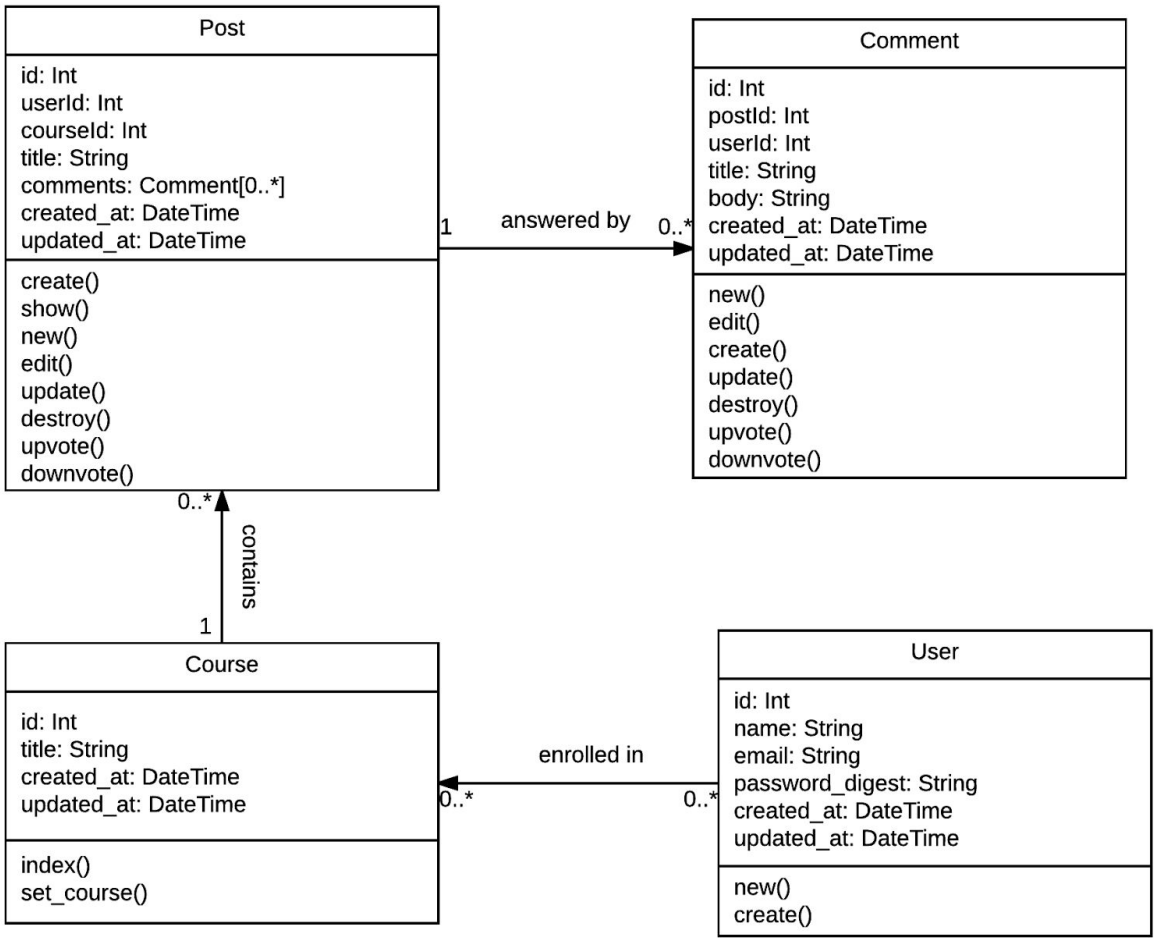
Table of Contents

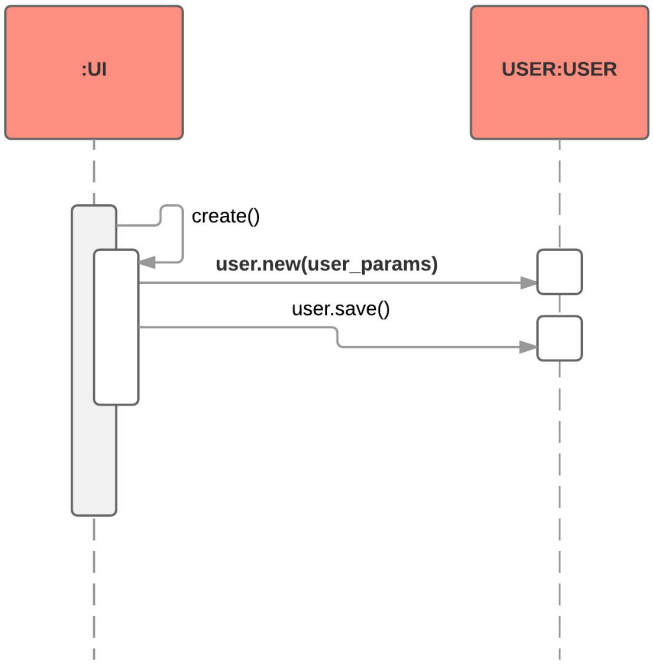
I. Design Class Diagram With All Classes & Associations

Design Class Diagram With All Classes & Associations

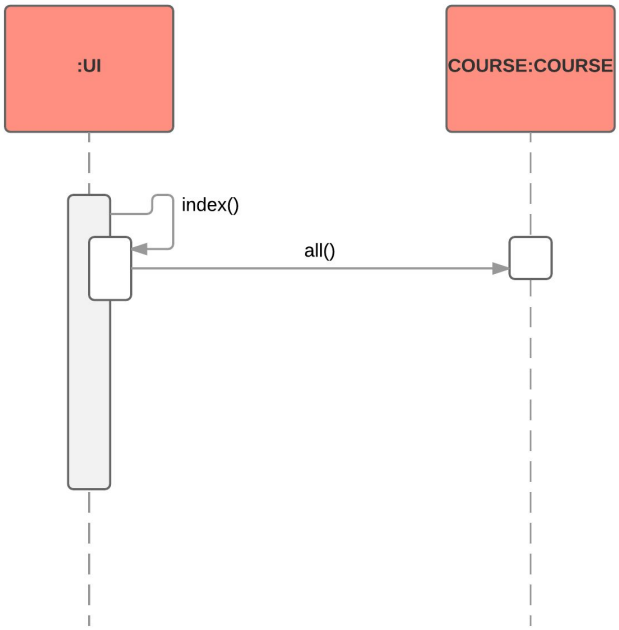


II. Design Level Sequence Diagrams

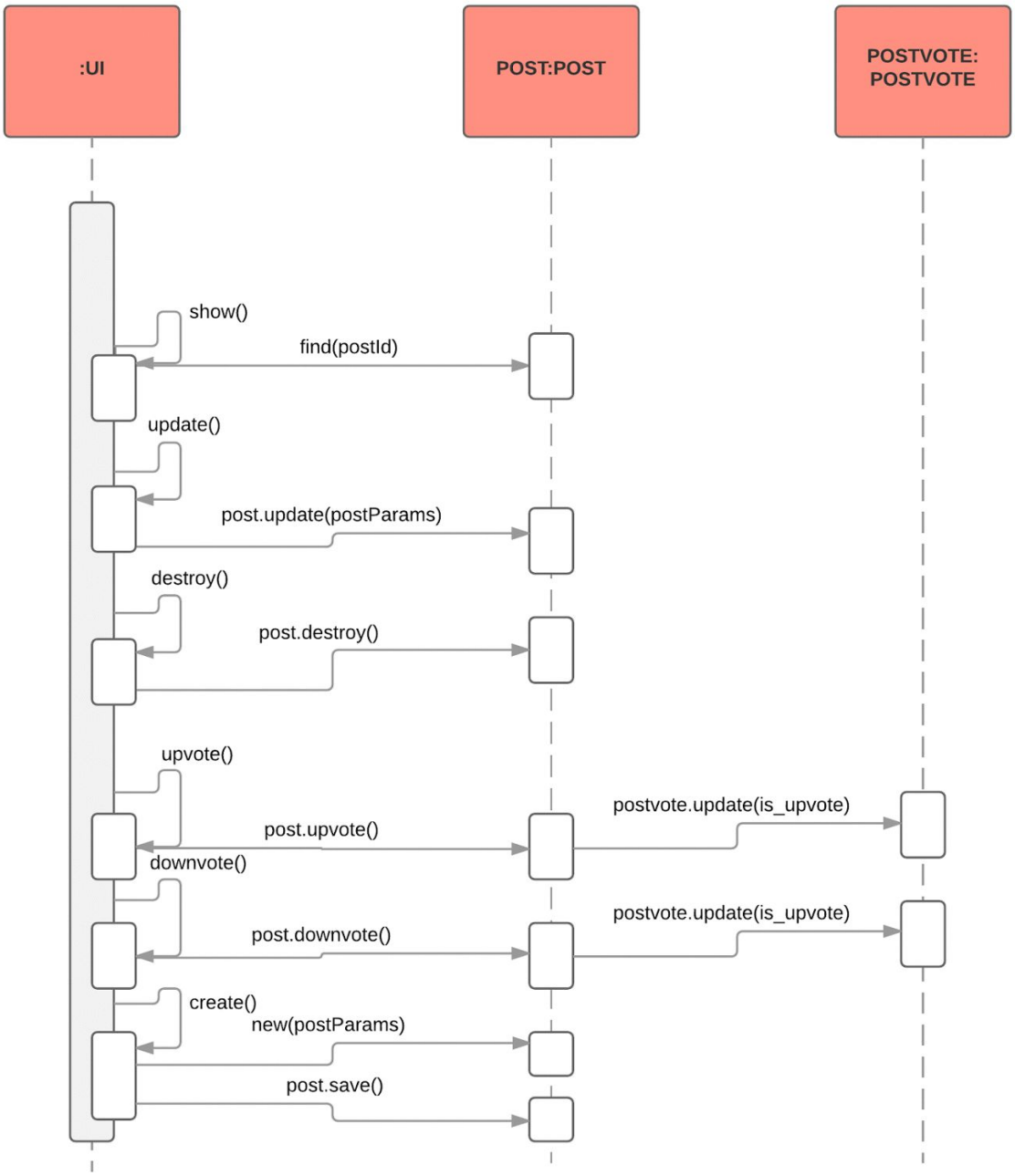
User Design Sequence Diagram:



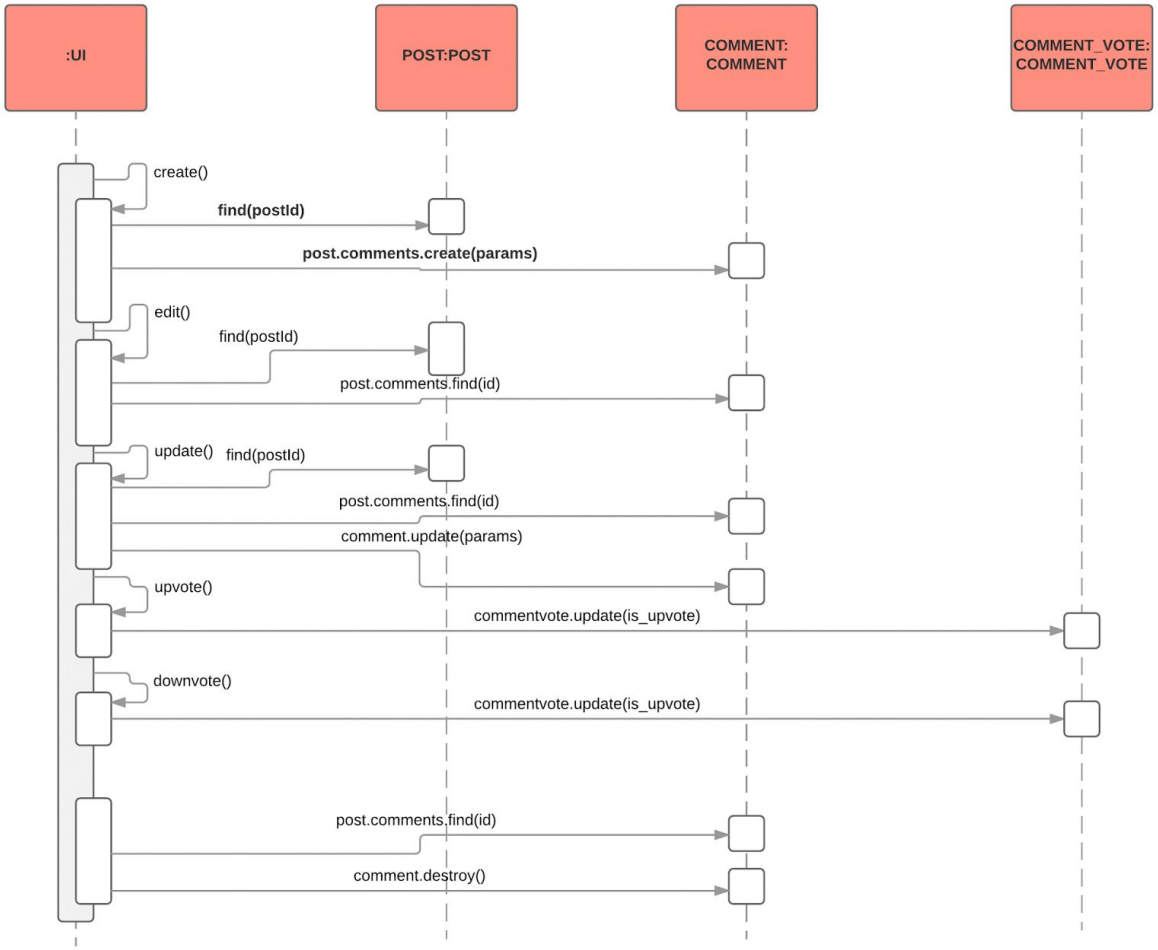
Course Design Sequence Diagram:



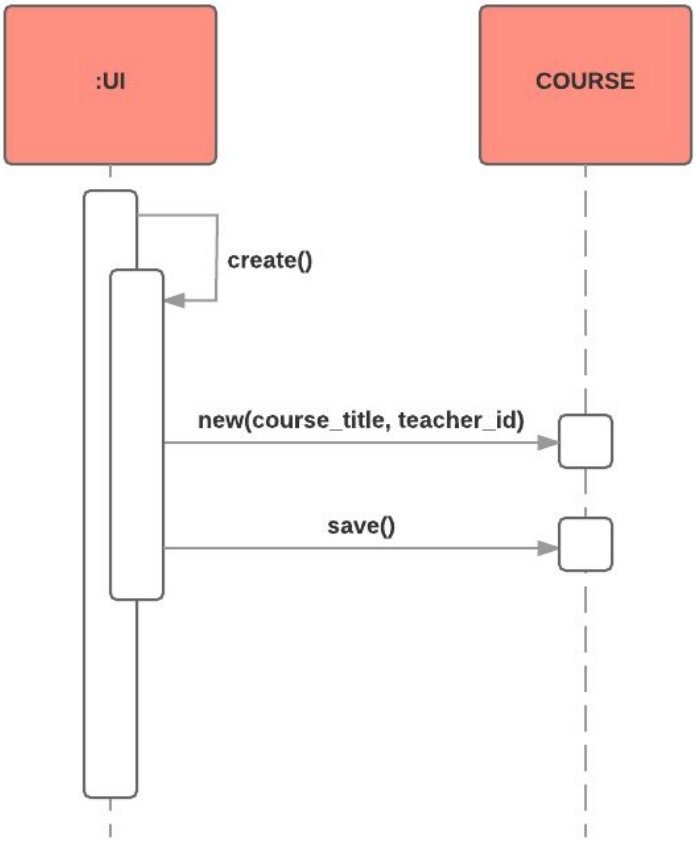
Question Design Sequence Diagram:



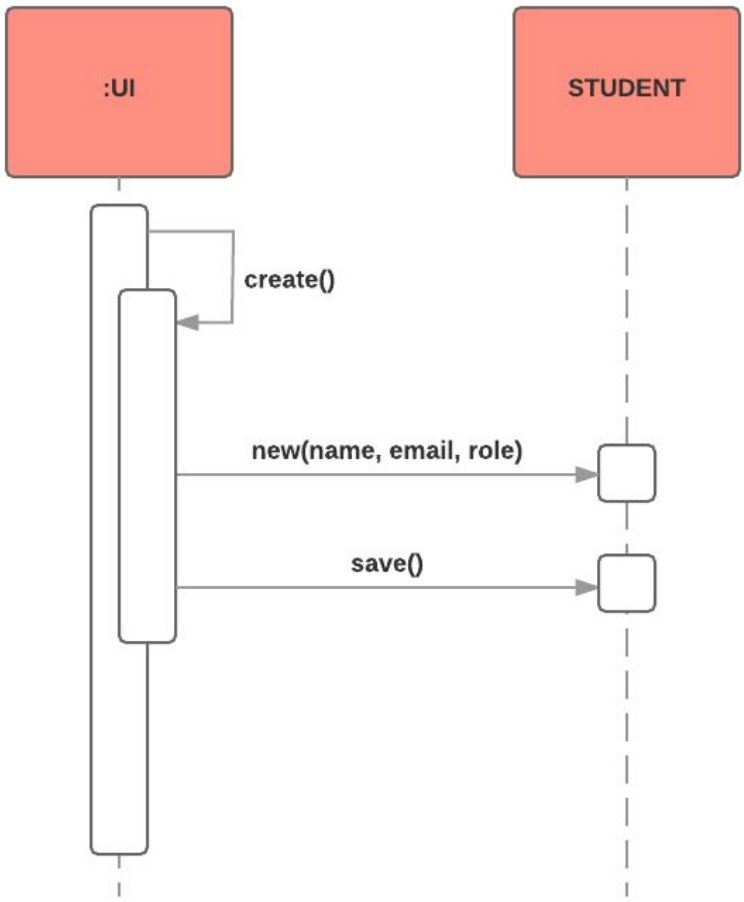
Answer Design Sequence Diagram:



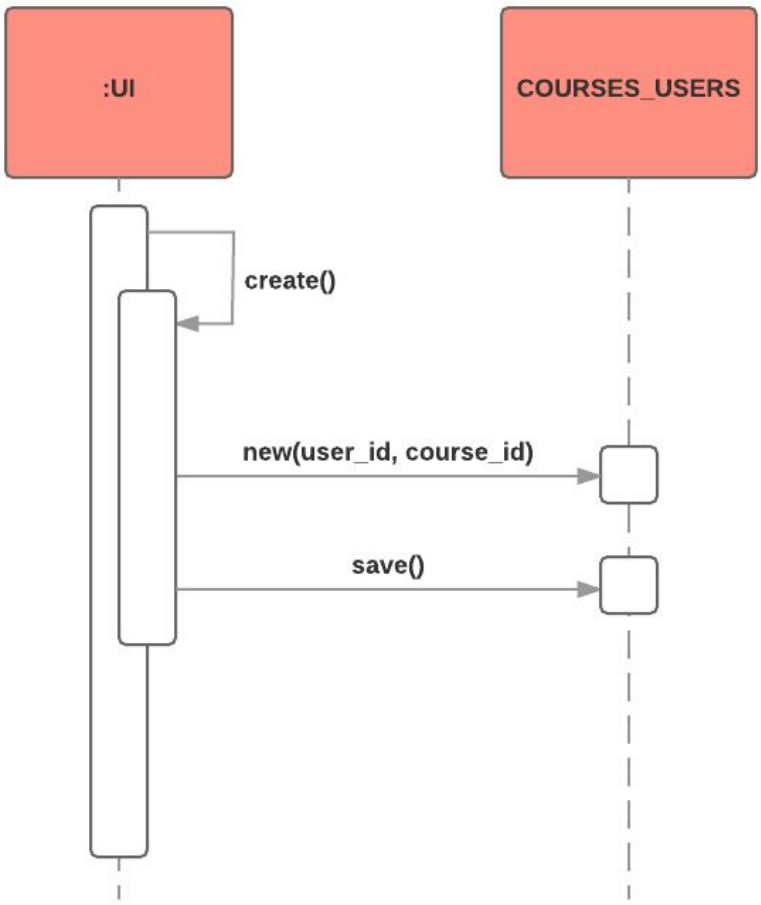
Add Course Design Sequence Diagram



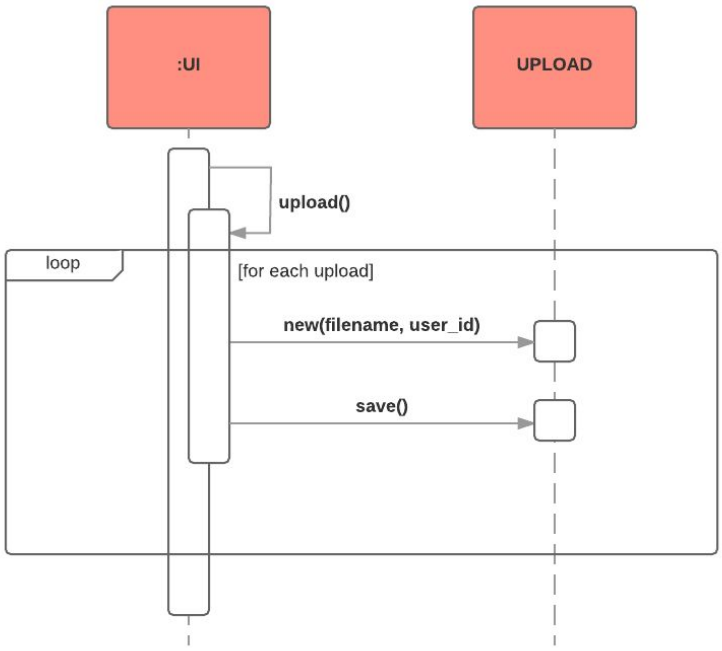
Add User Design Sequence Diagram



Add Student to Course Design Sequence Diagram



Upload Class Design Sequence Diagram



III. Discussion of Design Decisions

When designing our application for iteration 1, we wanted to implement the iteration as simple as possible. With this in mind, we took full advantage of Rails' scaffolding capabilities in order to implement the business logic and flow of our application. However, we did not want to finish iteration 1 with a bare-bones implementation of the use cases we designated for this iteration. So, using the views and designs provided to us from Rails' scaffolding, we wanted to use those as the starting point to begin working out a full-fledged design of our application. We decided to take this direction so that we could provide a demo at the end of this iteration that is more representative of the vision we have for our application and the future directions we intend to take in future development.

When designing our application for iteration 2, we had to begin by rethinking the way visitors get to posts. Instead of having all the posts available to anyone we created users and courses to limit what visitors can view and create. Doing this allowed us to force visitors to log on. We also decided to let users view, but not post/comment on, courses that they are not enrolled in with the "preview courses" feature. Finally, we wanted people to be able to vote on posts/comments in a way such that they could take back their votes later. So we allowed for NKUNet to remember what users voted on.

When designing our application for the final iteration, we had to begin by expanding upon users to create roles. We used these different roles to give different privileges to different users (Student, Teacher, Admin, Registrar). We also decided to add more functionality to the different roles. The student and teacher pages were revamped to add functionality for reputation, frequently answered questions, and uploading documents. The Registrar was given the ability to add users, courses, and add users to courses. To make this process easier, we allowed the ability to populate the site via a CSV file.

IV. Discussion of Database Design

The database for iteration 1, did not need to be complex. In whole, the database consists of two tables (*Posts* and *Comments*). For the time being, each table has a small number of data variables. For *Posts*, each record in this table will have an *ID*, *author*, *title*, and *body* (in addition to the auto-generated *created_at* and *updated_at* data). For *Comments*, each record in this table will have an *ID*, *post_ID*, *author*, and *body* (in addition to the auto-generated *created_at* and *updated_at* data).

The relationship between these two tables is simple. A *post* can have many comments, and a *comment* must belong to a single *post*.

In Iteration 2 we had to make few changes to our database. We added a Users table so that we could have users log in and out of the site, but also so we could limit which pages/permissions users are able to have. We also added a *PostVotes* and *CommentVotes* table to keep track of all the votes that a comment or post has. For *Users,* each record in this table will have an *ID, name, email,* and *password_digest.* In addition, the table has the auto-generated fields updated_at and created_at. All tables discussed will have these two extra fields. For *PostVotes*, each record in this table will have an *ID, post_id, user_id,* and *is_upvote* field. The table for *CommentVotes* is similar to the *post_votes* table. Each record in this table will have an *ID, comment_id, user_id,* and *is_upvote* field. Another table that we added to our database is the *Courses* table. The *Courses* table contains the *ID*, and *title* fields. The table *courses_users* acts as a junction table between *Course* and *User* and it only has two fields (*user_id* and *course_id*).

In iteration 3 we only had to make minor adjustments to accommodate the newest changes. We created an Uploads table to keep track of which documents were uploaded by which users. We added a directory where the file uploads are to be stored. The User table was modified to accommodate the roles student, teacher, admin, and registrar. Finally, we added the user_id to the courses table to set teachers to courses.

In our design courses are meant to be the containers for all questions. This is because when a user posts a question they are required to post it in a course. Also when a user wants to browse questions, they have to browse a subset of questions of a time in the form of a course. Below are listed some of the relationships between our tables:

- A course can have many questions but a question must have only one course.
- A user can belong to 0 or more courses and a course can belong to 0 or more users.
- An answer can have only one question but a question can have 0 or more answers
- A question can have many votes but a vote can only belong to one question.
- An answer can have many votes but a vote can only belong to one answer.

Rails Commands For Iteration 1

```
$ bin/rails g scaffold Post title:string  body:text author:string
$ bin/rails g model Comment author:string body:text post:references
$ bin/rake db:migrate
```

Rails Commands For Iteration 2

```
$ bin/rails g scaffold Course title:string
$ bin/rails g model User name:string email:text password_digest:string
$ bin/rails g migration CreateJoinTableUserCourses user course
$ bin/rails g model PostVotes post:references user:references
is_upvote:boolean
$ bin/rails g model CommentVotes comment:references user:references
is_upvote:boolean
$ bin/rake db:migrate
```

Rails Commands For Iteration 3

```
$bin/rails g model Uploads filename:text user:references

$bin/rails generate migration add_role_to_users role:integer
```