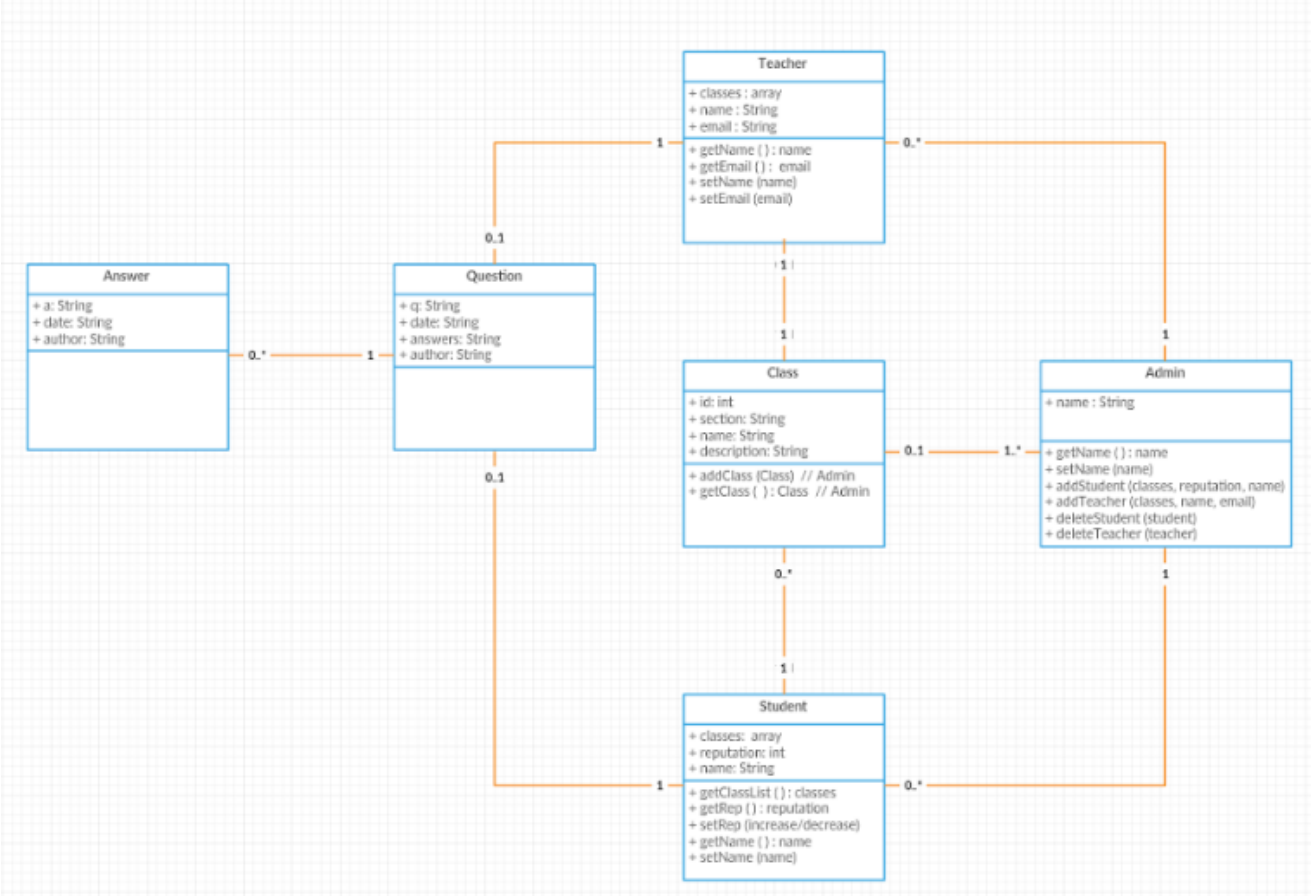
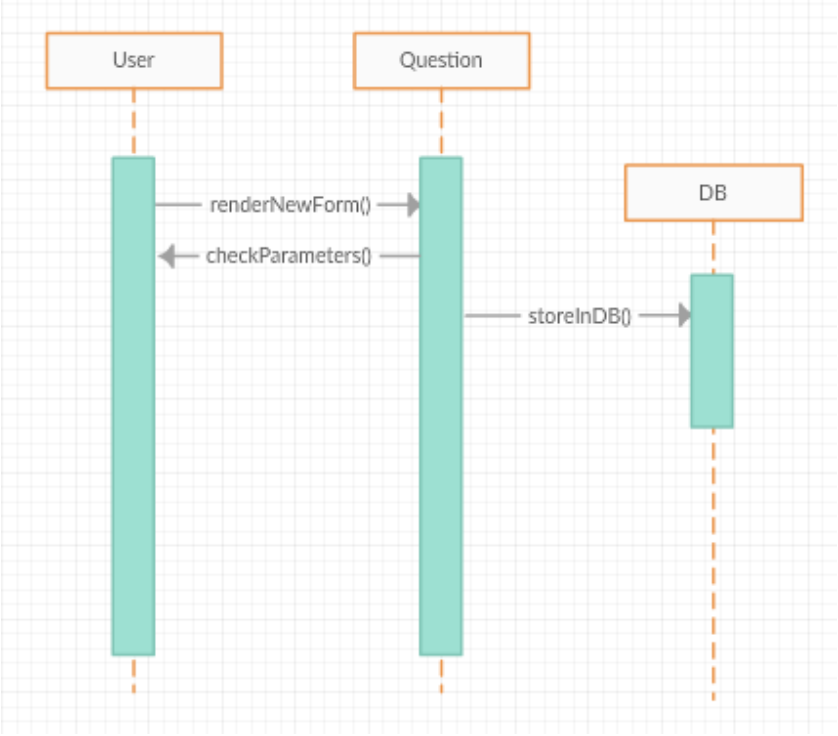
TABLE OF CONTENTS

Class Diagram

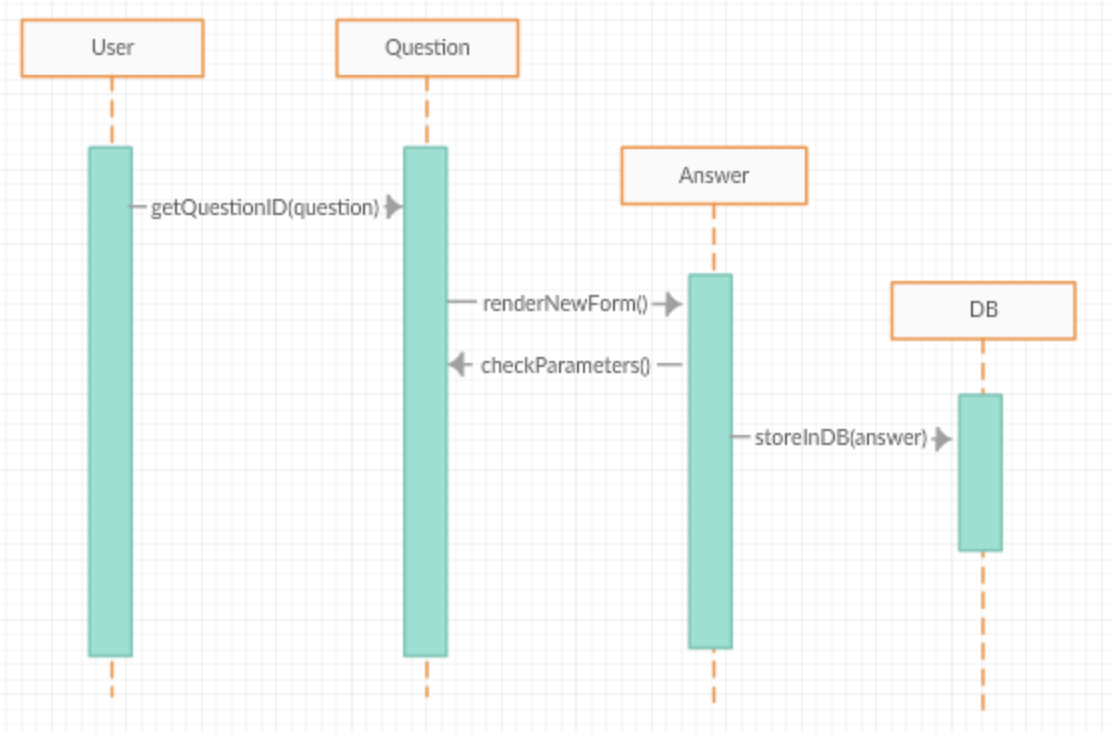


Interaction Sequence Diagrams

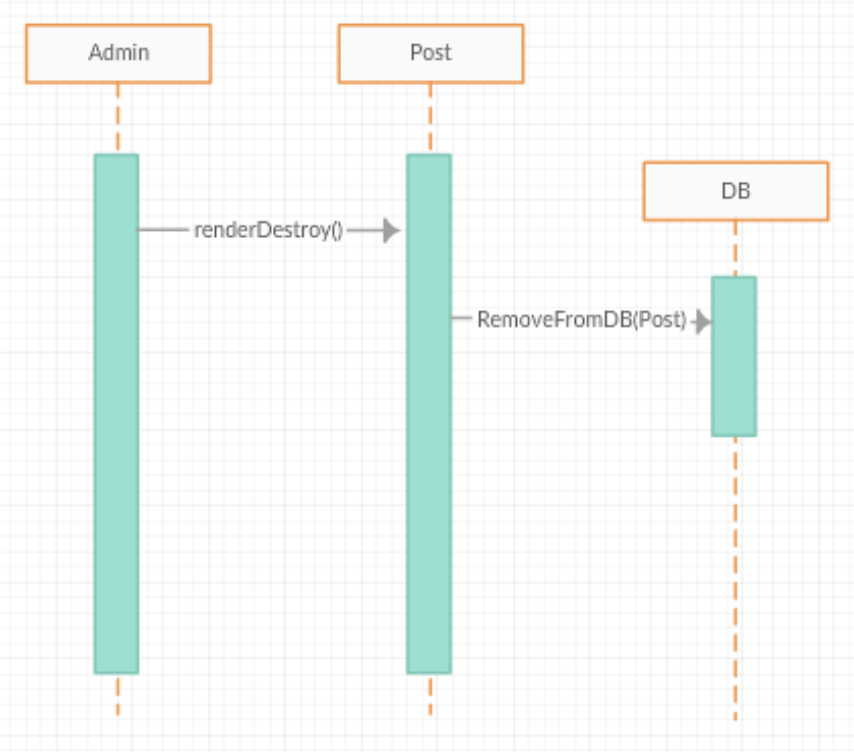
Post Question



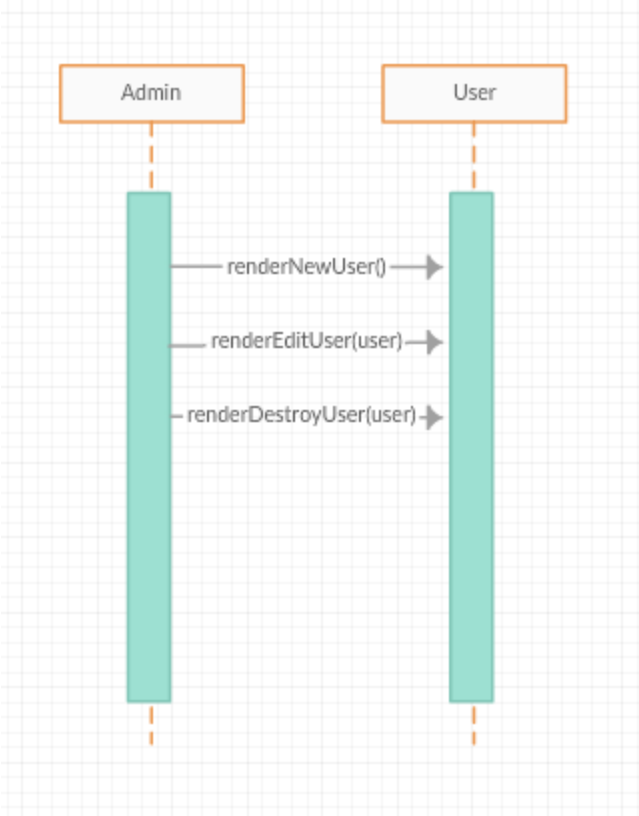
Answer Question



Manage Post



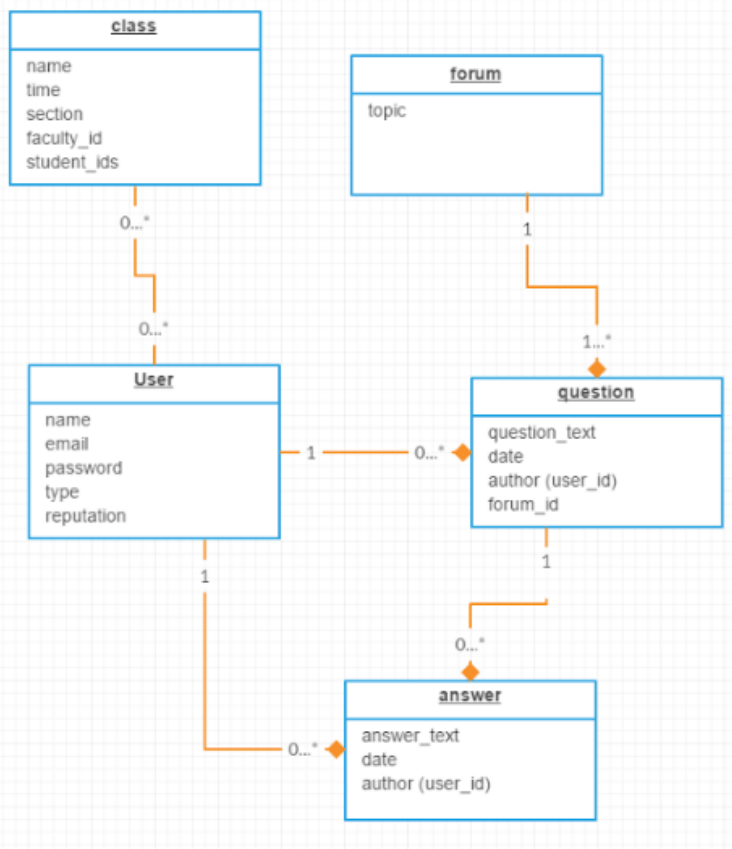
Manage Users



Design Decisions

For the class diagram, each conceptual entity was represented. For the system, this included all the types of users (students, faculty, admin), and the entities they could create (answers, questions, courses). These entities readily translated to objects in the system, with the slight difference that students, faculty, and admin were all lumped into one object of user, with an argument defining the type of user. As typical with Rails scaffolding, these database objects were quickly translated to implemented objects, each with its own helper, controller, and view. Therefore, the logic concerned with each object would be put in its respective helper or controller, while the user interface elements would be put in its respective view html files. For example, the user helper is responsible for finding all of the courses linked to that user, so that these courses can be displayed on the user view. Also, the view elements are split into different categories based on the type of view: show, edit, and new. The index html is the container for these different types of view files, while the show, edit, and new allow the end user to view details of the object in the case of show, or present a form for the enduser to fill out, in the case of edit and new.

Database Design



Most of our classes from the class diagram are represented as their own tables in our database. The only difference is that the “user” table encapsulates the student, faculty, and admin. These three users are differentiated by the field “type” which specifies if they are a student, faculty member, or an admin. The “answer” and “question” tables map to the answers and questions in the class diagrams. They have the basic fields of the person who wrote them (author) which is designated by the user_id, what they actually wrote (question_text and answer_text), and the date of when they wrote it. The “question” table also have a forum_id foreign key that maps the question to the forum that it’s apart of. The “forum” table only has one field which is the topic of the forum. The class table has the name of the class, the section, the time of the class, and the user_ids associated with this class (this is what the faculty_id and student_id are refering to). Lastly, the “user” table has information about the user which includes their name, email, password, what type of user they are, and their current reputation for answering questions.

These are the following rails commands we used to create the database:

Instantiating tables

- `bin/rails generate scaffold User name:string email:string password:string type:string reputation:Integer`
- `bin/rails generate scaffold class name:string time:string section:string user_id:Integer`
- `bin/rails generate scaffold forum topic:string`

- bin/rails generate scaffold question question_text:string date:string user_id:Integer forum_id:Integer
- bin/rails generate scaffold answer answer_text:string date:string user_id:Integer
- bin/rake db:migrate

We had to update the user table because type was a reserved word in the language so we used this command to do it:

```
rails g migration add_user_type_to_user user_type:integer
```

```
Rake db:migrate
```