## Table of Contents

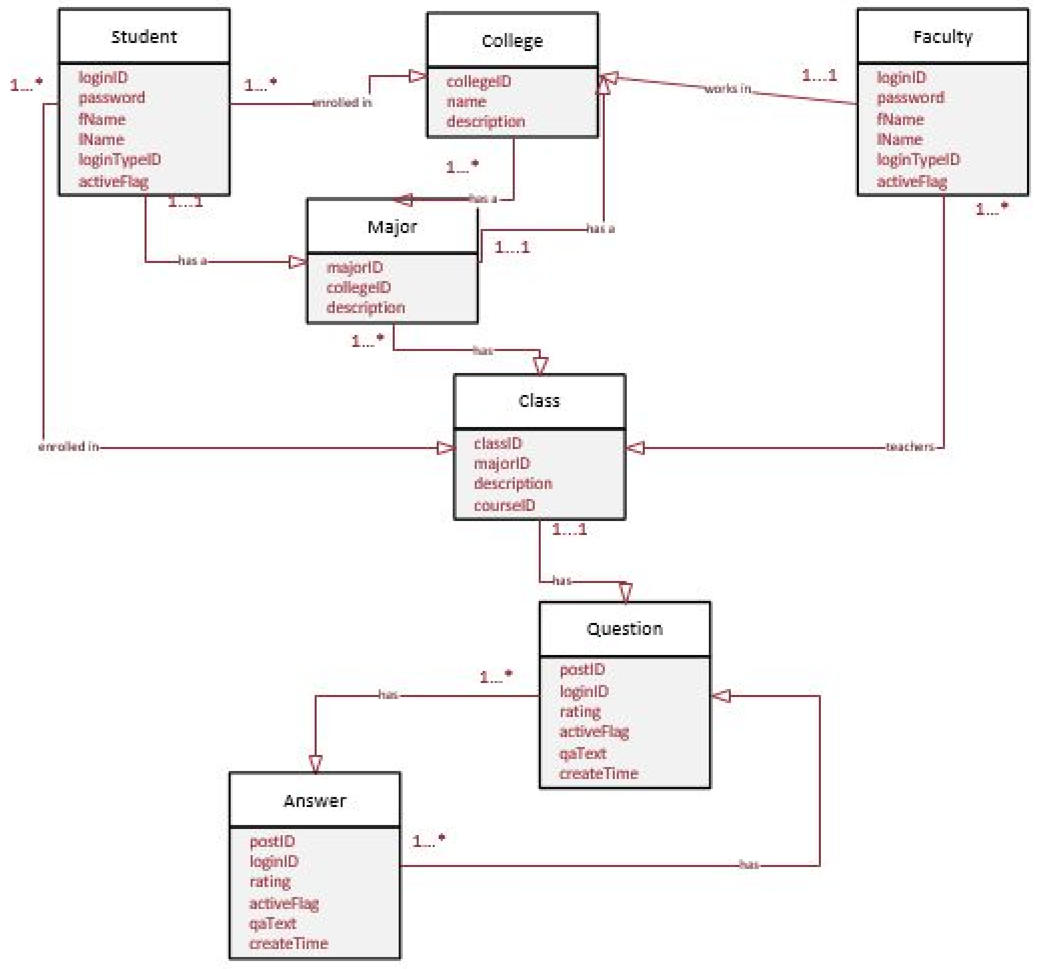
## Design Discussion

For the first iteration of the app development we decided to implement posting a question, posting an answer, removing a question and removing an answer. For this iteration there will be no login capabilities. You will simply be able to go to the site and be able to post questions and answers with the ability to edit or destroy the questions. The site hasn't been broken down into the sections based on college, major, or class that we envision the site having yet. Those will be done in later iterations. We have the framework for those things already built into the app, but they are just frameworks.
The classes that were created for this iteration all tie back to the four use cases that we implemented with some basic setup classes. The Logins and Login_types store the logins of users and the different types of logins. All the use cases require that a user is logged in. Q_and_as stores questions and answers that are posted. These connect to the post a question and post an answer use cases. They also connect to a remove a question and answer use cases as well. Courses stores all courses in the systems. Colleges stores all the colleges in the university. Majors stores all majors from all of the colleges. Enrolled_classes stores the classes that the students are enrolled in. Below are the diagrams for each class.

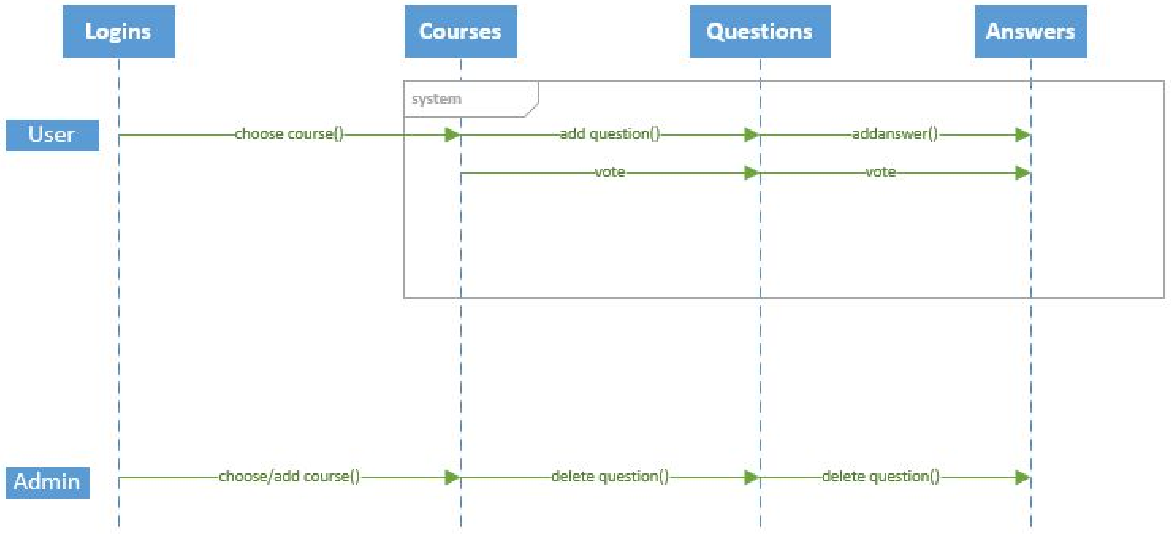
For the second iteration, we decided to implement attaching item and vote on an answer, as well as implementing two new use cases; add courses and delete courses. This iteration also implements login capabilities, where Students and Faculty users are able to access forums and post questions and answers, and Admin users are able to delete questions and answers as well as add and delete courses.
There were a few diagrams that had to be changed for this iteration because we had noticed that the way the diagrams were drawn they needed minor tweaks. We added diagrams for the new use cases along with the system sequence diagrams. We had to update the Use Case Model to fit the new use cases as well.

For the final iteration, we decided to just spruce up the existing site. We decided to add the ranking and the voting functionality. We planned on updating the homepage to show the "most recent" questions along with a course list. Again there were a few diagrams that needed to adjusted because the actual design did not look like the design that we had originally come up with.

## Design Class Diagram



## Design level Sequence Diagrams



## Design Decisions

We built upon the things that we had established for the first iteration for the design of this iteration. We made the look of the app look more like an app than a horrible 90s website. It has a more fluid design.
GRASP Patterns Used:

- Creator - having methods create other things used in the app.
- Low Coupling - things in the app aren't changed by changes in other components (i.e. the layout doesn't affect how the links work).
- High Cohesion - different methods collaborate with other parts to get the work done.
- Controller - goes to the system to get things accomplished.

GOF Patterns Used:

- Bridge - bridges exist between questions/answers/courses
- Iterator - iterates over questions/courses/user lists
- Strategy - each class has its own algorithm to do certain things.

## Database Design

The database is broken down into similar tables as the classes that we have : User, Courses, Questions, Answers. The first is the class and the second is its equivalent in the database. Rails allows for a scaffold to build the database for you, so we used that scaffold. Rails saves the

question and answers as title:string, body:text. The foreign keys were generated as author:string, body:text, question:reference. By using this model it doesn't generate all files associated, only the model/class file. The scaffold generates all files.
The following are the commands that were used:

- Rake db:migrate

To add columns

- Rails generate add_author_to_questions author:string
- Rake db:migrate