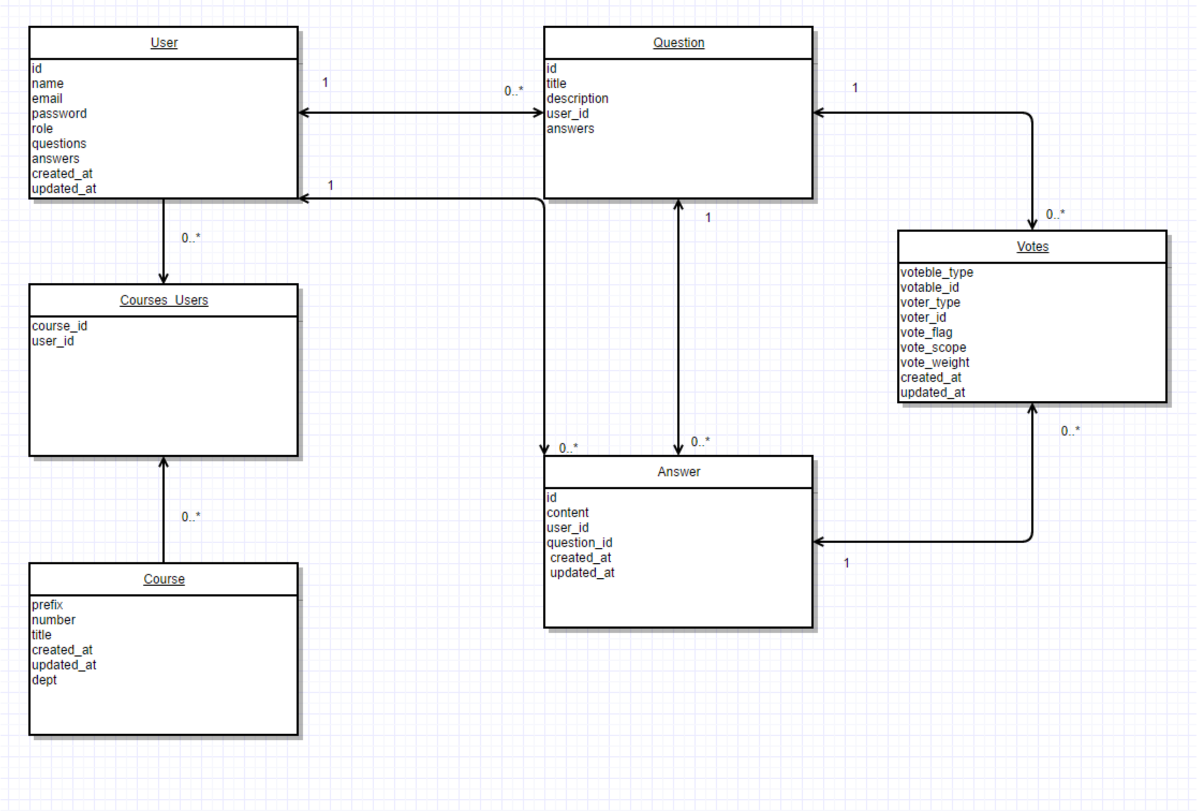
Design Document for 2b |! 2b

Table of Contents:

Discussion of Changes

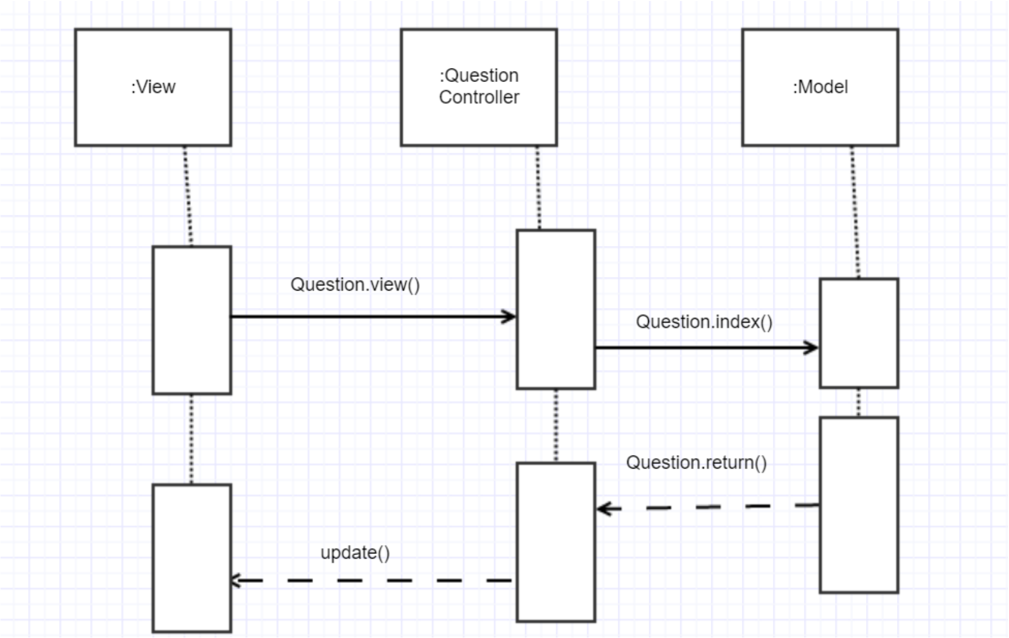
We added a new diagram which shows how the questions and answers can be voted upon. Essentially, the User interacts clicks on the vote, which tells the Question/Answer Controller that the database needs to be updated. We updated the diagram for creating a question. When a Question is made, there is a title, course ID, and a description passed as parameters.

Design Class Diagram

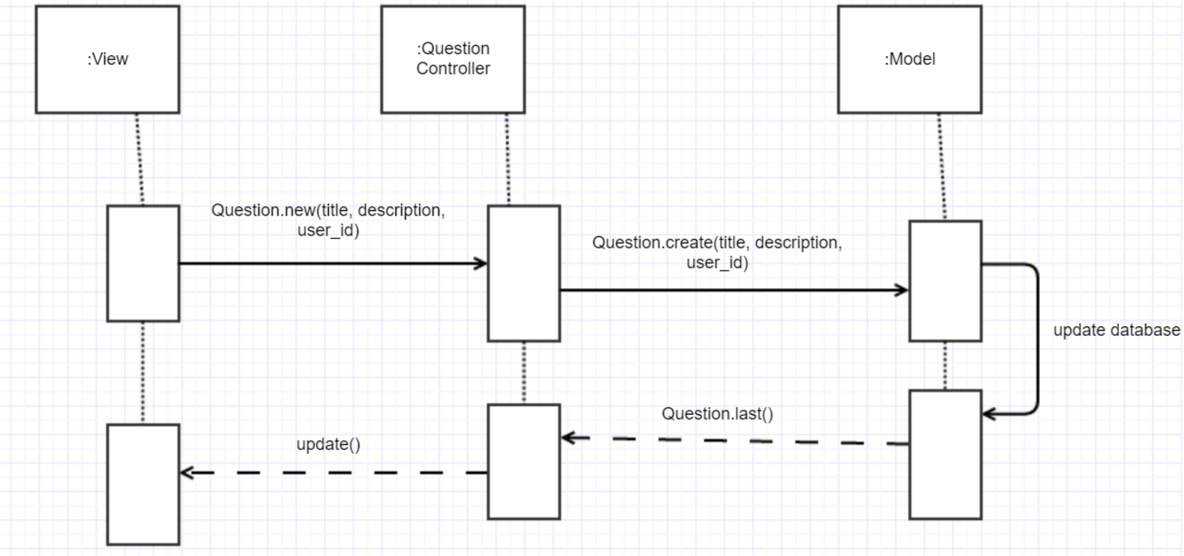


Design level sequence diagrams

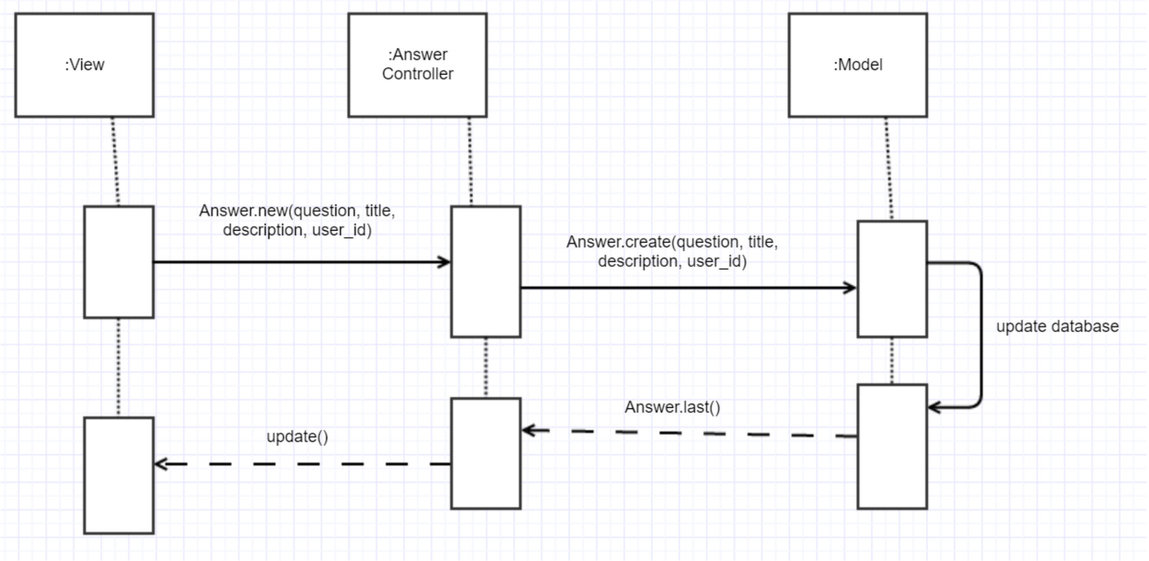
View Question



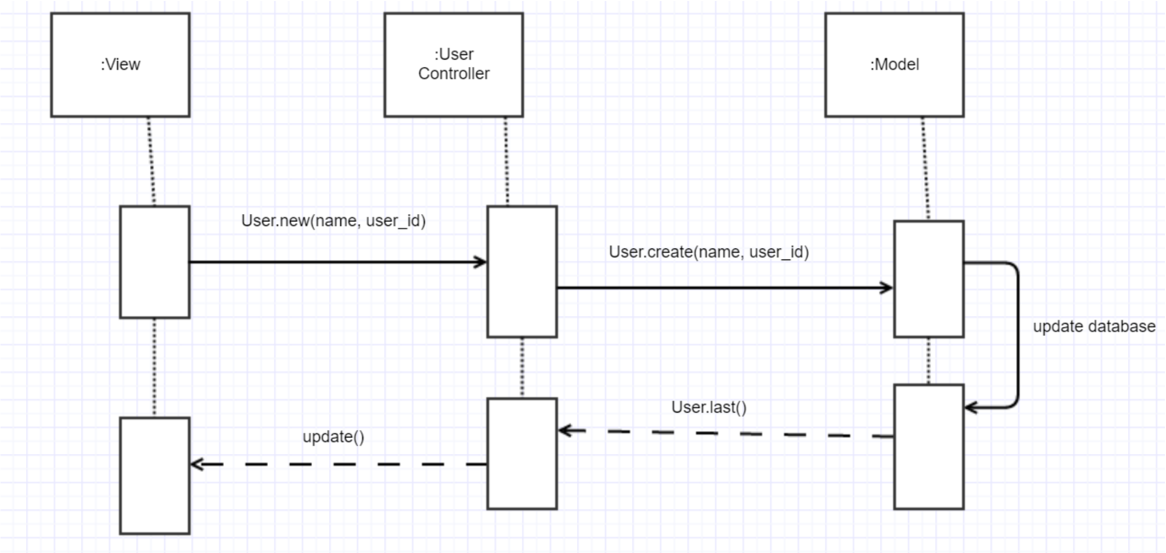
Create Question


:View
:Question Controller
:Model
Question.new(title, description, user_id)
Question.create(title, description, user_id)
update database
Question.last()
update()

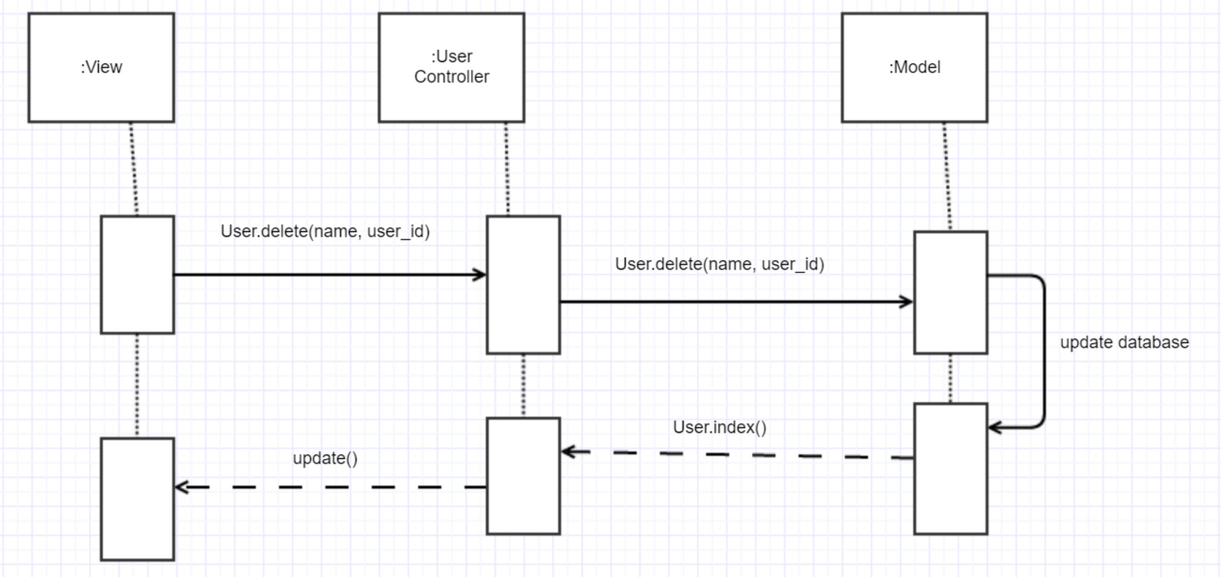
Answer Question


:View
:Answer Controller
:Model
Answer.new(question, title, description, user_id)
Answer.create(question, title, description, user_id)
update database
Answer.last()
update()

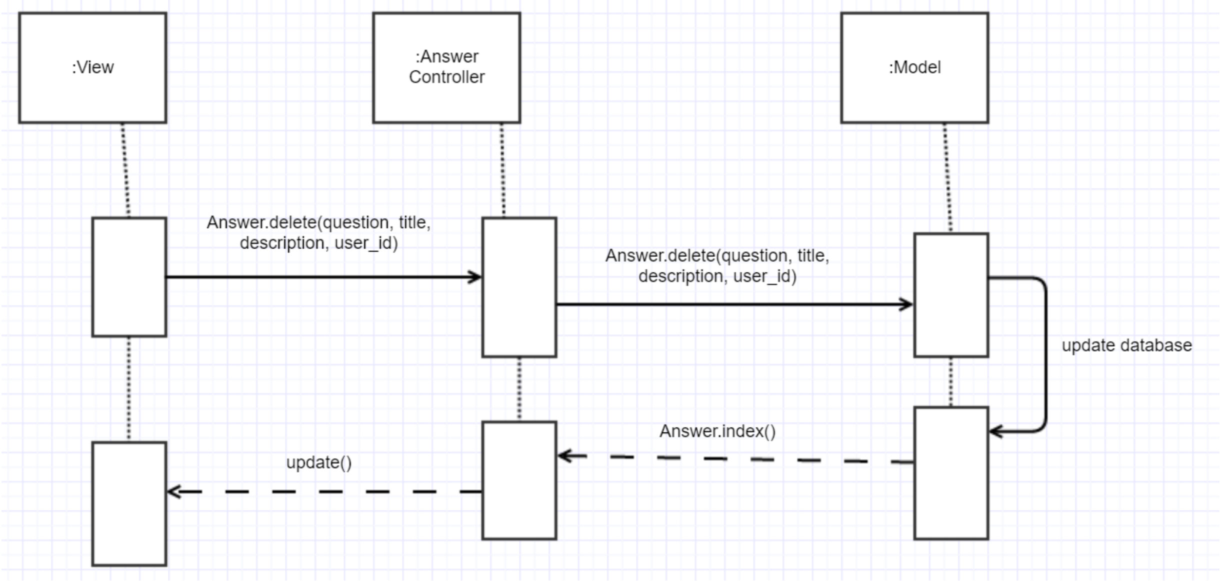
Create User



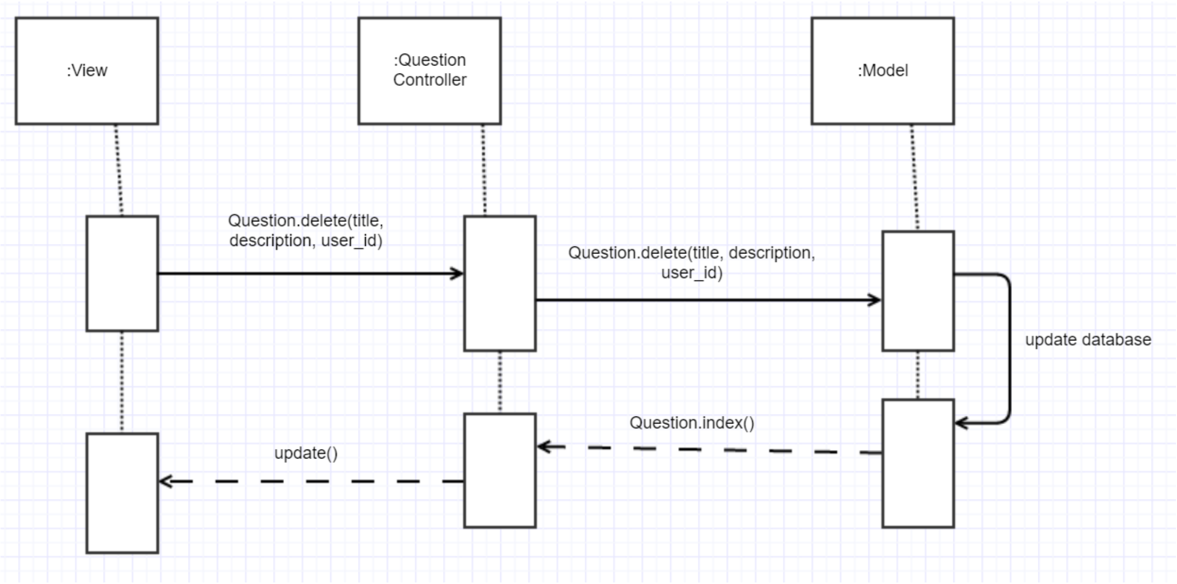
Delete User



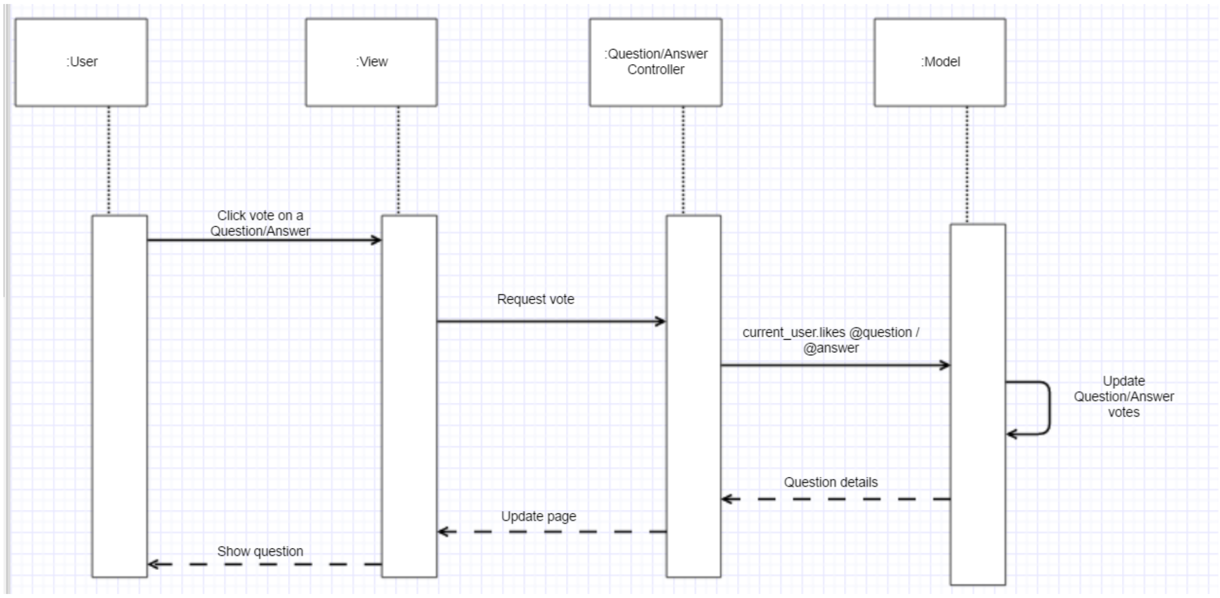
Delete Answer


:View
:Answer Controller
:Model
Answer.delete(question, title, description, user_id)
Answer.delete(question, title, description, user_id)
update database
Answer.index()
update()

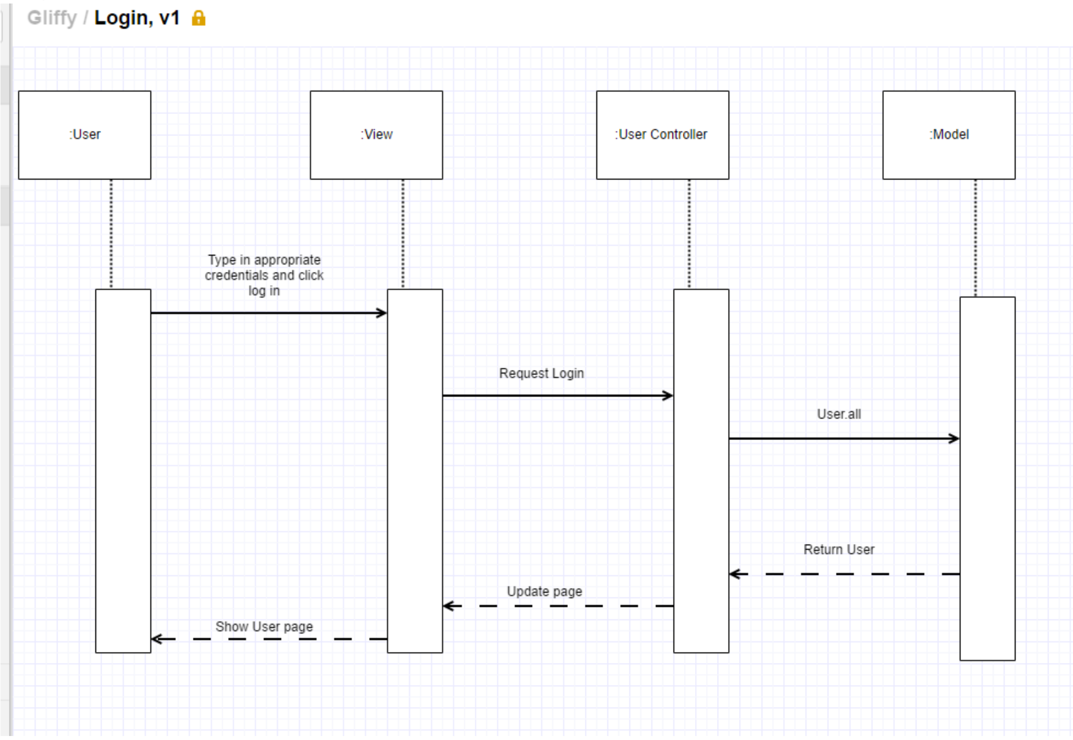
Delete Question


:View
:Question Controller
:Model
Question.delete(title, description, user_id)
Question.delete(title, description, user_id)
update database
Question.index()
update()

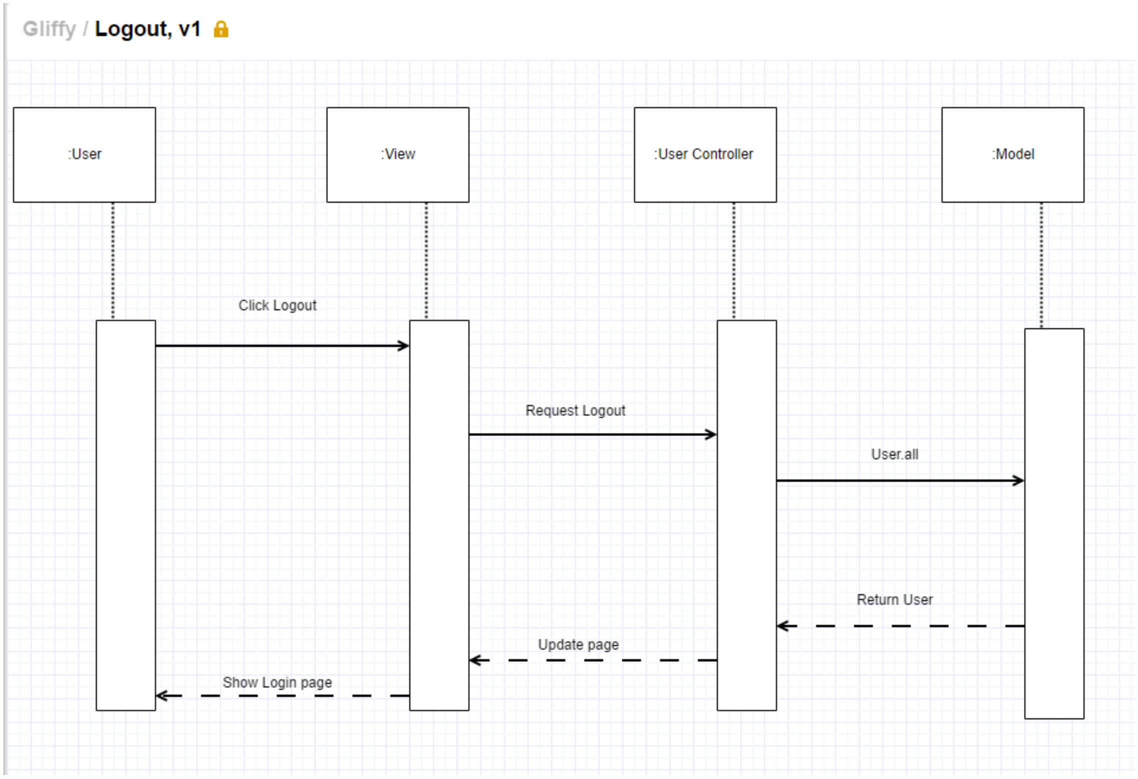
Vote Question/Answer


:User
:View
:Question/Answer Controller
:Model
Click vote on a Question/Answer
Request vote
current_user.likes @question / @answer
Update Question/Answer votes
Question details
Update page
Show question

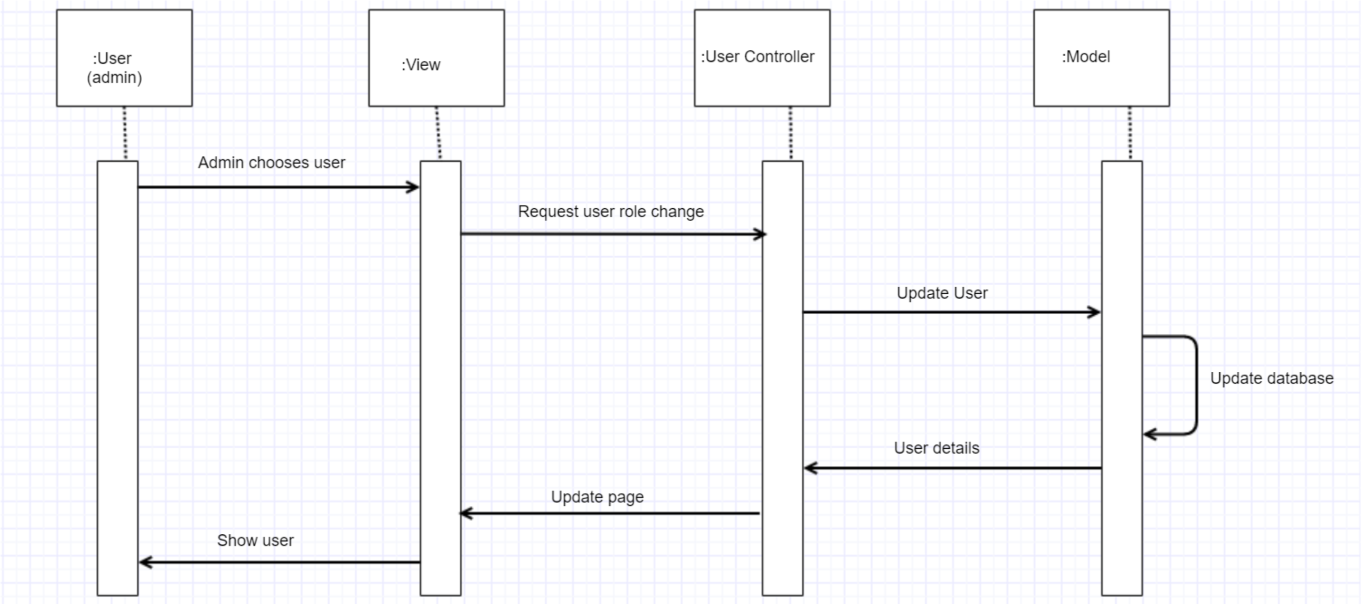
Login



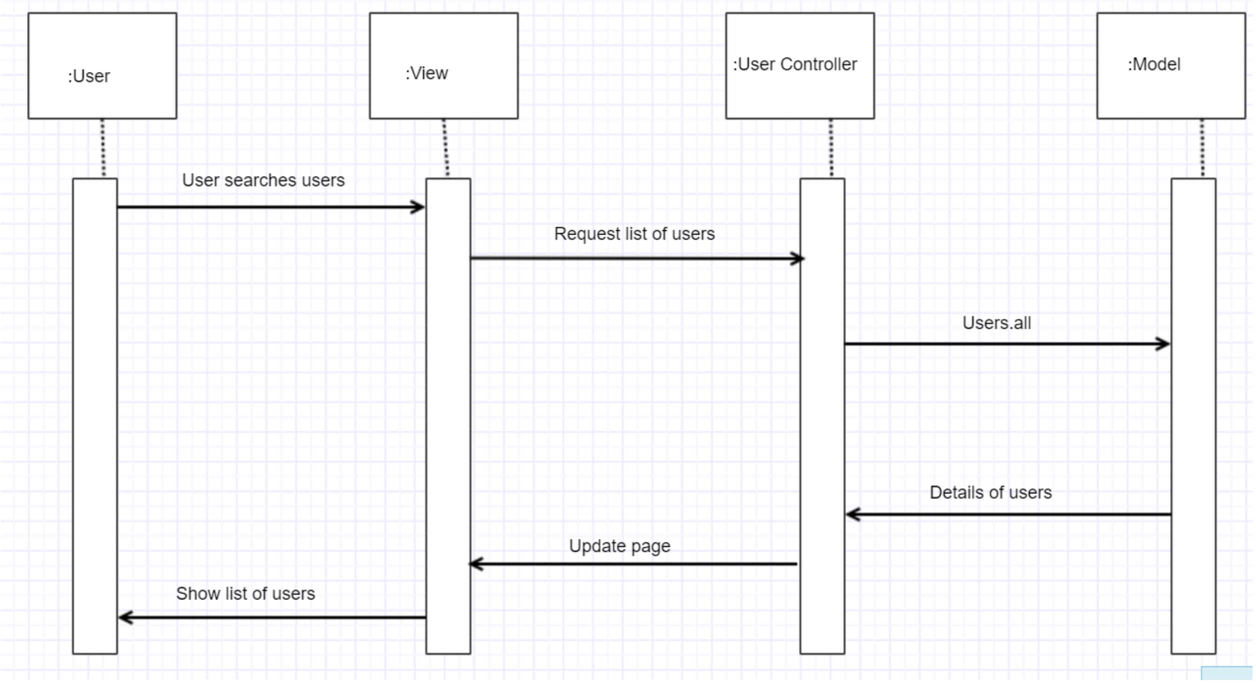
Logout



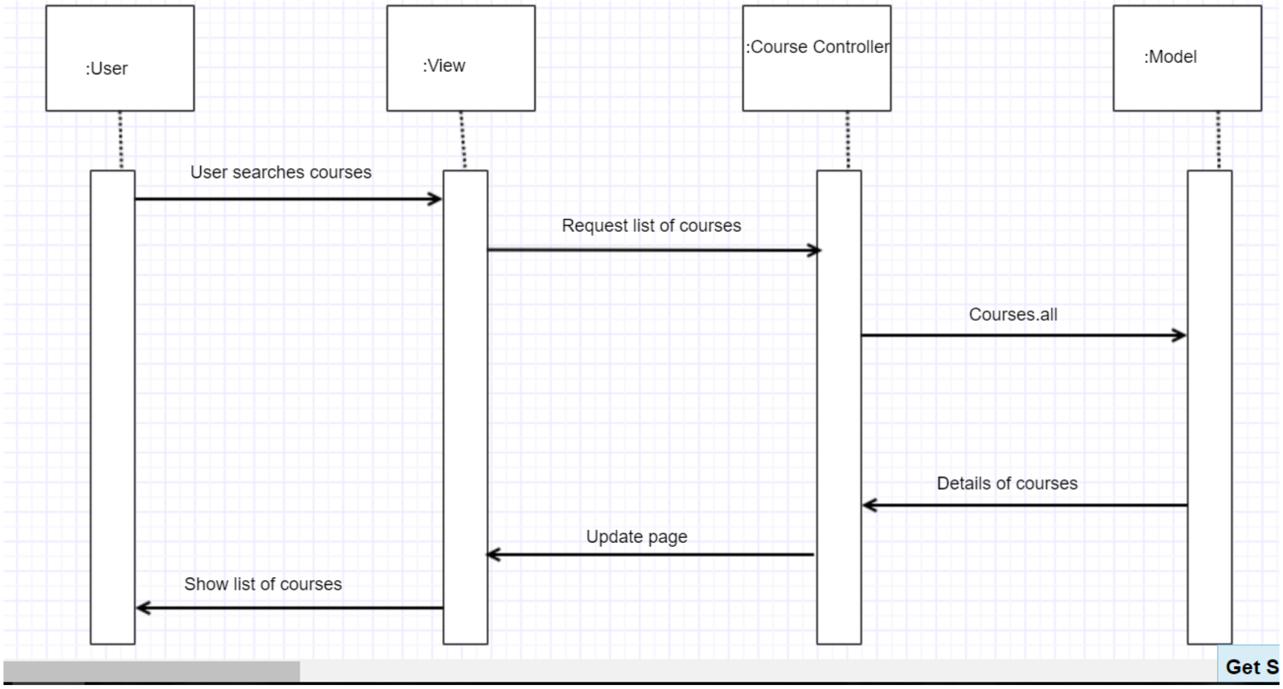
Change User Role



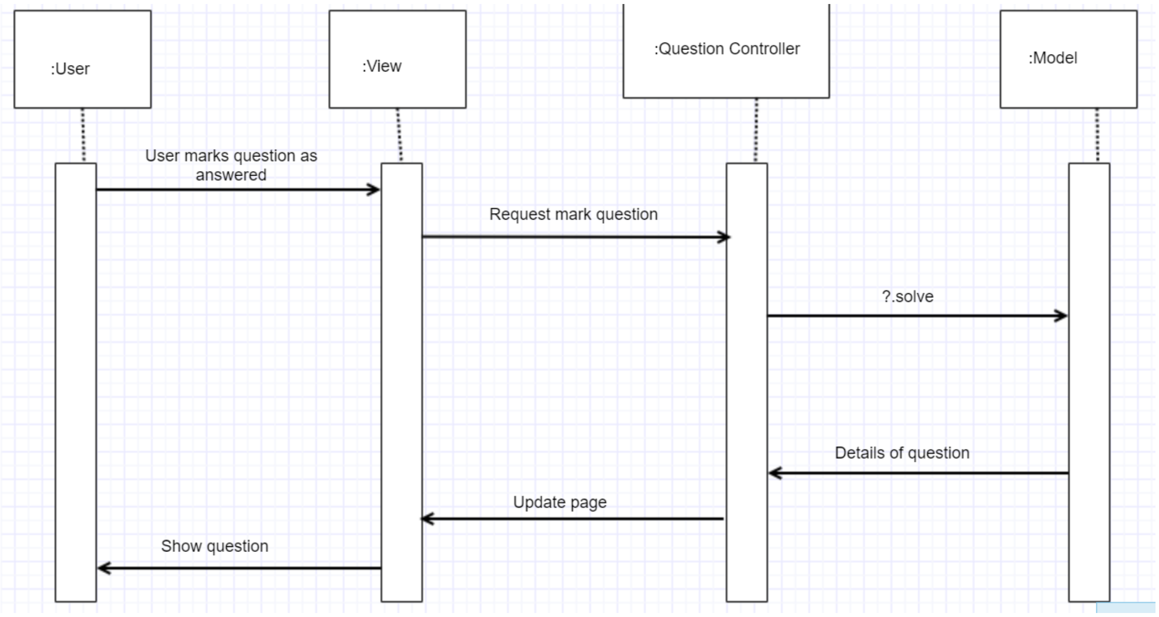
Search Users



Search Courses



Mark Question



Discussion of design decisions

For the first iteration of NKUNet we implemented question posting and answering. We also implemented a generic "User" class that allows us to create a general user from which we can then assign 'Roles'. This 'Role' system allows us to implement our "Student, Faculty, Registrar and Administrator" with their own given permissions. This way we have versatility in how we can create and add users, and if a faculty member becomes a admin or a student becomes faculty member. Instead of recreating a brand-new user and deleting the old account we can simply change the role of the user. This saves time and allows said user to keep the content they had previously. Users are also allowed to edit their answers and comments; students and faculty can delete answers to questions but are not allowed to delete questions once someone has commented/answered it. Registrar and Administrators can delete questions and answers if they deem them inappropriate.

For this iteration, we added a voting system as well as classes in which students and faculty can join and post questions and answers/comments. The voting system allows users to vote and show if the question or answer was helpful or if question or answer were off topic, it can be down voted. We also added file upload functionality during this iteration. The file upload allows students and faculty to post useful and educative documents.

Discussion of database design

The database now consists of Users, Questions, Answers, Courses, and Votes. Each User can have many Questions and many Answers, which they have posted to the site. Users also have multiple courses. Each Question can have many Answers posted in the question thread. Each course has Users and Questions. Both Answers and Questions now have Votes.

Iteration write up

During our design process, we had many ideas that we wanted to implement and many that we didn't. Our major design decision would be the use of a generic user class instead of creating each user class individually. We achieved this with the use of a role parameter, that we would assign to the generic user and from there the user would have access to permissions that the specific role has. We chose to go with this design because it provided a simple user creation. Instead of looking through a list of users and having to decide if the user is a student, faculty, registrar or admin. We can just make and account for the user and then give them their role. So, if we made a mistake or if their permissions are being elevated, instead of creating a new account and deleting the old one. We can just change the role of the account. Some other choices we had to make were how to handle forums and how posting questions and comments/answers would work. For forums, we decided to take each class that the user is enrolled in. This way all questions and comments/answers are in a centralized area and can be accessed by the student or faculty easily and with little search time. We set up our question and answers to have a primary id to make it easier to look up and call the question. The questions and answer also contain a user id, to track which user has posted what question/answer and link them together. Another decision that we had to do was agree on how to implement the voting system. We debated if Our last decision that we had to make for this iteration was how we are going to implement the file upload system. We originally thought of writing one from scratch, but decided that a premade gem would be a lot better and easier to use and implement. We decided on using a gem called Paperclip. Paperclip allows for basic file uploading, validations and callbacks, and post-processing. This had all the features that we needed and was all done, we just had to implement it.

The classes that we created for this iteration were User, Questions, Answers, Votes, Course Users and Courses. Each user has and ID, Name, Email, Password, Role, Questions, Answers, Created_At and Updated_At. Users can ask many questions and answer many questions. Questions and Answers can have only one asker. Each Question and Answer has a Vote associated with it, Questions and Answers can have multiple votes. Each User belongs to a Course, this is through the Course_Users and each Course belongs

Grasp Patterns

Controller- Our user class is a good example of controller, because we have it so the system can create and delete users as well as decide on roles.

High Cohesion- Users who post questions or answered questions are linked together. Votes are also linked and connected to questions and answers. Once a user asks a question, the user gets an update that the question is created, and if a user comments/answers a question. The question gets and update as well as the user.