Design Document

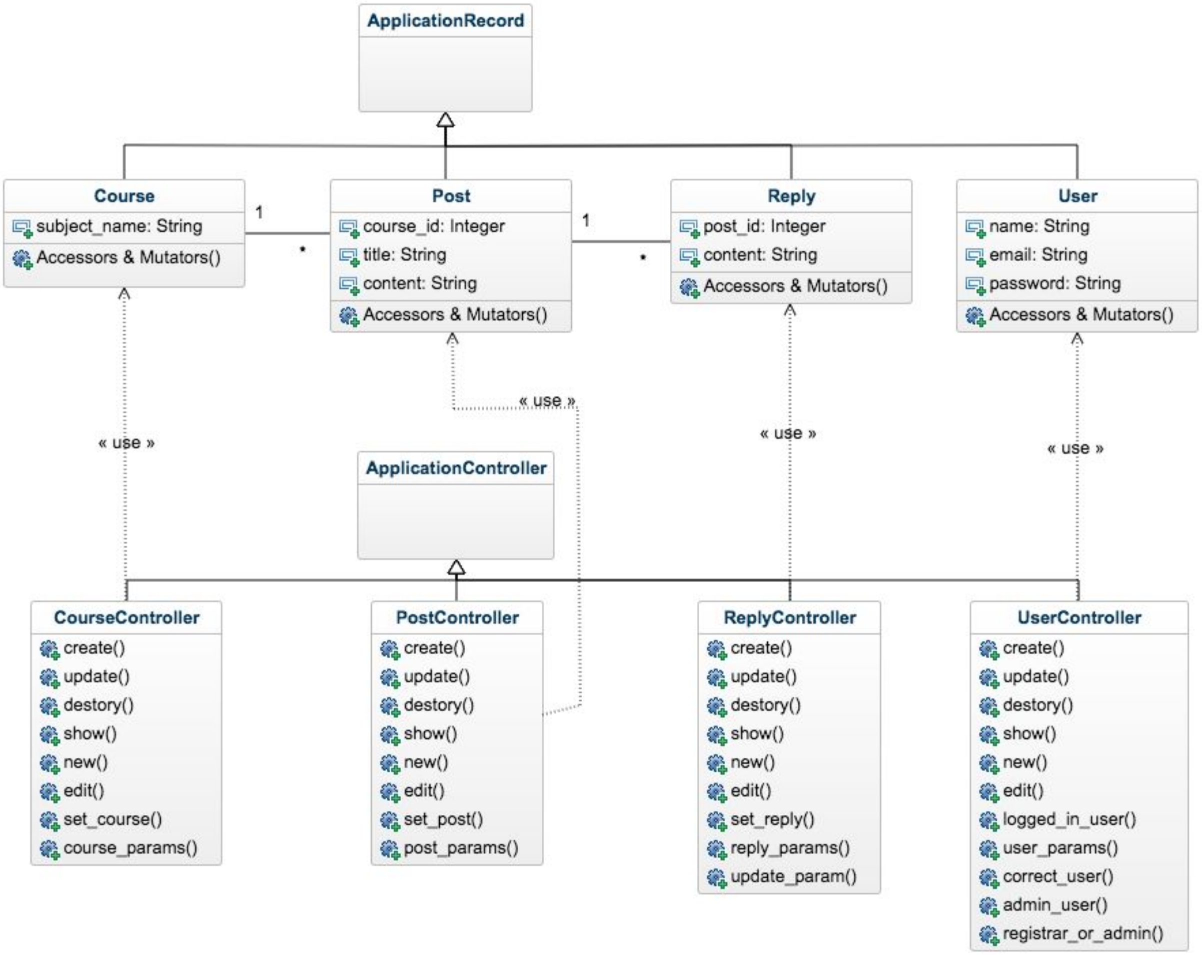
The Borne Collective: NKUNet

Change Discussion

Iteration 3

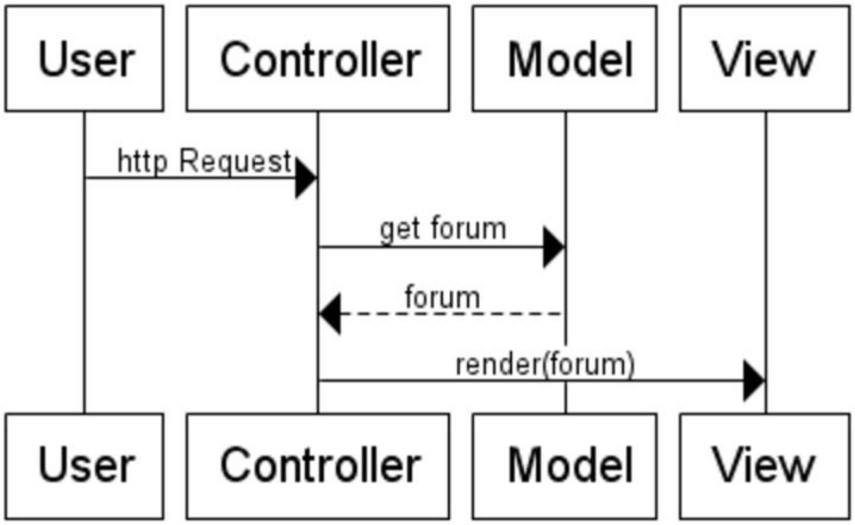
The goal of this iteration is to implement search. We want a user to be able to find posts given a keyword or phrase. In addition to search we have to finish the following use case functionalities: view user, manage schedule and vote. New additions to this document include: a new sequence diagram for UC12: Search and a discussion of the search use case at the bottom.

Class Diagram

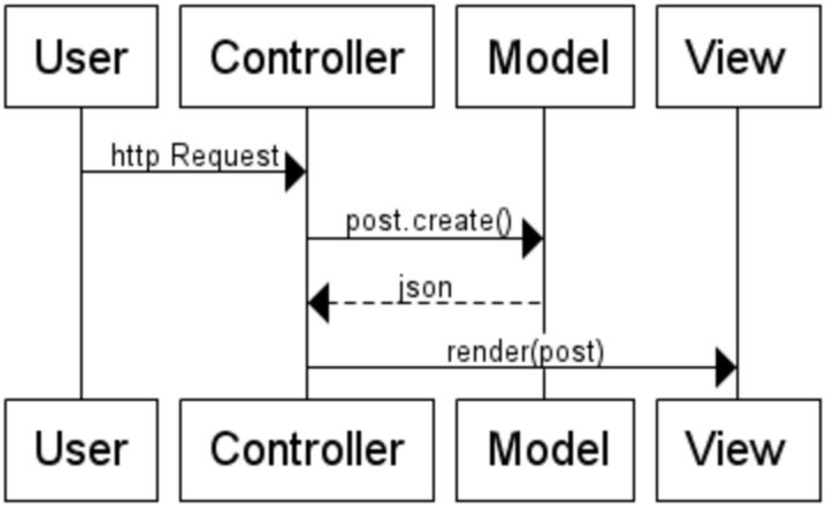


Sequence Diagrams

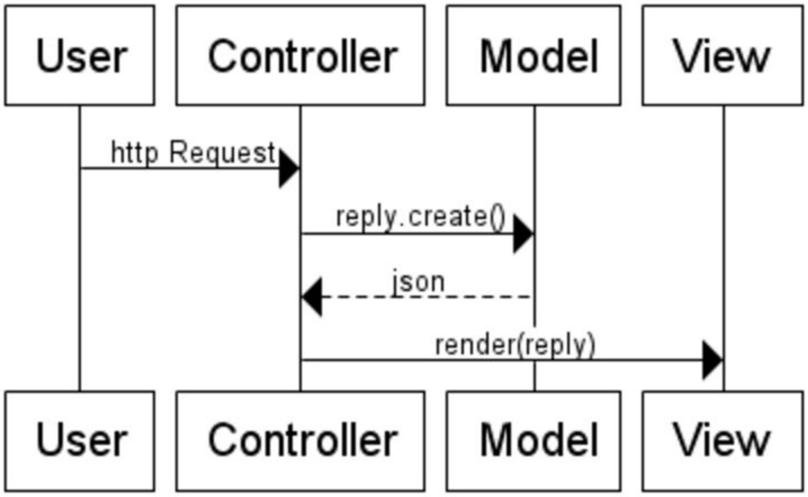
1. View Forum



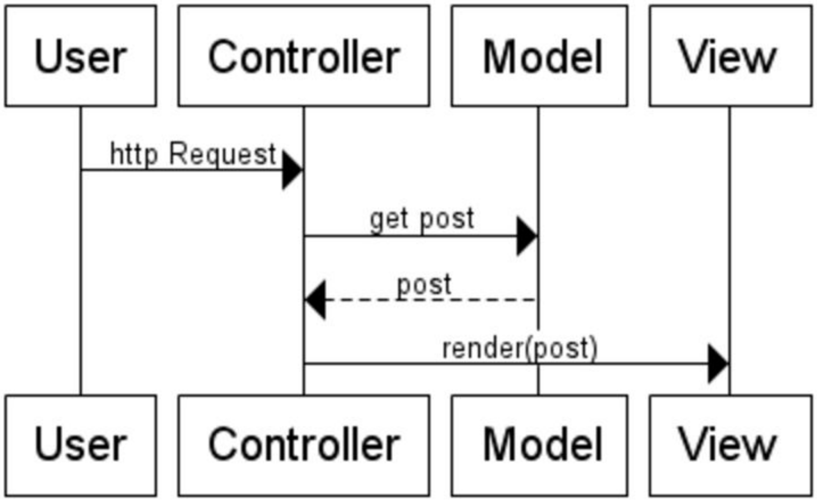
2. Ask Question



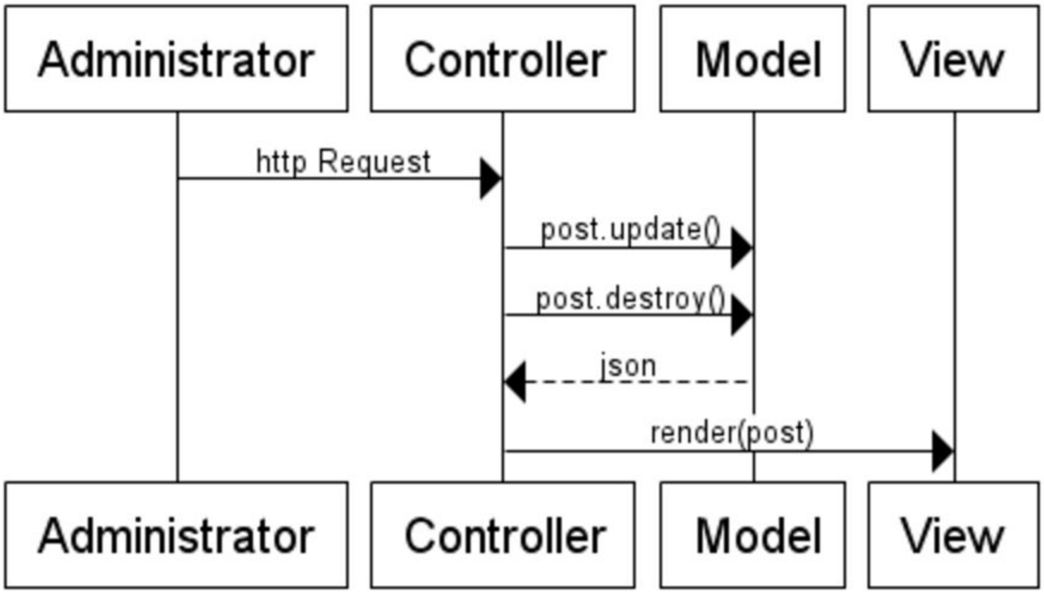
3. Answer Question



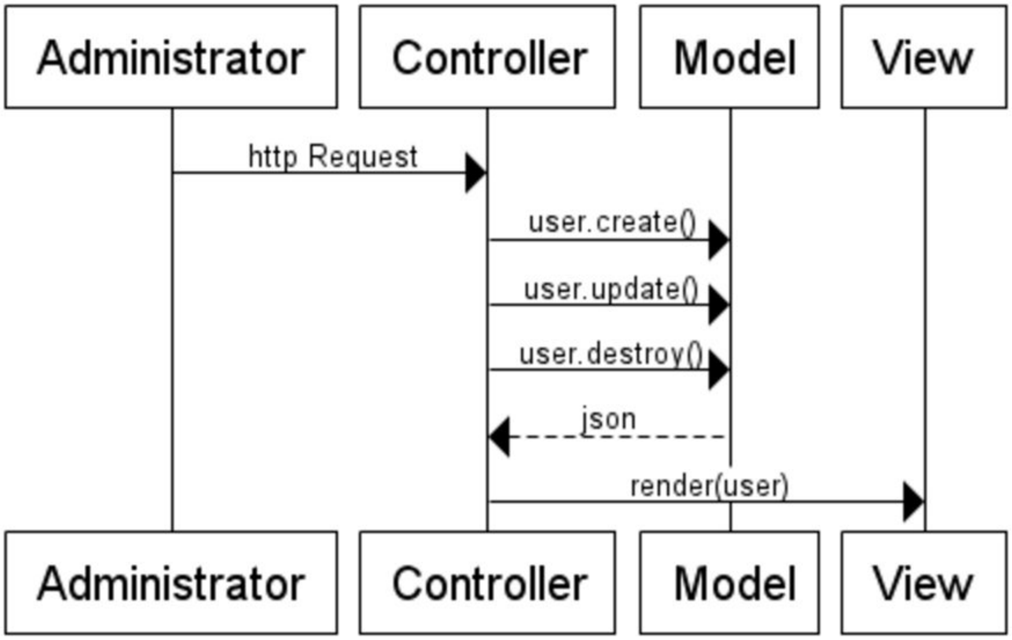
4. View Thread



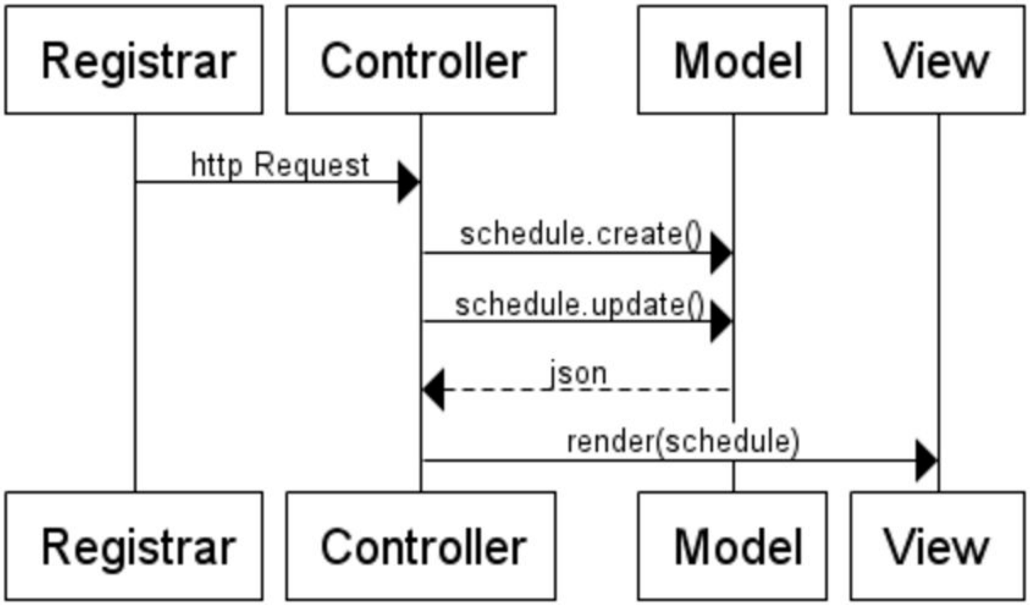
5. Manage Thread



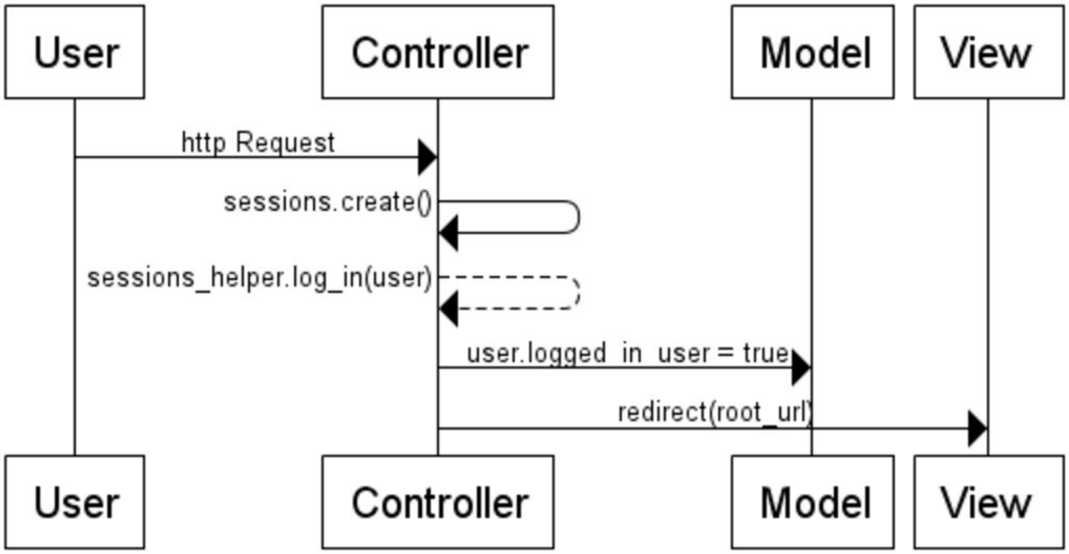
6. Manage Users



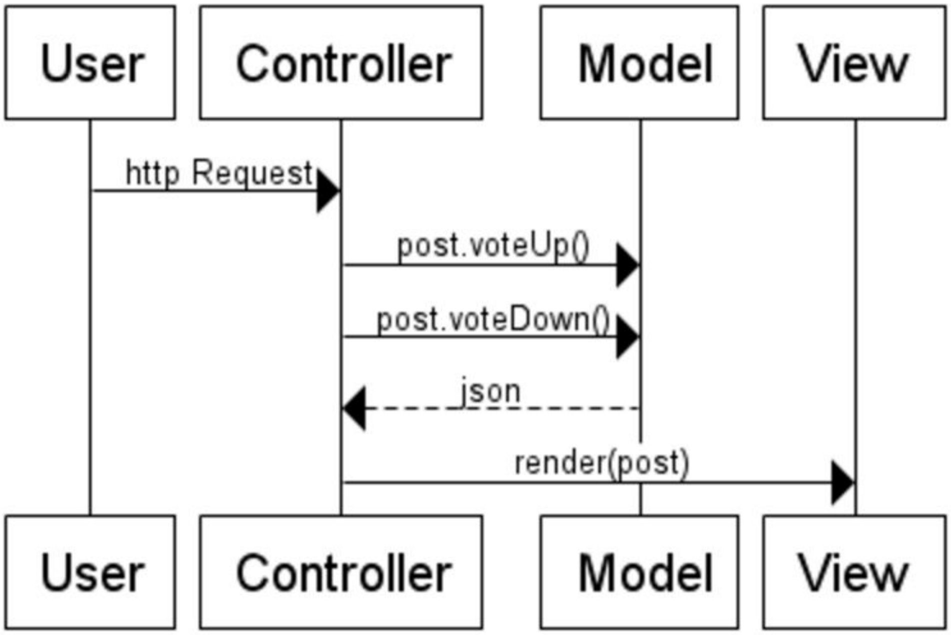
7. Manage Schedule



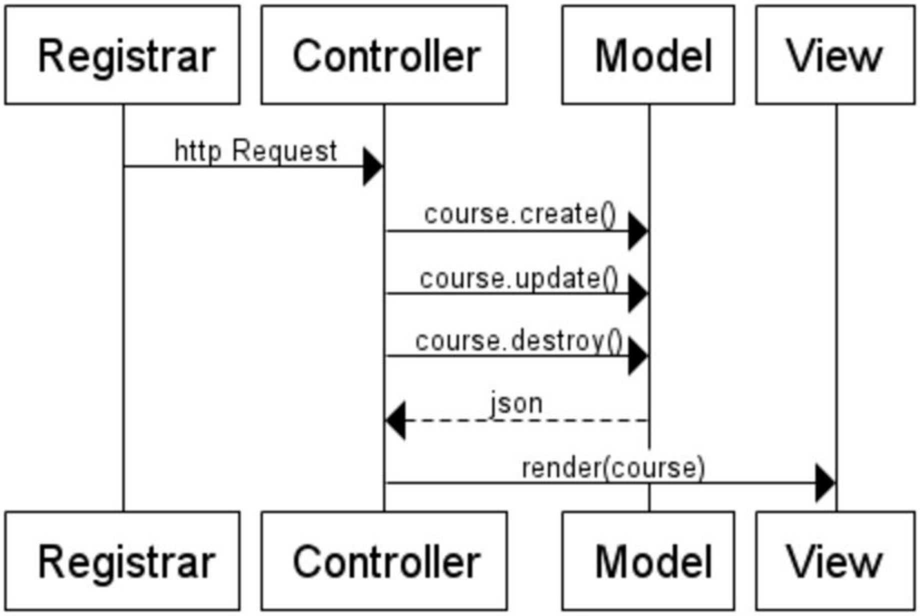
8. Authenticate User



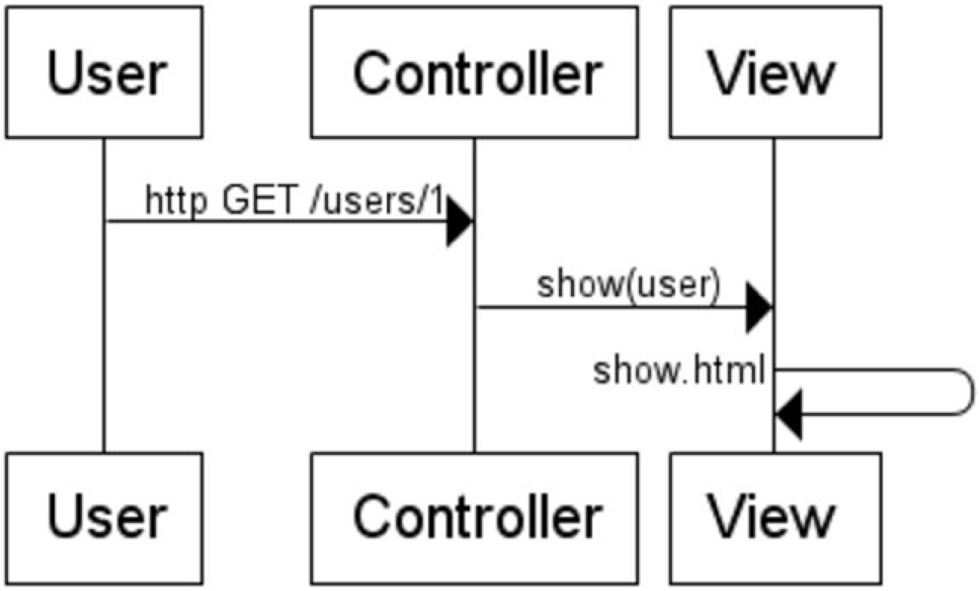
9. Vote



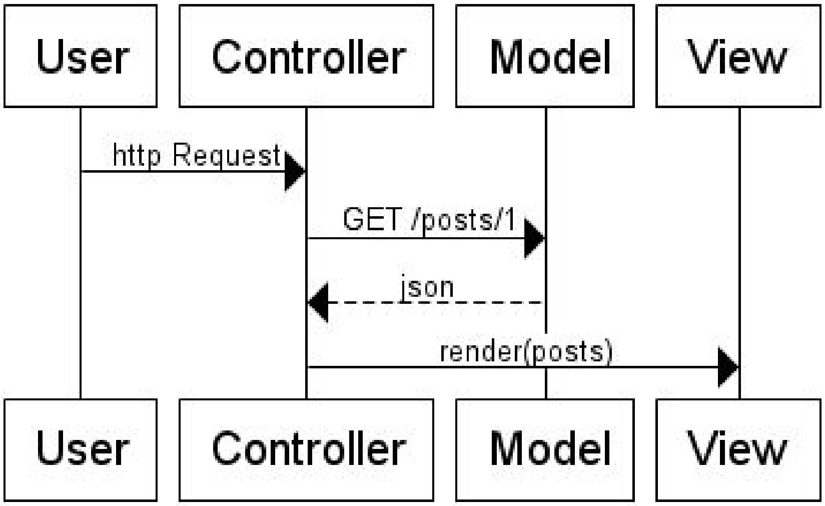
10. Manage Class



11. View User



12. Search



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 ⇒ * 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.

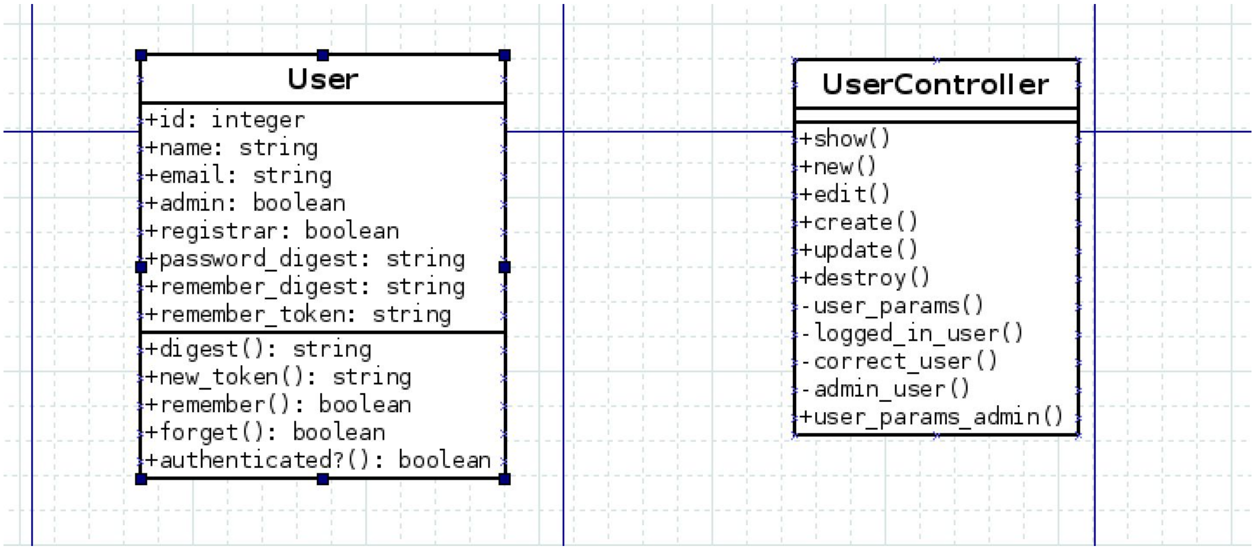
In addition, all the above controllers have CRUD operations to modify and delete entity instances (update and destroy respectively).

Design Discussion

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

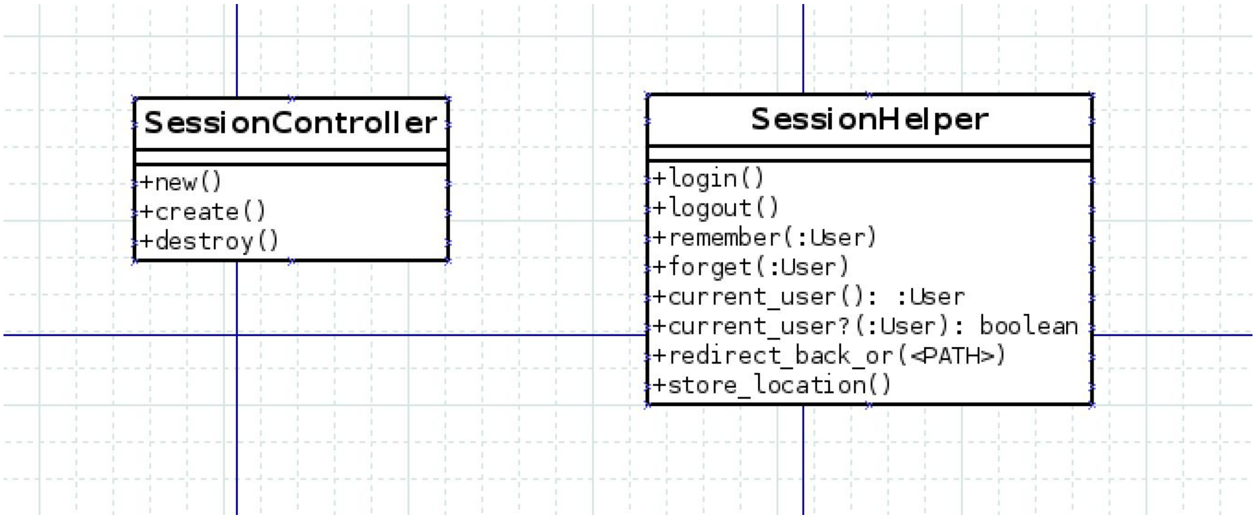


We decided to represent Users in our system with a User model and implemented a corresponding UserController to perform the standard RESTful actions with the User model. There’s currently 3 types of Users in our system: User, Administrator, and Registrar. The admin and registrar access rights are simply 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 were 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.

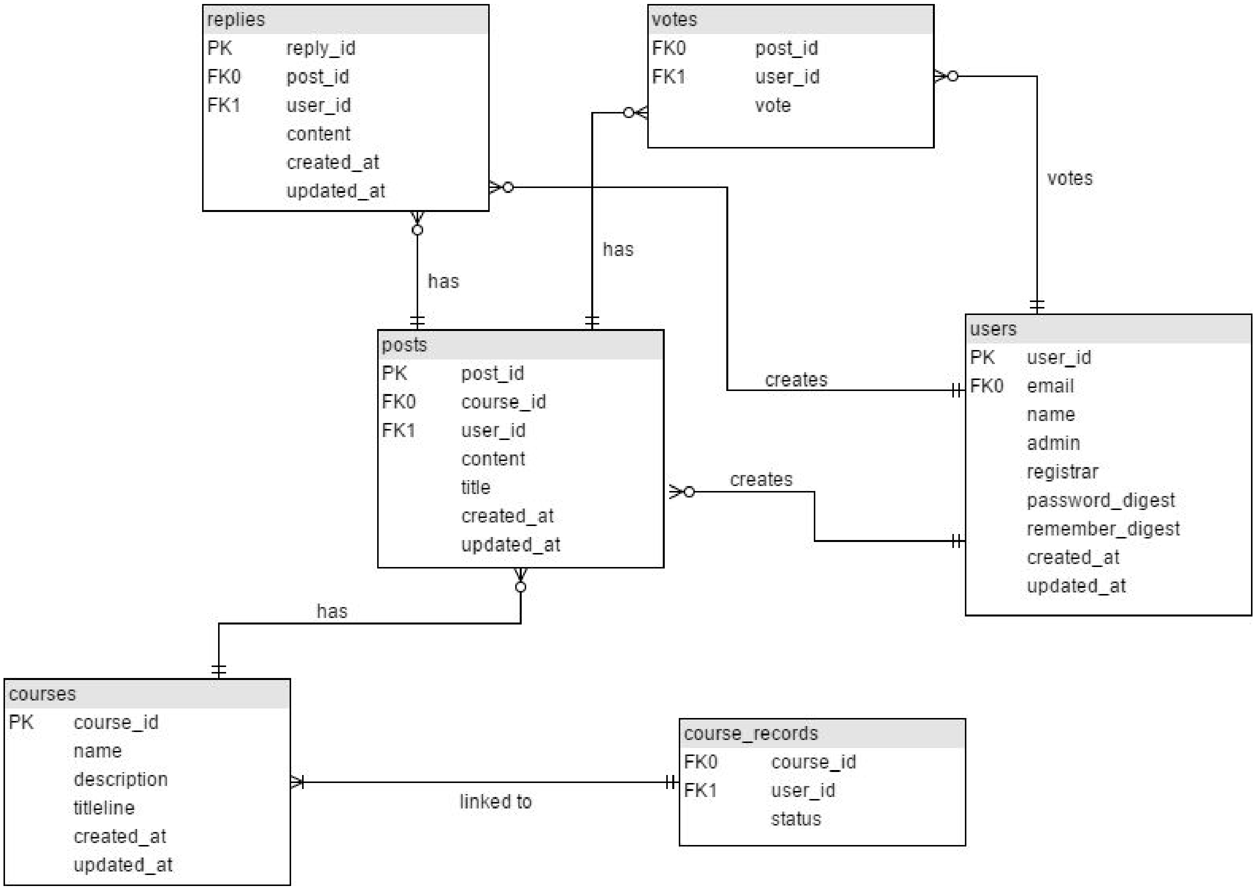
UC10: Manage Class

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.

UC12: Search

Search is a functionality accessible to all users. This allows a users to enter a search criteria then the system will retrieve items from the database (via an API call) that match the specified criteria. If nothing matches the search criteria then the system will display “Sorry, No results found.”

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”)
```