

# Design Document

The Borne Collective: NKUNet

[Change Discussion](#)

[Iteration 2](#)

[Class Diagram](#)

[Sequence Diagrams](#)

[Design](#)

[Classes](#)

[Controllers](#)

[Database](#)

[Tables](#)

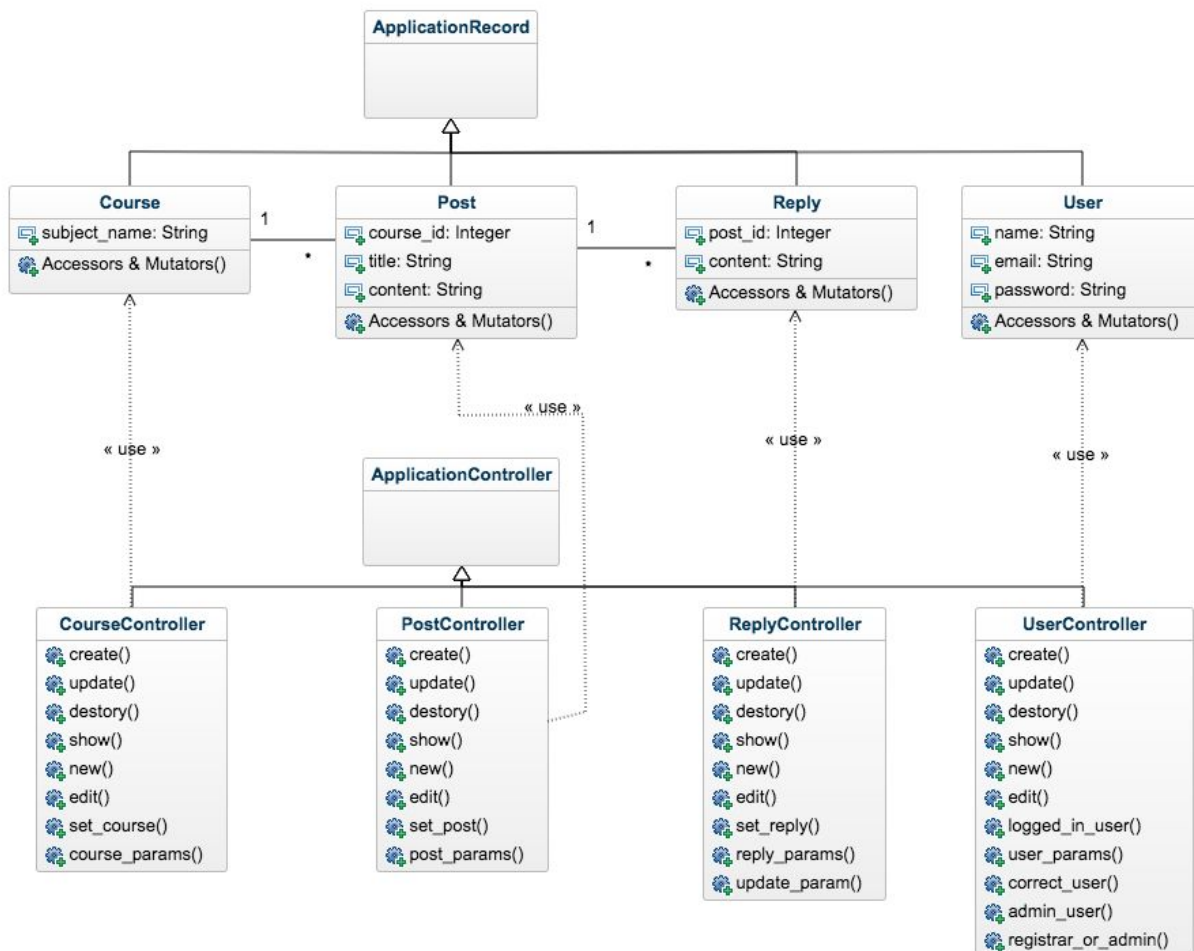
[Rails Console Commands](#)

# Change Discussion

## Iteration 2

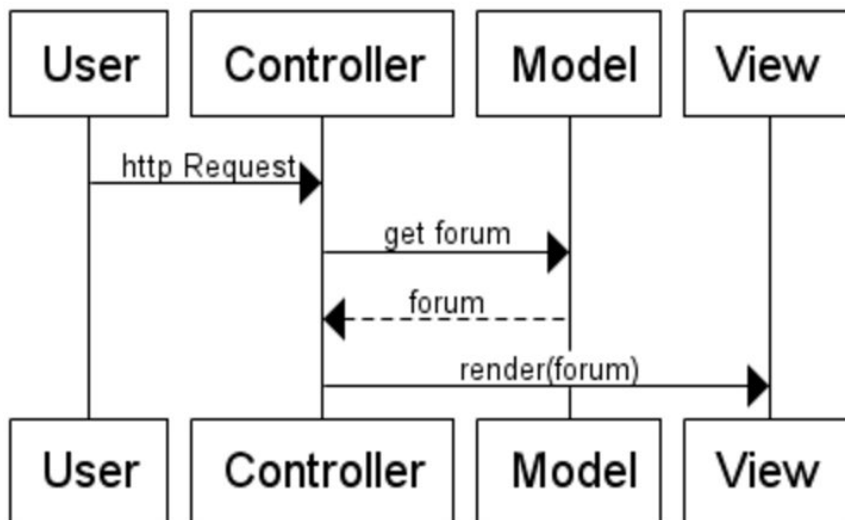
The goal of this iteration was to implement users into our system and associate these users with courses that the users have completed, are enrolled in, or have previously taken/taught. First we needed to implement some sort of login and authentication system so we can begin to assign roles and functions to those roles. From there we began implementation and improvements to user actions. Posts, replies, and votes are now linked to user id's. Replies and votes are also linked to their appropriate Posts. On the management side of things, admin and faculty users can add, view, and modify user accounts as well as classes assignment to students.

## Class Diagram

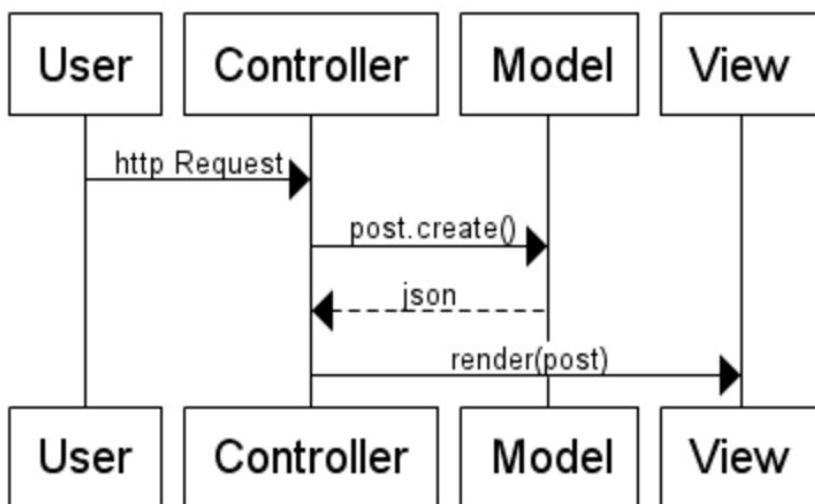


# Sequence Diagrams

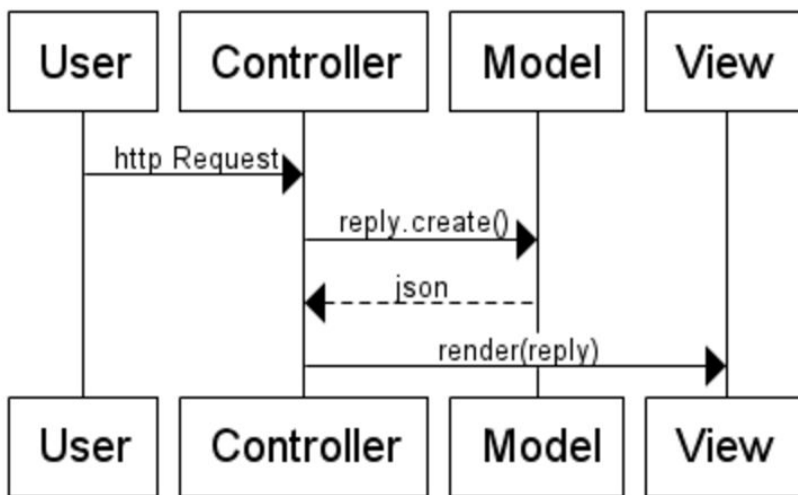
## 1. View Forum



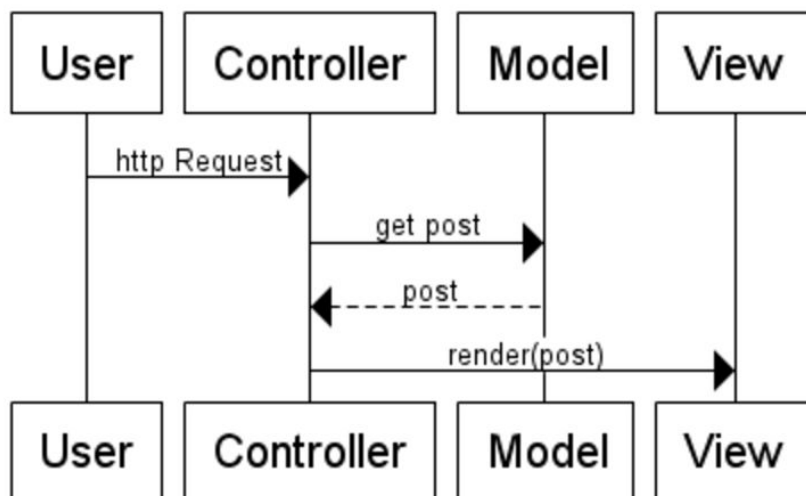
## 2. Ask Question



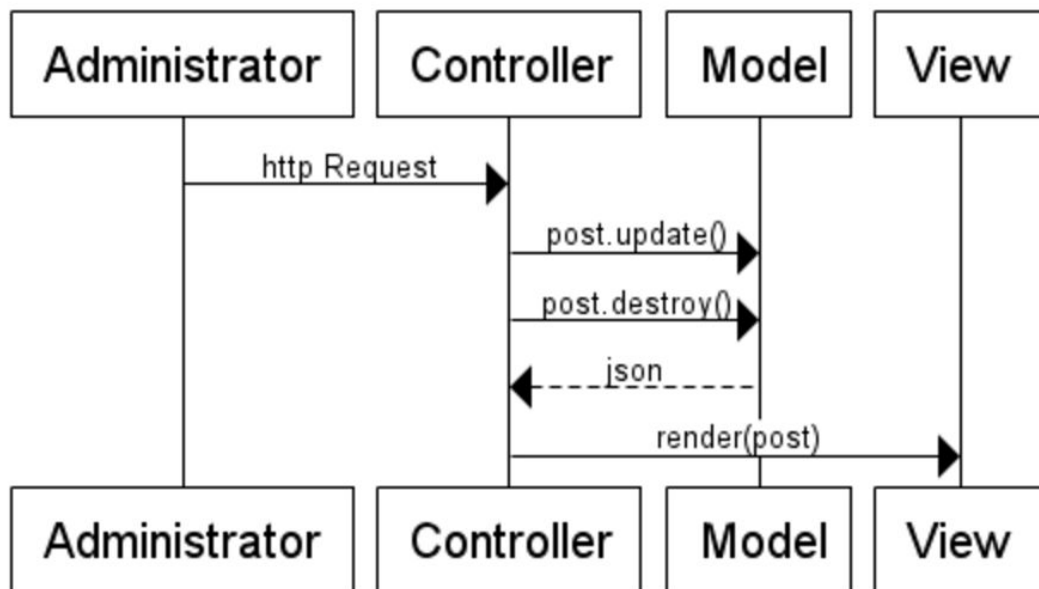
### 3. Answer Question



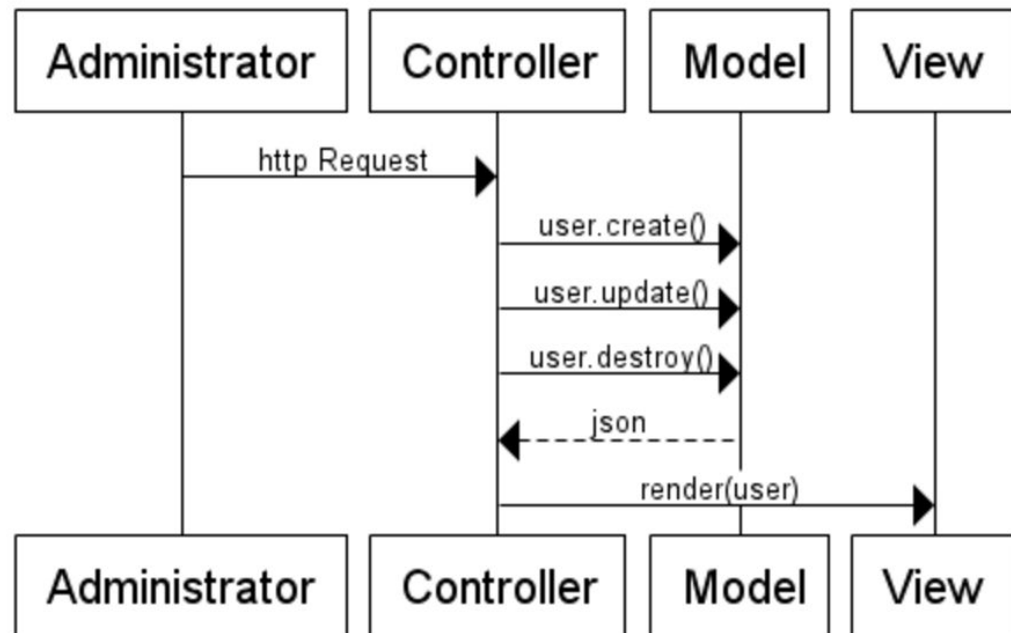
### 4. View Thread



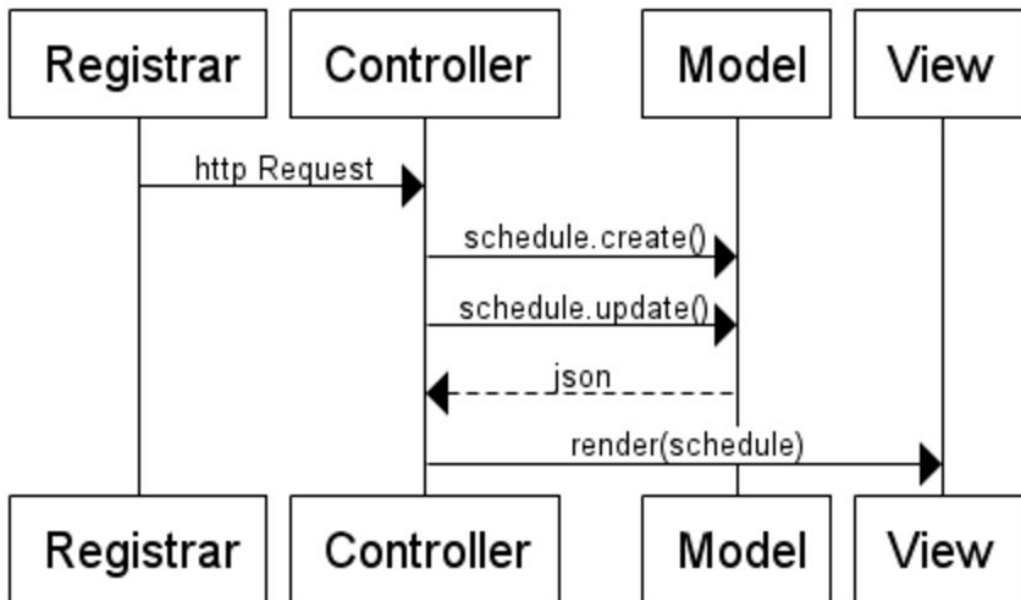
## 5. Manage Thread



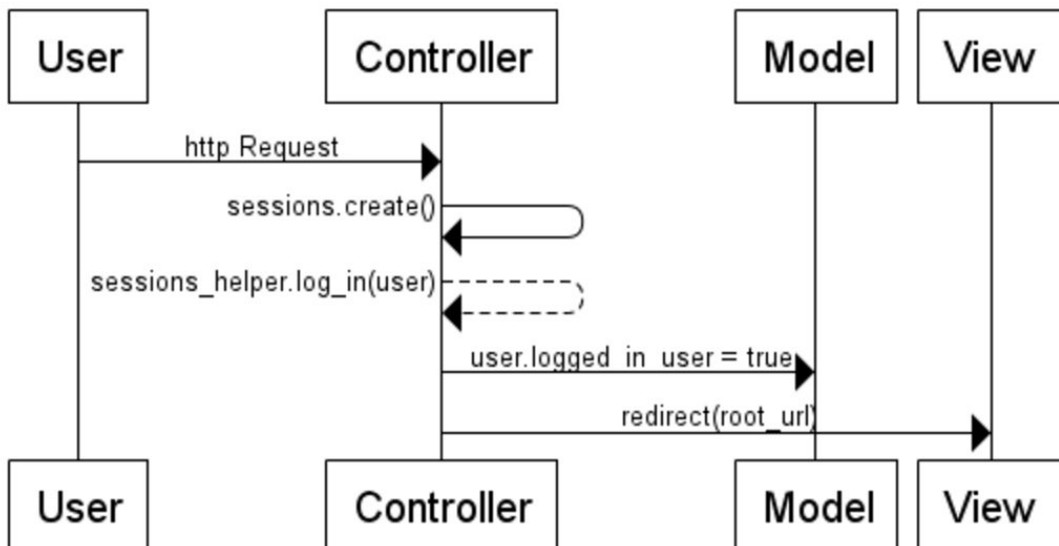
## 6. Manage Users



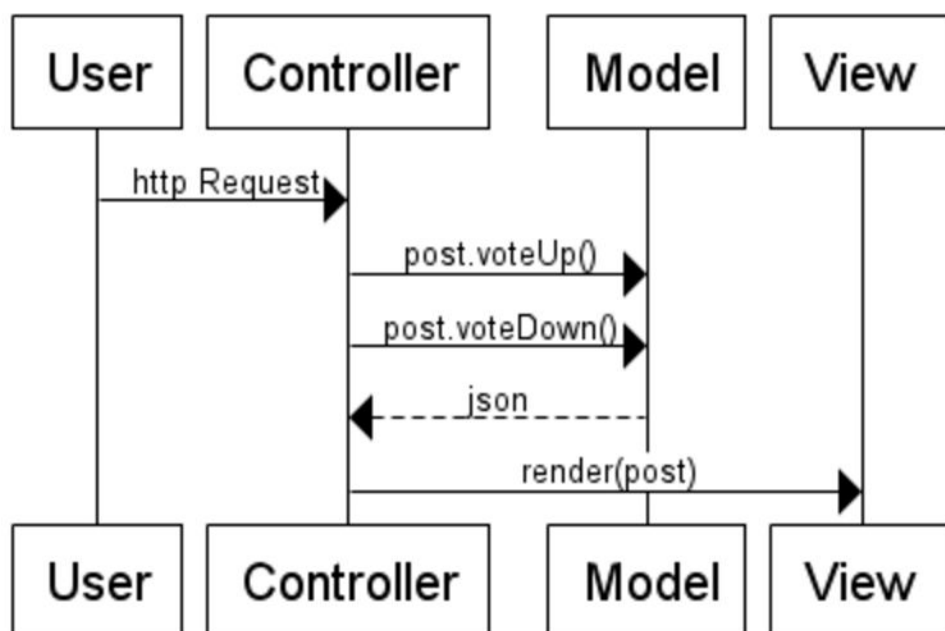
## 7. Manage Schedule



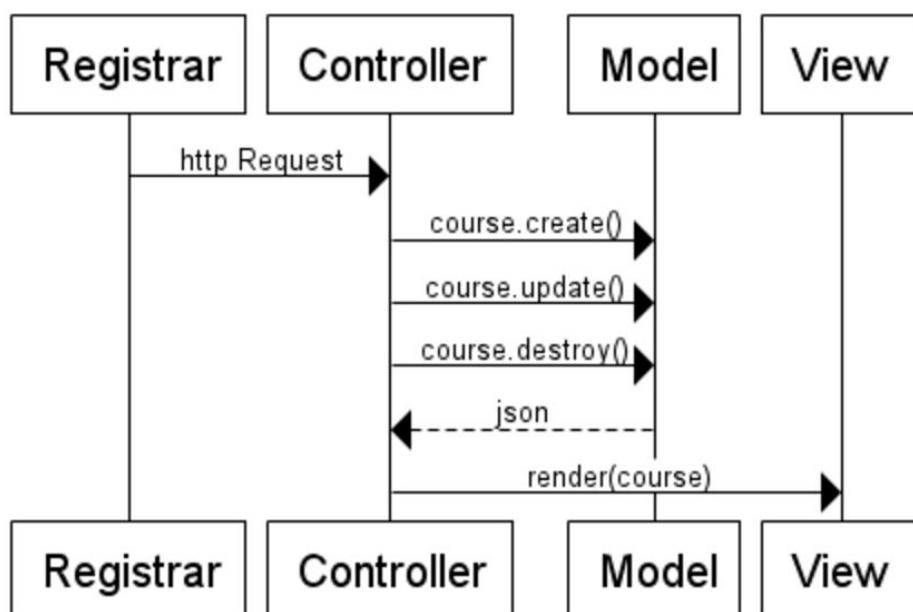
## 8. Authenticate User



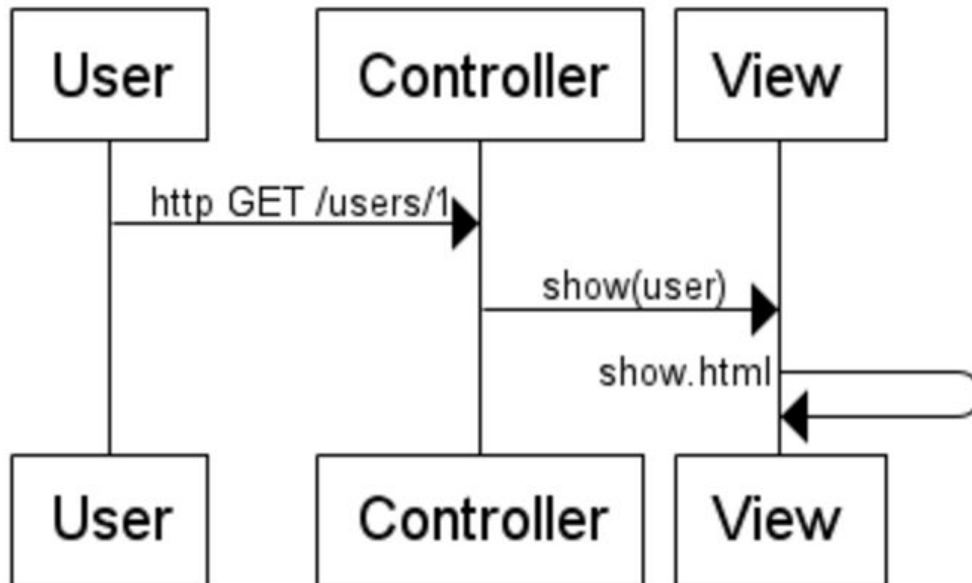
## 9. Vote



## 10. Manage Class



## 11. View User





# Design

## Classes

The basis of User interaction revolves around three main classes: forum, post and reply (all of which inherit from the ApplicationRecord).

**course** – Each instance of a course has an id and a course\_subject. Forum access is mediated by the forums\_controller.

**post** – Each instance has a content and a title and is linked to a forum. Post access is mediated by the posts\_controller.

**reply** – Each instance has a content and is linked to an associated post. Reply access is mediated by the replies\_controller.

**user** – Each instance is associated to 0  $\Rightarrow$  \* course(s), post(s) and replies. User access is mediated by the user\_controller.

## Controllers

Access to the above classes is moderated by three controllers: forums\_controller, post\_controller and replies\_controller (all of which inherit from the ApplicationController).

**course\_controller** – Given a subject\_name, the forum\_controller will use the necessary CRUD operation to create an instance of the course class.

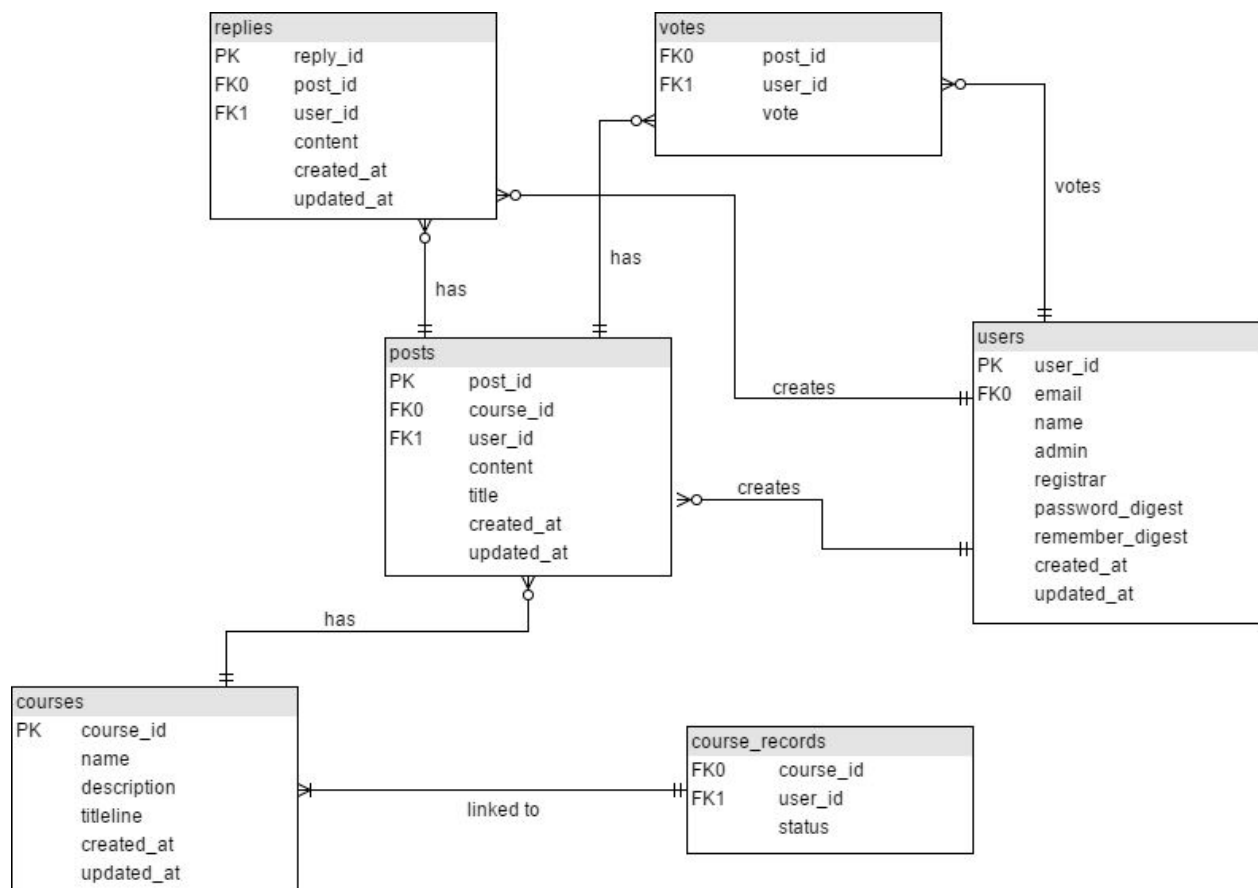
**post\_controller** – Given a title, content and the forum\_id of an existing forum instance, the post\_controller will call upon the proper CRUD operation to create a new instance of the post class.

**replies\_controller** – Given a content and the post\_id of an existing post instance, the replies\_controller will create an instance of the reply class.

**user\_controller** – Manages user roles, authentication and creates associations to user courses, posts and replies.

In addition, all the above controllers have CRUD operations to modify and delete entity instances (update and destroy respectively).

## Database



Forum has 0...\* Post entities associated to it. forum\_id is a foreign key in the Post table.

Post has 0...\* Reply entities associated to it. post\_id is a foreign key in the Reply table.

## Rails Console Commands

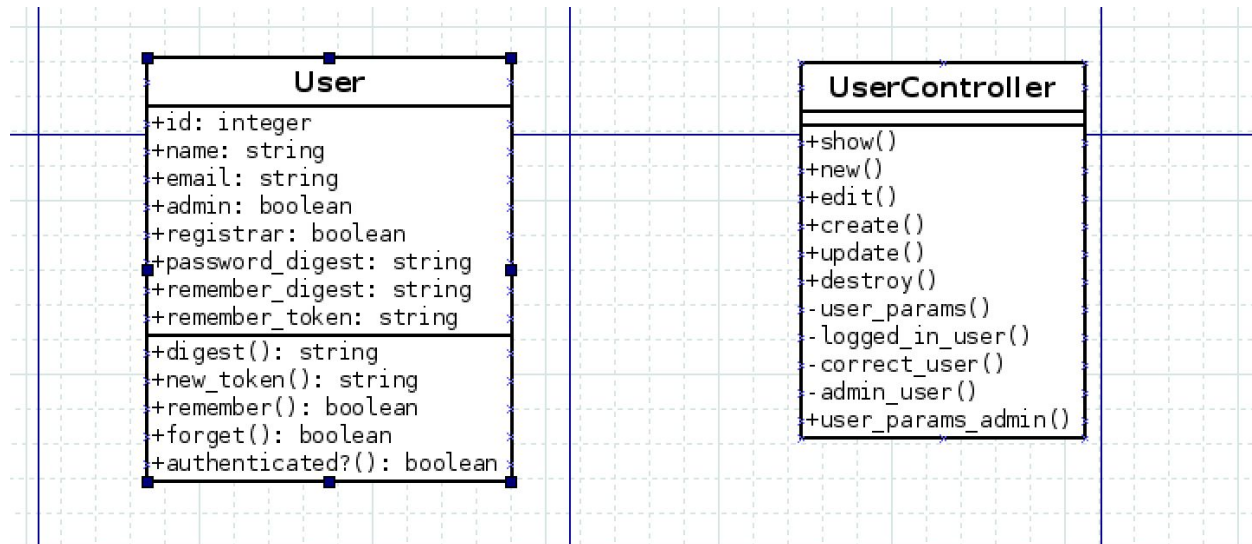
```
Post.create(forum_id: "forum_id", title: "title", content: "content")
```

```
Reply.create(post_id: "post_id", content: "content")
```

## UC5: Manage Thread

Manage Thread is a functionality only accessible to Administrator users. This allows admins to edit and delete user posts and replies. The system logic will simply use the `post_controller` and `replies_controllers`' CRUD operations to update or delete their respective models.

## UC11: View User & UC6: Manage User

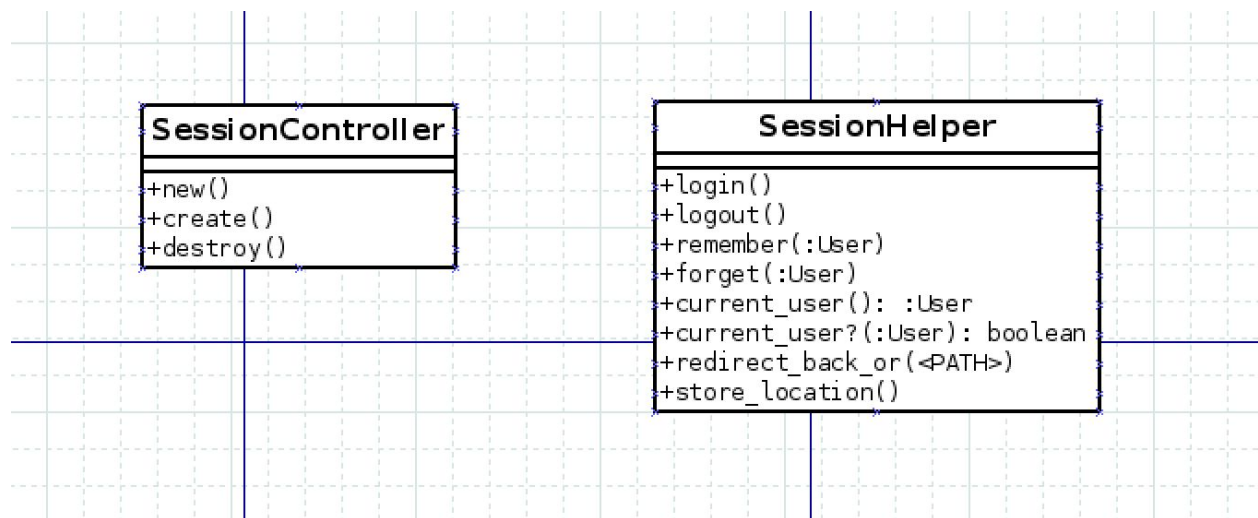


We decided to represent Users in our system with a User model and implemented a corresponding UserController to perform the standard RESTful actions with the User model. There's currently 3 types of Users in our system: User, Administrator, and Registrar. The admin and registrar access rights are simply indicated with boolean values in the User model. Due to Faculty and Users performing the same actions everywhere but their course and User page, we decided to use an ERR model (CourseRecord) to properly represent that relationship. The **user\_params** and **user\_params\_admin** are included in the controller to ensure proper strong parameter typing is used in updates based on the User type. **Correct\_user**, **logged\_in\_user**, **admin\_user**, are all used in callbacks to ensure that only users with the correct permissions are allowed to use restricted RESTful actions. It's also worth noting that **new\_token()**, **remember()**, and **forget()** are all methods used to implement permanent cookie sessions.

## **UC7: Manage Schedule**

Manage Schedule is a functionality only accessible to Registrar users. This functionality entails that they can upload a csv file of student schedules, which will be managed by the course\_controller.

## **UC8: Authenticate User**



In order to implement user authentication we first needed something to authenticate against, for that we chose Bcrypt. Bcrypt gave our project the ability to store password hashes to authenticate against credentials passed into our system. In order to pass said credentials into our system, we created a SessionController with a View to login to the system. Since authentication throughout a User session must be checked on each request, we created a SessionHelper file with auxiliary methods that all controllers can use to ensure that the user is properly authenticated.

## **UC9: Vote**

This functionality is accessible to all logged in users. It is registered on the front end and managed in the replies\_controller/post\_controllers' vote\_up and vote\_down methods. These methods then increment or decrement that question/answer's vote count. This method also uses user credentials to update the vote so that user can only vote once

for each question and answer.

### **UC10: Manage Class**

Manage Class is a functionality only accessible to Registrar users. This allows them to add, edit and delete classes. For this operation, the system will simply use the course\_controller's CRUD operations to create, update or destroy model instances.