Design Document

The Borne Collective: NKUNet

Change Discussion
- Iteration 2

Class Diagram

Sequence Diagrams

Design
- Classes
- Controllers

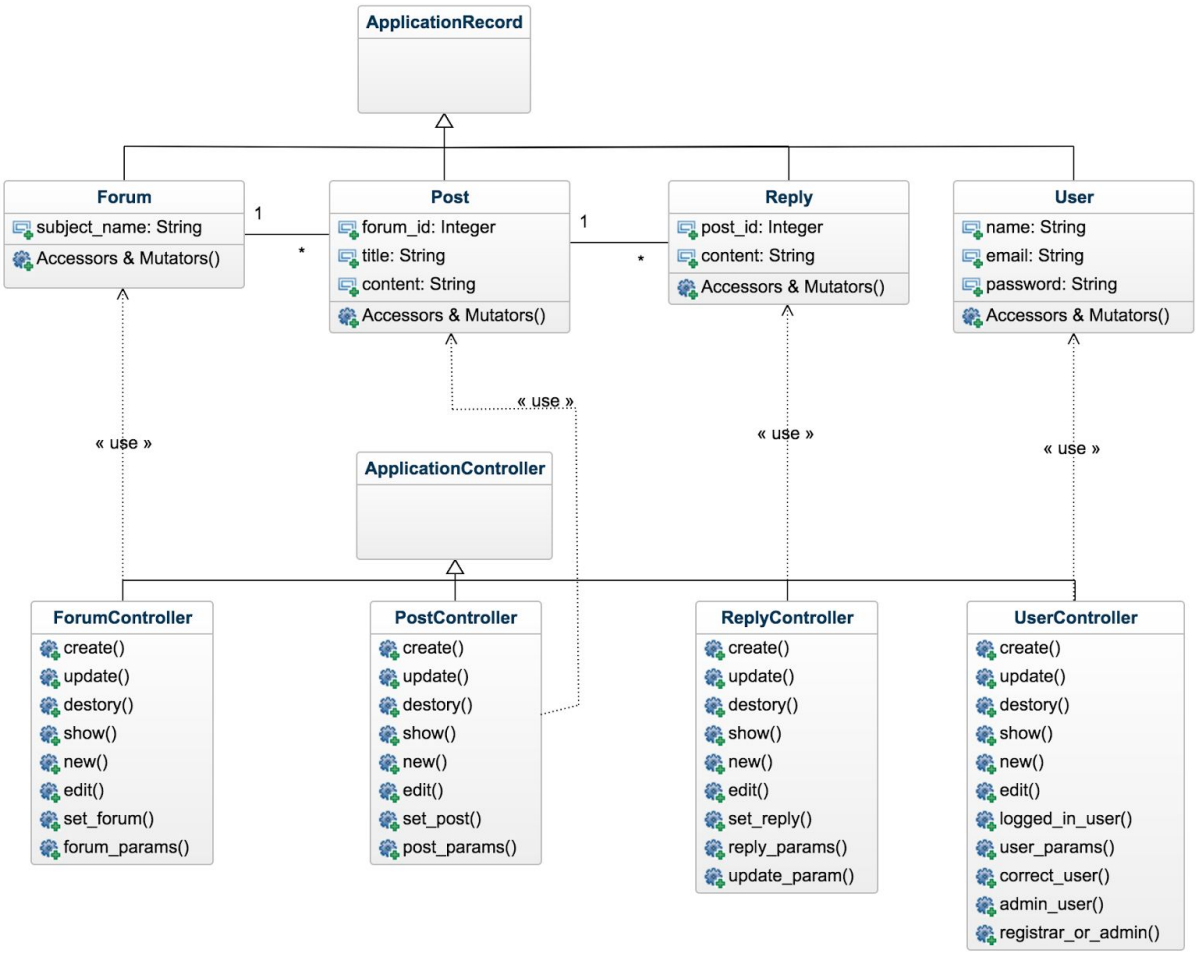
Database
- Tables
- Rails Console Commands

Change Discussion

Iteration 2

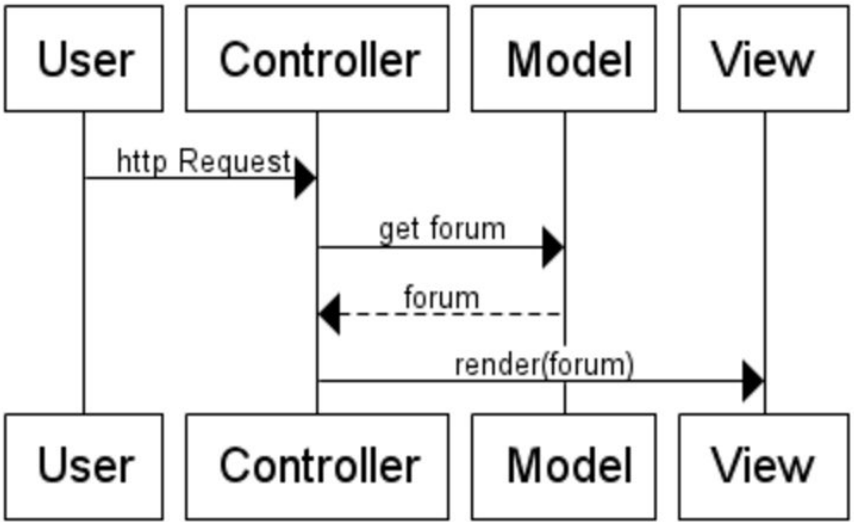
The goal of this iteration was to implement users into our system and associate these users with courses that the users have completed, are enrolled in, or have previously taken/taught. First we needed to implement some sort of login and authentication system so we can begin to assign roles and functions 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

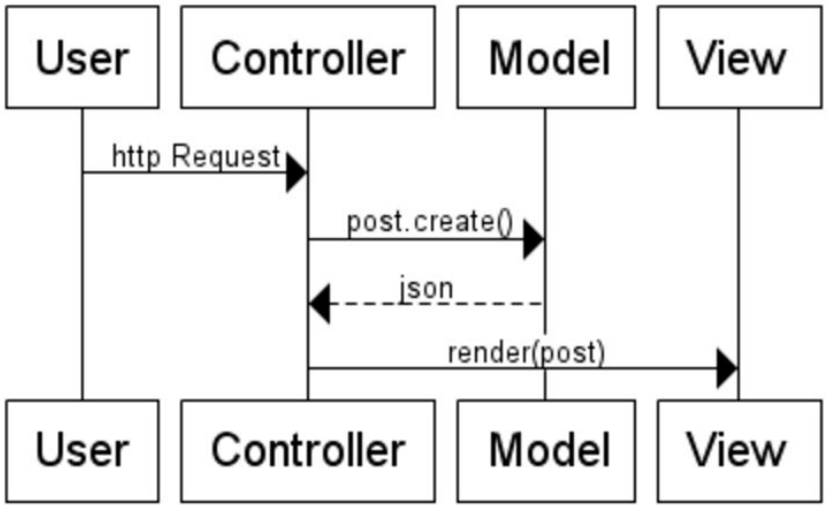


Sequence Diagrams

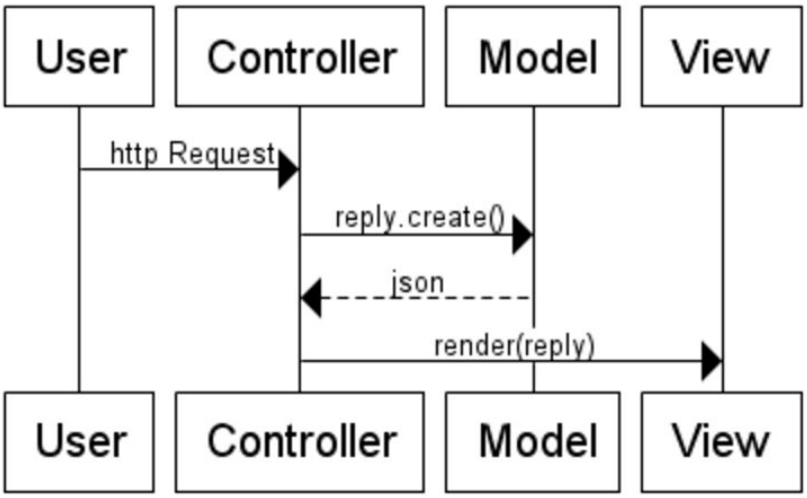
1. View Forum



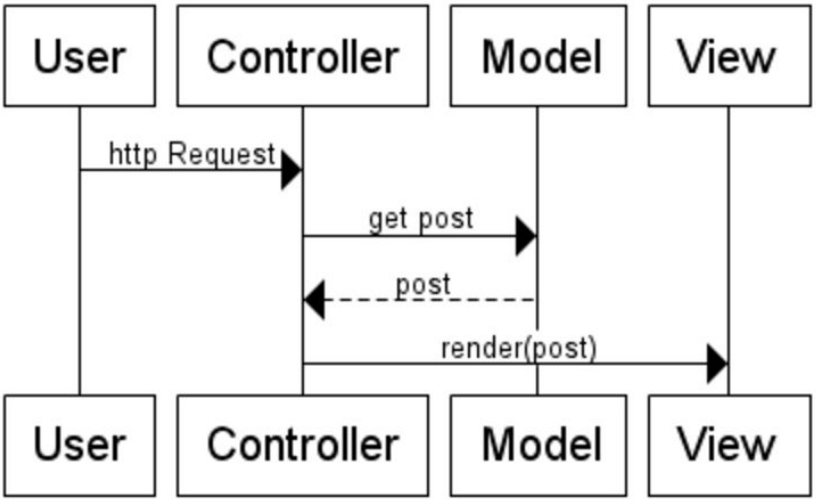
2. Ask Question



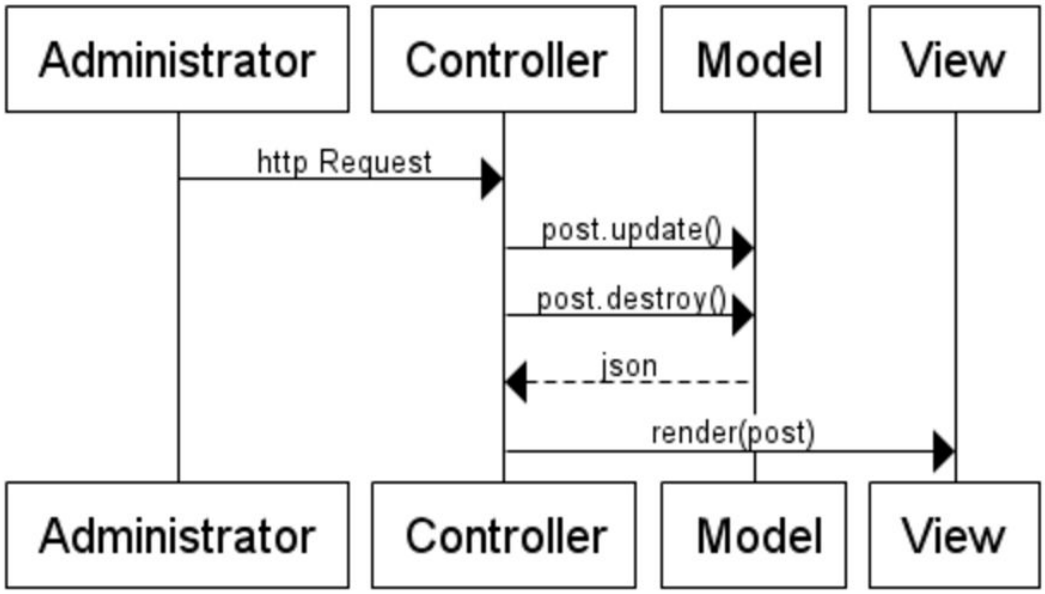
3. Answer Question



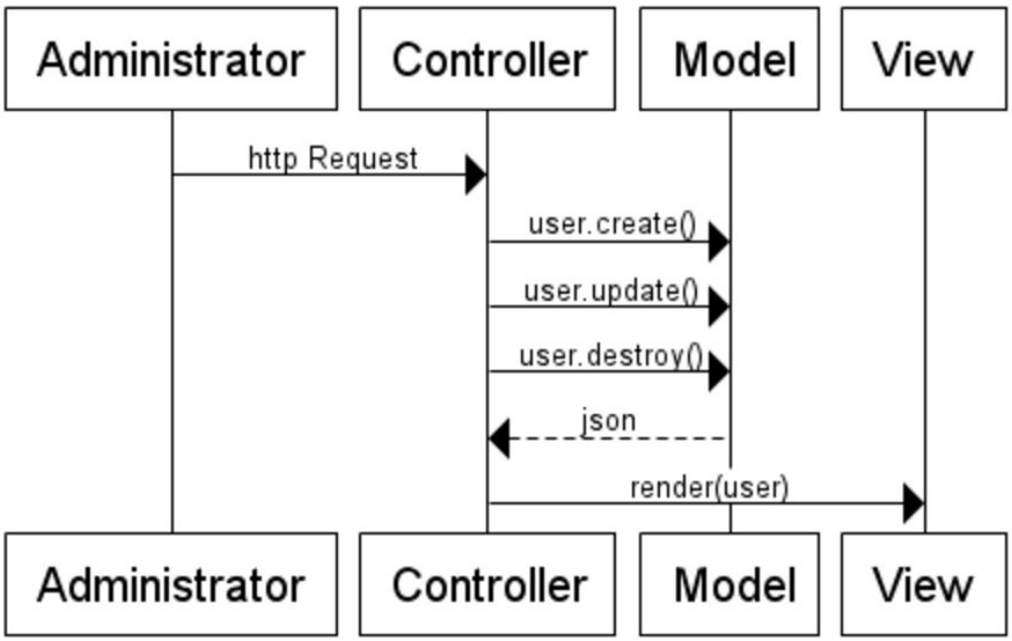
4. View Thread



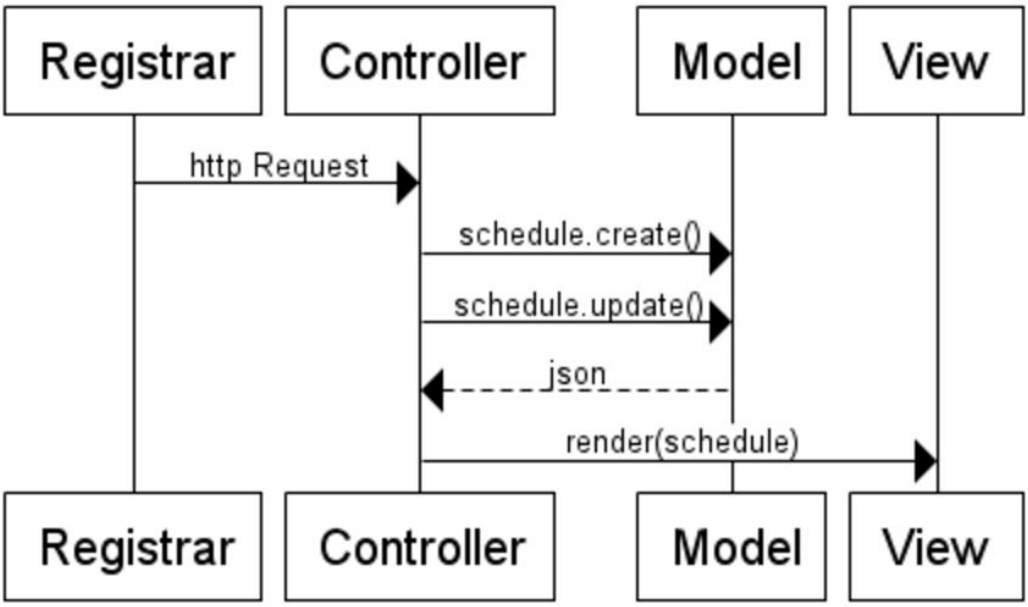
5. Manage Thread



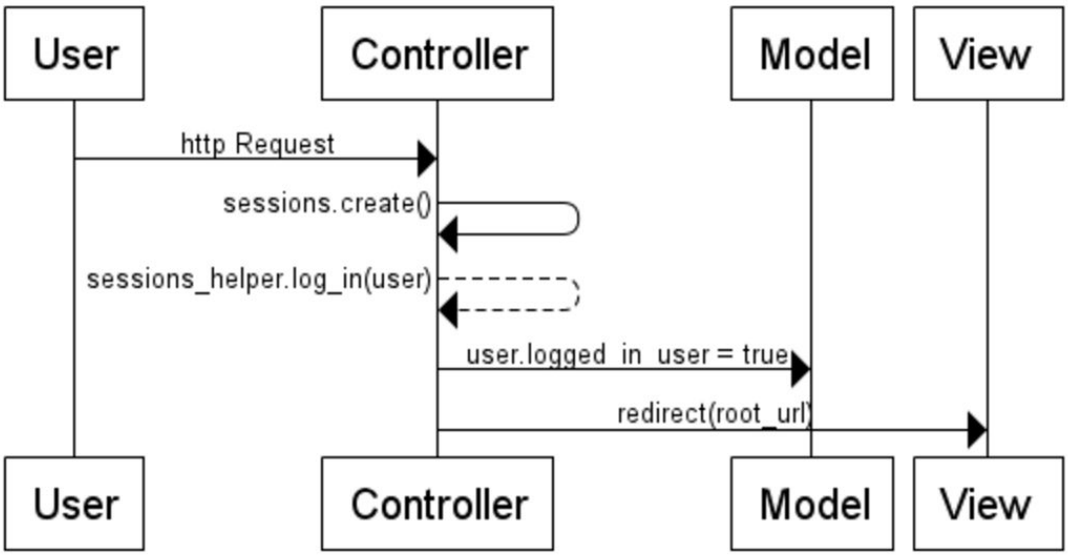
6. Manage Users



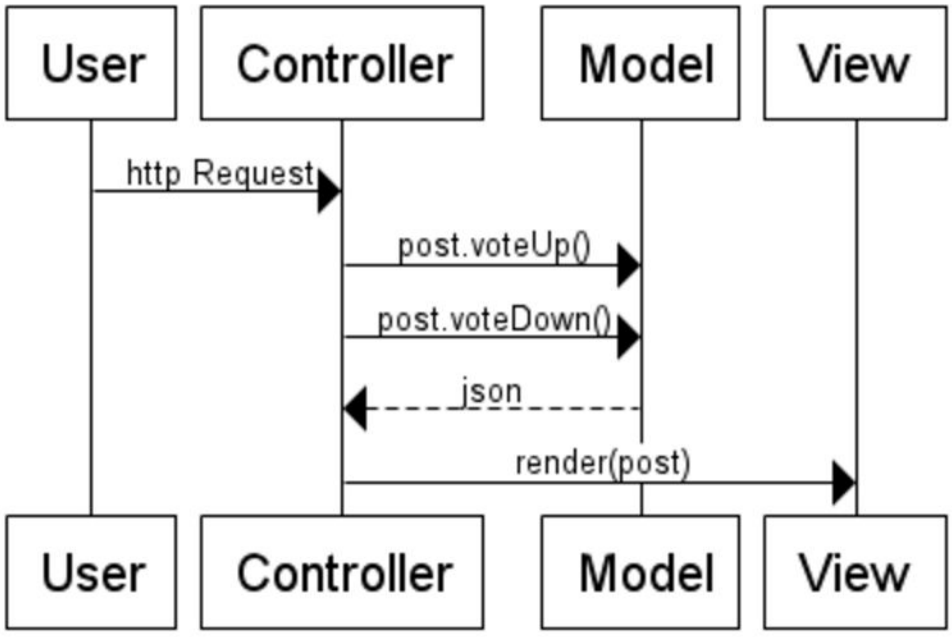
7. Manage Schedule



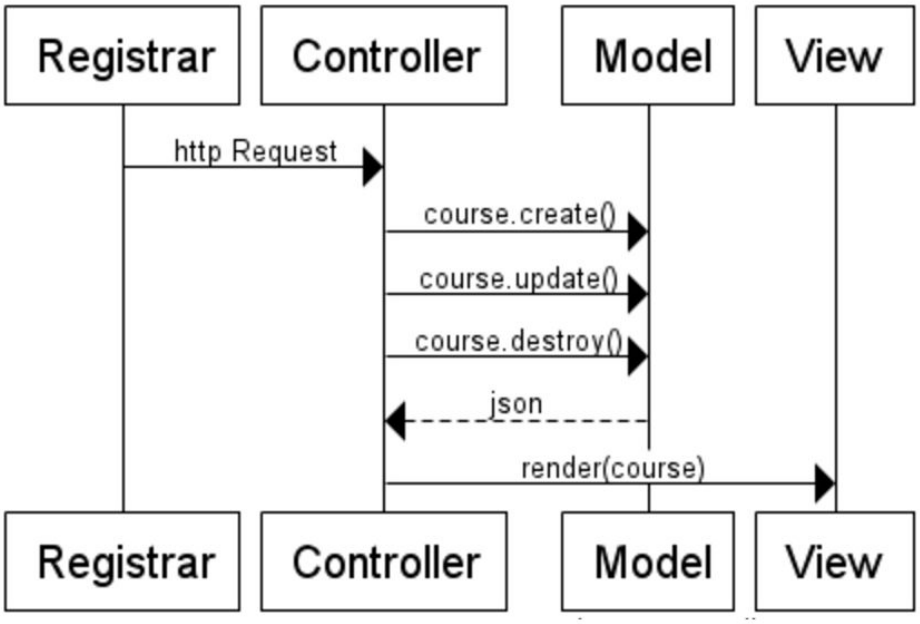
8. Authenticate User



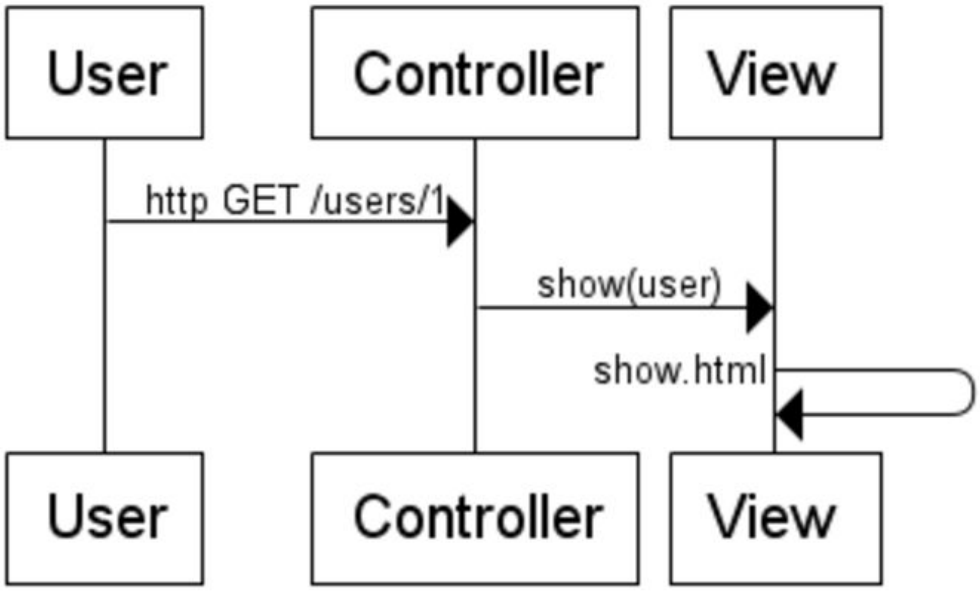
9. Vote



10. Manage Class



11. View User


User
Controller
View
http GET /users/1
show(user)
show.html
User
Controller
View

Design

Classes

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

forum – Each instance of a forum has an id and a forum_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.

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).

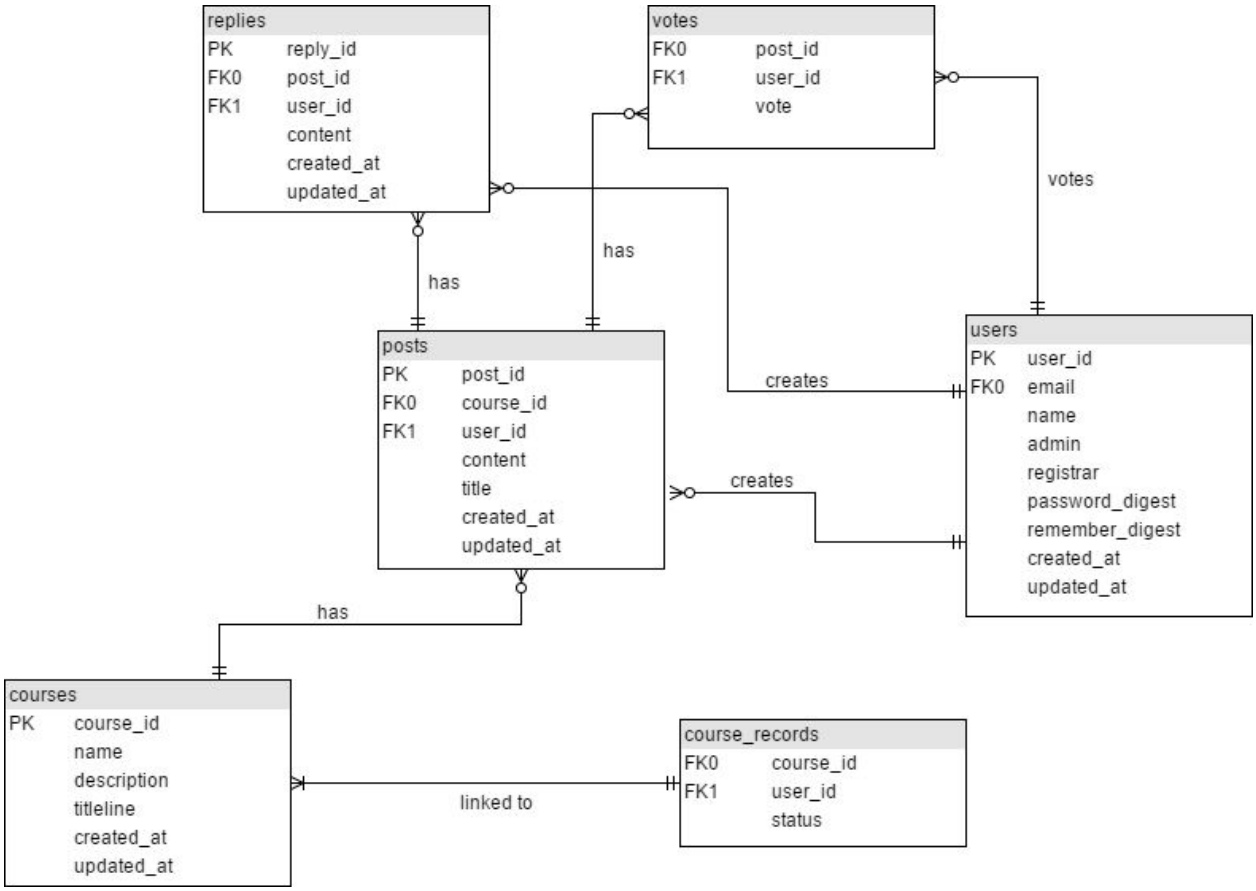
forum_controller – Given a subject_name, the forum_controller will use the necessary CRUD operation to create an instance of the forum class.

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.

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.

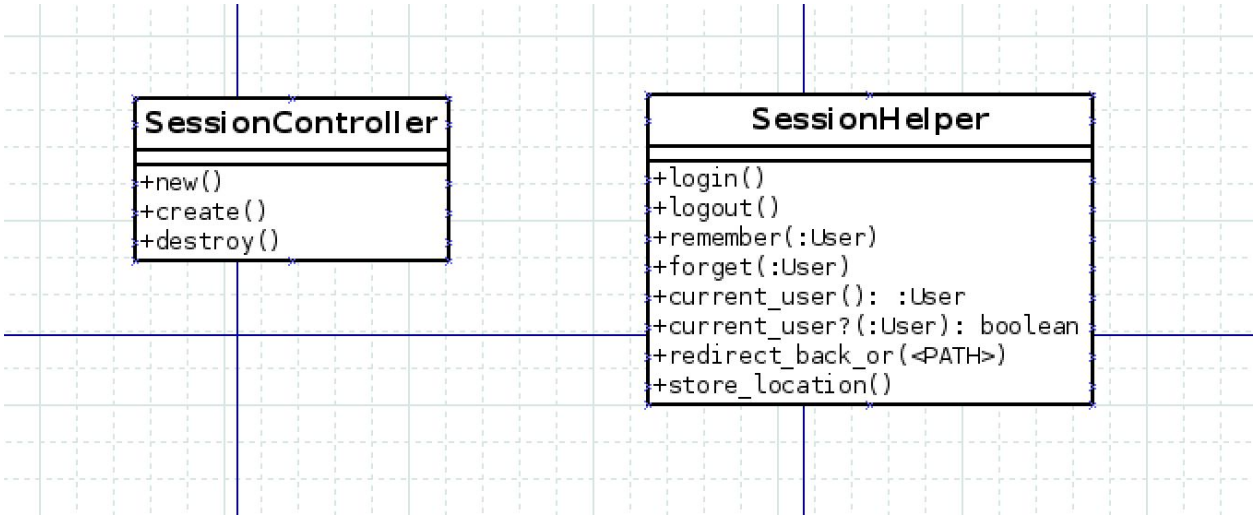
Post has 0...* Reply entities associated to it. post_id is a foreign key in the Reply table.

Rails Console Commands

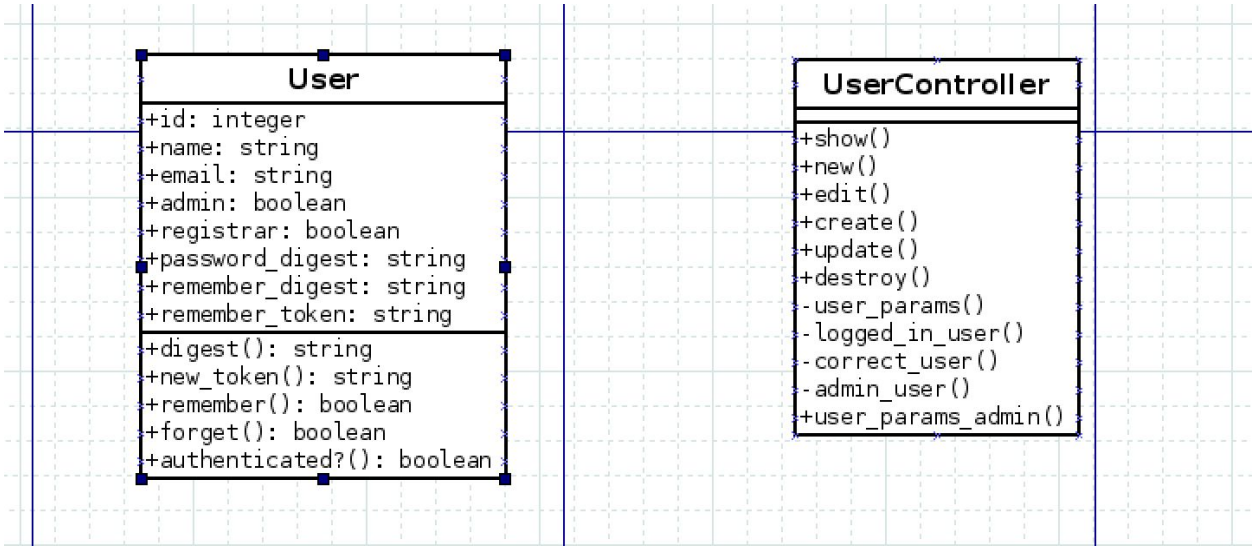
Post.create(forum_id: ”forum_id”, title: “title”, content: “content”)

Reply.create(post_id: “post_id”, content: “content”)

UC5: 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.



UC6: View User & UC7: 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.

UC8: Manage Thread

UC9: Vote

UC10: View Class

UC11: Manage Class

UC12: Manage Schedule