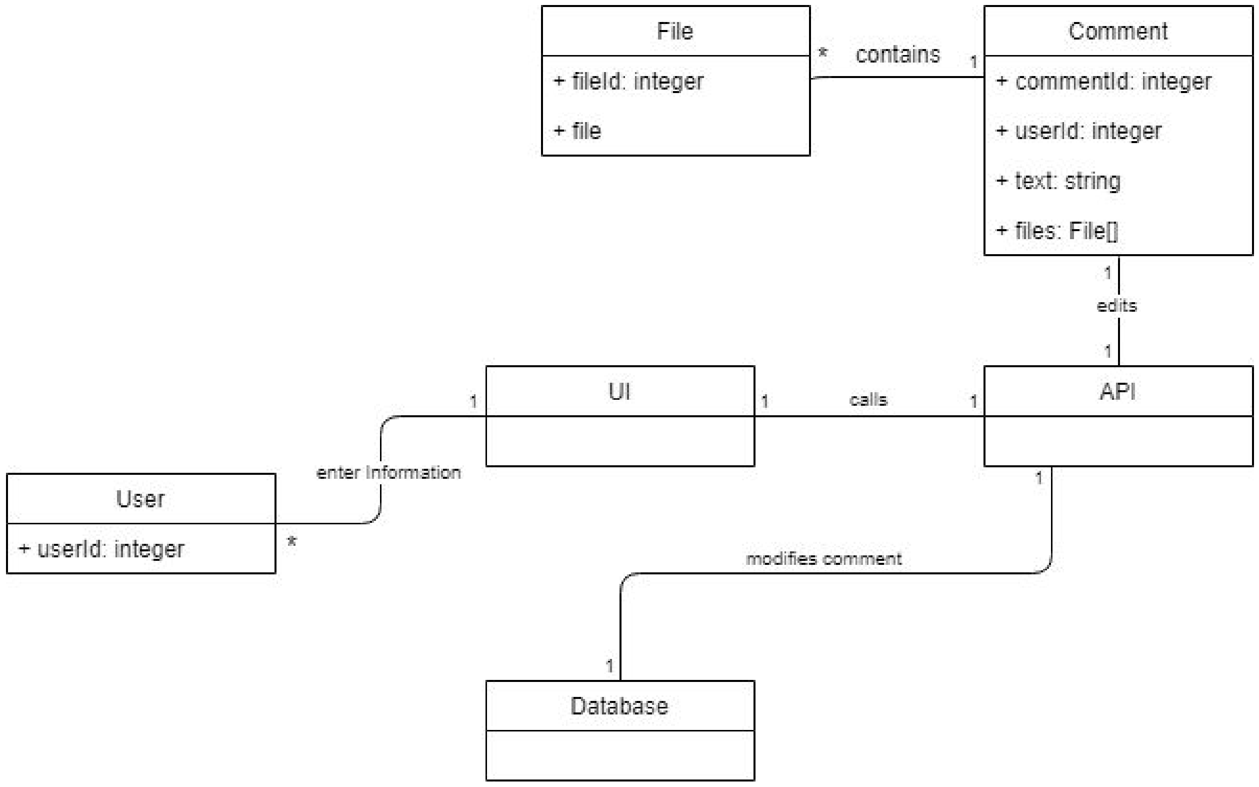
2.2.13 Edit Comment

Teacher/Student uses this feature to modify the contents of a particular Comment.

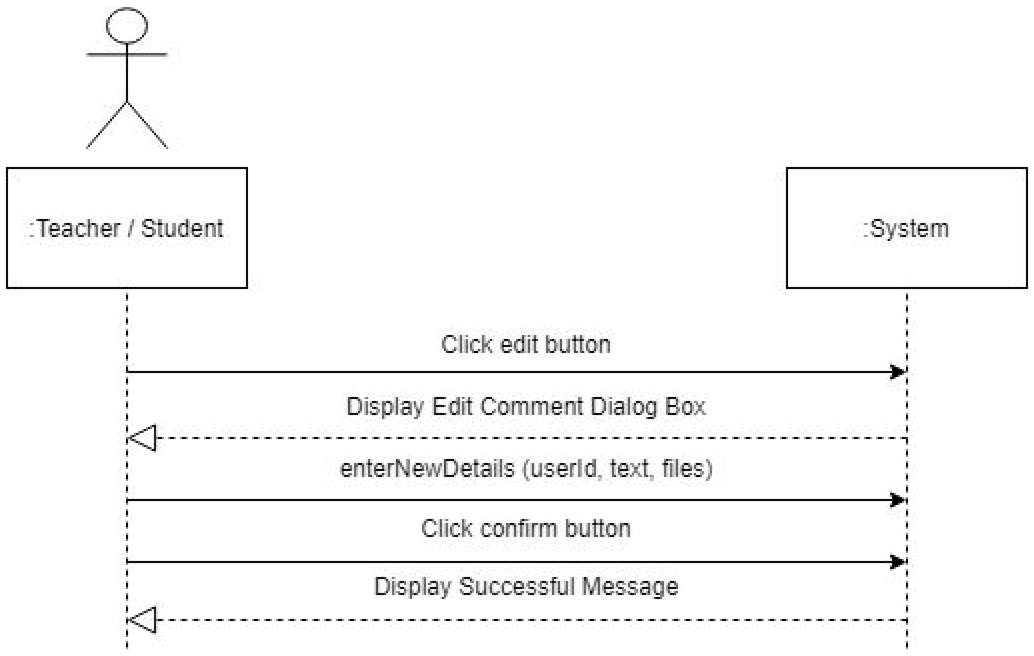
Use Case ID:	UC-1.13		
Use Case Name:	Edit Comment		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Teacher/Student		
Description:	End-user uses this feature to edit a Comment.		
Trigger:	When the end-user clicks the 'Edit' button besides his/her comment.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. The comment's information is updated on the cloud server.		
Normal Flow:	1. End-user clicks the 'Edit' button in front of a comment. 2. System displays an edit comment text and files dialog box. 3. End-user enters the changes in the provided field. 4. End-user clicks on Confirm button. 5. End-user is shown a successful message.		
Alternative Flows:	Nil		

Exceptions:	2a. In step 2 of the normal flow, if the end-user enters an invalid text (including empty) or if file size is greater than 5MB 1. Message to end-user about the error. 2. End-user enters correct subject, text or file(s). 3. Use case resumes on step 4 of normal flow.
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to update comment within 0.5 seconds after clicking the confirm button 4. The system shall accept 25 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

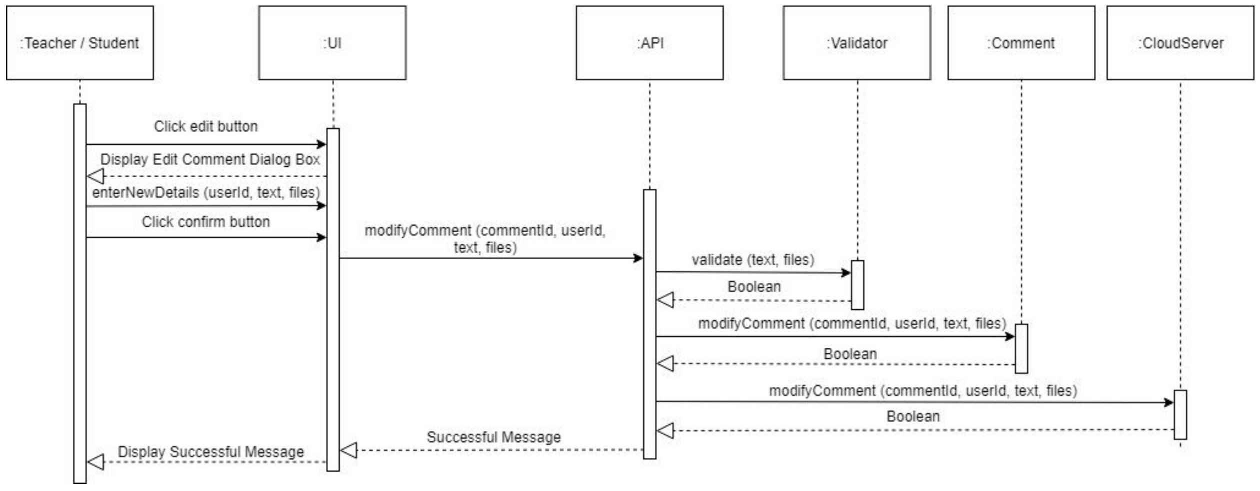
Domain Model:



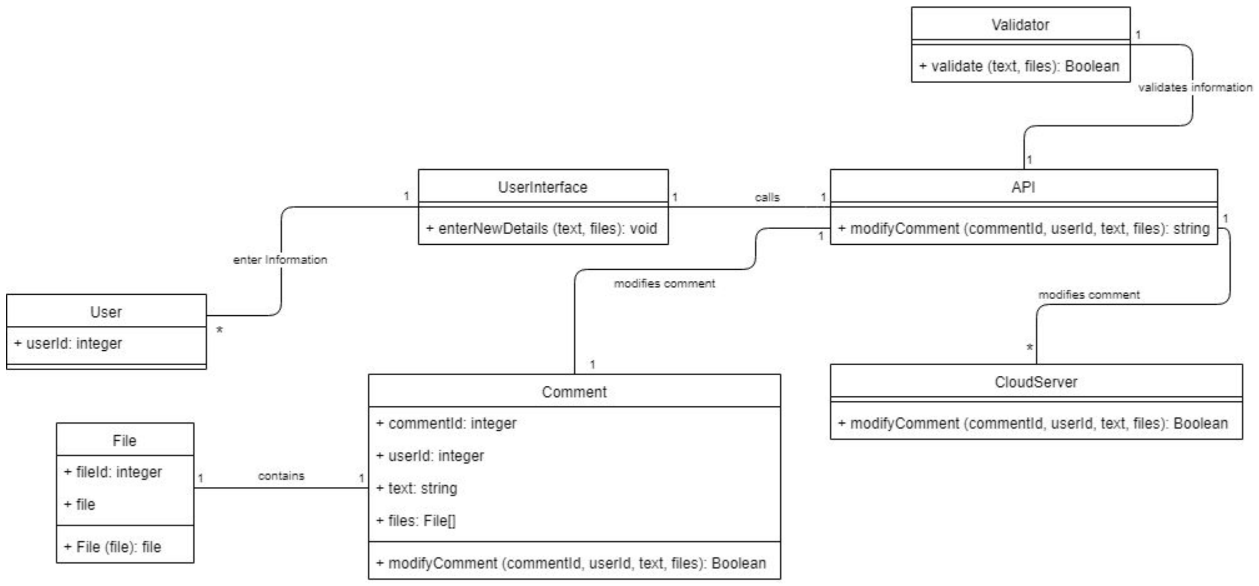
System Sequence Diagram:


:Teacher / Student
:System
Click edit button
Display Edit Comment Dialog Box
enterNewDetails (userId, text, files)
Click confirm button
Display Successful Message

Sequence Diagram:


:Teacher / Student
:UI
:API
:Validator
:Comment
:CloudServer
Click edit button
Display Edit Comment Dialog Box
enterNewDetails (userId, text, files)
Click confirm button
modifyComment (commentId, userId, text, files)
validate (text, files)
Boolean
modifyComment (commentId, userId, text, files)
Boolean
modifyComment (commentId, userId, text, files)
Boolean
Successful Message
Display Successful Message

Class Diagram:



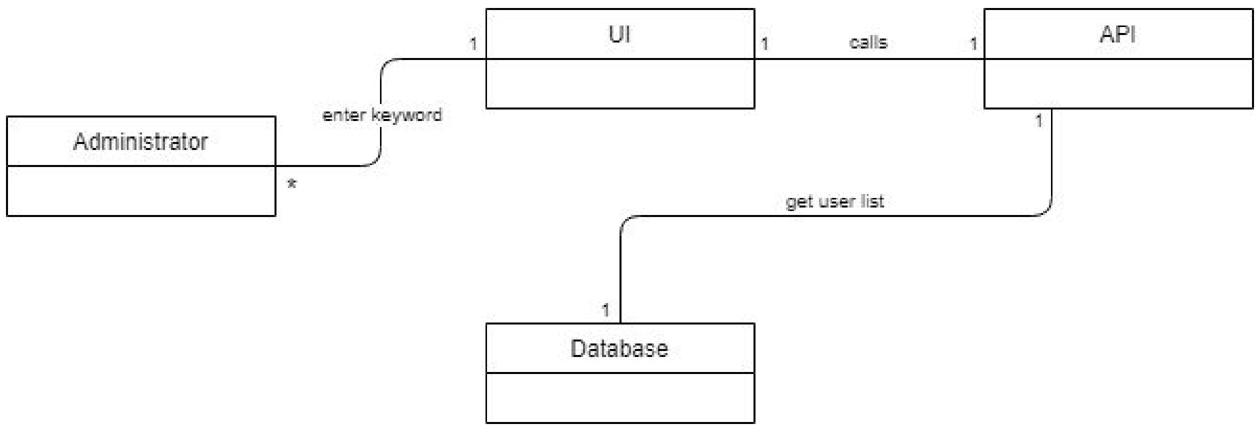
2.2.14 Search User

Administrator/Teacher uses this feature to search a particular user of the LMS.

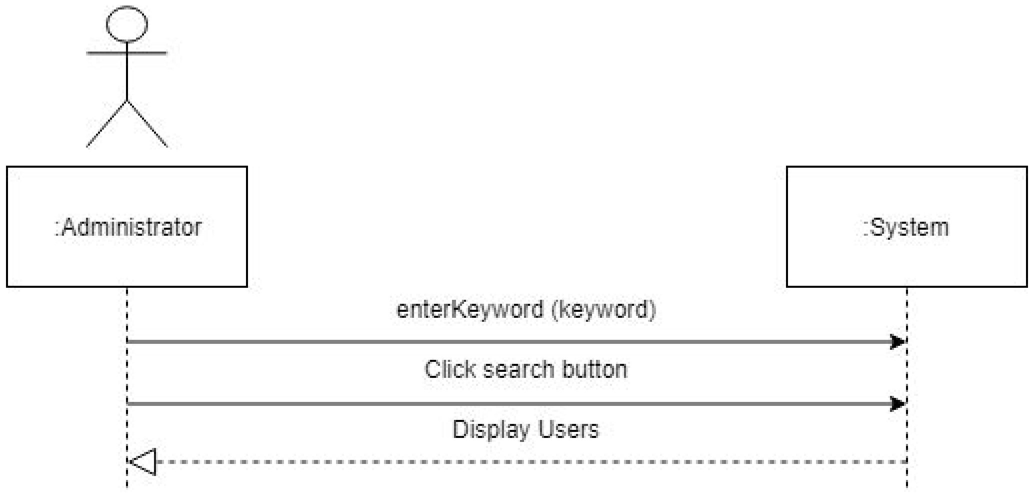
Use Case ID:	UC-1.14		
Use Case Name:	Search User		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Administrator		
Description:	End-user uses this feature to search for specific user in the LMS.		
Trigger:	When the end-user clicks on the search button against the search box		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. If found, the search results will be shown to the End-user.		
Normal Flow:	1. End-user enters the name of a user in the search box. 2. End-user clicks on Search button. 3. End-user is shown the search results.		
Alternative Flows:	1. End-user enters the name of a user in the search box. 2. End-user clicks on Search button. 3. The searched user does not exist in the LMS. 4. End-user is prompted with a message which says “No results found”.		
Exceptions:	Nil		
Includes:	Nil		

Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to display search results within 0.5 seconds after clicking the search button 4. The system shall accept 50 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

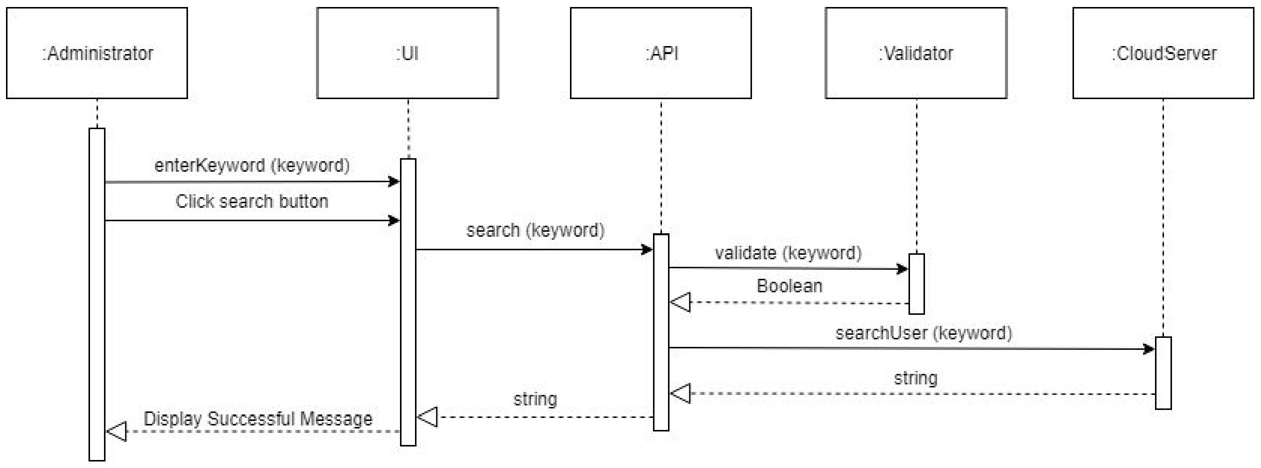
Domain Model:



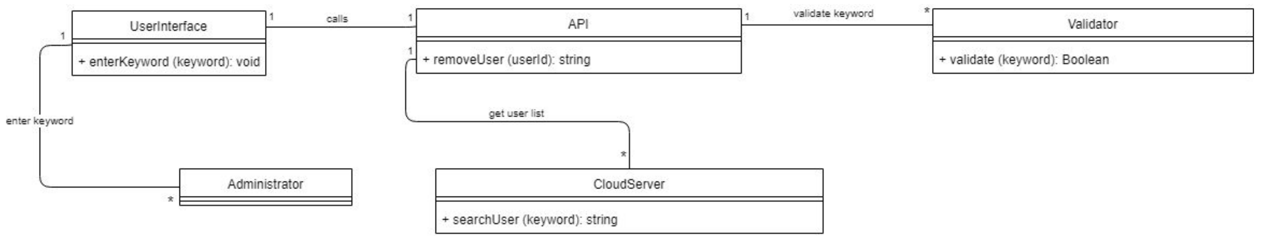
System Sequence Diagram:



Sequence Diagram:



Class Diagram:



UserInterface
+ enterKeyword (keyword): void
calls
API
+ removeUser (userId): string
validate keyword
Validator
+ validate (keyword): Boolean
enter keyword
get user list
Administrator
CloudServer
+ searchUser (keyword): string
1
1
1
1
1
*
*
*

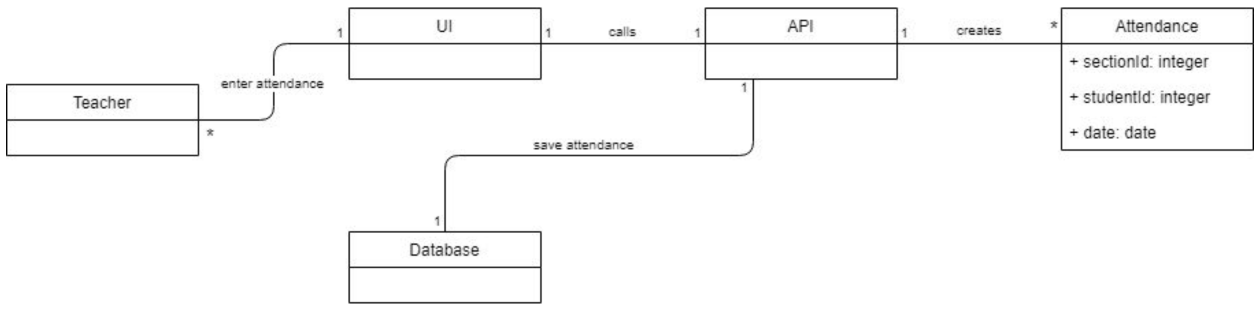
2.2.15 Record Attendance

Teacher uses this feature to record attendance of the students of a particular section.

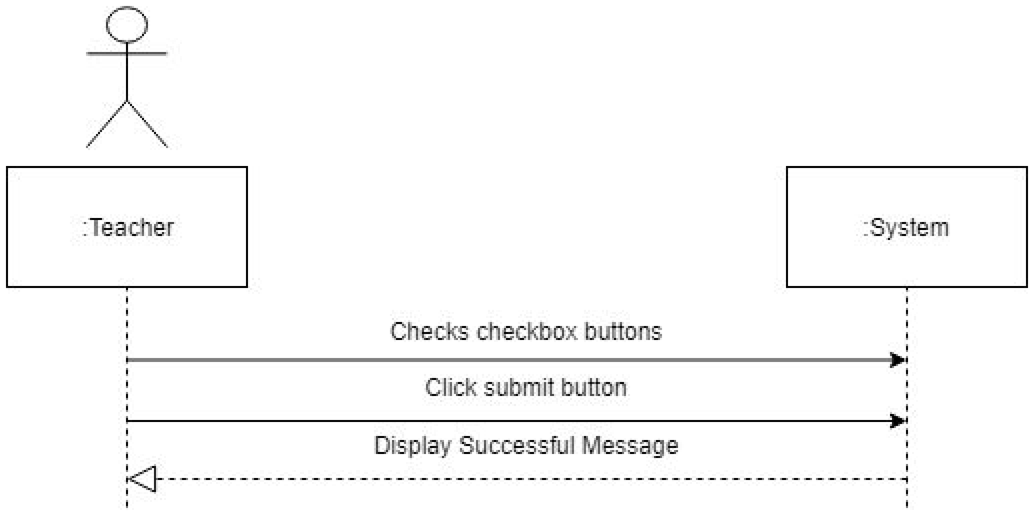
Use Case ID:	UC-1.15		
Use Case Name:	Record Attendance		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Teacher		
Description:	End-user uses this feature to record the Attendance of a particular user for a class.		
Trigger:	When the end-user clicks on the submit button at the end of a particular section attendance page.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. Attendance record for a particular section will be stored on cloud server.		
Normal Flow:	1. End-user checks checkbox buttons besides the name of the students. 2. End-user clicks on submit button.		

	3. End-user is shown a successful message.
Alternative Flows:	Nil
Exceptions:	Nil
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to record attendance within 0.5 seconds after clicking the submit button 4. The system shall accept 25 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

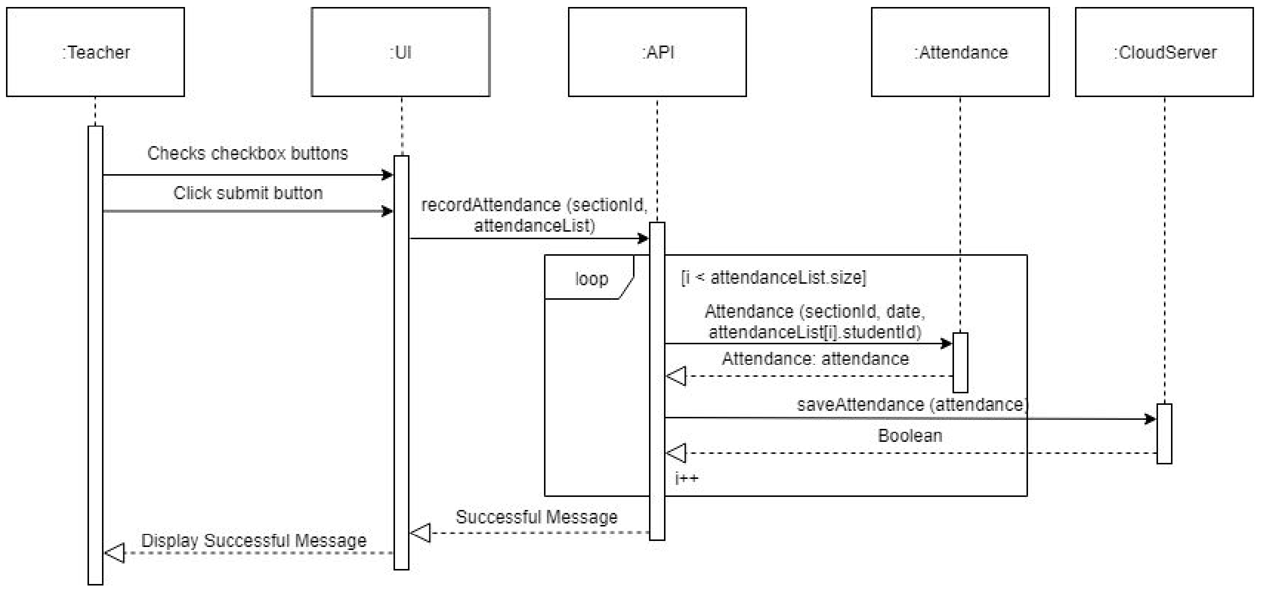
Domain Model:



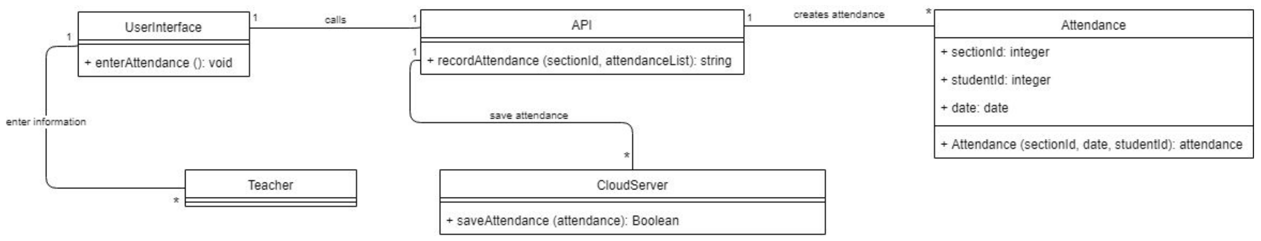
System Sequence Diagram:



Sequence Diagram:


:Teacher
:UI
:API
:Attendance
:CloudServer
Checks checkbox buttons
Click submit button
recordAttendance (sectionId, attendanceList)
loop
[i < attendanceList.size]
Attendance (sectionId, date, attendanceList[i].studentId)
Attendance: attendance
saveAttendance (attendance)
Boolean
i++
Successful Message
Display Successful Message

Class Diagram:


UserInterface
+ enterAttendance (): void
calls
API
+ recordAttendance (sectionId, attendanceList): string
creates attendance
Attendance
+ sectionId: integer
+ studentId: integer
+ date: date
+ Attendance (sectionId, date, studentId): attendance
enter information
Teacher
save attendance
CloudServer
+ saveAttendance (attendance): Boolean
1
1
1
1
1
1
*
*
*

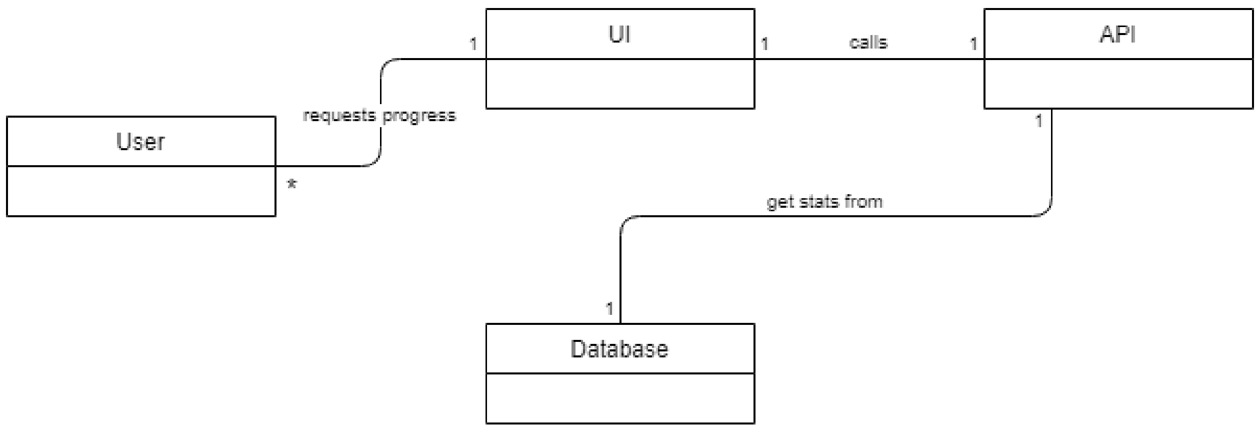
2.2.16 Check Progress

Teacher/Student uses this feature to check the current progress of a student in a particular section.

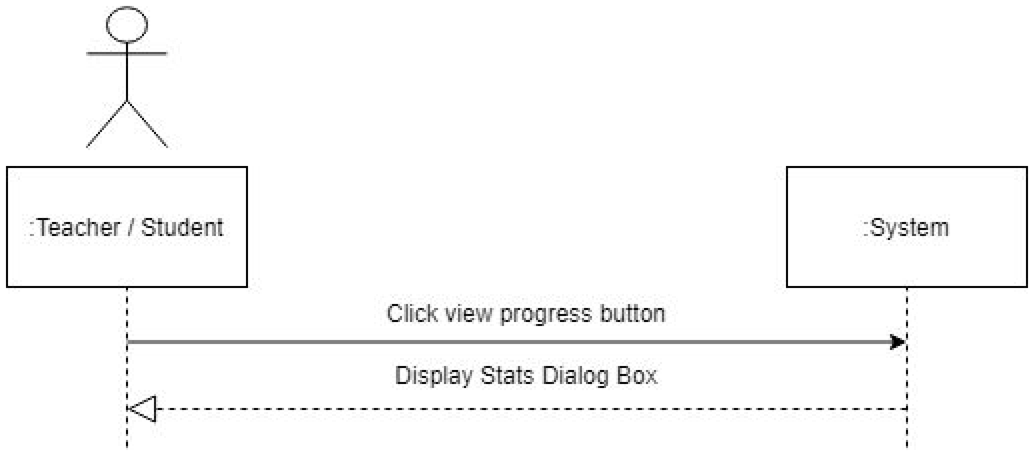
Use Case ID:	UC-1.16		
Use Case Name:	Check Progress		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	
Actors:	Teacher/Student		
Description:	End-user uses this feature to view the marks/grades of a student for a particular section.		
Trigger:	When the end-user clicks on the 'View Progress' button.		
Preconditions:	1. The end-user must be logged in.		
Postconditions:	1. The end-user can view the marks/grades of the student.		

Normal Flow:	1. End-user selects View Progress button besides name of a student. 2. The system shows a the stats of the student on a dialog box.
Alternative Flows:	Nil
Exceptions:	Nil
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to see progress within 0.5 seconds after clicking the view progress button 4. The system shall accept 25 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

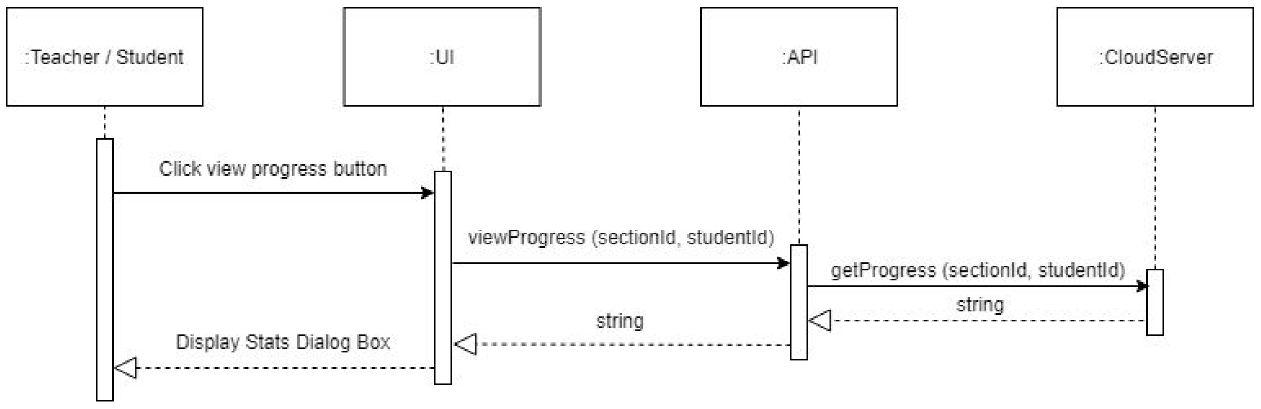
Domain Model:



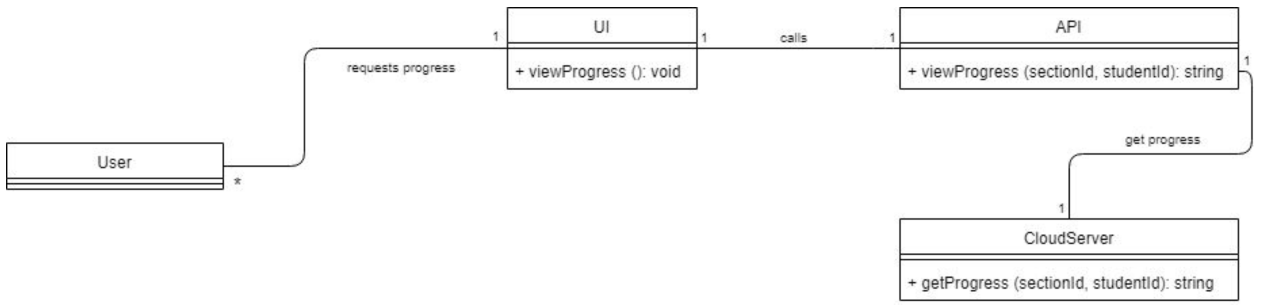
System Sequence Diagram:



Sequence Diagram:


:Teacher / Student
:UI
:API
:CloudServer
Click view progress button
viewProgress (sectionId, studentId)
getProgress (sectionId, studentId)
string
string
Display Stats Dialog Box

Class Diagram:


UI
+ viewProgress (): void
API
+ viewProgress (sectionId, studentId): string
User
CloudServer
+ getProgress (sectionId, studentId): string
requests progress
calls
get progress
1
1
1
1
1
1
*

2.2.17 Manage Section

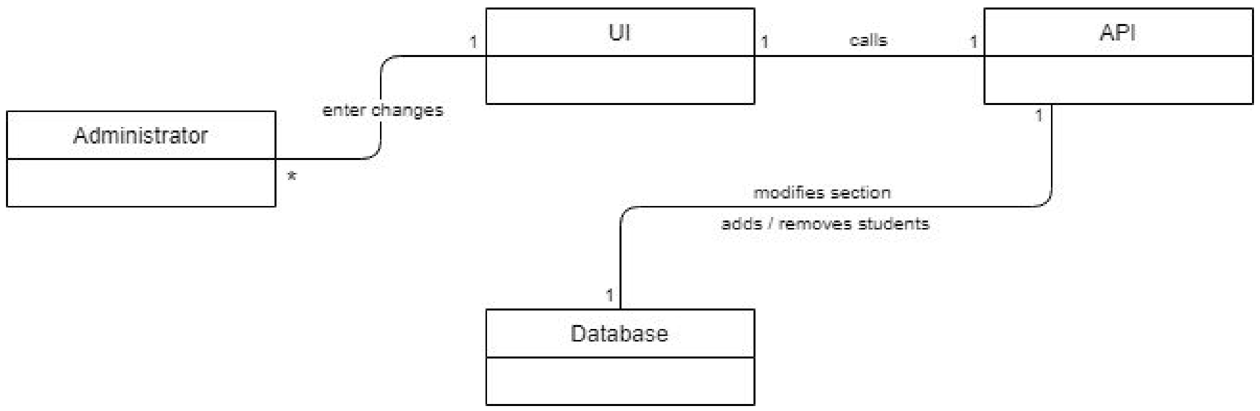
Adding/removing students from a particular section.

Use Case ID:	UC-1.17		
Use Case Name:	Manage Section		
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	

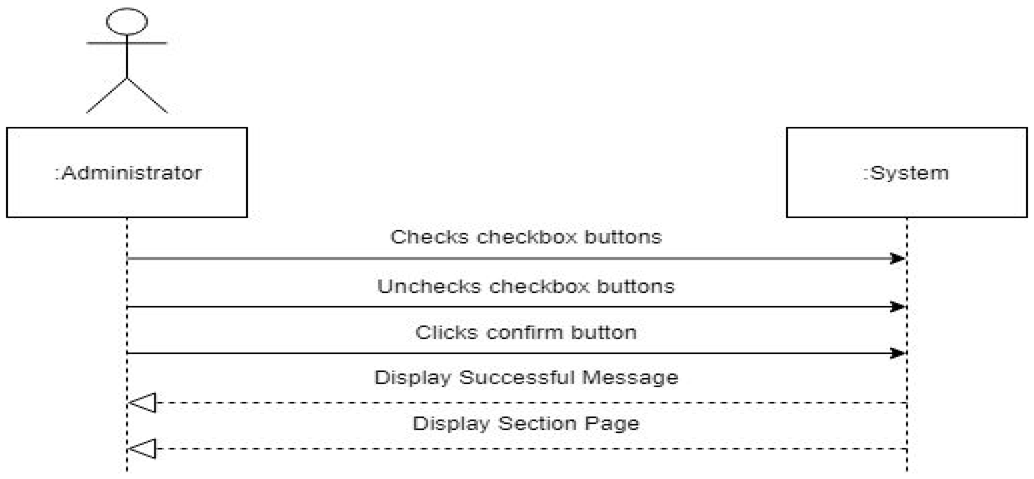
Actors:	Administrator
Description:	End-user uses this feature to add/remove students from a section.
Trigger:	When the end-user clicks on the manage section button besides a section name in the section list on sections page.
Preconditions:	1. The end-user must be logged in.
Postconditions:	1. All the changes made are synchronized with the cloud server.
Normal Flow:	1. End-user checks checkbox buttons besides name of students in order to add students under the heading Add Students.

	2. End-user unchecks checkbox buttons besides name of a students in order remove the student currently in the section under the heading Remove Students. 3. End-user clicks a 'Confirm' button.
Alternative Flows:	Nil
Exceptions:	Nil
Includes:	Nil
Frequency of Use:	500 per day
Special Requirements:	1. The mean time to failure should be at least one year 2. This use case shall be available 24 hours/day 3. The end-user shall be able to update end-user within 0.5 seconds after clicking the confirm button 4. The system shall accept 25 requests per second
Assumptions:	Nil
Notes and Issues:	Nil

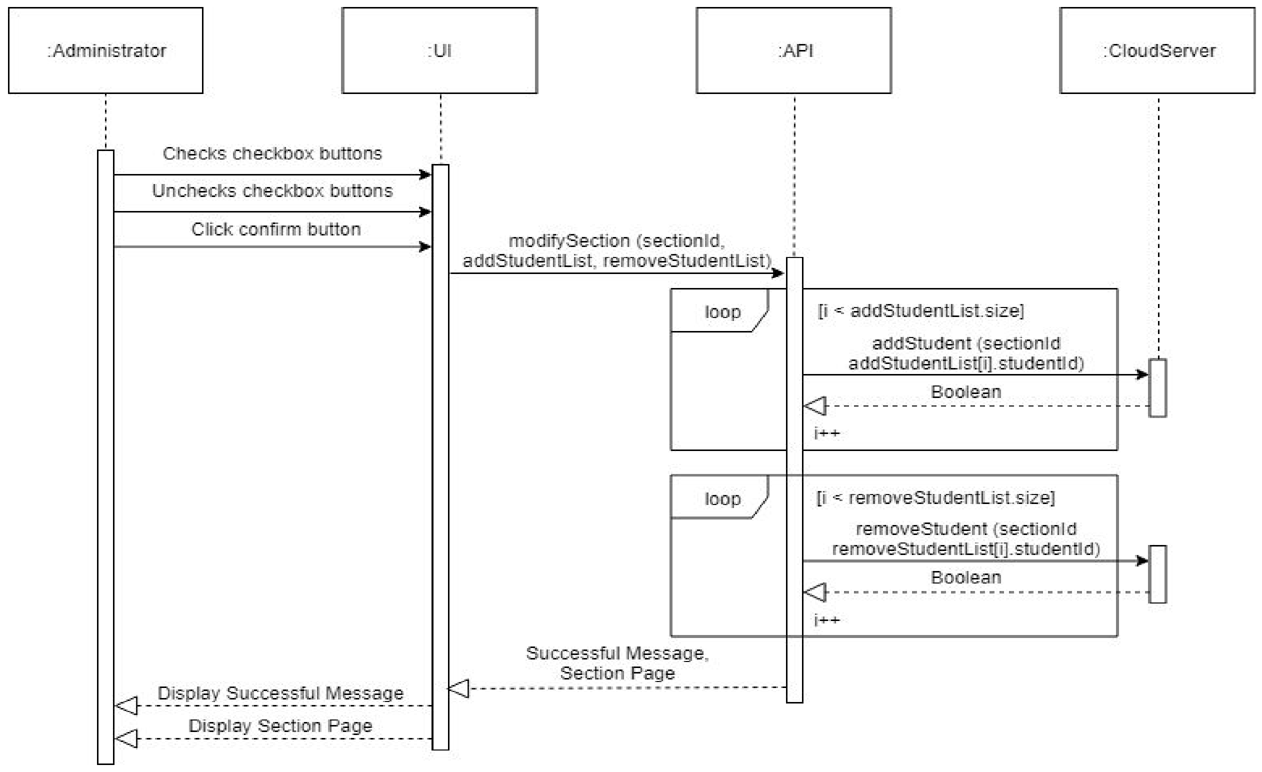
Domain Model:



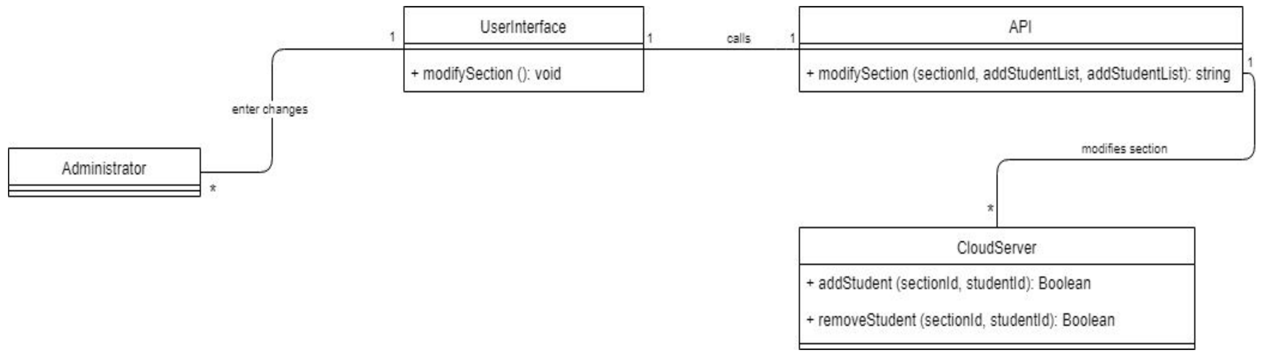
System Sequence Diagram:



Sequence Diagram:

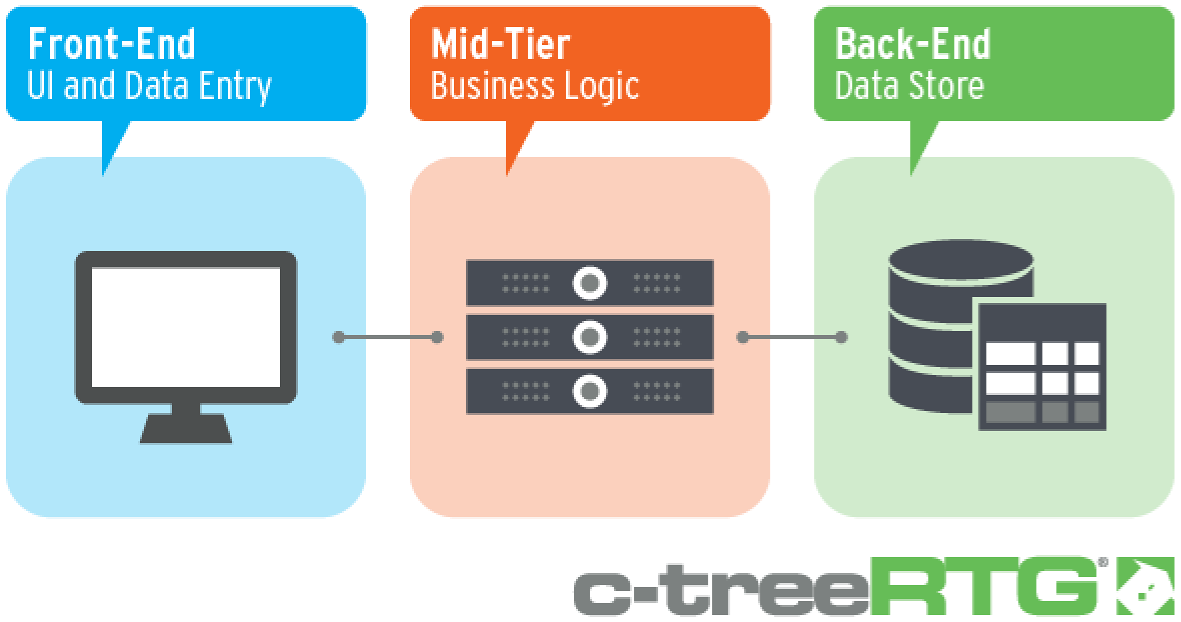

:Administrator
:UI
:API
:CloudServer
Checks checkbox buttons
Unchecks checkbox buttons
Click confirm button
modifySection (sectionId, addStudentList, removeStudentList)
loop
[i < addStudentList.size]
addStudent (sectionId addStudentList[i].studentId)
Boolean
i++
loop
[i < removeStudentList.size]
removeStudent (sectionId removeStudentList[i].studentId)
Boolean
i++
Successful Message, Section Page
Display Successful Message
Display Section Page

Class Diagram:


UserInterface
+ modifySection (): void
API
+ modifySection (sectionId, addStudentList, addStudentList): string
Administrator
CloudServer
+ addStudent (sectionId, studentId): Boolean
+ removeStudent (sectionId, studentId): Boolean
enter changes
calls
modifies section
1
1
1
1
*
*

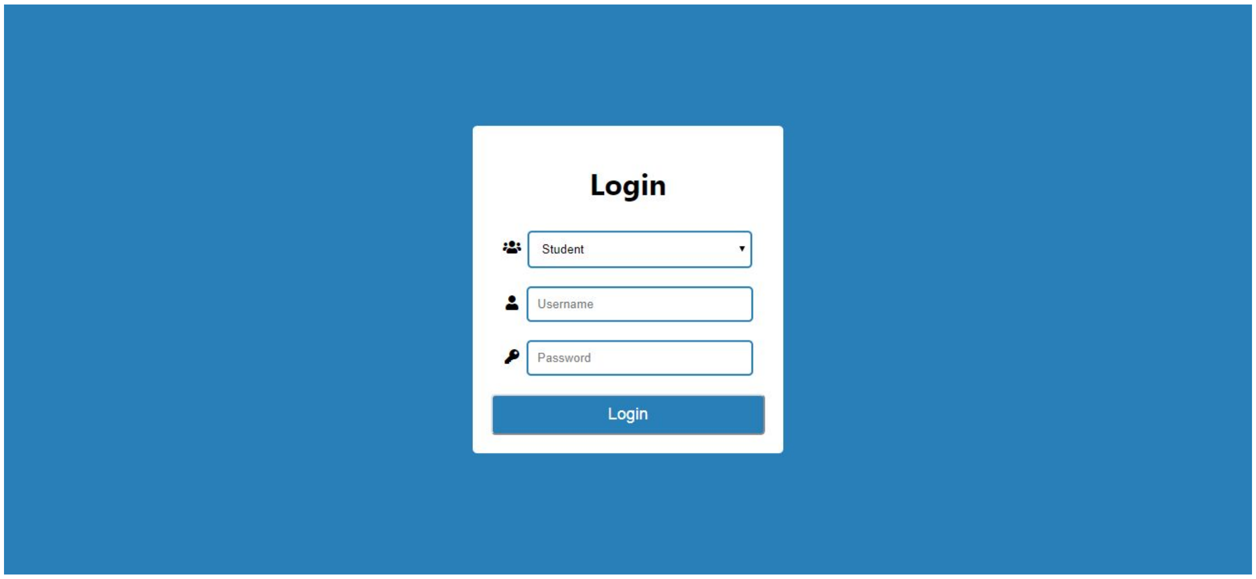
3. Software Architecture:

Our project is using 3-Tier Architecture with layers namely the UI Layer, Domain Layer and Database Layer.

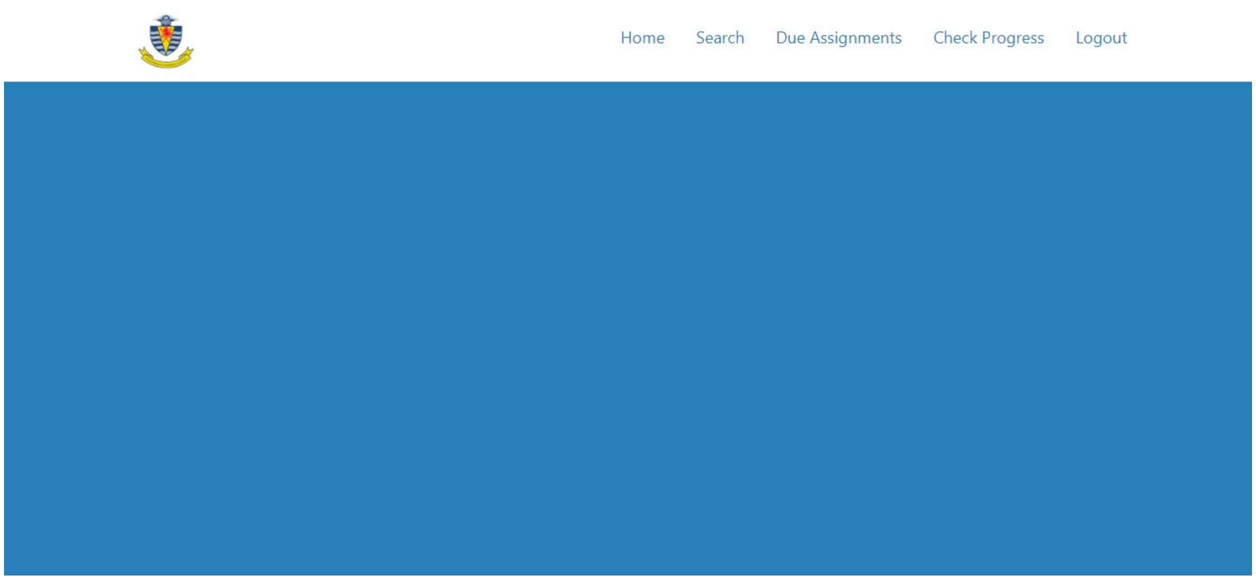


4.User Interface diagrams:

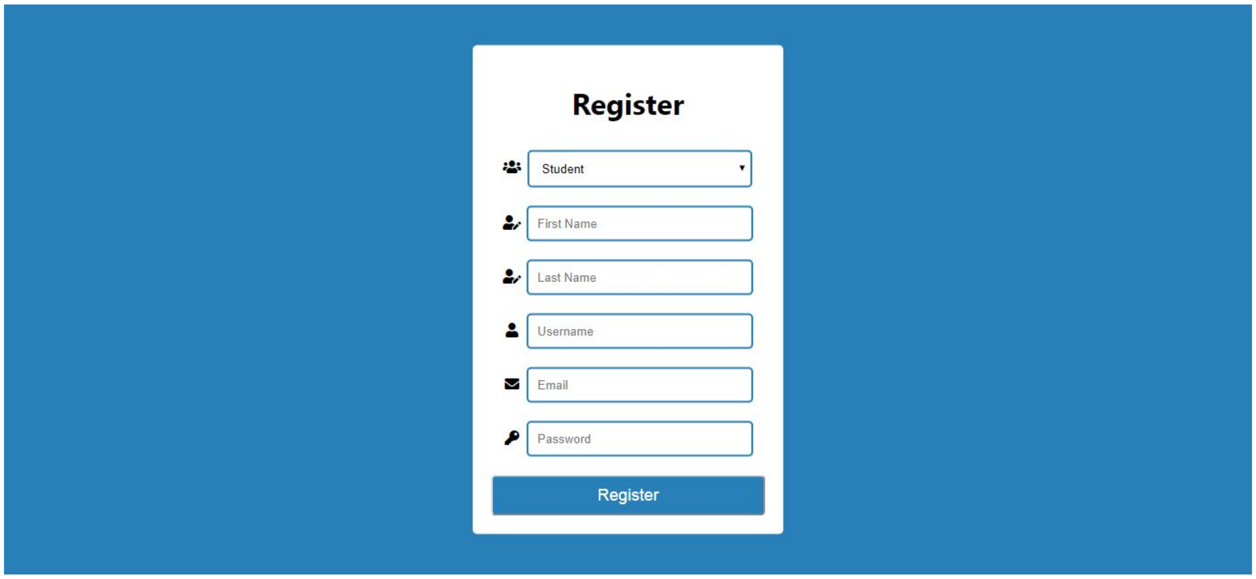
4.1 Login



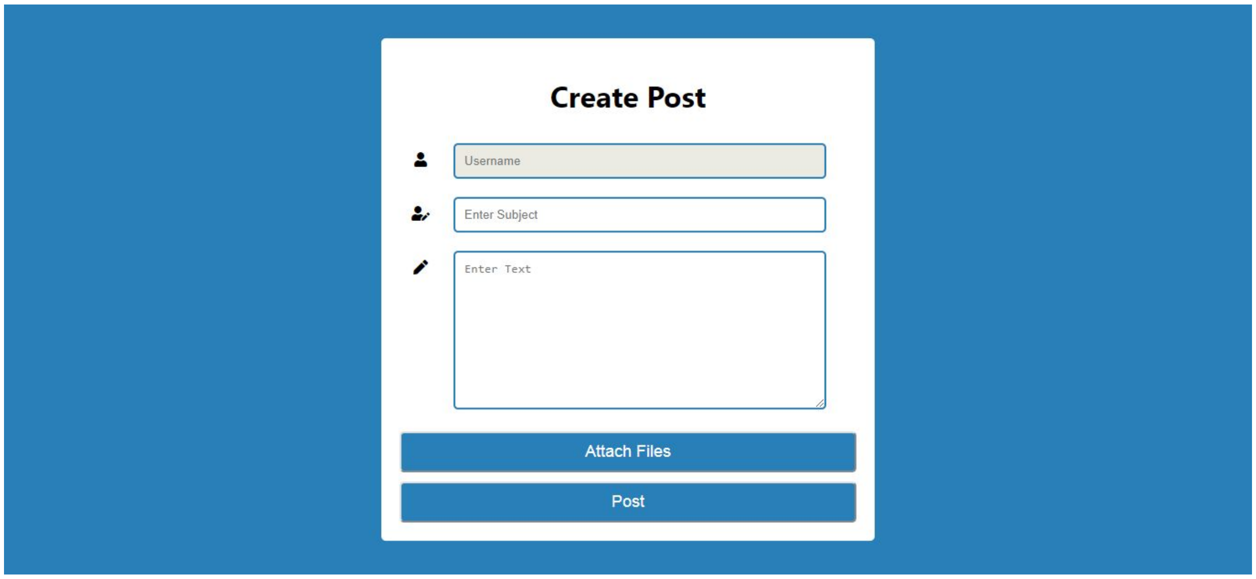
4.2 Logout



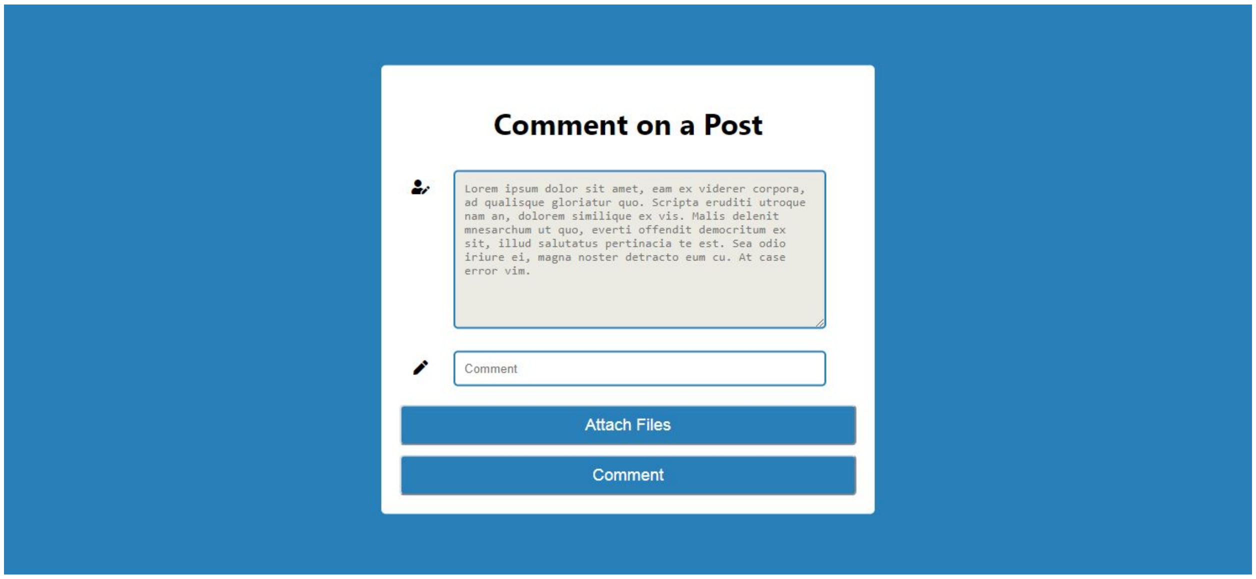
4.3 Register User



4.4 Create Post

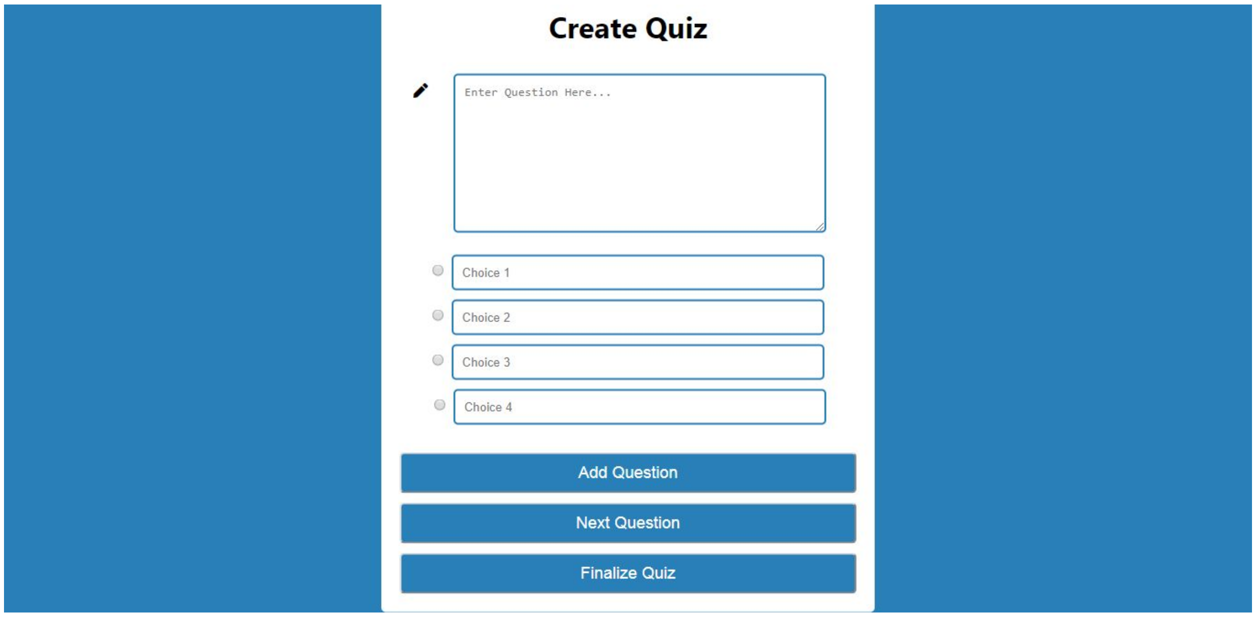


4.5 Comment

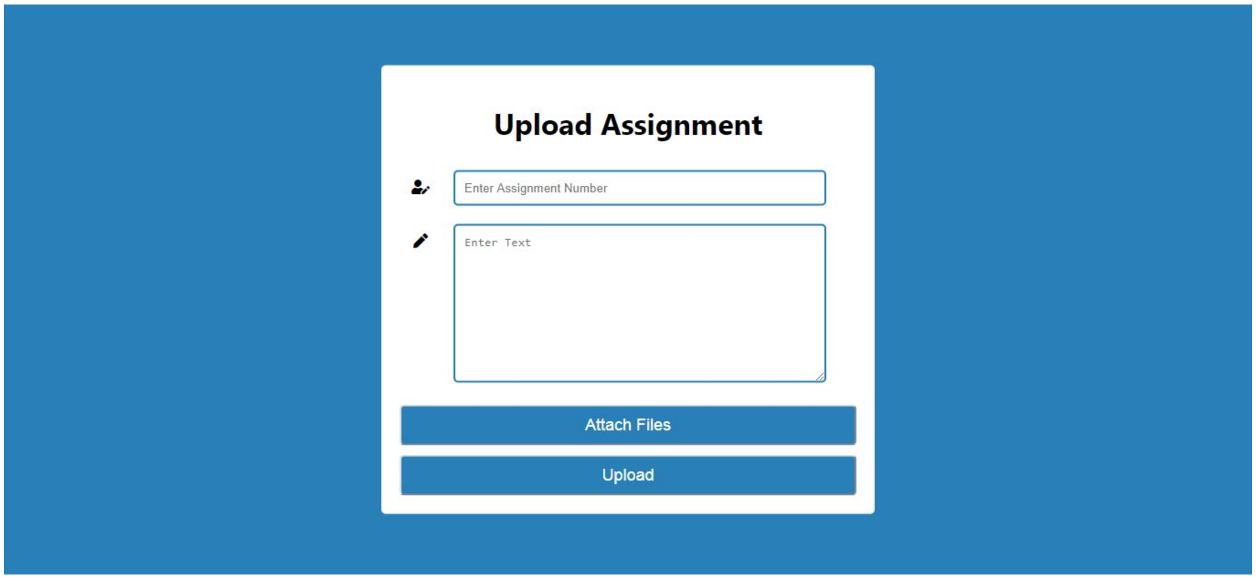

Comment on a Post

Lorem ipsum dolor sit amet, eam ex viderer corpora, ad qualisque gloriatur quo. Scripta eruditi utroque nam an, dolorem similique ex vis. Malis delenit mnesarchum ut quo, everti offendit democritum ex sit, illud salutatus pertinacia te est. Sea odio iriure ei, magna noster detracto eum cu. At case error vim.

4.6 Create Quiz


Create Quiz

4.7 Upload Assignment



5. Conclusion:

All things considered, the Learning Management System provides very reliable and quick services required for efficiently managing all the necessities of an educational system. It provides a platform for both teachers and students to interact with each other. LMS eliminates the need for a time consuming manual system where the instructor has to print out course materials for all the students and hand out grades to students individually. The advantages of using LMS by far outweigh its drawbacks.